SOEN 6441
Advanced Program Practices

Lab 3: MVC and Observer Design Pattern

Instructors:	Dr. Joey Paquet	joey.paquet@concordia.ca
	Dr. Amin Ranj Bar	amin.ranjbar@concordia.ca
TA:	Hamed Jafarpour	hamed.jafarpour@concordia.ca

Outline

Model-View-Controller (MVC)

- MVC, which stands for Model-View-Controller, is a software architectural pattern commonly used in the design and development of software applications, especially in the context of web and desktop applications.
- It is designed to separate the concerns of an application into three interconnected components, each with its own distinct role:
    - Model
    - **View**
    - **Controller**

Model

- The Model represents the application's data and business logic.
- It is responsible for managing and manipulating the data,
- It is responsible for responding to requests from the Controller to update itself.
- In essence, the Model encapsulates the core functionality of the application and ensures data consistency.

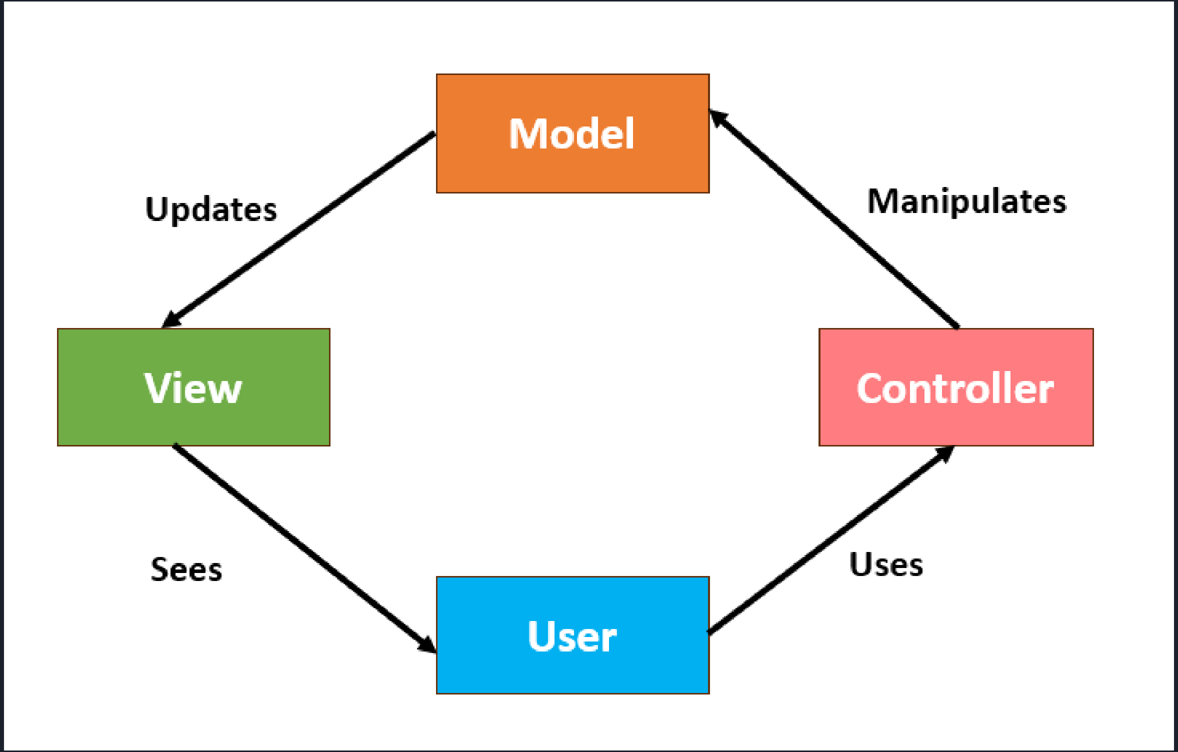
View

- The View is responsible for presenting the data to the user in a human-readable format.
- It represents the user interface and displays information from the Model to the user.
- Views can include elements like web pages, user interfaces, or any visual representation of data.

