الأكاديمية العربية الدولية
Arab International Academy

لغة البرمجة جافا

JAVA Programming Language

عبدالقادر العبدالله

كلية العلوم – تخصص البرمجة

المحاور

- إنشاء دالة
- المعلمات والوسيطات
- القيم المُعادة من الدالة
- التحميل الزائد للدوال (Method Overloading)
- النطاق (Scope في Java)
- نطاق الدالة
- نطاق الكتلة
- العودية (Recursionفي Java)
- شرط التوقف في العودية
- ما هي البرمجة الكائنية
- ما هي الكلاسات Classes والكائنات Objects؟
- إنشاء كلاس Class في Java
- إنشاء كائن Objectفي Java
- إنشاء عدة كائنات في Java
- استخدام عدة كلاسات في Java
- سمات الكلاس في Java (Java Class Attributes)
- الوصول إلى سمات الكلاس Accessing Attributes
- دوال الكلاس في Java
- الفرق بين static و public فيJava
- الـ Constructorsفي Java
- وسائط المُنشئ Constructor Parameters

المخرجات المتوقعة من الدرس

1. فهم أساسيات البرمجة الكائنية التوجه (OOP) في Java، بما يشمل مفهوم الكائن (Object) والصف(Class)
2. القدرة على إنشاء الأصناف (Classes) والكائنات(Objects)، واستخدامها في كتابة البرامج.
3. معرفة كيفية استخدام السمات (Attributes) والدوال (Methods) داخل الأصناف، والوصول إليها بشكل صحيح.
4. التمييز بين المفاهيم الأساسية مثل static و public، وفهم نطاق المتغيرات والدوال (Scope) .
5. القدرة على كتابة المُنشئات (Constructors) واستخدام الوسائط داخلها بشكل فعال.
6. فهم مفاهيم متقدمة مثل التحميل الزائد للدوال (Method Overloading) والتكرار الذاتي(Recursion).
7. اكتساب مهارات تنظيم الكود وتحديد أماكن المتغيرات والدوال في الكتل المختلفة ضمن البرنامج.

إنشاء دالة

يجب أن يتم **تعريف الدالة داخل فئة (Class)**في لغة Java، ولا يمكن أن تكون مستقلة خارجها.
يتم تعريف الدالة باستخدام **اسم تختاره**، متبوعًا **بأقواس ()**، ويمكن أن تحتوي هذه الأقواس على معلمات (parameters) أو تكون فارغة

Example

Get your own Java Server

Create a method inside Main:

```
public class Main {
  static void myMethod() {
    // code to be executed
  }
}
```

المعلمات والوسيطات

يمكن تمرير المعلومات إلى الدوال (أو الطرائق) من خلال ما يُعرف بـ **المعلمات**، حيث تعمل المعلمة كمتغيّر داخل الدالة يتم استخدامه لمعالجة البيانات المُرسلة.

يتم تحديد المعلمات بعد اسم الدالة، داخل الأقواس (). يمكنك تحديد عدد غير محدود من المعلمات، بشرط أن تفصل بينها باستخدام الفاصلة (،).

Example

Get your own Java Server

```
public class Main {
  static void myMethod(String fname) {
    System.out.println(fname + " Refsnes");
  }

  public static void main(String[] args) {
    myMethod("Liam");
    myMethod("Jenny");
    myMethod("Anja");
  }
}
// Liam Refsnes
// Jenny Refsnes
// Anja Refsnes
```

القيم المُعادة من الدالة

في الصفحة السابقة، استخدمنا الكلمة المفتاحية void في جميع الأمثلة، وهي تُشير إلى أن الدالة **لا تُعيد أي قيمة** عند تنفيذها.
لكن إذا كنت تريد أن تقوم الدالة **بإرجاع (إعادة) قيمة** معينة بعد تنفيذها، فيمكنك استخدام **نوع بيانات بدائي** (primitive data type) مثل:

- int) عدد صحيح
- char) حرف
- double) عدد عشري
- أو غيرها بدلًا من (void).

كما يجب عليك استخدام الكلمة المفتاحية return داخل الدالة لإرجاع القيمة المطلوبة.

