```java
// Observer (Subscriber)
interface Subscriber {
    void update(String channelName, String videoTitle);
}
```

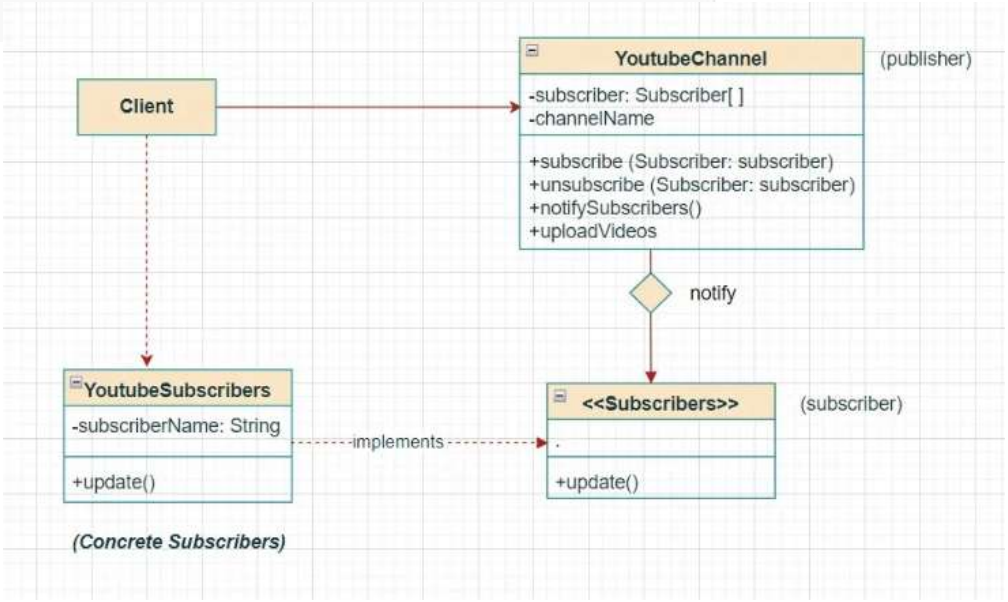
```java
// Concrete Observer (YouTube Subscriber)
class YouTubeSubscriber implements Subscriber {
    private String subscriberName;

    public YouTubeSubscriber(String subscriberName) {
        this.subscriberName = subscriberName;
    }

     @Override
    public void update(String channelName, String videoTitle) {
        System.out.println(subscriberName
                + " received a new video notification from "
                + channelName + ": " + videoTitle);
    }
}
```

```java
// Subject (YouTube Channel)
class YouTubeChannel {
    private List<Subscriber> subscribers = new ArrayList<>();
    private String channelName;

    // Creating a channel with name
    public YouTubeChannel(String channelName) {
        this.channelName = channelName;
    }

    // Subscribing the Channel
    public void subscribe(Subscriber subscriber) {
        subscribers.add(subscriber);
    }

    // UnSubscribing the Channel
    public void unsubscribe(Subscriber subscriber) {
        subscribers.remove(subscriber);
    }

    // Notifying all the subscriber of channel
    public void notifySubscribers(String videoTitle) {
        for (Subscriber subscriber : subscribers) {
            subscriber.update(channelName, videoTitle);
        }
    }

    // Channel Uploading a video
    public void uploadVideo(String videoTitle) {
        System.out.println("New video uploaded: " + videoTitle);
        notifySubscribers(videoTitle);
    }
}
```

1. **Client:** Client refers to the component or module that utilizes the observer pattern to achieve the desired behavior. It is responsible for creating the subject object, attaching observers to the subject, and interacting with the subject to trigger updates and receive notifications.
2. **Subject (also known as Publisher/Observable):** This is the object that *maintains a list of observers and provides methods* to attach, detach, and notify observers. The subject is responsible for managing the observers and notifying them of any changes in its state. *Example of Subject:* ***YouTube Channel***
3. **Observer (also known as Subscriber):** This is the ***interface or abstract*** class that defines the contract for the observers. It declares a single method (e.g., `update`) that is called by the subject when there is a state change. *Example of Observer:* ***Subscriber***
4. **Concrete Observer:** This is the concrete implementation of the Observer interface. It represents the actual observers that subscribe to and receive updates from the subject. Each concrete observer provides its own implementation of the `update` method to handle the received updates. *Example of Concrete Observer:* ***Subscriber of YouTube Channel***

