DESIGN PATTERNS IN JAVA

1.What is design patterns in Java?

A design patterns are **well-proved solution** for solving the specific problem/task. Every design pattern has **some specification or set of rules** for solving the problems. Design patterns are programming language independent strategies for solving the common object-oriented design problems. That means, a design pattern represents an idea, not a particular implementation.

By using the design patterns you can make your code more flexible, reusable and maintainable. It is the most important part because java internally follows design patterns.

2. Types of design patterns

a) Creational design patterns are concerned with**the way of creating objects.** These design patterns are used when a decision must be made at the time of instantiation of a class (i.e. creating an object of a class).

There are following 6 types of creational design patterns.

1. [Factory Method Pattern](https://www.javatpoint.com/factory-method-design-pattern)
2. [Abstract Factory Pattern](https://www.javatpoint.com/abstract-factory-pattern)
3. [Singleton Pattern](https://www.javatpoint.com/singleton-design-pattern-in-java)
4. [Prototype Pattern](https://www.javatpoint.com/prototype-design-pattern)
5. [Builder Pattern](https://www.javatpoint.com/builder-design-pattern)
6. [Object Pool Pattern](https://www.javatpoint.com/object-pool-pattern)

b)Behavioral Design Pattern

Behavioral design patterns are concerned with **the interaction and responsibility of objects.**

In these design patterns,**the interaction between the objects should be in such a way that they can easily talk to each other and still should be loosely coupled.**

3. Explain Factory method pattern with example

Factory method pattern defines an interface or abstract class for creating an object but let the subclasses decide which class to instantiate. In this subclasses are responsible to create the instance of the class. The Factory Method Pattern is also known as **Virtual Constructor.**

**Advantages:**

* It allows the sub-classes to choose the type of objects to create.
* It will work with any classes that implement that interface or that extends that abstract class.

Example: Calculation of Electricity Bill

* create a Plan abstract class and concrete classes that extends the Plan abstract class. A factory class GetPlanFactory is defined.
* 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.

**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);

          }

}

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

**class**  DomesticPlan **extends** Plan{

**public** **void** getRate(){

             rate=3.50;

        }

   }

**class**  CommercialPlan **extends** Plan{

**public** **void** getRate(){

        rate=7.50;

   }

**class**  InstitutionalPlan **extends** Plan{

**public** **void** getRate(){

        rate=5.50;

   }

**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**;

   }

}

**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: ");

      Scanner sc=new Scanner(System.in);

      String planName=sc.next();

      System.out.print("Enter the number of units for bill will be calculated: ");

**int** units=sc.nextInt();

      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);

            }

}