Assignment
Formal Methods in Software Engineering
(SE-313)

NED University of Engineering and Technology, Karachi

Department of Software Engineering

Group ID: *A-10*

Group Members:

Maria Ashfaq (SE-21005)

Fatima Zehra (SE-21028)

Batch: *2021*

Section: *A*

VDM – Elevator Control System

Scope

The Elevator Control System will effectively manage the operations of an elevator car within a building with floor numbers ranging from 0 to 10, as 0 being ground floor and 10th being the highest floor. The system will efficiently handle floor requests, allowing users to specify destinations within the floor range. To reduce the complexity, the scope of the system is confined to accommodate only one floor request at a time.

In this system, the initial floor of the elevator is recorded as a one-time configuration. Other operations on the system will not be able to go ahead until the initial floor is set.

The software will manage the elevator's current floor, providing real-time location information to users. It will record the current floor, requested floor, and the state of the elevator car (indicated by the Boolean variable "isMoving"). Initially, the current floor, requested floor are set to nil and isMoving state is set to false.

Elevator movement is controlled through commands such as "UP" to ascend by 1 floor, "DOWN" to descend by 1 floor, and "STOP" to halt the elevator. The control system also manages the opening and closing of elevator doors through commands like "OPEN" and "CLOSE".

The elevator car itself operates by moving up or down within the building and signalling the software each time a change of one floor has occurred. Upon receiving such a signal, the software records the new floor and provides response to the elevator car, indicating whether further changes are necessary to reach the requested floor. The principal goal of the Elevator Control System is to ensure efficient and secure control over elevator movement, respond promptly to user requests, manage door operations, and maintain accurate records of the elevator's state and location.

4 + 1 Architecture

The 4+1 view model is a way to explain how software-intensive systems are designed. It uses different views to show the system from various perspectives, considering the needs of end-users, developers, system engineers, and project managers.

There are four views in the model: process, logical, development, and physical. Additionally, the design is displayed using a few chosen use cases or scenarios called the "plus one" view. As a result, the model has 4+1 views. Hence the model is called the 4+1 Architectural View.

1. Logical View

The logical view focuses on how the system functions from the perspective of end-users. It involves using UML diagrams like class diagrams and state diagrams to illustrate the system's functionality.

The Logical View of Elevator Control System is demonstrated by the following Class Diagram;

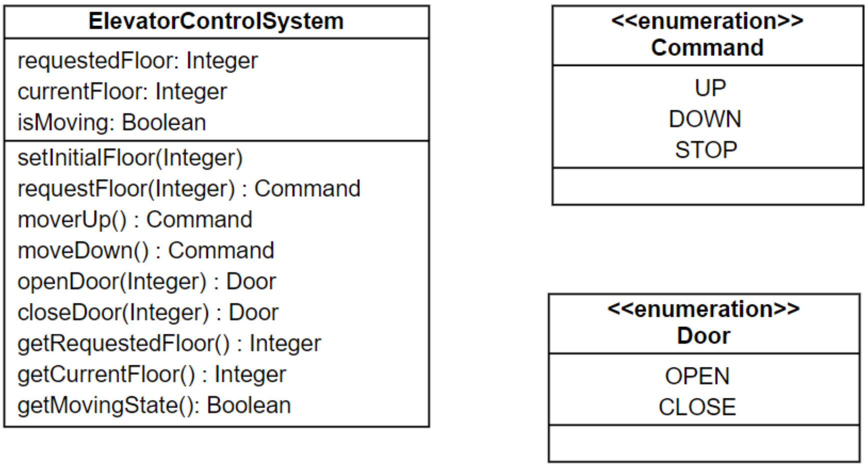


Figure 1. Class Diagram Elevator Control System

2. Process View

The process view emphasizes the real-time behaviour of the system, addressing its dynamic aspects. It elaborates on the system's processes and their communication. UML diagrams such as sequence diagrams, and activity diagrams are employed to describe the elements of the process view.

The Process View of Elevator Control System is presented by the following Sequence Diagram;

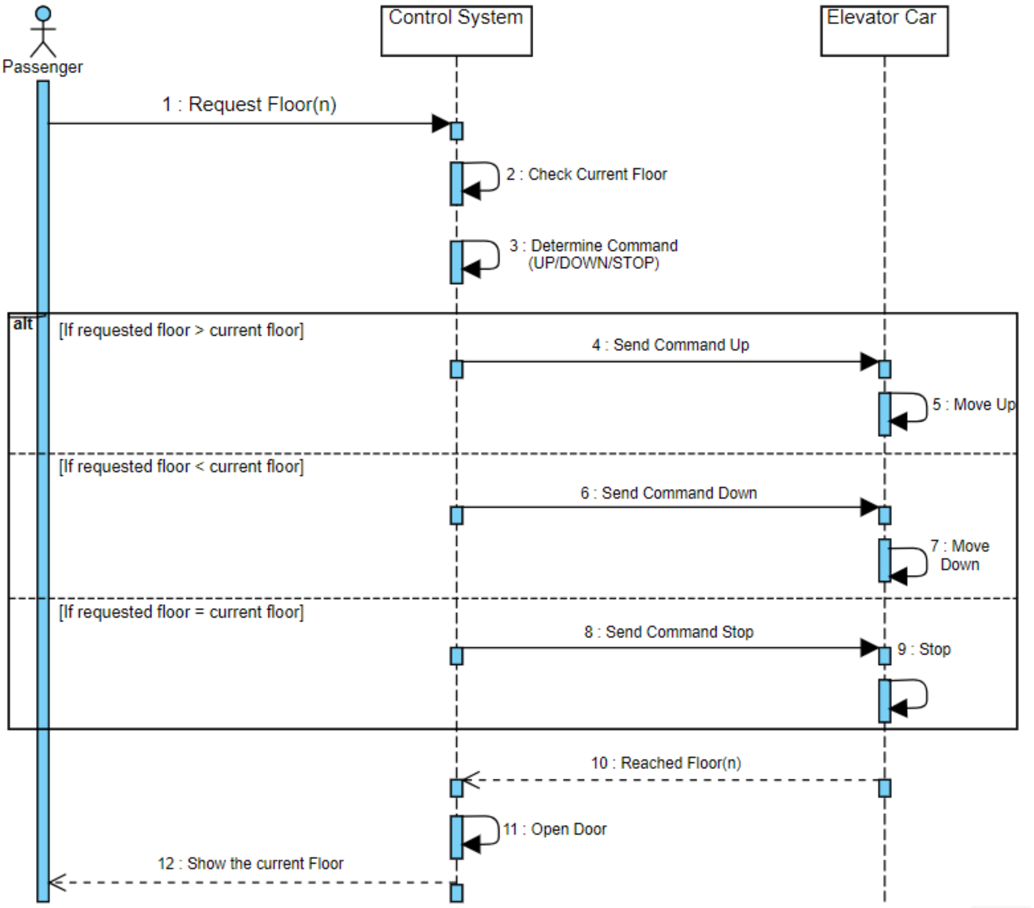


Figure 2. Sequence Diagram Elevator Control System

3. Development View

The development view presents a system from the perspective of a programmer and revolves around software management. This view is also referred to as the implementation view. The UML diagrams used to represent the development view, are Package diagram and Component Diagrams.
The Process View of Elevator Control System is illustrated using the following Component Diagram;

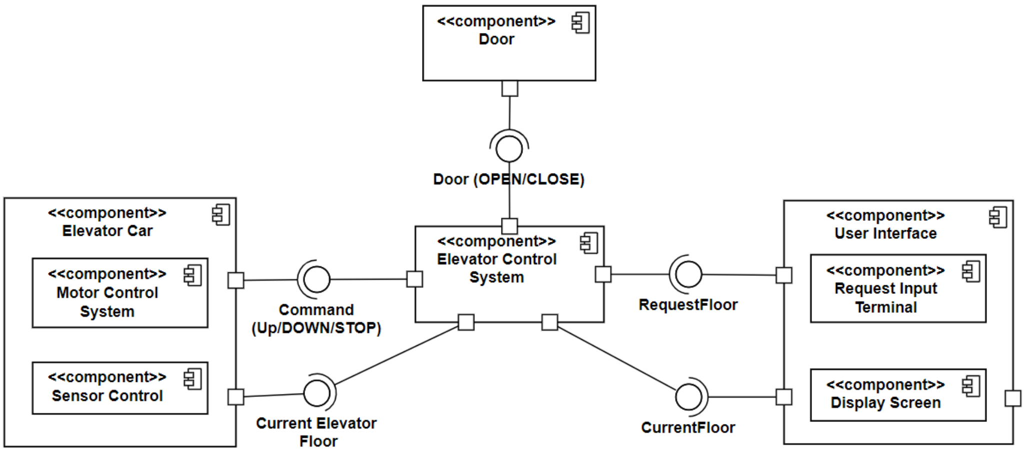

<<component>> Door
Door (OPEN/CLOSE)
<<component>> Elevator Car
<<component>> Motor Control System
<<component>> Sensor Control
Command (Up/DOWN/STOP)
<<component>> Elevator Control System
Current Elevator Floor
RequestFloor
CurrentFloor
<<component>> User Interface
<<component>> Request Input Terminal
<<component>> Display Screen

