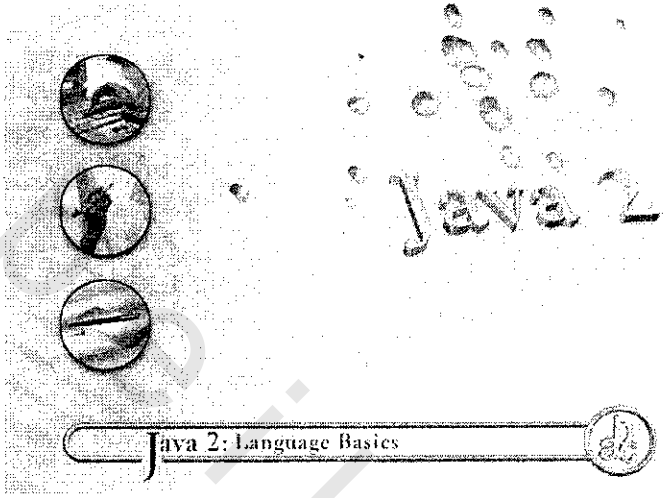
الفصل الثامن



قواعد التوصل للفصائل والدوال والمتغيرات
Access Modifiers

فى هذا الفصل سوف نتناول موضوع قواعد التوصل Access للفصائل Classes والدوال Methods والمتغيرات Variables وبعض الأمور الأخرى المتعلقة بذلك ، وذلك من خلال النقاط التالية:

- المعدلات Modifiers.
- تحديد درجة التوصل للدوال والمتغيرات Access Control for Methods and Variables.
- مفهوم الكبسلة Encapsulation.
- المعدل الخاص private .
- المعدل العام public.
- المعدل المحمي protected.
- مقارنة المعدلات Comparing Modifiers.
- الوراثة ودرجات التوصل Access Control and Inheritance.
- دوال التوصل Accessor Methods.
- المعدل النهائي final .
- الفصائل والدوال المجردة Abstract Classes and Methods.

المعدلات Modifiers:

- هى مجموعة من الكلمات المحجوزة Reserved keywords التى تستخدم فى تحديد درجة التوصل Access إلى الفصيلة class ودوالها methods ومتغيراتها ، وهذه الكلمات هى

```
static, public, protected, private, final, abstract, synchronized,
volatile, …
```

- تم وضع هذه المفاهيم ليستطيع المبرمج تحديد مواصفات أجزاء الفصائل classes والدوال methods والمتغيرات ، ومدي استطاعة مستخدم الفصيلة class التوصل Access لها والتعامل معها.
- ونستطيع تقسيم المعدلات Modifiers حسب الفئات التى تعمل معها وهى كالآتى:

1- معدلات Modifiers للسيطرة على الوصول Access Controlling للدوال methods والمتغيرات والفصائل classes وهى public, protected, private.

2- معدلات Modifiers لإنشاء المتغيرات والدوال methods وهى static.

3- معدلات Modifiers لإنهاء إنشاء الفصائل classes والدوال methods والمتغيرات وهى final.

4- معدلات Modifiers لإنشاء فصائل classes ودوال methods مجردة وهى abstract.

5- معدلات Modifiers تستخدم مع عمليات المهام المتعددة threads وهى synchronized, volatile.

- ولكى تستخدم المعدلات Modifiers فإننا نقوم بوضع المعدل Modifier ثم اسم المتغير أو الدالة method أو الفصيلة class كما فى الأمثلة التالية:

```
public class Emp {// Body of The class}
private boolean sex;
static final double salary = 9.5;
protected static final int Delta;
public static void min (string arg [ ])
```

إذا كنت تستخدم أكثر من معدل Modifier ، فلا يهم ترتيبهم طالما أنهم يسبقون اسم الفصيلة class أو الدالة method أو المتغير.

