

Puntos 3 del sprint 1

Patrones de diseño escogidos: singleton y states.

Singleton: El patrón singleton se utiliza para limitar la creación de una clase a un solo objeto. Esto es beneficioso cuando se necesita un objeto (y solo uno) para coordinar acciones en todo el sistema. Hay varios ejemplos en dónde debería existir una única instancia de una clase, incluidos cachés, grupos de subprocesos y registros.

este patrón nos garantiza que la clase tendrá una única instancia. Regularmente es utilizado para hacer controles de acceso a alguna base de datos o archivo.

El objetivo de los ejercicios en java aplicando este patrón es el siguiente:

En el paquete Singleton1, Crear un único objeto a una clase llamada persona con los atributos nombre y la edad de una persona e informar cuando se intente crear otro objeto de que ya existe un objeto, e imprimir por pantalla la información correspondiente.

En el paquete Singleton2 se resuelve de manera similar pero esta vez utilizando una clase Producto, con atributos como el código y el nombre del producto.

En ambos ejercicios se utiliza el método estático getIntance que nos controlara el acceso a la instancia singleton, y que también nos creara la instancia si aun no se ha creado.

Ejemplo 01

```

package singleton1;

/**
 *
 * @author luis
 */

public class Principal {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        Persona persona= Persona.getInstance("luis", "10");
        //Persona personal= Persona.getSingletonInstance("maria", "12");
        System.out.println(persona.getNombre());
        System.out.println(persona.getEdad());
    }

}

```

```

package singleton1;

public class Persona {
    private String nombre;
    private String edad;
    private static Persona persona;

    //el constructor es privado, lo cual no permitira que se genere un constructor por defecto
    private Persona(String nombre, String edad){
        this.nombre=nombre;
        this.edad=edad;
        System.out.println("el nombre es: "+this.nombre);
        System.out.println("la edad es: "+this.edad + " años");
    }

    public static Persona getInstance(String nombre, String edad){

        if(persona==null){
            persona= new Persona(nombre,edad);
        }else{

```

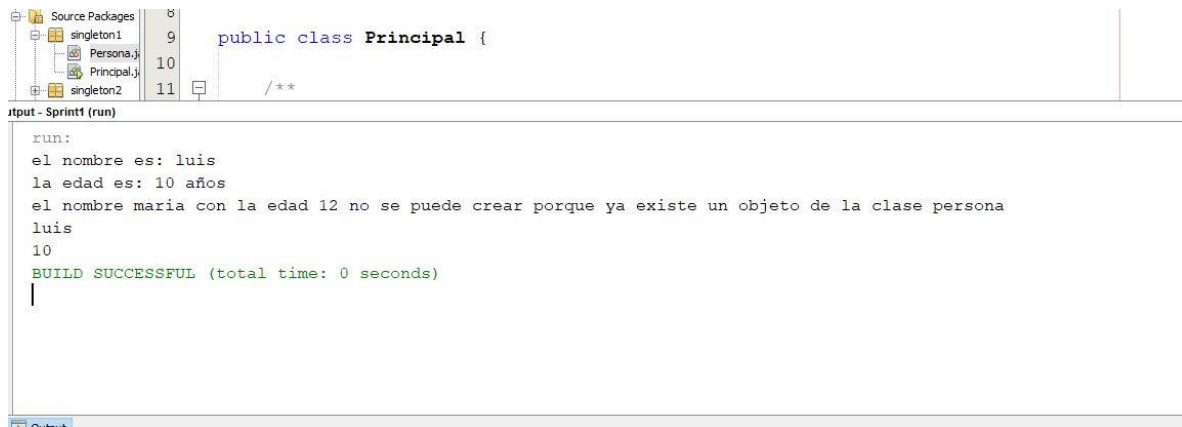
```
public static Persona getInstance(String nombre, String edad){  
    if(persona==null){  
        persona= new Persona(nombre,edad);  
    }else{  
        System.out.println("el nombre "+nombre+ " con la edad "+edad+" no se puede crear porque "  
            + "ya existe un objeto de la clase persona");  
    }  
    return persona;  
}  
  
public String getNombre() {  
    return nombre;  
}  
  
public void setNombre(String nombre) {  
    this.nombre = nombre;  
}  
  
public String getEdad() {
```

```
    public String getNombre() {  
        return nombre;  
    }
```

```
    public void setNombre(String nombre) {  
        this.nombre = nombre;  
    }
```

```
    public String getEdad() {  
        return edad;  
    }
```

