كلية الهندسة المعلوماتية

برمجة 3

Java Programming

ا. د. علي عمران سليمان

محاضرات الأسبوع الرابعة

Interfaces

الفصل الثاني 2023-2024

content

- Abstract Classes.
- Abstract Methods.
- Interfaces.
- Fields in Interfaces.
- Implementing Multiple Interfaces.
- Polymorphism with Interfaces.
- Default Methods

References

- Deitel & Deitel, Java How to Program, Pearson; 10th Ed(2015)
- د.علي سليمان، بنى معطيات بلغة JAVA، جامعة تشرين 2013-2014

Overriding method (replacement or expansion)

جامعة المنارة
MANARA UNIVERSITY

- يوجد نوعين لإعادة تعريف الطر ائق Overriding
-هما الاستبدال replacement والتوسيع expansion

- الاستبدال replacement: تطرقنا لإعادة كتابة طريقة ()get_salary سابقاً، وقلنا إن الكائن من إي صنف هو الذي يحدد الطريقة التي سيناديها والموجوده بنفس الصنف، وفي حال الرغبة بمنادات الطريقة من صنف الأب تسبق ; ()super.get_salary واستخدمت طريقة الاستبدال نظراً لإن كل طريقة جديدة هي استبدال للطريقة الموروثة.

- يستخدام constructors طريقة التوسيع، (مثال أخر: استخدام طريقة ()printAllDatails لطباعة معلومات Person وتوسيعها ضمن الصنف Employee لإكمال مايخص الموظف هنا أول سطر سيكون مناداة طريقة طباعة Person وفق التالي ;()super. printAllDatails ثم كتابة مايخص الموظف)، عند اشتقاق كائن من الموظف ومناداة طريقة الطباعة سينفذ طريقة الأم ويكمل بتنفيذ طريقة الابن.

Reference type and object type

- ليكن لدينا مرجع من نوع موظف ويشير لكائن من نوع موظف.

Employee e1 =new Employee (“adam”,30,”Hama”, …);

- ليكن لدينا مرجع من نوع Employee ويشير لكائن من صنف SalariedEmployee (موظف شهري صنف مشتق من الأول). حيث أن لكل منها المعادلة التي تحسب راتبه.

Employee e2 =new SalariedEmployee (“mona”,20,”Aleppo”, …, , ,800,50);

- عند مناداة الطريقة ;()e1.get_salary سيتم تنفيذ الطريقة الموجودة ضمن Employee

وعند مناداة الطريقة ;()e2.get_salary سيتم تنفيذ الطريقة ضمن SalariedEmployee أي أن الكائن هو من يحدد أية طريقه سيتم تنفيذها.

- إن ()get_salary موجوده ضمن الصنف الأب وبالتالي معرفة على كل الأصناف الوارثة له.
- <u>بفرض أن طريقة معرفة ضمن SalariedEmployee وتم نداءها من e2 لن يتم التعرف عليها رغم أن الكائن من نفس الصنف الموجوده به الطريقة، إن نوع المرجع e2 هو Employee ولن يتعرف إلا على الطرق التي لها تعريف ضمن الصنف الأساس Employee.</u>

Abstract Classes 1

جامعة المنارة
MANARA UNIVERSITY

- لقد تطرقنا لبناء صنف أساس واشتقاق أصناف منه (وراثته) وإمكانية الحصول على كائنات منه ومن الأصناف المشتقة.

- بفرض أننا نحتاج لصنف معمم **generalized** وهو بمثابة قالب **Template** ولا نرغب بأن يتم اشتقاق كائن منه بل من أجل أن يصف الصنف ومحتوياته ونجبر كل الوارثين بنمط محدد ويكون بجعله **abstract** أي بمثابة صنف أساس للاصناف الأخرى.

- الطريقة المجردة **abstract** ليس لها جسم ويجب **overridden** في كل الأصناف الفرعيه غيرالمجردة، وغير ذلك سنحصل على خطأ.

