Design Patterns In Java

Design Patterns:

Les modèles de conception représentent les meilleures pratiques utilisées par les développeurs de logiciels expérimentés orientés objet.Les modèles de conception sont des solutions aux problèmes généraux rencontrés par les développeurs de logiciels lors du développement de logiciels.Ces solutions ont été obtenues par essais et erreurs par de nombreux développeurs de logiciels sur une période assez longue.

l'audience:

Cette référence a été préparée pour les développeurs expérimentés afin de fournir les meilleures solutions à certains problèmes rencontrés lors du développement de logiciels et pour les développeurs non expérimentés d'apprendre à concevoir des logiciels de manière simple et rapide.

Qu'est-ce que le Gang of Four (GOF) ?

En 1994, quatre auteurs, Erich Gamma, Richard Helm, Ralph Johnson et John Vlissides, ont publié un livre intitulé Design Patterns - Eléments de logiciel orienté objet réutilisable, qui a initié le concept de Design Pattern dans le développement de logiciels.

Ces auteurs sont collectivement connus sous le nom de Gang of Four (GOF).Selon ces auteurs, les modèles de conception sont principalement basés sur les principes suivants de la conception orientée objet.

- Programmer une interface pas une implémentation
- Favoriser la composition d'objets par rapport à l'héritage

Utilisation du modèle de conception:

Les modèles de conception ont deux usages principaux dans le développement de logiciels.

Plateforme commune pour les développeurs
Les modèles de conception fournissent une terminologie standard et sont spécifiques à un scénario particulier.Par exemple, un modèle de conception singleton signifie l'utilisation d'un seul objet. Ainsi, tous les développeurs familiarisés avec un modèle de conception unique utilisent un seul objet et pourront se dire que le programme suit un modèle singleton.

Types de modèles de conception:

Selon le manuel de référence des modèles de conception Modèles de conception - Éléments d'un logiciel orienté objet réutilisable, il existe 23 modèles de conception pouvant être classés en trois catégories: modèles de création, modèles structurels et comportements.Nous aborderons également une autre catégorie de modèles de conception: les modèles de conception J2EE.

S.N. Modèle et description:

1- Modèles de création:
Ces modèles de conception permettent de créer des objets tout en masquant la logique de création, au lieu d'instancier directement des objets à l'aide d'un nouvel opérateur.Cela donne au programme plus de flexibilité pour décider quels objets doivent être créés pour un cas d'utilisation donné.

2- Structural Patterns:
Ces modèles de conception concernent la composition des classes et des objets.Le concept d'héritage est utilisé pour composer des interfaces et définir des méthodes pour composer des objets afin d'obtenir de nouvelles fonctionnalités.

3- Patterns comportementaux:
Ces modèles de conception concernent spécifiquement la communication entre les objets.

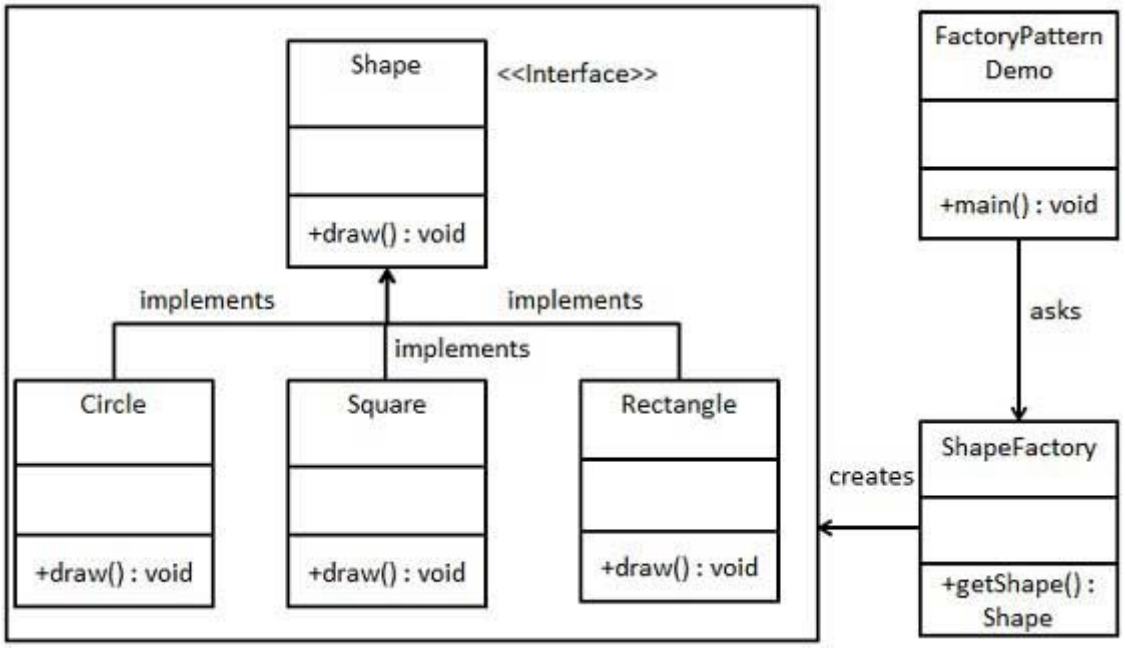
4- Patterns J2EE:
Ces modèles de conception concernent spécifiquement le niveau de présentation.Ces modèles sont identifiés par Sun Java Center.

