

الفصل الرابع

OOP

- Class and Objects
 - Methods
- Access Modifiers
- Encapsulation
- Inheritance

OOP

Object Oriented Programming

- البرمجة الموجهة نحو الكائنات (OOP)

هي طريقة برمجة تستخدم الأجسام وتفاعلاتها في تصميم التطبيقات وبرامج الحاسوب.

- كل كائن له خصائص وأفعال.

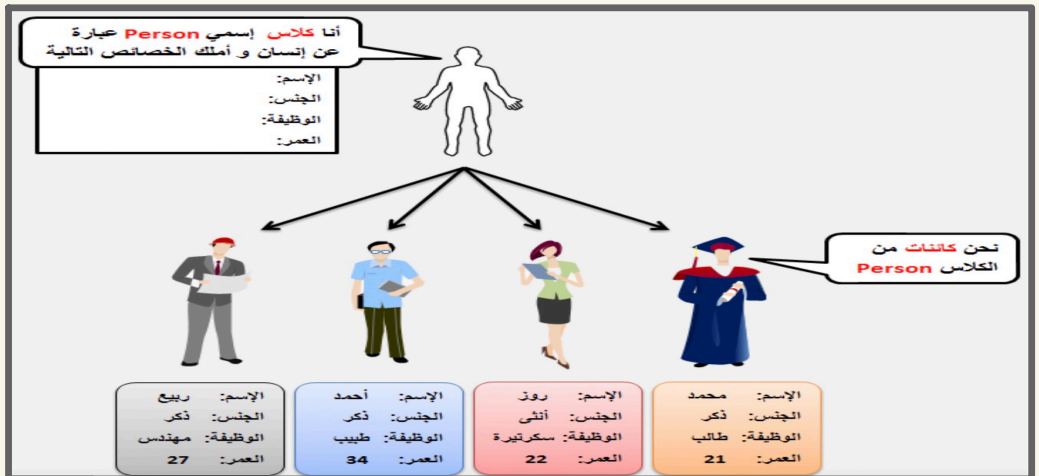
مفهوم Class & Objects

- **Class** : نكتبها كلاس في العربية ، و الكلاس عبارة عن حاوية كبيرة تستطيع أن نحتوي على كل

الكود من متغيرات و دوال و كائنات إلخ..

- **Objects** : تعني كائن في اللغة العربية ، و الكائن عبارة عن نسخة مطابقة لكلاس معين.

على سبيل المثال : علي هو كائن ، الشخص هو فئة.



طريقة التعامل مع الكائنات :

نقوم بإنشاء كائن من الكلاس .

بعدها نقوم بإدخال قيم لخصائصه ، إستدعاء دواله إلخ..

لإستدعاء أي شيء موجود في الكائن الذي أنشأناه

نضع اسم الكائن .

ثم نقطة .

ثم الشيء الذي نريد الوصول إليه (سواء اسم متغير أو دالة) .

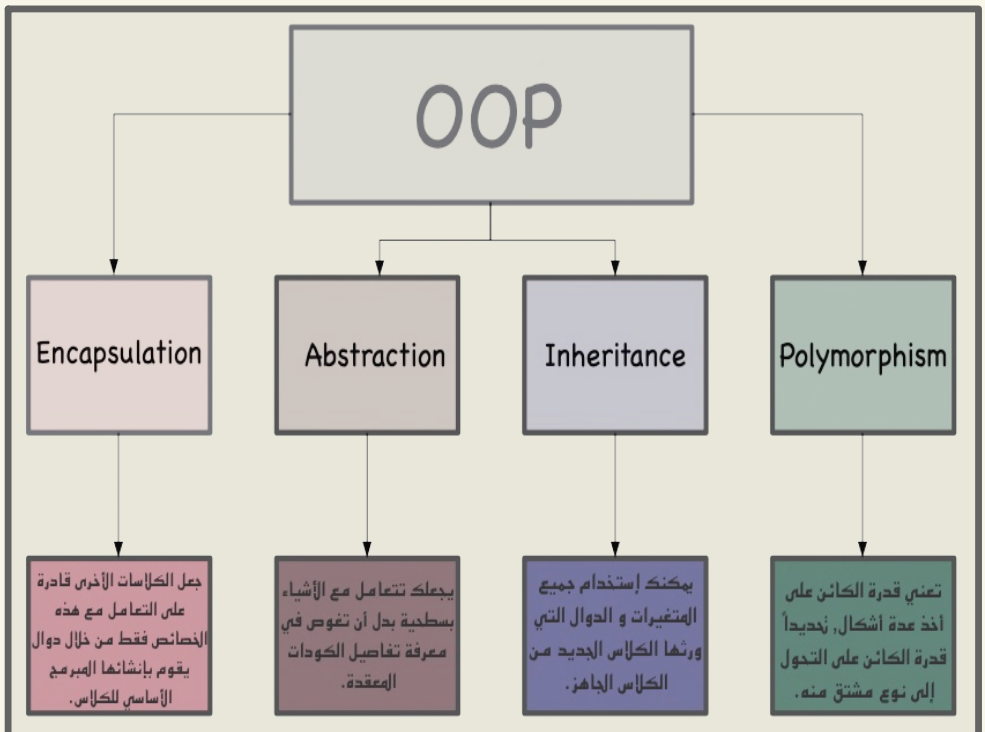
نصائح عليك إتباعها :

