CITS5501/CITS3501 Project 2025 phase 2

Version:	0.1
Date:	12 September 2025

Project Admin

For project administrative matters and rules, refer to the phase 1 project specification.

Deliverables (20 marks total)

There are three deliverables for phase 2 of the CITS5501 project:

- A text file containing an EBNF grammar. The grammar must compile using the BNF Playground website.
- A report (submitted as PDF), created from a Markdown file.
- Your Java source code with Javadoc for a JUnit test class, `SegmentSubcommandTest.java`

The source for all three deliverables should be checked into version control in your GitHub repository. We will use the submitted version for marking, but your facilitator may inspect the GitHub repository for additional information on your group's collaboration and quality assurance processes.

Task 4 – Tachi commands

Task 4.1 – Command grammar (3 marks)

Create a BNF or EBNF grammar that will parse (or generate) valid Tachi commands. Your grammar should be a plain text file (not MS Word, Markdown, or any other format).

Requirements for the grammar are as follows:

1. The syntax for Tachi commands is described in the Phase 1 specification, under "Tachi command description".
2. Your grammar should use the notation accepted by the BNF Playground website.
3. You should not attempt to validate semantic properties of commands.
4. The grammar should include a non-terminal, `<tachi_command>`, to be used as the start symbol for the grammar.
5. The grammar SHOULD specify any whitespace that needs to appear between other symbols in the command. But as a simplifying assumption: if symbols need to be separated by horizontal whitespace (tabs and spaces), you can and should assume that a single space character is sufficient.
6. Your grammar is a code artifact, and should be clear and well-presented, like any code.

Tips:

1. Make sure you read the BNF playground website thoroughly – it includes a description of the dialect of EBNF it uses, under "Grammar Help".
2. Read through the CITS5501 rubric guide on "Long answer" questions requiring code. Make sure you understand how those guidelines apply to EBNF grammars. (Your facilitator may be able to provide some assistance.) Ensure you know how to comment your grammar, structure it so it is clear to a reader, and make use of appropriate use of EBNF features which would make your grammar clearer or more concise.

Submission: You will submit your grammar as a text file in Moodle. You should also maintain it as a file within your project repository, so markers can see how it has been revised and reviewed.

Task 4.2 – Command grammar quality (3 marks)

Briefly explain (word limit: 1000 words) how you ensured your grammar from Task 4.1 is:

- correct;
- clear; and
- succinct.

What high-level strategies/approaches did you use? What concrete techniques did you apply? Did you adopt any coverage criteria, and if so, which ones and why? You should link to at least one point in your project repository where the grammar has been reviewed for quality.

You should discuss the degree to which your quality assurance techniques can be *automated* – could they be run automatically when changes are made to the code, or do they require manual steps? If the latter, discuss whether this would be appropriate for a real-world project, and what improvements could be made to them.

Submission: This should be a section in your report, and be given the heading "Task 4.2".

Task 5 – Java testing

For Task 5, you are provided with a Java library (in the form of several JAR files) containing compiled code and documentation for classes that process Tachi commands. The JAR files can be downloaded here. You are required to discuss approaches to testing the library and to implement several tests.

The JAR files are intended to be used by adding them to a Java project. (How this is done depends on your exact project setup and/or IDE, but if using VS Code, it should be sufficient to add them to a `lib` directory that is a sibling of any `test` directory you create to contain your JUnit test class.)

Task 5.1 – `ShopFlightFareCommand` class analysis (3 marks)

Describe in detail the preconditions and postconditions of the constructor for the `ShopFlightFareCommand` class, justifying your answer (word limit: 500 words).

Submission: This should be a section in your report, and be given the heading "Task 5.1".

Task 5.2 – Input Space Partitioning (4 marks)

Apply Input Space Partitioning (ISP) to two methods or constructors in the Java library. One of these should be the `SegmentSubcommand` constructor; the other can be a method or constructor of your choice.

Explain the steps you took in applying ISP (word limit: 1000 words). Your explanation should include:

- a level of coverage you think your test cases should achieve, and why you chose that level; and
- the characteristics and partitions you derived.

Your descriptions of characteristics and partitions can be fairly brief (bullet points, tables, etc. are sufficient), but you should pick three characteristics to discuss in more detail. Explain how you arrived at those characteristics and partitions, and what the rationale for them is.

You need only refine your characteristics and partitions into *four* test cases. At least three test cases should be for the `SegmentSubcommand` class. You should provide descriptions of those test cases. The test case descriptions should include:

1. A test ID, for easy reference later.
2. A description of all fixtures, test values and expected values.

If you need to make any reasonable assumptions in order to derive your characteristics or test cases, state clearly what they are.

Submission: This should be a section in your report, and be given the heading "Task 5.2".

Task 5.3 – Test implementation (4 marks)

Write a test class using JUnit 5 called `SegmentSubcommandTest`. The test class should contain JUnit test methods for the `SegmentSubcommand` class, which implement the 3–4 test cases you described in Task 5.2.

Your test cases should follow appropriate best practices for JUnit tests, including:

- Your tests must properly "Arrange, Act and Assert" your test case. Tests that (for instance) contain no assertions will not be awarded marks.
- Your test cases must be ones described in Task 5.2. For each test, include Javadoc documentation providing the test ID from Task 5.2 that it corresponds to.

Your code should compile using a recent version of the Java Development Kit and the JUnit libraries. (You may assume that the current Long-Term Support release of the Java Development Kit, JDK 21, is acceptable, as is the current version of the JUnit Jupiter library, 5.13.4.)

Submission: You should submit your `SegmentSubcommandTest` Java code via Moodle. Refer to the Moodle site for further submission details.

Task 5.4 – Test quality (3 marks)

Explain how you assessed the quality of the tests from Task 5.3 (word limit: 500 words).

You should include links to your GitHub repository (to specific pull requests, issues, commits, etc.) which demonstrate the tests being reviewed for quality.

Submission: This should be a section in your report, and be given the heading "Task 5.4".