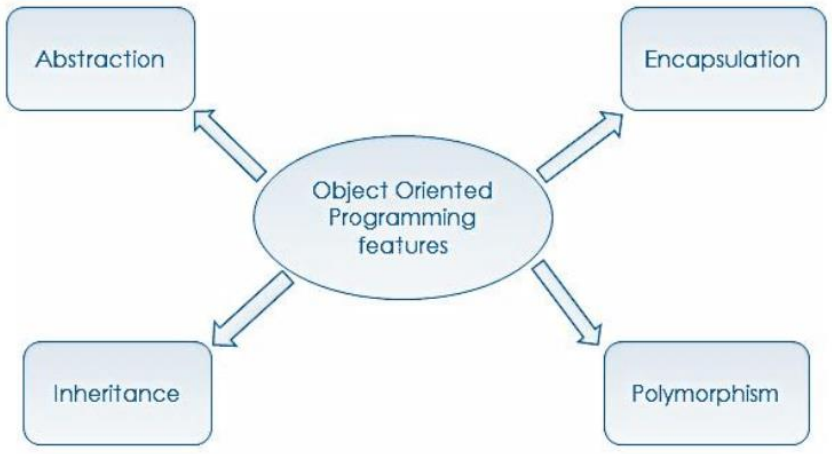
ملخص 5 (Encapsulation) lectuer

*الكلمات الي تحتها خط ركزوا عليها

❖ Overview

- **Object-originated Programming (OOP) concepts:**
  1. **Inheritance**
  2. **Polymorphism**
  3. **Abstraction**
  4. **Encapsulation**



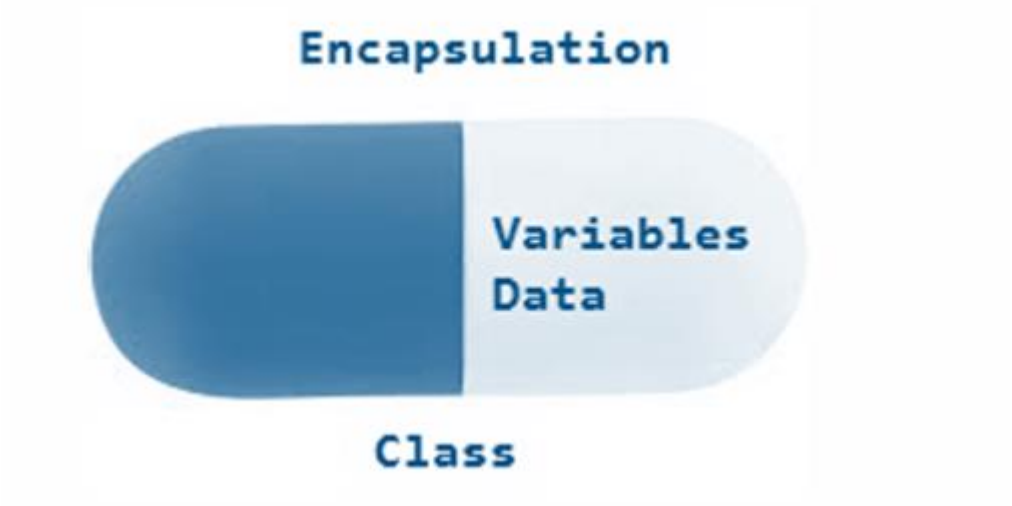
❖ Encapsulation

- **The Encapsulation is one of the important features of OOP. It refers to the grouping of variables and methods inside a single class.**

- **التغليف هو أحد السمات المهمة للبرمجة الشيئية الموجهة. وهو يشير إلى تجميع المتغيرات والدوال داخل كلاس واحد**

- **It helps us to keep related fields and methods together, which makes our code organized and easy to read.**
- **It also stops outer classes from accessing and changing variables and methods of a specific class. This also helps to achieve data hiding increasing the security of data.**

- **يساعدنا على إبقاء الحقول والدوال ذات الصلة معًا ، مما يجعل الكود منظم وسهل القراءة.**
- **كما أنه يمنع الكلاسات الخارجية من الوصول إلى متغيرات ودوال كلاس معين وتغييرها. وهذا يساعد أيضًا على تحقيق إخفاء البيانات مما يزيد من أمان البيانات**



❖ How to achieve Encapsulation? كيف يمكنا تطبيق التغليف

1. Declare the variables of a class as private.

1- تعريف جميع المتغيرات في الكلاس ك private

```
private String name;
private int age;
```

2. Provide public setter and getter methods to modify and view the variables values.

2- عمل دوال ال setter و getter ك public لتعديل وعرض قيم المتغيرات

```
public String getName() {
  return name;
}

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

❖ An example of Setters and Getters

```java
class Person {
    private String name; // Private field to store the name
    private int age; // Private field to store the age