- أي صنف يحتوي على طريقة مجردة **abstract** يصبح تلقائيًا صنفاً مجرداً **abstract** ويتم ذلك بإضافة الكلمة المفتاحية **abstract** في تعريف الصنف ما قبل الكلمة المفتاحية class وبعد نوع الوصول للصنف وعندها سيمنع المطابق اشتقاق كائن منه ويعطي خاطأ عند محاولة ذلك.

Abstract Classes 2

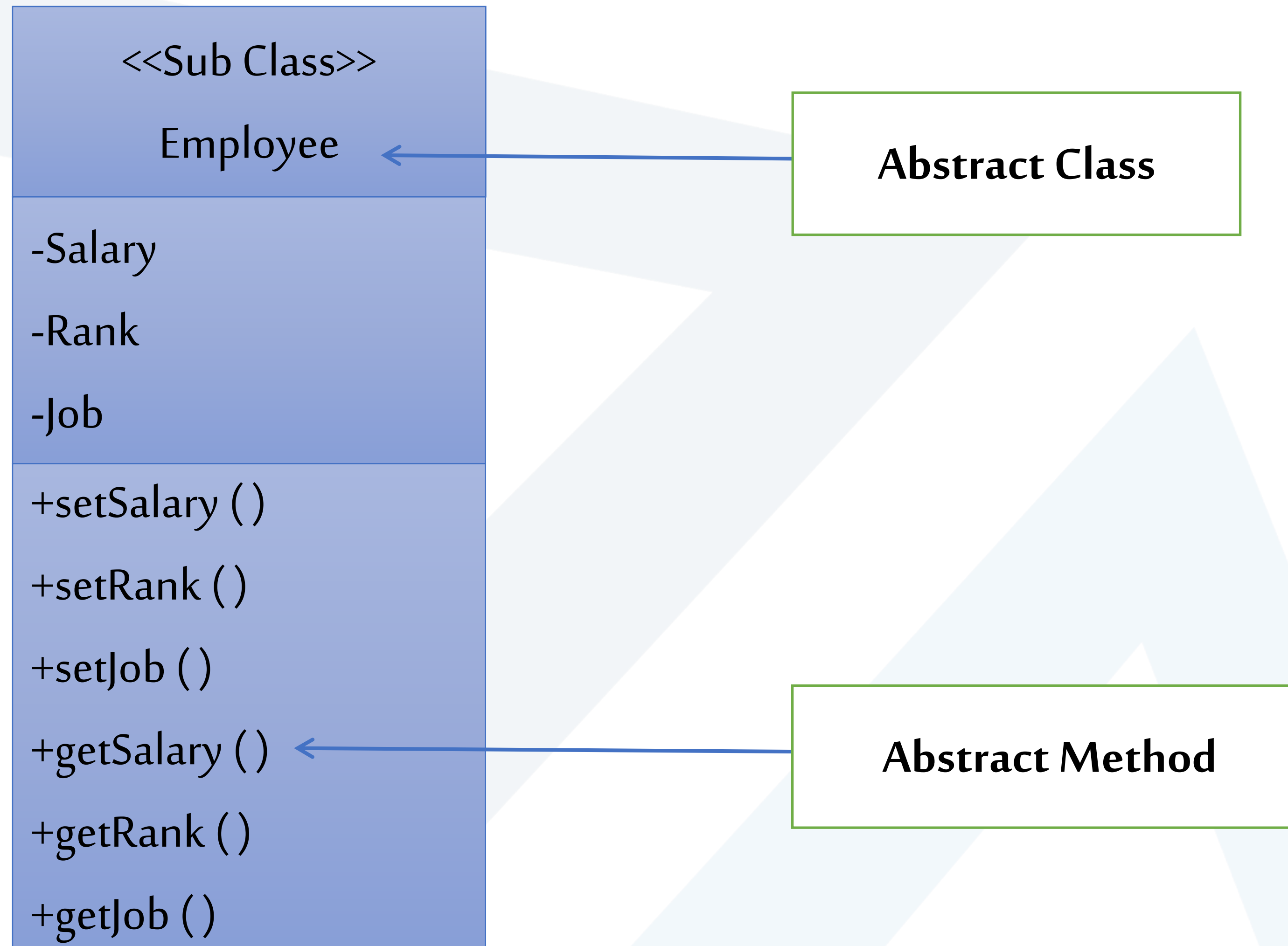
جامعة المنارة
MANARA UNIVERSITY

- يمثل الصنف المجرد الشكل العام أو الشكل المجرد لجميع الأصناف المشتقة منه.

```
AccessSpecifier abstract ReturenType MethodName(ParameterList);
Ex: public abstract void getSalary ( );
```