Controller

- The Controller acts as an intermediary between the Model and the View.
- It receives user input and translates it into commands for the Model or the View.
- It handles user interactions, processes requests, and updates the Model and View accordingly.
- The Controller is responsible for managing the flow of data and user interactions within the application.

MVC components



Advantages of MVC

1. **Simultaneous Development:** Developers can work on different components simultaneously without interference, thanks to the decoupling of MVC components.

2. **Reusability:** Views can be easily reused for different applications, as they focus solely on data presentation, making them adaptable to various data sources.

3. **Improved Scalability:** Scalability is enhanced because performance issues in one component, like database access, can be addressed independently without impacting other parts of the application.

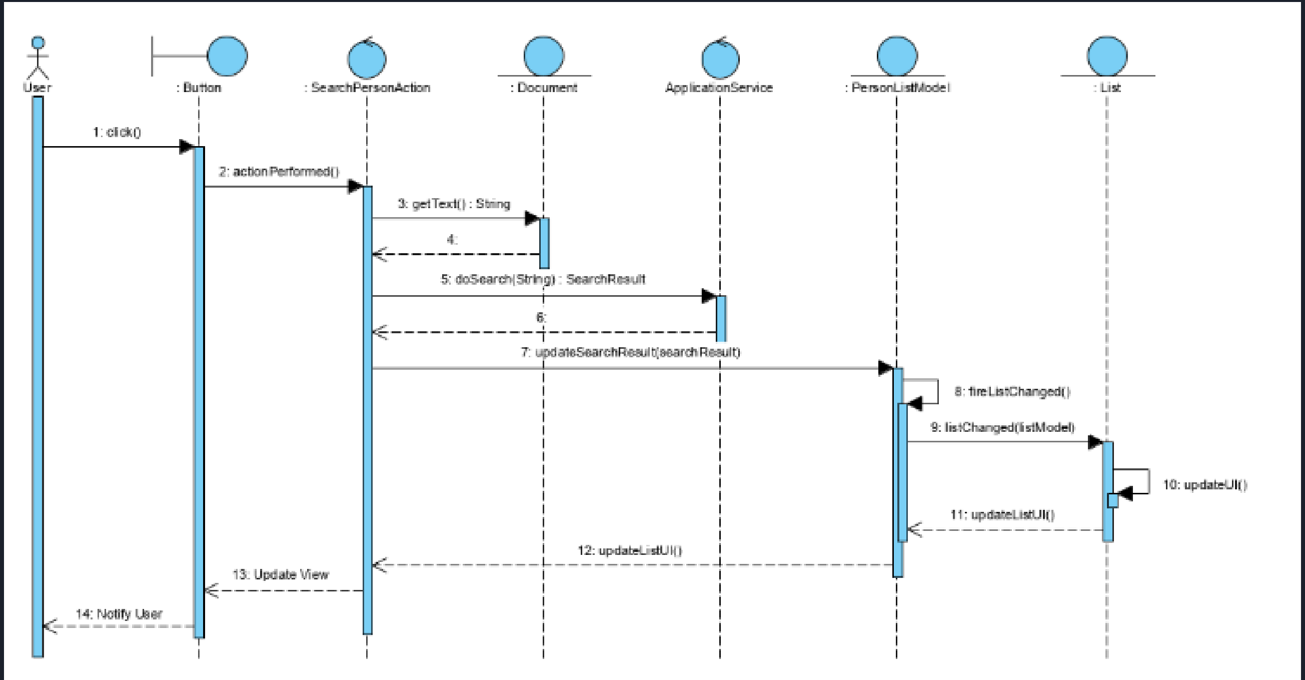
4. **Low Coupling:** MVC inherently maintains low coupling between Models, Views, and Controllers, reducing interdependence among these components.

5. **Better Extendibility:** Due to the minimal interdependence, modifications or enhancements to one component do not adversely affect the others, making the system more extendable and maintainable.

Example MVC Sequence Diagram

The “Search for Persons” use case Scenario is[1]:

1.The user enters a search string in the text field.
2.The user clicks the search button.
3.The search result is displayed in the result list.



