* What is design pattern in java?

A design pattern provides a general reusable solution for a common problem occurs in software design. The patterns typically show relationships and interactions between classes and objects. Design pattern are programming languages independent strategies for solving a common problem. By using the design patterns, we can make our code more flexible, reusable and maintainable.

* What is Gang of Four (GOF)?

In 1994, four authors published a book titled **Design Patterns - Elements of Reusable Object-Oriented Software** which initiated the concept of Design Pattern in Software development. These authors are collectively known as Gang of fours(GOF). According to them the principles of object oriented design:

1. Program to an interface not its implementation
2. Favor object composition over inheritance.

* Types of design pattern?

There are mainly three types of design patterns:

1. Creational -> These design patterns are all about class instantiation or object creation.
2. Structural -> These design patterns are about organizing different classes and objects to form larger structures and provide new functionality.
3. Behavioral -> Behavioral patterns are about identifying common communication patterns between objects and realize these patterns.

* What are the types of creational design pattern?

1. Singleton Pattern
2. Factory Method pattern
3. Abstract factory pattern
4. Prototype pattern
5. Builder pattern
6. Object pool pattern

* What are the types of Structural design pattern?

1. Adaptor Pattern
2. Bridge pattern
3. Composite pattern
4. Decorator pattern
5. Façade pattern
6. Flyweight pattern
7. Proxy pattern

* What are the types of Behavioral pattern?

1. Chain of responsibility pattern
2. Command pattern
3. Interpreter pattern
4. Iterator pattern
5. Mediator pattern
6. Memento Pattern
7. Observer pattern
8. State Pattern
9. Strategy pattern
10. Template pattern
11. Visitor pattern
12. Null Object

* What is Factory method pattern?

A Factory pattern or a Factory method pattern says that just define an interface or abstract class for creating an object but let the sub classes decide which class to instantiate. In other word, subclasses are responsible for creating the instance of the class.

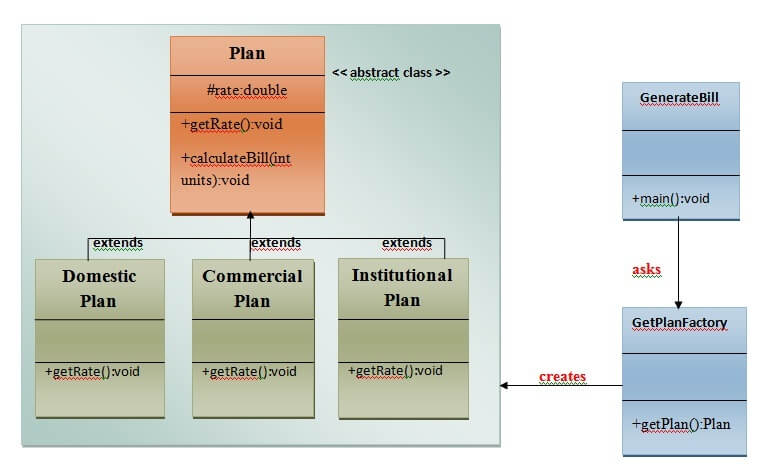
Advantage of factory design patter:

1. Factory method patter allows the subclasses to choose the type of objects to create.
2. It promotes the loose-coupling by eliminating the need to bind application-specific classes into the code.

Usage of factory design pattern

* When a class doesn't know what sub-classes will be required to create
* When a class wants that its sub-classes specify the objects to be created.
* When the parent classes choose the creation of objects to its sub-classes.

Example:



import java.io.\*;

abstract class Plan{

protected double rate;

abstract void getRate();

public void calculateBill(int units){

System.out.println(units\*rate);

}

}

class DomesticPlan extends Plan{

//@override

public void getRate(){

rate=3.50;

}

}

class CommercialPlan extends Plan{

//@override

public void getRate(){

rate=7.50;

}

class InstitutionalPlan extends Plan{

//@override

public void getRate(){

rate=5.50;

}

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;

}

}

import java.io.\*;

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);

}

}