DISEÑO DE SISTEMAS

Juan Carlos Molina Lozano

Docente

CONTENIDO

- Introducción
- Objetivos de la Clase
- Propósito de los Patrones Creacionales
- Características de los Patrones Creacionales
- Ventajas y Desventajas

OBJETIVOS DE LA CLASE

Que los estudiantes comprendan el concepto, propósito y aplicación de los patrones de diseño creacionales, analicen sus características, ventajas y desventajas, y sean capaces de identificar situaciones en las que se deben aplicar, mediante ejemplos prácticos que les permitan construir el conocimiento a partir de la experiencia.

INTRODUCCIÓN

En la programación orientada a objetos, uno de los desafíos más comunes es crear objetos de manera flexible y controlada, especialmente en aplicaciones complejas. Aquí es donde entran en juego los patrones de diseño creacionales: soluciones reutilizables a problemas comunes relacionados con la creación de instancias. Estos patrones permiten mejorar la escalabilidad, mantenimiento y reutilización del código.

QUÉ SON LOS PATRONES CREACIONALES?

Son patrones que abordan los problemas relacionados con la creación de objetos, evitando el uso directo del operador new. Permiten que el sistema sea independiente de cómo se crean, componen y representan los objetos.

Principales Patrones Creacionales

- Singleton
- Factory Method
- Abstract Factory
- Builder
- Prototype

PATRONES CREACIONALES

Singleton

- Propósito:

Garantizar que una clase tenga una única instancia y proporcionar un punto de acceso global a ella.

- Cuándo usarlo?

Cuando se necesita una única conexión (por ejemplo, a una base de datos).Cuando se requiere un control centralizado (como un gestor de configuración o log).

- Características:

Constructor privado.

Instancia estática y método público para obtenerla.

Controla su propia instancia.

PATRONES CREACIONALES

Singleton

☑ Ventajas:

Control estricto de instancias.

Ahorro de recursos.

Punto único de acceso.

❌ Desventajas:

Dificulta pruebas unitarias (acoplamiento).

Puede convertirse en un "objeto global" mal usado

```java
public class Logger {
    private static Logger instancia;

    private Logger() {}

    public static Logger getInstancia() {
        if (instancia == null) {
            instancia = new Logger();
        }
        return instancia;
    }

    public void log(String mensaje) {
        System.out.println("LOG: " + mensaje);
    }
}
```

PATRONES CREACIONALES

Factory Method

- Propósito:

Define una interfaz para crear un objeto, pero permite a las subclases decidir qué clase instanciar. Delega la responsabilidad de instanciación.

- Cuándo usarlo?

Cuando no se conoce de antemano qué tipo exacto de objeto se necesita.

Cuando las clases deben decidir qué subclase instanciar.

- Características:

Interfaz o clase abstracta con método crearProducto().Las subclases deciden qué objeto específico crear.

PATRONES CREACIONALES

Factory Method

☑ Ventajas:

Fomenta el principio de abierto/cerrado.

Flexibilidad para agregar nuevos productos.

❌ Desventajas:

Mayor número de clases.

Puede ser más complejo de entender inicialmente.

```java
abstract class Documento {
    public abstract void mostrar();
}

class PDF extends Documento {
    public void mostrar() {
        System.out.println("Mostrar PDF");
    }
}

class Word extends Documento {
    public void mostrar() {
        System.out.println("Mostrar Word");
    }
}

abstract class CreadorDocumento {
    public abstract Documento crearDocumento();
}

class CreadorPDF extends CreadorDocumento {
    public Documento crearDocumento() {
        return new PDF();
    }
}
```

PATRONES CREACIONALES

Abstract Factory

- Propósito:

Proporciona una interfaz para crear familias de objetos relacionados sin especificar sus clases concretas.

- Cuándo usarlo?

Cuando se necesitan conjuntos de objetos interrelacionados.

Para garantizar compatibilidad entre productos de una misma familia.

- Características:

Interfaz para crear múltiples tipos de objetos.

Cada fábrica concreta implementa la interfaz.

PATRONES CREACIONALES

Abstract Factory

☑ Ventajas:

Agrupa objetos relacionados.

Garantiza la coherencia entre objetos.

