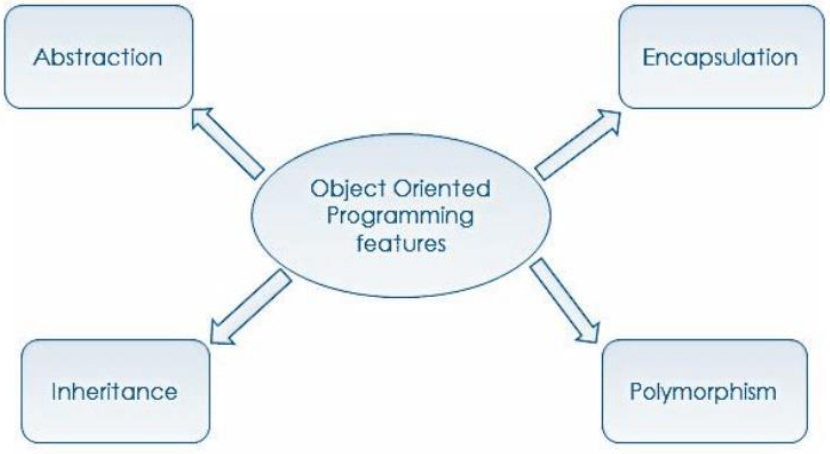
FREE

# ملخص 3 (Polymorphism) lectuer

*الكلمات الي تحتها خط ركزون عليها

## ❖ Overview

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



## ❖ Polymorphism

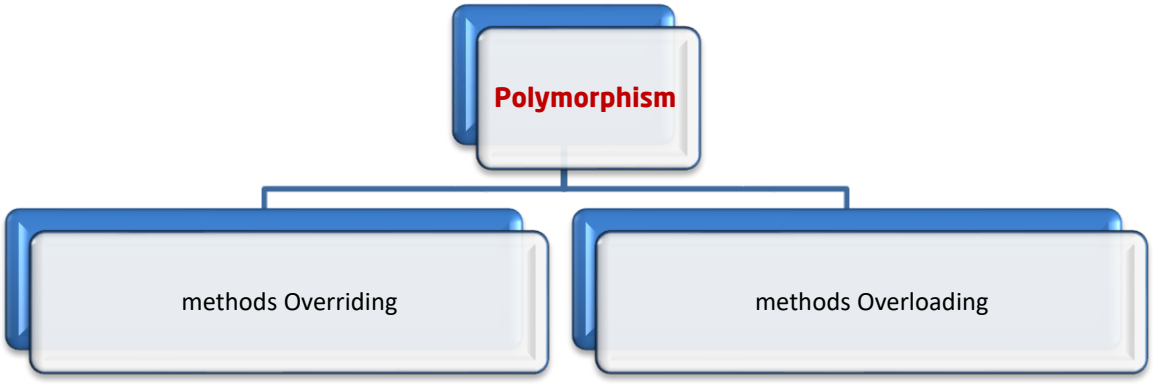
**Polymorphism is an important concept of object-oriented programming. It simply means more than one form.**

**مفهوم متعدد الأشكال (Polymorphism) من أهم المفاهيم في البرمجة الشيئية. وهو يعني ببساطة "أكثر من شكل واحد"**

**That is, the same entity (method or object) can perform different operations in different scenarios.**

**يعني ان الكيان نفسه (دالة أو كائن) يمكنه إجراء عمليات مختلفة في سيناريوهات مختلفة.**

- **We can achieve Polymorphism in Java using the following ways:**
- **يمكن تطبيق ال Polymorphism بطريقتين:**



## ❖ Why we need Polymorphism?

- Polymorphism allows us to create consistent code
- Assume that we have a Shape class (as a superclass) and we have two classes Circle & Square (as subclasses). The class Circle has a method drawCircle and the class Square has a method called drawSquare.
- The two methods work perfectly. However, for every shape, we need to create different methods. It will make our code inconsistent.
- To solve this, polymorphism in Java allows us to create a single method Draw() that will behave differently for different shapes.

- "تعدد الأشكال." يسمح لنا بكتابة كود أكثر مرونة وقابلية للتوسعة
- تخيل أن لدينا كلاس عام superclass يسمى "Shape" (شكل)، وكلاسين فرعيين subclass منها هما "Circle" (دائرة) و"Square" (مربع). كل كلاس من هذه الكلاسات الفرعية لديه دالة خاصة لرسم نفسه، مثل "drawCircle" و"drawSquare".