- على الرغم من أن استخدام المعدلات Modifiers اختياري - فكما لاحظت أننا لم نستخدمها إلا نادراً - لكن من المفيد استخدامها لأننا نستطيع عن طريقها تحديد طرق وصول مستخدمى الفصائل classes إلى الفصائل classes والمتغيرات والدوال methods.

تحديد درجة التوصل للدوال والمتغيرات

Access Control for Methods and Variables:

- إن أغلب استخداماتك من المعدلات Modifiers هى تلك التى تحدد درجة الوصول Access Control للدوال methods والمتغيرات وهى الكلمات protected, public, private ، وهذه المعدلات Modifiers تحدد أي من المتغيرات والدوال methods متاحة ومرئية للفصائل classes الأخرى.
- وباستخدام طرق التوصل Access Control فإنك تحدد كيفية استخدام فصيلتك class عن طريق الفصائل classes الأخرى ، فمثلاً بعض المتغيرات والدوال methods ستستخدم فقط من داخل الفصيلة class نفسها ويجب إخفاؤها عن باقى الفصائل classes التى ربما تتعامل مع فصيلتك class ، ويطلق على هذه العملية الكبسلة Encapsulation.
- والأصل فى تصميم الفصائل classes أن تكون متغيرات variables الفصيلة class محمية من الدرجة private أو من الدرجة protected (حتى لا يتم التعامل مباشرة مع هذه المتغيرات) ، وتكون دوال Methods الفصيلة class من النوع public حتى يمكن استدعاؤها.

مفهوم الكبسلة Encapsulation:

- هى فكرة من أفكار البرمجة الحديثة Object Oriented Programming تقوم على إنشاء الفصيلة Class كوحدة مغلقة تحتوى على المتغيرات Variables

والدوال Methods ، ويتم التعامل معهما دون الدخول فى تفاصيل تركيب الفصيلة class ولا أكوادها ولا كيفية قيام دوال methods الفصيلة class بتحقيق وظائفها ، وهذا هو ما يسمى بالكبسلة Encapsulation.

- وتقوم لغة Java بتوفير أربعة درجات من تحديد درجة التوصل Access Control وهم:
  1. المعدل العام public.
  2. المعدل الخاص private.
  3. المعدل المحمي protected.
  4. المعدل الافتراضى Default Modifier (والذي يتم تحديده بعدم كتابة أي معدل modifier).
- ولنأخذ مثالاً على المعدل الافتراضى Default level Modifier ، حيث أن هذا المستوى معناه أنك لا تستخدم أي معدل Modifier مع المتغير أو الدالة method مثلما استخدمناه فى معظم الأمثلة السابقة مثل:

```
String name = "Ahmed";
int clacAge (){
 // Body of The Method
}
```

- فكما ترى أننا لم نستعمل أي معدل Modifier ، وهذا المستوى يسمى الافتراضي Default ، والمتغير أو الدالة method الذى يعلن بدون أي معدل Modifier يصبح متاحاً لأى فصيلة class فى نفس الحزمة Package.
- وكما قلنا من قبل أن لغة Java عبارة عن مكتبة من الفصائل classes مرتبة فى حزم packages مثل java.awt، java.net ، وهذه الفصائل classes تخص موضوع محدد مثل الشبكات أو واجهة التطبيق.
- إذن فأى متغير عضو member variable فى الفصيلة class يعلن بدون أى معدل modifier ، تستطيع أى فصيلة class فى نفس الحزمة package أن تراه وتعدل فى قيمته ، وأى دالة method تعلن بدون أى معدل Modifier ، تستطيع أى فصيلة class فى نفس الحزمة Package أن تستدعيها.

- ولا تستطيع أى فصيلة class خارج الحزمة package التوصل إلى المتغيرات أو الدوال methods المعلنة بدون معدل Modifier.

