Actividad para el sprint 1 ciclo 4

2 patrones de diseño explicados con 2 ejemplos de la vida real y con código fuente.

The Abstract Factory Method: Es un patrón de diseño creacional que nos permite producir familias de objetos relacionados sin especificar sus clases concretas, retorna una de las varias familias de objetos. Una jerarquía que encapsula: muchas plataformas posibles y la construcción de una suite de productos.

Discusión:

La aplicación puede reemplazar al por mayor a toda la familia de productos simplemente creando una instancia concreta diferente de la fábrica abstracta.

Debido a que el servicio proporcionado por el objeto de fábrica es tan generalizado, se implementa de forma rutinaria como Singleton.

Estructura: Abstract Factory define un método de fábrica por producto. Cada método de fábrica encapsula al **new** operador y las clases de productos concretas, específicas de la plataforma. Luego, cada "plataforma" se modela con una clase derivada de Factory.

Ejemplo

El propósito de Abstract Factory es proporcionar una interfaz para crear familias de objetos relacionados, sin especificar clases concretas. Este patrón se encuentra en el equipo de estampado de chapa utilizado en la fabricación de automóviles. El equipo de estampado es una fábrica abstracta que crea piezas de carrocería. La misma maquinaria se utiliza para estampar puertas derechas, puertas izquierdas, guardabarros delanteros derechos, guardabarros delanteros izquierdos, capotas, etc. para diferentes modelos de coches.

Código fuente:

```
// class CPU
abstract class CPU {}

// class EmberCPU
class EmberCPU extends CPU {}

// class EnginolaCPU
class EnginolaCPU extends CPU {}


// class MMU
abstract class MMU {}

// class EmberMMU
class EmberMMU extends MMU {}

// class EnginolaMMU
class EnginolaMMU extends MMU {}

// class EmberFactory
class EmberToolkit extends AbstractFactory {
   @Override
   public CPU createCPU() {
      return new EmberCPU();
   }

   @Override
   public MMU createMMU() {
      return new EmberMMU();
   }
}

// class EnginolaFactory
class EnginolaToolkit extends AbstractFactory {
   @Override
   public CPU createCPU() {
      return new EnginolaCPU();
   }
```

```
    @Override
    public MMU createMMU() {
        return new EnginolaMMU();
    }
}
enum Architecture {
    ENGINOLA, EMBER
}
abstract class AbstractFactory {
    private static final EmberToolkit EMBER_TOOLKIT = new
EmberToolkit();
    private static final EnginolaToolkit ENGINOLA_TOOLKIT = new
EnginolaToolkit();

    // Returns a concrete factory object that is an instance of the
    // concrete factory class appropriate for the given architecture.
    static AbstractFactory getFactory(Architecture architecture) {
        AbstractFactory factory = null;
        switch (architecture) {
            case ENGINOLA:
                factory = ENGINOLA_TOOLKIT;
                break;
            case EMBER:
                factory = EMBER_TOOLKIT;
                break;
        }
        return factory;
    }
    public abstract CPU createCPU();

    public abstract MMU createMMU();
}

public class Client {
    public static void main(String[] args) {
        AbstractFactory                    factory                    =
AbstractFactory.getFactory(Architecture.EMBER);
        CPU cpu = factory.createCPU();
    }
}
```

Adapter: Es un patrón de diseño estructural que permite la colaboración entre objetos con interfaces incompatibles.

Convierte la interfaz de una clase en otra interfaz que esperan los clientes. El adaptador permite que las clases trabajen juntas que de otra manera no podrían debido a interfaces incompatibles.
Envuelve una clase existente con una nueva interfaz.

Discusión

La reutilización siempre ha sido dolorosa y difícil de alcanzar. Una razón ha sido la tribulación de diseñar algo nuevo, mientras se reutiliza algo viejo. Siempre hay algo que no está del todo bien entre lo viejo y lo nuevo. Pueden ser dimensiones físicas o desalineación. Puede ser sincronización o sincronización. Pueden ser suposiciones desafortunadas o estándares en competencia.

Es como el problema de insertar un nuevo enchufe eléctrico de tres clavijas en un viejo tomacorriente de pared de dos clavijas: se necesita algún tipo de adaptador o intermediario.

Ejemplo.

El patrón Adaptador permite que las clases que de otro modo serían incompatibles trabajen juntas al convertir la interfaz de una clase en una interfaz esperada por los clientes. Las llaves de tubo proporcionan un ejemplo del adaptador. Un enchufe se conecta a un trinquete, siempre que el tamaño de la unidad sea el mismo. Los tamaños de impulsión típicos son 1/2 "y 1/4". Obviamente, un trinquete de transmisión de 1/2 "no encajará en un dado de transmisión de 1/4" a menos que se utilice un adaptador. Un adaptador de 1/2 "a 1/4" tiene una conexión hembra de 1/2 "para encajar en el trinquete impulsor de 1/2" y una conexión macho de 1/4 "para encajar en el dado de impulsión de 1/4".