دليل الاحتراف

Oracle Java 10

المهام الأساسية

مرفق DVD

م/عزب محمد عزب

استشارى البرمجة وتطوير النظم

Easy Tutorials®

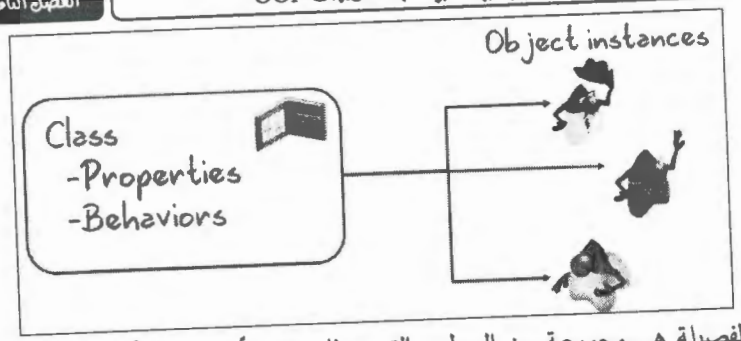
معنى البرمجة بواسطة الأهداف OOP(Object oriented programming)

تبني فكرة البرمجة بواسطة الأهداف (OOP) علي استعمال الهدف (Object) كوحدة برمجة بمعنى أنه بدلاً من استعمال الدوال والأوامر لبناء البرنامج مما يضطر المبرمج لإعادة كتابة الأوامر كل مرة لتحقيق فكرة معينة وهذه كانت فكرة البرمجة التقليدية .

ولكن أتت البرمجة بواسطة الأهداف (OOP) لتجعل وحدة بناء البرمجة كبيرة وهي هدف Object أو فصيلة class وبالتالي يتم إعداد مجموعة من الفصائل العامة classesالتي تلي معظم متطلبات إعداد برنامج والتي يحتاجها المبرمج حتى أن بعض المبرمجين يشبه البرمجة بواسطة الأهداف بالبناء باستعمال المباني الجاهزة والبرمجة بالطريقة التقليدية القديمة تشابه البناء باستعمال الأدوات الأولية وبالتالي الفرق بينهما في السرعة كبيرة جدا

معنى الفصيلة class

الفصيلة class هي أساس البرمجة بواسطة الأهداف (OOP) وهي التي يبنى عليها البرنامج وأخذت فكرة الفصيلة CLASS من الواقع فكل عنصر من عناصر الحياة عبارة عن فصيلة class ، فأنت تستطيع أن تطلق علي جميع السيارات إنها من فصيلة CAR مع بعض الاختلافات ، ويمكنك أن تطلق على الطيور فصيلة Birdأى طائر وهكذا تنتمى جميع العناصر إلى فصائل classes ،وكل فصيلة تستطيع تمثيلها بعنصرين هما البيانات والدوال (methods , data,) ، فمثلاً فصيلة الموظف Employee بياناتها هي بيانات الموظف العامة مثل كود الموظف ، اسم الموظف ، عنوان الموظف ، تليفون الموظف ، وباقي بياناته ، وكذلك الدوال (methods) هي دوال تحقيق العمليات التي يمكن أن تتم على الموظف مثل :عملية إضافة موظف جديد ، وحذف موظف موجود ،تعديل بيانات موظف وجميع العناصر يمكن تمثيلها بهذه الطريقة



إذن الفصيلة هي مجموعة من السطور التي تمثل عنصراً تمثيل تاماً من حيث بيانات العنصر والتي تسمى خصائص properties ،وكذلك دوال العنصر التي تسمى methods وبتقسيم البرنامج إلي فصائل يصبح اكثر نظاماً وأسرع في الإعداد ، حيث قامت ميكروسوفت في منتجها vb.net بإعداد مجموعة كبيرة من الفصائل التي تلي متطلبات المبرمج، وما عليك عند إعداد البرامج إلا أن تدرس مكتبة الفصائل class Lib الموجودة لتعرف المتوفر منها ،وكذلك تتعرف علي بيانات (خصائص) properties ودوال الفصيلة Methods

وما شاهدناه فى الفصول الأولي من الأوامر التي يكتبها vb.net عند إنشائك لبرنامج جديد هو استعمال فصائل كل عنصر تستعمله ،

فمثلاً عند إضافتك لنموذج (form) تلاحظ أنه بستورث الفصيلة form وبالمثل عند إضافتك لزر الأمر button1 يتم استوراث الفصيلة button وهكذا

وهذا الأسلوب لم يكن واضحاً في الإصدار السابق vb6 حيث كنت تستعمل ذلك بدون الشعور بالفصائل ولا التعامل معها ولا التعديل فيها ولكن أتاح لك vb.net التعامل مع الفصائل للاستفادة من مفهوم OOP .

والسؤال كيف أستطيع إنشاء فصيلة CLASS والتعامل معها وتطبيق مفاهيم OOP؟

الإجابة : هذا ما سوف نوضحه في هذا الفصل بعد الانتهاء أولاً من توضيح المفاهيم الأساسية لمفهوم OOP

ما هي دوال البناء و دوال الهدم Construction & destruction

دوال البناء (constructors)

هي دوال تنفذ تلقائياً عند استعمالك للفصيلة class

(عند تعريف عنصر object) وهي تشبه حدث form – load الموجود في ال form ، تستعمل في تسجيل أي قيم ابتدائية أو تعريف أي متغيرات أو أي شروط ابتدائية0

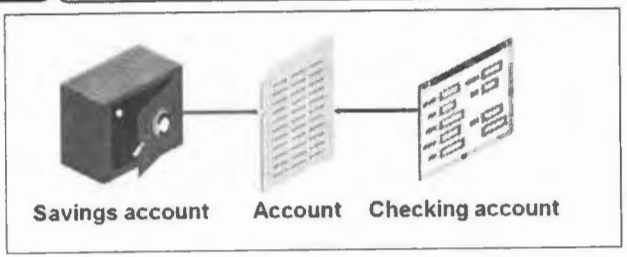
دوال الهدم destructors

هي دالة أو دوال تنفذ تلقائياً عند الانتهاء من استعمال الفصيلة class ويصبح هدف الفصيلة يشير إلي nothing .

وهي تشبه الحدث form-unload الموجود في الform وتستعمل للإنهاء أو التخلص من تعريف متغيرات أو اجراء أى عمليات قبل الخروج من البرنامج

ما هي خاصية التوريث inheritance

معنى خاصية التوريث (inheritance) هو توريث فصيلة Base class قديمة موجودة بالفعل لفصيلة class جديدة عند إنشائها بحيث يضاف تركيب الفصيلة Base class القديمة من بيانات و دوال إلي الفصيلة الجديدة new class ثم نكمل عليها في الفصيلة الجديدة وهذه الخاصية من أهم خصائص مفهوم OOP فعلى أساسه تبنى مكتبات الفصائل Class Lib حيث يتم بناء فصيلة أساس Base class ثم تستورثها الفصيلة الثانية والثالثة وتأخذ الفصائل من بعضها البعض حتى تتكون مكتبة فصائل عبارة عن شجرة فصائل Class Tree وهذا هو الحال في مكتبة الفصائل الشهيرة MFC الخاصة بميكروسوفت فى لغة Visual C++وكذلك مكتبة فصائل لغة Java المعروفة بالاسم JFC كما يظهر ذلك من الشكل



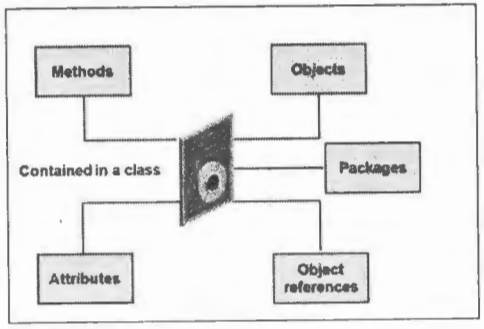
فى هذا المثال تلاحظ وجود class رئيسى وهو Account ثم عمل توريث منه إلى نوعين

