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 & Vehicle Class

Statement Type	Explanation
Pre-Conditions	• The **accelerate()** method should only be called if the bus is already running, i.e., isRunning = = true. • All important objects like **driver, students list, and stops list** should already be created and ready to use. • When calling **removeStudent()** or **removeStop(),** make sure the student or stop actually **exists** in the list. • While adding a student, make sure **no other student with the same ID** is already added to avoid duplicates.
Post-Conditions	• **Start():** Bus is now in running state. isRunning = true and it's ready to move. • **Stop():** Stops the bus and resets speed to zero. • **AddStudent():** A new student is added to the list, so the total number of students increases. • **RemoveStudent():** A student is removed from the list, so the total number of students decreases. • **Accelerate(10):** If the bus is running, its speed increases by 10. If it's not running, nothing changes. • **Brake():** Speed goes down, but never below 0. It's like slowing down safely. • **DeleteStop():** If the stop exists in the list, it is removed and a success message is shown. If not a "not found" message is printed the route list is updated accordingly.
Initialization Statements	• speed = 0 • isRunning = false, means bus starts at rest. • licensePlate = "Null", Default value. • students = new ArrayList<>() • route = new ArrayList<>(): is initialized in the field declaration. This setup ensures the bus object is safe, clean, and ready to use without errors.

Invariants	• If the bus is not running, then the speed must always be zero**.** • There should never be two students with the same ID in the list. • Two stops with the same name should not exist in the route. • If somehow speed becomes negative, it must be corrected back to zero like in **brake()** method.
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	• **removeStudent()** stops after finding and removing the target student Or if it reaches the end of the list. • **start()** and **stop()** immediately end after updating the bus status. • **Accelerate() and Brake():** End after one change in speed no loop inside. • **getStop():**Ends after match found or after checking all stops.

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:

- **accelerate(int speed)**: Increases speed by given value (e.g., 20 to 30 to 40).
- **brake()**: Reduces speed by 5 each time. Eventually hits 0 and never goes negative.
- **removeStudent(int id)**: Loops through student list to find and remove a student. If not found, ends after full check.
- **getStop(String name)**: Searches stop list until a match is found or it reaches the end.
- **Start() and stop**(): No loop Just action start and Stop **.**

Every method 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.