18. Interfaces

Objetivos:
a) Definir el concepto de interfaz
b) Interpretar el código fuente de una aplicación Java donde aparecen interfaces
c) Construir una aplicación Java sencilla, convenientemente especificada, que declare y utilice una interfaz

Una *interfaz* es una especie de plantilla para la construcción de clases. Normalmente una interfaz se compone de un conjunto de declaraciones de **cabeceras** de **métodos** (sin implementar, de forma similar a un método abstracto) que especifican un **protocolo de comportamiento** para una o varias clases. Además, una clase puede *implementar* una o varias interfaces: en ese caso, la clase debe proporcionar la declaración y definición de **todos** los métodos de cada una de las interfaces o bien declararse como clase `abstract`. Por otro lado, una interfaz puede emplearse también para declarar **constantes** que luego puedan ser utilizadas por otras clases.

Una *interfaz* puede parecer similar a una *clase abstracta*, pero existen una serie de diferencias entre una interfaz y una clase abstracta:

- **Todos** los métodos de una interfaz se declaran implícitamente como abstractos y públicos.
- Una clase abstracta no puede *implementar* los métodos declarados como abstractos, una interfaz no puede *implementar* ningún método (ya que todos son abstractos).
- Una interfaz no declara variables de instancia.
- Una clase puede implementar varias interfaces, pero sólo puede tener una clase ascendiente directa.
- Una clase abstracta pertenece a una jerarquía de clases mientras que una interfaz **no** pertenece a una jerarquía de clases. En consecuencia, clases sin relación de herencia pueden implementar la misma interfaz.

18.1. Declaración de una interfaz

La declaración de una interfaz es similar a una clase, aunque emplea la palabra reservada `interface` en lugar de `class` y no incluye ni la declaración de variables de instancia ni la implementación del cuerpo de los métodos (sólo las cabeceras). La sintaxis de declaración de una interfaz es la siguiente:

```
public interface IdentificadorInterfaz {
    // Cuerpo de la interfaz . . .
}
```

Una interfaz declarada como `public` debe ser definida en un archivo con el mismo nombre de la interfaz y con extensión .java. Las cabeceras de los métodos declarados en el cuerpo de la interfaz se separan entre sí por caracteres de punto y coma y todos son declarados implícitamente como `public` y `abstract` (se pueden omitir). Por su parte, todas las constantes incluidas en una interfaz se declaran implícitamente como `public`, `static` y `final` (también se pueden omitir) y es necesario inicializarlas en la misma sentencia de declaración.

Por ejemplo, la interfaz `Modificacion` declara la cabecera de un único método:

```
/**
 *  Declaracion de la interfaz Modificacion
 *  @author A. Garcia-Beltran
 *  Ultima revision: noviembre, 2007
 */
public interface Modificacion {
    void incremento(int a);
}
```

que se almacena en el archivo fuente `Modificacion.java` y que, al compilarse:

```
$>javac Modificacion.java↵
```

genera un archivo `Modificacion.class`. Al no corresponder a una clase que implementa un método `main`, este archivo no puede ejecutarse con el intérprete de Java.

Segundo ejemplo: la interfaz `constantes` declara dos constantes reales con el siguiente código fuente:

```
/**
 *  Declaracion de la interfaz Constantes
 *  @author A. Garcia-Beltran
 *  Ultima revision: noviembre, 2007
 */
public interface Constantes {
    double valorMaximo = 10000000.0;
    double valorMinimo = -0.01;
}
```

que se almacena en el archivo fuente `Constantes.java` y que, al compilarse, genera un archivo `Constantes.class`

Tercer ejemplo: la interfaz `Numerico` declara una constante real y dos cabeceras de métodos con el siguiente código fuente:

```
/**
 *  Declaracion de la interfaz Numerico
 *  @author A. Garcia-Beltran
 *  Ultima revision: noviembre, 2007
 */
public interface Numerico {
    double EPSILON = 0.000001;
    void establecePrecision(float p);
    void estableceMaximo(float m);
}
```

que se almacena en el archivo fuente `Numerico.java` y que, al compilarse, genera un archivo `Numerico.class`.

18.2. Implementación de una interfaz en una clase

Para declarar una clase que implemente una interfaz es necesario utilizar la palabra reservada `implements` en la cabecera de declaración de la clase. Las cabeceras de los métodos (identificador y número y tipo de parámetros) deben aparecer en la clase tal y como aparecen en la interfaz implementada. Por ejemplo, la clase `Acumulador` implementa la interfaz `Modificacion` y por lo tanto debe declarar un método `incremento`:

```
/**
 *  Declaracion de la clase Acumulador
 *  @author A. Garcia-Beltran
 *  Ultima revision: noviembre, 2007
 */
public class Acumulador implements Modificacion {
    private int valor;
    public Acumulador (int i) {
        valor = i;
    }
    public int daValor () {
        return valor;
    }
    public void incremento (int a) {
        valor += a;
    }
 }
```

