**Name\_\_\_Matt Andre\_\_\_\_Jeff Fisher\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**CSC 205 Fall 2012 LAB 5 Points 20**

1. **Demonstrate the use of Insertion, Selection and Bubble-sort algorithms using the templates below: (12)**

INSERTION SORT

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Initial | Insert 67 | Insert 12 | Insert 10 | Insert 75 | Insert 88 | Insert 93 | Insert 45 |
| 67 | 67 | 12 | 10 | 10 | 10 | 10 | 10 |
| 82 | 82 | 67 | 12 | 12 | 12 | 12 | 12 |
| 12 | 12 | 82 | 67 | 67 | 67 | 67 | 45 |
| 10 | 10 | 10 | 82 | 75 | 75 | 75 | 67 |
| 75 | 75 | 75 | 75 | 82 | 82 | 82 | 75 |
| 88 | 88 | 88 | 88 | 88 | 88 | 88 | 82 |
| 93 | 93 | 93 | 93 | 93 | 93 | 93 | 88 |
| 45 | 45 | 45 | 45 | 45 | 45 | 45 | 93 |

SELECTION SORT

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Initial |  |  |  |  |  |  |  |
| 67 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| 82 | 82 | 12 | 12 | 12 | 12 | 12 | 12 |
| 12 | 12 | 82 | 45 | 45 | 45 | 45 | 45 |
| 10 | 67 | 67 | 67 | 67 | 67 | 67 | 67 |
| 75 | 75 | 75 | 75 | 75 | 75 | 75 | 75 |
| 88 | 88 | 88 | 88 | 88 | 88 | 82 | 82 |
| 93 | 93 | 93 | 93 | 93 | 93 | 93 | 88 |
| 45 | 45 | 45 | 82 | 82 | 82 | 88 | 93 |

BUBBLESORT

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Initial | After  Pass1 | After  Pass2 | After  Pass3 | … | … | … |  |
| 67 | 10 | 10 | 10 | 10 | 10 |  |  |
| 82 | 67 | 12 | 12 | 12 | 12 |  |  |
| 12 | 82 | 67 | 45 | 45 | 45 |  |  |
| 10 | 12 | 82 | 67 | 67 | 67 |  |  |
| 75 | 45 | 45 | 82 | 75 | 75 |  |  |
| 88 | 75 | 75 | 75 | 82 | 82 |  |  |
| 93 | 88 | 88 | 88 | 88 | 88 |  |  |
| 45 | 93 | 93 | 93 | 93 | 93 |  |  |
| Swapped? | yes | yes | yes | yes | no |  |  |

INSERTION SORT

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Initial | Insert 74 | Insert 92 | Insert 12 | Insert 60 | Insert 36 | Insert 75 | Insert 10 |
| 82 | 74 | 74 | 12 | 12 | 12 | 12 | 10 |
| 74 | 82 | 82 | 74 | 60 | 36 | 36 | 12 |
| 92 | 92 | 92 | 82 | 74 | 60 | 60 | 36 |
| 12 | 12 | 12 | 92 | 82 | 74 | 74 | 60 |
| 60 | 60 | 60 | 60 | 92 | 82 | 75 | 74 |
| 36 | 36 | 36 | 36 | 36 | 92 | 82 | 75 |
| 75 | 75 | 75 | 75 | 75 | 75 | 92 | 82 |
| 10 | 10 | 10 | 10 | 10 | 10 | 10 | 92 |

SELECTION SORT

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Initial |  |  |  |  |  |  |  |
| 82 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| 74 | 74 | 12 | 12 | 12 | 12 | 12 | 12 |
| 92 | 92 | 92 | 36 | 36 | 36 | 36 | 36 |
| 12 | 12 | 74 | 74 | 60 | 60 | 60 | 60 |
| 60 | 60 | 60 | 60 | 74 | 74 | 74 | 74 |
| 36 | 36 | 36 | 92 | 92 | 92 | 75 | 75 |
| 75 | 75 | 75 | 75 | 75 | 75 | 92 | 82 |
| 10 | 82 | 82 | 82 | 82 | 82 | 82 | 92 |

