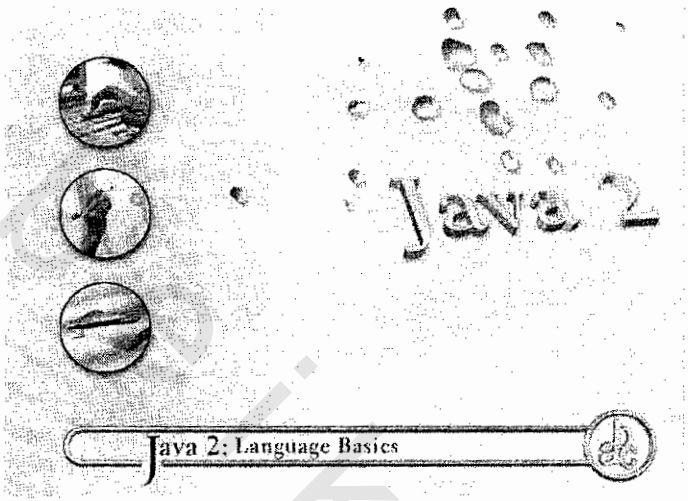
الفصل الحادى عشر



Interfaces الـ

Interfaces

في هـذا الفصـل نتعـرف علـي الـ interface
واستخداماته وذلك من خلال النقاط التالية:

- ما هو الـ Interface.
- تعريف الـ Interface.
- ما يحتويه الـ Interface.
- فائدة الـ Interface.
- كيفية استخدام الـ Interface.

ما هو الـ Interface:

- عرفنا فيما سبق أن لغة Java لا تدعم الوراثة المتعددة Multiple Inheritance ، ولكن فى الحياة العملية فإنه من الممكن أن تحتاج أن تنشىء فصيلة class ترث خواصها من أكثر من فصيلة class أخري ، فمثلاً في حياتنا العادية فإن الابن من الممكن أن يرث من أبيه وأمه ، فإذا استخدمنا الوراثة Inheritance فإننا في هذه الحالة لا نستطيع الوراثة Inheritance من أكثر من فصيلة class واحدة.
- إذن ما الحل إذا أردنا اكتساب خواص من أكثر من فصيلة class؟ الحل هو استخدام الـ Interface.

تعريف الـ Interface:

- هو صورة من صور الفصائل classes ولكن بدون كتابة جسم دوالها Method Body ، وبالتالى يمكنك القول بأن الـ Interface هو فصيلة class لا يتم فيها تعريف سطور دوالها Methods ، أى أنها مجرد إعلان عن الدوال Method Declaration.
- ويتم تعريف الـ Interface مثل الفصيلة class تماماً ولكن مع استبدال كلمة class بكلمة interface مثل interface Employee.

ما يحتويه الـ interface:

- من الممكن أن يحتوى الـ interface على متغيرات ودوال Methods مثله مثل الفصيلة class ولكن كل المتغيرات فيه تكون نهائية final ، بمعنى أنه لا يمكنك تغيير قيمتها الأبتدائية Initial Value وأيضاً كل الدوال تكون مجردة abstract ، بمعنى أنه فى الفصيلة class التى تستخدم هذا الـ interface فإنك لابد أن تعيد تعريف جميع الدوال Methods فيه وإلا اعتبرت الفصيلة class الجديدة فى هذه الحالة مجردة abstract وبالتالى لا يمكنك إنشاء هدف object منها.
- فى الـ interface كل المتغيرات تكون نهائية final وكل الدوال مجردة abstract حتى لو لم تذكر ذلك صراحة.
- لا يمكنك كتابة تعريف أى دالة Method فى الـ interface ولكنك تكتب نوع القيمة المرتجعة Return Value من الدالة Method ثم اسمها ثم المعلامات

Parameters ، أى كما أشرنا أننا نضع فقط الإعلان عن الدالة Method ولا نكتب سطور هذه الدالة Method.

