PATRÓN FACTORÍA

Objetivo

El objetivo de la guía es el de afianzar el conocimiento acerca del patrón Factoría para resolver problemas reales con el uso del lenguaje de programación Java. Además le presentará las diferentes variantes de dicho patrón y los pros y contras de cada una.

Competencias

- Entender el concepto de patrón Factoría
- Utilizar el lenguaje Java para implementar el patrón
- Resolver problemas reales mediante el uso del patrón

Definición

Definición 1

El patrón factoría es uno de los patrones de diseño más utilizados en Java. Este tipo de patrón de diseño está dentro de la categoría de patrones de creación ya que este patrón proporciona una de las mejores maneras de crear un objeto.

En el patrón Factory, creamos el objeto sin exponer la lógica de creación al cliente y nos referimos al objeto recién creado usando una interfaz común.

Definición 2

Un Patrón Factoría o Patrón de Método Factoría dice que sólo se define una interfaz o una clase abstracta para crear un objeto, pero deja que las subclases decidan qué clase instanciar. En otras palabras, las subclases son responsables de crear la instancia de la clase.

El patrón factoría también se conoce como Virtual Constructor.

Ventajas del patrón factoría

- El Patrón Factoría le permite a las subclases elegir el tipo de objetos a crear.
- Promueve el bajo acoplamiento al eliminar la necesidad de vincular clases específicas de aplicación en el código. Esto significa que el código interactúa

únicamente con la interfaz resultante o la clase abstracta, de modo que funcionará con cualquier clase que implemente esa interfaz o que extienda esa clase abstracta.

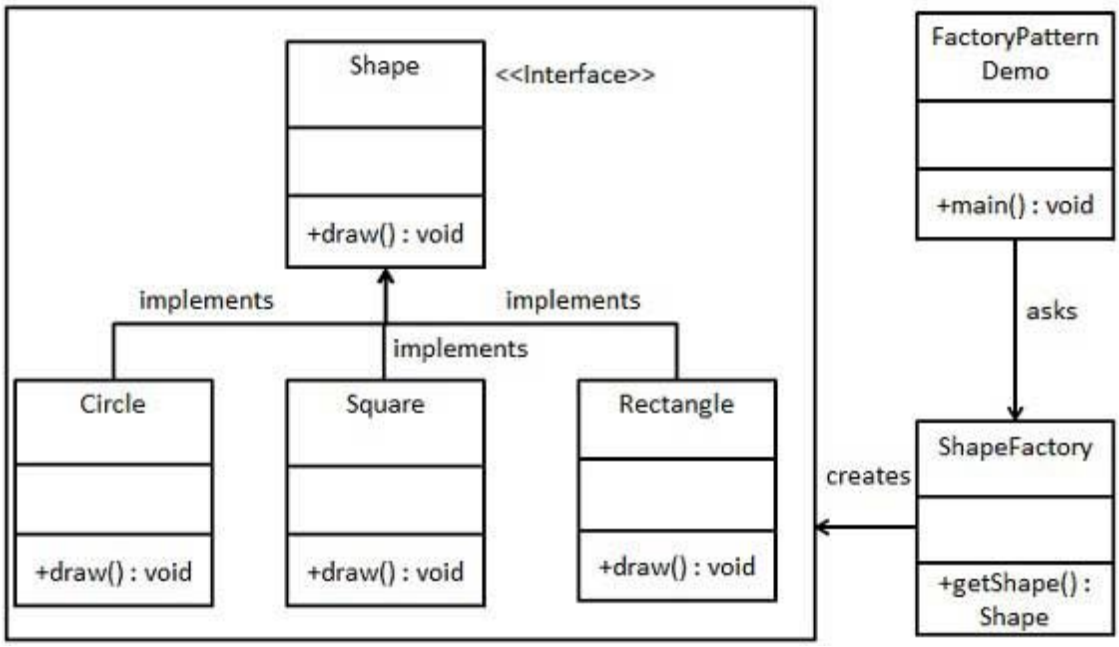
Uso del patrón

- Cuando una clase no sabe qué subclases se requerirán para crear.
- Cuando una clase desea que sus subclases especifiquen los objetos a crear.
- Cuando las clases padre eligen que la creación de objetos la realicen sus subclases.