Figure 3. Component Diagram Elevator Control System

4. Physical View

The physical view illustrates the system through the eyes of a system engineer, focusing on the topology of software components and the physical connections among them. This view is also known as the deployment view. The UML diagram commonly employed to represent the physical perspective is the deployment diagram.
The Deployment Diagram for Elevator Control System is as follows;

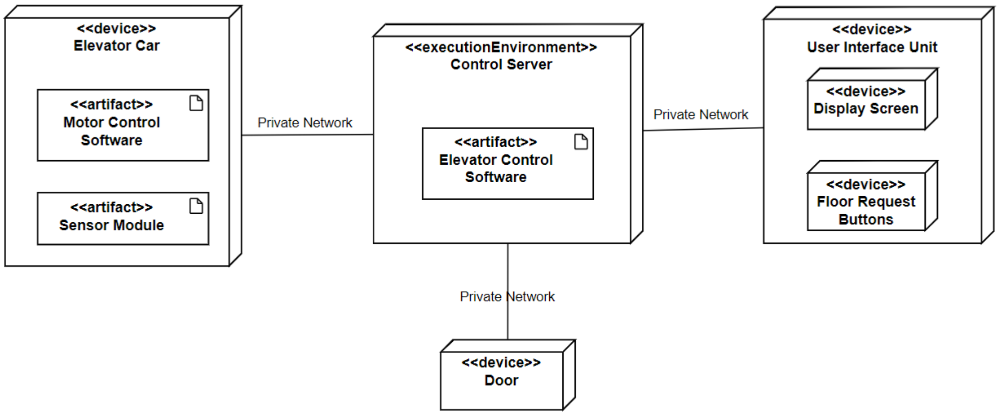

<<device>> Elevator Car
<<artifact>> Motor Control Software
<<artifact>> Sensor Module
Private Network
<<executionEnvironment>> Control Server
<<artifact>> Elevator Control Software
Private Network
<<device>> User Interface Unit
<<device>> Display Screen
<<device>> Floor Request Buttons
Private Network
<<device>> Door

Figure 4. Deployment Diagram Elevator Control System

5. +1 View

A limited set of use cases or scenarios, forming the fifth view, is employed to elaborate on the architectural description. These scenarios outline sequences of interactions between objects and processes. This view is alternatively referred to as the use case view.
Following is the Use Case Diagram of Elevator Control System;

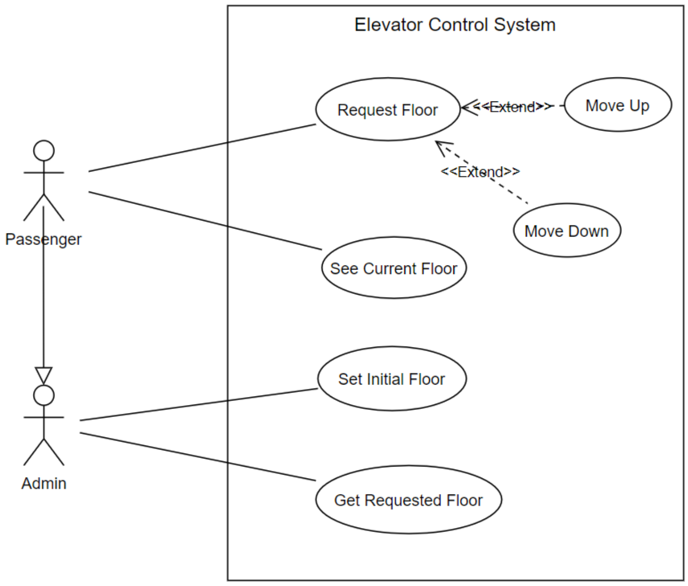


Figure 5. Use Case Diagram Elevator Control System

VDM Specification

values

$MIN : N = 0$

$MAX : N = 10$

types

$Command = < UP > | < DOWN > | < STOP >$

$Door = < OPEN > | < CLOSE >$

state ElevatorControlSystem **of**

$requested\ Floor : [N]$

$current\ Floor : [N]$

$isMoving$: B

- - both requested and current floor must be in range or equal to nil and moving state to be

- - either true or false

inv mk- ElevatorControlSystem $(req, cur, move) \underline{\Delta}$

$$((inRange(req) \lor req = nil)$$

$$\land (inRange(cur) \lor cur = nil)$$

$$\land (move = TRUE \lor move = FALSE))$$

- - both requested and current floor are undefined when system is initialized and moving state

- - to be false

init mk- ElevatorControlSystem $(req, cur, move)$ $\underline{\Delta}$

$$(req \ = \ nil \ \land \ cur \ = \ nil \land \ move = FALSE)$$

end

functions

inRange($val : \ N$) $result : B$

pre $TRUE$

post $result \ \leftrightarrow \ MIN \ \leq \ val \ \leq \ MAX$

operations

- - an operation that records the initial floor of the system

- - this operation will only be used once

setInitialFloor($floorNo : \ N$)

ext wr $currentFloor : [N]$

pre $inRange(floorNo) \land currentFloor \ = \ nil$

post $currentFloor \ = \ floorNo$

- - an operation that records the requested floor and signals hardware to move up or down as

- - appropriate

requestFloor $(floorNo : \ N)$ $movement : \ Command$

ext wr $requestedFloor : [N]$

rd $currentFloor : \ [N]$

pre $inRange(floorNo) \land currentFloor \ \neq \ NIL$

post $requestedFloor \ = \ floorNo \ \land$

$$(floorNo \ > \ currentFloor \ \land \ movement \ = < UP >)$$
$$\lor \ (floorNo \ < \ currentFloor \ \land \ movement \ = < DOWN >)$$
$$\lor \ (floorNo \ = \ currentFloor \ \land \ movement \ = < STOP >)$$

- - an operation that instructs the hardware to move upwards or stop if requested floor has

- - been reached

moveUp() $movement : Command$

ext **rd** $requestedFloor : [N]$

wr $currentFloor : [N]$

wr $isMoving : B$

pre $currentFloor < requestedFloor \wedge currentFloor \neq NIL \wedge requestedFloor \neq NIL$

post $currentFloor = \overline{currentFloor} + 1 \wedge isMoving = TRUE \wedge$

$(currentFloor < requestedFloor \wedge movement = < UP >)$

$\vee (currentFloor = requestedFloor \wedge movement = < STOP > \wedge isMoving = FALSE)$

- - an operation that instructs the hardware to move downwards or stop if requested floor has been reached

moveDown() $movement : Command$

ext **rd** $requestedFloor : [N]$

wr $currentFloor : [N]$

wr $isMoving : B$

pre $currentFloor > requestedFloor \wedge currentFloor \neq NIL \wedge requestedFloor \neq NIL$

post $currentFloor = \overline{currentFloor} - 1 \wedge isMoving = TRUE \wedge$

$(currentFloor > requestedFloor \wedge movement = < DOWN >)$

$\vee (currentFloor = requestedFloor \wedge movement = < STOP > \wedge isMoving = FALSE)$