    // Getter methods to retrieve the name and age
    public String getName() { return name; }

    public int getAge() { return age; }


    // Setter methods to set the name and age
    public void setName(String name) { this.name = name; }

    public void setAge(int age) {
            if (age >= 0) { // Validation to ensure age is non-negative
            this.age = age;
        }
        else {
           System.out.println("Invalid age!"); // Prints error message
        }
    }
} // end class

public class Main
{
    public static void main(String[] args) {
        Person person = new Person(); // Create a new Person object
        person.setName("John"); // Set the name using the setter method
        person.setAge(25); // Set the age using the setter method
        System.out.println("Name: " + person.getName());
        System.out.println("Age: " + person.getAge());
        person.setAge(-10); // Attempt to set an invalid age
        System.out.println("Age: " + person.getAge());
    }
}
```

```
Name: John
Age: 25
Invalid age!
Age: 25
```

- جرب الكود اونلاين **اونلاين**

❖ Lombok: Generating Boilerplate Code Using Annotations

- When you write Java code, there are lot of getter and setter methods. It is tedious to write all these boilerplate code. Project Lombok can help in this situation.
- Project Lombok is a Java library that provides annotations to tell the Java compiler to automatically generate boilerplate code such as getter and setter methods for data fields. To use Lombok, you need download a jar file named lombok.jar from:

https://projectlombok.org/download

- لومبوك: توليد أكواد قوالب باستخدام Annotations
- عند كتابة كود جافا، ستجد العديد من دوال الحصول والإعداد .(getter/setter) كتابة كل هذه الأكواد التكرارية أمر ممل. يمكن لمشروع لومبوك أن يساعد في هذه الحالة.
- مشروع لومبوك هو مكتبة جافا توفر تعليقات لإخبار مُجمع جافا بتوليد أكواد قوالب تلقائيًا، مثل دوال الحصول والإعداد لمجالات البيانات. لاستخدام لومبوك، تحتاج إلى تنزيل ملف وثم تثبتها jar باسم lombok.jar من:
  - https://projectlombok.org/download

- يمكن استخدام annotations Lombok لإضافة وظائف مختلفة إلى الكود .إليك بعض annotations الشائعة:
  - @Data يُستخدم لإنشاء getters و setters و constructors و equals() و toString().ghashCode()
  - @Getter يُستخدم لإنشاء getters فقط.
  - @Setter يُستخدم لإنشاء setters فقط.
  - @NoArgsConstructor يُستخدم لإنشاء constructor بدون parameters.
  - @AllArgsConstructor يُستخدم لإنشاء constructor مع parameters.
  - @EqualsAndHashCode يُستخدم لإنشاء equals() و hashCode().
  - @ToString يُستخدم لإنشاء toString().

*طبعا ذا راح تحتاجه لما تسوي مشاريع لكن في الدراسة ما تقدر تستخدمها

* والان شوف الفرق مع استخدام مكتبة ال Lombok وبدون استخدام المكتبة

هذا الكود مع استخدام المكتبة

```java
@Data
public class Person {

    private String name;
    private int age;

}
```

يعادل هذا الكود بدون استخدام المكتبة

```java
public class Person {

    private String name;
    private int age;

    public String getName() {
        return name;
    }

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

    public int getAge() {
        return age;
    }

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

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Person person = (Person) o;
        return Objects.equals(name, person.name) &&
                Objects.equals(age, person.age);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, age);
    }

    @Override
    public String toString() {
        return "Person{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }

}
```

❖ Inner-Classes (Nested Classes)

- هناك طريقة أخرى ممكنة لتغليف البيانات وهي استخدام الفئات الداخلية inner-classes او (nested classes) .

- Inner class refers to the class that is defined inside a class or an interface.

- Inner class يشير إلى الكلاس التي تم تعريف داخل كلاس أو interface

- There are four types of inner classes in Java:

- معانا اربع انواع من inner Class ويجب حفظهم

1. Nested Inner Class
2. Method Local Inner Class
3. Static Nested Class
4. Anonymous Inner Class

Syntax:

```
class Outer_class_name {
        class Inner_class_name { … }
}
```

❖ 1. Nested Inner Class

- **Nested inner class can access any private instance variable of the outer class.**
- **The inner class can have access modifier: private, protected, public, and default**

- **يمكن Nested inner class الوصول إلى أي متغير خاص للكلاس الخارجي.**
- **يمكن أن يكون inner class معدّل الوصول: خاص، ومحمي، وعامة، وافتراضي**

```
class Outer_class{
  class Inner_class{
    public void show(){
        System.out.println(“I’m an inner class”);
    }
  }
}
```

- **In the main method, we can do the following:**

```
Outer_class.Inner_class ob = new Outer_class().new Inner_class();
ob.show();
```

- **مثال آخر**

```
public class OuterClass {

    public class InnerClass {
        public void show() {
            System.out.println("I'm an inner class!");
        }
    }

    public static void main(String[] args) {
        OuterClass outer = new OuterClass();
        OuterClass.InnerClass inner = outer.new InnerClass();
        inner.show();
    }
}
```

Output : `I'm an inner class!`

❖ 2. Method Local Inner Class

- **Inner class can be declared inside a method of an outer class**
- **نستطيع عمل إعلان او declear للكلاس الداخلي inner class بداحل دالة من الكلاس الخارجي**

