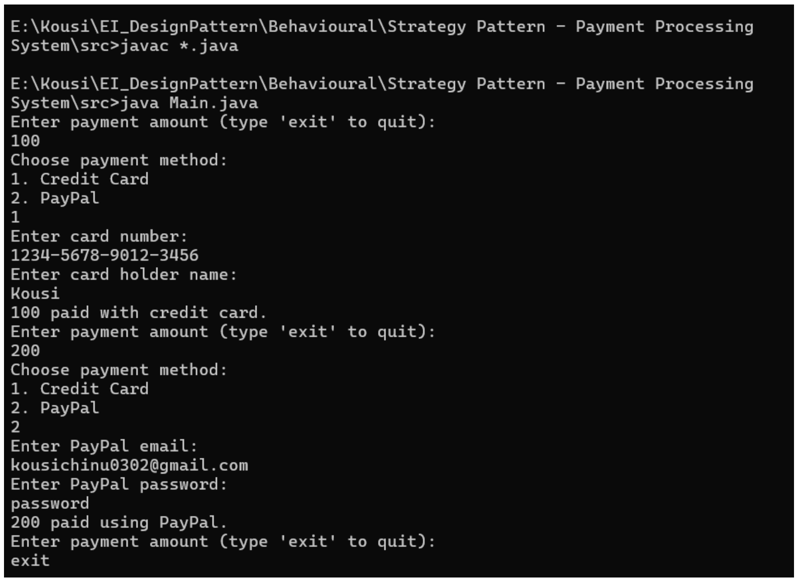
Exercise 1: Design Patterns

1. Behavioral Patterns

Strategy Pattern

- **Use Case**: Payment Gateway
  - **Explanation**: The Strategy Pattern allows a system to select an algorithm or strategy at runtime. In the context of a payment gateway, different payment methods (e.g., credit card, PayPal, bank transfer) can be used interchangeably. The payment gateway can choose the appropriate payment strategy based on the user's choice.
  - **Example**: If a customer chooses to pay using a credit card, the payment gateway will use the Credit Card Payment Strategy to process the payment. If the customer chooses PayPal, the PayPal Payment Strategy will be used instead.

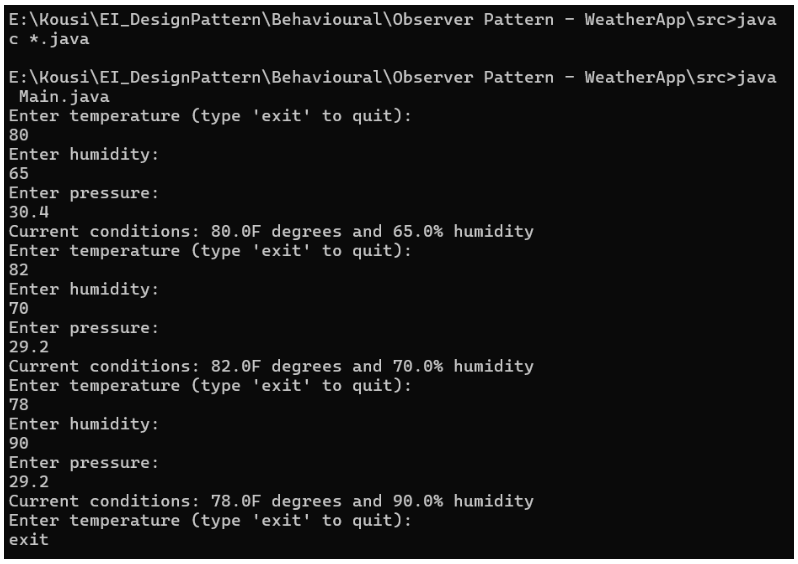


```
E:\Kousi\EI_DesignPattern\Behavioural\Strategy Pattern - Payment Processing
System\src>javac *.java

E:\Kousi\EI_DesignPattern\Behavioural\Strategy Pattern - Payment Processing
System\src>java Main.java
Enter payment amount (type 'exit' to quit):
100
Choose payment method:
1. Credit Card
2. PayPal
1
Enter card number:
1234-5678-9012-3456
Enter card holder name:
Kousi
100 paid with credit card.
Enter payment amount (type 'exit' to quit):
200
Choose payment method:
1. Credit Card
2. PayPal
2
Enter PayPal email:
kousichinu0302@gmail.com
Enter PayPal password:
password
200 paid using PayPal.
Enter payment amount (type 'exit' to quit):
exit
```

Observer Pattern

- **Use Case**: Weather App
  - **Explanation**: The Observer Pattern is used to notify multiple observers (users) about changes in the state of an object (weather conditions). When the weather changes, the app updates all subscribed users with the new weather information.
  - **Example**: A user subscribes to weather updates for New York City. Whenever the weather changes in New York City, the app notifies the user of the new weather conditions.