Implementación de un ejemplo

Vamos a crear una interfaz `Shape` y clases concretas implementando la interfaz `Shape`. Una clase factoría `ShapeFactory` se definirá en el siguiente paso.

La clase de demostración `FactoryPatternDemo`, usará `ShapeFactory` para obtener un objeto `Shape`. Se pasará información (CIRCLE / RECTANGLE / SQUARE) a `ShapeFactory` para obtener el tipo de objeto que se necesita.



Paso 1

Crear una interface.

Shape.java

```
public interface Shape {
   void draw();
}
```

Paso 2

Cree las clases concretas que implementan la misma interfaz.

Rectangle.java

```
public class Rectangle implements Shape {

   @Override
   public void draw() {
      System.out.println("Inside Rectangle::draw() method.");
   }
}
```

Square.java

```
public class Square implements Shape {

   @Override
   public void draw() {
      System.out.println("Inside Square::draw() method.");
   }
}
```

Circle.java

```
public class Circle implements Shape {

   @Override
   public void draw() {
      System.out.println("Inside Circle::draw() method.");
   }
}
```

Paso 3

Cree una factoría para generar un objeto de una clase concreta basada sobre la información pasada como parámetro.

ShapeFactory.java

```
public class ShapeFactory {

   //use getShape method to get object of type shape
   public Shape getShape(String shapeType){
      if(shapeType == null){
         return null;
      }
      if(shapeType.equalsIgnoreCase("CIRCLE")){
```

```
         return new Circle();

      } else if(shapeType.equalsIgnoreCase("RECTANGLE")){
         return new Rectangle();

      } else if(shapeType.equalsIgnoreCase("SQUARE")){
         return new Square();
      }

      return null;
   }
}
```

Paso 4

Use la factoría para obtener el objeto de una clase concreta pasando información como un tipo.

FactoryPatternDemo.java

```
public class FactoryPatternDemo {

   public static void main(String[] args) {
      ShapeFactory shapeFactory = new ShapeFactory();

      //get an object of Circle and call its draw method.
      Shape shape1 = shapeFactory.getShape("CIRCLE");

      //call draw method of Circle
      shape1.draw();

      //get an object of Rectangle and call its draw method.
      Shape shape2 = shapeFactory.getShape("RECTANGLE");

      //call draw method of Rectangle
      shape2.draw();

      //get an object of Square and call its draw method.
      Shape shape3 = shapeFactory.getShape("SQUARE");

      //call draw method of circle
      shape3.draw();
   }
}
```

Paso 5

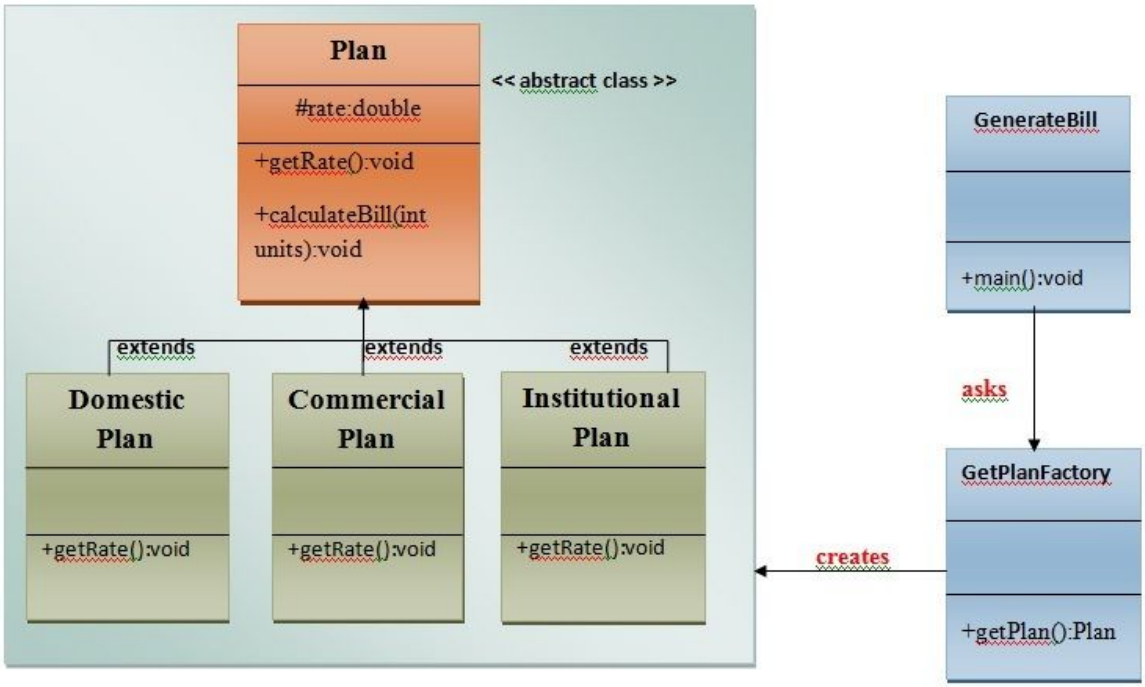
Verifique la salida.

```
Inside Circle::draw() method.
Inside Rectangle::draw() method.
Inside Square::draw() method.
```