القيم المُعادة من الدالة

Example

Get your own Java Server

```
public class Main {
  static int myMethod(int x) {
    return 5 + x;
  }

  public static void main(String[] args) {
    System.out.println(myMethod(3));
  }
}
// Outputs 8 (5 + 3)
```

التحميل الزائد للدوال (Method Overloading)

في Java، يسمح لك **التحميل الزائد للدوال** بأن تُعرّف **أكثر من دالة بنفس الاسم**، ولكن بشرط أن تختلف في **عدد المعاملات** (**Parameters**) أو **نوعها** أو **ترتيبها**.

بمعنى آخر: يمكن أن تحتوي نفس الفئة (class) على عدة دوال تحمل نفس الاسم، لكن كل واحدة منها **تؤدي وظيفة مختلفة** اعتمادًا على ما يتم تمريره لها من معطيات

Example

Get your own Java Server

```
int myMethod(int x)
float myMethod(float x)
double myMethod(double x, double y)
```

التحميل الزائد للدوال (Method Overloading)

في هذا المثال، لدينا **دالتان بنفس الاسم** تقومان بجمع الأرقام، لكن كل دالة تستقبل نوعًا مختلفًا من البيانات (مثل عدد صحيح أو عدد عشري).
هذا مثال عملي على مفهوم **التحميل الزائد للدوال (Method Overloading)**، حيث يمكننا استخدام نفس اسم الدالة مع أنواع مختلفة من المعاملات لتنفيذ عمليات مختلفة.

Example

```
static int plusMethodInt(int x, int y) {
  return x + y;
}

static double plusMethodDouble(double x, double y) {
  return x + y;
}

public static void main(String[] args) {
  int myNum1 = plusMethodInt(8, 5);
  double myNum2 = plusMethodDouble(4.3, 6.26);
  System.out.println("int: " + myNum1);
  System.out.println("double: " + myNum2);
}
```

التحميل الزائد للدوال (Method Overloading)

بدلًا من تعريف دالتين تقومان بنفس المهمة، من الأفضل استخدام التحميل الزائد (Overloading) لدالة واحدة.
عندما تكون لديك دالتان تؤديان **نفس الوظيفة تقريبًا** ولكن تتعاملان مع **أنواع مختلفة من البيانات**، مثل int وdouble ، فمن الأفضل أن **تُحمّل دالة واحدة** بأكثر من شكل (تستخدم نفس الاسم لكن بوسائط مختلفة).
هذا يجعل الكود **أكثر تنظيمًا وأسهل في القراءة والصيانة**

