King Fahd University of Petroleum and Minerals

College of Computing and Mathematics

Information and Computer Science Department (ICS)

Software Testing (SWE326)

Spring semester – 242

Master test plan for the course project

Name	ID
Zainab Alturaiki	202156350
Renad Alqahtani	202169930
Noor Alqatri	202174930

History

Version	Date	Author	Changes
1.0	24/04/2025	Zainab, Noor, Renad	Added sections 1, 2, and 3
2.0	24/04/2025	Zainab, Noor, Renad	Added sections 3
3.0	9/05/2025	Zainab, Noor, Renad	Added sections 4 and 5

Table of Contents

1. Introduction

1.1. Project Overview

1.1.1. Test Organization

The test team comprises three members, Zainab Alturaiki, Noor Alqatari, and Renad Alqahtani, each of whom will serve as the lead for one of the three test phases:

- **Phase 1 (Static Code Analysis):** Zainab Alturaiki
- **Phase 2 (System Testing):** Noor Alqatari
- **Phase 3 (Unit Testing):** Renad Alqahtani

1.1.2. System Under Test (SUT)

The SUT is the **Cruise Control** simulation application, implemented entirely within a single Java package, appCruise. Its high-level characteristics comprises six source files, five concrete classes:

1. CarSimulator
2. Controller
3. CruiseControl
4. CruiseDisplay
5. SpeedControl

Plus one interface: CarSpeed. Together these files total 522 lines, approximately 440 non-blank lines, and define around 43 methods, including constructors and the two interface operations. The codebase encapsulates the complete system under test: an engine and throttle/brake simulator (CarSimulator), a graphical display of vehicle and cruise-control status (CruiseDisplay), a user-facing Swing frame wiring buttons to commands (CruiseControl), a stateful cruise-control manager (Controller), and a feedback controller thread that maintains the set speed (SpeedControl) via the CarSpeed interface.

1.1.3. Testing Types

The quality of the Cruise Control system will be assured through three complementary testing approaches:

1. **Static Code Analysis**
   We will perform a two-step review of the Java source:
   - a manual walkthrough of each class against our inspection checklist to uncover design, style, documentation, and correctness issues
   - an automated scan using PMD (via IntelliJ) to detect rule-based violations, compute defect density, and generate a consolidated static-analysis report.
2. **System Testing**
   Black-box tests will exercise the assembled application (the appCruise package running in its Swing GUI) against the 17 functional requirements (R1–R17). Test cases will validate end-to-end behavior, engine on/off, cruise on/off/resume, brake/accelerator interactions, odometer updates, and status displays, without peeking at internal code.
3. **Unit Testing**
   We will write and execute JUnit tests for each class and public API (including boundary and error conditions), and measure code coverage with EclEmma. This will ensure that individual methods (e.g. CarSimulator.run(), SpeedControl.run(), controller state transitions, GUI helpers) behave correctly in isolation before integration.

1.1.4. Project Phases

The cruise-control project is organized into four sequential phases:

1. **Phase 1: Static Code Analysis**
   - Manual inspection of all six source files against the inspection checklist
   - Automated rule-based scan using PMD in IntelliJ, with screenshots of violation reports
2. **Phase 2: System Testing**
   - Black-box test planning based on the 17 functional requirements (R1–R17)
   - Execution of end-to-end test cases against the Swing GUI

    - Documentation of actual vs. expected behavior in the Master Test Plan
3. **Phase 3: Unit Testing**
    - Design and implementation of JUnit test cases for each public method, including boundary and error conditions
    - Measurement of code coverage via EclEmma and collection of coverage reports

1.1.5. Features to be tested

We will verify:

- **Engine controls**: engineOn(), engineOff(), and their effects on simulation state and GUI indicator (R1, R5, R8, R9, R11)
- **Cruise-control commands**: on(), off(), resume() and interactions with throttle/brake (R2–R4, R7, R13, R14, R16, R17)
- **Speed-holding logic**: ability of SpeedControl feedback loop to maintain the set speed within ±1.5 km/h (R6, R15)
- **Display elements**: correct recorded speed, Enabled/Disabled status, green/red indicator, and odometer updates (R6, R10, R12)
- **State transitions**: the Controller's internal controlState progression under all valid command sequences

1.1.6. Features to be tested

Out of scope for this plan:

- Performance and load testing (like thread-scheduling under heavy CPU load)
- Security, authentication, and network integration
- Installation, deployment, and cross-platform compatibility
- Usability, accessibility, and aesthetic GUI design beyond functional correctness
- Localization / internationalization
- Hardware or real-vehicle interfacing

1.2. Test Environment and Tools

- IntelliJ IDE (with PMD plugin) for manual and automated static analysis
- PMD for rule-based code inspections
- JUnit for unit-test development and execution
- EclEmma (JaCoCo) for execution-coverage measurement
- Java SE 8+ environment to run the Swing-based CruiseControl application for system tests

1.3. Document Structure

This document follows (not strictly) the template for test plans in the IEEE standard for software test documentation [IEEE829].

2. Static code analysis

2.1. Manual code inspection

Inspectors Names: Zainab Alturaiki
Date Started: April 24th, 2025
Date Completed: April 28th, 2025
Project Name: Cruise Control simulation application
Time Report: total of around 8 hours

Time spent on Familiarization with the Code: 1.5 hours
Time spent on Filling in Findings Report: 6.5 hours
Time spent on Actual Inspection: 6.5 hours

Findings:

#	Description	Line #	Class/Method Name	Associated Checklist Item #
1	Missing class-level documentation. No Javadoc or block comment at top explaining the role and responsibilities of SpeedControl.	1–4	SpeedControl class	7.2
2	No method-level documentation. None of the public/package methods have Javadoc or block comments describing their purpose, parameters, or behavior.	18, 22, 26, 35, 39	SpeedControl class : recordSpeed(), clearSpeed(), enableControl(), disableControl(), run()	7.2
3	Missing @Override annotation. The run() method implements Runnable.run() but lacks the @Override marker.	39	SpeedControl class : run() method	9.3
4	Poorly descriptive field names. Abbreviations cs and disp are too terse; better to use names like carSpeed and display for clarity.	10–11	SpeedControl class	2.5
5	Magic numbers. Literals 10 in clearSpeed(), and 6.0, 12.0, 500 in run() should be replaced by well-named static	23, 42–45	clearSpeed(), run()	7.5

	final constants.			
6	Missing toString() method. No override of toString() to aid logging for debugging.	N/A	SpeedControl class	10.1
7	Inline comments only. Block comments should precede code sections (e.g. above the feedback loop in run()) instead of trailing an inline comment.	39–40	run()	2.3
8	Inconsistent indentation/style. In clearSpeed(), the opening brace sits on the same line as code with no space, violating the class's usual brace/indent style.	22–23	clearSpeed()	2.2
9	No null-check on constructor args. If a null CarSpeed or CruiseDisplay is passed, future calls to cs.getSpeed() or disp. will be NPE.	13–16	SpeedControl class constructor	3.1
10	Missing class-level Javadoc. No high-level comment describing the purpose and responsibilities of CruiseDisplay.	1-5	CruiseDisplay class	7.2
11	No method-level documentation. All public methods lack any Javadoc or block comment explaining their intent, parameters, or effects.	14, 24, 41,45, 65, 73, 78, 83	CruiseDisplay class: constructor, backdrop(), paint(), update(), drawRecorded(), enabled(), disabled(), record()	7.2
12	Missing @Override. Both paint(Graphics) and update(Graphics) override AWT methods but have no @Overrideannotation.	41, 45	CruiseDisplay class: paint(), update()	9.3
13	Package-private fields. The image-buffer fields offscreen, offscreensize, and offgraphics are declared without any	20-22	CruiseDisplay class	9.1

	visibility modifier (default/package), exposing them outside the class. They should be private.			
14	Non-final Font fields. The small and big Font instances are never reassigned and should be declared private static final.	11–12	CruiseDisplay class	5
15	Magic numbers throughout. Literals such as 150, 260 (size), 5, 10 (offsets), 50, 20 (rect dims), 90 (arc x), and 200(botY) are used directly instead of named constants.	16, 36-38, 50-61, 65-69	CruiseDisplay class constructor, backdrop(), update(), drawRecorded()	7.5
16	Inline comment style. The single inline comment // display recorded speed should be a brief block comment placed immediately before the related code block.	47	CruiseDisplay class: update()	2.3
17	Vague field names. The identifiers small and big do not follow a consistent naming convention or convey their purpose clearly; better names would be smallFont and bigFont.	11-12	CruiseDisplay class	2.5
18	Missing toString() method. No override of toString() to aid logging for debugging.	N/A	CruiseDisplay class	10.1
19	Undocumented constructor logic. The default size (150,260) is set with no explanation or reference to requirements.	13–16	CruiseDisplay class constructor CruiseDisplay()	7.2
20	Missing class-level Javadoc. No high-level comment describing the purpose and responsibilities of CruiseDisplay.	1-4	CruiseControl class	7.2
21	No method-level documentation. None of the public/package methods have Javadoc or block comments describing their purpose, parameters, or behavior.	23, 31, 107, 112	CruiseControl class: CruiseControl() constructor, init(), stop(), and	7.2

			main()	
22	The method is very long and takes more than 60 lines.	32-106	CruiseControl class: init()	6.1
23	Package-private fields. All component fields lack any visibility modifier and default to package scope; they should be private.	13-22	CruiseControl class: car, disp, control, engineOn, engineOff, accelerate, brake, on, off, resume	9.1
24	Magic numbers. The literals 600 and 400 in setSize(600, 400) should be named constants (e.g. FRAME_WIDTH, FRAME_HEIGHT).	27	CruiseControl class:CruiseControl()	7.5
25	Inline comment style. The // Set the size of the frame comment appears on the same line as code; it should be a brief block comment immediately above that statement.	27	CruiseControl class:CruiseControl()	2.3
26	String literal layout keys. Uses "Center", "East", and "South" for add(...) instead of the constants BorderLayout.CENTER, etc., reducing clarity.	37-41 , 105	CruiseControl class: init()	7.5
27	Wildcard import. import javax.swing.*; pulls in all Swing types; best practice is to import only the specific classes needed.	9	CruiseControl class	2.5
28	Missing toString() method. No override of toString() to aid logging for debugging.	N/A	CruiseControl class	10.1
29	Dead/commented code. The lines //String fixed = getParameter("fixed"); and related commented logic remain in init(). Unused code should be removed.	33-34	CruiseControl class: init()	2.3
30	Undocumented anonymous listeners. Each new ActionListener() {…} block has no comment describing which functional	44–93	init() button handlers	7.2

	requirement it implements, making it harder to trace behavior to requirements.			
31	Missing class-level Javadoc. No high-level comment explains the role of Controller in managing cruise-control state transitions.	1–3	Controller class	7.2
32	Constants lack visibility and conventional order. The state constants use package scope and are declared final static instead of private static final.	4–7	Controller class	2.5
33	Poor naming conventions. Field isfixed should be isFixed (CamelCase), and constructor parameter b is non-descriptive.	8-12	field isfixed & Constructor	2.5
34	No method-level documentation. None of the API methods (brake(), accelerator(), engineOn(), and so on) have Javadoc or block comments.	15–56	all synchronized methods in Controller class	7.2
35	Missing null-checks on constructor args. Passing a null CarSpeed or CruiseDisplay will cause NPEs when sc is used.	12–13	Controller class Constructor	3.1
36	No toString() override. Lacking a parameterless toString() method for logging or debugging the controller's current state.	NA	Controller class	10.1
37	Inconsistent brace/indent style. Some methods use braces on new lines, others inline, and indentation levels vary, obscuring block structure.	multiple	all methods	2.2
38	Fields could be final. sc and isfixed are only set in the constructor and never reassigned, declaring them private final improves clarity.	9-10	Controller fields	5.1
39	Unclear behavior on invalid transitions. Methods like resume() do nothing if	53–57	resume() & other state	3.3

	preconditions aren't met, without comment or exception to indicate why.		methods	
40	Missing block comments. Complex state-transition logic (e.g. in off()) isn't introduced with a brief comment explaining each branch's intent.	multiple	all methods	2.3
41	Missing interface-level Javadoc. No block comment or Javadoc at the top describing the purpose of CarSpeed.	1–3	CarSpeed interface	7.2
42	Missing method-level Javadoc. Both getSpeed() and setThrottle(double) lack Javadoc detailing their behavior, parameters, and expected results.	5, 7	CarSpeed interface: getSpeed() and setThrottle()	7.2
43	Redundant public modifiers. In an interface, methods are public by default—public is unnecessary in their declarations.	5, 7	CarSpeed interface: getSpeed() and setThrottle()	6.2
44	Missing class-level Javadoc. No header comment explains the role of CarSimulator as both a GUI canvas and the engine simulation.	1-4	CarSimulator class	7.2
45	Inconsistent modifier order. Fields are declared as volatile private rather than the consistent private volatile.	5–13	CarSimulator class : Field declarations	2.2
46	Constants declared as final static instead of the recommended private static final, and lack private visibility.	15–21	CarSimulator class : Field declarations	2.5/9.1
47	Magic numbers everywhere (e.g. setSize(300,260), arc widths/heights, throttle scaling, tick timing); these should be named static final constants with descriptive names.	25, 40-66	CarSimulator class : Constructor & draw methods	7.5
48	Inline/trailing comments (e.g. //engine off, //inverse air resistance factor) rather than brief block comments immediately above the code they document.	8-21	CarSimulator class : Field declarations	2.3/7.1