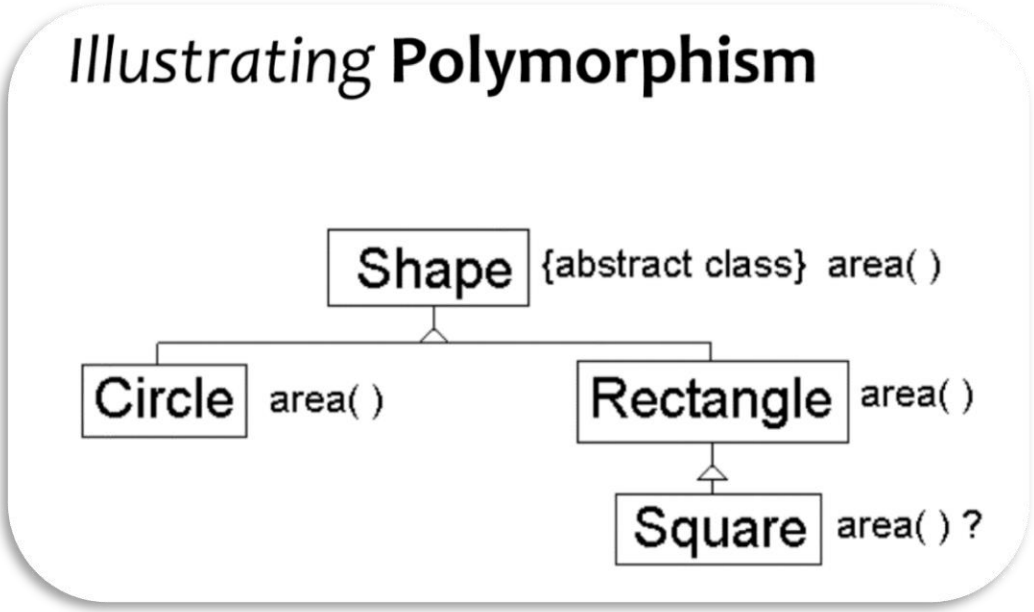
- بدون تعدد الأشكال، سنحتاج إلى كتابة كود منفصل لكل شكل لرسمه. هذا يجعل الكود أقل مرونة وأكثر صعوبة في التعديل.

- لكن تعدد الأشكال يحل هذه المشكلة!

- باستخدام تعدد الأشكال، يمكننا إنشاء دالة واحدة فقط في الكلاس superclass "Shape" تسمى "draw()". هذه الطريقة سيتم تنفيذها بشكل مختلف لكل فئة فرعية، اعتماداً على نوع الشكل الذي نتعامل معه.

```java
class Shape {
    public void draw() {
        System.out.println("draw shape")
    }
}

class Circle extends Shape {

    public void draw() {
        System.out.println("draw Circle");
    }
}

class Square extends Shape {

    public void draw() {
        System.out.println("draw Square");
    }
}
```

- برضو بنحتاج ال polymorphism في ال abstraction يعني اذا معانا دالة abstract في كلاس abstract نستخدم ال polymorphism لعمل override لتلك الدالة من كلاس آخر




## ❖ Method Overriding

- During inheritance in Java, if the same method is present in both the superclass and the subclass. Then, the method in the subclass overrides the same method in the superclass. This is called method overriding.

- في جافا، عندما نتحدث عن الوراثة، يمكن للفئات الفرعية أن ترث الخصائص والدوال من الفئات العامة. لكن في بعض الأحيان، نحتاج إلى تعديل سلوك بعض الدوال في الفئات الفرعية لتتناسب مع احتياجاتها الخاصة وتكون هذه الدالة موجودة في superclass and the subclass. هذه العملية تمسى method overriding

```
class A {
  public void printData() { System.out.print("I'm in Class A"); }
}

class B extends A {
  @Override // note that the keyword @Override is not mandatory
  public void printData() { System.out.print("I'm in Class B"); }
}
```

- في المثال السابق، لدينا دالة تسمى "printData()" في الكلاس A والكلاس B. يمكن القيام بتجاوز الطرق (Overriding) عندما يرث الكلاس B (child class) خصائص الكلاس A (parent class).
- عندما يعمل الكلاس B overrides للدالة "printData()" من الكلاس A ، تقوم الدالة في الكلاس B بتغيير محتوى الدالة body ليصبح مختلفاً عن محتوى الدالة في الكلاسA .

## ❖ Method Overloading

- Method overloading in Java allows to create methods that have the same name but with different parameters.