Design Pattern

✓The design pattern is a general reusable solution to a common problem that occurs in software design.

✓Design patterns are not specific to a particular programming language or technology but provide a template or guideline for solving recurring design problems.

Observer Design Pattern

- ✓ The Observer design pattern is one of the behavioral design patterns.
- ✓ It is used when you need to establish a one-to-many relationship between objects.
- ✓ In this pattern, an object (known as the subject or publisher) maintains a list of its dependents (known as observers or subscribers) and notifies them of any state changes, typically by calling one of their methods.

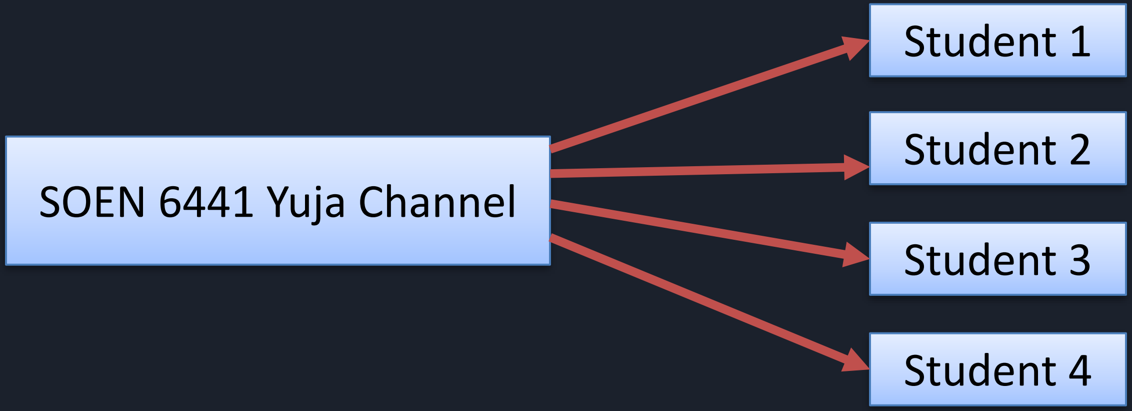
Advantages of Observer Design Pattern

- ✓ Loose Coupling: Promotes independence between subject and observers, enhancing modularity.
- ✓ Flexibility: Easily add or remove observers without altering existing code.
- ✓ Reusability: Observers can be used across different subjects, promoting code reusability.
- ✓ Decentralized Control: Allows for decentralized management of interactions, reducing complexity.
- ✓ Real-Time Updates: Ideal for scenarios requiring real-time updates and event handling, enhancing responsiveness.

