Eclipse Debugging

Objective(s)

- Learn to use the Eclipse IDE to debug Java programs

Lab setup

1. Download and import the provided Eclipse project.

Debugging

Debugging involves finding and resolving bugs preventing programs from running properly. We will focus on the tools provided in the Eclipse IDE that help us debug Java programs.

Types of errors

Debugging does not deal with compile time/syntax errors. It only deals with errors that occur while the program is running.

Runtime Errors
Runtime errors may occur with uncaught exceptions. The JVM will stop execution of the program and display the line where it encountered the exception. We might decide to handle the exception or track down what may be the logical error that causes the exception.

Logical Errors
Logical errors may not cause the program to terminate but will cause incorrect output/results. These errors can be hard to track down, but we can use debugging tools to make the job easier.

Debugging tools

While writing programs we work mostly in the Java perspective. This perspective includes a lot of features we may not touch, but mainly it includes the package explorer, code editor, Problems window, etc. Eclipse also has a Debug perspective which contains all the debugging tools.

To begin debugging, Right-click on the project and choose Debug As > Java Application. We can instead choose Debug configurations make changes to the execution of the program such as adding command-line arguments.

When the program terminates, Eclipse will switch to the Debug perspective. You will find the Debug view on the left of the window. This provides a visualization of the stack trace, where the program currently is. On the right, you can find the Variables, Breakpoints, and Expressions view. The Variables view will let you watch the values of variables as you step through the program. The Breakpoints view will show any breakpoints you have set in the program. The Expressions view allows you to create an expression and watch its value as you step through the program.

Breakpoints
Breakpoints can be set to tell the debugger when to pause the execution of the program. Once paused, you can inspect the state of the program using the Variables/Expressions view.

Debugging process

Debugging can be an arduous venture. It helps to have a process to follow.

1. Identify the type of error you are attempting to resolve.
2. Form a hypothesis.
    a. Where could the problem originate?
    b. What variables in the program could be causing the problem?
3. Use debugging tools to narrow the search.
    a. Set breakpoints before the section of code that you think is causing the problem.
    b. Use the debugger to step through each line of code.

Graded submission Task 1:
Let's try debugging on some real piece of code. The code you are given is for a discrete event simulation using a simulation library. The program occasionally throws an error due to a null value. See if you can identify the issue.

For this task, you do not need to submit any code. Instead, you will submit screenshots of your debugging process. Include screenshots of the Variables, Breakpoints, Expressions, and Debug views. Include screenshots with the program's initial error and during your debugging process. Compile these screenshots in a Word document and label each appropriately.

Rubrics
Task 1 (points allotted) - Required screenshots included (all views, program's initial error, debugging process) 10