- على كل الأصناف الوارثة لصنف مجرد وغير المجردة أن تقوم بكتابة كل الطرق المجردة الموروثة من الصنف الأساس كل بما ينسجم مع مهمته.
- يتم كتابة الطرق التي لن تتغير ضمن الأصناف الوارثة، ضمن الصنف المعمم لتوفير الكتابات المتكررة.
- يتم استخدام طرق مجردة للتأكد من أن الصنف الفرعي سينفذ **implements** الطريقة.
- إذا فشل الصنف الفرعي في **override** لأي طريقة مجردة ، سينتج خطأ في المترجم.

Abstract Methods



an exercise

- حول الطريقة ()getSalary في الصنف Employee إلى طريقة مجرده بوضع abstract ما قبل القيمة المعادة وحذف جسمها،
- سيحل خطأ يطالب بتحويل الصنف Employee المدروس سابقاً لصنف مجرد ويتم بوضع abstract مابعد نوع الوصول والكلمة المفتاحية class ستلاحظ عدم إمانية اشتقاق كائن منه وعند المحاولة سنحصل على خطأ.
- ستجد عند اشتقاق صنف SalariedEmployee من الصنف Employee ستجد وجود خطأ يطالب بإعادة كتابة الطريقة ()getSalary لإنها طريقة مجرده إلا إذا كان الصنف الجديد مجرد.
- هذا ينطبق على الصنف HourlyEmployee ايضاً، وكل الآصناف المشتقة من الصنف المجرد Employee.
- إذا كانت كل طرائق الصنف مجردة تحول إلى واجهه Interfaces وهذا ماسنهتم به الأن.
- الشكل العام لتعريف الواجهة:

```
public interface InterfaceName
  {      (Method headers…)    }
```

Interfaces 1

جَامعة المَنارة
MANARA UNIVERSITY

- الواجهة interface تشبه الاصناف المجردة وتحتوي جميع الطرق المجردة والمجردة فقط.
- الغاية من الواجهة interface هو تحديد السلوكيات للاصناف الأخرى المحققه لها.
- الواجهة interface. هي عبارة عن مجموعة من تصريحات الطرائق بدون أي أجسام لطرائقها، أي أن طرائق الواجهة يكتب نموزجها فقط proto type (أي مجرد تواقيع طرائق).
- إن هذا التحديد بدوره، يفرض من قبل المترجم أو نظام وقت التنفيذ، والذي يتطلب أن تكون أنواع البارامترات التي تمرر عادة إلى الطرائق تتوافق بصرامة مع النوع المحدد في الواجهة.
- عندما يقوم صنف بتحقيق أو بناء implements واجهة، فيجب أن يبني جميع الطرائق المصرح عنها فيها، بهذه الاسلوب، فإن الواجهات تتطلب وجود صنف بناء يملك طرائقها بنفس التواقيع المحددة.
- يقال غالبًا أن الواجهة تشبه "العقد" contract، وعندما تنفذ صنف لواجهة ، يجب أن يلتزم بالعقد.

Interfaces 2

Contract

Class Photograph

public String description()

{ return descript; }

.

.

.

.

.

public int lowestPrice()

{return price/2;}

interface Sellable

1 – description( )

.

.

.

.

.

.

.

.

