# Testing and Debugging

Professor Frank Kreimendahl

School of Computing and Data Science

Wentworth Institute of Technology

September 21, 2022

## Testing

## Levels of Testing

- Programs are difficult to write correctly
- Rather than guess or try just a few inputs, build methodical tests for each part of a program
- Three levels of testing in software:
  - Unit testing – verifying a single class or method
  - Integration testing – testing interactions between classes or multiple methods in a class
  - System testing – testing the entire program as if an end user was running it

## Types of Testing

- Black-box testing:
  - Testing a class based on its interface and specification
  - Many valid inputs are tested for correct behavior
  - Invalid inputs are also tested for correct exceptions/errors
- White-box testing:
  - Testing a class with knowledge of its implementation
  - Attempt to give inputs that cover every possible path of execution as possible
  - Ensure that every line of code runs at least once

## Testing Code Paths

### Consider inputs to all paths in the method

```
public void testMethod(char a, char b) {
  if (a < 'M') {
    if (b < 'R')
      System.out.println("path 1");
    else
      System.out.println("path 2");
  }
  else {
    if (b < 'H')
      System.out.println("path 3");
    else
      System.out.println("path 4");
  }
}
```

## Input Values

How many runs are required to test all paths?

What values can we use for `a` and `b`?

## Input Values

How many runs are required to test all paths?

What values can we use for `a` and `b`?

## Preparations for Code Testing

- Testing is an integral part of writing code, not just an afterthought
- Writing unit tests can verify that each part of your program works as expected
- Tests can also verify that code is still working after you have changed parts of your program
- Considering tests makes you consider edge cases and write better code

## Black-Box Code Testing

- Test all valid inputs
- Test invalid inputs
- For each test, specify or otherwise calculate expected results

## White-Box Code Testing

- Test every branch of code
- Test every possibility for switch statements
- Test loops for three possibilities: run 0 times, run 1 time, run multiple times
- Test that loops will terminate

## Test Boundary Conditions of Code

- Special cases on the cusp of two different behaviors
- Examples from double-linked lists:
  - Inserting at the beginning/end
  - Setting at the end/past the end
  - Operations on size 0 or size 1 lists
- General cusps:
  - Inputs at extremes of allowed values
  - Operations at extremes of a data structure
  - Extremes of number of inputs or data structure sizes

## Debugging

## Debugging

- Debugging is like detective work
- Debugging requires three steps to succeed:
  1. Understanding how your code should behave
  2. Determining where your code behaves in an undesired manner
  3. Changing your code so it behaves correctly
- There are several strategies for step 2

## Example Buggy Code

```
public static String getSentence() {
  Scanner in = new Scanner(System.in);
  StringJoiner stj = new StringJoiner(" ");
  int count = 0;
  while (count < 10) {
    System.out.println("Enter word or ** to quit");
    String word = in.next();
    if (word == "**") break;
    stj.add(word);
    count++;
  }
  return stj.toString();
}
```

## Sentinels, or `printf` debugging

- The read loop should stop when someone enters “**”, but it doesn’t
- We can write a print statement when we add a word to see if words are being read correctly:

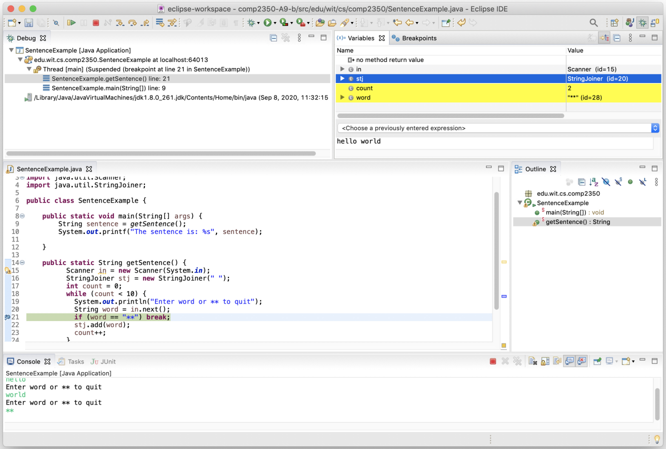
  -
    ```
    System.out.println("!!! Next word is " + word +
                                  ", count is " + count);
    ```
- We will see that words are read in correctly, but the loop doesn’t break when “**” is encountered
- Java’s == operator compares addresses of Objects like Strings, not values

This form of debugging works great on small projects and projects where it’s easy to print out the current data.

## Debugging Software

- A debugger is a piece of software that gives you control over your program's execution
- Most modern IDEs include a debugger
- Debuggers allow you to:
  - Execute code one line at a time
  - Set a breakpoint – a line of code that pauses your program's run
  - View the values of all variables in memory
  - Many, many more features…

# Eclipse Debugger


eclipse-workspace - comp2350-A9-b/src/edu/wit/cs/comp2350/SentenceExample.java - Eclipse IDE
Debug
Variables
Breakpoints
SentenceExample.java
Outline
Console
Tasks
JUnit

```
SentenceExample [Java Application]
  edu.wit.cs.comp2350.SentenceExample at localhost:64013
    Thread [main] (Suspended (breakpoint at line 21 in SentenceExample))
      SentenceExample.getSentence() line: 21
      SentenceExample.main(String[]) line: 9
  /Library/Java/JavaVirtualMachines/jdk1.8.0_261.jdk/Contents/Home/bin/java (Sep 8, 2020, 11:32:15
```

| Name | Value |
|---|---|
| no method return value | |
| in | Scanner  (id=15) |
| stj | StringJoiner  (id=20) |
| count | 2 |
| word | "**" (id=28) |

```
<Choose a previously entered expression>
hello world
```

```java
import java.util.Scanner;
import java.util.StringJoiner;

public class SentenceExample {

    public static void main(String[] args) {
        String sentence = getSentence();
        System.out.printf("The sentence is: %s", sentence);

    }

    public static String getSentence() {
        Scanner in = new Scanner(System.in);
        StringJoiner stj = new StringJoiner(" ");
        int count = 0;
        while (count < 10) {
          System.out.println("Enter word or ** to quit");
          String word = in.next();
          if (word == "**") break;
          stj.add(word);
          count++;
        }
```

```
edu.wit.cs.comp2350
  SentenceExample
    main(String[]) : void
    getSentence() : String
```

```
SentenceExample [Java Application]
hello
Enter word or ** to quit
world
Enter word or ** to quit
**
```

Debugger interface built into Eclipse