ما هي الخاصية Overloading

معنى خاصية overloading هو إمكانية إنشاء اكثر من دالة Method بنفس الاسم مع تغيير عدد المعاملات Parameters مثلاً ،وهذا يفيد بإنشاء اكثر من دالة لنفس الوظيفة بمعاملات مختلفة بنفس الاسم ، مثل إنشاء دالتين بالاسم (find cust إحداهما تأخذ معامل رقمي هو كود الموظف ،والثانية تأخذ عبارة حرفية هي اسم الموظف وفي الحالتين تبحث الدالة ،و بالتالي يشعر المبرمج كأنها دالة واحدة ولكنهما دالتين يتم استدعاء كل واحدة تلقائياً حسب المعامل المرسل لها

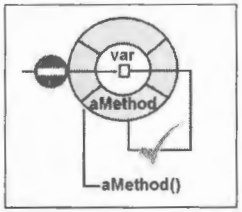
صديقى :
حاول تنظر حولك وتدبر ستجد أن كل شئ هو Object من فصيلة class
حاول مراجعة كل العناصر ال Objects من حولك وتحديد الفصيلة class التابع لها

وتأخذ الفصيلة class ومكوناتها العناصر التالية



معنى Encapsulation

وهى خاصية تعنى إنشاء الفصيلة Class به متغيرات ودوال ثم يتم استعمالها بدون ضرورة التعرف على مكونات الفصيلة ويظهر ذلك كما فى الشكل



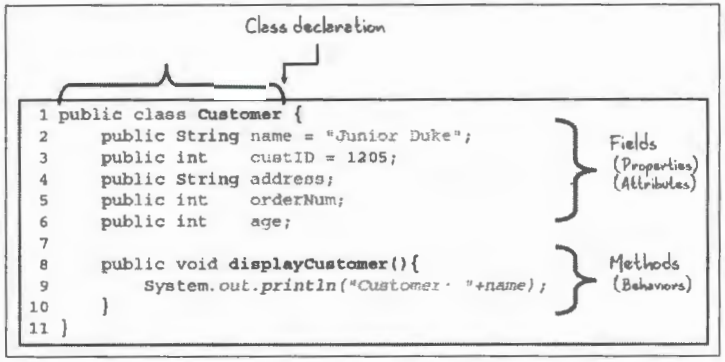
بحيث تكون المتغيرات Private والدوال public

إنشاء واستعمال الفصائل creating & Use class

بعد شرح بعض المفاهيم المهمة لمفهوم OOP تعال معنا نشرح عملياً كيفية إنشاء الفصائل CLASSES وتحقيق هذه المفاهيم .

الفصيلة class هى أساس مفهوم OOP لذلك أول ما تعلمه فى هذا المفهوم هو كيفية إنشاء واستعمال المصفوفة class

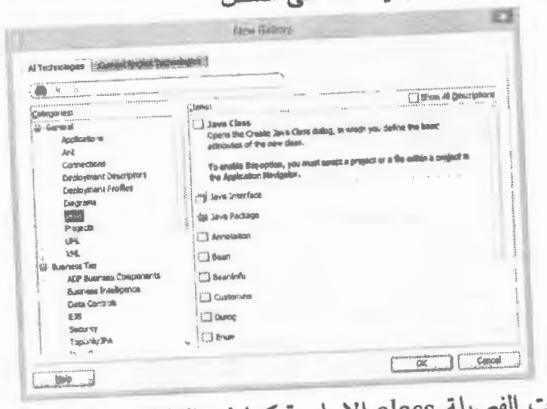
وتكون مكونات class كما فى الشكل



ولتحقيق ذلك تابع معى الخطوات التالية:

1- قم بإنشاء تطبيق جديد

2- قم بإنشاء فصيلة جديدة كما فى الشكل



ثم حدد مواصفات الفصيلة class الاساسية كما فى الشكل

3- اكتب سطور أبسط برنامج Java كما فى الشكل

```
1  package project1;
2
3  public class Class1 {
4      public static void main(String[] args)
5      {
6
7          System.out.println("Hello World ...");
8
9      }
10 }
11
```

فى هذه السطور تلاحظ ..انها أبسط أشكال برامج لغة Java ويتكون من فصيلة class واحدة وهى الفصيلة الرئيسية ، وبالتالى لابد أن يحتوى برنامج الـ Java على فصيلة واحدة على الاقل وبها الدالة main() التى يبدأ منها تنفيذ البرنامج

فى السطر رقم 3 يبدأ تعريف الفصيلة class بالاسم Class1 مع استعمال الكلمة المحجوزة class

فى السطر رقم 4 الدالة الرئيسية داخل الفصيلة main()

مثال 2 :

المثال التالى يتناول موضوع الفصائل classes بشكل أكثر ايضاحا كما فى السطور التالية

1- قم بإنشاء مشروع جديد

2- قم بإنشاء class جديد كما فى الشكل

3- قم بتعديل بيانات الكود لتصبح كما فى الشكل

```
1.  package project1;
2.  public class MathClass
3.  {
4.  int a,b,c;
5.  public void printABC()
6.  {
7.  a=10;
8.  b=20;
9.  c=30;
10. System.out.println("a="+a);
11. System.out.println("b="+b);
12. System.out.println("c="+c);
13. }
14. }
```

- فى هذا الشكل فى السطر رقم 1 تم إنشاء فصيلة بالاسم Mathclass وذلك باستعمال الكلمة المحجوزة class حتى يمكن التعامل مع الفصيلة class1
- فى السطر رقم 3 تم الاعلان عن المتغيرات a,b,c, من النوع Integer مع وضع كلمة public قبلها حتى يمكن التعامل مع هذه المتغيرات مباشرة من خارج الفصيلة class1 وسوف يتم توضيح هذه النقطة

- فى السطر رقم 5 تم إنشاء Method من النوع Sub اجراء بالاسم Printabc() مع وضع كلمة public قبلها حتى يمكن التعامل مع هذا الاجراء مباشرة من خارج الفصيلة class1 وسوف يتم توضيح هذه النقطة وداخل سطور هذا الاجراء تم استعمال الدالة Writeln() ثلاث مرات لطباعة قيم المتغيرات a,b,c, كما فى السطور 10.11,12

بهذا تم إنشاء فصيلة جديدة Class بالاسم class1 مع تعريف متغيرات data (a,b,c) ودالة Printabc() كأعضاء لهذة الفصيلة class1

من فضلك قم بمراجعة سطور إنشاء الفصيلة class1 جيدا وحاول استيعاب كيفية إنشاء وكتابة سطورالفصيلة class

والخطوة التالية هى كيفية استعمال الفصيلة الجديدة class1 لتوضيح ذلك تابع الخطوات التالية:

عد إلى البرامج وداخل سطور الدالة الرئيسية Main() اكتب سطور استعمال الفصيلة class1 كما فى الشكل

```
package project1;
public class Class1 {

  public static void main(String[] args) {
     MathClass obj1=new MathClass();
     obj1.printABC();
  }
}
```

فى هذا الشكل تم استعمال الفصيلة كما يلى:

فى السطر رقم 22 تم الاعلان عن متغير بالاسم nc من نوع الفصيلة class1 فهو فى هذه الحالة لا يسمى متغير بل يسمى هدف Object مع استعمال الكلمة new لإنشاء هدف object فعلى بالذاكرة وبالتالى تكون أول خطوة لاستعمال الفصيلة class هى تعريف هدف object من الفصيلة class كما فى السطر التالى:

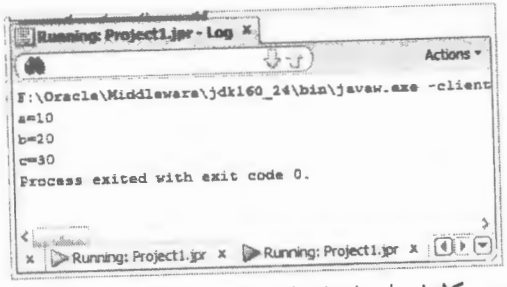

```
ClassName objectRef = new ClassName();
```

```
Myclass Obj1=new Myclass();
```

فى السطر رقم 5 تم وضع القيمة 20 فى المتغير a التابع للفصلة class1 ولكن مع الهدف nc وبالتالى أى تعامل مع الفصيلة لا يتم مع اسمها بل يتم مع اسم الهدف المعرف منها مع ملاحظة امكانية تعريف أكثر من هدف وبالتالى تغيير القيم فى كل هدف

فى السطر رقم 23 تم استدعاء الدالة Printabc() ولكن مع اسم الهدف من الفصيلة وهو nc حيث لا يصلح التعامل مع أى عضو داخل الفصيلة الاعن طريق متغير الهدف المعرف من الفصيلة

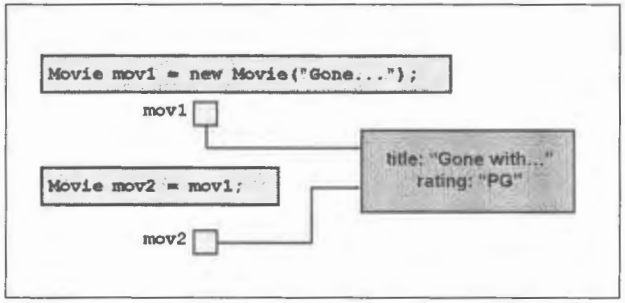
قم بتنفيذ البرنامج تلاحظ استدعاء السطور المكتوبة داخل الدالة الرئيسية وهى تعريف هدف من الفصيلة واعطاء قيم لمتغيرات الفصيلة ثم طباعة هذه القيم باستدعاء الدالة Printabc() كما فى الشكل



صديقى حاول التدرب كثيرا على إعداد الفصائل classes بما تحتويه من بيانات Data Members ودوال اعضاء Methods وكذلك التدريب على استعمال هذه الفصائل classes بتعريف متغير object ثم استدعاء الدوال الاعضاء Methods

استعمال Object Refrance

يمكن استعمال متغير Object Refrance للاشارة إلى نفس Object كما فى الشكل



ويمكن اعطاء متغير العنصر Object Refrance القيمة null لالغاء الاشارة إلى العنصر كما فى السطر التالى

```
Obj1=null;
```

ولزيادة توضيح إنشاء class والتعامل معه تابع معنا المثال التالى:

1- قم بإنشاء تطبيق جديد بالخطوات المعتادة كما سبق

2- قم بكتابة سطور إنشاء فصيلة class جديدة كما فى الشكل

```
Mathoperations.java *
1  class operations
2  {
3
4  }
5
6
7  public class Mathoperations {
8
9      public static void main(String[] args) {
10
11     }
12 }
```

فى هذا الشكل تم إنشاء فصيلة جديدة بالاسم mathoperations وذلك بالاستعمال الكلمة المحجوزة class وتلقائيا تم اغلاق هذه الفصيلة بالعبارة End class

3- داخل سطور الفصيلة الجديدة mathoperations قم بتعريف متغيرات جديدة كما فى السطر التالى

```
blic a, b, c As Double
```


4- داخل سطور الفصيلة الجديدة mathoperations قم بإنشاء دوال العمليات الحسابية فقم بتعريف دالة لعملية الجمع واخرى لعملية الطرح وعملية الضرب وعملية القسمة

لتصبح الفصيلة الجديدة mathoperations كما فى السطور التالية

```
Mathoperations.java *
1  class operations
2  {
3     public double a,b,c;
4
5     public double sum(double v1,double v2)
6     {
7        double res;
8        res=v1+v2;
9        return res;
10       }
11    public double sub(double v1,double v2)
12    {
13       double res;
14       res=v1-v2;
15       return res;
16       }
17    public double mul(double v1,double v2)
18    {
19       double res;
20       res=v1*v2;
21       return res;
22       }
23    public double div(double v1,double v2)
24    {
25       double res;
26       res=v1/v2;
27       return res;
28       }
29    }
30
31
32 public class Mathoperations {
33
34    public static void main(String[] args) {
35
36    }
37 }
```

راجع هذه السطور تلاحظ اننا أنشأنا فصيلة جديدة وبها مجموعة دوال العمليات الحسابية التى أشرنا اليها

5- قم باستعمال الفصية الجديدة واكتب به السطور التالية:

```
Mathoperations.java *
1   class operations
2   {
3     public double a,b,c;
4     public double sum(double v1,double v2)
5       {
6       double res;
7       res=v1+v2;
8       return res;
9           }
10    public double sub(double v1,double v2) {
11    double res;
12    res=v1-v2;
13    return res;
14          }
15    public double mul(double v1,double v2)
16  {
17    double res;
18    res=v1*v2;
19    return res;
20      }
21     public double div(double v1,double v2)
22  {
23    double res;
24    res=v1/v2;
25    return res;
26      }  }
27   public class Mathoperations {
28     public static void main(String[ ] args) {
29  operations obj=new operations();
30  double sumRes,subRes,mulRes,divRes;
31    sumRes=obj.sum(100,200);
32  subRes=obj.sub(300,100);
33  mulRes=obj.mul(10,20);
34  divRes=obj.div(200,5);
35    System.out.println("sumRes="+sumRes);
36  System.out.println("subRes"+subRes);
37  System.out.println( "mulRes="+mulRes);
38  System.out.println("divRes="+divRes);
39    }}
```

فى هذه السطور

فى السطر رقم 1 تم تعريف هدف object بالاسم mathobj من الفصيلة الجديدة mathoperations وذلك باستعمال كلمتى dim وNew كما سبق وشرحنا

فى السطر رقم 2 تم الاعلان عن ثلاثة متغيرات a,b,c من النوع Double

فى السطر رقم 3و4 تم وضع قيمة محتوى مربعى النص TextBox1,TextBox2 فى المتغيرين a,b وذلك باستعمال الدالة ()val

فى السطر رقم 5 تم استدعاء الدالة ()sum مع هدف الفصيلة mathobj مع ارسال معاملين هما a,b لجمع المتغيرين a,b ووضع النتيجة فى المتغير c

فى السطر رقم 6 تم وضع قيمة المتغير c وهو نتيجة الجمع فى مربع النص TextBox3 بهذا تم استعمال الدالة ()sum المعرفة داخل الفصيلة mathoperations

دوال البناء Constructors

من الخصائص المتوفرة فى مفاهيم البرمجة بالاهداف OOP فكرة وجود دالة البناء Constructor داخل الفصيلة class وهى دالة Method مثل أى دالة ولكن الفرق فى أنها دالة تنفذ تلقائيا بدون استدعاء أى بمجرد تعريف هدف object من الفصيلة class , والغرض من ذلك هو استعمال هذه الدالة فى تنفيذ أى عمليات أولية للفصيلة class مثل اعطاء قيم ابتدائية للمتغيرات و يتم تحديد دالة البناء Constructor داخل الفصلة بإنشاء دالة بنفس اسم الفصيلة أى تصبح كما فى بالشكل

```
Mathoperations.java  Constructor.java *
1  public class Constructor {
2      Constructor ()
3      {
4          System.out.println("this is the Constructor >>> it run without calling ");
5          }
6
7          public static void main(String[] args) {
8          Constructor obj=new Constructor();
9            }
10 }
11
```