Otro UML para el patrón Factoría

Vamos a crear una clase abstracta llamada `Plan` y clases concretas que extiendan la clase abstracta `Plan`. Luego definimos una clase factoría `GetPlanFactory`.

La clase `GenerateBill` utilizará `GetPlanFactory` para obtener un objeto `Plan`. Pasará la información (PLASTICPLAN / COMMERCIALPLAN / INSTITUTIONALPLAN) a `GetPlanFactory` para obtener el tipo de objeto que necesita.



Calcular factura de electricidad: Un ejemplo real de método factoría

Paso 1

Crear una clase abstracta Plan

Plan.java

```
import java.io.*;
abstract class Plan{
         protected double rate;
         abstract void getRate();

         public void calculateBill(int units){
              System.out.println(units*rate);
```

```
        }
}
```

Paso 2

Crear las clases concretas que extienden la clase abstracta Plan.

DomesticPlan.java

```
public class  DomesticPlan extends Plan{
        //@override
         public void getRate(){
             rate=3.50;
        }
}//end of DomesticPlan class.
```

CommercialPlan.java

```
public class CommercialPlan extends Plan{
   //@override
    public void getRate(){
        rate=7.50;
   }
}
```

InstitutionalPlan.java

```
public class  InstitutionalPlan extends Plan{
   //@override
    public void getRate(){
        rate=5.50;
   }
}
```

Paso 3

Crear una clase GetPlanFactory para generar objetos de clases concretas basados en información dada.

GetPlanFactory.java

```
public class GetPlanFactory{

       //use getPlan method to get object of type Plan
       public Plan getPlan(String planType){
            if(planType == null){
             return null;
            }
          if(planType.equalsIgnoreCase("DOMESTICPLAN")) {
                return new DomesticPlan();
```

```
            }
         else if(planType.equalsIgnoreCase("COMMERCIALPLAN")){
              return new CommercialPlan();
          }
        else if(planType.equalsIgnoreCase("INSTITUTIONALPLAN")) {
              return new InstitutionalPlan();
        }
      return null;
   }
}//end of GetPlanFactory class.
```

