الجمهورية العربية السورية
جامعة حمص
الكلية التطبيقية
قسم تقانات الحاسوب
السنة الرابعة

الفصل الاول
2026-2025

هندسة برمجيات متقدمة – القسم العملي

Class in Java

المحاضرة السادسة

مدرسو المقرر

م.روز المشرقي
م.جنان الكردي

مقدمة:

يعتمد مفهوم البرمجة غرضيّة التوجه على الفكرة الأساسية التي تقول بأن كل البرامج هي محاكاة حاسوبية لأغراض وأشياء من العالم الحقيقي أو لمفاهيم مجرّدة في هذا العالم.

- تتمتع الأغراض بصفات أساسية هي:

- لكل غرض خصوصية وهوية identity.
- لكل غرض متحولات تعكس حالة الغرض data fields.
- لكل غرض بعض طرائق المعالجة التي تعبر عن سلوكه method.

كل غرض ينتمي إلى شكل من أشكال البيانات يدعى بالصف ونقول إن الغرض هو شبيه صفه ويشكل مثالاً له.

الصفوف:

يتألف الصف من مجموعة حقول البيانات ومجموعة الطرائق المتعلقة بالصف، إضافة إلى البواني وهي تحمل اسم الصف نفسه، وتقوم بتشييد وبناء الأغراض من الصف، وجميعها يمكن أن تعرّف بأنها عامة أو خاصة أو محمية (public – private -protected).

مثال:

```
package classes;

public class Address{

private String street;
private int buildingNumber;

public Address() {   }
    public Address(String street, int n)
     {     this.street = street;    this.buildingNumber = n; }
```

تعريف الصف

تعريف المتغيرات

تعريف البواني

```
public String getStreet() { return street;    }
public void setStreet(String street) {   this.street = street;  }
public int getBuildingNumber() {   return buildingNumber;  }
public void setBuildingNmumber(int buildingNumber)
{    this.buildingNumber = buildingNumber;   }

@Override
public String toString() {  return street + " " + buildingNumber;  }
```

ملاحظة هامة:

لا يمكن استدعاء الباني الافتراضي عند وجود بوانٍ أخرى في الصف، وإلا لابد من تعريفه بدون وسطاء في هذه الحالة من أجل استخدامه.

```
package lab_4;
import classes. Address;


public class ClassTest {
    public static void main(String[] args) {
        Address ad1 = new Address("ALmhta", 30);
        System.out.println(ad1.getStreet());
        ad1.setBuildingNmumber(45);
        System.out.println(ad1.toString());
        }
```

الوراثة(Inheritance):

تعرف بعلاقة "is a" ، تتم عملية الوراثة بين الصفوف من خلال اشتقاق صف من صف آخر، حيث يرث الصف المشتق (الصف الابن) (Sub Class) جميع محتويات الصف المشتق منه (Super Class) باستثناء المحتويات المعرفة على أنها خاصة (private). تتم عملية اشتقاق الصفوف من بعضها باستخدام الكلمة المحجوزة extends.

ملاحظة: لا تدعم جافا الوراثة المتعددة.

مثال:

نتابع على المثال السابق (مثال تركيب الصفوف في المحاضرة السابقة)، تقسم الشركة الأشخاص إلى موظف ذو أجر ساعي وموظف ذو راتب ثابت حسب الفئة الوظيفية. أكتب الصفوف اللازمة للمسألة، مع اجراء التعديلات المناسبة للصفوف السابقة

```java
public class Employee {
    protected String name;
    protected int id;

    public Employee(String name, int id) {
        this.name = name;
        this.id = id;
    }

    public void displayInfo() {
        System.out.println("name: " + name);
        System.out.println(" id: " + id);
    }
}
```

```java
public class SalariedEmployee extends Employee {
    private double monthlySalary;

    public SalariedEmployee(String name, int id, double monthlySalary) {
        super(name, id);
        this.monthlySalary = monthlySalary;
    }

    public double calculateSalary() {
        return monthlySalary;
    }

    @Override
    public void displayInfo() {
        super.displayInfo();
        System.out.println("نوع الموظف: راتب ثابت");
        System.out.println("الراتب الشهري: " + calculateSalary());
    }
}
```

```java
public class HourlyEmployee extends Employee {
    private double hourlyRate;
    private int hoursWorked;

    public HourlyEmployee(String name, int id, double hourlyRate, int hoursWorked) {
        super(name, id);
        this.hourlyRate = hourlyRate;
        this.hoursWorked = hoursWorked;
    }

    public double calculateSalary() {
        return hourlyRate * hoursWorked;
    }

    @Override
    public void displayInfo() {
        super.displayInfo();
        System.out.println("نوع الموظف: أجر ساعي");
        System.out.println("الراتب: " + calculateSalary());
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        SalariedEmployee emp1 = new SalariedEmployee("Ahmed", 101, 3000);

        HourlyEmployee emp2 =new HourlyEmployee("Rama", 102, 20, 160);

        emp1.displayInfo();
        System.out.println("-------------------");
        emp2.displayInfo();
    }
}
```

