محتوى اليوم الرابع

فهرس المواضيع | TOC

تنقسم OOP الى اربع مفاهيم أساسية :

- **التغليف Encapsulation**
- **الوراثة Inheritance**
- **اشكال متعددة Polymorphism**
- **التجريد Abstraction**

التغليف Encapsulation

معنى التغليف هو التأكد من إخفاء البيانات "الحساسة" عن المستخدمين. لتحقيق ذلك ، يجب عليك:

- تعريف Attributes الخاصة بالكلاس على أنها private
- توفير طرق الحصول العامة وتعيينها للوصول إلى قيمة المتغير الخاص وتحديثها

تعرفنا في السابق على Modifiers ومن اجل تحقيق مفهوم Encapsulation سنحاج الى جعل جميل Attributes من نوع private كم في المثال التالي

```
class Animal{
   private String name;
   private int age;

   Animal(String name, int age) {
       this.name = name;
```

```
        this.age = age;
    }
}
```

ثم سوف نقوم بتوفير وسيلة للحصول على طريقة للوصول للعناصر الخاصة بهذا الكلاس وتسمى هذه الطريقة ب Getters/Setters كما في المثال التالي:

```
class Animal{
   private String name;
   private int age;

   Animal(String name, int age) {
       this.name = name;
       this.age = age;
   }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

- دوال getters
  - قمنا بتعريف getName و وظيفتها هي الحصول على قيمة name
  - قمنا بتعريف getAge و وظيفتها هي الحصول على قيمة age
- دوال setters
  - قمنا بتعريف setName و وظيفتها هي تغيير قيمة name
  - قمنا بتعريف setAge و وظيفتها هي تغيير قيمة age

من المهم ان يكون Modifiers الخاص بهذه الدوال من نوع public لنتمكن من الوصول له من اي موقع في مشروعنا

الوراثة Inheritance

في Java ، من الممكن أن ترث Attributes و methods من كلاس إلى كلاس آخر.

مفهوم Inheritance يندرج تحته ثلاث مفاهيم اساسية

- الكلاس الفرعي subclass (الطفل) - الكلاس الذي يرث من كلاس آخر
- الكلاس الأساسي superclass (الأصل) - الكلاس الموروث منه
- لنرث صفات كلاس نستخدم كلمة `extends`

في المثال أدناه ، ترث كلاسCar (الكلاس الفرعي) attributes و methods من فئةVehicle (الكلاس الأساسي):

```
class Vehicle{
    void moving(){
        System.out.println("The Vehicle is moving");
    }
}

class Car extends  Vehicle{
}
```

كما في المثال الأعلى يمكننا استخدام جميع methods الخاصة بكلاس Vehicle عند انشاء object من نوع Car

```
public class Main {
    public static void main(String[] args) {
    Car car1 = new Car();
    car1.moving();
    }
```

```
}
```

المخرج

```
The Vehicle is moving
```

رغم اننا لم نعرف اي method بإسم moving في الكلاس الخاص ب Car لكننا استطعنا استخدام دالة moving و السبب لأنها موروثة من كلاس Vehicle ونستطيع استخدامها.

استخدام super

عند وجود Constructor في الكلاس الأساسي و اردنا وراثة هذا الكلاس فإننا سنحتاج الى استخدام كلمة سوبر في Constructor الخاص بالكلاس الفرعي كما في المثال التالي

```
class Vehicle{
    String speed;
    int yearOfManufacture;
    public Vehicle(String speed, int yearOfManufacture) {
        this.speed = speed;
        this.yearOfManufacture = yearOfManufacture;
    }

    void moving(){
        System.out.println("The Vehicle is moving");
    }
}

class Car extends  Vehicle{
    String company;
    public Car(String speed, int yearOfManufacture, String com
pony) {
        super(speed, yearOfManufacture);
        this.company = compony;
```

```
    }
}
```

يمكننا بإستخدام super اسناد القيم الخاصة attributes الكلاس الأساسي في الكلاس الفرعي

اشكال متعددة Polymorphism

مفهوم Polymorphism او تعدد الأشكال، ويحدث عندما يكون لدينا العديد من الكلاسات التي ترتبط ببعضها البعض عن طريق الوراثة.

كما حددنا في الفصل السابق؛ تتيح لنا الوراثة وراثة Attributes و methods من الكلاس الاساسي الى الكلاس الفرعي. يستخدم Polymorphism هذه methods لأداء مهام مختلفة. هذا يسمح لنا بأداء عمل واحد بطرق مختلفة.

على سبيل المثال ، فكر في كلاس اساسي اسمه Animal و يحتوي method تسمى`animalSound()`. يمكن أن تكون الكلاسات الفرعية من كلاس Animal عبارة عن كلاس Cat و كلاس Dog ولديهم أيضًا method خاصة بنفس الاسم تسمى`animalSound()`:

