Programación Orientada a Objetos. Utilización avanzada de clases

Encarnación Sánchez Gallego

ÍNDICE

Introducción

- La POO es una herramienta para reducir la complejidad de los sistemas de software. Usando la POO un programa que ocupa miles y miles de líneas de código puede utilizarse como una colección de pequeñas unidades (objetos), cada una con cierta independencia y ciertas responsabilidades.
- Un programa OO consisten en un conjunto de objetos que intercambian **mensajes**, cada objeto decide si debe o no aceptar los mensajes que recibe, así como la interpretación de cada mensaje. En un programa OO mediante compilado, los objetos no son totalmente independientes: unos heredan propiedades y métodos de otros, algunos necesitan consultar a otros para desempeñar su tarea, otros llevan dentro de si más objetos.
- **La principal ventaja** de la POO es que un programa de objetos se extiende de manera más sencilla que uno sin ellos, el programador puede construir sus propios objetos de uno ya existente y personalizarlo de acorde a sus necesidades.

ÍNDICE

3. Clases y métodos abstractos y finales

La abstracción es una de las características de la POO. Con la abstracción lo que se hace es extraer la esencia básica y su comportamiento para luego después representarla en un lenguaje de programación.

Recuerda
Abstract en Java es sinónimo de genérico.

Clases y métodos abstractos

Clases pensadas para ser genéricas. Esto quiere decir que no va a haber objetos de esas clases puesto que no tiene sentido. La clase vehículo será genérica, ya que cuando se crea y la usamos crearemos objetos de esa clase como coche, bicicleta, etc. Es obvio que todos son vehículos y por lo tanto esta clase abstracta sólo define los atributos y comportamientos comunes.

Ejemplo:

```
public abstract class vehiculo{
    private int peso;
    public void setPeso(int p){peso=p;}
    public abstract int getVelocidadActual();
}
```

Se pueden implementar métodos abstractos y no abstractos.

3. Clases y métodos abstractos y finales

Recuerda

- De las clases abstractas no pueden crearse objetos.
- Si una clase tiene métodos abstract por fuerza tendrá que ser una clase abstracta.
- Un método *abstract* no puede ser *static*.
- Las subclases que implementen esta clase abstracta tendrán que redefinir estos métodos o bien declararlos también como *abstract*.

- **Objetos, clases y métodos finales**
- **Objetos finales**

Cuando un objeto se declara como final, este impedirá que hay otros objeto con la misma referencia. Por ejemplo cuando se realiza algo parecido a esto:

```
final cuadrado c1=new cuadrado(5);
cuadrado c2=new cuadrado(15);
c1=c2;
```

```
prueba.java:6: cannot assign a value to final variable c1
  c1=c2;
  ^
1 error
La compilación falló.
```

El compilador ,mostrará código de error en la 3º línea

3. Clases y métodos abstractos y finales

- **Métodos finales**
- Le decimos a ese método que no va a cambiar. No se sobrescribe.

```
public final void setColor(String s){color=s;}
```

- **Clases finales**

No pueden tener descendencia , no pueden tener subclases. Ejemplo:

```
public final class triangulo{
.....
}
```

ÍNDICE

4. Arrays de objetos

- Los arrays no solo pueden almacenar tipos primitivos. También pueden almacenar objetos. El funcionamiento es similar en los dos casos. Conviene destacar si no se especifica ningún valor los elementos de un array de objetos se inicializan automáticamente a **null**.
- Proporciona métodos **static** para la manipulación de arrays. Estos métodos incluyen:
    - **sort**: para ordenar un array en forma ascendente
    - **binarySearch**: para buscar en un array
    - **equals**: para comparar arrays
    - **fill**: para rellenar valores en un array.

4. Arrays de objetos

Cuando definimos e instanciamos un Array, especificamos el tipo de datos que contendrá. En esa especificación se pueden incluir objetos, obteniendo un array de objetos.

```java
public class POOarrayObjetos {

    public static void main(String[] args) {
        //array de 10 enteros
        int arrayInt[] = new int[10]; //un array de 10 "objetos número entero".

        //array de 10 objetos de la clase persona
        Persona arrayPersona[] = new Persona[10];
        arrayPersona[0] = new Persona("Pepe", "Pérez", 20);
        arrayPersona[1] = new Persona("Julia", "López", 26);
        //...

        arrayPersona[0].saludar();
    }
```

```
run:
Pepe es la persona número 1
Julia es la persona número 2
Hola, encantado. Mi nombre es Pepe
BUILD SUCCESSFUL (total time: 0 seconds)
```

Ventajas de este uso:

Se pueden definir múltiples objetos sin necesidad de asignarles un nombre, algo que parece trivial pero en Java no lo es.

Se podrán usar los objetos de manera sencilla haciendo referencia a un array.

