Santiago Bermudez

Lab04: Skills - Using a Debugger

Getting Ready: Before going any further, you should:

1. Setup your development environment.

2. Download the following files:
   `PhoneDriver.java`
   `PhoneCard.java`
   to an appropriate directory/folder. (In most browsers/OSs, the easiest way to do this is by right-clicking/control-clicking on each of the links above.)

3. If you don't already have one from earlier in the semester, create a project named eclipseskills.

4. Drag the file `PhoneCard.java` and `PhoneDriver.java` into the default package (using the "Copy files" option).

5. Open `PhoneCard.java` and `PhoneDriver.java`.

Part 1. Review: This part of the lab will review a few topics related to object-oriented programming in Java.

1. In the `main()` method in the `PhoneDriver` class, what kind of objects are end, now, and start?

   End, now, and start are declared variables of type Date.

2. In the `main()` method in the `PhoneDriver` class, what kind of object is card?

   Card is a declared variable of type PhoneCard.

3. Where is the code for the `PhoneCard` class?

   In the PhoneCard.java file, there is a public method called PhoneCard. This method has the following parameters: double initialBalance, int maxCalls, and double dollarsPerMinute.

4. Where is the code for the `Date` class?

   In the PhoneDriver.java file, there is a line on top that says “import java.util.date;”. That line gives us the Date class.

5. Read the documentation for the `Date` (https://docs.oracle.com/javase/7/docs/api/java/util/Date.html). Make sure you find the documentation for the `Date` class that is in `java.util`. (There are several `Date` classes in the Java library.)

6. When you construct a `Date` object using the default constructor (i.e., the constructor that has no parameters), what properties will it have?

   The default constructor of Date will simply just give the date and initialize it to represent the current date and time at which it was allocated, measured to the nearest millisecond.

Part 2. Setting a Breakpoint: One of the nice things about running an application in a debugger is that you can stop the execution at one or more pre-defined locations (called *breakpoints*). This part of the lab will teach you how.

1. Click on the tab containing `PhoneDriver.java` to make sure that it has the focus.
2. Right-click in line 29 of `PhoneDriver.java` and pull down to Toggle Breakpoint.
3. (this line# may be different for some of you with a probability chance of less than 0.05%. The line# should match :

   if (availableMillis > 0)

4. What happened?

   For me, it created a breakpoint at line 29. It added a blue circle or dot to the left of line 29.

5. Click on . This will run `PhoneDriver` and stop the execution at the breakpoint (i.e., line 29). Note: If prompted, allow Eclipse to enter the "Debug Perspective".
6. What happened?

   It highlighted the line of code that had a breakpoint on its left. It also added a debug window on the right which showed what the variable values were at that line and what their names were. It also showed the variable IDs and has a blue pointer arrow pointing at the line.

Part 3. Checking State Information: Another nice thing about running an application in a debugger is that, once you stop the execution at a breakpoint, you can check state information (e.g., the value of attributes and variables). This part of the lab will teach you how.

1. Click on the "Variables" tab on the left side of the debug window.
2. Click on the "tree icon" next to "Locals" to expand it.
3. What is the current value of `availableMillis`?

   5999999

Part 4. Stepping Over Lines: When running an application in a debugger, once you stop the execution at a breakpoint, you can continue the execution one "step" at a time. This part of the lab will teach you how.

1. Click on . This will run `PhoneDriver` again and stop the execution at the breakpoint (i.e., line 29).
2. Click on the button.
3. What happened?

   The blue pointer arrow moved to the next line of code, which was on line 31. It also highlighted line 31 and unhighlighted line 29.

4. Click on the button until the next if statement is highlighted.
5. What is the current value of `availableMillis`? (Hint: Look in the "Variables" tab. You may need to scroll.)

   5399999

6. Click on the button to run to the end of the application.

Part 5: Stepping Into Lines: So far, all of the "stepping" you have done has been in one method in one class. This is called "stepping over". You can also "step into" a line of code to see what happens there. This part of the lab will teach you how.

1. Click on . This will run `PhoneDriver` and stop the execution at the breakpoint (i.e., line 29).
2. Click on the button.
3. What happened?

   The blue pointer arrow moved over to the next line of code, which is line 31 in this case. It also highlighted line 31 and unhighlighted line 29.

4. Click on the button again.
5. What happened?

   The blue pointer arrow moved to the PhoneCard.java file. From there it moved to line 87 in the startCall method.

6. Look at the call stack in the "Debug" tab. It tells you what class and method you are in and where this method was called from.
7. What method is currently being executed (and what class is it in)?

   The startCall method is currently being executed and it is inside the PhoneCard class.

8. What line is currently being executed?

   Line 87 is currently being executed

9. Where was this method called from?

   From PhoneDriver class, line 31 inside the main method.

10. Click on the "triangle icon" next to this to expand it.
11. What is the current value of `balance`?

    10.0

12. Click on the button.
13. Click on the "triangle icon" next to callNumbers to expand it.
14. What is the current value of `callNumbers[0]`?

    “540-568-1671” (id = 32)

15. Click on the "triangle icon" next to callStarts to expand it.
16. What is the current value of `callStarts[0]`?

[If the variable's value says null, press the ***Step Over*** button or **F6** one more time so it shows.]

Date (id = 29)

17. Why does it have that value?

    Because date is being used to determine when the call has started.

18. Click on the button twice.
19. What happened?

    The blue arrow pointer got sent back to the PhoneDrive.java file. When I clicked the second time, from there it went to line 37.

20. Add a breakpoint at line 41 in `PhoneDriver.java` (i.e., the line that constructs a `Date`).
21. Click on the button. This will run the application to the next breakpoint (i.e., line 41).
22. Click on the button.
23. Why didn't the debugger step into the `Date` constructor?

    Because within the compiler, things that are inside parentheses tend to get higher priority. The line of code was:
    "start = new Date(end.getTime() + 1200000);"
    The call to the Date constructor was preceded by the getTime method.

24. Click on the button to run to the end.
25. Click on Window+Perspective+Close Perspective to close the "Debug Perspective".

Part 6: Advanced Topics: This part of the lab will help you use the debugger more efficiently.

1. How can you display all of the breakpoints?

   When you set your breakpoints and click on the debug button or the button, a debug window should appear with three different tabs on top. Click on the breakpoints tab and that should show you all of the breakpoints.

2. What is a conditional breakpoint?

   A conditional breakpoint is a feature in Eclipse that allows you to attach conditions to breakpoints if you wanted to stop at a breakpoint only when a certain condition applies. This is useful in that you won't have to add temporary code when debugging. You can utilize this by right-clicking on a breakpoint, selecting "Breakpoint properties…", checking the "Conditional" box, and then you can write your temporary code underneath.

3. How can you see variable references while debugging?

   In the debug window, there are variables under the variables tab. You can right-click anyone of those variables and then select the "All References…" option if you wanted to see a list of all the Java objects that have a reference to the selected variable.