Lab #8 - Debugging Programs in Eclipse

The ability to use a graphics debugger to debug program is crucial in programming. It could save you countless hours guessing on what went wrong.

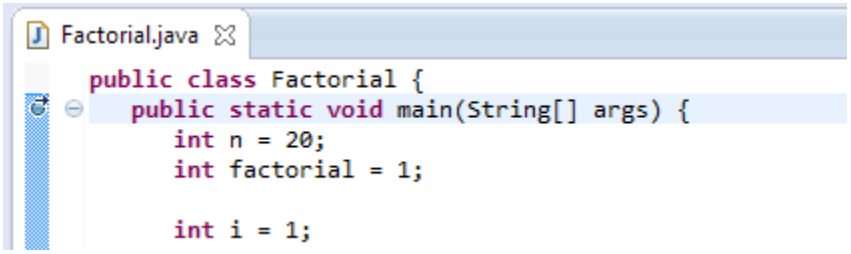
Write a Java Program

The following program computes and prints the factorial of *n* (`=1*2*3*...*n`). The program, however, has a logical error and produces a wrong answer for *n*=`20` ("`The Factorial of 20 is -2102132736`" - a negative number?!).

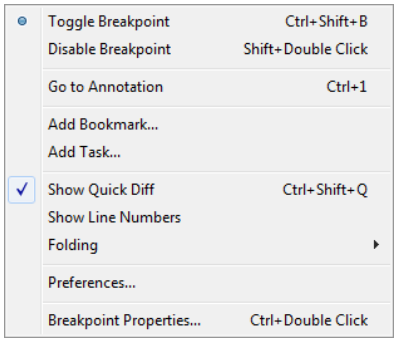
```
