PATRONES DE DISEÑO CREACIONALES

ETSII - URJC

Sebastián Ramiro Entierros García,
Adrián Martín Martín y
Álvaro Martínez Quiroga

Fundamentos del diseño del software

Índice

Contenido

Introducción y ficheros

En esta practica nos disponemos a desarrollar dos patrones de diseño, estos son el patrón Singleton y el patrón Factory Method. El paquete de la practica incluye:

- Código:
  - Punto 2D, hace uso del patrón de diseño de Singleton.
  - Login, hace uso del patrón de diseño de Singleton.
  - Factory, hace uso del patrón de diseño de Factory Method. Código sacado del aula virtual.
- Diagramas UML del método Factory Method.
- Memoria.

Patrón Singleton (I)

Clase Punto2D

```
public class Punto2D {

    private int x, y;

    public Punto2D(int x, int y){
        this.x = x;
        this.y = y;
    }

    @Override
    public String toString(){
        return "(" + getX() + "," + getY() + ")";
    }

    public boolean equals(Punto2D otroPunto){
        return getX() == otroPunto.getX() && getY() == otroPunto.getY();
    }

    public int getX(){return this.x;}
    public int getY(){return this.y;}

    public void setX(int x){this.x = x;}
    public void setY(int y){this.y = y;}
}
```

Clase Punto2DSingleton

```
public class Punto2DSingleton {

    private int x, y;
    private static Punto2DSingleton punto = null;

    private Punto2DSingleton(){
        this.x = 0;
        this.y = 0;
    }

    public static Punto2DSingleton getInstance(){
        if(punto == null){
            punto = new Punto2DSingleton();
        }
        return punto;
    }

    @Override
    public String toString(){
        return "(" + getX() + "," + getY() + ")";
    }

    public boolean equals(Punto2DSingleton otroPunto){
        return getX() == otroPunto.getX() && getY() == otroPunto.getY();
    }

    public int getX(){return this.x;}
    public int getY(){return this.y;}

    public void setX(int x){this.x = x;}
    public void setY(int y){this.y = y;}
}
```

Clase TestNormal

```
import java.util.Random;

public class TestNormal {

    private Punto2D punto1, punto2;
    private Random rdn = new Random();

    public TestNormal(){
        punto1 = new Punto2D(rdn.nextInt(100),rdn.nextInt(100));
        punto2 = new Punto2D(rdn.nextInt(100),rdn.nextInt(100));
    }
```

```
    public void ejecutarTest(){
        StringBuilder sb = new StringBuilder("---Prueba TestNormal:\n\n");

        sb.append("Punto1: ").append(punto1.toString()).append("\n");
        sb.append("Punto2: ").append(punto2.toString()).append("\n");
        sb.append("¿Puntos iguales?: ").append(punto1.equals(punto2));

        sb.append("\n\n-Cambiamos los valores de los puntos: ");
        int x1 = rdn.nextInt(100), x2 = rdn.nextInt(100), y1 = rdn.nextInt(100),
y2 = rdn.nextInt(100);
        sb.append("Nuevos valores -> x1:").append(x1).append(", x2:");
        sb.append(x2).append(", y1:").append(y1).append(", y2:").append(y2);

        punto1.setX(x1);
        punto1.setY(y1);
        punto2.setX(x2);
        punto2.setY(y2);

        sb.append("\n\n-Nuevos Puntos:\n\n");

        sb.append("Punto1: ").append(punto1.toString()).append("\n");
        sb.append("Punto2: ").append(punto2.toString()).append("\n");
        sb.append("¿Puntos iguales?: ").append(punto1.equals(punto2));

        sb.append("\n\n-Conclusiones:\n\n");

        sb.append("Podemos concluir que salvo coincidencia los puntos conservan
total desigualdad.");

        sb.append("\n\n---Fin Prueba TestNormal.\n\n");

        System.out.println(sb.toString());
    }
}
```