Example

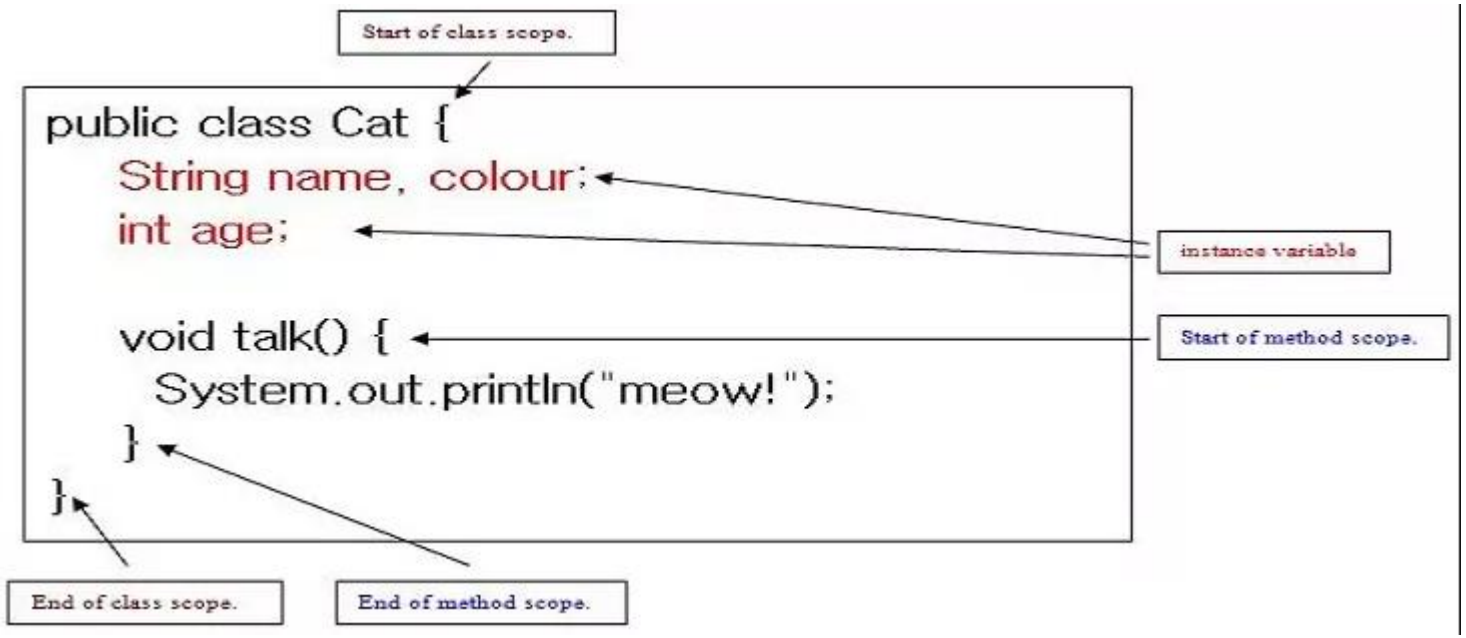
```
static int plusMethod(int x, int y) {
  return x + y;
}

static double plusMethod(double x, double y) {
  return x + y;
}

public static void main(String[] args) {
  int myNum1 = plusMethod(8, 5);
  double myNum2 = plusMethod(4.3, 6.26);
  System.out.println("int: " + myNum1);
  System.out.println("double: " + myNum2);
}
```

النطاق (Scope في Java)

- في لغة Java، لا يمكن الوصول إلى المتغيرات إلا **داخل النطاق الذي تم إنشاؤها فيه**.
هذا المفهوم يُعرف باسم **"النطاق" (Scope)**، ويعني حدود أو مجال ظهور المتغير داخل البرنامج.



نطاق الدالة

المتغيرات التي يتم إعلانها **مباشرة داخل دالة** (method) تكون متاحة (مرئية) **في أي مكان داخل نفس الدالة، بدءًا من السطر الذي تم إعلان المتغير فيه وحتى نهاية الدالة**.

بمعنى آخر: لا يمكنك استخدام المتغير قبل أن يتم تعريفه، لكنه يكون متاحًا **بعد التعريف** داخل كامل الدالة.

Example

```
public class Main {
  public static void main(String[] args) {

    // Code here CANNOT use x

    int x = 100;

    // Code here can use x
    System.out.println(x);
  }
}
```

نطاق الكتلة

في Java ، **الكتلة البرمجية** تشير إلى كل التعليمات التي تُكتب داخل **الأقواس المعقوفة.{}**
مثلًا: الكتل التي تستخدم في الشروطif ، أو الحلقات for وwhile، أو داخل الدوال

ما هو نطاق الكتلة؟

أي **متغير يتم تعريفه داخل كتلة {}** لا يمكن الوصول إليه **إلا من داخل تلك الكتلة فقط،** بدءًا من **السطر الذي تم فيه إعلان المتغير وحتى نهاية القوس.**
.بمعنى آخر: لا يمكنك استخدام المتغير خارج الكتلة التي تم تعريفه فيها

نطاق الكتلة

```
public class Main {
  public static void main(String[] args) {

    // Code here CANNOT use x

    { // This is a block

      // Code here CANNOT use x

      int x = 100;

      // Code here CAN use x
      System.out.println(x);

    } // The block ends here

  // Code here CANNOT use x

  }
}
```

Q1

1.ما الكلمة المفتاحية التي تُستخدم لإنشاء دالة لا تُعيد قيمة؟

2.أي مما يلي يُعد مثالًا صحيحًا لتعريف دالة فيJava ؟

أ) method my void()

