# EECS2030 (Sections B & G) Fall 2025
# Guide to Programming Test 2

| | |
|---|---|
| When: Wednesday, November 5 (during your enrolled lab session) | |
| Duration: 80 minutes | |
| Section G (LAB 01/02/03) | Section B (LAB 01/02/03) |
| 13:00 to 14:20 | 14:30 to 15:50 |
| WSC 106/108 | WSC 106/108 |
| Registered Accommodation (from both sections) | |
| Starts at 13:00 | |
| WSC 105 | |

Chen-Wei Wang

Last Update: October 29

## 1 Policies

- This test ***assumes proficiency*** with what are covered in:
  - Lab 2

    The actual test will **not** be as long as Lab2.
  - Lab 3
  - Exceptions (in both lecture and Lab2)
  - Object Equality (in both lecture and Lab3)
  - TDD via JUnit

  Being just able to (more or less) comprehend the lab solutions and example code given to you is a promising first step, but it is still **far** from reaching the ***passing criterion***.

  To pass or to perform well, you need to be capable of coding up solutions in Eclipse, **from scratch**, to problems at a similar level of difficulty as the lab and example problems.
- As Lab2 covers basic OOP, it is ***beneficial*** for you to also review/practice:
  - Lab0 Part 1
  - Lab0 Part 2
  - Lab1
  - Review Lecture on Classes and Objects Link
  - The required readings:
    - Inferring Classes/Methods from JUnit Tests Link
    - Declaring and Manipulating Reference-Typed, Multi-Valued Attributes Link
- You are fully responsible **for properly importing the starter project into Eclipse, exporting the completed project with the specified name, and submitting the required Eclipse project archive (.zip) file in time** (i.e., by the time when the lab machines are rebooted).

TAs and instructors may attempt to help you on the import/export/submission processes upon your requests. However, should any failure occur on submitting the wrong file(s) or on submitting the required file(s) in time, TAs and instructors will **not** be assumed the responsibility (as these are the **same** processes as those required by the assignments): you are **solely responsible** for the consequence (e.g., receiving a zero for the test).

- Once the test starts, each of the following actions is considered as an ***academic misconduct*** (and you will only be warned once before we report it to the **Lassonde office of academic integrity**):
  - Look at the screen of others.
  - Talk to anyone taking the test. Even if people around you need help and ask you directly, you are still **not** supposed to respond to them. Instead, just let the TAs or instructors handle.
- **Lab machines will be rebooted back to the Linux mode promptly at the specified end time of your test session: any work/file(s) not submitted in time will not be accepted.**
- There will be multiple test sessions running on the same day. Before all sessions end, it is considered **a breach of academic integrity** if you communicate about the content of your test with others.
- This programming test is in-person and **strictly** individual: plagiarism check may be performed and suspicious submissions will be reported to Lassonde for **a breach of academic integrity**.
- This programming test will account for **8%** of your course grade.
- This test is **purely** a programming test, assessing if you can write **valid** Java programs free of syntax, type, and logical errors.
- **Structure of the Test**:
  - You **must** show up in the room assigned to your **enrolled session** (LAB 01, LAB 02, or LAB 03).
  - At the start time, all lab machines will be rebooted to the "lab-test mode" (where there is **no** network connection and you are expected to use the **Eclipse tool only**).
  - During the test, you will be expected to:
    * Launch Eclipse on a designated workspace.
    * Download and import a starter project archive file (.zip file).
    * Develop Java classes in the `model` package, based on the given starter JUnit tests.
    * **Penalty will be applied if the Java classes you develop in the `model` package do not compile with the original, given starter tests. To avoid such penalty related to compilation errors, your model classes should at least properly declare all classes and methods (and supply some default implementation such as `return null;` whenever necessary) as required by the given starter JUnit tests.**
    * **When your developed code compiles but contains logical errors, you are expected to find them, using breakpoints and the debugger, and fix them on your own.**
    * You are **solely responsible** for:
      · **leaving enough time (≈ 3 minutes) to export the completed Java project and upload/submit the archive (.zip) file to WebSubmit**; and
      · **submitting the right project archive file for grading**.
      A **common mistake** is that one just uploads the initial starter project for grading, in which case the TAs cannot do anything about it.
- **Submission for Grading**:
  - Like your labs, submission (of an Eclipse Java archive `.zip` file) for this programming test must be through the WebSubmit link (which will be provided during the test).
  - It is your sole responsibility for making sure that the correct version of project archive file is submitted. **After** clicking on the `submit` button on WebSubmit, you should **re-download** the archive file and ensure it is the right version to be graded. **No** excuses will be accepted.