❌ Desventajas:

Puede ser excesivo si no se requiere una familia completa.

Añadir nuevos productos puede requerir cambios extensivos.

```
interface Boton {
    void dibujar();
}

class BotonWindows implements Boton {
    public void dibujar() {
        System.out.println("Botón estilo Windows");
    }
}

class BotonMac implements Boton {
    public void dibujar() {
        System.out.println("Botón estilo Mac");
    }
}

interface GUIFactory {
    Boton crearBoton();
}

class WindowsFactory implements GUIFactory {
    public Boton crearBoton() {
        return new BotonWindows();
    }
}

class MacFactory implements GUIFactory {
    public Boton crearBoton() {
        return new BotonMac();
    }
}
```

PATRONES CREACIONALES

Builder

- Propósito:

Separar la construcción de un objeto complejo de su representación, de modo que el mismo proceso pueda crear diferentes representaciones.

- Cuándo usarlo?

Cuando los objetos tienen muchos atributos opcionales.

Cuando se necesita crear variantes de un mismo objeto.

- Características:

Director que dirige el proceso de construcción.

Builder que define los pasos de construcción.Producto final ensamblado paso a paso.

PATRONES CREACIONALES

Builder

☑ Ventajas:

Control preciso sobre el proceso de construcción.

Permite crear objetos paso a paso.

Evita constructores con muchos parámetros.

❌ Desventajas:

Más clases y complejidad si se abusa.

Puede ser innecesario en objetos simples.

```
class Casa {
    private String puertas;
    private String ventanas;

    public void setPuertas(String puertas) { this.puertas = puertas; }
    public void setVentanas(String ventanas) { this.ventanas = ventanas; }
}

interface ConstructorCasa {
    void construirPuertas();
    void construirVentanas();
    Casa getCasa();
}

class ConstructorCasaModerna implements ConstructorCasa {
    private Casa casa = new Casa();

    public void construirPuertas() {
        casa.setPuertas("Puertas modernas");
    }

    public void construirVentanas() {
        casa.setVentanas("Ventanas grandes");
    }

    public Casa getCasa() {
        return casa;
    }
}
```

PATRONES CREACIONALES

Prototype

- Propósito:

Permite clonar objetos existentes sin acoplar el código a sus clases específicas.

- Cuándo usarlo?

Cuando la creación de objetos es costosa.

Cuando se necesita una copia exacta de un objeto con sus valores actuales.

- Características:

Requiere que el objeto implemente un método de clonación.

Crea nuevas instancias copiando un prototipo.

PATRONES CREACIONALES

Prototype

☑ Ventajas:

Reducción de tiempo de creación.

Facilita la duplicación de objetos.

```
abstract class Figura implements Cloneable {
    public String color;

    public Figura clonar() {
        try {
            return (Figura) super.clone();
        } catch (CloneNotSupportedException e) {
            return null;
        }
    }
}

class Circulo extends Figura {
    public int radio;
}
```

❌ Desventajas:

Requiere cuidado en la copia de objetos anidados (copia profunda vs superficial).

Puede ser difícil mantener la consistencia si el objeto cambia con el tiempo.

CARACTERÍSTICAS DE LOS PATRONES CREACIONALES

- Ocultan la lógica de instanciación.
- Fomentan la reutilización de código.
- Promueven la independencia del sistema respecto a cómo se crean los objetos.
- Facilitan la evolución del diseño sin grandes modificaciones.

VENTAJAS Y DESVENTAJAS

Ventajas

- ☑ Mejoran la modularidad del código.
- ☑ Aumentan la flexibilidad del sistema.
- ☑ Permiten implementar el principio de inversión de dependencias.
- ☑ Reducen el acoplamiento entre componentes.

Desventajas

- ❌ Pueden incrementar la complejidad del diseño si no se aplican adecuadamente.
- ❌ En algunos casos, generan sobrecarga innecesaria si el problema a resolver es sencillo.
- ❌ Requieren mayor experiencia y análisis previo para aplicar correctamente.

REFERENCIAS

Refactoring Guru

Patrones Creacionales

https://refactoring.guru/es/design-patterns/creational-patterns

PREGUNTAS