نفذ البرنامج تلاحظ ظهور رسالة دالة البناء Costructor بالرغم من عدم استدعائه ولكن تم الاستدعاء تلقائيا وتحصل على النتيجة كما فى

```
this is the Constructor >>> it run without calling
Press any key to continue...
```

إنشاء أكثر من دالة باسم واحد Method Overloading

من المفاهيم المشهورة فى البرمجة بالاهداف OOP مفهوم إنشاء أكثر من دالة باسم واحد ويطلق عليه Method Overloading حيث يسمح هذا المفهوم بإنشاء أكثر من دالة بنفس الاسم إذا دعت الحاجة لذلك مثل إنشاء مجموعة دوال رسم بالاسم draw()

تسطيع رسم خط Line ومربع Box ودائرة circle وغيره

ولكن يتوقف ذلك على معاملات الدالة

ومثال أخر يمكن إنشاء أكثر من دالة بحث بالاسم Search() واحدة للبحث عن موظف بمعلومية رقم الموظف والثانية للبحث عن الموظف باسم الموظف ويتم استدعاء الأولى أو الثانية حسب المعامل الذى تم ارساله ولتوضيح فكرة إنشاء أكثر من دالة بنفس الاسم Mehod Overloading تابع المثال التالى:

1- قم بإنشاء تطبيق جديد بالخطوات المعتادة كما سبق

2- عدل سطور البرنامج وقم بإنشاء فصيلة جديدة بالاسم Shapes كما فى السطور التالية:

```
1  class shapes
2  {
3      public void drawline ()
4      {
5      for(int i=0;i<10;i++)
6      System.out.print("*");
7      System.out.println("*");
8      }
9
10     public void drawline (int n)
11     {
12     for(int i=0;i<n;i++)
13     System.out.print("*");
14     System.out.println("*");
15     }
16
17     public void drawline (int n,char ch)
18     {
19     for(int i=0;i<n;i++)
20     System.out.print(ch);
21     System.out.println("*");
22     }
23
24 }
25
26 public class ShapesApp {
27
28     public static void main(String[] args) {
29     shapes obj=new shapes();
30     obj.drawline();
31     obj.drawline(20);
32     obj.drawline(30,'-');
33
34 }
35 }
```


فى هذه السطور تم إنشاء فصيلة بالاسم Shapes وداخل هذه الفصيلة class تم تعريف ثلاث دوال بالاسم drawLine() كلها باسم واحد ولكن الملاحظ أن الفرق بينها هو المعاملات

الأولى بدون معاملات وتقوم بطباعة الحرف *عدد 20 مرة

الثانية بمعامل واحد هو n من نوع Integer ويستخدم لتحديد عدد مرات طباعة الحرف *

الثالثة لها معاملين الأول هو n من نوع Integer ويستخدم لتحديد عدد مرات طباعة الحرف * والثانى ch لتحديد الحرف المطلوب طباعته

بعد إنشاء الدوال الثلاثة داخل الفصيلة Shapes قم بكتابة أوامر استعمال الفصيلة Shapes ودوالها الثلاثة داخل الدالة الرئيسية للبرنامج main() كما فى السطور

فى السطر رقم 29 تم تعريف هدف Object من الفصيلة Shapes كما سبق

فى السطر رقم 30 تم استدعاء الدالة drawLine() مع هدف الفصيلة ولكن الملاحظ اننا لم نكتب معاملات للدالة لذلك فأننا نستدعى الدالة الأولى التى لا تأخذ معاملات

فى السطر رقم 31 تم استدعاء الدالة drawLine(10) مع هدف الفصيلة ولكن الملاحظ اننا كتبنا معامل للدالة لذلك فأننا نستدعى الدالة الثانية التى تأخذ معامل واحد

فى السطر رقم 32 تم استدعاء الدالة drawLine(15, "-") مع هدف الفصيلة ولكن الملاحظ اننا كتبنا معاملين للدالة لذلك فأننا نستدعى الدالة الثالثة التى تأخذ معاملين

قم بتنفيذ البرنامج تحصل على نتيجة التنفيذ التى تعبر عن البرامج كما فى الشكل التالى

قواعد يفضل الالتزام بها عند كتابة البرامج

الشكل التالى يعرض التركيب المثال للبرنامج من حيث اسم البرنامج والتعليقات وصاحب البرنامج وتاريخ كتابة البرنامج ومكتبات البرنامج وبيئة التطوير ومتطلبات التطوير ، من فضلك حاول مراجعة هذه القواعد وحاول الاسترشاد بها عند كتابة البرامج

```
//*************************************************************
//*************************************************************
// Program name
// Copyright (c) 200? by
// ALL RIGHTS RESERVED
//*************************************************************
//*************************************************************
// Date:                    Coded by:
// Filename:                Module name:
//                          Source file:
// Program description:
//
//*************************************************************
// Libraries and software support:
//
//*************************************************************
// Development environment:
//
//*************************************************************
// System requirements:
//
//*************************************************************
// Start date:
// Update history:
//          DATE          MODIFICATION
//
//*************************************************************
// Test history:
```



```
// TEST PROTOCOL        DATE      TEST RESULTS
//
//*************************************************************
// Programmer comments:
//
//
//*************************************************************
```

ملاحظات يجب مرعاتها عند تصميم الفصائل

1- حاول أن تحافظ على أن تكون البيانات من النوع ال private

2- حاول أن تعطى قيم ابتدائية للمتغيرات .initialize data

3- لا تستعمل متغيرات منفردة كثيرا فبدلا من ذلك حاول تجميعها فى فصيلة class

فمثلا بالنظر إلى المتغيرات التالية :

```
private String street;
private String city;
private String state;
private int zip;
```

يمكن تجميعها فى فصيلة بالاسم Address

4- حاول استخدام الصورة القياسية لتعريف الفصيلة class كما فى التركيب التالى:

```
public features
package scope features
private features
Within each section, we list:
instance methods
static methods
instance fields
static fields
```

5- حاول تقسيم الفصائل حسب الوظائف .. وتلاشى إنشاء فصيلة تتناول أكثر من موضوع ولتوضيح ذلك راجع هذا التصميم

```
public class CardDeck // bad design
{
```

```
  public CardDeck() { . . . }
  public void shuffle() { . . . }
  public int getTopValue() { . . . }
  public int getTopSuit() { . . . }
  public void draw() { . . . }
  private int[] value;
  private int[] suit;
}
```

فى هذا التصميم تلاحظ احتوى على بيانات أكثر من موضوع وهذا التصميم سئ بل يجب فصل الموضوعات بحيث لا تحتوى الفصيلة الا على موضوع واحد ،وبالتالى يصبح التصميم الجيد كما فى السطور

```
public class CardDeck
{
  public CardDeck() { . . . }
  public void shuffle() { . . . }
  public Card getTop() { . . . }
  public void draw() { . . . }
  private Card[] cards;
}

public class Card
{
  public Card(int aValue, int aSuit) { . . . }
  public int getValue() { . . . }
  public int getSuit() { . . . }
  private int value;
  private int suit;
}
```

6- حاول اعطاء أسماء للفصائل والمتغيرات معبرة عن الغرض منها كمات يجب أن تلاحظ أنه من المشهور إنشاء الدوال METHODS بالاسماء SET_() و GET() حيث أن مجموعة دوال Set_() تقوم باعطاء قيم لمتغيرات الفصيلة ومجموعة دوال Get_() تقوم باعادة قيم متغيرات الفصيلة

تنظيف الذاكرة Reclaiming Memory

تقوم لغة Java بمسح المتغيرات الغير مستعملة من الذاكرة حتى لا تملئ الذاكرة، ويتم ذلك تلقائيا الا أنه يمكنك توجيه الامر بذلك كما فى السطر التالى

```
System.gc();
```

الدالة finalize()

يمكن إنشاء دالة بالاسم finalize() واستدعائها عكس constructor بعد الانتهاء من استعمال Object وبالتالى تكون أخر أوامر تنفذ ، وبالتالى يمكن استغلالها فى مسح المتغيرات واغلاق الملفات ، والشكل القادم يوضح مثال لهذه الدالة

```
public class Movie {

  public void finalize() {
    System.out.println("Goodbye");
  }
}
```

⌘⌘⌘

الفصل التاسع

خاصية التوريث

INHERITANCE

معنى خاصية التوريث Inheritance

هي خاصية قوية جداً في مفهوم البرمجة باستعمال الأهداف OOP وتستعمل لتقليل إعادة كتابة الأوامر ، وهي أنه باستعمال خاصية التوريث تستطيع إنشاء فصيلة (CLASS) تحتوى على خصائص Proprities و دوال methods ثم استعمالها كأساس لفصائل أخرى وبالتالي لا نحتاج لكتابة ما كتبناه في تعريف الفصيلة الأولي (والتي تسمىbasic class) مع كل فصيلة جديدة .