فائدة الـ Interface :

- فى حالة تطوير مجموعة من الفصائل classes ومطلوب التزامها بتركيب معين ، يتم إنشاء الـInterface بالتركيب المطلوب ثم جعل هذه الفصائل classes تستورث (تطبق) implement هذا الـ Interface ، وبالتالى تم إجبار مبرمجي الفصائل classes علي الالتزام بهذا التركيب.

ملحوظة:

يسمح الـ Interface بتعدد التوريث Multiple Inheritance وبالتالى يمكن لفصيلة class جديدة أن تستورث (تطبق) implement أكثر منInterface .

كيفية استخدام الـ interface:

- كما تعلم أنه يتم تحقيق الوراثة Inheritance من الفصيلة الأم Parent Class باستعمال كلمة extends ، ولكن فى الـ interface فإنك تستخدم كلمة implements.

مثال (1): استخدام الـ interface:

يوضح هذا المثال كيفية تعريف واستعمال الـ Interface.

كود البرمجة:

```
1:    interface Employee
2:    {
3:      float calculateTax ();
4:      int x = 10;
5:    }
6:
7:    class Salary
8:    {
9:     float salary;
10:    float Tax;
```

```
11:   }
12:
13:   class Engineer extends Salary implements Employee
14:   {
15:    Engineer ( float salary , float Tax )
16:    {
17:         this.salary = salary;
18:         this.Tax = Tax;
19:    }
20:
21:    public float calculateTax ()
22:    {
23:         //x = 20;//this line causes an error
24:         return salary * Tax;
25:    }
26:    public static void main ( String[] args )
27:    {
28:         Engineer eng = new Engineer ( 1000 , .05f );
29:         float f = eng.calculateTax();
30:         System.out.println("Tax = " + f );
31:    }
32:   }
```

شرح السطور:

- في السطر رقم 1 يتم تعريف interface باسم Employee.
- في السطر رقم 3 يتم تعريف دالة Method اسمها ()calculateTax. لاحظ عدم وجود كود الدالة Method نفسها. أيضاً هذه الدالة Method تكون مجردة abstract على الرغم من أننا لم نذكر ذلك صراحة.
- في السطر رقم 4 يتم تعريف متغير اسمه x ووضعنا له قيمة ابتدائية Initial Value تساوى عشرة. لاحظ أن هذا المتغير يكون نهائياً final على الرغم من أننا لم نذكر صراحة ، بمعنى أنه لا يمكنك تغيير قيمته أبداً بعد ذلك.
- في السطور من 7 إلى 11 يتم تعريف فصيلة class اسمها Salary تحتوى على متغيرين.

- في السطر رقم 13 تتضح أهمية وفائدة الـ interface حيث أنك ترث من الفصيلة Salary وتنفذ الـ interface المسمي Employee ، وبذلك فإن الفصيلة Engineer تكون لها صفات الاثنين.
- في السطر رقم 15 يتم تعريف دالة البناء constructor التى تستقبل قيمتين وتستخدمهم لإعطاء قيمة ابتدائية Initial Value للمتغيرين salary وTax. لاحظ أن المتغيرين salary وTax تم تعريفهم أصلاً فى الفصيلة Salary ، وحيث أنك ورثت من هذه الفصيلة class فإنك ورثت هاتين القيمتين.
- في السطر رقم 21 يتم إعادة تعريف الدالة calculateTax(). لاحظ أن هذه الخطوة ضرورية كما ذكرنا وإلا اعتبرت الفصيلة Engineer فى هذه الحالة مجردة abstract ولا يمكنك إنشاء هدف object منها.
- في السطر 23 يتم توضيح خطأ واضح ولذلك وضعناه كتعليق. هذا الخطأ نتج من أن المتغير x هو متغير نهائي final لأنه موجود فى الـ interface المسمى Employee ولذلك لا يمكنك تغيير قيمته أبداً.
- باقى السطور واضحة ولا تحتاج إلى شرح.