- - an operation that instructs the hardware to open the elevator door

openDoor $(floorNo : N)$ $doorStatus: Door$

ext rd $currentFloor : [N] , isMoving : B$

pre $inRange(floorNo) \wedge currentFloor = floorNo \wedge isMoving = FALSE$

post $doorStatus = < OPEN >$

- - an operation that instructs the hardware to close the elevator door

closeDoor ($floorNo\ :\ N$) $doorStatus : Door$

ext rd $currentFloor\ :\ [N]\ ,\ isMoving\ :\ B$

pre $inRange(floorNo) \wedge currentFloor = floorNo\ \wedge\ isMoving = FALSE$

post $doorStatus = < CLOSE >$

- - an operation that returns the requested floor

getRequestedFloor() $currentRequested\ :\ [N]$

ext rd $requestedFloor\ :\ [N]$

pre $TRUE$

post $currentRequested\ =\ requestedFloor$

- - an operation that returns the current floor

getCurrentFloor() $currentCurrent\ :\ [N]$

ext rd $currentFloor\ :\ [N]$

pre $TRUE$

post $currentCurrent\ =\ currentFloor$

- - an operation that returns the moving state of elevator

getMovingState() $movingState\ : B$

ext rd $isMoving\ : B$

pre $TRUE$

post $movingState = isMoving$

Java Code

(Command.java)

```
class Command
{
    //a single private attribute
    private int value;

    //class constants representing named quote values
    public static final Command UP = new Command(0);
    public static final Command DOWN = new Command(1);
    public static final Command STOP = new Command(2);

    //private constructor used by class constants
    private Command (int x)
    {
        value = x;
    }

    public boolean equals (Object objectIn)
    {
        Command c=(Command) objectIn;
        return value==c.value;
    }

    //useful for testing purposes
    public String toString()
    {
        switch (value) {
            case 0:
                return "UP";
            case  1:
                return "DOWN";
            default:
                return "STOP";
        }
    }

}
```

(Door.java)

```
public class Door
{
    //a single private attribute
    private int value;

    //class constants representing named quote values
    public static final Door OPEN = new Door(0);
    public static final Door CLOSE = new Door(1);


    //private constructor used by class constants
    private Door (int x)
    {
        value = x;
    }

    public boolean equals (Object objectIn)
    {
        Door d=(Door) objectIn;
        return value==d.value;
    }

    //useful for testing purposes
    public String toString()
    {
        switch (value) {
            case 0:
                return "OPEN";
            case  1:
                return "CLOSE";
            default:
                return "";
        }
    }

}
```

(ElevatorControlSystem.java)

```
/**
 * ElevatorControlSystem
 */
public class ElevatorControlSystem implements InvariantCheck{

    //constants
    public static final int NIL = -999;
    public static final int MAX = 10;
    public static final int MIN = 0;

    //state attributes
    private int requestedFloor ;
    private int currentFloor ;
    private boolean isMoving ;
    private Command movement;
    private Door doorStatus;

    //initialisation satisfied by constructor
    public ElevatorControlSystem()
    {
        requestedFloor=NIL;
        currentFloor=NIL;
        isMoving=false;
        //checking invariant class
        VDM.invTest(this);
    }

    //invariant
    public boolean inv()
    {
        return (inRange(requestedFloor)||requestedFloor==NIL) &&
(inRange(currentFloor)||currentFloor==NIL) && (isMoving==false ||
isMoving==true);
    }

    //inRange function added as a private method
    private boolean inRange(int val)
    {
        return (MIN <= val && val <=MAX);
    }

    //a function to set the initial floor
    public void setInitialFloor(int floorNo)
    {
        VDM.preTest(inRange(floorNo) && currentFloor==NIL);
        currentFloor=floorNo;
    }

    //a function to request a floor
    public Command requestFloor(int floorNo)
    {
        //check precondition
        VDM.preTest(inRange(floorNo) && currentFloor!=NIL);

        //implement post condition
        //satistfy 1st conjunct
        requestedFloor=floorNo;

        //satisfy 2nd conjunct
        if(floorNo>currentFloor)
        {
            movement = Command.UP;
        }
        if(floorNo<currentFloor)
        {
            movement = Command.DOWN;
        }
        if(floorNo==currentFloor)
        {
            movement = Command.STOP;
        }
        //check invariant before method ends
        VDM.invTest(this);
        //send back output value
        return movement;
    }

    //operation to move up
    public Command moveUP()
    {
        //pre-condition checked
        VDM.preTest(currentFloor<requestedFloor && currentFloor!=NIL &&
requestedFloor!=NIL );

        //post condition
        //satisfy 1st conjunct
        currentFloor=currentFloor+1;
        //satisfy 2nd conjunct
        isMoving=true;
        //satisfy 3rd conjunct
        if(currentFloor<requestedFloor)
        {
            movement = Command.UP;
        }
        if(currentFloor==requestedFloor)
        {
            movement = Command.STOP;
            System.out.println("You have reached the requested floor");
            isMoving=false;
        }
        //check invariant before method ends
        VDM.invTest(this);
        //send back output value
        return movement;
    }

    //operation to move down
    public Command moveDOWN()
    {
        //pre-condition checked
        VDM.preTest(currentFloor>requestedFloor && currentFloor!=NIL && 
requestedFloor!=NIL);

        //post condition
        //satisfy 1st conjunct
        currentFloor=currentFloor-1;
        //satisfy 2nd conjunct
        isMoving=true;
        //satisfy 3rd conjunct
        if(currentFloor>requestedFloor)
        {
            movement = Command.DOWN;
        }
        if(currentFloor==requestedFloor)
        {
            movement = Command.STOP;
            System.out.println("You have reached the requested floor");
            isMoving=false;
        }
        //check invariant before method ends
        VDM.invTest(this);
        //send back output value
        return movement;
    }


    //operation to open door
    public Door openDoor(int floorNo)
    {
        //pre-condition checked
        VDM.preTest((inRange(floorNo))&& (currentFloor==floorNo) && 
(isMoving==false));
        //post condition
        doorStatus = Door.OPEN;
        //check invariant before method ends
        VDM.invTest(this);
        //send back output value
        return doorStatus;
    }

    //operation to close door
    public Door closeDoor(int floorNo)
    {
        //pre-condition checked
        VDM.preTest((inRange(floorNo))&& (currentFloor==floorNo) && 
(isMoving==false));
        //post condition
        doorStatus = Door.CLOSE;
        //check invariant before method ends
        VDM.invTest(this);
        //send back output value
        return doorStatus;
    }

    public int getRequestedFloor()
    {
        return requestedFloor;
    }

    public int getCurrentFloor()
    {
        return currentFloor;
    }

    public boolean getMovingState()
    {
        return isMoving;
    }

}
```

(InvariantCheck.java)

```
public interface InvariantCheck {
    public boolean inv();
}
```

(VDM.java)

```
public class VDM {

    public static void preTest(boolean condition) {
        if (!condition) {
            throw new RuntimeException("Precondition violated");
        }
    }

    public static void postTest(boolean condition) {
        if (!condition) {
            throw new RuntimeException("Postcondition violated");
        }
    }

    public static void invTest(InvariantCheck object) {
        if (!object.inv()) {
            throw new RuntimeException("Invariant violated");
        }
    }
}
```

Testing Class

(EasyIn.java)

```
import java.util.InputMismatchException;
import java.util.Scanner;

public class EasyIn {
    private static Scanner scanner = new Scanner(System.in);

    public static char getChar() {
        String input = getString();
        if (input.length() > 0) {
            return input.charAt(0);
        } else {
            return '\0'; // Return null character for an empty input
        }
    }

    public static int getInt() {
        while (true) {
            try {
                int value = scanner.nextInt();
                scanner.nextLine(); // Consume the newline character
                return value;
            } catch (InputMismatchException e) {
                // Consume the invalid input
                scanner.next();
                System.out.print("Invalid input. Please enter an integer: ");
            }
        }
    }

    public static String getString() {
        return scanner.nextLine();
    }

    public static void pause(String message) {
        System.out.print(message);
        getString(); // Wait for user to press Enter
    }
}
```