1/** Compute the Factorial of n, where n=20.
2  *    n! = 1*2*3*...*n
3  */
4public class Factorial {
5   public static void main(String[] args) {
6      int n = 20;
7      int factorial = 1;
8
9      int i = 1;
10     while(i <= n) {
11        factorial = factorial * i;
12        i++;
13     }
14     System.out.println("The Factorial of " + n + " is " + factorial);
15  }
16}
```

Step 1: Set an Initial Breakpoint

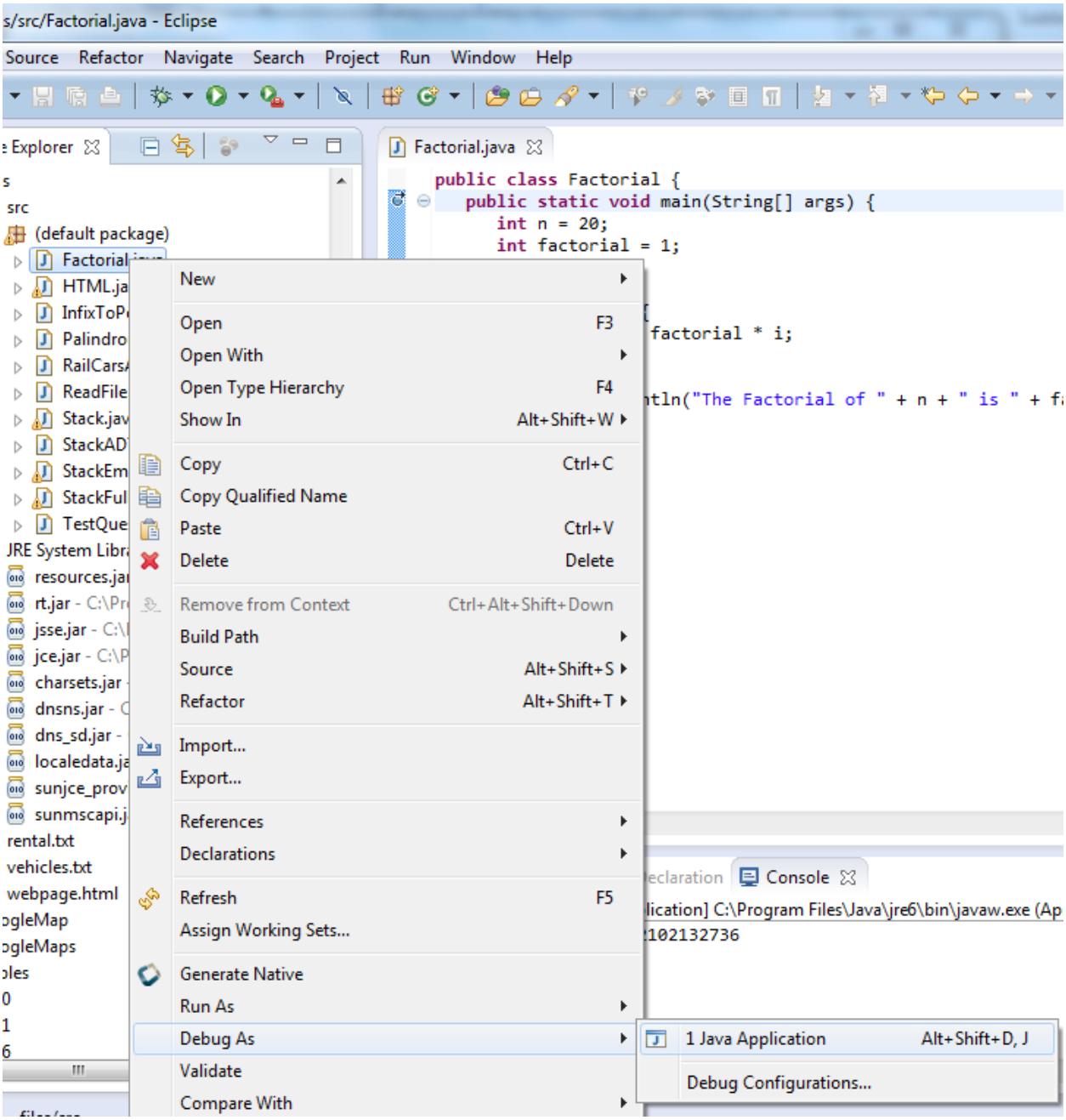
Double-click here
To set a Breakpoint
Or right-click and choose
Toggle Breakpoint