- يسمح Overloading للدوال في Java بإنشاء دوال تحمل الاسم نفسه، ولكنها تختلف في parameters. يُمكن أن تكون الاختلافات في نوع parameters أو في العدد أو في كليْهما. مثل :

```
void  sum(int x, int y) { ..... };
void  sum(double x, double y) { ..... } ;
float sum(int x, float y) { ..... } ;
```

- The methods have the same name sum. But, the parameters are different for each method.
- Note that the three methods of sum(..) are in one class. We differentiate between them in the method return type and the parameters numbers and/or types.
- يرجى ملاحظة أن الدوال الثلاثة sum(..) توجد في نفس الكلاس. نحن نميز بينها بناءً على نوع الإرجاع للطريقة وعدد أنواع معالماتها.

## ❖ An example of method Overloading

```
class Pattern{

 public void Show(){
  for (int i=0; i<10; i++){
    System.out.print("*");
  }
 }

 public void Show(char c){
  for (int i=0; i<10; i++){
    System.out.print(c);
  }
 }

} // end class
```

**Creating objects in the Main method:**

```
Pattern p = New Pattern();
p.Show();
System.out.print("\n");
p.Show('%');
```

**Output:**

```
**********
%%%%%%%%%%
```

## ❖ Overriding vs Overloading

```java
public class Test {
  public static void main(String[] args) {
    A a = new A();
    a.p(10);
    a.p(10.0);
  }
}

class B {
  public void p(double i) {
    System.out.println(i * 2);
  }
}

class A extends B {
  // This method overrides the method in B
  public void p(double i) {
    System.out.println(i);
  }
}
```

```java
public class Test {
  public static void main(String[] args) {
    A a = new A();
    a.p(10);
    a.p(10.0);
  }
}

class B {
  public void p(double i) {
    System.out.println(i * 2);
  }
}

class A extends B {
  // This method overloads the method in B
  public void p(int i) {
    System.out.println(i);
  }
}
```

- *ملاحظات
  - لايمكن تطبيق ال method overrides في نفس الكلاس لأن الدوال لديها نفس ال parameters.
  - يمكن تطبيق ال method overloading في نفس الكلاس لأن الدوال لديها اختلاف في ال parameters

## ❖ Dynamic Binding

```
public class PolymorphismDemo {
  public static void main(String[] args) {
    m(new GraduateStudent());
    m(new Student());
    m(new Person());
    m(new Object());
  }

  public static void m(Object x) {
    System.out.println(x.toString());
  }
}

class GraduateStudent extends Student {
}

class Student extends Person {
  public String toString() {
    return "Student";
  }
}

class Person extends Object {
  public String toString() {
    return "Person";
  }
}
```

Method m takes a parameter of the Object type. You can invoke it with any object.

**Output:**

```
Student
Student
Person
java.lang.Object@685f4c2e
```

- طبعا كلاس **Person** يرث من كلاس **Object** وهو غير موجود كيف كذا؟
  نحن درسنا في lectuer 2 انه في كلاس اسمه class object وهو كلاس موجود و تلقائيا
  كل الكلاسات في الجافا ترث منه سواء سويت extends او لا

- نلاحظ ان في هذا المثال موجود عدة كلاسات ترث من بعض بهذا الشكل

**GraduateStudent → Student → person → Object**

- ونلاحظ ان لكل كلاس دالة toString()وعاملين لها override لكي تكون خاصة بكل كلاس ما
  عدا كلاس **GraduateStudent** لاتوجد فيه بس هو راح ياخذ الدالة من كلاس **Student**

- ومن خلال الدالة **m()** يمكن انشاء وارسال لها ال object من دالة main ونلاحظ انه لما استدعينا
  الدالة من كلاس object كان ال output كذا **java.lang.Object@685f4c2e** لانه هنا استدعى
  الدالة الإفتراضية وطبع اسم الباكج واسم الكلاس والهاش كود