(ElevatorControlSystemTester.java)

```
public class ElevatorControlSystemTester
{
    public static void main(String[] args)
    {
        char choice;

        //to monitor for an invariant violation of initial object
        try
        {
            //generate a new elevator control system object
            ElevatorControlSystem elevator =new ElevatorControlSystem();
            do
            {
                System.out.println("\n Elevator Control System Tester");
                System.out.println("1.Initialize system");
                System.out.println("2.Request a floor");
                System.out.println("3.Display current floor");
                System.out.println("4.Display requested floor");
                System.out.println("5.Display moving state");
                System.out.println("6.Go up");
                System.out.println("7.Go down");
                System.out.println("8.Open Door");
                System.out.println("9.Close Door");
                System.out.println("Enter choice 1 to 9 or press 0 to quit");

                //accepts character enetered at keyboard
                choice =EasyIn.getChar();

                System.out.println(); //blank line
                try
                {
                    switch (choice) {
                        case '1':
                            option1(elevator);
                            break;
                        case '2':
                            option2(elevator);
                            break;
                        case '3':
                            option3(elevator);
                            break;
                        case '4':
                            option4(elevator);
                            break;
                        case '5':
                            option5(elevator);
                            break;
                        case '6':
                            option6(elevator);
                            break;
                        case '7':
                            option7(elevator);
                            break;
                        case '8':
                            option8(elevator);
                            break;
                        case '9':
                            option9(elevator);
                            break;
                        default:
                            break;
                    }

                }catch(VDMException e)
                {
                    e.printStackTrace(); //built in exception method
                }
            }while(choice !='0');

        }
        catch(VDMException e)//if initial object breaks invariant
        {
            System.out.println("Initial object breaks invariant"); //error
message
            EasyIn.pause("\nPress <Enter> to quit");//pause method of EasyIn
        }
    }
    //test VDM operation implementations

    public static void option1(ElevatorControlSystem elevator)
    {
        System.out.println("Enter floor number");
        int floor =EasyIn.getInt();
        elevator.setInitialFloor(floor);
        System.out.println("System initialized to floor  "+
elevator.getCurrentFloor());
    }

    public static void option2(ElevatorControlSystem elevator)
    {
        System.out.println("Enter floor number");
        int floor =EasyIn.getInt();
        elevator.requestFloor(floor);
    }
    public static void option3(ElevatorControlSystem elevator)
    {
        if(elevator.getCurrentFloor()==-999){System.out.println("Current floor
is unknown.\nPlease initialize the system");}
        else{System.out.println("Current floor is :
"+elevator.getCurrentFloor());}
    }

    public static void option4(ElevatorControlSystem elevator)
    {
        System.out.println("Requested floor is :
"+elevator.getRequestedFloor());
    }
    public static void option5(ElevatorControlSystem elevator)
    {
        if(elevator.getMovingState()==true){System.out.println("Elevator is
moving");}
        if(elevator.getMovingState()==false){System.out.println("Elevator is
not moving");}

    }
    public static void option6(ElevatorControlSystem elevator)
    {
        elevator.moveUP();//this method could throw a VDMException
    }
    public static void option7(ElevatorControlSystem elevator)
    {
        elevator.moveDOWN();//this method could throw a VDMException
    }
    public static void option8(ElevatorControlSystem elevator)
    {
        elevator.openDoor(elevator.getCurrentFloor());//this method could
throw a VDMException
        System.out.println("Door opened");
    }
    public static void option9(ElevatorControlSystem elevator)
    {
        elevator.closeDoor(elevator.getCurrentFloor());//this method could
throw a VDMException
        System.out.println("Door closed");
    }


}
```

(VDMException.java)

```
public class VDMException extends RuntimeException {
    public VDMException(String message) {
        super(message);
    }
}
```

Test Cases

Project Name	Elevator Control System
Priority	High
Description	The Elevator Control System will effectively manage the operations of an elevator car within a building
Test Objective	To test system functionalities

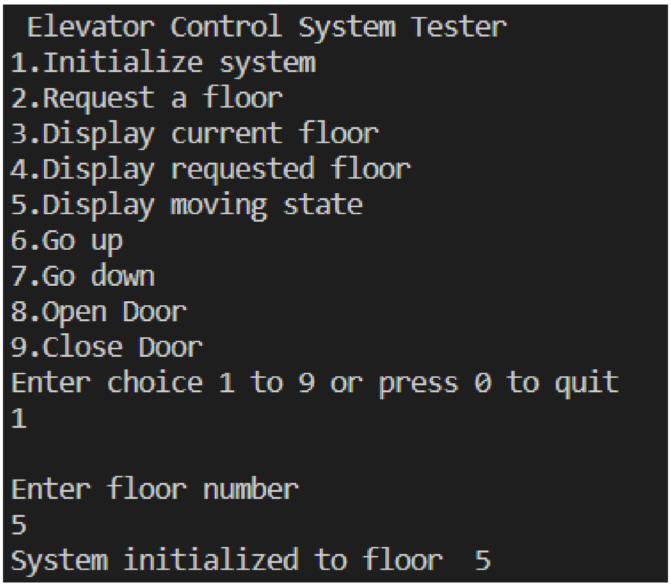
Test Case Author	Maria Ashfaq, Fatima Zehra
Test Case Reviewer	Sir Mustafa Latif
Test Case Version	1
Test Execution Date	10/1/2024

Test Case ID	Description	Test Steps	Input Data	Expected Results	Actual Results	Execution Status	Bug Severity
ECS_TC1	Initialize the system with a valid floor.	1. Press option 1 to initialize the system. 2. Enter floor number 5.	5	System initialized to floor 5.	System initialized to floor 5.	Pass	High
ECS_TC2	Initialize the system with an invalid floor.	1. Press option 1 to initialize the system. 2. Enter floor number 12.	12	System throws an Error	System throws an Error	Pass	High
ECS_TC3	Requesting a floor after initialization	1.Initialize the system 2.Press 2 to request a floor 3.Enter floor number 3 4.Press 4 to display requested floor	3	Requested floor is 3	Requested floor is 3	Pass	High
ECS_TC4	Requesting a floor before initialization	1. Press 2 to request a floor 2.Enter floor number 3	3	System throws an Error	System throws an Error	Pass	High
ECS_TC5	Test moving the elevator up.	1.Initialize the system to floor 0 2.Request a floor above the current floor 3.Press 6 to go up 4.Press 3 to Display current floor	5	System goes up by one floor from floor 0 so current floor is 1	System goes up by one floor from floor 0 so current floor is 1	Pass	High
ECS_TC6	Test moving the elevator down.	1.Initialize the system to floor 5 2.Request a floor below the current floor 3.Press 7 to go down 4.Press 3 to Display current floor	2	System goes down by one floor from floor 5 so current floor is 4	System goes down by one floor from floor 5 so current floor is 4	Pass	High

ECS_TC7	Test opening the elevator door	1.Initialize the system to floor 0 2.Request a floor. 3.Press 8 to select Open Door option	None	Door is opened	Door is opened	Pass	High
ECS_TC8	Test closing the elevator door	1.Open the door 2.Press 9 to close the door	None	Door is closed	Door is closed	Pass	High
ECS_TC9	Test opening the elevator door while elevator is moving	1.Initialize the system 2.Request a floor. 3.Go up /down 4.Press 8 to select Open Door option	None	System throws an Error	System throws an Error	Pass	High
ECS_TC10	Test checking system state while elevator is moving	1.Initialize the system 2.Request a floor. 3.Go up /down 4.Press 5 to display system state	None	System is moving	System is moving	Pass	High
ECS_TC11	Test checking system state while elevator is not moving	1.Initialize the system 2.Request a floor. 3.Reach requested floor 4. Press 5 to display system state	None	System is not moving	System is not moving	Pass	High
ECS_TC12	Go up without requesting a floor	1.Initialize system 2.Press 6 to go up	None	System throws an Error	System throws an Error	Pass	High
ECS_TC13	Go down without requesting a floor	1.Initialize system 2.Press 7 to go down	None	System throws an Error	System throws an Error	Pass	High

