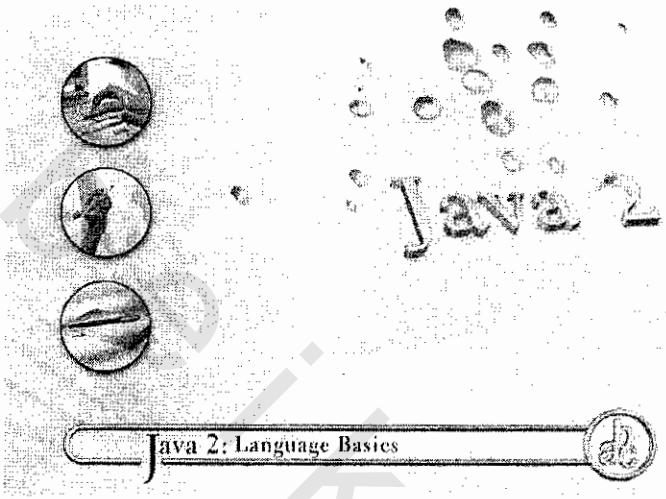
الفصل التاسع



تعدد الأشكال Polymorphism

في هذا الفصل سوف نتناول أحد خصائص البرمجة بواسطة الأهداف OOP وهي خاصية تعدد الأشكال Polymorphism وذلك من خلال النقاط التالية:

- تعدد الأشكال Polymorphism.
- التحميل الزائد للدوال Method overloading.
- نسخ الدوال بالتوريث Method Overriding Through Inheritance.
- مساواة الأهداف والتحويل بين الأنواع Assignment Compatibility and Type Conversion.
- تحقيق عملية تعدد الصور أثناء تنفيذ البرنامج Runtime Polymorphism.
- الفصيلة Object .

تعدد الأشكال Polymorphism:

- هى أحد خصائص البرمجة بالأهداف OOP ومعناها إمكانية استعمال متغير (هدف object) من الفصيلة الأم Parent class للإشارة إلي الفصائل classes الوارثة للفصيلة الأم Parent class ، وتسمى dynamic binding or late binding or run-time binding.
- ويمكن القول أن خاصية تعدد الأشكال Polymorphism معناها باختصار هو وجود اسم واحد بأكثر من شكل ، وبالطبع توجد أشكال متقاربة من هذا المعنى ناقشناها من قبل مثل:

- التحميل الزائد للدوال Method Overloading.
- نسخ الدوال بالتوريث Method Overriding Through Inheritance.
- نسخ الدوال باستخدام Method Overriding Through (Interface) Java Interface.

أولاً: التحميل الزائد للدوال Method overloading:

- وهذا يعني وجود أكثر من دالة Method بنفس الاسم داخل نفس الفصيلة class مع اختلاف المعاملات Parameters من حيث النوع أو العدد ، وهذا ما نسميه بالتحميل الزائد للدوال Method Overloading ، ويطلق عليها البعض Compile-time Polymorphism.
- ويتضح ذلك من المثال التالى:

```
1. class B {
2. public void m(int x){
3. System.out.println("m(int x)");
4. }
5. public void m(String y){
6. System.out.println("m(String y)");
7. }
8. }
```

شرح السطور:

- تم إنشاء دالتين Methods بالاسم m() ، الأولي بمعامل Parameter من النوع int والثانية بمعامل Parameter من النوع String.
- وهذه هى نظرية إنشاء أكثر من دالة Method التى تناولناها من قبل ، وهى أحد صور تعدد الأشكال Polymorphism.
- ويتم استدعاء دوال Methods هذه الفصيلة class كما في السطور التالية:

```
1.  public class Poly01{
2.  public static void main(String[] args)
3.  {
4.  B var = new B();
5.  var.m(3);
6.  var.m("String");
7.  }
8.  }
```

شرح السطور:

- تم تعريف هدف object اسمه var من نوع الفصيلة B ثم تم استدعاء الدالة m() مرتين ، المرة الأولى بمعامل Parameter رقم 3 وبالتالى يتم استدعاء الدالة Method الأولى ، والثانية بمعامل Parameter من نوع String ، وهذا ما يسمى بالتحميل الزائد للدوال Method Overloading ، وكما أشرنا هو أحد صور تعدد الأشكال Polymorphism.
- وعند تنفيذ البرنامج تحصل على نتيجة التنفيذ التالية:

```
m(int x)
m(String y)
```

ثانياً: نسخ الدوال بالتوريث Method Overriding Through Inheritance:

- ويتم ذلك عن طريق إنشاء دالة Method بنفس اسم ومعاملات Parameters الدالة Method الموجودة فى الفصيلة الأم Parent class وذلك بعد تحقيق عملية

التوريث Inheritance لفصيلة class جديدة ، ويتم إنشاء الدالة Method الجديدة فى الفصيلة الصغري Child class ، وهذا ما يسمى بنسخ الدوال Method Overriding.

- والصورة الجديدة لتعدد الأشكال Polymorphism تختلف في طريقة استدعاء هذه الدوال Methods ، حيث يتم استدعاء نفس الدالة Method ولكن مع أهداف objects من فصائل classes مختلفة ترث من نفس الفصيلة الأم Parent class.
- وبالطبع ليست الفكرة واضحة تماماً وسوف نقوم بتوضيح ذلك من خلال الفقرات التالية.

مساواة الأهداف والتحويل بين الأنواع

Assignment Compatibility and Type Conversion:

- قبل الاستمرار فى تناول موضوع تعدد الأشكال Polymorphism ، يجب فهم موضوع مساواة الأهداف objects والتحويل casting بين الأنواع ، بداية يمكن مساواة المتغيرات إذا كانت القيمة الموجودة فى المتغير الأول يمكن وضعها فى المتغير الثانى.
- أما في التحويل بين الأنواع Type Conversion and Casting ، فإنه يتم التحويل بين الأنواع إما تلقائياً Automatically أو إجبارياً Forced ، وذلك باستعمال مؤثر للتحويل Cast Operator المناسب للنوع ، وهذا المؤثر Operator هو اسم النوع المطلوب التحويل إليه ويوضع بين قوسين ، فمثلاً للتحويل إلى النوع int ، يتم كتابة مؤثر التحويل Cast Operator بالصورة (int) ، مع ملاحظة أن هذا التحويل casting ليس نهائياً ولا حقيقياً حيث يتم التحويل casting فقط أثناء العملية ولكن يبقى النوع الاصلى كما هو ، هو فقط يعامل معاملة النوع المطلوب التحويل إليه.
- أما عند مساواة متغيرات الأهداف Assignment.Compatibility.for References ، فنجد أن هذه العملية تختلف عن مساواة متغيرات البيانات الأساسية Primitives ، حيث لا تصلح المساواة مع متغيرات الأهداف objects إلا فى الحالات التالية:

1- عندما يكون الهدفين reference variables من نفس النوع ، أى هما أهداف Objects من نفس الفصيلة class.

2- عندما يكون متغير الهدف reference variable من نوع الفصيلة الأم Parent class لمتغير الهدف object الأخر.

3- عندما يكون متغير الهدف reference variable من نوع interface تم تنفيذه implement بفصيلة class الهدف object الثانى.

- وبالتالى فإن بعض بعض عمليات مساواة متغيرات الأهداف objects لا تحتاج مؤثر التحويل cast operator.

