Chapter 12
Interfaces

Lesson page 12-1. Interfaces

Activity 12-1-1 The interface

Question 1. Interface means, literally, between the faces. In general, an interface is a plane or other surface forming a common boundary of bodies or spaces.

Question 2. The form of an interface is:

```
interface <identifier> {
    <abstract methods and constant definitions>
}
```

Question 3. The interface is abstract, like an abstract class, in the sense that it cannot be instantiated. Method `frooble` is abstract, because it does not have a method body.

Question 4. False.

Activity 12-1-2 Implementing an interface

Question 5. `C`'s header: **public class** `C` **implements** `In`

Question 6. True.

Activity 12-1-3 The real use of interface`ActionListener`

Question 7. The header is:
public class `SubFrame` **extends** `Frame` **implements** `ActionListener`

Question 8. The statement that adds `b` is: `add(b);` .

Question 9. This call registers `SubFrame` as a listener of `b`:

```
b.addActionListener(this);
```

Question 10. This method will be called when the button is pressed:

```
public void actionPerformed (ActionEvent e) {
    System.out.println("b was pressed.");
}
```

Lesson page 12-2. The interface as a type

Activity 12-2-1 The interface

Question 1. Orthogonal: lying or intersecting at right angles; mutually perpendicular; completely independent.

Question 2. True.

Question 3. No type casts are necessary.

Question 4. An instance of an interface `In` (say) is really an instance of a class `C` (say) that implements the interface, so the instance of `In` has all the variables and methods that the instance of `C` has.

Activity 12-2-2 Implementing more than one interface

Question 5. **public class** `C` **implements** `In, Jn`

Question 6. Class `C` has to define all the methods that are defined in interfaces `In` and `Jn`.

Activity 12-2-3 Extending an interface

Question 7. In Java, multiple inheritance occurs when a class inherits the same method (with the same signature) from two different sources —from a superclass and an interface or from two interfaces.

Question 8. False.

Question 9. True.

Question 10. False.

Activity 12-2-4 Exercises on interfaces

Lesson page 12-3. Interface Comparable

Activity 12-3-1 Interface `Comparable`

Question 1. Here is interface `Comparable`.

```
public interface Comparable {
    // < 0 if b < this object, 0 if b = this
    // object, and > 0 if b > this object
    int compareTo(Object b);
}
```

Note that the parameter of `compareTo` is of type `Object`! In a few places, *ProgramLive* mistakenly says that it is of type `Comparable`.

Question 2. Interface `Comparable` is useful in any class in which there is an ordering of objects; method `Comparable` defines the ordering. Examples are

the wrapper classes `Integer` and `Double`, a class whose instances are dates, a class whose instances are times, and a class whose instances are colors, where some ordering of colors can be given.

Activity 12-3-2 Implementing class Comparable

Question 3. This is a bad question, because it can't be answered. Class `Comparable` has nothing to do with negative values, only with comparing values to see whether one is smaller than, equal to, or greater than another. For example, the standard implementations of time on our computers do not allow negative values. Our apologies.

In place of this question, we could have a question to write a nonstatic method (in a class `C`) with `Comparable` array `b` as a parameter that tests whether at least one element of `b` is less than the object in which the method occurs. We give the outline of class `C` as well:

```
import java.util.*;

public class C implements Comparable {
    // = <0 if this object is < x; 0 if this
    // object = x; > 0 if this object > b
    public int compareTo(Object x) {
        C c= (C) x;
        // put code here to do the comparison
        // and return -1, 0, or 1
    }

    // = "an element of b is less than this object"
    public boolean hasSmaller(Comparable[] b) {
        // inv: no element of b[0..i-1] is less than this
        for (int i= 0; i != b.length; i++) {
            if (compareTo(b[i]) > 0) {
                return true;
            }
        }
        return false;
    }
}
```

Question 4. The header is: **public class** `C` **implements** `Comparable`

Question 5. The answer depends on what version of Java you are using. The old version 1.1 did not have interface `Comparable`, and consequently, the wrapper classes like `Integer` did not implement `Comparable`. In version 1.2, wrapper class `Integer` does indeed implement interface `Comparable`, so the methods of class `Compares` will indeed work on `Integer`s, as well as `Pixel`s,

where class `Pixel` is defined on lesson page 12.3.

Activity 12-3-3 Casting between Pixel and Comparable

Question 6. False.

Question 7. True.

Lesson page 12-4. Interface Enumeration

Activity 12-4-1 Interface Enumeration

Question 1. An enumeration is a detailed list; an account of a number of things, in which mention is made of every one of them. One can have an enumeration of the natural numbers, 0, 1, 2, 3, ..., even though there are an unbounded number of them.

Question 2. Method `nextElement` has to yield an element of class `Object` (or of some subclass of it), and **char** is a primitive type, not a class type.

Question 3. The two methods in every `Enumeration` implementation:

```
// = ‘‘there are more objects to enumerate’’
boolean hasMoreElements()

// = the next object to enumerate. If there are
// no more, throw a NoSuchElementException
Object nextElement()
```

Question 4. Here's the method:

```
// = the number of blanks in s
public static int numberOfBlanks(String s) {
    StringEnumeration= new StringEnumeration(s);
    int x= 0;
    // {x = number of blanks in chars seen thus far}
    while (e.hasMoreElements()) {
        char c= ((Character)e.nextElement()).charValue();
        if (c == ' ') {
             x= x+1;
        }
    }
    return x;
}
```

Activity 12-4-2 A neat use of Enumeration

Question 5. False. The method `print` that is given in this activity can be used to print any instance of `Enumeration` (or its subclasses).

Question 6. Here is method `print` with an `Iterator` as a parameter:

```
// Print the contents of Iterator e.
public static void print(Iterator e) {
    while (e.hasNext()) {
        System.out.println(e.next());
    }
}
```

Question 7. Here is class `StringIterator`:

```
import java.util.*;
public class StringIterator implements Iterator {
    String s; // The String to be enumerated
    int k= 0; // s[k] is next char to be enumerated

    // Constructor: an instance to enumerate sp
    public StringIterator(String sp) {
         s= sp;
     }

    // = "there are more elements to enumerate"
    public boolean hasNext()
        { return k != s.length(); }

    // = The next element to enumerate
    public Object next() {
        if (!hasNext()) {
            throw new NoSuchElementException(
                "no more characters");
        }
        k= k+1;
        return new Character(s.charAt(k-1));
    }

    // Remove --not implemented
    public void remove() {
        throw new UnsupportedOperationException();
    }
}
```