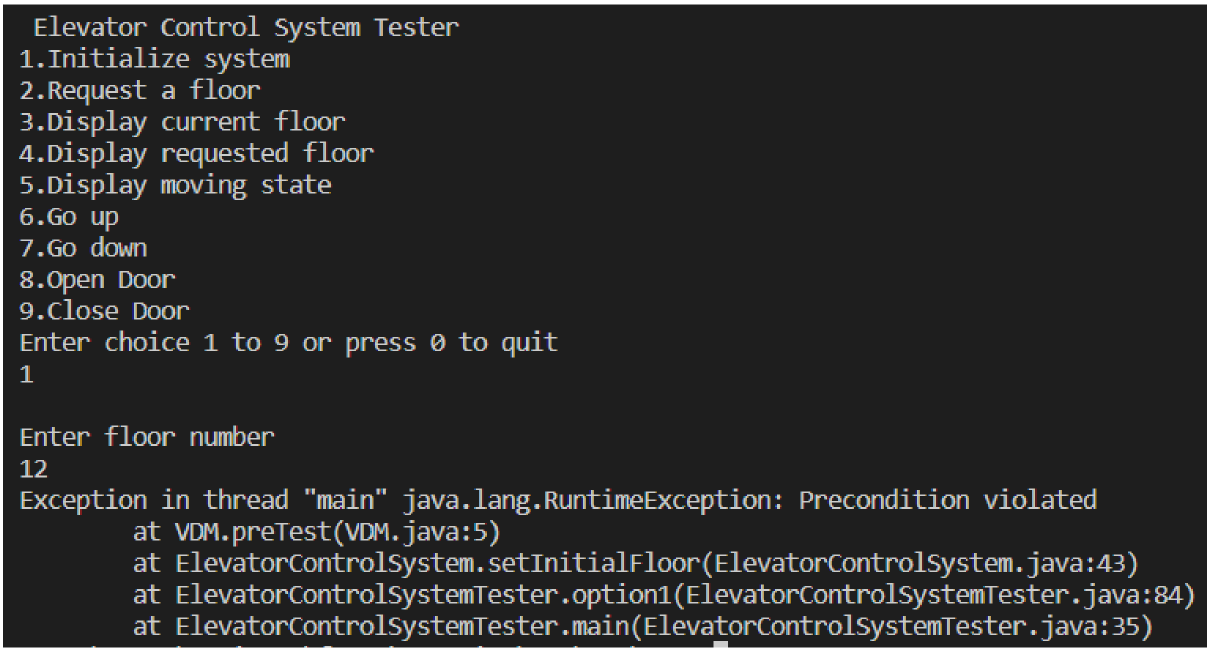
Output of Test Cases

- **ECS_TC1**



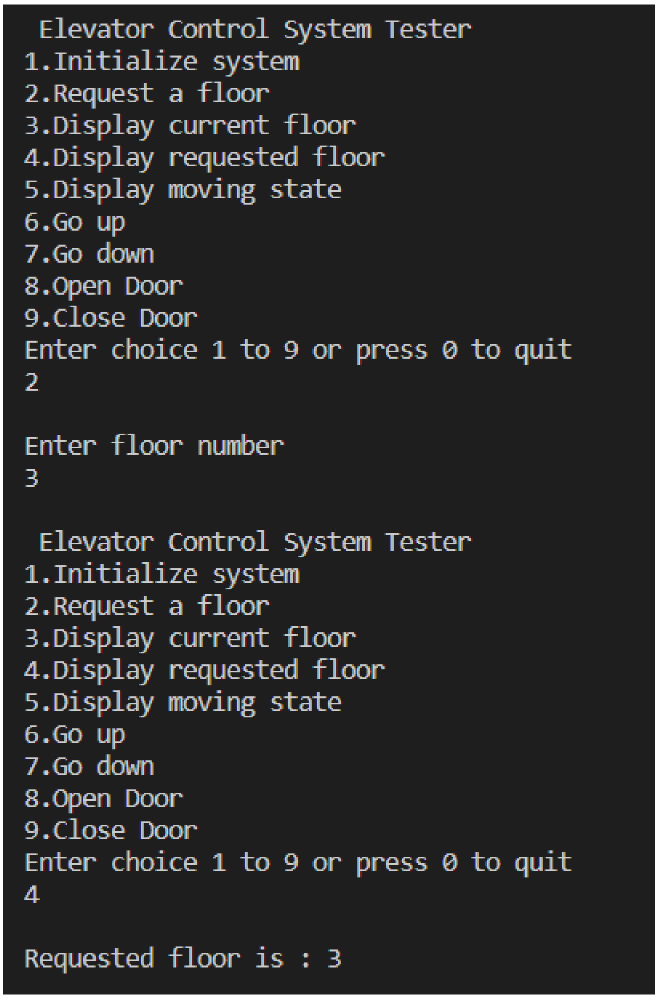
```
 Elevator Control System Tester
1.Initialize system
2.Request a floor
3.Display current floor
4.Display requested floor
5.Display moving state
6.Go up
7.Go down
8.Open Door
9.Close Door
Enter choice 1 to 9 or press 0 to quit
1

Enter floor number
5
System initialized to floor  5
```

- **ECS_TC2**



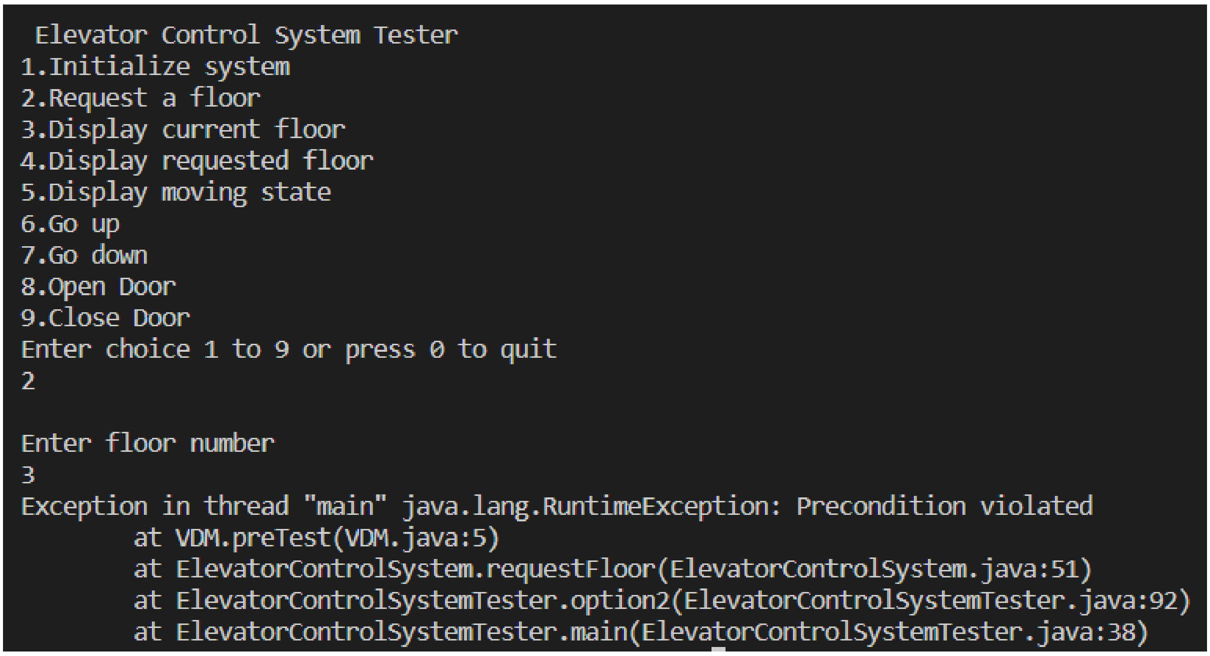
```
 Elevator Control System Tester
1.Initialize system
2.Request a floor
3.Display current floor
4.Display requested floor
5.Display moving state
6.Go up
7.Go down
8.Open Door
9.Close Door
Enter choice 1 to 9 or press 0 to quit
1

Enter floor number
12
Exception in thread "main" java.lang.RuntimeException: Precondition violated
        at VDM.preTest(VDM.java:5)
        at ElevatorControlSystem.setInitialFloor(ElevatorControlSystem.java:43)
        at ElevatorControlSystemTester.option1(ElevatorControlSystemTester.java:84)
        at ElevatorControlSystemTester.main(ElevatorControlSystemTester.java:35)
```

- **ECS_TC3**



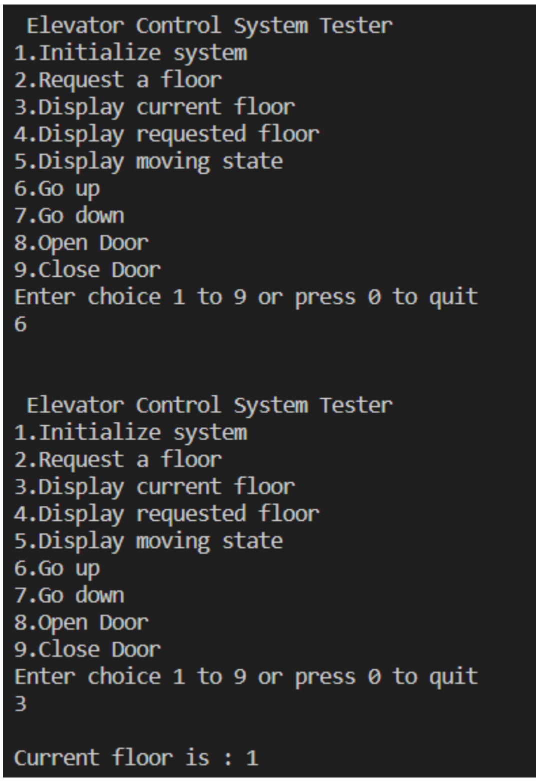
```
 Elevator Control System Tester
1.Initialize system
2.Request a floor
3.Display current floor
4.Display requested floor
5.Display moving state
6.Go up
7.Go down
8.Open Door
9.Close Door
Enter choice 1 to 9 or press 0 to quit
2

Enter floor number
3

 Elevator Control System Tester
1.Initialize system
2.Request a floor
3.Display current floor
4.Display requested floor
5.Display moving state
6.Go up
7.Go down
8.Open Door
9.Close Door
Enter choice 1 to 9 or press 0 to quit
4

Requested floor is : 3
```