Factory Pattern:

Le modèle d'usine est l'un des modèles de conception les plus utilisés en Java.Ce type de modèle de conception est associé à un modèle de création, car il constitue l'un des meilleurs moyens de créer un objet.

Dans le modèle Factory, nous créons un objet sans exposer la logique de création au client et faisons référence au nouvel objet créé à l'aide d'une interface commune.

L'implémentation:



Example:

Étape 1

Créez une interface.
Shape.java

```
public interface Shape {
   void draw();
}
```

Étape 2

Créer des classes concrètes implémentant la même interface.

Rectangle.java

```
public class Rectangle implements Shape {
  @Override
  public void draw() {
    System.out.println("Inside Rectangle::draw() method.");
  }}
```

Square.java

```
public class Square implements Shape {
  @Override
  public void draw() {
    System.out.println("Inside Square::draw() method.");
  }
}
```

Circle.java

```
public class Circle implements Shape {
  @Override
  public void draw() {
    System.out.println("Inside Circle::draw() method.");
  }
}
```

Étape 3

Créez une fabrique pour générer un objet de classe concrète en fonction d'informations données.

ShapeFactory.java

```
public class ShapeFactory {

  //use getShape method to get object of type shape
  public Shape getShape(String shapeType){
    if(shapeType == null){
      return null;
    }
    if(shapeType.equalsIgnoreCase("CIRCLE")){
      return new Circle();
```

```
    } else if(shapeType.equalsIgnoreCase("RECTANGLE")){
      return new Rectangle();

    } else if(shapeType.equalsIgnoreCase("SQUARE")){
      return new Square();
    }

    return null;
  }
}
```

Étape 4

Utilisez Factory pour obtenir un objet de classe concrète en transmettant une information telle que le type.

FactoryPatternDemo.java

```
public class FactoryPatternDemo {

  public static void main(String[] args) {
    ShapeFactory shapeFactory = new ShapeFactory();

    //get an object of Circle and call its draw method.
    Shape shape1 = shapeFactory.getShape("CIRCLE");

    //call draw method of Circle
    shape1.draw();

    //get an object of Rectangle and call its draw method.
    Shape shape2 = shapeFactory.getShape("RECTANGLE");

    //call draw method of Rectangle
    shape2.draw();

    //get an object of Square and call its draw method.
    Shape shape3 = shapeFactory.getShape("SQUARE");

    //call draw method of square
    shape3.draw();
  }}
```

Étape 5

Vérifiez la sortie.

```
Inside Circle::draw() method.
Inside Rectangle::draw() method.
Inside Square::draw() method.
```

En bref, les design patterns:

*** C'est...**
_ Une description d'une solution classique à problème récurrent...

_ Une description d'une partie de la solution...
Avec des relations avec système et les autres parties...

_ Une technique d'architecture logicielle

*** Ce n'est pas...**

_ Une brique:
- *Un pattern dépend de son environnement*

_ Une règle:
- *Un pattern ne peut pas s'appliquer mécaniquement*

_ Une méthode:
- *Ne guide pas une prise de décision: Un pattern est la décision prise*