Week 7 - Wednesday

COMP 2000

Last time

- What did we talk about last time?
- More recursion

Questions?

Project 2

Debugging

Debugging with print statements

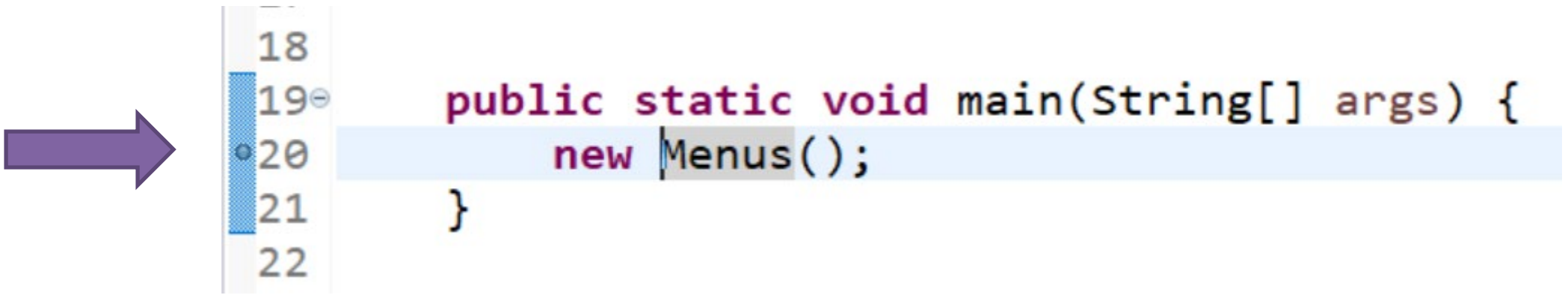
- When the output of your program isn't enough to tell you what the problem is, people sometimes use print statements
- By injecting print statements at key locations, you can see
  - If certain points in your code are reached
  - What the values of variables are
- Debugging with print statements can be useful and is nearly always available
- There are even libraries that allow you to turn off the statements when you're no longer debugging

Debugging with an IDE

- In modern times, there are more powerful (and easy to use) tools
- Most modern IDEs, including Eclipse, have tools for debugging code
- You can:
  - Set a breakpoint (where execution will pause)
  - Step over lines of code
  - Step into methods
  - Examine the values of variables
  - Much more!

Set a breakpoint

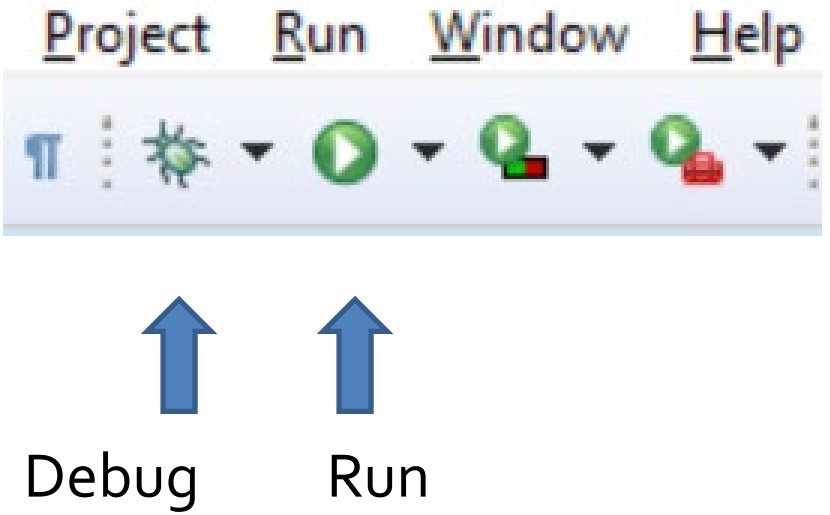
- Programs run at blinding speed, so you have to pause them to get information
- Setting a breakpoint is done in most debugging systems so that you can make the program pause where you want
- In Eclipse, the easiest way to do this is to double-click on the narrow column just left of the line numbers
- Doing so will create a breakpoint, shown with a blue dot