- **ECS_TC4**



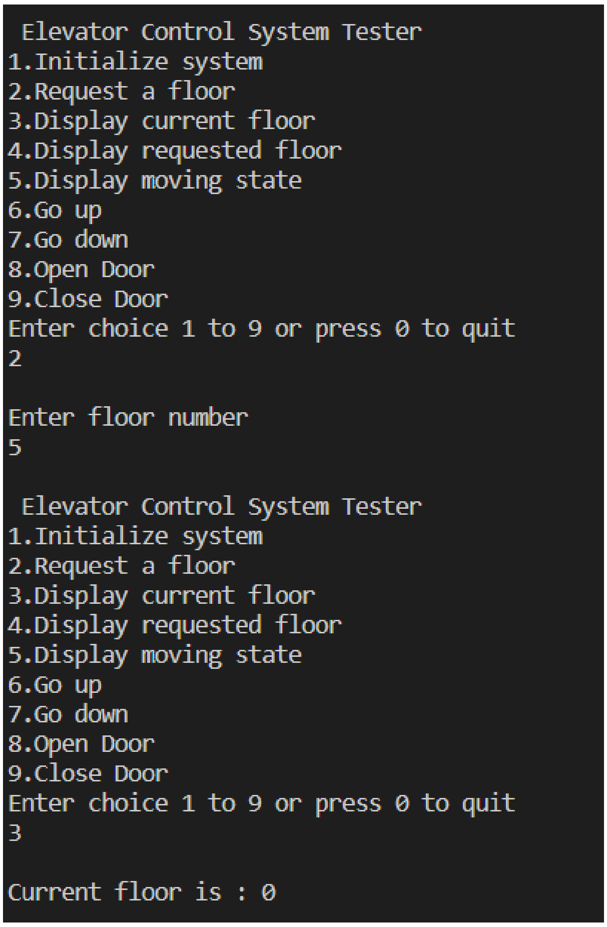
```
 Elevator Control System Tester
1.Initialize system
2.Request a floor
3.Display current floor
4.Display requested floor
5.Display moving state
6.Go up
7.Go down
8.Open Door
9.Close Door
Enter choice 1 to 9 or press 0 to quit
2

Enter floor number
3
Exception in thread "main" java.lang.RuntimeException: Precondition violated
        at VDM.preTest(VDM.java:5)
        at ElevatorControlSystem.requestFloor(ElevatorControlSystem.java:51)
        at ElevatorControlSystemTester.option2(ElevatorControlSystemTester.java:92)
        at ElevatorControlSystemTester.main(ElevatorControlSystemTester.java:38)
```

- **ECS_TC5**

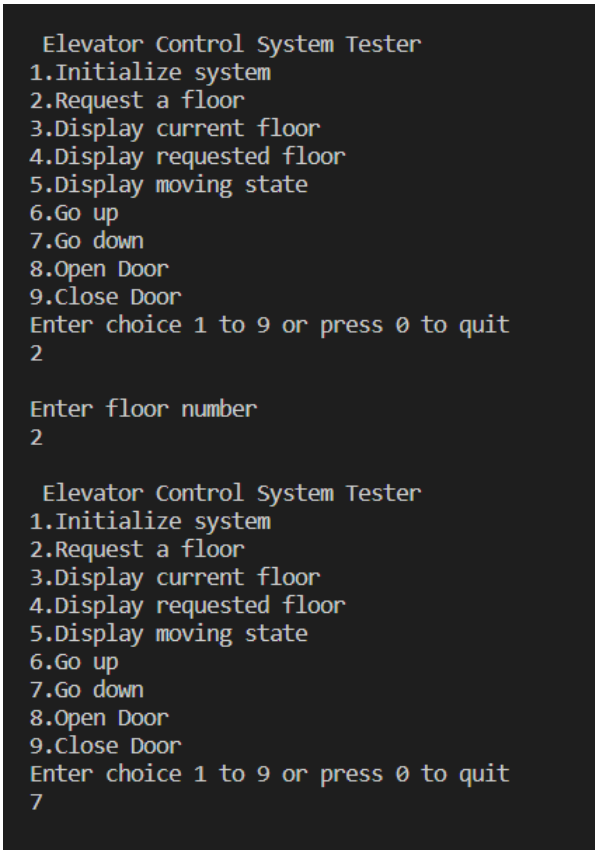


```
 Elevator Control System Tester
1.Initialize system
2.Request a floor
3.Display current floor
4.Display requested floor
5.Display moving state
6.Go up
7.Go down
8.Open Door
9.Close Door
Enter choice 1 to 9 or press 0 to quit
6


 Elevator Control System Tester
1.Initialize system
2.Request a floor
3.Display current floor
4.Display requested floor
5.Display moving state
6.Go up
7.Go down
8.Open Door
9.Close Door
Enter choice 1 to 9 or press 0 to quit
3

Current floor is : 1
```

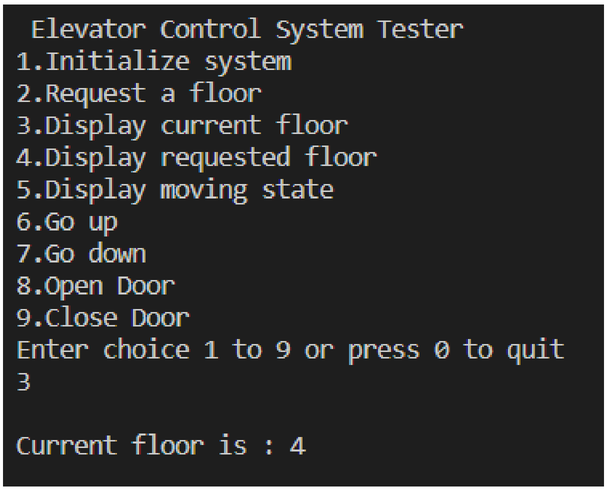
- **ECS_TC6**



```
 Elevator Control System Tester
1.Initialize system
2.Request a floor
3.Display current floor
4.Display requested floor
5.Display moving state
6.Go up
7.Go down
8.Open Door
9.Close Door
Enter choice 1 to 9 or press 0 to quit
2

Enter floor number
5

 Elevator Control System Tester
1.Initialize system
2.Request a floor
3.Display current floor
4.Display requested floor
5.Display moving state
6.Go up
7.Go down
8.Open Door
9.Close Door
Enter choice 1 to 9 or press 0 to quit
3

Current floor is : 0
```

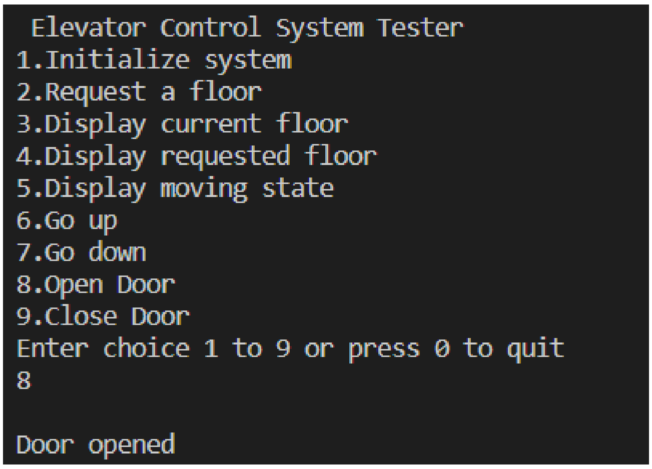


```
 Elevator Control System Tester
1.Initialize system
2.Request a floor
3.Display current floor
4.Display requested floor
5.Display moving state
6.Go up
7.Go down
8.Open Door
9.Close Door
Enter choice 1 to 9 or press 0 to quit
2

Enter floor number
2

 Elevator Control System Tester
1.Initialize system
2.Request a floor
3.Display current floor
4.Display requested floor
5.Display moving state
6.Go up
7.Go down
8.Open Door
9.Close Door
Enter choice 1 to 9 or press 0 to quit
7
```



