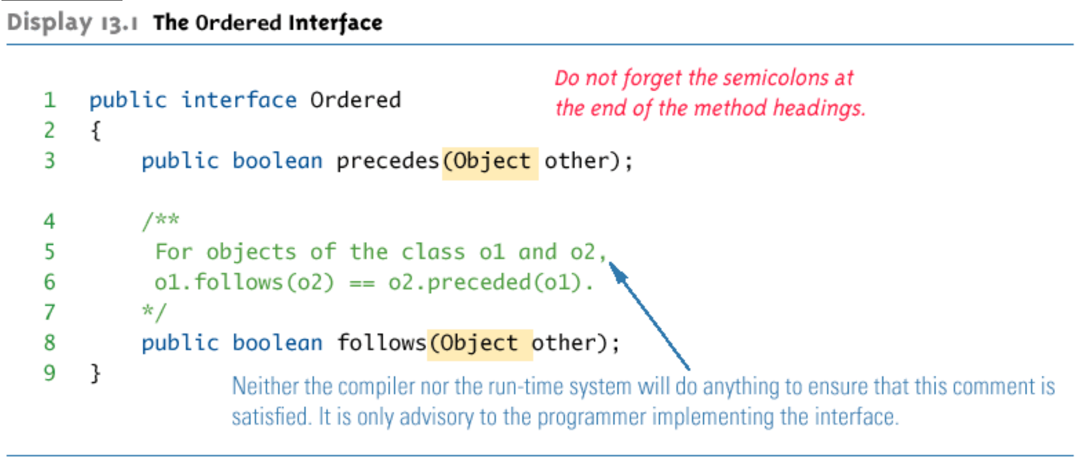
# Chapter 13 Interfaces and Inner Classes

`interface`:

1. an interface contains method headings and constant definitions only, no instance variables, no concrete methods
2. Java should do mutiple inheritance through interfaces
3. all method headings be public
4. to implement an interface…
   a. concrete class must implement all the method headings in the interface
   b. concrete class needs to include “`implements Interface_Name`” in the class declaration
5. 多重繼承會出問題，例如B和C繼承A，D繼承B和C，D在執行override method的時候不知道要用哪一個版本
6. implements, extends傻傻分不清楚？
   a. abstract class/class extends abstract class/class
   b. abstract class/class implements interface
   c. interface extends interface
   d. interface implements abstract class/class (不存在，因為interface不實作)

