



SOEN 6441

Advanced Program Practices

Lab 3: MVC and Observer Design Pattern

Instructors: Dr. Joey Paquet
Dr. Amin Ranj Bar

joey.paquet@concordia.ca
amin.ranjbar@concordia.ca

TA: Hamed Jafarpour

hamed.jafarpour@concordia.ca



Outline

- ❑ **Model-View-Controller (MVC)**
- ❑ **MVC components**
- ❑ **Advantages of MVC**
- ❑ **Example MVC Sequence Diagram**
- ❑ **Design Pattern**
- ❑ **Observer Design Pattern**
- ❑ **Advantages of Observer Design Pattern**
- ❑ **Example by Sample Java Code**



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

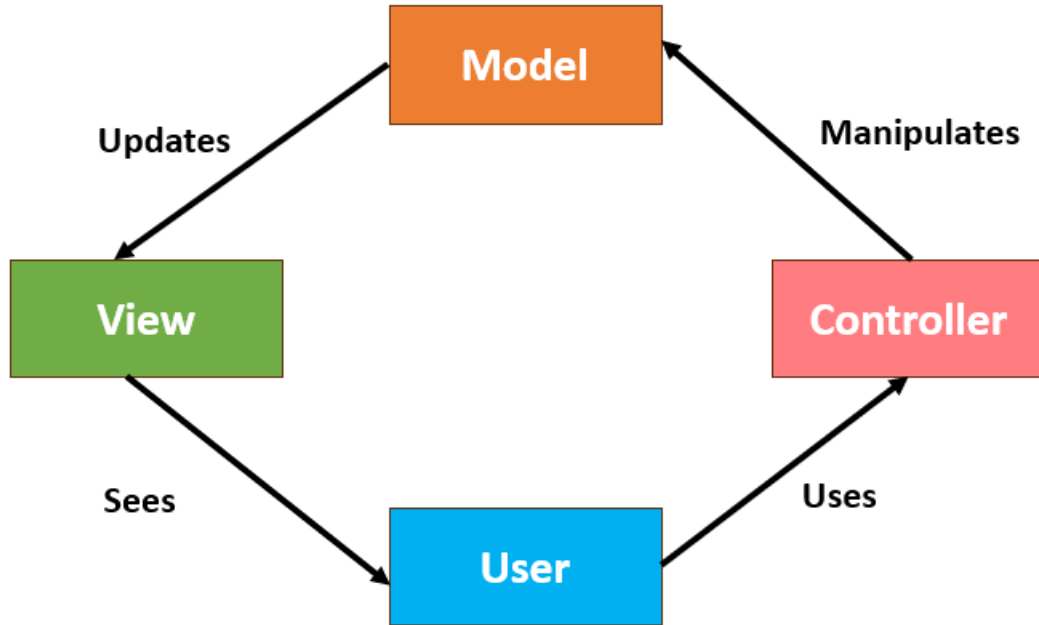
- The View is responsible for presenting the data to the user in a human-readable format.
- It represents the user interface and displays information from the Model to the user.
- Views can include elements like web pages, user interfaces, or any visual representation of data.



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.