4. Arrays de objetos

```
35      //método para obtener nombre y apellidos
36      public String getNombreApellidos(){
            return this.nombre this.apellido;
38      }
39  }
```

- En las funciones y métodos que tienen un valor

Resultado, el return sólo acepta un datos a devolver al Programa principal. Si se quiere devolver varios resultados, A priori no es posible.

Una posible solución es devolver un array de varios datos. Para ello:

- Se establece como dato de salida que será un array, especificando con [] tras el tipo de dato.
- Se define en el método el array interno (local) y se rellena con la lógica correspondiente.
- Se devuelve el valor de array en el return.
- La llamada al método se trata como un array desde el programa principal.

```
//método para obtener nombre y apellidos
public String[] getNombreApellidos(){
    String arrayNomApe[] = new String[2];
    arrayNomApe[0] = this.nombre;
    arrayNomApe[1] = this.apellido;
    return arrayNomApe;
}
```

```
public static void main(String[] args) {
    Persona Pepe = new Persona("Pepe","Pérez", 30);

    //llamada de nombre y apellidos
    System.out.println("Esta persona se llama " + Pepe.getNombreApellidos()[0] + " y se apellida " + Pepe.getNombreApellidos()[1]);
```

```
run:
Esta persona se llama Pepe y se apellida Pérez
```

ÍNDICE

4. Polimorfismo

- Permite “programar en forma general”, en vez de “programar en forma especifica” para una clase.
- Permite escribir programas que procesen objetos que compartan la misma superclase en una jerarquía de clases, como si todos fueran objetos de la superclase.
- El polimorfismo permite diseñar e implementar sistemas que puedan extenderse con facilidad.
- En tiempo de ejecución el tipo del objeto al que referencia la variable es el que determina método que se utilizara.
- La mayoría de las llamadas a los métodos se resuelven en tiempo de ejecución en base al tipo del objeto que se esta manipulando.
- Este proceso se conoce como vinculación dinámica o vinculación postergada.
- El operador instanceof se puede utilizar para determinar si el tipo de unobjeto especifico tiene la relación “es un” con un tipo especifico.
- Todos los objetos en Java conocen su propia clase y pueden acceder a esta información a través del método getClass, que todas las clases heredan de la clase Object.

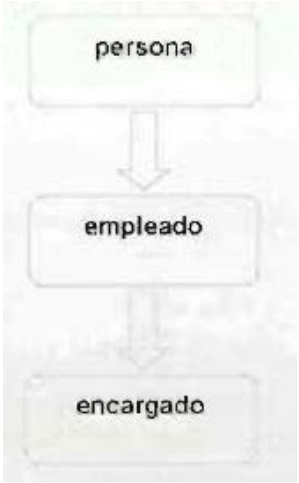
4. Polimorfismo

Permite abstraer y programar de forma general agrupando objetos con características comunes y jerarquizándolos en clases.

Recuerda

El polimorfismo se consigue en Java mediante las clases abstractas y las interfaces. Concretamente las interfaces amplían enormemente las posibilidades del polimorfismo.

Cuando se crea una referencia al objetos de la clase base,
esa misma referencia puede servir para referenciar a objetos
de la clase derivada.
Tenemos la clase persona de la cual desciende la clase empleado
La clase persona tiene métodos genéricos que pueden ser usados por
Cualquier persona como establecer y devolver nombre



4. Polimorfismo

- La implementación sería:

```
public class persona {
    private String nombre;
    public void setNombre(String nom){
        nombre = nom;
    }
    public String getNombre(){
        return nombre;
    }
}
```

```
public class empleado extends persona{
    protected int sueldoBase;
    public int getSueldo(){ return sueldoBase; }
    public void setSueldoBase(int s){ sueldoBase = s; }
}

public class encargado extends empleado{
    public int getSueldo(){
        Double d = new Double(sueldoBase*1.1);
        return d.intValue();
    }
}
```

4. Polimorfismo

- Realizamos lo siguiente con la clase test;

```
class test {
    public static void main(String[] args) {
        persona p1;
        p1 = new empleado();
        p1.setNombre("Isaac Sanchez");
        p1.setSueldoBase(100); //dará error al compilar
        empleado e1;
        e1 = new encargado();
        e1.setSueldoBase(500);
        e1.setPuesto("Jefe almacen"); //dará error al compilar
        System.out.println(e1.getSueldo());
    }
}
```

Comprueba lo que sucede pag 125

ÍNDICE

6. Clases abstractas

El paquete o

O importamos todas las clases de un paquete, Un ejemplo sería:

https://www.abrirllave.com/java/clases-abstractas.php

ÍNDICE

7. Interfaces

El paquete o

O importamos todas las clases de un paquete, Un ejemplo sería:

ÍNDICE

7. Sobreescritura de métodos

El paquete o

O importamos todas las clases de un paquete, Un ejemplo sería:

ÍNDICE

8. Acceso a métodos de la superclase

El paquete o

O importamos todas las clases de un paquete, Un ejemplo sería: