الأكاديمية العربية الدولية

الأكاديمية العربية الدولية

المقررات الجامعية

www.aiacademy.info | care@aiacademy.info

بسم الله الرحمن الرحيم

الكلية الجامعية للعلوم التطبيقية
University College of Applied Sciences

أساسيات برمجة تطبيقات الهواتف الذكية باستخدام نظام أندرويد

(متطلب كلية للحصول على درجة الدبلوم المتوسط)

تأليف

د. إياد محمد قاسم الأغا

أستاذ مساعد في كلية تكنولوجيا المعلومات – الجامعة الإسلامية

السنة/ 2015

قررت الكلية الجامعية للعلوم التطبيقية تدريس كتاب " **أساسيات برمجة تطبيقات الهواتف الذكية باستخدام نظام أندرويد**" لطلبتها

بدءاً من العام الدراسي 2014/2013م

الإشراف العام	
رئيس الكلية	أ . د. رفعت نعيم رستم
نائب الرئيس للشئون الأكاديمية	د. أحمد فؤاد عبد العال

مركز التطوير الأكاديمي	
إشراف إداري	د. سناء الصايغ

تحكيم علمي	
محكم علمي	
محكم علمي	

تدقيق لغوي	
مدقق لغوي	

العمل الفني	
تصميم	
رسوم	
تنضيد	

منشورات الكلية الجامعية للعلوم التطبيقية

يطلب هذا الكتاب من الكلية الجامعية للعلوم التطبيقية، غزة-فلسطين

بيانات الاتصال بالكلية الجامعية للعلوم التطبيقية

عنوان الكلية	بيانات الاتصال
تل الهوا-متفرع من شارع عون الشوا غزة-فلسطين	**Tel**: (+970) 8 2868999 **Fax**: (+970) 8 2847404 **P.O Box**: 1415 Gaza Palestine **بريد إلكتروني**: info@ucas.edu.ps **الموقع الإلكتروني**: www.ucas.edu.ps

بيانات الكتاب

بيانات الطبعة	
رقم الإيداع	
رقم المقرر	
بيانات النشر	

إهـــداء

أهدي هذا العمل المتواضع إلى والدتي العزيزة ووالدي رحمه الله والذين كانا خير داعمين لي في مسيرتي العلمية

كما أهدي هذا العمل إلى عائلتي الصغيرة: زوجتي وإبنائي لصبرهم على انشغالي عنهم ودعمهم لي لإنجاز هذا العمل

مقدمة الكتاب

في ظل التزايد المضطرد في استخدام الهواتف الذكية والأجهزة اللوحية، تعتبر تطبيقات الهواتف الذكية من أكثر البرمجيات التي يتطلبها سوق العمل في مجال تكنولوجيا المعلومات. على الرغم من وجود العديد من المراجع والكتب المتخصصة في تعليم برمجة تطبيقات الهواتف الذكية، إلا أن معظم هذه المراجع معدة باللغة الإنجليزية وتستهدف المطورين الذين يتقنون اللغة الانجليزية، حيث لايزال هناك قصور كبير في المراجع العربية في هذا المجال. يعتبر هذا العمل المتواضع من أوائل الأعمال التي تتناول تطوير تطبيقات الهواتف الذكية باستخدام نظام أندرويد. كما أنه من أوائل الأعمال الموجهة أكاديمياً وذلك ضمن تخصص تطوير تطبيقات الهواتف الذكية في الكلية الجامعية للعلوم التطبيقية – غزة.

هذا الكتاب هو المرجع الرئيس لمساق تطوير تطبيقات الهواتف الذكية 1 حيث تعتمد دراسة هذا المساق على متطلبين سابقين هما: أساسيات البرمجة والبرمجية الشيئية (Object Oriented Programming) وكذلك تصميم واجهات الهواتف النقالة. لذلك فإن هذا الكتاب لايغطي مبادئ البرمجة الشيئية التي يجب أن يكون الطالب ملماً بها. كذلك لن يتم التطرق في هذا الكتاب إلى الطرق المختلفة لتصميم الواجهات (Layouts) باستخدام XML، حيث سيتم التركيز على التعامل مع عناصر واجهة المستخدم برمجياً فقط. كذلك يجب أن يكون لدا الطالب معرفة بمفهوم قواعد البيانات وبناء الجداول وأوامر SQL المختلفة.

يراعي الكتاب في تقديمه وعرضه الإحتياجات الأكاديمية وذلك بالدمج ما بين المفاهيم النظرية التي يجب أن يلم بها الطالب بالإضافة إلى الجانب العملي الذي يتم تنميته من خلال عدد كبير من الأمثلة والتمارين. يحتوي الكتاب على عشر وحدات تغطي المواضيع الأساسية التي يتطلبها المطور المبتدئ. كذلك يحتوي الكتاب في وحداته العشر على أكثر من عشرين تمرين عملي محلول معدة من قبل مؤلف الكتاب و مفصلة خطوةً بخطوة. تهدف هذه التمارين إلى تدريب الطالب على بناء تطبيقات متكاملة تغطي مختلف المواضيع التي يتناولها الكتاب. كذلك تحتوي كل وحدة في نهايتها على عدد من الأسئلة للتقييم الذاتي.

محتويات المقرر

التراجم

ترجمة المصطلحات التقنية المتعلقة بالبرمجة الشيئية ونظام أندرويد كان من الصعوبات التي واجهت المؤلف في إخراج الكتاب. لشرح المصطلحات التقنية باللغة العربية بدون الإخلال بمفاهيم هذه المصطلحات قمنا بترجمة المصطلحات إلى العربية وقمنا أثناء الشرح بذكر الترجمة العربية ملحوقة بالمصطلح الإنجليزي بين قوسين. التراجم المستخدمة للمصطلحات المختلفة موضحة في الجدول التالي. نود التأكيد على أن بعض المصطلحات لم تترجم حرفياً حيث تم الاعتماد أحياناً على تراجم قريبة لكونها أقرب للمفهوم الفعلي من الترجمة الحرفية من وجهة نظر المؤلف.

من الضروري أيضاً أن نؤكد على أن ترجمة المصطلحات هي بغرض مساعدة القارئ العربي على فهم السياق وليس لاستبدال المصطلحات الأصلية بالتراجم العربية. لذلك نوصي كلاً من المدرس والطالب بالحرص على استعمال المصطلح الإنجليزي قدر الإمكان.

المصطلح	الترجمة المستخدمة في الكتاب
Activity	الفعالية
Service	الخدمة
Broadcast Receivent	مستقبل النشر
Manifest File	ملف الوثيقة
Layout File	ملف التصميم
Assets Folder	مجلد الممتلكات
Activity Lifecycle	دورة حياة الفعالية
Backstack	الحافظة الخلفية
Intent	الهدف
PendingIntent	الهدف المعلق
Explicit Intent	الهدف الصريح
Implicit Intent	الهدف المضمن
Intent Filter	مرشح الهدف
Action	حدث أو إجراء
Class	فئة
Subclass	فئة فرعية
Superclass	فئة رئيسية
Interface	الواجهة البرمجية
Object	كائن
Method	دالة
Callback Method	دالة الاتصال الراجع
Constructor	الباني
Package	حزمة
Listener	مستمع
Adapter	محول

ListView	قائمة العرض
Shared Preferences	التفضيلات المشتركة
Bundle	الحزمة
OptionsMenu	قائمة الخيارات
Context Menu	قائمة السياق
Popup Menu	القائمة المنبثقة
MenuItem	عنصر القائمة
Action Bar	شريط العمل
Action Overflow	ملحق شريط العمل
Navigation Drawer	درج التصفح
XML Attribute	خاصية XML
Permission	إذن
Notification	إشعار
Notification Drawer	درج الإشعارات
Content Provider	مزود المحتوى
Content Resolver	مستخرج المحتوى
Fragment	القطعة
Fragment Manager	مدير القطع
Dialog	مربع الحوار

الوحدة الأولى:

الفعالية (Activity)

يتعلم الطالب في هذه الوحدة:

- ✓ . مفهوم الفعالية (Activity) في نظام أندرويد وطريقة إنشائها.
- ✓ . دورة حياة الفعالية (Activity) وطريقة إدارتها بصورة سليمة.
- ✓ . حفظ حالة الفعالية (Activity) واستردادها.
- ✓ التعريف بأوضاع العمل الخاصة بالفعالية (Activity) واستخدام الحافظة الخلفية (Back stack).
- ✓ . التدريب العملي على إنشاء الفعالية (Activity) ومعالجة حالاتها المختلفة.

هناك أنواع مختلفة من المكونات يمكن أن يبنى منها تطبيق الأندرويد. أول هذه المكونات يسمى بالفعالية (Activity) وهي جزء من تطبيق أندرويد يوفر للمستخدم واجهة تفاعلية تمكنه من تنفيذ أمر ما مثل تصفح الأخبار، البحث عن معلومة، الاتصال الهاتفي، التقاط الصور، عرض خريطة أو أي مهمة أخرى. كل فعالية (Activity) تمثل نافذة مستقلة في التطبيق. هذه النافذة قد تعرض في وضع ملء الشاشة أو قد تحتل جزء صغير من الشاشة.

تطبيق الأندرويد يتكون عادة من مجموعة من الفعاليات (Activities) الغير مرتبطة ببعضها. كل تطبيق له عادةً فعالية رئيسية واحدة (Main Activity) وهي التي تعرض على الشاشة عن تشغيل التطبيق. بعد تشغيل الفعالية الرئيسية (Main Activity) ، يمكن من خلالها تشغيل فعاليات أخرى لتنفيذ مهمات مختلفة. على سبيل المثال، عند تشغيل تطبيق إخباري لأول مرة يتم عرض قائمة الأخبار في الواجهة الخاصة بالفعالية الرئيسية (Main Activity)، وعند اختيار أي خبر محدد بالنقر عليه، يتم تشغيل فعالية جديدة لعرض تفاصيل الخبر.

واجهة المستخدم يتم عادةً إنشاؤها وتصميمها من خلال ملفات XML كما درست مسبقاً. الفعالية (Activity) تقوم عند بدء تشغيلها بإنشاء الواجهة بناءً على محتوى ملف XML. في ملف الفعالية (Activity) يتم كتابة كود يمثل الإجراءات المختلفة التي يجب تنفيذها عند تفاعل المستخدم مع عناصر الواجهة. وبذلك يكون هناك فصل كامل بين تصميم الواجهات، والذي يتم من خلال ملفات XML، والإجراءات اللازم تنفيذها، والتي يتم تحديدها داخل الفعالية (Activity).

قبل الحديث عن التعامل مع واجهة المستخدم والتفاعل مع الأحداث المختلفة UI Events، سيتم شرح كيفية إنشاء الفعالية (Activity) وإدارتها.

إنشاء الفعالية (Creating the Activity)

لإنشاء فعالية جديدة، يجب عليك إنشاء فئة فرعية (subclass) من الفئة (Activity) أو أي فئة متفرعة من Activity). بعد ذلك عليك كتابة الكود الخاص بدوال الاتصال الراجع (Callback Methods). دوال الاتصال الراجع هي دوال يتم تنفيذها تلقائيا من قبل النظام (بدون تدخل المستخدم) بناء على التغير في دورة حياة الفعالية (Activity Life Cycle).

فمثلاً عند بدء تشغيل الفعالية يقوم النظام بتشغيل الدالة ()onCreate، وعند إزالة الفعالية (Activity) يتم تشغيل الدالة ()onDestroy. يمكن للمستخدم تنفيذ أي كود عند بدء تشغيل الفعالية (Activity) عن طريقة إضافة هذا الكود إلى الدالة ()onCreate. فمثلاً، نقوم عادة بكتابة الكود الخاص بإنشاء محتويات واجهة المستخدم داخل الدالة ()onCreate وذلك لتجهيز الواجهة للعرض عن بداية تشغيل الفعالية (Activity). أهم الدوال التي نحتاج عادةً إلى إضافة كود لها عند إنشاء الفعالية (Activity) هي:

- ()onCreate: يجب كتابة الكود الخاص بهذه الدالة والتي ينفذها النظام تلقائياً عند إنشاء الفعالية. نقوم عادةً في هذه الدالة بتهيأة وانشاء المكونات المختلفة الخاصة بالفعالية. من خلال هذه الدالة يتم تحديد هيكلية الواجهة الخاصة بالفعالية عن طريق استدعاء الدالة ()setContentView. نقوم عادة بتحديد معرف خاص بملف XML الخاص بالواجهة وتمريره إلى الدالة ()setContentView.
- ()onResume: يتم تنفيذ هذه الدالة عندما تصبح الفعالية مشاهدة visible وذلك بعد إنشائها أو استئناف عملها بعد توقف. أي أن هذه الدالة يتم تنفيذها بعد الدالة ()onCreate.قد تستخدم هذه الدالة لاسترجاع حالة الفعالية أو لربطها بمكونات التطبيق الأخرى مثل الخدمات (Services).
- ()onPause: يقوم النظام باستدعاء هذه الدالة تلقائياً عند ايقاف الفعالية. إيقاف الفعالية قد يكون نتيجة تشغيل فعالية اخرى تحجبها جزئياً عن المشاهدة، ولا يعني بالضرورة إنهاء الفعالية أو إخفائها. نحتاج إلى كتابة هذه الدالة لحفظ حالة الفعالية والبيانات التي تستخدمها حتى لا تفقد، وحتى يتم استعادتها عن الرجوع للفعالية.
- ()onStop: يتم تنفيذ هذه الدالة عندما تصبح الفعالية غير مشاهدة تماماً. يمكن استخدام هذه الدالة لإيقاف العمليات التي تنفذها الفعالية.

هناك المزيد من دوال الاتصال الراجع (Callback Methods) والتي سيتم التطرق لها عند الحديث عن دورة حياة الفعالية (Activity Life Cycle) لاحقاً في هذه الوحدة.

الإعلان عن الفعالية في ملف القائمة Manifest

بعد إنشاء الفئة (class) الخاصة بالفعالية، يجب الاعلان عن الفعالية في ملف الوثيقة Manifest. ملف الوثيقة Manifest يحدد الإعدادات المختلفة للتطبيق والمكونات التي يستخدمها وسيتم شرحه في الوحدة الثانية من هذا الكتاب. الإعلان عن فعالية ما (Activity) يتم عن طريق إضافة الخاصية <activity> داخل الخاصية <application> كما هو موضح في الكود التالي (ملاحظة: قد تقوم بيئة العمل مثل Eclipse بعمل ذلك تلقائياً):

```
<manifest ... >
  <application ... >
      <activity android:name=".ExampleActivity" />
      ...
  </application ... >
  ...
</manifest>
```

هناك أيضاً بعض الخصائص التي تضاف داخل المكون <activity> والخاصة بالفعالية مثل اسم الفعالية android:name والذي يجب اضافتها لتحديد اسم الفئة (class) الخاص بالفعالية.

إدارة دورة حياة الفعالية (Managing the Activity Life Cycle)

إدارة دورة حياة الفعالية (Activity) عن طريق كتابة الكود الخاص بدوال الاتصال الراجع (Callback Methods) إجراء مهم لبناء تطبيق صحيح. دورة حياة الفعالية (Activity) تتأثر مباشرةً بارتباطها بالفعاليات الأخرى. الفعالية يمكن أن تكون في حالة من ثلاثة حالات:

- وضع العمل running: وفي هذه الحالة تكون الواجهة في المقدمة ومفعلة من قبل المستخدم.
- وضع الإيقاف المؤقت paused: تنتقل فعالية ما إلى وضع paused عندما يتم تشغيل فعالية أخرى تنتقل إلى الواجهة وتكون هذه الفعالية الجديدة شفافة جزئياً أو لا تملأ الشاشة بحيث لا يزال بالإمكان رؤية الفعالية الأولى. الفعالية في وضع الإيقاف المؤقت تظل الفعالية في الذاكرة وتحتفظ بحالتها وقيم المتغيرات فيها، ولكن قد يلجأ النظام لتدميرها في حالة القصور الشديد في الذاكرة.
- وضع الإيقاف stopped: تنتقل الفعالية إلى هذه الحالة عندما تصبح غير مشاهدة تماماً ويتم حجبها خلف فعالية أخرى (أي أنها تصبح في المؤخرة). الفعالية في وضع الايقاف تحتفظ كذلك بحالتها وقيم المتغيرات فيها، ولكن قد يلجأ النظام لتدميرها في حالة القصور الشديد في الذاكرة.

عندما تنتقل الفعالية من حالة إلى أخرى من الحالات الموضحة أعلاه، يتم إعلام الفعالية من قبل النظام عن طريق تشغيل دوال الاتصال الراجع (Callback Methods) الملائمة حسب الحالة. لذلك، يمكن كتابة كود في هذه الدوال لتنفيذ مهام محددة عندما تتغير حالة الفعالية. الكود الموضح بالأسفل يوضح هيكلية الفعالية وأهم دوال الاتصال الراجع(Callback Methods):

```
public class ExampleActivity extends Activity{
    @Override
    public void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        // الفعالية في مرحلة الانشاء.
    }
    @Override
    protected void onStart(){
        super.onStart();
        // الفعالية على وشك أن تصبح مشاهدة. تنفذ هذه الدالة قبل العرض مباشرة.
    }
    @Override
    protected void onResume(){
        super.onResume();
        // تنفذ هذه الدالة بعد ان تصبح الدالة مشاهدة
    }
```

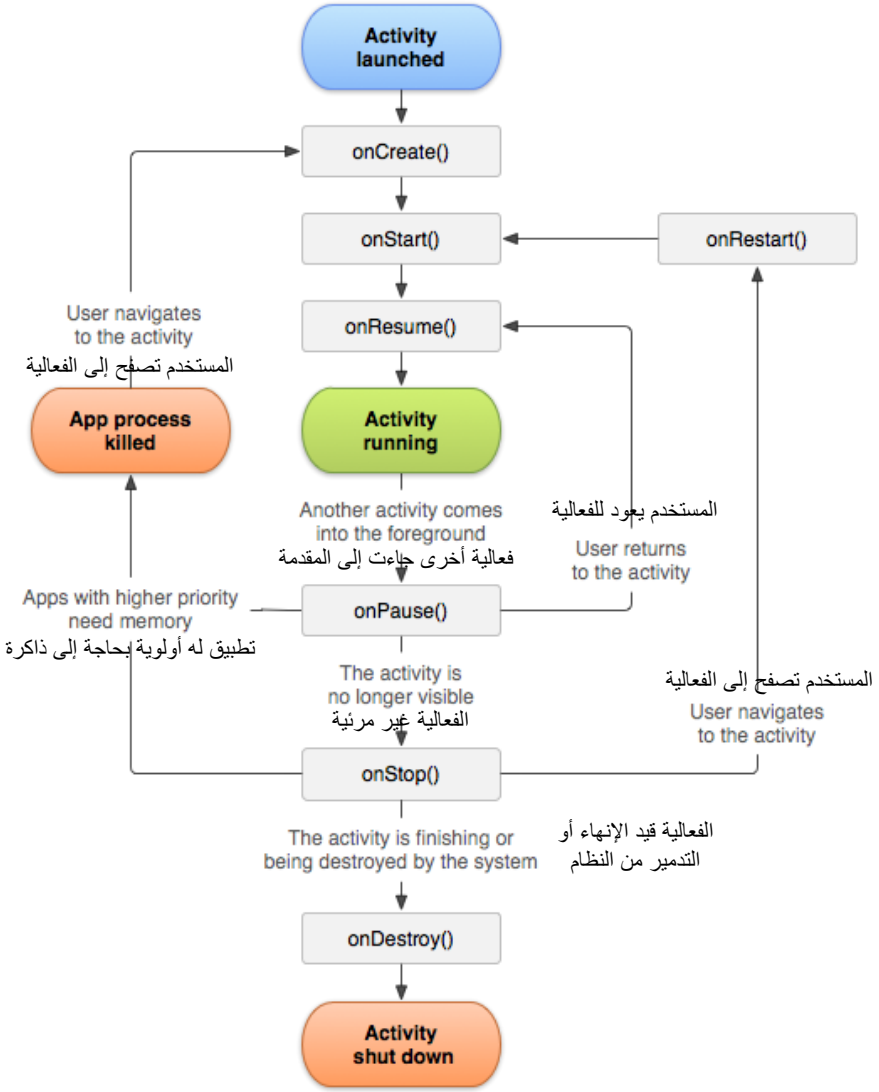
```
    @Override
    protected void onPause(){
        super.onPause();
        تنفذ هذه الدالة عن ايقاف الفعالية مؤقتا بسبب تشغيل فعالية أخرى//
    }
    @Override
    protected void onStop(){
        super.onStop();
        // الفعالية غير مشاهدة تماما وعليه تم ايقافها وتنفيذ هذه الدالة
    }
    @Override
    protected void onDestroy(){
        super.onDestroy();
        تنفذ هذه الدالة قبل تدمير الفعالية مباشرةً//
    }
}
```

ملاحظة: اذا أردت كتابة أي كود في دوال الاتصال الراجع فعليك أولاً تنفيذ دوال الاتصال الراجع في الفئة الأم (superclass) وذلك قبل تنفيذ أي كود جديد كما هو موضح بالشكل. فمثلاً، في الدالة ()onCreate يتم تنفيذ الدالة ()super.onCreate ثم يتم كتابة بقية الكود.

دورة حياة الفعالية كاملةً موضحة بالشكل 1-1 بالأسفل حيث تمثل المستطيلات دوال الاتصال الراجع (Callback Methods)، وتمثل الدوائر حالات الفعالية. دورة حياة الفعالية تحصل ما بين تنفيذ الدالة ()onCreate والدالة ()onDestroy. لذلك، كل ما يرتبط بإنشاء الفعالية من إنشاء هيكلية واجهة المستخدم والمتغيرات وتهيئة حالة الفعالية والاتصال بقواعد البيانات مثلا يتم في الدالة ()onCreate. كل ماهو مرتبط بانهاء حالة الفعالية من اغلاق الاتصال بالخدمات أو قواعد البيانات يمكن أن يتم في الدالة ()onDestroy.

هناك ما يسمى دورة الحياة المشاهدة للفعالية Visible lifecycle وتحصل ما بين الدالة ()onStart و ()onStop وخلالها تكون الفعالية مشاهدة من قبل المستخدم. لاحظ أن دورة الحياة المشاهدة هي جزء من دورة الحياة الكاملة. لاحظ أن الفعالية المشاهدة قد لا تكون بالضرورة في وضع العمل running (قد تكون في وضع الايقاف الموقت نتيجة تشغيل فعالية تحجبها جزئياً).

وهناك أيضا دورة الحياة في المقدمة Foreground lifecycle وهي جزء من الدورتين السابقتين وتحصل بين تنفيذ الدالة ()onResume والدالة ()onPause. خلال هذه الفترة تكون الفعالية في وضع العمل running، أي أنها تكون مشاهدة ويتم التفاعل معها من قبل المستخدم. لاحظ أن الكود الذي يتم كتابتة في الدالة ()onResume و الدالة ()onPause يتم تنفيذه عندما يحصل أي تغيير في حالة الفعالية، ولذلك تستخدم هذه الدوال بكثرة لحفظ حالة الفعالية والبيانات المتعلقة بها عند حصول أي تغيير في حالة الفعالية.

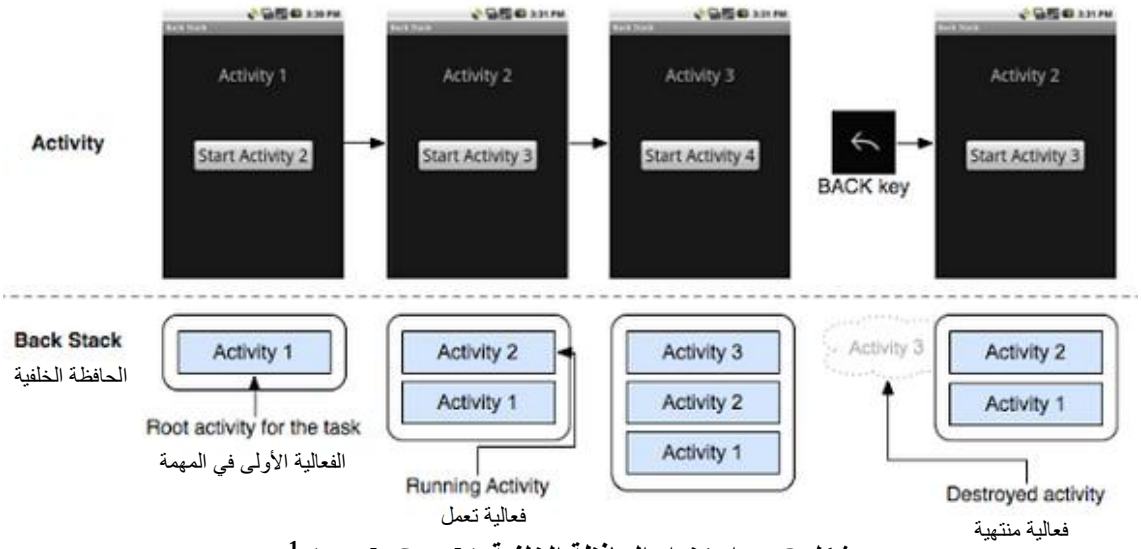


شكل 1-1: دورة حياة الفعالية (Activity Life Cycle) [1]

في كل مرة يتم فيها تشغيل فعالية جديدة، يتم ايقاف الفعالية السابقة. يقوم نظام الأندرويد بحفظ وترتيب الفعاليات في حافظة خلفية تسمى Back Stack كما هو موضح في شكل 2-1. كل فعالية جديدة يتم تشغيلها يتم إضافتها إلى مقدمة الحافظة تليها الفعاليات السابقة. الفعالية (Activity) التي في مقدمة الحافظة هي فقط تكون في وضع التشغيل running، بينما تكون كل الفعاليات التالية لها متوقفة عن العمل. عند إنهاء الفعالية التي في المقدمة عن طريق النقر على زر Back يتم إيقافها وإزالتها من الحافظة، وتصبح الفعالية التالية لها في مقدمة الحافظة الخلفية

[1] Activity, Android Developer, http://developer.android.com/reference/android/app/Activity.html

لتنتقل إلى وضع التشغيل. بالرجوع إلى مثال التطبيق الإخباري، عن تفعيل الفعالية الخاصة بأي خبر تنتقل هذه الفعالية إلى مقدمة الحافظة الخلفية، وتتوقف الفعالية الرئيسية عن العمل لتصبح في مؤخرة الحافظة الخلفية. عند انتهاء المستخدم من استخدام الفعالية الفرعية والنقر على زر Back، يتم ازالة الفعالية من الحافظة لتعود الفعالية الرئيسية للواجهة وتصبح في بداية الحافظة.



شكل 2-1: استخدام الحافظة الخلفية (Back Stack) [1]

تمرين عملي (1-1)

يهدف هذا التمرين إلى تتبع دورة حياة الفعالية (Activity Life Cycle) والتسلسل المستخدم في تنفيذ دوال الاتصال الراجع (Callback Methods). سنقوم في هذا التمرين بإنشاء فعالية بدون أي عناصر في واجهة، ومن ثم سنكتب كود طباعة في بعض دوال الاتصال الراجع. كود الفعالية موضح بالأسفل:

```
public class MainActivity extends Activity {

     @Override
     protected void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
          Toast.makeText(this, "onCreate",
Toast.LENGTH_SHORT).show();
     }
     @Override
     protected void onPause() {
```

[1] Android for 'Starters', http://techiezjunkyard-android.blogspot.com/2012/01/task-and-back-stack_8919.html

```
          super.onPause();
          Toast.makeText(this, "onPause",
Toast.LENGTH_SHORT).show();
     }
     @Override
     protected void onResume() {
          super.onResume();
          Toast.makeText(this, "onResume",
Toast.LENGTH_SHORT).show();
     }
     @Override
     protected void onStop() {
          super.onStop();
          Toast.makeText(this, "onStop",
Toast.LENGTH_SHORT).show();
     }
     @Override
     protected void onDestroy() {
          super.onDestroy();
          Toast.makeText(this, "onDestroy",
Toast.LENGTH_SHORT).show();
     }
}
```

قم بتشغيل التطبيق وتتبع ترتيب الرسائل التي يتم طباعتها.

بعد إكتمال تشغيل الفعالية وتوقف طباعة الرسائل، إنقر على زر Back لانهاء الفعالية، ولاحظ تسلسل الرسائل التي يتم طباعتها في هذه الحالة.

ملاحظة: لإنشاء رسالة ليتم طباعتها على الشاشة يمكن استخدام الدالة Toast.makeText(). هذه الدالة يمرر لها كائن من نوع Context وهو يمثل التطبيق الحالي ويمكن تمرير الفعالية الحالية. كما يمرر لها الرسالة المراد طباعتها ومدة العرض (Toast.LENGTH_SHORT أو Toast.LENGTH_LONG). بعد إنشاء الرسالة يتم طباعتها بتنفيذ الدالة Toast.show(). طباعة رسائل من نوع Toast سيتم استخدامه بكثرة في التمارين المذكورة في هذا الكتاب.

حفظ حالة الفعالية

ذكرنا مسبقاً أن الفعالية (Activity) في حالة الإيقاف stopped أو الإيقاف المؤقت paused يتم الإحتفاظ بحالتها ولا تفقد المعلومات الخاصة بها مثل حالة واجهة المستخدم ولذلك لأن الفعالية ما تزال في الذاكرة. ولكن عند إنهاء الفعالية من القبل النظام يتم أزالتها من الذاكرة وتفقد كل المعلومات الخاصة بها. عند الحاجة إلى تشغيل الفعالية مرة أخرى يقوم النظام بإنشاء نسخة جديدة من الفعالية. المستخدم يجب أن لا يلاحظ أن النظام قام بتدمير الفعالية السابقة وأنشاء أخرى جديدة، ولذلك يتوقع أن تحتفظ الفعالية بحالتها وبياناتها. ولذلك يجب على المبرمج أن يحرص على حفظ حالة الفعالية قبل تدميرها اذا كانت هناك حاجة لتشغيل الفعالية في وقت لاحق.

هناك أكثر من طريقة لحفظ حالة الفعالية والبيانات المتعلقة بها. يمكن للمبرمج استخدام الدالة ()onSaveInstanceState، والتي يقوم النظام تلقائياً بتنفيذها مباشرةً قبل إيقاف الفعالية. يمرر النظام لهذه الدالة حزمة Bundle وهي عبارة عن حافظة يمكن تخزين بيانات بداخلها عن طريق دوال إدخال مخصصة مثل ()putInt و ()putString. بعد تدمير الفعالية تظل كل البيانات المخزنة في الحزمة Bundle حتى يحتاج المستخدم إلى تشغيل الفعالية مرة أخرى. عندها، يقوم النظام بإنشاء نسخة جديدة من الفعالية ويقوم بتمرير الحزمة Bundle إلى الدوال ()onCreate و ()onRestoreInstanceState. باستخدام أي من هاتين الدالتين، يمكن استخراج الحالة والبيانات المحفوظة في الحزمة Bundle. إذا لم يكن هناك أن بيانات لاسترجاعها (كما في حالة تشغيل الفعالية لأول مرة)، تكون قيمة الحزمة Bundle هي null.

الكود التالي يوضح الدوال ()onSaveInstanceState و ()onRestoreInstanceState وكيفية التعامل مع الحزمة Bundle لحفظ واسترجاع البيانات، حيث يتم كتابة هذا الكود داخل الفعالية (Activity):

```
@Override
protected void onSaveInstanceState(Bundle outState) {
 super.onSaveInstanceState(outState);
 outState.putString("username", "someone.ucas.edu.ps");
}
@Override
protected void onRestoreInstanceState(Bundle
savedInstanceState) {
 super.onRestoreInstanceState(savedInstanceState);
 String username =
savedInstanceState.getString("username");
}
```

يجب الملاحظة أن حالة العناصر الموجودة في واجهة المستخدم UI Widgets مثل القيم الموجودة في EditText و CheckBox يتم حفظها في الحزمة Bundle واسترجاعها تلقائياً عند إعادة إنشاء الفعالية دون الحاجة إلى كتابة كود للقيام بذلك. كل ما على المبرج القيام به ليضمن الحفظ التلقائي لحالة عناصر واجهة المستخدم هو إعطاء اسم معرف غير مكرر لكل عنصر وذلك باستخدام الخاصية android:id في ملف التصميم Layout.

على الرغم من أن حالة عناصر الواجهة يتم حفظها تلقائياً، يمكن استخدام الدالة onSaveInstanceState() لحفظ بيانات أخرى مثل قيم المتغيرات والمصفوفات.

ملاحظة: ليس هناك ضمانة أن يتم تنفيذ الدالة onSaveInstanceState() من قبل النظام قبل تدمير الفعالية (كما في حالة أن يقوم المستخدم بإنهاء الفعالية بالنقر على زر back)، لذلك ينصح باستخدام هذه الدالة لكتابة الكود الخاص بحفظ حالة واجهة المستخدم فقط، ولا ينصح باستخدامها لحفظ بيانات أخرى مثل تلك التي يجب تخزينها في قواعد البيانات. في حالة الحاجة لحفظ بيانات يفضل استخدام onPause() لكتابة كود الحفظ في قواعد البيانات عندما يقرر المستخدم إنهاء الفعالية.

يمكنك إختبار قدرة تطبيقك على حفظ حالة فعالية ما عن طريق تدوير جهاز العرض (المحمول أو الجهاز اللوحي): عندما يتغير اتجاه العرض من رأسي إلى أفقي والعكس، يقوم النظام بتدمير الفعالية وانشائها مرة أخرى (وذلك بتنفيذ الدالة onDestroy() ثم تنفيذ الدالة onCreate() وذلك لتطبيق خصائص العرض الخاصة بالوضع الجديد للشاشة. في هذه الحالة، يجب أن تضمن استعادة حالة الفعالية وكل البيانات الخاصة بها عند تغيير اتجاه الجهاز، وذلك حتى لا يلاحظ المستخدم أي فقد للبيانات أو تغيير في حالة الفعالية.

أسئلة على الوحدة الأولى

1. ما المقصود بالفعالية (Activity)؟ ولماذا يعتبر من الضرورة إدارة وفهم دورة حياة الفعالية (Activity Life Cycle)؟
2. قم بتشغيل التطبيق الذي تم إنشاؤه في التمرين 1-1 ثم قم بتدوير الجهار لتغيير اتجاه العرض من أفقي إلى رأسي والعكس. ماذا تستنتج من تسلسل الرسائل التي يتم طباعتها؟
3. ما المقصود بالحافظة الخلفية Back stack في نظام أندرويد؟ وما وضع العمل للفعاليات المحفوظه بها؟.
4. لماذا يفضل استخدام الدالة onPause() لحفظ حالة الفعالية (Activity) على استخدام الدالة onSaveInstanceState()؟. قم بالتحقق من الفرق بين الدالتين عملياً.

الوحدة الثانية:

بنية التطبيق (Application Structure)

يتعلم الطالب في هذه الوحدة:

- ✓ . المكونات الأساسية لتطبيق أندرويد والفرق بينها.
- ✓ . ملف الوثيقة (Manifest File) ومكوناته وتعديله لتهيئة التطبيق للعمل بشكل صحيح.
- ✓ . أنواع المصادر (Resources) الموجودة ضمن تطبيق أندرويد والوصول إليها.
- ✓ .التعامل بشكل صحيح مع تغير الإعدادات مثل تغير جهة العرض (رأسي - أفقي) وشاشة العرض.

لدراسة هذه الوحدة لابد من الإلمام بمفهوم الفعالية (Activity) ودورة حياتها (إرجع إلى الوحدة الأولى)، كذلك لابد من الإلمام بتصميم واجهات التطبيق (Layouts) باستخدام XML.

مكونات تطبيق أندرويد

يتكون تطبيق أندرويد من عدة مكونات تشمل بعض أو كل ما يلي:

- الفعاليات (Activities): تستخدم الفعاليات لعمل الواجهات التفاعلية. تم شرح الفعالية (Activity) في الوحدة السابقة، وسيتم استخدامها بكثرة في التطبيقات الموجودة في هذا الكتاب.
- الخدمات (Services): الخدمة هي مكون يعمل في الخلفية لتنفيذ عمليات يحتاج تشغيلها لفترة طويلة.الخدمة (Service) لاتوفر واجهة للمستخدم حيث تعمل في الخلفية بدون تدخل من المستخدم. على سبيل المثال، يمكن تفعيل خدمة تشغيل الملفات الصوتية في الخلفية بينما يقوم المستخدم بالتفاعل مع تطبيق مختلف، أوخدمة تزيل بيانات عبرالشبكة دون عرقلة تفاعل المستخدم مع التطبيقات الأخرى. لن يتم التطرق لبرمجة الخدمات في هذا الكتاب.
- مزودات المحتوى (Content Providers): مزودالمحتوى يتحكم في مشاركة قواعد بيانات أو الملفات. يمكنك تخزين البيانات في نظام الملفات أو في قاعدة بيانات SQLite، والوصول إليها من خلال مزود المحتوى (Content Provider). يمكن أيضاً لتطبيقات أخرى الاستعلام أوحتى تعديل البيانات (إذا كان مزود المحتوى يسمح بذلك). على سبيل المثال، يوفر نظام أندرويد مزود المحتوى الذي يدير معلومات الاتصال الخاصة بالمستخدم (Contacts Content Provider). على هذا النحو،أي التطبيق مع الأذونات المناسبة يمكنه الاستعلام عن جزء من المعلومات الخاصة بشخص معين. سيتم التطرق لمزودات المحتوى (Content Providers) في الوحدة السابعة من هذا الكتاب.
- مستقبلات النشر (Broadcast Receivers): مستقبل النشر هو المكون الذي يستجيب للرسائل المرسلة من النظام أو التطبيقات الأخرى. على سبيل المثال، عند إنخفاض مستوى شحن البطارية أو إعادة تشغيل الجهاز، يقوم النظام ببث رسائل للإبلاغ عن هذا الحدث. يمكن لمستقبل النشر (Broadcast Receiver) الاستجابة لهذه الرسائل وتنفيذ إجراءات ما. مستقبل النشر (Broadcast Receiver) لا يشتمل على واجهة مستخدم،

ولكنه قد يرسل تنبيها لمستخدم (Notification) عند حدوث حدث ما. سيتم التطرق لمستقبلات النشر (Broadcast Receiver) في الوحدة الثامنة من هذا الكتاب.

ملف الوثيقة (Manifest File)

كل تطبيق أندرويد يجب أن يحتوي على ملف الوثيقة باسم AndroidManifest.xml في المجلد الأساسي للتطبيق. يحتوي ملف الوثيقة على معلومات أساسية عن التطبيق حتى يتمكن النظام من تشغيله بشكل صحيح. الكود بالأسفل يوضح مثال لملف وثيقة (Manifest) لتطبيق ما. يحتاج المطور في كثير من الأحيان إلى إجراء تعديلات في هذا الملف ليعمل التطبيق بالشكل المناسب.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="ps.edu.ucas.FirstAndroidApp"
    android:versionCode="1"
    android:versionName="1.0" >
<!--الصلاحيات-->
<uses-permission
android:name="android.permission.WRITE_INTERNAL_STORAGE" />
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-sdk
        android:minSdkVersion="11"
        android:targetSdkVersion="18" />

<application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
<!-- الفعاليات -->
<activity android:name="
ps.edu.ucas.FirstAndroidApp.MainActivity"
android:label="@string/main_activity_title"
android:screenOrientation="portrait" >
<intent-filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER"
/>
</intent-filter>
</activity>
```

```
</application>
</manifest>
```

سنقوم فيما يلي بشرح أهم المعلومات التي يحويها هذا الملف:

- وصف لمكونات التطبيق المختلفة من فعاليات (Activities) وخدمات (Services) وغيرها: كما ذكرنا في الوحدة السابقة، عند إنشاء أي فعالية يجب الإعلان عنها في ملف الوثيقة. أنظر المثال أعلاه حيث أن ملف الوثيقة يشتمل على فعالية باسم (MainActivity).
- الأذونات (Permission) والتي يحتاجها التطبيق من أجل الوصول إلى الأجزاء المحمية من النظام مثل الذاكرة الخارجية أو الاتصال بالانترنت.على سبيل المثال، هذه بعض الصلاحيات التي يتم إضافتها إلى ملف الوثيقة Manifest باستخدام الخاصية <uses-permission>:

```
<uses-permission
android:name="android.permission.WRITE_INTERNAL_STORAGE" />
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission
android:name="android.permission.INTERNET" />
```

حيث تمنح هذه الأذونات التطبيق إمكانية الوصول للذاكرتين الداخلية والخارجية والإنترنت على التوالي.

- الحد الأدنى و الأعلى لمستوى الواجهة البرمجية (API Level) التي يتطلبها التطبيق للعمل بشكل صحيح. API Level هو رقم يحدد إصدار الواجهة البرمجية التي توفرها منصة أندرويد (Android Platform) ويتم تحديده في ملف الوثيقة (Manifest) من خلال الخاصية <uses-sdk>. من الهام تحديد قيم هذه الخاصية لأنها تحدد أرقام الإصدارات المتوافقة مع التطبيق. هناك قيمتان ضمن الخاصية <uses-sdk> يجب تحديدهما وهما: android:minSdkVersion و android:targetSdkVersion. القيمة الأولى تحدد أدنى واجهة برمجية يحتاجها التطبيق ليعمل بشكل صحيح. يقوم نظام أندرويد تلقائياً برفض تنصيب التطبيق إذا كانت الواجهة البرمجية المستخدمة من قبل النظام أقل من تلك المحددة في الخاصية android:minSdkVersion. عند تحديد هذه الخاصية يجب مراعاة طبيعة الأجهزة المتوافقة مع التطبيق وانتشارها في السوق: فمثلاً تحديد قيمة عالية للواجهة البرمجية يعني أن التطبيق الخاص بك لن يعمل على كل الأجهزة التي تعمل بالواجهات الأقدم والتي قد يستخدمها عدد كبير من الأجهزة. في المقابل فإن استخدام قيمة متدنية جدا يحرمك من استخدام الخصائص البرمجية الحديثة المضافة للواجهات البرمجية الأحدث.

قيمة android:targetSdkVersion تحدد الواجهة البرمجية التي يستهدفها التطبيق ، وهي تعني أن التطبيق تم اختباره على الواجهة البرمجية المحددة وأنه ليس هناك حاجة لأن يقوم النظام بأي عملية مواءة للتطبيق إذا توافقت الواجهة البرمجية للنظام مع واجهة التطبيق. لا يزال التطبيق قابل للعمل على الأنظمة القديمة (حتى الرقم المحدد في android:minSdkVersion) ولكن قد تقوم الانظمة القديمة بعمل مواءمة تلقائية للتطبيق مما قد يؤثر على بعض الإضافات الغيرمدعومة من قبل النظام.

الجدول 1-2 يوضح اسم منصة أندرويد (Code Name)، رقم الإصدار (Version) ورقم الواجهة البرمجية (API Level).

جدول 1-2: اسم منصة أندرويد (Code Name)، رقم الإصدار (Version) ورقم الواجهة البرمجية (API Level)[1]

API Level	Version	Code Name
API level 21	5.0	Lollipop
API level 19	4.4 - 4.4.4	KitKat
API level 18	4.3.x	Jelly Bean
API level 17	4.2.x	Jelly Bean
API level 16	4.1.x	Jelly Bean
API level 15, NDK 8	4.0.3 - 4.0.4	Ice Cream Sandwich
API level 14, NDK 7	4.0.1 - 4.0.2	Ice Cream Sandwich
API level 13	3.2.x	Honeycomb
API level 12, NDK 6	3.1	Honeycomb
API level 11	3.0	Honeycomb
API level 10	2.3.3 - 2.3.7	Gingerbread
API level 9, NDK 5	2.3 - 2.3.2	Gingerbread
API level 8, NDK 4	2.2.x	Froyo
API level 7, NDK 3	2.1	Éclair
API level 6	2.0.1	Éclair
API level 5	2.0	Éclair
API level 4, NDK 2	1.6	Donut
API level 3, NDK 1	1.5	Cupcake
API level 2	1.1	(no code name)
API level 1	1.0	(no code name)

- اتجاه الشاشة (screen orientation) للفعالية والتي تكون في أحد وضعين: الوضع الرأسي (portrait) والوضع الأفقي (landscape). إذا أردت تثبيت وضع العرض لفعالية ما (Activity) بحيث لا تستجيب لتغير وضع الجهاز إن كان رأسياً أو أفقياً، يمكن القيام ذلك باستخدام الخاصية android:screenOrientation من داخل الخاصية activity كما هو موضح في المثال التالي:

```
android:screenOrientation="portrait"
```

كما هو الحال في ملف XML الخاص بالواجهات، يمكن من ملف الوثيقة الوصول للنصوص المعرفة في ملف strings.xml. فمثلاً، في ملف الوثيقة (Manifest) الموضح مسبقاً: الخاصية label الموجودة ضمن العنصر

[1] Android Developers, http://developer.android.com/guide/topics/manifest/uses-sdk-element.html

application تستخدم قيمة المتغير app_name المعرف في ملف strings.xml. بالمثل، الخاصية label الموجودة ضمن الخاصية activity تستخدم قيمة المتغير main_activity_title المعرف أيضاً في ملف strings.xml.

إنشاء المصادر (Creating Resources)

عند بناء تطبيق أندرويد، يوصى دائماً بإبقاء المصادر التي يتعامل معها التطبيق(Application Resources) منفصلة عن الكود. من أنواع المصادر التي يتعامل معها التطبيق: العبارات النصية strings، الصور images، أنماط التصميم styles، ملفات هيكلة الواجهات layout وغيرها. هيكلية تطبيق الأندرويد مشابهة لما هو موضح في شكل 1-2، حيث أن المجلد src مخصص لحفظ ملفات الجافا، بينما مجلد res يستخدم لحفظ الأنواع المختلفة من المصادر، حيث يخصص لكل نوع من المصادر مجلد خاص داخل المجلد res. فمثلاً الصور تحفظ في مجلد drawable بينما تحفظ العبارات النصية strings داخل الملف strings.xml داخل المجلد values.

```
MyProject/
    src/
        MyActivity.java
    res/
        drawable/
            icon.png
        layout/
            main.xml
            info.xml
        values/
            strings.xml
```

شكل 1-2: هيكلية تطبيق أندرويد

جدول 2-2 يوضح أهم مجلدات المصادر ونوعية البيانات التي تحفظ بها:

جدول 2-2: مجلدات المصادر (Resources) ضمن تطبيق أندرويد

مجلد المصدر	الاستخدام
drawable/	يحتوي على ملفات الصور
layout/	يحتوي على ملفات XML التي تحدد تصميم هيكلية الواجهات layout
menu/	يحتوي على ملفات XML التي تحدد تركيب القوائم المستخدمة في التطبيق
values/	يحتوي على ملفات XML تحدد قيم يتم استخدامها في التطبيق مثل الألوان والأرقام والعبارات النصية.
drawable/	يحتوي على ملفات الصور
layout/	يحتوي على ملفات XML التي تحدد تصميم هيكلية الواجهات layout

ملاحظة: المصادر التي يتم حفظها في المجلد res هي المصادر الإفتراضية التي يتم استخدامها في التطبيق. أحياناً قد تحتاج إلى مصادر مختلفة لتتلائم من الأوضاع المختلفة لعمل التطبيق. على سبيل المثال، التطبيق قد يعمل على أجهزة مختلفة الحجم، وبناءً على ذلك فإن تصميم الواجهات وأنواع الصور المستخدمة قد يختلف من جهاز لآخر. لتصميم التطبيق بحيث يتلاءم مع ظروف العرض المختلفة، لا بد من حفظ مصادر بديلة بالإضافة إلى تلك الإفتراضية ليتم استخدامها عند الحاجة لذلك. عند تشغيل التطبيق، يقوم نظام أندرويد تلقائياً باختيار المصادر المناسبة بما يتناسب مع إعدادات الجهاز.

لتحديد مصادر بديلة لإعدادات الجهاز المختلفة، يجب اضافة مجلد داخل المجلد res بالصيغة التالية:

`<resources_name>-<config_qualifier>`

حيث <resources_name> هو اسم مجلد المصدر مثل layout، values، drawable. <qualifier> يمثل الإعدادات التي يستخدم فيها المصدر. على سبيل المثال، في مجلد المصادر الموضح:

```
res/
    drawable/
        icon.png
        background.png
    drawable-hdpi/
        icon.png
        background.png
```

لاحظ أن المجلد drawable-hdpi يحوي الصور التي يتم استخدامها للشاشات ذات الدقة العالية وهو ما يعنيه المقطع hdpi.عند تشغيل التطبيق، يقوم النظام تلقائياً باختيار مجلد الصور المناسب لإعدادات الجهاز: حيث يتم استخدام الصور من المجلد drawable-hdpi إذا كان الجهاز ذو شاشة عالية الدقة، بينما يتم استخدام الصور من المجلد drawable لغير ذلك من الأجهزة.

مثال آخر على ذلك هو تحديد تصميم مختلف للواجهة إذا اختلف اتجاه الشاشة (Orientation) أو لغة الجهاز. في هذه الحالة يتم إنشاء مجلد باسم layout-port يحوي تصميم الواجهات للوضع الرأسي (portrait) أو مجلد باسم layout-land يحوي تصميم الواجهات للوضع الأفقي أو layout-ar ليحوي تصاميم واجهات متناسبة مع اللغة العربية.

الوصول للمصادر (Accessing Resources)

بعد إنشاء المصادر، قد تحتاج للوصول إليها أثناء بناء التطبيق. هناك طريقتين للوصول للمصادر: الوصول من خلال الكود أو من خلال XML.

- الوصول من خلال الكود:

كل المصادر التي يتم إنشاؤها يقوم النظام تلقائياً بإنشاء صيغ معرفة لها ID وحفظها في ملف خاص اسمه R. الصيغ المعرفة تأخذ الشكل التالي:

R.<resource_type>.<resource_name>

حيث <resource_type> يمثل نوع المصدر مثل string، layout، drawable وغيره، بينما <resource_name> تعني اسم المصدر. فمثلاً، المعرف التالي: R.string.activity_title يستخدم للوصول للعبارة النصية string التي تحمل اسم activity_title. المعرف R.drawable.background يستخدم للوصول لملف الصورة الذي يحمل اسم background داخل المجلد drawable. الأمثلة التالية توضح كيفية الوصول إلى المصادر برمجياً من خلال الكود:

مثال 1:الوصول إلى ملف صورة باسم myimage لعرضها في عنصر واجهة من نوع ImageView

```
imageView.setImageResource(R.drawable.myimage);
```

مثال 2: تغيير عنوان الفعالية (Activity) باستخدام العبارة النصية title

```
getWindow().setTitle(getResources().getText(R.string.title));
```

مثال 3: تحديد ملف هيكلية الواجهة باسم main_screen عند إنشاء الفعالية

```
setContentView(R.layout.main_screen);
```

- الوصول من خلال XML

عند إنشاء ملفات XML للواجهات وغيرها، يمكن اعطاء قيم لعناصر XML باستخدام قيم معرفة في ملفات أخرى. الصيغة العامة للوصول لأي مصدر من خلال XML تأخذ الشكل التالي:

@<resource_type>/<resource_name>

حيث <resource_type> يمثل نوع المصدر مثل string، layout، drawable وغيره، بينما <resource_name> تعني اسم المصدر.

المثال التالي يوضح كيف يتم إعطاء قيمة للنص المعروض على الزر Button باستخدام عبارة نصية string معرفة باسم submit (أنظر الخاصية android:text):

```
<Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/submit" />
```

المثال التالي يوضح إعطاء قيم للعنصر EditText باستخدام مصادر معرفة في أماكن أخرى (أنظر الخاصيتين: android:textColor و android:text):

```
<EditText
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:textColor="@color/opaque_red"
    android:text="@string/title" />
```

مجلد الممتلكات (Assets Folder)

يوفر نظام أندرويد مجلد آخر ضمن مجلدات التطبيق حيث يمكن حفظ الملفات المختلفة. يسمى هذا المجلد assets. الفرق بين مجلد assets والمجلد res أن النظام لا يقوم بإنشاء أرقام معرفة ID للملف الموجودة في المجلد assets. يستخدم مجلد assets لحفظ الملفات التي لايمكن حفظها في ملف res مثل ملفات بالامتداد txt، ملفات الصوت (.wav, .mp3, .mid) وغيرها. يتم الوصول إلى الملفات في مجلد assets عن طريق الكائن AssetManager والذي يمكن الوصول إليه بتطبيق الدالة getAssets() داخل الفعالية (Activity). يوفر AssetManage عدة دوال للتعامل مع الملفات مثل list() لاستعراض الملفات و open() لفتح ملف ما. الكود التالي يوضح كيفية قراءة محتوى ملف text.txt موجود في مجلد assets وطباعته كرسالة.

