Debugging

CMSC 202

Overview

- Debugging
- Error Types
- Stack Traces
- Probing
- Eclipse debugger

Debugging

- Debugging is a ***methodical process*** of finding and reducing the number of bugs, or defects, in a computer program or a piece of electronic hardware, thus making it behave as expected.
- Debugging tends to be ***harder when*** various subsystems are ***tightly coupled***, as changes in one may cause bugs to emerge in another.

—Wikipedia

Error/Bug Types

- Compile time errors
  - Bugs caught by compiler
    - Syntax errors
- Runtime errors that terminate program execution
  - Bugs caught by the runtime system
    - NullPointerException
    - ArrayIndexOutOfBoundsException
- Runtime errors that do not terminate the program
  - Bugs not caught by the runtime system, but hopefully caught by developer
    - Logic errors

Stack Trace

- A stack trace is a dump of the active stack frames at a given point in execution time.
- In Java, when the JVM detects an error condition (such as trying to invoke a method on a null reference), it raises an ***exception*** resulting in a stack trace.
- This stack trace shows you where the error originated from and how it came to be executed.

Reading a Stack Trace

Exception

Error Message

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at Bar.derp(Bar.java:8)
    at Foo.herp(Foo.java:6)
    at Test.main(Test.java:7)
```

Stack trace

The source of the error is at the top of the stack trace

Errors Inside a Java-Provided Class