```
class Animal {
    public void animalSound() {
        System.out.println("The animal makes a sound");
    }
}

class Dog extends Animal {
    @Override
    public void animalSound() {
        System.out.println("The dog says: bow wow");
    }
}
class Cat extends Animal {
    @Override
    public void animalSound() {
        System.out.println("The dog says: meow meow");
```

```
    }
}
```

يتم استخدام `@Override` للأشارة ان هذه method موجودة مسبقا في الكلاس الأساسي و لكن تم تغييرها بداخل الكلاس الفرعي.

تكمن قوة مفهوم Polymorphism عند تعريف object لهذه الأنواع كما في المثال التالي:

```
public class Main {
    public static void main(String[] args) {
        Animal animal = new Animal();
        Animal cat = new Cat();
        Animal dog = new Dog();
        animal.animalSound();
        cat.animalSound();
        dog.animalSound();
    }
}
```

قمنا بتعريف ٣ متغيرات من نوع Animal و لكن القيم المعطى لها تختلف

- في المتغير الأول تم اسناد له قيمة من نوع Animal
- في المتغير الأول تم اسناد له قيمة من نوع Cat
- في المتغير الأول تم اسناد له قيمة من نوع Dog

وهنا تم استخدام مفهوم Polymorphism بحيث انه نوع Animal يقبل اشكال متعددة للقيم و لكن يجب ان تكون كل القيم عبارة عن كلاسات فرعية لكلاس نوع المتغير.

عند القيام بمناداة دالة `animalSound()` سوف نلاحظ اختلاف القيمة المطبوعة و السبب يرجع الى نوع القيمة المسندة لكل متغير.

```
The animal makes a sound
The dog says: meow meow
The dog says: bow wow
```

التجريد Abstraction

مفهوم التجريد هو اخفاء البيانات الخاصة بشكل كامل و اظهار method فقط, يتم ذلك عن طريق استخدام احدى هذين الفهومين `interface` او `abstract classes`.

استخدام `abstract classes`

- مفهوم `abstract class` : هي كلاس لا يمكن استخدامه لإنشاء `objects` (للوصول إليها ، يجب أن يتم وراثته الى كلاسات فرعية).
- مفهوم `abstract method`: هي `method` فارغة يتم تعريفها في `abstract class` و لكن يجب كتابة الكود الخاص فيها في الكلاس الفرعي الذي يرث `abstract class`

في المثال التالي تم تعريف `abstract class` بإسم Animal وتم تعريف `abstract method` بإسم moving و نلاحظ انها فارغة و لكن تم تعريف هذه `method` في الكلاسات التي ترث من `abstract class` Animl

```
abstract class Animal {
    public void animalSound() {
        System.out.println("The animal makes a sound");
    }
    abstract void moving();
}

class Dog extends Animal {
    @Override
    void moving() {
        System.out.println("The Dog is Moving");
    }
}
class Cat extends Animal {
    @Override
    void moving() {
        System.out.println("The Cat is Moving");
    }
}
```

يتم استخدام `abstract class` بنفس الطريقة الموجودة في Polymorphism و لكن يجب ان ننتبه انه لا يمكن تعريفه بإستخدام نوع `abstract class`

```
public class Main {
    public static void main(String[] args) {
        //Animal animal = new Animal(); this Wrong You can't d
eclare it using the abstract class type
        Animal cat = new Cat();
        Animal dog = new Dog();
        cat.moving();
        dog.moving();
    }
}
```

المخرج

```
The Cat is Moving
The Dog is Moving
```

استخدام `interface`

الفرق الجوهري بين **`interface`** و `abstract class` ان **`interface`** هي عبارة عن **`abstract`** **`class`** يحتوي فقط على method فارغة.

يتم تعريفه بكتابة كلمة `interface` ثم كتابة اسمه ثم كتابة methods فارغة

كما في المثال التالي, تم تغيير `abstract class` السابق الى نوع **`interface`** و كتابة method فارغة بشكل طبيعي, لوراثة او استخدام interface نقوم بكتابة كلمة implements في الكلاس الفرعي (يمكن استخدام اكثر من **`interface`** في الكلاس)

