Design Patterns

Abstract Factory*, *Factory Method*, *Observer*, *Composite*, *Decorator*, *et* *Façade

Contexte : Gestion d'une bibliothèque multimédia

Vous développez un système de gestion pour une bibliothèque multimédia qui permet aux utilisateurs d'accéder à différents types de contenus : **livres**, **films** et **musiques**. Le système doit répondre aux exigences suivantes :

1. **Abstract Factory** : Fournir une manière standardisée de créer différents types de contenus selon le support (numérique ou physique).
2. **Factory Method** : Permettre à chaque contenu (livre, film, musique) de déterminer la manière spécifique de se créer.
3. **Observer** : Notifier les utilisateurs des nouvelles sorties dans des catégories spécifiques (livres, films ou musiques).
4. **Composite** : Permettre la gestion de collections contenant plusieurs contenus (par exemple, une collection de livres).
5. **Decorator** : Ajouter dynamiquement des fonctionnalités aux contenus, comme la mise en avant d'un contenu populaire ou l'ajout de notes/avis.
6. **Facade** : Fournir une interface simplifiée pour rechercher des contenus et gérer les abonnements des utilisateurs.

Étape 1 : Implémentation des Design Patterns

1. Abstract Factory

Créez une interface `ContentFactory` avec deux implémentations :

- **PhysicalContentFactory** : Crée des livres, films ou musiques physiques.
- **DigitalContentFactory** : Crée des contenus numériques.

Chaque type de contenu (livre, film, musique) est une classe dérivée de `Content`.

2. Factory Method

Chaque classe de contenu (ex. `Book`, `Movie`, `Music`) doit avoir sa propre méthode pour se créer, en utilisant un pattern Factory Method.

3. Observer

Mettez en place un système de notification où les utilisateurs s'abonnent à des catégories (livres, films, musiques). Lorsqu'un nouveau contenu est ajouté, les utilisateurs abonnés doivent être notifiés.

4. Composite

Créez une structure pour gérer des collections de contenus :

- Une collection peut contenir des livres, des films, et/ou des musiques.
- Une collection peut être imbriquée dans une autre collection.

5. Decorator

Ajoutez dynamiquement des fonctionnalités aux contenus :

- Par exemple, ajouter une fonctionnalité pour mettre en avant un contenu ou permettre aux utilisateurs de laisser des commentaires.

6. Facade

Implémentez une classe `LibraryFacade` qui fournit une interface simplifiée pour :

- Ajouter un contenu.
- Gérer les abonnements.
- Rechercher un contenu.

Étape 2 : Cahier des charges

Classes principales à implémenter :

1. **Abstract Factory** :
   - `ContentFactory` (interface)
   - `PhysicalContentFactory` (implémentation)
   - `DigitalContentFactory` (implémentation)
2. **Factory Method** :
   - `Content` (classe de base)
   - `Book`, `Movie`, `Music` (sous-classes)
3. **Observer** :
   - `User` (classe observateur)
   - `Category` (classe observable : livres, films, musiques)
4. **Composite** :
   - `ContentCollection` : peut contenir des objets de type `Content` ou d'autres collections.
5. **Decorator** :
   - `ContentDecorator` (classe abstraite)
   - `PopularContentDecorator`, `CommentableContentDecorator` (implémentations concrètes)
6. **Facade** :
   - `LibraryFacade` : fournit des méthodes comme `addContent()`, `subscribeUser()`, `searchContent()`.

Étape 3 : Instructions

Partie 1 : Code

- Implémentez le code des classes en respectant les patterns décrits.
- Testez chaque pattern indépendamment.

Partie 2 : Scénarios

1. **Abstract Factory** : Créez une liste de contenus physiques et numériques.
2. **Observer** : Ajoutez un contenu et notifiez les utilisateurs abonnés.
3. **Composite** : Créez une collection qui contient plusieurs types de contenus.
4. **Decorator** : Ajoutez des fonctionnalités comme les avis sur certains contenus.
5. **Facade** : Effectuez une recherche et gérez les abonnements via la façade.

Solution pour Abstract Factory

Interface `ContentFactory`

```
public interface ContentFactory {
    Book createBook(String title, String author);
    Movie createMovie(String title, String director);
    Music createMusic(String title, String artist);
}
```

Implémentation de `DigitalContentFactory`

```
public class DigitalContentFactory implements ContentFactory {
    @Override
    public Book createBook(String title, String author) {
        return new DigitalBook(title, author);
    }

    @Override
    public Movie createMovie(String title, String director) {
        return new DigitalMovie(title, director);
    }

    @Override
    public Music createMusic(String title, String artist) {
        return new DigitalMusic(title, artist);
    }
}
```

Classe abstraite `Content`

```
public abstract class Content {
    protected String title;
```

```
    public Content(String title) {
        this.title = title;
    }

    public String getTitle() {
        return title;
    }

    public abstract void display();
}
```

Classe concrète `DigitalBook`

```
public class DigitalBook extends Content {
    private String author;

    public DigitalBook(String title, String author) {
        super(title);
        this.author = author;
    }

    @Override
    public void display() {
        System.out.println("Digital Book: " + title + " by " + author);
    }
}
```