```
Exception in thread "main" java.lang.IndexOutOfBoundsException: Index: 12, Size: 0
    at java.util.ArrayList.RangeCheck(ArrayList.java:547)
    at java.util.ArrayList.remove(ArrayList.java:387)
    at Bar.derp(Bar.java:12)
    at Foo.herp(Foo.java:6)
    at Test.main(Test.java:7)
```

Sometimes errors originate in a class that's provided by Java.

This is where the error manifested itself, though the cause is almost always in your code up the stack.

Scan from the top down looking for the first reference to your code. That's usually a good place to start looking.

Tracing with Print Statements

- Print (a.k.a. *tracing*, *probing*) debugging is the act of watching (live or recorded) trace statements, or print statements, that indicate the flow of execution of a pr.ocess

—Wikipedia

Tracing with Print Statements

- Once you've identified the location of the error (by reading the stack trace), start printing out variables.
- This can be as basic as simply printing out all local variables, members, objects, parameters, etc. using `System.out.println()`.
- Having a working `toString()` method for all of your objects really aids in this debugging process.

Debugger

- A special program used to find errors (bugs) in other programs.
- A debugger allows a programmer to stop a program at any point and examine and change the values of variables.

—Webopedia

Eclipse Debugger

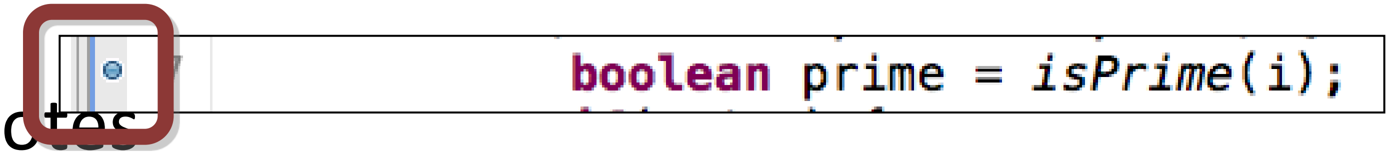
- Eclipse has a built-in perspective that is dedicated to debugging Java code.
- To run a program in the debugger, simply right click on the class to run and select…
  - Debug As → Java Application
- Allow Eclipse to open the Debug perspective if it asks.
- If you do nothing else, Eclipse will simply run your program just like a normal “Run As”.

Breakpoints

- Breakpoints can be used to pause your program at a certain point.
- Once paused, you can examine (and even change) the state of variables.
- There are many different ways to break…
  - Line
  - Method
  - Member change
  - Etc.

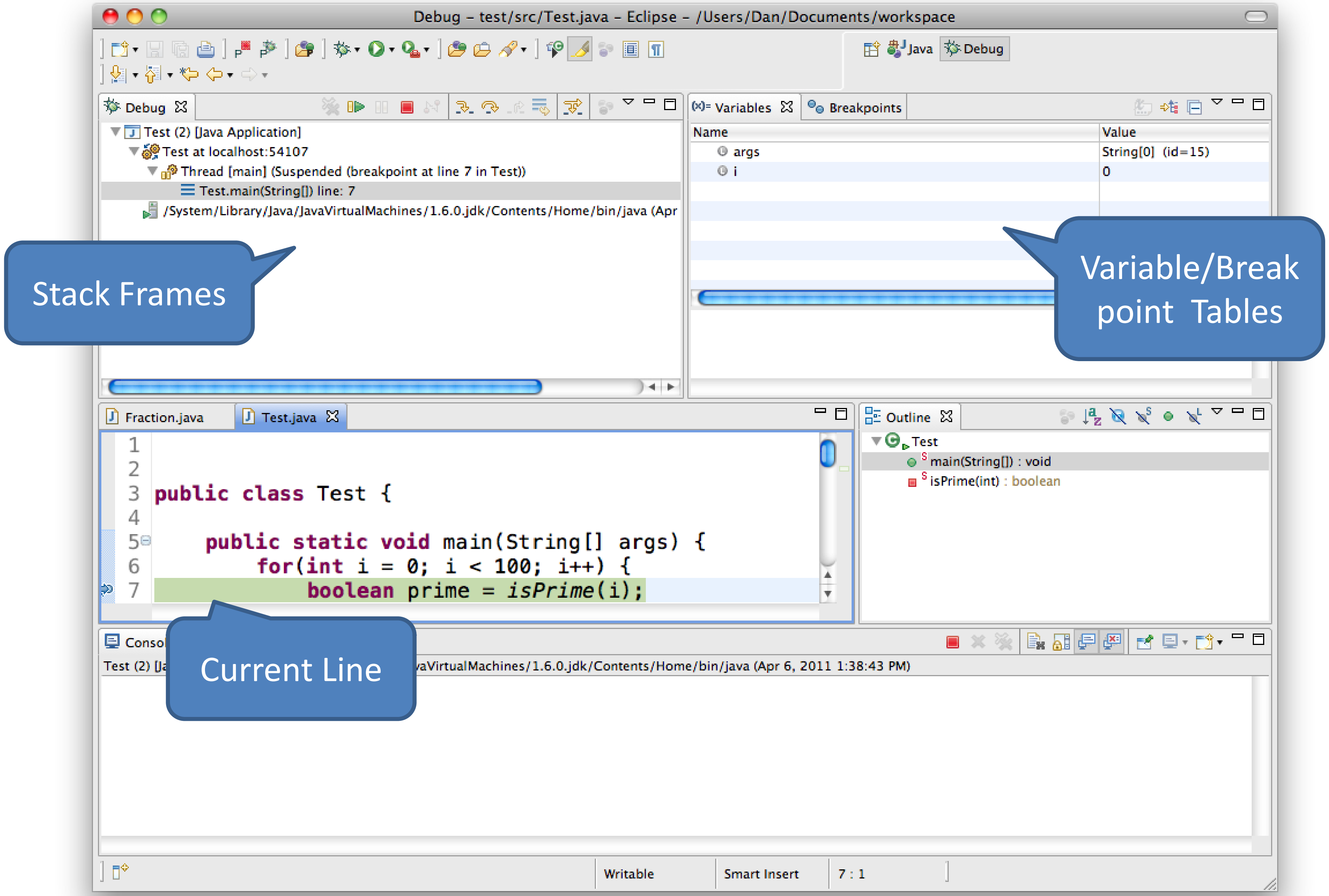
Line Breakpoints

- To set a breakpoint on a line, simply double click in the gutter left of the line to stop on.
- Once you do so, you'll see a small blue bubble in the gutter like so…