ب) public int myMethod()

ج) create function() int

د) define myMethod void()

3.ما هو الهدف من استخدام المعلمات (parameters) في الدالة؟

4.في حالة التحميل الزائد للدوال(Method Overloading) ، ما الذي يجب أن يختلف؟

5.أي من التالي يُمثل "شرط التوقف" في دالة عودية؟

أ) بداية الحلقة

ب) استخدام new

ج) حالة تُنهي تكرار الدالة

د) تعريف متغير جديد

الأجوبة

1. void
2. ب) public int myMethod()
3. لنقل البيانات إلى داخل الدالة
4. نوع أو عدد المعاملات
5. ج) حالة تُنهي تكرار الدالة

العودية (Recursion في Java)

العودية هي تقنية في البرمجة تُستخدم عندما تقوم الدالة **بنداء نفسها** داخل تعريفها.

لماذا نستخدم العودية؟

تُستخدم العودية لتقسيم المشاكل المعقدة إلى **مشاكل أصغر وأسهل في الحل**.
مثلًا: يمكن استخدام العودية لحساب المضروب (Factorial)، أو استعراض الملفات داخل المجلدات، أو حل المسائل التي تتطلب تكرارًا متداخلًا.

```java
public class Main {
  public static void main(String[] args) {
    int result = sum(10);
    System.out.println(result);
  }
  public static int sum(int k) {
    if (k > 0) {
      return k + sum(k - 1);
    } else {
      return 0;
    }
  }
}
```

شرط التوقف في العودية

كما أن الحلقات (loops) قد تقع في مشكلة الحلقة اللانهائية،

فإن الدوال العودية (recursive functions) قد تواجه مشكلة العودية اللانهائية، أي أن الدالة تستمر في نداء نفسها إلى ما لا نهاية هذا يحدث عندما **لا يوجد شرط يحدد متى يجب أن تتوقف الدالة عن نداء نفسها**.

ما هو شرط التوقف؟

شرط التوقف هو حالة محددة نقول فيها للدالة :

توقفي عن نداء نفسك وابدئي بإرجاع النتائج.«

بدون شرط التوقف، ستستمر العودية إلى أن ينهار البرنامج

شرط التوقف في العودية

Example

Use recursion to add all of the numbers between 5 to 10.

```
public class Main {
  public static void main(String[] args) {
    int result = sum(5, 10);
    System.out.println(result);
  }
  public static int sum(int start, int end) {
    if (end > start) {
      return end + sum(start, end - 1);
    } else {
      return end;
    }
  }
}
```

ما هي البرمجة الكائنية

-ما هي OOP؟

OOPهي اختصار لـ **Object-Oriented Programming**، أي **البرمجة كائنية التوجه** أو **البرمجة الشيئية**.

الفرق بين البرمجة الإجرائية Procedural والبرمجة الكائنية OOP :

البرمجة الإجرائية:
تعتمد على كتابة سلسلة من الإجراءات أو الدوال Methods التي تقوم بعمليات على البيانات. البيانات والدوال منفصلان عن بعضهما

البرمجة كائنية التوجه : OOP

تعتمد على إنشاء كائنات (Objects) تحتوي على كل من البيانات (المتغيرات) والوظائف (الدوال) المتعلقة بهذه البيانات.

ما هي البرمجة الكائنية (OOP)

فوائد البرمجة كائنية التوجه (OOP)مقارنة بالبرمجة الإجرائية:

أسرع وأسهل في التنفيذ:
تنظيم الكود ضمن كائنات يجعل التعامل مع البيانات والوظائف أكثر كفاءة.

تُوفر هيكلًا واضحًا للبرامج:
تقسيم المشروع إلى كائنات تسهّل فهمه وتطويره.

تُساعد في جعل الكود أكثر تنظيمًا ومرونة DRY - Don't Repeat Yourself
أي أنك **لا تكرر نفسك**، بل تعيد استخدام نفس الكائنات والدوال، مما يُسهل الصيانة والتعديل والتصحيح.

