6TO ENSAYO DE LA CLASE

PATRONES DE DISEÑO

Un design pattern o patrón de diseño consiste en un diagrama de objetos que forma una solución a un problema conocido y frecuente. El diagrama de objetos está constituido por un conjunto de objetos descritos por clases y relaciones que enlazan objetos.

Los patrones responden a problemas de diseño de aplicaciones en el marco de la programación orientada a objetos. Se trata de soluciones conocidas y probadas cuyo diseño proviene de la experiencia de los programadores. No existe un aspecto teórico en los patrones, en particular no existe una formalización(a diferencia de los algoritmos). Los patrones de diseño están basadoenlas buenas prácticas de la programación orientada a objetos.

Los patrones se organizan según su vocación en:

- Patrones de construcción
- Patrones de estructuración
- Patrones de comportamiento

Patrones de Construcción

Los patrones de construcción tienen la vocación de abstraer los mecanismos de creación deobjetos. Un sistema que utilice estos patrones se vuelve independiente de la forma en que se crean los objetos, en particular, de los mecanismos de instanciación de las clases concretas. Estos patrones encapsulan el uso de clases concretas y favorecen así el uso de las interfaces en las relaciones entre objetos, aumentando las capacidades de abstracción en el diseño global del sistema. Son en total 5:

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

Patrones de Estructuración

Los patrones de estructuración tienen como objetivo facilitar la independencia de la interfaz deun objeto o un conjunto de objetos respecto de su implementación. En el caso de un conjunto de objetos, se trata también de hacer que esta interfaz sea independiente de la jerarquía de clases y de la composición de los objetos. Son en total 7:

- Adapter
- Bridge
- Composite
- Decorator
- Facade
- Flyweight

- Proxy

Patrones de Comportamiento

Los patrones de estructuración aportan soluciones a los problemas de estructuración de datos y de objetos. El objetivo de los patrones de comportamiento consiste en proporcionar soluciones para distribuir el procesamiento y los algoritmos entre los objetos.
Estos patrones organizan los objetos así como sus interacciones especificando los flujos de control y de procesamiento en el seno de un sistema de objetos. Son un total de 11:

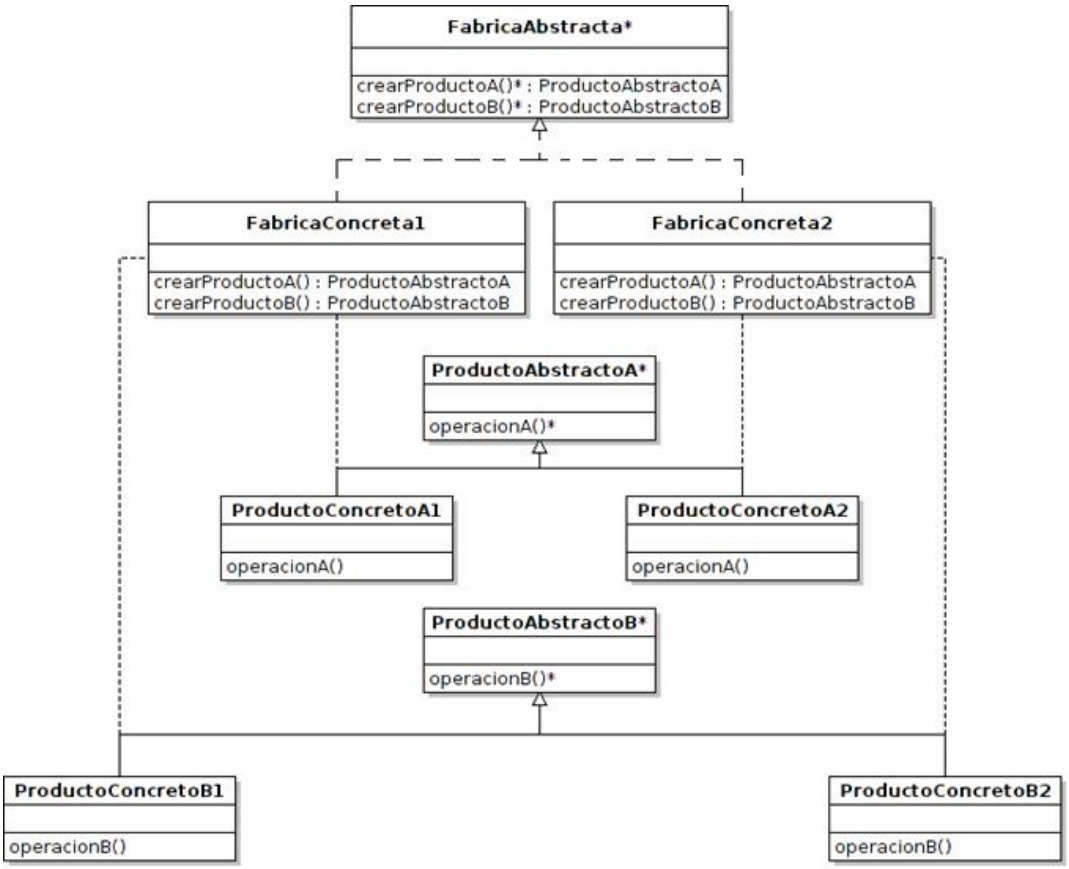
- Chain of responsibility
- Command
- Interpreter
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template Method
- Visitor

El patrón Abstract Factory

El objetivo del patrón Abstract Factory es la creación de objetos agrupados en familias sin tenerque conocer las clases concretas destinadas a la creación de estos objetos.