7 - listPrice( )

8 - lowestPrice( )

Class Photograph

Implements interface Sellable

Interface

interface Sellable

```java
/** Interface for objects that can be sold. */
public interface Sellable {

  /** description of the object */
  public String description();

  /** list price in cents */
  public int listPrice();

  /** lowest price in cents we will accept */
  public int lowestPrice();
}
```

الواجهة Sellable

class Photograph implements Sellable

```
/** Class for photographs that can be sold */
public class Photograph implements Sellable {
  private String descript;                                  // description of this photo
  private int price;                                        // price we are setting
  private boolean color;                                    // true if photo is in color
  public Photograph(String desc, int p, boolean c) {        // constructor
    descript = desc; price = p; color = c; }
  public String description()     { return descript; }
  public int listPrice()          { return price; }
  public int lowestPrice()        { return price/2;}
  public boolean isColor()        { return color; }
public String toString() {                                  // for printing
  return "( descript: " + descript +" SlistPrice: " + price + ", lowestPrice: " + price/2 + ", Color: " + color + ")";
  }
}
```

الصنف Photograph يحقق الواجهه Sellable

Interfaces 3

- لنفرض أننا نرغب بإنشاء جرد بالتحف التي نملكها، مصنفة كأغراض من أنواع مختلفة. وقد نرغب بتعريف بعض الأشياء على أنها قابلة للبيع، بحيث إننا قمنا ببناء الواجهة Sellable وبعدها بناء الصنف Photograph الذي يبني الواجهة Sellable والمحقق لكل طرق الواجهه Sellable بحسب الحاجة، للإشارة إلى أننا نرغب بأن نكون قادرين على بيع أي من الأغراض من الصنف Photograph، بالإضافة إلى أنه تمت إضافة الطريقة isColor الخاصة بالأغراض من النوع Photograph.

- لا يمكن إنشاء مثيل من الواجهه. Sellable sel1= new Sellable(); // ERROR!

- تكون الوراثة المتعددة في لغة الجافا متاحة من أجل الواجهات وليس من أجل الأصناف.

- إن طرائق واجهة ما ليس لها أجسام تعريف وبالتالي لن يحصل التباس عند وجود طريقتين بنفس التوقيع في واجهتين نظراً لعدم وجود اجساد لها في الواجهه المورثة.

جَامعة المَنارة
MANARA UNIVERSITY

Interfaces 3

- تحتوي جميع الكائنات في Java على طريقة خاصة تسمى toString والتي تعرض تمثيل الشريط المحرفي String لمحتويات الكائن.

- الطريقة toString موجوده ضمن الكائن Object وتسمى هذه الطرق بطرق الخدمات العامة أيضًا، أو الواجهة العامة التي يوفرها الصنف لعملائه.

- عندما يتم ربط كائن بسلسلة، يتم استدعاء طريقة الكائن toString ضمنيًا للحصول على تمثيل سلسلة للكائن.

```
Photograph Pho1 = new Photograph("adam", 100, true );
System.out.println(Pho1);
```

- يمكن أيضًا استدعاء طريقة toString بشكل صريح.

```
System.out.println(Pho1. toString());
```

Multiple Inheritance in Interfaces 2

يمكن أن تضم مجموعة أغراض ونوعاً آخر من الأغراض التي يمكن نقلها، من أجل هذه الأغراض، يمكن أن نعرف الواجهة التالية:

```
/** Interface for objects that can be transported. */
public interface Transportable
{
  /** weight in grams */
  public int weight();
  /** whether the object is hazardous */
  public boolean isHazardous();
}
```

الواجهة Transportable: يمكن أن نعرف الصنف BoxedItem من أجل التحف المتنوعة التي يمكن أن نبيعها، نحزمها أو نشحنها.

وبالتالي، يبني الصنف BoxedItem طرائق الواجهة Sellable والواجهة Transportable مع إضافة طرائق مخصصة لتحديد قيمة التأمين لشحن صندوق و أبعاد الصندوق المشحون.

