9. INTERFACES

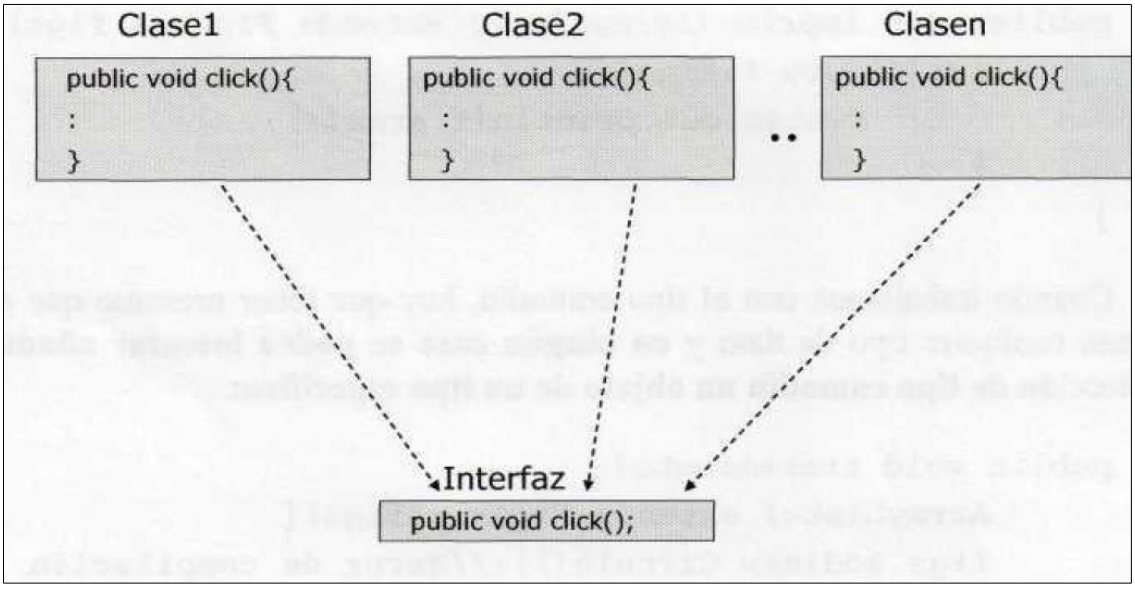
9.1. Definición de interfaz

Estrictamente hablando, una interfaz es un conjunto de métodos abstractos y de constantes públicas definidos en un archivo .java. Una interfaz es similar a una clase abstracta llevada al límite, en la que todos sus métodos son abstractos.

La finalidad de una interfaz es la de definir el formato que deben de tener determinados métodos que han de implementar ciertas clases.

Por ejemplo, para gestionar eventos en una aplicación basada en entorno gráfico, las clases donde se capturan estos eventos deben codificar una serie de métodos que se ejecutarán al producirse estos eventos. Cada tipo de evento tendrá su propio método de respuesta, cuyo formato estará definido en una interfaz. Así, aquellas clases que deseen responder a un determinado evento, deberán implementar el método de respuesta de acuerdo al formato definido en la interfaz.

Hay que insistir en el hecho de que una interfaz no establece lo que un método tiene que hacer y cómo hacerlo, sino el formato (nombre, parámetros y tipo de devolución) que éste debe tener.



Adherencia a una interfaz

9.2. Declaración de una interfaz

Una interfaz se define mediante la palabra **`interface`**, utilizando la siguiente sintaxis:

```
[public] interface NombreInterfaz {
      // Constantes
      tipo constante1 = valor1;
      tipo constante2 = valor2;
            . . .

      // Métodos abstractos
      tipo metodo1 (argumentos);
      tipo metodo2 (argumentos);
            .
            .
            .
}
```

Por ejemplo,

```
public interface Operaciones {
      void rotar();
      String serializar();
}
```

Al igual que las clases, las interfaces se definen en archivos .java y, como sucede con aquéllas, si la interfaz utiliza el modificador de acceso public, el nombre de la interfaz deberá coincidir con el del fichero .java donde se almacena. Como resultado de la compilación de una interfaz, se genera un archivo .class.

A la hora de crear una interfaz hay que tener en cuenta las siguientes consideraciones:

- **Todos los métodos definidos en una interfaz son públicos y abstractos,** aunque no se indique explícitamente. El uso de los modificadores `abstract` y `public` en la definición de los métodos de la interfaz es, por tanto, redundante si bien su uso no provocará ningún tipo de error.

- **En una interfaz es posible definir constantes.** Además de métodos, las interfaces pueden contener constantes, las cuales son, implícitamente, públicas y estáticas. De hecho, tanto los modificadores `public` como `static` como `final`, se pueden omitir en la definición de constantes dentro de una interfaz. Los siguientes son ejemplos válidos de definición de constantes en el interior de una interfaz:

```
int k = 23;
public String s = “hj”;
public static final double p = 4.5;
Object o = new Object();
```

- **Una interfaz no es una clase.** Las interfaces tan sólo pueden contener lo que ya se ha comentado: métodos abstractos y constantes. No pueden contener métodos con código, constructores o variables y, por supuesto, no es posible crear objetos de una interfaz.

