Observer and Mediator Pattern

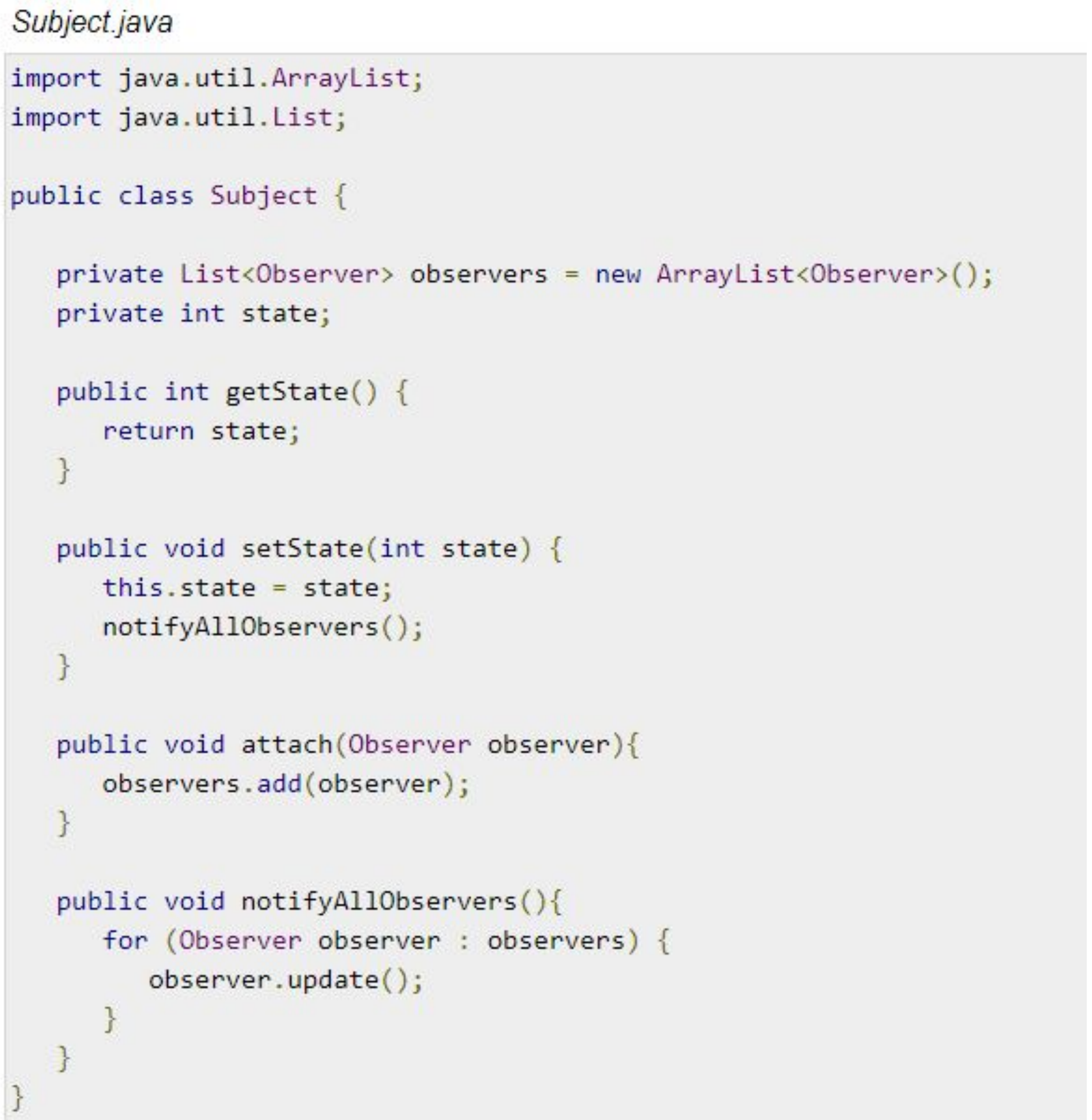
Observer Pattern

- 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.
- **The Observer Pattern provides an object design where subjects and observers are loosely coupled. WHY? How?**
- Observer pattern falls under behavioral pattern category.

