Patrones de Diseño en Java

Singleton

Descripción del patrón Singleton

El patrón Singleton asegura que una clase tenga una única instancia y proporciona un punto de acceso global a ella. Es útil cuando se necesita un control centralizado.

Implementación en Java

```
public class Singleton {
private static Singleton instance;
}
```

Ejemplo de uso

```
Singleton singleton =
Singleton.getInstance();
```

Builder

Descripción del patrón Builder

El patrón Builder permite construir un objeto complejo paso a paso. Separa la construcción de un objeto de su representación, facilitando la creación de diferentes representaciones.

Implementación en Java

```
class Product {
private String part1;
private String part2;
}
```

```
class Builder {
private Product product = new Product();
}
```

Ejemplo de uso

```
Builder builder = new Builder();
builder.buildPart1();
builder.buildPart2();
Product product = builder.getResult();
```

Abstract Factory

Descripción del patrón Abstract Factory

El patrón Abstract Factory proporciona una interfaz para crear familias de objetos relacionados sin especificar sus clases concretas. Es útil para mantener la consistencia entre productos.

Implementación en Java

```
interface AbstractFactory {
ProductA createProductA();
ProductB createProductB();
}
```

```
class ConcreteFactory1 implements
AbstractFactory {
public ProductA createProductA() {
return new ProductA1();
}
public ProductB createProductB() {
return new ProductB1();
}
}
```

```
class ConcreteFactory2 implements
AbstractFactory {
public ProductA createProductA() {
return new ProductA2();
}
public ProductB createProductB() {
return new ProductB2();
}
}
```

Ejemplo de uso

```
AbstractFactory factory = new
ConcreteFactory1();
ProductA productA =
factory.createProductA();
ProductB productB =
factory.createProductB();
```

Factory Method

Descripción del patrón Factory Method

El patrón Factory Method permite crear objetos sin especificar la clase exacta del objeto que se va a crear. Facilita la creación de productos relacionados.

Implementación en Java

```
abstract class Product {
abstract void use();
}
```

```
class ConcreteProductA extends Product
{
void use() {
System.out.println("Usando Producto A");
}
}
```

```
class ConcreteProductB extends Product
{
void use() {
System.out.println("Usando Producto B");
}
}
```

```
abstract class Creator {
abstract Product factoryMethod();
}
```

```
class CreatorA extends Creator {
Product factoryMethod() {
return new ConcreteProductA();
}
}
```

```
class CreatorB extends Creator {
Product factoryMethod() {
return new ConcreteProductB();
}
}
```

Ejemplo de uso

```
Creator creator = new CreatorA();
Product product =
creator.factoryMethod();
product.use();
```

Comparación de Patrones

Diferencias clave entre los patrones

Cada patrón tiene su propósito específico: Singleton para instancias únicas, Factory Method para creación de productos, Abstract Factory para familias de productos y Builder para construcción compleja.

Contexto de uso

La elección del patrón depende del problema a resolver, la necesidad de flexibilidad y la complejidad del objeto a crear.