```
public class AssetsActivity extends Activity {

     @Override
     protected void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
          try {
               InputStream is =
getAssets().open("text.txt");
               int size = is.available();
               byte[] buffer = new byte[size];
               is.read(buffer);
               is.close();
               String content = new String(buffer);
               Toast.makeText(getApplicationContext(),
content, Toast.LENGTH_LONG).show();
          } catch (IOException e) {
               e.printStackTrace();
          }
     }
}
```

التعامل مع تغيير الإعدادات (Handling Configuration Changes)

إعدادات الجهاز قد تتغير في أي وقت مثل تغير اتجاه العرض (أفقي – رأسي) أو تغير اللغة. عندما تحصل مثل هذه التغييرات، يقوم نظام أندرويد بإلغاء الفعالية الحالية (Activity) وإنشاء واحدة جديدة عن طريق تنفيذ الدالة onDestroy() ومن ثم onCreate(). إعادة تشغيل الفعالية يهدف إلى جعل التطبيق يتكيف مع التغيير الجديد باستخدام المصادر والواجهات التي تتلاءم مع الإعدادات الجديدة. عند إعادة تشغيل الفعالية يمكن استرجاع حالتها وبياناتها عن طريق تطبيق الدالة onSaveInstanceState() أو onPause() لحفظ حالة الفعالية قبل إيقافها، ومن ثم تطبيق الدالة onRestoreInstanceState() أو onStart() لاسترجاع البيانات.

يمكن تثبيت بعض الإعدادات للتطبيق بحيث لا يتم إعادة تشغيل الفعالية (Activity) تلقائياً عند تغيره. فمثلاً، يمكن إضافة الخاصية android:screenOrientation ضمن خصائص الفعالية (Activity) في ملف الوثيقة Manifest وذلك لتثبيت طريقة عرض التطبيق في وضع رأسي (أو أفقي) مهما تغير وضع الجهاز. كذلك من الممكن إيقاف إعادة تشغيل الفعالية عند حصول تغيير بالإعدادات بحيث يتم التعامل مع هذا التغيير برمجياً. يتم ذلك بإضافة الخاصية android:configChanges ضمن خصائص الفعالية (Activity) في ملف الوثيقة (Manifest) كما هو موضح:

```
<activity android:name=".MyActivity"
android:configChanges="orientation|keyboardHidden"
          android:label="@string/app_name">
```

في هذه المثال يتم التصريح بأن الفعالية ستتولى التعامل برمجياً مع تغيير اتجاه العرض orientation أو في حالة إخفاء لوحة المفاتيح، وذلك دون الحاجة لإعادة تشغيل الفعالية. في هذه الحالة يجب تطبيق الدالة onConfigurationChanges() داخل الفعالية والتي يتم تنفيذها تلقائياً عند حصول أي تغيير في الإعدادات. يمكن كتابة كود في هذه الدالة لتنفيذ الإجراء المطلوب كما بالمثال التالي:

```
@Override
public void onConfigurationChanged(Configuration
newConfig){
    super.onConfigurationChanged(newConfig);

    // Checks the orientation of the screen
if(newConfig.orientation
==Configuration.ORIENTATION_LANDSCAPE){

Toast.makeText(this,"landscape",Toast.LENGTH_SHORT).show();
}elseif(newConfig.orientation
==Configuration.ORIENTATION_PORTRAIT){

Toast.makeText(this,"portrait",Toast.LENGTH_SHORT).show();
    }
}
```

في هذا المثال أعلاه يتم عرض رسالة تبين اتجاه عرض التطبيق وذلك عند حصول أي تغيير في الإعدادات. لاحظ أنه يتم تمرير الإعدادات التي تغيرت إلى الدالة.

في الغالب لن تحتاج لتطبيق هذه الدالة حيث يفضل إعادة تشغيل الفعالية (Activity) عن تغير الإعدادات وذلك حتى يتم استخدام المصادر المناسبة للاعدادات الجديدة تلقائياً. على سبيل المثال، عن وجود تصميم واجهات خاص في وضع العرض الرأسي portrait في مجلد layout-port، يقوم النظام تلقائياً باستخدام هذه الواجهات إذا تم تغيير اتجاه العرض للوضع الرأسي.

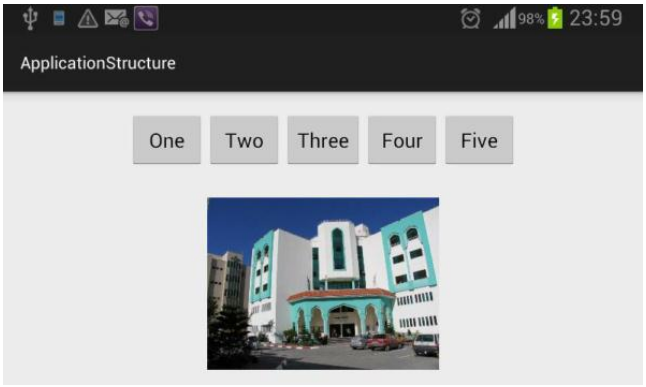
كذلك يمكن فحص إعدادات الجهاز برمجياً من داخل الفعالية (Activity) باستخدام الدالة getResources().getConfiguration(). المثال التالي يوضح كيفية معرفة اتجاه العرض برمجياً عند بدء تشغيل الفعالية (Activity):

```
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  if(this.getResources().getConfiguration().orientation ==
Configuration.ORIENTATION_LANDSCAPE){
     Toast.makeText(this, "landscape",
Toast.LENGTH_SHORT).show();
  }else
if(this.getResources().getConfiguration().orientation ==
Configuration.ORIENTATION_PORTRAIT){
     Toast.makeText(this, "portrait",
Toast.LENGTH_SHORT).show();
  }
…
}
```

تمرين عملي (1-2)

يهدف هذا التطبيق البسيط إلى التمرن على بعض المفاهيم الواردة في هذا الفصل مثل تغيير إعدادات التطبيق من خلال ملف الوثيقة (Manifest)، الوصول للمصادر وتحديد مصادر بديلة لتلائم الإعدادات المختلفة للتطبيق.

التطبيق له واجهتين كما هو موشح بشكل 2-2 : الواجهة الأفقى تعمل في وضع العرض الأفقي حيث تتكون من مجموعة من الأزرار المعروضة أفقياً وأسفلها صورة لمبنى الكلية الجامعية. الواجهة الثانية تعمل في وضع العرض الرأسي وفيها يتم عرض الأزرار رأسياً، ويتم عرض صورة لشعار الكلية الجامعية. يتم اختيار الواجهة وتغيير الصورة تلقائياً بناء على وضع العرض.



شكل 2-2: واجهتا التطبيق في الوضعين الرأسي (Portrait) والأفقي (Landscape)

الخطوات التالية توضح مراحل بناء التطبيق:

أولاً: إنشاء الواجهات: بناء على أن التطبيق له واجهتين مختلفيتين، نقوم بإنشاء ملفي XML لتصميم الواجهة: أحد الملفين مخصص للواجهة بالوضع الرأسي ونضعه في المجلد layout، والملف الآخر يحمل نفس الاسم ولكن في مجلد آخر بالاسم layout-land. لاحظ أن اسم المجلد يحدد الإعدادات التي يستخدم عندها ملف الواجهة، فالمجلد الذي ينتهي اسمه بـ land يستخدم في وضع العرض الأفقي (Landscape) بينما يستخدم المجلد الآخر في غير ذلك. ملفات التصميم موضحة بالأسفل:

أولاً: تصميم الواجهة الخاصة بالوضع الرأسي

```
// layout/activity_main.xml
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"

android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
```

```
tools:context="com.example.applicationstructure.MainActivit
y" >
<Button
        android:id="@+id/buttonOne"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:text="@string/buttonOne" />
<Button
        android:id="@+id/buttonTwo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_below="@+id/buttonOne"
        android:text="@string/buttonTwo" />
<Button
        android:id="@+id/buttonThree"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_below="@+id/buttonTwo"
        android:text="@string/buttonThree" />
<Button
        android:id="@+id/buttonFour"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_below="@+id/buttonThree"
        android:text="@string/buttonFour" />
<Button
        android:id="@+id/buttonFive"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_below="@+id/buttonFour"
        android:text="@string/buttonFive" />
<ImageView
        android:id="@+id/imageView"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:layout_below="@+id/buttonFive"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="24dp"/>
</RelativeLayout>
```

ثانياً: تصميم الواجهة الخاصة بالوضع الأفقي

```
// layout-land/activity_main.xml
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"

tools:context="com.example.applicationstructure.MainActivit
y" >
<Button
        android:id="@+id/buttonOne"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toLeftOf="@+id/buttonTwo"
        android:text="@string/buttonOne" />
<Button
        android:id="@+id/buttonTwo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/buttonOne"
        android:layout_toLeftOf="@+id/buttonThree"
        android:text="@string/buttonTwo" />
<Button
        android:id="@+id/buttonThree"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/buttonOne"
        android:layout_centerHorizontal="true"
        android:text="@string/buttonThree" />
<Button
```

```
        android:id="@+id/buttonFour"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_alignBottom="@+id/buttonOne"
        android:layout_toRightOf="@+id/buttonThree"
        android:text="@string/buttonFour" />
<Button
        android:id="@+id/buttonFive"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/buttonOne"
        android:layout_toRightOf="@+id/buttonFour"
        android:text="@string/buttonFive" />
<ImageView
        android:id="@+id/imageView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/buttonTwo"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="24dp"/>
</RelativeLayout>
```

ثانياً: إنشاء الفعالية (Activity): الكود بالأسفل يوضح الفعالية (MainActivity) المستخدمة في التطبيق:

```
1:  public class MainActivity extends Activity {
2:
3:  @Override
4:  protected void onCreate(Bundle savedInstanceState) {
5:     super.onCreate(savedInstanceState);
6:     setContentView(R.layout.activity_main);
7:     ImageView image = (ImageView)
8:     this.findViewById(R.id.imageView);
9:    if(this.getResources().getConfiguration().orientation
10:    == Configuration.ORIENTATION_LANDSCAPE)
11:       image.setImageResource(R.drawable.ucas_building);
12:   else if(this.getResources().getConfiguration().
13:   orientation == Configuration.ORIENTATION_PORTRAIT)
14:       image.setImageResource(R.drawable.ucas_logo);
15:   }
16: }
```

لاحظ أنه في الدالة onCreate() (عند بداية تشغيل الفعالية) يتم إنشاء الواجهة عن طريق الدالة setContentView() والتي يمرر لها المعرف الخاص بملف الواجهة وهو R.layout.activity_main (أنظر سطر رقم 6). لاحظ أيضاً أن هذا المعرف يمكن استخدامه للوصول لكلا ملفي الواجهة. يقوم النظام تلقائياً باختيار الملف المناسب بناءً على وضع العرض: فإذا كان وضع العرض أفقياً (Landscape) يتم استخدام ملف الواجهة activity_main.xml الموجود في المجلد layout-land، بينما إذا كان وضع العرض رأسياً (Portrait) يتم استخدام ملف الواجهة activity_main.xml الموجود في المجلد layout.

بعد ذلك يتم فحص وضع العرض برمجياً عن طريق الدالة getResources().getConfiguration() وبناءً عليه يتم تغيير الصورة (أنظر الكود من سطر 9 إلى 15). لاحظ أيضاً أن الوصول للصورة تم من خلال المعرف:R.drawable.<image_name>.

أسئلة على الوحدة الثانية

1. ما الفرق بين الفعالية (Activity) والخدمة (Service)؟ وفيم تستخدم كل منهما؟
2. ما الفرق بين استخدام المجلد Assets والمجلد res لحفظ المصادر؟
3. قم بإنشاء تطبيق أندرويد لعرض نص محدد باللغة العربية أو الإنجليزية في عنصر من نوع TextView وذلك بناءً على اللغة المحددة في إعدادات الجهاز. فمثلاً، عند تشغيل التطبيق يتم عرض جملة " The University College of Applied Sciences" إذا كانت اللغة المحددة في إعدادات الهاتف هي الإنجليزية. وعند تغيير اللغة إلى العربية من إعدادات الجهاز، يتم عرض الجملة "الكلية الجامعية للعلوم التطبيقية" تلقائياً.

   تلميح: استخدم مجلدين واحد باسم values-en والآخر باسم values-ar وذلك لتحديد نص إنجليزي وآخر عربي في ملف strings.xml الموجود في كلا المجلدين.
4. قم بالتعديل على التطبيق المنشأ في 2 وذلك لتغيير حجم الخط للنص المعروض تلقائيا عند عرضه باستخدام جهاز ذو شاشة كبيرة.

الوحدة الثالثة:

التعامل مع عناصر الواجهة برمجياً

(Interacting with UI Components Programmatically)

يتعلم الطالب في هذه الوحدة:

- ✓ معالجة أحداث التفاعل مع واجهة المستخدم.
- ✓ الوصول لعناصر واجهة المستخدم برمجياً والتعامل مع أنواع عناصر الواجهة المختلفة.
- ✓ التعامل مع قائمة العرض (ListView) وتعبئتها باستخدام أنواع المحولات المختلفة (Adapter).

لدراسة هذه الوحدة لابد من الإلمام بمفهوم الفعالية (Activity) ودورة حياتها (إرجع إلى الوحدة الأولى)، التعامل مع المصادر (إرجع إلى الوحدة الثانية)، كذلك لابد من الإلمام بعناصر الواجهة الأساسية وكذلك تصميم واجهات التطبيق (Layouts) باستخدام XML.

درست في مساق سابق إنشاء عناصر الواجهة المختلفة مثل TextView، EditText، Button وغيرها من العناصر. كذلك تعلمت طرق الهيكلة المختلفة Layouts لتنظيم عرض عناصر الواجهة (لمراجعة هذه المفاهيم يمكن الرجوع إلى الفصل الثالث في المرجع رقم 1 أو الفصل الثالث في المرجع رقم 7). سنتعلم الآن طريقة الوصول لعناصر الواجهة برمجياً والتفاعل مع الأحداث UI Events. كذلك سنتطرق للتعامل مع بعض عناصر الواجعة شائعة الإستخدام مثل القائمة (ListView) وطريقة عرض البيانات المختلفة بها باستخدام المحولات Adapters.

معالجة أحداث الوجهة برمجياً (Handling UI Events)

يقصد بأحداث واجهة المستخدم UI Events هي الأحداث الناتجة عن تفاعل المستخدم مع مكونات الواجهة مثل نقر الأزرار أو لمس الشاشة.

للتعامل مع أي حدث يحصل على أي عنصر في الواجهة يجب أولا الوصول لهذا العنصر من داخل الفعالية، وذلك يتم باستخدام الدالة findViewById() والتي يمرر لها الرقم المعرف للعنصر (له صيغة: R.id.<element_name>. لكل عنصر في الواجهة رقم خاص يختلف عن أي معرف للعناصر الأخرى. يتم إنشاء هذا المعرف تلقائياً بعد إدارج العنصر في ملف التصميم layout.xml. تقوم دالة findViewById() بإرجاع مؤشر لكائن من نوع View وهو نوع فئة الأب (Superclass type) لكل عناصر الواجهة. نقوم بعمل casting للعنصر المرجع وذلك لتحويله للنوع المطلوب. فمثلاً، للوصول للزر button1 يمكن استخدام الأمر التالي من داخل الفعالية (Activity). بعد الحصول على مؤشر للكائن الخاص بالزر، يمكن التعامل معه باستخدام الدوال المختلفة التي توفرها الفئة Button Class.

```
Button button = (Button) this.findViewById(R.id.button1);
button.setText("Click here");
```

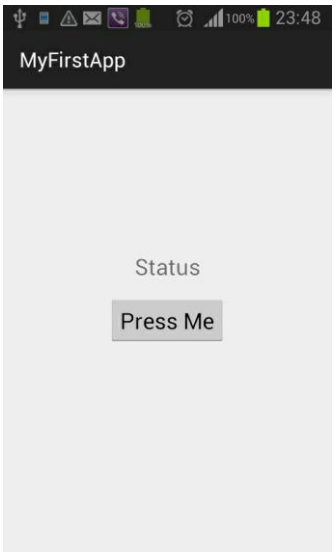
بعد الوصول لعناصر الواجهة برمجياً، يمكن التقاط الحدث (تفاعل المستخدم مع العنصر) أول وقوعه عن طريق تسجيل ما يسمى بالمستمع (Listener) والذي يعلمك تلقائياً بحصول حدث ما بتنفيذ دالة راجعة callback method. الفئة View، والتي تتفرع منها كل فئات عناصر واجهة المستخدم، تحتوي على مجموعة من الواجهات البرمجية (Interface) .كل واجهة برمجية (Interface) تحتوي على دوال مجردة والتي يمكن تطبيقها حسب الحاجة للاستماع للأحداث المختلفة.

الواجهة البرمجية (Interface) تحتوي فقط على دوال مجردة دون أي تطبيق، وعلى المستخدم تطبيق هذه الدوال بكتابة الكود المناسب. للتفاعل مع حدث معين على عنصر واجهة، يجب تسجيل واجهة مستمع جديدة لدى عنصر الواجهة وتطبيق الدوال الموجودة به لتنفيذ الإجراء المطلوب. فمثلاً، للاستجابة لحدث على زر معين Button، يجب على تسجيل واجهة مستخدم من نوع View.OnClickListener لدى الكائن Button (عن طريق تنفيذ الدالة setOnClickListener () ضمن الفئة Button، ومن ثم كتابة الكود الخاص بالدالة المجردة onClick() لتحديد الإجراء المطلوب تنفيذه عند الحدث). عند النقر على الزر يقول النظام تلقائياً بتنفيذ الدالة onClick() لينفذ الإجراء المطلوب.

```
button.setOnClickListener(
     new Button.OnClickListener() {
          public void onClick(View v) {
          // Action to be performed when button is clicked
          }
     }
);
```

تمرين عملي (1-3)

لتوضيح كيفية التعامل مع الأحداث بمثال عملي، سنقوم بعرض الخطوات الكاملة لتطبيق بسيط واجهته موضحة بالشكل. تتكون الواجهة من زر مكتوب عليه "Press Me" وعنصر عرض من نوع TextView. الحدث الذي نود تنفيذه هو تغيير محتوى النص المعروض في TextView عند النقر على الزر PressMe.



شكل1-3 : واجهة التطبيق الخاص بالتمرين 1-3

ملف هيكلية الواجهة لهذا المثال موضح بالأسفل، حيث أنه يحدد خصائص العنصر Button والمعرف بالإسم myButton، والعنصر TextView والمعرف بالاسم myTextView.

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myLayout"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >
<Button
        android:id="@+id/myButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="@string/mybutton_string" />
<TextView
        android:id="@+id/myTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_above="@+id/myButton"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="41dp"
        android:text="@string/mytextview_string"
```

```
android:textAppearance="?android:attr/textAppearanceLarge"
/>
</RelativeLayout>
```

بعد تصميم الواجهة، يجب التعديل على كود الفعالية (Activity) وذلك لتسجيل مستمع للحدث (Listener) ليقوم بتنفيذ الإجراء المطلوب عن النقر على الزر. كود الفعالية المعدل موضح بالأسفل:

```
1:  public class MainActivity extends Activity {
2:  @Override
3:  protected void onCreate(Bundle savedInstanceState) {
4:     super.onCreate(savedInstanceState);
5:     setContentView(R.layout.activity_main);
6:  }
7:  @Override
8:  protected void onStart(){
9:     super.onStart();
10:  Button button = (Button)findViewById(R.id.myButton);
11:     button.setOnClickListener(
12:       new Button.OnClickListener() {
13:            public void onClick(View v) {
14:              TextView myTextView = (TextView)
15:              findViewById(R.id.myTextView);
16:              myTextView.setText("Button clicked");
17:            }
18:       }
19:     );
20:  }
21: }
```

لاحظ أن الكود المسؤول عن التعامل مع الحدث تمت كتابته في الدالة ()onStart (أنظر سطر رقم 9). تم اختيار الدالة ()onStart لأنه عند تنفيذها يكون قد اكتمل إنشاء عناصر الواجهة ومن ثم أصبح في الإمكان الوصول للعناصر الموجودة بها عن طريق الدالة ()View.findViewById(أنظر سطر رقم 15). إنشاء واجهة المستخدم يتم في الدالة ()onCreate، وتصبح الفعالية (Activity) مرئية عند تنفيذ الدالة ()onStart (راجع دورة حياة الفعالية وتسلسل تنفيذ دوالة الاتصال الراجع callback). يمكن كتابة الكود أيضاً في الدالة ()onCreate ولكن بعد تنفيذ الدالة ()setContentView والمسؤولة عن إنشاء الواجهة.

بعد الوصول إلى الكائن myButton باستخدام الدالة ()findViewById، نضيف المستمع (Listener) للكائن myButton بتنفيذ الدالة () setOnClickListener، ونضيف الكود الخاص بالإجراء المطلوب داخل الدالة

onClick(). الإجراء المطلوب تنفيذه عند النقر على الزر هو تعديل النص المعروض في عنصر الواجهة myTextView ليصبح "Button Clicked". لتنفيذ هذا الإجراء يتم أولاً الوصول العنصر myTextView باستخدام الدالة findViewById()، ثم نعدل النص المعروض بتنفيذ الدالة setText().

التعامل مع الحدث عن طريقة ملف Layout

يمكن تنفيذ حدث معين عند النقر على الزر بطريقة أخرى وذلك بتعديل ملف هيكلية الواجهة layout بإضافة الخاصية android:onClick وإعطائها قيمة تمثل اسم الدالة التي سيتم تنفيذها تلقائياً عن النقر على الزر كما هو موضح في المثال التالي:

```
<Button
        android:id="@+id/myButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="@string/mybutton_string"
     android:onClick="myButtonClicked" />
```

بعد تعديل ملف الواجهة، يجب التعديل في ملف الفعالية (Activity) بإضافة الدالة myButtonClicked كما هو موضح بالمثال التالي: (يجب أن يكون اسم الدالة مطابق للإسم المحدد في ملف Layout).

```
public void myButtonClicked(View view){
TextView myTextView = (TextView)
findViewById(R.id.myTextView);
myTextView.setText("Button clicked");
}
```

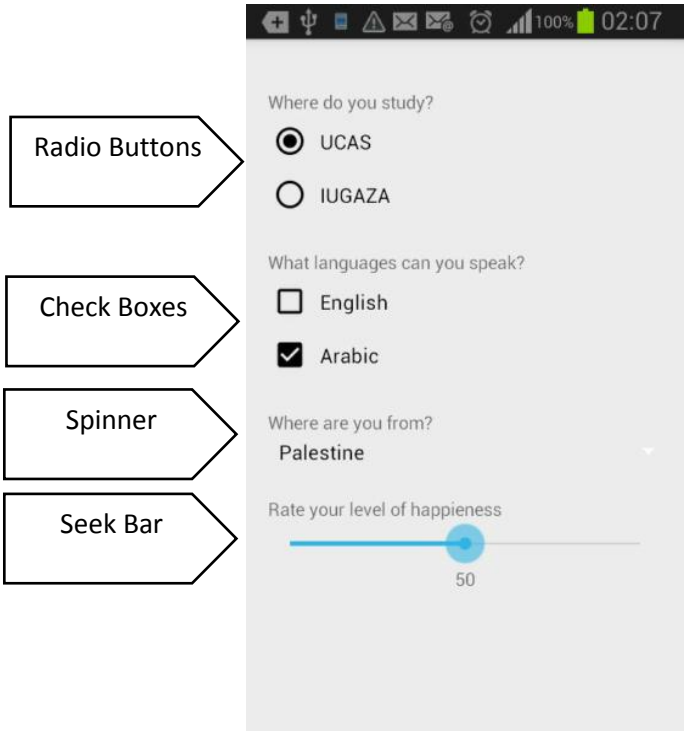
ملاحظة: يفضل استخدام الطريقة الأولى للاستماع لأحداث عناصر الواجهة وهي بإضافة المستمع (Listener) برمجياً وذلك لأنها تسمح بلاستماع لأحداث متنوعة عن طريقة دوال مختلفة مثل setOnClickListener()، setOnLongClickListener()، setOnTouchListener() وغيرها. كما أن هذه الطريقة تضمن فصل كامل ما بين مرحلة تصميم الواجهة، وهو ما يتم من خلال ملف هيكلية الواجهة، وبين مرحلة التعامل مع الأحداث والذي يتم برمجياً من خلال الفعالية (Activity).

في النهاية، نستعرض بعد مستمعات الأحداث (Event Listeners) التي توفرها عناصر الواجهة والتي يمكن استخدامها للاستماع لأنواع مختلفة من الأحداث، ومن ضمنها:

- onClickListener: ويستخدم لتنفيذ إجراء عند النقر ثم ترك عنصر الواجهة View. يتم كتابة كود الإجراء في الدالة ()onClick.
- onLongClickListener: ويستخدم لتنفيذ إجراء عند النقر لفترة على عنصر الواجهة. يتم كتابة كود الإجراء في الدالة ()onLongClick.
- onKeyListener: ويستخدم لتنفيذ إجراء عند الضغط على أحد المفاتيح في الجهاز بينما عنصر الواجهة View مفعّل من قبل المستخدم (has focus). يتم كتابة كود الإجراء في الدالة ()onKey.

تمرين عملي (2-3)

يتطرق هذا التمرين إلى عناصر واجهة متنوعة مثل زر الإنتقاء (RadioButton)، خانة الإختيار (CheckBox)، القائمة المنسدلة (Spinner) والشريط المنزلق (SeekBar) وكيفية معالجة الأحداث لهذه العناصر برمجياً وأنواع المستمعات (Listeners) المختلفة لهذا الغرض. الواجهة الخاصة بهذا التطبيق وأنواع العناصر المضافة موضحة في شكل 2-3.



شكل 2-3 : واجهة التطبيق الخاص بالتمرين 2-3

لتحديد مصفوفة النصوص (Stirng Array) التي يجب تعبئتها في القائمة المنزلقة (Spinner)، أضف المصفوفة countries_array إلى ملف strings كما هو موضح بالأسفل:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="app_name">UI Events</string>
<string name="action_settings">Settings</string>
<string-array name="countries_array">
<item>Palestine</item>
<item>Egypt</item>
<item>Algeria</item>
<item>Syria</item>
<item>Jordan</item>
<item>Saudi Arabia</item>
<item>Iraq</item>
</string-array>
</resources>
```

ملف تصميم الواجهة Layout الخاصة بهذا التمرين موضح بالأسفل:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"

android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.uievents.MainActivity" >
<TextView
    android:id="@+id/studyTextView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
     android:layout_marginTop="20dp"
    android:text="Where do you study?" />
<RadioGroup
```

```
    android:id="@+id/radioGroup"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" >
<RadioButton
        android:id="@+id/ucasRadio"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="UCAS" />
<RadioButton
        android:id="@+id/iugRadio"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="IUGAZA" />
</RadioGroup>
<TextView
    android:id="@+id/languageTextView"
    android:layout_marginTop="20dp"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="What languages can you speak?" />
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical" >
<CheckBox
        android:id="@+id/englishCheckBox"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="English" />
<CheckBox
    android:id="@+id/arabicCheckBox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Arabic" />
</LinearLayout>
<TextView
    android:id="@+id/CountryTextView"
    android:layout_marginTop="20dp"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

```
    android:text="Where are you from?" />
<Spinner
    android:id="@+id/countiresSpinner"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:entries="@array/countries_array"
     />
<TextView
    android:id="@+id/happienessTextView"
    android:layout_marginTop="20dp"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Rate your level of happieness" />
<SeekBar
    android:id="@+id/happienessSeekBar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
<TextView
    android:id="@+id/HappienessValue"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:text="" />
</LinearLayout>
```

لاحظ أن أزراء الانتقاء Radio Buttons تم جمعها في مجموعة واحدة RadioGroup وذلك حتى يتم اختيار عنصر واحد منها فقط. لاحظ أيضاً القائمة spinner وكيف تم تعبئتها بالمصفوفة المحددة مسبقاً في المصادر (ملف strings.xml) وذلك باستخدام الخاصية "android:entries="@array/countries_array (يمكن تعبئة القائمة spinner برمجياً باستخدام المحول Adapter والذي سيتم الحديث عنه لاحقاً في هذا الفصل). في أسفل الواجهة تم استخدام عنصر عرض TextView وذلك لعرض القيمة الراجعة من العنصر SeekBar. بعد تجهيز الواجهة، سيتم الآن معالجة أحداثها برمجياً داخل الفعالية (MainActivity) كما هو موضح:

```
1:  public class MainActivity extends Activity {
2:
3:  @Override
4:  protected void onCreate(Bundle savedInstanceState) {
5:     super.onCreate(savedInstanceState);
6:     setContentView(R.layout.activity_main);
7:  }
8:
9:  @Override
```

```
10: protected void onStart() {
11:    super.onStart();
12:    RadioGroup radioGroup = (RadioGroup)
13:    this.findViewById(R.id.radioGroup);
14:    radioGroup.setOnCheckedChangeListener(new
15:    android.widget.RadioGroup.OnCheckedChangeListener(){
16:
17:    @Override
18:    public void onCheckedChanged(RadioGroup group, int
19:    checkedId) {
20:       if(checkedId == R.id.ucasRadio)
21:          Toast.makeText(getApplicationContext(),
22:          "Choice : UCAS", Toast.LENGTH_SHORT).show();
23:       else if(checkedId == R.id.iugRadio)
24:          Toast.makeText(getApplicationContext(),
25:          "Choice : IUG", Toast.LENGTH_SHORT).show();
26:    }
27:  });
28:  CheckBox englishCheckBox = (CheckBox)
29:  this.findViewById(R.id.englishCheckBox);
30:  englishCheckBox.setOnCheckedChangeListener(new
31:  OnCheckedChangeListener(){
32:     @Override
33:     public void onCheckedChanged(CompoundButton
34:     buttonView, boolean isChecked) {
35:        Toast.makeText(getApplicationContext(),
36:        buttonView.getText()+" is "+ isChecked,
37:        Toast.LENGTH_SHORT).show();
38:    }
39:  });
40:  CheckBox arabicCheckBox = (CheckBox)
41:  this.findViewById(R.id.arabic(CheckBox));
42:  arabicCheckBox.setOnCheckedChangeListener(new
43:  OnCheckedChangeListener(){
44:    @Override
45:      public void onCheckedChanged(CompoundButton
46:      buttonView, boolean isChecked) {
47:
48:         Toast.makeText(getApplicationContext(),
49:         buttonView.getText()+" is "+ isChecked,
```

```
50:          Toast.LENGTH_SHORT).show();
51:       }
52:   });
53:
54:   Spinner spinner = (Spinner)
55:   this.findViewById(R.id.countiresSpinner);
56:   spinner.setOnItemSelectedListener(new
57:   OnItemSelectedListener(){
58:
59:     @Override
60:     public void onItemSelected(AdapterView<?> parent,
61:     View view, int position, long id) {
62:        Toast.makeText(getApplicationContext(),
63:        parent.getItemAtPosition(position).toString(),
64:        Toast.LENGTH_SHORT).show();
65:     }
66:     @Override
67:     public void onNothingSelected(AdapterView<?>
68:     parent){}
69:   });
70:   final TextView happienessValue = (TextView)
71:   this.findViewById(R.id.HappienessValue);
72:   SeekBar seekBar = (SeekBar)
73:   this.findViewById(R.id.happienessSeekBar);
74:   seekBar.setOnSeekBarChangeListener(new
75:   OnSeekBarChangeListener(){
76:     @Override
77:     public void onProgressChanged(SeekBar seekBar, int
78:     progress, boolean fromUser) {
79:       happienessValue.setText(String.valueOf(progress));
80:     }
81:
82:     @Override
83:     public void onStartTrackingTouch(SeekBar seekBar){}
84:
85:     @Override
86:     public void onStopTrackingTouch(SeekBar seekBar){}
87:   });
88: }
89:}
```

يتم معالجة الأحداث لكل عنصر كما يلي:

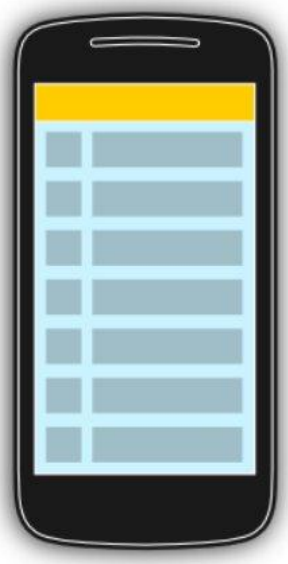
- Radio Buttons: لتحديد أي زر إنتقاء (Radio Button) تم اختياره، نقوم باستخدام المستمع RadioGroup.OnCheckedChangeListener وإضافته إلى مجموعة الأزرار RadioGroup (أنظر سطر رقم 14).عند حصول الحدث يتم تنفيذ الدالة ()onCheckedChanged والتي يمرر إليها رقم المعرف ID الخاص بالزر الذي تم اختياره حيث يتم طباعة رسالة تبين الإختيار (أنظر الكود من سطر 18 إلى 26).
- Check Boxes: للاستماع لحدث اختيار مربع الإختيار (CheckBox) نستخدم المستمع CompoundButton.OnCheckedChangeListener (أنظر سطور رقم 30 و 42)، وعند حصول الحدث يتم تنفيذ الدالة ()onCheckedChanged والتي يمرر إليها الكائن الخاص بمربع الإختياربالإضافة إلى قيمته (true-false) حيث يتم طباعة اسم مربع الاختيار وقيمته (أنظر الكود من سطر 33 إلى 38، ومن سطر 45 إلى 51).
- Spinner: نستخدم مستمع من نوع AdapterView.OnItemSelectedListener لمعالجة الحدث الخاص بهذا العنصر (أنظر سطر رقم 56). عند حصول الحدث يتم تنفيذ الدالة ()onItemSelected والتي من ضمن ما يمرر لها موقع العنصر الذي تم اختياره من القائمة. بمعرفة موقع العنصر يمكن الوصول له من القائمة عن طريق الدالة ()getItemAtPosition (أنظر الكود من سطر 60 إلى 65).
- Seek Bar: يتم معالجة الغيرات على الشريط المنزلق (SeekBar) عن طريق إضافة مستمع من نوع OnSeekBarChangeListener (أنظر سطر رقم 74). عند حصول الحدث يتم تنفيذ الدالة ()onProgressChanged والتي يمرر لها القيمة الحالية للشريط (0-100 مثلاً) بالإضافة إلى القيمة المنطقية fromUser والتي تحدد ما إذا كان تغير الشريط تم من المستخدم (fromUser=true) أو برمجياً (fromUser=false). في المثال الموضح يتم طباعة قيمة الشريط الممررة في عنصر من نوع TextView (أنظر الكود من سطر 77 إلى 80).

قائمة العرض (ListView)

عرض العناصر في قائمة هو أحد التصاميم الشائعة بكثرة في تطبيقات الهواتف النقالة. يشاهد المستخدم عدد من العناصر ويستطيع الانتقال لأعلى القائمة وأسفلها scroll up/down كما هو موضح بشكل 3-3 . عند إختيار أحد عناصر القائمة يتم عادة تنفيذ إجراء مثل فتح فعالية جديدة.

إضافة بيانات للقائمة (ListView)

لإضافة بيانات للقائمة (ListView) يتم عادةً استخدام المحول (Adapter)، وهو كائن وسيط بين عنصر العرض وهو القائمة (ListView) (أو أي عنصر واجهة يتفرع من الفئة AdaperView class) وبين البيانات المعروضة. يتحكم المحول (Adapter) في الوصول لبيانات القائمة، كما أنه مسؤول عن تحويل البيانات المدخلة للقائمة إلى عنصر عرض (View) يمكن تضمينه داخل القائمة. فمثلاً، عند إدخال مصفوفة من النصوص (Array String) لعرضها في القائمة، يقوم المحول (Adapter) بتحويل كل عبارة نصية في المصفوفة إلى كائن من نوع (TextView) والذي يتم إدراجه في القائمة، أي أن مصفوفة النصوص تتحول باستخدام المحول (Adapter) إلى مجموعة من عناصر العرض (TextView) المجمعة في قائمة من نوع (ListView).



شكل 3-3 : قائمة العرض (ListView) [1]

لاحظ أن محتوى القائمة قد يكون أكثر تعقيداً من مجرد عرض عبارات نصية. على سبيل المثال، قد يشتمل السطر الواحد في القائمة على صورة ونص وزر. في هذه الحالة يجب تصميم هيكلية الواجهة الخاصة بالسطر الواحد كملف Layout، ومن ثم نقوم بكتابة محول خاص (Custom Adapter) والذي يقوم باستقبال البيانات، الصور والنصوص، وإنشاء عنصر العرض (View) لعرضها بناءً على ملف layout. سنقوم لاحقاً في هذه الوحدة بتناول موضوع المحولات الخاصة (Custom Adapter).

المحولات الافتراضية (Default Adapter)

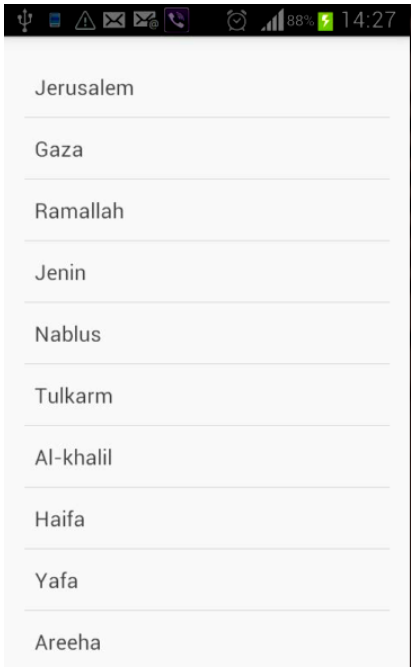
يوفر نظام أندرويد بعض المحولات الافتراضية، من أهمها ArrayAdapter و CursorAdapter. يستخدم ArrayAdapter لإدراجات البيانات الموجودة في مصفوفة Array أو List.المحول المخصص (CursorAdapter) يعالج البيانات المدخلة من قاعدة بيانات أو مزود المحتوى (Content Provider). كل هذه المحولات Adapters هي فئات فرعية (subclasses) من الفئة الأساسية BaseAdapter.

تمرين عملي (3-3)

يوضح هذا التمرين استخدام ArrayAdapter لتعبئة قائمة العرض (ListView) بمصفوفة من العبارات النصية. ArrayAdapter يتعامل مع مصفوفة من الكائنات objects حيث أن كل كائن سيمثل كسطر في القائمة (ListView).

[1] Using Lists in Android, atutorial by Lars Vogel, http://www.vogella.com/tutorials/AndroidListView/article.html

في هذا التمرين سنقوم بتعبئة قائمة (ListView) باسماء مدن فلسطينية كما هو موضح في شكل، وعند النقر على أي اسم يتم معالجة الحدث بطباعة رسالة على الشاشة.



شكل 4-3 : واجهة التطبيق الخاصة بالتمرين 3-3

ملف هيكلية الواجهة (Layout) الخاص بالبرنامج موضح بالأسفل، حيث تحتوي الواجهة على عنصر واحد فقط وهو القائمة (ListView):

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.example.simplelist.MainActivity" >
    <ListView
        android:id="@+id/listView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >
    </ListView>
</LinearLayout>
```

الكود الخاص بالفعالية MainActivity موضح في الأسفل:

```
1:  public class MainActivity extends Activity {
2:
3:   @Override
4:   protected void onCreate(Bundle savedInstanceState) {
5:        super.onCreate(savedInstanceState);
6:        setContentView(R.layout.activity_main);
7:        ListView listView = (ListView)
8:  this.findViewById(R.id.listView);
9:        String[] values = {"Jerusalem", "Gaza",
10: "Ramallah", "Jenin", "Nablus", "Tulkarm","Al-
11: khalil","Haifa","Yafa","Areeha"};
12:       ArrayList<String> listValues = new
13: ArrayList<String>();
14:       for(int i=0; i<values.length; i++)
15:            listValues.add(values[i]);
16:       ArrayAdapter<String> adapter = new
17: ArrayAdapter<String>(this,
18: android.R.layout.simple_list_item_1, listValues);
19:       listView.setAdapter(adapter);
20:       listView.setOnItemClickListener(new
21: OnItemClickListener(){
22:
23:           @Override
24:           public void onItemClick(AdapterView<?>
25: parent, View view, int position, long id) {
26:                Toast.makeText(getApplicationContext(),
27: parent.getItemAtPosition(position).toString(),
28: Toast.LENGTH_SHORT).show();
29:           }
30:       });
31:  }
32: }
```

لتعبئة قائمة العرض (ListView) يتم تنفيذ الخطوات التالية: يتم أولاً الوصول للقائمة عن طريقة الدالة findViewById() (أنظر سطر 7 و 8)، ثم نقول بإنشاء مصفوفة (List) تحتوي على أسماء المدن المراد إضافتها للقائمة (أنظر السطور من 9 إلى 15). لإدراج محتويات المصفوفة (List) في قائمة العرض (ListView) نستخدم ArrayAdapter ويتم إنشاءه كالتالي:

```
ArrayAdapter<String> adapter = new
ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1, listValues);
```

حيث يمرر له ثلاث قيم هي: context (الفعالية الحالية)، المعرف الخاص بتصميم هيكلية الواجهة (android.R.layout.simple_list_item_1)، وقائمة البيانات (listValues). تحتاج قائمة العرض (ListView) إلى تصميم يحدد كيفية عرض كل سطر وخصائص العرض (التنسيق والهوامش والخطوط وغيره من الخصائص). يمكن تصميم شكل السطر في القائمة في ملف Layout مستقل، ومن ثم تمريره للمحول (Adapter) عند إنشائه. في هذا التمرين، تم تمرير أحد التصميمات الافتراضية التي توفرها بيئة أندرويد والتي تحمل الصيغة :android.R.layout.<layout_name> . هناك المزيد من التصميمات والتي يمكن تجربتها وملاحظة التغير في شكل القائمة (ListView).

بعد إنشاء المحول (Adapter) وإدخال البيانات إليه، نقوم بتمريره إلى القائمة (ListView) عن طريق الدالة ()setAdapter من خلال (ListView) (أنظر سطر رقم 19). كما ذكرنا مسبقاً فإنه المحول (Adapter) يقوم بتحويل البيانات المدخلة إلى عناصر Views ومن ثم تجميعها وعرضها في القائمة (ListView).

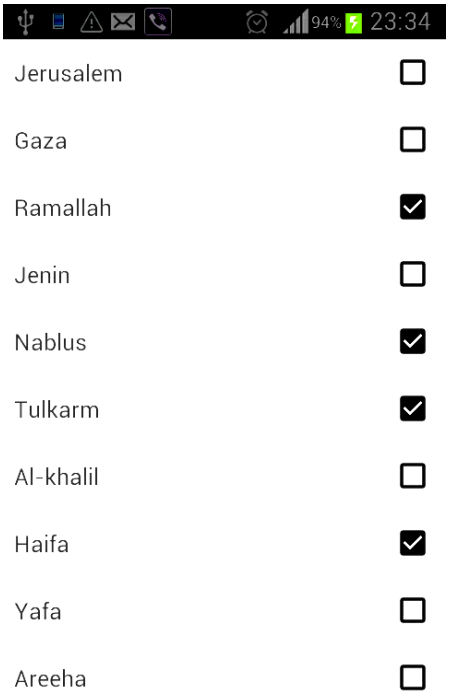
لمعالجة حدث النقر على أي عنصر من القائمة، قمنا بإضافة مستمع (Listener) من نوع OnItemClickListener (أنظر سطر رقم 20). عند حصول الحدث يتم تنفيذ الدالة ()onItemClicked والتي يمرر إليها موقع العنصر الذي تم اختياره من القائمة (ListView). يتم بعد ذلك الوصول و طباعة اسم العنصر الذي تم اختياره عن طريق الدالة ()getItemAtPosition (أنظر الكود من سطر 24 إلى 31).

لاحظ أن البيانات التي يتم تمريرها إلى المحول (Adapter) قد لا تكون بيانات نصية String، بل قد تكون كائن (object) من أي نوع. في حالة تمرير كائن غير نصي، يقوم المحول (Adapter) ضمنياً بتنفيذ الدالة ()toString لتمثيل الكائن كنص. لذلك احرص على تطبيق الدالة ()onString لأي كائن تقوم بإنشائه حتى تعمل القائمة (ListView) بعرض البيانات بالشكل الصحيح.

القائمة (ListView) في المثال السابق تسمح باختيار عنصر واحد من القائمة. يمكن تفعيل الإختيار المتعدد من القائمة عن طريق الدالة ()setChoiceModel بتنفيذ الأمر التالي:

```
listView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE);
```

كما يوفر نظام أندرويد تصماميم افتراضية للقوائم متعددة الإختيار مثل android.R.layout.simple_list_item_multiple_choice والتي يمكن تمريرها للمحول كما فعلنا في التمرين السابق، حيث تصبح واجهة التطبيق السابق عن تطبيق خاصية الاختيار المتعدد كما بالشكل

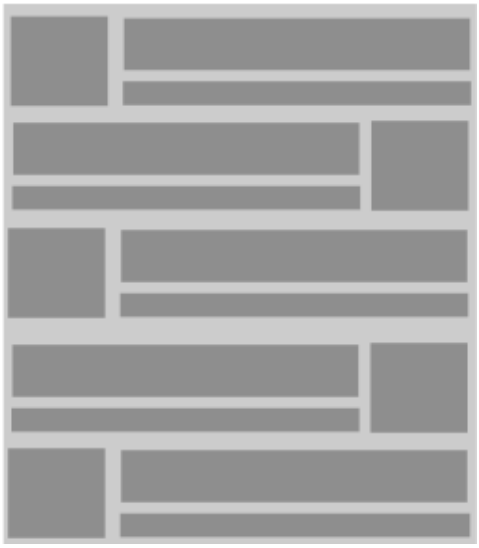


شكل 5-3 : الاختيار المتعدد من القائمة (ListView)

الجدير بالذكر أن نظام أندرويد يوفر فئة class باسم ListActivity، وهي فعالية ذو واجهة مستخدم إفتراضية تحتوي على قائمة (ListView). في حالة استخدام ListActivity بدلاً من Activity، لن تحتاج إلى تصميم ملف واجهة للتطبيق وإدراج قائمة جديدة (ListView) ، حيث يمكن الوصول إلى القائمة (ListView) الموجودة افتراضياً في الفعالية ListActivity عن طريقة الدالة () ListActivity.getListView.

المحول الخاص Custom Adapter

المحول الإفتراضي ArrayAdapter يوفر فقط إمكانية إضافة العناصر النصية من نوع String، وأي بيانات من نوع آخر يتم تمريرها للمحول ArrayAdapter يتم تحويلها تلقائياً لنص عن طريق الدالة ()toString. في كثير من الأحيان تحتاج لإضافة أنواع أخرى من البيانات للقائمة (ListView) مثل الصور. كذلك قد يكون السطر الواحد في القائمة مركباً حيث يتكون من أكثر من جزء كما هو موضح في الهيكلية الموضحة في شكل 6-3.



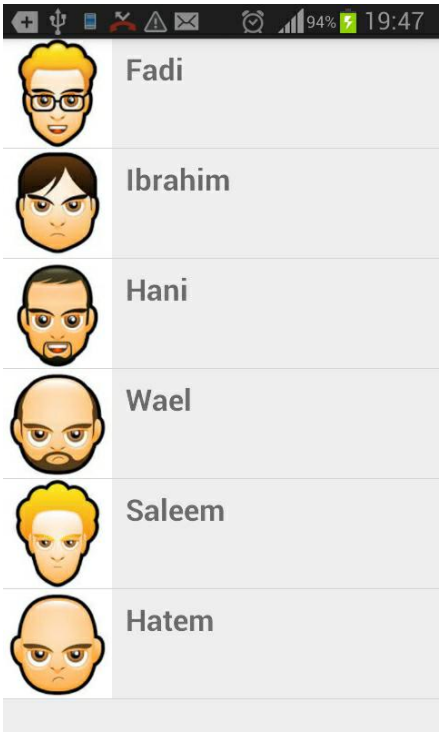
شكل 3-6 : قائمة عرض (ListView) حيث أن تصميم السطر يكون مركباً من مجموعة من العناصر[1]

في مثل هذه الحالات، نحتاج لإنشاء محول خاص (Custom Adapter) نحدد فيه تصميم الواجهة الخاصة بالسطر في قائمة العرض (ListView)، ونحدد كذلك كيف يتم معالجة البيانات المختلفة لعرضها في سطر القائمة. للقيام بذلك نقوم بإنشاء فئة (class) جديدة تمثل المحول الجديد بحيث تتفرع من الفئة BaseAdapter أو أي فئة متفرعة من BaseAdapter مثل ArrayAdapter، ثم نقوم بتطبيق الدالة ()getView والتي يتم من خلالها إنشاء الواجهة الخاصة بالسطر وعرض البيانات في عناصرها. التمرين التالي يوضح خطوات بناء واستخدام المحول الخاص (Custom Adapter) ضمن تطبيق بسيط.

تمرين عملي (3-4)

في هذا التمرين سيتم استخدام القائمة (ListView) بالإضافة لمحول خاص (Custom Adapter) لإنشاء واجهة التطبيق الموضحة بالشكل 3-7 : حيث تتكون الواجهة من قائمة (ListView) كل سطر فيها من مكون اسم شخص وصورته، وعند النقر على أي سطر يتم طباعة رسالة باسم الشخص.

[1] Using lists in Android, A tutorial by Lars Vogel, http://www.vogella.com/tutorials/AndroidListView/article.html



شكل 7-3: واجهة التطبيق الخاص بالتمرين 4-3

يتم في البداية تصميم واجهة السطر والتي تتكون من عنصر لعرض الصورة ImageView بالإضافة لعنصر نصي TextView لعرض الإسم. الملف التالي my_listview_layout.xml يوضح تصميم هيكلية الواجهة.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >
    <ImageView
        android:id="@+id/imageView"
        android:layout_width="80dp"
     android:layout_height="80dp"
        android:src="@drawable/ic_launcher" />
    <TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="44dp"
```

```
        android:textSize="24sp"
        android:textStyle="bold"
        android:layout_marginLeft="10dp"
        android:gravity="center_vertical"/>
</LinearLayout>
```

بعد ذلك يتم إنشاء المحول الخاص (Custom Adapter) والذي سيستخدم في الواجهة في الشكل 7-3 تصميمها لبناء القائمة (ListView). الكود الموضح بالأسفل يوضح الفئة (class) الخاصة بالمحول الخاص (Custom Adapter):

```
1:  public class MyCustomAdapter extends
2:  ArrayAdapter<String>{
3:   Context context;
4:   List<String> names;
5:   List<Integer> photos;
6:
7:   public MyCustomAdapter(Context context, List<String>
8:   names, List<Integer> photos) {
9:        super(context, R.layout.my_listview_layout,
10:  names);
11:       this.names = names;
12:       this.photos = photos;
13:       this.context = context;
14:  }
15:
16:  public View getView(int position, View view, ViewGroup
17:  parent){
18:       if(view == null){
19:            LayoutInflater inflater = (LayoutInflater)
20: context.getSystemService(
21: Context.LAYOUT_INFLATER_SERVICE);
22:            view =
23: inflater.inflate(R.layout.my_listview_layout, parent,
24: false);
25:            TextView tv = (TextView)
26: view.findViewById(R.id.textView1);
27:            ImageView iv = (ImageView)
28: view.findViewById(R.id.imageView1);
29:            tv.setText(names.get(position));
30:            iv.setImageResource(photos.get(position));
```

```
31:
32:             ViewHolder vh = new ViewHolder();
33:             vh.text = tv;
34:             vh.image = iv;
35:             view.setTag(vh);
36:        }else{
37:             ViewHolder vh = (ViewHolder) view.getTag();
38:             vh.text.setText(names.get(position));
39:
40:   vh.image.setImageResource(photos.get(position));
41:        }
42:        return view;
43:   }
44:
45:   static class ViewHolder{
46:        public TextView text;
47:        public ImageView image;
48:   }
49:}
```

كما تلاحظ فإن الفئة الجديدة MyCustomAdapter متفرعة من الفئة ArrayAdapter وتقوم بتطبيق الدالة getView().

من خلال الباني (Constructor) (أنظر السطور من رقم 7 إلى 14) يتم تمرير الكائن context (الفعالية مثلاً)، والمصفوفة الخاصة بالأسماء المراد عرضها (names) ثم مصفوفة الصور (photos) ممثلة بأرقام المعرفات IDs الخاصة بالصور. سيقوم المحول Adapter باستخدام هذه البيانات في بناء عناصر القائمة (ListView). لاحظ أن المحول ArrayAdapter يتطلب تمرير تصميم واجهة القائمة المعرفة بـ R.layout.my_listview_layout بالإضافة إلى قائمة من الجمل النصية String إلى الفئة الأم (superclass)وهي ArrayAdaper، وهو ما يتم من خلال الأمر التالي في الباني Constructor:

```
  super(context, R.layout.my_listview_layout, names);
```

الدالة الأساسية في المحول (Adapter) هي getView() (أنظر سطر رقم 16) والتي تحدد طريقة تحويل البيانات لكل سطر إلى عنصر عرض View ليكون ضمن القائمة (ListView). لاحظ أنه يتم استخدام كائن من نوع LayoutInflater لتحويل الواجهة الممثلة بملف XML إلى كائن نوع View. يمرر إلى الدالة inflate() المعرف الخاص بالواجهة المنشئة مسبقاً (R.layout.my_listview_layout) لتقوم بإرجاع كائن من نوع View (أنظر الكود من سطر 18 إلى 24).

يشتمل الكائن View داخله على عناصر الواجهة وهي ImageView و TextView. يتم بعد ذلك الوصول لكل من هذه العناصر عن طريق تنفيذ الدالة findViewById() على نطاق الكائن View (أنظر الكود من سطر 25 إلى 28).

