JAVA DESIGN PATTERNS

problem solving approches

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

Design patterns represent the best practices used by experienced object-oriented software developers. Design patterns are solutions to general problems that software developers faced during software development. These solutions were obtained by trial and error by numerous software developers over quite a substantial period of time.

This tutorial will take you through step by step approach and examples using Java while learning Design Pattern concepts.

Audience

This reference has been prepared for the experienced developers to provide best solutions to certain problems faced during software development and for un-experienced developers to learn software design in an easy and faster way.

Prerequisites

Before you start proceeding with this tutorial, we make an assumption that you are already aware of the basic concepts of Java programming.

Copyright & Disclaimer

Table of Contents

About the Tutorial i

Audience i

Prerequisites i

Copyright & Disclaimer i

Table of Contents ii

1. DESIGN PATTERN OVERVIEW 1

What is Gang of Four (GOF)? 1

Usage of Design Pattern 1

Types of Design Patterns 1

2. FACTORY PATTERN 3

Implementation 3

3. ABSTRACT FACTORY PATTERN 7

Implementation 7

4. SINGLETON PATTERN 15

Implementation 15

5. BUILDER PATTERN 18

Implementation 18

6. PROTOTYPE PATTERN 26

Implementation 26

7. ADAPTER PATTERN 32

Implementation 32

8. BRIDGE PATTERN 38

Implementation 38

9. FILTER PATTERN 42

Implementation 42

10. COMPOSITE PATTERN 50

Implementation 50

11. DECORATOR PATTERN 54

Implementation 54

12. FACADE PATTERN 59

Implementation 59

13. FLYWEIGHT PATTERN 63

Implementation 63

14. PROXY PATTERN 69

Implementation 69

15. CHAIN OF RESPONSIBILITY PATTERN 72

Implementation 72

16. COMMAND PATTERN 77

Implementation 77

17. INTERPRETER PATTERN 81

Implementation 81

18. ITERATOR PATTERN 85

Implementation 85

19. MEDIATOR PATTERN 88

Implementation 88

20. MEMENTO PATTERN 91

Implementation 91

21. OBSERVER PATTERN 95

Implementation 95

22. STATE PATTERN 100

Implementation 100

23. NULL OBJECT PATTERN 104

Implementation 104

24. STRATEGY PATTERN 108

Implementation 108

25. TEMPLATE PATTERN 112

Implementation 112

26. VISITOR PATTERN 116

Implementation 116

27. MVC PATTERN 121

Implementation 121

28. BUSINESS DELEGATE PATTERN 126

Implementation 126

29. COMPOSITE ENTITY PATTERN 131

Implementation 131

30. DATA ACCESS OBJECT PATTERN 136

Implementation 136

31. FRONT CONTROLLER PATTERN 141

Implementation 141

32. INTERCEPTNG FILTER PATTERN 145

Implementation 145

33. SERVICE LOCATOR PATTERN..151

Implementation **151**

34. TRANSFER OBJECT PATTERN...157

Implementation **157**

DESIGN PATTERN OVERVIEW

Design patterns represent the best practices used by experienced object-oriented software developers. Design patterns are solutions to general problems that software developers faced during software development. These solutions were obtained by trial and error by numerous software developers over quite a substantial period of time.

What is Gang of Four (GOF)?

In 1994, four authors Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides published a book titled **Design Patterns - Elements of Reusable Object-Oriented Software** which initiated the concept of Design Pattern in Software development.

These authors are collectively known as **Gang of Four (GOF)**. According to these authors design patterns are primarily based on the following principles of object orientated design.

- Program to an interface not an implementation
- Favor object composition over inheritance

Usage of Design Pattern

Design Patterns have two main usages in software development.

Common platform for developers

Design patterns provide a standard terminology and are specific to particular scenario. For example, a singleton design pattern signifies the use of single object so all developers familiar with single design pattern will make use of single object and they can tell each other that program is following a singleton pattern.

Best Practices

Design patterns have been evolved over a long period of time and they provide best solutions to certain problems faced during software development. Learning these patterns help unexperienced developers to learn software design in an easy and fast way.

Types of Design Patterns

As per the design pattern reference book **Design Patterns - Elements of Reusable Object-Oriented Software**, there are 23 design patterns which can be classified in three categories: Creational, Structural and Behavioral patterns. We will also discuss another category of design pattern: J2EE design patterns.

