INSTITUTO POLITÉCNICO NACIONAL

Escuela Superior de Cómputo

Unidad de aprendizaje

"Programación Orientada a Objetos"

Grupo:

2CM1

Profesor:

Daniel Cruz García

Tarea 8:

"Clases abstractas e interfaces"

Alumna:

Luciano Espina Melisa

Fecha de entrega:

04/Mayo/2018

Clases abstractas

Una clase que declara la existencia de métodos pero no la implementación de dichos métodos (o sea, las llaves { } y las sentencias entre ellas), se considera una clase abstracta.

Una clase abstracta puede contener métodos no-abstractos pero al menos uno de los métodos debe ser declarado abstracto.

Para declarar una clase o un metodo como abstractos, se utiliza la palabra reservada **abstract**.

```
abstract class Drawing
{
abstract void miMetodo(int var1, int var2);
String miOtroMetodo( ){ ... }
}
```

Una clase abstracta no se puede instanciar, pero si se puede heredar y las clases hijas serán las encargadas de agregar la funcionalidad a los métodos abstractos. Si no lo hacen así, las clases hijas deben ser también abstractas. [1]

Declarar una clase abstracta es distinto a tener una clase de la que no se crean objetos. En una clase abstracta, no existe la posibilidad. En una clase normal, existe la posibilidad de crearlos, aunque no lo hagamos. El hecho de que no creemos instancias de una clase no es suficiente para que Java considere que una clase es abstracta. Para lograr esto hemos de declarar explícitamente la clase como abstracta mediante la sintaxis que hemos indicado. Si una clase no se declara usando abstract se cataloga como “clase concreta”. En inglés abstract significa “resumen”, por eso en algunos textos en castellano a las clases abstractas se les llama resúmenes. Una clase abstracta para Java es una clase de la que nunca se van a crear instancias: simplemente va a servir como superclase a otras clases.

A su vez, las clases abstractas suelen contener métodos abstractos: la situación es la misma. Para que un método se considere abstracto ha de incluir en su signatura la palabra clave abstract. Además, un método abstracto tiene estas peculiaridades:

- No tiene cuerpo (llaves): sólo consta de signatura con paréntesis.
- Su signatura termina con un punto y coma.
- Sólo puede existir dentro de una clase abstracta. De esta forma se evita que haya métodos que no se puedan ejecutar dentro de clases concretas. Visto de otra manera, si una clase incluye un método abstracto, forzosamente la clase será una clase abstracta.
- Los métodos abstractos forzosamente habrán de estar sobreescritos en las subclases. Si una subclase no implementa un método abstracto de la superclase tiene un método no ejecutable, lo que la fuerza a ser una subclase abstracta. Para que la subclase sea concreta habrá de implementar métodos sobreescritos para todos los métodos abstractos de sus superclases [2]

Ejemplo de clases abstractas

```
1  public abstract class Principal{
2
3      /**Método concreto con implementación*/
4      public void metodoConcreto(){
5          ..............
6          ..............
7      }
8
9      /**Método Abstracto sin implementación*/
10     public abstract void metodoAbstracto();
11
12 }
13
14 class subClase extends Principal{
15
16  @Override
17  public void metodoAbstracto() {
18   /**Implementación definida por la clase concreta*/
19  }
20
21 }
```

[3]

Interfaz

Un interfaz es una lista de acciones que puede llevar a cabo un determinado objeto. Sorpresa, ¿eso no eran los métodos que se definen en una clase? Casi, en una clase además de aparecer los métodos aparecía el código para dichos métodos, en cambio en un interfaz sólo existe el prototipo de una función, no su código.

Veámoslo con un ejemplo: Pensemos en un interfaz que en su lista de métodos aparezcan los métodos despegar, aterrizar, servirComida y volar. Todos pensamos en un avión, ¿verdad? El motivo es sencillamente que avión es el concepto que engloba las acciones que hemos detallado antes, a pesar de que, existan muchos objetos avión diferentes entre sí, por ejemplo, Boeing 747, Boeing 737, MacDonell-Douglas.

Lo realmente interesante es que todos ellos, a pesar de pertenecer a clases distintas, poseen el interfaz avión, es decir poseen los métodos detallados en la lista del interfaz avión.

Esto significa también que a cualquier avión le podemos pedir que vuele, sin importarnos a que clase real pertenezca el avión, evidentemente cada clase especificará como volará el avión (porque proporciona el código de la función volar).