```
TextView tv = (TextView) view.findViewById(R.id.textView1);
ImageView iv = (ImageView)
view.findViewById(R.id.imageView1);
```

بعد الوصول إلى عناصر السطر في القائمة (ListView)، يتم إضافة البيانات إليه (أنظر سطر رقم 29 و30). الدالة getView() يتم تنفيذها لكل سطر في القائمة (ListView)، وعند تنفيذها لأي سطر يمرر إليها رقم هذا السطر في المتغيرposition. فمثلاً، لعرض السطر الأول في القائمة يتم تنفيذ الدالة getView() تلقائياً ويمرر إليها position =0. لإضافة بيانات لسطر ما، نضيف البيانات الموجود في المصفوفات الخاصة بالأسماء والصور في المكان position. على سبيل المثال السطر الثالث في القائمة (ListView) حيث قيمة position تساوي 2 يأخذ البيانات الموجودة في المصفوفات من المكان 2. هذا يضمن أن يكون ترتيب العناصر في القائمة (ListView) مطابق لترتيبها في المصفوفات الخاصة بالبيانات. يتم ذلك من خلال السطرين التاليين في الكود:

```
tv.setText(names.get(position));
iv.setImageResource(photos.get(position));
```

وأخيراً، لاحظ أن الكود المرفق للمحول الخاص Cutom Adapter يستخدم ما يسمى بـ ViewHolder Pattern وذلك لتجنب تنفيذ الدالة findViewById() والتي تستنفذ وقت قد يؤثر على سرعة عرض (ListView) وسلاسة تحريكها. تعتمد هذه الفكرة على تخزين مؤشرات لعناصر الواجهة View في كائن من نوع ViewHolder (أنظر الكود من سطر 45 إلى 48) وذلك في أول مرة يتم فيها إن شاء الواجهة باستخدام LayoutInflater. ثم يتم إلحاق الكائن من نوع ViewHolder بعنصر العرض View الخاص بالواجهة عن طريق الدالة setTag()، وهو ما يتم من خلال الكود:

```
ViewHolder vh = new ViewHolder();
vh.text = tv;
vh.image = iv;
view.setTag(vh);
```

عندما يتم تنفيذ الدالة getView() ثانياً (كما في حالة عمل scroll لعرض عنصر تم عرضه مسبقاً)، فإن الدالة getView() لن تقوم بإنشاء عنصر الواجهة ثانياً باستخدام Layout Inflater لأنه قد تم إنشاؤه مسبقاً، حيث أن قيمة الكائن View المدخل للدالة getView() لا يساوي null في حالة إعادة عرض العنصر. في هذه الحالة لن يتم إنشاء الواجهة ثانية، ويتم استخدام الكائن ViewHolder الملحق بعنصر الواجهة View للوصول للعناصر TextView و ImageView بدون الحاجة لاستخدام الدالة getViewById() (أنظر الكود من سطر 36 إلى 41)، وهو ما يتم من خلال الكود التالي:

```
ViewHolder vh = (ViewHolder) view.getTag();
vh.text.setText(names.get(position));
vh.image.setImageResource(photos.get(position));
```

بعد إنشاء الفئة (class) الخاصة المحول (MyCustomAdapter) ، نقوم الآن ببناء الفعالية (Activity) والتي سيتم من خلالها إنشاء كائن من المحول MyCustomAdapter وتمرير البيانات المطلوب تعبئتها في القائمة (ListView) وهي أسماء وصور الأشخاص. الكود التالي خاص بهذه الفعالية:

```
1: public class MainActivity extends ListActivity {
2:
3:   @Override
4:   protected void onCreate(Bundle savedInstanceState) {
5:        super.onCreate(savedInstanceState);
6:        List<String> names = new ArrayList<String>();
7:        names.add("Fadi");
8:        names.add("Ibrahim");
9:        names.add("Hani");
10:       names.add("Wael");
11:       names.add("Saleem");
12:       names.add("Hatem");
13:
14:       List<Integer> photos = new ArrayList<Integer>();
15:       photos.add(R.drawable.face1);
16:       photos.add(R.drawable.face2);
17:       photos.add(R.drawable.face3);
18:       photos.add(R.drawable.face4);
19:       photos.add(R.drawable.face5);
20:       photos.add(R.drawable.face6);
21:
22:       MyCustomAdapter adapter = new
23: MyCustomAdapter(this, names, photos);
24:
25:       this.setListAdapter(adapter);
26:       this.getListView().setOnItemClickListener(new
27: OnItemClickListener(){
28:
29:            @Override
30:            public void onItemClick(AdapterView<?>
```

```
31: adapterView, View view, int position, long id) {
32:       Toast.makeText(getApplicationContext(),
33: adapterView.getItemAtPosition(position).toString(),
34: Toast.LENGTH_SHORT).show();
35:             }
36:       });
37:   }
38:}
```

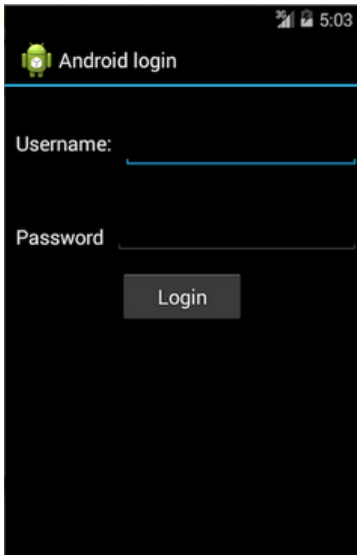
في الكود السابق، يتم في الدالة onStart() انشاء القوائم Lists الخاص بالأسماء والصور (أنظر السطور من 6 إلى 20) (لاحظ أن الصور يتم الوصول إليها باستخدام المعرفات بالصيغة: R.drawable.<image_file_name>). يجب أن تكون الصور محفوظة بنفس الأسماء في مجلد drawable. لاحظ أيضاً أن الفعالية متفرعة من (ListActivity) وهو ما يعني أننا لسنا بحاجة لتصميم واجهة خاصة بالفعالية، حيث أن الواجهة الافتراضية تشتمل على قائمة (ListView). يتم الوصول لهذه القائمة عن طريق الدالة ListActivity.getListView() من داخل الفعالية.

يتم إنشاء كائن من نوع المحول MyCustomAdapter وتمرير البيانات له، ومن ثم تمرير هذا الكائن إلى القائمة (ListView) عن طريق الدالة setListAdapter() (أنظر الكود من سطر 22 إلى 25).

وفي النهاية يتم الاستماع لحدث النقر على عناصر القائمة ListView عن طريق مستمع من نوع OnItemClickListener حيث يتم طباعة الإسم الذي تم النقر عليه (أنظر الكود من سطر 26 إلى 37).

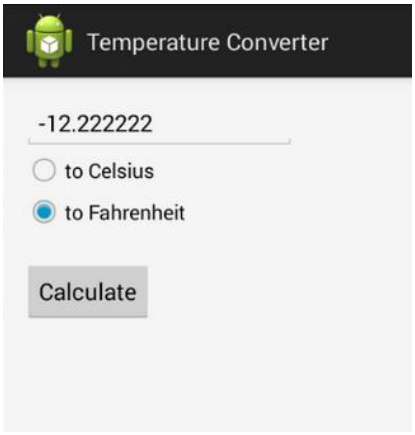
أسئلة على الوحدة الثالثة

1. قم بإنشاء التطبيق الوضح في الشكل التالي: يقوم المستخدم بإدخال اسم المستخدم "User Name" وكلمة المرور "Password"، وعند النقر على زر الإرسال "Submit" يتم قراءة القيم المدخلة وطباعتها على الشاشة باستخدام Toast.



2. قم بإنشاء تطبيق باسم "Temparature Converter" والذي يقوم بتحويل درجة الحرارة من سيليزيس (Celisius) إلى فهرنهايت (Fahrenheit) والعكس. واجهة التطبيق موضحة بالأسفل، حيث يتم إدخال درجة الحرارة في مربع الإدخال ومن ثم إختيار طريقة التحويل: "to Celisius" أو "to Fahrenheit". عند النقر على زر "Calculate" يتم حساب النتيجة وطباعتها برسالة Toast. مع العلم أن التحويل يتم بناءً على المعادلات التالية:

$$°F = °C \times 9/5 + 32$$
$$°C = (°F - 32) \times 5/9$$



3. قم بالتعديل على تمرين 3-4 كالتالي: كل سطر في قائمة العرض (ListView) يعرض صورة واسم الشخص بالإضافة إلى وضع الاتصال الخاص به: متصل (Online) أو غير متصل (Offline). يتم عرض الأيقونات الموضحة بالشكل بناء على وضع الاتصال. أوضاع الاتصال الخاصة بالمستخدمين يتم تمريرها كمصفوفة للمحول Adapter.

4. قم بالتعديل على المحول MyCustomAdapter المستخدم في تمرين 3-4 وذلك بعمل فئة فرعية (subclass) من الفئة (BaseAdapter) بدلاً من الفئة (ArrayAdapter) وقم بعمل التغييرات اللازمة. تأكد بأن تمرين 3-4 يعمل بشكل صحيح بعد إجراء التغيير.

الوحدة الرابعة:

الربط بين الفعاليات باستخدام الأهداف

(Linking Activities Using Intents)

يتعلم الطالب في هذه الوحدة:

- ✓ .مفهوم الهدف (Inent) وأنواعة واستخداماته.
- ✓ تشغيل فعالية (Activity) من فعالية أخرى.
- ✓ التواصل بين الفعاليات وتبادل البيانات بينها.
- ✓ .مفهوم مرشح الهدف (Intent Filter) واستخداماته.

تطبيق أندرويد قد يتكون من أكثر من فعالية (Activity)، بحيث يتم الإنتقال من فعالية إلى أخرى أثناء استخدام التطبيق. فمثلاً، قد تكون هناك فعالية (Activity) تعرض قائمة عرض (ListView)، وعند اختيار أي عنصر من القائمة يتم تشغيل واجهة جديدة (فعالية) لعرض بيانات متعلقة بالعنصر الذي تم اختياره. في مثل هذه الحالات، يجب تشغيل فعالية من فعالية أخرى، وكذلك قد يلزم نقل البيانات بين الفعاليات. الربط بين الفعاليات في نظام أندرويد يتم باستخدام ما يسمى بـIntent أو الهدف (ترجمة حرفية). يتناول هذا الفصل مفهوم الهدف (Intent) وأنواعة المختلفة وطريقة تشغيل الفعاليات باستخدام (Intent)، ويتم تدعيم كل هذه المفاهيم بتمارين عملية.

الأهداف (Intents)

الـ (Intent) أو الهدف هو كائن (Object) يستخدم للتواصل بين مكونات التطبيقات لطلب إجراء معين أو لتبادل المعلومات. يمكن تخيل الهدف بأنه رسالة يتم إرسالها إلى التطبيقات لطلب إجراء ما أو لتبادل المعلومات. هناك ثلاث استخدامات أساسية للهدف (Intent) وهي كالتالي:

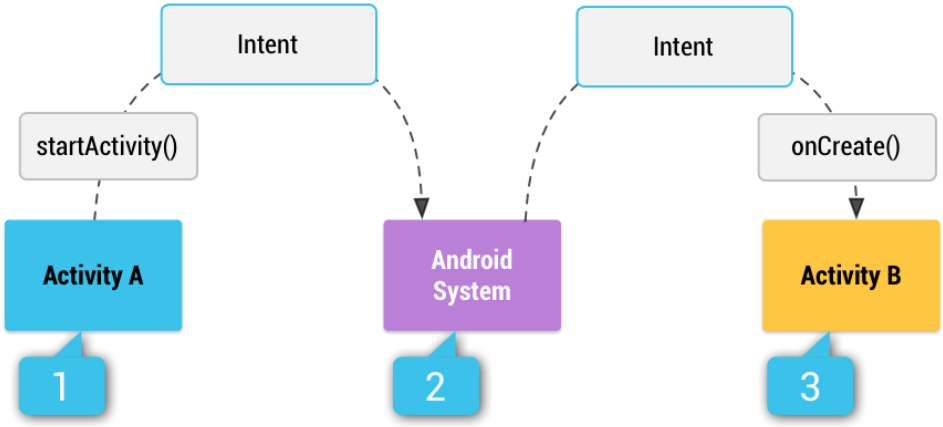
- تشغيل فعالية (Activity) واستقبال النتائج منها: الفعالية تمثل واجهة من واجهات التطبيق. يمكن تشغيل فعالية ما عن طريق تمرير هدف (Intent) إلى الدالة ()startActivity. في هذه الحالة يحدد الهدف (Intent) الفعالية المراد تشغيلها بالإضافة لأي بيانات أخرى تحتاجها الفعالية.
- تشغيل خدمة (Service): الخدمة (Service) هي مكون من مكونات تطبيق أندرويد يستخدم لتنفيذ مهام بدون واجهة للمستخدم، وغالباً ما تعمل الخدمة في الخلفية بدون تفاعل من المستخدم. يتم تشغيل خدمة معينة (Service) عن طريق الدالة ()startService والتي يمرر لها كائن من نوع الهدف (Intent) يحدد الخدمة المراد تشغيلها.
- إرسال منشور (Deliver Broadcast): المنشور (Broadcast) هي رسالة يتم ارسالها إلى تطبيقات مختلفة. على سبيل المثال، عن إعادة تشغيل الجهاز boot يقوم نظام أندرويد تلقائياً بارسال رسالة من نوع (Intent) وذلك لإعلام كل البرامج المعنية بحدث boot حتى تنفذ بناءً عليه إجراءات أخرى. أرسال المنشور Broadcast سيتم التطرق إليه في الوحدة الثامنة.

أنواع الهدف (Intent Types)

هناك نوعان للهدف (Intent)، وهما:

- الهدف الصريح (Explicit Intent) :وفيه يتم تحديد المكون المراد تشغيله (فعالية Activity أو خدمة Service أو منشور Broadcast). غالباً ما يتم استخدام الهدف الصريح لتشغيل مكون من مكون آخر في نفس التطبيق لأن الفئة (class) الخاصة بالمكون المراد تشغيله معروفة ويمكن الوصول إليها في نطاق التطبيق الواحد.
- الهدف المضمن (Implicit Intent): وفيه لا يتم تحديد المكون المراد تشغيله صراحةً، ويتم تحديد نوع الإجراء المراد تنفيذه بدلاً من ذلك. تحديد الإجراء يُمكّن أي مكون يتبع لأي تطبيق آخر أن ينفذ هذا الإجراء إذا كان مصمماً لذلك. على سبيل المثال، إذا اراد التطبيق عرض صفحة ويب معينة، يجب تشغيل فعالية (Activity) قادرة على عرض وتصفح الإنترنت، ويتم ذلك بتنفيذ الدالة startActivity() ويمرر للدالة هدف مضمن يحدد فيه إجراء باسم ACTION_VIEW وصفحة الويب المراد فتحها (لاحظ أنه لم يحدد اسم مكون محدد لعرض صفحة الويب). يقوم النظام تلقائياً باختيار الفعالية الملائمة لتشغيل الإجراء المطلوب (ACTION_VIEW)، وقد تكون الفعالية الملائمة تابعة لتطبيق آخر في نفس الجهاز.

عند إنشاء هدف صريح (Explicit Intent) بهدف تشغيل مكون ما فإن النظام يقوم مباشرة بتشغيل المكون المحدد في الهدف الصريح، أما عند إنشاء هدف مضمن (Implicit Intent) فإن النظام يبحث عن المكون المناسب لتنفيذ الإجراء المحدد في الهدف (Intent) وذلك في كل التطبيقات الموجودة في الجهاز. إذا توافق الهدف مع تطبيق معين، يقوم النظام بتشغيل التطبيق الموافق ويرسل له الهدف (Intent) والذي قد يحوي بيانات أخرى لإرسالها للتطبيق المطلوب.عند وجود أكثر من فعالية يمكنها تنفيذ الإجراء المطلوب يقوم النظام بعرض قائمة الفعاليات حتى يختار المستخدم الفعالية التي يريدها. شكل 1-4 يوضح إجراءات تشغيل فعالية (Activity B) من فعالية (Activity A) باستخدام هدف مضمن (Implicit Intent).



شكل 1-4: إنشاء فعالية باستخدام هدف مضمن: 1- من داخل الفعالية A يتم تنفيذ التعليمة startActivity() ويمرر إليها هدف مضمن يوضح الإجراء المراد تنفيذه. 2- يتقبل النظام الهدف ويبحث في كل التطبيقات في

الجهاز عن أي فعالية يمكنها تشغيل الإجراء المطلوب. فمثلاً، الفعالية B تحقق الغرض، لذلك يقوم النظام بتشغيلها ويرسل لها الهدف. 3- عند تشغيل الفعالية B يتم تنفيذ الدالة ()onCreate حيث يمكن خلالها استقبال الهدف الذي ارسله النظام والذي قد يحتوي على معلومات تلزم الفعالية B.[1]

مكونات الهدف (Intent)

الهدف (Intent) يحوي المعلومات اللازمة لتشغيل مكون ما وتشمل أهم هذه المعلومات ما يلي:

- المكون (Component): وهو يحدد المكون الذي يجب أن يستقبل الهدف (Intent). يجب تحديد اسم المكون فقط عند استخدام هدف صريح (Explicit Intent)، وهو ما يعني أن الهدف يجب أن يرسل فقط للمكون المحدد بالإسم. بدون تحديد اسم مكون يصبح الهدف مضمناً (Implicit Intent)، وهو ما يعني أن النظام هو من يحدد المكون الذي يمكنه استقبال الهدف بناءً على المعلومات الأخرى المضمنة في الهدف. لذلك إذا أردت تشغيل مكون محدد في تطبيقك فيجب عليك تحديد المكون من خلال هدف صريح (Explicit Intent) . يجب الملاحظة أن تحديد المكون يعني هنا الفئة (class) الخاصة بهذا المكون مثل الفعالية (Activity).
- الإجراء (Action): وهو جملة تحدد الإجراء المراد تنفيذه مثل عرض صفحة ويب أو الاتصال على رقم جوال وغيره. يتم تحديد الإجراء (Action) في حالة الهدف المضمن (Implicit Intent) فقط، حيث يترك للنظام مسؤولية إختيار المكون القادر على تنفيذ الإجراء. يمكن تحديد الإجراء الذي يستخدمه الهدف برمجياً إما بتمريره من خلال منشئ الباني constructor الخاص بالهدف أو بتنفيذ الدالة ()setAction. هناك بعض الإجراءات العامة التي يتعرف عليها النظام والتي يمكن استخدامها لأعراض عامة، ومن أهم هذه الإجراءات:
  - ACTION_VIEW: والذي يستخدم لعرض بيانات معينة للمستخدم. قد تكون هذه البيانات عبارة عن عنوان موقع ويب أو صورة يراد عرضها أو موقع على الخريطة وغيره.
  - ACTION_SEND: وهو يستخدم لمشاركة البيانات ارسالها كبريد إلكتروني أو تحميلها على شبكة إجتماعية.
- البيانات (Data): وهي عبارة عن كائن من نوع Uri يشير إلى البيانات المراد استخدامها. نوع البيانات المرسلة في الهدف (Intent) يعتمد على الإجراء المطلوب، فمثلا في حالة الإجراء (ACTION_VIEW) تكون البيانات على شكل عنوان الصفحة أو الصورة المراد عرضها.
- التصنيف (Category): وهو يحدد نوع المكون الذي يتلقى الهدف (Intent). في معظم الحالات لن تحتاج لتحديد التصنيف. من أنواع التصنيفات CATEGORY_LAUNCHER وهو يحدد الفعالية الرئيسية التي سيبدأ بها تشغيل التطبيق، حيث أن كل تطبيق له فعالية واحدة فقط من هذا التصنيف.
- الإضافات (Extras): وهو عبارة عن حافظة تستخدم لإرسال بيانات إضافية من خلال الهدف (Intent)، وترسل البيانات على شكل مفاتيح وقيم مقابلة (key-value pairs)، حيث يمكن الوصول لكل قيمة من خلال المفتاح المحدد لها. يتم إضافة البيانات للحافظة عن طريق الدوال من نوع ()putExtra. على سبيل المثال، اذا

[1] Intents and Intent Filters, Android Developer, http://developer.android.com/guide/components/intents-filters.html

اردت ارسال بريد إلكتروني من خلال التطبيق، يجب انشاء هدف وتحديد الإجراء له من نوع (ACTION_SEND). ثم يجب تحديد العنوان البريدي للمستقبل عن طريق الدالة putExtra وتستخدم المفتاح EXTRA_EMAIL كمفتاح لبيانات المستقبل، ويمكن أيضا تحديد موضوع الرسالة البريدية subject من خلال اضافة الموضوع بالمفتاح EXTRA_SUBJECT.

- Flag: وهو متغير يحدد كيفية إنشاء الفعالية وما المهمة التي ستتبع لها وكيف سيتم التعامل معها بعد الإنشاء. في الغالب لن تحتاج إلى تحديد قيم لهذا المتغير.

تشغيل الفعالية (Activity) باستخدام الهدف (Intent)

يمكن تشغيل فعالية عن طريق فعالية أخرى ما عن طريق تنفيذ الدالة startActivity() وتمرير هدف صريح (Explicit Intent) يحدد الفعالية المراد تشغيلها. الهدف (Intent) قد يحوي بالاضافة إلى الفعالية المراد تشغيلها بيانات أخرى مراد تمريرها للفعالية الجديدة.

الكود التالي يوضح تشغيل الفعالية SignInActivity باستخدام هدف صريح (Explicit Intent) وتمرير بيانات لها عن طريق حافظة الإضافات (Extras) الموجودة ضمن الهدف (Intent):

```
Intent intent =new Intent(getApplicationContext(),
SignInActivity.class);
intent.putExtras("username","Ahmed");
startActivity(intent);
```

لاحظ أن إضافة البيانات يتم في الحافظة Extras بطريقة المفتاح والقيمة (key-value)، حيث يستخدم المفتاح (key) لاحقاً لاسترجاع البيانات من الهدف (Intent). الفعالية التي تم تشغيلها باستطاعتها قراءة الهدف (Intent) عن طريق تنفيذ الدالة getIntent() ومن ثم استخراج البيانات المرسلة من خلال الهدف لمعالجتها.

في حالات أخرى قد تحتاج إلى تنفيذ إجراء معين (Action) مثل إرسال بريد إلكتروني أو رسالة نصية. في هذه الحالة قد لا يملك البرنامج أي فعالية محددة لتنفيذ الإجراء المطلوب، ويمكن استخدام فعاليات توفرها تطبيقات أخرى موجودة على الجهاز. يمكن تنفيذ ذلك باستخدام هدف مضمن (Implicit Intent) يحدد نوع الإجراء المراد تنفيذه حيث سيقوم النظام تلقائياً باختيار الفعالية المناسبة للإجراء المطلوب من أي تطبيق آخر على الجهاز. في حالة وجود أكثر من فعالية تدعم الإجراء المطلوب (مثل وجود أكثر من مستعرض لصفحات الانترنت)، يقوم النظام بطلب تدخل المستخدم لاختيار الفعالية التي يريدها لتنفيذ الإجراء المطلوب. على سبيل المثال، اذا أردت السماح لمستخدم بإرسال بريد إلكتروني من خلال تطبيق معين، يمكن تنفيذ ذلك بالكود التالي:

```
Intent intent =new Intent(Intent.ACTION_SEND);
intent.putExtra(Intent.EXTRA_EMAIL, recipientArray);
startActivity(intent);
```

لاحظ أنه تم إنشاء الهدف (Intent) باستخدام الإجراء المطلوب (ACTION_SEND) ولم يتم تحديد فعالية محددة لتشغيلها. الثابت EXTRA_EMAIL المضاف للهدف (Intent) يحدد المصفوفة recipientArray والتي تحوي العناوين البريدية المراد إرسال البريد إليها. هذه العناوين البريدية يتم إدارجها في خانة "إلى" (to) ضمن نموذج إرسال البريد.

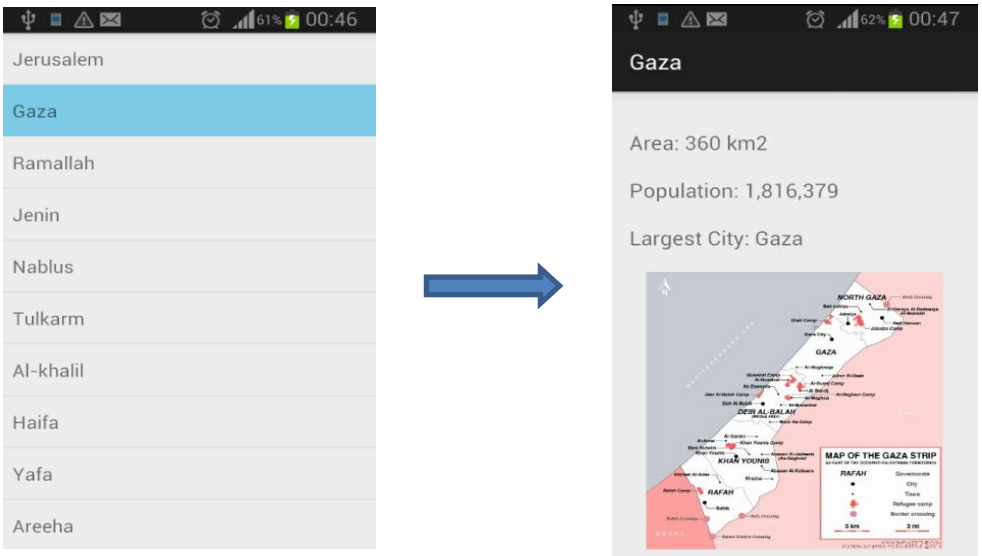
ملاحظة هامة: عند إنشاء الفئة (class) الخاصة بالفعالية الجديدة تأكد أن الفعالية تم تعريفها في ملف الوثيقة (Manifest) وذلك بإضافة الخاصية activity داخل الخاصية application في ملف الوثيقة Manifest كما هو موضح:

```
<application>
…
  <activity
     android:name="ps.edu.ucas.example.NewActivity"
     android:label="@string/title_activity_details" >
  </activity>
…
</application>
```

حيث أن قيمة الخاصية android:name تشير إلى مسار الفئة (classpath) الخاص بالفعالية الجديدة.

تمرين عملي (1-4)

في هذا التمرين سنقوم بالتعديل على تمرين (3-3) والذي يقوم بعرض قائمة (ListView) بأسماء المدن الفلسطينية. التعديل المطلوب إجراؤه هو تشغيل فعالية جديدة (Activity) عند النقر على اسم أي مدينة وذلك لعرض معلومات عن هذه المدينة في الفعالية الجديدة كما بالشكل:



شكل 2-4: واجهة التطبيق الخاص بتمرين 1-4.

بالأسفل ملف الواجهة الخاص بالفعالية الجديدة، والذي يتكون من عناصر TextView و ImageView لعرض التفاصيل النصية والصور الخاصة بالمدينة التي يتم اختيارها من القائمة (ListView).

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"

android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
tools:context="com.example.simplelist.DetailsActivity" >
    <TextView
        android:id="@+id/tvArea"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="20sp"
        android:layout_marginTop="20dp"
        android:text="TextView" />
    <TextView
        android:id="@+id/tvPopulation"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="20sp"
        android:layout_marginTop="20dp"
        android:text="TextView" />
    <TextView
        android:id="@+id/tvLargestCity"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="20sp"
        android:layout_marginTop="20dp"
        android:text="TextView" />
    <ImageView
        android:id="@+id/ivMap"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:textSize="20sp"
        android:layout_marginTop="20dp"/>
</LinearLayout>
```

الكود بالأسفل يوضح الفعالية الجديدة (DetailsActivity) والتي يتم فيها عرض تفاصيل المدينة التي يتم اختيارها من القائمة (ListView).

```
1:  public class DetailsActivity extends Activity {
2:
3:   @Override
4:   protected void onCreate(Bundle savedInstanceState) {
5:        super.onCreate(savedInstanceState);
6:        setContentView(R.layout.activity_details);
7:        TextView tvArea = (TextView)
8:  this.findViewById(R.id.tvArea);
9:        TextView tvPopulation = (TextView)
10:  this.findViewById(R.id.tvPopulation);
11:       TextView tvLargestCity = (TextView)
12:  this.findViewById(R.id.tvLargestCity);
13:       ImageView ivMap = (ImageView)
14:  this.findViewById(R.id.ivMap);
15:       Intent intent = this.getIntent();
16:       String cityName =
17:  intent.getExtras().getString("city");
18:       this.setTitle(cityName);
19:       if(cityName.equals("Gaza")){
20:          tvArea.setText("Area: 360 km2");
21:          tvPopulation.setText("Population: 1,816,379");
22:            tvLargestCity.setText("Largest City: Gaza");
23:            ivMap.setImageResource(R.drawable.gaza);
24:       }
25:  }
26: }
```

محتوى واجهة الفعالية (DetailsActivity) يتغير بتغير اسم المدينة الذي يتم إرسالة من الفعالية الاولى من خلال هدف (Intent). الجزء الأساسي في الكود هو قراءة الهدف (Intent) المرسل من فعالية البداية باستخدام الدالة getIntent () (أنظر سطر رقم 15). قراءة البيانات المرسلة في الهدف (Intent) يتم بقراءة الحافظة الموجودة ضمن الهدف باستخدام الدالة getExtras() ومن ثم تنفيذ أحد دوال القراءة حسب نوع البيانات المرسلة مثل

getInt() و getString() (أنظر سطر رقم 17). بناءً على اسم المدينة الذي تم ارساله من فعالية البداية يتم تعبئة محتوى الفعالية (DetailsActivity).

ملاحظة: تأكد من إضافة تعريف الفعالية (DetailsActivity) في ملف الوثيقة (Manifest).

بعد إنشاء الفعالية المراد تشغيلها، نقوم بالتعديل على فعالية البداية (MainActivity) والتي تحتوي على القائمة (ListView) وذلك بإضافة الكود التالي داخل الدالة onItemClick() الخاصة بالمستمع OnItemClickListener:

```
…
1:  listView.setOnItemClickListener(new
2:  OnItemClickListener(){
3:
4:  @Override
5:  public void onItemClick(AdapterView<?> parent, View
6:   view, int position, long id) {
7:   String cityName =
8:   parent.getItemAtPosition(position).toString();
9:   Intent intent = new Intent(getApplicationContext(),
10:  DetailsActivity.class);
11:  intent.putExtra("city", cityName);
12:  startActivity(intent);
13:  }
14: });
…
```

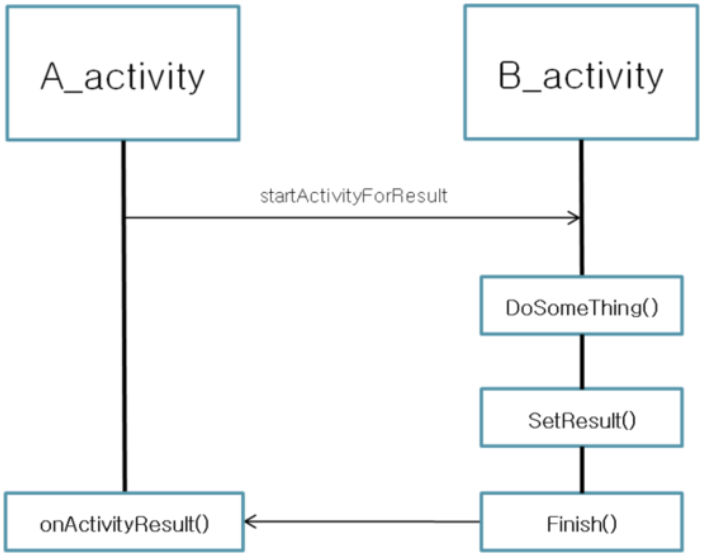
لاحظ أنه تم إنشاء هدف صريح (Explicit Intent) يحتوى فئة (class) الفعالية المراد تشغيلها وهي DetailsActivity (أنظر السطرين رقم 9 و10). تم بعد ذلك حفظ اسم المدينة الذي تم النقر عليه في الحافظة (Extras) باستخدام المفتاح city (أنظر سطر رقم 11). وفي النهاية تم تشغيل الفعالية باستخدام الدالة startActivity() (أنظر سطر رقم 12).

إنشاء فعالية من أجل نتيجة (Starting an activity for a result)

في بعض الأحيان قد تحتاج إلى استرجاع نتيجة من الفعالية (Activity) التي قمت بتشغيلها: فمثلاً قد تنشئ فعالية لتطلب من المستخدم الاختيار من قائمة ما و ترجع النتيجة إلى الفعالية الأساسية. في هذه الحالة يمكن استخدام الأمر startActivityForResult() بدلاً من startActivity(). لاستقبال النتيجة من الفعالية الجديدة.

لتوضيح آلية التواصل بين الفعاليات Activitites في هذه الحالة، افترض أن هناك فعاليتين A و B بحاجة للتواصل بحيث تقوم A بتشغيل B وانتظار نتائج منها. شكل 3-4 يوضح آلية التواصل بين الفعاليتين: تقوم الفعالية A بتشغيل الفعالية B باستخدام الدالة startActivityForResult(). الفعالية B بدورها تنفذ إجراء معين

DoSomeThing() ثم تقوم بتجهيز النتائج على شكل هدف (Intent) وترجعها بتنفيذ الدالة setResult(). عند إغلاق الفعالية B، بالنقر على زر Back مثلاً، تعمل الفعالية A ويتم تلقائياً تنفيذ الدالة onActivityResult() والتي يمكن من خلالها قراءة البيانات المرجعة من الفعالية B.

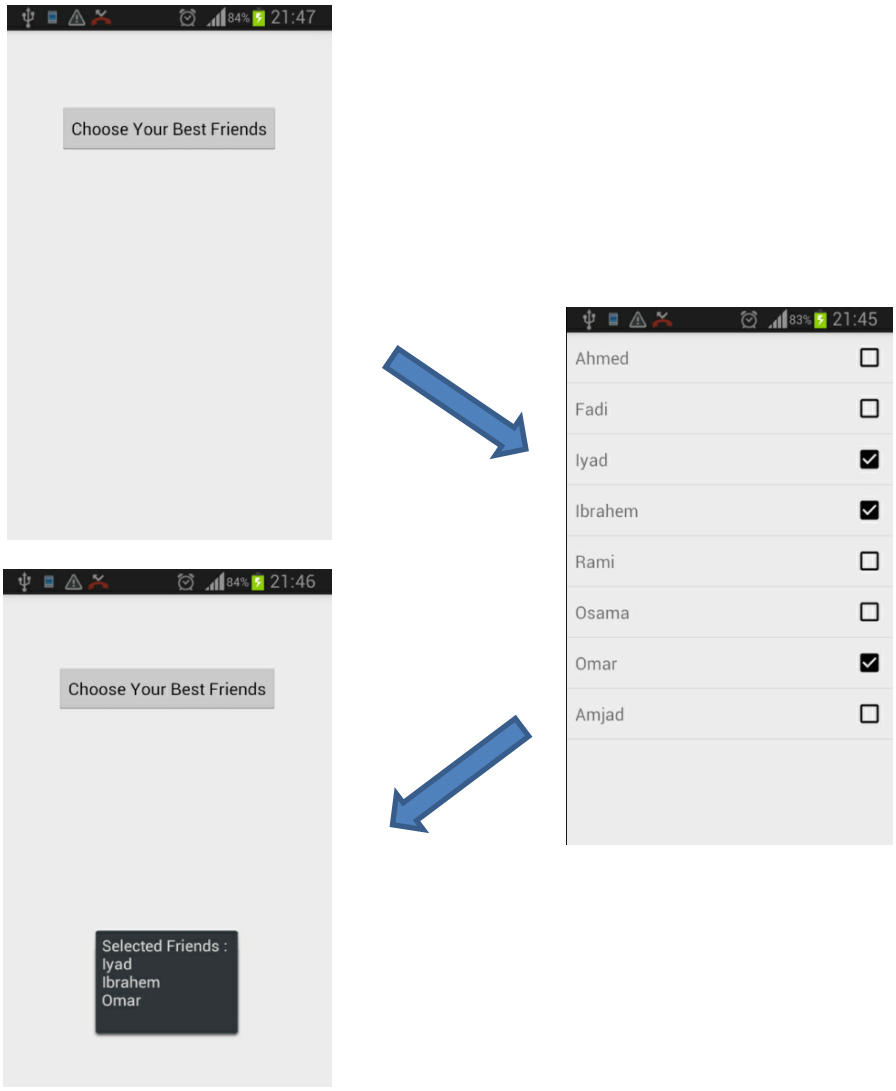


شكل 3-4: التواصل بين الفعاليات (Activities) في حالة تنفيذ الدالة startActivityForResult()

لتوضيح هذه الخطوات عملياً سنقوم ببناء تطبيق بسيط يوضح هذه الآلية.

تمرين عملي (2-4)

في هذا التمرين سنطبق مفهوم تشغيل فعالية (Activity) من أجل إرجاع نتيجة. واجهات التطبيق موضحة في الشكل، وهو مكون من فعاليتين (Activities): الفعالية الرئيسية تظهر زر عند النقر عليه يتم تشغيل فعالية جديدة تظهر قائمة من أسماء أصدقاء حيث يجب اختيار قائمة الأصدقاء المفضلة منها. بعد إنهاء الفعالية الثانية بالنقر على زر Back، يتم تشغيل الفعالية الرئيسية ويتم طباعة قائمة الأصدقاء التي تم اختيارها على الشاشة.



شكل 4-4: واجهات التطبيق الخاص بتمرين 2-4 والتواصل بينها.

الكود التالي يوضح الفعالية الرئيسية (MainActivity):

```
1:  public class MainActivity extends Activity{
2:
3:   public static int REQUEST_CODE = 50; // Any ID number
4:   @Override
5:   protected void onCreate(Bundle savedInstanceState) {
```

```
6:        super.onCreate(savedInstanceState);
7:        setContentView(R.layout.activity_main);
8:        Button btn = (Button)
9:  this.findViewById(R.id.button);
10:       btn.setOnClickListener(new OnClickListener(){
11:
12:            @Override
13:            public void onClick(View v) {
14:                 Intent intent = new
15:  Intent(getApplicationContext(),
16:  FriendsListActivity.class);
17:  startActivityForResult(intent, REQUEST_CODE);
18:            }
19:       });
20:  }
21:
22:  @Override
23:  protected void onActivityResult(int requestCode, int
24:  resultCode, Intent data) {
25:       super.onActivityResult(requestCode, resultCode,
26:  data);
27:       if(requestCode == REQUEST_CODE && resultCode ==
28:  FriendsListActivity.RESULT_CODE){
29:            ArrayList<String> selectedFriends =
30:  data.getStringArrayListExtra("selectedFriends");
31:            String msg = "Selected Friends :\n";
32:            for(String friend:selectedFriends)
33:                 msg+=friend+"\n";
34:            Toast.makeText(getApplicationContext(), msg,
35:  Toast.LENGTH_LONG).show();
36:       }
37:   }
38: }
```

لاحظ أنه عن النقر على الزر "Choose Your Best Friends" يتم إنشاء هدف صريح (Explicit Intent) بالفعالية المراد تشغيلها وهي (FreindsListActivity) (أنظر الكود من سطر 14 إلى 16) ، ثم يتم تشغيل الفعالية عن طريقة الدالة ()startActivityForResult والتي يمرر لها الهدف (Intent) بالإضافة إلى قيمة رقمية تستخدم كمعرف للطلب request، حيث يمكن استخدام أي قيمة رقمية من نوع integer (أنظر سطر رقم 17).

بعد الانتهاء من العمل في الفعالية الجديدة، يتم استقبال النتيجة في الدالة ()onActivityResult (أنظر سطر رقم 23) والتي تنفذ تلقائياً عند تلقي النتيجة. من خلال الدالة ()onActivityResult يتم تلقي ثلاث نتائج هي: request_code، result_code و data. القيمة request_code تحدد معرف الطلب الأساسي ويفترض أن يطابق نفس القيمة التي تم إرسالها عن تشغيل الفعالية (فحص هذه القيمة يفيد في تحديد الطلب الذي تم الرد عليه في حالة تنفيذ أكثر من طلب لارجاع نتيجة). القيمة الثانية result_code يتم إرسالها من الفعالية الثانية وتفيد كمعرف لمصدر النتيجة. القيمة الثالثة data هي هدف (Intent) يحتوي بداخله على النتائج المرجعة. لاحظ أن النتيجة المرجعة عبارة عن قائمة <ArrayLisy<String تمثل الأسماء التي تم إختيارها، وقد تم استرجاعها من الهدف (Intent) عن طريق تنفيذ الدالة ()getStringArrayListExtra (أنظر سطر رقم 29 و30). بعد ذلك يتم تحويل النتائج إلى رسالة وتطبع على الشاشة.

الكود التالي يوضح الفعالية الثانية FriendsListActivity وتكون من نوع ListActivity:

```
1:public class FriendsListActivity extends ListActivity {
2:
3:   public static int RESULT_CODE = 60; //Any ID number
4:
5:   @Override
6:   protected void onCreate(Bundle savedInstanceState) {
7:        super.onCreate(savedInstanceState);
8:        ArrayList<String> friendNames = new
9:  ArrayList<String>();
10:       friendNames.add("Ahmed");
11:       friendNames.add("Fadi");
12:       friendNames.add("Iyad");
13:       friendNames.add("Ibrahem");
14:       friendNames.add("Rami");
15:       friendNames.add("Osama");
16:       friendNames.add("Omar");
17:       friendNames.add("Amjad");
18:       ArrayAdapter<String> adapter= new
19:  ArrayAdapter<String>(this,android.R.
20:  layout.simple_list_item_multiple_choice, friendNames);
21:       this.setListAdapter(adapter);
22: getListView().setChoiceMode(ListView.
23:  CHOICE_MODE_MULTIPLE);
24:  getListView().setOnItemClickListener(new
25:  OnItemClickListener(){
26:
27:  @Override
```

```
28:   public void onItemClick(AdapterView<?> parent, View
29: view, int position, long id) {
30:       ListView listView = getListView();
31:       SparseBooleanArray checked =
32:  listView.getCheckedItemPositions();
33:  ArrayList<String> selectedItems = new
34:  ArrayList<String>();
35:  for (int i = 0; i < listView.getCount(); i++) {
36:    if(checked.get(i))
37:      selectedItems.add(listView.getItemAtPosition(i).
38:  toString());
39:  }
40:  Intent intent = new Intent();
41:  intent.putStringArrayListExtra("selectedFriends",
42:  selectedItems);
43:  setResult(RESULT_CODE, intent);
44:           }
45:       });
46:  }
47:}
```

لاحظ أنه تم تعبئة قائمة العرض (ListView) بمصفوفة الأسماء باستخدام ArrayAdapter كما هو موضح في الكود (أنظر الكود من سطر 8 إلى 20). لاحظ أيضاً أن القائمة (ListView) تسمح بالاختيار المتعدد من خلال تنفيذ الدالة setChoiceMode(ListView.CHOICE_MODE_MULTIPLE) (أنظر سطر رقم 22). لاحظ أيضاً الإجراء الذي يتم تنفيذه عند اختيار عناصر من القائمة (ListView) حيث يتم حفظ العناصر التي تم اختيارها في قائمة ArrayList<String> (أنظر الكود من سطر 28 إلى 39). لارجاع هذه القائمة للفعالية الرئيسية، يتم إنشاء هدف (Intent) وحفظ القائمة به عن طريقة تنفيذ الدالة putStringArrayListExtra() (أنظر سطر رقم 41). وأخيراً، يتم إرجاع النتيجة عن طريقة تنفيذ الدالة setResult() حيث ترسل النتائج بالإضافة إلى رقم result_code ليستخدم كمعرف لمصدر النتائج (أنظر سطر رقم 43).

مرشح الهدف (Intent Filter)

مرشح الهدف (Intent Filter) هي خصائص يتم كتابتها في ملف الوثيقة (Manifest File) التابع لتطبيق ما وذلك بهدف تحديد نوع الأهداف (Intents) التي يمكن للمكون (الفعالية ،الخدمة أو المنشور) استقبالها. على سبيل المثال، بالإعلان عن مرشح هدف (Intent Filter) لفعالية معينة داخل ملف الوثيقة، فإن التطبيقات الأخرى المحملة على الجهاز قد تتمكن من تشغيل هذه الفعالية إذا تم إرسال هدف (Intent) يوافق مرشح الهدف (Intent Filter). إذا لم يتم تحديد مرشح هدف للفعالية فلا يمكن تشغيل هذه الفعالية إلا باستخدام هدف صريح (Explicit Intent).

يمكن أن تعد تطبيقك لاستقبال أنواع محددة من الأهداف (Intents) عن طريق الإعلان عن مرشحات الأهداف (Intent) Filters في ملف الوثيقة (Manifest File) باستخدام الخاصية <intent-filter>.

كل مرشح هدف (Intent Filter) يحدد نوع الهدف (Intent) الذي يمكن لمكون التطبيق استقباله بناءً على نوع الإجراء المحدد في المرشح ونوع البيانات data والتصنيف category. يقوم النظام بإيصال الهدف (Intent) إلى مكون تطبيقك فقط إذا توافقت محتويات الهدف مع تفاصيل مرشح الهدف (Intent Filter). على سبيل المثال، يوضح الكود بالأسفل جزء من ملف الوثيقة (Manifest) والذي يبين مرشح الهدف لفعالية باسم ShareActivity:

```
<activity android:name="ShareActivity">
   <intent-filter>
     <action android:name="android.intent.action.SEND"/>
     <action
android:name="android.intent.action.SEND_MULTIPLE"/>
     <category
android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
</activity>
```

كما هو موضح، فإن مرشح الهدف في هذا المثال يحتوي على ثلاث عناصر:

- الإجراء الذي تنفذه الفعالية (Action)، وهو في هذا المثال من نوع ACTION_SEND أو ACTION_SEND_MULTIPLE.
- التصنيف category وهو في هذا المثال من نوع default، وهو التصنيف التلقائي لأي فعالية غير فعالية ابتداء البرنامج والتي تأخذ تصنيف LAUNCHER.
- نوع البيانات التي تتلقاها الفعالية من خلال الخاصية data (مثال: text/plain).

ذكرنا مسبقاً أنه عند تشغيل فعالية باستهدف هدف مضمن (Implicit Intent)، يقوم النظام تلقائياً بالبحث عن الفعالية المناسبة لتشغيل الإجراء (Action) المحدد في الهدف المضمن (Implicit Intent) ضمن كل التطبيقات المحملة على النظام. إذا توافق محتوى الهدف المرسل (Intent) مع تفاصيل مرشح الهدف (Intent Filter) الخاص بأي فعالية، يقوم النظام تلقائياً بتشغيلها، أو عرض خيارات في حالة وجود أكثر من فعالية يمكنها استقبال الهدف.

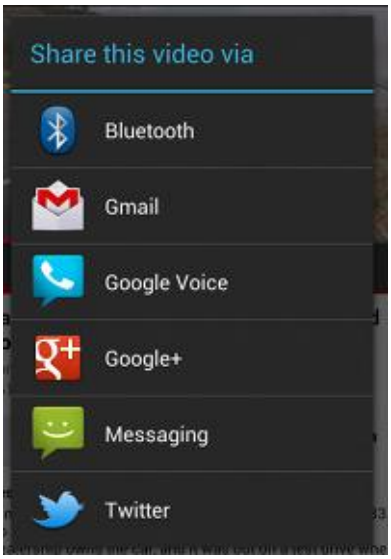
ملاحظة: يمكن تحديد أكثر من إجراء داخل مرشح الهدف، مما يعني أن الفعالية يمكن أن تعمل اذا توافق إي إجراء معرف في المرشح (Intent Filter) مع الإجراء المحدد في الهدف المرسل (Intent).

يوضح الكود التالي كيفية تشغيل فعالية إرسال رسالة باستخدام هدف مضمن (Implicit Intent). عند إنشاء كائن من نوع (Intent)، تم تمرير الإجراء المطلوب وهو إجراء العرض (ACTION_SEND)، ومن ثم تم تضمين البيانات المطلوب عرضها ,وهي الرسالة المطلوب إرسالها, من خلال الدالة ()putExtra. يقوم النظام تلقائياً بتحديد

الفعالية المناسبة للتشغيل عن طريق مطابقة نوع الإجراء ونوع البيانات المرسلة مع مرشح الهدف (Intent Filter) للفعاليات الأخرى . في هذه الحالة سيقوم النظام باختيار فعالية يمكنها عرض إرسال الرسائل.

```
Intent sendIntent =new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType(HTTP.PLAIN_TEXT_TYPE);
startActivity(sendIntent);
```

في حالة وجود أكثر من فعالية يمكنها تنفيذ نفس الإجراء، يقوم النظام بعرض خيارات للمستخدم ليختار منها الفعالية التي يريدها، كما هو موضح في الشكل:



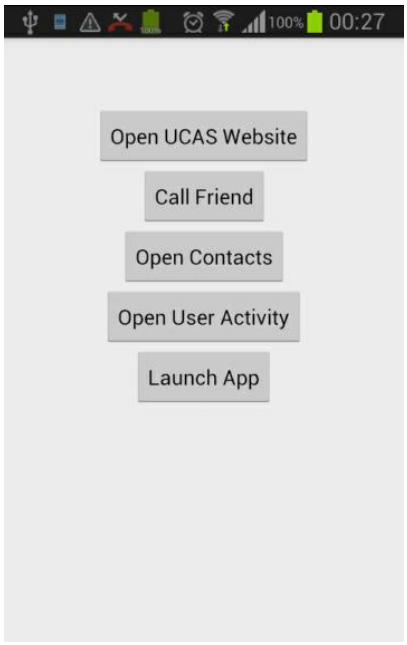
شكل 5-4: خيارات تشغيل متعددة للإجراء ACTION_SEND.[1]

تمرين عملي (3-4)

في هذا التمرين سيتم تشغيل عدة من الفعاليات (Activities) لتنفيذ بعض الإجراءات (Actions) شائعة الاستخدام مثل تشغيل متصفح الانترنت، الاتصال الهاتفي و عرض جهات الاتصال contacts. كل ذلك يتم باستخدام هدف مضمن (Implicit Intent) يحدد فيه الإجراء (Action) المراد تنفيذه. سنجرب أيضاً إمكانية تعريف مرشح هدف (Intent Filter) لفعالية (Activity) ضمن تطبيق تم إنشاؤه مسبقاً.

شكل 6-4 يوضح واجهة التطبيق وهي مكونة من مجموعة من الأزرار، حيث أن كل زر مسئول عن تنفيذ إجراء معين.

[1] Intents and Intent Filters, Android Developer, http://developer.android.com/guide/components/intents-filters.html



شكل 6-4: واجهة التطبيق الخاص بتمرين 3-4.

تصميم الواجهة الخاصة بهذا التصميم موضح بالأسفل:

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"

android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >
    <Button
        android:id="@+id/openWebpageButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="38dp"
        android:text="Open UCAS Website"
```

```
        android:onClick="openUCAS" />
    <Button
        android:id="@+id/callButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/openWebpageButton"
        android:layout_centerHorizontal="true"
        android:text="Call Friend"
        android:onClick="callFriend"
         />
    <Button
        android:id="@+id/contactsButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/callButton"
        android:layout_centerHorizontal="true"
        android:text="Open Contacts"
        android:onClick="openContacts"
         />
     <Button
        android:id="@+id/openActivityButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/contactsButton"
        android:layout_centerHorizontal="true"
        android:text="Open User Activity"
        android:onClick="openActivity" />
    <Button
        android:id="@+id/lauchAppButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/openActivityButton"
        android:layout_centerHorizontal="true"
        android:text="Launch App"
        android:onClick="lauchApp" />
</RelativeLayout>
```

أحد الإجراءات التي يشتمل عليها التطبيق يتضمن فتح قائمة جهات الاتصال (Contancts) من خلال النقر على "Open Contacts"، وهو ما يتطلب إذن permission. لذلك، يتم إضافة الإذن التالي إلى ملف الوثيقة Manifest:

```
<manifest ...>
<uses-permission
android:name="android.permission.CALL_PHONE" />
    ...
</manifest>
```

أحد الإجراءات التي يتم تنفيذها (عند النقر على الزر "Open User Activity") هو تشغيل فعالية (Activity) موجودة ضمن تمرين 2-4 في الوحدة الثالثة (أنظر الشكل 4-4). حتى نتمكن من تشغيل فعالية ضمن تطبيق آخر (الفعالية FreindsListActivity)، يجب أن يتم ذلك باستخدام هدف مضمن (Implicit Intent) (لا يمكن استخدام هدف صريح Explicit Intent) وذلك لأن الفئة (class) الخاصة بالفعالية الهدف ليست في نطاق التطبيق الحالي). لإنشاء هدف مضمن (Implicit Intent)، لابد من تعريف إجراء (Action) للفعالية المراد تشغيلها. لعمل ذلك، يجب إضافة (Intent Filter) للفعالية المراد تشغيلها ضمن ملف الوثيقة (Manifest) الخاص بالتمرين 2-4 كما هو موضح:

```
…
<activity android:name=".FriendsListActivity"
android:label="@string/title_activity_friends_list" >
 <intent-filter>
   <action
android:name="ps.edu.ucas.example.FriendsListActivity" />
   <category android:name="android.intent.category.DEFAULT"
/>
 </intent-filter>
</activity>
…
```

لاحظ أن اسم الإجراء المستخدم ضمن الخاصية android:name هو مسار الفعالية (Activity). يمكن استخدام أي اسم بديل بشرط عدم تطابقه مع اسماء الإجراءات المستخدمة في فعاليات أخرى.

الكود التالي يوضح الفعالية (MainActivity) والتي من خلالها يتم تنفيذ كل الإجراءات المطلوبة.