```
boolean prime = isPrime(i);
```

- Notes
  - Simply double click again to toggle the breakpoint off.
  - In order to break on a line, there must be an executable statement (e.g. you cannot break on a curly brace).



Debug – test/src/Test.java – Eclipse – /Users/Dan/Documents/workspace

```
Test (2) [Java Application]
  Test at localhost:54107
    Thread [main] (Suspended (breakpoint at line 7 in Test))
      Test.main(String[]) line: 7
  /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (Apr
```

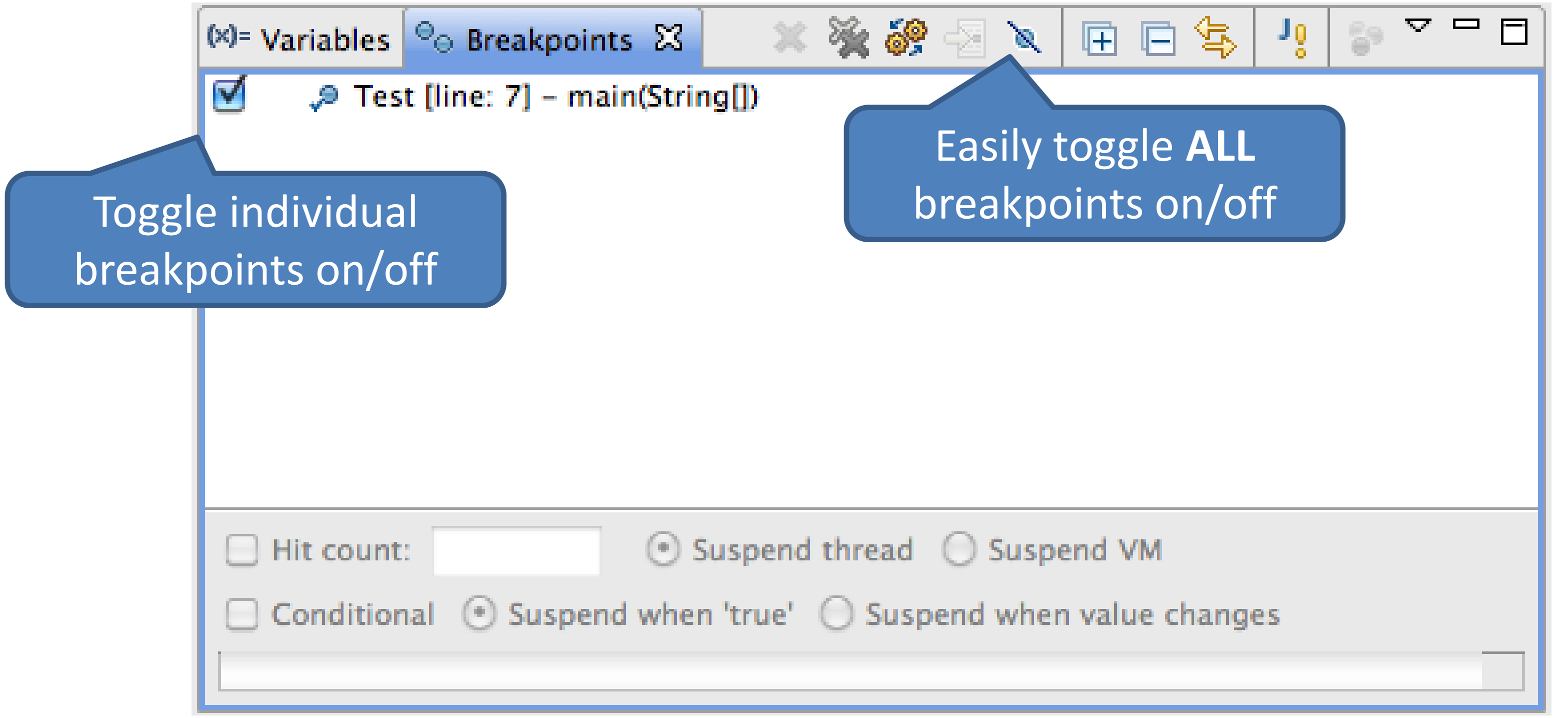
Name	Value
args	String[0] (id=15)
i	0