. . .

```
public class Client {
    public static void main(String[] args) {
        // Create a YouTube channel
        YouTubeChannel channel = new YouTubeChannel("TechTalks");

        // Create subscribers
        Subscriber subscriber1 = new YouTubeSubscriber("Mridul");
        Subscriber subscriber2 = new YouTubeSubscriber("Promi");
        Subscriber subscriber3 = new YouTubeSubscriber("Siam");

        // Subscribers subscribing to the channel
        channel.subscribe(subscriber1);
        channel.subscribe(subscriber2);
        channel.subscribe(subscriber3);

        // Channel uploads a new video
        channel.uploadVideo("Introduction to Java Programming");

 // Output:
// New video uploaded: Introduction to Java Programming
// Mridul received a new video notification from TechTalks: Introduction to Java
// Promi received a new video notification from TechTalks: Introduction to Java
// Siam received a new video notification from TechTalks: Introduction to Java P

        // Promi Decide to unsubscribe the channel
        channel.unsubscribe(subscriber2);

        // Channel uploads another video
        channel.uploadVideo("Advanced Java Concepts");

// Output:
// New video uploaded: Advanced Java Concepts
// Mridul received a new video notification from TechTalks: Advanced Java Concep
// Siam received a new video notification from TechTalks: Advanced Java Concepts
// Promi didn't receive any notification as she unsubscribed
    }
}
```



```java
// Step - 1 ; State
public interface DeliveryState {
   void Status();
}
```

```java
// Step - 2 : Concrete State
public class OrderedState implements DeliveryState {
    @Override
    public void Status() {
        System.out.println("Order State Running");
    }
}
```

```java
// Step - 2 : Concrete State
public class DeliveredState implements DeliveryState {
    @Override
    public void Status() {
        System.out.println("Switching to Delivery State ....");
    }
}
```

```java
// Step - 2 : Concrete State
public class ReceivedState implements DeliveryState{
    @Override
    public void Status() {
        System.out.println("Switching to Receiving State ....");
    }
}
```

```java
// Step -  3 : Context
public class DeliverySystem {
    private DeliveryState state;

    public DeliverySystem() {
        this.state = new OrderedState();
        // Initially on Ordered State
    }
    // swtiching to another state
    public void setState(DeliveryState state) {
        this.state = state;
    }

    public void status() {
        state.Status();
    }
}
```

```java
public class Client {
    public static void main(String[] args) {

        // Creating a Delivery System Context
        DeliverySystem system = new DeliverySystem();
        system.status(); // Initially on Ordered State

        // switch to another State
        system.setState(new DeliveredState());
        system.status(); // now state switch to Delivered State

        // again Move to another State
        system.setState(new ReceivedState());
        system.status(); // now state switch to Received State

        /*
        Order State Running
        Switching to Delivery State ....
        Switching to Receiving State ....
         */
    }
}
```

```java
// Step - 1 : Receiver
public class Light {
    public void turnOn() {
        System.out.println("Light is turned on");
    }

    public void turnOff() {
        System.out.println("Light is turned off");
    }
}
```

```java
// Step - 2: Command
public interface Command {
    void execute();
}
```

```java
// Step - 3 : Concrete Command
public class LightOnCommand implements Command {
    private Light light;

    public LightOnCommand(Light light) {
        this.light = light;
    }

    public void execute() {
        light.turnOn();
    }
}
```

```java
// Step - 4 : Invoker
public class RemoteControl {
    private Command command;

    public void setCommand(Command command) {
        this.command = command;
    }

    public void pressButton() {
        command.execute();
    }
}
```

```java
// Step - 3 : Concrete Command
public class LightOffCommand implements Command {
    private Light light;

    public LightOffCommand(Light light) {
        this.light = light;
    }

    public void execute() {
        light.turnOff();
    }
}
```

```java
// Step - 5 : Client
public class Client {
    public static void main(String[] args) {
        // Create the receiver objects
        Light livingRoomLight = new Light();

   // Create the concrete command objects and associate them with receivers
        Command lightOnCommand = new LightOnCommand(livingRoomLight);
        Command lightOffCommand = new LightOffCommand(livingRoomLight);

        // Create the invoker (remote control)
        RemoteControl remoteControl = new RemoteControl();

        // Associate commands with the remote control buttons
        remoteControl.setCommand(lightOnCommand); // Light On button
        remoteControl.pressButton();
        // Executes LightOnCommand's execute() method

        remoteControl.setCommand(lightOffCommand); // Light Off button
        remoteControl.pressButton();
        // Executes LightOffCommand's execute() method
    }
}
```

```java
// Visitor interface
public interface DocumentVisitor {
    void visit(PdfDocument document);
    void visit(WordDocument document);
    void visit(PlainTextDocument document);
}
```

```java
// Element interface
public interface Document {
    void accept(DocumentVisitor visitor);
}
```