Observer Pattern

Observer pattern uses three actor classes. Subject, Observer and Client.

Observer Pattern

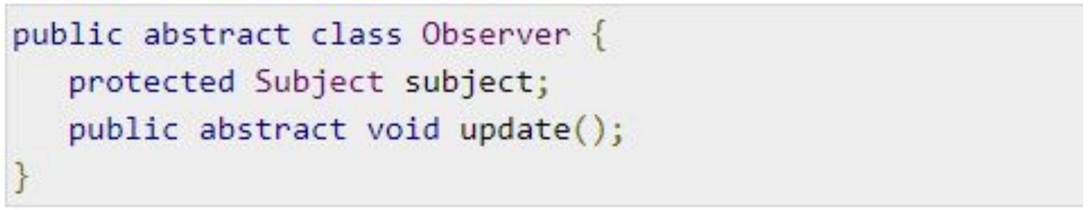


Subject.java

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

public class Subject {

   private List<Observer> observers = new ArrayList<Observer>();
   private int state;

   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();
      }
   }
}
```

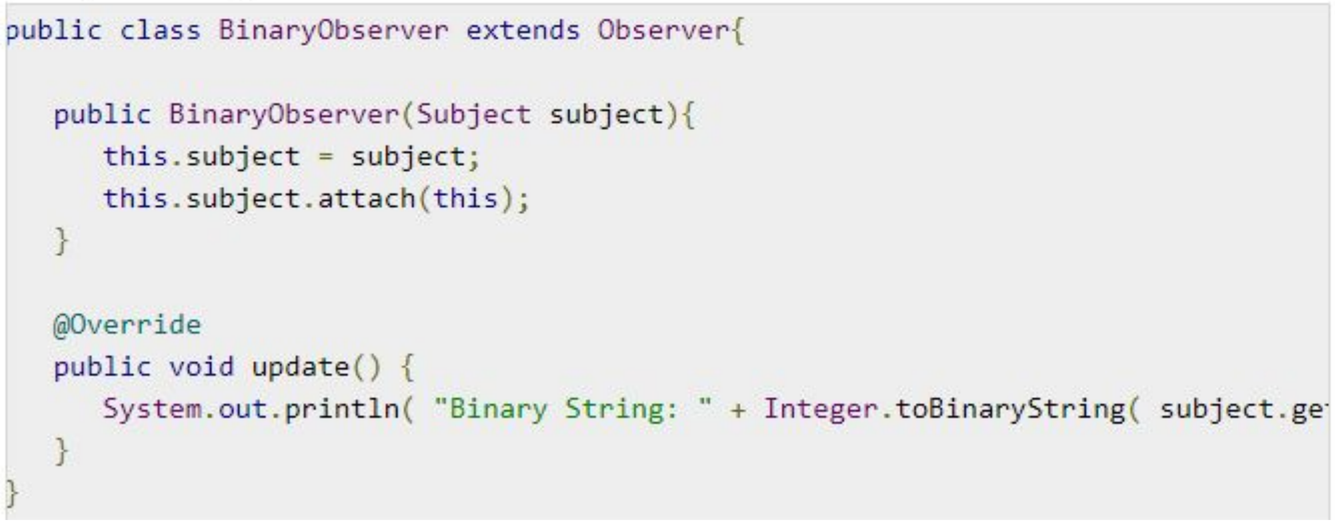
Observer Pattern

Observer.java



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

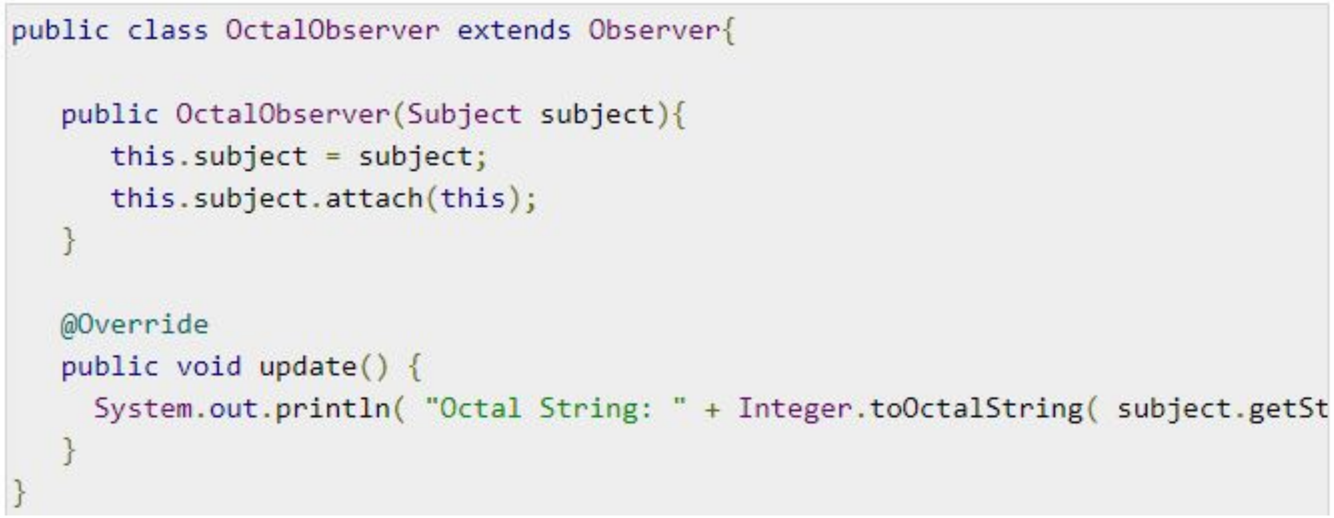
BinaryObserver.java



