Repaso de interfaces

1 Interfaces. Clases abstractas e interfaces.

Una interfaz y una clase abstracta son muy parecidas. La diferencia está en que **en una interfaz, todos los métodos tienen que ser abstractos**, es decir, estarán formados por las cabeceras de diferentes métodos (prototipos).

Con lo cual, una **interfaz**, al igual que las clases abstractas, sirve para definir el comportamiento **mínimo que tiene que tener una clase**.

Las **interfaces se declaran** usando la palabra clave `interface` en vez de `class`.

```
public interface Obligatorio {
    public void pedirTodosLosDatos();
    public void visualTodosLosDatos();
}
```

Las interfaces no son heredadas por una clase, sino que **una clase implementa una interfaz**. Esto se indica de esta forma:

```
class NombreClase implements Nombrelnterface
```

Al poner esto, se está obligando a que la clase "`NombreClase`" implemente todos los métodos que tenga dicha interfaz.

Por ejemplo:

Si una clase implementa la interfaz `Obligatorio` (definida anteriormente), estará obligada a implementar los métodos: `pedirTodosLosDatos()` y `visualTodosLosDatos()`.

```
class MiClase implements Obligatorio {
    void pedirTodosLosDatos() {
      <código>
    }
    void visualTodosLosDatos(){
      <código>
    }
}
```

Las interfaces siguen los principios de la herencia múltiple, ya que permiten que una clase pueda implementar varias interfaces. Es decir, podemos escribir esto:

```
Class NombreClase implements Nombrelnterface1,
      Nombrelnterfae2, Nombrelnterface3,...
```

Con esto, estamos obligando a que la clase "`NombreClase`" implemente todos los métodos que tengan todas las interfaces indicadas.

Las propiedades de las interfaces son:

- Una interfaz no puede tener atributos salvo que sean estáticos (*`static`*) y constantes (*`final`*). Además, **en una interfaz los atributos deben ser públicos e inicializados con un valor en el momento de la declaración**.
- A los métodos de una interfaz no se les puede poner los modificadores **`private`** ni **`protected`**. Implícitamente, son **`public`**.
- Una interfaz **no puede tener constructor**.
- A los métodos de una interfaz no hace falta ponerles el modificador **`abstract`**; porque implícitamente lo son.
- Dentro de la clase que implementa una interfaz, todos los métodos que sobrescribe de dicha interfaz tienen que ser públicos.
- **Se puede declarar un objeto con el nombre de una interfaz**, pero **no se puede instanciar indicando el nombre de la interfaz**. De este modo, si se define una atributo/variable cuyo tipo es una interfaz, **se le puede asignar un objeto que sea una instancia de una clase que implementa la interfaz**. Esto significa que, utilizando interfaces como tipos, se puede aplicar el mecanismo de polimorfismo a clases que no están relacionadas por el mecanismo de herencia, pero sí por el mecanismo de implementación de una interfaz.
- Una interfaz **puede heredar de otra interfaz**. En Java y C#, debido a la restricción de la herencia simple, solo podrá heredar de una interfaz:

```
public interface Interfaz2 extends Interfaz1 {
    //Firmas de los métodos
}
```

Si una clase implementa una interface, puede suceder:

a) Que implemente los métodos de la interface sobrescribiéndolos (puede ser una clase concreta).
b) Que no implemente los métodos de la interface: obligatoriamente será una clase abstracta y, por tanto, obligatoriamente ha de llevar la palabra clave *abstract* en su encabezado.

2 Referencias de interfaz

Se pueden declarar variables como referencias a objetos que usan una interfaz en lugar de un tipo de clase. Cualquier instancia de cualquier clase que implemente la interfaz declarada puede ser referida por dichas variables. Cuando se llama a un método a través de una de estas referencias, la versión correcta se llamará en función de la clase de objeto al que está apuntando la variable (esto es el **polimorfismo con**

interfaces). Esta es una de las características clave de las interfaces. El método que se ejecutará se busca dinámicamente en tiempo de ejecución, permitiendo que las clases sean creadas más tarde que el código que llama a los métodos de dichas clases. El código de llamada a un método puede enviarse a través de una interfaz sin tener que saber nada (excepto la firma) sobre el código "Llamado".

El polimorfismo con interfaces sólo hay que usarlo cuando sea necesario ya que requiere un consumo importante de recursos del sistema.

Ejemplo:

```
public interface Hablar {
	public void hablar();
}
public class Persona implements Hablar {
	private String nombre;
	public Persona() {
		nombre = "Ana";
	}
	public void hablar(){
		System.out.println("Hola me llamo " + nombre);
	}
}
public class Perro implements Hablar {
	private String nombre;
	public Perro() {
		nombre = "Atila";
	}

	public void hablar(){
		System.out.println("Guau, guau");
	}
}
public class PruebaInterface {
	public static void main(String[] args) {
		Persona persona = new Persona();
		Perro perro = new Perro();
		hacerHablar(persona);
		hacerHablar(perro);
	}//main
	public static void hacerHablar(Hablar v){
		v.hablar();
	}

}
```

En la ejecución veremos:

```
Hola me llamo Ana
Guau, guau
```