المدرسة الحسنية للأشغال العمومية
ECOLE HASSANIA DES TRAVAUX PUBLICS

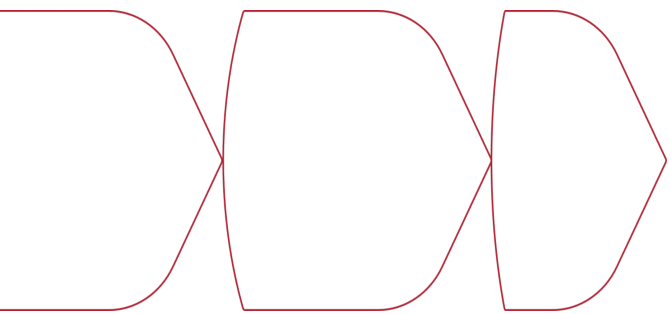
Introduction aux Design Patterns

Comprendre les modèles de conception en programmation orientée objet

Slimani Mohamed Amine

EHTP

February 12, 2025



Sommaire

Qu'est-ce qu'un Design Pattern ?

- **Définition** : Un design pattern est une solution générique et réutilisable à un problème récurrent de conception logicielle.
- **Objectif** : Fournir une structure de code éprouvée pour résoudre des problèmes courants.
- **Avantages** : Réutilisabilité, maintenabilité, et meilleure communication entre développeurs.

Pourquoi utiliser des Design Patterns ?

- **Réutilisabilité** : Éviter de réinventer la roue pour des problèmes courants.
- **Maintenabilité** : Faciliter la compréhension et la modification du code.
- **Communication** : Utiliser un vocabulaire commun pour décrire des solutions.

Catégories de Design Patterns

- **Créationnels** : Concernent la création d'objets (ex. Singleton, Factory).
- **Structurels** : Concernent la composition d'objets (ex. Adapter, Decorator).
- **Comportementaux** : Concernent l'interaction entre objets (ex. Observer, Strategy).

Exemples de Design Patterns

- **Singleton** : Garantit qu'une classe n'a qu'une seule instance.
- **Factory** : Fournit une interface pour créer des objets.
- **Observer** : Permet à des objets de s'abonner à des événements.

Exemple de code : Singleton

Implémentation du Singleton en Java

```
Comment Code
public class Singleton {
    private static Singleton instance;

    private Singleton() {}

    public static Singleton getInstance() {
        if (instance == null) {
            instance = new Singleton();
        }
        return instance;
    }

    Run | Debug
    public static void main(String[] args) {
        // Utilisation
        Singleton singleton1 = Singleton.getInstance();
        Singleton singleton2 = Singleton.getInstance();
        System.out.println(singleton1 == singleton2); // true
    }
}
```

Exemple de code : Factory

Implémentation du Factory en Java

```java
interface Product {
    void use();
}

Comment Code
class ConcreteProduct implements Product {
    public void use() {
        System.out.println(x:"Using ConcreteProduct");
    }
}

Comment Code
class ProductFactory {
    public static Product createProduct() {
        return new ConcreteProduct();
    }
}

Comment Code
public class Main {
    Run | Debug
    public static void main(String[] args) {
        // Utilisation
        Product product = ProductFactory.createProduct();
        product.use();
    }
}
```

Exemple de code : Observer

Implémentation de l'Observer en Java

```java
interface Observer {
    void update(String message);
}
Comment Code
class ConcreteObserver implements Observer {
    public void update(String message) {
        System.out.println("Received: " + message);
    }
}
Comment Code
public class Subject {
    private List<Observer> observers = new ArrayList<>();

    public void addObserver(Observer observer) {
        observers.add(observer);
    }

    public void notifyObservers(String message) {
        for (Observer observer : observers) {
            observer.update(message);
        }
    }
    Run | Debug
    public static void main(String[] args) {
        // Utilisation
        Subject subject = new Subject();
        subject.addObserver(new ConcreteObserver());
        subject.notifyObservers(message:"Hello, Observers!");
    }
}
```

Bonnes pratiques

- **Choix du pattern** : Utiliser le bon pattern pour le bon problème.
- **Simplicité** : Ne pas surcharger le code avec des patterns inutiles.
- **Documentation** : Bien documenter l'utilisation des patterns.

Outils pour travailler avec les Design Patterns

- **UML** : Utiliser des diagrammes UML pour visualiser les patterns.
- **IDE** : Utiliser des IDE modernes pour faciliter l'implémentation.
- **Livres** : Se référer à des ouvrages comme "Design Patterns: Elements of Reusable Object-Oriented Software".

Défis des Design Patterns

- **Complexité** : Certains patterns peuvent rendre le code plus complexe.
- **Surcharge** : Utiliser trop de patterns peut rendre le code difficile à comprendre.
- **Adaptation** : Adapter les patterns à des contextes spécifiques peut être difficile.

Pourquoi c'est important ?

- Les design patterns sont essentiels pour écrire du code robuste et maintenable.
- Ils permettent de résoudre des problèmes courants de manière efficace.
- Comprendre les design patterns est crucial pour les développeurs expérimentés.

Résumé

Les Design Patterns sont des outils puissants pour améliorer la qualité du code. Explorez, apprenez, et appliquez-les dans vos projets !