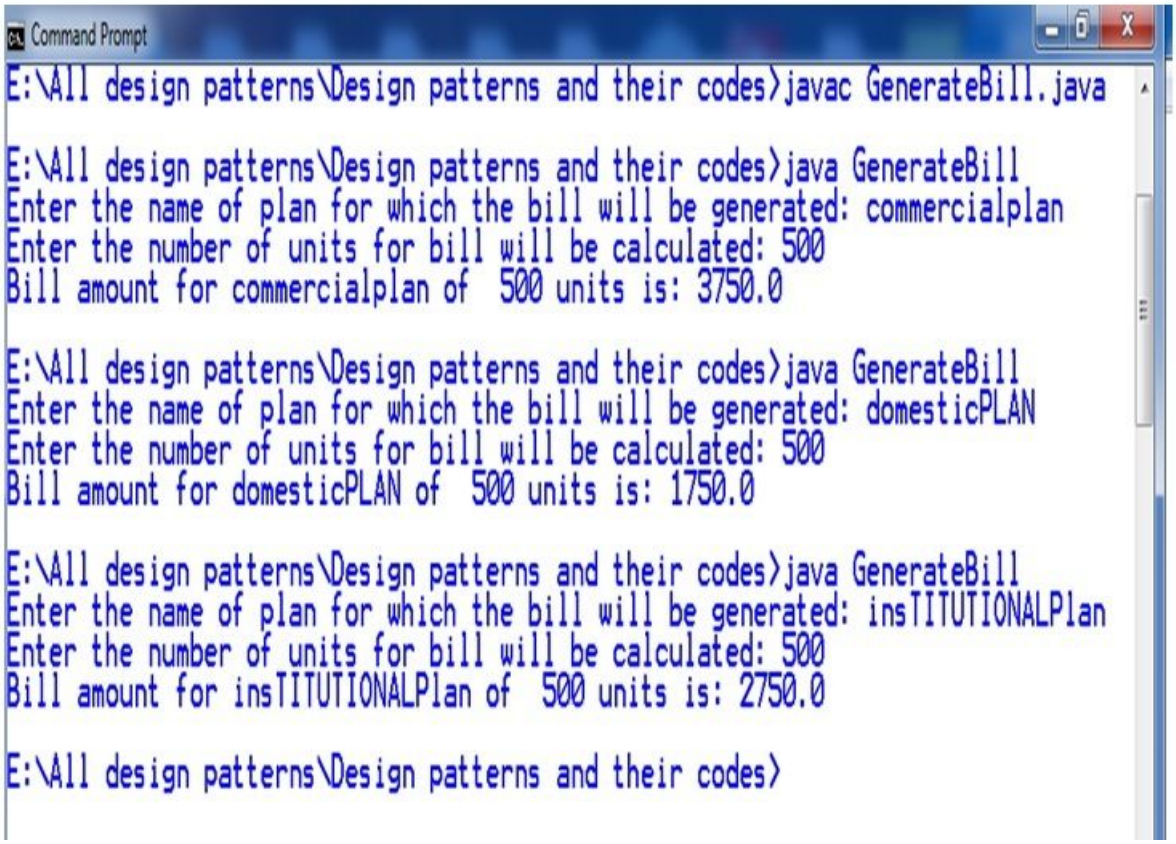
Paso 4

Generar Bill usando GetPlanFactory para obtener el objeto de una clase concreta pasando una información como el tipo de plan DOMESTICPLAN ó COMMERCIALPLAN ó INSTITUTIONALPLAN.

GenerateBill.java

```
import java.io.*;
public class GenerateBill {
    public static void main(String args[])throws IOException{
      GetPlanFactory planFactory = new GetPlanFactory();

      System.out.print("Enter the name of plan for which the bill will be
generated: ");
      BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

      String planName=br.readLine();
      System.out.print("Enter the number of units for bill will be calculated: ");
      int units=Integer.parseInt(br.readLine());

      Plan p = planFactory.getPlan(planName);
      //call getRate() method and calculateBill()method of DomesticPaln.

       System.out.print("Bill amount for "+planName+" of  "+units+" units is: ");
           p.getRate();
           p.calculateBill(units);
    }
}//end of GenerateBill class.
```

Salida



Lecturas recomendadas

- Patrón factoría https://www.tutorialspoint.com/design_pattern/factory_pattern.htm
- Patrón Factoría http://www.javatpoint.com/factory-method-design-pattern

Ejercicios

Requisitos:

Instale mysql server y mysql server workbench https://dev.mysql.com/downloads/mysql/
Instale la base de datos de prueba llamada sakila.
https://dev.mysql.com/doc/sakila/en/sakila-installation.html
Obtenga la consulta Find Overdue DVD de los ejemplos de Sakila
https://dev.mysql.com/doc/sakila/en/sakila-usage.html

Pasos a seguir

1. En netbeans cree un nuevo proyecto de tipo Java Web -> Web Application. El nombre de la aplicación será Sakila1. Ubique el directorio donde colocará su proyecto y no utilice ningún framework.
2. Una vez creado el proyecto proceda a crear una clase llamada “Conexion” la cual debe instanciar sólo un objeto de tipo conexión el cual se utilizará a lo largo del proyecto para realizar operaciones con la base de datos Sakila.
3. Luego cree una página denominada index.jsp donde obtenga la conexión y ejecute la consulta “Find Overdue DVD” de los ejemplos de sakila para obtener los datos y presentarlos en la pantalla.

Bibliografía

1. Java Singleton Design Pattern Best Practices with Examples
http://www.journaldev.com/1377/java-singleton-design-pattern-best-practices-examples
2. Simply Singleton
http://www.javaworld.com/article/2073352/core-java/simply-singleton.html