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.javainto 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?

   They are Date objects.

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

   A PhoneCard object.

3. Where is the code for the PhoneCard class?

   In the current working directory.

4. Where is the code for the Date class?

   In the Java library (package java.util).

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?

   It will contain the current date.

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

   It will contain the current date.

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. What happened?

   A blue circle appeared on the left side of line 29.

4. 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".

5. What happened?

   Eclipse changed to the "debug perspective". A blue arrow appeared on the left side of line 29 and the line was highlighted.

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 arrow and highlight moved down to the next executable statement on line 31.

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 arrow moved down to the next executable statement. In this case, we stepped into the if statement (line 31).

4. Click on the ⤵ button again.

5. What happened?
   The blue arrow moved into the startCall() method in the PhoneCard class.

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 in the PhoneCard class.

8. What line is currently being executed?
   Line 87 in the PhoneCard class.

9. Where was this method called from?
   The main() method in the PhoneDriver class (line 31).

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"

15. Click on the "triangle icon" next to callStarts to expand it.

16. What is the current value of `callStarts[0]`?
    callStarts[0]=Date (id=29)

17. Why does it have that value?
    The value of the start is stored into callStarts[0]. Start contains the date the call began.

18. Click on the ↷ button twice.

19. What happened?
    Execution returned to PhoneDriver.

20. Add a breakpoint at line 46 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 46).
22. Click on the ⤵ button.
23. Why didn't the debugger step into the `Date` constructor?
    The getTime() method is being called.

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?
   Using the Breakpoints View.

2. What is a conditional breakpoint?
   A breakpoint that allows you to break inside a code block when a defined expression evaluates to true.

3. How can you see variable references while debugging?

   By managing the Variables View