9.3. Implementación de una interfaz

Como ya se ha dicho antes, el objetivo de las interfaces es proporcionar un formato común de métodos para las clases. Para forzar a que una clase defina el código para los métodos declarados en una determinada interfaz, la clase deberá *implementar* la interfaz.

En la definición de una clase, se utiliza la palabra **`implements`** para indicar qué interfaz se ha de implementar:

```
public class MiClase implements MiInterfaz { ... }
```

Por ejemplo,

```
public class Triangulo implements Operaciones {
      public void rotar() {
            // implementación del método
      }
      public String Serializar() {
            //implementación de método
      }
}
```

Sobre la implementación de interfaces, se ha de tener en cuenta lo siguiente:

- Al igual que sucede al heredar una clase abstracta, **cuando una clase implementa una interfaz, está obligada a definir el código (implementar) de todos los métodos existentes en la misma.** De no ser así, la clase deberá ser declarada como abstracta.

- **Una clase puede implementar más de una interfaz,** en cuyo caso, deberá implementar los métodos existentes en todas las interfaces. El formato utilizado en la definición de la clase será:

  public class MiClase implements Interfaz1, Interfaz2,… { … }

- **Una clase puede heredar otra clase e implementar al mismo tiempo una o varias interfaces.** En este sentido, las interfaces proporcionan una gran flexibilidad respecto a las clases abstractas a la hora de "forzar" a una clase a implementar ciertos métodos, ya que el implementar una interfaz no impide que la clase pueda heredar las características y capacidades de otras clases. Por ejemplo, si la clase `Triangulo` necesitara tener los métodos `rotar()` y `serializar()`, podría implementar la interfaz `Operaciones` y seguir heredando la clase `Figura`.

  Al mismo tiempo, el hecho de aislar esos métodos en la interfaz y no incluirlos en la clase `Figura`, permite que otras clases que no sean figuras puedan implementar esos métodos sin necesidad de heredar a ésta, mientras que el resto de clases que heredan `Figura` (`Rectangulo`, `Circulo`, etc.) si no necesitaran disponer de la capacidad de rotar y serializar, no se verían en la obligación de proporcionar dichos métodos.

  La sintaxis utilizada para heredar de una clase e implementar interfaces es:

  *public class MiClase **extends** Superclase **implements** Interfaz1, Interfaz2 { … }*

- **Una interfaz puede heredar otras interfaces.** No se trata realmente de un caso de herencia como tal, pues lo único que "adquiere" la subinterfaz es el conjunto de métodos abstractos existentes en la superinterfaz. La sintaxis utilizada es la siguiente:

  *public interface MiInterfaz **extends** Interfaz1, Interfaz2 { . . . }*

9.4. Interfaces y polimorfismo

Como ya ocurriera con las clases abstractas, el principal objetivo que persiguen las interfaces con la definición de un formato común de métodos es el polimorfismo.

Una **variable de tipo interfaz puede almacenar cualquier objeto de las clases que la implementan,** pudiendo utilizar esta variable para invocar a los métodos del objeto que han sido declarados en la interfaz e implementados en la clase:

```
Operaciones op = new Triangulo();
op.rotar();
op.serializar();
```

Esta capacidad, unida a su flexibilidad, hacen de las interfaces una estructura de programación tremendamente útil.

Las interfaces son muy útiles para asignar comportamientos comunes a clases que no están relacionadas, ya que sirven para forzar a las clases que las implementan a tener un comportamiento a seguir.

El operador instanceof también funciona con variables de tipo interfaz de forma análoga a como lo hace con las clases.

Al igual que se habla de superclases y subclases, se puede emplear la terminología superinterfaces y subinterfaces. A diferencia de las clases, un interfaz puede extender simultáneamente a varios superinterfaces, lo que supone una aproximación a la posibilidad de realizar herencia múltiple.

Aunque las interfaces tienen cierto parecido con las clases abstractas, hay que tener en cuenta que una clase abstracta puede contener métodos no abstractos (implementados) y además las clases abstractas no soportan la herencia múltiple. Por otro lado, las interfaces no pueden contener métodos no abstractos y además soportan la herencia múltiple.

9.5. Interfaces en el Java SE

Además de clases, los paquetes de Java estándar incluyen numerosas interfaces, algunas de ellas son implementadas por las propias clases del Java SE y otras están diseñadas para ser implementadas en las aplicaciones.