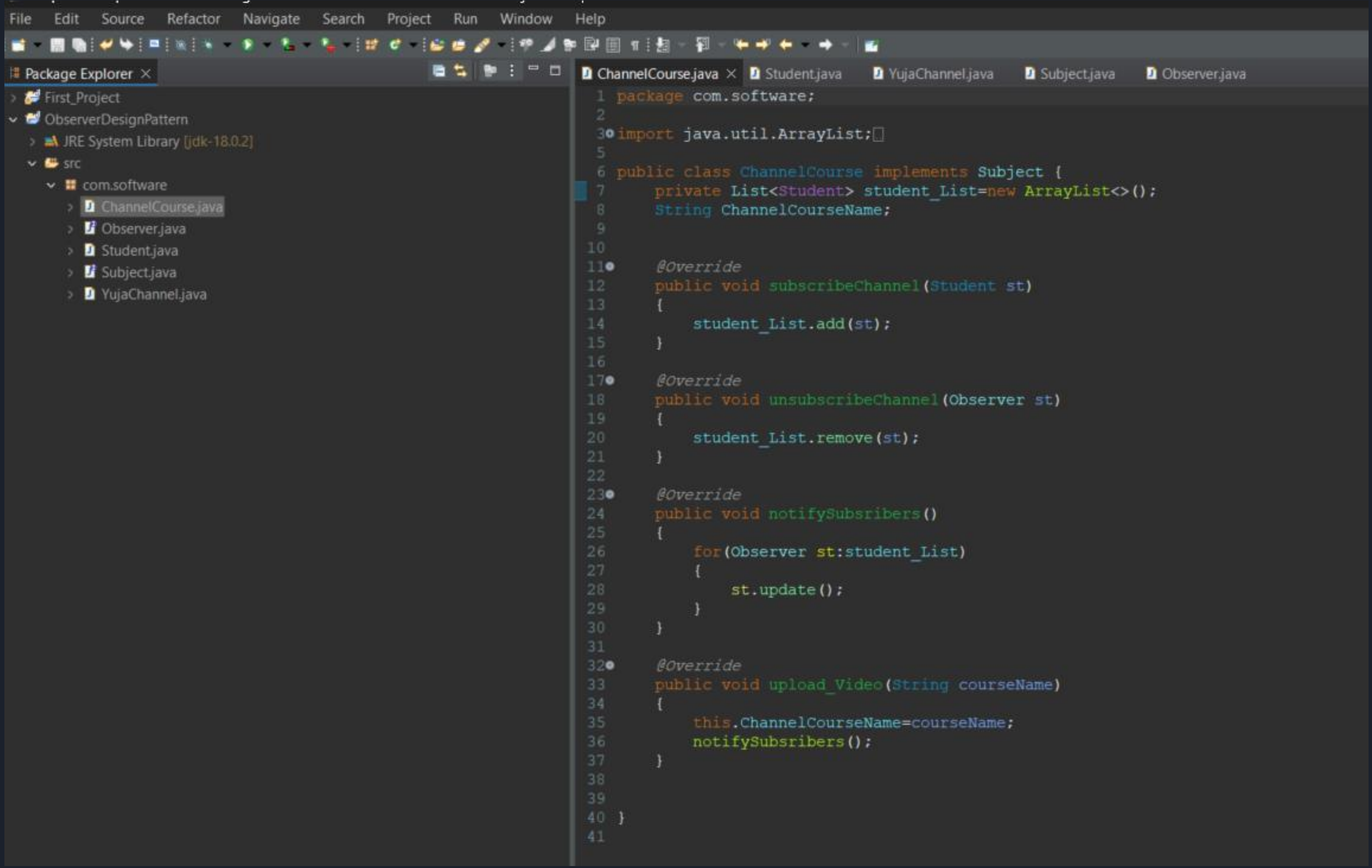
Example of Observer Design Pattern

Yuja Channel

- ✓ Students are observer
- ✓ Channel courses are subject (e.g. Channel for SOEN6441)