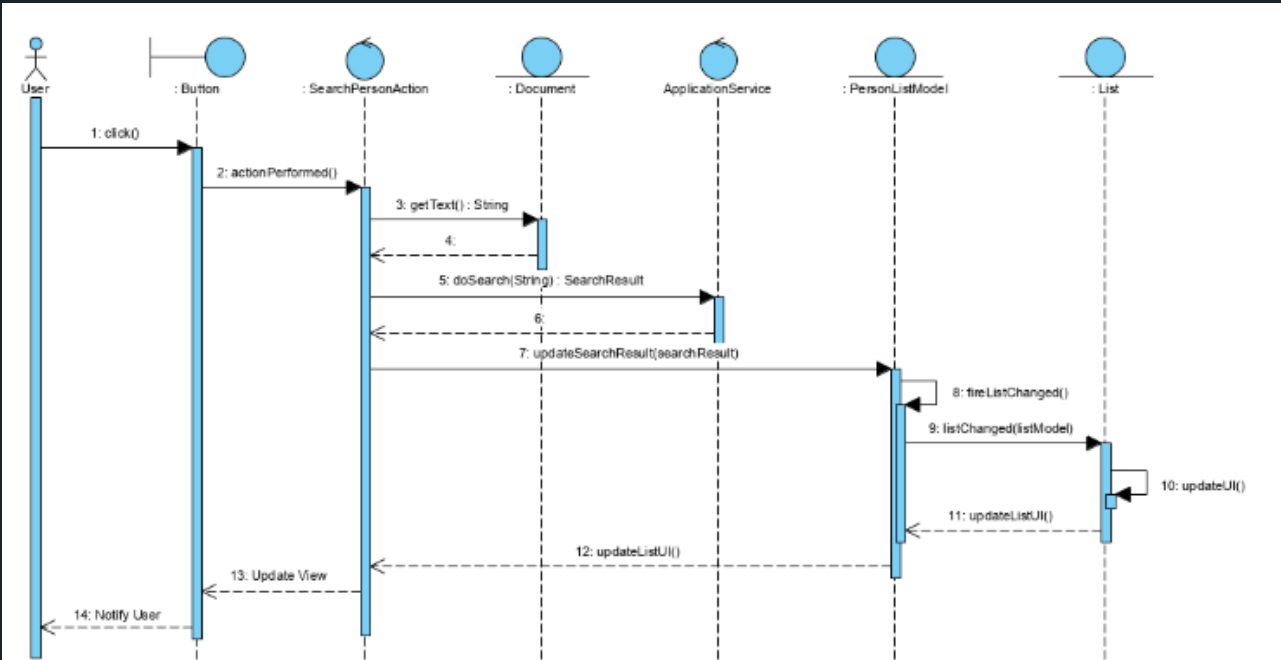
4. Low Coupling: MVC inherently maintains low coupling between Models, Views, and Controllers, reducing interdependence among these components.

5. Better Extensibility: 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.



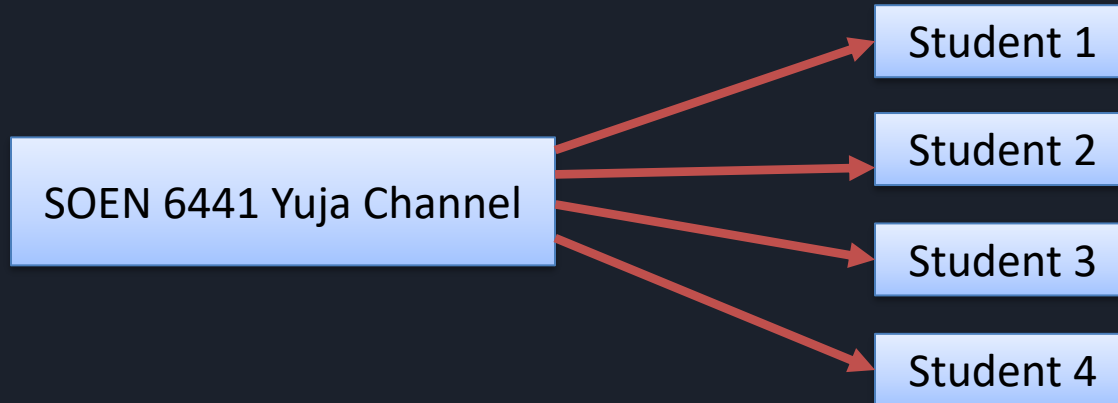
Advantages of Observer Design Pattern

- ✓ **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.