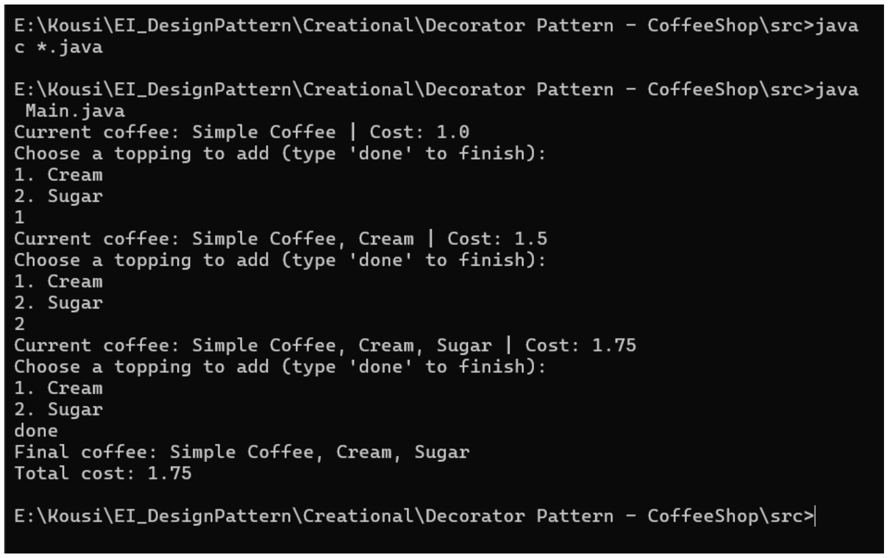
```
E:\Kousi\EI_DesignPattern\Behavioural\Observer Pattern - WeatherApp\src>java
c *.java

E:\Kousi\EI_DesignPattern\Behavioural\Observer Pattern - WeatherApp\src>java
 Main.java
Enter temperature (type 'exit' to quit):
80
Enter humidity:
65
Enter pressure:
30.4
Current conditions: 80.0F degrees and 65.0% humidity
Enter temperature (type 'exit' to quit):
82
Enter humidity:
70
Enter pressure:
29.2
Current conditions: 82.0F degrees and 70.0% humidity
Enter temperature (type 'exit' to quit):
78
Enter humidity:
90
Enter pressure:
29.2
Current conditions: 78.0F degrees and 90.0% humidity
Enter temperature (type 'exit' to quit):
exit
```

2. Structural Patterns

Decorator Pattern

- **Use Case**: Coffee Shop
  - **Explanation**: The Decorator Pattern allows adding new responsibilities to an object dynamically without altering its structure. In a coffee shop scenario, this pattern can be used to add toppings (e.g., milk, sugar, whipped cream) to a coffee.
  - **Example**: You have a basic coffee object. Using the Decorator Pattern, you can dynamically add milk or sugar to this coffee object, resulting in a new coffee variant without changing the original coffee class.