S.N.	Pattern & Description
1	**Creational Patterns** These design patterns provide a way to create objects while hiding the creation logic, rather than instantiating objects directly using new operator. This gives more flexibility to the program in deciding which objects need to be created for a given use case.
2	**Structural Patterns** These design patterns concern class and object composition. Concept of inheritance is used to compose interfaces and define ways to compose objects to obtain new functionalities.
3	**Behavioral Patterns** These design patterns are specifically concerned with communication between objects.
4	**J2EE Patterns** These design patterns are specifically concerned with the presentation tier. These patterns are identified by Sun Java Center.

FACTORY PATTERN

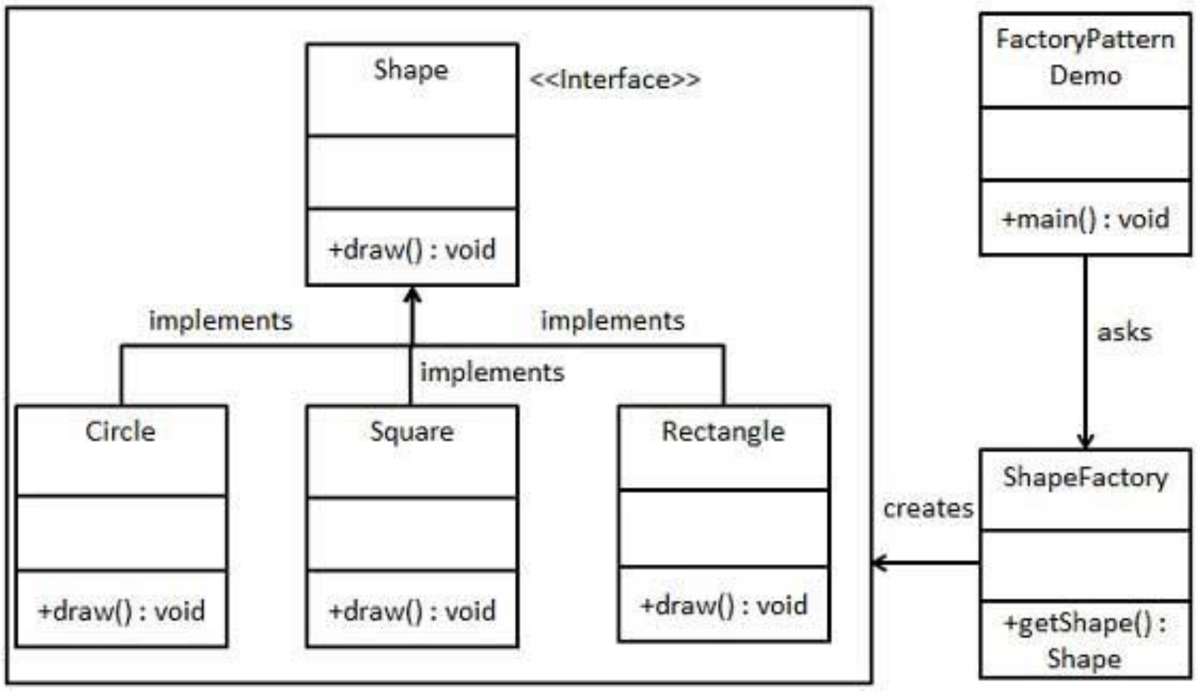
Factory pattern is one of most used design patterns in Java. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

In Factory pattern, we create objects without exposing the creation logic to the client and refer to newly created object using a common interface.

Implementation

We are going to create a Shape interface and concrete classes implementing the Shape interface. A factory class ShapeFactory is defined as a next step.

FactoryPatternDemo, our demo class, will use ShapeFactory to get a Shape object. It will pass information (CIRCLE / RECTANGLE / SQUARE) to ShapeFactory to get the type of object it needs.



Step 1

Create an interface.

Shape.java

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

Step 2

Create concrete classes implementing the same 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.");
   }
}
```

Step 3

Create a Factory to generate object of concrete class based on given information.

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;
   }
}
```

Step 4

Use the Factory to get object of concrete class by passing an information such as 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 circle
      shape3.draw();
   }
}
```

Step 5

Verify the output.

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