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

© Copyright 2015 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>About the Tutorial</td>
<td>i</td>
</tr>
<tr>
<td>Audience</td>
<td>i</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>i</td>
</tr>
<tr>
<td>Copyright &amp; Disclaimer</td>
<td>i</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>ii</td>
</tr>
<tr>
<td>1. DESIGN PATTERN OVERVIEW</td>
<td>1</td>
</tr>
<tr>
<td>What is Gang of Four (GOF)?</td>
<td>1</td>
</tr>
<tr>
<td>Usage of Design Pattern</td>
<td>1</td>
</tr>
<tr>
<td>Types of Design Patterns</td>
<td>1</td>
</tr>
<tr>
<td>2. FACTORY PATTERN</td>
<td>3</td>
</tr>
<tr>
<td>Implementation</td>
<td>3</td>
</tr>
<tr>
<td>3. ABSTRACT FACTORY PATTERN</td>
<td>7</td>
</tr>
<tr>
<td>Implementation</td>
<td>7</td>
</tr>
<tr>
<td>4. SINGLETON PATTERN</td>
<td>15</td>
</tr>
<tr>
<td>Implementation</td>
<td>15</td>
</tr>
<tr>
<td>5. BUILDER PATTERN</td>
<td>18</td>
</tr>
<tr>
<td>Implementation</td>
<td>18</td>
</tr>
<tr>
<td>6. PROTOTYPE PATTERN</td>
<td>26</td>
</tr>
<tr>
<td>Implementation</td>
<td>26</td>
</tr>
<tr>
<td>7. ADAPTER PATTERN</td>
<td>32</td>
</tr>
<tr>
<td>Implementation</td>
<td>32</td>
</tr>
<tr>
<td>8. BRIDGE PATTERN</td>
<td>38</td>
</tr>
<tr>
<td>Implementation</td>
<td>38</td>
</tr>
</tbody>
</table>
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. INTERCEPTING FILTER PATTERN ............................................... 145
    Implementation .......................................................................... 145
33. SERVICE LOCATOR PATTERN ........................................................................................................ 151
   Implementation ............................................................................................................................ 151

34. TRANSFER OBJECT PATTERN .................................................................................................. 157
   Implementation ............................................................................................................................ 157
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.

<table>
<thead>
<tr>
<th>S.N.</th>
<th>Pattern &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>Creational Patterns</strong>&lt;br&gt;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.</td>
</tr>
<tr>
<td>2</td>
<td><strong>Structural Patterns</strong>&lt;br&gt;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.</td>
</tr>
<tr>
<td>3</td>
<td><strong>Behavioral Patterns</strong>&lt;br&gt;These design patterns are specifically concerned with communication between objects.</td>
</tr>
<tr>
<td>4</td>
<td><strong>J2EE Patterns</strong>&lt;br&gt;These design patterns are specifically concerned with the presentation tier. These patterns are identified by Sun Java Center.</td>
</tr>
</tbody>
</table>
Factory pattern is one of the most used design patterns in Java. This type of design pattern comes under the 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

```java
public class Rectangle implements Shape {

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

Square.java

```java
public class Square implements Shape {

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

Circle.java

```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**

```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**

```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.