Interface + POO -> Gestion d’un système de notifications

On va combiner :

- **POO (héritage, redéfinition, encapsulation)**
- **Interface (contrat de comportement, polymorphisme)**
- **Et un petit scénario métier concret**

💼

Exercice — Interface + POO : Gestion d’un système de notifications

🎯 Objectif :

Modéliser un système capable d’envoyer des **notifications** à des **utilisateurs**,

en utilisant **des interfaces** pour rendre le code flexible et extensible.

🧩

Énoncé :

Tu veux créer un programme qui peut **notifier des utilisateurs** de différentes manières (Email, SMS, Push, etc.).

Chaque **utilisateur** a un **nom** et un **type de notification préféré**.

Tu dois modéliser :

- une **interface Notifier** (le contrat pour envoyer une notification)
- plusieurs **implémentations concrètes** (EmailNotifier, SmsNotifier, PushNotifier)
- une **classe User** qui contient le nom et le Notifier
- une **classe NotificationService** qui gère l’envoi de messages à tous les utilisateurs.

📦 UML — Vue d’ensemble

```
classDiagram
    class Notifier {
        <<interface>>
        +sendNotification(String message)
    }

    class EmailNotifier {
        +sendNotification(String message)
    }

    class SmsNotifier {
        +sendNotification(String message)
    }

    class PushNotifier {
        +sendNotification(String message)
    }

    Notifier <|.. EmailNotifier
    Notifier <|.. SmsNotifier
    Notifier <|.. PushNotifier

    class User {
        -String name
        -Notifier notifier
        +User(String name, Notifier notifier)
        +notify(String message)
    }

    class NotificationService {
        -List<User> users
        +addUser(User user)
        +notifyAll(String message)
    }

    User --> Notifier
    NotificationService --> User
```

🧠

Réponses

Étape 1 — L'interface

```
public interface Notifier {
    void sendNotification(String message);
}
```

Étape 2 — Les implémentations concrètes

```
public class EmailNotifier implements Notifier {
    @Override
    public void sendNotification(String message) {
        System.out.println("✉️ Email sent: " + message);
    }
}

public class SmsNotifier implements Notifier {
    @Override
    public void sendNotification(String message) {
        System.out.println("📱 SMS sent: " + message);
    }
}

public class PushNotifier implements Notifier {
    @Override
    public void sendNotification(String message) {
        System.out.println("🔔 Push notification: " + message);
    }
}
```

Étape 3 — La classe

User

```
public class User {
    private String name;
    private Notifier notifier; // interface = contrat, pas dépendance forte
```

```
    public User(String name, Notifier notifier) {
        this.name = name;
        this.notifier = notifier;
    }

    public void notify(String message) {
        System.out.println("To " + name + ":");
        notifier.sendNotification(message);
    }
}
```

Étape 4 — Le service

```
import java.util.ArrayList;
import java.util.List;

public class NotificationService {
    private List<User> users = new ArrayList<>();

    public void addUser(User user) {
        users.add(user);
    }

    public void notifyAll(String message) {
        for (User user : users) {
            user.notify(message);
        }
    }
}
```

Étape 5 — Le Main

```
public class Main {
    public static void main(String[] args) {
        User vincent = new User("Vincent", new EmailNotifier());
        User isabelle = new User("Isabelle", new SmsNotifier());
        User tom = new User("Tom", new PushNotifier());

        NotificationService service = new NotificationService();
        service.addUser(vincent);
```

```
        service.addUser(isabelle);
        service.addUser(tom);

        service.notifyAll("⚡ System maintenance tonight at 10 PM!");
    }
}
```

🧾 Exemple de sortie console

```
To Vincent:
✉️ Email sent: ⚡ System maintenance tonight at 10 PM!

To Isabelle:
📱 SMS sent: ⚡ System maintenance tonight at 10 PM!

To Tom:
🔔 Push notification: ⚡ System maintenance tonight at 10 PM!
```

🧠 Concepts travaillés

Concept	Exemple	Rôle
Interface	Notifier	définit un contrat d'envoi de notification
Implémentation	EmailNotifier, SmsNotifier, PushNotifier	concrétise l'interface
Polymorphisme	User appelle notifier.sendNotification() sans savoir lequel	flexibilité
Composition	User possède un Notifier	lien faible (injection de dépendance)
Encapsulation	User cache son Notifier derrière la méthode notify()	propreté
Extensibilité	ajouter SlackNotifier ou TeamsNotifier sans changer le code existant	principe **OCP**

💬 Pour aller plus loin :

- Ajoute une interface Loggable avec une méthode logAction(String message) et fais implémenter cette interface à NotificationService.
- Crée une version DatabaseNotifier qui enregistre les notifications au lieu de les afficher.
- Teste le polymorphisme avec une List.