يفضل إنشاء كل كلاس في ملف جافا خاص .

ابدأ بإسم الكلاس دائماً بحرف كبير .

ابدأ بإسم الكائن دائماً بحرف صغير .

oop concepts



Method

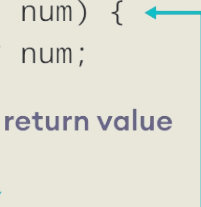
Method : تعني الدوال ، وهي عبارة عن مجموعة أوامر مجمعة في مكان واحد و تنفذ عندما نقوم باستدعائها .

- يمكن استدعائها في أي نقطة في البرنامج من خلال كتابة اسمها في البرنامج ، وبعد تنفيذها يتم الرجوع إلى نقطة الاستدعاء في الدالة الرئيسية .
- تساعد الدوال في تنظيم وتنسيق هيكل البرنامج من خلال تقسيم البرنامج الرئيسي إلى مجموعة برامج فرعية (دوال) بحيث يكون لكل منها وظيفة محددة .
- استخدام الدوال يسهل عملية متابعة وصيانة الشفرة المصدرية للبرنامج .
- تساعد في التقليل من تكرار كتابة الأكواد .

تنقسم الدوال في لغة الجافا إلى نوعان رئيسيان هما :

- **Bulid-In** (الدوال الجاهزة) : وهي مجموعة الدوال المبرمجة مسبقا ضمن حزمة المترجم الخاص بلغة الجافا مثل (الدوال العامة ، الدوال الرياضية ، دوال التعامل مع النصوص) .
- **User-defined** (الدوال المعرفة من قبل المبرمج) : وهي مجموعة الدوال التي يتم إنشائها من قبل المبرمج لأداء وظيفة معينة .

```
int square(int num) {  
    return num * num;  
}  
...  
...  
result = square(10);  
// code
```



Syntax

```
modifier returnType methodName ( Parameter List ) {  
  
    // Method Body  
  
}
```

Modifier : يحدد طريقة الوصول للدالة.

Return Type : يحدد النوع الذي سترجعه الدالة عندما تنتهي أو إذا كانت لن ترجع أي قيمة.

Name of Method : يمثل الاسم الذي نعطيه للدالة ، و الذي من خلاله يمكننا استدعاءها.

parameter lis : المقصود بها الباراميترات (وضع الباراميترات إختياري) .

Method Body : تعني جسم الدالة، و المقصود بها الأوامر التي نضعها في الدالة.

الـ **Return Type** في الدالة يمكن أن يكون أي نوع من أنواع البيانات الموجودة في جافا
(**String - boolean - double - int** إلخ..) .

و يمكن وضع اسم لكلاس معين ، و هنا يكون القصد أن الدالة ترجع كائن من هذا الكلاس.

في حال كانت الدالة لا ترجع أي قيمة ، يجب وضع الكلمة **void** مكان الكلمة **Return Type**.

Example :

```
public int addNumbers(int a, int b) {  
  
    int sum = a + b;  
  
    return sum;  
  
}
```

Method Signatures

التوقيع : عبارة عن اسم الدالة ، ونوع المعاملات وترتيبها ، بالإضافة إلى نوع البيانات الراجعة منها.

