EECS2030 (Section F) Fall 2022
Guide to Programming Test 1
When: 16:10 to 17:20, Tuesday, October 4

Chen-Wei Wang

1 Policies

- 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 honesty**.
- This programming test will account for **5%** 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**:
  - At 16:10, all WSC 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.
      When errors exist in your developed code, 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 ($\approx$ 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 make sure it is the right version to be graded. **No** excuses or submissions will be accepted after your attempt times out.

– **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

For this programming test, you will **also** be graded by an additional list of Junit tests (e.g., you are given 5 tests, and there are another additional five tests not given, and your submission will be graded by all 10 tests).

Therefore, it is up to you 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.

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 most important 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 write *runnable* programs (for which you are assessed through computer tests)

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.

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.

6 Coverage for the Test

- Object Oriented Programming

  Note. There will **not** be any written questions, but you may review your instructor's lecture materials to clarify the concepts.

- Lab0 (Part 1 and Part 2): Review Tutorial Series
- Lab1 (the actual test will not be as long as Lab1)
- Exceptions will **not** be covered.
- Required Written Notes:

  - ***Inferring Classes from JUnit Tests*** [Pdf]
  - ***Manipulating Multi-Valued, Reference-Typed Attributes*** [Pdf]

- The concepts about Github, remote labs, and terminal commands are **not** covered in the test.

7 Study Tips for the Test

- The actual test will require methods to be implemented with object creations, method calls, and loops (possibly embedded with selections).
- Finish the given example test (for as many times as needed) to **familiarize yourself with the workflow of the test**.
- Review examples covered in the tutorial videos, lectures, and written notes. Make modifications to the example test accordingly by adding more methods and tests. For example, some methods in the actual test will be at the similar difficulty level as the `getPointsInQuadrantI` method as explained in the required written notes.

8 Practice Test

- The starter project (`.zip` file) of a practice test is made available under the **Programming Tests** section on the ***Section F eClass site***. You can attempt this test for as many times as you wish.
- This practice test will **not** be graded.
- It is important to note that these questions are meant for familiarizing yourself with the **format** and **workflow** of the test, and they represent **only** as an example: you are expected to study **all** materials as listed in Section 6.
- The practice test contains many test methods for you to work on. Nonetheless, the actual test **will be** suitable for the given time limit.

9 Simulating the Lab Test

It is highly recommended that you simulate taking the programming test by following these steps:

Preparation

- Login into a machine under remotelabs (using your EECS account): `https://remotelab.eecs.yorku.ca/`. Choose a machine under the `ea` category.
- Launch the Firefox web browser (under Activities) and login into the Section eClass site.
- Open a copy of this test guide (so that you can click on the WebSubmit link at the end).

Start the Test

- Start a timer (say for 90 minutes).
- Download the (`PracticeTest1_Starter.zip`) file from eClass onto the Desktop.
- Launch Eclipse (under Activities) and choose the default workspace:

  `/eecs/home/??/eclipse-workspace`

  where ?? denotes your EECS account name. It is ok to select a different directory as the workspace, as long as you know how to locate it.
- **Import** the starter project to Eclipse.
- Tackle the test by implementing classes/methods into the `model` package, based on the starter tests given. You are only expected to read the starter tests, as well as comments in the same Java file.
- **Before you submit, you should make sure that there is no compilation error in any of the files (including the original starter JUnit test file given to you) in the project.**

Submission

- By the end of the time limit, **export** your developed project (entirely) as an archive file (with the designated name different from the starter project): `PracticeTest1.zip`. Save the archive file to the Desktop. Only properly exported `.zip` archive file will be graded. It is assumed that you already gained familiarity with importing and exporting projects in Eclipse.
- **It is a recommended practice that you export, upload, and submit intermediate versions of your developed project (e.g., every 20 to 30 minutes).**
- Upload the `PracticeTest1.zip` file to the WebSubmit link for grading:

  `https://webapp.eecs.yorku.ca/submit/?acadyear=2022-23&term=F&course=2030F&assignment=PT1`

  Be careful about **not** uploading the initial downloaded starter project file `PracticeTest1_Starter.zip` for grading.