ما هو الـ Abstraction؟

إذا بحثنا في القاموس عن معنى كلمة (Abstract-تجريد-)، فسنجد انها تعني [خاصية التعامل مع الفكرة لا الحدث]. بمعنى إهمال التفاصيل الغير لازمة واستبدالها بما هو مهم وواضح.

مثال على ذلك: عندما تحاول إرسال ايميل الى شخص ما، فإنك لن تهتم بالتفاصيل الصغيرة مثل: ما الذي يحدث بالضبط عندما تضغط على زر إرسال أو ما البروتوكول المستخدم لنقل الرسالة، كل ما تريد عمله هو أن تكتب عنوان الرسالة والمحتوى ومستقبل الرسالة وترسلها .

نفس الشيء ينطبق على مفاهيم الـ Object-Oriented في الـ abstraction نهدف الى إخفاء تفاصيل الـ implementation عن المستخدم.

في جافا، يمكننا تطبيق هذا المفهوم عن طريق إنشاء abstract class

الصفوف المجردة (Abstract Classes):

- يعتبر الصف الذي يحتوي على كلمة abstract في تعريفه abstract class، ويحقق ما يلي:
- من الممكن للـ Abstract class أن يحتوي على abstract methods (ليس إجبارياً)، وهي الطريقة التي لا تحتوي على تعريف مثل

```
public abstract void get( )
```

- إذا احتوى الصف على abstract method ، فيجب تعريفه كـ abstract class .
- اذا تم تعريف الصف كـ abstract class ، فلا يمكن إنشاء objects منه.
- لاستخدام الـ abstract class وخصائصه، يجب عليك أن ترثه (inherit it) من صف آخر،بحيث يقوم الصف الابن بتحقيق كل الـ abstract methods و إلا سيكون الصف الابن مجرد أيضاً.

مثال: اكتب صفاً مجرداً يعبر عن الأشكال الهندسية يحمل الاسم Shape.

```
public abstract class Shape {
   protected  String color;
   public Shape(String c){ color=c;}
   public String get(){return color;}
   public void set(String c){color=c;}
}
```

- abstract class لا يختلف عن الصف العادي، فهو يحتوي على attributes ، constructor ، methods الفرق الوحيد هو كلمة abstract بالتعريف.
- الآن لنحاول إنشاء غرض من الصف السابق:

```
public class Test {
    public static void main(String[] args)
      {  Shape s =new Shape("yallow") ;   }
}
```

عند تشغيل البرنامج سنحصل على الخطأ التالي:

Shape is abstract; cannot be instantiated

ملاحظة هامة: لا يمكن إنشاء غرض من abstract class حتى وإن احتوى على constructor

وراثة الصفوف المجردة (Inheriting Abstract Classes)

نستطيع إعادة استخدام خصائص وطرائق الـ abstract class كأي صف آخر عن طريق وراثته.

مثال: عرف الصف Circle كصف ابن للصف Shape

```
public class Circle extends Shape {
    private int x, y, r;
    public Circle(String color, int x, int y, int r)
    {   super(color);
        this.x = x;       this.y = y;       this.r = r;    }
    public int getx() {    return x;    }
    public int gety() {    return y;    }
    public int getr() {    return r;  }
    public void setx(int x) {   this.x = x;    }
    public void sety(int y) {   this.y = y;    }
    public void setr(int r) {    this.r = r;   }
}
```

- لا يمكننا إنشاء غرض من صف Shape، ولكن يمكننا إنشاء غرض من صف Circle واستخدام جميع خصائص وطرائق الصف Shape

```
public class Test {
    public static void main(String[] args) {
        Circle c=new Circle("red", 2, 3, 15);
        Shape s =new Circle("yellow",0,0,5) ;
        System.out.println("circle color= "+ c.get()+"\tcircle center ("+
                c.getx()+","+c.gety()+")"+" radius= " + c.getr() );
    System.out.println("Shape color= "+ s.get());
}
```

Output:

circle color= red circle center (2,3) radius= 15

Shape color= yellow

واجهات الصفوف (Interfaces):

تقوم واجهة الصف بتوصيف بعض النواحي للصف غير تلك التي يتم وراثتها من الصف الأب، وكل واجهة صف تتضمن ثوابت وطرائق (تصريحاً فقط دون أي جسم)، لذلك يطلب من الصف الذي يحقق هذه الواجهة تعريف أجسام هذه الطرائق.

