Lab 6
BlueJ, a java IDE

Overview

In the first week of the lab, a java editor was presented as a tool for creating java source files. Hopefully you understand how jEdit works and are comfortable with the purpose it has in the process of creating programs. Fortunately there are lots of other tools for development, some of which are specialized and others that are complicated and feature-rich for more advanced programmers.

Now we turn to an editor that supports the development of java programs with a special emphasis on creating objects. You should be at a point in your reading where you are being introduced to objects, or you will be in the near future, and we will be using BlueJ to develop programs for most of our activities during the rest of the semester.

You will also find that this IDE (integrated development environment) will allow you to execute and test programs without leaving the IDE. I.e., you can run the programs from within BlueJ without using the command-line.

Recall the idea of abstraction. BlueJ allows us to more effectively create these abstractions critical to good programming. Nonetheless, some of this may seem a little odd at first, so be patient as you learn about BlueJ and objects. Both are needed to get the full impact of object oriented programming (OOP).

In this lab we will cover the basics of creating and running programs with BlueJ, then investigate how to use the debugger to test programs and look for errors. Initially little attention will be placed on objects, with more attention placed on OOP in the labs that follow.

BlueJ has already been installed on the lab machines, but videos on installing and configuring BlueJ on your own machine are located on the course website.
Before proceeding, print or obtain a copy of the last page of this lab for the purpose of grading your lab. This lab will NOT be graded with WebCAT. As you complete each step of the lab, obtain your instructor’s signature and submit the sheet to your instructor upon completing the lab.

Watch the video

*Running BlueJ*

for an introduction to the use of the BlueJ IDE.

When you finish watching the video, enter the program HelloWorld, compile and run it.

Show your instructor your work when you are finished.

*(This is NOT submitted to WebCAT)*

There are a number of strategies for debugging programs; some simple and moderately effective, and others that are more effective but more complicated to use. In this lab three strategies will be examined:

- Use of println to examine the value of variables and trace the flow of control.
- Use of the debugger in an IDE like bluej.
- Use of jUnit testing.

Each of these have advantages based on what the programmer is trying to determine. In the first case (using println) there is great flexibility and little overhead in learning how to use it and incorporating it into the application.
1. Intro to String methods

So begin with this example that will use Strings. Look at the “java 6 docs” and find the specifications for Strings. Examine the methods substring(), length(), and trim() and determine what they do.

Once you have read the definitions of the methods, guess what you think these statements will print and then add the statements to your HelloWorld application and find out what they really do.

```java
String One=" Bobby ";
System.out.println(One.length());
System.out.println((One.trim()).length());
String Two="Bobby Paul Smith Jr. ";
System.out.println(Two.substring(1,3));
System.out.println(Two.substring(4,7));
```

2. Using the string operations

Finally, add statements to print the first and last name of the person from the variable “Two” using the substring commands (similar to the statements above).

Show your instructor the modified HelloWorld application with the code above and the statements that you have added to print the first and last names when you are finished.

3. Finding errors in use of the string operations

Use the java program PersonName.java for the next part. This program has a method NumberNames() which will count the number of names in a String. For example, “Bobby Paul Smith Jr.” should return 4.

The problem is that there is a (are) mistake(s) in the application and the program will become stuck in an infinite loop. This loop is not a simple count controlled loop, so it is more difficult to find the error, but use your basic looping principles.

- Look for loop termination.
- What is necessary for this loop to terminate?
- How can we assure that the loop is getting closer to termination on each iteration of the loop?
This part of the lab will begin the process of learning to use the tools listed above to debug the program.

**BEFORE RUNNING THE PROGRAM, NOTE:**
If you execute a program that is stuck in an infinite loop in the BlueJ IDE, there is a “barber pole” in the bottom left corner of the IDE. In order to stop a program that is in a loop, you can right-click the “pole” icon and “reset the machine” which will halt the program but leave BlueJ running.

Run the PersonName application and note that it is in an infinite loop.

Stop the program by resetting the IDE as explained above.

This program is written with a loop that attempts to extract one name at a time from the string, counting the names as it progresses. When it removes all of the names, the loop stops. …. But it doesn’t!