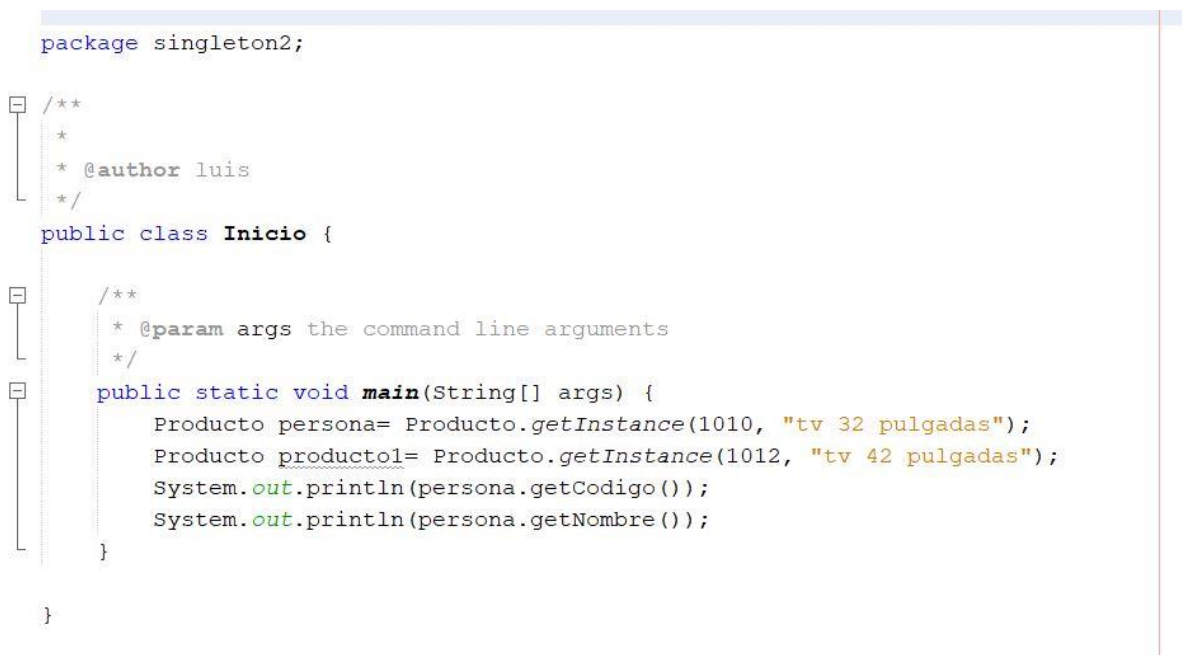
```
    public void setEdad(String edad) {  
        this.edad = edad;  
    }
```



The screenshot shows an IDE with a project structure on the left and a console output on the right. The project structure includes a package named 'singleton2' containing two classes: 'Persona.java' and 'Principal.java'. The 'Principal.java' file is open in the editor, showing a public class 'Principal' with a comment '/**'. The console output shows the following text:

```
run:
el nombre es: luis
la edad es: 10 años
el nombre maria con la edad 12 no se puede crear porque ya existe un objeto de la clase persona
luis
10
BUILD SUCCESSFUL (total time: 0 seconds)
```

Ejemplo 02



The screenshot shows an IDE with a Java code file open in the editor. The code is as follows:

```
package singleton2;

/**
 *
 * @author luis
 */
public class Inicio {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Producto persona= Producto.getInstance(1010, "tv 32 pulgadas");
        Producto producto1= Producto.getInstance(1012, "tv 42 pulgadas");
        System.out.println(persona.getCodigo());
        System.out.println(persona.getNombre());
    }
}
```

```
package singleton2;
```

```
/**  
 *  
 * @author luis  
 */
```

```
public class Producto {
```

```
    private int codigo;  
    private String nombre;  
    private static Producto producto;
```

```
//el constructor es privado, lo cual no permitira que se genere un constructor por defecto
```

```
    private Producto(int codigo, String nombre){  
        this.codigo=codigo;  
        this.nombre=nombre;  
        System.out.println("el codigo es: "+this.codigo);  
        System.out.println("el nombre es: "+this.nombre);  
    }  
}
```

```
public static Producto getInstance(int codigo, String nombre){
```

```
    if(producto==null){  
        producto= new Producto(codigo,nombre);
```

```
    }else{
```

```
        System.out.println("el nombre "+nombre+ " con el codigo "+codigo+" no se puede crear "  
        + "porque ya existe un objeto de la clase producto");
```

```
    }
```

```
    return producto;
```

```
}
```

```
/**
```

```
 * @return the codigo  
 */
```

```
public int getCodigo() {  
    return codigo;  
}
```

```
/**
```

```
 * @return the nombre to set
```

```

L      */
      public void setCodigo(int codigo) {
      [      this.codigo = codigo;
      ]      }

      /**
      [      * @return the nombre
      ]      */
      public String getNombre() {
      [      return nombre;
      ]      }

      /**
      [      * @param nombre the nombre to set
      ]      */
      public void setNombre(String nombre) {
      [      this.nombre = nombre;
      ]      }

      }

```

Output - Sprinter (run)

```

run:
el codigo es: 1010
el nombre es: tv 32 pulgadas
el nombre tv 42 pulgadas con el codigo 1012 no se puede crear porque ya existe un objeto de la clase producto
1010
tv 32 pulgadas
BUILD SUCCESSFUL (total time: 0 seconds)
|

```

States: su nombre en español “estado” nos da pista de lo que realiza y es el cambio de estado o comportamiento de un objeto cuando su estado interno cambia, el objetivo de este patrón es disminuir la cantidad de condicionales que se puedan presentar por causa de la cantidad de estados, y mejorar así la legibilidad y el mantenimiento del código.