وتسمى الفصائل الجديدة التي تستورث فصيلة قديمة فصائل مشتقة derived class وغالباً يتم التعامل مع الفصائل classes بأن نستورث فصيلة ونكمل عليها .

ولتوضيح ذلك نضرب لك المثال التالي:

نفرض انك تريد تعريف فصيلتان الأولي لتعريف بيانات الطالب و الثانية لتعريف بيانات المدرس فبدون استعمال خاصية التوريث سنضطر لإنشاء الفصيلتان برغم من تشابه الفصيلتان واحتوائهما على بيانات مُتشابه فمثلاً فصيلة الطالب تحتوى على البيانات التالية :

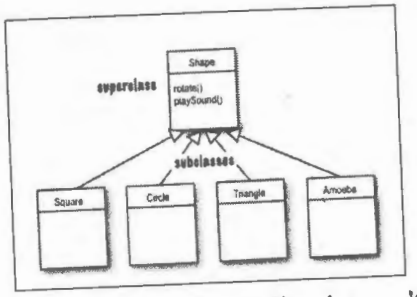
الكود ، الاسم ، العنوان ، التليفون ، المؤهل ، تاريخ الميلاد ، وكثير من البيانات

وتحتوى على الدوال التالية :

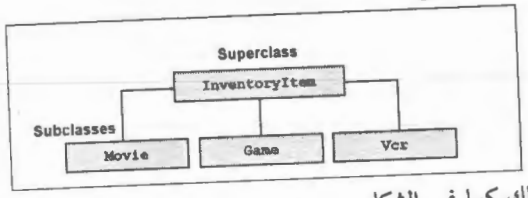
- دالة تسجيل بيانات الطالب
- دالة حذف بيانات الطالب
- دالة تعديل بيانات الطالب (ودوال أخرى)

وعند النظر إلي فصيلة المدرس سوف تجد أنها تشتمل علي كثير من البيانات و الدوال الأعضاء في فصيلة الطالب بالاضافة لبعض الاعضاء الجديدة وبالتالي باستعمال خاصية التوريث علينا توريث فصيلة الطالب لفصيلة المدرس وإضافة الجديد إليها مما يوفر علينا إعادة كل شيء ويمكن تكرار هذه الفكرة مع فصائل جديدة أخرى مما يوفر

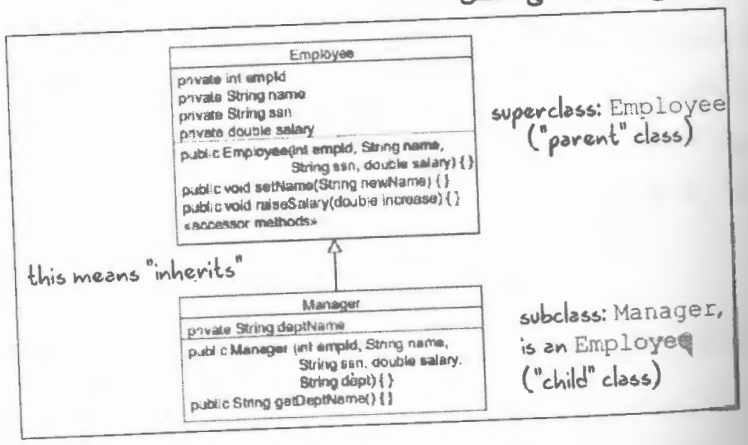
الكثير من الوقت بالاضافة أن هذا الاسلوب يسمح لك باستعمال فصائل classes تم إعدادها وتم اختبارها



فى هذا المثال تلاحظ وجود فصيلة رئيسية superclass وهى الفصيلة Shape ثم توريثها لفصائل فرعية subclasses تأخذ المواصفات العامة لها مثل الفصيلة Square و Circle ولزيادة التوضيح الشكل التالى يعرض مثال أخر للتوريث كما تعرضه المادة التعليمية Oracle كما يلى



ويظهر تطبيق ذلك كما فى الشكل



التوريث في Java

يتم إنشاء فصائل كاملة بكل دوالها (تعتبر كفصائل أساس Base class) ثم توريث هذه الفصائل بكل دوالها ومتغيراتها لفصائل جديدة وذلك يوفر عليك الكثير وهذا يعني أنه يمكنك أن تنشأ form جديدة بناء علي form قديمة لأن form في حد ذاتها فصيلة.

1- ابدأ تطبيق كما سبق

2- اكتب سطور تعريف فصيلة class بالاسم Person كما فى السطور التالية:

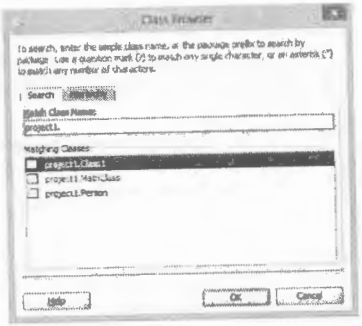
```
class Person
{
  public String Pname,Paddress;
  public void Set_Data(String Name_V,String address_V)
    {
      Pname=Name_V;
      Paddress=address_V;
    }
  public String Get_Name ( )
    {
      return Pname;
    }
  public String Get_Address ( )
    {
      return Paddress;
    }
}
public class InheritApp1 {
  public static void main(String[] args) {
  }
}
```

في هذه السطور تم إنشاء فصيلة بالاسم person وتم تعريف متغيرات وتم كتابة فصيلة متكاملة تحتوى على المتغيرات Pname, Paddress وتحتوى على الدالة () SetData التى تقوم بتسجيل بيانات الفصيلة وتحتوى على الدالة ()getName التى تعيد قيمة المتغير pname الخاص بالفصيلة والخاص باسم الشخص وكذلك الدالة ()getAddress التى تعيد قيمة المتغير paddress وعند إنشاء فصيلة جديدة وتوريثها هذه الفصيلة Person يتم استعمال متغيراتها ودوالها مباشرة دون الحاجة إلي إنشائها مرة أخرى .

يمكن توريث هذه الفصيلة لفصيلة جديده بطريقتين

1. استعمال معالج JDeveloper فى إنشاء الفصيلة والتوريث فى نفس الوقت وذلك كما فى الشكل

ويمكن عرض فصائل الحزمة project1 بالضغط على رمز العدسة المجاور Extends ثم نكتب .project1 تظهر فصائل هذه الحزمة ثم نختار منها كما فى الشكل



فيتم إنشاء الفصيلة الجديدة والتوريث فى نفس الوقت وتحصل على الفصيلة الجديدة كما فى الشكل

```
package project1;
public class Students extends Person
{

}
```

كما يمكن كتابة سطور فصيلة جديدة بالاسم student مع استعمال الامر extends كما يلي:

```
class Student extends Person
{
 int degree;
    public void Set_Degree(int degree_V)
    {

     degree=degree_V;

    }
 pubic int RetDegree()
 {
   return degree;
 }

 }
```

في هذه السطور

تم إنشاء فصيلة جديدة بالاسم student

وتم استعمال العبارة extends person ومعناها قم بتوريث الفصيلة person بكل ما فيها للفصيلة الجديدة (student) و بالتالي أصبحت نفس إعضاء الفصيلة القديمة person اعضاء فى الفصيلة الجديدة student و هذا يسمح لك باستعمالها مباشرة وذلك كما يلي :

فى الدالة الرئيسية Main() قم بكتابة سطور استعمال كلا من الفصيلتين person,student كما فى السطور التالية:

```
MethodOperations.java  ShapesApp.java  InhertApp1.java
public class InhertApp1 {

    public static void main(String[] args) {
        String vname,vaddress,
        Person no=new Person();
        no.Set_Data("Azab","Cairo");
         vname=no.Get_Name ( );
          vaddress=no.Get_Address ( );
          System.out.println(vname);
          System.out.println(vaddress);

    Student st=new Student();
    st.Set_Degree(100);
    int vd;
    String vaddress2,
    vd=st.RetDegree(),
    st.Set_Data("Omar","Giza"),

        vname=st.Get_Name ( );
          vaddress2=st.Get_Address ( );

          System.out.println("vd:"+vd);
        System.out.println("vname:"+vname),
          System.out.println("vaddress:"+vaddress);

          }
}
```

في هذه السطور

في السطر رقم 45 تم تعريف المتغيرات vname, vaddrs

في السطر رقم 46 تم تعريف المتغير no(هدف) في الفصيلة person

فى السطور 47و48و49 تم التعامل مع هدف الفصيلة person باستدعاء الدوال والتعامل معها كما سبق

في السطر رقم 53 تم تعريف المتغير st من الفصيلة student

فـي السـطر رقـم 54 تـم استدعاء الدالـة الجديـدة المضـافة للفصـيلة بالاسـم setdegree

في السطر رقم 55و56 تم الاعلان عن متغيرات

فـي السـطر رقـم 58 تـم اسـتدعاء الدالـة SetData() المعرفـة داخـل الفصـيلة الاساسية Person فبالرغم من عدم وجود الدالة داخل الفصيلة الجديدة الا اننا استعملناها لانها معرفة فى الفصيلة التى استورثناها بالاستعمال الامر Inherits

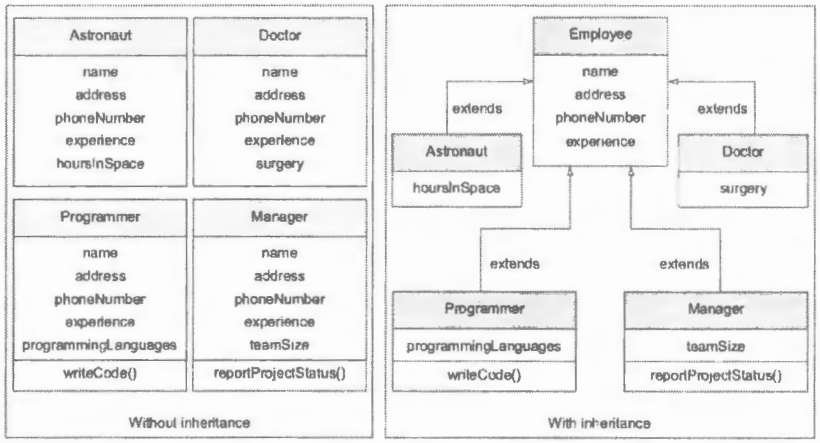
في السطر رقم 60و61 استدعاء دوال الفصيلة الاساسية Person فبالرغم من عدم وجود هذه الدوال داخل الفصيلة الجديدة الا اننا استعملناها لانها معرفة فى الفصيلة التى استورثناها بالاستعمال الامر Inherits

من هذا المثال تلاحظ استعمال الخصائص المستورثه من الفصيلة person مع الفصيلة student بالرغم من عدم تعريفها أو عدم الإعلان عنها

نفذ البرنامج تحصل على نتيجة التنفيذ كما فى الشكل

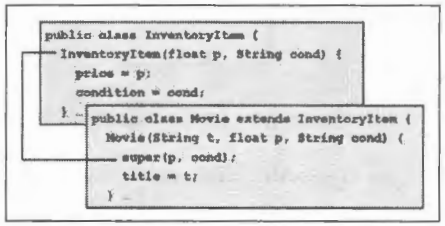
```
Azab
Cairo
vd:100
vname:Omar
vaddress:Cairo
Press any key to continue...
```

والشكل التالى يوضح الفرق بين استعمال خاصية الوراثة Inhertance وعدم استعمالها



استعمال Super

يمكن استعمال كلمة super للاشارة إلى الفصيلة الاساسية superClass لاستدعاء دالة منها كما فى الشكل



```
public class InventoryItem {
  InventoryItem(float p, String cond) {
    price = p;
    condition = cond;
  }
  public class Movie extends InventoryItem {
    Movie(String t, float p, String cond) {
      super(p, cond);
      title = t;
    }
```

فى هذا الشكل يتم استعمال super() لاستدعاء دالة البناء constructor الخاص بالفصيلة الرئيسية مع ارسال المعاملات لها

تغيير الدوال في الفصيلة الجديدة overriding Methods

من الإمكانيات المتاحة في مفهوم OOP البرمجة بواسطة الأهداف خاصية تسمى Overriding وهذه الخاصية تعنى إمكانية كتابة دوال جديده في الفصائل الجديدة بنفس اسم الدوال الموجودة في الفصيلة الأساس (Base class) التي تم توريثها أي أن الدالة موجودة في الفصيلة الأساسية ومع ذلك تم إنشائها مرة أخرى بنفس الاسم في

الفصيلة الجديدة و بالتالي يتم إلغاء الدالة من الفصيلة الجديدة واعتماد الدالة الجديدة وهذا يسمى تركيب دالة علي أخرى overriding ولتوضيح ذلك ننشأ فصيلة بها دالة ثم نستورث هذه الفصيلة و ننشأ الدالة السابقة في الفصيلة الجديدة و يظهر ذلك من السطور التالية:

Override.java

```
1  class One
2  {
3      public void Say()
4      {
5      System.out.println("This Msg from class One");
6
7      }
8  }
9  class Two extends One
10 {
11     public void Say()
12     {
13     System.out.println("This Msg from class Two");
14
15     }
16 }
17 public class Override {
18     public static void main(String[] args) {
19
20
21     }
22 }
```

في هذه السطور

في السطر رقم 1 تم تعريف فصيلة جديدة بالاسم one

في السطر رقم 3 تم تعريف دالة بالاسم ()say مع وضع الكلمة overiridable أمام الدالة وهذا يعنى إمكانية التركيب عليها أي إنشاء دالة بنفس الاسم في الفصيلة المشتقة ، ثم تم إنشاء الدالة ()Say فى الفصيلة الجديدة برسالة جديدة وإنهاء الفصيلة .

في السطر رقم 9 تم إنشاء فصيلة بالاسم two

في السطر رقم 11 تم إنشاء دالة للفصيلة الجديدة بنفس الاسم say وهو اسم الدالة الموجودة في الفصيلة الأساس التي تم توريثها (one) مع وضع الكلمة overrides أمام الدالة ليعنى انها بديل الدالة موجودة في الفصيلة الأساس .

في السطر رقم 9 تم توريث الفصيلة القديمة one .

بهذا يصبح للفصيلة الأساس (one) دالة بالاسم say

ويصبح للفصيلة الجديدة (two) دالة بالاسم say

وعند تعريف متغير من كل منهما واستدعاء الدالة say يتم استدعاء الدالة الخاصة بكل منهما بالرغم من وجود خاصية التوريث بها.