Como muestra, comentamos algunas de las más importantes:

- **java.lang.Runnable,** Contiene un método para ser implementado por aquellas aplicaciones que van a funcionar en modo multitarea.
- **java.util.Enumeratíon.** La utilizamos en el apartado dedicado a las colecciones. Proporciona métodos que son implementados por objetos utilizados para recorrer colecciones.
- **java.awt.event.WindowListener.** Proporciona métodos que deben ser implementados por las clases que van a gestionar los eventos (clases manejadoras) producidos en la ventana, dentro de una aplicación basada en entorno gráfico. Además de esta interfaz, hay otras muchas más para la gestión de otros tipos de eventos en los diversos controles gráficos Java.
- **java.sqL.Connection**. Interfaz implementada por los objetos utilizados para manejar conexiones con bases de datos. Además de ésta, el paquete java.sql contiene otras interfaces relacionadas con el envío de consultas SQL y la manipulación de resultados, como es el caso de Statement o ResultSet.
- **java.io.Serializable**. Esta interfaz no contiene ningún método que deba ser definido por las clases que la implementan, sin embargo, la JVM requiere que dicha interfaz deba ser implementada por aquellas clases cuyos objetos tengan que ser transferidos a algún dispositivo de almacenamiento, como por ejemplo un archivo de disco.

9.6. Ejemplos:

Ejemplo de interfaz que solo contiene **constantes**:

```
public interface DatosCentroDeEstudios {
      byte numeroDePisos = 5;
      byte numeroDeAulas = 25;
      byte numeroDeDespachos = 10;
}
```

Ejemplo de interfaz que solo contiene **métodos**:

```
public interface CalculosCentroDeEstudios {
      short numeroDeAprobados(float[] notas);
      short numeroDeSuspensos(float[] notas);
      float notaMedia(float[] notas);
      float desviacion(float[] notas);
}
```

Ejemplo de interfaz que contiene **constantes** y **métodos**:

```
public interface CentroDeEstudios {
      // Constantes
      byte numeroDePisos = 5;
      byte numeroDeAulas = 25;
      byte numeroDeDespachos = 10;

      // Métodos
      short numeroDeAprobados(float[] notas);
      short numeroDeSuspensos(float[] notas);
      float notaMedia(float[] notas);
      float desviacion(float[] notas);
}
```

También se puede definir un interfaz basado en otro. Utilizando la palabra reservada `extends`, tal y como se hace en la herencia de clases:

```
public interface CentroDeEstudiosHeredero extends
CalculosCentroDeEstudios {
           byte numeroDePisos = 5;
           byte numeroDeAulas = 25;
           byte numeroDeDespachos = 10;
}
```

Al igual que se habla de superclases y subclases, se puede emplear la terminología superinterfaces y subinterfaces. A diferencia de las clases, un interfaz puede heredar simultáneamente de varias superinterfaces, lo que supone una aproximación a la posibilidad de realizar **herencia múltiple**:

```
public interface CentroDeEstudiosHerederoMultiple extends
           DatosCentroDeEstudios, CalculosCentroDeEstudios {

}
```

Para poder utilizar los miembros de un interfaz es necesario implementarlo en una clase; se emplea la palabra reservada `implements` para indicarlo:

```
public class ClaseCentroDeEstudios implements CentroDeEstudios {
	@Override
	public short numeroDeAprobados(float[] notas) {
		short numAprobados = 0;

		for (int i = 0; i < notas.length; i++)
			if (notas[i] >= 5)
				numAprobados++;
		return numAprobados;
	}

	@Override
	public short numeroDeSuspensos(float[] notas) {
//		short numSuspensos = 0;
//
//		for (int i = 0; i < notas.length; i++) {
//			if (notas[i] < 5) {
//				numSuspensos++;
//			}
//		}
//		return numSuspensos;

		// También se podría implementar como
		return (short) (notas.length -
				this.numeroDeAprobados(notas));
	}

	@Override
	public float notaMedia(float[] notas) {
		float suma = 0;

		for (int i = 0; i < notas.length; i++) {
			suma += notas[i];
		}
		return suma / (float) notas.length;
	}

	@Override
	public float desviacion(float[] notas) {
		float media = this.notaMedia(notas);
		float suma = 0;

		for (int i = 0; i < notas.length; i++) {
			suma += Math.abs(media - notas[i]);
		}
		return suma / (float) notas.length;
	}
}
```

Para probar la clase y ver cómo funcionan las interfaces, se crea una nueva clase `Pruebas`:

```
public class Pruebas {
      public static void main(String[] args) {
           float[] notas = { 3, 6, 8, 10, 2, 5, 7, 9, 10 };
           ClaseCentroDeEstudios grupo = new ClaseCentroDeEstudios();
           System.out.println("Aprobados: " +
                                grupo.numeroDeAprobados(notas));
           System.out.println("Suspensos: " +
                                grupo.numeroDeSuspensos(notas));
           System.out.println("Media: " + grupo.notaMedia(notas));
           System.out.println("Desviación: " +
                                grupo.desviacion(notas));
      }
}
```

Al ejecutar el programa se muestran los resultados calculados. Si ahora se redefine la Clase `ClaseCentroDeEstudios` para que, en vez de implementar la interfaz `CentroDeEstudios`, implemente la interfaz `CentroDeEstudiosHeredero` o implemente la interfaz `CentroDeEstudiosHerederoMultiple` o incluse implemente las dos interfaces `DatosCentroDeEstudios` y `CalculosCentroDeEstudios`, se puede comprobar que los resultados van a ser los mismos, pues todas estas interfaces (o combinación de interfaces) están representando lo mismo, simplemente son distintas formas de definirlas.