```
1
2
3  public class Test {
4
5      public static void main(String[] args) {
6          for(int i = 0; i < 100; i++) {
7              boolean prime = isPrime(i);
```

```
Test
  main(String[]) : void
  isPrime(int) : boolean
```

```
Test (2) [Ja…vaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (Apr 6, 2011 1:38:43 PM)
```

Breakpoint View

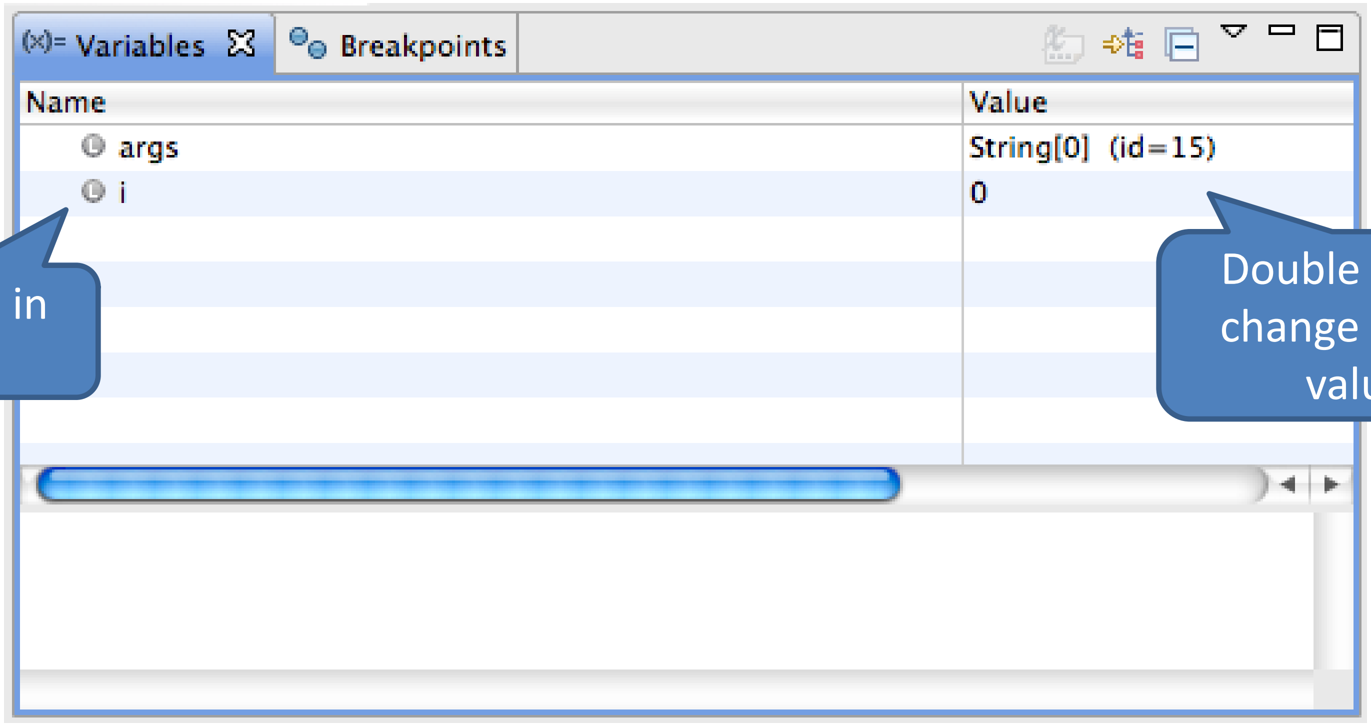

(x)= Variables
Breakpoints
