

Factory Method Pattern

A Factory Pattern or Factory Method Pattern says that just **define an interface or abstract class for creating an object but let the subclasses decide which class to instantiate**. In other words, subclasses are responsible to create the instance of the class.

The Factory Method Pattern is also known as **Virtual Constructor**.

Advantage of Factory Design Pattern

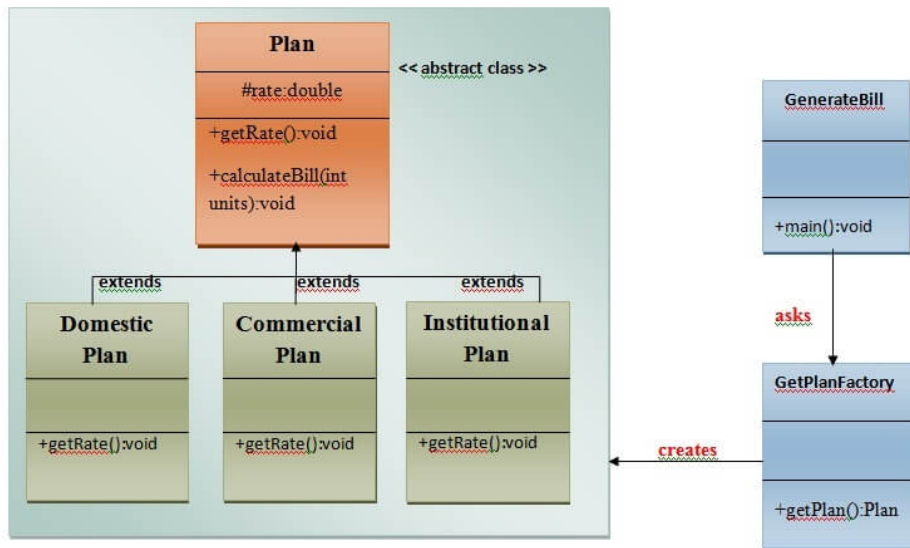
- Factory Method Pattern allows the sub-classes to choose the type of objects to create.
- It promotes the **loose-coupling** by eliminating the need to bind application-specific classes into the code. That means the code interacts solely with the resultant interface or abstract class, so that it will work with any classes that implement that interface or that extends that abstract class.

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.

UML for Factory Method Pattern

- We are going to create a Plan abstract class and concrete classes that extends the Plan abstract class. A factory class GetPlanFactory is defined as a next step.
- GenerateBill class will use GetPlanFactory to get a Plan object. It will pass information (DOMESTICPLAN / COMMERCIALPLAN / INSTITUTIONALPLAN) to GetPlanFactory to get the type of object it needs.



accenture

**HERE, YOUR PRIORIT
ARE OURS.**

EQUALITY=INNOVATION



Calculate Electricity Bill : A Real World Example of Factory Method

Step 1: Create a Plan abstract class.

```

import java.io.*;

abstract class Plan{
    protected double rate;
    abstract void getRate();
}
  
```



```
    public void calculateBill(int units){
        System.out.println(units*rate);
    }
} //end of Plan class.
```

Step 2: Create the concrete classes that extends Plan abstract class.

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

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

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

Step 3: Create a GetPlanFactory to generate object of concrete classes based on given information..

```
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.
```

Step 4: Generate Bill by using the GetPlanFactory to get the object of concrete classes by passing an information such as type of plan DOMESTICPLAN or COMMERCIALPLAN or INSTITUTIONALPLAN.

```
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);
    }
} //end of GenerateBill class.
```

download this Electricity bill Example

Output

```
Command Prompt
E:\All design patterns\Design patterns and their codes>javac GenerateBill.java

E:\All design patterns\Design patterns and their codes>java GenerateBill
Enter the name of plan for which the bill will be generated: commercialplan
Enter the number of units for bill will be calculated: 500
Bill amount for commercialplan of 500 units is: 3750.0

E:\All design patterns\Design patterns and their codes>java GenerateBill
Enter the name of plan for which the bill will be generated: domesticPLAN
Enter the number of units for bill will be calculated: 500
Bill amount for domesticPLAN of 500 units is: 1750.0

E:\All design patterns\Design patterns and their codes>java GenerateBill
Enter the name of plan for which the bill will be generated: instITUTIONALPlan
Enter the number of units for bill will be calculated: 500
Bill amount for instITUTIONALPlan of 500 units is: 2750.0

E:\All design patterns\Design patterns and their codes>
```

[< prev](#)[next >](#)

Help Others, Please Share















Join Javatpoint Test Series






Placement Papers	AMCAT	Bank PO/Clerk	GATE
TCS	eLitmas	UPSSSC	NEET
HCL	Java	Government	CAT
Infosys	Python	Exams	Railway
IBM	C Programming	SSC	CTET
Accenture	Networking	Civil Services	IIT JEE
		SBI	









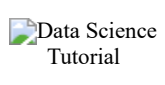
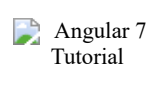
Learn Latest Tutorials

			
Ansible	Mockito	Talend	Azure
			
SharePoint	Powershell	Kali Linux	OpenCV
			
Kafka	Pandas	Joomla	Reinforcement





Preparation

			
Aptitude	Reasoning	Verbal A.	Interview
			
Company			























Trending Technologies

			
AI	AWS	Selenium	Cloud
			
Hadoop	ReactJS	D. Science	Angular 7



 Blockchain Tutorial Blockchain	 Git Tutorial Git	 Machine Learning Tutorial ML	 DevOps Tutorial DevOps
---	---	--	---

B.Tech / MCA

 DBMS tutorial DBMS	 Data Structures tutorial DS	 DAA tutorial DAA	 Operating System tutorial OS
 Computer Network tutorial C. Network	 Compiler Design tutorial Compiler D.	 Computer Organization and Architecture COA	 Discrete Mathematics Tutorial D. Math.
 Ethical Hacking Tutorial E. Hacking	 Computer Graphics Tutorial C. Graphics	 Software Engineering Tutorial Software E.	 html tutorial Web Tech.
 Cyber Security tutorial Cyber Sec.	 Automata Tutorial Automata	 C Language tutorial C	 C++ tutorial C++
 Java tutorial Java	 .Net Framework tutorial .Net	 Python tutorial Python	 List of Programs Programs
 Control Systems tutorial Control S.	 Data Mining Tutorial Data Mining		

