Singleton

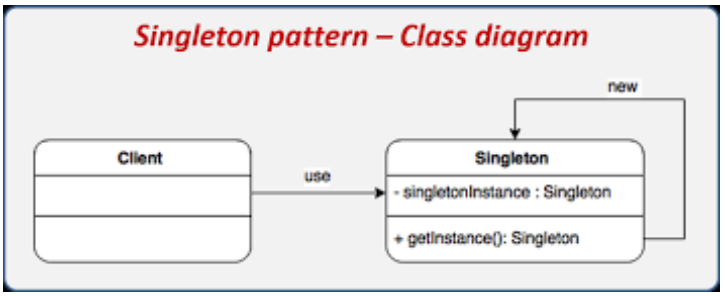
Es un patrón que restringe la creación a un único objeto la creación de objetos pertenecientes a una clase y asegura de que sólo haya esta instancia única.

¿Cómo se utiliza?
Se crea una clase con:

- Constructor privado,
- Una instancia estática privada de sí misma,
- Un método público estático que devuelve esa instancia.

Código:

```
public class Singleton {
   private static Singleton instancia;
   private Singleton() {} // constructor privado
   public static Singleton getInstancia() {
      if (instancia == null) {
         instancia = new Singleton();
      }
      return instancia;
   }
}
```



Ventajas:

- Control total sobre la instancia.
- Útil para recursos compartidos (configuraciones, conexión a bases de datos).
- Mejora la eficiencia evitando múltiples instancias innecesarias.

Decorator

Permite añadir funcionalidades a un objeto en tiempo de ejecución sin modificar su clase.

¿Cómo se utiliza en Java?

- Se define una interfaz común.
- Se crean clases base concretas.
- Los decoradores implementan la misma interfaz y contienen una instancia del objeto decorado.

Código

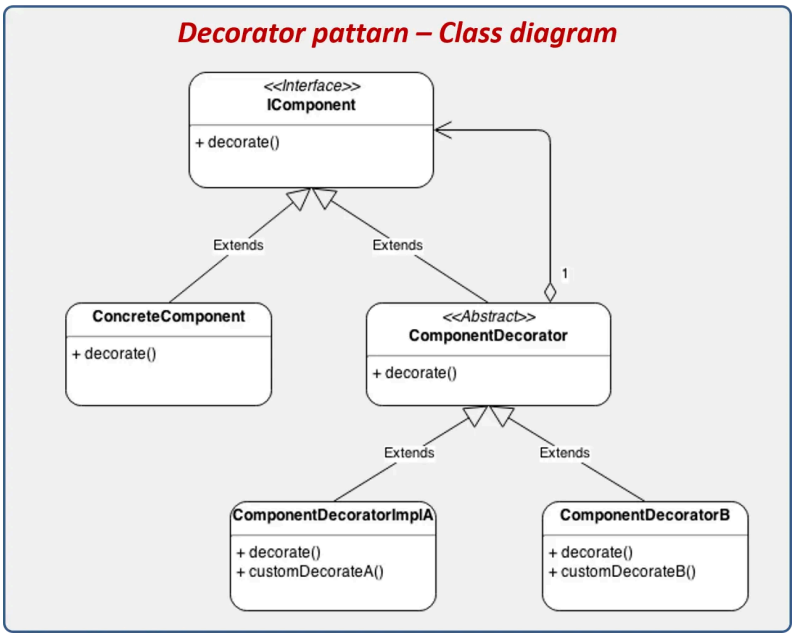
```
public interface Cafetera {
```

```
    String servir();
}

public class CafeSimple implements Cafetera {
    public String servir() {
        return "Café";
    }
}

public class ConLeche implements Cafetera {
    private Cafetera cafetera;
    public ConLeche(Cafetera c) { this.cafetera = c; }

    public String servir() {
        return cafetera.servir() + " con leche";
    }
}
```



Ventajas:

- Añade responsabilidades de forma flexible.
- Cumple con el principio de abierto/cerrado (OCP).
- Evita jerarquías de herencia rígidas.

Command

Es una técnica que encapsula una solicitud como un objeto. Así se puede parametrizar el emisor con diferentes operaciones y permitir funcionalidades como deshacer.

¿Cómo se utiliza en Java?

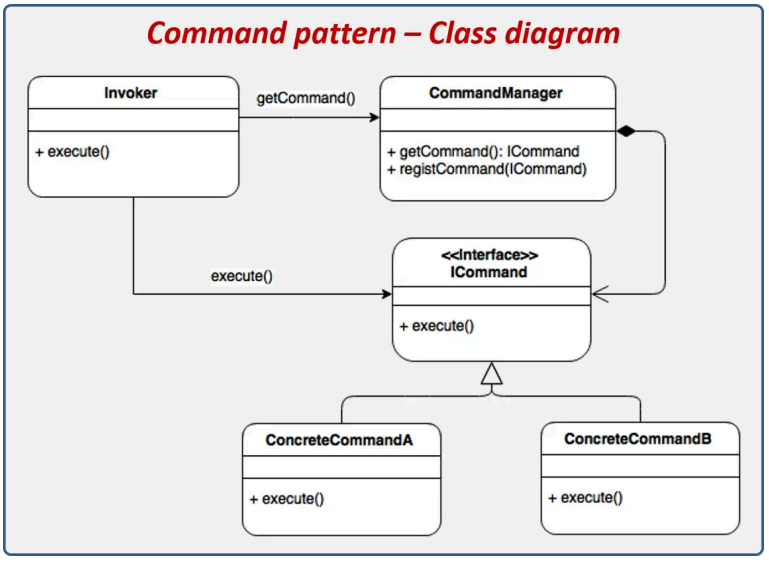
- Se define una interfaz `Command` con un método `execute()`.
- Se crean implementaciones concretas de `Command`.
- Un "invocador" almacena los comandos y los ejecuta.

Código

```
public interface Command {
   void execute();
}

public class EncenderLuz implements Command {
   public void execute() {
      System.out.println("Luz encendida");
   }
}

public class ControlRemoto {
   private Command comando;
   public void setCommand(Command c) { this.comando = c; }
   public void presionarBoton() { comando.execute(); }
}
```



Ventajas:

- Desacopla emisor y receptor.
- Facilita funcionalidades como colas de tareas, logs, deshacer/rehacer.
- Fomenta el uso de operaciones como objetos reutilizables.