EECS1022 (M,N,O) Winter 2021
Guide to Programming Test 2
When: Wednesday (Mar 3) & Thursday (Mar 4)

Chen-Wei Wang

In this programming test you will be required to code Java **utility methods** in Eclipse. A class declared with utility methods (i.e., ones starting with `public static`) will be given to you. You **only** need to worry about writing lines of code that serve as the body of implementation of each of these methods. You will **not** be asked to build console applications for grading.

- This programming test is **strictly** individual: plagiarism check will be performed on all submissions, and suspicious submissions will be reported to Lassonde for **a breach of academic honesty**.
- You are given **90 minutes** to complete the submission. The time limit is **strict** so you are solely responsible for leaving enough time (e.g., 10 minutes) to export the completed Java project and upload/submit the archive (.zip) file to eClass.
- This programming test will be graded out of 100 marks, and it accounts for 8% of your course grade.
- This programming test is **purely** a programming test, assessing if you can write **valid** Java programs free of syntax, type, and logical errors.
- You will be given a starter Eclipse Java project archive file (.zip) to import, which contains **no errors** (under the `Problems` panel) to begin.

  Stringent Requirements:

  - Given that you start with a project with no compile-time errors, it is absolutely critical that you submit Java code that compiles (i.e., no red crosses shown on the Eclipse editor).

    If your code 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 **60% penalty** on the resulting marks (e.g., if the revised submission received 50 marks, then the final marks for your test would be 20 marks).

    The amount of penalty will increase in later tests so as to have you be responsible for submitting code that at least compiles.
  - In the `Utilities` class, it is strictly forbidden for you to use any Java library class or method (e.g., `ArrayList`, `Arrays.sort`).

    That is, there must not be an import statement in the beginning of this class.

    You will receive an **immediate zero** for each task where this requirement is violated.

    Use primitive arrays (e.g., `int[]`, `[boolean[]]`), selections, and loops only.

1 Rationale for the Grading Standard

The two most important learning outcome of this course are:

1. Computational thinking (for which you build through lab exercises and assessed by quizzes and 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, this is only your second programming course, and you're still learning. But it is exactly this mind set that restricts your potential of becoming a competent programmer. If we want to train you to be a competent programmer, NOW is the time to enforce the strict (but justifiable) standard.

2 Rules

Timing Constraints of the Test

- Programming Test 2 will be ***opened*** at **02:00pm EST** on **Wednesday**, March 3.
- Programming Test 2 will be ***closed*** at **02:00pm EST**, on **Thursday**, March 4.
- During the 24-hours submission period, there is a **single attempt of 90 minutes** for you to complete the test. That is, once you click on the test link and choose to start it, a timer of 90 minutes will start.
- The time given (90 minutes) includes what it takes for you to:
  - Download the starter project archive file.
  - Import it to Eclipse.
  - Complete the assigned methods.
  - Export the project to an archive file.
  - Upload/submit it to eClass.
- Submission (of an Eclipse Java archive `.zip` file) must be through the ***common (M,N,O) eClass sites***.
- It is your sole responsibility for making sure that the correct version of project archive file is submitted. **Before** you click on the `submit` button on eClass, 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.

Instructions, Starter Project, and Submission

- Both the test instructions and a starter Eclipse Java project archive file will be made available on eClass.
- You will download and import the stater project to the Eclipse (either from your own machine or from a remote lab machine) and complete assigned methods there.
- By the end of the **90-minutes** interval, you must export the Java project as an archive file (.zip) and submit it to eClass.

  This timing requirement is **strict**: if you did not upload and submit the required Java project archive file (with the specified name) by the end of the 90 minutes, you would receive a **zero**.

  Emailing the project archive file to your instructor or TAs will **not** be acceptable.

3 Coverage for the Test

- Elementary Programming, Selections, and Loops.

  Note. There will **not** be any written questions, but you may review your lecture materials to clarify the concepts. Refer to your instructor's lecture materials covered in Weeks 1 – 6.

- The level of difficulty of the actual test will be somewhere between the practice tests (see below) and your labs.
- Java Tutorial Videos: Week 1 to Week 6 (only parts related to loops)
- The concepts about Github, remote labs, and terminal commands are **not** covered in the test.

4 Study Tips for the Test

- The test is meant for **testing your ability of writing valid Java programs**.
- Finish the given example test (for as many times as needed) to **familiarize yourself with the workflow of the test**.
- Reviewing examples covered in the tutorial videos and your lectures may help.
- Make sure you can write programs (without syntax or type errors) to the similar difficulty of your assigned programming exercises for labs.

5 Format

Structure of the starter project you will import is identical to that of Programming Test 1:

- The `model` package contains the `Utilities` class containing a list of utility methods to be implemented by you.
- The `junit_tests` package contains a collection of JUnit tests to help you get started implementing the required utility methods.

  Note. Unlike your Lab1 and Lab2, where you were graded solely by the JUnit tests, for your 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, you are expected to test your program with extra inputs by writing more JUnit tests. To write a new JUnit test, as shown repeatedly in the weekly Java tutorial videos, add a block of code like this:

```
1 @Test
2 public void test_areaOfCircle_01() {
3   double result = Utilities.areaOfCircle(5);
4   assertEquals(78.5, result, 0.1);
5 }
```

  - **Line 1**: It is case-sensitive: `@Test`
  - **Line 2**: Declare the test method, where you always start by writing `public void`, followed by a different method name (e.g., `test_areaOfCircle_02`), followed by an empty list of input parameters (`()`).

    Line 3:
    - For the right-hand side of the assignment, invoke the utility method with some input (e.g., `Utilities.areaOfCircle(5)`).
    - For the left-hand side of the assignment, the type of variable `result` must match the return type of the utility method.
  - **Line 4**: The first input `78.5` to `assertEquals` is the **expected** value, the second input is the return value from the utility method. In the case of comparing two floating-point numbers, a third input `0.1` specifies some tolerance.

  Bottom Line. You can always add a new test by copying, pasting, and modifying a test give to you.
- The `console_apps` package contains a single console application class `App` with the `main` method. A scanner is already declared for you, but should you decide to use it, you are expected to write lines of code for prompting user for inputs, reading inputs, invoking a utility method, and printing to the console.

  Note. It is **not** recommended for you to use the console application class for your testing, as **your submission will only be graded using JUnit tests**.

6 Example Tests

- See under the **Practice Programming Tests** section on the M,N,O eClass site.
- It is expected that you complete these three practice tests (2a, 2b, and 2c), as many time as necessary, to **familiarize yourself with the workflow of the test**.
- You should **practice writing your own JUnit tests**.
- A tutorial series walking over the solution to each utility method is available here:

  `https://www.youtube.com/playlist?list=PL5dxAmCmjv_4UZNiLzeFPAgDDv2vLCGb4`

  Note:

  - The IDE used in the tutorials is not Eclipse, but the lines of solution code will work on your Eclipse.
  - The only exception is that when you wish to invoke a helper method, there is no need to write `this.` as shown in the tutorial video (e.g., the last method `getIndices`).