BUBBLESORT

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Initial | After  Pass1 | After  Pass2 | After  Pass3 | … | … | … |  |
| 82 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| 74 | 82 | 12 | 12 | 12 | 12 | 12 | 12 |
| 92 | 74 | 82 | 36 | 36 | 36 | 36 | 36 |
| 12 | 92 | 74 | 82 | 60 | 60 | 60 | 60 |
| 60 | 12 | 92 | 74 | 82 | 74 | 74 | 74 |
| 36 | 60 | 36 | 92 | 74 | 82 | 75 | 75 |
| 75 | 36 | 60 | 60 | 92 | 75 | 82 | 82 |
| 10 | 75 | 75 | 75 | 75 | 92 | 92 | 92 |
| Swapped? | yes | yes | yes | yes | yes | yes | no |

**DEBUGGING USING Eclipse**

Debugger Tutorial Available at

<http://www.vogella.de/articles/EclipseDebugging/article.html>

**Debugger:**  A debugger is a tool that lets you control the execution of your program. You can suspend the execution at any point and observe the values of variables. This is very useful in debugging – i.e. finding logical errors in your program.

**Breakpoint:** A breakpoint is an executable line in your code at which the program can be paused.

**How to setup a breakpoint?**

Right-click on the small left column in the code (It should be an executable line) and click on “toggle breakpoint”

OR

Simply double-click in the left column of an executable line.

**Starting the Debugger: Run>Debug (Or click on the ‘bug’ icon)**

At this point, if you have not set any breakpoints, the program will run normally. If you have setup break points, it will pause at the first break point. (If Eclipse asks you if you want to switch to the “debugger perspective” – say Yes.)

**There are several actions you can take when the execution has paused at a breakpoint.**

**Step Over:** This means that you want to execute this line and stop at the next line in this code. For example, if the code is: (Note: line numbers are not a part of the code)

Line#

**10** **X = method1(a, b, c);**

**11 Y = method2(a, b, c);**

and it has paused at the first line (Line# 10), if you say step-over, it will execute the first line and then pause at the second line (Line# 11). You can examine the value of X when it has paused at Line# 11. (**Function Key: F6)**

**Step Into:** If the current line is a method call, the debugger will jump to the first executable line of the method. After this you can again use step-over or step-into.

In the above example, if the code for method1 was:

**public int method1(int p, int q, int r)**

**{**

**int s = 0;**

**s = (p+q+r) / 3;**

**return s;**

**}**

If you chose “step-into” at Line# 10, the debugger will jump to the line int s = 0; (**Function Key: F5)**

**Skip to the next breakpoint:** Sometimes, you may not want to go step by step. You may want it to run until hits the next breakpoint (or the end of the program). Use the **F8** key for this.

**Return to the caller:** (**Function Key: F7)** This will take the control back to the caller. We rarely ever use this.

**Stack:** The current call stack is shown in the “Debug view”

**Variables:** Values of the currently visible variables is shown in the “Variables view”

### Skip all breakpoints: If you want to temporarily de-activate all your breakpoints you can press the button "Skip all breakpoints" which is visible if you select the tab breakpoints. If you press this button again the breakpoints will get activated again.

### Watchpoint: A watchpoint is a breakpoint at which it stops whenever a field read or changed. You can set a watchpoint through a double-click on the left side before the field declaration. Via the properties of the watchpoint you can define if the breakpoint should be hit during read access (Field Access) or during write access (Field Modification).

YouTube Videos: Watch these videos at your own time:

<http://www.youtube.com/watch?v=D10Mc9Uvs0E> Java for Total Beginners (Lesson 1)

<http://www.youtube.com/watch?v=WeSitNPAExg&feature=related> About Using Debugger (Lesson 1)

**Your Assignment: Correct the logical errors in the following programs using the debugger: (8)**

In each of the problems, comments are used to specify what it is supposed to do. However, the programs have bug(s) in them. You are asked to find what the bugs are and correct them using the Eclipse debugger.