7. for example

<u>interface:</u>



Display 13.1 **The Ordered Interface**

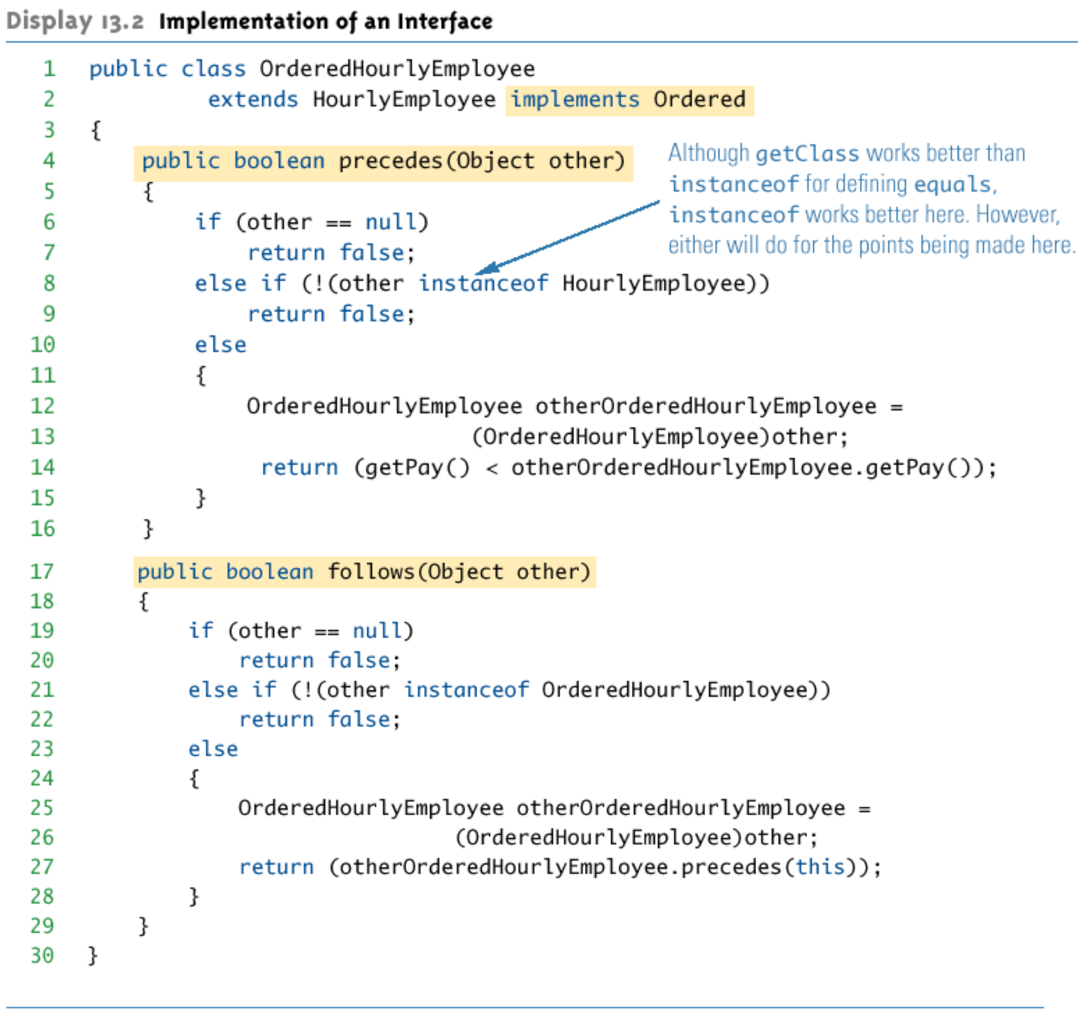
*Do not forget the semicolons at the end of the method headings.*

```java
public interface Ordered
{
    public boolean precedes(Object other);

    /**
     For objects of the class o1 and o2,
     o1.follows(o2) == o2.preceded(o1).
    */
    public boolean follows(Object other);
}
```

Neither the compiler nor the run-time system will do anything to ensure that this comment is satisfied. It is only advisory to the programmer implementing the interface.

<u>class implements interface:</u>



Display 13.2 **Implementation of an Interface**

```java
public class OrderedHourlyEmployee
          extends HourlyEmployee implements Ordered
{
    public boolean precedes(Object other)
    {
        if (other == null)
            return false;
        else if (!(other instanceof HourlyEmployee))
            return false;
        else
        {
            OrderedHourlyEmployee otherOrderedHourlyEmployee =
                                 (OrderedHourlyEmployee)other;
             return (getPay() < otherOrderedHourlyEmployee.getPay());
        }
    }

    public boolean follows(Object other)
    {
        if (other == null)
            return false;
        else if (!(other instanceof OrderedHourlyEmployee))
            return false;
        else
        {
            OrderedHourlyEmployee otherOrderedHourlyEmployee =
                                 (OrderedHourlyEmployee)other;
            return (otherOrderedHourlyEmployee.precedes(this));
        }
    }
}
```

Although `getClass` works better than `instanceof` for defining `equals`, `instanceof` works better here. However, either will do for the points being made here.

abstract class implements interface:

**Display 13.3 An Abstract Class Implementing an Interface**

```java
public abstract class MyAbstractClass implements Ordered
{
    int number;
    char grade;

    public boolean precedes(Object other)
    {
        if (other == null)
            return false;
        else if (!(other instanceof HourlyEmployee))
            return false;
        else
        {
            MyAbstractClass otherOfMyAbstractClass =
                                              (MyAbstractClass)other;
            return (this.number < otherOfMyAbstractClass.number);
        }
    }

    public abstract boolean follows(Object other);

}
```

interface extends interface:

**Display 13.4 Extending an Interface**

```java
public interface ShowablyOrdered extends Ordered
{
    /**
      Outputs an object of the class that precedes the calling object.
    */
    public void showOneWhoPrecedes();
}
```

Neither the compiler nor the run-time system will do anything to ensure that this comment is satisfied.