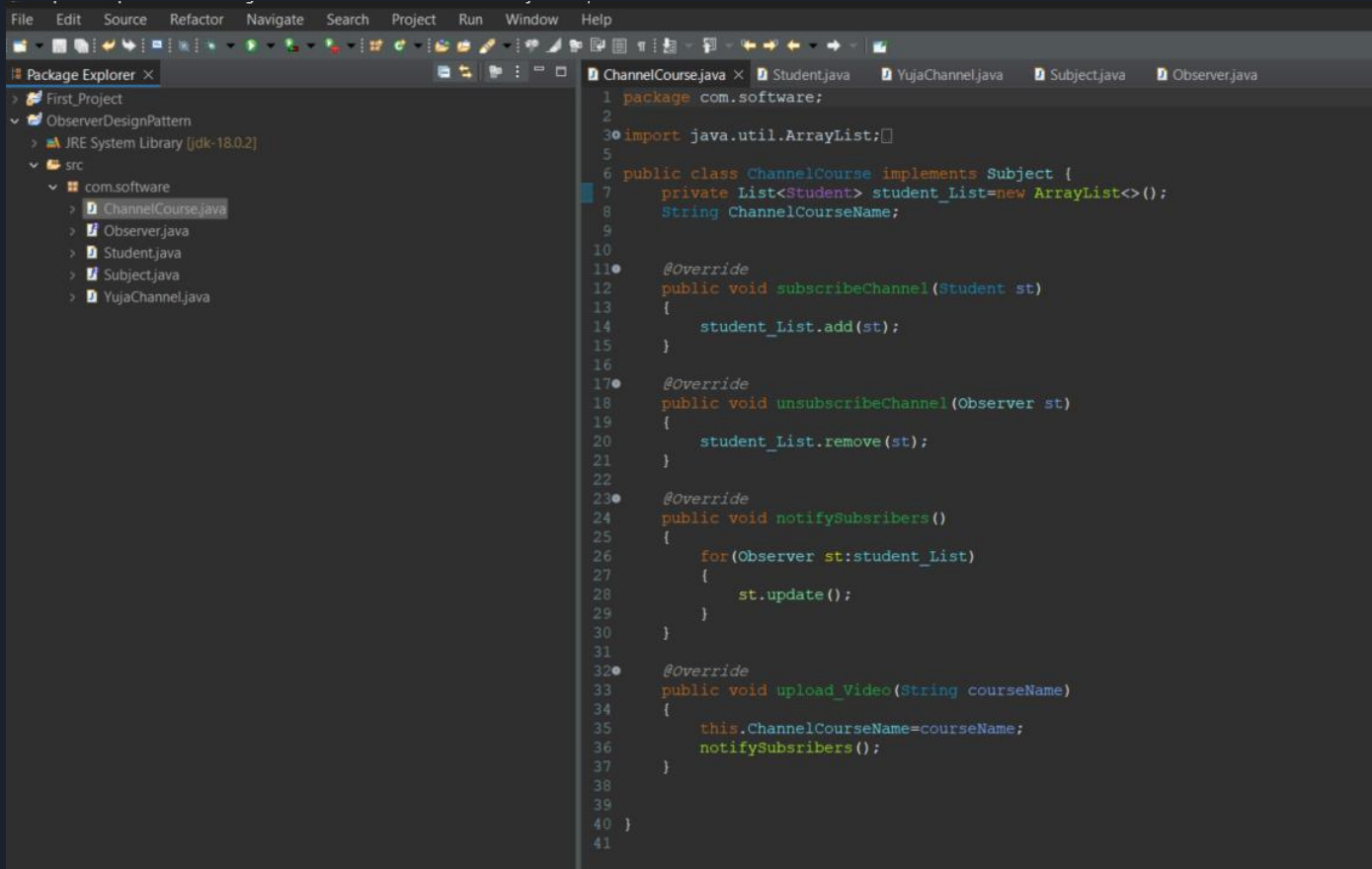
Example of Observer Design Pattern

Yuja Channel

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



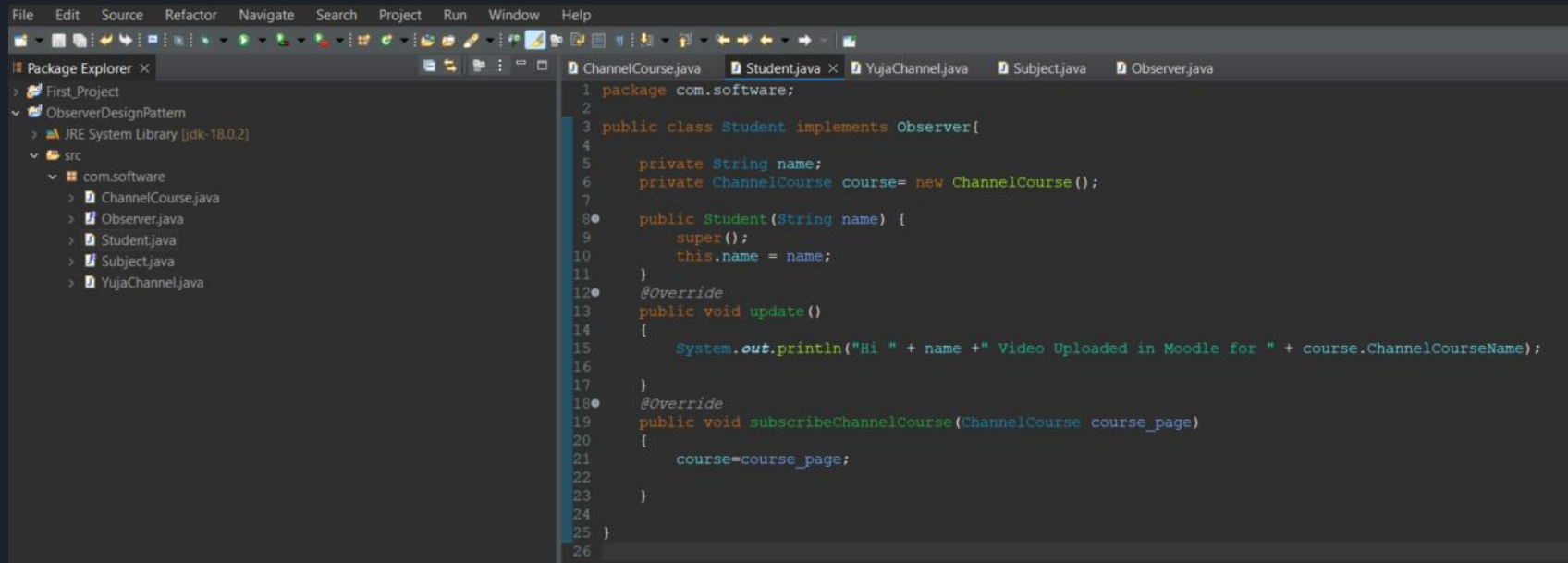
Java sample code for observer design pattern



```
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer X
First_Project
ObserverDesignPattern
JRE System Library [jdk-18.0.2]
src
com.software
ChannelCourse.java
Observer.java
Student.java
Subject.java
YujaChannel.java

ChannelCourse.java x Student.java YujaChannel.java Subject.java Observer.java
1 package com.software;
2
3 import java.util.ArrayList;
4
5
6 public class ChannelCourse implements Subject {
7     private List<Student> student_list=new ArrayList<>();
8     String ChannelCourseName;
9
10
11     @Override
12     public void subscribeChannel(Student st)
13     {
14         student_list.add(st);
15     }
16
17     @Override
18     public void unsubscribeChannel(Observer st)
19     {
20         student_list.remove(st);
21     }
22
23     @Override
24     public void notifySubscribers()
25     {
26         for(Observer st:student_list)
27         {
28             st.update();
29         }
30     }
31
32     @Override
33     public void upload_Video(String courseName)
34     {
35         this.ChannelCourseName=courseName;
36         notifySubscribers();
37     }
38
39
40 }
41
```