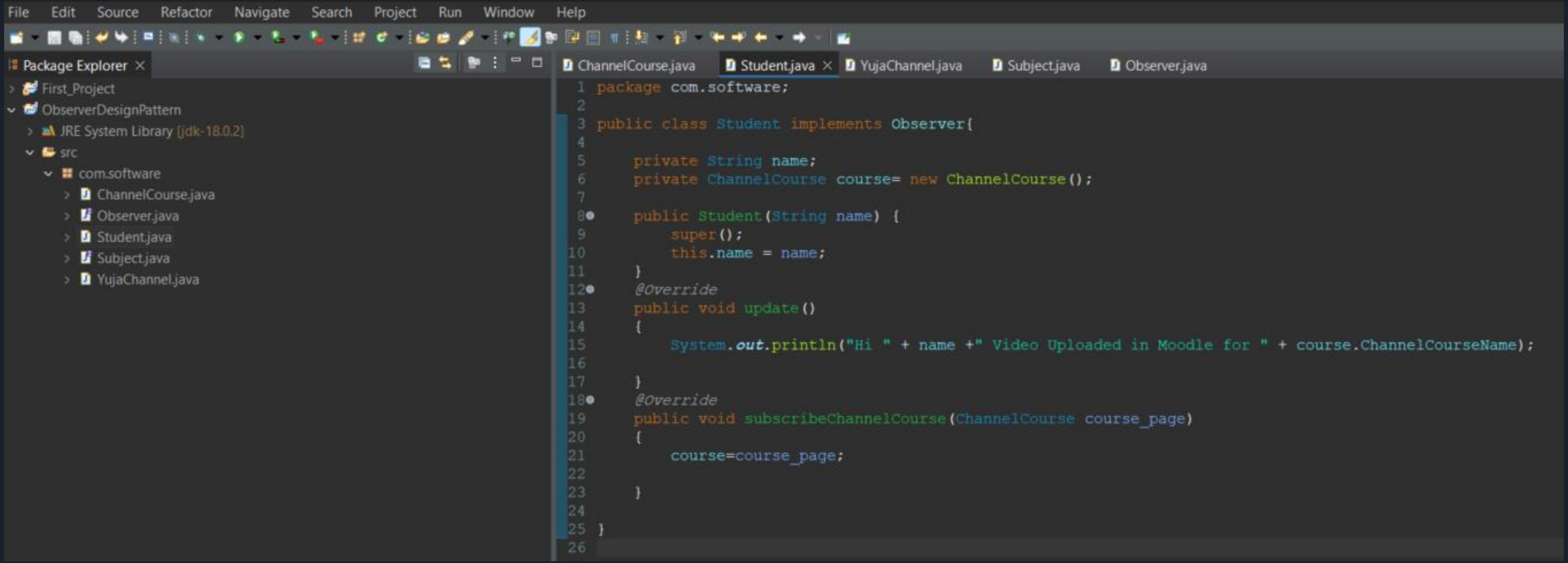


Java sample code for observer design pattern



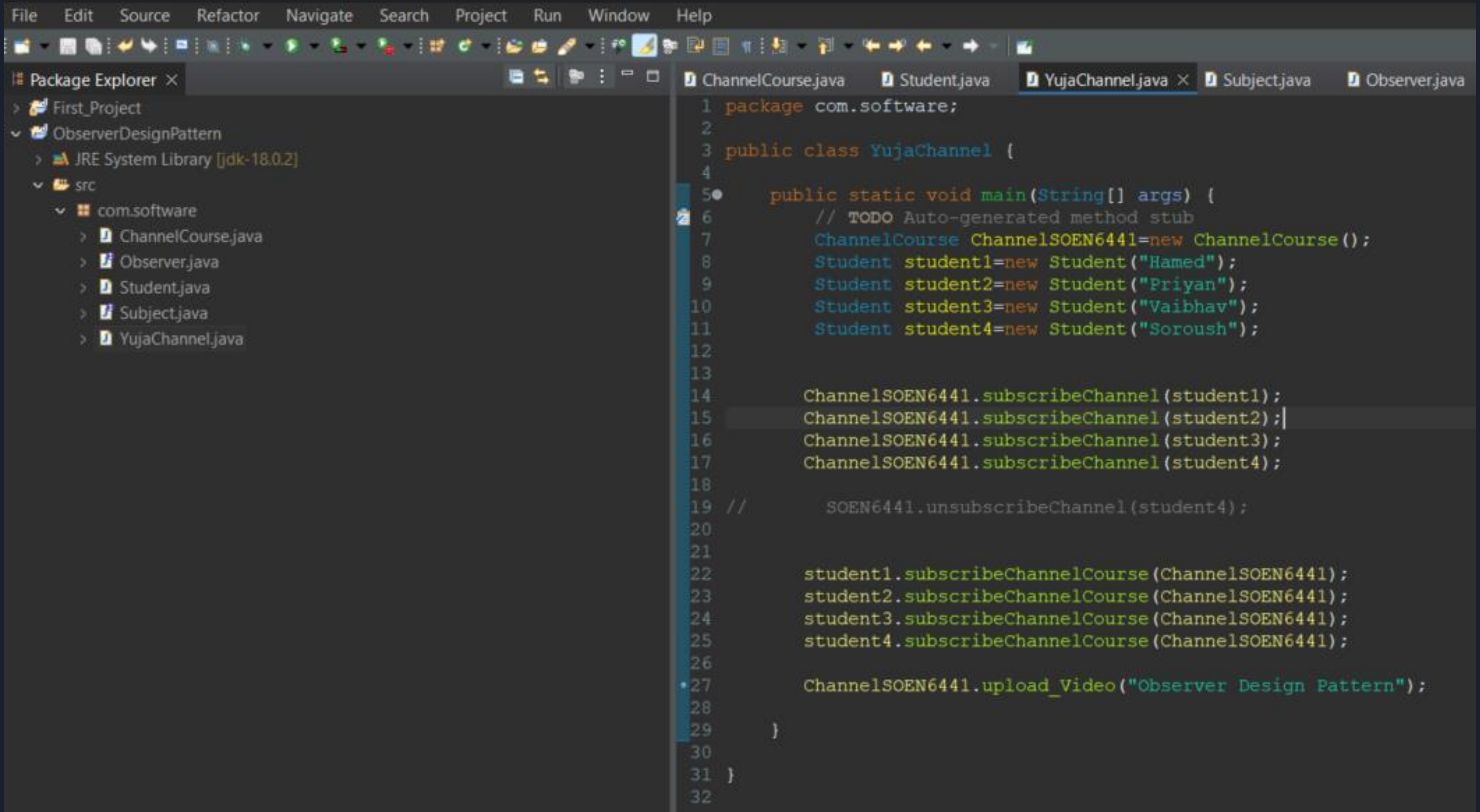
```java
package com.software;

import java.util.ArrayList;

public class ChannelCourse implements Subject {
    private List<Student> student_List=new ArrayList<>();
    String ChannelCourseName;


    @Override
    public void subscribeChannel(Student st)
    {
        student_List.add(st);
    }

    @Override
    public void unsubscribeChannel(Observer st)
    {
        student_List.remove(st);
    }

    @Override
    public void notifySubsribers()
    {
        for(Observer st:student_List)
        {
            st.update();
        }
    }

    @Override
    public void upload_Video(String courseName)
    {
        this.ChannelCourseName=courseName;
        notifySubsribers();
    }


}
```

