Programación de Aplicaciones. Introducción al diseño basado en componentes

Anexo I Interfaces Java

1. Definición de una interfaz en Java

La definición de una interfaz en Java se realiza de la siguiente forma:

```
[public ] interface nombre_interfaz {

<cuerpo de la interfaz>

}
```

Donde ***<cuerpo de la interfaz>*** es igual al conjunto de métodos abstractos que define la interfaz. **<cuerpo de interfaz>** puede ser solamente declaraciones de métodos sin cuerpo y constantes .

El modificador **public** indica que la interfaz puede ser utilizada por cualquier clase de cualquier otro paquete. Si no se indica public la interfaz sólo estará disponible para las clases del mismo paquete

Ejemplo de declaración de una interfaz denominada IActivación con un único método abstracto

```
public interface IActivacion{

void activar();

}
```

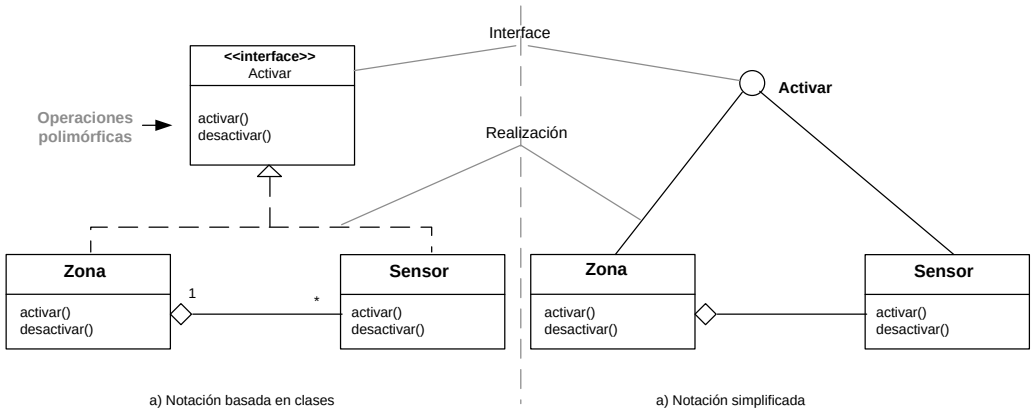
Una **interfaz puede heredar o extender otra interfaz**. Para ello se usa la siguiente sintaxis:

```
[public ] interface nombre_interfaz extends nombre_interfaz_base {

<cuerpo de la interfaz>

}
```

Nota importante: El nombre de la interfaz se puede usar en cualquier lugar donde se pueda usar el nombre de una clase.

Las interfaces al igual que las clases y métodos abstractos proporcionan plantillas de comportamiento que se espera que sea implementada por otras clases. Una interfaz recordemos no implementa métodos, sólo los define (también puede definir constantes).

Por ejemplo para este diagrama de clases la interfaz Activar proporciona un forma de activar y desactivar cualquier dispositivo



2. Implementar una interfaz Java

Para que una clase implemente una interfaz definida, hay que usar la palabra reservada **implements** como se muestra a continuación:

```
[public ] class nombre_clase  implements  nombre_interfaz {

<cuerpo de la clase>

}
```

Nota : La palabra reservada **implements** sigue a **extends** si esta existiera

En el ejemplo anterior

```
class Zona implements Activar{

public Zona(){//constructor}

void activar() {

//Implementación

}

void desactivar(){

//Implementación

}

 }

class Sensor implements Activar{

public Sensor(){//constructor}

void activar() {

//Implementación

}
```

```
void desactivar(){
//Implementación
}
}
```

Una **clase puede implementar varias interfaces**

Ejemplo

```
class Identificacion implements OperacionesVerificaciónFirma,
OperacionesCifrado
```

Como una interfaz sólo aporta declaraciones de métodos abstractos es obligado implementarlos. No podemos elegir y definir sólo aquellos métodos que necesitemos, en el momento de definir la clase de esta forma, se obligará a que la clase implemente todos y cada uno de los métodos definidos por la interfaz, si no es así el programa no compilará.