```
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.ge
   }
}
```

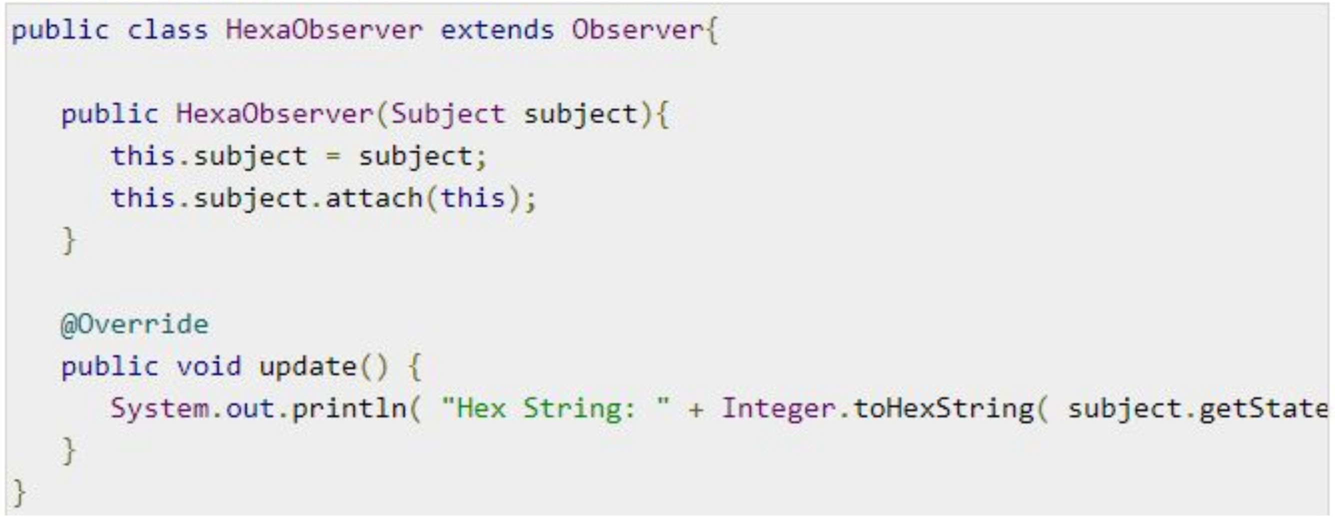
Observer Pattern

OctalObserver.java



```java
public class OctalObserver extends Observer{

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

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

HexaObserver.java



```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
   }
}
```