Java sample code for observer design pattern



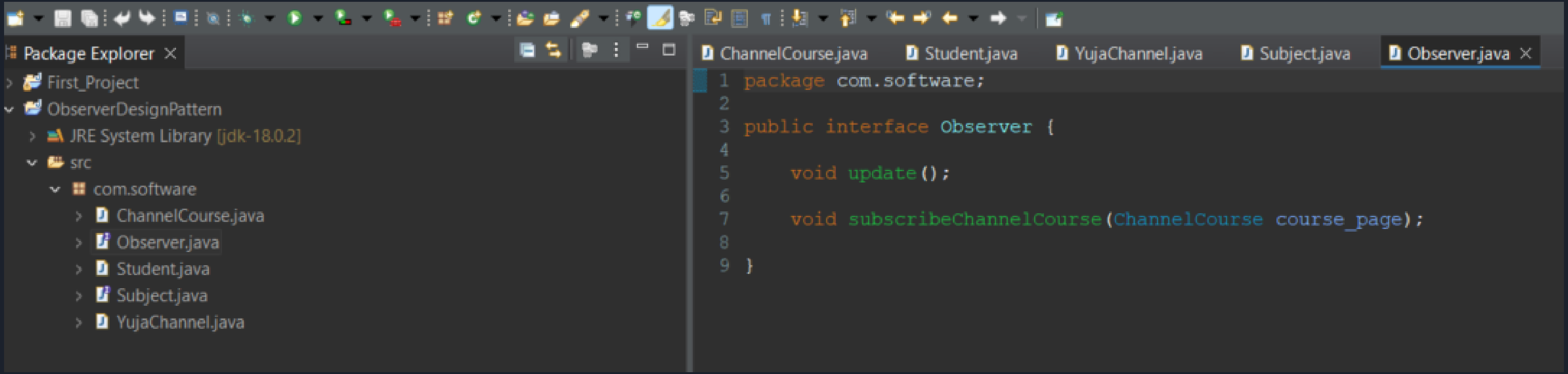
```java
package com.software;

public class Student implements Observer{

    private String name;
    private ChannelCourse course= new ChannelCourse();

    public Student(String name) {
        super();
        this.name = name;
    }
    @Override
    public void update()
    {
        System.out.println("Hi " + name +" Video Uploaded in Moodle for " + course.ChannelCourseName);

    }
    @Override
    public void subscribeChannelCourse(ChannelCourse course_page)
    {
        course=course_page;

    }

}

