Formal Methods In Software Engineering (SE-306-A)

Stage – 1 (Code Analysis)

Project Title:

University Smart Bus & Route Management System

Submitted by:

Maida Kosser (221400091)

Muhammad Asad Ullah (221400089)

Muhammad Noman (221400087)

Zoya Khalid (221400095)

Submitted to:

Mam Sajiya Tariq

BS Software Engineering (VI)

GIFT University, Gujranwala

Member 1: Maida Kosser – Bus Class

Code of Bus Class:

```
class Bus implements Vehicle{

    //Fields
    private ArrayList<Stops> route = new ArrayList<Stops>();
    private Driver driver;
    private ArrayList<Student> students = new ArrayList<Student>();
    private String licensePlateNumber;
    private int speed;

    //Constructor
    public Bus() {
        this.licensePlateNumber = "Null";
        this.speed = 0;
     }

    public Bus(Driver driver, ArrayList<Student> students,  ArrayList<Stops> route, String licensePlateNumber) {
        this.driver = driver;
        this.route = route;
        this.students = students;
        this.licensePlateNumber = licensePlateNumber;
     }

    public String getLicensePlateNumber() {
        return licensePlateNumber;
    }

    public void addStops(Stops stop){
        route.add(stop);
    }//M

    public ArrayList<Stops> getroute(){
        return route;
    }//M

    public Stops getStop(String name){
        for(int i = 0; i <= route.size()-1; ++i){
            if(route.get(i).getName().equals(name)){
                return route.get(i);
            }//if
        }//for
        return null;
    }//M

    public void changeStop(String name, Stops stop){
        for(int i = 0; i <= route.size()-1; ++i){
            if(route.get(i).getName() == name){
                route.set(i, stop);
            }//if
        }//for
    }//M

    public void removeStop(String name){
        for(int i = 0; i <= route.size()-1; ++i){
            if(route.get(i).getName() == name){
                route.remove(i);
            }//if
        }//for
    }//M
```

```
    public void changeDriver(Driver driver){
        this.driver = driver;
    }

    public Driver getDriver(){
        return driver;
    }//M

    public void addStudent(Student student){
        students.add(student);
    }//M

    public void removeStudent(int idNumber){
        for(int i = 0; i <= students.size()-1; ++i){
            if(students.get(i).getIdNumber() == idNumber){
                students.remove(i);
            }//if
        }//for
    }//M

    public Student getStudent(int idNumber){
        for(int i = 0; i <= students.size()-1; ++i){
            if(students.get(i).getIdNumber() == idNumber){
                return students.get(i);
            }//if
        }//for
        return null;
    }//Student

    public void start(){
        System.out.println("The bus is started");
    }

    public void stop() {
        System.out.println("The bus is stopped ");
    }

    public void accelerate(int speed) {
        this.speed += speed;
    }

    public void brake() {
        this.speed -= 5;
    }

    public void deleteRoute(ArrayList<Stops> route) {
        if (route.remove(this)) {
            System.out.println("Stop deleted successfully.");
        } else {
            System.out.println("Stop not found in the list.");
        }
    }//M
}//class
```

Statement Type	Explanation
Pre-Conditions	• driver, students, route, licensePlateNumber all parameters must be non-null. • **getLicensePlateNumber():** licensePlateNumber must be initialized. • **addStops(stop)**: should not be null. • **getroute():** route list must be initialized. • **getStop(name):** name not be null and route initialized. • **changeStop():** Both name and stop should not null. • **removeStop(name):** name not be null. • **changeDriver(driver):** driver not null. • **getDriver():** driver must be initialized. • **addStudent(student):** student should be added not null. • **removeStudent(idNumber):** students list initialized and with valid idNumber. • **getStudent(idNumber):** students list initialized with valid idNumber. • **accelerate(speed):** speed cannot pass a negative number. • **brake():** current speed greater then 5 to avoid negative speed. • **deleteRoute(route):** route not null and should contain the item to be removed.
Post-Conditions	• licensePlateNumber = "Null" and speed = 0 • **getLicensePlateNumber():** Returns current value of licensePlateNumber. • **addStops(stop):** Adds a Stop object to route. • **getroute():** Returns current route list. • **getStop(name):** Returns Stop object with matching name or null if not found. • **changeStop(name, stop):** Replaces stop with new one if found. • **removeStop(name**): Removes stop if found. • **changeDriver(driver):** Updates the driver. • **getDriver():** Returns the current driver object. • **addStudent(student):** Add student to the students list. • **removeStudent(idNumber):** Removes student with matching ID if exists. • **getStudent(idNumber):** Returns Student with given ID or null. • **start() & Stop():** Prints the bus state. • **brake():** Reduces current speed by.

	• **deleteRoute(route):** Prints state of route depending on stop removal.
Initialization Statements	• speed = 0 • licensePlate = "Null", Default value. • students = new ArrayList<>() • route = new ArrayList<>(): is initialized in the field declaration. • driver initialized through constructor.
Invariants	• route and students are never null. They are initialized in constructor. • driver may be null, but replaced with a valid object for route. • No duplicate students allowed. • speed greater then 0 at all times cannot be negative. • **brake()** never reduces speed below 0. • If a student or stop is added and removed the list update accordingly.
Bound Function	• In the **removeStudent()** method, the loop runs from **i = 0** to **students.size() - 1.** As soon as a match is found and removed, loop breaks. If not found, the loop ends after reaching the last element. • **getStop(String name):** It either returns a stop when found or ends after checking all stops. • **getStudent(int id):** Check each student if match is found ends. • **changeStop() & removeStop():**Work the same way iterate with a clear end.
Done Condition	• **getStop(name):** Done when matching stop is found or loop completes. • **changeStop(name, stop):** Done when stop replaced or loop completes. • **removeStop(name):** Done when stop removed or loop completes. • **getStudent(idNumber), removeStudent(idNumber):** Done when match found or list ends. • **deleteRoute(route):** Done when this bus object is removed from route list or list ends

Inductive Observation:

Inductive observation means we check each method's steps to confirm that it makes progress and will eventually finish. Just like in Bus Class the methods shows:

- **Simple methods** (getLicensePlateNumber(), getDriver(), start(), stop(), accelerate(), brake().
- **Loop based methods (getStop, changeStop, removeStop, getStudent, removeStudent)**

These use loops that increment i every iteration, and eventually i == list.size() and loops end naturally. Every method like above takes one or more steps toward a result. No method causes an infinite loop or crash.

Conclusion:

From observing the methods in the Bus class, I concluded that all of them are designed to make proper progress and achieve a goal without causing any errors or infinite loops. Each loop has a clear stopping point, such as finding a matching student or stop. The bus speed increases or decreases step-by-step, it never goes negative and only increases when the bus is actually running. Also the state of the bus whether it's running or stopped, or how many students are on board, updates logically after each action. This makes the Bus class **highly reliable**, **logically structured**, and **perfect for managing vehicle operations in a real system** and follows a safe and reliable structure where methods work efficiently, terminate properly, making the code correct, dependable, and easy to maintain.