Factory Design Pattern

13 November 2021 19:53

- The factory design pattern says that define an interface (A java interface or an abstract class) and let the subclasses decide which object to instantiate. The factory method in the interface lets a class defer the instantiation to one or more concrete subclasses

- In Factory pattern, we create objects without exposing the creation logic to the client and the client uses the same common interface to create a new type of object.

- In Factory pattern, we create object without exposing the creation logic to the client and refer to newly created object using a common interface.

- 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**

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

Below is the Application in which we use Factory Design Pattern
Whenever we develop an app, we do have different type of notifications, like SMS Notification, Email Notification and Push Notification.
Here Factory will decide which class to be instantiated.

```
public interface Notification {
    void notifyUser();
}
```

```
public class SMSNotification implements Notification {

    @Override
    public void notifyUser()
    {
        // TODO Auto-generated method stub
        System.out.println("Sending an SMS notification");
    }
}

public class EmailNotification implements Notification {

    @Override
    public void notifyUser()
```

```
    {
        // TODO Auto-generated method stub
        System.out.println("Sending an e-mail notification");
    }
}

public class NotificationFactory {
    public Notification createNotification(String channel)
    {
        if (channel == null || channel.isEmpty())
            return null;
        if ("SMS".equals(channel)) {
            return new SMSNotification();
        }
        else if ("EMAIL".equals(channel)) {
            return new EmailNotification();
        }
        else if ("PUSH".equals(channel)) {
            return new PushNotification();
        }
        return null;
    }
}

public class NotificationService {
    public static void main(String[] args)
    {
        NotificationFactory notificationFactory = new
NotificationFactory();
        Notification notification =
notificationFactory.createNotification("SMS");
        notification.notifyUser();
    }
}
```

➢ Real time Uses of Factory Design Pattern:

- ➢ getInstance() method of java.util.Calendar, NumberFormat, and ResourceBundle uses factory method design pattern.

- ➢ Consider an example of using multiple database servers like SQL Server and Oracle. If you are developing an application using SQL Server database as backend, but in future need to change backend database to oracle, you will need to modify all your code, if you haven't written your code following factory design pattern.
- ➢ In factory design pattern you need to do very little work to achieve this. A class implementing factory design pattern takes care for you and lessen your burden. Switching from database server won't bother you at all. You just need to make some small changes in your configuration file. (In Factory class pick the configuration file or data base connection based on the type of database then, in future there is no need to modify the change whenever data base is switched)