إمكانية إنشاء تطبيقات قابلة لإعادة الاستخدام Reusable Applications

من خلال إنشاء كائنات يمكن استخدامها في عدة أجزاء من المشروع أو في مشاريع أخرى، مما يقلل من كمية الكود ويُسرّع التطوير.

ما هي الكلاسات Classesوالكائنات Objects ؟

في البرمجة كائنية التوجه OOP، **الكلاسات والكائنات** هما الأساس الذي تُبنى عليه التطبيقات.

ما هو الـ Class الفئة ؟

الكلاس هو **قالب أو نموذج Blueprint**نستخدمه لتعريف شكل الكائن Object
بمعنى آخر، الكلاس يحدد **ما هي البيانات (المتغيرات)** و**ما هي الوظائف (الدوال)** التي يمتلكها الكائن.

ما هو الـ Objectالكائن ؟

الكائن هو **نسخة حقيقية** من الكلاس – كأنك صنعت سيارة فعلية من التصميم الهندسي.
يمكنك إنشاء عدة كائنات من نفس الكلاس، ولكل كائن خصائصه الخاصة.

ما هي الكلاسات Classesوالكائنات Objects ؟

class	objects
Fruit	Apple Banana Mango

Another example:

class	objects
Car	Volvo Audi Toyota

الكلاسات والكائنات في Java

كل شيء في Java مرتبط بالكلاسات والكائنات

في Java، كل عنصر تتعامل معه تقريبًا يكون إما **كائنًا** أو جزءًا من **كلاس**.
كل كائن يحتوي على:

- **خصائص Attributes :** وهي البيانات مثل اللون، الوزن، الاسم...
- **وظائف Methods:** وهي الأفعال أو المهام التي يمكن أن يؤديها الكائن، مثل الطباعة، القيادة، التوقف...

إنشاء كلاس Class في Java

لإنشاء كلاس في Java ، نستخدم الكلمة المفتاحية**class** ، وهي الطريقة الأساسية لتعريف نوع جديد يحتوي على خصائص ودوال .

Main.java

Create a class named " Main " with a variable x:

```
public class Main {
  int x = 5;
}
```

إنشاء كائن Object في Java

في لغةJava ، يتم إنشاء الكائنات من **الكلاسات**.**(Classes)**
بمجرد أن تقوم بإنشاء كلاس (مثلMain) ، يمكنك استخدامه لإنشاء **كائن** يمثل نسخة حقيقية من هذا الكلاس.

- ClassName: اسم الكلاس الذي تريد إنشاء كائن منه.
- objectName: اسم الكائن الذي تختاره.
- new: كلمة مفتاحية تُستخدم لإنشاء كائن جديد.
- ClassName(): هذا يُستدعى المُنشئ (constructor) الخاص بالكلاس.

إنشاء كائن Object في Java

Example

Create an object called " myObj " and print the value of x:

```
public class Main {
  int x = 5;

  public static void main(String[] args) {
    Main myObj = new Main();
    System.out.println(myObj.x);
  }
}
```

Q2

1. ما الفرق الأساسي بين class و object في Java ؟
2. ما وظيفة الكلمة المفتاحية new في Java ؟
3. أي من التالي يُعد سِمة (Attribute) في كائن Java ؟
   أ print()
   ب color
   ج return
   د void
4. ما الذي يُشير إليه استخدام static مع دالة أو سِمة؟
5. كيف يتم الوصول إلى سمة في كائن باستخدام Java ؟

الأجوبة

1. الكلاس هو القالب، والكائن هو نسخة حقيقية منه
2. لإنشاء كائن جديد من كلاس
3. color
4. تنتمي للكلاس نفسه وليس للكائن
5. object.attribut

إنشاء عدة كائنات في Java

يمكنك إنشاء عدة كائنات من نفس الكلاس:

في Java، يمكنك إنشاء **أكثر من كائن Object** من نفس الكلاس Class
كل كائن يتم إنشاؤه يكون **مستقلًا تمامًا** عن الكائنات الأخرى، وله نسخة خاصة به من الخصائص (المتغيرات) والوظائف (الدوال).

