INTERFACES

¿Qué es y para qué sirve un interfaz?

Un interfaz es un mecanismo que permite definir un conjunto de métodos y constantes, que proporcionarán una forma de acceso común a todas aquellas clases que implementen dicho interfaz. Puede verse un interfaz como un **mecanismo estándar de acceso para un conjunto de clases**.

Se trata de declarar métodos abstractos y constantes que, posteriormente puedan ser implementados y usados de diferentes maneras según las necesidades de un programa.

Por ejemplo, el interfaz Comparable de la API de Java (puedes verlo en la documentación de Java) permite comparar objetos de forma estándar, sea cual sea la clase a la que pertenezcan, siempre y cuando dicha clase implemente **el interfaz Comparable**. En concreto, este interfaz solo contiene un método (*int compareTo(Object o)*). Ello nos asegura que, en cualquier objeto perteneciente a una de las clases que implementan el interfaz Comparable (por ejemplo, Integer, Date, Character, File, String, etc.), podemos invocar el método *compareTo* para compararlo con otro objeto.

¿Cómo se programa un interfaz?

La API de Java proporciona bastantes interfaces ya programados. Si para tu aplicación necesitas definir un nuevo interfaz, puedes declararlo utilizando la palabra clave interface. En el interior de la declaración, debes declarar todas las constantes y métodos que deseas que tenga el interfaz. No puedes implementarlos: **en el interfaz solo se declaran los métodos**. Son las clases que implementan el interfaz las responsables de programar estos métodos.

```
interface MiInterfaz {
  int CONSTANTE = 100;
  int metodoAbstracto( int parametro );
}
```

Se observa que las variables (atributos) adoptan la declaración en mayúsculas, pues en realidad actuarán como constantes *final*. En ningún caso estas variables actuarán como variables de instancia.

Por su parte, los métodos tras su declaración presentan un punto y coma, en lugar de su cuerpo entre llaves. Son métodos abstractos, por tanto, métodos sin implementación. **Ej:**

```
public interface Almacen {
         // constantes del interfaz
         public final static int CAPACIDAD_MINIMA= 10;

         // métodos del interfaz
         public void almacenar(Object dato, int identificador);
         public Object recuperar(int identificador);
      }
```

final static no es necesario ponerlo, por defecto son **public final static**.

En el ejemplo anterior se declara un interfaz para almacenar y recuperar objetos. El interfaz, como se puede ver, solo declara los métodos, y deben ser las clases que implementen el interfaz Almacen las que programen estos dos métodos. Además de los métodos, se declara la constante CAPACIDAD_MINIMA, que puede ser accedida mediante la expresión Almacen.CAPACIDAD_MINIMA.

¿Cómo se implementa un interfaz?

Una clase, para implementar un interfaz, debe indicarlo en su declaración, utilizando la **palabra clave implements**. Pero no es suficiente con indicar que implementa el interfaz, debe declarar y programar todos los métodos que indica el interfaz.

```
class ImplementaInterfaz implements MiInterfaz{
  int multiplicando=CONSTANTE;
  int metodoAbstracto( int parametro ){
    return ( parametro * multiplicando );
  }
}
```

En este ejemplo se observa que han de codificarse todos los métodos que determina la interfaz (m*etodoAbstracto()*), y la validez de las constantes (*CONSTANTE*) que define la interfaz durante toda la declaración de la clase.

Una interfaz no puede implementar otra interfaz, aunque sí extenderla (*extends*) ampliándola.

Veamos otro ejemplo para la interfaz Almacen. Clase que implementa el interfaz Almacen:

```
      public class Armario implements Almacen {
         // Declaración de atributos
         // Necesitamos una colección o array para guardar los objetos
         // Otros métodos: constructores, etc.

         // Otros métodos que no tienen nada que ver
         // Con el interfaz
         public void abrirPuertas() {
             // Código del método
         }
```

```
        // Métodos del interfaz Almacen
        public void almacenar(Object dato, int identificador) {
            // Código necesario para almacenar el dato
        }

        public Object recuperar(int identificador) {
            // Código necesario para recuperar el dato
        }
    }
```

Es muy importante recalcar que un interfaz obliga a aquellas clases (no abstractas) que indican que lo implementan a programar todos sus métodos. Sin embargo, la clase puede, libremente, programar cualquier otro método no especificado en el interfaz.

¿Si una clase implementa un interfaz, sus subclases lo heredan?

Dada una clase que implementa un interfaz, toda subclase de esta implementa implícitamente el mismo interfaz, sin necesidad de indicarlo con la palabra clave implements, y sin necesidad de volver a programar o declarar los métodos del interfaz.