BoxedItem implements Sellable, Transportable

```
/** Class for objects that can be sold, packed, and shipped. */
public class BoxedItem implements Sellable, Transportable
{ private String descript;          // description of this item
  private int price;                // list price in cents
  private int weight;               // weight in grams
  private boolean haz;              // true if object is hazardous
  private int height;               // box height in centimeters
  private int width=0;              // box width in centimeters
  private int depth=0;              // box depth in centimeters
  public BoxedItem( String desc,int p, int w,  boolean h)
              {descript= desc;  price= p;  weight= w; haz= h;}   //Constructor
  public String description()       { return descript; }
  public int listPrice()  { return price; }
  public int lowestPrice()          {  return price/2; }
  public int weight()               { return weight; }
  public boolean isHazardous()      { return haz; }
  public int insuredValue()         {  return price*2;}
```

BoxedItem implements Sellable, Transportable and main()

```
public void setBox(int h, int w, int d)
   {  height = h;  width = w;  depth = d;  }              // end setBox
public String toString()  {                               // for printing BoxedItem
   return "descript: " + descript +"  SlistPrice: " + price + ", lowestPrice: " +   price/2 + ",\n weight: " + weight +", isHazardous: " + haz +",
insuredValue: " + price*2 + ",\n height: " + height +", width: " + width +", depth: " + depth +")";    }  //end toString in BoxedItem
public static void main( String args[] )
         {      Photograph Pho1 = new Photograph("adam", 100, true );
                 System.out.println(Pho1);
                System.out.println();System.out.println();
                 BoxedItem box1 = new BoxedItem("adam", 100,10, true );
                 System.out.println("\n");
                 box1.setBox(3, 5, 7); System.out.println("\n\n");
                System.out.println(box1.toString());
        } // end maim
}// end class BoxedItem
```

Multiple Inheritance in Interfaces 1

- عندما ينفذ صنف واجهات متعددة، يجب أن توفير الطرق المحددة من قبل كل منهم.
- لتحديد واجهات متعددة في تعريف صنف، تسرد أسماء الواجهات ، مفصولة عن بعضها بعضاً بفواصل، بعد الكلمة المفتاحية implements.
- بالعودة إلى مثال التحف، يمكن أن نعرف واجهة للأغراض المؤمن عليها كما يلي:

```
public interface InsurableItem extends Transportable, Sellable
{           /** Returns insured Value in cents */
    public int insuredValue();
}
```

- تدمج هذه الواجهة طرائق الواجهة Transportable مع طرائق الواجهة Sellable وتضيف طريقة أخرى insuredValue()

- إن مثل هذه الواجهة يمكن أن تتيح لنا تعريف الصنف BoxedItem2 كما يلي:

```
public class BoxedItem2 implements InsurableItem {
        // ... same code as class BoxedItem
}
```

- في هذه الحالة، لاحظ أن الطريقة insuredValue ليست اختيارية، في حين أنها كانت اختيارية في النسخة السابقة للصنف BoxedItem، وكذلك كل الطرق من الواجهتين الموروثتين.
- تتيح لنا الواجهات إرغام الأغراض على بناء طرائق محددة، إلا أن استخدام متحولات الواجهات مع أغراض حقيقية يتطلب أحيانا استخدام تحويل الأنماط.
- بفرض أننا صرحنا عن الواجهة Person المبينة في المقطع البرمجي التالي. لاحظ أن الطريقة EqualTo للواجهة Person تأخذ بارامتراً واحداً من النوع Person. وبالتالي، يمكن أن نمررغرض من أي صنف يبني الواجهة Person إلى هذه الطريقة.

Multiple Inheritance in Interfaces 4

```
public interface Person {
  public boolean equalTo (Person other);        // is this the same person?
  public String getName();                      //get this person's name
  public int getAge();                          // get this person's age
}
```

- عندما يشير متغيرواجهة إلى كائن:
  - فقط الطرق المعلنة في الواجهة تكون متاحة،بما يحاكي المعروض في الشرحة الرابعة.
  - مطلوب تحويل قسري casting للنوع الصحيح للوصول إلى الطرق الأخرى للكائن المشار إليه بواسطة مرجع الواجهة.

