EECS2030 (Section F) Fall 2022
Guide to Programming Test 2
When: 16:10 to 17:20, Tuesday, November 8

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 **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**:
  - 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** and **Lab2**: 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

- Lab2
- Exceptions
- TDD via JUnit

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

- Object Equality, Aggregations, Compositions, and Lab3 will **not** be covered.

7 Study Tips

- The actual test will be a variant of your Lab2, simplified to suit for the given time limit.
- The best preparation you can do is to make sure that you understand how to solve Lab2:
  - Run the grading tests of Lab2 (see eClass) to see what mistakes you made in your submission.
  - Make use of the solution walkthrough video.
- If time permits, re-doing Lab1 can also help.
- Discuss your ProgTest1 with Jackie if you find the need.

8 Simulating the Lab Test

You are advised to re-do Lab2 in the remote labs. See the instructions in your Programming Test 1's guide to see how to simulate a programming test.