Un ejemplo muy común y utilizado para explicar este patrón es realizando el proceso de un semáforo que cuenta principalmente con los estados verde, amarillo y rojo.

En los ejercicios utilizando el patrón state se busca ejemplificar el comportamiento de este patrón de diseño, por ejemplo, en el paquete

state1, se simula un inicio de sesión de un usuario que de acuerdo con su estado se tendrá un comportamiento.

Se establece inicialmente un estado con la sesión cerrada, lo cual cuando la persona intenta realizar un envío o un reenvío, el programa le informa que debe realizar el inicio de sesión, una vez el estado es cambiado (es decir, a inicia sesión), el programa ya le permite realizar las respectivas acciones de enviar y reenviar.

Ejemplo 01

```
package state1;

/**
 *
 * @author luis
 */
public class Usuario {
    Estado estado=new CerrarSesion();

    public void setEstado(Estado estado){
        this.estado =estado;
    }

    public void enviar(){
        estado.enviar();
    }

    public void reenviar(){
        estado.reenviar();
    }

}
```

```
package statel;
```

```
/**
 *
 * @author luis
 */
public class CerrarSesion implements Estado{
    @Override
    public void enviar(){
        System.out.println("iniciar la sesion ");
    }
    @Override
    public void reenviar(){
        System.out.println("iniciar la sesion ");
    }
}
```

```
package statel;
```

```
/**
 *
 * @author luis
 */
public class Cliente {

    public static void main(String[] args) {
        Usuario usuario= new Usuario();
        usuario.enviar();
        usuario.reenviar();
        usuario.setEstado(new IniciarSesion());
        usuario.enviar();
        usuario.reenviar();
    }
}
```



```
package statel;
```

```
/**
 *
 * @author luis
 */
public interface Estado {
    void enviar();
    void reenviar();
}
```

```
package statel;
```

```
/**
 *
 * @author luis
 */
public class IniciarSesion implements Estado{
    @Override
    public void enviar(){
        System.out.println("Enviar");
    }
    @Override
    public void reenviar(){
        System.out.println("Reenviar");
    }
}
```

```
: Output - Sprint1 (run)
run:
iniciar la sesion
iniciar la sesion
Enviar
Reenviar
BUILD SUCCESSFUL (total time: 0 seconds)
|
```