Observer Pattern

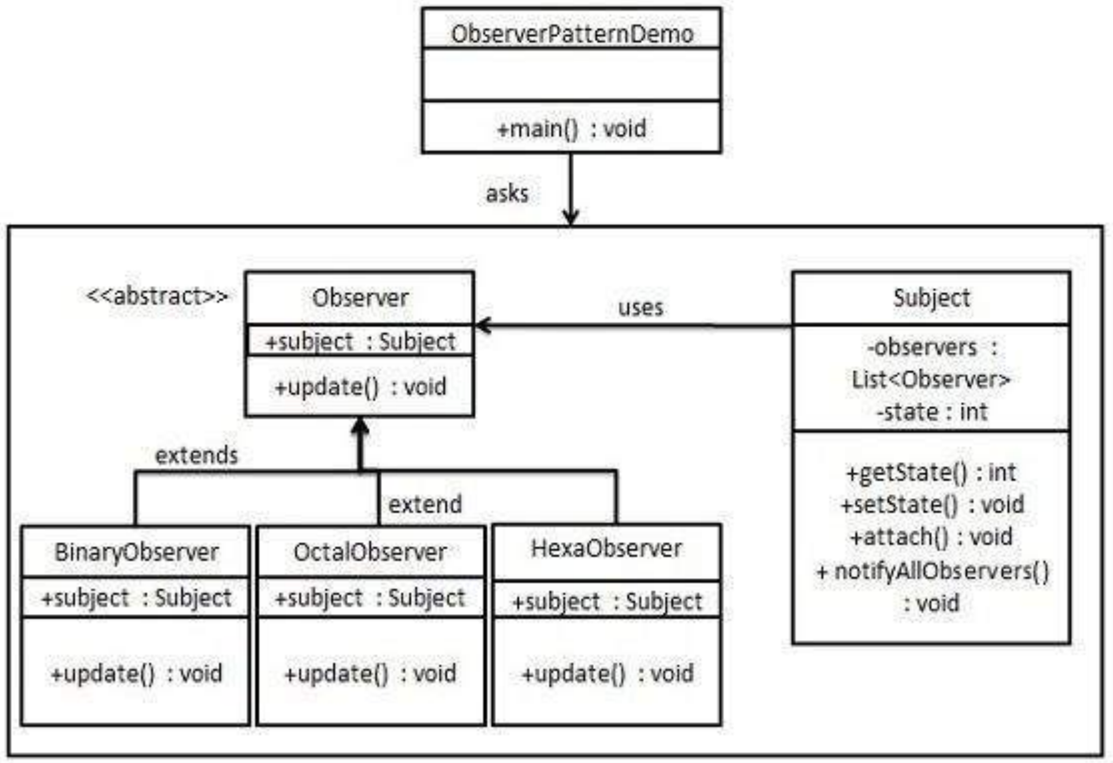
ObserverPatternDemo.java

```
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);
   }
}
```

Verify the output.

```
First state change: 15
Hex String: F
Octal String: 17
Binary String: 1111
Second state change: 10
Hex String: A
Octal String: 12
Binary String: 1010
```