- You can also choose **Toggle Breakpoint** from the **Run** menu

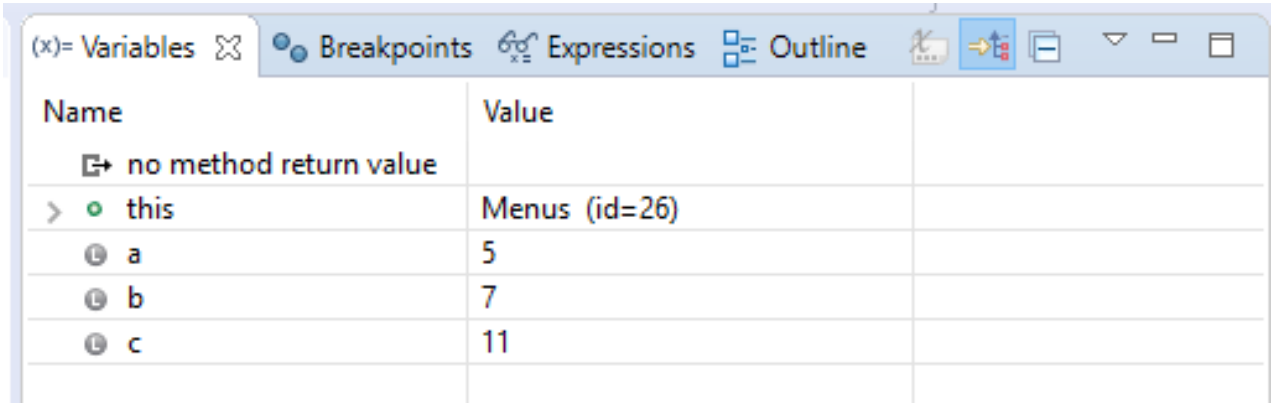
Debug mode

- After you set one or more breakpoints and run your program…
  - Nothing changes!
- In order for execution to pause at breakpoints, you have to run the program in debug mode
- In Eclipse this is done by clicking on the bug icon (instead of the green circle with the white triangle)
- Or you can right click on the .java file you want to debug and choose **Debug As… Java Application** instead of **Run As… Java Application**
- Debugging changes you over to the Debug Perspective, reorganizing Eclipse's panes slightly
- You might get a message about that



Inspecting variables

- Now that you've paused execution, you can look at the value of local and member variables
- The Variables tab will show you the value of local variables
- It will also give you **`this`**, which allows you to explore the object you're inside of
- Note that tabs can be moved around in Eclipse, so it might not be where you expect it
- If it's hidden, you can go under **Window > Show View > Variables**
- If you hover over a variable in code that's in scope, it will also show you its value



Step Over

- Are you stuck at the point of code where the breakpoint is?
- No!
- You can **Step Over** the next line of code, executing it
- This option is listed in the Run menu, but you usually Step Over code over and over, so use the F6 hotkey
- If the line of code you're Stepping Over is a method, it will call the method and return, not go into the method
  - Unless there's a breakpoint inside the method

Step Into

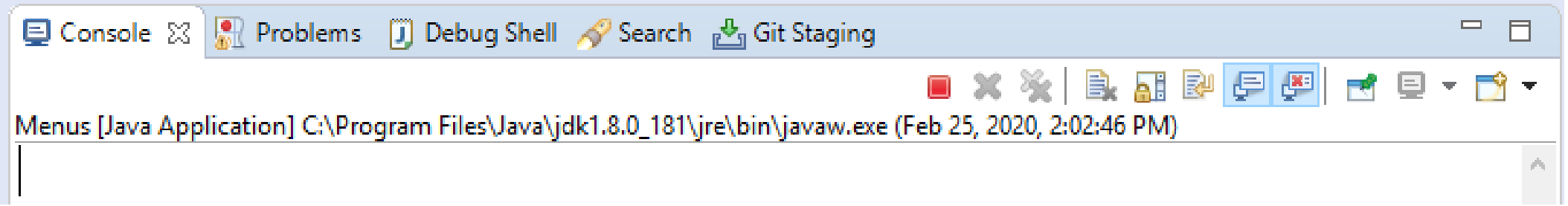
- On the other hand, what if you want to step into a method and see what's going on there?
- That's what **Step Into** is for
- Step Into (hotkey F5) will go inside of a method call instead of stepping over it
- Gotchas:
  - Sometimes there are multiple method calls in a single line: you'll step into the first (even if that's not what you wanted)
  - If the method is library code that your system doesn't have source code for, you won't be able to see anything

Step Return

- Sometimes you've gone into a method and have seen what you need to see
- Rather than using Step Over to get through all the code in the method, you can immediately return using Step Return (hotkey F7)
- Note that these hotkeys might vary from OS to OS, but they're always listed in the **Run** menu