Toggle individual breakpoints on/off
Easily toggle **ALL** breakpoints on/off

☑ Test [line: 7] - main(String[])

☐ Hit count: ◉ Suspend thread ○ Suspend VM

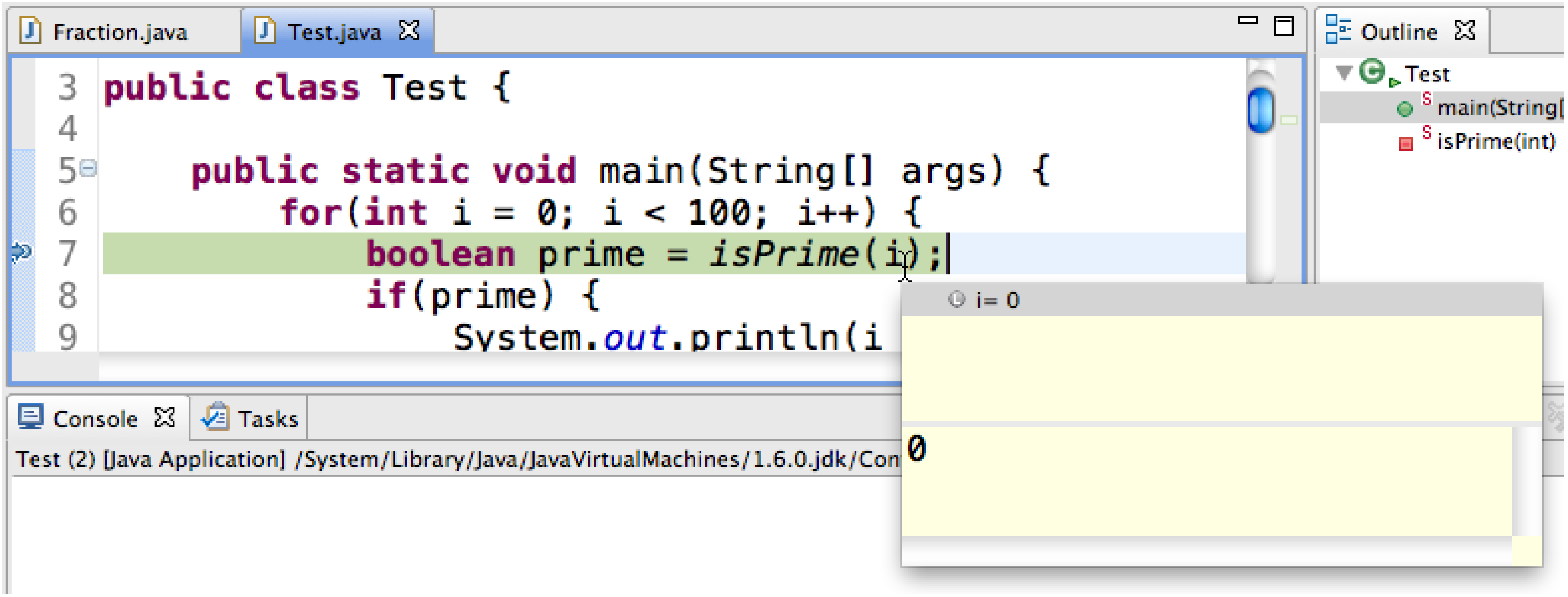
☐ Conditional ◉ Suspend when 'true' ○ Suspend when value changes

Variable View



Name	Value
args	String[0] (id=15)
i	0

Hover to View Variable Values



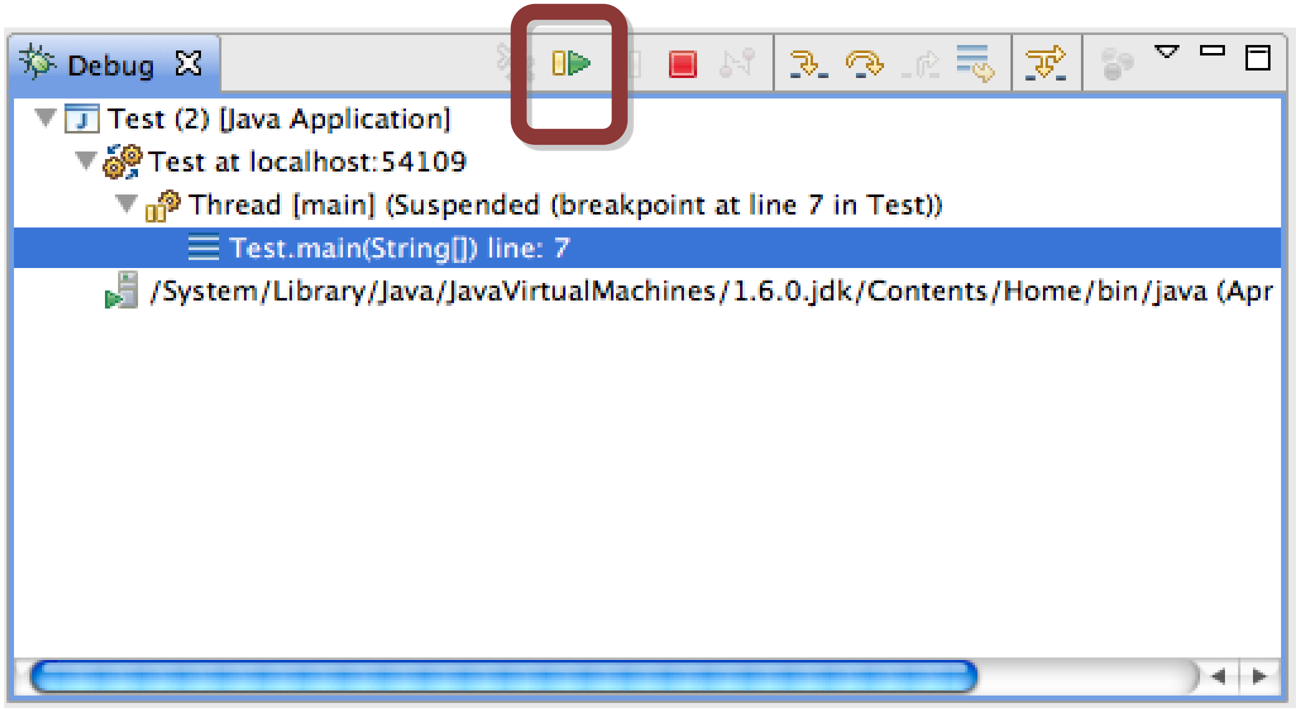
```
public class Test {