```

Java sample code for observer design pattern



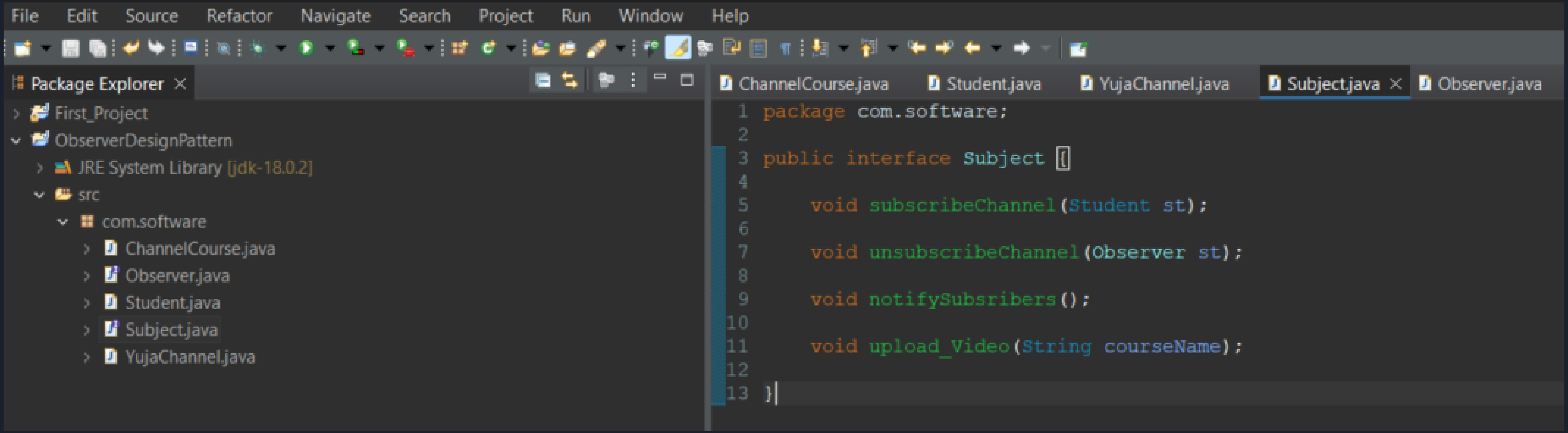
```java
package com.software;

public class YujaChannel {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        ChannelCourse ChannelSOEN6441=new ChannelCourse();
        Student student1=new Student("Hamed");
        Student student2=new Student("Priyan");
        Student student3=new Student("Vaibhav");
        Student student4=new Student("Soroush");


       ChannelSOEN6441.subscribeChannel(student1);
       ChannelSOEN6441.subscribeChannel(student2);
       ChannelSOEN6441.subscribeChannel(student3);
       ChannelSOEN6441.subscribeChannel(student4);

//        SOEN6441.unsubscribeChannel(student4);


       student1.subscribeChannelCourse(ChannelSOEN6441);
       student2.subscribeChannelCourse(ChannelSOEN6441);
       student3.subscribeChannelCourse(ChannelSOEN6441);
       student4.subscribeChannelCourse(ChannelSOEN6441);

       ChannelSOEN6441.upload_Video("Observer Design Pattern");

    }

}
```

Java sample code for observer design pattern



```java
package com.software;

public interface Observer {

    void update();

    void subscribeChannelCourse(ChannelCourse course_page);

}
```

Java sample code for observer design pattern



```java
package com.software;

public interface Subject {

    void subscribeChannel(Student st);

    void unsubscribeChannel(Observer st);

    void notifySubsribers();

    void upload_Video(String courseName);

}
```

Thank You
For Your Attention

Reference

[1] https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-model-view-control-mvc/