Examble :

```
public int sum(int x, int y) {
```

```
    return sum;
```

```
}
```

Parameters

Return type

Name of method

Method Overloading

التحميل الزائد (Overloading) : هي دوال تحمل نفس الاسم، ولكن معاملتها مختلفة .

Example :

```
class MethodOverloading {

    // this method accepts int
    private static void display(int a){
        System.out.println("Got Integer data");
    }

    // this method  accepts String object
    private static void display(String a){
        System.out.println("Got String object");
    }

    public static void main(String[] args) {
        display(1);
        display("Hello");
    }
}
```

Method Constructors

دالة البناء (Constructors) : هذه الدالة ليست void ولا ترجع قيمة ، ويكون

نوعها public.

- يكون اسم الدالة نفس اسم الكلاس ، وليس لها Data Type

- تُستخدم لإعطاء قسم أولية عند إنشاء الـ Object.

Examble :

```
class Circle {  
    public static final double PI = 3.14;  
    public double r;  
    // public constructor  
    public Circle() {  
        this.r = r;  
    }  
    public double circumference() {  
        return 2 * PI * r;  
    }  
    public double area() {  
        return PI * r * r;  
    }  
}
```


Access Modifiers

- **public (عامة)** : الكلاس أو الدالة أو المتغير الذي يتم تعريفه كـ **public** يمكن الوصول إليه مباشرة .

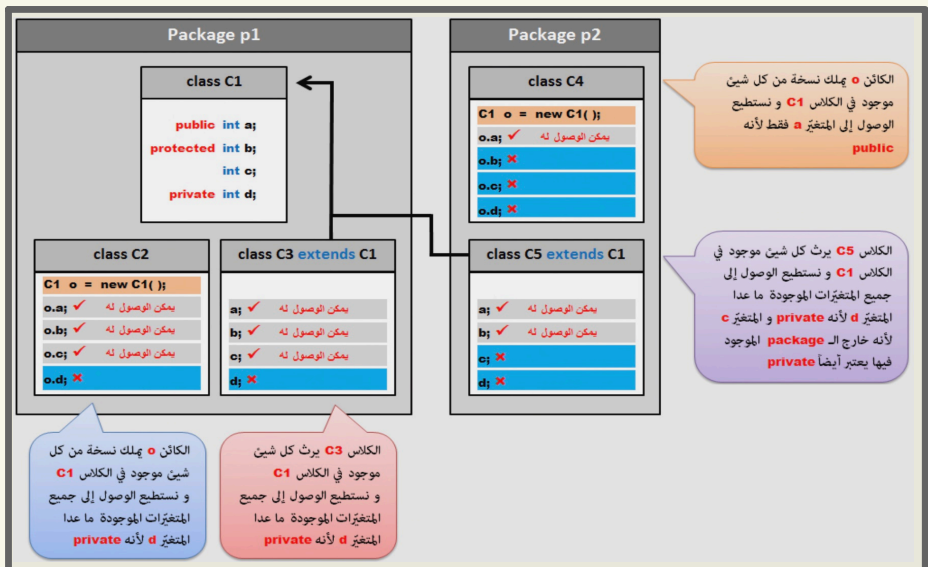
- **private (خاصة)** : هو أعلى مستوى من حيث الحماية. المتغيرات و الدوال التي يتم تعريفها كـ **private** يمكن الوصول لها فقط من داخل الكلاس الذي تم تعريفها فيه .
ملاحظات :

لا يمكنك تعريف كلاس كـ **private** .

لا تقم بتعريف دالة كـ **private** إذا كان نوعها أيضاً **abstract** لأنك لن تستطيع أن تفعل لها **override** .

- **protected (محمية)** : الدالة أو المتغير الذي يتم تعريفه كـ **protected** يمكن الوصول إليه فقط من الكلاسات الموجودة في نفس الـ **package** أو من الكلاسات التي ترث منه .

ملاحظة : لا يمكنك تعريف كلاس كـ **protected** .



This keyword

الكلمة **this** هي كلمة محجوزة في لغة جافا ، و هي تستخدم للإشارة إلى الـ **Global Variables**

وتستخدم أيضاً للإشارة إلى الكائن الحالي .