1. **Using “println” to examine intermediate results.**

Begin the process of using our most primitive tool, “println”, to debug the problem by printing the counter to see if it is actually running the loop. The statement is already in the code, simply uncomment the print statement in the loop and run it again.

You should see that it is still in a loop and that `numnames` is simply getting larger and larger.

Stop the program again.

While printing is useful in many cases, it has limited use in a program with an infinite loop. It is unclear why the program is continuing and whether or not it is making any progress towards solving the problem because the output is so large that the initial results are not visible in the window.

Before examining another way to debug the program, add a statement to print the value of `testname` on each iteration.

**Question 3**  
**Explain what is wrong with the application and tell the instructor your explanation.**
2. Using the BlueJ Debugger: stopping loops, examining intermediate results.

Before you correct the problem, reflect on the use of the println. Incorporating it into the program was very easy and this confirmed that the program was actually in a loop. In addition, the first output led us to examine more information that eventually led to a theory regarding the cause of the error. Debugging programs is a process of deduction that is directed by the information we gather, so it is important to determine what information would help to resolve the problem and then ask for it.

In the previous example we used a println statement that helped us to understand the flow of control in addition to how the value of variables change. However, the program continued to run without pausing when we would have preferred to have the program pause and allow us to ask questions about what was happening.

Debuggers like the one in BlueJ will allow us to do just that:

1. Slow the pace of execution
2. Stop the program on each statement
3. Stop the program when it reaches a certain point
4. Examine the value of certain variables.
5. And MORE!.

Consider using the debugger to investigate the previous error. Open the source of the PersonName.java file. Simply use the source editor window (double click on the PersonName icon/class). Be sure that the program will compile. Then go to the line:

\[ \text{testname}=\text{testname}.substring(\text{posBlank}+1,\text{testname}.length()); \]

and click once in the narrow column on the left, on the same line as this statement. A STOP sign should pop up. **If it doesn’t, get some help before moving on.**

Now run the program (main), just like you ran the HelloWorld. When execution reaches the line with the STOP, the program will halt, pop up a window with the values of all of your variables and wait for you to tell it what to do.

Does this confirm your theory about what was going on or give you some other options to consider?

Before ending this use of the debugger, try using the “Step” button a couple of times. Note that the program executes one line at a time (under your control) and updates the value of variables as it progresses. You should be able to see which line has executed by looking at the source file where you placed the STOP sign.

Now press “Continue”. Where did it stop? Why?

**After you have answered these questions for yourself, show your instructor the BlueJ IDE with the debugger halted in a state where it displays the values of the variables.**
Fix the PersonName program and show your instructor when the main routine correctly runs all four test cases.

Now that you have used both approaches to this, observe how easy it was to add the println, but that you had to plan what you wanted to do before running the test. When using the debugger, much more information and control is available to you and you are able to react more than plan. Also don’t miss the obvious.. without an IDE you can’t use a debugger, but you can always use println.

3. Testing with Junit

The last part of this lab is to investigate how testing is used in finding errors and alerting us to their existence. Picture yourself working on a program that is 20000 lines long. You are asked to make some changes to the program with very limited knowledge of the operation of the complete program. Once you make your changes, how do you convince yourself that all of the old program components still function properly? Testing all of that code would take a long time, but automating the testing process will help to keep the program functioning properly while maintaining it.

Enter JUnit.

JUnit testing allows one to build complex testing tools that are easy to rerun and assure integrity of the program functions while undergoing changes. The form for this is somewhat intuitive and very similar to things you have already been doing.

The simplest way to view JUnit is to consider it a tool to test methods. Pass a method certain parameters for which you already know the expected results. Then compare the results the method returns with the results expected.

Example:

Factorial of 4 is 24, factorial of 5 is 120, and factorial of 6 is 720. We can build a JUnit tester that will call the factorial method three different times to see if it returns the proper value for each of these cases.

The last activity for this lab is to build a JUnit mechanism to test the PersonName class and specifically the NumberNames method with the four cases from the Person Name class.

View the video that shows you how to create a JUnit tester and create one to test the four cases.

When you are finished, show your instructor.
Lab 6: Completion Table

<table>
<thead>
<tr>
<th>Question</th>
<th>Completed</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Instructor’s signature _______________________________ Grade ____________