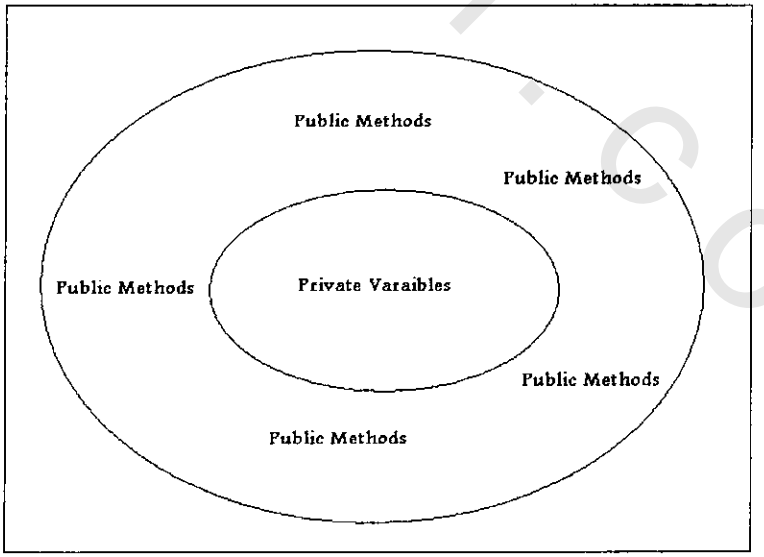
المعدل الخاص private :

- عند استخدام private مع أى متغير عضو member variable أو دالة عضو member method ، فإنك تخفيه بالكامل ، فلا يمكن أن يستخدم من أى فصيلة class أخرى.
- وهو يستخدم عند الرغبة فى منع أي من الفصائل classes الأخرى من التعامل المباشر مع هذه المتغيرات أو الدوال methods.
- والفصيلة class الوحيدة التى تستطيع رؤيته أو التعامل معه هى الفصيلة class المعلن فيها هذا المتغير أو الدالة method.
- وهذا ينطبق أيضاً على الفصائل classes التي ترث من هذه الفصيلة class ، فهذه الفصائل classes لا تستطيع التوصل لهذا المتغير أو الدالة method المعلن عنها بـ private.
- وعموماً المتغيرات المعلن عنها private مفيدة جداً فى حالتين:

1- عندما تكون الفصائل classes الأخرى لا تحتاج إلى استخدام هذا المتغير.

2- عندما نخاف أن تقوم فصيلة class أخرى بتغيير قيم هذا المتغير بقيم غير مرغوب فيها.

- الشكل التالي يوضح الصورة العامة للفصيلة class.



(الشكل 8-1)

وكمثال لو أن عندنا متغير n يحمل عدد الطلاب الناجحين فى فصيلة class تسمى SuccessStudents وقامت فصيلة class أخرى تسمى SubStudent بتغيير قيمة المتغير n ، فعندئذ سوف يختل عمل الفصيلة class الأولى ، ففى هذه الحالة يجب إعلان هذا المتغير private بحيث تستطيع الدوال methods والمتغيرات داخل الفصيلة SuccessStudents فقط التعامل معه ، ولتوضيح ذلك تابع معنا نص البرنامج التالى.

مثال (1): المعدل الخاص private:

قم بانشاء برنامج جديد ثم قم بكتابة هذا البرنامج.

```
1- class Student
2- {
3-    int x;
4- private int success;
5- private float salary;
6- void calcSuccess(int s)
7- {
8-      success = s;
9- }
10-    public void print ( )
11-   {
12-    System.out.println("The Number of Succeeded Students
       ="+ success);
13-   }
14- }
15- class mainx
16- {
17-    public static void main (String arg [])
18-    {
19-    Student s = new Student();
20-    //s.success = 100; //This statement will cause an Error
21-    s.print();
22-    s.calcSuccess (10); // This is right
23-    s.x=10;
24-    }
25- }
```