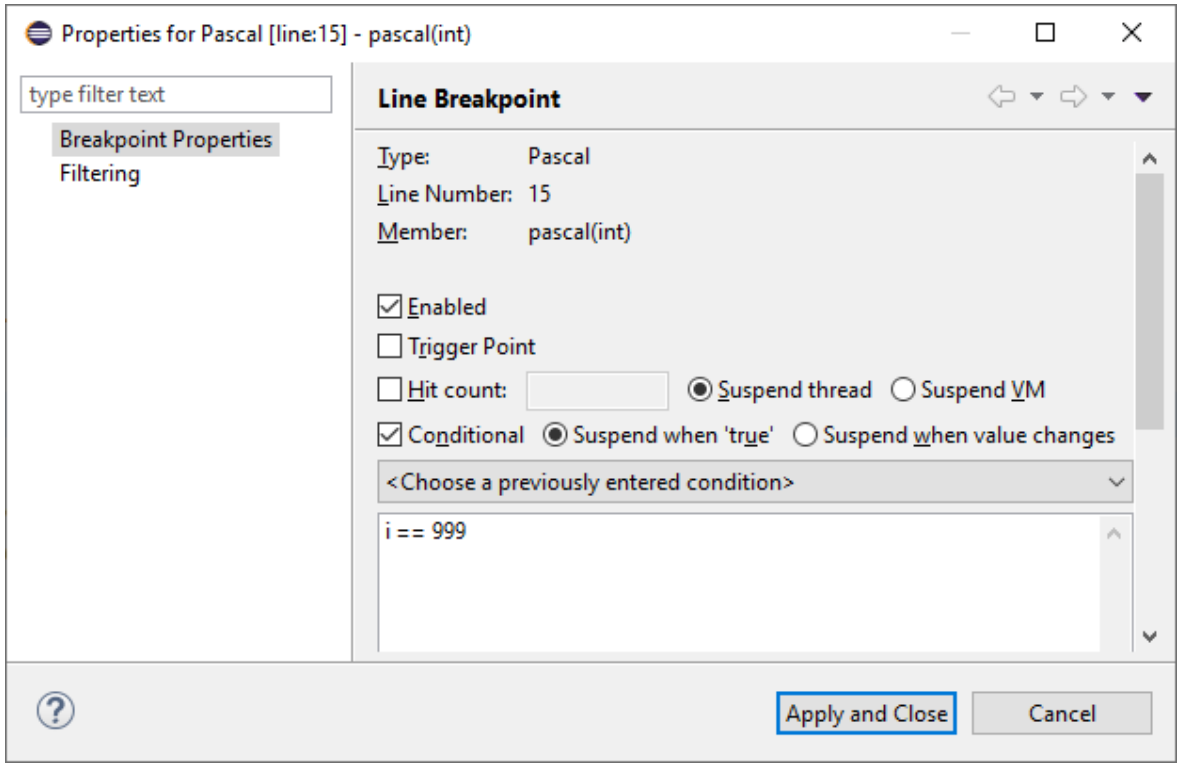
Resume

- And if you're really tired of going through code line-by-line, you can **Resume** execution
- The Resume command has hotkey F8
- Execution will continue until you hit another breakpoint
- You can also Terminate the program by selecting the Terminate option from the Run menu or hitting the little red square on the Console or on the ribbon



