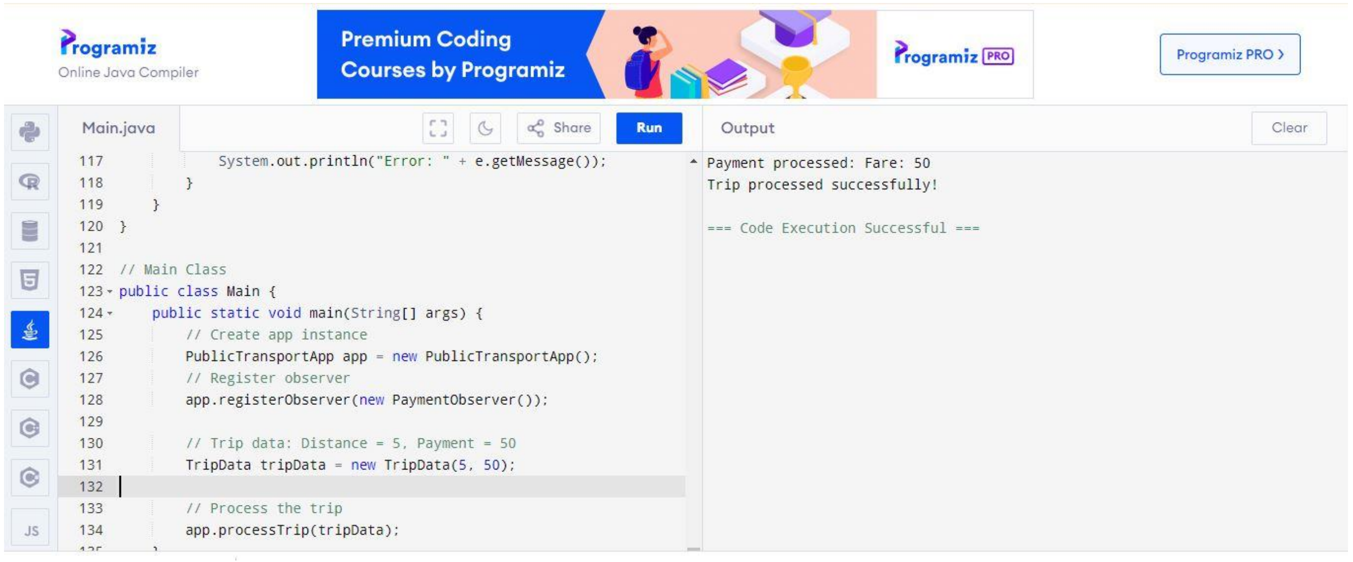
Code

```
import java.util.ArrayList;

import java.util.List;


// Observer Pattern

interface Observer {

    void update(String message);

}


class PaymentObserver implements Observer {

    @Override

    public void update(String message) {

        System.out.println("Payment processed: " + message);

    }

}


// Subject Class

class Subject {

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


    public void registerObserver(Observer observer) {

        observers.add(observer);

    }


    public void notifyObservers(String message) {

        for (Observer observer : observers) {

            observer.update(message);

        }
    }
}

// Pipe and Filter
class Pipe {
    private List<Filter> filters = new ArrayList<>();

    public void addFilter(Filter filter) {
        filters.add(filter);
    }

    public TripData process(TripData data) throws Exception {
        for (Filter filter : filters) {
            data = filter.execute(data);
        }
        return data;
    }
}

interface Filter {
    TripData execute(TripData data) throws Exception;
}

class FareFilter implements Filter {
    @Override
    public TripData execute(TripData data) {
        data.setFare(data.getDistance() * 10); // Fare = Distance * 10
        return data;
    }
}


class ValidationFilter implements Filter {
    @Override
    public TripData execute(TripData data) throws Exception {
        if (data.getPayment() < data.getFare()) {
            throw new Exception("Insufficient payment!");
        }
        return data;
    }
}


// Data Model
class TripData {
    private int distance;
    private int payment;
    private int fare;

    public TripData(int distance, int payment) {
        this.distance = distance;
        this.payment = payment;
    }

    public int getDistance() {
        return distance;
    }
    public int getPayment() {
        return payment;
    }

    public void setFare(int fare) {
        this.fare = fare;
    }

    public int getFare() {
        return fare;
    }
}

// Layered Architecture
class PublicTransportApp {
    private Subject subject = new Subject();
    private Pipe pipe = new Pipe();

    public PublicTransportApp() {
        pipe.addFilter(new FareFilter());
        pipe.addFilter(new ValidationFilter());
    }

    public void registerObserver(Observer observer) {
        subject.registerObserver(observer);
    }

    public void processTrip(TripData tripData) {
        try {
            TripData processedData = pipe.process(tripData);
            subject.notifyObservers("Fare: " + processedData.getFare());
            System.out.println("Trip processed successfully!");
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}

// Main Class
public class Main {
    public static void main(String[] args) {
        // Create app instance
        PublicTransportApp app = new PublicTransportApp();
        // Register observer
        app.registerObserver(new PaymentObserver());

        // Trip data: Distance = 5, Payment = 50
        TripData tripData = new TripData(5, 50);

        // Process the trip
        app.processTrip(tripData);
    }
}
```



```java
                System.out.println("Error: " + e.getMessage());
            }
        }
    }

    // Main Class
    public class Main {
        public static void main(String[] args) {
            // Create app instance
            PublicTransportApp app = new PublicTransportApp();
            // Register observer
            app.registerObserver(new PaymentObserver());

            // Trip data: Distance = 5, Payment = 50
            TripData tripData = new TripData(5, 50);

            // Process the trip
            app.processTrip(tripData);
```

```
Payment processed: Fare: 50
Trip processed successfully!

=== Code Execution Successful ===
```

In this code I use layer architecture

Pipe and filter

And observer pattern

From above 3 I choose public transport.