```
1:  public class MainActivity extends Activity {
2:
3:   @Override
4:   protected void onCreate(Bundle savedInstanceState) {
```

```
5:        super.onCreate(savedInstanceState);
6:        setContentView(R.layout.activity_main);
7:    }
8:
9:    public void openUCAS(View view){
10:       Intent intent = new Intent(Intent.ACTION_VIEW);
11:   intent.setData(Uri.parse("http://www.ucas.edu.ps"));
12:       this.startActivity(intent);
13:   }
14:
15:   public void callFriend(View view){
16:       Intent intent = new Intent(Intent.ACTION_CALL);
17:       intent.setData(Uri.parse("tel:0591234567"));
18:       this.startActivity(intent);
19:   }
20:
21:   public void openContacts(View view){
22:       Intent intent = new Intent(Intent.ACTION_VIEW);
23:   intent.setData(Uri.parse("content://contacts
24:   /people/"));
25:       this.startActivity(intent);
26:   }
27:
28:   public void openActivity(View view){
29:       Intent intent = new
30:   Intent("ps.edu.ucas.example.FriendsListActivity");
31:       this.startActivity(intent);
32:   }
33:   public void lauchApp(View view){
34:       Intent intent = new Intent(Intent.ACTION_MAIN);
35:       intent.setComponent(new
36:   ComponentName("ps.edu.ucas.example","ps.edu.ucas.
37:   example.MainActivity");
38:       intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
39:       this.startActivity(intent);
40:   }
41:}
```

في ما يلي يتم توضيح طريقة تنفيذ كل إجراء:

- الدالة openUCAS() (أنظر الكود من سطر 9 إلى 13) يتم تشغيل متصفح إنترنت Web browser لفتح موقع الانترنت الخاص بالكلية الجامعية. يتم إنشاء هدف (Intent) و تحديد الإجراء (ACTION_VIEW) وتمرير عنوان موقع الكلية عن طريق الدالة setData(). في حالة وجود أكثر من متصفح تظهر قائمة لتمكين المستخدم من اختيار المتصفح المطلوب.

- الدالة callFriend() (أنظر الكود من سطر 15 إلى 19) يتم فيها إجراء اتصال هاتفي برقم محدد. يتم إنشاء هدف (Intent) وتحديد الإجراء (ACTION_CALL) وتمرير الرقم المطلوب الاتصال به عن طريق الدالة setData(). لاحظ أن هذا الإجراء يتطلب إضافة إذن اتصال في ملف الوثيقة (Manifest).

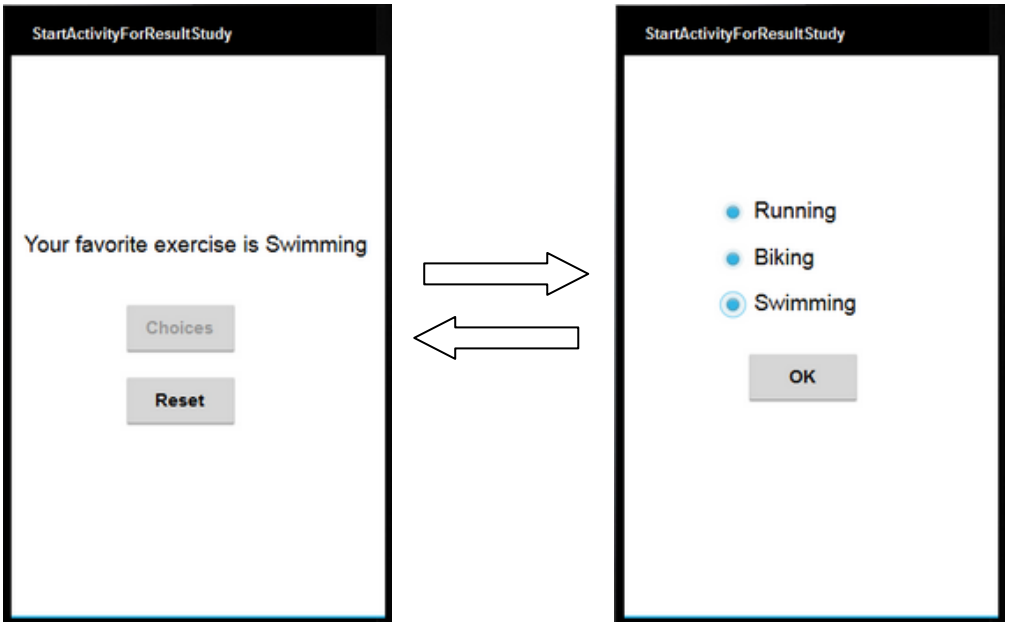
- الدالة openContacts() (أنظر الكود من سطر 21 إلى 26) يتم فيها فتح قائمة جهات الاتصال Contacts. يتم إنشاء هدف (Intent) و تحديد الإجراء (ACTION_VIEW) وتحديد القائمة المراد فتحها (مثل: content://contacts/people/) عن طريق الدالة setData(). لاحظ أن الإجراء (ACTION_VIEW) يتطابق مع الإجراء المستخدم لفتح متصفح الإنترنت. تذكر أنه في حالة تطابق الإجراء بين فعاليات مختلفة، فإن النظام يستخدم البيانات الأخرى ضمن الهدف (Intent) لتحديد الفعالية المطلوب تشغيلها. في هذه الحالة ،الفعالية الخاصة بفتح المتصفح لها نفس الهدف (ACTION_VIEW) الخاص بفتح قائمة جهات الاتصال Contacts، ولكن نوع البيانات المرسلة ضمن الهدف مختلف (في الحالة الأولي يستخدم البروتوكول http:// بينما في الحالة الثانية يستخدم البروتوكول content://). لذلك، يستخدم النظام نوع البيانات للتحديد الإجراء الصحيح.

- الدالة openActivity() (أنظر الكود من سطر 28 إلى 32) يتم من خلالها تشغيل الفعالية FriendsListActivity الموجودة ضمن تطبيق سابق تم إنشاؤه. لاحظ أن الإجراء المستخدم يطابق الإجراء المحدد في مرشح الهدف (Intent Filter) الذي تم تعريفه مسبقا ضمن ملف الوثيقة (Manifest) الخاص بالتطبيق الآخر.

- الدالة launchApp() (أنظر الكود من سطر 33 إلى 40) توضح طريقة تشغيل تطبيق آخر. في هذه الحالة فإن الإجراء المستخدم يكون من نوع (ACTION_MAIN) وهو يعني تشغيل فعالية رئيسية (تذكر أن كل تطبيق له فعالية رئيسية واحدة فقط). الدالة setComponent تحديد المجلد package الخاص بالتطبيق المطلوب تشغيله وكذلك مسار الفئة (classpath) الخاصة بالفعالية الرئيسية للتطبيق. الأمر setFlags(Intent.FLAG_ACTIVITY_NEW_TASK) يعني أن الفعالية سيتم تشغيلها كتطبيق مستقل غير مرتبط بالتطبيق الحالي.

أسئلة على الوحدة الرابعة

1. ما الفرق بين الهدف الصريح (Explicit Intent) والمضمن (Implicit Intent)؟ ومتى يستخدم كل منهما.
2. ما هي المكونات الأساسية التي يتم تعريفها داخل مرشح الهدف (Intent Filter)؟
3. ماذا يحدث إذا كان هناك فعاليتان (Activities) لهما نفس نوع الحدث (Action) داخل مرشح الهدف (Intent Filter)؟
4. قم ببناء تطبيق مكون من فعاليتين (Activities) كما هو موضح بالشكل. الفعالية الرئيسية على اليسار تحتوي على زر "Choices" عند النقر عليه يتم تشغيل الفعالية على اليمين. يقوم المستخدم من خلال الفعالية الجديدة باختيار هوايته المفضلة من ضمن مجموعة من الخيارات الأخرى (يسمح باختيار واحد فقط). عند النقر على زر "OK" ضمن الفعالية الجديدة يتم إرجاع الإختيار إلى الفعالية الرئيسية وعرضه ضمن عنصر عرض (TextView). عند النقر على زر "Reset" يتم حذف النص المكتوب ضمن عنصر العرض (TextView).



الوحدة الخامسة:

القوائم وشريط العمل

(Menus and Action Bar)

يتعلم الطالب في هذه الوحدة:

- ✓ أنواع القوائم المختلفة في تطبيق الأندرويد (قائمة الخيارات Options Menu، قائمة السياق Content Menu، والقائمة المنبثقة) واستخداماتها.
- ✓ إنشاء القوائم المختلفة برمجياً وربطها بالتطبيق
- ✓ تهيئة شريط العمل (Action Bar) والتحكم به برمجياً.
- ✓ إنشاء درج التصفح (Navigation Drawer) وربطه بالتطبيق

القوائم Menus من المكونات شائعة الاستخدام في تصميم واجهات التطبيقات. تتطرق هذه الوحدة إلى الأنواع المختلفة للقوائم وطريقة إنشائها.

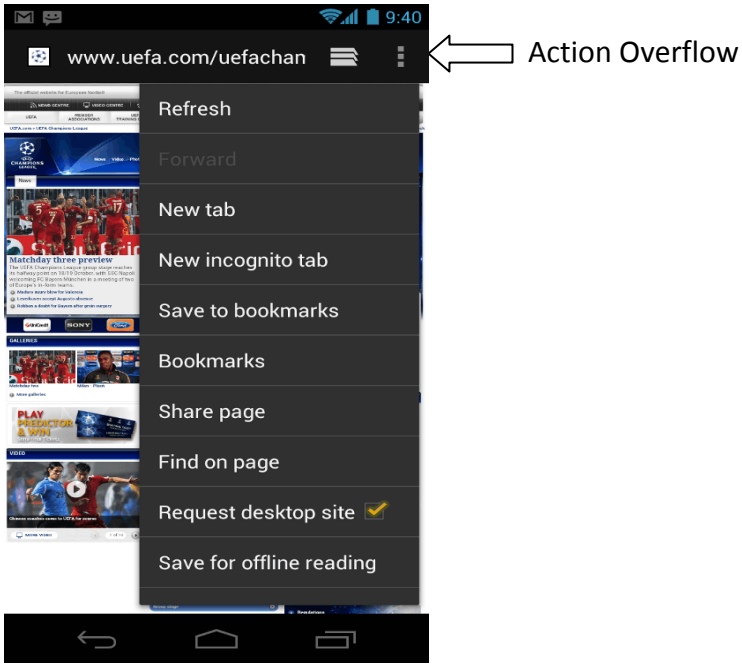
قائمة الخيارات (Options Menu) وشريط المهام (Action Bar)

تستخدم قائمة الخيارات Options Menu عادةً للوصول إلى الإجراءات العامة التي يتطلب الوصول إليها كثيراً أثناء استخدام التطبيق مثل إجراء البحث Search والإعدادات Settings. (ابتداءً من اصدار أندرويد 3.0 API level 11) يتم تضمين قائمة الخيارات ضمن شريط العمل (Action Bar)). يقوم النظام تلقائياً بإضافة عناصر القائمة الخيارات (Options Menu) إلى ملحق شريط العمل المسمى (Action Overflow) (أنظر شكل 5.1). لا تظهر عناصر القائمة تلقائياً، حيث يمكن عرضها بالنقر على اللإيقونة على يمين شريط العمل. يمكن أيضاً إظهار قائمة الخيارات بالنقر على زر "Menu" إذا كان الجهاز يوفر مثل هذا الزر.

لإنشاء قائمة اختيارات Options Menu لفعالية معينة (Activity)، يمكن في البداية تصميم محتويات القائمة خارجياً باستخدام XML. على الرغم من إمكانية بناء محتويات القائمة برمجياً، إلا أنه يفضل تصميم القائمة باستخدام XML وحفظها في مجلد المصادر. يمكن بعد ذلك إنشاء القائمة عند تشغيل الفعالية عن طريق الدالة ()Inflater.inflate. إنشاء القائمة في ملف XML يتيح إمكانية فصل تصميم القائمة عن عمل التطبيق. كما يتيح عمل إعدادات أو تصميمات مختلفة للقائمة بناءً على إعدادات الجهاز المختلفة مثل حجم شاشة العرض. كما ذكرنا في الوحدة الثالثة يمكن مواءمة التطبيق مع الإعدادات المختلفة عن طريق حفظ وترتيب الملفات الموجودة في مجلدات المصادر.

لتصميم قائمة خيارات جديدة، قم بإنشاء ملف XML جديد في المجلد /res/menu وقم ببناء القائمة داخله. الملف التالي my_menu.xml يوضح مثال لتصميم قائمة خيارات:

```
<?xml version="1.0" encoding="utf-8"?>
<menu
xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/ucas_homepage"
          android:icon="@drawable/ucas_logo"
          android:title="@string/ucas_homepage"
          android:showAsAction="ifRoom"/>
    <item android:id="@+id/help"
          android:icon="@drawable/ic_help"
          android:title="@string/help" />
</menu>
```



شكل 1-5: قائمة الخيارات (OptionsMenu)

كما هو موضح في المثال، تصميم القائمة كاملةً يكون ضمن الخاصية <menu> والتي تحتوي على عنصر أو أكثر من الخصائص <item> والتي تمثل عناصر القائمة. الخاصية <item> تستخدم لبناء عنصر القائمة (MenuItem).

من داخل الخاصية <item> يتم تعريف شكل عنصر القائمة (MenuItem) باستخدام الخصائص التالية:

- android:id : وهو يحدد رقم معرف (ID) خاص بعنصر القائمة (MenuItem). الرقم المعرف (ID) يمكّن التطبيق من معرفة عنصر القائمة (MenuItem) الذي تم النقر عليه، حيث كل عنصر قائمة له معرف خاص به.
- android:icon : وهو يضاف إختياريا لتحديد صورة الأيقونة الخاصة بعنصر القائمة. يجب أن تكون هذه الصورة ضمن مجلدات drawable.
- android:title: وهو يحدد النص المعروض في العنصر.
- android:showAsAction: وهو يحدد كيف ومتى يظهر عنصر القائمة ضمن شريط العمل (Action Bar). هناك قيم مختلفة يمكن تحديدها لهذه الخاصية مثل القيمة "ifRoom" وذلك لإظهار عنصر القائمة ضمن شريط العمل إذا كان هناك متسع به، وكذلك القيمة "never" لعدم إظهاره ضمن شريط العمل، والقيمة "always" لإظهاره دائماً ضمن شريط العمل

تمثل هذه أهم الخصائص التي ينصح باستخدامها، ولكن هناك العديد من الخصائص الأخرى التي لن نتطرق لها في هذا الكتاب.

يمكن أيضاً إضافة قائمة فرعية (submenu) داخل أي عنصر في القائمة (Menu) وذلك بإضافة الخاصية <menu> ضمن الخاصية <item>. القوائم الفرعية (submenus) مفيدة في حالة تعدد وتفرع الإجراءات التي تنفذ من خلال القائمة أو لترتيب عناصر القائمة ضمن مجموعات. المثال التالي يوضح إضافة قائمة فرعية إلى عنصر قائمة:

```
<?xml version="1.0" encoding="utf-8"?>
<menu
xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/file"
          android:title="@string/file" >
        <!-- "file" submenu -->
        <menu>
            <item android:id="@+id/create_new"
                  android:title="@string/create_new" />
            <item android:id="@+id/open"
                  android:title="@string/open" />
        </menu>
    </item>
</menu>
```

بعد تصميم قائمة الخيارات (Options Menu) يتم استخدامها في الفعالية (Activity) عن طريق الدالة MenuInflater.inflate() والتي نقوم بتحويل التصميم في ملف XML إلى عنصر واجهة يتم الوصول إليه برمجياً. يتم كتابة الدالة onCreateOptionsMenu()، وهي أحدى الدوال الموجودة ضمن الفعالية (Activity)

والتي يتم تنفيذها تلقائياً عند إنشاء الفعالية، وذلك لبناء القائمة من ملف XML باستخدام الكائن MenuInflater كما هو موضح:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.my_menu, menu);
    return true;
}
```

معالجة أحداث القائمة (Handling Click Events)

عندما يختار المستخدم عنصر من قائمة الخيارات (Options Menu) (بما فيها العناصر في شريط العمل Action Bar)، يقوم النظام بتنفيذ الدالة ()onOptionsItemSelected والتي يمرر لها عنصر القائمة MenuItem الذي تم اختياره (أنظر الكود بالأسفل). يمكن تمييز هذا العنصر عن طريق تنفيذ الدالة ()getItemId والتي ترجع المعرف ID الخاص به والمنشأ باستخدام الخاصية android:id في ملف XML الخاص بتصميم القائمة. بعد معالجة الحدث يتم إرجاع القيمة true. إذا لم يتم معالجة الحدث يجب تنفيذ الدالة ()onOptionsItemSelected الموجودة في الفئة الأم (superclass).

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
        case R.id.new_game:
            newGame();
            return true;
        case R.id.help:
            showHelp();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

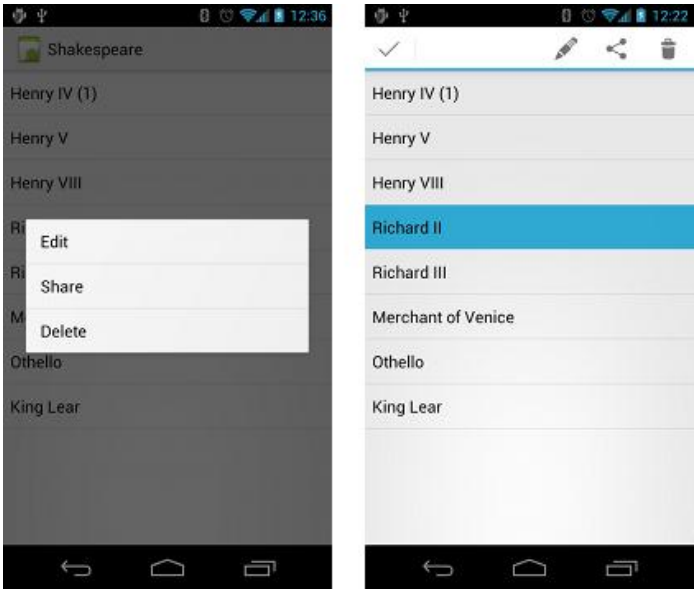
يوفر نظام أندرويد أيضاً إمكانية تعريف دالة لمعالجة حدث النقر click من خلال ملف XML الخاص بتصميم القائمة باستخدام الخاصية android:onClick. القيمة التي تأخذها هذه الخاصية هي اسم الدالة التي تعالج الحدث والتي يجب تكون عامة public ويمرر لها متغير من نوع MenuItem.

كما ذكرنا مسبقاً، يتم إنشاء القائمة Menu من خلال الكود الموجود في الدالة ()onCreateOptionsMenu. إذا أردت تعديل محتويات القائمة بالإضافة أو التعديل أو الحذف أثناء عمل الفعالية (Activity)، يمكن القيام بذلك من خلال الدالة ()onPrepareOptionsMenu. هذه الدالة يمرر إليها كائن من النوع Menu والخاص بالقائمة التي تم انشاؤها مسبقاً في الدالة ()onCreateOptionsMenu، حيث يمكن التعديل عليها بالإضافة أو الحذف.

قائمة السياق (Context Menu)

قائمة السياق Context Menu يتم من خلالها تنفيذ الإجراءات التي تؤثر على عنصر محدد في واجهة المستخدم. يمكنك ربط قائمة السياق Context Menu بأي عنصر واجهة View، ولكن غالبا ما تستخدم لتنفيذ اجراءات على العناصر في القائمة (ListView) أو (GridView)، أو غيرها من عناصر عرض المجموعات التي يمكن للمستخدم تنفيذ إجراءات مباشرة على كل بند فيها. هناك طريقتان لعرض قائمة السياق (Context Menu)، كما هو موضح بشكل 2-5، وهما:

- Floating Context Menu: حيث تظهر كقائمة عائمة عندما يقوم المستخدم بنقرة وطويل على عنصر واجهة View. باستخدام هذا الوضع (Floating Context Menu)، يمكن تنفيذ الإجراءات على عنصر واحد فقط في الوقت الواحد.
- Contextual Action Mode: وفيه يتم عرض عناصر القائمة (Menu)،التي تؤثر على العنصر الذي تم اختياره ،ضمن شريط العمل (Action Bar) في الجزء العلوي من الشاشة. عندما يكون هذا الوضع مفعلاً، يمكن للمستخدمين تنفيذ إجراء على عناصر متعددة في وقت واحد (إذا كان التطبيق الخاص بك يسمح بذلك) (لن يتم التطرق للقوائم من نوع Contextual Action Mode في هذا الكتاب).



شكل 2-5: قائمة سياق عائمة (Floating Context Menu) على اليسار، وشريط عمل السياق (Contextual Action Bar) على اليمين.

إنشاء قائمة سياق عائمة (Floating Context Menu)

يتم إنشاء قائمة السياق العائمة (Floating Context Menu) باستخدام الخطوات التالية:

1. كما ذكرنا مسبقاً، قائمة السياق العائمة ترتبط بعنصر واجهة (View) محدد وتظهر عند النقر عليه. عنصر الواجهة (View) المراد ربط قائمة السياق (Context Menu) به يجب تسجيله بتنفيذ الدالة registerForContextMenu() ضمن الفعالية (Activity) وتمرير العنصر (View) لها. إذا كانت الفعالية (Activity) تستخدم قائمة عرض (ListView) وأردت استخدام نفس قائمة السياق Context Menu لكل القائمة (ListView)، يجب في هذه الحالة تسجيل القائمة (ListView) بتمريرها للدالة registerForContextMenu().
2. يتم كتابة كود في الدالة onCreateContextMenu() والموجودة ضمن الفعالية (Activity). يتم من خلال هذه الدالة إنشاء القائمة من مصدرها في ملف XML وذلك باستخدام كائن من نوع MenuInflater كما هو موضح في المثال التالي:

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
                                ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.context_menu, menu);
}
```

لاحظ أن الدالة onCreateContextMenu() يمرر لها ،بالإضافة للقائمة (ContextMenu)، العنصر View والذي تم اختياره لتفعيل قائمة السياق (Context Menu) وكائن من النوع ContextMenu.ContextMenuInfo والذي يحتوي على معلومات إضافية خاصة بالعنصر الذي تم اختياره. هذه المعلومات يتم استخدامها اذا كان هناك عناصر Views متعددة وكل منها مرتبط بقائمة سياق Context Menu مختلفة. في هذه الحالة تفيد هذه المعلومات في تحديد القائمة التي يجب إنشاؤها لكل عنصر عرض View.

3. كتابة كود الدالة onContextItemSelected() ضمن الفعالية (Activity) وذلك لتحديد الإجراء المراد تنفيذه عند النقر على أي من عناصر القائمة Context Menu. المثال التالي يوضح ذلك:

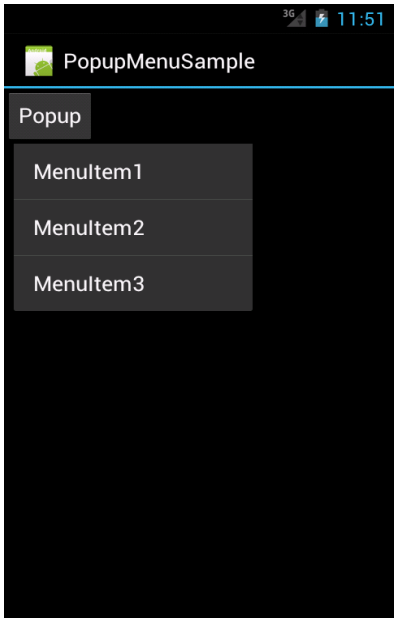
```
@Override
public boolean onContextItemSelected(MenuItem item) {
    AdapterContextMenuInfo info = (AdapterContextMenuInfo)
item.getMenuInfo();
    switch (item.getItemId()) {
        case R.id.edit:
            editNote(info.id);
```

```
            return true;
        case R.id.delete:
            deleteNote(info.id);
            return true;
        default:
            return super.onContextItemSelected(item);
    }
}
```

القائمة المنبثقة (Popup Menu)

القائمة المنبثقة (Popup Menu) (أنظر شكل 5.3) هي قائمة ملحقة بعنصر واجهة View وتظهر أسفله (أو أعلاه إلى لم تتوفر مساحة لعرضها) كما بشكل. القائمة المنبقة (Popup Menu) يمكن أن تستخدم في الحالات التالية:

1. إظهار قائمة من الإجراءات المتعلقة بمحتوى معين. على سبيل المثال، عند النقر على زر في شريط العمل (Action Bar) يتم اظهار قائمة منبثقة تتضمن مجموعة من الإجراءات.
2. إظهار قائمة من الإجراءات التفصيلية التي تتفرع من إجراء أساسي. فمثلاً، عند النقر على زر "إرسال"، تظهر قائمة منبثقة من إجراءات "إرسال" أخرى مثل إرسال إلى الجميع، إرسال إلى صديق، إرسال إلى مجموعة، وهكذا.



شكل 3-5: القائمة المنبثقة (Popup Menu)

لاحظ أن القائمة المنبقة متوفرة في ٌإصدار أندرويد API Level 11 وما بعده.

إظهار القائمة المنبثقة (Popup Menu) يتم بالخطوات التالية:

1. أنشئ كائن من نوع PopupMenu بحيث يمرر إلى الباني constructor الخاص بها الكائن Context الخاص بالتطبيق بالإضافة إلى عنصر الواجهة view التي يراد إلحاق القائمة به كما هو موضح:
2. استخدام كائن من نوع MenuInflater لإنشاء القائمة باستخدام ملف التصميم XML الخاص بالقائمة.
3. إظهار القائمة بتنفيذ الدالة Poupmenu.show().

على سبيل المثال، الكود بالأسفل يوضح التصميم الخاص بزر Button والذي عند النقر عليه يتم تنفيذ الدالة showPopup والمسؤولة عن إظهار قائمة منبثقة (Popup Menu).

```
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_overflow_holo_dark"

android:contentDescription="@string/descr_overflow_button"
    android:onClick="showPopup" />
```

الكود التالي يوضح الدالة showPopup() داخل الفعالية (Activity):

```
public void showPopup(View v) {
    PopupMenu popup = new PopupMenu(this, v);
    MenuInflater inflater = popup.getMenuInflater();
    inflater.inflate(R.menu.actions, popup.getMenu());
    popup.show();
}
```

ملاحظة: إبتداءً من الإصدار API Level 14 يمكن استخدام الدالة PopupMenu.inflate() بدلاً من استخدام MenuInflater.inflate().

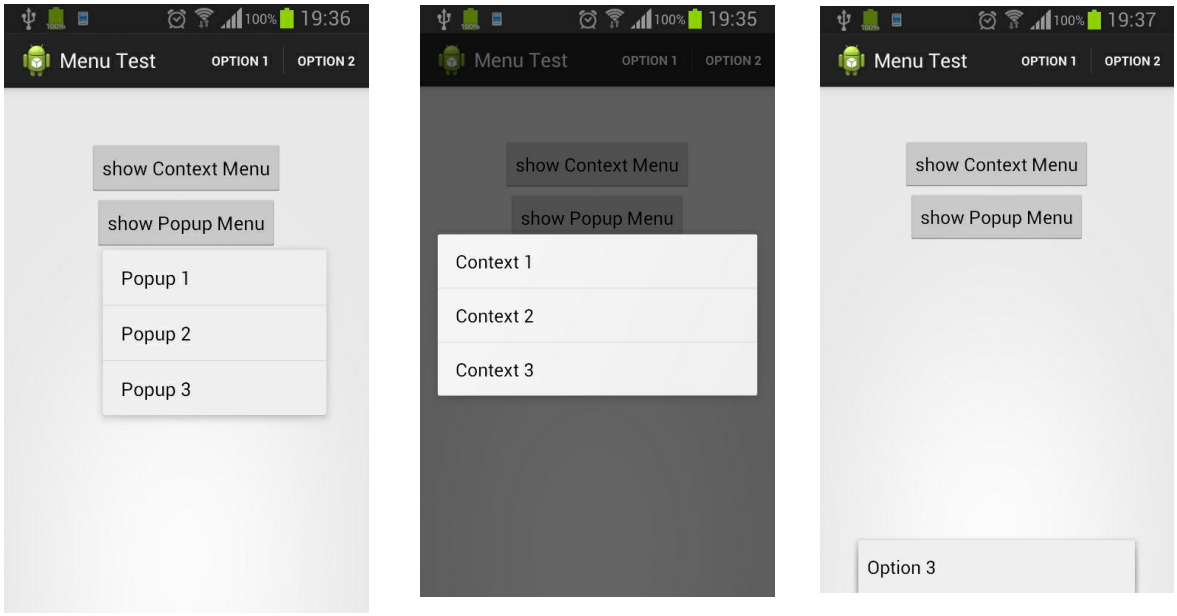
لتنفيذ إجراء معين عند النقر على أي عنصر في القائمة المنبثقة Popup يجب إنشاء مستمع (Listener) من نوع MenuPopupMenu.OnMenuItemClickListener وتسجيله باستخدام الدالة PopupMenu.setOnMenuItemClickListener() ومن ثم كتابة كود الإجراء المطلوب في الدالة onMenuItemClick(). الكود التالي يوضح هذه الخطوة:

```
public void showMenu(View v) {
    PopupMenu popup = new PopupMenu(this, v);

    popup.setOnMenuItemClickListener(new
OnMenuItemClickListener(){

     @Override
public boolean onMenuItemClick(MenuItem item) {
      switch (item.getItemId()) {
          case R.id.archive:
              archive(item);
              return true;
          case R.id.delete:
              delete(item);
              return true;
          default:
              return false;
      }
    }

  });
  popup.inflate(R.menu.actions);
  popup.show();
}
```

تمرين عملي (1-5)

يهدف هذا التمرين إلى التدريب على إنشاء أنواع القوائم (Menus) المختلفة التي تم تفصيلها سابقاً في هذه الوحدة. واجهة التطبيق والقوائم التي يتم عرضها موضحة في شكل 4-5. يستخدم التطبيق قائمة خيارات (Options Menu) تظهر في شريط العمل (option 1, option 2,..) بينما تظهر بقية العناصر عند النقر على أيقونة Action Overflow أو النقر على زر "Menu" في الجهاز.

تحتوي واجهة التطبيق على زرين: "Show Context Menu"، "Show Popup Menu" عند النقر عليهما تظهر قائمة السياق (Context Menu) و القائمة المنبثقة (Popup Menu) كما هو موضح في شكل 4-5. عند النقر على عنصر في أي من القوائم الثلاث يتم طباعة رسالة باستخدام Toast تظهر اسم العنصر.



شكل 4-5: واجهة التطبيق الخاص بتمرين 1-5: قائمة الخيارات Options Menu تظهر في شريط العمل Action Bar (يمين)، قائمة السياق Context Menu (وسط)، القائمة المنبثقة Popup Menu (يسار)

تصميم الواجهة الرئيسية (MainActivity) موضح فيما يلي:

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"

android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.menutest.MainActivity" >

    <Button
        android:id="@+id/contextBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
```

```
        android:layout_centerHorizontal="true"
        android:text="show Context Menu"
        android:layout_marginTop="30dp" />

    <Button
        android:id="@+id/popupBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/contextBtn"
        android:layout_centerHorizontal="true"
        android:text="show Popup Menu"
        android:onClick="showPopup" />

</RelativeLayout>
```

ملفات XML الخاصة بالقوائم الثلاثة موضحة فيما يلي:

أولاً: قائمة الخيارات (Options Menu)

```
//options_menu.xml
<?xml version="1.0" encoding="utf-8"?>
<menu
xmlns:android="http://schemas.android.com/apk/res/android"
>
    <item android:id="@+id/option1" android:title="Option
1" android:showAsAction="ifRoom"></item>
    <item android:id="@+id/option2" android:title="Option
2" android:showAsAction="ifRoom"></item>
    <item android:id="@+id/option3" android:title="Option
3" android:showAsAction="ifRoom"></item>
</menu>
```

ثانياً: قائمة السياق (Context Menu)

```
// context_menu.xml
<?xml version="1.0" encoding="utf-8"?>
<menu
xmlns:android="http://schemas.android.com/apk/res/android"
>
    <item android:id="@+id/context1" android:title="Context
1"></item>
    <item android:id="@+id/context2" android:title="Context
2"></item>
```

```
    <item android:id="@+id/context3" android:title="Context
3"></item>
</menu>
```

ثالثاً: القائمة المنبثقة (Popup Menu)

```
<?xml version="1.0" encoding="utf-8"?>
<menu
xmlns:android="http://schemas.android.com/apk/res/android"
>
    <item android:id="@+id/popup1" android:title="Popup
1"></item>
    <item android:id="@+id/popup2" android:title="Popup
2"></item>
    <item android:id="@+id/popup3" android:title="Popup
3"></item>
</menu>
```

الكود التالي يوضح الفعالية (MainActivity):

```
1:  public class MainActivity extends Activity {
2:
3:   @Override
4:   protected void onCreate(Bundle savedInstanceState) {
5:        super.onCreate(savedInstanceState);
6:        setContentView(R.layout.activity_main);
7:
8:  // Linking the button contextBtn with the context menu
9:        Button contextBtn = (Button)
10: this.findViewById(R.id.contextBtn);
11:       this.registerForContextMenu(contextBtn);
12:  }
13:
14:  @Override
15:  public boolean onCreateOptionsMenu(Menu menu) {
16:
17:       getMenuInflater().inflate(R.menu.options_menu,
18:  menu);
19:       return true;
20:  }
```

```
21:
22:  @Override
23:  public boolean onOptionsItemSelected(MenuItem item) {
24:     switch (item.getItemId()) {
25:       case R.id.option1:
26:            Toast.makeText(this, "Option 1 selected",
27:  Toast.LENGTH_LONG).show();
28:            return true;
29:       case R.id.option2:
30:           Toast.makeText(this, "Option 2 selected",
31:  Toast.LENGTH_LONG).show();
32:            return true;
33:       case R.id.option3:
34:           Toast.makeText(this, "Option 3 selected",
35:  Toast.LENGTH_LONG).show();
36:            return true;
37:       default:
38:           return super.onOptionsItemSelected(item);
39:     }
40:  }
41:
42:  @Override
43:  public void onCreateContextMenu(ContextMenu menu, View
44:  v, ContextMenuInfo menuInfo) {
45:       super.onCreateContextMenu(menu, v, menuInfo);
46:  this.getMenuInflater().inflate(R.menu.context_menu,
47:  menu);
48:  }
49:
50:  @Override
51:  public boolean onContextItemSelected(MenuItem item) {
52:       switch (item.getItemId()) {
53:         case R.id.context1:
54:              Toast.makeText(this, "Context 1 selected",
55:              Toast.LENGTH_LONG).show();
56:              return true;
57:         case R.id.context2:
58:             Toast.makeText(this, "Context 2 selected",
59:  Toast.LENGTH_LONG).show();
60:             return true;
```

```
61:         case R.id.context3:
62:             Toast.makeText(this, "Context 3 selected",
63: Toast.LENGTH_LONG).show();
64:             return true;
65:         default:
66:             return super.onContextItemSelected(item);
67:          }
68:  }
69:
70:  public void showPopup(View view) {
71:      PopupMenu popup = new PopupMenu(this, view);
72:      popup.setOnMenuItemClickListener(new
73:  OnMenuItemClickListener(){
74:
75:            @Override
76:            public boolean onMenuItemClick(MenuItem
77:  item) {
78:                    switch (item.getItemId()) {
79:                  case R.id.popup1:
80:
81:  Toast.makeText(getApplicationContext(), "Popup 1
82:  selected", Toast.LENGTH_LONG).show();
83:                       return true;
84:                  case R.id.popup2:
85:
86:  Toast.makeText(getApplicationContext(), "Popup 2
87:  selected", Toast.LENGTH_LONG).show();
88:                       return true;
89:                  case R.id.popup3:
90:
91:  Toast.makeText(getApplicationContext(), "Popup 3
92:  selected", Toast.LENGTH_LONG).show();
93:                       return true;
94:                  default:
95:                       return false;
96:                    }
97:            }
98:      });
99:      MenuInflater inflater = popup.getMenuInflater();
100:     inflater.inflate(R.menu.popup_menu,
```

```
101:  popup.getMenu());
102:     popup.show();
103: }
104:}
```

لاحظ أنه في الدالة ()onCreate عند تشغيل الفعالية، يتم الوصول للزر "contextButton" وربطه بقائمة السياق عند طريقة الدالة ()registerForContextMenu (أنظر سطر رقم 11).

قائمة الخيارات (Options Menu) يتم إنشاؤها ضمن الدالة ()onCreateOptionsMenu (أنظر الكود من سطر 15 إلى 20) ويتم معالجة الأحداث المرتبطة بها ضمن الدالة ()onOptionsItemSelected (أنظر الكود من سطر 23 إلى 40).

قائمة السياق (Context Menu) يتم إنشاؤها ضمن الدالة ()onCreateContextMenu (أنظر الكود من سطر 43 إلى 48) ويتم معالجة الأحداث المرتبطة بها ضمن الدالة ()onContextItemSelected (أنظر الكود من سطر 51 إلى 68).

القائمة المنبثقة (Popup Menu) يتم إنشاؤها ضمن الدالة ()showPopup (أنظر الكود من سطر 70 إلى 103) والتي تنفذ عن النقر على الزر المخصص.

درج التصفح (Navigation Drawer)

درج التصفح (Navigation Drawer) هو لائحة تشتمل على عناصر للتحكم بالتطبيق، مثل الأزرار أو القائمة (ListView)، علي يمين أو يسار الشاشة. تكون هذه اللائحة مخفية تلقائياً ويتم إظهارها عند السحب (swipe) باتجاه العرض (أنظر شكل 5-5). درج التصفح من الأدوات شائعة الاستخدام في تطبيقات أندرويد.

لإنشاء درج تصفح (Navigation Drawer) ضمن تطبيقك، يجب تصميم الواجهة باستخدام تصميم من نوع DrawerLayout. يجب أن يحتوي التصميم على جزئين:

1. المحتوى الأساسي (Main Content View): هذا الجزء يمثل المحتوى الأساسي للواجهة والذي يظل ثابتاً، ويجب أن يأتي في الترتيب الأول داخل ملف التصميم. يمكن أن يكون هذا الجزء عبارة عن تصميم (Layout) داخل التصميم (DrawerLayout) يحوي داخله مجموعة من عناصر العرض .
2. محتوى الدرج (Drawer Content) الجزء الثاني من التصميم DrawerLayout يمثل محتوى درج التصفح والذي يكون مخفياً ويظهر عند السحب فقط. يمكن أن يكون محتوى درج التصفح عبارة عن عنصر عرض وحيد (مثل ListView) أو مجموعة من العناصر يجمعها تصميم واحد (Layout) . محتوى الدرج يجب أن يستخدم خاصية XML التالية: android:layout_gravity والتي تأخذ القيمة "start" أو "end". هذه الخاصية تعني أن محتوى العرض يقع على يسار الشاشة (”start“) أو يمينها (”end“). كما يجب أن يستخدم محتوى الدرج الخاصية android:layout_width لتحديد عرض درج الحوار عند عرضه. لاحظ أن درج التصفح يجب أن لايخفي المحتوى الأساسي كاملاً عند عرضه.

شكل 5-5: درج التصفح (Navigation Drawer) [1]

المثال التالي يوضح تصميم لواجهة بسيطة: لاحظ أن تصميم الواجهة يستخدم DrawerLayout. بداخله تم استخدام تصميم من نوع RelativeLayout (أو أي تصميم آخر) والذي يجب أن تحتوي العناصر الأساسية الثابتة للواجهة (Main Content View). الجزء الثاني من التصميم DrawerLayout هو قائمة العرض (ListView) والتي تمثل محتويات درج التصفح. لاحظ أن القائمة تستخدم الخاصية android:layout_gravity.

```
<android.support.v4.widget.DrawerLayout

xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:theme="@android:style/Theme.WithActionBar"
    >
     <RelativeLayout >
     <!—Place main UI components here -->
     </RelativeLayout>
```

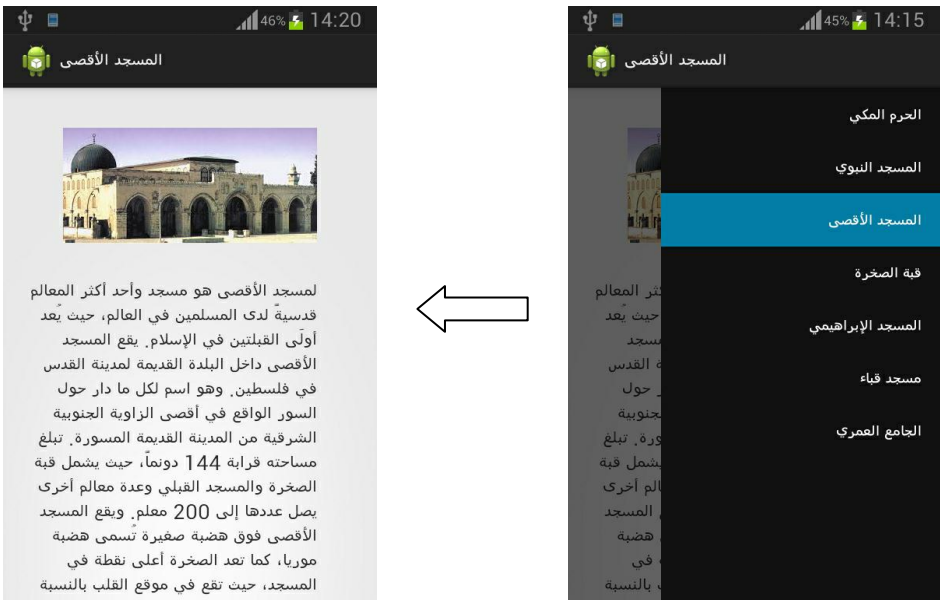
[1] Navigation Drawer, Android Developers, https://developer.android.com/design/patterns/navigation-drawer.html

```
    <ListView
        android:id="@+id/left_drawer"
         android:layout_gravity="start"
        android:layout_width="250dp"
        android:layout_height="match_parent"
        android:choiceMode="singleChoice"
        android:divider="@android:color/transparent"
        android:dividerHeight="0dp"
        android:background="#111"/>
</android.support.v4.widget.DrawerLayout>
```

ملاحظة: يمكن إدراج المحتوى الرئيسي (Main Content View) ضمن تصميم خارجي يسمى بالقطعة (Fragment) ومن ثم إدراج القطعة (Fragment) داخل التصميم DrawerLayout. سيتم التطرق للقطع (Fragments) في الوحدة التاسعة.

تمرين عملي (2-5)

في هذا التمرين سنقوم ببناء تطبيق معلوماتي عن المساجد الشهيرة في فلسطين وواجهته موضحة في شكل 6-5. يستخدم التطبيق درج التصفح (Navigation Drawer) لعرض قائمة بالمساجد، وعند النقر على اسم مسجد تظهر تفاصيله ضمن الواجهة الرئيسية. درج التصفح يكون مخفياً، ويتم عرضه عن طريق السحب من اليمين لليسار.



شكل 6-5: واجهة التطبيق الخاص بتمرين 2-5.

نقوم في البداية بإنشاء مصفوفة نصوص string-array باسم mosques_array ضمن المصادر تحتوي أسماء المساجد لعرضها ضمن قائمة (ListView) داخل درج التصفح (Navigation Drawer). سيتم لاحقاً الوصول لهذه المصفوفة برمجياً لتعبئة القائمة (ListView).

```
<resources>
    <string name="app_name">Navigation Drawer
Example</string>
    <string-array name="mosques_array">
        <item>المكي الحرم</item>
        <item>النبوي المسجد</item>
        <item>الأقصى المسجد</item>
        <item>الصخرة قبة</item>
        <item>الإبراهيمي المسجد</item>
        <item>قباء مسجد</item>
        <item>العمري الجامع</item>
    </string-array>
</resources>
```

فيما يلي نعرض تصميم الواجهة الرئيسية والتي تتكون من المحتوى الرئيسيي (تفاصيل المسجد) و درج التصفح (Navigation Drawer):

```
//activity_main.xml
<android.support.v4.widget.DrawerLayout

xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:theme="@android:style/Theme.WithActionBar"
    >
     <ScrollView android:layout_width="fill_parent"
android:layout_height="fill_parent">
          <RelativeLayout
              android:layout_width="match_parent"
              android:layout_height="match_parent"
android:paddingBottom="@dimen/activity_vertical_margin"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin">
              <ImageView
                  android:id="@+id/imageView1"
```

```
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_alignParentTop="true"
                    android:layout_centerHorizontal="true"
                    android:layout_marginTop="20dp"
                    android:src="@drawable/ic_launcher" />
                <TextView
                    android:id="@+id/textView1"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"

android:layout_alignRight="@+id/imageView1"
                    android:layout_below="@+id/imageView1"

android:layout_gravity="center_vertical|right"
                    android:textSize="20sp"
                    android:layout_marginTop="30dp"
                    android:text="TextView" />

            </RelativeLayout>
     </ScrollView>
    <ListView
        android:id="@+id/right_drawer"
         android:layout_gravity="end"
        android:layout_width="250dp"
        android:layout_height="match_parent"
        android:choiceMode="singleChoice"
        android:divider="@android:color/transparent"
        android:dividerHeight="0dp"
        android:background="#111"/>
</android.support.v4.widget.DrawerLayout>
```

لاحظ أن التصميم السابق (DrawerLayout) مكون من جزئين: الجزء الأول هو التصميم ScrollView ويحتوي داخله على تصميم RelativeView يضم عناصر العرض ImageView و TextView. العنصر ImageView يستخدم لعرض صورة المسجد بينما العنصر TextView يعرض معلومات عن المسجد. الجزء الثاني هو قائمة العرض (ListView) والتي تستخدم الخاصية "android:layout_gravity="end وذلك حتى يتم اخفاؤها في يمين الشاشة.

الكود التالي خاص بالفعالية MainActivity والتي يتم من خلالها إنشاء الواجهة ودرج التصفح ومعالجة الأحداث المختلفة:

```
1:  public class MainActivity extends Activity {
2:      private DrawerLayout mDrawerLayout;
3:      private ListView mDrawerList;
4:      private ImageView mosqueImage;
5:      private TextView mosqueDetails;
6:      private String[] mosqueNames;
7:
8:      @Override
9:      protected void onCreate(Bundle savedInstanceState)
10:    {
11:        super.onCreate(savedInstanceState);
12:        setContentView(R.layout.activity_main);
13:
14:        mosqueNames =
15:  getResources().getStringArray(R.array.mosques_array);
16:        mDrawerLayout = (DrawerLayout)
17:  findViewById(R.id.drawer_layout);
18:        mosqueImage = (ImageView)
19:  findViewById(R.id.imageView1);
20:        mosqueDetails = (TextView)
21:  findViewById(R.id.textView1);
22:        mDrawerList = (ListView)
23:  findViewById(R.id.right_drawer);
24:        mDrawerList.setAdapter(new
25:  ArrayAdapter<String>(this, R.layout.drawer_list_item,
26:  mosqueNames));
27:        mDrawerList.setOnItemClickListener(new
28:  ListView.OnItemClickListener() {
29:            @Override
30:            public void onItemClick(AdapterView<?>
31:  parent, View view, int position, long id) {
32:     mDrawerList.setItemChecked(position, true);
33:     String mosqueName = mosqueNames[position];
34:     setTitle(mosqueName);
35:     if(mosqueName.equalsIgnoreCase("المسجد الأقصى")){
36:       mosqueImage.setImageResource(R.drawable.aqsa);
37:       mosqueDetails.setText("...");
```

```
38:     }else if(mosqueName.equalsIgnoreCase("قبة الصخرة")){
39:        mosqueImage.setImageResource(R.drawable.dome);
40:        mosqueDetails.setText("...");
41:     }
42:     mDrawerLayout.closeDrawer(mDrawerList);
43:   }
44:});
45:         ActionBarDrawerToggle mDrawerToggle = new
46:  ActionBarDrawerToggle(
47:                 this,
48:                 mDrawerLayout,
49:                 R.drawable.ic_drawer,
50:                 R.string.drawer_open
51:                 ) {
52:     public void onDrawerClosed(View view) {}
53:
54:     public void onDrawerOpened(View drawerView){}
55:    };
56:  mDrawerLayout.setDrawerListener(mDrawerToggle);
57:     }
58:}
```

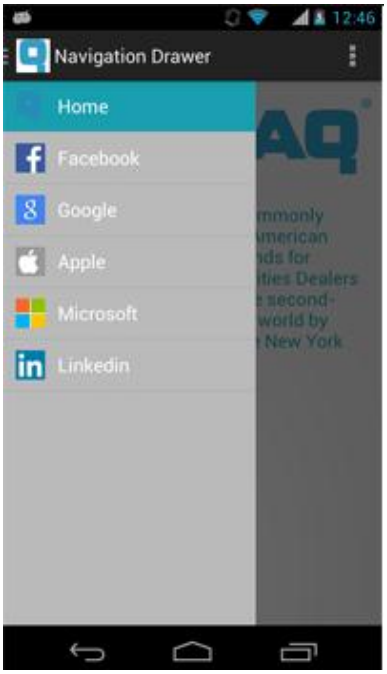
في الدالة onCreate() يتم الوصول لمصفوفة الأسماء mosques_array عن طريق الكود التالي:

mosqueNames = getResources().getStringArray(R.array.mosques_array); (أنظر سطر رقم 14). ومن ثم يتم تعبئة القائمة mDrawerList عن طريق استخدام ArrayAdapter (أنظر سطر رقم 24). بعد ذلك يتم الاستماع لحدث النقر على عناصر القائمة باستخدام الدالة setOnItemClickListener() وكتابة الدالة onItemClicked() (أنظر الكود من سطر 27 إلى 30). الإجراء الذي يتم تنفيذه عند النقر على القائمة هو تغيير عنوان الفعالية بتنفيذ الدالة setTitle() وتغيير محتوى ImageView و TextView لعرض معلومات عن المسجد الذي تم اختياره (أنظر الكود من سطر 32 إلى 41). كذلك يتم إخفاء درج التصفح (Navigation Drawer) عن طريق الدالة DrawerLayout.closeDrawer(); (سطر رقم 42).

يمكن كذلك الاستماع لأحداث فتح وإغلاق درج التصفح (Navigation Drawer) عن طريق الدالة DrawerLayout.setDrawerListener() والتي يمرر لها كائن من نوع ActionBarDrawerToggle. الإجراء المراد تنفيذه عن إغلاق أو فتح درج التصفح يتم كتابته في الدالتين onDrawerClose() و onDrawerOpened() على الترتيب. الكود في الأعلى (من سطر رقم 45 إلى 56) يوضح هذه الخطوة بالرغم من أنه لم يتم استخدامها لتنفيذ أي إجراء في هذا التطبيق.

أسئلة على الوحدة الخامسة

1. ما الفرق بين قائمة السياق (Context Menu) والقائمة المنبثقة (Popup Menu)؟ ومتى تستخدم كل منهما؟
2. قم بتعديل تمرين 4-3 والذي يعرض قائمة (ListView) تضم أسماء وصور مجموعة من الأصدقاء. عند الضغط لفترة على اسم صديق في القائمة يتم عرض قائمة سياق (Context Menu) تضمن زر حذف "Delete". عند النقر على زر الحذف يتم حذف السطر الخاص بالصديق من القائمة (ListView).
3. قم ببناء التطبيق الموضح بالشكل والذي يستخدم عنصر عرض من نوع WebView كمحتوى رئيسي ثابت بالإضافة إلى درج تصفح (Navigation Drawer) يضم قائمة بأسماء بعض المواقع الالكترونية الشهيرة مثل Facebook, Google, Apple وغيرها. درج التصفح يظهر عند السحب من اليسار لليمين، و عند النقر على أي منها، يتم إخفاء درج التصفح ومن ثم فتح الموقع في عنصر العرض WebView.



الوحدة السادسة:

استدامة البيانات في نظام أندرويد

(Data Persistence in Android)

يتعلم الطالب في هذه الوحدة:

- ✓ الطرق المختلفة لحفظ البيانات الخاصة بتطبيقات أندرويد ومتى يجب استخدام كل منها.
- ✓ إنشاء تطبيقات ترتبط بقواعد البيانات.
- ✓ تنفيذ إجراءات مختلفة على قواعد البيانات وعرض النتائج في واجهة المستخدم.

لدراسة هذه الوحدة لابد من الإلمام بمبادئ التعامل مع قواعد البيانات تعليمات SQL المختلفة. كذلك لابد من الإلمام بأساسيات القراءة والكتابة للملفات بلغة جافا.

يوفر نظام أندرويد أكثر من طريقة لحفظ البيانات الخاصة بالتطبيقات. أهم هذه الطرق تشمل ما يلي:

- التفضيلات المشتركة (Shared Preferences): وتشمل حفظ بيانات أساسية بمفاتيح محددة (key-value pairs).
- الذاكرة الداخلية (Internal Storage) وفيها يتم تخزين البيانات في الذاكرة الداخلية للجهاز.
- الذاكرة الخارجية (External Storage) وفيها يتم تخزين البيانات في ذاكرة خارجية مشتركة بين التطبيقات.
- قواعد البيانات (SQLite Databases): وفيها يتم تخزين البيانات في قواعد بيانات خاصة بالتطبيق.

استخدام أي من هذه الطرق يعتمد على احتياجات التطبيق مثل ما إذا كانت البيانات المراد حفظها خاصة بالتطبيق أو يمكن الوصول إليها من تطبيقات أخرى، وكذلك المساحة المطلوبة لتخزين البيانات. سيتم فيما يلي تفصيل كل من هذه الطرق:

التفضيلات المشتركة (Shared Preferences)

التفضيلات المشتركة (SharedPreferences) هي مكون يوفر حفظ واسترجاع بيانات أساسية بسيطة. كل معلومة يتم حفظها يجب أن تكون على شكل مفتاح وقيمة (key-value) بحيث يحدد المفتاح (key) اسم مميز ليتم استعادة القيمة (value) فيما بعد عن طريقه. يمكن استخدام التفضيلات المشتركة لحفظ بيانات من الأنواع البدائية (primitive types) وتشمل: booleans, floats, ints, longs, and Strings. يتم الاحتفاظ بالبيانات المخزنة من تطبيق ما في التفضيلات المشتركة حتى بعد إغلاق هذا التطبيق.

للوصول إلى الكائن الخاص بالتفضيلات المشتركة واستخدامه، يمكن استخدام واحدة من إحدى دالتين:

- getSharedPreferences(): استخدم هذه الدالة إذا أردت انشاء أكثر من ملف لحفظ البيانات. لإنشاء ملف تفضيلات جديد يتم تمرير اسم جديد من خلال الدالة getSharedPreferences().
- getPreferences(): في هذه الحالة يتم استخدام ملف واحد لكل فعالية لحفظ التفضيلات، ولا يتم تمرير أي اسم لهذه الدالة.

ولكتابة بيانات إلى التفضيلات المشتركة يجب القيام بالخطوات التالية:

1. تنفيذ الدالة edit() والتي تقوم بإرجاع كائن من نوع SharedPreferences.Editor والذي يتم من خلاله الكتابة.
2. أضف البيانات عن طريق تنفيذ مهام مثل: putBoolean(), putString(), putInt().
3. إعتماد تنفيذ عملية حفظ البيات المدخلة عن طريق تنفيذ الدالة commit().

المثال التالي يوضح كيفية حفظ متغير باسم mSilentMode من نوع boolean في التفضيلات المشتركة قبل إنهاء الفعالية مباشرة، ثم يتم استرجاع هذه القيمة عند تشغيل الفعالية مرة أخرى:

```
public class Example extends Activity{
    public static final String PREFS_NAME ="MyPrefsFile";
    private Boolean mSilentMode;

    @Override
    protectedvoid onCreate(Bundle state){
       super.onCreate(state);
       ...

       // Restore preferences
       SharedPreferences settings =
getSharedPreferences(PREFS_NAME,0);
       mSilentMode =
settings.getBoolean("silentMode",false);
       setSilent(mSilentMode); // Do something with silent
    }

    @Override
    protectedvoid onStop(){
       super.onStop();

      // We need an Editor object to make preference
changes.
      SharedPreferences settings =
```

```
getSharedPreferences(PREFS_NAME,0);
      SharedPreferences.Editor editor = settings.edit();
      editor.putBoolean("silentMode", mSilentMode);

      // Commit the edits!
      editor.commit();
    }
}
```

. أنظر الدالة ()onStop ولاحظ كيف تم تطبيق الخطوات في الأعلى لحفظ قيمة المتغير mSilentMode حيث تم استخدام المفتاح "silentMode" كمعرف. لاحظ أن ملف التفضيلات محدد باسم MyPrefsFile والذي تم تمريره للدالة ()getSharedPreferences. عند تشغيل الفعالية مرة أخرى، يتم استرجاع البيانات من التفضيلات المشتركة أثناء تنفيذ الدالة ()onCreate. لاحظ أنه تم الرجوع إلى نفس الملف والذي اسمه MyPrefsFile.

ملاحظة: عند تشغيل الفعالية لأول مرة لن يكون هناك بيانات مخزنة خاصة بالفعالية. في هذه الحالة سيأخذ المتغيرالقيمة الافتراضية وهي false والتي تم تحديدها في المدخل الثاني للدالة getBoolean("silentMode", false).

الذاكرة الداخلية (Internal Storage)

يمكن تخزين البيانات مباشرةً في الذاكرة الداخلية للجهاز، وتكون البيانات المخزنة خاصة بالتطبيق الذي تم التخزين من خلاله ولا يمكن للتطبيقات الأخرى الوصول لهذه البيانات. عند حذف التطبيق تحذف البيانات المتعلقة به تلقائياً.

لإنشاء وتخزين ملف في الذاكرة الداخلية يمكن استعمال الخطوات التالية:

1. نفذ الدالة ()openFileOutput ومرر لها اسم الملف المراد انشاؤه وتخزينه ووضع الوصول Operating mode. تقوم هذه الدالة بإنشاء الملف وإرجاع كائن من نوع FileOutputStream والذي من خلاله يتم الكتابة إلى الملف.
2. نظراً لأن FileOutputStream يستخدم لكتابة البيانات بالصيغة الأولية Bytes، و حتى تتمكن من كتابة أي نص، أنشي كائن من نوع PrintWriter ومرر له الكائن من نوع FileOutputStream.
3. اكتب إلى الملف باستخدام الدالة ()print.
4. أغلق الكائنات من نوعFileOutputStream و PrintWriter ذلك حتى لا يبقى الملف مفتوحاً بعد الانتهاء منه.

هذه الخطوات موضحة في الكود التالي:

```
String fileName ="hello_file";
String string="hello world!";
FileOutputStream fos =
openFileOutput(fileName,Context.MODE_PRIVATE);
PrintWriter pr = new PrintWriter(fos);
pr.print(string);
pr.close();
fos.close();
```

لاحظ أن وضع الوصول MODE_PRIVATE يعني أن الوصول للملف خاص بالتطبيق الحالي فقط.

لقراءة ملف سبق تخزينه في الذاكرة الداخلية يمكن اتباع الخطوات التالية:

1. نفذ الدالة openFileInput() ومرر لها اسم الملف المراد فتحه. هذه الدالة تقوم بإرجاع كائن من نوع FileInputStream.
2. حتى تتمكن من قراءة محتويات الملف سطراً سطراً، أنشئ كائن من نوع BufferedReader ومرر له الكائن FileInputStream
3. اقرأ محتويات الملف سطراً سطراً باستخدام الدالة readLine حتى تصل إلى نهاية الملف (عندها ترجع الدالة readLine قيمة null).
4. أغلق الكائنات من نوع BufferedReader و FileInputStream.

هذه الخطوات موضحة بالكود التالي:

```
FileInputStream in = openFileInput("hello_file");
BufferedReader br = new BufferedReader(new
InputStreamReader(in));
String strLine;
while((strLine = br.readLine())!= null){
System.out.println(strLine);
}
br.close();
in.close();
```

الذاكرة الخارجية (External Storage)

يدعم نظام أندرويد حفظ الملفات في الذاكرة الخارجية والتي قد تكون ذاكرة يمكن إزالتها وتركيبها removable مثل SD Card، أو ذاكرة ثابتة لا يمكن إزالتها. الملفات المخزنة في الذاكرة الخارجية يمكن الوصول إليها من أي تطبيق آخر وذلك بعكس الملفات المخزنة بالذاكرة الداخلية والتي تكون خاصة بالتطبيق.

قبل القراءة أو الكتابة للذاكرة الخارجية يجب إضافة إذن (permission) في ملف الوثيقة (Manifest) الخاص بالتطبيق، حيث أن هناك إذن بالقراءة (READ_EXTERNAL_STORAGE) و إذن بالكتابة (WRITE_EXTERNAL_STORAGE) كما هو موضح بالأسفل:

```
<manifest ...>
    <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
android:name="android.permission.READ_EXTERNAL_STORAGE"/>
  ...
</manifest>
```

ملاحظة: ابتداء من Android 4.4 لا تحتاج لإذن permission إذا أردت كتابة أو قراءة ملفات خاصة بالتطبيق فقط.