* For each problem first run the program as is and observe the output. Is it correct?
* If not correct, find the bugs and explain what the bug was and how you fixed it

**1.**

**public class Debugging1 {**

**/\*\***

**\* Find how many times the letter g occurs in a String.**

**\* This program has bugs in it. Use the debugger to find**

**\* what the bugs are and fix them.**

**\*/**

**public static void main(String[] args)**

**{**

**int letterCount = 0;**

**String checkWord = "Debugging";**

**String singleLetter = "";**

**for (int k = 0; k < checkWord.length(); k++)**

**{**

**singleLetter = checkWord.substring(1,1);**

**if (singleLetter == "g") letterCount++;**

**}**

**System.*out*.println("g was found " + letterCount + " times");**

**}**

**}**

Explain the bugs and how you fixed them below:

Original output was “g was found 0 times” which is incorrect.

Errors:

Using incorrect indices for creating substring of checkWord.

Should be **singleLetter = checkWord.substring(k,k+1);**

Using incorrect comparison for comparing singleLetter to “g”.

Should be **singleLetter.equals("g")**

Output after Corrections: “g was found 3 times”

**2. This class might have an infinite loop. Use the red square button in Eclipse to terminate the program.**

**public class Debugging2 {**

**public static void main(String[] args) {**

**// ------Problem 2a. Print the numbers 1 to 10------------**

**int i = 0;**

**for (i=1; i<=10; i++) ;**

**{**

**System.*out*.println("Number is " + i);**

**}**

**//--------Problem 2b. Comparing two Strings--------------**

**String s1 = "abcd";**

**String s2 = "abcd";**

**String s3 = "abcdef";**

**if (s1 == s2)**

**System.*out*.println(s1 + " and " + s2 + " are equal");**

**else**

**System.*out*.println(s1 + " and " + s2 + " are NOT equal");**

**if (s1 == s3)**

**System.*out*.println(s1 + " and " + s3 + " are equal");**

**else**

**System.*out*.println(s1 + " and " + s3 + " are equal");**

**//-------------------------------------------------------**

**/\*\***

**\* Problem 2c. The following two loops are supposed to do**

**\* the same thing. The while loop works fine, prints numbers**

**\* 10 down to 0. However, the for loop has a problem. Fix it.**

**\*/**

**int num = 10;**

**while(num >= 0)**

**{**

**System.*out*.println(num);**

**num--;**

**}**

**num = 10;**

**for(int k = num; k <= num; k--)**

**{**

**System.*out*.println(num);**

**num--;**

**}**

**}**

**}**

**Explain the bugs and how you fixed them below:**

Misplaced Semicolon after for loop. Should be: **for (i=1; i<=10; i++)**

Incorrect String comparisons using “==” Should be **s1.equals(s2)** and **s1.equals(s3)**

Output says s1 and s3 are always equal. Change output in else statement to:

**System.out.println(s1 + " and " + s3 + " are NOT equal");**

Incorrect for loop conditions. Change to **for(int k = num; k >= 0; k--)**

**3.**

**import java.util.Scanner; // Scanner class used for getting user input**

**public class Debugging3 {**

**// AverageAndSmallest**

**// This program will get three numbers from the user**

**// and determine the smallest number and the average**

**// The manin method that begins execution of Java application**

**public static void main(String args[])**

**{**

**// variable declarations**

**int num1; // first number to compare**

**int num2; // second number to compare**

**int num3; // third number to compare**

**int smallest = 0; // variable to hold smallest**

**double average; // variable to hold average**

**// create Scanner to capture input from console**

**Scanner input = new Scanner(System.*in*);**

**// get user input, num1 and num2**

**System.*out*.print("Enter first number: ");**

**num1 = input.nextInt();**

**System.*out*.print("Enter second number: ");**

**num2 = input.nextInt();**

**System.*out*.print("Enter third number: ");**

**num3 = input.nextInt();**

**// Compare and determine the smallest**

**if (num1 > num2)**

**smallest = num2;**

**if (num1 < num2)**

**smallest = num1;**

**if (num3 < smallest)**

**smallest = num3;**

**// Calculate the average**

**average = num1 + num2 + num3 / 3;**

**// Display average and the samllest**

**System.*out*.printf("Average is %.2f\n", average);**

**System.*out*.printf("Smallest is %d\n", smallest);**

**}**

**}**

**Explain the bugs and how you fixed them below:**

Doesn’t work if inputs are equal. Change first if statement condition to: **if (num1 >= num2)**

Calculates average incorrectly. Change to: **average = (num1 + num2 + num3) / 3.0;**