PRÁCTICA 7. Diseño de Patrones | ***2 H (1 Ses)***

Objetivos:

- Comprender la importancia de crear distintos patrones en el diseño en nuestras aplicaciones.

Contenido:

El objetivo de la sesión práctica será la incorporación de distintos patrones en el proyecto del Desguace.

- **Se recomienda usar un nuevo paquete y copiar todo el código fuente de la sesión 4. No es necesario toda la lógica de desguace para esta sesión.**
- **Se ha dejado parte del código en el repositorio bitbucket público de la asignatura**

Patrón Singleton

La clase Teclado en su ejecución en el campusvirtual alternativo provoca errorres cuando es instancia más de una vez. Por ese motivo es necesario garantizar que solo haya una instancia de dicha clase para lo cual convertiremos la clase Teclado en Steclado (Singleton Teclado). Deberéis plantearos las siguientes preguntas:

- ¿Resulta adecuado aplicar Singleton para la gestión de la práctica? Razone su respuesta.
- Adapta la clase a las condiciones específicas de la práctica.Pasos:
  1. Copiar la clase Teclado y denominarla STeclado
  2. Definir un atributo STeclado en STeclado
  3. Definir el constructor privado
  4. Definir un método estático getInstance() que devuelva el atributo STeclado.
  5. Implementar un método main
- Implemente el modelo en Java y realice una sencilla clase main donde se vea el patrón funcionando.
  - Método que permita añadir Pieza
  - Método que permita añadir Vehículo
  - Método que permita añadir Pieza a Vehiculo
  - Método que permita mostrar las piezas del catálogo y los vehículos
- En cada uno de los métodos que se defina deberá aparecer la siguiente línea para poder usar la clase Teclado definida como singleton.

```
		STeclado st= Steclado.getInstance();
public class MainSingleton {
	private Desguace d= new Desguace();
	//No hace falta. Usaremos con singleton en cada método
	//private Teclado t = new Teclado();
	public static void main(String[] args) {
		MainSingleton m = new MainSingleton();
		m.ejecutar();
	}
	private int menu() throws IOException {
		STeclado st= STeclado.getInstance();
		int opcion = st.Menu(new String[] { "1. Añadir Pieza", "2. Añadir
Vehiculo", "3. Añadir Pieza Vehiculo","4. Mostrar Piezas/Vehiculo", "5. Salir" }, 1, 5);
```

```
		return opcion;

	}
	private boolean addPieza() throws IOException {
		STeclado st= STeclado.getInstance();
		String id = st.literalConString("Dame el id");

		Pieza p2 = d.getPiezaDesguace(id);
		if (p2 == null) {
			String nombre = st.literalConString("Dame elnombre");
			int stock = st.literalConEntero("Dame el número");
			Pieza paux = new Pieza(id, nombre, stock);
			d.addPiezaDesguace(paux);
			return true;
		}
		return false;
	}
	private boolean addVehiculo() throws IOException {
		STeclado st= STeclado.getInstance();
		Integer bastidor = st.literalConEntero("Dame el bástidor");
		String marca = st.literalConString("Dame el marca");
		String modelo = st.literalConString("Dame el modelo");
		int tipo = st.literalConEntero("1. Coche 2. Moto 3. Camion");
		Vehiculo v=null;
		if (tipo == 1) {
			String color= st.literalConString("Dame el color");
			v= new Coche(marca,modelo, new Persona(),bastidor,color);
		}
		return d.addVehiculo(v);
	}
	private boolean addPiezaVehiculo() throws IOException {
		STeclado st= STeclado.getInstance();
		String id2 = st.literalConString("Dame el id");
		Integer b2 = st.literalConEntero("Dame el bástidor");
		return d.addPiezaVehiculo(id2, b2);
	}
	public void ejecutar() {
		int opcion;
		do {
			try {
				opcion=menu();
				switch (opcion) {
				case 1:
					if (addPieza())
						System.out.println("Se ha añadido");
					else System.out.println("No se ha añadido");
					break;
				case 2:
					if (addVehiculo())
						System.out.println("Se ha añadido");
					else System.out.println("No se ha añadido");
					break;

				case 3:
					if (addPiezaVehiculo())
						System.out.println("Se ha añadido");
					else System.out.println("No se ha añadido");
					break;
				case 4:
					System.out.println(d.getCatalogo().toString());
					System.out.println(d.getVehiculos().toString());

					break;
				}
			} catch (IOException e) {
				opcion = 0;
				System.out.println("Excepción");
			}
		} while (opcion != 5);
```

```
        }
}
```

Patrón Factoría

Dado que el taller dispone de diversos tipos de vehículos similares entre sí: coches, motos y camiones. Se pide hacer uso de un patrón factoría para la instanciación de dichos objetos. Deberéis plantearos las siguientes preguntas:

- ¿Resulta adecuado aplicar SimpleFactory o Factory Method para instanciar los diferentes vehículos en el taller? ¿Cuál de los dos patrones sería más adecuado utilizar?
- Rehacer el diagrama de clases genérico aplicando el patrón (sería necesario una clase FactoriaVehiculo con el método buildVehiculo(String tipo,bastidor, marca, modelo))
- Implemente el modelo en Java y realice una sencilla clase main donde se vea el patrón funcionando. En este caso solo es necesario modificar el método addVehiculo para hacer la llamada a la factoria

```
// NO estatico
FactoriaVehiculo f = new FactoriaVehiculo();
Vehiculo v = f.buildVehiculo(tipo, bastidor, marca, modelo);
//
// SI Estático
// Vehiculo v = FactoriaVehiculo.buildVehiculo(tipo, bastidor, marca, modelo);

 if (tipo == 1) {
        String color= st.literalConString("Dame el color");
        ((Coche) v).setColor(color);
 }
```

Patrón Prototipo

Supondremos que el desguace esta especializado únicamente en tres productos: Freno, Volante y Ruedas. En este caso parece lógico utilizar el patrón prototipo para clonar estas piezas plantillas y modificar únicamente aquello que sea necesario.

- Rehacer el diagrama de clases genérico aplicando el patrón. Será necesario hacer la clase Freno (con marca), Rueda (con modelo) y Volante (con model). Deberéis cambiar en Pieza los atributos de private a protected.
- Para implementar dicho patrón, estas clases: Pieza, Freno, Volante y Rueda deberán implementar el interface Cloneable así como implementar el método clone. Se recomienda ver las transparencias del tema (hay un ejemplo).
- Implementar la clase PrototypePieza donde se define un mapa y se añadan las tres plantillas ha dicho mapa (freno, volante, rueda). Se recomienda ver las transparencias.
- Implemente el modelo en Java y realice una sencilla clase main donde se vea el patrón funcionando.

```
if (p2 == null) {
                PrototypePieza ptp= new PrototypePieza();
                Pieza paux=null;
                int tipo = st.literalConEntero("1. Freno 2. Volante 3. Rueda");
                if (tipo==1) {
                        paux=(Pieza) ptp.prototipo("freno"); //
                        String marca = st.literalConString("Dame el marca");
                        ((Freno)paux).setMarca(marca);
                        ((Freno)paux).setId(id);

                }
                d.addPiezaDesguace(paux);
                return true;
        }
```