كما يجب قبل محاولة الوصول للذاكرة الخارجية التأكد من أن الذاكرة متوفرة عن طريق تنفيذ الدالة getExternalStorageState(). الذاكرة الخارجية قد تكون في حالة "مركبة" Mounted، "مفقودة" missing, "القراءة فقط" read only بالإضافة لحالات أخرى.المثال التالي يوضح دالتين لفحص حالة الذاكرة الخارجية: الأولى تفحص إذا كانت الذاكرة متوفرة للكتابة والقراءة بينما تفحص الثانية إذا كانت الذاكرة متوفرة للقراءة فقط.

```
/* Checks if external storage is available for read and
write */
public boolean isExternalStorageWritable(){
    String state =Environment.getExternalStorageState();
    if(Environment.MEDIA_MOUNTED.equals(state)){
        return true;
    }
    return false;
}

/* Checks if external storage is available to at least read
*/
public boolean isExternalStorageReadable(){
    String state =Environment.getExternalStorageState();
    if(Environment.MEDIA_MOUNTED.equals(state)||
        Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)){
        return true;
    }
    return false;
}
```

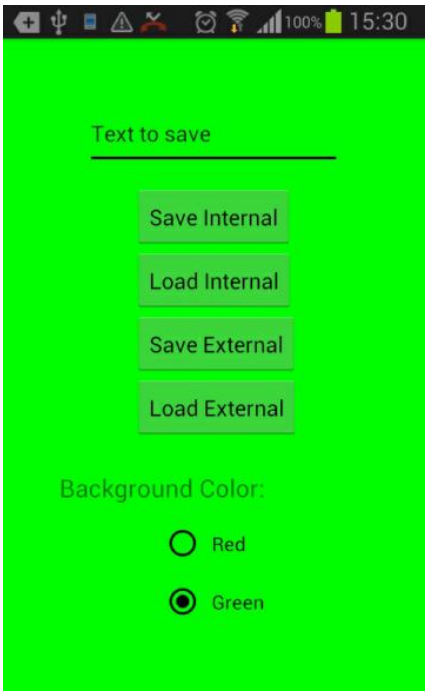
المثال التالي يوضح كيفية تخزين ملف في الذاكرة الخارجية. الدالة Environment.getExternalStorageDirectory() تقوم بإرجاع مسار المجلد الذي يمثل الذاكرة الخارجية حيث يمكن حفظ الملفات.

```
File directory = Environment.getExternalStorageDirectory();
File file = new File(directory, "externalFile.txt");
if(!file.exists())
     file.createNewFile();
FileOutputStream fos = new FileOutputStream(file);
OutputStreamWriter osr = new OutputStreamWriter(fos);
PrintWriter pr = new PrintWriter(osr);
pr.write();
osr.close();
fos.close();
```

تمرين عملي (1-6)

يهدف هذا التمرين إلى التدرب على حفظ بيانات تطبيق ما باستخدام الطرق الموضحة مسبقاً وهي التفضيلات المشتركة (SharedPreferences)، الذاكرة الداخلية (Intenral Storage) والذاكرة الخارجية (External Storage). واجهة التطبيق موضحة في شكل 1-6، وتتكون من مربع إدخال (EditText) يتم من خلاله إدخال نص ليتم حفظه و استرجاعه بالطرق المختلفة (حفظ في الذاكرة الداخلية، استرجاع من الذاكرة الداخلية، حفظ في الذاكرة الخارجية، استرجاع من الذاكرة الخارجية). لنتأكد من حفظ البيانات واسترجاعها بشكل سليم، نقوم بحفظ البيانات وإغلاق التطبيق ثم تشغيله مرة أخرى واسترجاع البيانات بنفس الطريقة التي حفظت بها، ونكرر هذه التجربة لكل طريقة

تحتوي واجهة التطبيق أيضاً على زري انتقاء (RadioButtons) لتحديد لون الخلفية (background)، حيث أن اختيار أي لون يؤدي إلى تغيير لون الخلفية. سنقوم باستخدام المفضلات المشتركة (SharedPreferences) لحفظ اللون واسترجاعه عند إعادة تشغيل البرنامج وذلك باستخدام الدالتين onPause() و onResume() على التوالي.



شكل 1-6: واجهة التطبيق الخاص بتمرين 1-6

ملف تصميم الواجهة موضح في ما يلي:

```
<RelativeLayout
xmlns:tools="http://schemas.android.com/tools"

xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"

android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <EditText
        android:id="@+id/editText"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="40dp"
        android:ems="10" >
        <requestFocus />
    </EditText>
    <Button
        android:id="@+id/loadInBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/saveInBtn"
        android:layout_below="@+id/saveInBtn"
        android:onClick="loadInternal"
        android:text="Load Internal" />
    <Button
        android:id="@+id/saveExBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/loadInBtn"
        android:layout_below="@+id/loadInBtn"
        android:onClick="saveExternal"
        android:text="Save External" />
    <Button
        android:id="@+id/loadExBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/saveExBtn"
        android:layout_below="@+id/saveExBtn"
        android:onClick="loadExternal"
        android:text="Load External" />
    <Button
        android:id="@+id/saveInBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/editText"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="15dp"
        android:onClick="saveInternal"
        android:text="Save Internal" />
```

```
    <TextView
        android:id="@+id/tvBgColor"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignRight="@+id/radioGroup"
        android:layout_below="@+id/loadExBtn"
        android:layout_marginTop="24dp"
        android:text="Background Color:"
        android:textSize="20sp" />
    <RadioGroup
        android:id="@+id/radioGroup"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/tvBgColor"
        android:layout_centerHorizontal="true" >
        <RadioButton
            android:id="@+id/rdRed"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="10dp"
            android:text="Red" />
        <RadioButton
            android:id="@+id/rdGreen"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignLeft="@+id/rdRed"
            android:layout_below="@+id/rdRed"
            android:layout_marginTop="5dp"
            android:text="Green" />
    </RadioGroup>
</RelativeLayout>
```

تذكر أننا سنحتاج إلى تغيير لون الخلفية أثناء عمل التطبيق، مما يستلزم الوصول للهيكلية (Layout) برمجياً لتغيير لونها بناءً على اختيار المستخدم من قائمة الألوان. للوصول للهيكلية (Layout)، يجب أن يكون لها رقم معرف (ID) في ملف R حتى نتمكن من تنفيذ الدالة findViewById(). لذلك تم إضافة الخاصية android:id="@+id/container" لملف التصميم حتى يتم إنشاء معرف Id لها.

تذكر أيضاً أنه للوصول للذاكرة الخارجية وحفظ البيانات يجب إضافة الإذن WRITE_EXTERNAL_STORAGE للقيام بذلك.

الكود التالي يوضح الفعالية (MainActivity) الخاصة بالتطبيق.

```
1: public class MainActivity extends Activity {
2:
3:   public static final String PREFS = "myprefs";
4:   public String bgColor;
5:
6:   EditText editText;
7:   @Override
8:   protected void onCreate(Bundle savedInstanceState) {
9:        super.onCreate(savedInstanceState);
10:       setContentView(R.layout.activity_main);
11:       editText = (EditText)
12:  this.findViewById(R.id.editText);
13:       RadioGroup rdGroup = (RadioGroup)
14:  this.findViewById(R.id.radioGroup);
15:       rdGroup.setOnCheckedChangeListener(new
16:  OnCheckedChangeListener(){
17:
18:            @Override
19:            public void onCheckedChanged(RadioGroup
20:  group, int checkedId) {
21:                 RadioButton selected = (RadioButton)
22:  findViewById(checkedId);
23:                 bgColor =
24:  selected.getText().toString();
25:                 setBackgroundColor(bgColor);
26:            }
27:       });
28:  }
29:
30:  public void saveInternal(View view){
31:       try {
32:            FileOutputStream fos =
33:  this.openFileOutput("myfile.txt",
34:  Context.MODE_PRIVATE);
35:            PrintWriter pr = new PrintWriter(fos);
36:            pr.write(editText.getText().toString());
37:            pr.close();
38:            fos.close();
```

```
39:             editText.setText("");
40:         } catch (FileNotFoundException e) {
41:             e.printStackTrace();
42:         } catch (IOException e) {
43:             e.printStackTrace();
44:         }
45:   }
46:
47:   public void loadInternal(View view){
48:         try {
49:             FileInputStream fis =
50:   this.openFileInput("myfile.txt");
51:             BufferedReader br = new BufferedReader(new
52:   InputStreamReader(fis));
53:             String txt="", tmp=null;
54:             while((tmp=br.readLine()) != null){
55:                 txt+=tmp;
56:             }
57:             br.close();
58:             fis.close();
59:             editText.setText(txt);
60:         } catch (FileNotFoundException e) {
61:             e.printStackTrace();
62:         } catch (IOException e) {
63:             e.printStackTrace();
64:         }
65:   }
66:
67:   public void saveExternal(View view){
68:         try {
69:             if(isExternalStorageWritable()){
70:                 File directory =
71:   Environment.getExternalStorageDirectory();
72:                 File file = new File(directory,
73:   "externalFile.txt");
74:                 if(!file.exists())
75:                     file.createNewFile();
76:                 FileOutputStream fos = new
77:   FileOutputStream(file);
78:                 OutputStreamWriter osr = new
```

```
79:  OutputStreamWriter(fos);
80:                 PrintWriter pr = new PrintWriter(osr);
81:
82:  pr.write(editText.getText().toString());
83:                 pr.close();
84:                 osr.close();
85:                 fos.close();
86:                 editText.setText("");
87:            }
88:       } catch (IOException e) {
89:            e.printStackTrace();
90:       }
91:  }
92:
93:  public void loadExternal(View view){
94:       try {
95:            if(isExternalStorageReadable()){
96:                 File directory =
97: Environment.getExternalStorageDirectory();
98:                 File file = new File(directory,
99:  "externalFile.txt");
100:                if(file.exists()){
101:                     FileInputStream fis = new
102:  FileInputStream(file);
103:                     InputStreamReader isr = new
104:  InputStreamReader(fis);
105:                     BufferedReader br = new
106:  BufferedReader(isr);
107:                     String txt="", tmp=null;
108:                     while((tmp=br.readLine()) !=
109:  null){
110:                          txt+=tmp;
111:                     }
112:                     br.close();
113:                     editText.setText(txt);
114:                }
115:           }
116:      } catch (IOException e) {
117:           e.printStackTrace();
118:      }
```

```
119: }
120:
121: @Override
122: protected void onResume() {
123:      super.onResume();
124:      SharedPreferences sp =
125:  this.getSharedPreferences(PREFS,
126:  Context.MODE_PRIVATE);
127:      bgColor = sp.getString("bgColor", null);
128:      if(bgColor != null)
129:           this.setBackgroundColor(bgColor);
130: }
131:
132: @Override
133: protected void onPause() {
134:      super.onPause();
135:      if(bgColor != null){
136:           SharedPreferences sp =
137:  this.getSharedPreferences(PREFS,
138:  Context.MODE_PRIVATE);
139:           Editor editor = sp.edit();
140:           editor.putString("bgColor", bgColor);
141:           editor.commit();
142:      }
143: }
144:
145: protected void setBackgroundColor(String bgColor) {
146:      RelativeLayout bgElement = (RelativeLayout)
147:  findViewById(R.id.container);
148:      if(bgColor.equals("Red"))
149:           bgElement.setBackgroundColor(Color.RED);
150:      else if(bgColor.equals("Green"))
151:           bgElement.setBackgroundColor(Color.GREEN);
152: }
153:...
154: }
```

يحتوي الكود على أربعة دوال رئيسية كل منها مسؤول عن تنفيذ إجراء محدد، وهي كما يلي:

- الدالة saveInternal() (أنظر الكود من سطر رقم 30 إلى 45): وفيها يتم قراءة النص من العنصر EditText وحفظه في الذاكرة الداخلية في ملف اسمه myfile.txt.
- الدالة loadInternal() (أنظر الكود من سطر رقم 47 إلى 65): وفيها يتم قراءة الملف myfile.txt والذي تم حفظه مسبقاً في الذاكرة الداخلية، ومن ثم عرض محتويات الملف في العنصر EditText.
- الدالة saveExternal() (أنظر الكود من سطر رقم 67 إلى 91): ويتم تنفيذها عند النقر على زر " Save External" ليتم حفظ النص المكتوب في العنصر EditText إلى الملف myfile.txt والذي يتم تخزينه في الذاكرة الخارجية. لاحظ أنه يتم فحص حالة الذاكرة الخارجية للتأكد من إمكانية الكتابة إليها قبل عملية الحفظ عن طريق تنفيذ الدالة isExternalStorageWritable().
- الدالة loadExternal():(أنظر الكود من سطر رقم 93 إلى 119) ويتم تنفيذها عند النقر على زر " Load External" ليتم استرجاع محتوى الملف myfile.txt وعرضه في العنصر EditText. لاحظ أنه يتم فحص حالة الذاكرة الخارجية للتأكد من إمكانية القراءة منها قبل عملية الحفظ عن طريق تنفيذ الدالة isExternalStorageReadable().

في الدالة onCreate() تم إنشاء مستمع من نوع OnCheckedChangeListener للاستماع لحدث النقر على أحد أزرار الانتقاء (RadioButton)، حيث يتم بناءً على ذلك تغيير لون الخلفية عن طريق تنفيذ الدالة setBackgroundColor() (أنظر الكود من سطر رقم 145 إلى 152)، وكذلك حفظ لون الخلفية الحالي في المتغير bgColor (أنظر الكود من سطر رقم 8 إلى 28).

تم أيضاً تطبيق الدالتين onPause() (من سطر رقم 133 إلى 143) و onResume() (من سطر رقم 122 إلى 130) وذلك حتى يتم حفظ واسترجاع لون الخلفية الحالي تلقائياً عند إغلاق التطبيق وإعادة تشغيله، حيث يتم استخداع التفضيلات المشتركة ShardPreferences لهذا الغرض.

قواعد البيانات (SQLite Databases)

SQLite هي مكتبة برمجية مفتوحة المصدر (Open Source) تستخدم محرك قواعد بيانات (Database Engine) يدعم قواعد البيانات العلاقية (Relational Databases) وما يتعلق بها من خصائص مثل SQL، والتعاملات (Transactions) و التصريحات المعدة مسبقاً (Prepared statements).

يدعم SQLite ثلاثة أنواع من البيانات هي النص TEXT وهو يقابل String في لغة الجافا، INTEGER وهو يشابة long أو int في الجافا، و REAL وهو يقابل double في الجافا. أي أنواع أخرى من البيانات يجب تحويلها إلى أحد هذه الأنواع الثلاثة حتى يمكن تخزينها في قاعدة البيانات. كما أن SQLite لا يتحقق من نوعية البيانات قبل تخزينها، بمعنى أنه يمكن تخزين بيانات رقمية في عمود يقبل بيانات نصية والعكس صحيح.

مكتبة SQLite مضمنة في كل جهاز يعمل بنظام أندرويد، وبذلك لا يحتاج إلى إي إعدادات أو تحميل ملفات لتشغيله. قاعدة البيانات الخاصة بأي تطبيق يتم تخزينها في المسار:

DATA/data/APP_NAME/databases/DATABASE_NAME

حيث DATA في المسار السابق ترمز إلى المسار الذي ترجعه الدالة (Environment.getDataDirectory().
APP_NAME في المسار السابق يرمز إلى اسم التطبيق، DATABASE_NAME يرمز إلى اسم قاعدة البيانات.

الرزمة android.database package تحتوي على كل الفئات (classes) اللازمة للتعامل مع قواعد البيانات، بينما الرزمة android.database.sqlite package تحتوي على الفئات (classes) الخاصة بالتعامل مع SQLite.

إنشاء قواعد البيانات

لإنشاء قاعدة بيانات جديدة لتطبيق أندرويد ما يجب إنشاء فئة جديد (class) متفرعة من الفئة SQLiteOpenHelper كما هوموضح في المثال التالي حيث تم إنشاء الفئة MySQLiteHelper المتفرعة من SQLiteOpenHelper:

```
public class MySQLiteHelper extends SQLiteOpenHelper{

     public MySQLiteHelper(Context context, String name,
     int version) {
          super(context, name, null, version);
     }

     @Override
     public void onCreate(SQLiteDatabase db) {
          db.execSQL("create table student(id integer
primary key autoincrement not null, name text not null, age
integer);");
     }

     @Override
     public void onUpgrade(SQLiteDatabase db, int
oldVersion, int newVersion) {
          db.execSQL("drop table student");
          this.onCreate(db);
     }
}
```

لاحظ أنه من الباني (Constructor) يتم استدعاء الباني (Constructor) الخاص بالفئة الأم SQLiteOpenHelper حيث يمرر لها ثلاث متغيرات أساسية وهي:

- المتغير context يشير إلى مكون التطبيق إلذي يستخدم قاعدة البيانات مثل مؤشر لكائن الفعالية المضيفة (Activity) أو الكائن الذي يمكن الحصول عليه من الدالة ()getApplicationContext.
- المتغير name يشير إلى اسم قاعدة البيانات المراد إنشاؤها أو تعديلها.
- المتغير version يشير إلى رقم إصدار قاعدة البيانات،وهو رقم يبدأ من 1،ويجب زيادته في كل مرة يحدث فيها تغيير في هيكلية قاعدة البيانات.

في الفئة الجديدة (MySQLiteHelper Class) يجب إعادة كتابة الدوال التالية:

- ()onCreate: ويتم استدعاؤها تلقائياً من النظام (لا تنفذ من التطبيق) عند إنشاء قاعدة البيانات لأول مرة. في الكود الموضح في الأعلى يتم إنشاء جدول باسم student مكون ثلاثة أعمدة وهي id, name, age.
- ()onUpgrade: ويتم استدعاؤها تلقائياً اذا تغير رقم الإصدار version وذلك في حالة تغيير هيكلية قاعدة البيانات مثل حدوث تعديل على جدول معين.

على سبيل المثال، عند إنشاء قاعدة البيانات لأول مرة يتم تنفيذ الدالة ()onCreate والتي يتم فيها إنشاء الجداول والربط بينها. عند حدوث أي تغيير في قاعدة البيانات مثل تعديل جدول ما ،يجب زيادة رقم الإصدار عند إنشاء كائن من نوع MySQLiteHelper. يقوم SQLite تلقائيا بمقارنة هذا الرقم مع رقم آخر اصدار مسبق لقاعدة البيانات، وإذا كان مختلفاً يقوم بتنفيذ الدالة ()onUpgrade حيث يتم فيها إجراء التعديل اللازمة على هيكلية قاعدة البيانات. في المثال السابق تقوم الدالة ()onUpgrade بحذف الجدول وإنشائه مرةً أخرى.

الوصول والتعامل مع قواعد البيانات

الفئة SQLiteOpenHelper توفر دالتين تمكن الوصول لقاعدة البيانات هما: ()getReadableDatabase و ()getWriteableDatabase حيث تسمح الأولى بالقراءة فقط بينما توفر الثانية إمكانية القراءة والكتابة من قاعدة البيانات. كلتا الدالتين ترجعان مؤشر إلى الكائن SQLiteDatabase وهي المكون الأساسي الذي يتم من خلاله يتم تنفيذ أي تعليمات SQL.

يوفر SQLiteDatabase الدوال ()insert(), update(), delete وذلك للإضافة والحذف والتعديل على محتوى قواعد البيانات. على سبيل المثال، الكود التالي يوضح كيفية اضافة بيانات إلى الجدول student الذي تم إنشاؤه مسبقاً. يتم تجهيز البيانات المراد إضافتها ككائن من نوع ContentValues وفيه يتم إدخال البيانات على شكل مفتاح-قيمة (key-value)، حيث يحدد المفتاح (key) اسم العمود في الجدول بينما القيمة (value) تمثل محتوى السجل في العمود. الكائن ContentValues يمكن استخدامه لإضافة أو تعديل البيانات في الجدول.

```
// Create or access a database with the name "studentdb"
MySQLiteHelper  sqlHelper = new
MySQLiteHelper(getApplicationContext(),"studentdb",1);
SQLiteDatabase  db = sqlHelper.getWritableDatabase();
ContentValues values = new ContentValues();
values.put("name", name);
values.put("age", age);
```

```
db.insert("student", null, values);
db.close();
sqlHelper.close();
```

تنفيذ الإستعلامات (Queries)

تنفيذ الاستعلامات يتم باستخدام إحدى دالتين من الكائن SQLiteDatabase. هاتين الدالتين هما:

- الدالة rawQuery() وهذه الدالة يمرر إليها الإستعلام والقيم المستعلم عنها كما في المثال التالي:

```
Cursor cursor = database.query("SELECT * from student where
name = ? and age > ?",new String[]{"Ahmed","20"});
```

حيث database هو كائن من نوع SQLiteDatabase.

- الدالة query() وهي توفر واجهة تمكن من إدخال الاستعلام وتفاصيله بشكل مرتب، وهي تأخذ الصيغة التالية:

```
public Cursor query (String table, String[] columns, String
 selection, String[] selectionArgs, String groupBy, String
having, StringorderBy, String limit);
```

حيث تفاصيل هذه المتغيرات موضحة بجدول 1-6 التالي:

جدول 1-6: تفاصيل المتغيرات المدخلة للدالة SQLiteDatabase.query()

اسم المتغير	وصف المتغير
Table	اسم الجدول الذي ينفذ عليه الاستعلام
Columns	اسماء الأعمدة المطلوب إرجاع قيمها. في حال الاستعلام عن كل الأعمدة تعطى القيمة null
Selection	هذا الجزء يمثل جملة WHERE والتي تحدد طريقة تصفية البيانات المستعلم عنها.
selectionArgs	هذا الجزء يعطي قيم لأي علامات استفهام ? قد تكون ضمن المتغير السابق Selection
groupBy	تكافئ group by في استعلام SQL
Having	تكافئ having في استعلام SQL
orderBy	تكافئ order by في استعلام SQL
Limit	قيمته تحدد عدد النتائج التي يتم إرجاعها.

مثال لاستخدام الدالة query() للاستعلام عن بيانات من الجدول student موضح بالأسفل:

```
String[] allColumns = {"id","name","age"};
```

```
Cursor cursor = database.query("student", allColumns, "
WHERE name =? and age >?", new String[]{"Ahmed","20"},
null, null, "name", null);
```

حيث database هو كائن من نوع SQLiteDatabase، وهذا الاستعلام يكافئ استعلاء SQLالتالي:

```
SELECT id,name, age FROM student WHERE name="Ahmed" and
age>20 order by name
```

كلا الدالتين السابقتين ()rawQuery و ()query ترجعان كائن من نوع Cursor وهو مؤشر يؤشر على جدول النتائج الناتج عن تنفيذ الاستعلام. يمكن قراءة البيانات عن طريق تحريك المؤشر من سطر لآخر وقراءة النتائج من كل سطر كما هو موضح في المثال التالي:

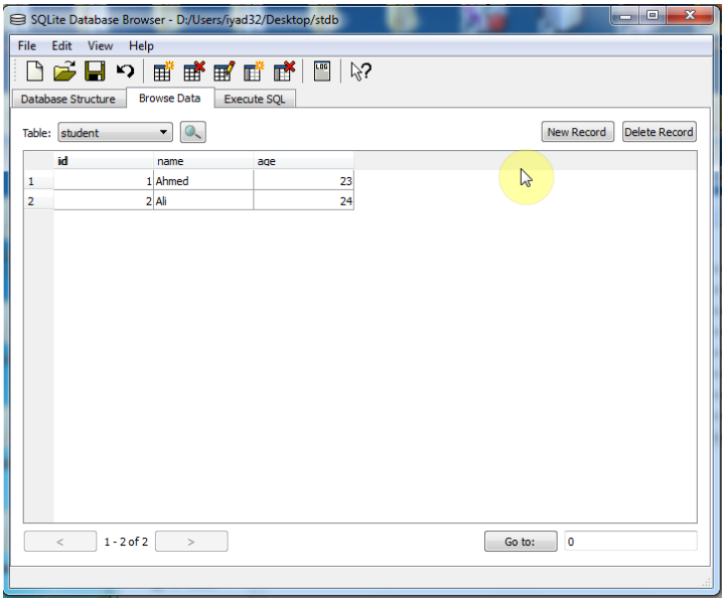
```
cursor.moveToFirst();
while(!cursor.isAfterLast()){
     int id = cursor.getInt(0);
String name = cursor.getString(1);
     int age = cursor.getInt(2);
…
     cursor.moveToNext();
}
cursor.close();
```

الدالة ()moveToFirst تنقل المؤشر لأول سطر في النتائج. الدالة ()moveToNext تحرك المؤشر خطوةً للأمام بعد قراءة السطر. الدالة ()isAfterLast تفحص انتهاء عملية القراءة وهو ما يتحقق عند وصول المؤشر cursor لما بعد السطر الأخير.

يتم قراءة قيم الأعمدة لكل سطر باستخدام الدوال ()getInt أو ()getString وغيرها من دوال قراءة القيم، وهذه الدوال تأخذ رقم العمود المراد قراءته كمدخل. من المهم بعد الانتهاء من قراءة النتائج إغلاق المؤشر cursor عن طريق الدالة ()close.

الإنشاء المسبق لقواعد البيانات

في كثير من الأحيان، يفضل إنشاء قاعدة البيانات في مرحلة تصميم التطبيق بدلاً من إنشاءها أثناء وقت التنفيذ كما هو الحال عند استخدام الكائن من نوع SQLiteOpenHelper. لإنشاء قواعد بيانات من نوع SQLite، يمكن استخدام بعض الأدوات المجانية التي تدعم ذلك مثل SQLite Database Browser (أنظر شكل 2-6) والمتوفر مجاناً من خلال الرابط /sourceforge.net/projects/sqlitebrowser.



شكل 2-6: واجهة الأداة البرمجية (SQLite Database Browser) المستخدمة في إنشاء قواعد البيانات

قاعدة البيانات المنشأة يتم حفظها كملف. لربط ملف قاعدة البيانات بتطبيق اللأندرويد، يجب وضع الملف في مجلد assets، ثم يجب كتابة الكود التالي في الدالة ()onCreate عن بداية تشغيل الفعالية حتى يتم نسخ الملف إلى الجهاز. المثال التالي يوضح الكود اللازم لنسخ ملف قاعدة البيانات stddb إلى الجهاز ومن ثم إنشاء كائن من نوع SQLiteDatabase.

```
…
protected void onCreate(Bundle savedInstanceState) {
…
String destPath = "/data/data/" + getPackageName() +
"/databases";
try {
     File f = new File(destPath);
     if (!f.exists()) {
          f.mkdirs();
          f.createNewFile();
           //---نسخ ملف قاعدة البيانات من مجلد assets إلى الجهاز---

     CopyDB(getBaseContext().getAssets().open("stddb"),
               new FileOutputStream(destPath + "/stddb"));
               }
          } catch (FileNotFoundException e) {
```

```
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }


SQLiteDatabase database =
SQLiteDatabase.openDatabase(destPath + "/stddb", null,
SQLiteDatabase.OPEN_READWRITE);
…
}

public void CopyDB(InputStream inputStream, OutputStream
outputStream) throws IOException {
     //---copy 1K bytes at a time---
     byte[] buffer = new byte[1024];
     int length;
     while ((length = inputStream.read(buffer)) > 0) {
          outputStream.write(buffer, 0, length);
     }
     inputStream.close();
     outputStream.close();
}
```

تمرين عملي (2-6)

في هذا التمرين سنقوم ببناء تطبيق أندرويد متكامل للربط مع قاعدة بيانات وتنفيذ إجراءات مختلفة عليها مثل تنفيذ الاستعلامات والإضافة والخذف. قاعدة البيانات باسم studentdb وتحتوي جدول واحد باسم student يتكون من ثلاثة أعمدة id,name, age. الواجهة الرئيسية للتطبيق موضحة في شكل 3-6 وتحتوي على ثلاثة أزرار: "View All" لعرض كل أسماء الطلاب من قاعدة البيانات، "Insert" لإضافة بيانات طالب جديد، و "Delete" لحذف طالب من قاعدة البيانات.

عرض أسماء الطلاب يتم من خلال قائمة (ListView) في فعالية (Activity) منفصلة. إضافة بيانات طالب جديد يتم من خلال نموذج إدخال يظهر عند النقر على زر "Insert" حيث يطلب من المستخدم إدخال اسم الطالب وعمره ومن ثم النقر على زر "Insert" للحفظ. حذف البيانات يتم عن طريقة عرض قائمة الطلاب في (ListView) حيث أن النقر على أي سطر في القائمة يظهر قائمة جانبية فيها أمر الحذف "Delete" ، وعند النقر عليه يتم إزالة بيانات الطالب من القائمة (ListView) ومن قاعدة البيانات.

شكل 3-6: واجهات التطبيق الخاص بتمرين 2-6.

قبل الحديث عن تصميم الواجهات والكود الخاص بالفعاليات، سنعرض كود الأجزاء المتعلقة بقواعد البيانات، كما ذكرنا مسبقاً، الخطوة الأولى للوصول إلى أو إنشاء قاعدة البيانات هو بإنشاء فئة (class) متفرعة من SQLiteOpenHelper. الكود التالي يوضح الفئة (class) باسم MySQLiteHelper والتي تؤدي هذا الغرض:

```
1:  public class) MySQLiteHelper extends SQLiteOpenHelper{
2:
3:    public static final String DATABASE_NAME =
4:  "studentdb";
5:    public static final int DATABASE_VERSION = 1;
6:    public static final String STUDENT_TABLE = "student";
7:    public static final String[] allColumns =
8:  {"_id","name","age"};
9:
10:  public MySQLiteHelper(Context context, String name,
11:  int version) {
12:       super(context, name, null, version);
13:  }
14:
15:  @Override
16:  public void onCreate(SQLiteDatabase db) {
17:       db.execSQL("create table student(_id integer
18:  primary key autoincrement not null, name text not
19:  null, age integer);");
20:  }
21:
22:  @Override
23:  public void onUpgrade(SQLiteDatabase db, int
24:  oldVersion, int newVersion) {
25:       db.execSQL("drop table student");
26:       this.onCreate(db);
27:  }
28:}
```

لاحظ الثوابت المعرفة في بداية الفئة (class) (أنظر الكود من سطر رقم 3 إلى 8) والتي تحدد اسم قاعدة البيانات studentdb ورقم إصدارها 1 واسم الجدول الذي سيتم الوصول إليه في قاعدة البيانات وهو student واسماء الأعمدة في الجدول في مصفوفة باسم allColumns. تم تعريف هذه الثوابت لأن سيتم استخدامها بكثرة في التطبيق في كل مرة يتم الوصول إلى قاعدة البيانات.

لتسهيل التعامل مع البيانات الخاصة لكل طالب والتعامل مع كل بيانات طالب كوحدة واحدة، تم إنشاء الفئة (class) باسم Student كالتالي:

```
1: public class Student {
2:
3:   private int id;
4:   private String name;
5:   private int age;
6:
7:   public Student(){}
8:
9:   public Student(int id, String name, int age) {
10:       this.id = id;
11:       this.name = name;
12:       this.age = age;
13:  }
14:  public int getId() {
15:       return id;
16:  }
17:  public void setId(int id) {
18:       this.id = id;
19:  }
20:  public String getName() {
21:       return name;
22:  }
23:  public void setName(String name) {
24:       this.name = name;
25:  }
26:  public int getAge() {
27:       return age;
28:  }
29:  public void setAge(int age) {
30:       this.age = age;
31:  }
32:}
```

لتسهيل تنفيذ الإجراءات المختلفة على قاعدة البيانات تم إنشاء فئة (class) باسم StudentDBUtility والتي يتم من خلالها استخدام كائن من الفئة MySQLiteHelper المنشأة مسبقاً لتنفيذ الإجراءات المختلفة على قاعدة البيانات. كود الفئة StudentDBUtility موضح بالأسفل:

```
1:  public class StudentDBUtility {
2:
3:   MySQLiteHelper myHelper;
```

```
4:   SQLiteDatabase db;
5:
6:   public StudentDBUtility(Context context, String
7:  databaseName, int version){
8:       myHelper = new MySQLiteHelper(context,
9:  databaseName, version);
10:  }
11:
12:  public void open(){
13:       db = myHelper.getWritableDatabase();
14:  }
15:
16:  public void close(){
17:       if(db != null && db.isOpen())
18:            db.close();
19:       myHelper.close();
20:  }
21:
22:  public List<Student> getAllStudents(){
23:       List<Student> students = new
24:  ArrayList<Student>();
25:       Cursor cursor =
26:  db.query(MySQLiteHelper.STUDENT_TABLE,
27:  MySQLiteHelper.allColumns, null, null, null, null,
28:  "name");
29:       cursor.moveToFirst();
30:       while(!cursor.isAfterLast()){
31:            int id = cursor.getInt(0);
32:            String name = cursor.getString(1);
33:            int age = cursor.getInt(2);
34:            Student s = new Student(id, name, age);
35:            students.add(s);
36:            cursor.moveToNext();
37:       }
38:       cursor.close();
39:       return students;
40:  }
41:
42:  public long insertStudent(String name, int age){
43:       ContentValues values = new ContentValues();
```

```
44:       values.put("name", name);
45:       values.put("age", age);
46:       return db.insert(MySQLiteHelper.STUDENT_TABLE,
47:  null, values);
48:  }
49:
50:  public long deleteStudent(int id){
51:       return db.delete(MySQLiteHelper.STUDENT_TABLE,
52:  "_id = ?", new String[]{String.valueOf(id)});
53:  }
54: }
```

لاحظ أن الفئة StudentDBUtility تحتوي على ثلاثة دوال رئيسية تمثل الإجراءات المطلوب تنفيذها على الجدول student في قاعدة البيانات. هذه الدوال هي:

- الدالة ()getAllStudents (أنظر الكود من سطر رقم 22 إلى 40): وتستخدم للاستعلام عن كل الطلاب الموجودين في الجدول student ومن ثم إرجاع قائمة List تحتوي على بيانات جميع الطلبة ممثلةً بكائنات من نوع Student. لاحظ أن هذه الدالة تخفي تفاصيل الاستعلام والوصول لقاعدة البيانات حيث تقوم بإرجاع قائمة List ببيانات الطلبة مخزنة في كائنات من نوع Student. لاحظ أيضاً صيغة الاستعلام وهي:

```
db.query(MySQLiteHelper.STUDENT_TABLE,
MySQLiteHelper.allColumns, null, null, null, null, "name");
```

حيث تكافئ استعلاء SQL التالي:

```
SELECT _id, name, age from student ORDER BY name
```

ملاحظة: في حالة الاستعلامات عن كل الأعمدة في الجدول يمكن تمرير null بدلاً من المصفوفة allColumns. كما أنه عند الحاجة لبيانات محددة، يفضل تمرير أسماء الأعمدة التي تحتوي البيانات المطلوبة فقط وذلك لأن الاستعلام عن كل الأعمدة يتطلب مزيد من الوقت.

- الدالة ()insertStudent (أنظر الكود من سطر رقم 42 إلى 48): وفيها يتم إدخال بيانات طالب جديد بتمرير قيم الإسم name والعمر age. قيمة id_الخاصة بالطالب يتم إنشاؤها تلقائياً في الجدول عن إضافة سطر جديد.
- الدالة ()deleteStudent (أنظر الكود من سطر رقم 50 إلى 53): وهي لحذف بيانات الطالب حسب قيمة id التي يتم تمريرها للدالة.
- الدوال ()open و ()close لفتح الاتصال بقاعدة البيانات وإغلاقه (أنظر الكود من سطر رقم 12 إلى 20).

لاحظ أنه سيتم استخدام كائن من الفئة StudentDBUtility من داخل الفعاليات المختلفة (Activities) للوصول إلى قاعدة البيانات.

تم أيضاً إنشاء محول خاص (Custom Adapter) ولذلك لعرض بيانات الطلاب في القائمة (ListView) في كلا الفعاليتين: (ViewActivity) و (DeleteActivity) (أنظر شكل 3-6). السبب في انشاء محول خاص (Custom Adapter) هو أننا نريد تمرير مصفوفة الطلاب المكونة من مجموعة من الكائن Student لعرض أسماء الطلبة في القائمة (ListView) بدلاً من استخدام المحول الافتراضي ArrayAdapter هو أن القائمة ستعرض وتتعامل مع كائن من نوع Student وليس String. الهدف هو حفظ واسترجاع الكائن Student الخاص بكل طالب عن النقر على أي سطر في قائمة العرض (ListView). استرجاع الكائن Student سيمكن لاحقا من معرفة المعرف الخاص به id حتى نتمكن من حذفه مثلاً.

المحول الافتراضي ArrayAdapter يدعم عرض جمل نصية فقط في القائمة (ListView) لذلك يمكن استخدامه لعرض أسماء الطلاب ولكن لن نستطيع عند استخدامه من استرجاع معلومات الطالب الأخرى من القائمة (ListView). الكود الخاص بالمحول StudentCustomAdapter موضح بالأسفل (لاحظ أنه متفرع من الفعالية BaseAdapter).

```
1:  public class StudentCustomAdapter extends BaseAdapter{
2:
3:   Context context;
4:   List<Student> students;
5:
6:   public StudentCustomAdapter(Context context,
7:  List<Student> students) {
8:        this.students = students;
9:        this.context = context;
10:  }
11:
12:  public View getView(int position, View view, ViewGroup
13:  parent){
14:       if(view == null){
15:            LayoutInflater inflater = (LayoutInflater)
16:       context.getSystemService(Context.LAYOUT_INFLATER_
17:       SERVICE);
18:            view =
19:  inflater.inflate(R.layout.student_list_layout, parent,
20:  false);
21:            TextView tv = (TextView)
22:  view.findViewById(R.id.viewName);
23:
24:  tv.setText(students.get(position).getName());
```

```
25:  ViewHolder vh = new ViewHolder();
26:            vh.text = tv;
27:            view.setTag(vh);
28:       }else{
29:            ViewHolder vh = (ViewHolder) view.getTag();
30:
31:  vh.text.setText(students.get(position).getName());
32:       }
33:       return view;
34:  }
35:
36:  static class ViewHolder{
37:       public TextView text;
38:  }
39:
40:  @Override
41:  public int getCount() {
42:       return students.size();
43:  }
44:
45:  @Override
46:  public Object getItem(int position) {
47:       return students.get(position);
48:  }
49:
50:  @Override
51:  public long getItemId(int position) {
52:       return position;
53:  }
54:
55:  public void remove(Student student){
56:       students.remove(student);
57:       this.notifyDataSetChanged();
58:  }
59:}
```

لاحظ أن الدالة getItem() (أنظر الكود من سطر رقم 46 إلى 48) تقوم بإرجاع كائن من نوع Student بناءً على المكان position التي يتم اختياره في القائمة (ListView)، وهو ما يمكن من الاستماع لحدث النقر على القائمة (ListView) واسترجاع الكائن الخاص بالطالب Student حسب المكان الذي تم النقر عليه.

لاحظ أيضاً أنه تم إضافة دالة باسم remove() (أنظر الكود من سطر رقم 55 إلى 58) حيث يمرر لها كائن من نوع Student لحذفه، ومن ثم تحديث محتوات القائمة (ListView) بتنفيذ الدالة notifyDataSetChanged().

لاحظ أيضاً أنه في الدالة getView() (أنظر الكود من سطر رقم 12 إلى 34) يتم انشاء الواجهة الخاصة بكل سطر باستخدام الدالة inflate() حيث يتم تمرير المعرف الخاص بملف تصميم الهيكلية وهو R.layout.student_list_layout حيث يستخدم التصميم التالي:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"

android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
     >

    <TextView
        android:id="@+id/viewName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="16dp"
        android:text="TextView" />
</LinearLayout>
```

تصميم الواجهات الخاصة بالتطبيق يتم باستخدام ملفات التصميم Layout التالية:

أولاً: تصميم الواجهة الرئيسية (MainActivity) موضح فيما يلي:

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/RelativeLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:paddingBottom="@dimen/activity_vertical_margin"
```

```
    android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/tvStudentDB"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="28dp"
        android:text="Student Database"
        android:textSize="28sp"
        android:textStyle="bold" />
    <Button
        android:id="@+id/viewAllBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/tvStudentDB"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="5dp"
        android:text="View All" />
    <Button
        android:id="@+id/insertBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/viewAllBtn"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="5dp"
        android:text="Insert" />
    <Button
        android:id="@+id/deleteBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/insertBtn"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="5dp"
        android:text="Delete" />
</RelativeLayout>
```

ثانياً: تصميم واجهة الإدخال (InsertActivity) موضح فيما يلي:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"

android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.sqlite.InsertActivity" >
    <TextView
        android:id="@+id/tvTitle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="24sp"
        android:textStyle="bold"
        android:layout_gravity="center"
        android:text="Insert Student" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
         >
        <TextView
            android:id="@+id/tvStdName"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="18sp"
            android:text="Name" />
        <EditText
            android:id="@+id/insertName"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
```

```
            android:ems="10" >
            <requestFocus />
        </EditText>
    </LinearLayout>
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="2dp" >

        <TextView
            android:id="@+id/tvStdAge"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="18sp"
            android:text="Age"/>

        <EditText
            android:id="@+id/insertAge"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:ems="10" />
    </LinearLayout>
    <Button
        android:id="@+id/confirmInsertBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="10dp"
        android:text="OK" />
</LinearLayout>
```

واجهتا العرض (ViewActivity) و الحذف (DeleteActivity) تستخدمان التصميم الافتراضي للفعالية (ListActivity)، أي أنه ليس لهما ملفي تصميم للهيكلية.

قبل كتابة الكود الخاص في الفعاليات (Activities) يجب التأكد من إضافة إذن الكتابة للذاكرة الخارجية لملف الوثيقة Manifest. كذلك يجب التأكد من الإعلان عن كل الفعاليات التي يتم إنشاؤها في ملف الوثيقة Manifest.

سنقوم الآن بتفصيل الكود الخاص بكل فعالية (Activity):

أولاً: الفعالية MainActivity والكود الخاص بها موضح بالأسفل. تعرض هذه الفعالية أزرار تنفيذ الإجراءات المختلفة. عند النقر على كل زر يتم تشغيل فعالية جديدة لتنفيذ الإجراء المطلوب. فمثلاً، عند النقر على الزر "viewAllBtn" (أنظر الكود من سطر رقم 9 إلى 18) يتم إنشاء هدف صريح (Explicit Intent) يحدد الفعالية (ViewActivity) والتي سيتم فيها عرض الأسماء. بعد ذلك يتم تشغيل الفعالية (ViewActivity) بتنفيذ الدالة startActivity() حيث يمرر الهدف (Intent) كمدخل للدالة (سطر رقم 16).

```
1:  public class MainActivity extends Activity {
2:
3:   @Override
4:   protected void onCreate(Bundle savedInstanceState) {
5:        super.onCreate(savedInstanceState);
6:        setContentView(R.layout.activity_main);
7:        Button viewAllBtn = (Button)
8:  this.findViewById(R.id.viewAllBtn);
9:        viewAllBtn.setOnClickListener(new
10:  OnClickListener(){
11:
12:            @Override
13:            public void onClick(View v) {
14:                 Intent intent = new
15:  Intent(MainActivity.this, ViewActivity.class);
16:                 startActivity(intent);
17:            }
18:       });
19:       Button insertBtn = (Button)
20:  this.findViewById(R.id.insertBtn);
21:       insertBtn.setOnClickListener(new
22:  OnClickListener(){
23:
24:            @Override
25:            public void onClick(View v) {
26:                 Intent intent = new
27:  Intent(getApplicationContext(),
28:  InsertActivity.class));
29:                 startActivity(intent);
30:            }
31:       });
32:       Button deleteBtn = (Button)
33:  this.findViewById(R.id.deleteBtn);
34:       deleteBtn.setOnClickListener(new
```

```
35:  OnClickListener(){
36:
37:            @Override
38:            public void onClick(View v) {
39:                 Intent intent = new
40:  Intent(getApplicationContext(), DeleteActivity.class);
41:                 startActivity(intent);
42:            }
43:       });
44:  }
45:}
```

ثانياً: الفعالية ViewActivity وهي المسؤولة عن عرض أسماء الطلاب والكود الخاص بها موضح بالأسفل. لاحظ أن هذه الفعالية تقوم بإنشاء كائن من نوع StudentDBUtility للربط مع قاعدة البيانات (أنظر سطر رقم 6)، ومن ثم الاستعلام عن كل الطلاب من خلال الدالة ()getAllStudents (أنظر الأسطر 10 و11). قائمة بيانات الطلبة المرجعة List<Student> students يتم تمريرها إلى محول من نوع StudentCustomAdapter لتعبئة القائمة ListView (أنظر سطر رقم 12).

```
1: public class ViewActivity extends ListActivity {
2:
3:   @Override
4:   protected void onCreate(Bundle savedInstanceState) {
5:        super.onCreate(savedInstanceState);
6:        StudentDBUtility dbUtility = new
7:  StudentDBUtility(this, MySQLiteHelper.DATABASE_NAME,
8:  MySQLiteHelper.DATABASE_VERSION);
9:        dbUtility.open();
10:       List<Student> students =
11:  dbUtility.getAllStudents();
12:       StudentCustomAdapter adapter = new
13:  StudentCustomAdapter(this, students);
14:       this.setListAdapter(adapter);
15:       dbUtility.close();
16:  }
17:}
```

ثالثاً: الفعالية InsetActivity والموضحة بالأسفل ومن خلالها تتم عملية إدخال بيانات الطالب. لاحظ أنه في هذه الدالة يتم أيضاً استخدام كائن من نوع StudentDBUtility (أنظر سطر رقم 11) لإدخال بيانات الطالب من خلال الدالة ()insertStudent. عند النقر على زر الإضافة "Insert" يتم تنفيذ إجراء الموضح في الأسطر من

26 إلى 43 حيث يتم قراءة العناصر المدخلة (الاسم والعمر) ثم تنفيذ الدالة insertStudent() (أنظر سطر رقم 33). اذا كانت قيمة result المرجعة تساوي -1 فهذا يعني أن عملية الإضافة لم تتم بنجاح حيث يتم طباعة رسالة Toast تبين ذلك (أنظر الأسطر من 34 إلى 43). لاحظ أيضاً أن الاتصال مع قاعدة البيانات يبدأ في الدالة onCreate() (أنظر سطر رقم 8). يبقى الاتصال مفتوحاً حتى يتم إغلاق أو ايقاف الفعالية عند تنفيذ الدالة onPause() (أنظر سطر رقم 51) والتي يتم فيها إغلاق الاتصال مع قاعدة البيانات.

```
1:  public class InsertActivity extends Activity {
2:
3:   EditText nameTxt;
4:   EditText ageTxt;
5:   StudentDBUtility dbUtility;
6:
7:   @Override
8:   protected void onCreate(Bundle savedInstanceState) {
9:        super.onCreate(savedInstanceState);
10:       setContentView(R.layout.activity_insert);
11:       dbUtility = new StudentDBUtility(this,
12:  MySQLiteHelper.DATABASE_NAME,
13:  MySQLiteHelper.DATABASE_VERSION);
14:       dbUtility.open();
15:       nameTxt = (EditText)
16:  this.findViewById(R.id.insertName);
17:       ageTxt = (EditText)
18:  this.findViewById(R.id.insertAge);
19:       Button confirmInsertBtn = (Button)
20:  this.findViewById(R.id.confirmInsertBtn);
21:       confirmInsertBtn.setOnClickListener(new
22:  OnClickListener(){
23:
24:            @Override
25:            public void onClick(View v) {
26:                 if(nameTxt.getText() !=null &&
27:  ageTxt.getText() !=null){
28:                      String name =
29:  nameTxt.getText().toString();
30:                      int age =
31:  Integer.parseInt(ageTxt.getText().toString());
32:                      long result =
33:  dbUtility.insertStudent(name, age);
34:                      if(result == -1){
```

```
35:
36:  Toast.makeText(getApplicationContext(), "Error occured
37:  while inserting data", Toast.LENGTH_SHORT).show();
38:                       }else{
39:
40:  Toast.makeText(getApplicationContext(), "Item inserted
41:  successfully", Toast.LENGTH_SHORT).show();
42:                            nameTxt.setText("");
43:                            ageTxt.setText("");
44:                       }
45:                  }
46:             }
47:        });
48:  }
49:
50:  @Override
51:  protected void onPause() {
52:       super.onPause();
53:       dbUtility.close();
54:  }
55:}
```

رابعاً: الفعالية DeleteActivity والتي يتم من خلالها الحذف عن طريق اختيار العنصر المراد حذفه من قائمة (ListView) ومن ثم النقر على زر Delete في القائمة الجانبية. الكود الخاص بهذه الفعالية موضح فيما يلي:

```
1:  public class DeleteActivity extends ListActivity {
2:
3:   StudentDBUtility dbUtility;
4:   StudentCustomAdapter adapter;
5:
6:   @Override
7:   protected void onCreate(Bundle savedInstanceState) {
8:        super.onCreate(savedInstanceState);
9:        dbUtility = new StudentDBUtility(this,
10:  MySQLiteHelper.DATABASE_NAME,
11:  MySQLiteHelper.DATABASE_VERSION);
12:       dbUtility.open();
13:       List<Student> students =
14:  dbUtility.getAllStudents();
15:       adapter = new StudentCustomAdapter(this,
16:  students);
```