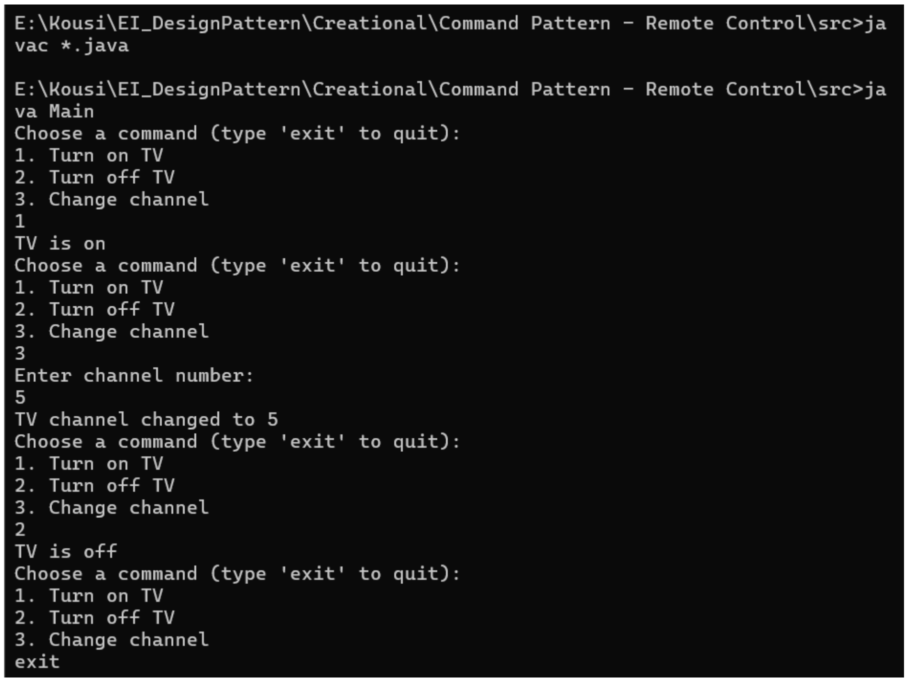
```
E:\Kousi\EI_DesignPattern\Creational\Decorator Pattern - CoffeeShop\src>java
c *.java

E:\Kousi\EI_DesignPattern\Creational\Decorator Pattern - CoffeeShop\src>java
 Main.java
Current coffee: Simple Coffee | Cost: 1.0
Choose a topping to add (type 'done' to finish):
1. Cream
2. Sugar
1
Current coffee: Simple Coffee, Cream | Cost: 1.5
Choose a topping to add (type 'done' to finish):
1. Cream
2. Sugar
2
Current coffee: Simple Coffee, Cream, Sugar | Cost: 1.75
Choose a topping to add (type 'done' to finish):
1. Cream
2. Sugar
done
Final coffee: Simple Coffee, Cream, Sugar
Total cost: 1.75

E:\Kousi\EI_DesignPattern\Creational\Decorator Pattern - CoffeeShop\src>
```

Command Pattern

- **Use Case**: Remote Control
  - **Explanation**: The Command Pattern encapsulates a request as an object, allowing users to parameterize clients with queues, requests, and operations. In a remote control system, each button press can be encapsulated as a command object.
  - **Example**: The remote control has buttons for different actions (e.g., turn on TV, change channel). Each button's action is encapsulated in a Command object that the remote control executes when the button is pressed.

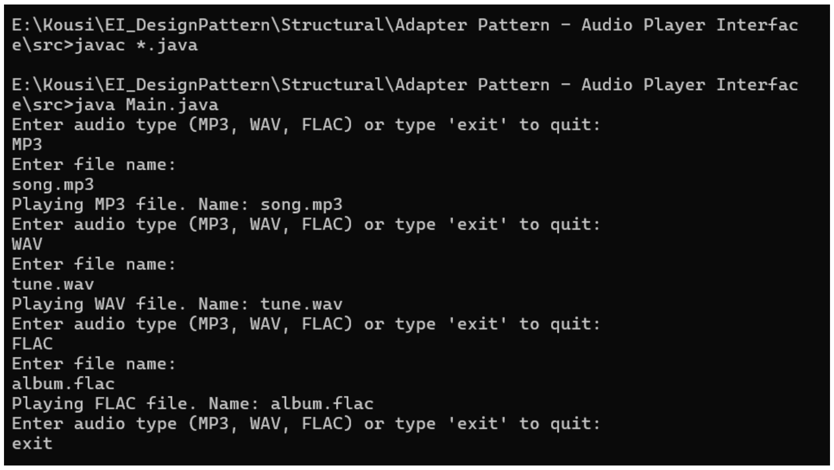


```
E:\Kousi\EI_DesignPattern\Creational\Command Pattern - Remote Control\src>ja
vac *.java

E:\Kousi\EI_DesignPattern\Creational\Command Pattern - Remote Control\src>ja
va Main
Choose a command (type 'exit' to quit):
1. Turn on TV
2. Turn off TV
3. Change channel
1
TV is on
Choose a command (type 'exit' to quit):
1. Turn on TV
2. Turn off TV
3. Change channel
3
Enter channel number:
5
TV channel changed to 5
Choose a command (type 'exit' to quit):
1. Turn on TV
2. Turn off TV
3. Change channel
2
TV is off
Choose a command (type 'exit' to quit):
1. Turn on TV
2. Turn off TV
3. Change channel
exit
```