En java un interfaz define la lista de métodos, pero para que una clase posea un interfaz hay que indicar explícitamente que lo implementa mediante la claúsula implements. Pero veamos primero la estructura de un interfaz:

```
 [modif.visibilidad] interface nombreInterfaz [extends listaInterfaces]
{
        prototipo método1;

..
        prototipo método1;
}
```

Donde modif.visibilidad puede ser public o bien sin especificar, es decir visibilidad pública (desde cualquier clase se puede emplear el interfaz) o de paquete (sólo se puede emplear desde clases del mismo paquete).

nombreInterfaz por convenio sigue las mismas reglas de nomenclatura que las clases, y en muchos casos acaba en able (que podíamos traducir como: ser capaz de).

La claúsula opcional extends se emplea para conseguir que un interfaz hereda las funciones de otro/s interfaces, simplemente listaInterfaces es una lista separaada por coma de interfaces de los que se desea heredar.

En muchas ocasiones un interfaz es empleado para definir un comportamiento, que posteriormente será implementado por diversas clases, que podrían no tener nada que ver

entre ellas, pero que todas se comportarán igual de cara al interfaz. Es decir, todas tendrán las funciones indicadas por el interfaz.

Cuando varios objetos de distintas clases pueden responder al mismo mensaje (función), aun realizando cosas distintas se denomina polimorfismo. [4]

Ejemplo de interfaces

Vamos a definir el interfaz Cantante, un interfaz muy simple que sólo posee un método: cantar.

Crear el fichero Cantante.java

Agregar el siguiente código:

```
public interface Cantante
{
        public void cantar();
}
```

Cojamos la clase Persona y hagamos que implemente el interfaz Cantante:

```
public class Persona implements Cantante
```

Además agreguemos el código para el método que define el interfaz cantante:

```
public void cantar()
{
    System.out.println("La laa la raa laaa!");
}
```

Construyamos ahora una clase con función main (ArranqueInterfaz.java)para ejecutar:

```
public class ArranqueInterfaz
{
        public static void main(String arg[])
        {
```

```
        Persona p = new Persona();
        hacerCantar(p);
    }
    public static void hacerCantar(Cantante c)
    {
        c.cantar();
    }
}
```

Podemos ver que construimos un objeto (p) de tipo persona y se lo pasamos a la función hacerCantar. Esta función espera recibir un objeto Cantante, y una persona lo es, por tanto la recibe y llama al método cantar del objeto recibido.

Probemos a intentar pasar a la función hacerCantar en lugar del objeto Persona (p) un objeto String (texto), resultado: error de compilación.

Contruyamos ahora la clase Canario (Canario.java), pensando que también sabe cantar:

```
public class Canario implements Cantante
{
    private int peso;
    /* Aqui vendrían el resto de atributos y funciones propias de un canario */


    public void cantar()
    {
        System.out.println("Pio Pio Pio");
    }
}
```

Y ahora agreguemos en la clase ArranqueInterfaz el siguiente código, para crear un objeto canario y pasarselo a la función hacerCantar:

```
Canario c = new Canario();
```

```
hacerCantar(c);
```

Tras ejecutar comprobaremos que podemos pasar tanto una Persona como un Canario a la función hacerCantar, de tal manera que dentro de dicha función sólo accedamos a las funciones del interfaz y no habrá problemas. Por ejemplo, si pusieramos:

```
c.SetNombre("Luis")
```

dentro de la función hacerPersona, podría funcionar si pasasemos un objeto Persona, pero no si pasamos uno de tipo Canario. [4]

Bibliografías

[1] S/A IV - Clases abstractas | Profesores.fi-b.unam.mx [Online] Available: http://profesores.fi-b.unam.mx/carlos/java/java_basico4_8.html

[2] S/A Aprendeaprogramar.com [Online] Available: https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=668:clases-y-metodos-abstractos-en-java-abstract-class-clases-del-api-ejemplos-codigo-y-ejercicios-cu00695b&catid=68&Itemid=188

[3] Cristian Heano | Clases abstractas | Codejavu.blogspot.mx [Online] Available: http://codejavu.blogspot.mx/2013/05/clases-abstractas.html

[4] 4.5 Interfaces | Curso de Introducci&#65533;n a Java | Mundojava.net [Online] Available: http://www.mundojava.net/interfaces.html?Pg=java_inicial_4_5.html