*A (concrete) class that implements the `ShowablyOrdered` interface must have a definition for the method `showOneWhoPrecedes` and also have definitions for the methods `precedes` and `follows` given in the `Ordered` interface.*

8. `Comparable` Interface
    a. in `java.lang` package
    b. all Wrapper class has `Comparable` interface
    c. `Comparable` has only one method heading:

```
/** returns a negative number if the calling object “comes
     before” the parameter

    returns zero if the calling object “equals” the
     parameter

    returns a positive number if the calling object “comes
    after” the parameter
*/
public int compareTo(Object other)
```

    d. 宣告interface的地方必須要打上註解，因為在interface內沒有實作出來，所以需要註明如何讓繼承者使用
    e. for example: `GeneralizedSelectionSort` class，它負責接收`Comparable`類型的reference

```java
public class GeneralizedSelectionSort
{
    /**
     Precondition: numberUsed <= a.length;
                   The first numberUsed indexed variables have values.
     Action: Sorts a so that a[0, a[1], ... , a[numberUsed - 1] are in
     increasing order by the compareTo method.
    */
    public static void sort(Comparable[] a, int numberUsed)
    {
        int index, indexOfNextSmallest;
        for (index = 0; index < numberUsed - 1; index++)
        {//Place the correct value in a[index]:
            indexOfNextSmallest = indexOfSmallest(index, a, numberUsed);
            interchange(index,indexOfNextSmallest, a);
             //a[0], a[1],..., a[index] are correctly ordered and these are
             //the smallest of the original array elements. The remaining
             //positions contain the rest of the original array elements.
        }
    }

    /**
     Returns the index of the smallest value among
     a[startIndex], a[startIndex+1], ... a[numberUsed - 1]
    */
    private static int indexOfSmallest(int startIndex,
                                       Comparable[] a, int numberUsed)
    {
        Comparable min = a[startIndex];
        int indexOfMin = startIndex;
        int index;
        for (index = startIndex + 1; index < numberUsed; index++)
            if (a[index].compareTo(min) < 0)//if a[index] is less than min
            {
                min = a[index];
                indexOfMin = index;
                //min is smallest of a[startIndex] through a[index]
            }
        return indexOfMin;
    }

    /**
     Precondition: i and j are legal indices for the array a.
     Postcondition: Values of a[i] and a[j] have been interchanged.
    */
    private static void interchange(int i, int j, Comparable[] a)
    {
        Comparable temp;
        temp = a[i];
        a[i] = a[j];
        a[j] = temp; //original value of a[i]
    }

}
```

使用GeneralizedSelectionSort的方法：

```java
/**
 Demonstrates sorting arrays for classes that
 implement the Comparable interface.
*/
public class ComparableDemo
{
    public static void main(String[] args)
    {
        Double[] d = new Double[10];
        for (int i = 0; i < d.length; i++)
            d[i] = new Double(d.length - i);

        System.out.println("Before sorting:");
        int i;
        for (i = 0; i < d.length; i++)
            System.out.print(d[i].doubleValue() + ", ");
        System.out.println();

        GeneralizedSelectionSort.sort(d, d.length);

        System.out.println("After sorting:");
        for (i = 0; i < d.length; i++)
            System.out.print(d[i].doubleValue() + ", ");
        System.out.println();

        String[] a = new String[10];
        a[0] = "dog";
        a[1] = "cat";
        a[2] = "cornish game hen";
        int numberUsed = 3;

        System.out.println("Before sorting:");
        for (i = 0; i < numberUsed; i++)
            System.out.print(a[i] + ", ");
        System.out.println();

        GeneralizedSelectionSort.sort(a, numberUsed);

        System.out.println("After sorting:");
        for (i = 0; i < numberUsed; i++)
            System.out.print(a[i] + ", ");
        System.out.println();
    }
}
```

*The classes Double and String do implement the Comparable interface.*

**Sample Dialogue**

```
Before Sorting
10.0, 9.0, 8.0, 7.0, 6.0, 5.0, 4.0, 3.0, 2.0, 1.0,
After sorting:
1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0,
Before sorting;
dog, cat, cornish game hen,
After sorting:
cat, cornish game hen, dog,
```

9. 當一個interface沒有method headings也無constants（completely empty），其用途僅作分類、標記，例如：
    a. `Serializable` Interface
    b. `Cloneable` Interface：
        i. 所有繼承Object class都有的`clone`，但是不一定有被override。若想要標示有override `clone` method，可以implement `Cloneable` interface。所以`Cloneable` interface是用來指示`clone` method是否有被使用
        ii. (以後才會回來看exception)