Adapter Pattern

- **Use Case**: Audio Player
  - **Explanation**: The Adapter Pattern allows an interface to be compatible with another interface. For an audio player that supports multiple audio formats, adapters can be used to convert different audio formats (e.g., MP3, WAV, FLAC) into a format that the audio player can handle.
  - **Example**: The audio player uses an MP3 adapter to play MP3 files and a WAV adapter to play WAV files. Each adapter translates the audio file format into a format that the player understands.

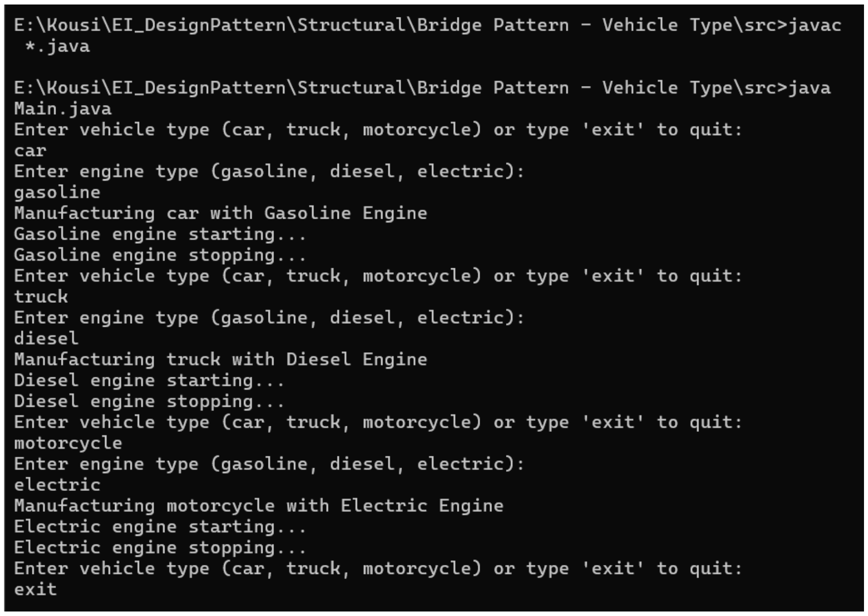


```
E:\Kousi\EI_DesignPattern\Structural\Adapter Pattern - Audio Player Interfac
e\src>javac *.java

E:\Kousi\EI_DesignPattern\Structural\Adapter Pattern - Audio Player Interfac
e\src>java Main.java
Enter audio type (MP3, WAV, FLAC) or type 'exit' to quit:
MP3
Enter file name:
song.mp3
Playing MP3 file. Name: song.mp3
Enter audio type (MP3, WAV, FLAC) or type 'exit' to quit:
WAV
Enter file name:
tune.wav
Playing WAV file. Name: tune.wav
Enter audio type (MP3, WAV, FLAC) or type 'exit' to quit:
FLAC
Enter file name:
album.flac
Playing FLAC file. Name: album.flac
Enter audio type (MP3, WAV, FLAC) or type 'exit' to quit:
exit
```