```
17:       this.setListAdapter(adapter);
18:       this.getListView().setOnItemClickListener(new
19:  OnItemClickListener(){
20:
21:            @Override
22:            public void onItemClick(AdapterView<?>
23:  parent, View view,
24:                        int position, long id) {
25:                  Student student =
26:  (Student)parent.getItemAtPosition(position);
27:                  showPopup(view, student);
28:            }
29:       });
30:  }
31:
32:  public void showPopup(View view, final Student
33:  student){
34:       PopupMenu popUpMenu = new PopupMenu(this, view);
35:  popUpMenu.getMenuInflater().inflate(R.menu.
36:  contextmenu, popUpMenu.getMenu());
37:       popUpMenu.setOnMenuItemClickListener(new
38:  OnMenuItemClickListener(){
39:
40:            @Override
41:            public boolean onMenuItemClick(MenuItem
42:  menuItem) {
43:
44:  dbUtility.deleteStudent(student.getId());
45:                  adapter.remove(student);
46:                  return false;
47:            }
48:       });
49:       popUpMenu.show();
50:  }
51:  @Override
52:  protected void onPause() {
53:       super.onPause();
54:       dbUtility.close();
55:  }
56:}
```

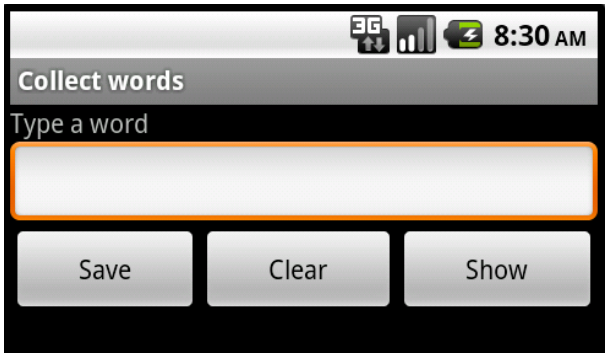
لاحظ أن الفعالية تستخدم أيضاً كائن من نوع StudentDBUtility للوصول لقاعدة البيانات (أنظر سطر رقم 9)، حيث يتم في البداية الاستعلام عن كل الطلاب من خلال الدالة ()getAllStudentsوعرضها في القائمة (ListView) باستخدام محول من نوع StudentCustomAdapter (أنظر الأسطر من 14 إلى 17). تم أيضاً إضافة مستمع (Listener) من نوع (OnItemClickListener) للاستماع لحدث النقر على أي سطر في القائمة (ListView) (أنظر سطر رقم 18). عند اختيار أي طالب من القائمة يتم استرجاع الكائن Student الذي تم اختياره ومن ثم يتم عرض قائمة سياق Context Menu تظهر زر الحذف Delete. يتم إنشاء قائمة السياق بتنفيذ الدالة ()showPopup والتي يمرر لها كائن Student المراد حذفه (أنظر الأسطر من 25 إلى 27). من خلال الدالة ()onMenuItemClick (أنظر الأسطر 44 و45) يتم تنفيذ أمر الخذف عند النقر على زر Delete حيث يتم تنفيذ الدالة ()deleteStudent الموجودة في StudentDBUtility ويمرر لها رقم المعرف الخاص بالطالب.

أسئلة على الوحدة السادسة

1. ما الفرق بين تخزين البيانات في الذاكرة الداخلية وتخزينها في الذاكرة الخارجية؟

2. قم ببناء التطبيق الموضح بالأسفل: يقوم المستخدم بإدخال كلمات وحفظها بالنفر على الزر "Save". الكلمات المدخلة يتم حفظها في قائمة من نوع ArrayList<String>. عند النقر على زر "Show" يتم طباعة الكلمات المحفوظة باستخدام Toast.

أحد مشاكل هذا التطبيق أنه عند حفظ بعض الكلمات وتدوير الجهاز، يتم فقدان كل الكلمات المحفوظة. السبب في ذلك أن النظام يقوم تلقائياً بإنهاء الفعالية (Activity) وإعادة تشغيلها عند تغيير إعدادات العرض. قم بعلاج هذه المشكلة باستخدام التفضيلات المشتركة (SharedPreferences) لحفظ الكلمات واسترجاعها عند إعادة تشغيل الفعالية.



3. قم بالتعديل على تمرين 2-6 وذلك بإضافة فعالية جديدة باسم SearchActivity يمكن من خلالها البحث عن طالب باستخدام الأسم (أو جزء من الأسم).

4. في تمرين 2-6، يتم تنفيذ إجراءات الاستعلام والإضافة والحذف على قاعدة البيانات. قم بتعديل التطبيق لينفذ إجراء التعديل (Update) على بيانات طالب محدد. قم أولاً بتعديل الفئة StudentDBUtility وذلك بإضافة دالة باسم updateStudentDetails() والتي يمرر لها ثلاث قيم هي: رقم المعرف ID الخاص بالطالب المراد التعديل على بيانات، اسم الطالب المعدل وعمره المعدل. قم بعد ذلك بالتعديل على القائمة المنبثقة التي تظهر في الفعالية (DeleteActivity) وذلك ليتم إظهار زر باسم "Update" عند النقر على اسم أي طالب. عند النقر على زر "Update" يتم إظهار فعالية لتعديل على البيانات القديمة التي تظهر في عناصر الإدخال من نوع EditText.

5. في تمرين 2-6 تم إنشاء قاعدة البيانات باستخدام فئة متفرعة (subclass) من الفئة SQLiteOpenHelper. قم بإنشاء قاعدة بيانات مطابقة لتلك المستخدمة في التمرين باستخدام الأداة البرمجية SQLite Database Browser، ثم قم بعمل التعديلات اللازمة لاستخدامها بدلاً من الفئة SQLiteOpenHelper.

الوحدة السابعة:

مزودات المحتوى

(Content Providers)

يتعلم الطالب في هذه الوحدة:

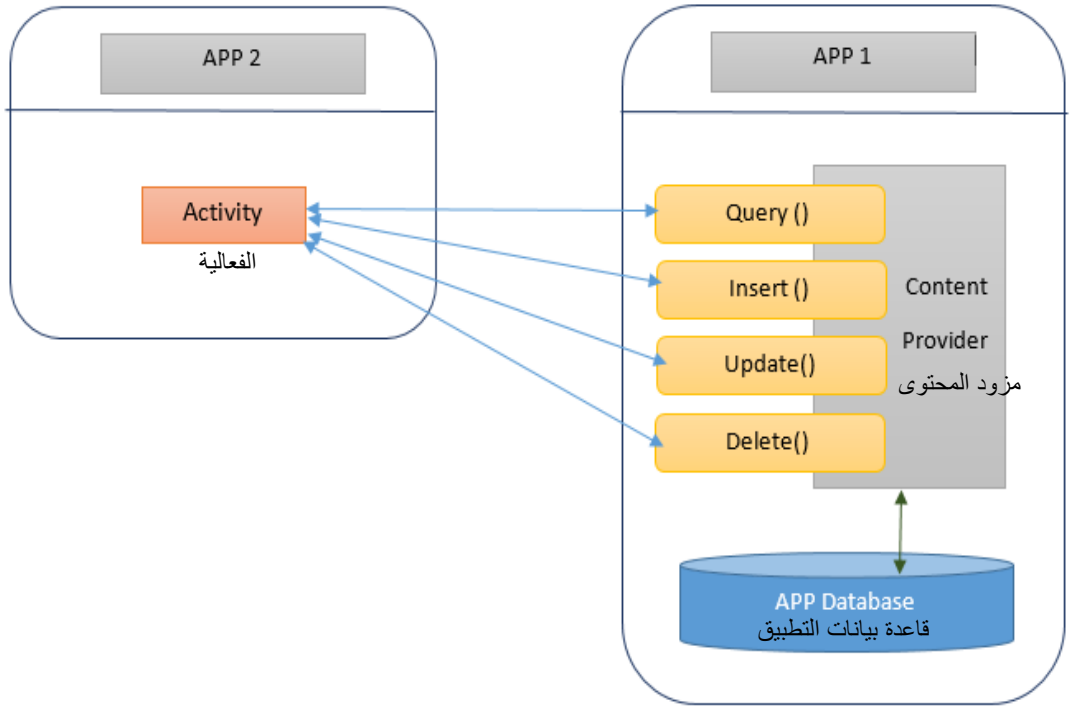
- ✓ مفهوم مزود المحتوى (Content Provider) واستخداماته.
- ✓ مكونات مزود المحتوى وطريقة الوصول إليه للتواصل مع قواعد البيانات الموجودة ضمن تطبيقات أخرى.
- ✓ التدريب العملي على إنشاء و استخدام مزود محتوى يتيح الوصول إلى قاعدة بيانات.

لدراسة هذه الوحدة لابد من الإلمام بإنشاء تطبيقات أندرويد المعتمدة على قواعد البيانات كما تم تفصيله في الوحدة السادسة.

توفر مزودات المحتوى (Content Providers) الإمكانية للوصول للبيانات (قواعد البيانات أو الملفات) من تطبيقات مختلفة. فمثلاً، اذا أردت مشاركة قاعدة البيانات المستخدمة في تطبيق ما، يجب إنشاء مزود محتوى (Content Provider) حول قاعدة البيانات ليوفر الوصول إليها من تطبيقات أخرى. فكرة مزود المحتوى (Content Provider) تشبه إلى حد كبير هيكلية العميل/المزود (Client-Server Architecture) حيث يمثل التطبيق الذي يسعى للوصول للبيانات دور العميل (Client) بينما يمثل مزود المحتوى دور المزود (Server). شكل 1-7 يوضح فكرة عمل مزود المحتوى (Content Provider): التطبيق App1 يستخدم قاعدة بيانات حيث لا يمكن الوصول إليها إلا من خلاله. لتمكين تطبيق آخر App2 مثلاً من الوصول لقاعدة البيانات الخاصة بـ App1 واستخدامها، يجب إنشاء مزود محتوى (Content Provider) يوفر عدد من الدوال (query, insert, update and delete) التي يمكن تنفيذها من أي تطبيق بعيد للوصول لقاعدة البيانات.

يوفر نظام أندرويد عدد من مزودات المحتوى (Built-in Content Providers) للوصول إلي بعض قواعد البيانات الخاصة بالنظام مثل جهات الاتصال (Contacts)، مخزن الوسائط (MediaStore)، المفضلات (Bookmarks)، والإعدادات (settings) وغيرها. على سبيل المثال، يستطيع أي تطبيق طبقاً للصلاحيات الممنوحة له من الوصول لجهات الاتصال الموجودة في الجهاز وذلك لقراءتها، التعديل عليها أو استعمالها لأي أمر آخر.

في هذا الفصل سنطرق إلى إنشاء مزود المحتوى (Content Provider) للوصول إلى قاعدة البيانات الخاصة بتطبيق ما.



شكل 1-7: الاتصال بين الفعالية (Activity) ومزود المحتوى (Content Provider)[1]

إنشاء مزود المحتوى (Content Provider)

يتم إنشاء مزود المحتوى (Content Provider) عن طريق إنشاء فئة فرعية (subclass) من الفئة android.content.ContentProvider، ومن ثم كتابة كود في بعض الدوال التي تتحكم بالوصول للبيانات وهي:

- الدالة onCreate(): وهذه الدالة يتم تنفيذها تلقائياً من قبل النظام عند إنشاء مزود المحتوى لأول مرة ويتم فيها تنفيذ الإعدادات الضرورية لتشغيل مزود المحتوى.
- الدالة query(): يتم تنفيذ هذه الدالة من قبل الزبون Client لاسترجاع بيانات من مزود المحتوى. الكود الذي يجب كتابته في هذه الدالة يحدد نوعية البيانات وطريقة استخراجها ومن ثم إرجاعها للزبون مغلفة بكائن من نوع Cursor.
- الدالة insert(): هذه الدالة يتم استدعاؤها عند إضافة سطر row إلى في قاعدة البيانات. يجب كتابة الكود اللازم في هذه الدالة ليتم الإضافة إلى قاعدة البيانات ومن ثم إرجاع معرف URI للسطر الجديد.

[1] CompileTimeError, http://www.compiletimeerror.com/2013/12/content-provider-in-android.html

- الدالة ()update: يتم تنفيذ هذه الدالة عند الحاجة لتعديل بعض البيانات من قبل المستخدم. يمرر لهذه الدالة القيم المعدلة لسطور معينة ومن ثم يتم إرجاع عدد السطور التي تم تعديلها.
- الدالة ()delete: وتستخدم في حالة حذف سطور محددة في قاعدة البيانات حيث ترجع هذه الدالة عدد السطور التي تم حذفها.
- ()getType: وترجع نوعالبيانات MIME Type (مثل text، HTML وغيره).

قبل شرح طريقة إنشاء مزود المحتوى عملياً، لا بد من شرح بعض المفاهيم الأساسية المتعلقة باستخدام مزود المحتوى (Content Provider) والوصول إليها.

معرف الوصول للمحتوى (The Content URI)

قد يحتوي الجهاز على أكثر من مزود محتوى (Content Provider) يتم الوصول إليها من تطبيقات مختلفة. لذلك، لا بد من طريقة لتمييز كل مزود محتوى. بالإضافة إلى ذلك، فإن مزود المحتوى قد يوفر الوصول لمحتويات متعددة (أكثر من جدول في قاعدة البيانات)، لذلك لابد أن تقوم التطبيقات التي تريد الوصول لمزود المحتوى من تحديد المحتوى المحدد المراد الوصول إليه (جدول أو سطر داخل الجدول)، وهو ما يتم باستخدام معرف الوصول للمحتوى (Content URI).

معرف الوصول للمحتوى (Content URI) مشابه لروابط الصحفات على شبكة الإنترنت، وهو عبارة عن جملة معرفة تبدأ بالمقطع content:// وتتكون من أكثر من جزء: الجزء الأول يسمى Authority ويحدد مزود المحتوى. عادةً يستخدم اسم الرزمة package للفئة الخاصة بمزود المحتوى. مثال:

content://ps.edu.ucas.mycontentprovider

الجزء الثاني يتم إضافته للجزء الأول ليحدد الوصول إلى جدول معين في قاعدة البيانات. فمثلاً، للوصول إلى الجدول student الموجود في قاعدة البيانات لمزود المحتوى ps.edu.ucas.mycontentprovider، يتم إضافة اسم الجدول student بعد Authority ليصبح:

content://ps.edu.ucas.mycontentprovider/student

للوصول إلى سطر محدد في قاعدة البيانات، مثل تعديل أو حذف سطر محدد، يتم إضافة معرف السطر، وهو عادةً رقم يزداد تلقائياً وممثل بعمود في الجدول، إلى نهاية معرف الوصول Content URI ليصبح:

ps.edu.ucas.mycontentprovider.StudentProvider/student/3

حيث الرقم الأخير يحدد معرف السطر المراد الوصول إليه.

عند القيام بأي إجراء على قاعدة البيانات مثل الإضافة، الحذف، التعديل أو الاستعلام، يتم تمرير معرف الوصول للمحتوى Content URI للدالة المطلوبة لتحديد المحتوى المطلوب تنفيذ الإجراء عليه. فمثلاً، لحذف السطر ذي المعرف id=3، يمكن تنفيذ الدالة ()delete وتمرير معرف الوصول الموضح بالأعلى.

الإعلان عن مزود المحتوى في ملف الوثيقة (Manifest)

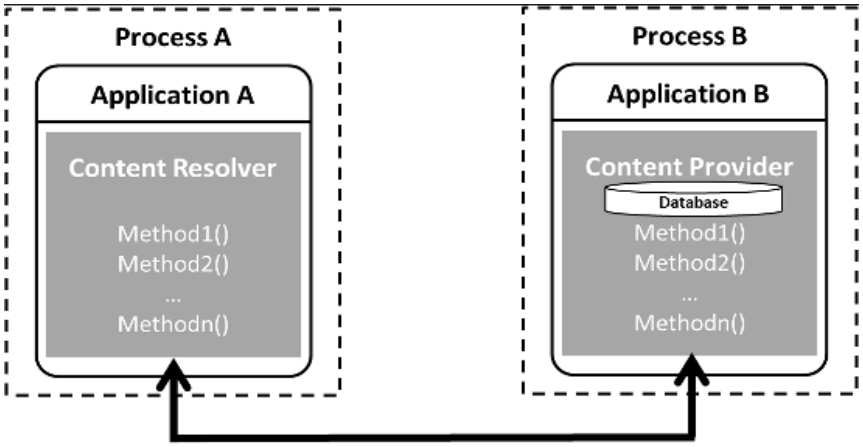
حتى يتعرف النظام على مزود المحتوى، يجب الإعلان عنه في ملف الوثيقة (Manifest) الخاص بالتطبيق الذي أنشئ به مزود المحتوى. على سبيل المثال، الكود التالي يتم إضافته إلى ملف (Manifest):

```
<provider android:name="
ps.edu.ucas.mycontentprovider.StudentProvider"
android:authorities="
ps.edu.ucas.mycontentprovider.StudentProvider"
android:exported="true"/>
```

حيث أن الخاصية android:authorities تحدد معرف مزود المحتوى، بينما الخاصية android:name تحدد اسم الفئة (class) التي يتم فيها تطبيق دوال مزود المحتوى، وفي الغالب تستخدم نفس القيمة المحددة لـ android:authorities. الخاصية android:exported تحدد ما إذا كان مزود المحتوى (Content Provider) متاح للتطبيقات الأخرى (android:exported="true") أم لا (android:exported="false").

مستخرج المحتوى (Content Resolver)

الوصول لمزود المحتوى (Content Provider) واستخدامه يتم باستخدام محلل المحتوى (Content Resolver)، وهو كائن (Object) يمكن الوصول إليه بتنفيذ الدالة getContentResolver() الموجودة ضمن Application Context. باستخدام مستخرج المحتوى (ContentResolver)، يمكن تنفيذ كل الدوال التي يوفرها مزود المحتوى مثل insert، query، delete وغيره. أي أن مستخرج المحتوى (ContentResolver) يتواصل مع مزود المحتوى (Content Provider) لتنفيذ الإجراءات التي يطلبها التطبيق الزبون (Client Application) (أنظر شكل 2-7).



شكل 2-7: الوصول لمزود المحتوى (Content Provider) باستخدام مستخرج المحتوى (Content Resolver) [1]

[1] Pact Publishing, https://www.packtpub.com/books/content/what-content-provider

تمرين عملي (1-7)

سنقوم في هذا التمرين العملي باستكمال التمرين الخاص بقواعد البيانات (تمرين 2-6) ،والتي تم تفصيله في الوحدة السادسة ،وذلك بإنشاء مزود محتوى (Content Provider) لتمكين التطبيقات الأخرى للوصول إلى قاعدة البيانات المنشأه مسبقاً باسم studentdb.

الخطوة الأولى في إنشاء مزود المحتوى (Content Provider) هي إنشاء فئة (class) متفرعة من الفئة ContentProvider. بعد ذلك نقوم بكتابة الكود في الدوال الموجودة بها للتواصل مع قاعدة البيانات واستخدامها. انتبه أن الفئة الجديد (class) يتم إنشاؤها ضمن تطبيق قواعد البيانات السابق وليس ضمن تطبيق جديد. الشكل يوضح الكود الخاص بالفئة الجديدة.

```
1:  public class StudentProvider extends ContentProvider{
2:
3:   MySQLiteHelper helper;
4:   public static final String AUTHORITY =
5:  "ps.edu.ucas.sqlite.contentprovider.StudentProvider";
6:
7:   UriMatcher uriMatcher = new
8:  UriMatcher(UriMatcher.NO_MATCH);
9:
10:  public static final int CHOICE_STUDENT_TABLE = 1;
11:  public static final int CHOICE_STUDENT_RECORD = 2;
12:
13:  @Override
14:  public boolean onCreate() {
15:       helper = new MySQLiteHelper(this.getContext(),
16:  MySQLiteHelper.DATABASE_NAME,
17:  MySQLiteHelper.DATABASE_VERSION);
18:       uriMatcher.addURI(AUTHORITY,
19:  MySQLiteHelper.STUDENT_TABLE, CHOICE_STUDENT_TABLE);
20:       uriMatcher.addURI(AUTHORITY,
21:  MySQLiteHelper.STUDENT_TABLE+"/#",
22:  CHOICE_STUDENT_RECORD);
23:
24:       return false;
25:  }
26:
27:  @Override
28:  public Uri insert(Uri uri, ContentValues values) {
29:       int result = uriMatcher.match(uri);
```

```
30:       long id = -1;
31:       if(result == CHOICE_STUDENT_TABLE){
32:            SQLiteDatabase db =
33:  helper.getWritableDatabase();
34:            id = db.insert(MySQLiteHelper.STUDENT_TABLE,
35:  null, values);
36:
37:  this.getContext().getContentResolver().notifyChange(
38:  uri, null);
39:       }else{
40:            throw new IllegalArgumentException("Invalid
41:  Uri");
42:       }
43:       return Uri.parse("content://"+AUTHORITY +
44:  MySQLiteHelper.STUDENT_TABLE+"/"+id);
45:  }
46:
47:  @Override
48:  public int delete(Uri uri, String selection, String[]
49:  selectionArgs) {
50:       int result = uriMatcher.match(uri);
51:       int deleteRecord = -1;
52:       if(result == CHOICE_STUDENT_TABLE){
53:            SQLiteDatabase db =
54:  helper.getWritableDatabase();
55:            deleteRecord =
56:  db.delete(MySQLiteHelper.STUDENT_TABLE, selection,
57:  selectionArgs);
58:
59:  this.getContext().getContentResolver().notifyChange(
60:  uri, null);
61:       }else if(result == CHOICE_STUDENT_RECORD){
62:            SQLiteDatabase db =
63:  helper.getWritableDatabase();
64:            String id = uri.getLastPathSegment();
65:            deleteRecord =
66:  db.delete(MySQLiteHelper.STUDENT_TABLE, "_id = "+id+"
67:  and "+selection, selectionArgs);
68:
69:  this.getContext().getContentResolver().notifyChange(
```

```
70:  uri, null);
71:       }else{
72:            throw new IllegalArgumentException("Invalid
73:  uri");
74:       }
75:       return deleteRecord;
76:  }
77:
78:  @Override
79:  public String getType(Uri uri) {
80:       return null;
81:  }
82:
83:  @Override
84:  public Cursor query(Uri uri, String[] projection,
85:  String selection,
86:            String[] selectionArgs, String sortOrder) {
87:       int result = uriMatcher.match(uri);
88:       if(result == CHOICE_STUDENT_TABLE){
89:            SQLiteDatabase db =
90:  helper.getReadableDatabase();
91:            return
92:  db.query(MySQLiteHelper.STUDENT_TABLE, projection,
93:  selection, selectionArgs, null, null, sortOrder);
94:       }
95:       return null;
96:  }
97:
98:  @Override
99:  public int update(Uri uri, ContentValues values,
100:  String selection,
101:            String[] selectionArgs) {
102:       int result = uriMatcher.match(uri);
103:       int updatedRecord = -1;
104:       if(result == CHOICE_STUDENT_TABLE){
105:            SQLiteDatabase db =
106:  helper.getWritableDatabase();
107:            updatedRecord =
108:  db.update(MySQLiteHelper.STUDENT_TABLE, values,
109:  selection, selectionArgs);
```

```
110:
111: this.getContext().getContentResolver().notifyChange(
112:  uri, null);
113:      }else if(result == CHOICE_STUDENT_RECORD){
114:           SQLiteDatabase db =
115:  helper.getWritableDatabase();
116:           String id = uri.getLastPathSegment();
117:           updatedRecord =
118:  db.update(MySQLiteHelper.STUDENT_TABLE, values, "_id
119:  = "+id+" and "+selection, selectionArgs);
120:
121:  this.getContext().getContentResolver().notifyChange(
122:  uri, null);
123:      }else{
124:           throw new IllegalArgumentException("Invalid
125:  uri");
126:      }
127:      return updatedRecord;
128: }
129:}
```

الثابت AUTHORITY (أنظر سطر رقم 4 و5) يحدد المعرف الرئيس لمزود المحتوى (Content Provider) وهو: "ps.edu.ucas.sqlite.contentprovider.StudentProvider". لاحظ أن مزود المحتوى (Content Provider) يستخدم كائن من الفئة MySQLiteHelper (أنظر سطر رقم 3 و15) والتي تم إنشاؤها في تمرين 2-6 للوصول إلى قاعدة البيانات من خلاله. لاحظ أنه من خلال الدالة ()onCreate (أنظر الأسطر من 14 إلى 25) تم إنشاء الكائن من نوع MySQLiteHelper وتمرير اسم قاعدة البيانات، studentdb، ورقم إصدارها 1.

قبل شرح الكود الذي يجب كتابته في دوال مزود المحتوى (Content Provider)، يجب التوضيح أن التطبيق الزبون Client يقوم بإرسال المعرف URI الخاص بالمحتوى الذي يريد الوصول إليه للتعديل أو الحذف أو الإضافة. لذلك فإن الدوال query, delete, update, insert كلها تحتاج إلى مدخل URI يحدد المحتوى المراد تنفيذ الإجراء عليه. لاحظ أن المحتوى المراد الوصول إليه قد يكون الجدول كاملاً أو سطر معين في الجدول. لذلك يجب أن يقوم مزود المحتوى بترجمة وتحليل معرف المحتوى Content URI لمعرفة ما إذا كان يشير إلى جدول أو سطر محدد في الجدول، وللقيام بذلك نستعين بكائن من نوع UriMatcher (أنظر سطر رقم 7 و8) والذي نقوم من خلاله بتحديد الصيغ المختلفة لمعرف المحتوى URI والقيم التي يتم إرجاعها مع كل صياغة (أنظر الأسطر من 18 إلى 22). مزود المحتوى الخاص بنا يتعامل مع صيغتين لمعرف المحتوى Content URI هما:

- content://<AUTHORITY><TABLE_NAME> (معرف مزود المحتوى متبوعاً باسم الجدول)

- content://<AUTHORITY><TABLE_NAME>/# (معرف مزود المحتوى متبوعا باسم الجدول متبوعاً برقم السطر)

عند تمرير أي قيمة معرف محتوى Content URI من خلال الدالة query مثلاً، يتم مطابقة القيمة الواردة بالصيغ الموجودة في UriMatcher وعند تطابقها مع أي صيغة تقوم بإرجاع رقم يدل على الصيغة المطابقة. فمثلاً اذا تم تمرير المعرف التالي Content URI:

content://ps.edu.ucas.sqlite.contentprovider.studentprovider/Student

يقوم UriMatcher بإرجاع القيمة CHOICE_STUDENT_TABLE

بينما إذا تم تمرير المعرف التالي Content URI:

content://ps.edu.ucas.sqlite.contentprovider.studentprovider/Student/5

يقوم UriMatcher بإرجاع القيمة CHOICE_STUDENT_RECORD

حيث أن طريقة تنفيذ الإجراء المطلوب على قاعدة البيانات يعتمد على قيمة المطابقة المرجعة من UriMatcher.

سنتطرق الآن إلى الكود الذي سيتم كتابته في بعض دوال مزود المحتوى:

في الدالة insert() (أنظر الأسطر من 28 إلى 45) نقوم بكتابة الكود لتنفيذ الخطوات التالية:

1. استخدام UriMatcher لتحديد نوع URI ما إذا كان يشير إلى الجدول أو إلى سطر في الجدول.في المثال الحالي سنتعامل مع URI الخاص بالجدول فقط (الأسطر من 29 إلى 31).
2. رمي استثناءException إذا كان URI غير صالح (الأسطر من 39 إلى 42).
3. الحصول على مؤشر من نوع SQLiteDatabase قابل للكتابة عن طريق الدالة getWritableDatabase() (سطر 32 و33).
4. إجراء عملية insertSQL لإدراج البيانات الموجودة ضمن ContentValues في جدول قاعدة البيانات (الأسطر 34 و35).
5. إخطار مستخرج المحتوى ContentResolver بالتعديل الذي تم عن طريق تنفيذ الدالة notifyChange() (سطر 37 و38).
6. إعادة معرف محتوى Content URI يمثل سطر الجدول المضاف حديثاً (سطر رقم 43 و44).

في الدالة query() (أنظر الأسطر من 84 إلى 96) نقوم بكتابة كود لتنفيذ الخطوات التالية:

1. استخدام UriMatcher لتحديد نوع URI ما إذا كان يشير إلى الجدول أو إلى سطر في الجدول.في المثال الحالي سنتعامل مع URI الخاص بالجدول فقط (الأسطر 87 و88).

2. الحصول على مؤشر من نوع SQLiteDatabase قابل للقراءة فقط عن طريق الدالة getReadableDatabase() (سطر رقم 89 و90).
3. تنفيذ الاستعلاء باستخدام المدخلات الأخرى للدالة query() ومن ثم إرجاع الناتج مغلفاً بكائن من نوع Cursor (سطر رقم 92 و93).

لاحظ في الدالة update() (أنظر الأسطر من 99 إلى 129) يتم تحديد نوع URI والتعامل مع كل نوع بتنفيذ التعليمة SQL Update المناسبة. فإذا كان معرف المحتوى (Content URI) يشير إلى الجدول في قاعدة البيانات (يتم فحص هذا الشرط في سطر رقم 104)، فمن المفترض أن شرط التعديل (جملة WHERE) ،والذي يحدد الأسطر التي يتم التعديل عليها ،موجود في المتغير selection، أما إذا كان معرف المحتوى يشير إلى سطر معين، فيتم اضافة رقم الصف id إلى الجملة selection لتكون ضمن جملة WHERE (أنظر الأسطر من 116 إلى 119). الدالة delete() (أنظر الأسطر من 48 إلى 76) تقوم بنفس الإجراء وتنفذ تعليمة SQL Delete بناء على نوع معرف المحتوى.

بعد إنشاء الفئة StudentProvider الخاصة بمزود المحتوى، يجب الإعلان عن مزود المحتوى في ملف Manifest كالتالي:

```
<provider android:name="
ps.edu.ucas.sqlite.contentprovider.StudentProvider"
android:authorities="
ps.edu.ucas.sqlite.contentprovider.StudentProvider"
android:exported="true"/>
```

بعد ذلك يصبح مزود المحتوى StudentProvider جاهزاً للاستخدام من أي تطبيق آخر، حيث يمكن الوصول إليه من خلال المعرف URI التالي:

content:// ps.edu.ucas.sqlite.contentprovider.StudentProvider

ومن ثم إلحاق اسم الجدول والسطر المراد الوصول إليه.

بعد توضيح خطوات إنشاء مزود المحتوى (Content Provider) سنقوم بإنشاء تطبيق جديد للوصول لمزود المحتوى واستخدامه. التطبيق الجديد سيكون له نفس واجهات تطبيق قواعد البيانات الموضح بالشكل، ويقوم بنفس الوظائف، وهي عرض بيانات الطلاب والإضافة والحذف، مع اختلاف واحد وهو الوصول إلى قاعدة البيانات عن بعد من خلال مزود المحتوى StudentProvider الذي تم إنشاؤه.

التطبيق الجديد لن يقوم باستخدام كائن من نوع SQLiteOpenHelper بل فقط كائن نوع ContentResolver للوصول لقاعدة بيانات التطبيق البعيد من خلال مزود المحتوى Content Provider. التطبيق الجديد يستخدم الفئة StudentDBUtility لتنظيم الوصول لمزود المحتوى وإخفاء تفاصيل الاتصال والاستعلام. الفئة StudentDBUtility مشابهة بتلك المستخدمة في تطبيق قواعد البيانات – تمرين 2-6، باستثناء أنها هنا تستخدم

كائن من نوع ContentResolver بدلاً من SQLiteOpenHelper. الكود الخاص بالفئة StudentDBUtility الجديدة موضح فيما يلي:

```
1:  public class StudentDBUtility {
2:
3:   ContentResolver resolver;
4:   public static final String AUTHORITY =
5:  "content://ps.edu.ucas.sqlite.studentprovider.
6:  StudentProvider";
7:   public static final Uri STUDENT_PROVIDER_URI =
8:  Uri.parse(AUTHORITY+"/student");
9:   public static final String[] allColumns =
10:  {"_id","name","age"};
11:
12:  public StudentDBUtility(Context context){
13:       resolver = context.getContentResolver();
14:  }
15:
16:  public Uri insertStudent(String name, int age){
17:       ContentValues values = new ContentValues();
18:       values.put("name", name);
19:       values.put("age", age);
20:       return resolver.insert(STUDENT_PROVIDER_URI,
21:  values);
22:  }
23:
24:  public List<Student> getAllStudents(){
25:       List<Student> students = new
26:  ArrayList<Student>();
27:       Cursor cursor =
28:  resolver.query(STUDENT_PROVIDER_URI, allColumns, null,
29:  null, "name");
30:       cursor.moveToFirst();
31:       while(!cursor.isAfterLast()){
32:            int id = cursor.getInt(0);
33:            String name = cursor.getString(1);
34:            int age = cursor.getInt(2);
35:            Student s = new Student(id, name, age);
36:            students.add(s);
37:            cursor.moveToNext();
```

```
38:         }
39:         cursor.close();
40:         return students;
41:   }
42:
43:   public long deleteStudent(int id){
44:         return resolver.delete(STUDENT_PROVIDER_URI, "_id
45:   = ?", new String[]{String.valueOf(id)});
46:   }
47:   // Not used for contentprovider
48:   public void open(){}
49:   // Not used for contentprovider
50:   public void close(){}
51:}
```

لاحظ أنه في الباني Constructor الخاصة بالفئة StudentDBUtility (أنظر سطر رقم 12)، يتم الوصول للكائن من نوع ContentResolver والمتوفر في context من خلال تنفيذ الأمر التالي:

```
resolver = context.getContentResolver();
```

الدوال الأخرى مثل ()getAllStudents، ()insertStudent و ()deleteStudent مطابقة لتلك الموجودة في التطبيق السابق، باستثناء استخدام مستخرج المحتوى (Content Resolver) لتنفيذ الإجراء المطلوب على قاعدة البيانات من خلال مزود المحتوى.

بعد إنشاء StudentDBUtility يمكن استخدام نفس كود الفعاليات (Activities) وتصميماتها الموجودة في التطبيق السابق دون أي تغيير، ومن المفترض أن يعمل التطبيق تماماً مثل التطبيق السابق باستثناء أن قاعدة البيانات موجودة في تطبيق آخر.

أسئلة على الوحدة السابعة

1. ما الفائدة من إنشاء واستخدام مزود المحتوى (Content Provider)؟
2. أذكر الدوال التي يجب كتابتها عند إنشاء مزود محتوى جديد؟
3. كيف يتم تسجيل مزود المحتوى في ملف الوثيقة (Manifest)؟
4. قم بالتعديل على تمرين 1-7 وذلك بإضافة فعالية جديدة للبحث عن بيانات طالب من خلال مزود المحتوى (Content Provider) ومن ثم عرض النتائج على الشاشة.

الوحدة الثامنة:

مستقبلات النشر

(Broadcast Receivers)

يتعلم الطالب في هذه الوحدة:

- ✓ مفهوم مستقبل النشر (Broadcast Receiver) واستخداماته.
- ✓ إنشاء مستقبلات نشر (Broadcast) للاستماع والاستجابة لأحداث النظام المختلفة
- ✓ تهيئة التطبيقات لنشر (Broadcast) رسائل إلى مستقبلات النشر الموجودة ضمن تطبيقات أخرى.
- ✓ مفهوم الإشعارات (Notifications) واستخداماتها.

لدراسة هذه الوحدة لابد من الإلمام بمفهوم الهدف (Intent) وأنواعه، واستخدامه للتواصل بين الفعاليات (إرجع إلى الوحدة الرابعة).

Broadcast Receiver أو مستقبل النشر هو مكون من مكونات تطبيق أندرويد يستخدم للاستماع للأحداث (Events) الصادرة من النظام أو من تطبيقات أخرى. عندما يحصل حدث معين، يقوم النظام، أو تطبيق آخر، بارسال رسائل من نوع (Intent) وذلك لإشعار كل مستقبلات النشر (Broadcast Receivers) التي تستمع لهذا الحدث.

تقوم مستقبلات النشر بدورها بتنفيذ إجراء معين عند حصول الحدث المستمع له. على سبيل المثال، يمكن إنشاء مستقبل نشر (Broadcast Receiver) للاستماع لحدث إعادة تشغيل الجهاز (Boot) أو الحصول على إشارة الاتصال بالانترنت. عند حصول أي من هذه الأحداث يقوم النظام تلقائياً بنشر (broadcast) رسالة من نوع (Intent) وذلك لإشعار مستقبل النشر (Broadcast Receiver) بذلك والذي يقوم بدوره بتنفيذ إجراء معين مثل تشغيل تطبيق ما أو استئناف تحميل ملف من الانترنت أو إرسال إشعار (Notification) بالحدث للمستخدم. مستقبل النشر (Broadcast Receiver) هومجرد "بوابة" للمكونات الأخرى ويهدف إلى القيام بكمية ضئيلة جدا من العمل. على سبيل المثال ، بدء خدمة (Service) لأداء بعض الأعمال على أساس هذاالحدث.

هناك خطوتان لتفعيل مستقبل النشر (Broadcast Receiver) لاستماع للرسائل الصادرة من النظام وهما:

1. إنشاء مستقبل النشر: وهذا يتم بإنشاء فئة متفرعة (subclass) من الفئة BroadcastReceiver، وكتابة الدالة ()onReceive وفيها يحدد الإجراء الذي يتم تنفيذه عند استقبال رسالة من نوع (Intent) والتي تفيد بحصول حدث معين. المثال التالي يوضح طريقة إنشاء مستقبل نشر يقوم تلقائياً بتشغيل فعالية (Activity) معينة، مثل متصفح الإنترنت، عند استقبال رسالة (Intent) من النظام.

```
public class MyReceiver extends BroadcastReceiver {
@Override
  public void onReceive(Context context, Intent intent) {
    Intent newIntent = new Intent(Intent.ACTION_VIEW);
    newIntent.setData(Uri.parse("http://www.ucas.edu.ps"));
    context.startActivity(newIntent));
  }
}
```

لاحظ أن المتغير (Intent) يمثل الرسالة التي تم نشرها من النظام، حيث يتم استقباله كمدخل للدالة ()onReceive، بينما المتغير (newIntent) يتم إنشاؤه لتنفيذ الإجراء المطلوب بعد استقبال رسالة النظام وهو في المثال السابق تشغيل فعالية.

2. تسجيل مستقبل النشر: بعد إنشاء مستقبل النشر كمكون لتطبيق أندرويد، لا بد من تسجليه في ملف الوثيقة (Manifest) التابع للتطبيق وتحديد نوع الحدث الذي يستمع لها. فمثلاً، اذا اردنا لمستقبل النشر المنشأ في المثال السابق الاستماع لحدث تشغيل الجهاز والمسمى (ACTION_BOOT_COMPLETED) بحيث يتم إشعاره تلقائياً من قبل النظام عند حصول هذا الحدث، يجب إضافة الخاصية receiver داخل الخاصية application من ملف الوثيقة Manifest كما هو موضح:

```
<application
…
<receiver android:name="MyReceiver">
<intent-filter>
<action
android:name="android.intent.action.BOOT_COMPLETED">
</action>
</intent-filter>
</receiver>
…
</application>
```

عند اكتمال تشغيل الجهاز، سيتم تلقائياً اشعار مستقبل النشر MyReceiver ومن ثم تنفيذ الدالة ()onReceive.

لاحظ أن تسجيل مستقبل النشر يتضمن تحديد مرشح الهدف (Intent Filter) والذي يحدد نوع الرسائل التي يستمع لها مستقبل النشر. في المثال الموضح، تم إعداد مستقبل النشر للاستماع لحدث إعادة التشغيل المحدد بالقيمة "android.intent.action.BOOT_COMPLETED". عند إعادة التشغيل يقوم النظام تلقائياً ببث رسائل تفيد بحصول حدث التشغيل، حيث يتم استقبال هذه الرسائل من قبل مستقبلات النشر المسجلة لاستماع لهذا الحدث فقط.

ملاحظة: بعض الأحداث يتطلب الاستماع لها إضافة إذن (Permision) إلى ملف الوثيقة (Manifest). فمثلاً، للاستماع لحدث اكتمال تشغيل الجهاز لابد من إضافة الإذن التالي:

```
<uses-permission
android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
```

هناك عدد من الأحداث (Event) الصادرة من قبل نظام أندرويد ويمكن الاستماع لها من خلال تسجيل مستقبلات النشر. يوضح الجدول التالي بعض أحداث النظام وأسمائها الثابته.

جدول 1-8: بعض الأحداث (Events) الخاصة بنظام أندرويد والتي يمكن الاستماع لها باستخدام مستقبلات النشر (Broadcast Receivers)

اسم الحدث	وصف الحدث
android.intent.action.BOOT_COMPLETED	اكتمال إعادة تشغيل الجهاز
android.net.conn.CONNECTIVITY_CHANGE	تغيير في حالة الاتصال بالإنترنت
android.intent.action.ACTION_POWER_CONNECTED	توصيل الشاحن بالجهاز
android.intent.action.ACTION_POWER_DISCONNECTED	فك توصيل الشاحن بالجهاز
android.intent.action.CALL	الاتصال بجهة اتصال
android.intent.action.BATTERY_LOW	انخفاض سعة البطارية
android.intent.action.DATE_CHANGED	تغيير في تاريخ ووقت الجهاز

الجدير ذكره أنه يمكن تسجيل، وإلغاء تسجيل، مستقبل النشر برمجياً بدلاً من التعديل على ملف الوثيقة (Manifest). يتم ذلك من خلال الدوال ()registerReceiver و ()unregisterReceiver. يفضل عادة تسجيل مستقبل النشر من خلال ملف (Manifest).

تمرين عملي (1-8)

يهدف هذا التمرين إلى إنشاء وتسجيل مستقبل نشر (Broadcast Receiver) للاستماع لحدث من أحداث النظام ومن ثم تنفيذ إجراء محدد عند حصول هذا الحدث. الحدث الذي سيتم الاستماع له هو تغير حالة الاتصال بالشبكة اللاسلكية (WIFI)، حين يصبح الجهاز متصلاً بشبكة (WIFI) يقوم الجهاز بالاهتزاز (Vibrate) وعرض رسالة تبين أن الجهاز أصبح متصلاً.

الخطوة الأولي لتنفيذ هذا التمرين هي إنشاء الفئة (MyConnectivityReceiver) الخاصة بمستقبل النشر (Broadcast Receiver) وهي موضحة بالأسفل:

```
1:  public class MyConnectivityReceiver extends
2:  BroadcastReceiver{
3:
```

```
4:    @Override
5:    public void onReceive(Context context, Intent intent)
6:   {
7:         ConnectivityManager conMngr =
8:   (ConnectivityManager)
9:   context.getSystemService(Context.CONNECTIVITY_SERVICE);
10:        NetworkInfo wifi =
11:   conMngr.getNetworkInfo(ConnectivityManager.TYPE_WIFI);
12:        if(wifi.isConnected()){
13:             Vibrator vibrator = (Vibrator)
14:   context.getSystemService(Context.VIBRATOR_SERVICE);
15:             vibrator.vibrate(500);
16:             Toast.makeText(context, "Connected to WIFI",
17:   Toast.LENGTH_LONG).show();
18:        }
19:   }
20: }
```

لاحظ أنه في الدالة ()onReceive (أنظر الكود من سطر رقم 5 إلى 20)، والتي يتم تنفيذها تلقائياً عندما يستقبل مستقبل النشر (BroadcastReceiver) رسالة هدف (Intent) من النظام تفيد بتغير حالة الاتصال بالشبكة، يتم استخدام أحد خدمات نظام أندرويد وهي (Connectivity Service) لفحص حالة الشبكة (أنظر الأسطر من 7 إلى 9). إذا أصبح الجهاز متصلاً يتم استخدام خدمة أخرى من خدمات النظام وهي (Vibrator Service) لتقوم بعمل اهتزاز لنصف ثانية. أخيراً، يتم طباعة رسالة لتبين أن الجهاز أصبح متصلاً (أنظر الأسطر من 12 إلى 18).

الخطوة التالية هي تسجيل مستقبل النشر في ملف (Manifest) وتحديد الأحداث التي يستجيب لها. الشكل يوضح ملف الوثيقة (Manifest) الخاص بالتطبيق:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="ps.edu.ucas.broadcastreceiver"
    android:versionCode="1"
    android:versionName="1.0" >
     <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE" />
     <uses-permission
android:name="android.permission.ACCESS_WIFI_STATE" />
     <uses-permission
android:name="android.permission.VIBRATE" />
    <uses-sdk
        android:minSdkVersion="8"
```

```
        android:targetSdkVersion="21" />
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
…
        <receiver android:name=".MyConnectivityReceiver"
android:enabled="true">
            <intent-filter>
                <action
android:name="android.net.conn.CONNECTIVITY_CHANGE" />
            </intent-filter>
        </receiver>
….
    </application>
```

لاحظ الجزء الأخير الخاص بتسجيل مستقبل النشر (Broadcast Receiver) حيث تم تحديد حدث النظام android.net.conn.CONNECTIVITY_CHANGE للاستماع له. لاحظ أيضاً اضافة أذونات خاصة للاستماع لحالة الشبكة ولاستخدام الهزاز (Vibrator).

قم بتشغيل التطبيق وتجربته وذلك بتغيير حالة الشبكة (WIFI) من إعدادات الجهاز وملاحظة ما يحدث.

بث رسائل خاصة (Broadcasting Custom Intents)

يمكن تهيأت التطبيق الخاص بك لبث رسائل (Intent) إلى تطبيقات أخرى أو إلى مكونات أخرى في نفس التطبيق. في هذه الحالة يتم بث رسالة (Intent) عن طريق تنفيذ أحدى الدوال الخاصة بذلك مثل الدالة ()sendBroadcast. الرسالة المرسلة يتم استقبالها من قبل مستقبلات النشر (Broadcast Receivers) المعدة لاستقبال نفس نوع (Intent) والمسجلة من قبل تطبيقات أخرى أو في نفس التطبيق. فمثلاً، قد ترغب في إنشاء تواصل بين تطبيق ما وأحد التطبيقات الأخرى على الجهاز. في هذه الحالة ما عليك إلا إنشاء وتسجيل مستقبل بث (Broadcast Receiver) في البرنامج المستقبل، ثم تقوم بنشر رسالة (Intent) عن طريق البرنامج المرسل بحيث يوافق نوع الرسالة (Intent) النوع المحدد في مرشح الهدف (Intent Filter) لمستقبل النشر. يمكن استخدام هذه الطريقة للتواصل بين أجزاء التطبيق الواحد من الواجهة (Activity) وخدمة (Service) تعمل في الخلفية مثلاً. يوضح الكود التالي كيفية نشر رسالة عن طريق الدالة ()broadcastIntent تنفذ عن النقر على زر:

```
public void broadcastIntent(View view){
Intent intent =new Intent();
Intent.setAction("ps.edu.ucas.broadcastreceiver.CUSTOM_INTE
NT)");
sendBroadcast(intent);
```

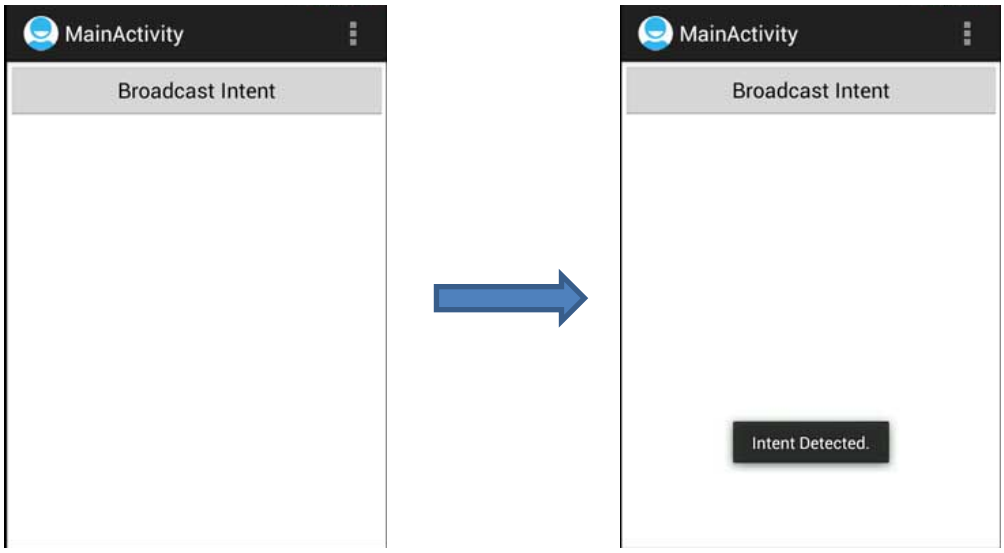
لاستقبال هذه الرسالة من تطبيق آخر، لا بد من تسجيل مستقبل نشر في التطبيق المستقبل بحيث يوافق مرشح الهدف (Intent Filter) نوع رسالة الهدف. الشكل يوضح تسجيل مستقبل النشر MyReceiver المنشأ في المثال السابق لاستقبال الرسائل المرسلة من التطبيق في المثال السابق

```
<application
…
<receiver android:name="MyReceiver">
  <intent-filter>
    <action
android:name="ps.edu.ucas.broadcastreceiver.CUSTOM_INTENT">
    </action>
  </intent-filter>
</receiver>
…
</application>
```

ملاحظة: لبث رسائل (Intent) يمكن استخدام الدالة broadcastIntent () أو broadcastStickyIntent (). في الدالة الثانية، الرسالة التي يتم نشرها تبقى في النظام لفترة، مما يمكن التطبيقات التي تسجل مستقبل النشر برمجياً من استقبال الرسالة عندما تصبح جاهزةً لذلك. أي أن الدالة الثانية تستهدف التطبيقات التي تستمع متأخراً للرسائل التي يتم نشرها.

تمرين عملي (8-2)

سنقوم بتنفيذ تطبيق بسيط نقوم فيه بإنشاء مستقبل نشر لاعتراض رسالة هدف مرسلة من أحد التطبيقات. واجهة التطبيق موضحة بشكل 8-1، حيث تتكون من زر واحد (Broadcast Intent)، عند النقر عليه يتم نشر رسالة بحدث (Action) محدد. عند استقبال الرسالة من قبل أحد مستقبلات النشر (Broadcast Receivers) يتم طباعة رسالة على الشاشة. إذا كان باستطاعتك تنفيذ استقبال الرسائل الخاصة باستخدام مستقبلات النشر، يمكنك بنفس الكيفية استقبال رسائل النظام.



شكل 1-8: واجهة التطبيق الخاص بتمرين 8-2[1]

كود الفعالية الخاص بالتطبيق موضح في الأسفل:

```
1:  public class MainActivity extends Activity{
2:
3:  @Override
4:  public cvoid onCreate(Bundle savedInstanceState){
5:  super.onCreate(savedInstanceState);
6:      setContentView(R.layout.activity_main);
7:  }
8:  // broadcast a custom (Intent).
9:  public void broadcastIntent(View view){
10:  Intent intent =new Intent();
11: intent.setAction("ps.edu.ucas.broadcastreceiver.CUSTOM_
12:  INTENT");
13:  sendBroadcast(intent);
14: }
15:}
```

لاحظ أنه عند النقر على زر "Broadcast Intent" يتم إنشاء هدف مضمن (Implicit Intent) ويتم تحديد حدث خاص باسم "ps.edu.ucas.broadcastreceiver.CUSTOM_INTENT" (يمكن اختيار أي اسم

[1] Tutorialspoint, http://www.tutorialspoint.com/android/android_broadcast_receivers.htm

للحدث بشرط أن لا يتكرر مع اسم حدث آخر). يتم نشر (Broadcast) هذا الهدف باستخدام الدالة setBroadcast() (أنظر الأسطر من 10 إلى 13)

نقوم بعد ذلك بانشاء مستقبل النشر (Broadcast Receiver) الذي من المفترض أن يستمع للحدث الخاص ويستجيب له بطباعة رسالة على الشاشة. الكود التالي يوضح الفئة (MyCustomActionReceiver) الخاصة بذلك:

```
1:  public class MyCustomActionReceiver extends
2:  BroadcastReceiver{
3:  @Override
4:  public void onReceive(Context context,Intent
5:  intent){
6:    Toast.makeText(context,"Intent
7:    Detected.",Toast.LENGTH_LONG).show();
8:  }
9:}
```

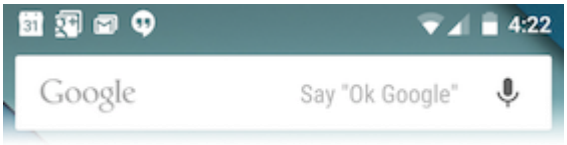
وفي الخطوة الأخيرة يتم تسجيل مستقبل النشر (Broadcast Receiver) في ملف الوثيقة (Manifest). لاحظ أن اسم الحدث (Action) المستخدم في مرشح الهدف (Intent Filter) يطابق اسم الحدث الذي تم ارساله مسبقاً ضمن الهدف (Intent) باستخدام الدالة setBroadcast().

```
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
package="ps.edu.ucas.customaction"
android:versionCode="1"
android:versionName="1.0">
<uses-sdk
android:minSdkVersion="8"
android:targetSdkVersion="15"/>
<application
android:icon="@drawable/ic_launcher"
android:label="@string/app_name"
android:theme="@style/AppTheme">
…
<receiver android:name="MyCustomActionReceiver">
  <intent-filter>
    <action
android:name="ps.edu.ucas.broadcastreceiver.CUSTOM_INTENT">
    </action>
  </intent-filter>
</receiver>
```

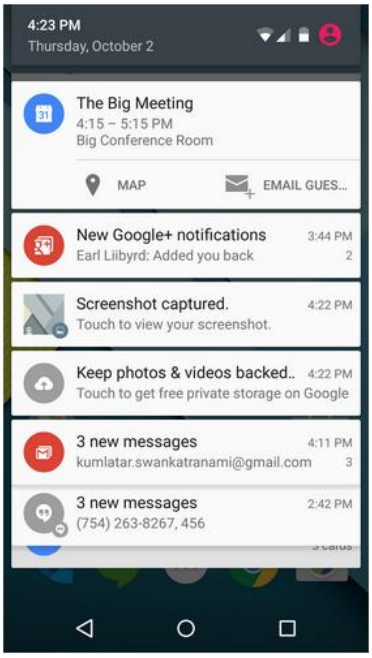
```
</application>
</manifest>
```

الإشعارات (Notifications)

الإشعار (Notification) هي رسالة يمكن إظهارها للمستخدم خارج إطار واجهة التطبيق، بمعنى أنه يمكن للنظام إرسال إشعارات (Notifications) للمستخدم دون الحاجة إلى تشغيل تطبيق معين. عندما يقوم النظام بارسال اشعار (Notification) للمستخدم، مثل إعلام المستخدم بوصول رسالة ما SMS أو اكتمال تحميل ملف من شبكة الإنترنت، يتم إظهار الإشعار (Notification) كأيقونة في منطقة الإشعارات (Notification Area) (أنظر شكل 2-8)، وعند النقر على أيقونة الإشعار يتم إظهار التفاصيل في درج الإشعارات (Notification Drawer) (أنظر شكل 3-8).



شكل 2-8: منطقة الأشعارات (Notification Area)



شكل 3-8: درج الأشعارات (Notification Drawer)

إنشاء الإشعار (Notification)

لإنشاء إشعار (Notification) يجب في البداية إنشاء كائن من نوع Notification.Builder ومن خلاله يتم تحديد خصائص الإشعار وواجهته والإجراء الذي ينفذ عند النقر على الإشعار. بعد تحديد هذه الخصائص، يتم إنشاء الكائن الخاص بالإشعار Notification عن طريق تنفيذ الدالة ()Notification.Builder.build. لإرسال الإشعار، يتم تنفيذ الدالة ()NotificationManager.notify.

الإشعار (Notification) يجب أن يشتمل على الأقل على ما يلي، والتي يتم تحديدها من خلال الكائن Notification.Builder:

- أيقونة صغيرة small icon يتم تحديدها عن طريق الدالة ()setSmallIcon.
- عنوان Title يتم تحديده عن طريق الدالة ()setContentTitle.
- التفاصيل detail text، يتم تحديده عن طريق الدالة ()setContentText.

كل الإعدادات والخصائص الأخرى للإشعار (Notification) هي إختيارية ويمكن التعرف على تفاصيلها بالاطلاع على تفاصيل الفئة Notification.Builder. المثال التالي يوضح طريقة إنشاء إشعار (Notification) وإرساله:

```
1:  Notification.Builder builder = new
2:  Notification.Builder(getApplicationContext());
3:  builder.setContentTitle("News from UCAS");
4:  builder.setContentText("Graduation ceremonies started
5:  for UCAS");
6:  builder.setSmallIcon(R.drawable.ic_launcher);
7:  Notification notification = builder.build();
8:  NotificationManager manager =
9:  (NotificationManager)getSystemService(
10:  Context.NOTIFICATION_SERVICE);
11:  manager.notify(id, notification);
```

لاحظ أنه في البداية يتم إنشاء كائن من نوع (Notification.Builder) (أنظر سطر رقم 1 و2) ومن ثم تحديد الخصائص الأساسية للإشعار بتنفيذ الدوال ()setContentTitle، ()setContentText، ()setSmallIcon (أنظر الأسطر من 3 إلى 6).

بعد تحديد خصائص الإشعار (Notification) من خلال Notification.Builder، يتم إنشاء كائن من نوع Notification بتنفيذ الدالة ()Notification.Builder.build (سطر رقم 7).

لإرسال الإشعار (Notification)، يجب في البداية الوصول للكائن (NotificationManager) وهو أحد خدمات نظام أندرويد المتوفرة في الخلفية (الأسطر من 8 إلى 10)، ثم يتم إرسال الإشعار بتنفيذ الدالة

NotificationManager.notify() (سطر رقم 11). لاحظ أن الدالة notify() يمرر لها بالإضافة للكائن من نوع (Notification) رقم معرف id يتم اختياره من قبل المستخدم. يمكن من خلال هذا المعرف الوصول للإشعار لاحقاً لتعديله أوحذفه.

الإجراء المرتبط بالإشعار (Notification Action)

على الرغم من كونه اختيارياً، يمكن إضافة إجراء واحد أو أكثر للإشعار (Notification). فمثلاً، عند النقر على الإشعار المستلم، يمكن تشغيل فعالية (Activity) لإظهار تفاصيل إضافية عن الإشعار.

داخل الكائن من نوع (Notification)، يتم تعريف الإجراء (Action) المراد تنفيذه عن طريق تعريف كائن من نوع (Pending Intent) والذي يحتوي داخله على هدف (Intent) يحدد الإجراء المراد تنفيذه. بعد ذلك يتم تمرير الكائن من نوع الهدف المعلق (PendingIntent) عند طريق الدالة setContentIntent() من خلال Notification.Builder.

المثال التالي يستكمل المثال السابق وذلك بإضافة حدث (Action) للإشعار حيث يتم تشغيل الفعالية (Activity) الخاصة بمتصفح الإنترنت لفتح صفحة الكلية الجامعية:

```
1:  Notification.Builder builder = new
2:  Notification.Builder(getApplicationContext());
3:  builder.setContentTitle("News from UCAS");
4:  builder.setContentText("Graduation ceremonies started
5:  for UCAS");
6:  builder.setSmallIcon(R.drawable.ic_launcher);
7:  builder.setContentInfo("Click for more info");
8:  Intent intent = new Intent(Intent.ACTION_VIEW);
9:  intent.setData(Uri.parse("http://www.ucas.edu.ps"));
10:  PendingIntent pIntent =
11:  PendingIntent.getActivity(getApplicationContext(), 0,
12:  intent,0);
13:  builder.setContentIntent(pIntent);
14:  Notification notification = builder.build();
15:  NotificationManager manager =
16:  (NotificationManager)getSystemService(
17:  Context.NOTIFICATION_SERVICE);
18:  manager.notify(id, notification);
```

لإنشاء حدث (Action) وربطه بالإشعار (Notification) ،يتم إنشاء كائن من نوع (Intent) لتحديد الحدث (Action) المراد تنفيذه وهو في المثال الموضح من نوع (ACTION_VIEW) (أنظر سطر رقم 8) ويتم أيضاً تحديد عنوان موقع الويب المراد فتحه من خلال الدالة setData() (أنظر سطر رقم 9).

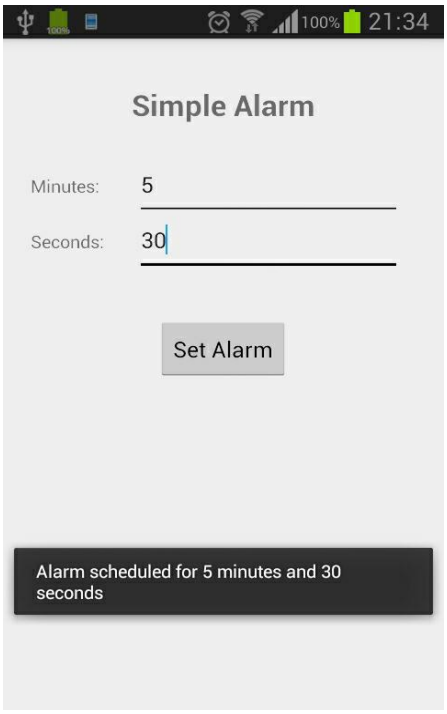
بعد ذلك يتم إنشاء هدف معلق (PendingIntent) والذي يمرر له كائن الهدف (Intent) الخاص بالإجراء المراد تنفيذه (أنظر الأسطر من 10 إلى 12). بعد ذلك يتم تمرير الكائن من نوع (PendingIntent) للكائن (Notification.Builder) من خلال الدالة setContentIntent() (أنظر سطر رقم 13)

يمكن فهم الهدف المعلق (PendingIntent) على أنه رسالة موجهة للنظام لنطلب منه تشغيل إجراء محدد عند النقر على الإشعار Notification عند وصوله. الإجراء الذي يتم تنفيذه محدد عن طريق الكائن Intent، الذي ممرناه لكائن الهدف المعلق (PendingIntent) أثناء إنشائه. أي أن الإجراء المحدد في الهدف المعلق (PendingIntent) ينفذ من قبل النظام ويكون معلقاً لحين النقر على الإشعار (Notification) المستلم.

تمرين عملي (3-8)

في هذا التمرين سنقوم ببناء تطبيق منبه بسيط يقوم بتنبيه المستخدم بعد وقت محدد ليقوم بتنفيذ بمتابعة آخر الأخبار على موقع الكلية الجامعية للعلوم التطبيقية. التنبيه يتم بإرسال إشعار (Notification) عندما يحين الوقت المحدد، وعند النقر على هذا الإشعار يتم فتح الموقع الإلكتروني للكلية الجامعية.

واجهة التطبيق موضحة بشكل 4-8 حيث يقوم المستخدم بتحديد وقت الانتظار والذي بعد انقضائه يتم إرسال الإشعار (Notification). كما هو موضح يقوم المستخدم بإدخال وقت الانتظار بالدقائق والثواني، ثم يضغط على زر "Set Alart" وذلك لتفعيل خدمة التنبيه.