UML Diagram for the Observer Pattern



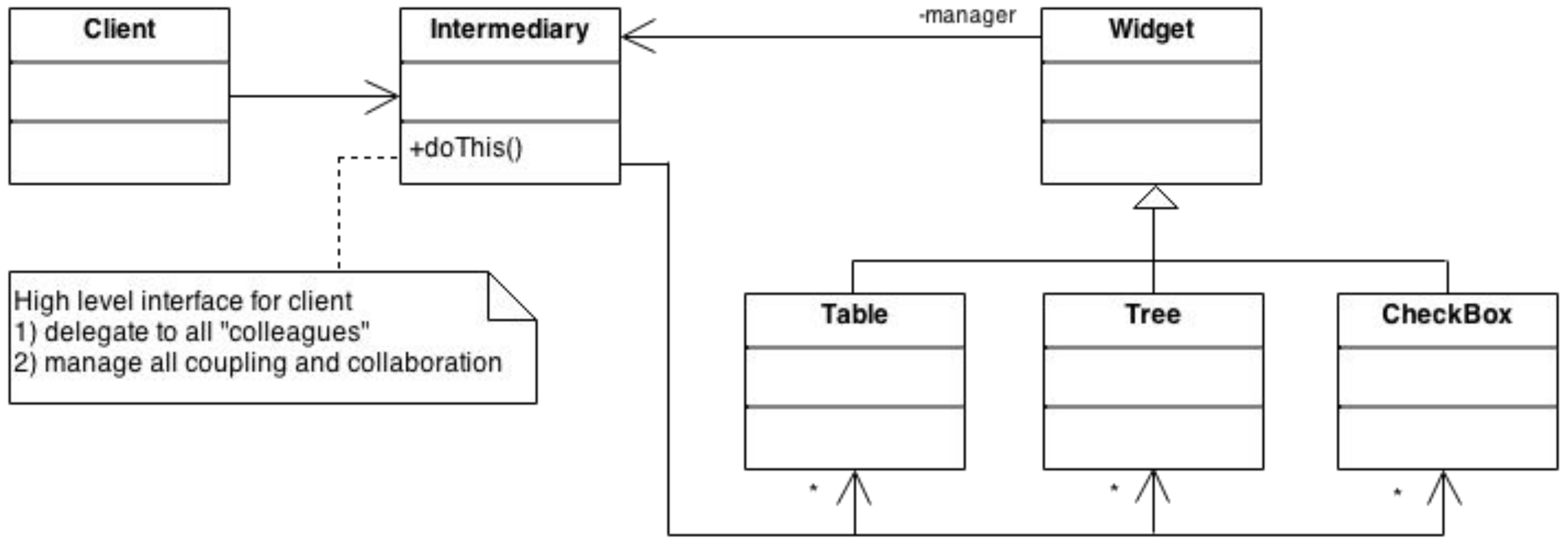
Problem

- What if the relationship is many-to-many?

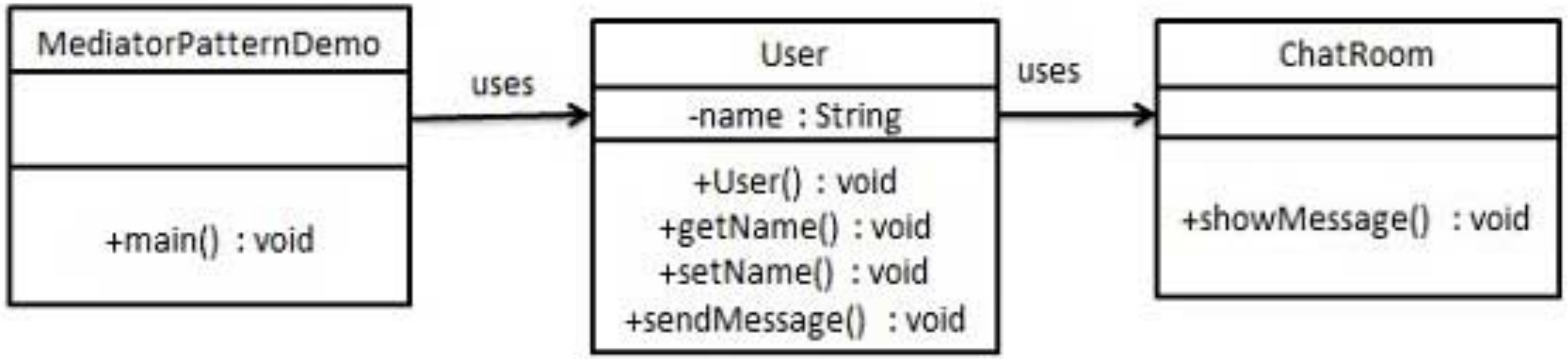
Mediator Pattern

- Mediator pattern is used to reduce communication complexity between multiple objects or classes.
- This pattern provides a mediator class which normally handles all the communications between different classes and supports easy maintenance of the code by loose coupling.
- Mediator pattern falls under behavioral pattern category.

Mediator Pattern


Client
Intermediary
+doThis()
-manager
Widget
Table
Tree
CheckBox
*
*
*
High level interface for client
1) delegate to all "colleagues"
2) manage all coupling and collaboration

Mediator Pattern



HW

- Learn writing a code on mediator.
- Think on the situations where we can use observer or mediator pattern.
- Differences between observer and mediator.

HW

- Prepare a 3 minute presentation on you assignment 1 (combining strategy, factory and abstract factory)