Bridge Pattern

- **Use Case**: Vehicle System
  - **Explanation**: The Bridge Pattern separates an abstraction from its implementation, allowing them to vary independently. For a vehicle system, this pattern can be used to separate vehicle types (cars, trucks, motorcycles) from engine types (gasoline, diesel, electric).
  - **Example**: You can have a `Car` class that uses different engine types (gasoline, electric). The Bridge Pattern allows you to combine different types of vehicles with various engine types without creating a large number of classes.



```
E:\Kousi\EI_DesignPattern\Structural\Bridge Pattern - Vehicle Type\src>javac
 *.java

E:\Kousi\EI_DesignPattern\Structural\Bridge Pattern - Vehicle Type\src>java
Main.java
Enter vehicle type (car, truck, motorcycle) or type 'exit' to quit:
car
Enter engine type (gasoline, diesel, electric):
gasoline
Manufacturing car with Gasoline Engine
Gasoline engine starting...
Gasoline engine stopping...
Enter vehicle type (car, truck, motorcycle) or type 'exit' to quit:
truck
Enter engine type (gasoline, diesel, electric):
diesel
Manufacturing truck with Diesel Engine
Diesel engine starting...
Diesel engine stopping...
Enter vehicle type (car, truck, motorcycle) or type 'exit' to quit:
motorcycle
Enter engine type (gasoline, diesel, electric):
electric
Manufacturing motorcycle with Electric Engine
Electric engine starting...
Electric engine stopping...
Enter vehicle type (car, truck, motorcycle) or type 'exit' to quit:
exit
```

Problem Statement: Rocket Launch Simulator

Overview

The Rocket Launch Simulator is a terminal-based application designed to simulate a rocket launch mission. The goal is to manage and track the rocket's progress through various stages, from pre-launch checks to orbit placement, with real-time updates and user interactions.

Key Features

1. **Pre-Launch Checks**:
   - The user initiates system checks by typing `start_checks`.
   - The simulator confirms if all systems are ready for launch.
2. **Launch**:
   - After the checks, the user can start the launch by typing `launch`.
   - The simulator updates the rocket's status every second, including stage, fuel level, altitude, and speed.
3. **Fast Forward**:
   - Users can advance the simulation by typing `fast_forward X`, where `X` represents the number of seconds to fast forward.
   - The simulator calculates and displays the rocket's status after the specified number of seconds.
4. **Stage Separation and Orbit Placement**:
   - The rocket progresses through different stages. When a stage completes, the simulator announces the transition.
   - The mission succeeds if the rocket reaches orbit; otherwise, it fails due to insufficient fuel.

Functional Requirements

1. **Pre-Launch Checks**:Output: "All systems are 'Go' for launch."
2. **Launch and Stage Updates**:

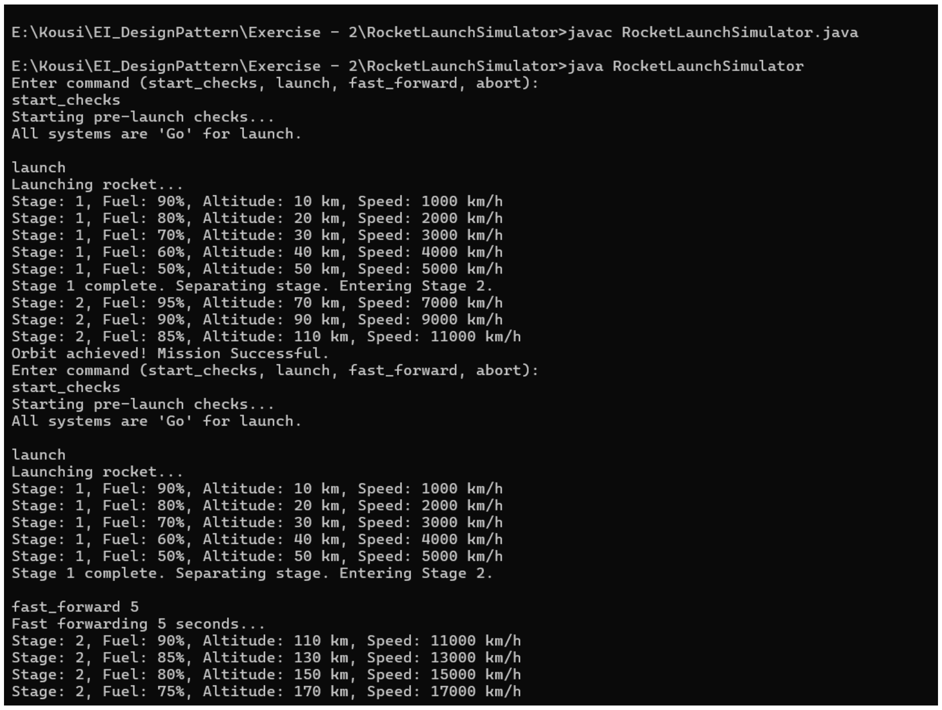
Output: Real-time updates showing:
`Stage: [current stage], Fuel: [current fuel], Altitude: [current altitude], Speed: [current speed]`

3. **Fast Forward**:
   - Output: Status after X seconds of fast forwarding.