Examble :

```
class Main {
    String name;

    // setter method
    void setName( String name ) {
        this.name = name;
    }

    // getter method
    String getName(){
        return this.name;
    }

    public static void main( String[] args ) {
        Main obj = new Main();

        // calling the setter and the getter method
        obj.setName("Toshiba");
        System.out.println("obj.name: "+obj.getName());
    }
}
```

Encapsulation

Encapsulation : تعني التغليف ، و هو عبارة عن أسلوب يمكن اتباعه لإخفاء البيانات الأساسية في

الكلاس ، أي لإخفاء الخصائص الموجودة في (Global Variables) ، و جعل الكلاسات الأخرى قادرة على التعامل مع هذه الخصائص فقط من خلال دوال يقوم بإنشائها المبرمج الأساسي للكلاس .

الأسلوب المتبع في عملية التغليف

بما أن فكرة التغليف الأساسية هي إخفاء البيانات من جهة و إتاحة التعامل معها من جهة أخرى .
أول ما يجب أن يخطر في بالك هو أنه يجب تعريف جميع الخصائص (أي المتغيرات التي ستحفظ البيانات) الموجودة في الكلاس كـ **private** لأن تعريف الخصائص كـ **private** يعني أنه يمكن الوصول إليهم فقط من داخل الكلاس الموجودين فيه .

ثاني شيء عليك التفكير فيه هو إيجاد طريقة للوصول إلى هذه الخصائص من الخارج . لذلك عليك تجهيز دوال نوعها **public** للتعامل مع هذه الخصائص ، لأن الدوال التي نوعها **public** يمكن الوصول إليهم من أي مكان .

إذاً لتحقيق مبدأ التغليف ، عليك تعريف الخصائص كـ **private** و تعريف الدوال التي تستخدم للوصول إليهم كـ **public** .

فوائد Encapsulation :

- يمكنه جعل الأشياء الموجودة في الكلاس قابلة للقراءة أو للكتابة من قبل الكلاسات الخارجية .
- يسمح للكلاس بوضع شروط أثناء تخزين البيانات .
- التغليف يساعد أيضاً في جعل البرنامج قابل للتطوير من مبرمجين آخرين بدون حاجة هؤلاء المبرمجين إلى معرفة تفاصيل الكود الأساسي في البرنامج .

Setter & Getter

مفهوم دوال ال Setter و ال Getter

Setter : دورها الحصول على قيمة العنصر.

Getter : دورها إسناد قيمة للعنصر.

عند التعامل مع أي متغير (أو خاصية) فعندك خيارين و هما إما إعطاءه قيمة جديدة و إما الحصول على القيمة الموجودة فيه ، و بما أنه يجب بناء دوال للتعامل مع كل خاصية من الخصائص الموجودة في الكلاس ، ينصح باعتماد أسماء متعارف عليها كالتالي :

ابدأ اسم كل دالة الهدف منها إعطاء قيمة للخاصية بالكلمة **set** ثم اسم الخاصية.

ابدأ اسم كل دالة الهدف منها الحصول على قيمة الخاصية بالكلمة **get** ثم اسم الخاصية.

Examble :

```
public class Employee {  
    private double salary;  
  
    public double getSalary() { // salary هذه الدالة ترجع قيمة المخزنة الخاصية  
        return salary;  
    }  
  
    public void setSalary(double s) { // salary هذه الدالة تعطيه رقم فتقوم بوضعه للخاصية  
        salary = s;  
    }  
}
```

Inheritance

Inheritance : الكلاس يمكنه أن يرث من كلاس آخر، وبالتالي يحصل على الدوال و المتغيرات الموجودة في هذا الكلاس.

فكرة الوراثة بسيطة، لكن فائدتها قوية جداً، فمثلاً إذا كنت تريد إنشاء كلاس جديد ولاحظت أنه يوجد كلاس جاهز يحتوي على عناصر قد تفيدك يمكنك استغلالها بدل كتابتها من الصفر.