- جرب هذا المثال مرتب اونلاين **اونلاين**

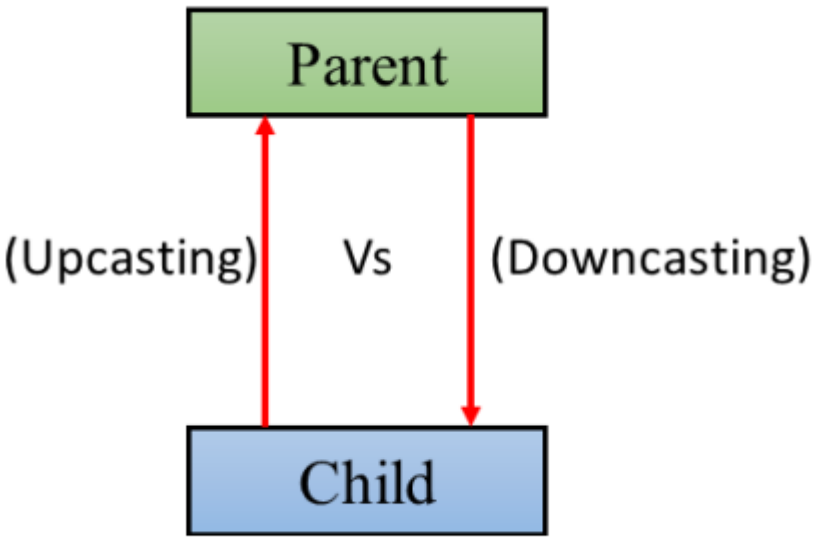
## ❖ Upcasting and Downcasting

- **Upcasting is when an object of a derived class is assigned to a variable of a base class (or any ancestor class):**

- **(Upcasting) هو عملية تعيين كائن من derived class إلى متغير من base class**

- **Syntax:**

```
Parent p = new Child();
```



## ❖ Upcasting Example

```java
class Shape {
    int xpos, ypos;

    public Shape(int x, int y) {
        xpos = x;
        ypos = y;
    }

    public void draw() {
        System.out.println("Draw method called of class Shape");
    }
}

class Circle extends Shape {
    int r;

    public Circle(int x1, int y1, int r1) {
        super(x1, y1);
        r = r1;
    }


    public void draw() {
        System.out.println("Draw method called of class Circle");
    }
}


public class Main
{
     public static void main(String[] args) {
         Shape s = new Circle(10, 20, 4);
        s.draw();
     }
}
```

**Output:**
Draw method called of class Circle

• جرب هذا المثال مرتب اونلاين **اونلاين**

- When we wrote Shape S = new Circle(10 , 20 , 4), we have cast Circle to the type Shape.
- This is possible because Circle has been derived from Shape

- في السطر(Shape s = new Circle(10, 20, 4 ، قمنا بتعيين كائن من فئة الدائرة (Circle) إلى مرجع من نوع الشكل (Shape). هذا يعني أننا نعامل كائن الدائرة على أنه كائن من نوع الشكل العام.
- هذا ممكن لأن Circle مشتق او يرث من Shape. في هذه الحالة، نستطيع فقط استخدام الطرق (methods) الموجودة في كلاس الشـــكل (Shape) بشـــكل مباشـــر. وذلك لأن كلاس الدائرة (Circle) ورث جميع الخصـائص والطرق من كلاس الشـكل (Shape)، ولكن العكس ليس صـحيحاً، يعني اذا استدعينا دوال Circle الخاصة راح يظهر error

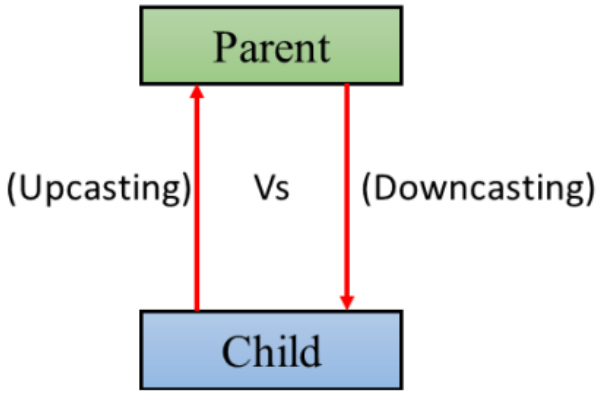
## ❖ Downcasting

- Downcasting is when a type cast is performed from a base class to a derived class (or from any ancestor class to any descendent class)

- (Downcasting) هو عملية تحويل نوعية تُجرى على مرجع لكائن ما من base class إلى derived class

- Syntax:

```
Child c = (Child)p;
```



## ❖ Downcasting Example

```java
class Shape {
    int xpos, ypos;

    public Shape(int x, int y) {
        xpos = x;
        ypos = y;
    }

    public void Draw() {
        System.out.println("Draw method called of class Shape");
    }
}

class Circle extends Shape {
    int r;

    public Circle(int x1, int y1, int r1) {
        super(x1, y1);
        r = r1;
    }


    public void Draw() {
        System.out.println("Draw method called of class Circle");
    }

    public void Surface() {
        System.out.println("The surface of the circle is " + (Math.PI * r * r));
    }
}



public class Main
{
    public static void main(String[] args) {
        Shape s = new Circle(10, 20, 4);
        ((Circle) s).Surface();
    }
}
```

**Output:**
The surface of the circle is 50.26

• جرب هذا المثال مرتب اونلاين **اونلاين**

- When we wrote Shape s = new Circle(10 , 20 , 4), we have cast Circle to the type shape.
- In that case, we are only able to use methods found in Shape, that is, Circle has inherited all the properties and methods of Shape.