- فى هذا المثال نقوم بإنشاء فصيلة class مهمتها عرض عدد الطلبة الناجحين.
- فى هذا المثال تم الإعلان عن المتغير success عن أنه private كما فى السطر رقم 4 ، إذن فهو مرئى فقط داخل الفصيلة Student ، ولذلك استطاعت الدالة calcSuccess() التعامل معه كما فى السطر رقم 8.
- لكن فى السطر رقم 20 سيقوم المترجم compiler بإصدار رسالة خطأ لأن الهدف s ليس له الحق الآن فى التوصل للمتغير success ، بينما له الحق فى التوصل للمتغير x وأيضاً الدالة calcSuccess() لأنهما بدون معدلات Modifiers أي أن لهم المستوى الافتراضى Default level Modifier.

قم بإزالة السطر رقم 20 وستجد أن البرنامج يتم ترجمته Compiling بشكل صحيح

المعدل العام public:

- فى بعض الأحيان نحتاج أن نجعل متغيراً أو دالة method متاحة لجميع الفصائل classes التى تريد استخدام هذا المتغير أو الدالة method ، فكيف يتم ذلك؟
- سيقفز فى ذهنك على الفور أن نجعل المتغير أو الدالة method بدون معدل Modifier ولكن تذكر أنك إذا فعلت ذلك ، فسيكون المتغير أو الدالة method متاحة لأى فصيلة class موجودة فى الحزمة Package الواقع فيها فصيلة class المتغير أو الدالة method فقط ، إذن ماذا عن بقية الفصائل classes الموجودة فى بقية الحزم Packages؟
- الحل هو أن نستخدم المعدل public ، فعند الإعلان عن متغير أو دالة public ، يصبح هذا المتغير أو هذه الدالة method عامة لجميع الفصائل classes فى أى حزمة package ويكون لها الحق فى التوصل لهذا المتغير أو الدالة method.
- وللإعلان عن متغير عضو member variable أو دالة عضو member method باستخدام المعدل public ، فإن ذلك يأخذ الشكل التالى:

```
public int x;
public void printData();
```

- ومن أشهر الدوال methods التى استخدمناها وتستخدم المعدل public هى الدالة الرئيسية ()main كما فى السطر:

```
public static void main (String arg[])
```

المعدل المحمي protected:

- يقوم المعدل modifier من نوع protected بالعمل على الحد من التوصل للمتغيرات والدوال methods إلا عن طريق المجموعتين الآتيتين:
  1-الفصيلة الفرعية sub class من هذه الفصيلة class.
  2-كل الفصائل classes الموجودة فى نفس الحزمة package.
- أي باختصار أن المعدل protected مثل المعدل الافتراضي default modifier باستثناء أن الفصائل classes المورثة - سواء الموجودة فى نفس الحزمة package أو فى حزمة package أخري- تستطيع أن تتوصل إلى المتغيرات أو الدوال methods المسبوقة بـ protected.
- ويتم الإعلان عنها كما فىالشكل التالى:

```
protected int x;
protected int calacStudentNo () {}
```

- ويعتبر هذا النوع مفيد جداً فى الوراثة inheritance.

مقارنة المعدلات Comparing Modifiers:

- بعد أن تناولنا الدرجات المختلفة لدرجة الوصول إلى البيانات والدوال methods وهى الدرجات protected, public, private, default ، قد يصبح من الصعب التفرقة بينهم خصوصاً مع protected ، لذلك قمنا بجمعهم فى جدول واحد والمقارنة بينهم.
- وقبل عرض الجدول تابع معنا المثال التالى:

```
protected  int x;
```

- هذا الإعلان يعنى أنه يمكننا التوصل (التعامل) مع هذا المتغير من خلال الفصيلة class التى أعلن فيها المتغير ، كما يمكن التوصل لهذا المتغير من أي فصيلة class فى نفس الحزمة Package ، ولكن لا يمكن التوصل لهذا المتغير من أي فصيلة

class لا ترث من هذه الفصيلة class وتقع خارج الحزمة Package ، كما يمكن التوصل لهذا المتغير من خلال فصيلة class ترث من الفصيلة class المعلن فيها المتغير x فى نفس الحزمة package أو في حزمة package أخرى.