لاحظ أنه لا يمكنك أن ترث من أكثر من فصيلة class واحدة وهذا على عكس الـ interface الذى يمكنك تنفيذ أكثر من interface بشرط الفصل بين اسمهم بفصلة comma ، فمثلاً إذا افترضنا وجود interface يسمى Person وأخر يسمى Adult ونريد أن ننشىء فصيلة class اسمها Engineer تنفذهما ، فإننا تكتب

```
class Engineer implements Adult, Person
{
        // rest of class Engineer code
}
```

مثال (2): استخدام الـ interface:

هذا المثال لا يوجد به جديد ولكننا نستخدمه لبيان مشكلة سوف نوضحها وذلك كما فى السطور التالية:

```
1: class Employee
2: {
3:       Engineer eng;
4:       float TaxPercentage;
5:       float Salary;
6:
 7:      Employee ( Engineer eng , float Salary , float TaxPercentage )
8:      {
9:           this.eng = eng;
10:               this.TaxPercentage = TaxPercentage;
11:               this.Salary = Salary;
12:          }
13:
14:          public void calculateTax ()
15:          {
16:          System.out.println ( "Tax = " + TaxPercentage
* Salary /100 );
17:     }
18:   }
19:
20:   class Engineer
21:   {
22:    Engineer ( )
23:    {
24:          Employee eng = new Employee ( this , 1000 , 5 );
25:          eng.calculateTax ();
26:    }
27:
28:    public static void main ( String[] args )
29:    {
30:          new Engineer ();
31:    }
32:       32: }
```

شرح السطور:

في السطر رقم 1 يتم تعريف فصيلة class اسمها Employee.

- في السطور من 3 إلى 5 يتم تعريف متغيرات الفصيلة class.
- في السطر رقم 7 يتم تعريف دالة البناء constructor حيث تستقبل ثلاث قيم ، القيمة الأولى تمثل هدفاً object من نوع الفصيلة Engineer الذى سيلى شرحه فيما بعد. القيمتان الثانية والثالثة هما قيم المرتب ونسبة الضريبة على الترتيب.
- في السطور من 9 إلى 11 يتم تحديد قيم ابتدائية Initial Values لمتغيرات الفصيلة class عن طريق القيم التى تم تمريرها لدالة البناء constructor.
- في السطور من 14 إلى 17 يتم تعريف دالة اسمها calculateTax() تقوم بطباعة قيمة الضريبة.
- في السطر رقم 20 يتم تعريف فصيلة class اسمها Engineer.
- في السطر رقم 22 يتم تعريف دالة البناء constructor.
- في السطر رقم 24 يتم إنشاء هدف object من نوع الفصيلة Employee مع تمرير ثلاث قيم لدالة البناء constructor للفصيلة Employee ، القيمة الأولى: this التي - كما نعلم - تشير إلى الهدف object الحالى ، وحيث أننا فى الفصيلة Engineer ، إذن فـ this تشير إلى هدف object من نوع الفصيلة Engineer. القيمتان الثانية والثالثة هما قيم المرتب ونسبة الضريبة على الترتيب.
- في السطر رقم 25 يتم استدعاء الدالة calculateTax() الموجودة فى الفصيلة Employee.
- في السطر رقم 28 يتم تعريف الدالة الرئيسية main() وفيها ننشىء هدفاً object من نوع الفصيلة Engineer.
- إلى هنا ولا شئ جديد ولكن المشكلة تظهر كالآتي:
- نريد أن ننشئ فصيلة class اسمها Salesman تقوم بحساب الضريبة ، فإذا أنشأنا الفصيلة class كالآتى:

```
class Salesman
{
   Salesman ()
   {
```

```
        Employee Sales = new Employee (this, 1000,5);
        Sales.calculateTax ();
    }
}
```