A *breakpoint* suspends program execution so one can examine the internal states (e.g., values of variables) of the program. Before starting the debugger, you need to set a breakpoint to suspend the execution inside the program. Set a breakpoint at the beginning of the **`main()`** method by double-clicking on the *left margin* of the line containing **`main()`** or right-click on the left margin and select **Toggle Breakpoint** from the menu. A *blue circle* appears in the left-margin indicating a breakpoint is set at that line.

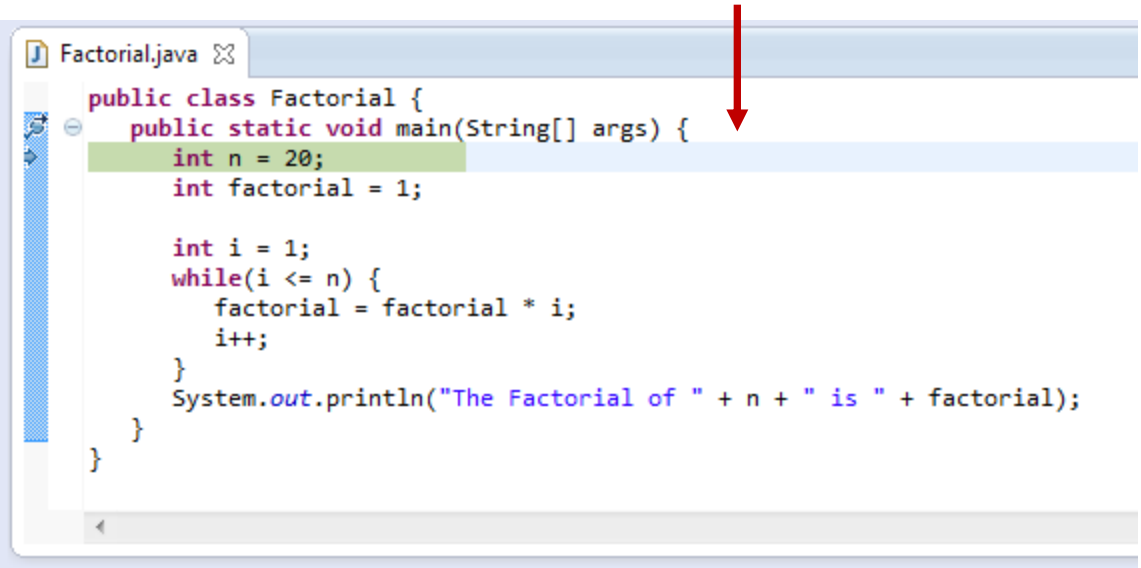


Step 2: Start the Debugger

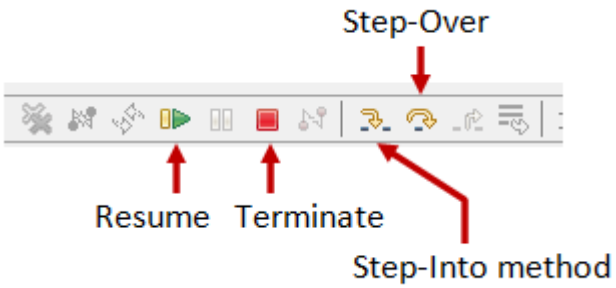


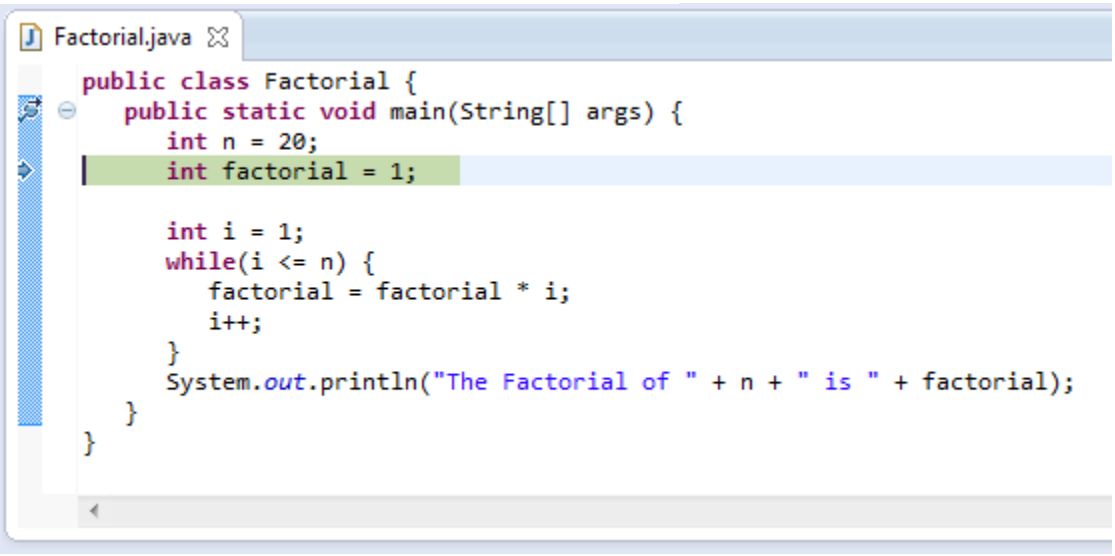
Right click anywhere in the source code and choose **Debug As** followed by **Java Application**.
Click the **Yes** button in the **Confirm Perspective Switch** dialog box to switch to the "Debug" perspective.

The program begins execution and stops at the first breakpoint encountered - `int n = 20;`.
Note the highlighted line indicates the next statement to be executed.