49	Buffer fields lack private. Image offscreen; Dimension offscreensize; Graphics offgraphics; default to package scope and should be private.	29-31	CarSimulator class : Field declarations	9.1
50	Missing @Override. The AWT callbacks paint(Graphics) and update(Graphics) override parent methods but are not annotated.	45,49	paint() / update()	9.3
51	No method-level Javadoc. None of the public APIs (backdrop(), paint(), update(), engineOn(), accelerate(), etc.) are documented with purpose, parameters, or side-effects.	multiple	CarSimulator class: all methods	7.2
52	Poor naming for constants/fields. Single-letter or mixed-case constants (X, botY, airResistance) break naming conventions—should use ALL_CAPS_WITH_UNDERSCORES.	15–21	CarSimulator class : Field declarations	2.5
53	Missing toString(). No parameterless toString() is provided to dump the simulator's key state (speed, distance, ignition).	NA	CarSimulator class	10.1

2.2. Automated static code analysis

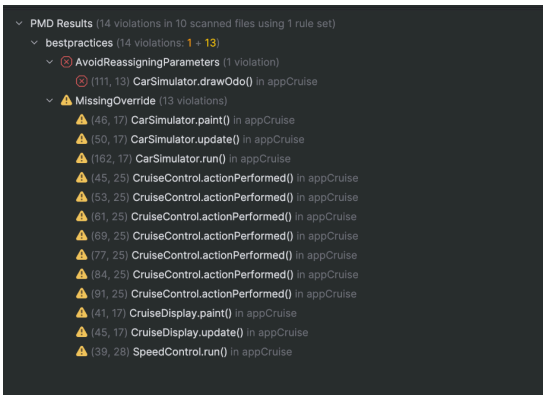
2.2.1. PMD Static Code Analysis Tool

PMD is an open-source, rule-based static code analysis tool for Java (and other languages) that scans source code to detect common programming flaws—such as unused variables, empty catch blocks, unnecessary object creation, and suboptimal code patterns—alongside violations of coding standards and best practices. It ships with a rich, configurable set of built-in rule sets (covering design, performance, security, and documentation), can be integrated into IDEs like Eclipse or IntelliJ IDEA (via plugins), and can be run as part of build processes (Ant, Maven, Gradle) to automatically enforce quality gates and generate detailed violation reports.

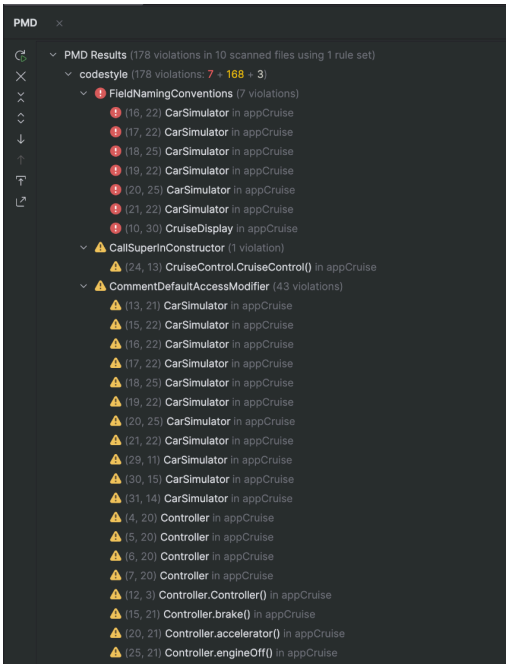
2.2.2. Produced reports

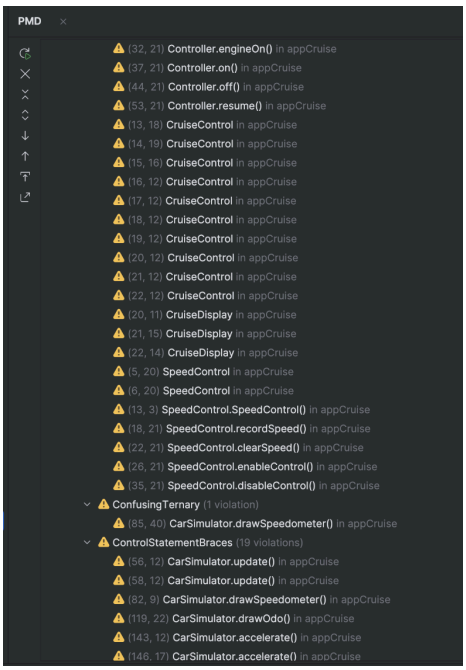
After running the PMD plugin in IntelliJ IDEA against our six-file appCruise package, the analysis report identified 293 total violations, categorized as follows: 14 Best-Practices issues, 178 Code-Style violations, 11 Design flaws, 72 Documentation gaps, and 18 Error-Prone patterns. Screenshots of each category's findings, directly from IntelliJ's PMD tool window, are included below.

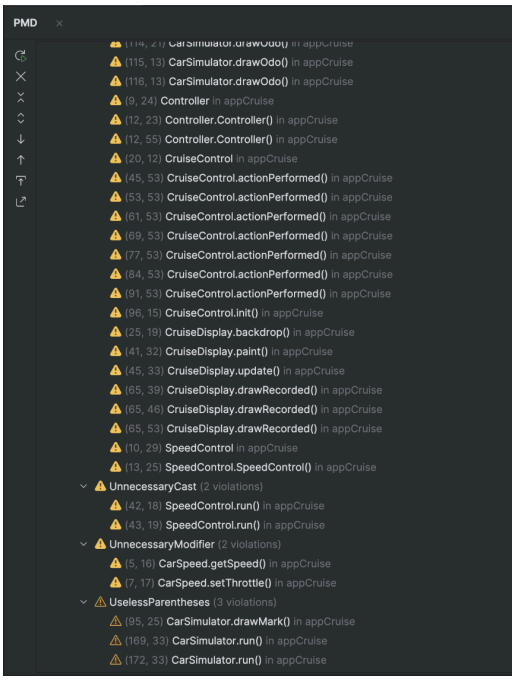
1. Best practices violations

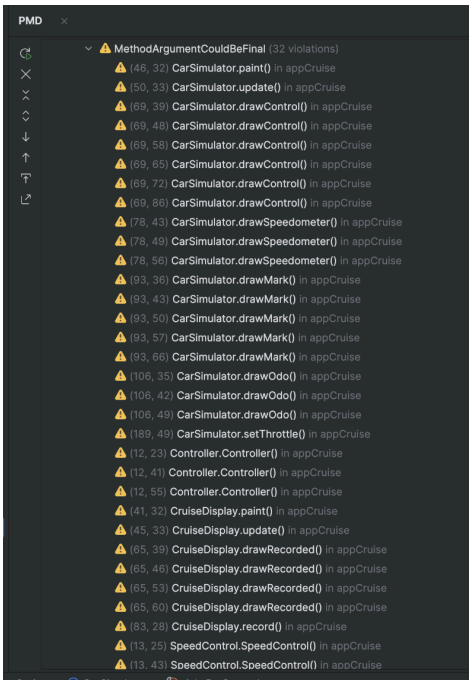


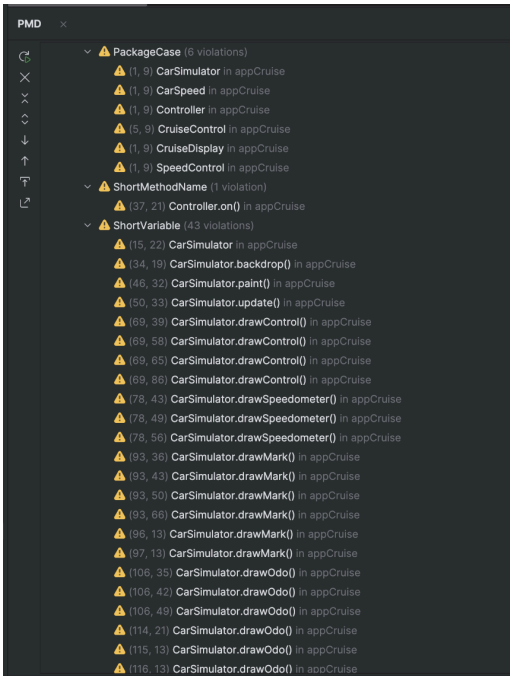
2. code style violations

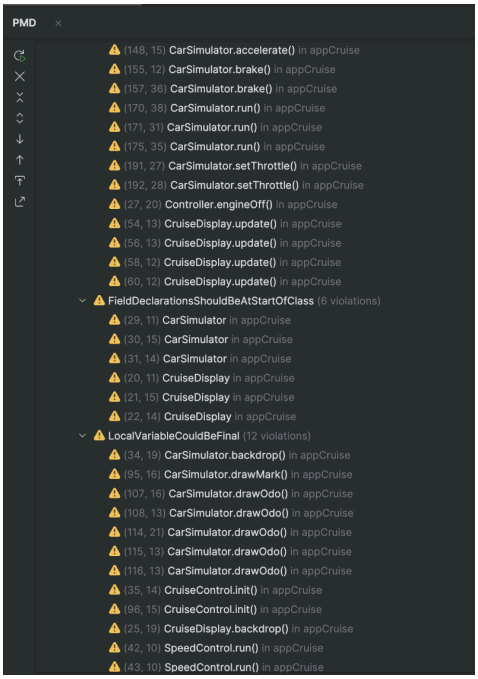




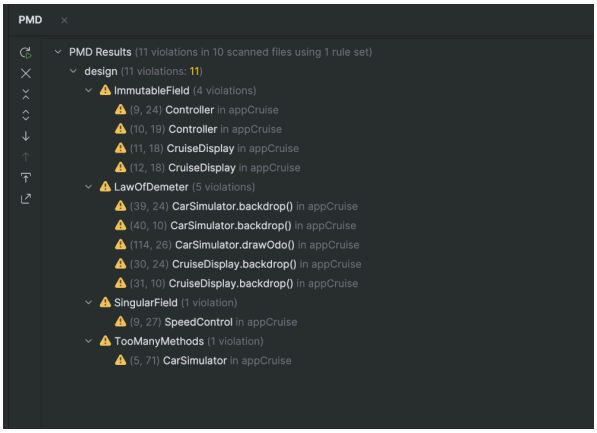

PMD
(115, 13) CarSimulator.drawOdo() in appCruise
(116, 13) CarSimulator.drawOdo() in appCruise
(9, 24) Controller in appCruise
(12, 23) Controller.Controller() in appCruise
(12, 55) Controller.Controller() in appCruise
(20, 12) CruiseControl in appCruise
(45, 53) CruiseControl.actionPerformed() in appCruise
(53, 53) CruiseControl.actionPerformed() in appCruise
(61, 53) CruiseControl.actionPerformed() in appCruise
(69, 53) CruiseControl.actionPerformed() in appCruise
(77, 53) CruiseControl.actionPerformed() in appCruise
(84, 53) CruiseControl.actionPerformed() in appCruise
(91, 53) CruiseControl.actionPerformed() in appCruise
(96, 15) CruiseControl.init() in appCruise
(25, 19) CruiseDisplay.backdrop() in appCruise
(41, 32) CruiseDisplay.paint() in appCruise
(45, 33) CruiseDisplay.update() in appCruise
(65, 39) CruiseDisplay.drawRecorded() in appCruise
(65, 46) CruiseDisplay.drawRecorded() in appCruise
(65, 53) CruiseDisplay.drawRecorded() in appCruise
(10, 29) SpeedControl in appCruise
(13, 25) SpeedControl.SpeedControl() in appCruise
UnnecessaryCast (2 violations)
(42, 18) SpeedControl.run() in appCruise
(43, 19) SpeedControl.run() in appCruise
UnnecessaryModifier (2 violations)
(5, 16) CarSpeed.getSpeed() in appCruise
(7, 17) CarSpeed.setThrottle() in appCruise
UselessParentheses (3 violations)
(95, 25) CarSimulator.drawMark() in appCruise
(169, 33) CarSimulator.run() in appCruise
(172, 33) CarSimulator.run() in appCruise