Java sample code for observer design pattern

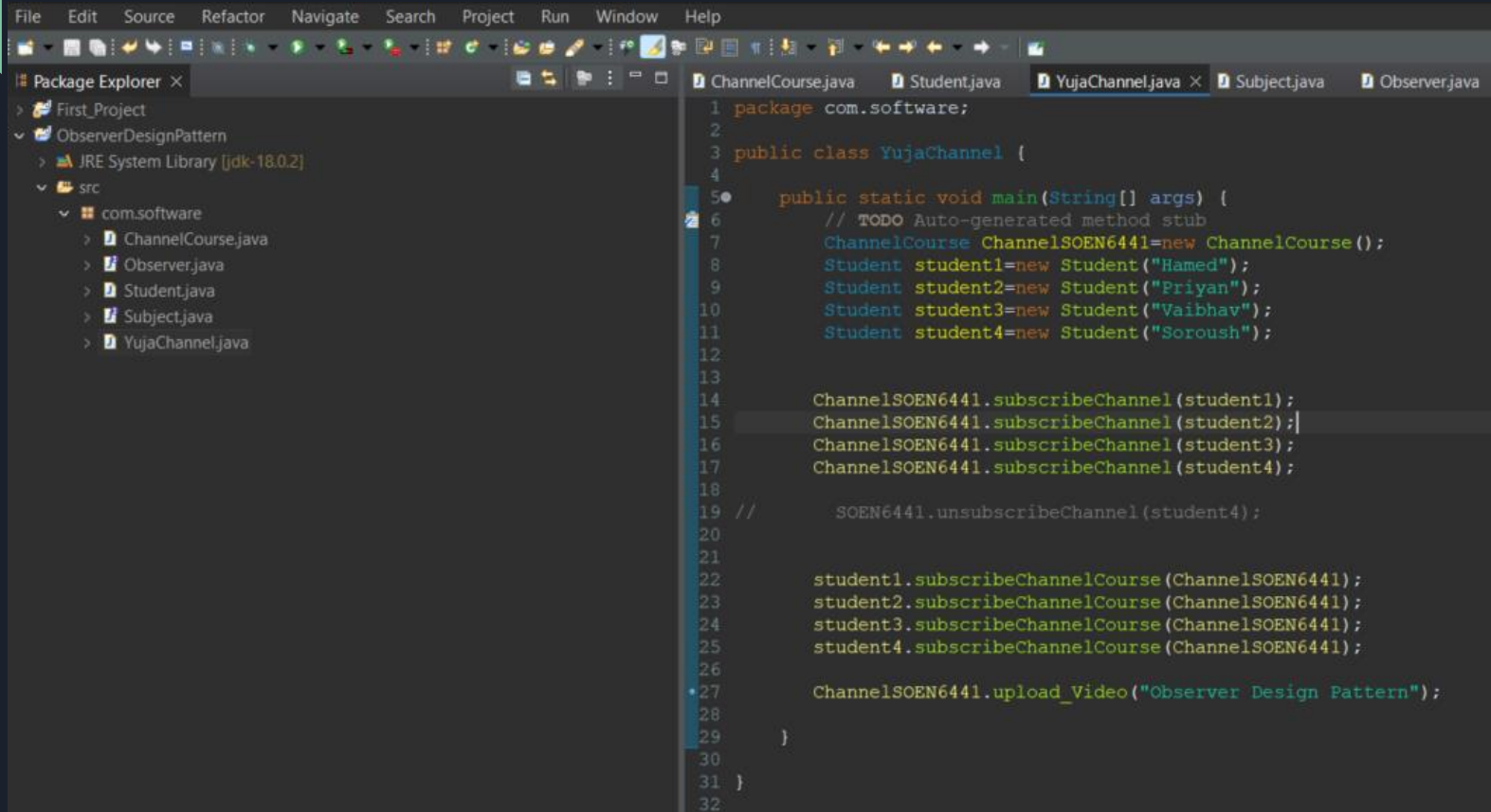


The screenshot shows an IDE with the following components:

- Package Explorer:** Displays a project structure with a package `com.software` containing files `ChannelCourse.java`, `Observer.java`, `Student.java`, `Subject.java`, and `YujaChannel.java`.
- Editor:** Shows the `Student.java` file with the following code:

```
1 package com.software;
2
3 public class Student implements Observer{
4
5     private String name;
6     private ChannelCourse course= new ChannelCourse();
7
8     public Student(String name) {
9         super();
10        this.name = name;
11    }
12    @Override
13    public void update()
14    {
15        System.out.println("Hi " + name + " Video Uploaded in Moodle for " + course.ChannelCourseName);
16    }
17
18    @Override
19    public void subscribeChannelCourse(ChannelCourse course_page)
20    {
21        course=course_page;
22    }
23
24
25 }
26
```

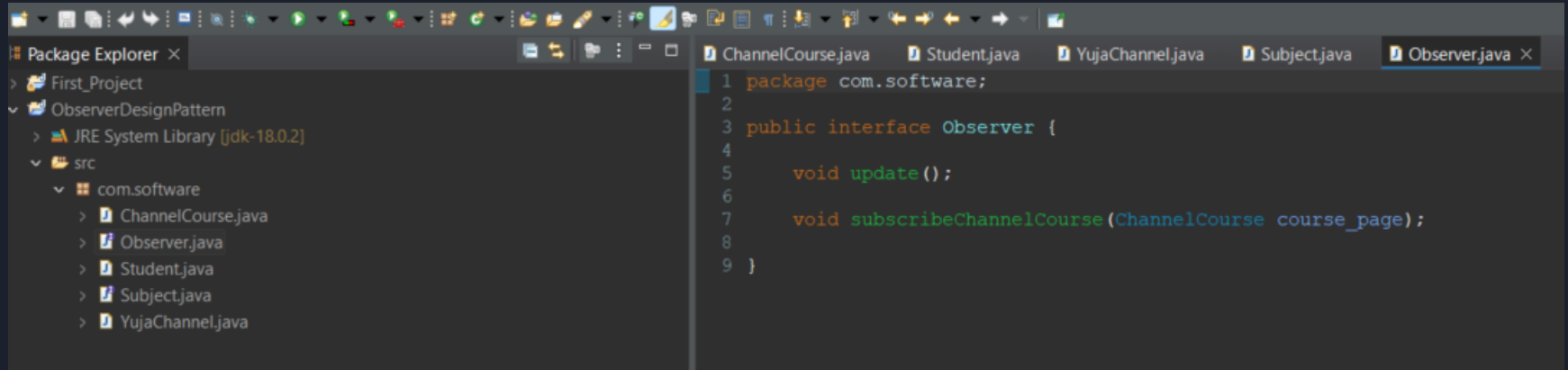
Java sample code for observer design pattern



The screenshot displays an IDE with the Package Explorer on the left and the Source Editor on the right. The Package Explorer shows a project named 'First_Project' with a package 'ObserverDesignPattern' containing a 'src' folder. Inside 'src', there is a package 'com.software' with five Java files: 'ChannelCourse.java', 'Observer.java', 'Student.java', 'Subject.java', and 'YujaChannel.java'. The Source Editor shows the code for 'YujaChannel.java'.

```
1 package com.software;
2
3 public class YujaChannel {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         ChannelCourse ChannelSOEN6441=new ChannelCourse();
8         Student student1=new Student("Hamed");
9         Student student2=new Student("Priyan");
10        Student student3=new Student("Vaibhav");
11        Student student4=new Student("Soroush");
12
13
14        ChannelSOEN6441.subscribeChannel(student1);
15        ChannelSOEN6441.subscribeChannel(student2);
16        ChannelSOEN6441.subscribeChannel(student3);
17        ChannelSOEN6441.subscribeChannel(student4);
18
19        // SOEN6441.unsubscribeChannel(student4);
20
21
22        student1.subscribeChannelCourse(ChannelSOEN6441);
23        student2.subscribeChannelCourse(ChannelSOEN6441);
24        student3.subscribeChannelCourse(ChannelSOEN6441);
25        student4.subscribeChannelCourse(ChannelSOEN6441);
26
27        ChannelSOEN6441.upload_Video("Observer Design Pattern");
28    }
29
30
31 }
32
```