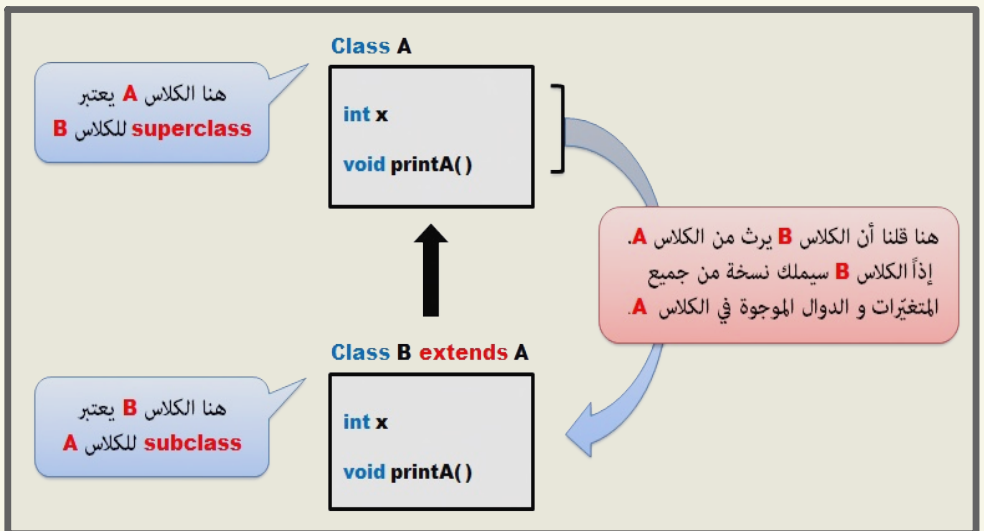
مفهوم Superclass & Subclass

Subclass : وهو الكلاس الذي يرث من كلاس آخر، ويسمى أيضاً (child class).

Superclass : الكلاس الذي يورث محتوياته لكلاس آخر، ويسمى أيضاً (parent class).

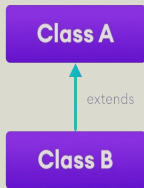
مثال :

الآن لنفترض أننا قمنا بتعريف كلاس اسمه **A** يحتوي على متغير اسمه **x** ودالة اسمها **printA**.
بعدها قمنا بإنشاء كلاس جديد فارغ اسمه **B** وقلنا أنه يرث من الكلاس **A**، إذاً هذا يعني أن الكلاس **B** أصبح يملك نسخة من جميع المتغيرات والدوال الموجودة في الكلاس **A**.



Types of inheritance

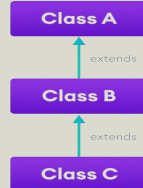
Single inheritance (وراثة فردية)



EX :

```
class A { ..... }  
class B extends A { ..... }
```

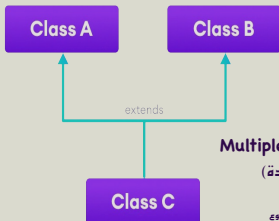
Multi Level inheritance (وراثة متتالية)



EX :

```
class A { ..... }  
class B extends A { ..... }  
class C extends B { ..... }
```

Multiple Inheritance (وراثة متعددة)

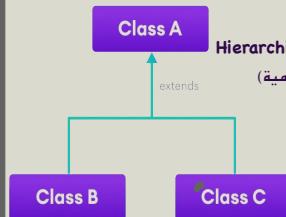


ملاحظة : إذا لم تدعم هذا النوع
ولذلك لتقليل التعقيد وتبسيط اللغة.

EX :

```
class B { ..... }  
class C { ..... }  
class A extends B,C { ..... }
```

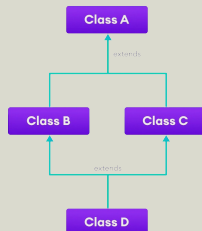
Hierarchical inheritance (وراثة هرمية)



EX :

```
class A { ..... }  
class B extends A { ..... }  
class C extends A { ..... }
```

Hybrid inheritance (وراثة هجينة)



ملاحظة : إذا لم تدعم هذا النوع
ولذلك لتقليل التعقيد وتبسيط اللغة.

Super keyword

الكلمة **super** تستخدم للأهداف التالية:

- للتمييز بين الأشياء (المتغيرات و الدوال) الموجودة في الـ **Superclass** و **Subclass** في حال كانت الأسماء مستخدمة في كلا الكلاسيين.
- لإستدعاء الكونستركتور الموجود في الـ **Superclass**.
- إذا الكلمة **super** تستخدم لإستدعاء الأشياء الموجودة في الـ **Superclass**.

طريقة استخدام الكلمة **super** لإستدعاء متغير من الـ **Superclass**.

```
super.variableName
```

طريقة استخدام الكلمة **super** لإستدعاء دالة من الـ **Superclass**.

```
super.methodName();
```