Clase TestSingleton

```
import java.util.Random;

public class TestSingleton {
    private Punto2DSingleton punto1, punto2;
    private Random rdn = new Random();

    public TestSingleton(){
        punto1 = Punto2DSingleton.getInstance();
        punto2 = Punto2DSingleton.getInstance();
```

```
    }

    public void ejecutarTest(){
        StringBuilder sb = new StringBuilder("---Prueba TestSingleton:\n\n");

        sb.append("Punto1: ").append(punto1.toString()).append("\n");
        sb.append("Punto2: ").append(punto2.toString()).append("\n");
        boolean equal1 = punto1.equals(punto2);
        sb.append("¿Puntos iguales?: ").append(equal1);

        sb.append("\n\n-Cambiamos los valores del segundo punto: ");
        int x = rdn.nextInt(100), y = rdn.nextInt(100);
        sb.append("Nuevos valores -> x:").append(x).append(", y:").append(y);

        punto2.setX(x);
        punto2.setY(y);

        sb.append("\n\n-Nuevos Puntos:\n\n");

        sb.append("Punto1: ").append(punto1.toString()).append("\n");
        sb.append("Punto2: ").append(punto2.toString()).append("\n");
        boolean equal2 = punto1.equals(punto2);
        sb.append("¿Puntos iguales?: ").append(equal2);

        sb.append("\n\n-Conclusiones:\n\n");
        if(equal1 && equal2) sb.append("Podemos concluir que el patron singleton
funciona sin problemas.");
        else sb.append("Podemos concluir que el patron singleton tiene un fallo
en su escritura.");

        sb.append("\n\n---Fin Prueba TestSingleton.\n\n");

        System.out.println(sb.toString());
    }
}
```

Clase Main

```
public class Main {
    public static void main(String[] args){
        TestNormal tn = new TestNormal();
        tn.ejecutarTest();
        TestSingleton ts = new TestSingleton();
        ts.ejecutarTest();
    }
}
```