- المشكلة التى تظهر الآن أننا نريد أن نستدعى الدالة calculateTax() الموجودة فى الفصيلة Employee ، والمشكلة تكمن تحديداً فى أن this تشير إلى الهدف object الحالى ، وفي هذه الحالة فإن this تشير إلى هدف object من نوع Salesman ، ولكن دالة البناء Constructor للفصيلة Employee تستقبل هدفاً object من نوع الفصيلة Engineer.
- أيضاً قد نريد حساب الضريبة للفصيلة Salesman بطريقة مختلفة.
- إذن ما الحل؟
- أحد الحلول أننا فى الفصيلة ننشىء دالة بناء constructor أخري بحيث نجعلها تستقبل هدفاً object من نوع Salesman.
- حسناً هذا الحل سوف يعمل ، ولكن افترض أن عندك 100 مهنة ، فمن غير المعقول أن تعيد تعريف دالة البناء constructor مئة مرة.
- إذن الحل الأخر هو أنك إذا لم تجد شيئاً مشتركاً بين أكثر من فصيلة class ، فإنك تستطيع إنشاء interface يكون بمثابة الشئ المشترك بين الفصائل classes ، وهذا سيظهر فى المثال التالى.

مثال (3): استخدام الـ interface:

نريد أن ننشئ فصيلة class تسمى Engineer وفصيلة class أخري تسمى Salesman بحيث أن كل فصيلة class منهما تستطيع حساب الدخل الصافى لكل موظف بحيث أن الدخل الصافى = المرتب – التأمينات.

كود البرمجة:

```
1:  interface Person
2:  {
3:    void calculateNetSalary ( float ins );
```

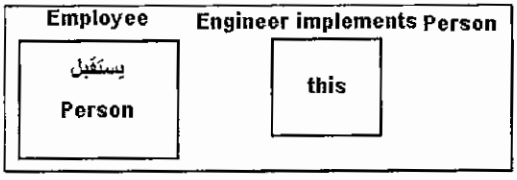
```
4:    }
5:
6:    class Employee
7:    {
8:     Person p;
9:     float TaxPercentage;
10:   float Salary;
11:
12:   Employee ( Person p , float Salary , float TaxPercentage )
13:   {
14:       this.p = p;
15:       this.TaxPercentage = TaxPercentage;
16:       this.Salary = Salary;
17:   }
18:
19:   public void calculateInsurance ( float salary )
20:   {
21:       float ins = 0.03f * salary;
22:       p.calculateNetSalary(ins);
23:   }
24:   }
25:
26:   class Engineer implements Person
27:   {
28:    float insurance;
29:
30:    Engineer ( )
31:    {
32:        Employee eng = new Employee ( this , 1000 , 5 );
33:        eng.calculateInsurance(1000);
34:    }
35:
36:    public static void main ( String[] args )
37:    {
38:        new Engineer ();
39:        new Salesman ();
```

```
40:     }
41:
42:     public void calculateNetSalary ( float ins )
43:     {
44:         float net = 1000 - ins;
45:         System.out.println("Net salary = " + net );
46:     }
47:   }
48:
49:   class Salesman implements Person
50:   {
51:    float insurance;
52:
53:    Salesman ( )
54:    {
55:         Employee sales = new Employee ( this , 700 , 3 );
56:         sales.calculateInsurance(700);
57:    }
58:
59:    public void calculateNetSalary ( float ins )
60:    {
61:         float net = 700 - ins;
62:         System.out.println("Net salary = " + net );
63:    }
64:   }
```

شرح السطور:

- في السطر رقم 1 يتم إنشاء interface جديد باسم Person يحتوى على دالة Method وحيدة هى calculateNetSalary() ، وبالطبع هذه الدالة Method ستكون مجردة abstract حتى لو لم نذكر ذلك صراحة.
- في السطر رقم 6 يتم إنشاء الفصيلة class المسماة Employee.
- في السطور من 8 إلى 10 يتم تعريف المتغيرات الخاصة بالفصيلة Employee بحيث يكون المتغير p هو هدف object من نوع الـ interface المسمى Person.