¿Cómo se declara una referencia a un objeto que implementa un interfaz?

Se puede declarar una referencia a un objeto que implementa un interfaz, como en el caso general, del tipo de la clase a la cual pertenece el objeto. Por ejemplo, si Armario es una clase que implementa el interfaz Almacen:

```
    Armario miArmario = new Armario();
    miArmario.almacenar(new String("camisa blanca"), N_ID);
```

Pero hay situaciones en que no resulta recomendable la solución anterior. Por ejemplo, si deseamos que una misma referencia pueda apuntar a, según sea el caso, objetos de distintas clases (pero todas ellas implementando el mismo interfaz), podemos utilizar directamente el nombre del interfaz en la declaración. Por ejemplo, si tenemos la clase Caja, que también implementa Almacen:

```
    Almacen miContenedor= new Armario();
    miContenedor.almacenar(new String("camisa blanca"),56356);
    ...
    // supongamos que ahora no se necesita más este objeto
    //   de la clase Armario, y se necesita uno de la clase
    //   Caja, utilizando la misma referencia:
    miContenedor= new Caja();
    miContenedor.almacenar(new Integer(34), 334233);
```

Otro caso en que resulta útil es cuando necesitamos un array de objetos que implementan un interfaz, pero cada uno de ellos puede ser de distinta clase:

```
Almacen[] contenedores= new Almacen[3];
contenedores[0]= new Armario();
contenedores[1]= new Caja();
contenedores[2]= new Armario();
```

¿Cómo se declara un parámetro de un método para que este pueda ser un objeto de cualquier clase que implemente un determinado interfaz?

Si se declara el parámetro del método con el nombre del interfaz como tipo, entonces este parámetro podrá recibir un objeto de cualquier clase, siempre que esta implemente dicho interfaz. Por ejemplo, si las clases Armario y Caja implementan el interfaz Almacen:

```
public void guardarTodoEn(Almacen contenedor) {
   contenedor.almacenar (objeto1, 23);
   contenedor.almacenar (objeto2, 33);
}
...
// en otra parte de la clase
if (guardarEnCaja) guardarTodoEn(new Caja());
else guardarTodoEn(new Armario());
```

En el ejemplo, el método guardarTodoEn puede almacenar sus objetos en un objeto de cualquier clase que implemente el interfaz Almacen. Una restricción que tiene esto es que, de esta forma, solo se puede acceder directamente en el método guardarTodoEn, a aquellos métodos que declara el interfaz, pero no al resto de los métodos que posean las clases que lo implementan. Para poder acceder habría que realizar un casting al objeto apuntado.

Todo lo explicado anteriormente es igualmente aplicable al valor de retorno de un método.

Diferencias entre un interface y una clase abstracta

Un **interface** es simplemente una lista de métodos no implementados, además puede incluir la declaración de constantes. Una **clase abstracta** puede incluir métodos implementados y no implementados o abstractos, miembros dato constantes y otros no constantes.

Un interface es parecido a una clase abstracta en Java, pero con las siguientes diferencias:

- Todo método es abstracto y público sin necesidad de declararlo. Por lo tanto, un interface en Java no implementa ninguno de los métodos que declara.

- Un interface se implementa (implements) no se extiende (extends) por sus subclases.

- Una clase puede implementar más de un interfaz en Java, pero solo puede extender una clase. Es lo más parecido que tiene Java a la herencia múltiple, que de clases normales está prohibida.

- Podemos declarar variables del tipo de clase del interfaz, pero para inicializarlas tendremos que hacerlo con objetos de una clase que lo implemente.

Así, por ejemplo, podemos declarar el siguiente interfaz en Java:

```
public interface Figura {
        int area();
}
```

Y una clase que lo implementa:

```
public class Cuadrado implements Figura {
        int lado;
        public Cuadrado (int ladoParametro) {
                lado = ladoParametro;
        }
        public int area(){
                return lado*lado;
        }
}
```

Más adelante podemos:

```
public class PruebaInterfaz{
        public static void main(String args[]){
          Figura figura=new Cuadrado (5);
          // Podemos crear una referencia de interface(variable figura) y que un objeto
          // que   pertenezca a una clase que la implementa le sea asignada a la variable
          System.out.println(figura.area());
        }
}
```

Una clase solamente puede derivar **extends** de una clase base, pero puede implementar varios interfaces. Los nombres de los interfaces se colocan separados por una coma después de la palabra reservada **implements**. El lenguaje Java no fuerza, por tanto, una relación jerárquica, simplemente permite que **clases no relacionadas** puedan tener algunas características de su comportamiento similares.