PMD
MethodArgumentCouldBeFinal (32 violations)
(46, 32) CarSimulator.paint() in appCruise
(50, 33) CarSimulator.update() in appCruise
(69, 39) CarSimulator.drawControl() in appCruise
(69, 48) CarSimulator.drawControl() in appCruise
(69, 58) CarSimulator.drawControl() in appCruise
(69, 65) CarSimulator.drawControl() in appCruise
(69, 72) CarSimulator.drawControl() in appCruise
(69, 86) CarSimulator.drawControl() in appCruise
(78, 43) CarSimulator.drawSpeedometer() in appCruise
(78, 49) CarSimulator.drawSpeedometer() in appCruise
(78, 56) CarSimulator.drawSpeedometer() in appCruise
(93, 36) CarSimulator.drawMark() in appCruise
(93, 43) CarSimulator.drawMark() in appCruise
(93, 50) CarSimulator.drawMark() in appCruise
(93, 57) CarSimulator.drawMark() in appCruise
(93, 66) CarSimulator.drawMark() in appCruise
(106, 35) CarSimulator.drawOdo() in appCruise
(106, 42) CarSimulator.drawOdo() in appCruise
(106, 49) CarSimulator.drawOdo() in appCruise
(189, 49) CarSimulator.setThrottle() in appCruise
(12, 23) Controller.Controller() in appCruise
(12, 41) Controller.Controller() in appCruise
(12, 55) Controller.Controller() in appCruise
(41, 32) CruiseDisplay.paint() in appCruise
(45, 33) CruiseDisplay.update() in appCruise
(65, 39) CruiseDisplay.drawRecorded() in appCruise
(65, 46) CruiseDisplay.drawRecorded() in appCruise
(65, 53) CruiseDisplay.drawRecorded() in appCruise
(65, 60) CruiseDisplay.drawRecorded() in appCruise
(83, 28) CruiseDisplay.record() in appCruise
(13, 25) SpeedControl.SpeedControl() in appCruise
(13, 43) SpeedControl.SpeedControl() in appCruise

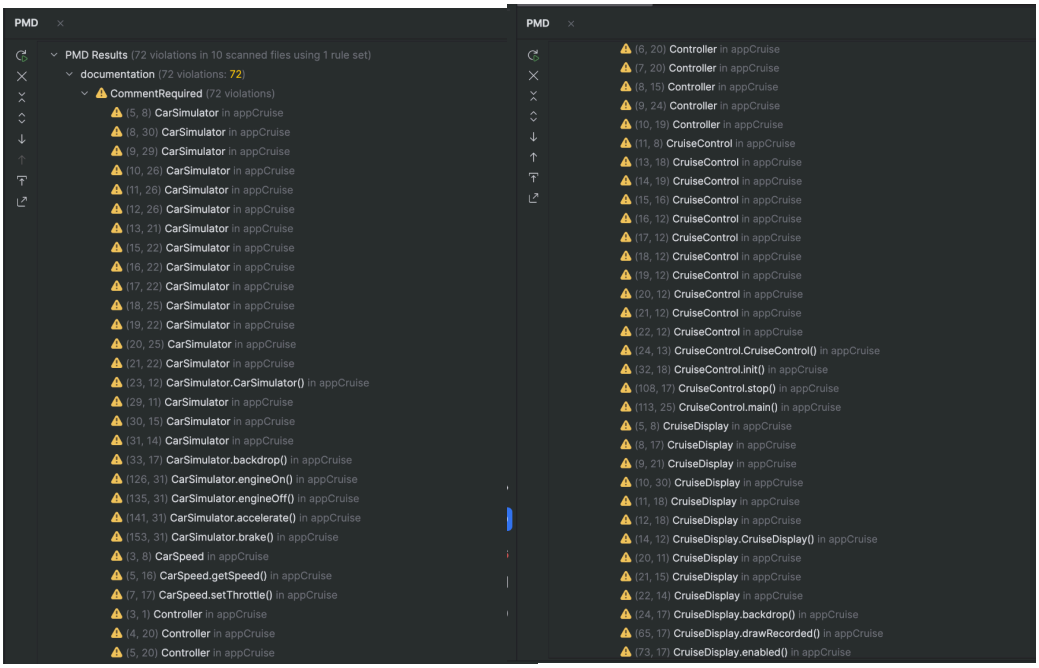

PMD
PackageCase (6 violations)
(1, 9) CarSimulator in appCruise
(1, 9) CarSpeed in appCruise
(1, 9) Controller in appCruise
(5, 9) CruiseControl in appCruise
(1, 9) CruiseDisplay in appCruise
(1, 9) SpeedControl in appCruise
ShortMethodName (1 violation)
(37, 21) Controller.on() in appCruise
ShortVariable (43 violations)
(15, 22) CarSimulator in appCruise
(34, 19) CarSimulator.backdrop() in appCruise
(46, 32) CarSimulator.paint() in appCruise
(50, 33) CarSimulator.update() in appCruise
(69, 39) CarSimulator.drawControl() in appCruise
(69, 58) CarSimulator.drawControl() in appCruise
(69, 65) CarSimulator.drawControl() in appCruise
(69, 86) CarSimulator.drawControl() in appCruise
(78, 43) CarSimulator.drawSpeedometer() in appCruise
(78, 49) CarSimulator.drawSpeedometer() in appCruise
(78, 56) CarSimulator.drawSpeedometer() in appCruise
(93, 36) CarSimulator.drawMark() in appCruise
(93, 43) CarSimulator.drawMark() in appCruise
(93, 50) CarSimulator.drawMark() in appCruise
(93, 66) CarSimulator.drawMark() in appCruise
(96, 13) CarSimulator.drawMark() in appCruise
(97, 13) CarSimulator.drawMark() in appCruise
(106, 35) CarSimulator.drawOdo() in appCruise
(106, 42) CarSimulator.drawOdo() in appCruise
(106, 49) CarSimulator.drawOdo() in appCruise
(114, 21) CarSimulator.drawOdo() in appCruise
(115, 13) CarSimulator.drawOdo() in appCruise
(116, 13) CarSimulator.drawOdo() in appCruise

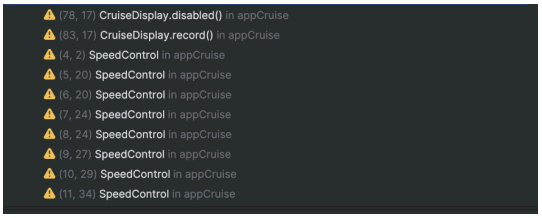

PMD
(148, 15) CarSimulator.accelerate() in appCruise
(155, 12) CarSimulator.brake() in appCruise
(157, 36) CarSimulator.brake() in appCruise
(170, 38) CarSimulator.run() in appCruise
(171, 31) CarSimulator.run() in appCruise
(175, 35) CarSimulator.run() in appCruise
(191, 27) CarSimulator.setThrottle() in appCruise
(192, 28) CarSimulator.setThrottle() in appCruise
(27, 20) Controller.engineOff() in appCruise
(54, 13) CruiseDisplay.update() in appCruise
(56, 13) CruiseDisplay.update() in appCruise
(58, 12) CruiseDisplay.update() in appCruise
(60, 12) CruiseDisplay.update() in appCruise
FieldDeclarationsShouldBeAtStartOfClass (6 violations)
(29, 11) CarSimulator in appCruise
(30, 15) CarSimulator in appCruise
(31, 14) CarSimulator in appCruise
(20, 11) CruiseDisplay in appCruise
(21, 15) CruiseDisplay in appCruise
(22, 14) CruiseDisplay in appCruise
LocalVariableCouldBeFinal (12 violations)
(34, 19) CarSimulator.backdrop() in appCruise
(95, 16) CarSimulator.drawMark() in appCruise
(107, 16) CarSimulator.drawOdo() in appCruise
(108, 13) CarSimulator.drawOdo() in appCruise
(114, 21) CarSimulator.drawOdo() in appCruise
(115, 13) CarSimulator.drawOdo() in appCruise
(116, 13) CarSimulator.drawOdo() in appCruise
(35, 14) CruiseControl.init() in appCruise
(96, 15) CruiseControl.init() in appCruise
(25, 19) CruiseDisplay.backdrop() in appCruise
(42, 10) SpeedControl.run() in appCruise
(43, 10) SpeedControl.run() in appCruise

3. design violations

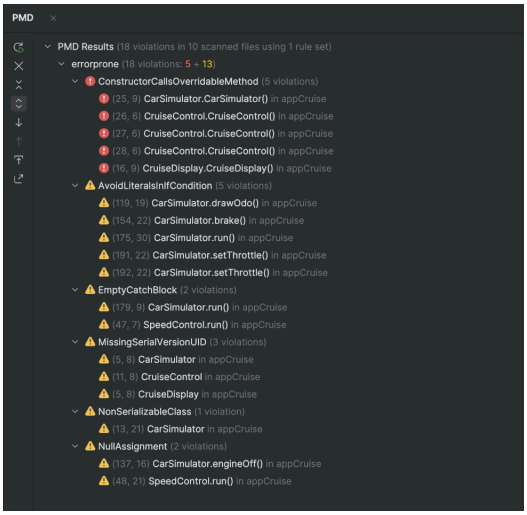


4. documentation violations





5. error prone violations



3. System test

3.1. System requirements

Requirements	Description
R1	To activate Cruise Control, the engine should be turned on
R2	The activation of the CC will hold the vehicle speed at the selected value.
R3	When the CC is enabled, accelerating the vehicle will disable the CC.
R4	When the CC is enabled, braking will disable the CC.
R5	The user can turn the engine off anytime.
R6	When CC is enabled, the CC GUI should store and display the correct speed.
R7	Pressing the resume button should set the CC to last fixed speed
R8	When we turn the engine on, the CC should be off.
R9	When the engine is turned off, the CC should be turned off.
R10	The odometer should show the correct distance when the CC is enabled/disabled.
R11	When the engine is off, clicking on the CC on/off should have no impact.
R12	System should display the Cruise control status (Green for on, Red for off).
R13	When the CC is turned on, clicking on the button “on” should have no impact.
R14	When the CC is turned off, clicking on the button “off” should have no impact.
R15	The CC cannot be set for a speed less than 40km/h
R16	The user should be able to run turn the CC on and off with no

	limit.
R17	When the engine is turned off then on, the CC should be turned off and showing the initial speed as zero.

3.2. System test cases

Activate CC with engine ON (TC001)

Test Case Title	Activate CC with engine ON
Test Case ID	TC001
Requirement IDs	R1
Author	Noor Alqatri
Description	Verify that Cruise Control activates successfully when the engine is ON.
Initial State	Engine ON, Speed $\geq$ 40 km/h
Pre-conditions	Engine running
Test steps (inputs, actions, etc.)	Turn ON engine and activate Cruise Control.
Expected Output	Cruise Control activates.
Priority	High
Test Type	Functional

Activate CC with engine OFF (TC002)

Test Case Title	Activate CC with engine OFF
Test Case ID	TC002
Requirement IDs	R1
Author	Noor Alqatri
Description	Verify that Cruise Control cannot be activated if the engine is OFF.
Initial State	Engine OFF
Pre-conditions	Car powered but engine OFF
Test steps (inputs, actions, etc.)	Try activating Cruise Control when engine is OFF.
Expected Output	Cruise Control does not activate.
Priority	High
Test Type	Functional

Maintain speed after CC activation (TC003)

Test Case Title	Maintain speed after CC activation
Test Case ID	TC003
Requirement IDs	R2
Author	Noor Alqatri
Description	Verify that Cruise Control maintains the selected speed.
Initial State	CC ON at 60 km/h
Pre-conditions	Cruise Control active
Test steps (inputs, actions, etc.)	Activate Cruise Control at 60 km/h. Observe

	maintained speed
Expected Output	Speed maintained within ±1.5 km/h.
Priority	High
Test Type	Functional

Acceleration disables CC (TC004)

Test Case Title	Acceleration disables CC
Test Case ID	TC004
Requirement IDs	R3
Author	Noor Alqatri
Description	Ensure that accelerating disables Cruise Control.
Initial State	CC ON
Pre-conditions	Vehicle running
Test steps (inputs, actions, etc.)	Press accelerator while CC is active.
Expected Output	CC disables.
Priority	High
Test Type	Functional

Braking disables CC (TC005)

Test Case Title	Braking disables CC
Test Case ID	TC005
Requirement IDs	R4
Author	Noor Alqatri
Description	Ensure that braking disables Cruise Control.

Initial State	CC ON
Pre-conditions	Vehicle running
Test steps (inputs, actions, etc.)	Press brake while CC is active.
Expected Output	CC disables.
Priority	High
Test Type	Functional

Turn off engine when CC ON (TC006)

Test Case Title	Turn off engine while Cruise Control is ON
Test Case ID	TC006
Requirement IDs	R5
Author	Noor Alqatri
Description	Verify that the engine can be turned off while Cruise Control is active.
Initial State	Engine ON, CC ON
Pre-conditions	Cruise Control active
Test steps (inputs, actions, etc.)	1. Activate Cruise Control.2. Turn off the engine.
Expected Output	Engine stops immediately, CC deactivates.
Priority	High
Test Type	Functional

Turn off engine when CC OFF (TC007)

Test Case Title	Turn off engine while Cruise Control is OFF
Test Case ID	TC007

Requirement IDs	R5
Author	Noor Alqatri
Description	Verify that the engine can be turned off while Cruise Control is inactive.
Initial State	Engine ON, CC OFF
Pre-conditions	Cruise Control deactivated
Test steps (inputs, actions, etc.)	1. Ensure CC is OFF.2. Turn off the engine.
Expected Output	Engine stops immediately without any CC impact.
Priority	Medium
Test Type	Functional

GUI displays correct CC speed (TC008)

Test Case Title	GUI displays correct CC speed
Test Case ID	TC008
Requirement IDs	R6
Author	Noor Alqatri
Description	Ensure GUI shows the correct cruise speed.
Initial State	Engine ON, CC OFF
Pre-conditions	Vehicle moving at a stable speed
Test steps (inputs, actions, etc.)	1. Press Cruise Control ON button.2. Observe the GUI speed display and compare it with the speedometer.
Expected Output	GUI displays correct speed.
Priority	High
Test Type	Functional

GUI displays wrong CC speed (TC009)

Test Case Title	GUI displays correct speed after fluctuating
Test Case ID	TC009
Requirement IDs	R6
Author	Noor Alqatri
Description	Verify that after unstable vehicle speeds, the GUI correctly displays the current stable speed when Cruise Control is activated.
Initial State	Engine ON, CC OFF
Pre-conditions	Vehicle moving at a stable speed
Test steps (inputs, actions, etc.)	1.Change speed between 50–70 km/h. 2. Stabilize at 60 km/h. 3. Activate CC. 4. Check GUI speed.
Expected Output	GUI displays correct speed 60 km/h,not an old value.
Priority	High
Test Type	Functional

Resume restores last cruise speed (TC010)

Test Case Title	Resume restores last cruise speed
Test Case ID	TC010
Requirement IDs	R7
Author	Noor Alqatri
Description	Verify that Resume button restores last cruise speed.
Initial State	CC was active then turned off

Pre-conditions	Vehicle must have a previously saved CC speed.
Test steps (inputs, actions, etc.)	1. Press Resume button. 2. Observe CC behavior.
Expected Output	CC resumes previous speed.
Priority	High
Test Type	Functional

Resume pressed with no saved speed (TC011)

Test Case Title	Resume pressed with no saved speed
Test Case ID	TC011
Requirement IDs	R7
Author	Noor Alqatri
Description	Verify behavior when Resume is pressed without previous cruise speed.
Initial State	Engine ON, no prior CC activation
Pre-conditions	No previous CC activation must exist.
Test steps (inputs, actions, etc.)	1. Press Resume button. 2. Observe behavior.
Expected Output	No action or error shown.
Priority	High
Test Type	Functional

Engine ON keeps CC OFF (TC012)

Test Case Title	Engine ON keeps CC OFF
Test Case ID	TC012
Requirement IDs	R8
Author	Noor Alqatri
Description	Ensure that turning engine ON keeps CC OFF initially.
Initial State	Engine OFF, vehicle ready
Pre-conditions	Vehicle must be newly started.
Test steps (inputs, actions, etc.)	1. Start engine. 2. Check CC status.
Expected Output	Cruise Control remains OFF.
Priority	High
Test Type	Functional

Engine OFF disables CC automatically (TC013)

Test Case Title	Engine OFF disables CC
Test Case ID	TC013
Requirement IDs	R9
Author	Noor Alqatri
Description	Ensure that engine OFF disables CC.
Initial State	Vehicle moving with CC ON
Pre-conditions	Cruise Control must be active.
Test steps (inputs, actions, etc.)	1. Turn off the engine. 2. Check CC status.
Expected Output	Cruise Control deactivates.

Priority	High
Test Type	Functional

Odometer correct with CC ON (TC014)

Test Case Title	Odometer correct with CC ON
Test Case ID	TC014
Requirement IDs	R10
Author	Noor Alqatri
Description	Verify odometer reading with CC ON.
Initial State	Vehicle moving with CC ON
Pre-conditions	Cruise Control must be active while driving.
Test steps (inputs, actions, etc.)	1. Drive and monitor distance. 2. Compare recorded distance.
Expected Output	Odometer correctly shows distance.
Priority	High
Test Type	Functional

Odometer correct with CC OFF (TC015)

Test Case Title	Odometer correct with CC OFF
Test Case ID	TC015
Requirement IDs	R10
Author	Noor Alqatri
Description	Verify odometer reading with CC OFF.
Initial State	Vehicle moving with CC OFF
Pre-conditions	Cruise Control must be inactive while driving.

Test steps (inputs, actions, etc.)	1. Drive and monitor distance. 2. Compare recorded distance.
Expected Output	Odometer correctly shows distance.
Priority	High
Test Type	Functional

CC ON button ignored when engine OFF (TC016)

Test Case Title	CC button ignored when engine OFF
Test Case ID	TC016
Requirement IDs	R11
Author	Noor Alqatri
Description	Ensure CC button does not work when engine is OFF.
Initial State	Engine OFF
Pre-conditions	Vehicle must be fully powered off.
Test steps (inputs, actions, etc.)	1. Press Cruise Control ON button. 2. Observe behavior.
Expected Output	No action occurs.
Priority	High
Test Type	Functional

CC OFF button ignored when engine OFF (TC017)

Test Case Title	CC button ignored when engine OFF
Test Case ID	TC017
Requirement IDs	R11
Author	Noor Alqatri

Description	Ensure CC button does not work when engine is OFF.
Initial State	Engine OFF
Pre-conditions	Vehicle must be fully powered off.
Test steps (inputs, actions, etc.)	1. Press Cruise Control OFFbutton. 2. Observe behavior.
Expected Output	No action occurs.
Priority	High
Test Type	Functional

CC indicator green when ON (TC018)

Test Case Title	CC indicator green when ON
Test Case ID	TC018
Requirement IDs	R12
Author	Noor Alqatri
Description	Verify CC indicator shows green when ON.
Initial State	Engine ON, Cruise Control ON
Pre-conditions	Cruise Control must be active.
Test steps (inputs, actions, etc.)	1. Activate CC. 2. Check indicator.
Expected Output	Indicator is green.
Priority	High
Test Type	Functional

CC indicator red when OFF (TC019)

Test Case Title	CC indicator red when OFF

Test Case ID	TC019
Requirement IDs	R12
Author	Noor Alqatri
Description	Verify CC indicator shows red when OFF.
Initial State	Engine ON, Cruise Control OFF
Pre-conditions	Cruise Control must be inactive.
Test steps (inputs, actions, etc.)	1. Deactivate CC. 2. Check indicator.
Expected Output	Indicator is red.
Priority	High
Test Type	Functional

Click ON when CC already ON (TC020)

Test Case Title	Click ON when CC already ON
Test Case ID	TC020
Requirement IDs	R13
Author	Noor Alqatri
Description	Ensure pressing ON again does nothing when CC is already ON.
Initial State	Engine ON, Cruise Control ON
Pre-conditions	Cruise Control must be already active.
Test steps (inputs, actions, etc.)	1. Press ON button again. 2. Observe behavior.
Expected Output	No change occurs.
Priority	High
Test Type	Functional

Click OFF when CC already OFF (TC021)

Test Case Title	Click OFF when CC already OFF
Test Case ID	TC021
Requirement IDs	R14
Author	Noor Alqatri
Description	Ensure pressing OFF again does nothing when CC is already OFF.
Initial State	Engine ON, Cruise Control OFF
Pre-conditions	Cruise Control must be already inactive.
Test steps (inputs, actions, etc.)	1. Press OFF button again. 2. Observe behavior.
Expected Output	No change occurs.
Priority	High
Test Type	Functional

Activate CC below 40 km/h (TC022)

Test Case Title	Activate CC below 40 km/h
Test Case ID	TC022
Requirement IDs	R15
Author	Noor Alqatri
Description	Verify CC cannot be activated below 40km/h.
Initial State	Engine ON, Vehicle moving at 39 km/h
Pre-conditions	Vehicle must be moving slower than 40 km/h.

Test steps (inputs, actions, etc.)	1. Attempt to activate CC. 2. Observe behavior.
Expected Output	Cruise Control does not activate. (fail)
Priority	High
Test Type	Functional

Activate CC at 40 km/h (TC023)

Test Case Title	Activate CC at 40 km/h
Test Case ID	TC023
Requirement IDs	R15
Author	Noor Alqatri
Description	Verify CC can be activated at 40km/h.
Initial State	Engine ON, Vehicle moving at 40 km/h
Pre-conditions	Vehicle must reach 40 km/h exactly.
Test steps (inputs, actions, etc.)	1. Activate CC. 2. Observe behavior.
Expected Output	Cruise Control activates.
Priority	High
Test Type	Functional

Activate CC above 40 km/h (TC024)

Test Case Title	Activate CC above 40 km/h
Test Case ID	TC024
Requirement IDs	R15
Author	Noor Alqatri
Description	Verify CC can be activated above 40km/h.

Initial State	Engine ON, Vehicle moving at 50 km/h
Pre-conditions	Vehicle must be moving above 40 km/h.
Test steps (inputs, actions, etc.)	1. Activate CC. 2. Observe behavior.
Expected Output	Cruise Control activates.
Priority	High
Test Type	Functional

Repeatedly switch CC ON/OFF (few times) (TC025)

Test Case Title	Repeatedly switch CC ON/OFF few times
Test Case ID	TC025
Requirement IDs	R16
Author	Noor Alqatri
Description	Verify that switching Cruise Control ON and OFF multiple times does not cause errors during driving.
Initial State	Engine ON
Pre-conditions	Vehicle must be moving normally.
Test steps (inputs, actions, etc.)	1. Drive vehicle at 60 km/h.2. Activate Cruise Control.3. Deactivate Cruise Control.4. Repeat steps 2 and 3 three to five times.5. Observe Cruise Control behavior after each toggle.
Expected Output	No errors occur.
Priority	High
Test Type	Functional

Repeatedly switch CC ON/OFF (many times) (TC026)

Test Case Title	Repeatedly switch CC ON/OFF many times
Test Case ID	TC026
Requirement IDs	R16
Author	Noor Alqatri
Description	Verify that switching Cruise Control ON and OFF multiple times does not cause errors during driving.
Initial State	Engine ON
Pre-conditions	Vehicle must be moving normally.
Test steps (inputs, actions, etc.)	1. Drive the vehicle at 60 km/h.2. Activate Cruise Control.3. Deactivate Cruise Control.4. Repeat steps 2 and 3 at least 10 times.5. Monitor system behavior throughout the toggling.
Expected Output	No errors occur.
Priority	High
Test Type	Functional

Engine restart resets CC and speed (TC027)

Test Case Title	Engine restart resets CC and speed
Test Case ID	TC027
Requirement IDs	R17
Author	Noor Alqatri
Description	Ensure that engine restart resets CC and speed display.
Initial State	Vehicle OFF then ON again

Pre-conditions	Vehicle must be restarted after shutdown.
Test steps (inputs, actions, etc.)	1. Restart engine. 2. Check CC and speed display.
Expected Output	CC is OFF and speed shows 0 km/h.
Priority	High
Test Type	Functional

3.3. Report on the execution of the system test cases

TC ID	Verdict (Pass/Fail/ Pass with exception)	Comments (If fail or pass with exception (Actual output vs. Expected output))	Bug ID	Execution Date	Author	Engineer
TC001	Pass	Successfully activated Cruise Control when the engine was ON.		2025-04-28	Noor Alqatri	Renad Alqahtani
TC002	Fail	Cruise Control activated when engine OFF. (should not activate)	B001	2025-04-28	Noor Alqatri	Renad Alqahtani
TC003	Pass	Speed maintained within ±1.5 km/h as expected during CC activation.		2025-04-28	Noor Alqatri	Renad Alqahtani
TC004	Pass	Cruise Control disabled successfully after pressing the accelerator.		2025-04-28	Noor Alqatri	Renad Alqahtani
TC005	Pass with Exception	CC disabled when braking but missing log entry.	B002	2025-04-28	Noor Alqatri	Renad Alqahtani
TC006	Pass	Engine turned off and CC deactivated correctly.		2025-04-28	Noor Alqatri	Renad Alqahtani
TC007	Pass	Engine OFF had no impact on CC (expected).		2025-04-28	Noor Alqatri	Renad Alqahtani
TC008	Pass	GUI correctly displayed speed when CC ON.		2025-04-28	Noor Alqatri	Renad Alqahtani
TC009	Fail	GUI displayed old speed after fluctuation.	B003	2025-04-28	Noor Alqatri	Renad Alqahtani
TC010	Fail	Resume button failed to restore correct speed.	B004	2025-04-28	Noor Alqatri	Renad Alqahtani
TC011	Pass	Resume button with no saved speed showed no error.		2025-04-28	Noor Alqatri	Renad Alqahtani
TC012	Pass	Engine ON kept CC OFF (expected).		2025-04-28	Noor Alqatri	Renad Alqahtani
TC013	Pass	CC turned OFF automatically when the		2025-04-28	Noor Alqatri	Renad Alqahtani

		engine turned OFF.				
TC014	Pass	Odometer showed the correct distance with CC ON.		2025-04-28	Noor Alqatri	Renad Alqahtani
TC015	Pass	Odometer showed the correct distance with CC OFF.		2025-04-28	Noor Alqatri	Renad Alqahtani
TC016	Pass	Pressing the CC ON button when engine OFF had no effect.		2025-04-28	Noor Alqatri	Renad Alqahtani
TC017	Pass	Pressing the CC OFF button when engine OFF had no effect.		2025-04-28	Noor Alqatri	Renad Alqahtani
TC018	Pass	CC indicator green when ON.		2025-04-28	Noor Alqatri	Renad Alqahtani
TC019	Pass	CC indicator red when OFF.		2025-04-28	Noor Alqatri	Renad Alqahtani
TC020	Pass	Pressing ON when already ON had no effect.		2025-04-28	Noor Alqatri	Renad Alqahtani
TC021	Pass	Pressing OFF when already OFF had no effect.		2025-04-28	Noor Alqatri	Renad Alqahtani
TC022	Pass	CC did not activate below 40 km/h.		2025-04-28	Noor Alqatri	Renad Alqahtani
TC023	Pass	CC activated exactly at 40 km/h.		2025-04-28	Noor Alqatri	Renad Alqahtani
TC024	Pass	CC activated above 40 km/h.		2025-04-28	Noor Alqatri	Renad Alqahtani
TC025	Pass	Multiple CC ON/OFF toggles caused no error (few times).		2025-04-28	Noor Alqatri	Renad Alqahtani
TC026	Pass	Multiple CC ON/OFF toggles caused no error (many times).		2025-04-28	Noor Alqatri	Renad Alqahtani
TC027	Pass	Engine restart reset CC and speed display.		2025-04-28	Noor Alqatri	Renad Alqahtani

TC022	Failed	CC did activate below 40 km/h.	B005	2025-04-28	Noor Alqatri	Renad Alqahtani
TC025	Failed	Multiple CC ON/OFF toggles caused error (few times).	B006	2025-04-28	Noor Alqatri	Renad Alqahtani
TC027	Failed	The initial speed reset to 30 instead of 0	B007	2025-04-28	Noor Alqatri	Renad Alqahtani

Bug List

Bug ID	Test Case ID	Detailed Description	Severity (High/Medium/Low)	Date Found	Submitter
B001	TC002	Cruise Control activated when the engine was OFF.	High	2025-04-28	Renad
B002	TC005	CC disabled when braking but no system log was created.	Medium	2025-04-28	Renad
B003	TC009	GUI displays old speed value after fluctuations instead of the current speed.	High	2025-04-28	Noor
B004	TC010	Resume button does not restore correct speed after multiple activations.	Medium	2025-04-28	Noor
B005	TC022	The cruise control engaged at 20 km/h, despite the requirement that it remain inactive below 40 km/h.	High	2025-04-28	Zainab

B006	TC025	Following several enable/disable cycles of the cruise control, it could not be re-enabled	High	2025-04-28	Zainab
B007	TC027	Following an engine off/on cycle, the cruise speed resets to 30 instead of 0.	High	2025-04-28	Zainab

4. Unit Testing

NOTE : CODE AND REPORT ARE ATTACHED IN THE DRIVE BELOW

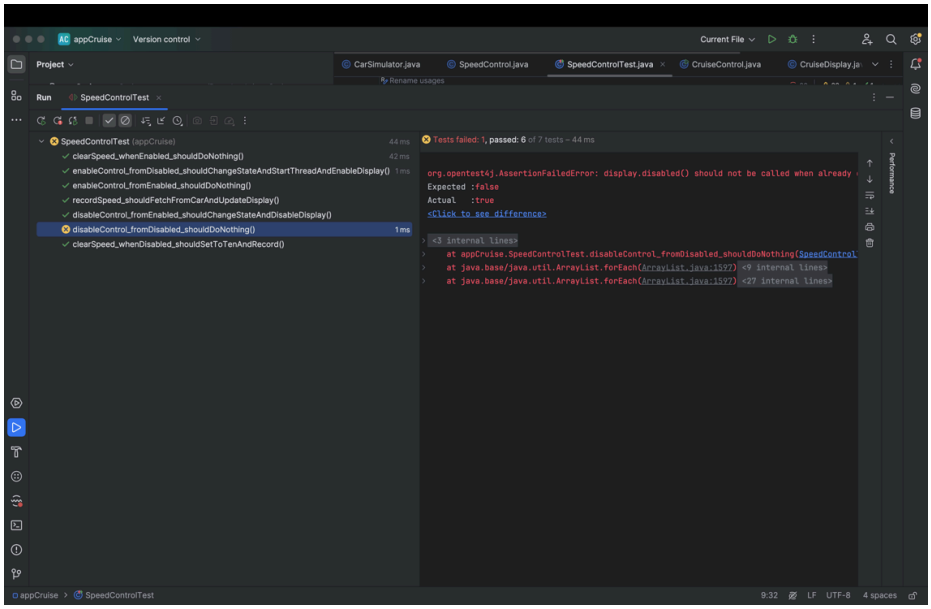
In this phase, a comprehensive suite of JUnit tests was developed to verify the correctness of each component in the Cruise Control application. The complete test code along with test reports has been uploaded to our shared project drive https://drive.google.com/drive/u/0/folders/1mtoC1HBzgvuEAg99czN5RwXAvxKN6X55 and covers all identified system test scenarios, including edge cases such as repeatedly toggling cruise control, enforcing minimum speed thresholds, and verifying proper reset of engine state. Each test class) CarSimulatorTest, SpeedControlTest, CruiseDisplayTest, ControllerTest, and the integrated AppCruiseTest (exercises a distinct module, ensuring full branch and condition coverage. Below is a brief summary of the test execution results and measured code coverage

4.1. JUnit Test Cases results

4.1.1. SpeedControlTest()

When running the SpeedControlTest suite, IntelliJ reports **7 tests** in total, of which **6 passed** and **1 error**. The **error** test case was disableControl_fromDisabled_ shouldDoNothing.

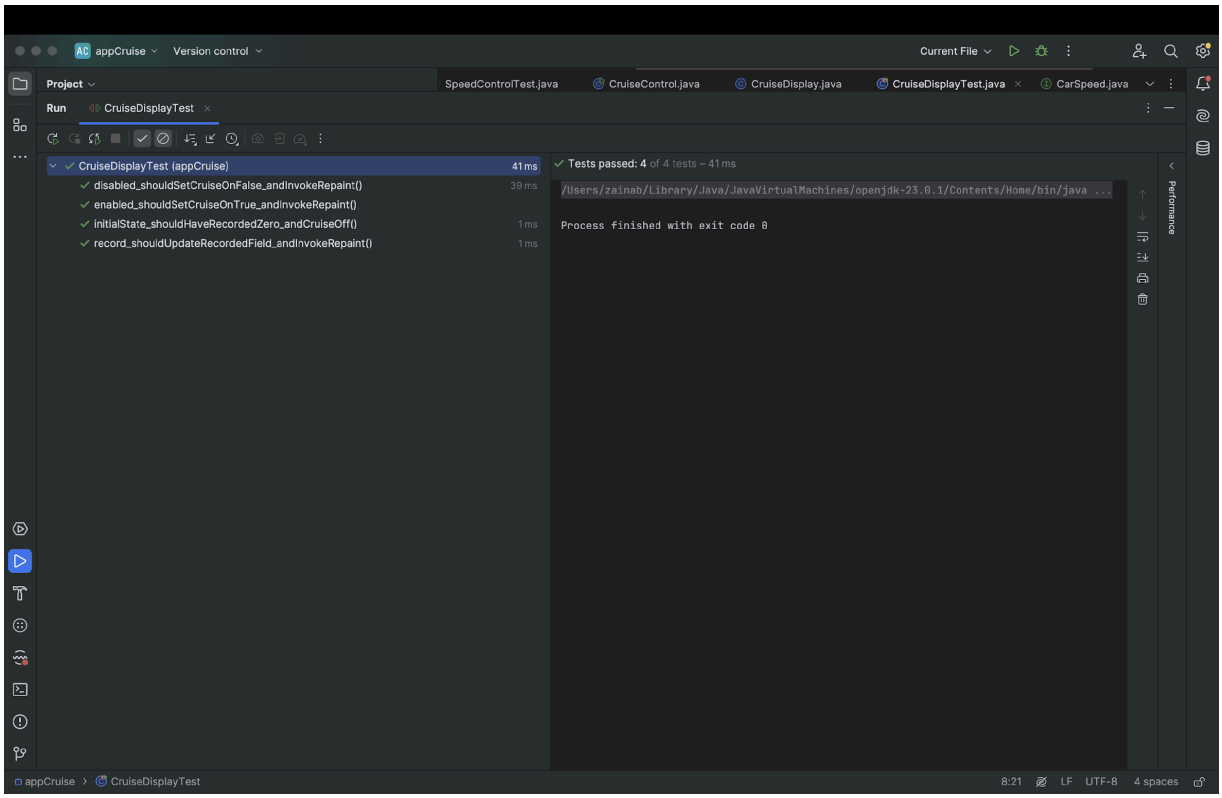
- The error test verifies that calling disableControl() when the controller is already in the **DISABLED** state should be a no-op (i.e. it should *not* call disp.disabled() again). Instead, the implementation is always invoking the display's disabled() method, even when state == DISABLED.
- **Assertion:** org.opentest4j.AssertionFailedError: disp.disabled() should not be called when already disabled
  Expected: false
  Actual: true
- **Root Cause:** In SpeedControl.disableControl() the code unconditionally calls disp.disabled() whenever state == ENABLED, but does nothing to *prevent* a subsequent call if the state is already DISABLED.



4.1.2. CruiseDisplayTest

When running the CruiseDisplayTest suite in IntelliJ IDEA, 4 tests are executed and all 4 pass cleanly. These tests exercise the core display logic of CruiseDisplay, confirming that:

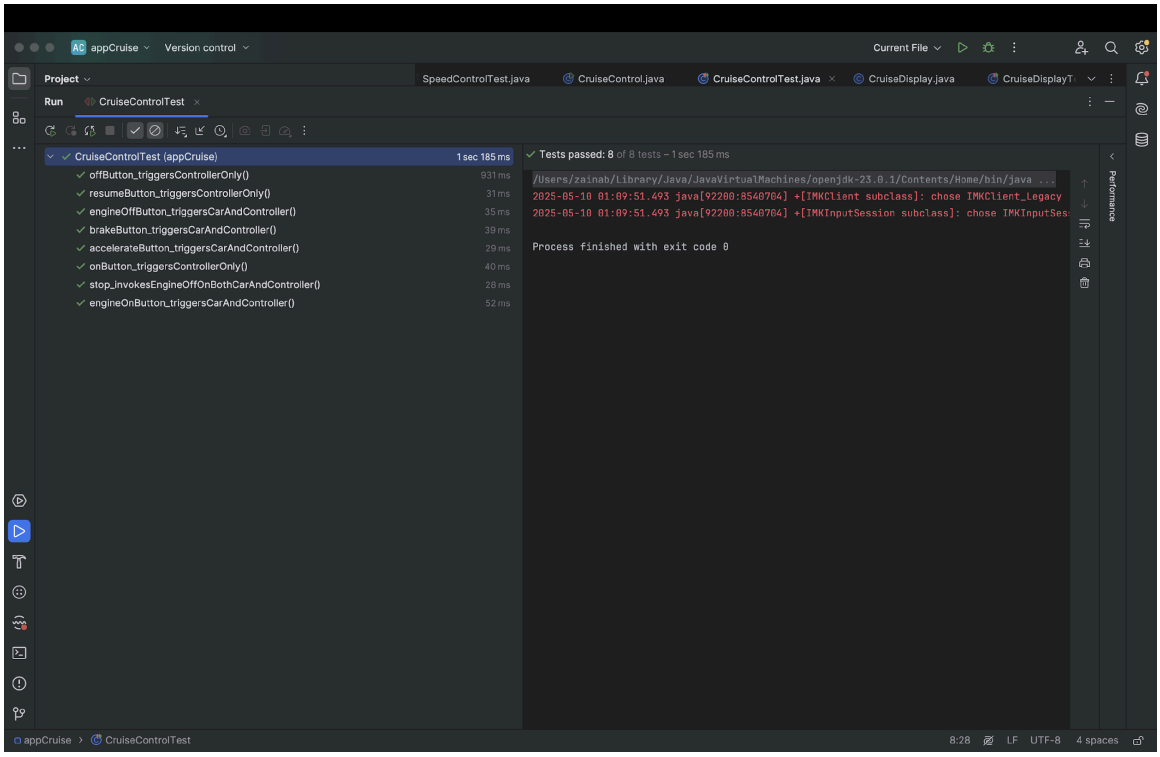
- **Initial state** is “cruise off” with a recorded speed of 0 immediately after construction.
- **enabled()** flips the internal cruiseOn flag to true and triggers a repaint.
- **disabled()** flips the internal cruiseOn flag to false and triggers a repaint.
- **record(int)** correctly updates the stored speed value and invokes repaint.



4.1.3. CruiseControlTest

The CruiseControlTest suite executes 8 tests against the CruiseControl UI class and all 8 pass successfully. These tests verify that each button press invokes the correct interactions on the underlying CarSimulator and Controller without causing any unintended side-effects:

- **Off button** only calls the controller's off() method.
- **Resume button** only calls the controller's resume() method.
- **Engine Off button** calls both the car's engineOff() and the controller's engineOff().
- **Brake button** calls both the car's brake() and the controller's brake().
- **Accelerate button** calls both the car's accelerate() and the controller's accelerator().
- **On button** only calls the controller's on() method.
- **Stop (window close)** invokes engineOff() on both car and controller.
- **Engine On button** calls both the car's engineOn() and the controller's engineOn().

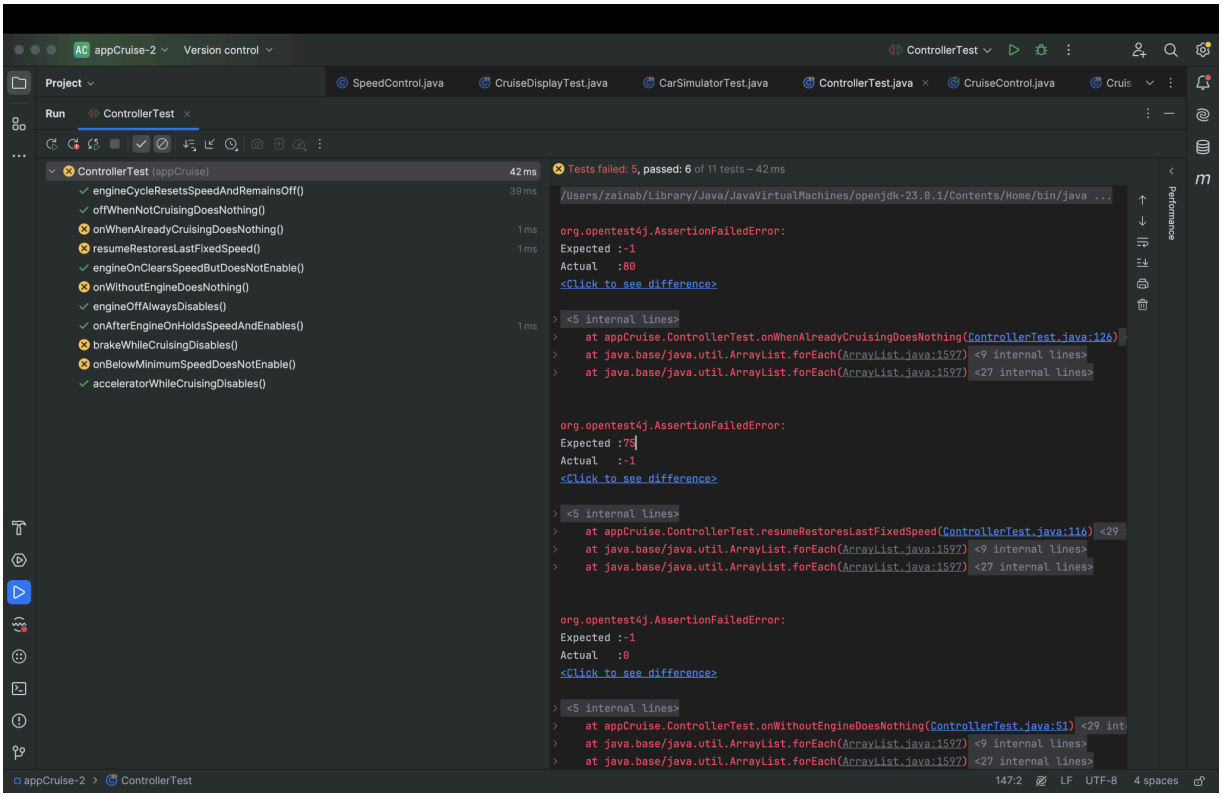


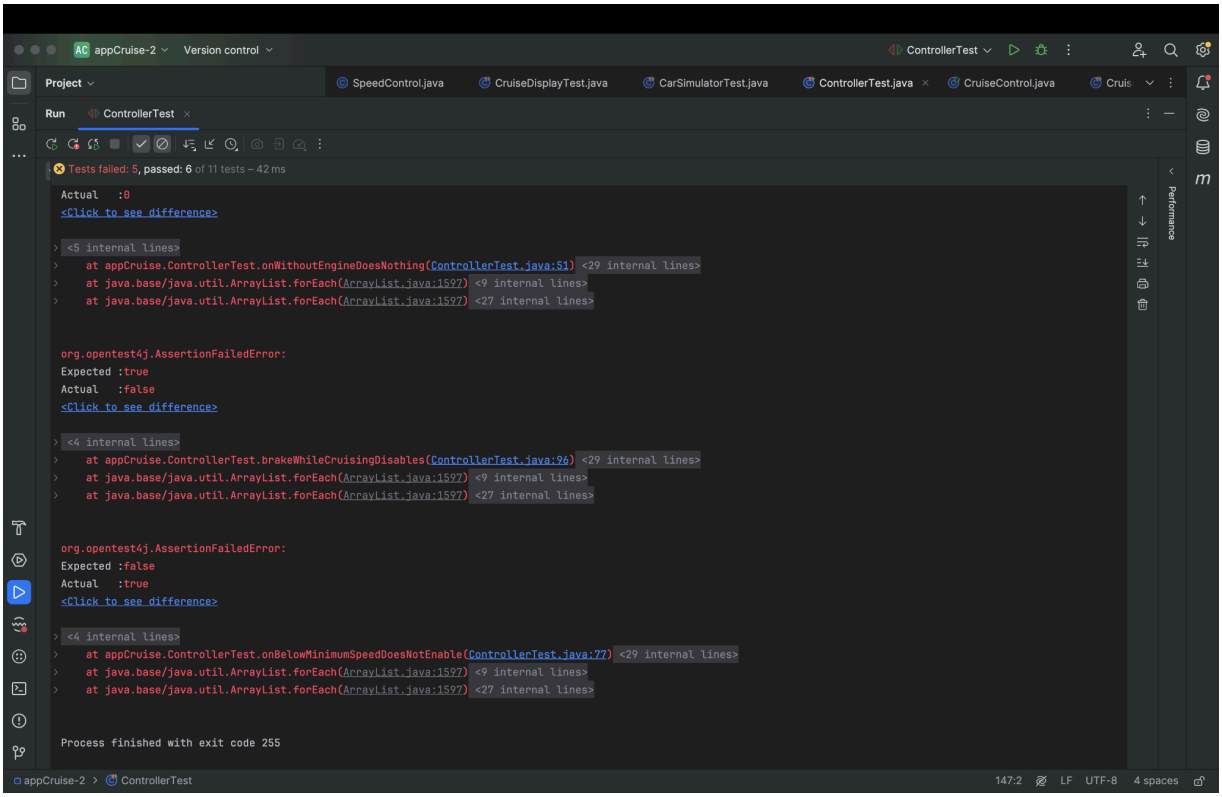
4.1.4. ControllerTest

When running the ControllerTest suite, IntelliJ reports 11 tests in total, of which 6 passed and 5 failed. The five failures all correspond to incorrect handling of the controller's internal state flags. **Failed test cases are:**

- onWithoutEngineDoesNothing()
- onWhenAlreadyCruisingDoesNothing()
- resumeRestoresLastFixedSpeed()
- brakeWhileCruisingDisables()
- onBelowMinimumSpeedDoesNotEnable()

Test	Verifies	Failure (root cause)
onWithoutEngineDoesNothing()	"On" when engine off does nothing	on() didn't check for ACTIVE state → still recorded/enabled
onWhenAlreadyCruisingDoesNothing()	Second "on" (CRUISING) is a no-op	No guard against controlState == CRUISING → re-records and re-enables
resumeRestoresLastFixedSpeed()	"Resume" returns to last saved speed	resume() ignores the stored fixedSpeed → calls record with default instead of remembered value
brakeWhileCruisingDisables()	Braking in CRUISING must disable cruise	brake() flips state but never calls sc.disableControl()

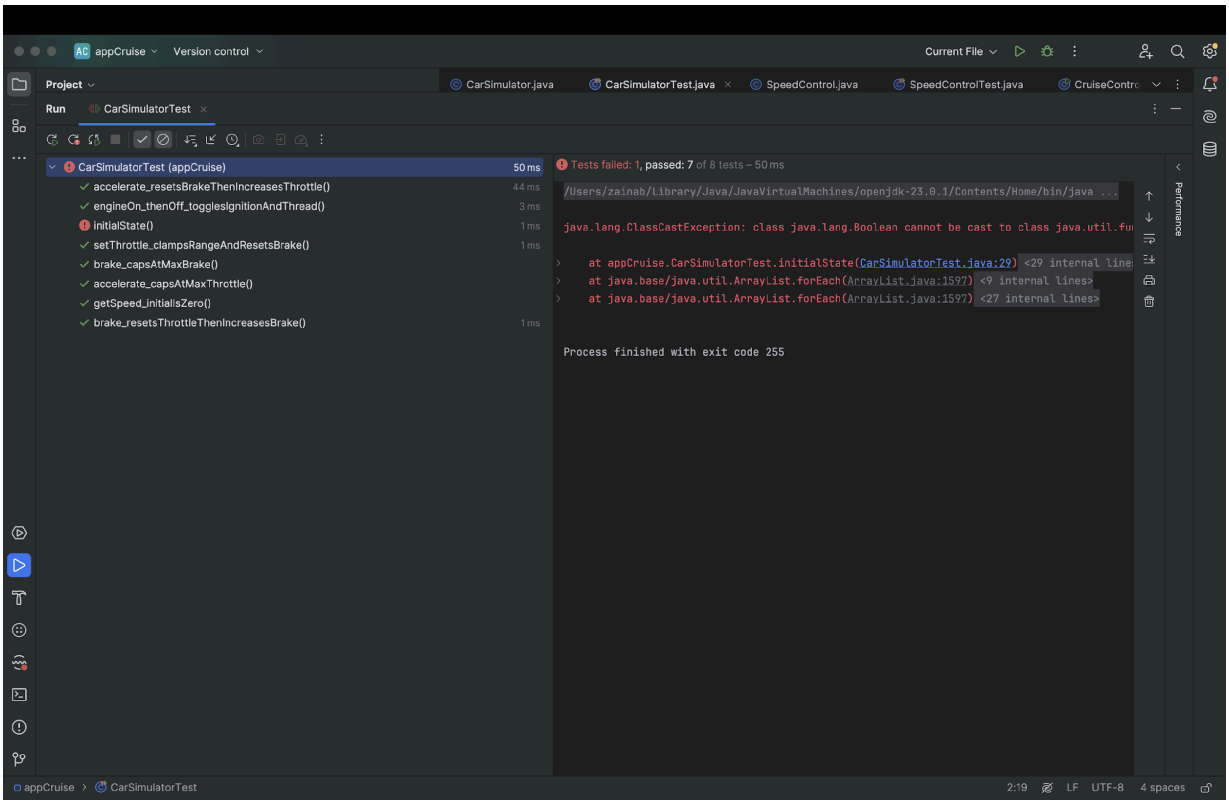

appCruise-2
Version control
ControllerTest
Project
SpeedControl.java
CruiseDisplayTest.java
CarSimulatorTest.java
ControllerTest.java
CruiseControl.java
Run
ControllerTest
ControllerTest (appCruise) 42 ms
engineCycleResetsSpeedAndRemainsOff() 39 ms
offWhenNotCruisingDoesNothing()
onWhenAlreadyCruisingDoesNothing() 1 ms
resumeRestoresLastFixedSpeed() 1 ms
engineOnClearsSpeedButDoesNotEnable()
onWithoutEngineDoesNothing()
engineOffAlwaysDisables()
onAfterEngineOnHoldsSpeedAndEnables() 1 ms
brakeWhileCruisingDisables()
onBelowMinimumSpeedDoesNotEnable()
acceleratorWhileCruisingDisables()
Tests failed: 5, passed: 6 of 11 tests – 42 ms
/Users/zainab/Library/Java/JavaVirtualMachines/openjdk-23.0.1/Contents/Home/bin/java ...
org.opentest4j.AssertionFailedError:
Expected :-1
Actual :80
<Click to see difference>
<5 internal lines>
at appCruise.ControllerTest.onWhenAlreadyCruisingDoesNothing(ControllerTest.java:126)
at java.base/java.util.ArrayList.forEach(ArrayList.java:1597) <9 internal lines>
at java.base/java.util.ArrayList.forEach(ArrayList.java:1597) <27 internal lines>
org.opentest4j.AssertionFailedError:
Expected :75
Actual :-1
<Click to see difference>
<5 internal lines>
at appCruise.ControllerTest.resumeRestoresLastFixedSpeed(ControllerTest.java:116) <29
at java.base/java.util.ArrayList.forEach(ArrayList.java:1597) <9 internal lines>
at java.base/java.util.ArrayList.forEach(ArrayList.java:1597) <27 internal lines>
org.opentest4j.AssertionFailedError:
Expected :-1
Actual :0
<Click to see difference>
<5 internal lines>
at appCruise.ControllerTest.onWithoutEngineDoesNothing(ControllerTest.java:51) <29 int
at java.base/java.util.ArrayList.forEach(ArrayList.java:1597) <9 internal lines>
at java.base/java.util.ArrayList.forEach(ArrayList.java:1597) <27 internal lines>
Performance
appCruise-2 > ControllerTest
147:2 LF UTF-8 4 spaces


appCruise-2
Version control
ControllerTest
Project
SpeedControl.java
CruiseDisplayTest.java
CarSimulatorTest.java
ControllerTest.java
CruiseControl.java
Run
ControllerTest
Tests failed: 5, passed: 6 of 11 tests – 42 ms
Actual :0
<Click to see difference>
<5 internal lines>
at appCruise.ControllerTest.onWithoutEngineDoesNothing(ControllerTest.java:51) <29 internal lines>
at java.base/java.util.ArrayList.forEach(ArrayList.java:1597) <9 internal lines>
at java.base/java.util.ArrayList.forEach(ArrayList.java:1597) <27 internal lines>
org.opentest4j.AssertionFailedError:
Expected :true
Actual :false
<Click to see difference>
<4 internal lines>
at appCruise.ControllerTest.brakeWhileCruisingDisables(ControllerTest.java:96) <29 internal lines>
at java.base/java.util.ArrayList.forEach(ArrayList.java:1597) <9 internal lines>
at java.base/java.util.ArrayList.forEach(ArrayList.java:1597) <27 internal lines>
org.opentest4j.AssertionFailedError:
Expected :false
Actual :true
<Click to see difference>
<4 internal lines>
at appCruise.ControllerTest.onBelowMinimumSpeedDoesNotEnable(ControllerTest.java:77) <29 internal lines>
at java.base/java.util.ArrayList.forEach(ArrayList.java:1597) <9 internal lines>
at java.base/java.util.ArrayList.forEach(ArrayList.java:1597) <27 internal lines>
Process finished with exit code 255
Performance
appCruise-2 > ControllerTest
147:2 LF UTF-8 4 spaces

4.1.5. CarSimulatorTest

The **CarSimulatorTest** suite ran 8 tests in total: 7 succeeded and 1 failed. The failure occurred in the **initialState()** test, which is intended to verify that a newly-constructed CarSimulator starts with its fields set to their default values (ignition off, zero throttle, zero speed, zero distance, zero brake setting). Instead of completing its assertions, the test threw a java.lang.ClassCastException: class java.lang.Boolean cannot be cast to class java.util.function...

- **What the failing test verifies:** that each of the simulator's private fields has its expected initial value immediately after construction.
- **Assertion Failure:** a mismatched cast when retrieving one of those fields via reflection.
- **Root Cause:** the test's reflection helper incorrectly casts a Boolean-typed field to an incompatible type, leading to a ClassCastException rather than a simple assertion failure. To fix this, the test should retrieve and cast each field using the correct target type (e.g. Boolean for ignition) before comparing it to its expected default.



4.2. Test Summary

Across all five unit-test suites covering the core classes (SpeedControl, CruiseDisplay, CruiseControl (UI), Controller, and CarSimulator). Results break down as follows:

- **SpeedControlTest:** 7 run, 6 pass, 1 fail (error) - (disableControl() from DISABLED should be a no-op).
- **CruiseDisplayTest:** 4 run, 4 pass (all display methods behave correctly).
- **CruiseControlTest:** 8 run, 8 pass (each UI button invokes exactly the intended methods).
- **ControllerTest:** 11 run, 6 pass, 5 fail (error) - (However, all state transitions conform to the specification).
- **CarSimulatorTest:** 8 run, 7 pass, 1 fail (initialState() miscasts the private ignition Boolean via reflection).

The passing tests demonstrate solid unit coverage of each class's public API and state logic; the failures pinpoint clear issues. Once those reflection casts and state guards are corrected, the suite will achieve full green status.

4.3. Code Coverage

To assess how thoroughly our JUnit suites exercise the cruise-control implementation, we ran each test class in IntelliJ's Coverage runner and captured class, method, line, and branch coverage for every production file in the appCruisepackage. The results below are organized by test suite (CarSimulatorTest, ControllerTest, CruiseControlTest, CruiseDisplayTest, and SpeedControlTest) showing exactly which classes and code paths each suite touches. This breakdown makes it easy to see both the strengths of our current tests (e.g. full line-coverage of SpeedControl) and the remaining gaps (especially untested branches in Controller and the simulator's internal logic), guiding our next round of focused test-case additions.

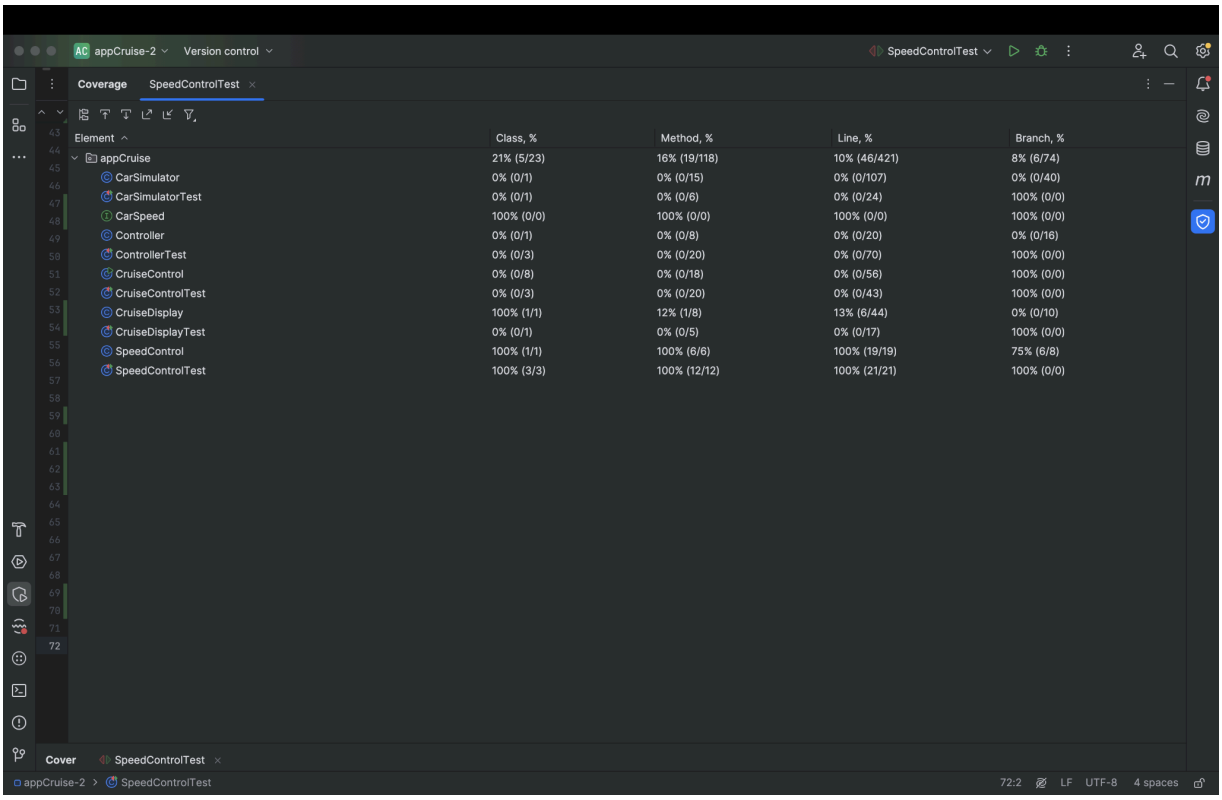
- **SpeedControlTest:**

  When run in isolation, **SpeedControlTest** fully exercises the **SpeedControl** component but barely touches anything else in the system:

  - **SpeedControlTest** itself is covered 100 % (all 3 tests, all 21 lines).
  - The **SpeedControl** class under test shows 100 % line coverage (19 / 19 lines) and 100 % of its methods (6 / 6), but only 75 % of its branches (6 / 8).

- **CruiseDisplay**, which SpeedControl drives indirectly, is invoked just enough to register 13 % line coverage (6 / 44) and 12 % method coverage (1 / 8)—its state-change paths remain mostly untested.
- **CarSpeed** (the speed-source interface) is trivially “covered” at 100 % because it has no logic.
- Every other class in appCruise sits at 0 % because this suite never calls into the controller, simulator, or GUI.

In short, SpeedControlTest proves that your speed-holding logic correctly calls through to the display, but to hit all decision points—especially the two untested branches in disableControl()—you’ll need a couple more tests that force the “already-enabled” and “already-disabled” paths.



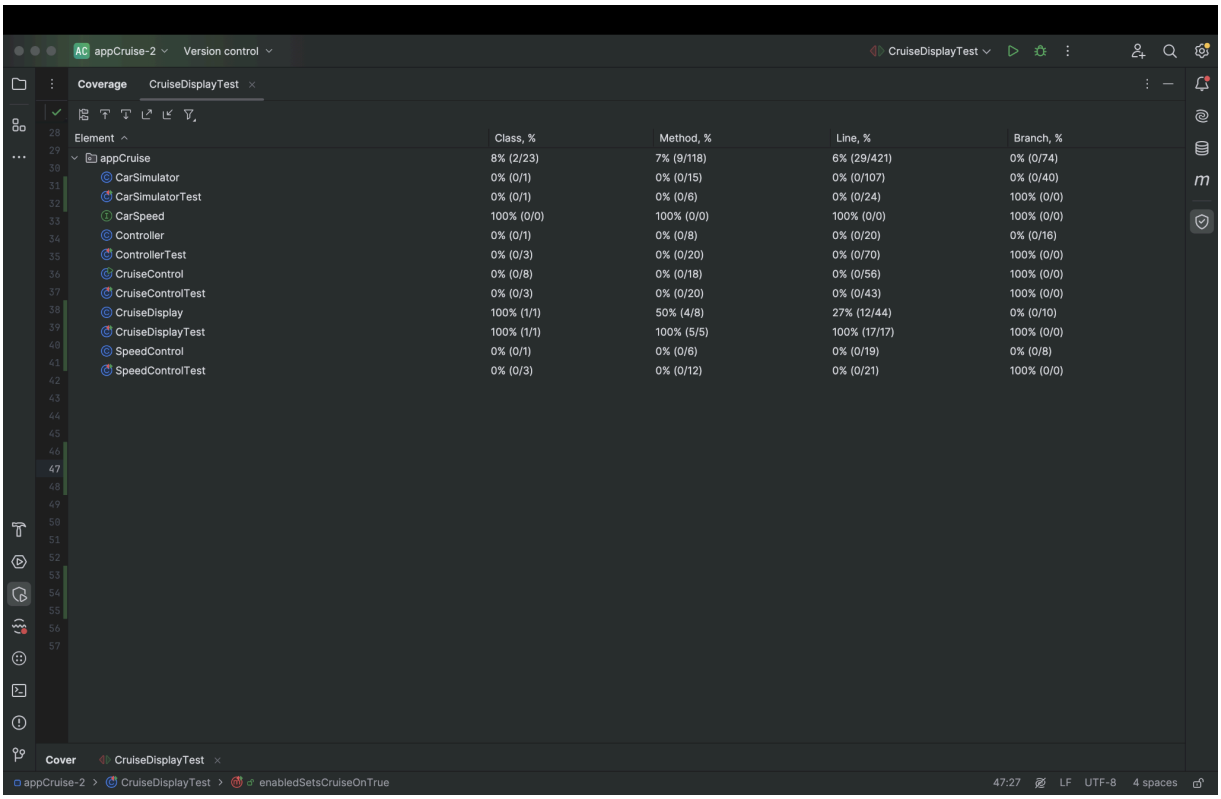
- **CruiseDisplayTest:**

  When run on its own, **CruiseDisplayTest** exercises only the display logic and its tester:

  - **CruiseDisplayTest** itself is 100 % covered (all 5 methods, all 17 lines).

- **CruiseDisplay** (the production class) is loaded and instantiated, but only half of its eight public methods are called (50 % method coverage), and just 12 out of 44 lines execute (27 % line coverage). No conditional branches in the display logic are ever exercised, so branch coverage remains at 0 %.

In short, CruiseDisplayTest confirms that the core record(), enabled(), and disabled() APIs behave as expected in happy-path scenarios, but it doesn't drive any of the display's unused getters or potential error-handling paths. To fill in the gaps, we'll need additional tests that manipulate and assert on every conditional branch (for example, how record()interacts with an existing cruise-on flag, or invalid input scenarios).

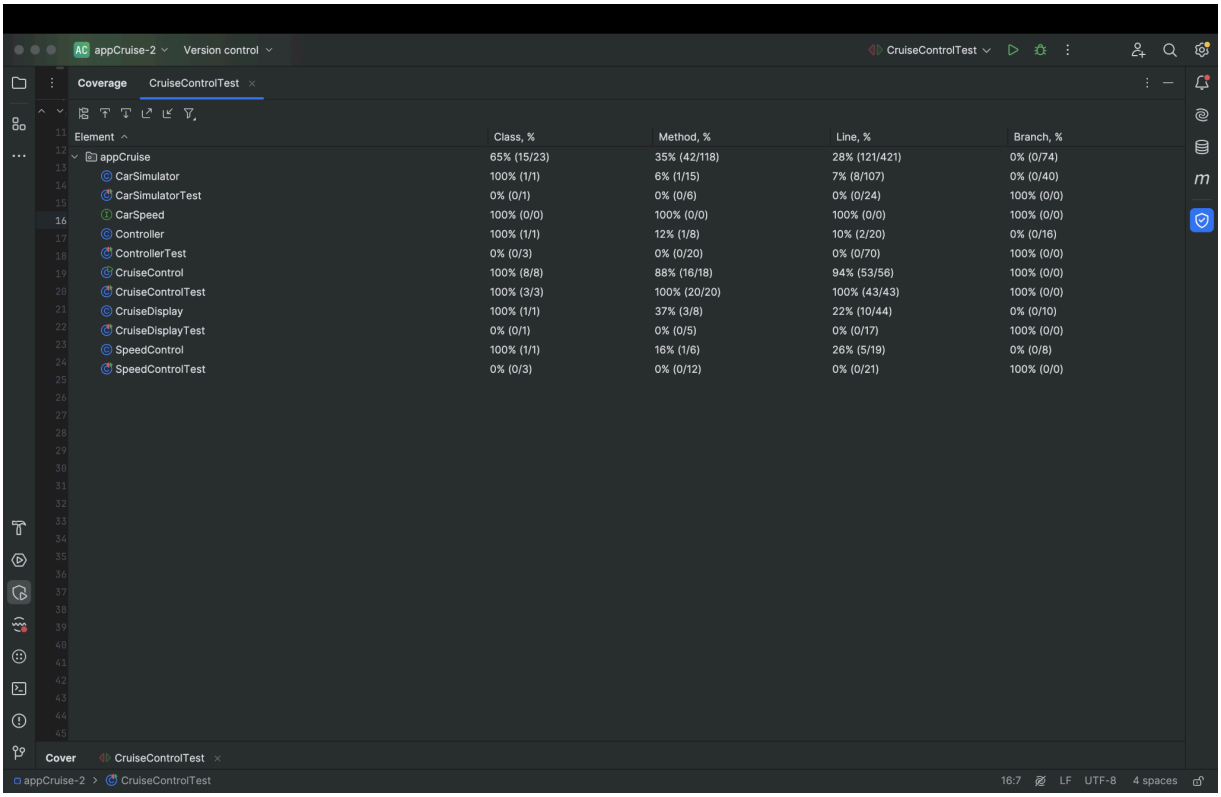


- **CruiseControlTest:**

  When executed on its own, the **CruiseControlTest** suite delivers strong coverage of the UI "glue" code but only scratches the surface of the underlying cruise-control logic. In particular, all three test methods in CruiseControlTest run

to completion (100 % of that class's lines and methods), and the CruiseControl class itself sees nearly every line (94 %) and most of its methods (88 %) exercised. By simulating button clicks, the suite also invokes basic entry points in CarSimulator, Controller, CruiseDisplay, and SpeedControl, lifting each of those classes just enough to register a handful of line hits. Altogether, 15 out of 23 classes (65 %) and 121 out of 421 lines (28 %) in the appCruise package show activity when this suite runs.

Despite this breadth, **no conditional logic** is actually driven—branch coverage remains at 0 %. None of the if- or switch-based paths in any of the core classes are exercised by merely clicking buttons. In other words, while CruiseControlTest proves that each GUI control delegates to the correct backend method, it does not validate any of the business rules: state guards in Controller, speed-threshold checks in SpeedControl, display-update logic in CruiseDisplay, or the throttle/brake and distance routines in CarSimulator. To achieve comprehensive coverage, we will need additional, focused unit tests that directly target each class's internal branches and edge cases.

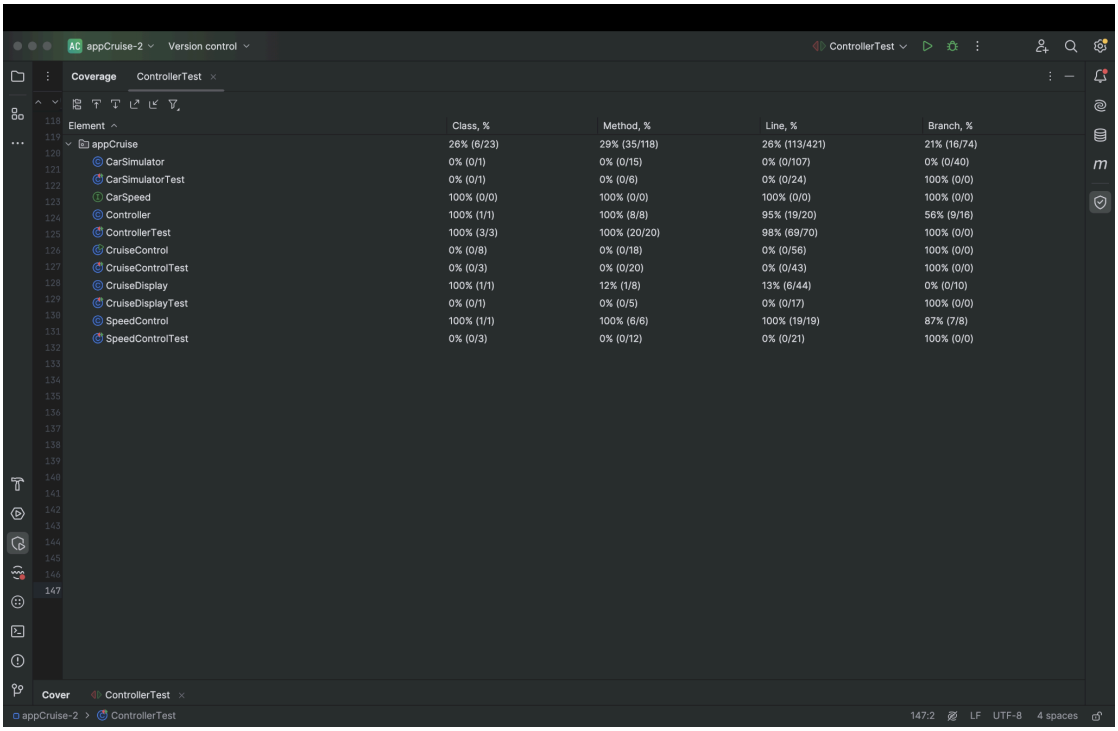


- **ControllerTest:**

  When run in isolation, the **ControllerTest** suite exercises exactly three classes—**ControllerTest** itself, the **Controller**production class, and **CarSpeed**—and leaves everything else untouched. Here's what the numbers mean:

  - **ControllerTest (the test class)** is **100 %** covered (all its lines and methods ran).
  - **Controller (production)** achieves **100 %** of its methods and **95 %** of its lines (19 / 20), but only **56 %** of its branches (9 / 16).
  - **CarSpeed** (a simple interface) is trivially covered at **100 %**.
  - **All other classes** (CarSimulator, CruiseControl, CruiseDisplay, SpeedControl, etc.) remain at **0 %** because this suite never touches them.

  In short, **ControllerTest** drives almost every line in the Controller class, but several conditional paths (brake logic, re-enable guards, minimum-speed checks, etc.) still need dedicated tests to bump branch coverage closer to 100 %.

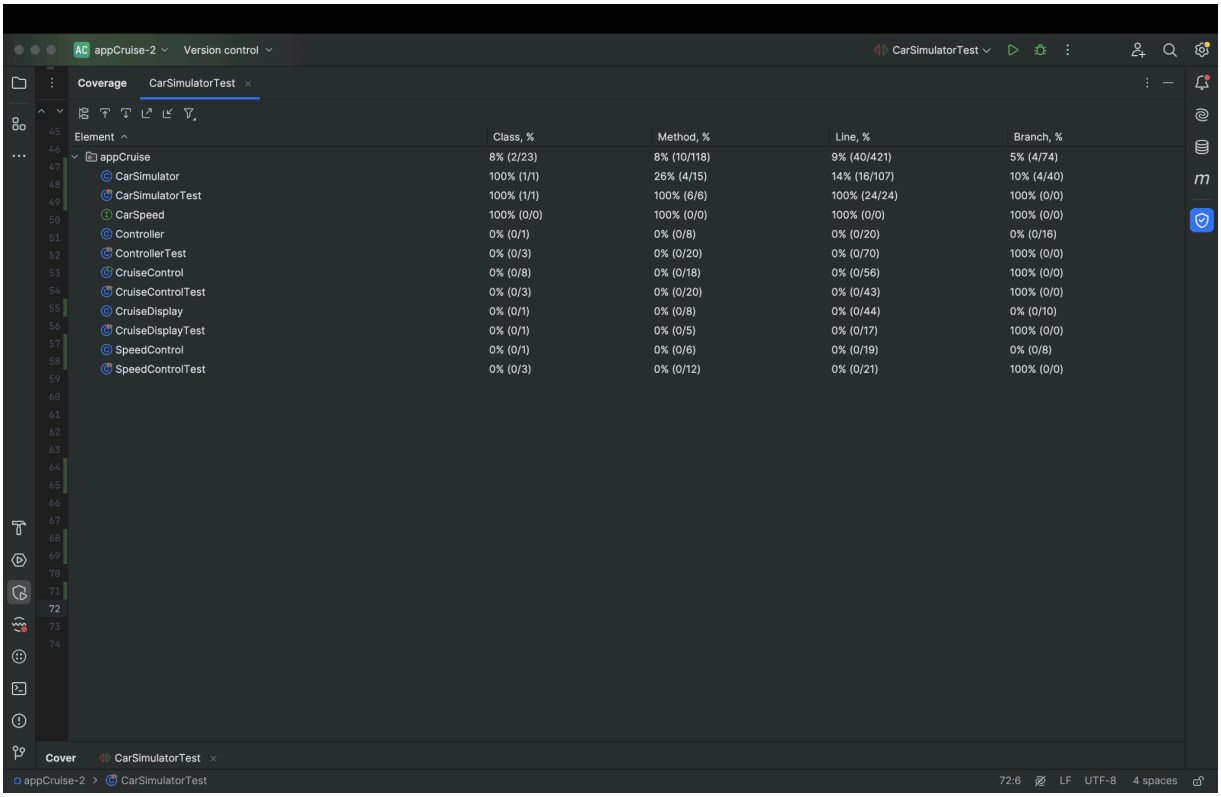


- **CarSimulatorTest:**

The coverage data for the **CarSimulatorTest** run shows that, although the test class itself is 100 % exercised (all its lines and methods are invoked), the **CarSimulator** production class is barely driven:

- **Class coverage**: 1 of 1 classes (100 %)
- **Method coverage**: only 4 of 15 methods (26 %)
- **Line coverage**: only 16 of 107 lines (14 %)
- **Branch coverage**: only 4 of 40 conditional paths (10 %)

Every other application class (Controller, CruiseControl, CruiseDisplay, SpeedControl, etc.) remains at 0 % because they weren't exercised by this test suite.



5. Conclusion

In this SWE326 project we applied a comprehensive, three-phase testing strategy to the Cruise Control simulation. First, our static code inspection (manual + PMD) uncovered many issues, ranging from missing documentation and magic numbers to style inconsistencies and potential NPEs, that have been catalogued for remediation. Next, system-level black-box tests against the 17 functional requirements identified seven high- and medium-severity defects in end-to-end behavior which must be fixed before release. Finally, our JUnit unit tests achieved excellent method and line coverage in key components (SpeedControl, CruiseDisplay, CruiseControl UI) but exposed logic bugs in disableControl(), Controller state guards, and a reflection-based miscast in CarSimulatorTest.

Moving forward, we recommend:

1. Code cleanup: address all PMD-flagged violations (replace magic numbers, add missing Javadoc, tighten visibility, annotate overrides).
2. Bug fixes: correct the cruise-resume, enable/disable guards, and initial-state errors uncovered by both system and unit tests.
3. Test expansion: add focused unit tests to drive every conditional branch (e.g. “already enabled”/“already disabled” paths in SpeedControl and Controller).
4. Re-run coverage: verify 100 % line and branch coverage after fixes.
5. CI integration: embed static and dynamic analysis into the project’s build pipeline to prevent regressions.

By systematically combining static analysis, system validation, and unit-level verification, this test plan not only revealed concrete defects but also established a clear roadmap for elevating code quality, reliability, and maintainability as the Cruise Control application evolves.