- في السطر رقم 12 يتم تعريف دالة البناء constructor للفصيلة class المسماة Employee حيث تستقبل هدفاً object من نوع الـ interface المسمى Person بالإضافة إلي قيمة المرتب ونسبة الضريبة ، وتستخدم القيم الثلاثة لإعطاء قيم ابتدائية Initial Values لمتغيرات الفصيلة class.
- في السطر رقم 19 يتم تعريف الدالة ()calculateInsurance التى تستقبل قيمة المرتب وتحسب التأمينات كنسبة 3% من المرتب.
- في السطر رقم 22 يتم استدعاء الدالة ()calculateNetSalary الموجودة فى الـ interface الذى أنشأنا منه الهدف object المسمى p وتمرر لها قيمة التأمينات. ولكن السؤال الآن: أين تعريف هذه الدالة Method؟ هذا ما سيتضح فى باقى البرنامج.
- في السطر رقم 26 يتم إنشاء فصيلة class جديدة اسمها Engineer وتنفذ الـ interface المسمى Person ، أى أنك تتوقع أنه لابد أن يتم إعادة تعريف الدالة ()calculateNetSalary التي تم تعريفها فى الـ interface المسمى Person.
- في السطر رقم 30 يتم إنشاء دالة البناء constructor الخاصة بالفصيلة class المسماة Engineer.
- في السطر رقم 32 يتم إنشاء هدف object من نوع الفصيلة Employee ويتم تمرير القيم التالية لدالة البناء constructor الخاصة بالفصيلةEmployee : القيمة الأولى هي this. لاحظ أن this تشير إلى الهدف object الحالى ، وهنا نحن فى الفصيلة Engineer وليس Employee. إذن this تشير إلي هدف object من نوع الفصيلة Engineer. إذا نظرت لتعريف دالة البناء constructor الخاصة بالفصيلة Employee ، تجد أن القيمة الأولى هى هدف object من نوع الـ interface المسمى Person. قد تكون الآن فى حيرة. لا تقلق فالأمر بسيط ويظهر كما فى الشكل التالى:



- كما ذكرنا فإن كلمة this تشير إلى الهدف object الحالى ، أى تشير إلى هدف object من نوع الفصيلة Engineer. ولكن حيث أن الفصيلة Engineer هي فصيلة class لها نفس صفات الـ interface المسمى Person ، إذن فإن this تعتبر إشارة إلى هدف object من نوع Person أيضاً ، ولذلك عند تمرير قيمة this لدالة البناء constructor الخاصة بالفصيلة Employee فلا يوجد تناقض ويعمل البرنامج بشكل صحيح.
- في السطر رقم 33 يتم استدعاء الدالة ()calculateInsurance الموجودة فى الفصيلة Employee ونمرر لها القيمة 1000.
- باقي السطور حتي السطر رقم 47 لا جديد فيها.
- في السطور من 49 إلى 64 يتم تعريف الفصيلة class المسماة Salesman ، ومثلها مثل الفصيلة Engineer تماماً ، ولكن انظر إلى السطر رقم 55 ، لاحظ أن this هنا تشير إلى الهدف object الحالى ، إذن this تشير إلى هدف object من نوع الفصيلة Salesman ، وحيث أن الفصيلة Salesman تنفذ الـ interface المسمى Person ، إذن this تشير إلى هدف object من نوع Person.
- باقى السطور لا جديد فيها.
- النتيجة الملحوظة من البرنامج أننا أنشأنا فصيلة class اسمها Employee ودالة البناء constructor الخاصة بها تستقبل هدفاً object من نوع الـ interface المسمى Person.
- إذن بدلاً من إنشاء دالة بناء constructor تستقبل هدفاً object من نوع الفصيلة Engineer ودالة بناء constructor أخري تستقبل هدفاً object من نوع الفصيلة Salesman ، فإننا أنشأنا دالة بناء Constructor تستقبل هدفاً object من نوع الـ interface المسمي Person وجعلنا الفصائل Engineer و Salesman من نوع الـ interface المسمي Person عن طريق كلمة implements وبالتالي تم توفير جهد كتابة دالة بناء Constructor لكل فصيلة class مطلوبة.