Evaluation Criteria

1. **Code Quality**: Follow best practices and SOLID principles.
2. **Functionality**: Accurate simulation of the rocket launch process.

3. **Global Convention**: Code should be clear and maintainable.
4. **Gold Standards**: Implement logging, exception handling, and error management.
5. **Code Walkthrough**: Explain your code and design decisions.



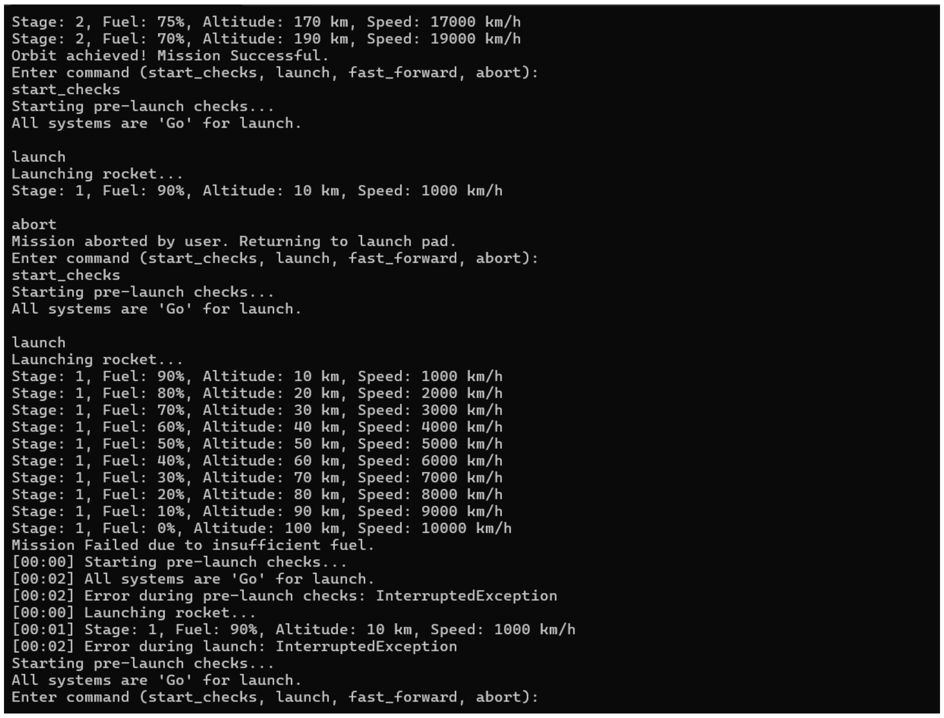
```
E:\Kousi\EI_DesignPattern\Exercise - 2\RocketLaunchSimulator>javac RocketLaunchSimulator.java

E:\Kousi\EI_DesignPattern\Exercise - 2\RocketLaunchSimulator>java RocketLaunchSimulator
Enter command (start_checks, launch, fast_forward, abort):
start_checks
Starting pre-launch checks...
All systems are 'Go' for launch.

launch
Launching rocket...
Stage: 1, Fuel: 90%, Altitude: 10 km, Speed: 1000 km/h
Stage: 1, Fuel: 80%, Altitude: 20 km, Speed: 2000 km/h
Stage: 1, Fuel: 70%, Altitude: 30 km, Speed: 3000 km/h
Stage: 1, Fuel: 60%, Altitude: 40 km, Speed: 4000 km/h
Stage: 1, Fuel: 50%, Altitude: 50 km, Speed: 5000 km/h
Stage 1 complete. Separating stage. Entering Stage 2.
Stage: 2, Fuel: 95%, Altitude: 70 km, Speed: 7000 km/h
Stage: 2, Fuel: 90%, Altitude: 90 km, Speed: 9000 km/h
Stage: 2, Fuel: 85%, Altitude: 110 km, Speed: 11000 km/h
Orbit achieved! Mission Successful.
Enter command (start_checks, launch, fast_forward, abort):
start_checks
Starting pre-launch checks...
All systems are 'Go' for launch.

launch
Launching rocket...
Stage: 1, Fuel: 90%, Altitude: 10 km, Speed: 1000 km/h
Stage: 1, Fuel: 80%, Altitude: 20 km, Speed: 2000 km/h
Stage: 1, Fuel: 70%, Altitude: 30 km, Speed: 3000 km/h
Stage: 1, Fuel: 60%, Altitude: 40 km, Speed: 4000 km/h
Stage: 1, Fuel: 50%, Altitude: 50 km, Speed: 5000 km/h
Stage 1 complete. Separating stage. Entering Stage 2.

fast_forward 5
Fast forwarding 5 seconds...
Stage: 2, Fuel: 90%, Altitude: 110 km, Speed: 11000 km/h
Stage: 2, Fuel: 85%, Altitude: 130 km, Speed: 13000 km/h
Stage: 2, Fuel: 80%, Altitude: 150 km, Speed: 15000 km/h
Stage: 2, Fuel: 75%, Altitude: 170 km, Speed: 17000 km/h
```