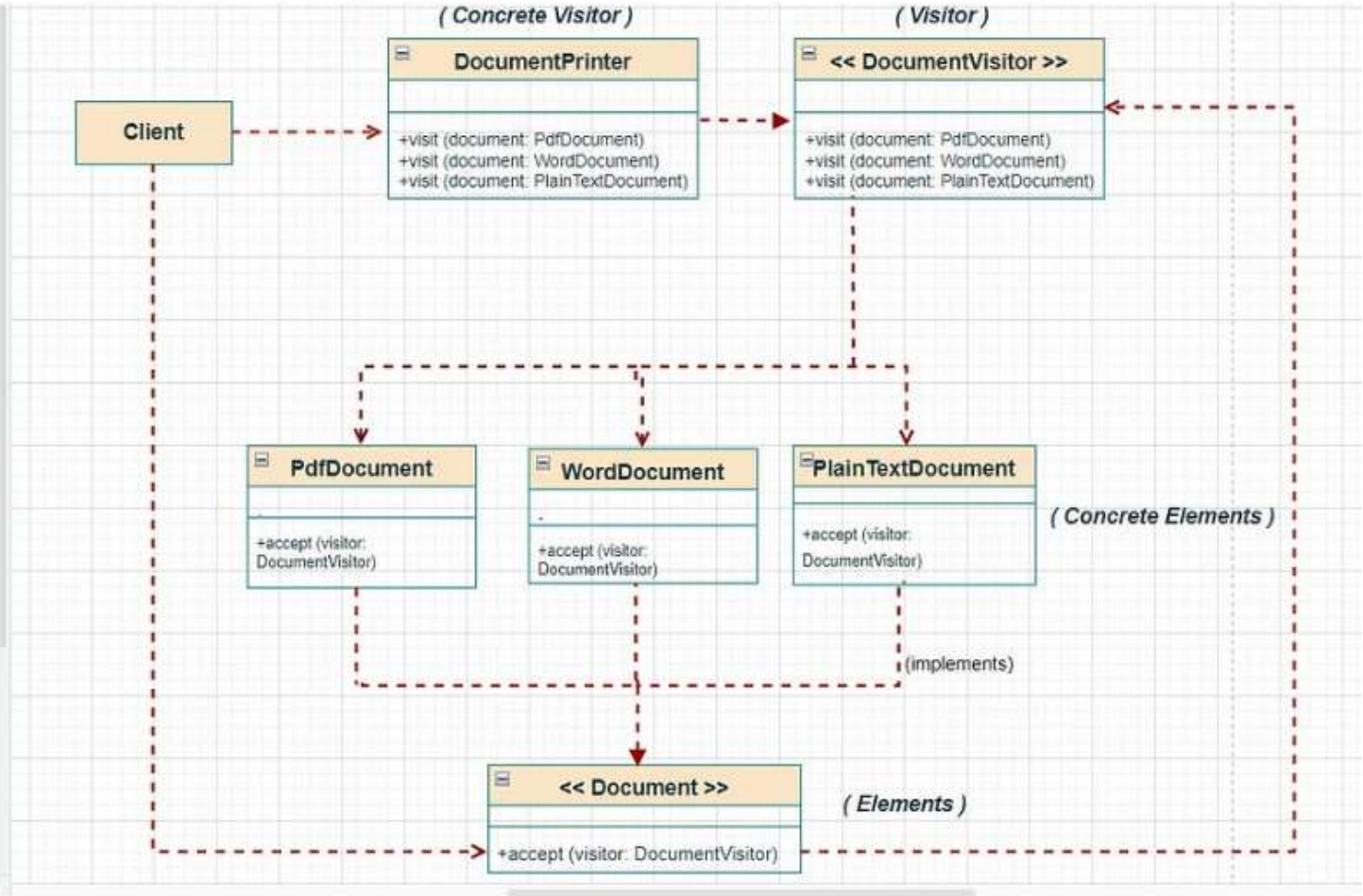
```java
// ConcreteVisitor
public class DocumentPrinter implements DocumentVisitor {
    @Override
    public void visit(PdfDocument document) {
        System.out.println("Printing PDF document: "
                           + document.getTitle());
        // Print PDF-specific details
    }

    @Override
    public void visit(WordDocument document) {
        System.out.println("Printing Word document: "
                           + document.getTitle());
        // Print Word-specific details
    }

    @Override
    public void visit(PlainTextDocument document) {
        System.out.println("Printing Plain Text document: "
                           + document.getTitle());
        // Print Plain Text-specific details
    }
}
```

```java
// ConcreteElement
public class PdfDocument implements Document {
    private String title;

    public PdfDocument(String title) {
        this.title = title;
    }

    public String getTitle() {
        return title;
    }

    @Override
    public void accept(DocumentVisitor visitor) {
        visitor.visit(this);
    }
}
```

```java
// ConcreteElement
public class WordDocument implements Document {
    private String title;

    public WordDocument(String title) {
        this.title = title;
    }

    public String getTitle() {
        return title;
    }

    @Override
    public void accept(DocumentVisitor visitor) {
        visitor.visit(this);
    }
}
```

```java
// ConcreteElement
public class PlainTextDocument implements Document {
    private String title;

    public PlainTextDocument(String title) {
        this.title = title;
    }

    public String getTitle() {
        return title;
    }

    @Override
    public void accept(DocumentVisitor visitor) {
        visitor.visit(this);
    }
}
```

```java
public class Client {

    public static void main(String[] args) {
        Document pdfDocument = new PdfDocument("Sample.pdf");
        Document wordDocument = new WordDocument("Sample.docx");
        Document plainTextDocument = new PlainTextDocument("Sample.txt");

        DocumentVisitor documentPrinter = new DocumentPrinter();

        pdfDocument.accept(documentPrinter);
        // Output: Printing PDF document: Sample.pdf
        wordDocument.accept(documentPrinter);
        // Output: Printing Word document: Sample.docx
        plainTextDocument.accept(documentPrinter);
        // Output: Printing Plain Text document: Sample.txt
    }
}
```



```java
// Originator
class Browser {
    private String currentPage;

    public void goToPage(String page) {
        System.out.println("Navigating to " + page);
        currentPage = page;
    }

    public Memento save() {
        return new Memento(currentPage);
    }

    public void restore(Memento memento) {
        currentPage = memento.getState();
        System.out.println("Restored to page: " + currentPage);
    }

    public String getCurrentPage() {
        return currentPage;
    }
}
```

```java
// Memento
class Memento {
    private final String state;

    public Memento(String state) {
        this.state = state;
    }

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