```
1  class Outer{
2      void outer_method(){
3          System.out.println("Inside outer method");
4          class Inner{
5              public void inner_method(){
6                  System.out.println("Inside inner method");
7              }
8          }
9
10         Inner I = new Inner();
11         I.inner_method();
12     }
13 }
14
15 public class Main
16 {
17     public static void main(String[] args) {
18         Outer o = new Outer();
19         o.outer_method();
20     }
21 }
22
23
```

```
Inside outer method
Inside inner method
```

- **جرب الكود اونلاين** اونلاين

- **لاحظ هنا انه في دالة main سوينا استدعاء للدالة حق الكلاس الخارجي وهذه الدالة يوجد فيها print لذلك يستم تنفيذها , وبرضو في inner class داخل الدالة , وبرضو يوجد اوبجكت واستدعاء للدالة حق الكلاس الداخلي لذلك سيتم تنفذ دالة print الموجودة داخل الدالة**

❖ 3. Static Nested Class

- Static nested classes are not technically inner classes. They are like a static member of outer class.
- Static nested classes ليست inner classes تقنيًا. إنها أشبه بأعضاءٍ ثابتة (static) في الكلاس الخارجي.

```java
class Outer {
     public static void outer_method() {
        System.out.println("Inside outer method");
     }

     static class Inner {
         public static void show() {
            System.out.println("Inside inner class method");
            outer_method();
         }
     }

 }

public class Main
{
    public static void main(String[] args) {
        Outer.Inner.show();

    }
}
```

```
Inside inner class method
Inside outer method
```

- جرب الكود اونلاين [اونلاين]

❖ 4. Anonymous Inner Class

- are not In Java, a class can contain another class known as nested class. It's possible to create a nested class without giving any name known as an anonymous class.
- anonymous class هو عبارة عن كلاس بدون اسم

- An anonymous class must be defined inside another class. So, it is also known as an anonymous inner class.
- يجب تعريف الكلاس المجهول داخل كلاس أخر. لذلك، يعرف أيضًا بالكلاس الداخلي المجهول.

- Anonymous classes usually extend superclasses or implement interfaces as follows:
  - Case A: It can be a superclass that an anonymous class extends
  - Case B: It can be an interface that an anonymous class implements

- تُستخدم الكلاسات المجهولة عادةً لتمديد extend الكلاسات العليا أو تنفيذ implement الواجهات كما يلي:
  - الحالة أ: يمكن للكلاس المجهول ان يسوي extend للكلاس العليا
  - الحالة ب: يمكن للكلاس المجهول ان يسوي implements للواجهة او interface

❖ Case A: a superclass that an anonymous class extends

```java
class Class_A {
    public void show() {
        System.out.println("Inside class A");
    }
}

class AnonymousDemo {
    public void createClass() {
        Class_A c1 = new Class_A() {
            public void show() {
                System.out.println("Inside class anonymous");
            }
        };
        c1.show();
    }
}

class Main {
    public static void main(String[] args) {
        AnonymousDemo an = new AnonymousDemo();
        an.createClass();
    }
}
```

```
Inside class anonymous
```

- جرب الكود اونلاين **اونلاين**

- قمنا بإنشاء كلاس Class_A يحتوي على دالة واحدة show() تطبع "Inside class A"
- ثم قمنا بإنشاء كلاس مجهول يسوي extends من كلاس Class_A و يسوي override لدالة show() الموجودة في كلاس الأب
- عند تشغيل البرنامج، يتم إنشاء كائن c1 من الكلاس المجهول.
- ثم يقوم الكائن باستدعاء دالة show() الخاصة بالكلاس المجهول.

❖ Case B: an interface that an anonymous class implements

```java
interface interface_A {
    public void show();
}

class AnonymousDemo {
    public void createClass() {
        interface_A i1 = new interface_A() {

            public void show() {
                System.out.println("Inside class anonymous");
            }
        };
        i1.show();
    }
}

class Main {
    public static void main(String[] args) {
        AnonymousDemo an = new AnonymousDemo();
        an.createClass();
    }
}
```

```
Inside class anonymous
```

- جرب الكود اونلاين **اونلاين**

- قمنا بإنشاء واجهة interface_A يحتوي على دالة واحدة ()show مجردة (no body)
- ثم قمنا بإنشاء كلاس مجهول يسوي implement من كلاس interface_A و يسوي override لدالة ()show الموجودة في كلاس الأب
- عند تشغيل البرنامج، يتم إنشاء كائن c1 من الكلاس المجهول.
- ثم يقوم الكائن باستدعاء دالة show() الخاصة بالكلاس المجهول.

ALZEEKA Tutorial

شروحات - مشاريع - خدمات - تصاميم

إنضم الآن

053 359 7191