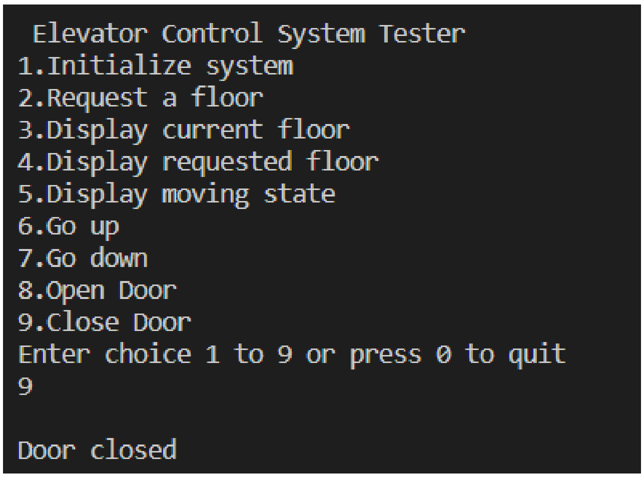
```
 Elevator Control System Tester
1.Initialize system
2.Request a floor
3.Display current floor
4.Display requested floor
5.Display moving state
6.Go up
7.Go down
8.Open Door
9.Close Door
Enter choice 1 to 9 or press 0 to quit
3

Current floor is : 4
```

- **ECS_TC7**



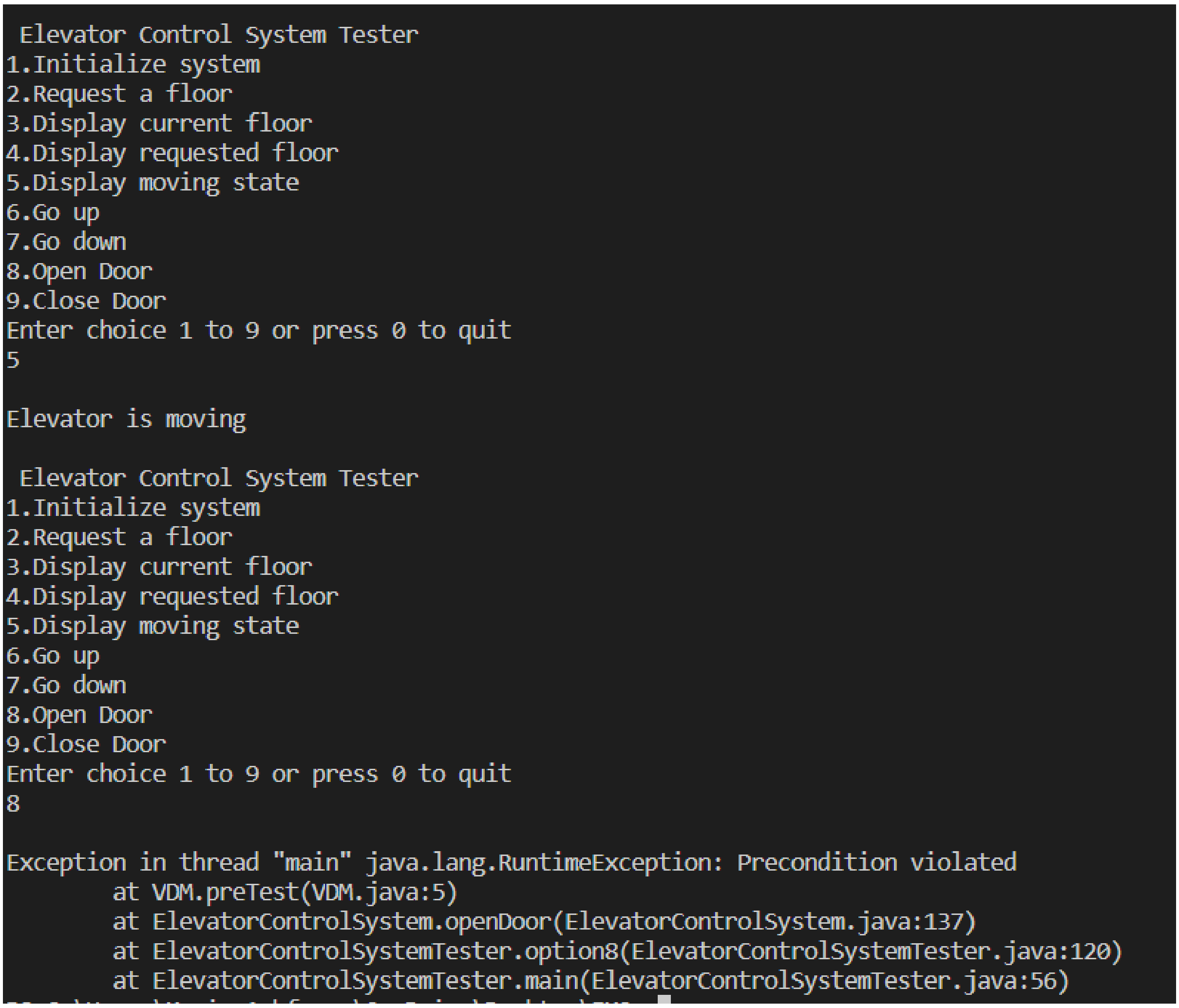
```
 Elevator Control System Tester
1.Initialize system
2.Request a floor
3.Display current floor
4.Display requested floor
5.Display moving state
6.Go up
7.Go down
8.Open Door
9.Close Door
Enter choice 1 to 9 or press 0 to quit
8

Door opened
```

- **ECS_TC8**



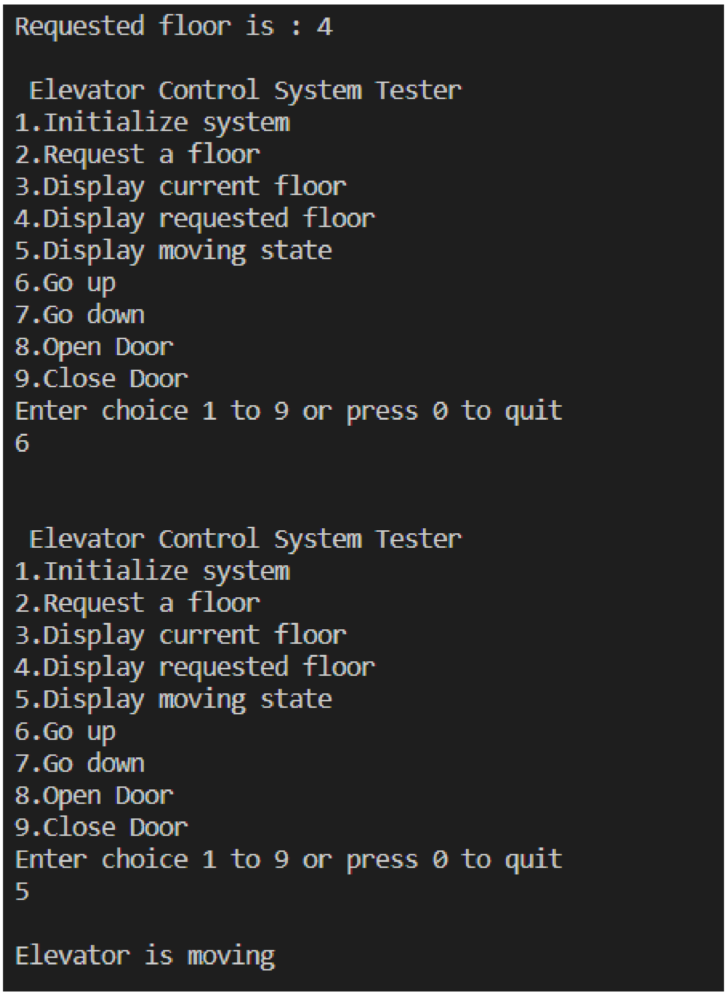
```
 Elevator Control System Tester
1.Initialize system
2.Request a floor
3.Display current floor
4.Display requested floor
5.Display moving state
6.Go up
7.Go down
8.Open Door
9.Close Door
Enter choice 1 to 9 or press 0 to quit
9

Door closed
```

