Patrón de diseño Método Fábrica

Permite crear objetos sin tener que especificar su clase exacta, permitiendo que el objeto creado puede intercambiarse con flexibilidad y facilidad. Es decir, supongamos que una fábrica produce distintos tipos de vehículos de transporte, por ejemplo: motos y autos, aunque son entidades distintas, comparten algunas características y funcionalidades como Encender, Arrancar, Detenerse, apagar. Etc. Esto permite que la fábrica pueda construirlos a pesar de que sean distintos y que, además implementen las funcionalidades y características que comparten sin necesidad de crear Fábricas independientes.

Ejemplo en código:

Fabrica.java

```
package sprint1;
public class Fabrica {
    public static Transporte construirTransporte(String tipoTransporte){

        switch (tipoTransporte){
            case "Motocicleta":
                return new Motocicleta();
            case "Automovil":
                return new Automovil();
            default:
                System.out.println("No se fabrica ese tipo de transporte en la Fábrica");
                return null;
        }

    }
}
```

Transporte.java

```
package sprint1;

public interface Transporte {

    public void arrancarTransporte();
    public void detenerTransporte();
    public void tipoTransporte();
}
```

Automovil.java

```java
package sprint1;
public class Automovil implements Transporte {
    @Override
    public void arrancarTransporte(){
        System.out.println("El automovil está arrancando");
    }

    @Override
    public void detenerTransporte(){
        System.out.println("El automovil se está deteniendo");
    }

    @Override
    public void tipoTransporte(){
        System.out.println("Automovil");
    }
}
```

Motocicleta.java

```java
package sprint1;
public class Motocicleta implements Transporte {
    @Override
    public void arrancarTransporte(){
        System.out.println("La motocicleta está arrancando");
    }

    @Override
    public void detenerTransporte(){
        System.out.println("La motocicleta se está deteniendo");
    }

    @Override
    public void tipoTransporte(){
        System.out.println("Motocicleta");
    }
}
```

Application.java (MainClass)

```java
package sprint1;
public class Application {
    public static void main(String[] args) {
        proceso("Automovil");
        proceso("Motocicleta");
    }

    public static void proceso(String tipoTransporte){
        Transporte transporte = Fabrica.construirTransporte(tipoTransporte);
        transporte.arrancarTransporte();
        transporte.detenerTransporte();
        transporte.tipoTransporte();
    }

}
```

Prueba

```
run:
El automovil está arrancando
El automovil se está deteniendo
Automovil
La motocicleta está arrancando
La motocicleta se está deteniendo
Motocicleta
```

Patrón de diseño Singleton

Es un patrón de diseño creacional que nos permite asegurarnos de que una clase tenga una única instancia, a la vez que proporciona un punto de acceso global a dicha instancia.

El gobierno es un ejemplo excelente del patrón Singleton. Un país sólo puede tener un gobierno oficial. Independientemente de las identidades personales de los individuos que forman el gobierno, el título “Gobierno de X” es un punto de acceso global que identifica al grupo de personas a cargo, es decir, todo los habitantes tendrán el mismo gobierno.

GobiernoSnglenton.java

```java
package singleton;

public class GobiernoSngleton {
    public String nombreGobierno;
    private static GobiernoSngleton instance = new GobiernoSngleton();

    private GobiernoSngleton(){}

    public static GobiernoSngleton getInstance(){
        return instance;
    }
}
```

Application.java

```java
package singleton;

public class Application {
    public static void main(String[] args) {
        GobiernoSngleton habitanteUno = GobiernoSngleton.getInstance();
        GobiernoSngleton habitanteDos = GobiernoSngleton.getInstance();
        GobiernoSngleton habitanteTres = GobiernoSngleton.getInstance();
        habitanteUno.nombreGobierno = "Gobierno de colombia";

        System.out.println(habitanteUno.nombreGobierno);
        System.out.println(habitanteDos.nombreGobierno);
        System.out.println(habitanteTres.nombreGobierno);
    }

}
```