Ejemplo 02

Nota: para este ejemplo dos del patrón state se hizo uso de la web para entender y complementar conocimiento sobre el comportamiento o funcionamiento de este patrón.

```
package state2;

public class Abierta implements EstadoVentanilla{
    @Override
    public void atender(Persona persona){
        System.out.println("Atendiendo a: "+persona.getNombre());
    }
}
```

```
package state2;
```

```
public class Banco {
```

```
    private String nombre;
```

```
    private String direccion;
```

```
    private Ventanilla ventanilla;
```

```
    public Banco() {
```

```
        ventanilla =new Ventanilla();
```

```
    }
```

```
    public void atender(Persona persona){
```

```
        System.out.println(persona.getNombre()+ " ingresa a la fila");
```

```
        ventanilla.atender(persona);
```

```
    }
```

```
    public void suspenderVentanilla() {
```

```
        ventanilla.suspender();
```

```
    }
```

```
    public void cerrarVentanilla() {
```

```
        ventanilla.cerrar();
```

```
    }
```

```
    public void abrirVentanilla() {
```

```
        ventanilla.abrir();
```

```
    }
```

```
    /**
```

```
     * @return the nombre
```

```
     */
```

```
    public String getNombre() {
```

```
        return nombre;
```

```
    }
```

```
    /**
```

```
     * @return the direccion
```

```
     */
```

```
    public String getDireccion() {
```

```
        return direccion;
```

```
    }
```

```
}
```

```
package state2;
```

```
public class Cerrada implements EstadoVentanilla{  
    @Override  
    public void atender(Persona persona){  
        System.out.println("Ventanilla cerrada!! ");  
    }  
}
```

```
package state2;
```

```
public interface EstadoVentanilla {  
    public void atender(Persona persona);  
}
```

```
package state2;
```

```
public class Main {
```

```
    public static void main(String[] args) {  
        // TODO code application logic here  
        Persona persona1= new Persona("maria", "rodriguez",21);  
        Persona persona2= new Persona("juan", "velez",71);  
        Persona persona3= new Persona("lorena", "marquez",88);  
        Persona persona4= new Persona("teresa", "bermudez",38);  
  
        Banco banco= new Banco();  
        banco.atender(persona1);  
        banco.suspenderVentanilla();  
        banco.atender(persona2);  
        banco.atender(persona3);  
        banco.cerrarVentanilla();  
        banco.atender(persona4);  
    }  
}
```

```
package state2;
```

```
public class Persona {
```

```
    private String nombre;  
    private String apellido;  
    private int edad;
```

```
    public Persona(String nombre,String apellido,int edad){  
        setApellido(apellido);  
        setNombre(nombre);  
        setEdad(edad);  
    }
```

```
    /**  
     * @return the nombre  
     */
```

```
    public String getNombre() {  
        return nombre;  
    }
```

```
package state2;
```

```
public class Suspendida implements EstadoVentanilla{
    @Override
    public void atender(Persona persona){
        //el cajero esta ocupado, pero si es una persona mayor la atiende.
        if (persona.getEdad()>65){
            System.out.println("atendiendo a: "+persona.getNombre());
        }else {
            System.out.println("espere 5 minutos por favor...");
        }
    }
}
```

```
package state2;
```

```
public class Ventanilla {

    private String cajero;
    private EstadoVentanilla estado;

    public Ventanilla(){
        estado=new Abierta();
    }

    public void suspender(){
        estado=new Suspendida();
    }

    public void cerrar(){
        estado=new Cerrada();
    }

    public void abrir(){
        estado=new Abierta();
    }
}
```

```

public void atender(Persona persona) {
    estado.atender(persona);
}

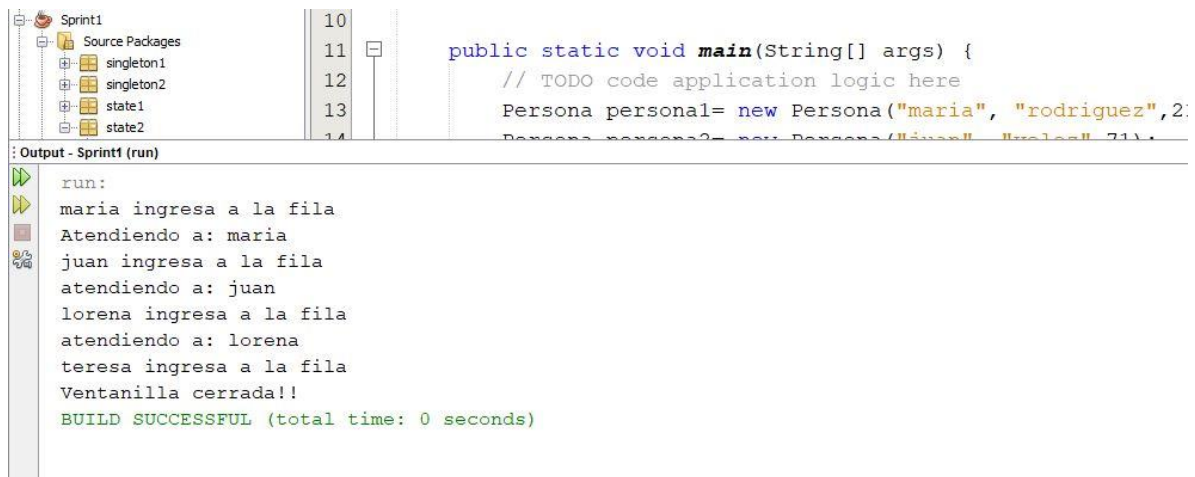
public String getCajero() {
    return cajero;
}

public EstadoVentanilla getEstadoVentanilla() {
    return estado;
}

/**
 * @param cajero the cajero to set
 */
public void setCajero(String cajero) {
    this.cajero = cajero;
}

public void setEstado(EstadoVentanilla estado) {
    this.estado = estado;
}

```



```

Sprint1
├── Source Packages
│   ├── singleton1
│   ├── singleton2
│   ├── state1
│   └── state2
└── Main
    └── public static void main(String[] args) {
        // TODO code application logic here
        Persona personal= new Persona("maria", "rodriguez",2);
        Persona persona2= new Persona("juan", "rodriguez", 2);
    }

```

```

run:
maria ingresa a la fila
Atendiendo a: maria
juan ingresa a la fila
atendiendo a: juan
lorena ingresa a la fila
atendiendo a: lorena
teresa ingresa a la fila
Ventanilla cerrada!!
BUILD SUCCESSFUL (total time: 0 seconds)

```

Realizado por:

Luis Alfredo Molina Socarras