Elementos del patrón Abstract Factory:

- **FábricaAbstracta.-**Es una interfaz que define las firmas de los métodos que crean los distintos productos.
- **FábricaConcreta1, FábricaConcreta2**.-Son las clases concretas que implementan los métodos que crean los productos para cada familia de producto. Conociendo la familia y el producto, son capaces de crear una instancia del producto para esta familia.
- **ProductoAbstractoA,ProductoAbstractoB**.-Son las clases abstractas de los productos independientemente de su familia. Las familias se introducen en las subclases concretas.



Ejemplo:

Hay dos clases de productos libros y agendas, que pueden ser virtual o físicas utilizamos el patrón Abstract Factory para su implementación en Java.

```java
public abstract class Libro {  // Es nuestro ProductoAbstracto A
    private String nombre;
    private String autor;
    private Double precio;

    public Libro(String nombre, String autor, Double precio) {
        this.nombre = nombre;
        this.autor = autor;
        this.precio = precio;
    }
}
```

```java
public abstract class Agenda { //Es nuestro ProductoAbstracto B
    private String propietario;
    private String marca;

    public Agenda(String propietario, String marca) {
        this.propietario = propietario;
        this.marca = marca;
    }
}
```

```java
public class LibroFisico extends Libro { //ProductoConcreto A1
    public LibroFisico(String nombre, String autor, Double precio) {
        super(nombre, autor, precio);
    }
}
```

```java
public class LibroVirtual extends Libro{ //ProductoConcreto A2
    public LibroVirtual(String nombre, String autor, Double precio) {
        super(nombre, autor, precio);
    }

}
```

```java
public class AgendaFisica extends Agenda{ ///ProductoConcreto B1
    public AgendaFisica(String propietario, String marca) {
        super(propietario, marca);
    }
}
```

```java
public class AgendaVirtual extends Agenda{//ProductoConcreto B2
    public AgendaVirtual(String propietario, String marca) {
        super(propietario, marca);
    }
}
```

```java
public interface Fabrica {//Interfaz Fabrica
    Libro crea_libro(String nombre, String autor, Double precio);
    Agenda crea_agenda(String propietario, String marca);
}
```

```java
public class FabricaFisica implements Fabrica {// FabricaConcreta 1
    @Override
    public Libro crea_libro(String nombre, String autor, Double precio) {
        return new LibroFisico(nombre,autor,precio);
    }

    @Override
    public Agenda crea_agenda(String propietario, String marca) {
        return new AgendaFisica(propietario,marca);
    }
}
```

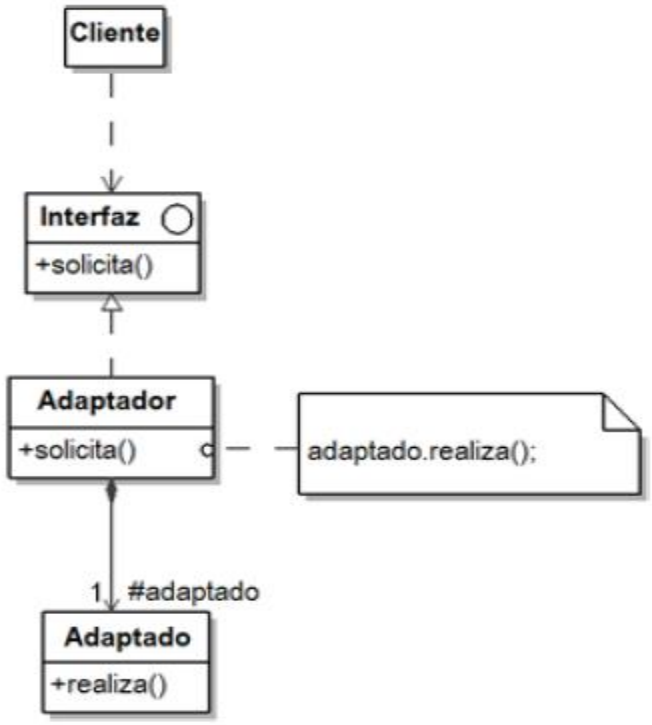
```java
public class FabricaVirtual implements Fabrica {// FabricaConcreta 2
    @Override
    public Libro crea_libro(String nombre, String autor, Double precio) {
        return new LibroVirtual(nombre,autor,precio);
    }

    @Override
    public Agenda crea_agenda(String propietario, String marca) {
        return new AgendaVirtual(propietario,marca);
    }
}
```

El patrón Adapter

El objetivo del patrón Adapter es convertir la interfaz de una clase existente en la interfaz esperada por los clientes también existentes de modo que puedan trabajar de manera conjunta. Se trata de conferir a una clase existente una nueva interfaz para responder a las necesidades de los clientes.

Elementos del patrón Adapter:

- **Interfaz.-**Incluye la firma de los métodos del objeto.
- **Cliente.-**Interactúa con los objetos respondiendo a la interfaz Interfaz.
- **Adaptador.-**Implementa los métodos de la interfaz Interfaz invocando a los métodos del objeto adaptado.
- **Adaptado.-**Incluye el objeto cuya interfaz ha sido adaptada para corresponder a la interfaz Interfaz.



Mendoza Medrano Adrián Esteban