- نبين في المقطع البرمجي التالي صنفاً Student يبني الواجهة Person. تفترض الطريقة equalTo أن الوسيط (المصرح عنه من النوع Person) هو أيضاً من النوع Student ويقوم بتحويل تضييق من النمط Person (الواجهة) إلى النمط Student (الصنف) باستخدام تحويل صريح. إن عملية التحويل مسموحة في هذه الحالة.

```
public class Student implements Person {
String id;    String name;    int age;                    // simple constructor
public Student (String i, String n, int a){id=i; name=n; age=a;}

 // protected int studyHours() {return age/2;}
 public String getID () {return id;}                  //ID of the student
 public String getName(){return name;}             //Person interface
 public int getAge() {return age;}                    // Person interface
 public boolean equalTo (Person other) {
  //Person interface cast Person to Student
 Student otherStudent = (Student) other;    return (id.equals (otherStudent.getID()));  }

 public String toString(){    return "Student(ID: " + id + ", Name: " + name +", Age: " + age + ")";
                 }           // end toString
 }                           // end class Student
```

Student بناء الصنف

وبسبب الافتراض الذي قمنا به في بناء الطريقة equalTo، يجب علينا أن نتحقق أن تطبيقاً يستخدم أغراضاً من النوع Student لن تحاول إجراء مقارنة بين أغراض مع أنواع أخرى من الأغراض. وإلا، فإن تحويل الأنماط في الطريقة equalTo سيفشل.

إن القدرة على إجراء تحويلات تضييق من أنماط واجهات إلى أنماط اصناف تتيح لنا كتابة أنواع عامة من بنى المعطيات التي تتضمن افتراضات دنيا حول العناصر التي تقوم بتخزينها. نبين في المقطع التالي، كيف نقوم ببناء مجلد يخزن أزواجاً من الأغراض التي تبني الواجهة Person، تقوم الطريقة remove بعملية بحث ضمن محتويات المجلد وتحذف الزوج المحدد، إذا كان موجوداً، وكما في الطريقة findOther فإنها تستخدم الطريقة equalTo للقيام بذلك.

```
public class PersonPairDirectory
{
        // ... instance variables would go here ...
        public PersonPairDirectory()
                { /* default constructor goes here */}
        public void insert (Person person, Person other)
                { /* insert code goes here */}
        public Person findOther (Person person)
                {return null; }   // stub for find
        public void remove (Person person, Person other)
                { /* remove code goes here */}
 }// end class PersonPairDirectory
```

مثال عن صنف

Multiple Inheritance in Interfaces 8

الآن، بفرض أننا ملأنا المجلد myDirectory، بأزواج من الأغراض Student التي تمثل أزواجاً متجاورة. من أجل إيجاد مجاور غرض smart_one من النوع Student، يمكن أن نحاول القيام بمايلي (وهو خاطئ):

```
Student cute_one=myDirectory.findOther (smart_one); // wrong!
```

إن الأمر السابق يسبب خطأ ترجمة يدعى ""explicit-cast-required، المشكلة هنا هي أننا نحاول القيام بتحويل تضييق بدون معامل تحويل صريح. بمعنى، أن القيمة المعادة من الطريقة findOther هي من النوع Person في حين أن المتحول cute_one التي يتم إسنادها إليه هو من النوع الأضيق Student، وهو صنف يبني الواجهة Person. وبالتالي، يجب استخدام عملية تحويل صريحة لتحويل النمط Person إلى النمط Student، كما يلي:

```
Student cute_one=(Student)myDirectory. findOther(smart_one);
```

إن تحويل القيمة من النوع Personالمعادة من الطريقة findOther إلى النمط Student تعمل بشكل جيد طالما أننا متأكدين من أن الاستدعاء لـ myDirectory.findOther يعطينا حقيقة غرضاً من النوع Student. عموماً، يمكن أن تكون الواجهات أداة قيمة لتصميم بنى معطيات عامة، والتي يمكن تخصيصها من قبل مبرمجين آخرين من خلال استخدام تحويل الأنماط.