- **ECS_TC9**



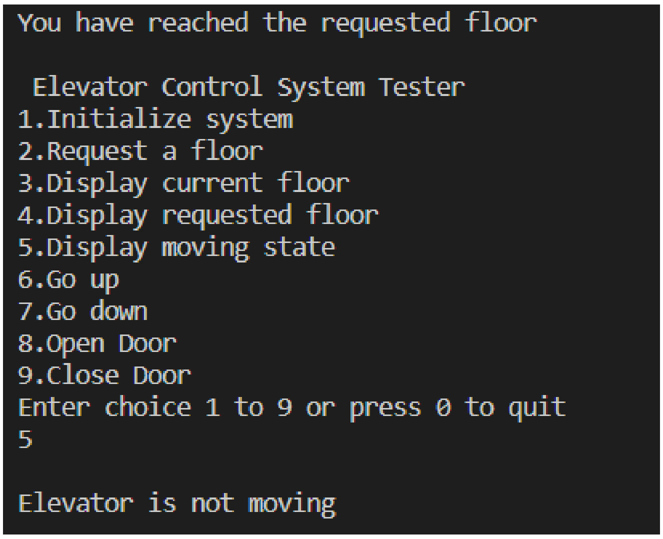
```
 Elevator Control System Tester
1.Initialize system
2.Request a floor
3.Display current floor
4.Display requested floor
5.Display moving state
6.Go up
7.Go down
8.Open Door
9.Close Door
Enter choice 1 to 9 or press 0 to quit
5

Elevator is moving

 Elevator Control System Tester
1.Initialize system
2.Request a floor
3.Display current floor
4.Display requested floor
5.Display moving state
6.Go up
7.Go down
8.Open Door
9.Close Door
Enter choice 1 to 9 or press 0 to quit
8

Exception in thread "main" java.lang.RuntimeException: Precondition violated
        at VDM.preTest(VDM.java:5)
        at ElevatorControlSystem.openDoor(ElevatorControlSystem.java:137)
        at ElevatorControlSystemTester.option8(ElevatorControlSystemTester.java:120)
        at ElevatorControlSystemTester.main(ElevatorControlSystemTester.java:56)
```

- **ECS_TC10**



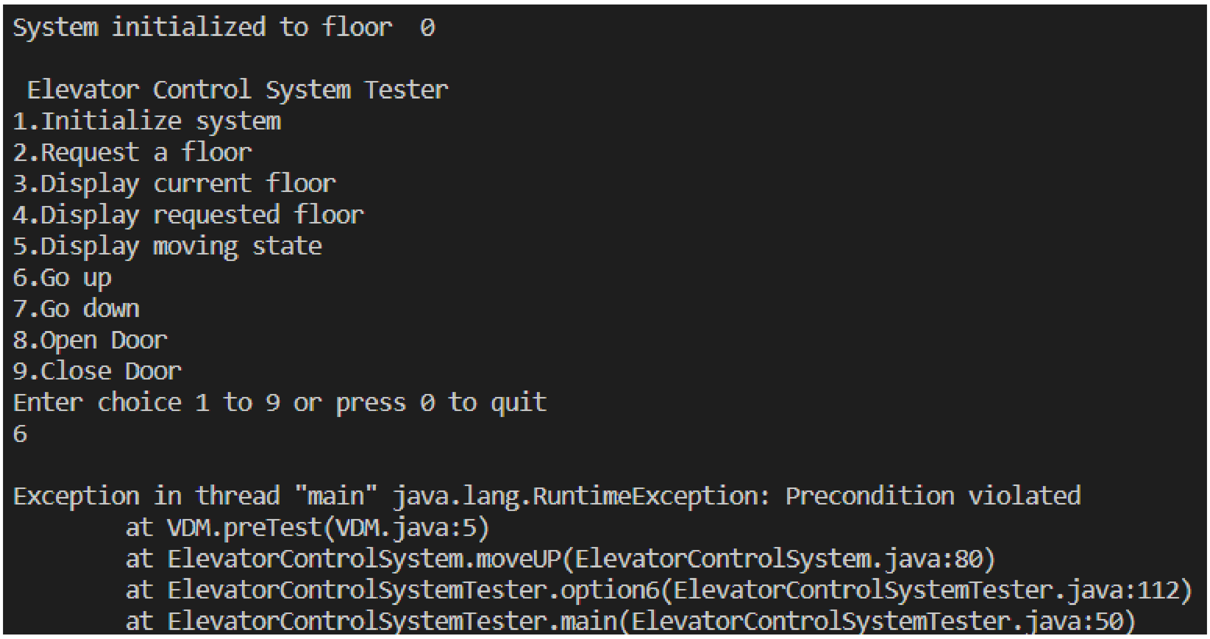
```
Requested floor is : 4

 Elevator Control System Tester
1.Initialize system
2.Request a floor
3.Display current floor
4.Display requested floor
5.Display moving state
6.Go up
7.Go down
8.Open Door
9.Close Door
Enter choice 1 to 9 or press 0 to quit
6


 Elevator Control System Tester
1.Initialize system
2.Request a floor
3.Display current floor
4.Display requested floor
5.Display moving state
6.Go up
7.Go down
8.Open Door
9.Close Door
Enter choice 1 to 9 or press 0 to quit
5

Elevator is moving
```

- **ECS_TC11**



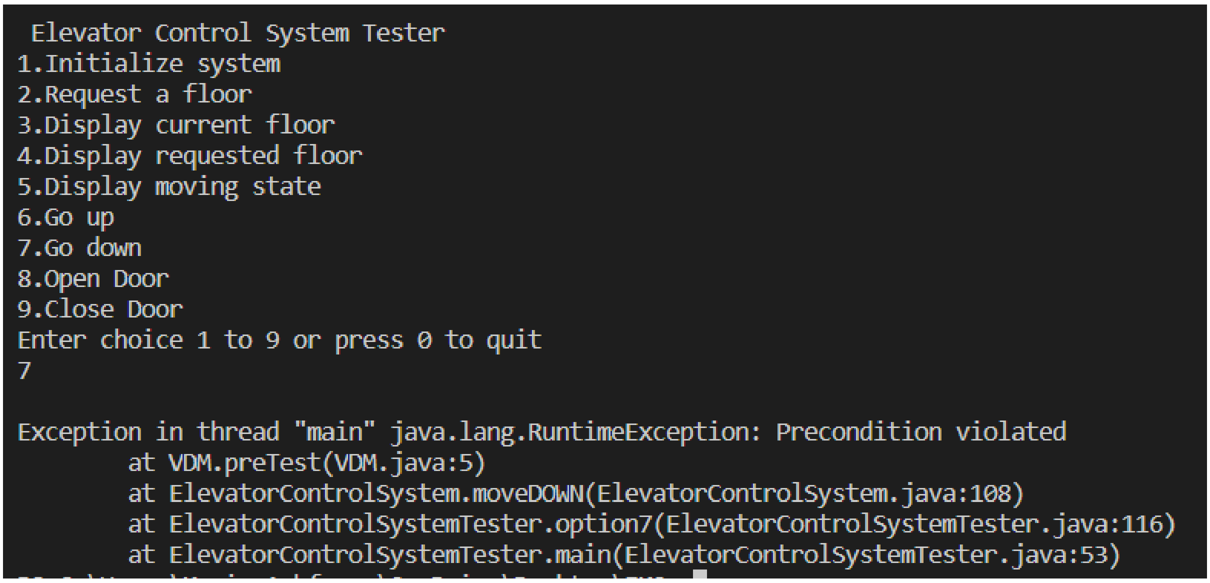
```
You have reached the requested floor

 Elevator Control System Tester
1.Initialize system
2.Request a floor
3.Display current floor
4.Display requested floor
5.Display moving state
6.Go up
7.Go down
8.Open Door
9.Close Door
Enter choice 1 to 9 or press 0 to quit
5

Elevator is not moving
```

- **ECS_TC12**



```
System initialized to floor  0

 Elevator Control System Tester
1.Initialize system
2.Request a floor
3.Display current floor
4.Display requested floor
5.Display moving state
6.Go up
7.Go down
8.Open Door
9.Close Door
Enter choice 1 to 9 or press 0 to quit
6

Exception in thread "main" java.lang.RuntimeException: Precondition violated
        at VDM.preTest(VDM.java:5)
        at ElevatorControlSystem.moveUP(ElevatorControlSystem.java:80)
        at ElevatorControlSystemTester.option6(ElevatorControlSystemTester.java:112)
        at ElevatorControlSystemTester.main(ElevatorControlSystemTester.java:50)
```

- **ECS_TC13**



```
 Elevator Control System Tester
1.Initialize system
2.Request a floor
3.Display current floor
4.Display requested floor
5.Display moving state
6.Go up
7.Go down
8.Open Door
9.Close Door
Enter choice 1 to 9 or press 0 to quit
7

Exception in thread "main" java.lang.RuntimeException: Precondition violated
        at VDM.preTest(VDM.java:5)
        at ElevatorControlSystem.moveDOWN(ElevatorControlSystem.java:108)
        at ElevatorControlSystemTester.option7(ElevatorControlSystemTester.java:116)
        at ElevatorControlSystemTester.main(ElevatorControlSystemTester.java:53)
```

GitHub Link

https://github.com/mariaashfaq02/ElevatorControlSystem