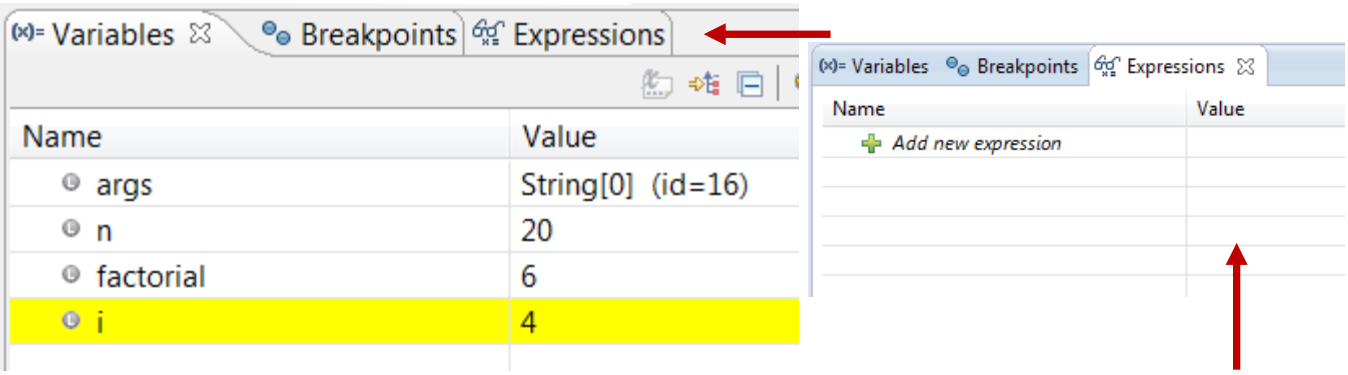


Step into (F5) will highlight the next executable statement.





Step 3: Step-Over and Watch the Variables and Outputs



In addition to the Variables tab, the Expressions tab can be used to watch the values of expressions.

Click the "Step Over" button (or select "Step Over" from "Run" menu) to *single-step* thru your program. At each of the step, examine the value of the variables (in the "Variable" panel) and the outputs produced by your program (in the "Console" Panel), if any. You can also place your cursor at any variable to inspect the content of the variable.
Single-stepping thru the program and watching the values of internal variables and the outputs produced is the *ultimate* mean in debugging programs - because it is exactly how the computer runs your program!

Step 4: Breakpoint, Run-To-Line, Resume and Terminate

A breakpoint *suspends* program execution and let you examine the internal states of the program. To set a breakpoint on a particular statement, double-click the left-margin of that line (or select "Toggle Breakpoint" from "Run" menu).

"Resume" continues the program execution, up to the next breakpoint, or till the end of the program.

"Single-step" thru a loop with a large count is time-consuming. You could set a breakpoint at the statement immediately outside the loop (e.g., Line 11 of the above program), and issue "Resume" to complete the loop.

Alternatively, you can place the cursor on a particular statement, and issue "Run-To-Line" from the "Run" menu to continue execution up to the line.

"Terminate" ends the debugging session. Always terminate your current debugging session using "Terminate" or "Resume" till the end of the program.

Step 5: Switching Back to Java perspective

Click the "Java" perspective icon on the upper-right corner to switch back to the "Java" perspective for further programming (or "Window" menu ⇒ Open Perspective ⇒ Java).

Other Debugger's Features

Modify the Value of a Variable: You can modify the value of a variable by entering a new value in the "Variable" panel. This is handy for temporarily modifying the behavior of a program, without changing the source code.

Step-Into and Step-Return: To debug a *method*, you need to use "Step-Into" to step into the *first* statement of the method. ("Step-Over" runs the function in a single step without stepping through the statements within the function.) You could use "Step-Return" to return back to the caller, anywhere within the method. Alternatively, you could set a breakpoint inside a method.

Use the debugger to trace through the following programs. Show the value of each variable and the returned value in each program.

Write a recursive method for converting a string of digits into the integer it represents. For example, "12341" represents the integer 12341. Hint: Process the string left to right.

```
public int strToNum(String str) {
   if(str.length() < 1)
      return 0;
   else
      return ((str.charAt(str.length() - 1) - '0')
         + (10 * strToNum(str.substring(0, str.length() - 1))));
}
```

Write a recursive method that counts the number of nodes in a singly linked list.

```
private int countNodes(Node<T> trav){
   if(trav == null)
      return 0;
   return 1 + countNodes(trav.next);
}
```

Write a recursive method that finds the smallest integer value in an array of int..

```
public int findMin(int array[], int size, int index) {
   if(index == size - 1)
      return array[index];
   int result = findMin(array, size, index + 1);
   if (array[index] < result)
      return array[index];
   else
      return result;
}
```

Write a recursive method that determines if a string s is a palindrome, that is, it is equal to its reverse. For example, "racecar" is a palindrome.
Hint: **Be careful to return the correct value for both odd- and even-length strings.**

```
public boolean isPalindrome(String str, int low, int high) {
   if(high <= low)
      return true;
   else if (str.charAt(low)!= str.charAt(high))
      return false;
   else
      return isPalindrome(str,low+1,high-1);
}
```