Inner Classes：

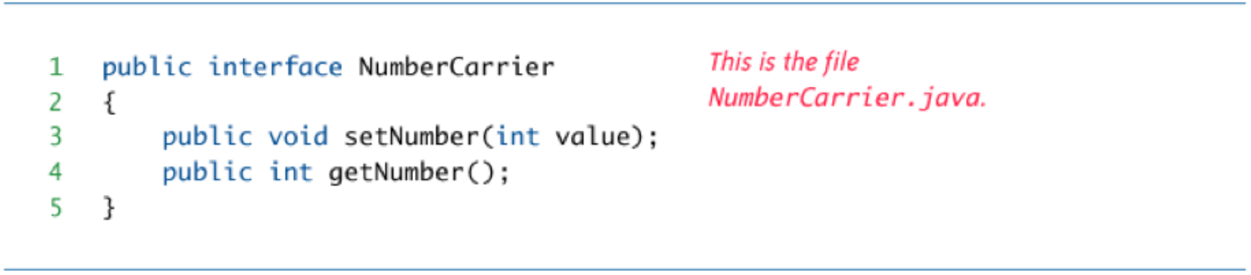
1. inner class is a member of the outer class
2. if inner class is private, then outside the outer class is unaccessable to inner class
3. advantage:
    a. inner class can be hidden
    b. inner class can be used as helping class
4. inner and outer classes have access to each other's private members, for example:

```
public class Outer{
        private int x1;
        private int x3;
        private void m1(){
                Inner inRef = new Inner();
                inRef.x2 = 1;               //private member access
        }
        private class Inner{
                private int x2;
                private int x3;
                private void m2(){
                        x1 = 2;             //private member access
                        x3 = 3;             //自己的x3
                        Outer.this.x3 = 4;  //Outer private access
                }
        }
}
```

5. Java produces a .class file for any class named `ClassName`.class for outer class and `ClassName$InnerClassName`.class for inner class
6. static inner class
    a. instance variables of the outer class cannot be referenced
    b. nonstatic methods of the outer class cannot be invoked
7. public inner class
    a. can be used outside of the outer class
    b. 宣告方法：
    `OuterClass.InnerClass inObject=new OuterClass.InnerClass()`
8. derived inner class is allowed, for example

```
public class Outer{
      private int x1;
      private int x3;
      private void m1(){
            Inner inRef = new Inner();
            inRef.x2 = 1;    //private member access
      }
      private class Inner extends Outer{
            private int x2;
            private int x3;
            private void m2(){
                  x1 = 2;                //private member access
                  x3 = 3;                //自己的x3
                  Outer.this.x3 = 4;     //Outer private access
            }
      }
}
//Inner class即包含了Outer class所有的data member和member function
```

9. Anonymous Class
    a. temporary use of class
    b. class definition is embedded inside the expression with the `new` operator
    c. for example:



```
public interface NumberCarrier
{
    public void setNumber(int value);
    public int getNumber();
}
```

*This is the file NumberCarrier.java.*

*This is just a toy example to demonstrate the Java syntax for anonymous classes.*

```java
public class AnonymousClassDemo
{
    public static void main(String[] args)
    {
        NumberCarrier anObject =
                    new NumberCarrier()
                    {
                        private int number;
                        public void setNumber(int value)
                        {
                            number = value;
                        }
                        public int getNumber()
                        {
                           return number;
                        }
                     };

        NumberCarrier anotherObject =
                    new NumberCarrier()
                    {
                        private int number;
                        public void setNumber(int value)
                        {
                            number = 2*value;
                        }
                        public int getNumber()
                        {
                            return number;
                        }
                    };

        anObject.setNumber(42);
        anotherObject.setNumber(42);
        showNumber(anObject);
        showNumber(anotherObject);
        System.out.println("End of program.");
    }

    public static void showNumber(NumberCarrier o)
    {
        System.out.println(o.getNumber());
    }

}
```

*This is still the file AnonymousClassDemo.java.*

10. inner class規則
    a. interface內<u>不</u>可有inner class (因為interface不實作東西)
    b. interface內可有inner interface
    c. class內可有inner abstract class
    d. class內可有inner interface
    e. abstract內可有inner class
    f. abstract內可有inner interface