مثال (1): التحويل Casting:

يوضح هذا المثال استخدام مؤثر التحويل cast operator with references وذلك كما فى السطور التالية:

```
class A extends Object{

}

class B extends A{
  public void m(){
    System.out.println("m in class B");
  }
}
class C extends Object{

}
public class NewClass{
  public static void main(String[] args){
    Object var = new B();
    //Following will not compile
    //var.m();
    //Following will not compile
```

```
    //((A)var).m();
    //Following will compile and run
    ((B)var).m();

    //Following will compile and run
    B v1 = (B)var;
    //Following will not execute
    //C v2 = (C)var;
    //Following will not compile
    //C v3 = (B)var;
  }//end main
}
```

شرح السطور:

- تم الإعلان عن فصيلة class جديدة بالاسم A ترث من الفصيلة الأساسية للفصائل Object ثم تم الإعلان عن فصيلة class جديدة بالاسم B ترث من الفصيلة A وكذلك تم الإعلان عن فصيلة class بالاسم C .
- ثم تم إنشاء الفصيلة الرئيسية NewClass التى تحتوى على الدالة الرئيسية ()main.
- داخل الدالة الرئيسية ()main يتم تعريف هدف object بالاسم var من نوع الفصيلة Object وتم استعماله فى الإشارة إلى هدف object من الفصيلة B ، وذلك صحيح تماماً نظراً لأن الفصيلة Object هى الفصيلة الأم Parent class لجميع الفصائل classes.
- داخل التعليقات يتم الإشارة إلى أن الاستدعاء بالطريقة ;(var.m()// تسبب خطأ في الترجمة compile لأن متغير الهدف object من نوع الفصيلة class الأقل فى التركيب ، والصحيح هو ;((B)var).m() ، وهكذا يتم التوضيح بالتعليقات ما يصلح وما لا يصلح ، وبالتالي يتم توضيح فكرة تحويل casting النوع أثناء المساواة.

تحقيق عملية تعدد الصور أثناء تنفيذ البرنامج Runtime Polymorphism:

- يتم تحقيق عملية تعدد الأشكال Polymorphism أثناء تنفيذ البرنامج Runtime

Polymorphism من خلال خاصية التوريث Inheritance وخاصية نسخ الدوال Method Overriding ، ولتوضيح ذلك تابع معنا الخطوات التالية:

- بافتراض أننا قمنا بتعريف فصيلة class بالاسم SuperClass وتعريف دالة Method بداخلها بالاسم method.
- ثم قمنا بتعريف فصيلة class بالاسم SubClass ترث من الفصيلة SuperClass وتم تعريف دالة Method بداخلها بالاسم method وبالتالى تم تحقيق خاصية النسخ Overriding.
- ثم قمنا بتعريف متغير reference للفصيلة SuperClass بالاسم ref واستخدمناه للإشارة إلى هدف object من الفصيلة SubClass.
- بافتراض أننا قمنا باستدعاء الدالة method مع متغير الهدف ref بالصورة ref.method() ، السؤال: هل يتم استدعاء الدالة method() الأصلية التابعة للفصيلة الأم SuperClass ، أم يتم استدعاء الدالة method() الجديدة التى تم تعريفها فى الفصيلة SubClass.
- النتيجة: يتم استدعاء الدالة method() الجديدة والموجودة فى الفصيلة SubClass وذلك لأن المتغير يشير إلى هدف object من نوع هذه الفصيلة class ، وهذا هو أشهر شكل من أشكال تعدد الأشكال Polymorphism والتى تسمى Runtime Polymorphism ويطلق عليها أحياناً late-binding.

ملحوظة:

يتم تحديد الدالة Method التى يتم استدعاؤها فى حالة Runtime Polymorphism حسب الهدف object الذى يشير إليه المتغير وليس علي حسب نوع المتغير ، ويتم تحديد الدالة Method وقت التنفيذ Runtime.

مثال (2): تعدد الأشكال Polymorphism:

هذا المثال يوضح الخاصية Runtime Polymorphism وذلك كما فى السطور التالية.

```
/*

This program illustrates downcasting
and polymorphic behavior

Program output is:

I am in class B
I am in class B
I am in class A
**********************************/

class A extends Object{
  public void m(){
    System.out.println("I am in class A");
  }//end method m()
}//end class A
//=====================================//

class B extends A{
  public void m(){
    System.out.println("I am in class B");
  }//end method m()
}//end class B
//=====================================//

public class C{
  public static void main(String[] args){
    Object var = new B();
    //Following will compile and run
    ((B)var).m();
    //Following will also compile
    // and run due to polymorphic
    // behavior.
```

```
    ((A)var).m();
    //Following will not compile
    //var.m();
    //Instantiate obj of class A
    var = new A();
    //Invoke the method on it
    ((A)var).m();
  }//end main
}//end class C
//===================================//
```

شرح السطور:

- تم إنشاء ثلاث فصائل classes بالأسماء A, B, C وتم إنشاء دالة Method بالاسم ()m في الثلاث فصائل classes ، أي تم استخدام خاصية نسخ الدوالMethod Overriding .
- يتم التوضيح بالتعليقات ما يصلح وما لا يصلح كما تم توضيح ناتج تنفيذ البرنامج في بداية الكود.

الفصيلة Object :

- تعتبر الفصيلة Object هي الفصيلة الأم Parent class لجميع الفصائل classes ، وهي موجودة على رأس شجرة الفصائل classes ، وتحتوى هذه الفصيلة class على مجموعة من الدوال Methods هي:

- clone()
- equals(Object obj)
- finalize()
- getClass()
- hashCode()
- notify()
- notifyAll()
- toString()

- wait()
- wait(long timeout)
- wait(long timeout, int nanos)

وبالطبع فإن جميع الفصائل classes ترث هذه الدوال Methods.

المثال التالي يوضح استعمال الفصيلة Object.

مثال (3): الفصيلة Object:

هذا المثال يوضح كيفية استعمال الفصيلة Object لتوضيح خاصية تعدد الأشكال Polymorphism وذلك كما فى السطور التالية:

```
1.  class A extends Object
2.  {
3.  }
4.  class B extends A
5.  {
6.  public String toString(){
7.  return "toString in class B";
8.  }
9.  }
10. class C extends B
11. {
12. public String toString(){
13. return "toString in class C";
14. }
15. }
16. public class Polymorphism04
17. {
18. public static void main(
19. String[] args){
20. Object varA = new A();
21. String v1 = varA.toString();
22. System.out.println(v1);
```

```
23.  Object varB = new B();
24.  String v2 = varB.toString();
25.  System.out.println(v2);
26.  Object varC = new C();
27.  String v3 = varC.toString();
28.  System.out.println(v3);
29.  }//end main
30.  }
```

شرح السطور:

- فى السطر رقم 1 تم تعريف فصيلة class جديدة بالاسم A ترث من الفصيلة Object.
- فى السطر رقم 4 تم تعريف فصيلة class جديدة بالاسم B ترث من الفصيلة A وتقوم بإعادة كتابة الدالة ()toString باستعمال خاصية نسخ الدوال Method Overriding.
- بالمثل فى السطر رقم 10 تم إنشاء الفصيلة C التى ترث من الفصيلة B وتقوم بإعادة كتابة الدالة ()toString باستعمال خاصية نسخ الدوال Method Overriding.
- فى السطر رقم 16 تبدأ الفصيلة class الأساسية وبها الدالة الرئيسية ()main التى تستعمل الفصائل classes السابقة.
- وعند تنفيذ هذا البرنامج تحصل على نتيجة التنفيذ التالية:

```
A@111f71
toString in class B
toString in class C
```

ملخص الفصل:

- تعرضنا في هذا الفصل لشرح أحد أهم مبادئ البرمجة بواسطة الأهداف OOP وهو مبدأ تعدد الأشكال Polymorphism.
- في الفصل القادم سوف نتعرف - بإذن الله - علي كيفية استخدام الحزم Packages ، فتابع معنا الفصل القادم.