**CS2110, Spring 2017. Assignment A0**
**The Java assert statement and Eclipse. Due on the CMS —see CMS for date**

### Introduction

This assignment will help you learn more about Eclipse and Java. It is in your best interest to do it as soon as possible —ideally when you have Java and Eclipse working on your computer. The assignment serves two purposes:

1. It introduces you to the Java **assert** statement, which you will use in assignment A1.
2. It helps you get started with Eclipse including: creating a new project, running method `main`, adding a "VM argument" (VM stands for *Virtual Machine*") to the run configuration, and understanding the error messages returned by the Java compiler.

### Getting help

If you don't know where to start, don't understand some of the terms used in this assignment, feel lost, etc., please SEEK HELP FROM THE COURSE STAFF IMMEDIATELY. Or ask a question and look for answers on Piazza. Do not wait. A little in-person help can do wonders. See the course homepage for the contact information for the instructors, TAs, and consultants.

### The Java `assert` statement

The Java assert statement has the form

> **assert** <boolean expression> ;

To execute an **assert** statement, first evaluate the <boolean expression>. If it is evaluates to **true**, then do nothing. If it evaluates to **false**, "throw" a **java.lang.AssertionError** error. Throwing that error (also called an "exception") causes the program to stop executing and print an error message indicating that the line of the program containing the **assert** statement whose <boolean expression> evaluated to **false**.

For example, suppose you have the following on lines 17 and 18.

```
17  x= 5;
18  assert x == 6;
```

and execute the program (we see later how to do that). You get output like the following:

```
Exception in thread "main" java.lang.AssertionError
at Bee.main(Bee.java:18)
```

Parentheses are *not* needed around the <boolean expression>. Do *not* write "**assert** (x==6);" —while it is legal Java, it is harder to read. Put spaces around the operator ==.

### Creating an Eclipse project

To create a new project, do the following:

1. Launch Eclipse
2. Create a new project using menu item **File —> New —> Java Project**. In the window that opens:
   - Give it project name **a0**

- Check that it is using execution environment JavaSE-1.8 or some other version 8 Java Runtime Environment (JRE)
- Click the **Finish** button

Some notes:

- We create separate folders for source (.java) and class (.class) files
- We don't add the projects to Working sets

3. Add a new class to the project using menu item **File —> New —> Class**. In the window that opens:
   - Give it the name **A0**
   - MAKE SURE THE FIELD LABELED "Package:" IS EMPTY! Delete all text in it.
   - Under "Which method stubs would you like to create?" check only the box "**public static void main(…)**"
   - Click the **Finish** button
   - You will see class A0 appear in the main pane of the Eclipse window.
4. Class A0 contains a definition of a method called `main`. When a certain menu item is used, method `main` will be called, resulting in its body (the text between `{` and `}`) being executed. We'll do that in a minute. First, copy the following lines and paste them into the body, in place of the comment "// TODO Auto-generated method stub" —you can remove that comment:

```
System.out.println("Executing method main.");
int x= 5;
System.out.println("x is now " + x);
assert x == 6;
System.out.println("The assert statement was not executed");
```

5. If the lines are not indented well —for example, the first `}` should appear under the **p** of the word **public** above it— select all lines by using control-A (pc) or command-A (mac) and then using control-I (pc) or command-I (mac). The class is now properly indented. It should have no errors in it and the program can be executed.
6. Use menu item **Run —> Run**. This causes method `main` to be executed, and you should see three lines of output:

```
Executing method main.
x is now 5
The assert statement was not executed
```

   This indicates that the assert statement was *not* executed. We next show you how to fix it so that the assert statement is executed.

**Making sure assert statements are executed**

A nice thing about **assert** statements is that their execution can be turned on or off. Thus, after testing a program thoroughly using **assert** statements to help test and debug, when you want to actually use the program to get something done, you can leave **assert** statements in the program but not have them executed during program execution. Then, if an error is detected later on, or changes have to be made in the program, you can turn on assert-statement execution to again help in testing and debugging.

Here is how to turn assert-statement execution on. First, make sure that A0.java is selected in the Package Explorer pane. Then:

1. In Eclipse, choose menu item **Run —> Run Configurations**
2. In the window that opens, click tab **Arguments**
3. In the field titled **VM arguments**, type: -ea
4. Click button **Apply**, near the bottom of the pane
5. Click button **Close** at the bottom of the window.

Having done that, run the program again using menu item **Run —> Run**. The output should now be:

```
Executing method main.
Exception in thread "main" x is now 5
java.lang.AssertionError
   at A0.main(A0.java:11)
```

or

```
Executing method main.
x is now 5
Exception in thread "main"
java.lang.AssertionError
   at A0.main(A0.java:11)
```

The last three lines indicate that an "exception" was "thrown" in method main. In this case, the exception was an **AssertionError**, and it occurred on line 11. You will learn about exceptions and throwing them later on in the course.

**Note**: on your Eclipse, the line number may be different, depending on, for example, you putting extra blank lines at the beginning.

Fix your Eclipse so that line numbers always show. To do that, choose menu item **Eclipse —> Preferences**; then select **General -> Editors -> TextEditors**. Make sure **Text Editors** is highlighted. Check box "Show line numbers". Click **Apply** and then **OK**.

### Fixing Eclipse so that new JUnit run configurations always have argument -ea

In assignment A1 and perhaps other projects, you will be creating new "JUnit test classes" to help you test and debug your program. It is good to *always* have argument -ea present in JUnit testing run configurations. To achieve this, do the following:

1. Choose menu item **Eclipse —> Preferences**
2. In the window that opens, choose item **Java —> JUnit**
3. Near the top of the window, check the box "Add '-ea' to VM arguments when creating a new JUnit launch configuration.

### What to Submit

On the CMS for the course, submit source file A0.java by the due date given by the CMS.