- في السطر(Shape s = new Circle(10, 20, 4 ، قمنا بتعيين كائن من فئة الدائرة (Circle) إلى مرجع من نوع الشكل .(Shape) هذا يعني أننا نعامل كائن الدائرة على أنه كائن من نوع shape
- في هذه الحالة، نستطيع فقط استخدام الطرق (methods) الموجودة في كلاس الشكل (Shape) بشكل مباشر. وذلك لأن كلاس الدائرة (Circle) ورث جميع الخصائص والطرق من كلاس الشكل (Shape)، ولكن العكس ليس صحيحاً. يعني اذا استدعينا دوال Circle الخاصة لن يظهر error

- If we want to call Surface() method, we need to downcast our type to Circle. In this case, we will move down the object hierarchy from Shape toCircle :
- لكن إذا أردت استدعاء دالة Surface() ، فأنت بحاجة إلى إجراء عملية التحويل إلى الأسفل (downcasting)لإعادة التعامل مع الكائن على أنه من نوع الدائرة.(Circle)

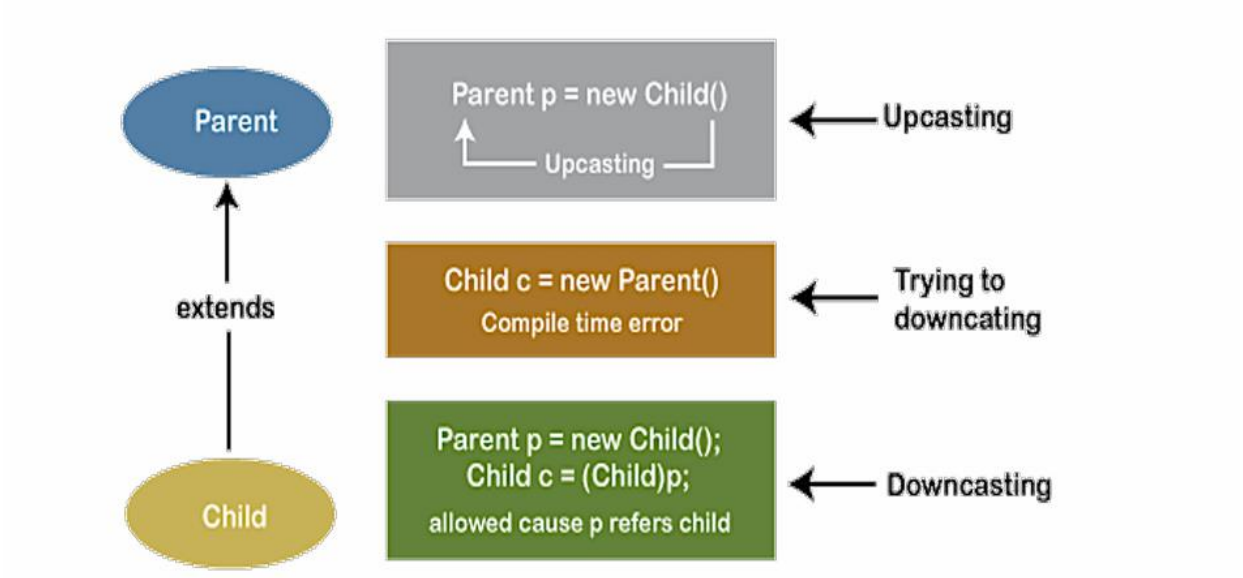
```
((Circle) s).Surface() ;
```

- If you use: Child c = new Parent(); will give a compile-time error.
- واذا سويت بهذا الشكل يعني object من كلاس الأب جعلته ك object لكلاس الإبن راح يظهر لك error بال compile-time error

## ❖ Why we need Upcasting and Downcasting?

- In Java, we rarely use Upcasting. We use it when we need to develop a code that deals with only the parent class.
  - التحويل إلى الأعلى:(Upcasting)
    - نادراً ما نستخدمه في Java.
    - نستخدمه عندما نحتاج إلى تطوير كود يتعامل فقط مع الفئة الأصلية.(parent class)

- Downcasting is used when we need to develop a code that accesse behaviors of the child class.
  - التحويل إلى الأسفل:(Downcasting)
    - نستخدمه عندما نحتاج إلى تطوير كود يستفيد من سلوكيات الكلاسات الفرعية.
    - يسمح لنا هذا التحويل باسترداد الميزات الخاصة بالكلاس الفرعي



# ALZEEKA Tutorial

## شروحات - مشاريع - خدمات - تصاميم

إنضم الآن

053 359 7191