ولتوضيح ذلك تابع ما يلي :

بعد كتابة سطور الفصيلتان اكتب السطور التالية فى الدالة الرئيسية للبرنامج Main()

```
17  public class Override {
18
19      public static void main(String[] args) {
20
21          One a=new One();
22          Two b=new Two();
23          a.Say();
24          b.Say();
25      }
```

في هذه السطور

في السطر رقم 22،21 المتغيرات a , b من الفصائل one ,two

في السطر رقم 23 تم استدعاء الدالة say مع المتغير a لمأخوذ في الفصيلة one

وبالتالي يتم استدعاء الدالة الأولي وتظهر الرسالة الأولي

في السطر رقم 24 يتم استدعاء الدالة say مع المتغير a المعرف من الفصيلة b

وبالتالي يتم استدعاء الدالة الثانية وتظهر الرسالة الثانية بالرغبة في توريث الفصيلة الأولي المحتوية علي الدالة say للفصيلة الثانية الإ أن إنشاء دالة جديدة بنفس الاسم أدي إلي إلغاء الدالة say مع الفصيلة الجديدة واستعمال الدالة الخاصة بها.

نفذ البرنامج تحصل على نتيجة التنفيذ كما فى الشكل

```
This Msg from class One
This Msg from class Two
Press any key to continue...
```

الأمر abstract

هذا الامر عندما يوضع أمام دالة في الفصيلة فإنه يعني عدم إمكانية استدعاء هذه الدالة مباشرة بل لابد من تعريف دالة جديده بنفس الاسم حتى يمكن استعمالها وبالتالي

فهذه الدالة تكتب للتعريف فقط أى لابد من تحقيق خاصية الـ Method Overriding

ويتضح ذلك من السطور التالية:

```
abstract class WashingMachine
{
  public WashingMachine()
  {
    // Code to initialize the class goes here.
  }
  abstract public void Wash();
  abstract public void Rinse(int loadSize);
  abstract public long Spin(int speed);
}
```

```
class MyWashingMachine extends WashingMachine
{
  public MyWashingMachine()
  {
    // Initialization code goes here.
  }
  override
// public void Wash()
  {
    // Wash code goes here.
  }

// override
public void Rinse(int loadSize)
   {
      // Rinse code goes here.
   }

//   override
public long Spin(int speed)
   {
      // Spin code goes here.
   }
}
```

متى تستعمل خاصية التوريث ومتى لا نستعملها ؟

هذا السؤال ربما يرد إلي ذهنك عندما تعمل مع الفصائل وإجابته كما يلي:

إذا كانت الوظيفة المطلوبة تحقيقها ليست بكبيرة بحيث لا تحتاج فصيلة جديدة تقوم فيها بإنشاء دوال و تعريف متغيرات فلا تستخدم خاصية التوريث بل تكتب الأوامر التي تحقق العملية فقط ،مثال لذلك إذا كان لديك أداة نص textbox وتريد تحويل لون الأرقام السالبة عندما تكتب به إلى اللون الأحمر فلا تقوم بتوريث فصيلة textbox إلى فصيلة جديدة ،وتعريف دالة لذلك ،ولكن اكتب الأوامر التي تحقق العملية لأنها لا تحتاج إلي فصيلة .

إذا كانت الفصيلة الجديدة (التي تستورث فصيلة قديمة) بها لن تستفيد كثيراً من الفصيلة الأساس (المستورثة) فقط سوف تستورثها لإنشاء دوالها من جديد بخاصية override فهذه الحالة يكون التوريث فيها غير جيد،

ولكن يمكنك إنشاء فصيلة بدوال بدون كتابة سطور الدوال (interface) ثم توريثها و بالتالي ننشأ الفصائل الجديدة حسب تركيب (interface) وخاصة إذا كان هناك أكثر من فصيلة سوف نشترك في التركيب فقط،

إذا كانت الفصيلة الجديدة (التي تستورث) تحتاج إلي دوال الفصيلة القديمة بنفس السطور المكتوبة فيها وتحتاج إلي الخواص والمتغيرات ولكن تريد الإضافة إليها فهذه هي الحالة المناسبة للتوريث وهو الاستفادة الكبيرة من الفصيلة المستورثها وإضافة الجديد بما يناسب العمل .

تعدد صور الدوال مع الاهداف Polymorphism

تتناول أحد خصائص البرمجة الشيئية OOP وهى خاصية تعدد صور الدوال مع الاهداف Polymorphism وذلك من خلال النقاط التالية:

معنى Polymorphism

طرق التحكم فى التوصل للمتغيرات Variables Access Control

الكبسلة Encapsulation

طرق التحكم فى التوصل للدوال Member Method Access Control

دوال التوصل Accessor Methods

طرق التحكم فى التوصل للفصائل

نظرية التجريد Abstraction وعلاقته بالفصائل

هى أحد خصائص البرمجة بالاهداف OOP ومعناها إمكانية استعمال متغير (هدف) الفصيلة الاب Base class للاشارة إلى الفصائل الوارثة للفصيلة الاب ، وتسمى (also called dynamic binding or late binding or run-time binding)

ويمكن القول أن الخاصية polymorphism معناها باختصار هو اسم واحد بأكثر من شكل ، وبالطبع يوجد أشكال متقاربة من هذا المعنى ناقشناها من قبل مثل :

- Method overloading
- Method overriding through inheritance
- Method overriding through the Java interface

وجود أكثر من دالة بنفس الاسم داخل نفس الفصيلة مع اختلاف المعاملات من حيث النوع أو العدد ووهذا ما نسميه الـ Function Overloading مع ملاحظة أن الدوال Methods موجودة فى نفس الفصيلة 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.  }//end method m(String y)
8.  }
```

فى هذه السطور

تم إنشاء دالتين بالاسم m() ، الأولى بمعامل من النوع int والثانية بمعامل من النوع string

وهذه هى نظرية إنشاء أكثر من دالة التى تناولناها من قبل ،وهى أحد صور تعدد الصور polymorphism

ويتم استدعاء سطور هذه الفصيلة ...كما فى 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. }
```

فى هذه السطور

تم تعريف متغير هدف var من نوع الفصيلة B ثم استدعاء الدالة ()m مرتين الأولى بمعامل رقم 3 وبالتالى استدعاء الدالة الأولى والثانية بمعامل string وهذا ما يسمى Function Overloading وكما أشرنا هو احد صور تعدد الصور Polymorphism وعند تنفيذ البرنامج تحصل على نتيجة التنفيذ التالية :

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

1- إنشاء دالة بنفس اسم ومعاملات الدالة الموجودة فى فى الفصيلة الام Base class وذلك بعد تحقيق عملية التوريث لفصيلة جديدة وذلك باستعمال خاصية التوريث Inheritance ويتم إنشاء الدالة الجديدة فى الفصيلة الجديدة Child class وهذا ما يسمى Function Overriding مع ملاحظة أن الدالة الجديدة تنشأ فى الفصيلة الابن Child class

2- والصورة الجديدة لتعدد الصور polymorphism هى نفس الصورة الثانية Function Overriding أى إنشاء دالة فى الفصيلة الجديدة بنفس اسم ومعاملات الدالة الموجودة فى الفصيلة الاصل Base class ولكن يزيد على ذلك طريقة استدعاء هذه الدوال ،حيث يتم استدعاء نفس الدالة ولكن مع

أهداف من فصائل مختلفة ترث نفس الفصيلة الاصل وبالطبع ليست واضحة تماما وسوف نقوم بتوضيح ذلك من خلال الفقرات التالية

مساواة الاهداف والتحويل بين الانواع Assignment compatibility and type conversion

قبل الاستمرار فى تناول موضوع تعدد الصور polymorphism يجب فهم موضوع مساواة الاهداف والتحويل بين الانواع ويمكن مساواة المتغيرات إذا كانت القيمة الموجودة فى المتغير الأول يمكن وضعها فى المتغير الثانى