التعميم Generics

- في إطار عمل التعميمات generics framework لاستخدام الأنماط المجردة بطريقة تتجنب العديد من التحويلات الصريحة.
- النوع العمومي generic type هو نوع لايعرف في زمن الترجمة، وإنما يصبح محدداً كلياً في زمن التنفيذ.
- يتيح لنا إطار عمل التعميمات تعريف الصنف بدلالة مجموعة من البارامترات ذات الأنواع الشكلية formal type parameters التي يمكن أن تستخدم، على سبيل المثال، *لتجريد أنماط بعض المتحولات الداخلية للصنف، تستخدم أقواس زاوية angle brackets لحصر قائمة بالبارامترات ذات الأنماط الشكلية*. على الرغم من أن أي معرف صالح يمكن أن يستخدم من أجل بارامترذو نمط شكلي.
- إذا كان لدينا صنف قد عرف مع هذه البارامترات، عندها يمكن أن نقوم بتهيئة أو إنشاء غرض من هذا الصنف يستخدم بارامترات ذات أنماط فعلية للإشارة إلى الأنماط الحقيقية المستخدمة.
- يبين المقطع البرمجي التالي الصنف Pair الذي يخزن أزواجاً قيمة-مفتاح key-value pairs، حيث إن أنواع القيمة والمفتاح محددة بالبارامترات V و Kعلى التوالي. تقوم الطريقة main بإنشاء مثلين من هذا الصنف، الأول من أجل زوج String-Integer والثاني Student-Double: .

جَامعة المَنارة
MANARA UNIVERSITY

interface Sellable

```
public class Pair<K, V> { K key;  V value;
public void set(K k, V v){key= k;value= v;}
public K getKey() { return key; }
public V getValue() { return value; }
public String toString() {    return "[" + getKey() + ", " + getValue() + "]";  }

public static void main (String[] args)
{          Pair<String,Integer> pair1=new  Pair<String,Integer>();
          pair1.set(new String("height"), new Integer(36));
          System.out.println(pair1);
          Pair<Student,Double> pair2=new Pair<Student,Double>();
          pair2.set(new Student("A5976","Sue",19),new Double(9.5));
          System.out.println(pair2);
          System.out.println("\n\n"+pair2.equals(pair1));
                }// end main
}               // class Pair
```

```
[height, 36]
[Student(ID: A5976, Name: Sue, Age: 19), 9.5]

false
```

Polymorphism with Interfaces

- كما ذكرنا سابقاً !Sellable pho= new Sellable (); // wrong هي عملية غير مسموحة.
- تسمح Java بإنشاء متغيرات مرجعية للواجهة تشير إلى كائنات من الأصناف المحققة لها.
- يمكن للمتغير المرجعي للواجهة أن يشير إلى أي كائن يقوم بتنفيذ تلك الواجهة، بغض النظر عن نوع صنفه.
- في كود المثال ، تم التصريح عن متغيرين مرجعيين ، هما pho2, pho1 .
- يشير المتغير المرجعي pho1 إلى كائن Photograph ويشير متغير pho2 إلى كائن BoxedItem.
- عندما ينفذ صنف واجهة ، يتم إنشاء علاقة وراثة تعرف باسم وراثة الواجهة.

```
Sellable pho1= new Photograph ();
Sellable pho2= new BoxedItem ();
```

- عند ذكر .pho1 ستظهر كل الطرق المتاحة والمعرفة صمن الواجهه المحققة.
- الطرق المعرفة ضمن الصنف المحقق للواجهه لن يتم التعرف عليها لعدم معرفة الواجهه بها مثل ;()pho1.isColor على غرار مرجع من نوع ويشير على كائن من نوع مشق لن يصل للطرق التي لم تعرف في الأصل الشريحة أربعة.

Fields in Interfaces

- يمكن أن تحتوي الواجهة على حقول تصريحات:

  - يتم التعامل مع جميع الحقول في الواجهة على أنها final and static.

- لأنها تصبح نهائية بشكل تلقائي، يجب توفير قيمة تهيئة لها.

```
public interface Doable
{
        int FIELD1 = 1, FIELD2 = 2;
        (Method headers...)
}
```

- في هذه الواجهة ، FIELD1 و FIELD2هما متغيران صحيحان نهائيان ثابتان final and static.

- أي صنف ينفذ هذه الواجهة لديها حق الوصول إلى هذه المتغيرات.

جَـامعة
الْمَـنارة
MANARA UNIVERSITY

انتهت محاضرة الأسبوع 4