**Design Patterns and Principles**

**EXERCISE – 2 : Implementing the Factory Method Pattern**

**SCENARIO :**

In a messaging app, users can choose to receive notifications via Email, SMS, or Push depending on their preference. To handle this dynamically without changing code repeatedly, a factory method is used to create the appropriate notification type based on user input.

**SOURCE CODE :**

interface Notification {

void notifyUser();

}

class EmailNotification implements Notification {

public void notifyUser() {

System.out.println("Sending an Email Notification");

}

}

class SMSNotification implements Notification {

public void notifyUser() {

System.out.println("Sending an SMS Notification");

}

}

class PushNotification implements Notification {

public void notifyUser() {

System.out.println("Sending a Push Notification");

}

}

class NotificationFactory {

public Notification createNotification(String type) {

if (type == null || type.isEmpty())

return null;

switch (type.toLowerCase()) {

case "email":

return new EmailNotification();

case "sms":

return new SMSNotification();

case "push":

return new PushNotification();

default:

return null;

}

}

}

public class Main {

public static void main(String[] args) {

NotificationFactory factory = new NotificationFactory();

Notification n1 = factory.createNotification("email");

n1.notifyUser();

Notification n2 = factory.createNotification("sms");

n2.notifyUser();

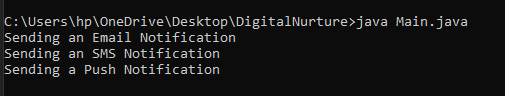
Notification n3 = factory.createNotification("push");

n3.notifyUser();

}

}

**OUTPUT :**

****