Si una clase implementa una interfaz y posteriormente se crean subclases, **las subclases heredarán todos los nuevos métodos que haya implementado la clase base.**

El tipo de un objeto nunca es un interfaz. Sin embargo una variable puede ser de tipo interfaz. La variable contendrá una referencia a un objeto que implementa la interfaz

3. Clases abstractas e interfaces

¿Sería equivalente un diseño con una clase abstracta?. La respuesta es no. Si se define una clase abstracta una subclase podría heredar métodos públicos de la clase base, asociaciones y atributos. Sin embargo una clase que implementa una interfaz debe sobrescribir métodos abstractos y no hereda nada (métodos públicos, relaciones etc). Hay que recordar que una interfaz no define asociaciones.

Aunque ambos conceptos tienen algo en común y es que permiten definir una especie de contrato. Una clase abstracta impone la restricción de que toda subclase debe implementar las operaciones abstractas, en este sentido si son iguales. Por otro lado una clase abstracta forma un concepto general que da sentido a una relación jerárquica. Sine embargo una interfaz no tiene esa finalidad

4. Utilizar una interfaz como un tipo de dato

Una interfaz es un nuevo tipo de dato, por lo tanto el nombre de una interfaz puede aparecer en cualquier sitio donde aparezca un tipo de datos.

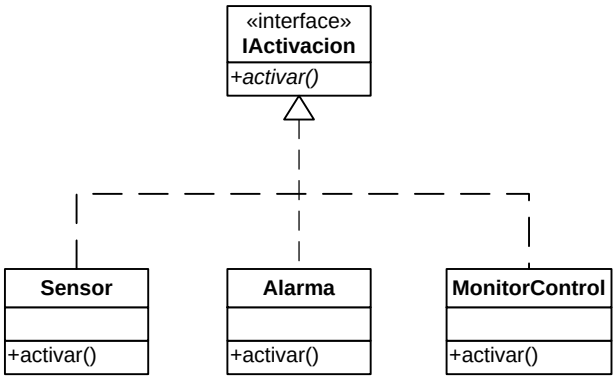
```
public class Controlador{
IActivacion dispositivos[];
public void registrarDispositivo (IActivacion disp){…}
```

```
}
```

Una variable de tipo interfaz espera referenciar a un objeto que tenga implementada dicha interfaz, de lo contrario el compilador de java mostrará un error.

Así mismo se puede convertir implícitamente referencias a objetos que implementan una interfaz en referencias a esa interfaz y viceversa, pero en este caso la conversión debe ser explicita.

El siguiente ejemplo muestra tres clases no relacionadas que por el hecho de implementar una interfaz pueden formar parte de la misma colección y aplicar la definición del polimorfismo para tratarlas de forma homogenea




```
//…

IActivacion dispositivos[];

dipositivos[0]= new Sensor();

dispositivos[1]= new Alarma();

dispositivos[2]= new MonitorControl();

dispositivos[0].activar();

//…
```

5. Interfaces y herencia múltiple

A menudo se piensa en los interfaces como una alternativa a la herencia múltiple. Pero la realidad es que ambos conceptos, **herencia e interfaces son bastantes diferentes, más concretamente las diferencias son**:

- Desde la interfaz una clase sólo puede heredar constantes
- Desde la interfaz una clase no puede heredar métodos definidos
- La jerarquía de interfaces es independiente a la jerarquía de clases. De hecho varias clases pueden implementar la misma interfaz y no pertenecer a la misma jerarquía de

clases. En cambio cuando se hable de herencia múltiple todas las clases pertenecen a la misma jerarquía.

C++ si permite herencia múltiple pero Java y C# no la permiten aunque en la práctica no llega a ser un problema real.

6. Para que sirve una interfaz

Una interfaz se utiliza para definir **un comportamiento abstracto**, un pseudocontrato entre una clase que va a usar la interfaz y una clase que deberá dar una implementación a cada uno de los métodos definidos en la interfaz. Resumimos a continuación algunas de las situaciones donde las interfaces pueden ser útiles.

- Captar comportamiento similar entre clases no relacionadas sin forzar entre ellas una relación artificial. Una acción de este tipo permite tratar a este conjunto de clase como un conjunto de objetos homogéneo, usando el polimorfismo.
- Declarar un conjunto de operaciones que varias clases deben implementar en determinadas situaciones (Por ejemplo en Java toda clase que debe ejecutarse como un hilo concurrente debe implementar la interfaz Runnable)
- Desacoplar un clase cliente de la clase que implementa la interfaz
- Permite definir un contrato sin preocuparnos por los detalles de la implementación