



Design Patterns Tutorial

- ▣ Design Patterns - Home
- ▣ Design Patterns - Overview
- ▣ Design Patterns - Factory Pattern
- ▣ Abstract Factory Pattern
- ▣ Design Patterns - Singleton Pattern
- ▣ Design Patterns - Builder Pattern
- ▣ Design Patterns - Prototype Pattern
- ▣ Design Patterns - Adapter Pattern
- ▣ Design Patterns - Bridge Pattern
- ▣ Design Patterns - Filter Pattern
- ▣ Design Patterns - Composite Pattern
- ▣ Design Patterns - Decorator Pattern
- ▣ Design Patterns - Facade Pattern
- ▣ Design Patterns - Flyweight Pattern
- ▣ Design Patterns - Proxy Pattern
- ▣ Chain of Responsibility Pattern
- ▣ Design Patterns - Command Pattern
- ▣ Design Patterns - Interpreter Pattern
- ▣ Design Patterns - Iterator Pattern



▣ Design Patterns - Observer Pattern

▣ Design Patterns - State Pattern

▣ Design Patterns - Null Object Pattern

▣ Design Patterns - Strategy Pattern

▣ Design Patterns - Template Pattern

▣ Design Patterns - Visitor Pattern

▣ Design Patterns - MVC Pattern

▣ Business Delegate Pattern

▣ Composite Entity Pattern

▣ Data Access Object Pattern

▣ Front Controller Pattern

▣ Intercepting Filter Pattern

▣ Service Locator Pattern

▣ Transfer Object Pattern

Design Patterns Resources

▣ Design Patterns - Questions/Answers

▣ Design Patterns - Quick Guide

▣ Design Patterns - Useful Resources

▣ Design Patterns - Discussion

Design Patterns - Observer Pattern

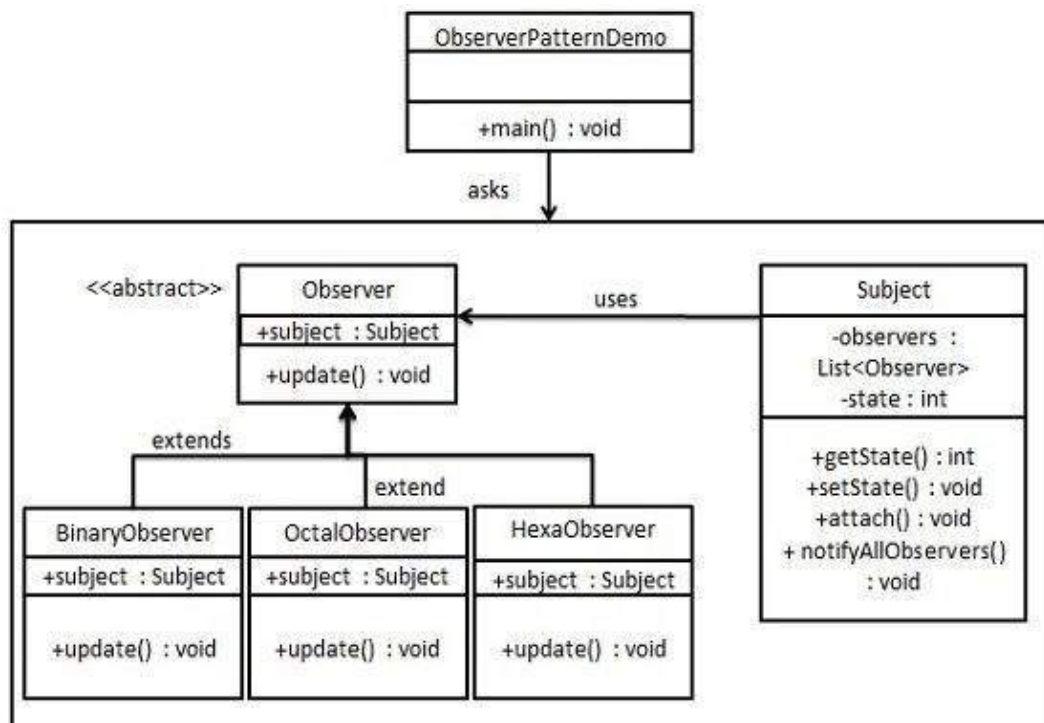

[⬅ Previous Page](#)
[Next Page ➡](#)

Observer pattern is used when there is one-to-many relationship between objects such as if one object is modified, its dependent objects are to be notified automatically. Observer pattern falls under behavioral pattern category.

Implementation

Observer pattern uses three actor classes. Subject, Observer and Client. Subject is an object having methods to attach and detach observers to a client object. We have created an abstract class *Observer* and a concrete class *Subject* that is extending class *Observer*.

ObserverPatternDemo, our demo class, will use *Subject* and concrete class object to show observer pattern in action.



Step 1

Create Subject class.

Subject.java

```
import java.util.ArrayList;
import java.util.List;
```



```
public int getState() {
    return state;
}

public void setState(int state) {
    this.state = state;
    notifyAllObservers();
}

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

public void notifyAllObservers(){
    for (Observer observer : observers) {
        observer.update();
    }
}
}
```

Step 2

Create Observer class.

Observer.java

```
public abstract class Observer {
    protected Subject subject;
    public abstract void update();
}
```

Step 3

Create concrete observer classes

BinaryObserver.java

```
public class BinaryObserver extends Observer{

    public BinaryObserver(Subject subject){
        this.subject = subject;
        this.subject.attach(this);
    }

    @Override
    public void update() {
        System.out.println( "Binary String: " + Integer.toBinaryString( subject.getState() ) );
    }
}
```

OctalObserver.java



```
}

@Override
public void update() {
    System.out.println( "Octal String: " + Integer.toOctalString( subject.getState() ) );
}
}
```

HexaObserver.java

```
public class HexaObserver extends Observer{

    public HexaObserver(Subject subject){
        this.subject = subject;
        this.subject.attach(this);
    }

    @Override
    public void update() {
        System.out.println( "Hex String: " + Integer.toHexString( subject.getState() ).toUpperCase() );
    }
}
```

Step 4

Use *Subject* and concrete observer objects.

ObserverPatternDemo.java

```
public class ObserverPatternDemo {
    public static void main(String[] args) {
        Subject subject = new Subject();

        new HexaObserver(subject);
        new OctalObserver(subject);
        new BinaryObserver(subject);

        System.out.println("First state change: 15");
        subject.setState(15);
        System.out.println("Second state change: 10");
        subject.setState(10);
    }
}
```

Step 5

Verify the output.

```
First state change: 15
Hex String: F
```



Octal String: 12

Binary String: 1010

[⬅ Previous Page](#)[Next Page ➡](#)

Advertisements

The advertisement collage features a variety of diagrams:

- Organizational charts showing hierarchical structures.
- Network graphs with nodes and edges.
- Flowcharts illustrating processes and decision paths.
- State machine diagrams with states and transitions.
- A mind map titled "Mind Map" with branches for "Check messages less", "Message filters", "Too many meetings", "Time wasting", "Message fatigue", "Methods", "Deadlines", "Checkpoints", "Getting more time", "Wakeup early", "Delegate", "Simplify", "Priorities", "More effective use", "Planning", "Goals", and "Ways to focus".
- A diagram titled "JavaScript Diagrams" with the text "Click Here for GoJS".



© Copyright 2015. All Rights Reserved.

Enter email for newsletter

go