Write a recursive method that takes a character string s and returns its reverse. So for example, the reverse of "**pots&pans**" would be "**snap&stop**". ***Hint*: Swap the first and last characters.**

```
public String reverseString(String s){
      if(s.length() == 0)
            return s;
      return reverseString(s.substring(1)) + s.charAt(0);
}
```

Write a recursive method to display the directories on your computer - **example: J:**.

```
public static void traverse(File file) {
      if(file.isDirectory()) {
            System.out.println(file);
            String dirContents[] = file.list();
            if (dirContents != null)
                  for (String directory : dirContents)
                        traverse(new File(file, directory));
      }
}
```

Towers of Hanoi (Iterative)

```
public static void hanoi(int discs){
      for (int x = 1; x < (1 << discs); x++) {
            int from = (x & x - 1) % 3;
            int to = ((x | x - 1) + 1) % 3;
            System.out.println("Move " + from + " to " + to);
      }
}
```

Recursion Problems - Write Recursive Methods for the Following:

1. Modify the method that calculates the sum of the integers between 1 and N shown below. Have the new version match the following recursive definition: The sum of 1 to N is the sum of 1 to (N/2) plus the sum of (N/2 + 1) to N. Trace your solution using an N of 7.

```
public static int sum(int num) {
   int result;
   if(num == 1)
         result = 1;
   else
         result = num + sum(num - 1);
   return result;
}
```

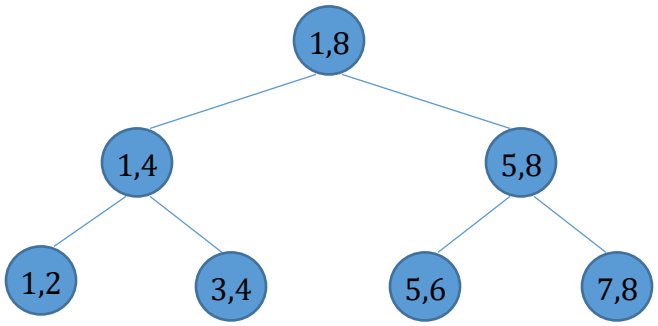


Figure 1 - Binary Tree of sum of the integers from 1 to 8

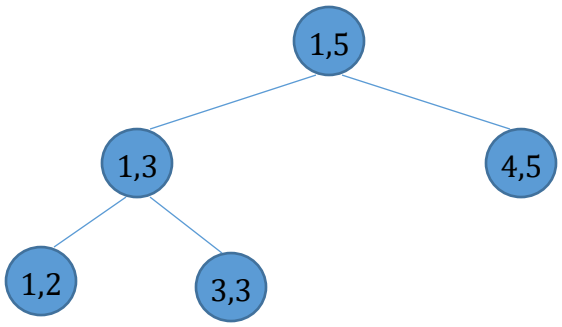


Figure 2 - Binary Tree of sum of the integers from 1 to 5

2. Modify the method, ***hanoi***, above, and produce an Excel chart showing the number of moves required to solve the Towers of Hanoi puzzle using the following number of discs: 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 30 and 31.
   What happens when you try a value over 30?
   How can the program be modified to speed up the execution?

3. Write a recursive definition of x^y, where x and y are integers and y >= 0. In addition, write the recursive method.

$$pow(x,y) = x^y = \begin{cases} 1 & if\ y = 0 \\ x * pow\left(x,(y-1)/2\right)^2 & if\ y > 0\ is\ odd \\ pow(x,y/2)^2 & if\ y > 0\ is\ even \end{cases}$$

4. Write a recursive method to display the contents (data) of a linked-list in reverse order.

5. Write a recursive method to convert a number, n, to a base, b, and return result as a String.

```
private static final char[] table =
      {'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};

System.out.println(convert(12,16));  // convert 12 to base 16 – result should be C
```

$2\overline{)12} = 6 \quad remainder \quad 0$

$2\overline{)6} = \ 3 \quad remainder \quad 0$

$2\overline{)3} = \ 1 \quad remainder \quad 1$

$2\overline{)1} = \ 0 \quad remainder \quad 1$

$12_{10} = 1100_2$

Binary – Base 2

$16\overline{)12} = 0 \quad remainder \quad C$

$12_{10} = C_{16}$

Hexadecimal – Base 16