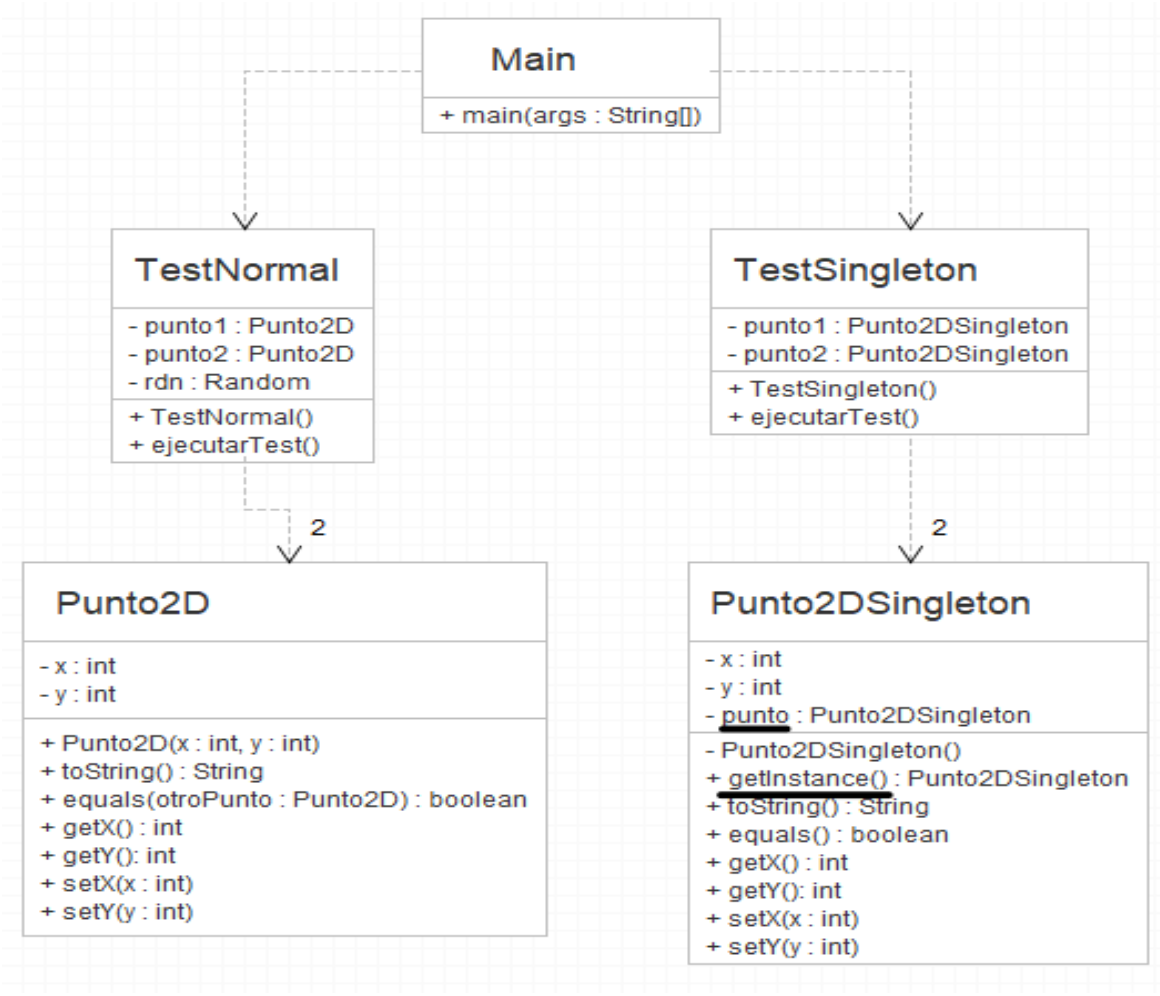


Diagrama 1: Punto2D

Patrón Singleton (II)

Clase Usuario

```
public class Usuario {

    private String username, password;

    public Usuario(String username, String password){
        this.username = username;
        this.password = password;
    }

    @Override
    public String toString(){
        return "Usuario: " + getUsername();
    }
```

```
    @Override
    public boolean equals(Object o) {
        if (o == this) {
            return true;
        }
        if (!(o instanceof Usuario)) {
            return false;
        }
        Usuario c = (Usuario) o;
        return (username.equals(c.username)) && (password.equals(c.password));
    }

    public String getUsername(){return this.username;}
    public String getPassword(){return this.password;}

    public void setUsername(String username){this.username = username;}
    public void setPassword(String password){this.password = password;}
}
```

Clase LoginSingleton

```
import java.util.ArrayList;

public class LoginSingleton {

    private static LoginSingleton ls = null;
    private ArrayList<Usuario> usuarios;

    private LoginSingleton(){
        usuarios = new ArrayList<>();
    }

    public static LoginSingleton getInstance(){
        if(ls == null){
            ls = new LoginSingleton();
        }
        return ls;
    }

    public int registrar(String nombre, String contra){
        for(Usuario us : usuarios){
            if(us.getUsername() == nombre){
                System.out.println("Ya existe el usuario con este nombre.");
                return 0;
            }
        }
        Usuario u = new Usuario(nombre, contra);
```

```
        usuarios.add(u);
        System.out.println("Usuario añadido.");
        return 1;
    }

    public int acceder(String nombre, String contra){
        Usuario u = new Usuario(nombre, contra);
        if(usuarios.contains(u)){
            System.out.println("Bienvenido usuario "+nombre+".");
            return 1;
        }
        System.out.println("No existe el usuario.");
        return 0;
    }
}
```

Clase TestLogin

```
public class TestLogin {
    private LoginSingleton log1, log2;

    public TestLogin(){
        log1 = LoginSingleton.getInstance();
        log2 = LoginSingleton.getInstance();
    }
    public void ejecutarTest(){
        System.out.println("---Pruebas TestLogin:\n\n");
        System.out.println("-Registramos al usuario Pepe de contraseña 1234 en
los dos logins:\n");
        log1.registrar("Pepe", "1234");
        log2.registrar("Pepe", "1234");
        System.out.println("\nComo podemos observar los dos log comparten los
datos.\n\n");

        System.out.println("-Registramos al usuario Antonio de contraseña 1234
en un login:\n");
        log1.registrar("Antonio", "1234");

        System.out.println("\n\n-Accedemos a Pepe en log1 y Antonio en
log2:\n\n");
        log1.acceder("Pepe", "1234");
        log2.acceder("Antonio", "1234");
        System.out.println("Seguimos viendo la comparticion de datos.\n\n");

        System.out.println("---Fin Pruebas TestLogin.\n\n");
    }
}
```

Clase Main

```
public class Main {
    public static void main(String[] args){
        TestLogin tl = new TestLogin();
        tl.ejecutarTest();
    }
}
```

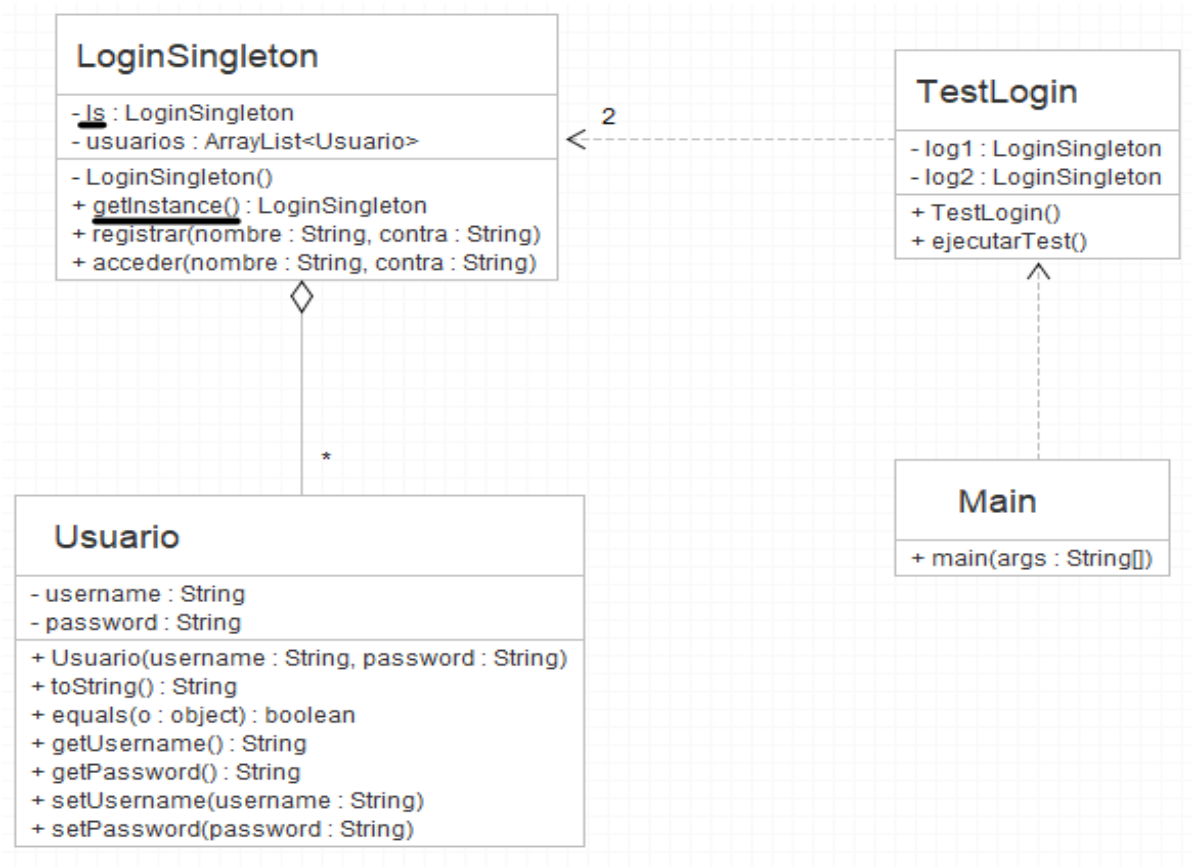
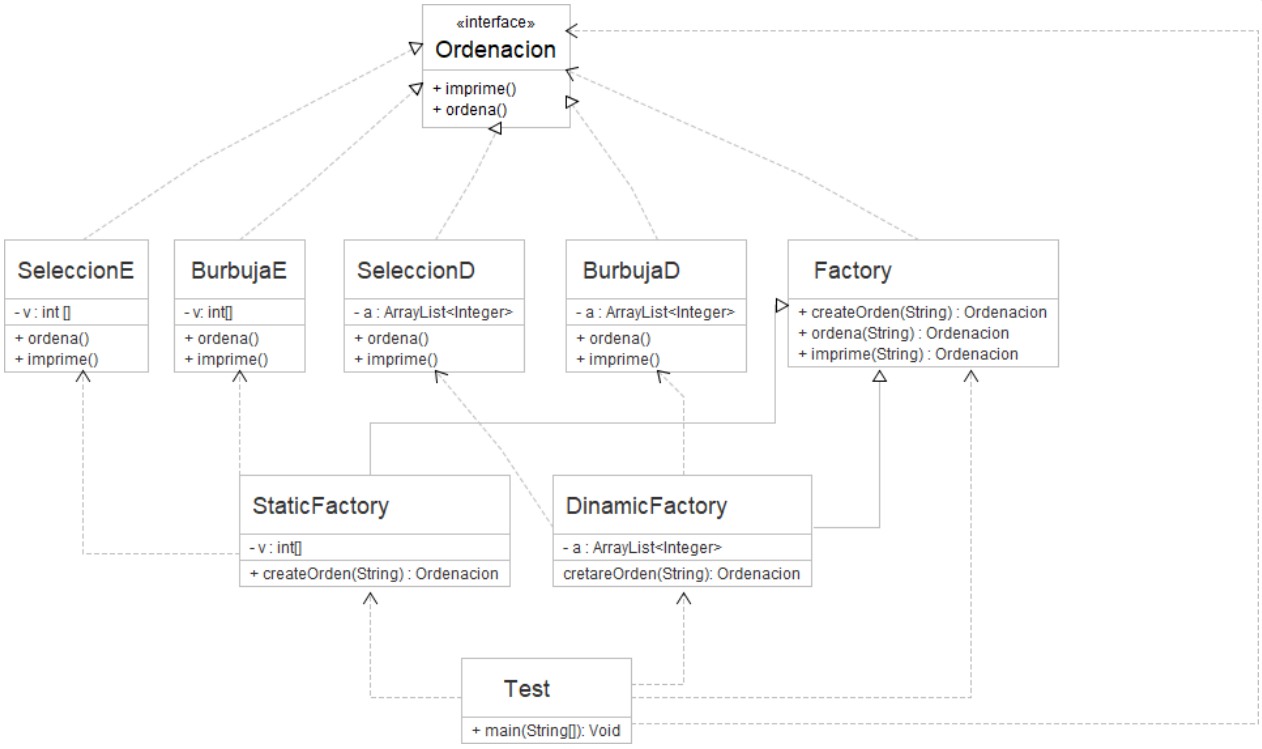


Diagrama 2: Log

Ejercicio 3: Patrón Factory Method


«interface»
Ordenacion
+ imprime()
+ ordena()
SeleccionE
- v : int []
+ ordena()
+ imprime()
BurbujaE
- v: int[]
+ ordena()
+ imprime()
SeleccionD
- a : ArrayList<Integer>
+ ordena()
+ imprime()
BurbujaD
- a : ArrayList<Integer>
+ ordena()
+ imprime()
Factory
+ createOrden(String) : Ordenacion
+ ordena(String) : Ordenacion
+ imprime(String) : Ordenacion
StaticFactory
- v : int[]
+ createOrden(String) : Ordenacion
DinamicFactory
- a : ArrayList<Integer>
cretareOrden(String): Ordenacion
Test
+ main(String[]): Void

Conclusiones de la práctica.

Consideramos esta practica muy interesante para tener y comprender la idea de código limpio. Gracias a esta practica hemos sido capaces de recapacitar en nuestra anterior forma de programar sin seguir ningún patrón de diseño, algo que resultaba en código no mantenibles en para el futuro de estos.

Nos ha parecido interesante en especial el método Singleton y sus posibles implementaciones de manera profesional para sistemas como puede ser un login (como se explica en la práctica), para diseñar un único logger para una aplicación, etc…

El patrón del Factory Method también nos ha parecido muy útil, sobre todo para dividir código y conseguir que un programa este muy bien dividido en diferentes módulos.

Referencias

- Temario del aula virtual.