التحويل بين الانواع Type conversion and the cast operator

يتم التحويل بين الانواع اما تلقائى automatically أو يتم بتحديد ذلك forced وذلك باستعمال مؤثر للتحويلcast operator المناسب للنوع وهذا المؤثر هو اسم النوع المطلوب التحويل اليه ويوضع بين قوسين فمثلا للتحويل إلى int يكتب مؤثر التحويل بالصورة

```
(int)
```

مع ملاحظة أن هذا التحويل ليس نهائيا ولا حقيقيا حيث يتم التحويل فقط أثناء العملية ولكن يبقى النوع الاصلى كما هو ، هو فقط يعامل معاملة النوع المطلوب التحويل اليه

مساواة متغيرات الاهداف Assignment compatibility for references

تختلف مساواة متغيرات الاهداف عن مساواة متغيرات البيانات الاساسية primitives حيث لا تصلح المساواة مع متغيرات الاهداف الا فى الحالات التالية:

1- متغيرين الاهداف variable reference من نفس النوع أى هما متغيرات Objects من نفس الفصيلة class

2- متغير الهدف variable reference من نوع الفصيلة الاب superclass لمتغير الهدف الاخر

3- متغير الهدف reference variable من نوع interface تم تحقيقه بفصيلة الهدف الثانى

4- متغير الهدف reference variable من نوع interface تم تحقيقه بفصيلة الاب superclass للمتغير الاخر

وبالتالى بعض عمليات مساواة متغيرات الاهداف لا تحتاج مؤثر التحويل cast operator

متغير الفصيلة Object

كما تعلم أن الفصيلة Object هى الاصل superclass لجميع الفصائل وبالتالى يعتبر متغير الهدف منه عام generic

مساواة متغيرات أهداف باستعمال Converting reference types with a cast

يجب استعمال مؤثر التحويل cast operator فى مساواة متغيرات االاهداف التى لا تحقق الشروط السابقة

مثال

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
}//end class Poly02
```

فى هذه السطور

تم الاعلان عن فصيلة جديدة بالاسم A ترث الفصيلة الاساسية للفصائل Object ثم فصيلة جديدة بالاسم B ترث الفصيلة A وكذلك فصيلة بالاسم C

ثم تم إنشاء الفصيلة الرئيسية NewClass التى تحتوى على الدالة الرئيسية ()main داخل الدالة الرئيسية ()main يتم تعريف متغير هدف بالاسم var من نوع الفصيلى الاب Object وتم استعماله فى الاشارة إلى هدف من الفصيلة B نظرا لأن الفصيلة Object هى الاب لكل هذه الفصائل

بعد تم داخل التعليقات الاشارة إلى أن الاستدعاء بالطريقة ;()//var.m لان متغير الهدف من نوع الاب الاقل فى التركيب ولكن الصحيح هو ;()((B)var).m وهكذا توضيح بالتعليقات ما يصلح وما لا يصلح

وهذا يوضح فكرة تحويل النوع أثناء المساواة

تحقيق عملية تعدد الصور أثناء تنفيذ البرنامج

يتم تحقيق عملية تعدد الصور أثناء تنفيذ البرنامج runtime polymorphism وذلك من خلال خاصية التوريث inheritance والخاصية method overriding ولتوضيح ذلك تابع معنا الخطوات التالية:

1. بافتراض أننا قمنا بتعريف فصيلة class بالاسم SuperClass وتعريف دالة بداخله بالاسم method
2. بافتراض أننا قمنا بتعريف فصيلة class بالاسم SubClass ولكن ترث الفصيلة السابقة SuperClass و تم تعريف دالة بداخله بالاسم method وبالتالى تحقيق الخاصية overrides
3. بافتراض أننا قمنا بتعريف متغير reference للفصيلة SubClass بالاسم ref ولكن استخدمناه للاشارة إلى هدف من الفصيلة SuperClass
4. بافتراض أننا قمنا باستدعاء الدالة method مع متغير الهدف ref بالصورة ref.method()
5. السؤال :هل يتم استدعاء الدالة ()method الاصلية التابعة للفصيلة الاب SuperClass أم يتم استدعاء الدالة ()method الجديدة التى عرفت فى الفصيلة SubClass
6. النتيجة: يتم استدعاء الدالة ()method الجديدة والموجودة فى الفصيلة SubClass وذلك لأن المتغير يشير إلى هدف من نوع هذه الفصيلة وهذه هى أشهرشكل لتعدد الصور polymorphism والتى تسمى runtime polymorphism ويطلق عليها أحيانا late-binding ويتم تحقيقها بنجاح مع وجود الخصائص class inheritance, interfaces, and method overriding.

ملحوظة:

تحدد الدالة التى يتم استدعائها فى حالة runtime polymorphism حسب الهدف الذى يشير اليه المتغير وليس نوع المتغير وتسمى runtime polymorphism حيث يتم تحديد الدالة وقت التنفيذ

والمثال التالى يوضح الخاصية runtime polymorphism مع وجود الخصائص class inheritance and overridden methods وذلك كما فى السطور التالية :


```
/*

This program illustrates downcasting
and polymorphic behavior

Program output is:

m in class B
m in class B
m in class A
**************************************/

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

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

public class PolyExam03{
  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 Poly03
//===================================//
```

فى هذه السطور

تم إنشاء ثلاث فصائل بالاسماء A,B,C وتم إنشاء دالة بالاسم ()m فى الثلاث فصائل مع توريث الفصائل مما يعنى وجود method Overriding

الفصيلة Object

Methods in the Object class

كما أشرنا من قبل أن الفصيلة Object هى الفصيلة الاب لجميع الفصائل فى لغة Java وهى موجودة على رأس شجرة الفصائل ، وتحتوى هذه الفصيلة على مجموعة من الدوال هى:

- clone()
- equals(Object obj)
- finalize()
- getClass()
- hashCode()
- notify()
- notifyAll()
- toString()
- wait()
- wait(long timeout)
- wait(long timeout, int nanos)

وبالطبع جميع الفصائل بشكل أو بأخر فأنها ترث هذه الدوال ومعظم هذه الدوال يفضل اعادة كتابتها بخاصية Method Overriding

مثال :

والمثال التالى يوضح أيضا كيفية استعمال الفصيلة Object الفصيلة الاب لجميع الفصائل classes لتوضيح نظرية تعدد صور الدوال polymorphism بين الفصائل 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. }
```

فى هذه السطور

فى السطر رقم 1 تم تعريف فصيلة class جديدة بالاسم A ترث الفصيلة الاب للفصائل Object

فى السطر رقم 4 تم تعريف فصيلة جديدة بالاسم B و ترث الفصيلة A ولكن تقوم باعادة كتابة الدالة ()tostring باستعمال الخاصية Function overriding كما اشرنا من قبل

بالمثل فى السطر رقم 10 تم إنشاء الفصيلة C التى ترث الفصيلة B ولكن تقوم باعادة كتابة الدالة ()tostring باستعمال الخاصية overriding Function كما اشرنا من قبل أيضا

فى السطررقم 16 تبدا الفصيلة الاساسية وبها الدالة الرئيسية ()main التى تستعمل الفصائل السابقة

وعند تنفيذ هذا البرنامج تحصل على نتيجة التنفيذ التالية:

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

تعدد الصور مع الـ Polymorphism and Interfaces

كما تعلم توجد امكانية توريث أكثر من Interfaces بما يقارب الوراثة المتعددة Multiple inheritance وكما تعلم يمكن للـ Interface أن يرث أكثر من Interface وكذلك لايحتوى الـ Interface الا على دوال Methods بدون سطور مجرد تعريف أى abstract متغيرات Variables وتكون ثوابت أى final

⌘⌘⌘

الفصل العاشر

بناء هيكل البرنامج باستعمال

Abstracrt Class
Interface