



Clases y métodos finales. Clases abstractas

Clases y métodos finales

En ocasiones es necesario evitar que alguien construya una subclase a partir de una de nuestras clases. Las clases que no se pueden heredar se llaman clases finales y se utiliza el modificador final en la definición de la clase para indicarlo.

Por ejemplo, imaginemos que queremos evitar que otros extiendan la clase Jefes. Bastaría con definir la clase de la siguiente manera: **final class Jefes {...**

Todos los métodos declarados en una clase final, son final automáticamente.

Los campos también pueden ser declarados como final. Si lo hacemos, estos no podrán ser modificados una vez construido el objeto, pero hay que tener en cuenta un detalle importante: cuando declaramos una clase como final, los métodos automáticamente pasan a ser final, pero no los campos.

¿Por qué declarar una clase o un método/s como final? En realidad, solo hay una buena razón para ello y es evitar que su semántica no se pueda modificar en ninguna subclase.

Clases abstractas

A medida que ascendemos por la jerarquía de la herencia, las clases se van volviendo más generales, más abstractas. Llega un momento según ascendemos por la jerarquía en el que la clase antecesora es tan general que pensamos en ella como base de otras clases más que como una clase con objetos concretos.

Consideremos un ejemplo: imaginemos que tenemos una clase Persona de la cual descienden dos clases más, Empleados y Alumnos. Un Empleado entonces es una Persona y también lo es un Alumno.

Hay ciertos atributos que tienen sentido para todas las personas, como por ejemplo el nombre de las mismas. Todos los alumnos tienen nombre y también los empleados. Al introducir una clase superior se nos permite en la misma implementar un método que nos de el nombre, por ejemplo el método `getNombre()`.

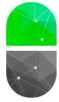
Sin embargo al querer por ejemplo implementar un método que nos de una descripción breve de la persona, resultaría sencillo implantarlo para las clases de Alumno y Empleado, pero ¿qué información de la descripción podríamos ofrecer de la clase Persona.

Hay una forma de hacer esto muy cómoda, para no tener que implementar el método de forma alguna que es utilizar la palabra reservada **abstract**.


Ejemplo:

```
public abstract String getDescripcion(); // no se requiere una implementación.
```

Toda clase que tenga uno o más métodos **abstract** ha de declararse también como **abstract**.

**Ejemplo:**

```
abstract class Personas {  
    public Personas(String nom) {  
        nombre=nom;  
    }  
    public String getNombre() {  
        return nombre;  
    }  
    public abstract String getDescripcion();  
    private String nombre;  
}
```



Los métodos abstractos actúan como reservas de espacio para los métodos que se implementan en las subclases. Cuando se extiende una clase abstracta hay dos opciones:

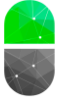
- Dejar sin definir alguno o todos los métodos abstractos. Si tomamos esta opción la clase extendida deberá ser declarada también abstracta.
- Definir todos los métodos. Si tomamos esta opción la subclase ya no es abstracta.

Por ejemplo, vamos a definir una clase llamada Alumno que extiende a la clase abstracta Persona e implementa el método getDescripcion(). Como ninguno de los métodos de la clase Alumno es abstracto, no es necesario declararla como abstracta.

Las clases abstractas no pueden tener objetos. Si se declara una clase como abstracta, no se podrán crear objetos de la misma pues daría error a la hora de compilar.

Definimos una clase Alumno que extiende a la clase abstracta Persona:

```
class Alumno extends Personas {  
    public Alumno (String n, String m) {  
        super(n); especialidad =m;  
    }  
    public String getDescripcion() {  
        return "Un alumno que estudia " + especialidad;  
    }  
    private String especialidad;  
}
```



La clase Alumno define el método `getDescripcion()`. Todos los métodos de la clase Alumno están implementados luego entonces ya no será abstracta.

Ejemplo:

```
package es.pildorasinformaticas.pooAbstractas;

import java.util.*;

public class PruebaPersonas {

    public static void main (String args[]) {

        Persona [] gente=new Persona[2];

        //se rellena la matriz gente con objetos de los tipos Alumno y Empleado

        gente[0]=new Empleado("Paco", 50000, 1992, 10, 1);
        gente[1]=new Alumno("María", "informática");

        //se imprimen los nombres y descripciones de todos los objetos Persona

        for (Persona p: gente) {

            System.out.println(p.getNombre() + ", " + p.getDescripcion());

        }

    }

    abstract class Persona {

        public Persona (String n) {

            nombre=n;

        }

        public abstract String getDescripcion();

        public String getNombre() {

            return nombre;

        }

        private String nombre;

    }

    class Empleado extends Persona {

        public Empleado (String n, double s, int anno, int mes, int dia){

            super(n); sueldo=s;

            GregorianCalendar calendario=new GregorianCalendar(anno, mes -1, dia);

            fechaContrato=calendario.getTime();

        }

    }

}
```



```
    public double getSueldo() {  
        return sueldo;  
    }  
  
    public Date getFechaContrato() {  
        return fechaContrato;  
    }  
  
    public String getDescripcion() {  
        return "Un empleado con un sueldo de " + sueldo;  
    }  
  
    public void subirSueldo (double porcentaje) {  
        double aumento=sueldo * porcentaje/100; sueldo+=aumento;  
    }  
  
    private double sueldo;  
    private Date fechaContrato;  
}  
  
class Alumno extends Persona{  
    public Alumno (String n, String m){  
  
        //se pasa n al constructor de la superclase  
        super(n); especialidad=m;  
    }  
  
    public String getDescripcion() {  
        return "Un alumno que estudia " + especialidad;  
    }  
  
    private String especialidad;  
}
```

Revisando este ejemplo cuidadosamente nos encontraremos con gran parte de los conceptos hasta ahora aprendidos.