مثال (4): استخدام الـ interface:

فى هذا المثال يتم عرض شكل مطول لاستعمال الـ interface.

```
// Interfaces.
1.    import java.util.*;
2.    interface Instrument
3.    {
4.    int i = 5;
5.    void play();
6.    String what();
7.    void adjust();
8.    }
9.    class Wind implements Instrument
10.   {
11.   public void play()
12.   {
13.   System.out.println("Wind.play()");
14.   }
15.   public String what() { return "Wind"; }
16.   public void adjust() {}
17.   }
18. class Percussion implements Instrument
19.   {
20.   public void play() {
21.   System.out.println("Percussion.play()");
22.   }
23.   public String what() { return "Percussion"; }
24.   public void adjust() {}
25.   }
26. class Stringed implements Instrument
27.   {
28.   public void play() {
29.   System.out.println("Stringed.play()");
30.   }
31.   public String what() { return "Stringed"; }
```

```
32.  public void adjust() {}
33.  }
34.  class Brass extends Wind {
35.  public void play() {
36.  System.out.println("Brass.play()");
37.  }
38.  public void adjust() {
39.  System.out.println("Brass.adjust()");
40.  }
41.  }
42.  class Woodwind extends Wind {
43.  public void play() {
44.  System.out.println("Woodwind.play()");
45.  }
46.  public String what() { return "Woodwind"; }
47.  }
48.  public class MainClass {
49.  static void tune(Instrument i) {
50.  i.play();
51.  }
52.  static void tuneAll(Instrument[] e) {
53.  for(int i = 0; i < e.length; i++)
54.  tune(e[i]);
55.  }
56.  public static void main(String[] args) {
57.  Instrument[] orchestra = new Instrument[5];
58.  int i = 0;
59.  orchestra[i++] = new Wind();
60.  orchestra[i++] = new Percussion();
61.  orchestra[i++] = new Stringed();
62.  orchestra[i++] = new Brass();
63.  orchestra[i++] = new Woodwind();
64.  tuneAll(orchestra);
65.  }
66.  }
```

شرح السطور:

- في السطور من 2 إلى 8 يتم تعريف interface جديد بالاسم Instrument يحتوى على متغير وثلاث دوال Methods ، وبالطبع لم يتم كتابة كود الدوال Method ، بل هو مجرد إعلان ، ثم يتم تعريفها فى الفصائل classes التي تنفذ implement هذا الـ interface بافتراض أن هذا الـ interface هو الشكل العام للأجهزة وبه الوظائف العامة للأجهزة مثل التشغيل ()play والضبط ()adjust ، وعلى فصيلة كل جهاز أن تقوم بتعريف هذه الدوال Methods بما يناسب طبيعة الجهاز ... هذا المثال لتقريب فكرة الـinterface .
- فى السطور 9 و 18 و 26 يتم إنشاء فصائل classes تقوم بتنفيذ implement الـ interface المسمي Instrument وتقوم بتعريف جميع الدوال Methods الموجودة فيه حسب ما يناسب الفصيلة class.
- فى السطر رقم 42 يتم تعريف فصيلة class جديدة بالاسم Woodwind ترث أحد الفصائل classes المعرفة مسبقاً وهى الفصيلة Wind وبالتالى ترث جميع خواصها ولكن تقوم بإعادة تعريف الدوال Methods باستخدام خاصية نسخ الدوال Method Overriding.
- فى السطر رقم 48 يتم انشاء الفصيلة class الرئيسية التى تحتوى على الدالة الرئيسية ()main وفيها يتم استعمال الفصائل classes التي سبق تعريفها كعناصر في مصفوفة Array مكونة من خمسة عناصر ، ثم يتم استدعاء الدالة ()tuneAll التي تقوم باستدعاء الدالة ()tune والتي بدورها تستدعي الدالة ()play ، وهنا سيتم استدعاء الدالة ()play علي حسب نوع الهدف object المسمي i الذي استدعينا الدالة ()play منه.

ملخص الفصل:

- تعرضنا في هذا الفصل لشرح استخدامات الـ interface.
- في الفصل القادم سوف نتعرف - بإذن الله - علي الاستثناءات وكيفية معالجتها Exception Handling ، فتابع معنا الفصل القادم.