Polimorfismos y clases abstractas/interfaces

Regina Rodríguez

Creamos un sistema de pago para una empresa que tiene diferentes tipos de trabajadores; hay trabajadores de tiempo completo, tiempo parcial y contratista. Cada trabador tiene su propia forma para calcular su pago.

```
public abstract class Trabajador implements Pagable { //aqui estammos creando nuestra clase abstracta

    private String nombre;//atributos
    private int identificador;
    private int SalarioBase;


     Trabajador(String nombre, int identificador, int SalarioBase){//constructores
    this.nombre = nombre;
    this.identificador = identificador;
    this.SalarioBase = SalarioBase;

    }


   abstract int CalcularPago(); //metodo abstracto
```

Generamos nuestra clase abstracta con su debido constructor y un método abstracto para calcular nuestro pago.

```
 */
public interface Pagable { //interfaz

    void RealizarPago();

}
```

De igual forma creamos nuestra interfaz con un método **RealizarPago()**

Creamos nuestra clase **TrabajorTiempoTompleto**, **TrabajorTiempoParcial** y **Contratista** con sus debidos atributos y métodos, cada clase usara los métodos de **CalcularPagor**() y **RealizarPago**() a su debida forma, aquí estamos usando el polimorfismo porque tenemos que un mismo método pero que hace diferente trabajo.

```java
@Override
int CalcularPago() {//Polimorfismo porque en cada clase hace cosas diferentes
   Pago =  TarifaHora * HorasTrabajadas;
    return Pago;
}

@Override
 public void RealizarPago() {
     System.out.println("Empleado: " + getNombre() + " con un sueldo de:  " + CalcularPago());

 }
```

(Clase **Contratista**)

```java
@Override
public int CalcularPago() {//Polimorfismo porque en cada clase hace cosas diferentes
    Pago =  getSalarioBase() * bono;
    return Pago;
}

@Override
public void RealizarPago() {
     System.out.println("Empleado: " + getNombre() + " con un sueldo de:  " + CalcularPago());

}
```

Clase **TrabajorTiempoTompleto**

```java
    @Override
    int CalcularPago() {//Polimorfismo porque en cada clase hace cosas diferentes
       Pago =  getSalarioBase() * HorasTrabajadas;
        return Pago;
    }

    @Override
    public void RealizarPago() {
         System.out.println("Empleado: " + getNombre() + " con un sueldo de:  " + CalcularPago());

    }

}
```

Clase **TrabajorTiempoParcial**

En nuestra clase principal también usamos polimorfismos por lo mismo que tr nos da diferente mensaje

```java
for (Trabajador tr : trabajadores){ //aqui usamos uso del for each para que esta se este repitiendo dependiendo de la longityd del array
tr.RealizarPago(); //usamdo el polimorfismos porque el tr nos permite hacer diferentes tareas
System.out.println();
}
```

Singletor

Se quiere crear un sistema que permita personalizar el diseño de dispositivos, manteniendo un mismo idioma para todos, pero permitiendo a los usuarios cambiar el tema visual según sus preferencias.

```
*/
public class Configuracion {

    private String idioma;//atributos
    private String color;//atributos
     private static Configuracion configuracion; //instancia clase


     private Configuracion(String idioma) {//es un contructor privado para no crear multipleas instancias
        this.idioma = idioma;
    }

    public String getColor() {
        return color;
    }

    public void setColor(String color) {
        this.color = color;
    }


    public static Configuracion getInstance(String color) { //aqui obtenemos la instancia de la clase
        if(configuracion == null){
            configuracion = new Configuracion("español");//se determina el idioma de la configuracion
          configuracion.setColor(color);//agarra el color que tiene actualemte
            return configuracion;
        }else{
             configuracion.setColor(color);//toma el color por el que se cambio
            return configuracion;}
    }
```

Aquí usamos el uso de singletor porque queremos que todos los dispositivos tengan el mismo idioma, creamos nuestro constructor privado para no permite crear ultimes instancias del idioma, y ya en nuestro método es donde instanciamos la clase para definir el idioma determinado para el sistema.

Inyector de dependencias

Queremos desarrollar una aplicación de comercio electrónico que necesita enviar correos electrónicos a los clientes cuando realizan una compra. Para enviar estos correos electrónicos, necesitamos utilizar un servicio de envío de correos electrónicos como Gmail, Hotmal y Yahoo!.

```
*/
public class Persona {

    private String nombre;
    private String mensaje;
    private ServicioEmail servicioemail;


     public Persona(String nombre, String mensaje, ServicioEmail servicioemail) { //aqui usamos el inyector mediante el constrcutor
            this.nombre = nombre;
            this.mensaje = mensaje;
            this.servicioemail = servicioemail;
        }
```

Definimos en una clase **Persona** una variable de la interfaz

```
/**
 *
 */
public interface ServicioEmail {


    public void mensajae();


}
```

Creamos nuestra interfaz con un método público para enviar mensajes

```
/**
 * @author Regina
 */
public class inyectar {

    static Persona InyectarServicio(String nombre, String mensaje, Servicio servicioemail){//aqui creamos nuestro metodo donde se estan inyectando las opciones a las clases

        switch(servicioemail) {
                case GMAIL:
                    return new Persona(nombre,mensaje, new Gmail("Gmail"));
                case HOTMAIL:
                    return new Persona(nombre,mensaje,new Hotmail("Hotmail "));
                case YAHOO:
                    return new Persona(nombre,mensaje,new Yahoo("Yahoo!"));
                default:
                    throw new UnsupportedOperationException("Servicio de correo no existente");
                }


    }

}
```

De aquí creamos nuestro inyector que nos dirá con qué servicio de correo nos están dando el mensaje, y se lo agregamos a nuestra clase persona mediante una inyección por medio del constructor

```
public Persona(String nombre, String mensaje, ServicioEmail servicioemail) { //aqui usamos el inyector mediante el constrcutor
        this.nombre = nombre;
        this.mensaje = mensaje;
        this.servicioemail = servicioemail;
    }
```