الجدول التالى يوضح هذه الدرجات والمقارنة بينها.

مكان التوصل Visibility	درجات معدلات التوصل			
	public	protected	default	private
من نفس الفصيلةclass	Yes	Yes	Yes	Yes
من أي فصيلة class واقعة فى نفس الحزمة package	Yes	Yes	Yes	No
من أي فصيلة class تقع خارج الحزمة package	Yes	No	No	No
من فصيلة فرعية sub class واقعة فى نفس الحزمة package	Yes	Yes	Yes	No
من فصيلة فرعية sub class واقعة خارج الحزمة package	Yes	Yes	No	No

الوراثة ودرجات التوصل Access Control and Inheritance:

بعد أن تناولنا درجات التوصل Access control ، نود أن نلقى بعض الضوء على موضوع بسيط يربط بين الوراثة Inheritance ودرجات التوصل access control ، فتعال معى نستوضح هذه النقطة الصغيرة.

عندما تنشئ فصيلة فرعية sub class وتقوم بعملية نسخ override لدالة method موجودة فى الفصيلة العليا super class ، لابد أن تأخذ فى الاعتبار موضوع درجة التوصل Access control فى الدالة method الأصلية ، ويتضح ذلك من القواعد التالية:

1. الدوال methods المعلن عنها public فى الفصيلة العليا super class لابد أن تكون public فى كل الفصائل الفرعية sub classes.

2. الدوال methods المعلن عنها protected فى الفصيلة العليا super class لابد وأن تكون أيضاً إما protected أو public فى الفصائل الفرعية sub classes ، ولا يمكن بأى حال من الأحوال أن تكون private.
3. الدوال methods المعلن عنها بدون معدل Access Modifier فى الفصيلة العليا super class يمكن أن تكون أيضاً بدون معدل Access Modifier أو public أو protected فى الفصائل الفرعية subclasses.

دوال التوصل Accessor Methods:

- لكى نفهم ماذا نعنى بدوال التوصل Accessor Methods ، تعال معى نتخيل السيناريو التالى ، بفرض أنك قد أعلنت عن متغير عضو member variable فى فصيلة class ما تسمى مثلاً classx1 ، وكان هذا المتغير يمثل بيانات لا تريد أن تتعامل معه الفصائل classes الأخرى بحيث لا يمكنها أن تغير فى قيمته حتى لا يؤثر ذلك فى عمل الفصيلة class x1 ، فماذا تفعل؟
- الحل هو - كما تتوقع - بأن نقوم بالإعلان عنه باستخدام private ، فبذلك لن تستطيع الفصائل classes الأخرى التوصل له ، وهذا صحيح.
- ولكن ماذا لو أردنا أن نجعل فصائل classes تتوصل access لهذا المتغير بحيث يمكنها تغيير قيمته ، ففى هذه الحالة فإن الإعلان عن المتغير بالدرجة private سيصبح مشكلة.
- والحل هو فى دوال التوصل Methods Accessor ، ونعنى أن يكون المتغير private كما هو ولكن ننشئ دوال للتوصل Accessor Methods لهذا المتغير ، وهذه الدوال methods تكون public كما يوضحه نص البرنامج التالى.
- فى هذا البرنامج تم الإعلان عن متغير اسمه zipCode ومستوى الوصول له private وتم إنشاء دالة التوصل له Accessor Method.

```
1. class x1
2. {
3.         private int zipCode = 0;
4.  public int getZipCode()
5.  {
```

```
6.          return zipCode;
7.  }
8.  public void setZipCode (int zipCode)
9.  {
10.     this. zipCode = zipCode;
11. }
12.     // Body of the class
13. }
14. class  y1 extends x1
15. {
16.     int f;
17.     public static void main (string arg [])
18. {
19.     f=100;
20.  y1  n= new  y1();
21.     n. setZipCode(f);
22.     // Body of the class
23. }
```

- وتنقسم دوال التوصل Accessor Method إلى نوعين ، واحدة للقراءة Read ويبدأ اسمها بالكلمة get ثم اسم المتغير مثل getZipCode كما هو واضح فى السطر رقم 4 ، فهذه الدالة method مهمتها إرجاع قيمة المتغير zipCode.
- والنوع الثانى للكتابة ويبدأ اسمها بالكلمة set ثم اسم المتغير مثل setZipCode كما هو واضح فى السطر رقم 8 ، وهذه الدالة method مهمتها وضع قيم المتغير zipCode.
- وعن طريق هاتين الدالتين نستطيع التوصل إلى المتغير من نوع private من خلال فصيلة class أخرى كما هو موضح فى السطر رقم 21 ، فقد استعملنا دوال التوصل Accessor Methods لكى نتوصل إلى المتغير من نوع private.
- وكما أشرنا من قبل ، أن هذا الشكل هو الشكل التقليدى لبناء الفصيلة class.

المعدل النهائي final :

- توجد خاصية يمكن تحديدها على الفصائل classes والدوال Methods والمتغيرات variables وهى استعمال كلمة final وذلك لتوضح بعض المعانى كما يلى:

1. class final تعنى أننا لن نستطيع أن ننشئ منها فصيلة فرعية sub class أى لا نستطيع القيام بالوراثة Inheritance منها ، وتصبح هى أخر فصيلة class فى شجرة هذه الفصائل Class Tree.
2. final method تعنى أننا لن نستطيع أن نقوم بعملية نسخ override لهذه الدالة method فى فصيلة أخرى فرعية sub class ويجب استعمالها كما هى.
3. final variable معناه أننا لن نستطيع تغيير قيمة هذا المتغير ويجب استعماله كما هو ويكون هذا المتغير في هذه الحالة من الثوابت Constants.

- والسؤال الآن ماذا نستفيد لو قمنا بالإعلان عن دالة method أنها final كما فى المثال التالى:

```
public final void getData()
{
   // Body of the Method
}
```

- والإجابة أنها تجعل عملية التنفيذ أسرع لأنها لا تجبر المترجم compiler عن البحث عن عمليات نسخ override قد تكون تمت لهذه الدالة method داخل فصيلة class أخري.
- وأيضاً تنطبق الإجابة السابقة على الفصائل classes لو أعلن عنها final كالآتى:

```
final public class x {    }
```

- فالإعلان عن الفصيلة class بهذا الشكل يجعل تنفيذ الفصيلة class أكثر سرعة.

لو أعلنت عن فصيلة class أنها final ، فتكون بالتالى جميع الدوال methods بداخلها final ولا نحتاج أن نضع المعدل final معهم.

الفصائل والدوال المجردة Abstract Classes and Methods:

- إن مبدأ التجريد Abstraction هو الوصول لأبسط صورة من صور شئ ما ، فمثلاً عندما تقول أن الإنسان يتبع فصيلة الثدييات ، والثدييات تتبع الكائنات الحية ، إذن فالكائنات الحية هى التجريد للإنسان ، فلا يوجد أبسط من هذه الصورة لوصف الإنسان.