– **Programming Requirements**

1. You are **only allowed** to use primitive arrays (e.g., `int[]`, `String[]`, `Facility[]`) for implementing classes and methods to solve problems related lists/collections. Any use of a Java library class or method is forbidden (that is, use selections and loops to build your solution from scratch instead):
   - Some examples of ***forbidden*** classes/methods: `Arrays` class (e.g., `Arrays.copyOf`), `System` class (e.g., `System.arrayCopy`), `ArrayList` class, `String` class (e.g., `substring`), `Math` class.
   - The use of some library classes does not require an `import` statement, but these classes are ***also forbidden*** to be used.
   - Here are the exceptions (library methods which you **are allowed** to use if needed):
     * `String` class (`equals`, `format`)

   You will receive a ***30% penalty*** if this requirement is violated.
2. If your submitted project (including the initial starter test file) contains any compilation errors (i.e., syntax errors or type errors), TAs will attempt to fix them (if they are quick to fix); once the revised submission is graded, your submission will receive a ***30% penalty*** on the resulting marks (e.g., if the revised submission received 50 marks, then the final marks would be 30 marks).

   A common compilation error is that some of the given **starter tests** do **not** compile because the expected classes and/or methods are not added/implemented. To avoid this error, for those classes/methods which you cannot manage to implement, at least provide the **proper method headers** (with empty body of implementation) to make the starter tests compile. For example, say part of the starter test reads:

```
1 ...
2 A oa = new A();
3 String s = oa.m(23);
4 ...
```

Line 3 suggests that a method `m` should be implemented in class `A`. To make Line 3 compile, you should at least declare the method in class `A`:

```
public String m(int i) {
  return null;
}
```

## 2 Format

The format of this programming test will be **identical** to that of your **Lab1**: given a JUnit test class containing compilation errors begin with, derive, declare, and implement classes and methods in the `model` package. You will **not** be asked to build console applications for grading.

– The `model` package is empty (to be added classes derived from the given JUnit tests).

– The `junit_tests` package contains a collection of JUnit tests suggesting the required classes and methods.

## 3 Grading Criteria

– For this programming test, you will **also** be graded by an additional list of Junit tests (which are **still consistent** with the problem descriptions). For example: if you are given 5 tests, and there are another additional 5 tests not given, then your submission will be graded by all 10 tests).

– Your test marks will be determined **solely** based on the input-output correctness of your implemented Java classes (i.e., the number of JUnit tests passed).

– **No** partial marks will be considered by inspecting code that does **not** pass the starter/grading test cases.

# 4 How the Test Should be Tackled

- Your **expected workflow** should be:

  1. **Step 1: Eliminate compilation errors.** Declare all the required classes and methods (returning default values if necessary), so that the project contains no compilation errors (i.e., no red crosses shown on the Eclipse editor). See Steps 1.1 to 1.3 of Section 2.2 in the written notes ***Inferring Classes from JUnit Tests***.
  2. **Step 2: Pass all unit tests.** Add **private** attributes and complete the method implementations accordingly, so that executing all tests result in a ***green*** bar.

     If necessary, you are free to declare (private or public) helper methods.

- ***It is critical that you complete Step 1 first, so that you will not receive a penalty for submitting a project containing compilation errors.***

# 5 Rationales: Grading Standard & Time Constraint

The two primary learning outcome of this course are:

1. Computational thinking (for which you build through labs and assessed by written tests and the exam)
2. Being able to construct *working* software (for which you are assessed through programming tests)

**Why is the compilation penalty?** When you write an essay, if there are grammatical mistakes, it can still be interpreted by a human. Computer programs are unlike essays: when your program contains compile-time syntax or type errors, it just cannot be run, end of story. When a computer program cannot be run, its runtime behaviour is simply unknown; and this is particularly the case when your program contains if-statements and loops. Furthermore, when you land a job upon graduation, you would not expect your supervisor or colleagues to read your code that does not run, because it does not even compile, would you? True, you're still learning. But it is exactly this mind set that restricts your potential of becoming a competent programmer. This is already your third programming course. If we want to train you to be a competent programmer, NOW is the time to enforce the strict (but justifiable) standard.

**Why is the time constraint?** Working under stress is unavoidable. Your future programming interviews for jobs will expect you to do the same: given problems, program your solutions in front of a work station or a whiteboard within some (short) set time limit. More critically, after landing a job, whenever being called upon by your perspective workplace supervisor for some customer-reported bugs, most likely they need to be fixed within a short time interval. Arguably, not being able to perform well under stress can be a indication of a lack of enough practice, which is surely unpleasant at first but also suggests how you can improve your skills fundamentally.

**Why are the additional grading test cases?** All such additional tests are derived from the **same** problem descriptions that are given to you during the test, and they are meant to assess if your method implementations are able to handle all **implied** input scenarios (rather than just those specific to the starter tests given to you). **As a competent engineer, it is your sole responsibility for ensuring that the built product stamped with your name—in this case, your submitted Java classes—is correct and fit for use.** Therefore, you are advised to test your program with extra inputs by writing more JUnit tests. You can always add a new test by copying, pasting, and modifying a test give to you.

# 6 Practice Test

- A practice test (**not** to be graded) has been made available on eClass.
- This practice test is only meant as an **example** for familiarizing yourself with the **format** and **workflow** of the test: you are expected to study **all** covered materials.