```java
// Caretaker
class BrowserHistory {
    private List<Memento> history = new ArrayList<>();
    private int currentIndex = -1;

    public void save(Memento memento) {
        history.add(memento);
        currentIndex = history.size() - 1;
    }

    public Memento undo() {
        if (currentIndex > 0) {
            currentIndex--;
            return history.get(currentIndex);
        }
        return null;
    }

    public Memento redo() {
        if (currentIndex < history.size() - 1) {
            currentIndex++;
            return history.get(currentIndex);
        }
        return null;
    }
}
```

```
Overall Output :
   Navigating to www.example.com
   Navigating to www.openai.com
   Current page: www.openai.com
   Restored to page: www.example.com
   Current page: www.example.com
   Restored to page: www.openai.com
   Current page: www.openai.com
```

```java
public class Client {
    public static void main(String[] args) {
        Browser browser = new Browser();
        BrowserHistory history = new BrowserHistory();

        browser.goToPage("www.example.com");
        history.save(browser.save());

        browser.goToPage("www.openai.com");
        history.save(browser.save());

        System.out.println("Current page: " + browser.getCurrentPage());
        // Output: www.openai.com

        browser.restore(history.undo());
        System.out.println("Current page: " + browser.getCurrentPage());
        // Output: www.example.com

        browser.restore(history.redo());
        System.out.println("Current page: " + browser.getCurrentPage());
        // Output: www.openai.com
    }
}
```