- الشكل العام لواجهة الصف:

```
interface  interfaceName
{
        Constant definitions
       Method declarations (without implementations)
}
```

- لو طلب من صف تنفيذ واجهة صفية فيجب الإشارة إلى ذلك في عنوان الصف بالكلمة **implements**

```
class  className implements interfaceName
```

مثال:

```
interface tax{
    double x=0.05;
    double calc ();
}

class Book implements tax
{
   String name;
   double price;
   public Book(String name, double price){ this.name=name; this. Price=price ;}
   @Override
   public double calc()  {   return price*x;   }
   public void printInfo(){
    System.out.println("name is "+ name+" price is "+price+" tax value is"+calc());
```

```
public class InterfaceTest {
    public static void main(String[] args) {
        Book b=new Book("Java",500);
        Tax t=new Book("SQL", 900);
        System.out.println(b.calc());
        System.out.println(t.calc());
        b.printInfo();
        ((Book)t).printInfo();
    }
}
```

في الواجهات الصفية:

- جميع أعضاء البيانات هي public static final حتى لو لم يذكر ذلك صراحةً .
- جميع الطرائق هي public abstract حتى لو لم يذكر ذلك صراحةً .

التعددية الشكلية (Polymorphism)

التعددية الشكلية (Polymorphism) في Java هي أحد مبادئ البرمجة كائنية التوجه (OOP)، وتعني قدرة الكائن على اتخاذ أكثر من شكل، أي أن نفس الدالة (method) يمكن أن تتصرف بطرق مختلفة حسب الكائن الذي يستدعيها.

أولاً: لماذا نستخدم التعددية الشكلية؟

تسهّل إعادة استخدام الكود

تجعل البرامج مرنة وقابلة للتوسعة

تقلل الاعتماد المباشر بين الأصناف (Loose Coupling)

أنواع التعددية الشكلية في Java

في Java يوجد نوعان رئيسيان:

1. التعددية الشكلية وقت الترجمة (Compile-time Polymorphism)

👉 عبر زيادة التحميل (Method Overloading)

2. التعددية الشكلية وقت التشغيل (Runtime Polymorphism)

👉 عبر إعادة التعريف (Method Overriding) والوراثة

1. التعددية الشكلية وقت الترجمة (Method Overloading)

تعني وجود عدة دوال بنفس الاسم ولكن: تختلف في عدد المعاملات أو نوع المعاملات

```java
class Calculator {
    int add(int a, int b) {
        return a + b;
    }

    double add(double a, double b) {
        return a + b;
    }

    int add(int a, int b, int c) {
        return a + b + c;
    }
}
```

الاستخدام:

```
1   Calculator calc = new Calculator();
2
3   System.out.println(calc.add(5, 3));         // 8
4   System.out.println(calc.add(2.5, 1.5));     // 4.0
5   System.out.println(calc.add(1, 2, 3));      // 6
6
```

التعددية الشكلية وقت التشغيل (Method Overriding)

تحدث عند: وجود وراثة عند قيام الصنف الابن بإعادة تعريف دالة موجودة في الصنف الأب

مثال أساسي:

```java
class Animal {
    void sound() {
        System.out.println("Animal makes a sound");
    }
}

class Dog extends Animal {
    @Override
    void sound() {
        System.out.println("Dog barks");
    }
}

class Cat extends Animal {
    @Override
    void sound() {
        System.out.println("Cat meows");
    }
}
```

التعددية الشكلية هنا:

```
1    Animal a1 = new Dog();
2    Animal a2 = new Cat();
3
4    a1.sound(); // Dog barks
5    a2.sound(); // Cat meows
6
```

مثال عملي : تطبيق أشكال هندسية

```
2   class Shape {
3       double area() {
4           return 0;
5       }
6   }
7
8   class Rectangle extends Shape {
9       double length = 5, width = 4;
10
11      @Override
12      double area() {
13          return length * width;
14      }
15  }
```

```java
class Circle extends Shape {
    double radius = 3;

    @Override
    double area() {
        return Math.PI * radius * radius;
    }
}

public class Main {
    public static void main(String[] args) {
        Shape s1 = new Rectangle();
        Shape s2 = new Circle();

        System.out.println(s1.area()); // 20
        System.out.println(s2.area()); // 28.27
    }
}
```