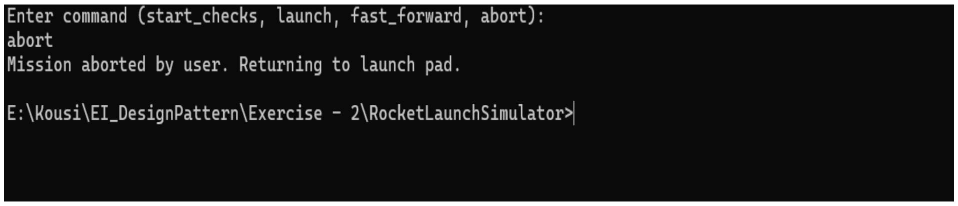


```
Stage: 2, Fuel: 75%, Altitude: 170 km, Speed: 17000 km/h
Stage: 2, Fuel: 70%, Altitude: 190 km, Speed: 19000 km/h
Orbit achieved! Mission Successful.
Enter command (start_checks, launch, fast_forward, abort):
start_checks
Starting pre-launch checks...
All systems are 'Go' for launch.

launch
Launching rocket...
Stage: 1, Fuel: 90%, Altitude: 10 km, Speed: 1000 km/h

abort
Mission aborted by user. Returning to launch pad.
Enter command (start_checks, launch, fast_forward, abort):
start_checks
Starting pre-launch checks...
All systems are 'Go' for launch.

launch
Launching rocket...
Stage: 1, Fuel: 90%, Altitude: 10 km, Speed: 1000 km/h
Stage: 1, Fuel: 80%, Altitude: 20 km, Speed: 2000 km/h
Stage: 1, Fuel: 70%, Altitude: 30 km, Speed: 3000 km/h
Stage: 1, Fuel: 60%, Altitude: 40 km, Speed: 4000 km/h
Stage: 1, Fuel: 50%, Altitude: 50 km, Speed: 5000 km/h
Stage: 1, Fuel: 40%, Altitude: 60 km, Speed: 6000 km/h
Stage: 1, Fuel: 30%, Altitude: 70 km, Speed: 7000 km/h
Stage: 1, Fuel: 20%, Altitude: 80 km, Speed: 8000 km/h
Stage: 1, Fuel: 10%, Altitude: 90 km, Speed: 9000 km/h
Stage: 1, Fuel: 0%, Altitude: 100 km, Speed: 10000 km/h
Mission Failed due to insufficient fuel.
[00:00] Starting pre-launch checks...
[00:02] All systems are 'Go' for launch.
[00:02] Error during pre-launch checks: InterruptedException
[00:00] Launching rocket...
[00:01] Stage: 1, Fuel: 90%, Altitude: 10 km, Speed: 1000 km/h
[00:02] Error during launch: InterruptedException
Starting pre-launch checks...
All systems are 'Go' for launch.
Enter command (start_checks, launch, fast_forward, abort):
```



```
Enter command (start_checks, launch, fast_forward, abort):
abort
Mission aborted by user. Returning to launch pad.

E:\Kousi\EI_DesignPattern\Exercise - 2\RocketLaunchSimulator>
```

Purpose

This exercise aims to test your ability to design and implement a complex system involving multiple states and interactions. It evaluates coding skills, problem-solving, and application of design patterns in a real-world scenario.