- كذلك الحال فى البرمجة ، فعندما يكون عندنا مجموعة من الفصائل classes مرتبطة بعلاقة الوراثة Inheritance ، فيكون ترتيبها هرمياً ونلاحظ أن أعلى فصيلة class فى هذا الترتيب تسمي الفصيلة العليا Super class وتكون هى الأقرب لمفهوم التجريد Abstraction ، بمعنى أن هذه الفصيلة class مفيدة فقط لننشئ منها فصائل فرعية sub classes ، أما هى فى حد ذاتها فإنها لا تستخدم.
- ويكون شكل الإعلان كالتالى:

```
public abstract class myday
{
// Body of the class
 }
```

- ولنضرب مثالاً عن الحاجة إلي إعلان فصيلة class ما أن تكون مجردة abstract.
- والمثال التالى يوضح بناء فصائل classes لتمثيل بيانات أفراد فى مؤسسة ما.

```
1- //create class Employee
2- //This is the super class for Engineer and Accountant
3- public abstract class Employee {
4- private int code;
5- private String name;
6- private int age;
7- private double salary;
8- public int getCode() {
9- return code;
10- }
 11- public void setCode(int code) {
12- this.code = code;
13- }
14- public String getName() {
15- return name;
16-  }
17- public void setName(String name) {
18- this.name = name;
19- }
20- public int getAge() {
```

```
21- return age;
22- }
23- public void setAge(int age) {
24- this.age = age;
25- }
26- public double getSalary() {
27- return salary;
28- }
29- public void setSalary(double salary) {
30- this.salary = salary;
31- }
32- }
33- class Engineer extends Employee{
34- private String field;
35- private String location;
36- public String getField() {
37- return field;
38- }
39- public void setField(String field) {
40- this.field = field;
41- }
42- public String getLocation() {
43- return location;
44- }
45- public void setLocation(String location) {
46- this.location = location;
47- }
48- }
49- //statring Accountant Class
50- class Accountant extends Employee{
51- private String AccountantField;
52- private String typeOfMachineHeuse;
53- public String getAccountantField() {
54- return AccountantField;
55- }
56- public void setAccountantField(String accountantField) {
```

```
57- AccountantField = accountantField;
58- }
59- public String getTypeOfMachineHeuse() {
60- return typeOfMachineHeuse;
61- }
62- public void setTypeOfMachineHeuse(String
63- typeOfMachineHeuse) {
64- this.typeOfMachineHeuse = typeOfMachineHeuse;
65- }
66- }
67- //Main class
68- class UseClasses
69- {
70- public static void main(String args[])
71- {
72- Engineer   e=new Engineer();
73- Accountant a=new Accountant();
74- e.setAge(40);
75- e.setCode(101);
76- e.setName("Ahmed");
77- e.setSalary(3500);
78- e.setField("civil");
79- e.setLocation("Cairo");
80- System.out.println("Engineer Code= "+e.getCode());
81- System.out.println("Engineer Name= "+e.getName());
82- System.out.println("Engineer Salary= "+e.getSalary());
83- System.out.println("Engineer Age= "+e.getAge());
84- System.out.println("Engineer Field= "+e.getField());
85- a.setAge(50);
86- a.setCode(151);
87- a.setName("Mohamed A.Elfattah");
88- a.setSalary(1500);
89- a.setTypeOfMachineHeuse("Calculators & Computer");
90- a.setAccountantField("Payable");
91- System.out.println("Accountant Cod= "+a.getCode());
92- System.out.println("Accountant Name= "+a.getName());
```

```
93- System.out.println("Accountant Salary= "+a.getSalary());
94- System.out.println("Accountant Age= "+a.getAge());
95- System.out.println("Accountant Machine Which he use= -
"+a.getTypeOfMachineHeuse());
96-System.out.println("Accountant Field=
"+a.getAccountantField());
97- }
98- }
```

شرح السطور:

- قمنا بإنشاء ثلاث فصائل مساعدة Helper Classes (أي لا تحتوى على الدالة الرئيسية main() وهم Employee, Engineer, Accountant).
- قمنا بإنشاء فصيلة رئيسية هى UseClasses وذلك لأنها تحتوى الدالة الرئيسية main().
- فى السطر رقم 3 قمنا بالإعلان عن الفصيلة Employee أنها مجردة abstract ، فلماذا؟
- هذا لأن كل فرد فى تخصص ما - مثل الهندسة أو المحاسبة - يشتركون فى العديد من الصفات مثل السن والاسم والرقم الكودى والمرتب ، كما أنهم على اختلاف تخصصاتهم يندرجون تحت فئة الموظفين ، لذلك أعلنت أن الفصيلة Employee هى فصيلة مجردة abstract أي أنها موجودة لكي أقوم بالوراثة inheritance منها فقط ، إذ لا يمكن أن تصف فرد ما بأنه موظف فقط ، بل تقول أنه محاسب أو مدير أو مهندس وفى نفس الوقت هو أيضاً موظف.
- فى السطور من 4 إلي 7 قمنا بتعريف مجموعة من المتغيرات الأعضاء member variables وهى الخواص المشتركة بين جميع الموظفين مثل رقم الكود والاسم والسن والمرتب ، فجميع الموظفين مهما اختلفوا فى التخصصات لابد أن يشتركوا فى هذه الخواص.
- فى السطور من 8 إلي 31 قمنا بإنشاء دوال التوصل Accessor Methods للمتغيرات المعلن عنها فى السطور من 4 إلي 7 حيث أننا استخدمنا المعدل الخاص private معهم.
- بعد ذلك قمنا ببناء فصيلتين classes ، واحدة للتعامل مع بيانات الموظفين من فئة المهندسين Engineer ، والثانية لبيانات الموظفين من فئة المحاسبين Accountant.
- وستلاحظ أننا أعلنا عن المتغيرات الأعضاء member variables داخل الفصائل classes أنها private وذلك لأن هذه الطريقة آمنة للتعامل مع المتغيرات

وستلاحظ أننا استعملنا دوال التوصل Accessor Methods في فصيلتي Engineer, Accountant كما في السطور من سطر 36 إلى سطر 48 في الفصيلة Engineer والسطور من سطر 53 إلى 75 في الفصيلة Accountant ، وستلاحظ أيضاً أننا قمنا بتعريف ما يميز كل من المهندسين عن المحاسبين.

- ففي الفصيلة Engineer مثلاً نلاحظ المتغير العضو instance variable المسمي "location" الذي يعبر عن الموقع الذى يعمل فيه المهندس.
- بينما في الفصيلة Accountant نجد المتغير العضو AccountantField الذى يحدد تخصص المحاسب في عملية المحاسبة.
- في السطور من 68 إلي 98 يتم بناء الفصيلة class الرئيسية.
- في السطرين 71 و 72 نقوم بإنشاء هدفين e, a من الفصيلتين Engineer, Accountant لكى نستطيع التعامل مع دوال methods الفصيلتين classes.
- في السطور من 74 إلي 77 نقوم باستدعاء الدوال ()setAge و ()setCode و ()setName و ()setSalary من خلال الهدف e على الرغم من أن هذه الدوال methods غير موجودة في الفصيلة Engineer ، ولكن لأننا أعلنا أن الفصيلة Engineer ترث من الفصيلة Employee ، فلذلك يصبح من حق الفصيلة Engineer أن تتوصل للدوال methods في الفصيلة Employee ، وهذا ما يقوم به الهدف e في السطور من 74 إلي 77 ، وكذلك الحال مع الهدف e في السطور من 85 إلي 90.
- في السطور من 80 إلي 84 يتم عرض البيانات الخاصة بالهدف e.
- في السطور من 91 إلي 97 يتم عرض البيانات الخاصة بالهدف a.

ملخص الفصل:

- تعرضنا في هذا الفصل لشرح قواعد التوصل للفصائل classes والمتغيرات Access Modifiers.
- في الفصل القادم سوف نتعرف - بإذن الله - علي أحد أهم مبادئ البرمجة بواسطة الأهداف OOP وهو مبدأ تعدد الأشكال Polymorphism ، فتابع معنا الفصل القادم.