Example

Create two objects of `Main`:

```
public class Main {
  int x = 5;

  public static void main(String[] args) {
    Main myObj1 = new Main();  // Object 1
    Main myObj2 = new Main();  // Object 2
    System.out.println(myObj1.x);
    System.out.println(myObj2.x);
  }
}
```

استخدام عدة كلاسات في Java

إنشاء كائن من كلاس معين Classواستخدامه داخل **كلاس آخر**.
هذا الأسلوب يُستخدم كثيرًا من أجل **تنظيم الكود** بشكل أفضل، خصوصًا في المشاريع الكبيرة.

لماذا نستخدم عدة كلاسات؟

- لجعل الكود أكثر تنظيمًا ووضوحًا.
- لتقسيم المهام: كلاس يحتوي على **الخصائص والدوال**، وكلاس آخر يحتوي على **الدالة main()** نقطة التشغيل الرئيسية.
- لتسهيل إعادة استخدام الكود.(Reuse)

Main.java

```
public class Main {
  int x = 5;
}
```

استخدام عدة كلاسات في Java

Second.java

```
class Second {
  public static void main(String[] args) {
    Main myObj = new Main();
    System.out.println(myObj.x);
  }
}
```

استخدام عدة كلاسات في Java

عندما يتم تجميع كلا الملفين:

عند القيام بعملية **التجميع (compilation)** لكلا الملفين (Main.java و Second.java) ، فإن مترجم Java (javac) سيقوم بتحويل كل ملف إلى ملف بصيغة — **class.** أي إلى **كود بايت (bytecode)** يمكن تشغيله على آلة جافا الافتراضية.(JVM)

```
C:\Users\Your Name>javac Main.java
C:\Users\Your Name>javac Second.java
```

استخدام عدة كلاسات في Java

تشغيل ملف Second.java:

في Java ، **لا يمكنك تشغيل أي ملف يحتوي على كلاس عادي فقط** ما لم يكن فيه **دالة** — **main()** وهي نقطة البداية لأي برنامج Java.

```
C:\Users\Your Name>java Second
```

استخدام عدة كلاسات في Java

والناتج سيكون:

```
5
```

Try it Yourself »

سمات الكلاس في Java Java Class Attributes

في الفصل السابق، استخدمنا مصطلح "متغير (variable) "للمتغير x

لكن في الواقع، هذا المتغير x هو **سِمة (Attribute)** من سمات الكلاس.

ما معنى Attribute؟

السِمة Attributeهي ببساطة **متغير يتم تعريفه داخل كلاس (class)**.

يمكننا القول إن **سمات الكلاس هي المتغيرات التي تُمثّل خصائص الكائن object**

Example

Create a class called "`Main`" with two attributes: `x` and `y`:

```
public class Main {
  int x = 5;
  int y = 3;
}
```

الوصول إلى سمات الكلاس Accessing Attributes

إنشاء كائن Object من الكلاس.

ثم استخدام **صيغة النقطة Dot Syntax** لقراءة أو تعديل تلك السِّمات.

Example

Create an object called " `myObj` " and print the value of `x` :

```
public class Main {
  int x = 5;

  public static void main(String[] args) {
    Main myObj = new Main();
    System.out.println(myObj.x);
  }
}
```

Try it Yourself »

دوال الكلاس في Java

الدوال Methods يتم تعريفها داخل **كلاس Class**

تُستخدم هذه الدوال لتنفيذ **إجراءات أو مهام محددة** في البرنامج.

- myMethod() هي **دالة معرفة داخل الكلاس MyClass**
- الكلمة المفتاحية static تعني أنه يمكننا استدعاء هذه الدالة مباشرة من داخل main() **بدون إنشاء كائن**.
- عند استدعاء الدالة، يتم تنفيذ التعليمات داخلها، مثل طباعة الجملة على الشاشة.

Example

Create a method named `myMethod()` in Main:

```
public class Main {
  static void myMethod() {
    System.out.println("Hello World!");
  }
}
```

الفرق بين static و public فيJava

في برامجJava ، غالبًا ما ترى أن **السمات (attributes)** أو **الدوال (methods)** تحمل الكلمتين المفتاحيتين static أو — public وأحيانًا الاثنتين معًا.