    public static void main(String[] args) {
        for(int i = 0; i < 100; i++) {
            boolean prime = isPrime(i);
            if(prime) {
                System.out.println(i
```

```
Test (2) [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Con
```

```
0
```

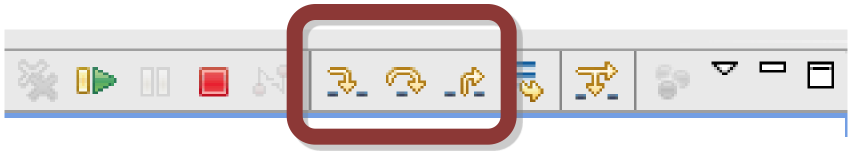
Continuing Execution

- If you click the resume button in the Debug view, execution will continue until the next breakpoint is encountered.



```
Test (2) [Java Application]
  Test at localhost:54109
    Thread [main] (Suspended (breakpoint at line 7 in Test))
      Test.main(String[]) line: 7
  /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (Apr
```

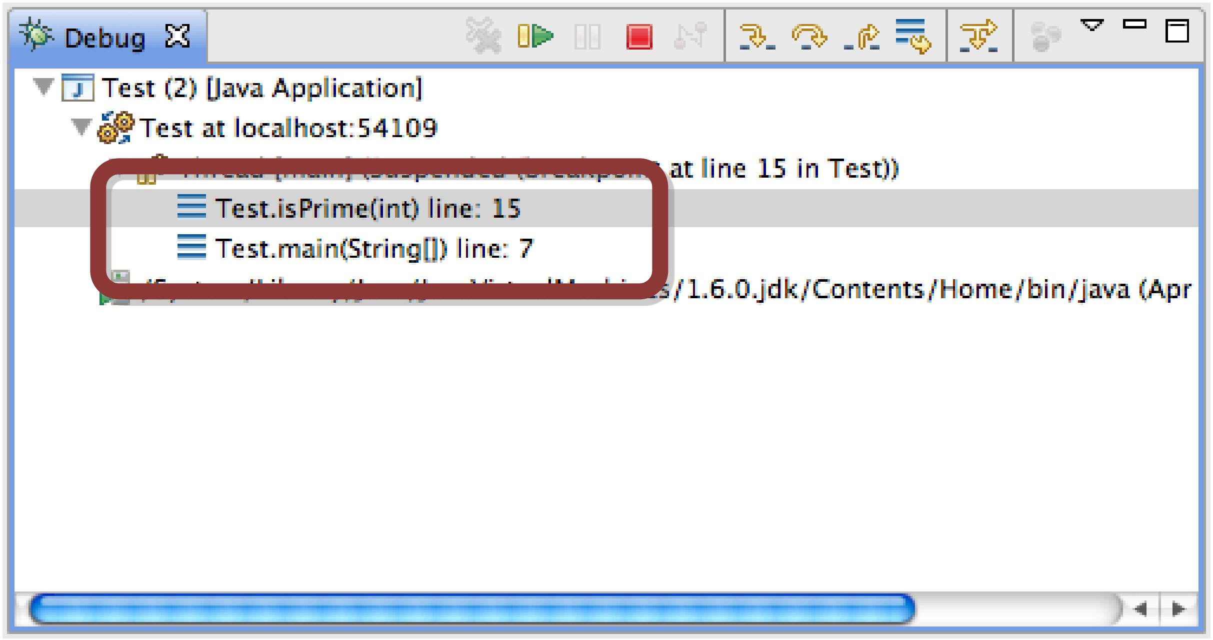
Stepping



- There are several step buttons that allow you to walk through the execution of your code.
  - Step Into
    - If the line contains a method call, step into that method and pause execution.
  - Step Over
    - Completely execute this line (including any method calls) and pause execution at the next line.
  - Step Return
    - Complete the current method and pause execution where the method was called from.

Stack Frames

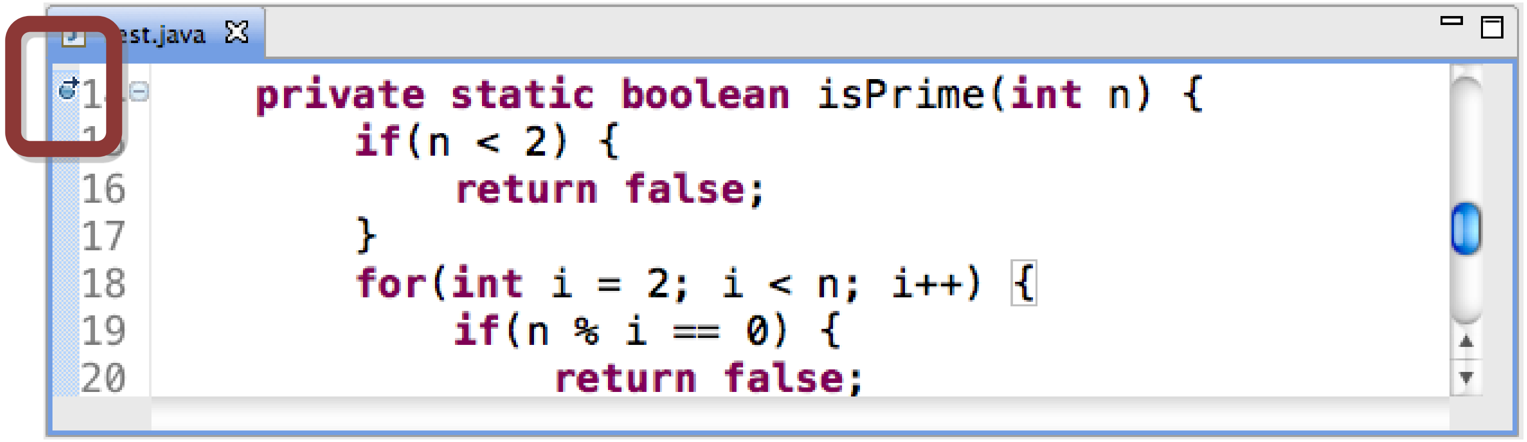
- Eclipse's Debug view also shows you stack frames so you can see how you got somewhere.
  - Current stack frame is at the top, main should be at the bottom