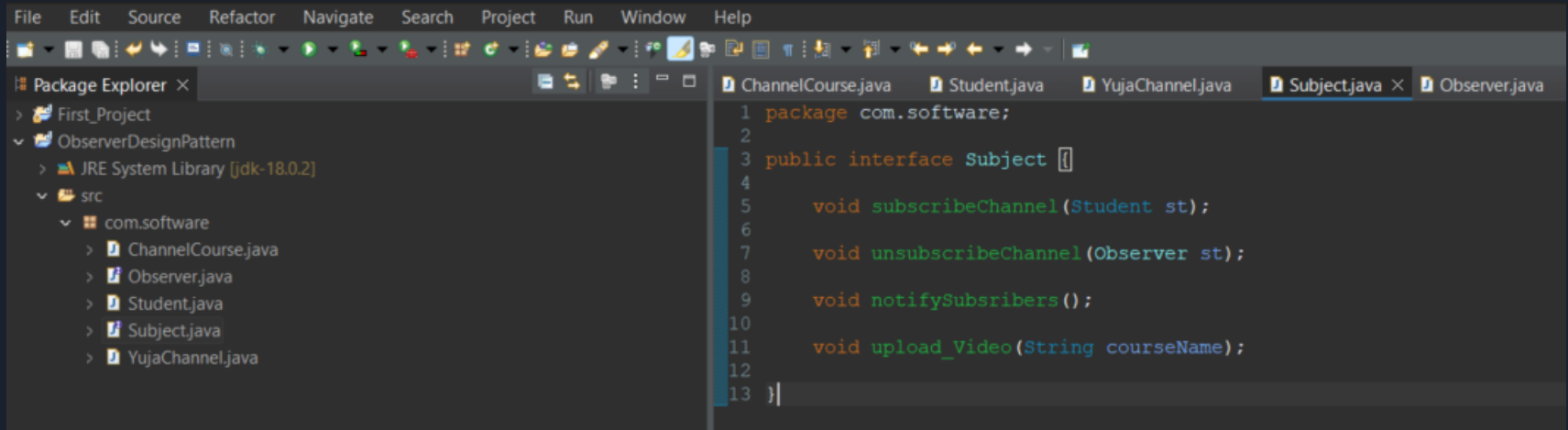

Java sample code for observer design pattern



The screenshot displays an IDE interface with a dark theme. On the left, the Package Explorer shows a project named 'First_Project' containing a package 'ObserverDesignPattern'. Inside this package, there is a 'src' folder which contains a 'com.software' package. The 'com.software' package lists several Java files: 'ChannelCourse.java', 'Observer.java', 'Student.java', 'Subject.java', and 'YujaChannel.java'. The main editor area on the right shows the code for 'Observer.java'. The code defines a 'public interface Observer' with two methods: 'void update();' and 'void subscribeChannelCourse(ChannelCourse course_page);'. The code is as follows:

```
1 package com.software;
2
3 public interface Observer {
4
5     void update();
6
7     void subscribeChannelCourse(ChannelCourse course_page);
8
9 }
```

Java sample code for observer design pattern



```
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer X
First_Project
ObserverDesignPattern
  JRE System Library [jdk-18.0.2]
  src
    com.software
      ChannelCourse.java
      Observer.java
      Student.java
      Subject.java
      YujaChannel.java
ChannelCourse.java Student.java YujaChannel.java Subject.java X Observer.java
1 package com.software;
2
3 public interface Subject {
4
5     void subscribeChannel(Student st);
6
7     void unsubscribeChannel(Observer st);
8
9     void notifySubscribers();
10
11     void upload_Video(String courseName);
12
13 }
```



**Thank You
For Your Attention**



Reference

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