Esta cabecera con la palabra `implements`... implica la obligación de la clase `Acumulador` de definir el método `incremento` declarado en la interfaz `Modificacion`. El siguiente código muestra un ejemplo de uso de la clase `Acumulador`.

```
/**
 *  Demostracion de la clase Acumulador
 *  @author A. Garcia-Beltran
 *  Ultima revision: noviembre, 2007
 */
public class PruebaAcumulador {
    public static void main (String [] args) {
        Acumulador p = new Acumulador(25);
        p.incremento(12);
        System.out.println(p.daValor());
    }
}
```

La compilación y posterior ejecución del código anterior origina la siguiente salida por pantalla:

```
$>javac PruebaAcumulador.java↵

$>java PruebaAcumulador↵
37
```

La clase `Acumulador` tendría también la posibilidad de utilizar directamente las constantes declaradas en la interfaz si las hubiera.

Para poder emplear una constante declarada en una interfaz, las clases que no implementen esa interfaz deben anteponer el identificador de la interfaz al de la constante.

18.3. Jerarquía entre interfaces

La *jerarquía* entre interfaces permite la *herencia* simple y múltiple. Es decir, tanto la declaración de una clase, como la de una interfaz pueden incluir la implementación de otras interfaces. Los identificadores de las interfaces se separan por comas. Por ejemplo, la interfaz `Una` implementa otras dos interfaces: `Dos` y `Tres`.

```
public interface Una implements Dos, Tres {
    // Cuerpo de la interfaz . . .
}
```

Las clases que implementan la interfaz `Una` también lo hacen con `Dos` y `Tres`.

Otro ejemplo: pueden construirse dos interfaces, `Constantes` y `Variaciones`, y una clase, `Factura`, que las implementa:

```
// Declaracion de la interfaz Constantes
public interface Constantes {
    double valorMaximo = 10000000.0;
    double valorMinimo = -0.01;
}

// Declaracion de la interfaz Variaciones
public interface Variaciones {
    void asignaValor(double x);
    void rebaja(double t);
}

// Declaracion de la clase Factura
public class Factura implements Constantes, Variaciones {
    private double totalSinIVA;
    public final static double IVA = 0.16;
    public double sinIVA() {
        return totalSinIVA;
    }
    public double conIVA() {
        return totalSinIVA * (1+IVA);
    }
    public void asignaValor(double x) {
        if (valorMinimo<x) totalSinIVA=x;
           else totalSinIVA=0;
    }
    public void rebaja(double t) {
        totalSinIVA *= (1-t/100);
    }
    public static void main (String [] args) {
        factura a = new factura();
        a.asignaValor(250.0);
        System.out.println("El precio sin IVA es: " + a.sinIVA());
        System.out.println("El precio con IVA es: " + a.conIVA());
        System.out.println("Rebajado durante el mes de mayo un 20%");
        a.rebaja(20);
        System.out.println("Rebajado sin IVA es: " + a.sinIVA());
        System.out.println("Rebajado con IVA es: " + a.conIVA());
    }
}
```

Si una interfaz implementa otra, incluye todas sus constantes y declaraciones de métodos, aunque puede redefinir tanto constantes como métodos.

Nota: Es peligroso modificar una interfaz ya que las clases dependientes dejan de funcionar hasta que éstas implementen los nuevos métodos.

Una clase puede simultáneamente descender de otra clase e implementar una o varias interfaces. En este caso la seccion `implements` se coloca a continuación de `extends` en la cabecera de declaración de la clase. Por ejemplo:

```
public class ClaseDescendiente extends ClaseAscendiente implements Interfaz {
    ...
}
```

18.4. Utilización de una interfaz como un tipo de dato

Al declarar una interfaz, se declara un nuevo tipo de referencia. Pueden emplearse identificadores de interfaz en cualquier lugar donde se pueda utilizar el identificador de un tipo de dato (o de una clase). El objetivo es garantizar la sustituibilidad por cualquier instancia de una clase que la implemente. Por ejemplo, puede emplearse como tipo de un parámetro de un método:

```
public class Calculos {
    public void asignacion(Variaciones x); {
        . . .
    }
}
```

Sólo una instancia de una clase que implemente la interfaz puede asignarse al parámetro cuyo tipo corresponde al identificador de la interfaz. Esta facultad se puede aprovechar dentro la propia interfaz. Por ejemplo:

```
public interface Comparable {
   // La instancia que llama a esMayor (this) y el parametro otra
   // deben ser de la misma clase o de clases que implementen esta interfaz
   // La funcion devuelve 1, 0, -1 si this es mayor, igual o menor que otra
   public int esMayor(Comparable otra);
}
```

En algún caso puede ser útil declarar una interfaz vacía como, por ejemplo:

```
public interface Marcador {
}
```

Esta declaración es totalmente válida ya que no es obligatorio incluir dentro de una interfaz la declaración de una constante o la cabecera de un método. La utilidad de estas interfaces reside en la posibilidad de ser empleadas como tipos de dato para especificar clases sin necesidad de obligar a éstas a implementar algún método en concreto. Una interfaz no es una clase pero se considera un tipo en Java y puede ser utilizado como tal.