```
Test (2) [Java Application]
  Test at localhost:54109
    at line 15 in Test))
      Test.isPrime(int) line: 15
      Test.main(String[]) line: 7
    s/1.6.0.jdk/Contents/Home/bin/java (Apr
```

Method Breakpoints

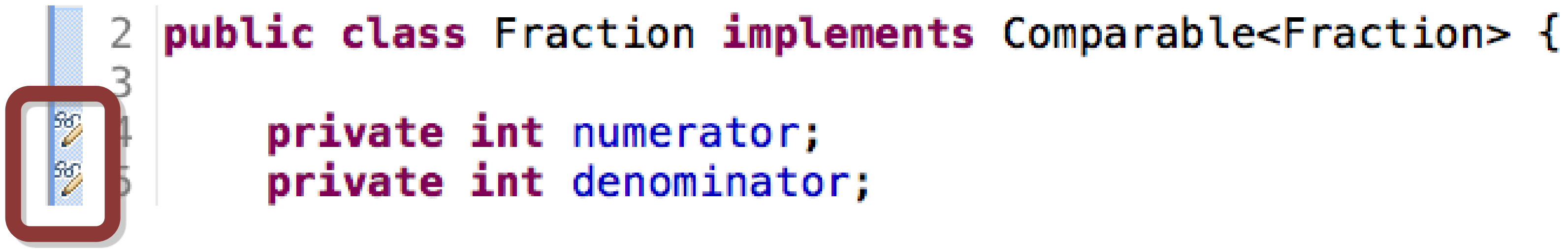
- Double clicking on the margin next to method will create a method entry breakpoint.
  - Right click → Breakpoint Properties… allows you to also set exit breakpoint.



```
    private static boolean isPrime(int n) {
        if(n < 2) {
            return false;
        }
        for(int i = 2; i < n; i++) {
            if(n % i == 0) {
                return false;
```

Watching Members

- You can also set breakpoints (also called *watch points*) to see when a member is being accessed or changed.
- Simply double click next to the member and it will set both breakpoints.
  - Double click again to toggle off
  - Right click → Breakpoint Properties... to change



```
public class Fraction implements Comparable<Fraction> {

    private int numerator;
    private int denominator;
```

Additional References

- Java Logging Overview
  - http://download.oracle.com/javase/1.5.0/docs/guide/logging/overview.html
- Lars Vogel’s Java Logging API Tutorial
  - http://www.vogella.de/articles/Logging/article.html
- Lars Vogel’s Java Debugging with Eclipse Tutorial
  - http://www.vogella.de/articles/EclipseDebugging/article.html