```
interface Animal {
     void moving();
}

class Dog implements Animal {
    @Override
    public void moving() {
        System.out.println("The Dog is Moving");
```

```
    }
}
class Cat implements Animal {
    @Override
    public void moving() {
        System.out.println("The Cat is Moving");
    }
}
```

يتم استخدام `interface` بنفس الطريقة الموجودة في `Polymorphism` و لكن يجب ان ننتبه انه لا يمكن تعريفه بإستخدام نوع `interface`

```
public class Main {
    public static void main(String[] args) {
//Animal animal = new Animal(); Wrong You can't declare it usi
ng the interface type
        Animal cat = new Cat();
        Animal dog = new Dog();
        cat.moving();
        dog.moving();
    }
}
```

المخرج

```
The Cat is Moving
The Dog is Moving
```

مفهوم Arraylist

هي مكتبة لـ `Array` متغيرة الحجم من الممكن ان نجدها داخل مجلد `java.util`

الاختلاف الرئيسي بين `Array` و `ArrayList` في Java هو أن حجم `Array` لا يمكن تعديله (إذا كنت تريد إضافة عناصر إلى / من `Array` أو إزالتها منها ، فيجب عليك إنشاء واحدة جديدة). بينما يمكن إضافة العناصر وإزالتها من `ArrayList` وقتما تشاء.

اهم الـ `Method` في الـ **Arraylist :**

دالة **add**

تستعمل لإضافة عنصر الى الـ Arraylist

مثال :

```
import java.util.ArrayList;

public class Main {
  public static void main(String[] args) {
    ArrayList<String> users = new ArrayList<String>();
    users.add("Saleh");
    users.add("Ali");
    users.add("Khalid");

    System.out.println(users);
  }
}
```

دالة **get**

تستعمل للإستعلام عن عنصر الى الـ Arraylist

مثال :

```
    System.out.println(users.get(0)); // Saleh
```

دالة **remove**

تستعمل لحذف عنصر من الـ Arraylist

مثال :

```
    users.remove(0);
    System.out.println(users); // [ Ali , Khalid ]
```

دالة **size**

تستعمل للإستعلام عن عدد العناصر داخل الـ Arraylist

مثال :

```
  System.out.println(users.size()); // 2
```

مفهوم **Scanner**

مفهوم الـ `Scanner` هو مكتبة مستخدمة لقرائة مدخل من المستخدم ويتواجد داخل مجلد `java.util` .

لإستعمال الـ `Scanner` نقوم في البداية بإنشاء Object من Class Scanner كما هو موضح في المثال التالي :

```
import java.util.Scanner;  // Import the Scanner class
```

```
class Main {
  public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in);  // Create a Scann
er object
  }
}
```

ملاحظة : System.in تعني إستقبال المدخل عن طريق الـ CLI .

ثم بعد ذلك نحدد نوع المدخل الذي نريد قرائتة بحسب الدوال الموجودة داخل الجدول التالي :

Description	Method
Reads a `boolean` value from the user	`nextBoolean()`
Reads a `byte` value from the user	`nextByte()`
Reads a `double` value from the user	`nextDouble()`
Reads a `float` value from the user	`nextFloat()`
Reads a `int` value from the user	`nextInt()`
Reads a `String` value from the user	`nextLine()`
Reads a `long` value from the user	`nextLong()`
Reads a `short` value from the user	`nextShort()`

مثال شامل :

```
import java.util.Scanner;
```

```
class Main {
  public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in);

    System.out.println("Enter name, age and salary:");

    // String input
    String name = myObj.nextLine();

    // Numerical input
    int age = myObj.nextInt();
    double salary = myObj.nextDouble();

    // Output input by user
    System.out.println("Name: " + name);
    System.out.println("Age: " + age);
    System.out.println("Salary: " + salary);
  }
}
```

مفهوم Enum

الـ Enum هو "فئة" خاصة من الـ `Class` تمثل مجموعة من الثوابت (المتغيرات غير القابلة للتغيير ، مثل المتغيرات النهائية `final`).

مثال :

```
enum Days {
        MONDAY,
        TUESDAY,
        WEDNESDAY,
        THURSDAY,
        FRIDAY,
        SATURDAY,
        SUNDAY
}

public static void main(String[] args) {

        System.out.println(Days.MONDAY);

    }
```