– public تعني "عام":

- تستخدم **مستوى الوصول**.(Access Modifier)
- تعني أن **السمة أو الدالة يمكن الوصول إليها من أي مكان في البرنامج**، سواء من داخل نفس الكلاس لتحديدأو من كلاس آخر.

– static تعني "ثابت":

- تعني أن **السمة أو الدالة تنتمي للكلاس نفسه وليس لكائن معين**.
- يمكن استخدامها **بدون إنشاء كائن (object)** من الكلاس.

الفرق بين static و public في Java

```java
public class Main {
  // Static method
  static void myStaticMethod() {
    System.out.println("Static methods can be called without creating objects");
  }

  // Public method
  public void myPublicMethod() {
    System.out.println("Public methods must be called by creating objects");
  }

  // Main method
  public static void main(String[] args) {
    myStaticMethod(); // Call the static method
    // myPublicMethod(); This would compile an error

    Main myObj = new Main(); // Create an object of Main
    myObj.myPublicMethod(); // Call the public method on the object
  }
}
```

الـ Constructors في Java

في لغة Java، الـ **constructor** البنّاء هو **دالة خاصة** داخل الكلاس تُستخدم **لتهيئة الكائنات Objects** عند إنشائها.

ماهو الـConstructor ؟

هو **يشبه الدالة**، لكن اسمه يجب أن يكون **مطابقًا لاسم الكلاس**.

- يتم **استدعاؤه تلقائيًا** عند إنشاء كائن جديد من الكلاس باستخدام new.
- يمكن استخدامه **لتحديد قيم ابتدائية لسمات الكائن** (Attributes)

الـ Constructors في Java

Create a constructor:

```
// Create a Main class
public class Main {
  int x;  // Create a class attribute

  // Create a class constructor for the Main class
  public Main() {
    x = 5;  // Set the initial value for the class attribute x
  }

  public static void main(String[] args) {
    Main myObj = new Main(); // Create an object of class Main (This will call the constructo
    System.out.println(myObj.x); // Print the value of x
  }
}

// Outputs 5
```

وسائط المُنشئ Constructor Parameters

يمكن للـ **constructor** في Javaأن يستقبل **وسائط ((parameters**، وهي مفيدة جدًا لـ **تهيئة السمات attributes** بقيم يتم تحديدها عند إنشاء الكائن.

- بدلاً من تعيين قيم ثابتة داخل الكلاس، يمكنك جعل الـ constructor **مرنًا** ويقبل **قيمًا من المستخدم أو من الكود** عند إنشاء الكائن.
- ثم تقوم بإسناد هذه القيم إلى السمات داخل الكلاس

وسائط المُنشئ Constructor Parameters

Example

```
public class Main {
  int x;

  public Main(int y) {
    x = y;
  }

  public static void main(String[] args) {
    Main myObj = new Main(5);
    System.out.println(myObj.x);
  }
}

// Outputs 5
```

Q3

1. **ما هو الـ Constructor في Java ؟**

2. **ما الهدف من تمرير وسائط إلى Constructor ؟**

3. **أي من التالي صحيح عند تعريف Constructor ؟**
   أ) يستخدم الكلمة المفتاحية void
   ب) يُكتب باسمه main
   ج) لا يحتوي على اسم
   د) يجب أن يطابق اسمه اسم الكلاس
4. **ما هي فائدة استخدام عدة كلاسات في برنامج Java ؟**

5. **ما الذي يحدث عند تجميع ملفي Java يحتويان على كلاسين؟**

الأجوبة

1. دالة خاصة لها نفس اسم الكلاس وتُستخدم لتهيئة الكائن
2. لإعطاء قيم أولية لسمات الكائن
3. يجب أن يطابق اسمه اسم الكلاس
4. تنظيم المشروع وتسهيل إعادة الاستخدام
5. يتم تحويل كل ملف إلى ملف Bytecode مستقل

المصادر

تم الإعتماد على W3SCOOLS في تجهيز هذه المحاضرة

https://www.w3schools.com/java/java_methods.asp

شكرا لكم