شكل 4-8: واجهة التطبيق الخاص بتمرين 3-8

لبناء هذا التطبيق، نبدأ أولاً بعرض ملف التصميم الخاص بالفعالية (MainActivity):

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/RelativeLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/tvAlarmTitle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="35dp"
        android:textSize="24sp"
        android:textStyle="bold"
        android:text="Simple Alarm" />

    <TextView
        android:id="@+id/tvMinutes"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_marginRight="26dp"
        android:layout_marginTop="102dp"
        android:layout_marginLeft="20dp"
        android:text="Minutes:" />

    <TextView
        android:id="@+id/tvSeconds"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/tvMinutes"
        android:layout_below="@+id/tvMinutes"
        android:layout_marginTop="23dp"
        android:text="Seconds:" />

    <EditText
```

```
        android:id="@+id/edMinutes"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBaseline="@+id/tvMinutes"
        android:layout_alignBottom="@+id/tvMinutes"
        android:layout_toRightOf="@+id/tvMinutes"
android:inputType="number"
        android:ems="10" >

        <requestFocus />
    </EditText>

    <EditText
        android:id="@+id/edSeconds"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBaseline="@+id/tvSeconds"
        android:layout_alignBottom="@+id/tvSeconds"
        android:layout_alignLeft="@+id/edMinutes"
android:inputType="number"
        android:ems="10" />

    <Button
        android:id="@+id/setAlarmBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/edSeconds"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="48dp"
        android:text="Set Alarm" />

</RelativeLayout>
```

ملف الكود الخاص بالفعالية (MainActivity) موضح بالأسفل:

```
1:  public class MainActivity extends Activity {
2:
3:   @Override
4:   protected void onCreate(Bundle savedInstanceState) {
5:        super.onCreate(savedInstanceState);
6:        setContentView(R.layout.activity_main);
```

```
7:        final EditText minutesTxt = (EditText)
8:   this.findViewById(R.id.edMinutes);
9:        final EditText secondsTxt = (EditText)
10:  this.findViewById(R.id.edSeconds);
11:       Button setAlarmBtn = (Button)
12:  this.findViewById(R.id.setAlarmBtn);
13:       setAlarmBtn.setOnClickListener(new
14:  OnClickListener(){
15:
16:            @Override
17:            public void onClick(View v) {
18:                 int minutes =
19:  Integer.parseInt(minutesTxt.getText().toString());
20:                 int seconds =
21:  Integer.parseInt(secondsTxt.getText().toString());
22:                 scheduleAlarm(minutes, seconds);
23:            }
24:       });
25:  }
26:
27:  public void scheduleAlarm(int minutes,int seconds){
28:            long currentTime = new
29:  GregorianCalendar().getTimeInMillis();
30:            long scheduleTime = currentTime +
31:  minutes*60*1000 + seconds * 1000;
32:            Intent intentAlarm = new
33:  Intent("ps.edu.ucas.alarmnotify.ALARM_ACTION");
34:            AlarmManager alarmManager = (AlarmManager)
35:  getSystemService(Context.ALARM_SERVICE);
36:
37:  alarmManager.set(AlarmManager.RTC_WAKEUP,scheduleTime,
38:  PendingIntent.getBroadcast(this,1,  intentAlarm,
39:  PendingIntent.FLAG_UPDATE_CURRENT));
40:            Toast.makeText(this, "Alarm scheduled for "+
41:  minutes +" minutes and "+ seconds +" seconds" ,
42:  Toast.LENGTH_LONG).show();
43:     }
44: }
```

لاحظ أنه عند النقر على زر "Set Alarm" يتم قراءة القيم المدخلة في عناصر الإدخال وتحويلها إلى قيم رقمية كما هو موضح:

```
int minutes =
Integer.parseInt(minutesTxt.getText().toString());
int seconds =
Integer.parseInt(secondsTxt.getText().toString());
```

يتم بعد ذلك تنفيذ الدالة scheduleAlarm() والتي يمرر لها قيمة الدقائق والثواني. في هذه الدالة يتم في البداية تحديد وقت التنبيه ويساوي الوقت الحالي مضافاً إليه مجموع الدقائق والثواني الدمخلة، وهو ما يتم من خلال هذين السطرين:

```
Long currentTime=new GregorianCalendar().getTimeInMillis();
long scheduleTime = currentTime + minutes*60*1000 + seconds
* 1000;
```

لتفعيل خدمة التنبيه يتم استخدام أحد الخدمات المتوفرة في نظام أندرويد وهي خدمة مدير التنبيه (Alarm Manager) والذي يمكن إعداده ليقوم بنشر هدف معين (broadcast Intent) عندما يحين وقت التنبيه. لتهيئة مدير التنبيه يتم إنشاء هدف مضمن (Implicit Intent) باسم (intentAlarm) وذلك ليتم نشره تلقائياً من قبل مدير التنبيه عندما يحين الوقت المحدد كالتالي:

```
Intent intentAlarm = new
Intent("ps.edu.ucas.alarmnotify.ALARM_ACTION");
```

اسم الحدث (Action) المستخدم هو اسم افتراضي خاص بالتطبيق. لاحقاً سيتم بناء مستقبل نشر (Broadcast Receiver) ليقوم باستقبال الهدف (Intent) المرسل بنفس الهدف المحدد.

يتم بعد ذلك إعداد مدير التنبيه (Alarm Manager) بإنشاء هدف معلق (PendingIntent) وذلك لنطلب منه نشر الهدف (intentAlarm) عندما يحين الوقت المحدد وذلك من خلال الكود التالي:

```
alarmManager.set(AlarmManager.RTC_WAKEUP,scheduleTime,
PendingIntent.getBroadcast(this,1,  intentAlarm,
PendingIntent.FLAG_UPDATE_CURRENT));
```

الخطوة التالية تتضمن إنشاء مستقبل نشر (Broadcast Receiver) ليقوم تلقائياً باستقبال الهدف المرسل من مدير التنبية ومن ثم إرسال الإشعار (Notification) إلى المستخدم. الكود التالي خاص بمستقبل نشر باسم MyAlarmReceiver:

```
1:  public class MyAlarmReceiver extends BroadcastReceiver{
2:
3:   public static final int NOTIFICAION_ID = 1;
4:
5:   @Override
6:   public void onReceive(Context context, Intent intent)
7:   {
8:        Notification.Builder builder = new
```

```
9:  Notification.Builder(context);
10:       builder.setContentTitle("Alarm sent off");
11:       builder.setContentText("Time to check latest news
12:  from UCAS");
13:       builder.setSmallIcon(R.drawable.ic_launcher);
14:       builder.setContentInfo("Click for more info");
15:       Intent ucasIntent = new
16:  Intent(Intent.ACTION_VIEW);
17:       ucasIntent.setData(Uri.parse("http://www.ucas
18:  .edu.ps"));
19:       PendingIntent pIntent =
20:  PendingIntent.getActivity(context, 0, ucasIntent,0);
21:       builder.setContentIntent(pIntent);
22:       Notification notification = builder.build();
23:       NotificationManager manager =
24:  (NotificationManager) context.getSystemService(
25:  Context.NOTIFICATION_SERVICE);
26:       manager.notify(NOTIFICAION_ID, notification);
27:  }
28: }
```

وأخيراً، لا ننسى تسجيل مستقبل النشر (MyAlarmReceiver) في ملف الوثيقة (Manifest) كما هو موضح:

```
<receiver android:name=".MyAlarmReceiver"
android:enabled="true">
  <intent-filter>
     <action
android:name="ps.edu.ucas.alarmnotify.ALARM_ACTION" />
     </intent-filter>
</receiver>
```

لاحظ أن اسم الحدث (Action) المحدد في مرشح الهدف (Intent Filter) يطابق اسم الحدث الخاص بالهدف (Intent) الذي يرسله مدير التنبيه (Alarm Manager).

قم بتشغيل التطبيق وتجربته بإدخال وقت الانتظار ثم انتظار وصول الإشعار (Notification).

أسئلة على الوحدة الثامنة

1. ما فائدة مستقبل النشر (Broadcast Receiver)؟ وما الفرق بينه وبين الفعالية (Activity)؟
2. قم ببناء تطبيق يقوم تلقائياً بإرسال إشعار (Notification) للمستخدم عند تغيير الوقت من إعدادات الهاتف.
3. قم بالتعديل على تمرين 2-6 الخاص بقواعد البيانات وذلك بنشر رسالة (sendBroadcasr) عند إضافة أو حذف بيانات أي طالب. قم بعد ذلك بإنشاء مستقبل نشر (Broadcast Receiver) لاستقبال هذه الرسائل وإرسال إشعار (Notification) للمستخدم للإعلامه بالإجراء الذي تم تنفيذه على قاعدة بيانات الطلاب.
4. ما الفرق بين الرسالة التي يتم طباعتها باستخدام Toast والإشعار Notification؟
5. ما الفرق بين Intent و PendingIntent؟

الوحدة التاسعة:

تصميم الوجهات باستخدام القطع

(User Interface Design by Using Fragments)

يتعلم الطالب في هذه الوحدة:

- ✓ مفهوم القطع (Fragments) واستخداماتها.
- ✓ استخدام القطع (Fragments) لإنشاء تطبيقات موائمة تتكيف مع حجوم الأجهزة المختلفة وأوضاع العرض.
- ✓ دورة حياة القطعة (Fragment) وعلاقتها بالفعالية المضيفة (Activity).
- ✓ طريقة إنشاء القطع (Fragments) وربطها بالفعاليات (Activities).
- ✓ التواصل وتبادل البيانات بين القطع (Fragments).
- ✓ إنشاء مربعات الحوار (Dialogs) المختلفة وتخصيصها.

القطعة (Fragment) واستخداماتها

القطعة (Fragment) تمثل جزء من واجهة المستخدم المعروضة ضمن الفعالية (Activity)، حيث يمكن تجميع عدة قطع (Fragments) لتكوين واجهة واحدة تعرض في الفعالية. بالإضافة إلى ذلك، يمكن استخدام القطعة (Fragment) في أكثر من واجهة.

يمكن استخدام القطع (Fragments) وإعادة هيكلتها للتلاءم من أحجام الشاشات المختلفة .فمثلاً، شاشات الهواتف الصغيرة لا تتسع لعرض الكثير من المكونات، لذلك يتم عرض مكون واحد، على هيئة قطعة واحدة Fragment، داخل واجهة الفعالية (Activity)، حيث من الممكن أن يشتمل التطبيق على عدة فعاليات (Activities) تعرض كل منها قطعة واحدة (Fragment) نظراً لعدم إمكانية عرض أكثر من قطعة لصغر مساحة الشاشة. أما في حالة الأجهزة اللوحية، فمن الممكن تجميع أكثر من قطعة (Fragment) في نفس الواجهة التي تتسع لعرض عدد أكبر من المكونات. في كلا الحالتين، يتم إعادة استخدام نفس القطع ولكن بترتيب وهيكلية مختلفة. القدرة على إعادة استخدام قطع الواجهة Fragments توفر الجهد والوقت اللازم في حالة إنشاء واجهة جديدة لكل جهاز مختلف.

كمثال عملي على استخدام القطع Fragments في تصميم الواجهات للأجهزة المختلفة، افترض ان هناك تطبيق يعرض قائمة من العناصر، وعند اختيار أي عنصر يتم عرض تفاصيله (مشابه لفكرة تطبيق إخباري). واجهة هذا البرنامج يمكن تقسيمها إلى قطعتين: قطعة 1 خاصة بعرض القائمة الرئيسية، وقطعة 2 لعرض تفاصيل أي عنصر يتم اختياره من القائمة. في حالة الجهاز اللوحي، يمكن عرض كلا القطعتين في نفس الواجهة حيث تتسع المساحة لذلك. أما في حالة الهاتف ذي الشاشة الأصغر، فيتم عرض كل قطعة (Fragment) في فعالية (Activity): حيث تحتوي الفعالية الأولى على قائمة العناصر (أنظر شكل 1-9). وعند النقر على أي عنصر في القائمة يتم عرض التفاصيل في فعالية جديدة تحل محل القائمة. لاحظ أن نفس القطع (Fragments) تم استخدامها ولكن بترتيب مختلف في كل حالة.

يمكن اعتبار أن القطعة (Fragment) هي وحدة مستقلة بذاتها ولها دورة حياتها ويمكن إضافتها إلى أو حذفها من الفعالية(Activity) أثناء تنفيذ التطبيق. القطعة (Fragment) يجب أن تكون مضمنة داخل فعالية (Activity)، أي أنه لا يمكن عرضها بدون (Activity). كذلك دورة حياة القطعة (Fragment) تتأثر بشكل مباشر بدورة حياة الفعالية (Activity). عن إيقاف أو تدمير الفعالية يتم إيقاف أو تدمير القطع المضمنة داخلها.

شكل 1-9: استخدام القطع (Fragments) لمواءمة التطبيق مع حجوم الشاشات المختلفة [1]

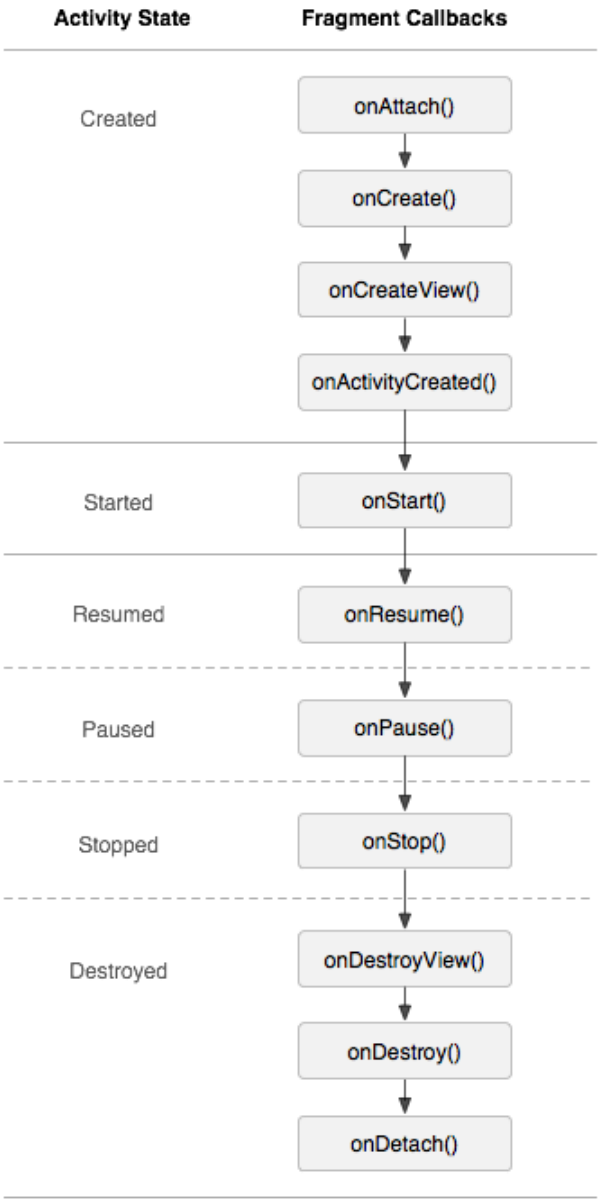
استخدام القطع (Fragments) لا يفيد فقط في مواءمة واجهة المستخدم مع الأجهزة ذات المساحات المختلفة، ولكن أيضا في مواءمة الواجهة مع اتجاه العرض لنفس الجهاز. فمثلاً يمكن تركيب الواجهة في الوضع الرأسي Portrait بشكل مختلف عن تركيبها في الوضع الأفقي Landscape، حيث يتم استخدام نفس القطع ولكن بتركيب مختلف.

إنشاء القطع (Fragments)

لإنشاء قطعة (Fragment) يجب إنشاء فئة جديدة متفرعة (subclass) من الفئة Fragment أو أي فئة أخرى متفرعة منها مثل ListFragment، DiaglogFragment وغيرها. الفئة الجديدة (class) الخاصة بالقطعة Fragment تحتوي على عدد من دوال الاتصال الراجع (Callback Methods) شبيهة بتلك الموجودة في الفعالية (Activity) مثل onCreate()، onStart()، onPause()، onStop() وغيرها من الدوال. عادةً، يتم كتابة بعض هذه الدوال لتحديد الإجراء المراد تنفيذه في كل مرحلة من دورة حياة القطعة (Fragment). شكل 2-9

[1] Tutorial on Fragments by Lars Vogel, http://www.vogella.com/tutorials/AndroidFragments/article.html

يوضح الدوال التي يتم تنفيذها خلال دورة حياة القطعة (Fragment)، وحالة الفعالية المضيفة. ونوضح فيما يلي أهم الدوال التي يحتاج المطور إلى كتابتها لتنفيذ بعض الإجراءات.



شكل 2-9: دورة حياة القطعة (Fragment Life Cycle) وعلاقتها بالفعالية[1]

[1] Fragments, Android Developer, http://developer.android.com/guide/components/fragments.html

- onCreate(): يتم تنفيذ هذه الدالة تلقائياً من قبل النظام عن إنشاء القطعة (Fragment) وقبل عرضها. يتم عادةً استخدام هذه الدالة لاسترجاع البيانات التي تم حفظها عند توقف القطعة (Fragment). يمرر إلى هذه الدالة حزمة (Bundle) والتي يتم منها استرجاع البيانات. لاحظ أنه في حالة استخدام القطع (Fragments) لبناء الواجهة، فإن كل قطعة مسؤولة عن حفظ واسترجاع البيانات الخاصة بها. يجب أيضاً الانتباه أن دورة حياة القطعة مرتبطة بحياة الفعالية: بمعنى توقف الفعالية يترتب عليه توقف القطعة مثلاً.

- onCreateView(): يتم تنفيذ الدالة تلقائياً عندما يحين وقت إنشاء الواجهة الخاصة بالقطعة. عادةً، كل قطعة يكون لها ملف تصميم خاص (Layout.xml)، ومن خلال الدالة onCreateView() يجب إنشاء واجهة القطعة عن طريق تنفيذ الدالة Inflater.inflate()، حيث يتم تحويل ملف الواجهة إلى كائن من نوع View يحوي كل عناصر الواجهة الخاصة بالقطعة. الكود التالي يوضح مثال لتنفيذ الدالة onCreateView() حيث يتم إنشاء واجهة القطعة والوصول برمجياً لعناصر الواجهة الموجودة داخله.

```
public View onCreateView(LayoutInflater inflater,
ViewGroup container, Bundle savedInstanceState) {
View                          view                          =
inflater.inflate(R.layout.fragment_rsslist_overview,
        container, false);
    Button            button            =            (Button)
view.findViewById(R.id.button1);
    button.setOnClickListener(new View.OnClickListener() {
@Override
publicvoid onClick(View v) {
     // Do something
     }
    });
return view;
  }
```

لاحظ أنه يتم تمرير متغير من نوع ViewGroup للدالة inflate(). هذا المتغير يمثل الحاوية container الخاصة بالفعالية المضيفة (Activity) والتي سيتم فيها تجميع كل القطع (Fragments) لتكوين واجهة النهائية للفعالية. الدالة inflate() تقوم بإنشاء واجهة القطعة (Fragment) للتتلاءم مع إعدادات الحاوية container.

لاحظ أن الدالة inflate() تقوم بإعادة الكائن View والذي يمثل الواجهة الخاصة بالقطعة كاملةً. بعد ذلك يمكن البحث عن أي عنصر داخل الكائن View لعمل إجراء معين. فمثلاً، يوضح الكود السابق كيف تم الوصول للزر الذي اسمه button1 الموجود داخل واجهة القطعة Fragment باستخدام الدالة findViewById() ومن ثم إضافة مستمع (Listener) لتنفيذ إجراء ما عن النقر على الزر.

يجب الانتباه أنه من خلال الدالة onCreateView() لا يمكن الوصول والتعامل مع الفعالية (Activity) حيث أن إنشاءها لم يكتمل بعد.

- onActivityCreated(): يتم تنفيذ هذه الدالة بعد onCreateView() وذلك عندما يكتمل إنشاء الفعالية (Activity). تكمن أهمية هذه أنه يمكن استخدامها لتنفيذ إي إجراءات تحتاج للكائن context، أو الفعالية (Activity). على سبيل المثال، إذا أردت تنفيذ الدالة findViewById() للوصول لأي عنصر واجهة في الفعالية (Activity)، طباعة رسالة Toast أو إنشاء Adapter خاض بقائمة العرض (ListView)، يجب تنفيذ كل ذلك باستخدام الكائن context أو الكائن acitivty. عند تنفيذ هذه الدالة، يمكن الوصول للفعالية من خلال الدالة getActivity()، ومن ثم يمكن تنفيذ كل الإجراءات السابقة.الكود التالي يوضح تطبيق الدالة onActivityCreated() حيث يتم فيها تعبئة قائمة (ListView) باستخدام المحول ArrayAdapter.

```
public void onActivityCreated(Bundle savedInstanceState) {
super.onActivityCreated(savedInstanceState);
String[] values = newString[] { "Android", "iPhone",
"WindowsMobile", "Blackberry", "WebOS", "Ubuntu",
"Windows7", "Max OS X", "Linux", "OS/2" };
ArrayAdapter<String> adapter = new
ArrayAdapter<String>(getActivity(),
        android.R.layout.simple_list_item_1, values);
    setListAdapter(adapter);
}
```

- onPause(): يتم تنفيذ هذه الدالة عند الإيقاف المؤقت للقطعة (Fragment) حيث تكون القطعة مرئية ولكن تم حجبها جزئياً من قبل فعالية (Activity) أخرى. تستخدم هذه الدالة في حفظ حالة القطعة أو بيانات أخرى قبل إيقاف أو إغلاق القطعة. كما ذكرنا مسبقاً في حالة الفعالية، يفضل تنفيذ كود حفظ الحالة في هذه الدالة بدلاً عن الدالة onStop()، حيث أن الدالة الأخيرة قد لا يتم تنفيذها من قبل النظام في بعد الأحيان، أما الدالة onPause() فيتم تنفيذها دائماً من عند ايقاف أو تدمير القطعة (Fragment).

- onAttach(): يتم تنفيذ هذه الدالة عند ربط القطعة (Fragment) بالفعالية (Activity). من المهم الملاحظة أنه عند تنفيذ هذه الدالة لايكون إنشاء عناصر الواجهة الخاصة القطعة والفعالية قد اكتمل وبذلك لا يمكن الوصول إلى محتواهما من عناصر الواجهة. يتم استخدام هذه الدالة للحصول على مؤشر لكائن الفعالية (Activity)، وهو ما يلزم في حالة التواصل بين القطعة و الفعالية. سيتم الحديث لاحقاً عن استخدام هذه للتواصل بين القطع (Fragments).

إضافة القطع إلى الفعالية

في الخطوات السابقة تم شرح كيفية إنشاء الفعالية (Fragment) والدوال التي نحتاج لتطبيقها عن الإنشاء أو أثناء عمل القطعة. الخطوة التالية هو إضافة القطعة داخل الواجهة الخاصة بالفعالية (Activity) والذي يتم بإحدى طريقتين:

1. إضافة القطع إلى الفعالية بطريقة ثابتة: يتم ذلك بإضافة خاصية من نوع fragment داخل ملف layout الخاص بالفعالية. يتم إضافة القطعة fragment تماماً كما يتم إضافة أي عنصر واجهة آخر كما هو موضح بالمثال التالي. لاحظ أن الخاصية android:name تشير إلى المسار الكامل للفئة (class) الخاصة بالقطعة

المضافة. لاحظ أيضاً أنه تم إعطاء كل قطعة (Fragment) رقم معرف من خلال الخاصية android:id، مما يتيح الوصول للقطعة برمجياً فيما بعد.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayoutxmlns:android="http://schemas.android.com/apk/
res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:baselineAligned="false"
android:orientation="horizontal">

<fragment
android:id="@+id/listFragment"
android:layout_width="0dp"
android:layout_weight="1"
android:layout_height="match_parent"
android:name="ps.edu.ucas.fragmentApp.MyListFragment"></fra
gment>
<fragment
android:id="@+id/detailFragment"
android:layout_width="0dp"
android:layout_weight="2"
android:layout_height="match_parent"
android:name=" ps.edu.ucas.fragmentApp.MyDetailFragment">
</fragment>

</LinearLayout>
```

2. إضافة القطع برمجياً أثناء التشغيل: إذا اردت التغيير في القطع Fragments أثناء تشغيل البرنامج، يمكن استخدام الدوال التي يوفرها Fragment Manager والذي يمكن الوصول إليه من خلال الفعالية (Activity) بتنفيذ الدالة ()getFragmentManager. يوفر Fragment Manager دوال لإضافة وحذف واستبدال القطع(Fragments).

قبل تنفيذ ذلك برمجياً، لا بد من تحديد مكان أو قالب فارغ في واجهة الفعالية ليتم إضافة القطعة (Fragment) إليه أو حذفه منها. المثال بالأسفل يبين ملف الواجهة Layout الخاص بالفعالية حيث تم إضافة عنصرين FrameLayout فارغين، وكل منهما له معرف android:id. سيتم لاحقاً تعديل محتوى هذين العنصرين بإضافة وحذف قطع (Fragments) وذلك باستخدام FragmentManager.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayoutxmlns:android="http://schemas.android.com/apk/
res/android"
```

```
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="horizontal">

<FrameLayout
android:id="@+id/listcontainer"
android:layout_width="match_parent"
android:layout_height="match_parent" />

<FrameLayout
android:id="@+id/detailscontainer"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:visibility="gone" />

</LinearLayout>
```

بعد ذلك يتم تنفيذ الكود التالي من داخل الفعالية (من داخل الدالة onCreate() مثلاً) والذي يوضح كيفية إضافة قطع (Fragments) داخل القوالب من نوع FrameLayout. لاحظ مثلاً أن الدالة FragmentTransaction.add() يمرر لها المعرف الخاص بالقالب (R.id.listcontainer) بالإضافة إلى الكائن الخاص بالقطعة (MyListFragment).

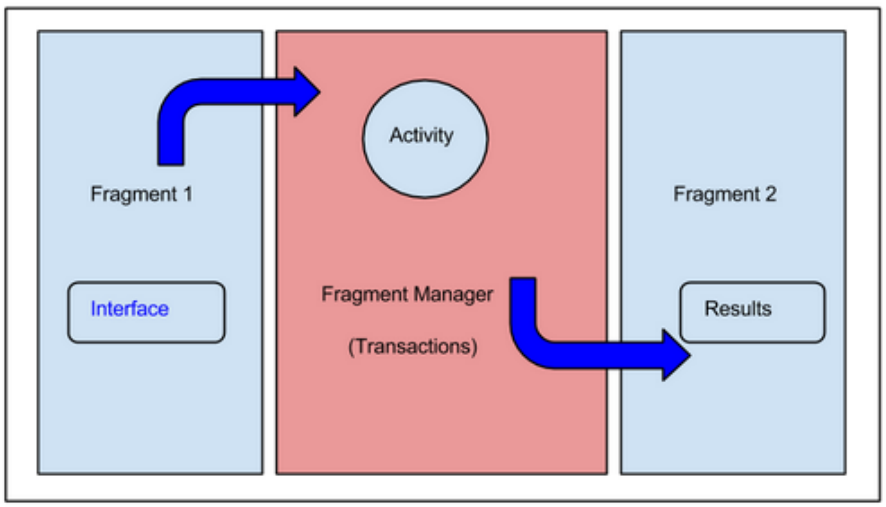
```
FragmentManager fm = getFragmentManager();

// الإضافة
FragmentTransaction ft = fm.beginTransaction();
ft.add(R.id.listcontainer, new MyListFragment());
ft.commit();

// استبدال
FragmentTransaction ft = fm.beginTransaction)();
ft.replace(R.id.listcontainer, new MyListFragment());
ft.commit();

// حذف
Fragment fragment =
fm.findFragmentById(R.id.listcontainer);
FragmentTransaction ft = fm.beginTransaction();
ft.remove(fragment);
ft.commit();
```

يمكن أيضاً أن نفحص برمجياً ما إذا كان الفعالية (Activity) تحتوي على قطعة معينة (Fragment) أم لا أثناء عمل التطبيق. يتم ذلك بالوصول للقطعة Fragment من خلال الدالة findFragmentById() ضمن مدير القطع Fragment Manager. ومن ثم ننفذ الدالة isInLayout() والتي تتحقق ما إذا كانت القطعة Fragment معروضة أم لا. الكود التالي يوضح هذه العملية:

```
DetailFragment fragment = (DetailFragment)
getFragmentManager().findFragmentById(R.id.detailscontainer
);
if (fragment==null || ! fragment.isInLayout()) {
// Do something
}
```

التواصل بين القطع (Fragments)

قد تحتاج القطع (Fragments) للتواصل فيما بينها لتبادل البيانات أو الأوامر. على سبيل المثال، عند اختيار خبر موجود في قائمة الأخبار ضمن قطعة ما، يتم عرض تفاصيل الخبر في قطعة أخرى. لتحقيق التواصل بين القطع، يفضل أن لا يتم التواصل بين القطع (Fragments) مباشرةً بل عن طريق الفعالية المضيفة (Activity). أي أن القطعة تقوم بتمرير أمر ما أو بيانات إلى الفعالية المضيفة (Activity)، ومن ثم تقوم الفعالية بتمرير ما وصل إليها إلى القطعة الهدف. شكل يوضح كيفية التواصل بين قطعتين (Fragments) من خلال الفعالية المضيفة (Activity) حيث تستقبل الفعالية أمر أو بيانات من القطعة الأولى، وتقوم من خلال ما يسمى بمدير القطع (Fragment Manager) بالوصول إلى القطعة الثانية. مدير القطع (Fragment Manager) يمكن عن طريقه الوصول لأي قطعة (Fragment)، كما يستخدم لإضافة أو حذف القطع برمجياً أثناء عمل التطبيق.



شكل 3-9: التواصل بين القطع (Fragments) من خلال الفعالية المضيفة (Activity)[1]

[1] Simple Developer, http://simpledeveloper.com/how-to-communicate-between-fragments-and-activities

الكود التالي يوضح كيفية تهيئة القطعة (Fragment) للتواصل مع الفعالية المضيفة:

```
1:  public class MyFragment extends Fragment {
2:
3:  private MyFragmentListener listener;
4:
5:  @Override
6:  public View onCreateView(LayoutInflater inflater,
7:  ViewGroup container,
8:      Bundle savedInstanceState) {
9:    View view =
10:  inflater.inflate(R.layout.fragment_layout,
11:        container, false);
12:  Button button = (Button)
13:  view.findViewById(R.id.button1);
14:    button.setOnClickListener(new View.OnClickListener()
15:  {
16:  @Override
17:  public void onClick(View v) {
18:  // Inform the Listener
19:  listener.buttonClicked();
20:      }
21:    });
22:   return view;
23:  }
24:  //Any class that needs to listen to the fragment
25:   //changes should implement this listener
26:   public interface MyFragmentListener {
27:     public void buttonClicked();
28:  }
29:
30:  @Override
31:  public void onAttach(Activity activity) {
32:  super.onAttach(activity);
33:  if(activity instanceof MyFragmentListener) {
34:       listener = (MyFragmentListener) activity;
35:    } else {
36:  throw new ClassCastException(activity.toString()
37:          + " Activity  must implemenet
38:  MyFragmentListener");
```

```
39:     }
40:   }
41:
42:   @Override
43:   public void onDetach() {
44:      super.onDetach();
45:      listener = null;
46:   }
47: }
```

فيما يلي سنقوم بشرح الأجزاء الرئيسية من الكود السابق: لتنفيذ التواصل بين الواجهة (Activity) والقطعة (Fragment) يجب أن يتم تعريف واجهة برمجية (Interface) تمثل المستمع (Listener) الذي سوف يستمع لأي تغيير يحصل في القطعة Fragment (أنظر الأسطر من 26 إلى 28)، وهو ما يتم في الجزء التالي من الكود:

```
public interface MyFragmentListener {
     public void buttonClicked();
}
```

ومن ثم تقوم الفعالية بعمل implement للواجهة البرمجية (MyFragmentListener). يمكن لأي مكون من مكونات التطبيق الاستماع للتغييرات التي تحصل في القطعة (Fragment) من خلال تطبيق (Implement) للواجهة البرمجية (MyFragmentListener). فائدة إنشاء الواجهة (MyFragmentListener) هو إبقاء القطعة قابلة للاستخدام دون أو تكون معتمدة على فعالية محددة.

بعد ذلك يتم تمرير الفعالية (Activity) إلى القطعة (Fragment) من خلال الدالة ()onAttach وهو ما يتم أثناء إنشاء القطعة كما هو موضح بالكود السابق (أنظر الأسطر من 31 إلى 40). في الدالة ()onAttach يتم الفحص ما إذا كانت الفعالية تطبق الواجهة البرمجية (MyFragmentListener) عن طريق instanceof (تفحص ما إذا كانت الفعالية لها نفس نوع (MyFragmentListener) (أنظر سطر 33). اذا تحقق ذلك، يتم حفظ موشر للفعالية في متغير باسم listener من نوع MyFragmentListener:

```
if(activity instanceof MyFragmentListener) {
          listener = MyFragmentListener) activity;
    }
```

عن طريق الكائن listener، والذي يمثل الفعالية المضيفة في هذه الحالة، يمكن تمرير الأوامر أو البيانات بتنفيذ أي من الدوال المعرفة في الواجهة البرمجية (MyFragmentListener). فمثلاً، في الكود السابق يتم إعلام الفعالية (Activity) عند النقر على الزر وذلك بتنفيذ الدالة ()buttonClicked (أنظر الأسطر من 14 إلى 23).

تمرين عملي (1-9)

هذا التطبيق هو مثال متكامل يوضح استخدام القطع (Fragments) لبناء تطبيق معلوماتي عن المدن الفلسطينية. المخرج النهائي للتطبيق موضح في شكل 4-9: في الوضع الأفقي Landscape، يتم عرض قائمة بأسماء المدن الفلسطينية على يسار الشاشة، وعند النقر على أي عنوان تظهر معلومات عن المدينة في يمين الشاشة. في الوضع الرأسي Portrait يتم عرض أسماء المدن في فعالية (Activity)، ومعلومات المدينة في فعالية جديدة.

لتصميم هذا التطبيق باستخدام القطع Fragments، سنقوم بانشاء قطعتين: واحدة لعرض أسماء المدن والثانية لعرض معلومات كل مدينة. واجهتا التطبيق في الوضع الرأسي Portait والأفقي Landscape تستخدمان نفس القطع ولكن بترتيب مختلف: حيث أنه في الوضع الرأسي يتم استخدام كل قطعة Fragment في فعالية منفصلة، بينما في الوضع الأفقي يتم جمع القطعتين Fragments في نفس واجهة الفعالية (Activity).

لاحظ أن التطبيق يتطلب تواصل بين القطع Fragments: حيث أن اختيار اسم مدينة من خلال القطعة الأولى Fragment يتطلب تعبئة القطعة الثانية Fragment بمعلومات المدينة التي تم اختيارها.

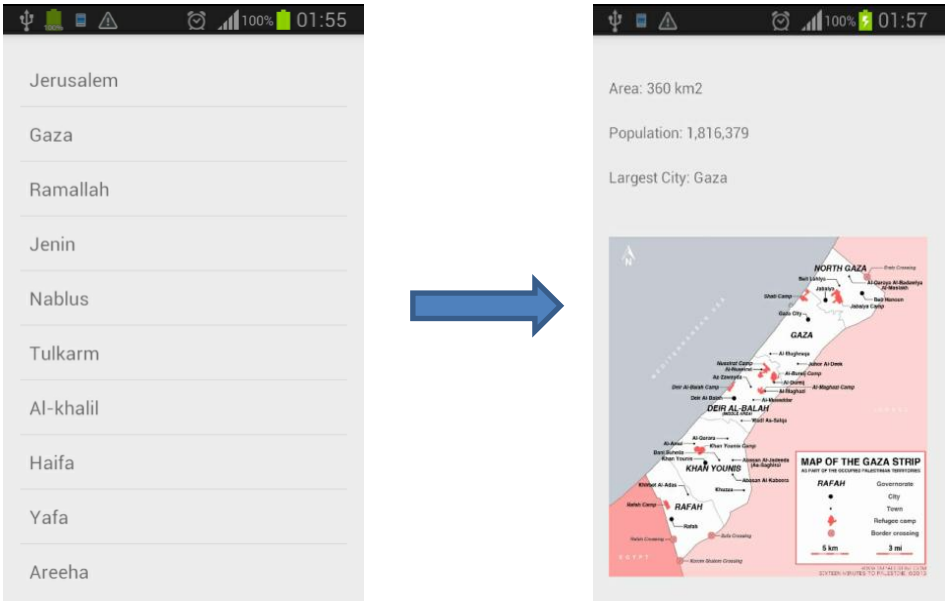
لبناء هذا التطبيق ننفذ الخطوات التالية:

أولاً: انشاء ملفات الواجهة الخاصة بالقطع (Fragments)

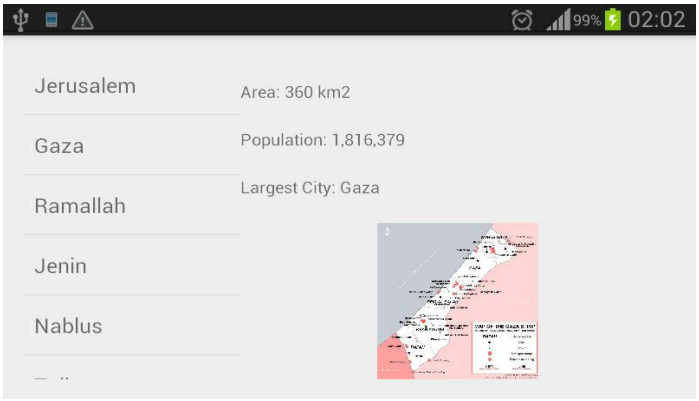
في المجلد layout ننشئ الملفات التالية:

- Listfragment_layout.xml: وهو يمثل تصميم الواجهة الخاصة بالقائمة (ListView) التي تعرض أسماء المدن. هذا التصميم سيكون ضمن قطعة واحدة (Fragment):

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <ListView
        android:id="@+id/cityListView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >
    </ListView>
</LinearLayout>
```



واجهات التطبيق الخاص بتمرين 1-9 في وضع العمل الرأسي (Portrait): التطبيق مكون من قطعتين وكل قطعة تعرض في واجهة منفصلة



واجهة التطبيق الخاص بتمرين 1-9 في وضع العمل الأفقي (Landscape): التطبيق مكون من قطعتين يتم عرضهما معاً في نفس الواجهة

شكل 4-9: واجهات وأوضاع العمل للتطبيق الخاص بتمرين 1-9

- Detailsfragment_layout.xml: وهو يمثل الواجهة الخاصة بعرض معلومات المدينة التي يتم اختيارها من القائمة، وسيمثل أيضاً قطعة واحدة (Fragment). يتكون التصميم من عناصر عرض (TextView)

لعرض مساحة المدينة وعدد سكانها وأكبر تجمعاتها، بالإضافة لعنصر عرض (ImageView) لعرض خارطة المدينة.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView
        android:id="@+id/tvArea"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:text="TextView" />
    <TextView
        android:id="@+id/tvPopulation"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:text="TextView" />
    <TextView
        android:id="@+id/tvLargestCity"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:text="TextView" />
    <ImageView
        android:id="@+id/ivMap"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:textSize="20sp"
        android:layout_marginTop="20dp"
        android:src="@drawable/ic_launcher"
        />
</LinearLayout>
```

ثانياً: إنشاء الفئات (classes) الخاصة بالقطع Fragments كالتالي:

- MyDetailsFragment: وتمثل الفئة (class) الخاصة بالقطعة (Fragment) التي تعرض معلومات المدينة التي يتم اختيارها من القائمة.

```
1:  public class MyDetailsFragment extends Fragment{
2:
3:   TextView tvArea;
4:   TextView tvPopulation;
5:   TextView tvLargestCity;
6:   ImageView ivMap;
7:
8:   @Override
9:   public View onCreateView(LayoutInflater inflater,
10:  ViewGroup container,
11:            Bundle savedInstanceState) {
12:       View view =
13:  inflater.inflate(R.layout.detailsfragment_layout,
14:  container, false);
15:       tvArea = (TextView)
16:  view.findViewById(R.id.tvArea);
17:       tvPopulation = (TextView)
18:  view.findViewById(R.id.tvPopulation);
19:       tvLargestCity = (TextView)
20:  view.findViewById(R.id.tvLargestCity);
21:       ivMap = (ImageView)
22:  view.findViewById(R.id.ivMap);
23:       return view;
24:  }
25:
26:  public void itemSelected(String cityName){
27:       getActivity().setTitle(cityName);
28:       if(cityName.equals("Gaza")){
29:            tvArea.setText("Area: 360 km2");
30:            tvPopulation.setText("Population:
31:  1,816,379");
32:            tvLargestCity.setText("Largest City: Gaza");
33:            ivMap.setImageResource(R.drawable.gaza);
34:       }else if(…){
35:        // If another city do something else …
36:    }
37:  }
38:}
```

لاحظ أن الدالة inflate() (أنظر الأسطر من 12 إلى 14) التي يتم تنفيذها داخل الدالة onCreateView() تستخدم ملف الواجهة المنشأ مسبقاً detailsfragment_layout.xml لإنشاء واجهة القطعة أثناء تنفيذ الدالة onCreateView().

الدالة itemSelected() (أنظر الأسطر من 26 إلى 37) يتم من خلالها تعبئة عناصر الواجهة بناء على اسم المدينة المدخل إلى هذه الدالة. من المفترض أن يتم استدعاء هذه الدالة عند النقر على اسم المدينة من القائمة (ListView) الموجودة ضمن قطعة Fragment أخرى.

- MyListFragment: وتمثل الفئة (class) الخاصة بالقطعة (Fragment) التي تعرض قائمة بأسماء المدن بحيث يتم عرض معلومات عن المدينة التي يتم اختيارها ضمن القطعة MyDetailsFragment.

```
1:  public class MyListFragment extends Fragment{
2:
3:   MyListFragmentListener listener;
4:
5:   @Override
6:   public View onCreateView(LayoutInflater inflater,
7:  ViewGroup container,
8:             Bundle savedInstanceState) {
9:        return
10:  inflater.inflate(R.layout.listfragment_layout,
11:  container, false);
12:  }
13:
14:  @Override
15:  public void onActivityCreated(Bundle
16:  savedInstanceState) {
17:       super.onActivityCreated(savedInstanceState);
18:       String[] values =
19:  {"Jerusalem","Gaza","Ramallah","Jenin","Nablus",
20:  "Tulkarm","Al-khalil","Haifa","Yafa","Areeha"};
21:       ArrayList<String> listValues = new
22:  ArrayList<String>();
23:       for(int i=0; i<values.length; i++)
24:            listValues.add(values[i]);
25:       ArrayAdapter<String> adapter = new
26:  ArrayAdapter<String>(getActivity(),
27:  android.R.layout.simple_list_item_1, listValues);
28:       final ListView list = (ListView)
29:  this.getView().findViewById(R.id.cityListView);
30:       list.setAdapter(adapter);
```

```
31:       list.setOnItemClickListener(new
32:  OnItemClickListener(){
33:
34:            @Override
35:            public void onItemClick(AdapterView<?>
36:  adapterView, View view, int position,long id) {
37:                 String cityName =
38:  list.getItemAtPosition(position).toString();
39:                 if(listener != null)
40:                      listener.itemSelected(cityName);
41:            }
42:       });
43:  }
44:
45:  public interface MyListFragmentListener{
46:       public void itemSelected(String selectedItem);
47:  }
48:
49:  @Override
50:  public void onAttach(Activity activity) {
51:       super.onAttach(activity);
52:       if(activity instanceof MyListFragmentListener){
53:            listener = (MyListFragmentListener)
54:  activity;
55:       }else{
56:            throw new
57:  ClassCastException(activity.toString()
58:                      + " Activity must implemenet
59:  MyListFragmentListener");
60:       }
61:  }
62:}
```

الدالة inflate() (أنظر الأسطر من 9 إلى 11) التي يتم تنفيذها داخل الدالة onCreateView() تستخدم ملف الواجهة المنشأ مسبقاً listfragment_layout.xml لإنشاء واجهة القطعة أثناء تنفيذ الدالة onCreateView().

لاحظ أنه عند ربط القطعة بالفعالية الأم يتم تلقائياً تنفيذ الدالة onAttach() حيث تم كتابة هذه الدالة ليتم حفظ مؤشر للفعالية الأم (Activity) وذلك لتمرر إليها الأوامر لاحقاً (أنظر الأسطر من 50 إلى 61). القطعة MyListFragment من المفترض أن تمرر اسم المدينة التي يتم اختيارها إلى القطعة الأخرى MyDetailsFragment وهو ما يجب أن يتم عن طريق الفعالية الأم.

في الدالة onActivityCreated() (أنظر الأسطر من 15 إلى 43) يتم إنشاء قائمة بأسماء المدن ومن ثم تمريرها إلى محول من نوع ArrayAdapter (أنظر الأسطر من 25 إلى 27). يتم بعد ذلك الوصول لقائمة العرض (ListView) عن طريق الدالة findViewById() ومن ثم تمرير المحول ArrayAdapter إلى قائمة العرض (ListView) لتعبئها (أنظر الأسطر من 28 إلى 30). لاحظ أنه تم اختيار الدالة onActivityCreated() تحديداً لأن الفعالية (Activity) تكون قد اكتمل إنشاؤها عند تنفيذها وبالتالي يمكن تنفيذ الدالة Fragment.getActivity() والتي تلزم عن إنشاء المحول ArrayAdapter.

ثالثاً: إنشاء واجهات الفعاليات (Activites)

بعد انشاء القطع (Fragments)، سنقوم الآن بإنشاء الفعاليات (activites) التي ستستخدم القطع لإنشاء واجهة التطبيق. لاحظ أن التطبيق يعمل بطريقتين بناءً على اتجاه شاشة الجهاز: اذا كان الجهاز أفقياً (Landscape) يتم عرض واجهة واحدة تشمل كلاً من MyListFragment و MyDetailsFragment. أما إذا كان الجهاز رأسياً (Portrait)، يتم عرض MyListFragment في فعالية، وMyDetailsFragment في فعالية أخرى.

لتنفيذ ذلك ننشئ ملف التصميم التالي activity_main.xml ،والخاص بالواجهة الرئيسية في الوضع الأفقي (Landscape)، ونضعه في المجلد layout:

```
<LinearLayoutxmlns:android="http://schemas.android.com/apk/
res/android"
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/LinearLayout1"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:paddingBottom="@dimen/activity_vertical_margin"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
tools:context=".MainActivity">

<fragment
android:id="@+id/listFragment"
android:name="ps.edu.ucas.fragments.MyListFragment"
android:layout_width="0dp"
android:layout_weight="1"
android:layout_height="match_parent"/>

<fragment
android:id="@+id/detailsFragment"
android:name="ps.edu.ucas.fragments.MyDetailsFragment"
android:layout_width="0dp"
android:layout_weight="2"
```

```
android:layout_height="match_parent"/>

</LinearLayout>
```

لاحظ أن الواجهة تضم القطعتين MyListFragment و MyDetailsFragment وكل منهما يشير إلى الفئة (class) الخاصة بالقطعة من خلال android:name.

بعد ذلك نقوم بإنشاء ملف الواجهات الخاصة بعرض التطبيق في الوضع الرأسي (Portrait). كما ذكرنا مسبقاً، يستخدم التطبيق في الوضع الرأسي (Portrait) واجهتين تستخدم كل منهما قطعة مختلفة. الملف التالي activity_main.xml يوضح تصميم الفعالية التي تتضمن (MyListFragment)، حيث يتم وضع الملف في مجلد باسم layout-port حتى يتم استخدامه تلقائياً في الوضع الرأسي (Portrait).

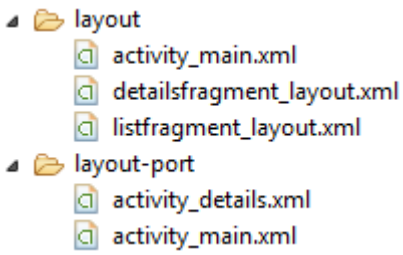
لاحظ أن اسم ملف الواجهة activity_main.xml يطابق اسم ملف الواجهة المنشأ مسبقا، ولكنه موجود في مجلد آخر. عند تشغيل التطبيق يتم اختيار الملف من المجلد الصحيح بناء على وضع العرض: فإذا كان العرض رأسياً يتم إنشاء الواجهة من الملف الموجود في المجلد layout_port وإذا كان العرض أفقياً يتم إنشاء الواجهة من الملف الموجود في المجلد layout.

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"

android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <fragment
        android:id="@+id/cityFragment"
        android:name="ps.edu.ucas.fragments.MyListFragment"
        android:layout_width="0dp"
        android:layout_weight="1"
        android:layout_height="match_parent" />
</LinearLayout>
```

ونقوم أيضاً بانشاء ملف واجهة آخر خاص بالفعالية التي تعرض معلومات المدينة في وضع Portrait ويحتوي على MyDetailsFragment:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LinearLayout1"
    android:tag="portrait"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"

android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".DetailsActivity" >

    <fragment
        android:id="@+id/detailsFragment"
        android:name="
ps.edu.ucas.fragments.MyDetailsFragment"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

</LinearLayout>
```

الترتيب النهائي لملفات تصميم الهيكلية للفعاليات (Activities) و القطع (Fragments) ضمن ملفات التطبيق يظهر كما بشكل 5-9:



شكل 5-9: ترتيب ملفات الواجهة الخاصة بتمرين 1-9

رابعاً: إنشاء الفعاليات (Activites): بعد إنشاء الواجهات الخاصة بالفعاليات Activites، يتم إنشاء الفئات (classes) الخاصة بها:

- الفعالية MainActivity: وتمثل الواجهة الرئيسية للتطبيق:

```
1:  public class MainActivity extends Activity implements
2:  MyListFragmentListener{
3:
4:   @Override
5:   protected void onCreate(Bundle savedInstanceState) {
6:        super.onCreate(savedInstanceState);
7:        setContentView(R.layout.activity_main);
8:   }
9:
10:  @Override
11:  public void itemSelected(String cityName){
12:       MyDetailsFragment fragment = (MyDetailsFragment)
13:  this.getFragmentManager().findFragmentById(
14:  R.id.detailsFragment);
15:       if(fragment != null && fragment.isInLayout()){
16:            fragment.itemSelected(cityName);
17:       }else{
18:            Intent intent = new
19:  Intent(getApplicationContext(),
20:  DetailsActivity.class);
21:            intent.putExtra("cityName", cityName);
22:            startActivity(intent);
23:       }
24:  }
25:}
```

لاحظ أن الدالة setContentView() (أنظر سطر رقم 7) المسؤولة عن ربط كود الفعالية بملف الواجهة يستخدم الملف المعرف بـ (activity_main). نظراً لوجود ملفي واجهة بنفس الاسم (activity_main)، أحدها في المجلد layout والآخر في المجلد layout-port، يقوم النظام تلقائياً باستخدام الملف المناسب بناء على وضع الجهاز: ففي وضع الشاشة الأفقي يتم استخدام ملف الواجهة (activity_main) الذي يتضمن MyListFragment و MyDetailsFragment. في الوضع الرأسي يتم استخدام ملف الواجهة activity_main الموجود في المجلد layout-portait والذي يحوي MyListFragment فقط. لاحظ أيضاً أن الفعالية تقوم بعمل implement للواجهة البرمجية MyFragmentListListener وذلك حتى تتمكن من الاستماع للحدث الصادر من MyListFragment.

عند استقبال حدث ما من MyListFragment (عند النقر على اسم مدينة من القائمة (ListView)، يتم تنفيذ الدالة itemSelected() والتي يمرر لها اسم المدينة التي تم اختيارها من القائمة (ListView) (أنظر سطر رقم 11).

الإجراء المتبع بعد ذلك يعتمد على وضع شاشة الجهاز: فإذا كانت الشاشة أفقية فإن القطعة MyDetailsFragment من المفترض أن يكون ضمن الواجهة الرئيسية وهو ما يتم التحقق منه من خلال تنفيذ الدالة ()isInLayout (أنظر سطر رقم 15). إذا كان MyDetailsFragment موجود ضمن الواجهة يتم إرسال النص له من خلال الدالة ()itemSelected الموجودة ضمن MyDetailsFragment (أنظر سطر رقم 16). أما إذا لم يكن MyDetailsFragment موجوداً فهذا يعني أن الجهاز في الوضع الرأسي، ومن ثم يتم إنشاء فعالية من نوع DetailsActivity وإرسال نص الخبر لها من خلال هدف (Intent) (أنظر الكود من سطر 18 إلى 23).

- الفعالية DetailsActivity: هذه الفعالية يتم عرضها فقط في الوضع الرأسي لإظهار تفاصيل الخبر:

```
1:  public class DetailsActivity extends Activity {
2:
3:   @Override
4:   protected void onCreate(Bundle savedInstanceState) {
5:        super.onCreate(savedInstanceState);
6:
7:   if(this.getResources().getConfiguration().
8:  orientation == Configuration.ORIENTATION_LANDSCAPE){
9:            this.finish();
10:           return;
11:       }
12:       setContentView(R.layout.activity_details);
13:       Intent intent = this.getIntent();
14:       Bundle extras = intent.getExtras();
15:       if(extras != null){
16:           String cityName =
17:  intent.getExtras().getString("cityName");
18:           MyDetailsFragment fragment =
19:  (MyDetailsFragment)
20:  this.getFragmentManager().findFragmentById(
21:  R.id.detailsFragment);
22:           fragment.itemSelected(cityName);
23:       }
24:  }
25:}
```

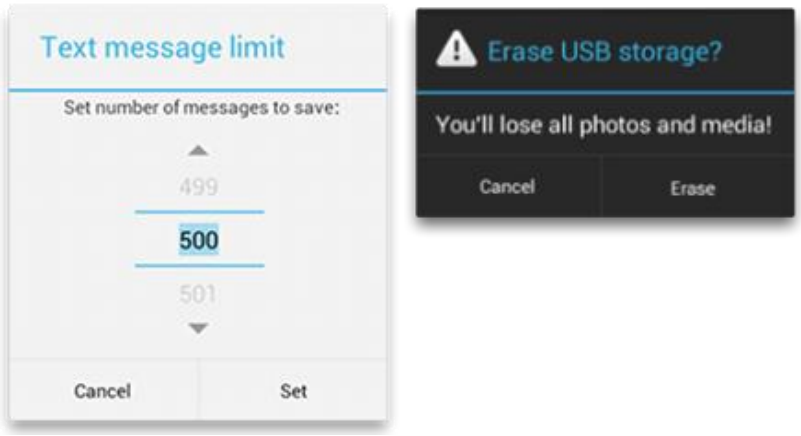
لاحظ أنه عند إنشاء هذه الفعالية يتم تنفيذ إجراءين أساسيين في الدالة ()onCreate وهما:

أولاً: التحقق من وضع شاشة الجهاز، فإذا كان الجهاز أفقياً لا يتم عرض الفعالية ويتم إنهاؤها بتنفيذ الدالة ()Activity.finish (أنظر الكود من سطر 7 إلى 11).

ثانياً: استقبال الهدف (Intent) المرسل من MyListFragment من خلال الفعالية المضيفة MainActivity، ومن ثم استخراج البيانات المرسلة وتتضمن أسم المدينة التي تم اختيارها (أنظر الأسطر من 13 إلى 17). من المفترض بعد ذلك إرسال اسم المدينة إلى القطعة MyDetailsFragment والمسئولة عن عرض التفاصيل. لذلك يتم الوصول للقطعة MyDetailsFragment عن طريق مدير القطع ومن ثم تمرير اسم المدينة من خلال الدالة itemSelected() (أنظر الأسطر من 18 إلى 22).

مربعات الحوار (Dialogs)

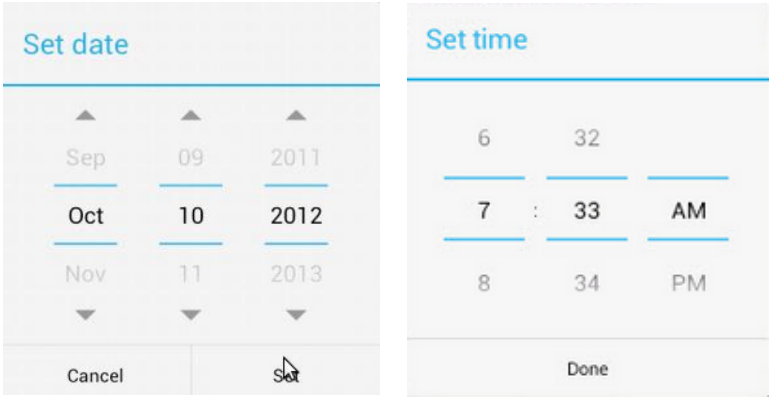
مربع الحوار (Dialog) هو نافذة صغيرة تطلب عادةً من المستخدم اتخاذ قرار أو إدخال معلومات إضافية. مربع الحوار لا يملأ الشاشة كما هو الحال في واجهة الفعالية (Activity)، ويستخدم عادة في الحالات التي تتطلب من المستخدمين اتخاذ إجراء ما قبل أن يتمكنوا من المضي قدما. شكل 6-9 يوضح أمثلة على مربعات الحوار Dialogs:



شكل 6-9: نماذج لمربعات الحوار (Dialogs)

الفئة (Dialog class) هي الفئة الأساسية التي تتفرع منها الأنواع المختلفة لمربعات الحوار. ينصح بتجنب استخدام الفئة (Dialog class) لإنشاء مربع الحوار Dialog، وبدلاً منه ينصح باستخدام أي من الفئات الفرعية (subclasses) التالية:

- AlertDialog: وهو مربع حوار يمكنه افتراضياً عرض عنوان، ثلاثة أزرار بحد أقصى، قائمة من عناصر الاختيار أو تصميم مخصص Custom layout.
- DatePickerDialog و TimePickerDialog: وهي مربعات حوار تستخدم لتحديد الوقت والتاريخ، وتظهر في شكل 7-9:

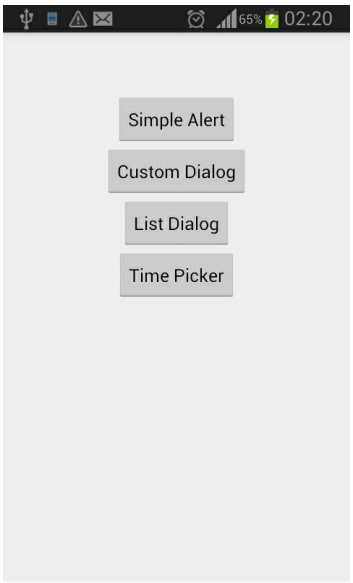


شكل 7-9: مربعات حوار من نوع DatePickerDialog و TimePickerDialog

هذه الفئات (classes) تحدد شكل وبناء مربع الحوار Dialog، ولكن يجب استخدام كائن من نوع DialogFragment كحاوية Container يتم من خلاله إنشاء وعرض مربع الحوار (Dialog). استخدام DialogFragment كحاوية لمربع الحوار يضمن معالجة الأحداث النائجة عن دورة حياة مربع الحوار بشكل صحيح، حيث ترتبط دورة حياة مربع الحوار بدورة حياة القطعة DialogFragment. يمكن مثلاً تنفيذ إجراء عند النقر على زر Back بإضافة كود في الدالة DialogFragment.onPause(). بالإضافة إلى ذلك، استخدام DialogFragment كحاوية لمربع الحواية يوفر إمكانية إعادة استخدام مربع الحوار في تطبيقات أخرى تماماً كما يحدث في حالة القطعة (Fragment).

تمرين عملي (9-2)

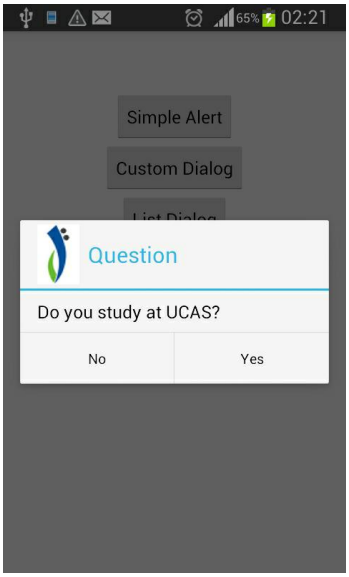
يهدف هذا التمرين إلى التدرب على إنشاء مربعات حوار (Dialogs) من أنواع مختلفة. الواجهة الرئيسية للتطبيق تظهر في شكل 8-9، وتتكون من مجموعة من الأزرار عند النقر على أي منها يظهر مربع حوار Dialog بخصائص محددة.



شكل 8-9: واجهة التطبيق الرئيسية الخاصة بالتمرين 2-9

في ما يلي نستعرض مربعات الحوار Dialogs المختلفة وطريقة إنشائها:

أولاً: مربع الحوار Alert Dialog: وهو يظهر عند النقر على الزر "Simple Alert" كما بالشكل 9-9:



شكل 9-9: مربع الحوار (AlertDialog) الخاص بالتمرين 2-9

لإنشاء هذا المربع (Dialog)، قم بإنشاء فئة (class) باسم MySimpleAlertDialog والتي تتفرع من الفئة DialogFragment وتحتوي الكود التالي:

```
1:  public class MySimpleAlertDialog extends
2:  DialogFragment{
3:   @Override
4:   public Dialog onCreateDialog(Bundle
5:  savedInstanceState) {
6:        AlertDialog.Builder builder = new
7:  AlertDialog.Builder(getActivity());
8:        builder.setMessage("Do you study at UCAS?");
9:        this.setCancelable(false);
10:       builder.setTitle("Question");
11:       builder.setIcon(R.drawable.ucaslogo);
12:       builder.setPositiveButton("Yes", new
13:  OnClickListener(){
14:
15:           @Override
16:           public void onClick(DialogInterface dialog,
17:  int which) {
18:                Toast.makeText(getActivity(), "You are
19:  a UCAS student", Toast.LENGTH_LONG).show();
20:           }
21:       });
22:       builder.setNegativeButton("No", new
23:  OnClickListener(){
24:
25:           @Override
26:           public void onClick(DialogInterface dialog,
27:  int which) {
28:                Toast.makeText(getActivity(), "You are
29:  not a UCAS student", Toast.LENGTH_LONG).show();
30:           }
31:
32:       });
33:       AlertDialog dialog = builder.create();
34:       dialog.setCanceledOnTouchOutside(false);
35:       return dialog;
36:  }
37:}
```

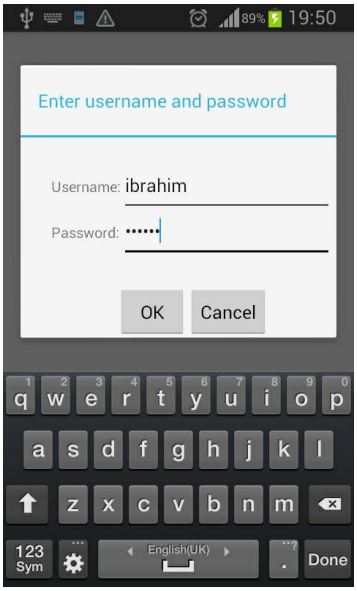
لإنشاء مربع الحوار AlertDialog المطلوب يتم كتابة الكود في الدالة ()onCreateDialog والتي ترجع كائن من نوع Dialog. كما تلاحظ، يتم في البداية إنشاء كائن من نوع AlertDialog.Builder (أنظر سطر رقم 6 و7) والذي يتم من خلاله تحديد خصائص مربع الحوار. لاحظ أنه تم تحديد الرسالة المعروضة في مربع الحوار باستخدام الدالة ()setMessage، وتم تحديد العنوان باستخدام الدالة ()setTitle والإيقونة الظاهرة في مربع الحوار باستخدام الدالة ()setIcon (أنظر الأسطر من 8 إلى 11).

يمكن أيضا من خلال الكائن من نوع Builder تحديد الأزرار والنصوص المعروضة بها والإجراءات التي تنفذ عند النقر علي أي من هذه الأزرار. يمكن إفتراضياً إضافة ثلاثة أزرار بحص أقصى داخل AletDialog وذلك باستخدام الدوال ()setPositiveButton، ()setNegativeButton و ()setNeutralButton (أنظر الأسطر من 12 إلى 32). كل دالة من الدوال السابقة يمرر لها النص الذي يكتب على الزر (مثل OK أو Yes) بالإضافة إلى كائن من نوع (OnClickListener) يحدد الإجراء المراد تنفيذه عند النقر على الزر. كود الإجراء المطلوب يتم كتابته في الدالة OnClickListener.onClick.

بعد تحديد خصائص مربع الحوار AlertDialog ،يتم إنشاؤه بتنفيذ الدالة ()Builder.create والتي ترجع كائن من نوع Dialog. في النهاية يتم إرجاع الكائن Dialog كمخرج من الدالة ()onCreateDialog (أنظر الأسطر من 33 إلى 35).

لاحظ أننا استخدمنا الأمر (Dialog.setCanceledOnTouchOutside(false (أنظر سطر رقم 34) لتجنب إغلاق مربع الحوار عند النقر خارجه كما يحدث إفتراضياً. في هذه الحالة لن يتم الإغلاق إلى من خلال الأزرار الموجودة داخل مربع الحوار.

ثانياً: مربع الحوار المخصص Custom Dialog (أنظر شكل 10-9) ويظهر عند النقر على الزر "Custom Dialog" الظاهر في شكل 9-8 .



شكل 10-9: مربع الحوار المخصص (Custom Dialog) الخاص بالتمرين 2-9

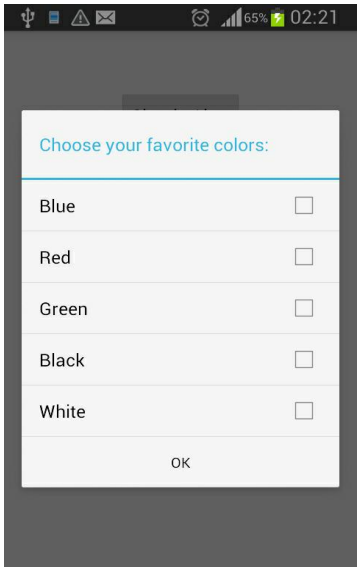
مربع الحوار المخصص يوفر إمكانية استخدام تصميم معد مسبقاً في ملف هيكلية من نوع XML لبناء واجهة مربع الحوار، وهو ما يعطي مرونة أكبر في تصميم الواجهة من مربع الحوار AlertDialog الإفتراضي. الكود التالي يمثل الفئة (class) الخاصة بمربع الحوار المخصص:

```
1:  public class MyCustomDialog extends DialogFragment{
2:
3:   @Override
4:   public Dialog onCreateDialog(Bundle
5:  savedInstanceState) {
6:        AlertDialog.Builder builder = new
7:  AlertDialog.Builder(getActivity());
8:        builder.setTitle("Enter username and password");
9:        LayoutInflater inflater = (LayoutInflater)
10:  this.getActivity().getLayoutInflater();
11:       View view =
12:  inflater.inflate(R.layout.dialog_layout, null);
13:       builder.setView(view);
14:       Button btnOK =
15:  (Button)view.findViewById(R.id.btnOK);
16:       btnOK.setOnClickListener(new OnClickListener(){
17:            @Override
18:            public void onClick(View arg0) {
```

```
19:                     Toast.makeText(getActivity(), "Thank
20:   you", Toast.LENGTH_SHORT).show();
21:                     MyCustomDialog.this.dismiss();
22:                 }
23:             });
24:             return builder.create();
25:   }
26:}
```

لاحظ أن إنشاء مربع الحوار المخصص مشابه إلى حد كبير لطريقة إنشاء مربع الحوار الإفتراضي المنشأ في الخطوة السابقة، فهو من نوع AlertDialog ومضمن في DialogFragment. الاختلاف الوحيد أنه تم استخدام كائن من نوع LayoutInflater لإنشاء الواجهة باستخدام ملف واجهة بالمعرف R.layout.dialog_layout (أنظر سطر رقم 11 و12). الواجهة المنشأة تكون مضمنة في كائن من نوع View يتم إضافته لمربع الحوار باستخدام الدالة Builder.setView() (أنظر سطر رقم 13). بعد إنشاء الواجهة، يمكن الوصول برمجياً لأي عنصر داخلها بتنفيذ الدالة findViewById() وذلك لإضافة مستمع (Listener) لحدث معين. في المثال الموضح يتم الوصول إلى الزر المعرف بـ R.id.btnOK ومن تسجيل مستمع (Listener) لمعالجة إجراء النقر عليه (أنظر الكود من سطر 14 إلى 23).

ثالثاً: مربع حوار يعرض قائمة من عناصر الإختيار كما يظهر في شكل 11-9 ، ويتم إنشاؤه عند النقر على زر "List Dialog" الموجود ضمن الواجهة في شكل 8-9.



شكل 11-9: مربع الحوار يعرض قائمة من العناصر، الخاص بالتمرين 2-9

لإنشاء القائمة المضمنة في مربع الحوار، نقوم بإنشاء مصفوفة من النصوص في ملف المصادر string باستخدام الخاصية string-array كما هو موضح:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">dialogtest</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string-array name="color_array">
        <item >Blue</item>
        <item >Red</item>
        <item >Green</item>
        <item >Black</item>
        <item >White</item>
    </string-array>
</resources>
```

الفئة (MyListDialog class) التالية توضح طريقة إنشاء مربع الحوار :

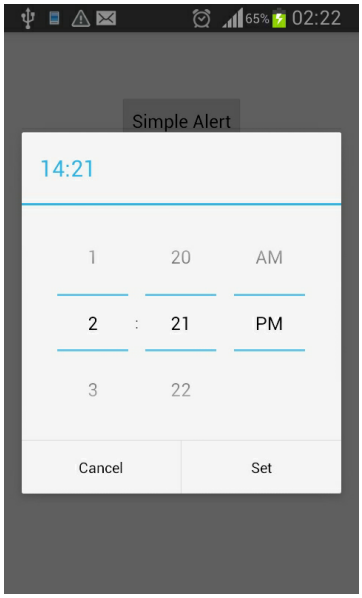
```
1:  public class MyListDialog extends DialogFragment{
2:
3:   Boolean[] selections = {false, false, false, false,
4:  false};
5:
6:   @Override
7:   public Dialog onCreateDialog(Bundle
8:  savedInstanceState) {
9:        AlertDialog.Builder builder = new
10:  AlertDialog.Builder(getActivity());
11:       builder.setTitle("Choose your favorite colors:");
12:       builder.setMultiChoiceItems(R.array.color_array,
13:  null, new OnMultiChoiceClickListener(){
14:
15:           @Override
16:           public void onClick(DialogInterface dialog,
17:  int which, boolean isChecked) {
18:                selections[which] = isChecked;
19:           }
20:       });
21:       builder.setPositiveButton("OK", new
22:  OnClickListener(){
23:
24:           @Override
```

```
25:             public void onClick(DialogInterface dialog,
26:  int which) {
27:                   String result = "";
28:                   for(Boolean selection: selections)
29:                        result+=selection+" , ";
30:                   Toast.makeText(getActivity(), result,
31:  Toast.LENGTH_LONG).show();
32:             }
33:       });
34:       return builder.create();
35:  }
36:}
```

لاحظ أن مربع الحوار تم أنشاؤه بطريقة مشابهة لمربعات الحوار السابقة باستثناء استخدام الدالة Builder.setMultiChoiceItems() والتي يمرر لها معرف المصفوفة التي تم إنشاؤها وهو R.array.color_array وكائن من نوع (OnMultiChoiceClickListener) والذي يستخدم للاستماع لحدث اختيار عنصر من القائمة حيث يتم تلقائياً تنفيذ الدالة onClick() عن النقر على أي عنصر (أنظر الكود من سطر 12 إلى 20).

لاحظ أن الدالة Builder.setMultiChoiceItems() تستخدم لتفعيل خاصية الاختيار المتعدد من قائمة العرض. يمكن استخدام الدالة Builder.setSingleChoiceItems() لدعم اختيار عنصر منفرد من القائمة.

رابعاً: واجهة تحديد الوقت TimePickerDialog (أنظر شكل 12-9) وهو أحد مربعات الحوار جاهزة التصميم والذي يمكن استخدامه في التطبيقات التي تتطلب تحديد الوقت، ويظهر عند النقر على الزر "Time Picker" في شكل 8-9:



شكل 12-9: مربع حوار من نوع TimePickerDialog خاص بالتمرين 2-9

هناك أيضاً مربع حوار مشابه باسم DatePickerDialog وهو يستخدم لتحديد التاريخ. إنشاء مربع حوار من نوع TimePickerDialog يتم باستخدام الكود التالي:

```
1:  public class TimerPickerFragment extends
2:  DialogFragment{
3:
4:   @Override
5:   public Dialog onCreateDialog(Bundle
6:  savedInstanceState) {
7:        Calendar c = Calendar.getInstance();
8:        int hour = c.get(Calendar.HOUR_OF_DAY);
9:        int minute = c.get(Calendar.MINUTE);
10:       TimePickerDialog dialog = new
11:  TimePickerDialog(this.getActivity(), new
12:  OnTimeSetListener(){
13:            @Override
14:            public void onTimeSet(TimePicker view, int
15:  hour, int minute) {
16:                 Toast.makeText(getActivity(), "You set:
17:  "+hour+" hour, "+minute+" minutes",
18:  Toast.LENGTH_LONG).show();
19:            }
```

```
20:         }, hour, minute, false);
21:         return dialog;
22:   }
23: }
```

لاحظ أنه تم إنشاء كائن من نوع TimePickerDialog وتم تمرير كائن الفعالية ()getActivity بالإضافة لكائن من نوع OnTimeSetListener والذي يحدد الإجراء الذي يتم تنفيذه عن تحديد توقيت جديد (في المثال أعلاه يتم طباعة الوقت) (أنظر الأسطر من 12 إلى 22). تم أيضاً تمرير الوقت الحالي (بالساعة والدقيقة) والذي تم قراءته باستخدام كائن من نوع Calendar (أنظر الأسطر من 7 إلى 9) إلى الكائن TimePickerDialog وذلك لتحديد الوقت الافتراضي عن فتح مربع الحوار.

بعد توضيح الكود الخاص بمربعات الحوار (Dialogs) المختلفة، سنقوم الآن بعرض الكود الخاص بالفعالية MainActivity والتي يتم من خلالها تشغيل مربعات الحوار:

```
1:  public class MainActivity extends Activity {
2:
3:   @Override
4:   protected void onCreate(Bundle savedInstanceState) {
5:         super.onCreate(savedInstanceState);
6:         setContentView(R.layout.activity_main);
7:   }
8:
9:   public void showSimpleAlert(View view){
10:        MySimpleAlertDialog dialog = new
11:  MySimpleAlertDialog();
12:        FragmentTransaction) ftr =
13:  this.getFragmentManager().beginTransaction();
14:        dialog.show(ftr, "mytag");
15:  }
16:
17:  public void showCustomAlert(View view){
18:        MyCustomDialog dialog = new MyCustomDialog();
19:        FragmentTransaction ftr =
20:  this.getFragmentManager().beginTransaction();
21:        dialog.show(ftr, "mytag");
22:  }
23:
24:  public void showListDialog(View view){
25:        MyListDialog dialog = new MyListDialog();
26:        FragmentTransaction ftr =
27:  this.getFragmentManager().beginTransaction();
```

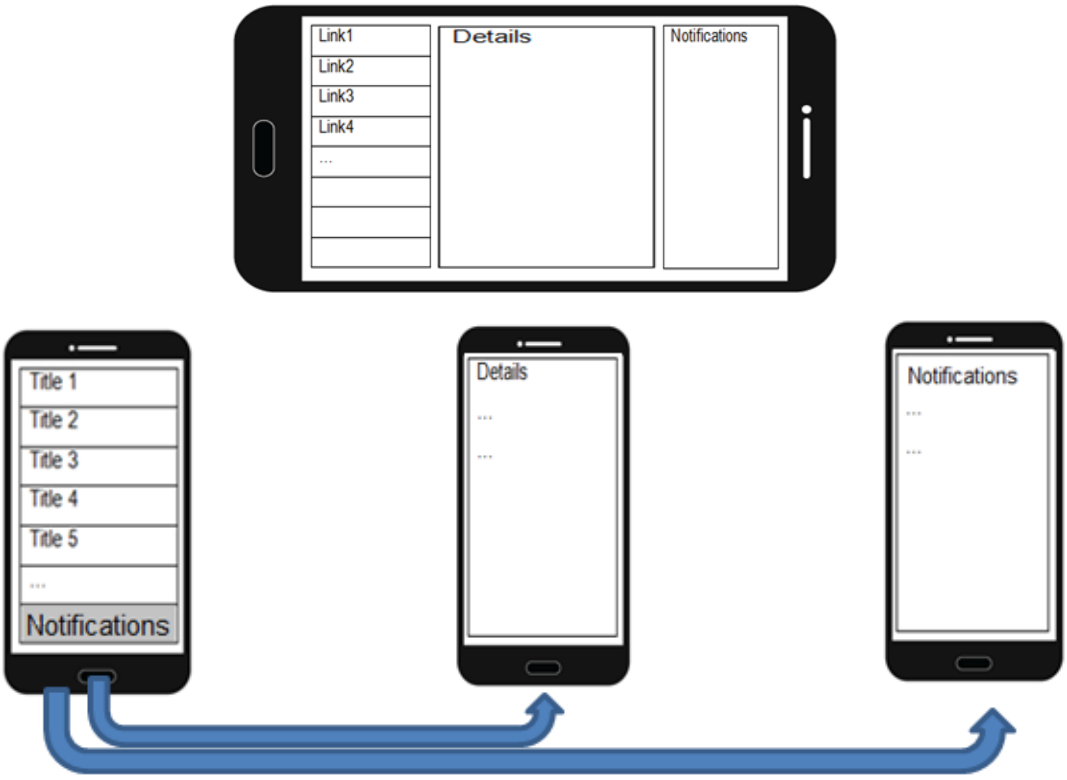
```
28:       dialog.show(ftr, "mytag");
29:   }
30:
31:   public void showTimePickerDialog(View view){
32:       TimerPickerFragment dialog = new
33:   TimerPickerFragment();
34:       FragmentTransaction ftr =
35:   this.getFragmentManager().beginTransaction();
36:       dialog.show(ftr, "mytag");
37:   }
38: }
```

لعرض مربعات الحوار المختلفة، يجب بدايةً الوصول للكائن FragmentManager المسؤول عن إدارة كل ما له علاقة بالقطع (Fragments) ومن ثم إنشاء كائن من نوع FragmentTransaction. لاحظ أن طريقة عرض أي مربع حوار Dialog تتم باستخدام الدالة Dialog.show() والتي يمرر لها كائن من نوع (FragmentTransaction) بالإضافة إلى اسم معرف Tag.

أسئلة على الوحدة التاسعة

1. ما مميزات استخدام القطع (Fragments) لتصميم واجهات تطبيقات أندرويد؟
2. ما الفرق بين القطعة (Fragment) والفعالية (Activity)؟
3. أذكر طريقتين لإضافة القطع (Fragments) للفعالية (Activity)؟
4. لماذا ينصح بتضمين مربع الحوار (AlertDialog) أو أي أنواع أخرى من مربعات الحوار (Dialog) داخل قطعة من نوع (DialogFragment)؟
5. قد بتصميم تطبيق إخباري بالخصائص التالية: واجهة التطبيق في الوضع الأفقي (Landscape) موضحة في الأسفل، حيث يتم عرض قائمة الروابط (Links) على اليسار، وعند النقر على أي رابط يتم عرض التفاصيل في المنتصف، بينما على يمين الشاشة يتم عرض قائمة بالإشعارات و التحديثات.

   في الوضع الرأسي (Portrait) يتم عرض كل جزء من الأجزاء السابقة في فعالية (Activity) منفصلة كما هو موضح بالأسفل، حيث تعرض الفعالية الرئيسية قائمة من الروابط في (ListView) وأسفلها زر باسم "Notifications". عند النقر على أي رابط يتم إظهار التفاصيل في فعالية جديدة، وعند النقر على زر "Notifications" يتم عرض الإشعارات في فعالية أخرى كما هو موضح.



6. في تمرين 2-9 تم إنشاء مربع حوار مخصص (CustomDialog) (أنظر شكل 10-9) حيث يتم إدخال اسم مستخدم وكلمة مرور من خلاله. قد بتعديل التمرين بحيث يتم إعادة البيانات المدخلة إلى الفعالية الرئيسية (MainActivity).

تلميح: بما أن مربع الحوار (AlertDialog) مضمّن داخل قطعة من نوع (DialogFragment)، يمكن استخدام نفس الطريقة التي تم شرحها للتواصل بين القطعة (Fragment) والفعالية المضيفة (Activity).

7. في الوحدة الخامسة، تمرين 2-5 قمنا بإنشاء درج تصفح (Navigation Drawer). ذكرنا أن تصميم الواجهة مكون من جزئين هما الجزء الثابت (Main content view) والجزء المضمن في درج التصفح. ذكرنا أيضاً أن الجزء الثابت يمكن إدراجه كقطعة (Fragment) حيث يتيح ذلك إمكانية تغيير المحتوى الأساسي إثناء عمل التطبيق عند طريق استبدال القطع (Fragments). قم بتعديل تمرين 2-5 لاستخدام القطع (Fragment) لتحقيق هذا الغرض.

الوحدة العاشرة:

نشر تطبيقات أندرويد

(Publishing Android Applications)

يتعلم الطالب في هذه الوحدة:

- ✓ خطوات ومستلزمات نشر تطبيق أندرويد على متجر جوجل (Google Play).
- ✓ طريقة تهيئة التطبيق للنشر وتوقيعه رقمياً.
- ✓ إنشاء سجل المطور (Developer Profile).

متجر جوجل (Google Play Store)

متجر جوجل (Google Play Store) يمثل الآلية الرسمية لنشر تطبيقات الأندرويد. إن نشر تطبيقك على متجر جوجل يجعل من التطبيق قابل للتحميل والاستخدام من قبل المستخدمين في جميع أنحاء العالم. يمكن أيضا للمستخدمين إضافة تعليقات أو إعطاء تقييم للتطبيق بما يمكن من اكتشاف الأخطاء أو تحسين التطبيق لاحقاً. كما يوفر متجر جوجل بض الإحصائيات التي يمكن الاستفادة منها لقياس نجاح أي تطبيق.

في هذه الوحدة سيتم شرح خطوات نشر التطبيق على متجر جوجل (Google Play Store).

يمكن تلخيص خطوات نشر التطبيق على متجر جوجل بما يلي:

1. تصدير التطبيق (Export) كملف من نوع APK (Android Package).
2. عمل توقيع رقمي (Digital Signature) للتطبيق باستخدام شهادة (Certificate). توقيع التطبيق رقمياً يساعد نظام أندرويد في تحديد هوية مالك التطبيق.
3. رفع التطبيق على المتجر.
4. استخدام سوق أندرويد (Android Market) لاستضافة وبيع التطبيق.

في ما يلي سنقوم بشرح خطوات إعداد التطبيق للتوقيع ومن ثم سنقوم بشرح طريقة نشر التطبيق:

تحديد إصدار التطبيق

ابتداءً من إصدار أندرويد 1.0، يحتوي ملف الوثيقة (Manifest) لكل تطبيق على خاصيتي: android:versionCode و android:versionName. قيمة الخاصية الأولى تحدد رقم الإصدار الخاص بالتطبيق. لكل نسخة جديدة معدلة من التطبيق يجب إضافة 1 لهذه القيمة وذلك للتفريق بين النسخة الجديدة والقديمة. هذه القيمة غير مستخدمة من قبل نظام أندرويد وهي مفيدة فقط للمطورين لمعرفة إصدارات التطبيق.

قيمة الخاصية android:versionName تحتوي على معلومات الإصدار التي تظهر للمستخدمين وتأخذ الصيغة:

<major>.<minor>.<point>

إذا تم إجراء تغيير كبير على التطبيق يتم إضافة واحد للقيمة major، أما إذا تم إضافة تعديلات أو إضافات صغيرة أو ثانوية يتم زيادة قيمة minor أو point. فمثلاً، التطبيق الجديد قد يأخذ اسم الإصدار (Version Name) التالي: "1.0.0" و عدن عمل تعديل بسيط يأخذ اسم الإصدار "1.0.1" أو "1.1.0". عند عمل تعديل كبير يمكن أن يأخذ التطبيق اسم الإصدار: "2.0.0" وهكذا.

إذا كنت تنوي نشر التطبيق على سوق أندرويد الإلكتروني (www.android.com/market/) ، يجب أن تتوفر جميع البيانات التالية ضمن ملف الوثيقة (Manifest) الخاص بالتطبيق:

- android:versionCode (ضمن الخاصية <Manifest>).
- android:versionName (ضمن الخاصية <Manifest>).
- android:icon (ضمن الخاصية <application>).
- android:label (ضمن الخاصية <application>).

المثال التالي يوضح هذه البيانات ضمن ملف وثيقة (Manifest) لتطبيق باسم "UCAS APP":

```
<?xml version=”1.0” encoding=”utf-8”?>
<manifest
xmlns:android=”http://schemas.android.com/apk/res/android”
package=”ps.ucas.edu.ps.publishexample”
android:versionCode=”1”
android:versionName=”1.0” >
<uses-sdk android:minSdkVersion=”13” />
<uses-permission
android:name=”android.permission.INTERNET”/>
<uses-permission
android:name=”android.permission.ACCESS_FINE_LOCATION”/>
<uses-permission
android:name=”android.permission.ACCESS_COARSE_LOCATION”/>
<application
android:icon=”@drawable/ic_launcher”
android:label=”UCAS APP” >
<uses-library android:name=”com.google.android.maps” />
<activity
android:label=”@string/app_name”
android:name=”.LBSActivity” >
<intent-filter >
<action android:name=”android.intent.action.MAIN” />
<category android:name=”android.intent.category.LAUNCHER”
/>
</intent-filter>
```

```
</activity>
</application>
</manifest>
```

يجب التأكد من إضافة كل الأذونات (Permissions) التي يتطلبها التطبيق ضمن ملف الوثيقة (Manifest) كما هو موضح في المثال السابق. لاحظ أيضا أن التطبيق السابق يتطلب إصدار أندرويد 13 على الأقل (Android 3.2.1). يفضل استخدام أدنى رقم إصدار يتوافق مع تطبيقك حتى يتمكن أكبر عدد من المستخدمين الاستفادة منه.

التوقيع الرقمي للتطبيق

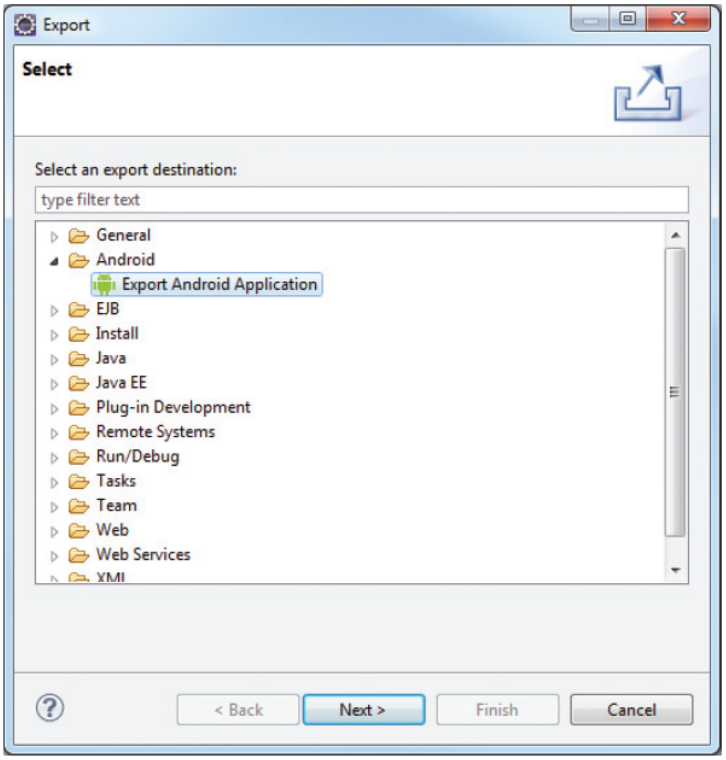
كل تطبيقات الأندرويد يجب توقيعها رقمياً قبل أن يتم تشغيلها على أجهزة أندرويد. توقيع التطبيق رقمياً يضمن ضمان موثوقية التطبيق عن طريق تحديد هوية المطور أو مالك التطبيق. توقيع التطبيق يتطلب ما يسمى بشهادة رقمية (Digital Certificate) تكون بمثابة التوقيع المميز للمطور والذي يتم إلحاقه بالتطبيق عند نشره. يمكن للمطور إنشاء الشهادة الرقمية الخاصة به واستخدامها لتوقيع تطبيقاته.

عند تجربة التطبيقات على برنامج المحاكاة (Emulator) تقوم الواجهة التطبيقية المستخدمة مثل Eclipse بإصدار شهادة للتجربة Debug Certificate ولذلك لا يتطلب أن يقوم المطور بأي خطوات لتوقيع التطبيق رقمياً. أما عند نشر التطبيق، فلا يمكن استخدام الشهادة التجريبية Debug Certificate ويجب إنشاء شهادة جديدة. يمكن إنشاء الشهادة باستخدام الأداة keytool.exe الموجودة ضمن Java SDK. كذلك توفر الواجهة التطبيقية مثل Eclipse امكانية إنشاء الشهادة الرقمية وتوقيع التطبيق بها باستخدام Wizard.

المثال التالي يوضح خطوات تصدير تطبيق أندرويد وتوقيعه رقمياً:

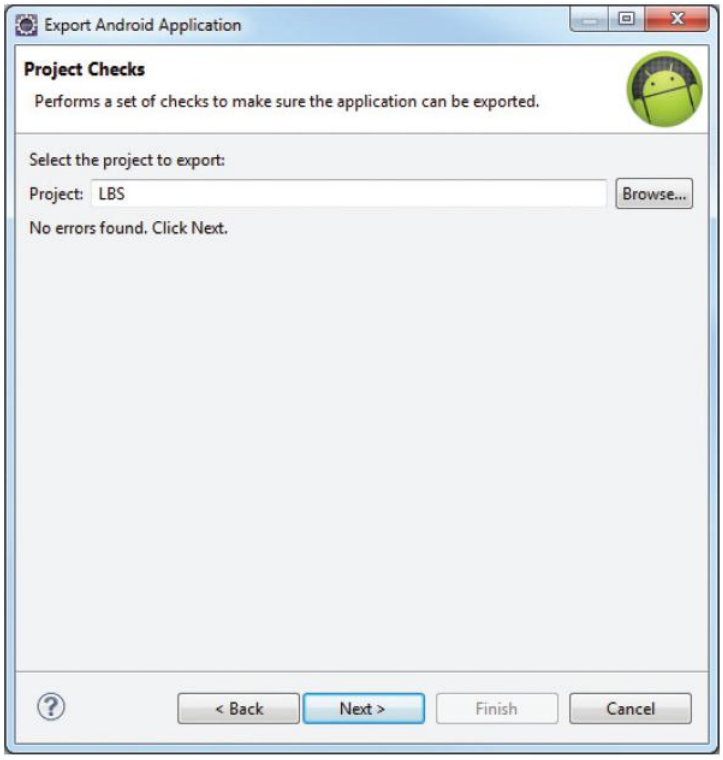
1. اختر التطبيق الذي تريد نشره ومن ثم اختر File -> Export.

ضمن مربع الحوار الذي يظهر افتح القائمة باسم Android وقم بالنقر على الخيار الفرعي Export Android Application كما هو موضح في شكل 10.1. اضغط على الزر Next.



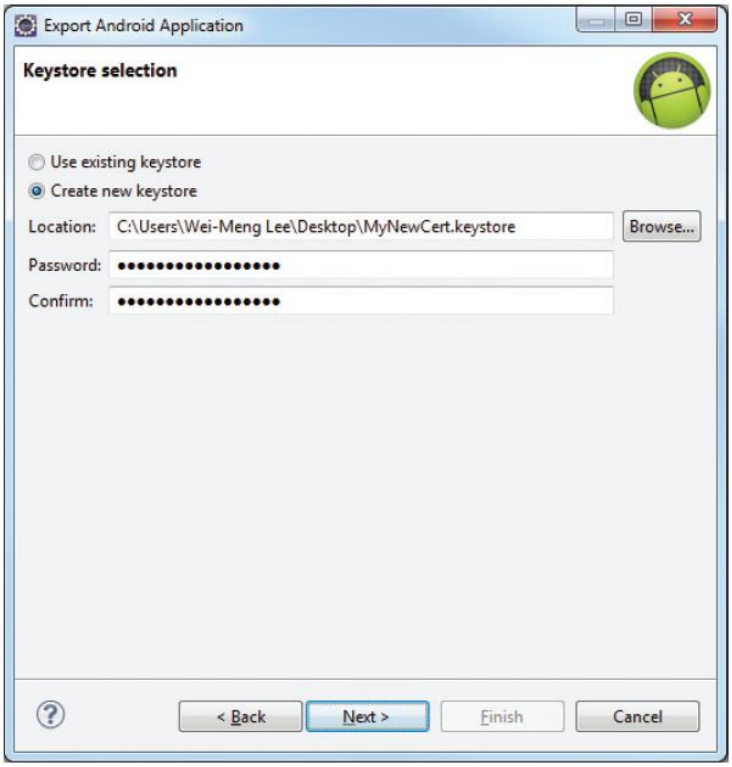
شكل 10.1: تصدير التطبيق من خلال eclipse

يظهر مربع الحوار بالشكل 10.2 والذي يبين اسم التطبيق الذي سيتم تصديره، ومن ثم اضغط على زر Next.



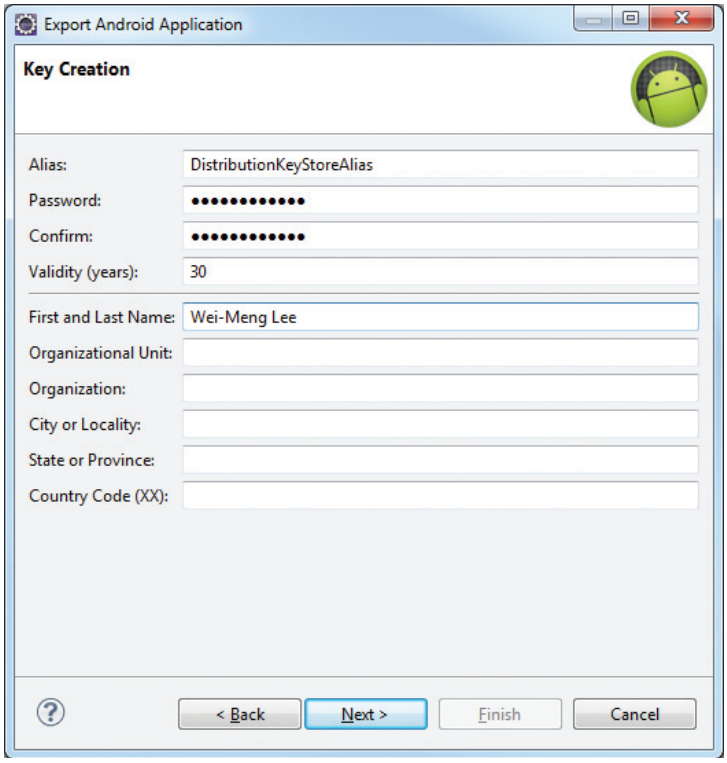
شكل 10.2: اختيار التطبيق ليتم تصديره

ضمن مربع الحوار الذي سيظهر والموضح في شكل 10.3 قم باختيار Create new keystrore وذلك حتى يتم إنشاء شهادة جديدة (Certificate or Keystore) لتوقيع التطبيق، كذلك حدد المسار ليتم حفظ ملف الشهادة به. يجب أيضاً إدخال كلمة مرور لحفظ الشهادة الجديدة حتى لا يتم استخدامها من قبل شخص غير مصرح له. قم بالنقر على زر Next.



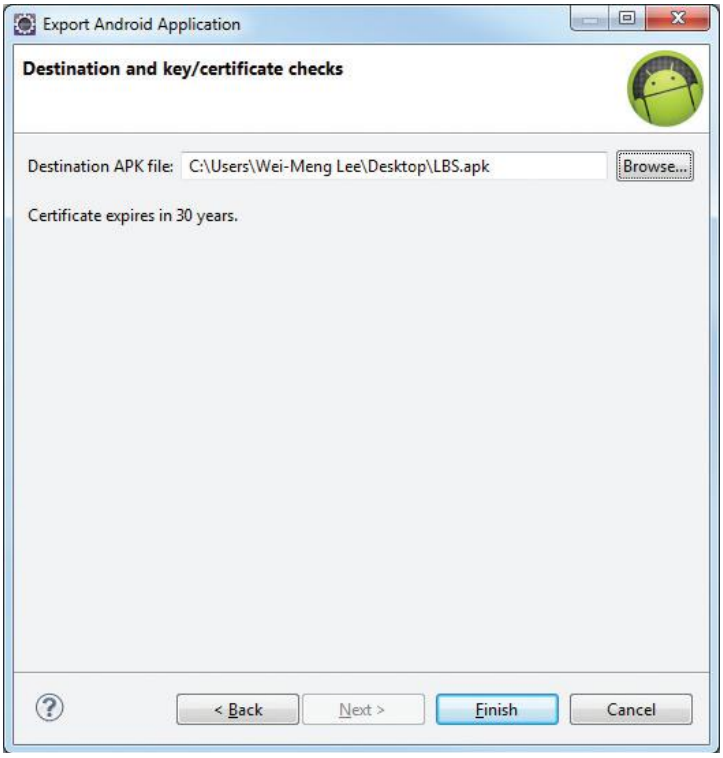
شكل 10.3: إنشاء الشهادة الرقمية لتوقيع التطبيق

ضمن مربع الحوار الذي يظهر (أنظر شكل 10.4) أدخل اسم للمفتاح السري private key المستخدم ضمن الشهادة Certificate كذلك أدخل كلمة مرور لحفظه. كذلك يجب أن تحدد فترة صلاحية المفتاح private key والتي يجب أن تنتهي بعد تاريخ 22-10-2033. لذلك أدخل أي عدد بحيث يكون أكبر من الفرق بين سنة 2033 والسنة الحالية. بعد ذلك أنقر على زر Next.



شكل 10.4: إدخال البيانات الخاصة بالشهادة الرقمية

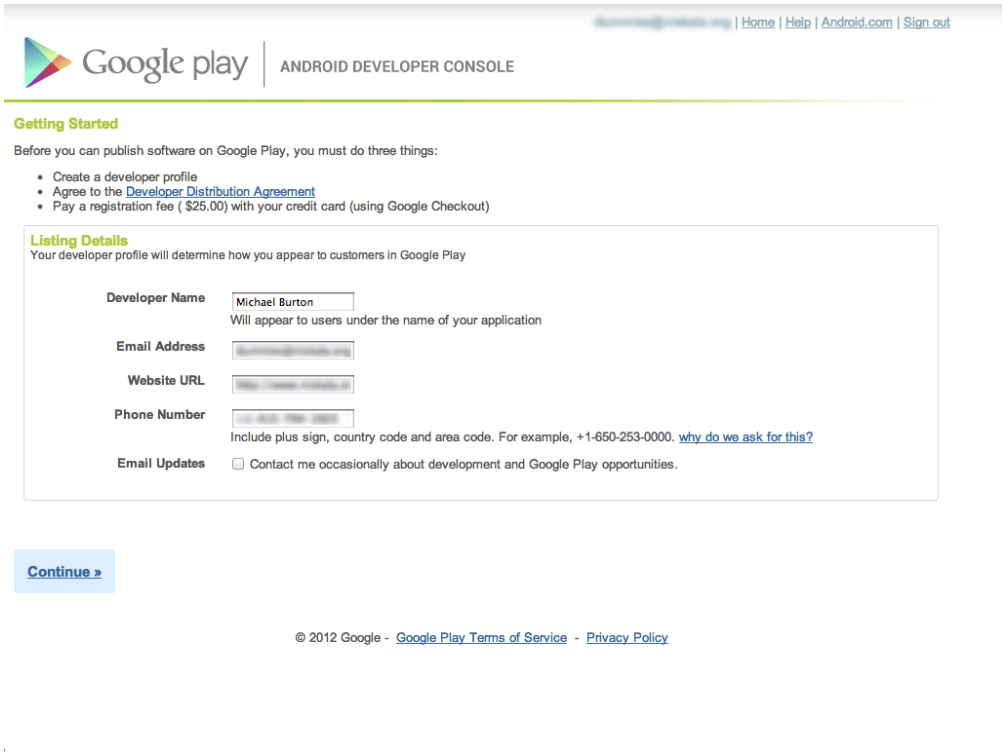
ضمن مربع الحوار الذي يظهر كما في شكل 10.5، حدد المسار الذي يتم فيه حفظ ملف APK، ومن ثم انقر على زر الإنهاء Finish.



شكل 10.5: اختيار المسار لحفظ ملف APK

نشر التطبيق على سوق أندرويد (Android Market)

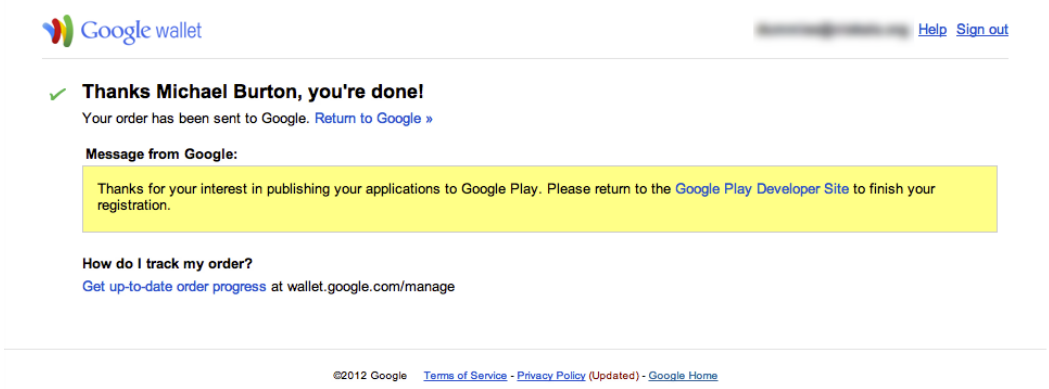
الخطوة الأولى لنشر التطبيق على سوق أندرويد (Android Market) هو إنشاء السجل الخاص بالمطور (Developer Profile) وذلك من خلال الرابط: http://market.android.com/publish/Home. يجب في البداية تسجيل الدخول بحساب جوجل (Google Account)، ومن ثم يمكن إنشاء سجل مطور جديد (Developer Profile) كما هو موضح في شكل 10.6. بعد الانتهاء من إدخال بيانات المطور أنقر على Continue.



شكل 10.6: إنشاء سجل مطور جديد (Developer Profile)

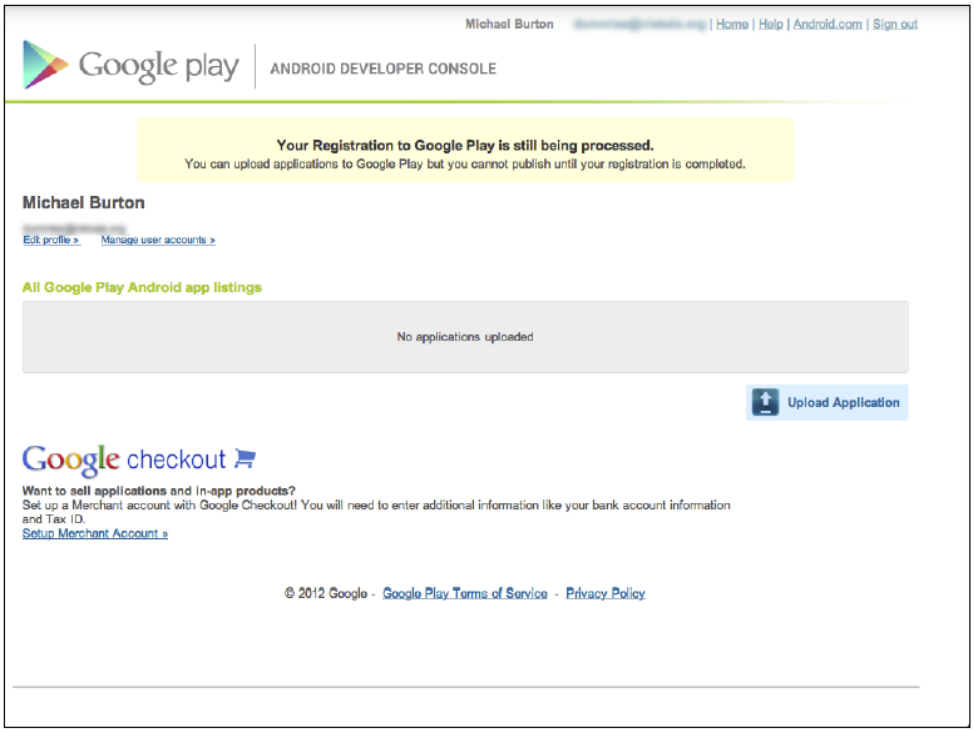
لنشر تطبيقات على سوق أندرويد يجب أن يدفع المطور رسوم تسجيل وتبلغ 25 دولار تدفع لمرة واحدة، ولا يمكن النشر بدون دفع هذه الرسوم. يمكن النقر على زر Google Checkout لدفع رسوم التسجيل. كذلك يجب الموافقة على اتفاقية جوجل الخاصة بالنشر (Distribution agreement) بعد قراءتها وذلك بتحديد مربع الاختيار I Agree.

بعد الانتهاء من إنشاء سجل المطور (Developer Profile) يتم إظهار الرسالة الموضحة في شكل 10.7 والتي تبين أن التسجيل تم بنجاح.



شكل 10.7: شاشة توضح إنشاء سجل المطور (Developer Profile) بنجاح

قم الآن بالنقر على الرابط Google Play Developer Site والذي يؤدي إلى الصفحة الرئيسية للمطور (أنظر شكل 10.8) والتي يمكن من خلالها نشر التطبيقات.



شكل 10.8: الواجهة الرئيسية لصفحة المطور والتي يتم من خلالها نشر التطبيقات

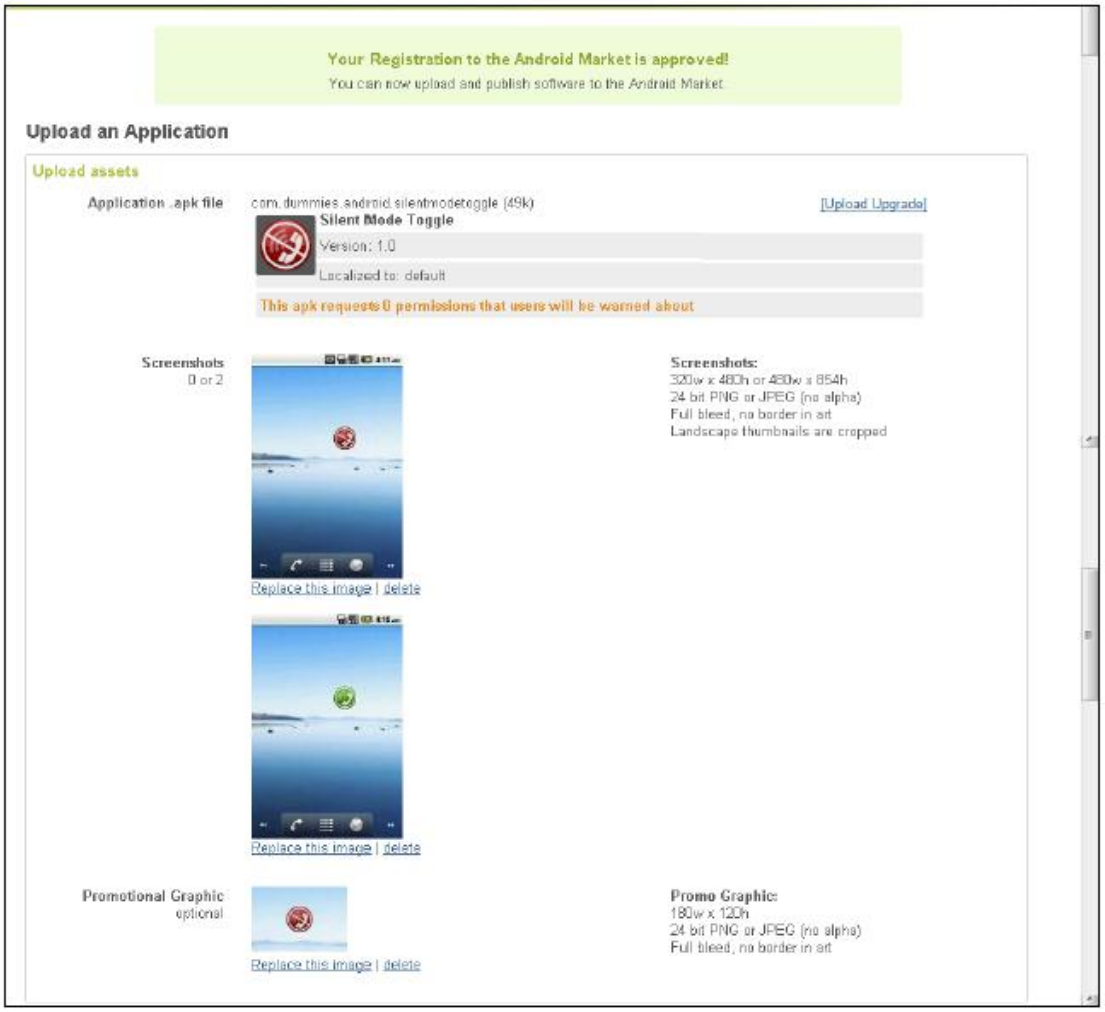
رفع التطبيق

بعد إنشاء سجل المطور (Developer Profile) يمكنك نشر التطبيق على متجر جوجل. تأكد من توفر المكونات التالية:

- ملف APK الخاص بالتطبيق والذي تم توقيعه رقمياً.
- صور (screenshots) توضح واجهات التطبيق (صورتان على الأقل).
- وصف مختصر للتطبيق ووظائفه.

من الصفحة الرئيسية للمطور (أنظر شكل 10.8) قم بالنقر على الرابط Upload Application حيث ستظهر الصفحة في شكل (10.9) والتي من خلالها يتم رفع ملف APK وكذلك ملفات الصور. يجب إيضاً إدخال بعض البيانات الخاصة بالتطبيق مثل عنوان التطبيق ،وصف له، آخر التعديلات على التطبيق (في حالة تعديل تطبيق منشور مسبقاً)، نوع التطبيق وتصنيفه.

بعد نشر التطبيق على متجر جوجل، يمكن تتبع أي تعليقات يرسلها المستخدمون وكذلك أي أخطاء محتملة في التطبيق وعدد مرات تنزيله.



شكل 10.9: الصفحة التي يتم من خلالها رفع ملف التطبيق وملحقاته.

المراجـــع

[1]. Joseph Annuzzi, Lauren Darcey and Shane Conder, "Introduction to Android Application Development: Android Essentials", 4th Edition, Developer's Library, December 6, 2013.

[2]. PawPrints Learning Technologies, "Beginning Android Development: Create Your Own Android Apps", September 25, 2014

[3]. Joseph Annuzzi, Lauren Darcey and Shane Conder, "Advanced Android Application Development", 4th Edition, Developer's Library, November 24, 2014.

[4]. Greg Nudelman, "Android Design Patterns: Interaction Design Solutions for Developers", Wiley, March 11, 2013.

[5]. Neil Smyth, "Android 4.4 App Development Essentials", January 27, 2014.

[6]. Bill Phillips, Brian Hardy, "Android Programming: The Big Nerd Ranch Guide", April 7,2013.

[7]. Wei-Meng Lee, "Beginning Android 4 Application Development", March 6, 2012.

[8]. The Developer's Guide – Android Developers, available on https://developer.android.com/guide/index.html (Last access: January 30, 2015).

[9]. Andoid Development Tutorial by Lars Vogel, available on http://www.vogella.com/tutorials/Android/article.html (Last access: January 30, 2015).