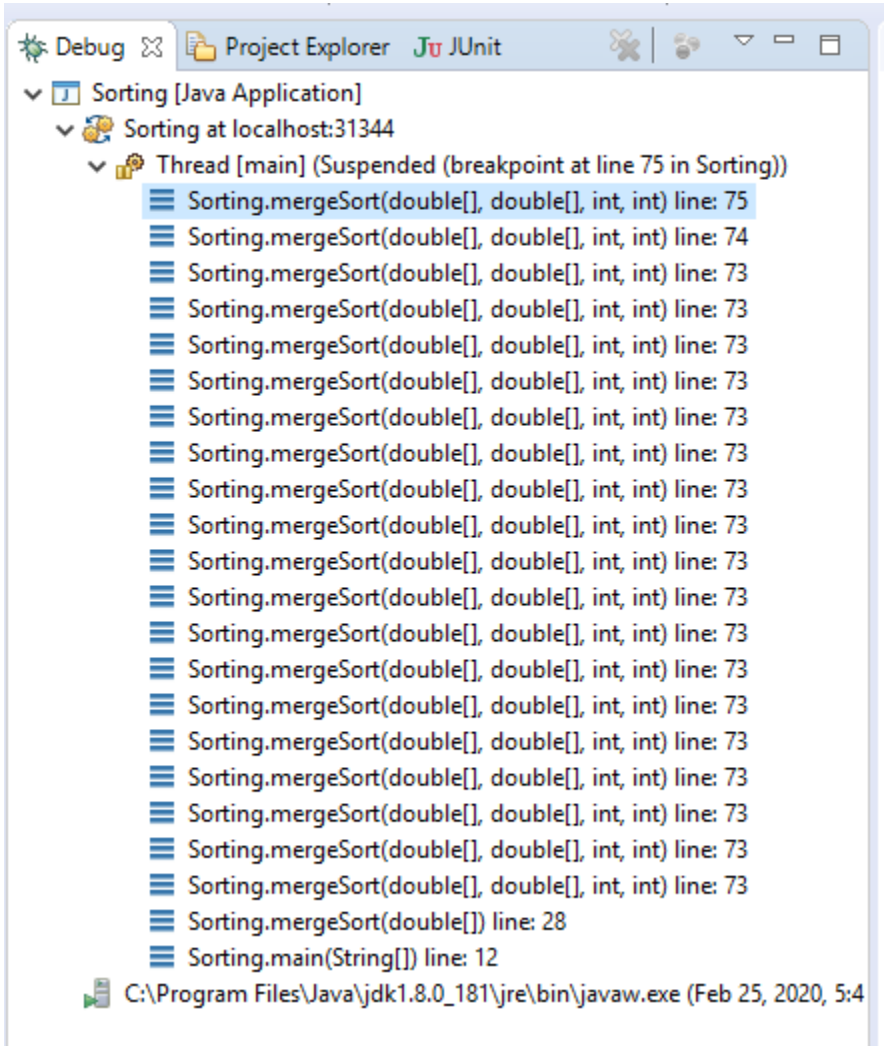
Conditional breakpoints

- What if you want to figure out a problem in a loop…
- But the problem happens after the loop runs 999 times
- Well, it would be infuriating to try to Step Over or Resume code 999 times
- Breakpoints can be **conditional**, meaning that they only pause code under certain circumstances
- Simply right-click on the breakpoint and choose **Breakpoint Properties…**
- You can add a condition or a hit count or other properties



Moving through the stack

- Whether doing recursion or not, many methods might be on the stack
- The currently executing line is only in one location
- However, the whole stack is active
- Using the Debug pane, you can jump around to different places on the stack and see what the values of local variables are



Strange stuff

- Using the debugger, you can *change* variables as well as inspecting them
- You shouldn't do that often, but it sometimes lets you test some weird case that's otherwise hard to achieve

Debugging example

- Let's use the debugger on the following method that's supposed to reverse an array and see what the problems are

```
public void reverse(int[] array) {
    for(int i = 1; i < array.length; ++i) {
        int temp = array[i];
        array[i] = array[array.length - i];
        array[array.length - i] = temp;
    }
}
```

Tower of Hanoi

Tower of Hanoi

- The Tower of Hanoi is a mathematical puzzle invented by the mathematician Édouard Lucas in the 19th century
- It is a board with three rods
- On the first rod sits a stack of *n* disks in increasing order of size, with the smallest disk on the top
- The goal is to move all of the disks to the third rod
- There are three rules:
  1. You can only move one disk at once
  2. Each move takes the top disk from one rod and puts it on the top of another (possibly empty) stack on another rod
  3. No larger disk may be placed on top of a smaller disk

Solving Tower of Hanoi

- Professor Stucki has a wooden set you can play with
- It's fun to move the disks around, but how can we come up with an algorithm that solves the problem?
- Recursion!

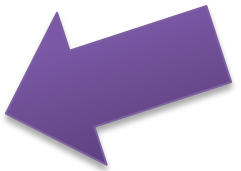
Recursive solution

- Base case (***n*** = 1):
  - If there is only one disk, move it to its destination
- Recursive case (***n*** > 1):
  - First move ***n*** – 1 disks to a temporary pole
  - Then move the n^{th} disk to the destination
  - Then move ***n*** – 1 disks from the temporary pole to the destination

Tower of Hanoi code

```
public static void hanoi(int n, char from, char to,
  char temp) {

  if( n == 1 )
     System.out.println("Move disk from " + from +
          " to " + to);
  else {
     hanoi(n - 1, from, temp, to);
     hanoi(1, from, to, temp);
     hanoi(n - 1, temp, to, from);
  }
}
```



Base Case

Recursive Case

Lessons from Tower of Hanoi

- The recursion is pretty interesting
- You can prove that there's no faster way to do it than the given approach
- But it's very slow!
- 100 disks would take longer than the Universe has been in existence, even on the faster modern computers
- How can we understand how long recursion takes?
- Take COMP 2100 and COMP 4500 to find out!

Quiz

Upcoming

Next time…

- *n* queens
- Mergesort

Reminders

- Keep reading Chapter 19
- Finish Project 2
  - Due Friday