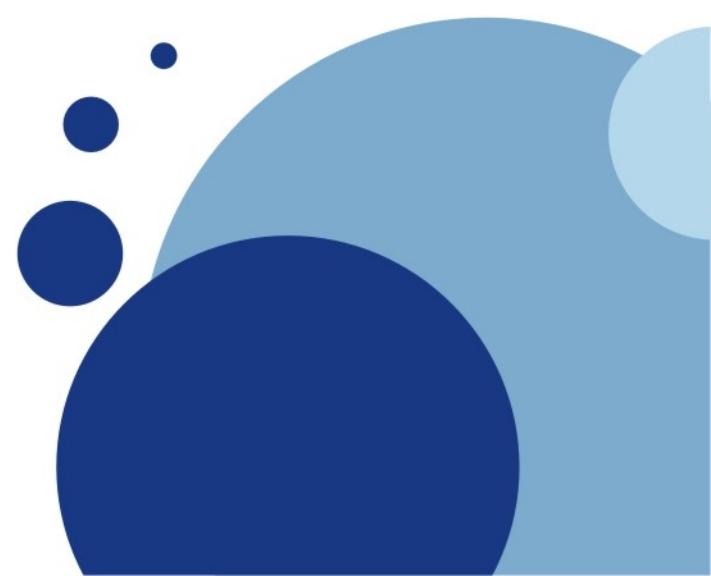


# **GEOG 178/258**

## **Week 2:**

### Variables, Debugging, and Loops

*mike johnson*



# OVERVIEW

Week

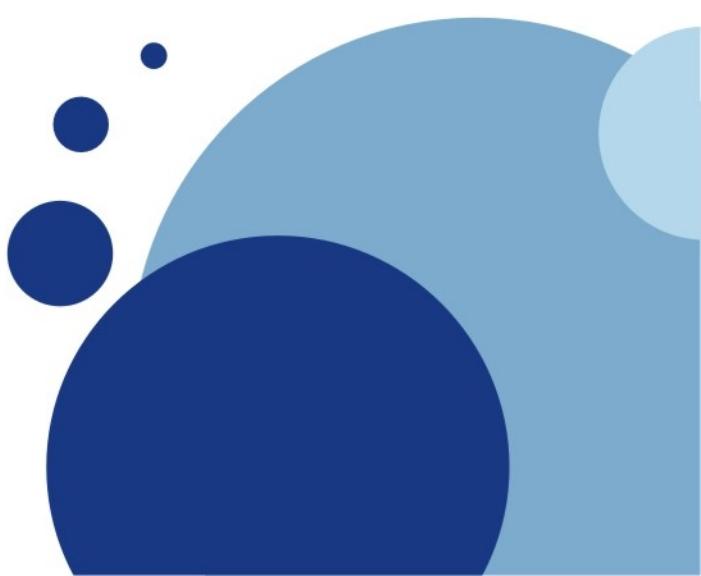
2

## Contents

*\*\*Click on titles to jump to the corresponding section:*

1. [Variables and their primitive types](#)
2. [Practice problems to declare, manipulate and print variables](#)
3. [Learn to import an existing program file](#)
4. [Launch and navigate the Eclipse Debugger](#)
5. [Look at the syntax and logic of the for and while loop](#)

# 1. Variables



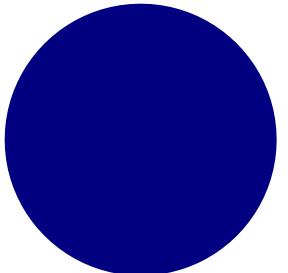
# What are Variables??

Week

2

## Variables

- Variables represent space reserved in memory to store values
  - So, creating a variable is reserving a set amount of memory space, and defining what can be stored there...
- Every variable is made up of three components:
  - (1) A **type** – i.e. how much memory to save
  - (2) A **name** – i.e. what it's called (human reference)
  - (3) A **value** – what it represents or is equal to
- An example: `int x = 100;`
- Here we are creating an **integer value** called **x** that is equal to **100**



# Primative Variable Types

Week

2

## Variables

- In Java there are 8 type of primitive variables
- Each of these reserves a different length of space in memory AND allows different types of data to be stored.
- These are predefined by Java and are represented by a key word type:
  1. Byte
  2. Short
  3. Int
  4. Long
  5. Float
  6. Double
  7. Char (character)
  8. Boolean (true/false)

# Variable Types

Week

2

## Variables

### 1. Byte

- 8-bit signed two's complement integer
- Minimum value: -128 ( $-2^7$ )
- Maximum value: 127 (inclusive) ( $2^7 - 1$ )
- Default value is 0
- Byte data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an integer.

### 2. Short

- 16-bit signed two's complement integer
- Minimum value: -32,768 ( $-2^{15}$ )
- Maximum value is 32,767 (inclusive) ( $2^{15} - 1$ )
- Short data type can also be used to save memory as byte data type.
- A short is 2 times smaller than an integer
- Default value is 0.

# Variable Types

Week

2

## Variables

- Int

- 32-bit signed two's complement integer.
- Minimum value is - 2,147,483,648 ( $-2^{31}$ )
- Maximum value is 2,147,483,647(inclusive) ( $2^{31} - 1$ )
- Integer is generally used as the default data type for integral values unless there is a concern about memory.
- The default value is 0

- Short

- 64-bit signed two's complement integer
- Minimum value is -9,223,372,036,854,775,808( $-2^{63}$ )
- Maximum value is 9,223,372,036,854,775,807 (inclusive)( $2^{63} - 1$ )
- This type is used when a wider range than int is needed
- Default value is 0L

# Variable Types

Week

2

## Variables

- **Float**

- Single-precision 32-bit IEEE 754 floating point
- Float is mainly used to save memory in large arrays of floating point numbers
- Default value is 0.0f
- Float data type is never used for precise values such as currency

- **Double**

- Double-precision 64-bit IEEE 754 floating point
- This data type is generally used as the default data type for decimal values, generally the default choice
- Double data type should never be used for precise values such as currency
- Default value is 0.0d

# Variable Types

Week

2

## Variables

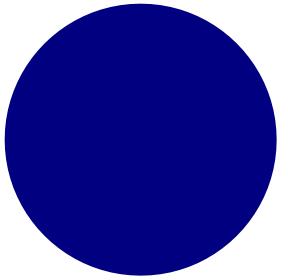
- Boolean

- One bit
- Two possible values: true (1) and false (0)
- This data type is used for simple flags that track true/false conditions
- Default value is false

- Char

- Single 16-bit Unicode character
- Minimum value is '\u0000' (or 0)
- Maximum value is '\uffff' (or 65,535 inclusive)
- Used to store any SINGLE character
- **A variable type 'String' must be used to store multiple characters**

# 2. Examples



# Download / Load Sample Code for this week

Option 1) If you have cloned the classes repo, be sure to fetch the new data

```
> cd ... working directory....          ## Enter the location you want the repo to go  
> git clone https://github.com/mikejohnson51/geog178.git ## Clone (copy the repo) into that location  
> cd ./geog178.                                ## Enter the new geog178 folder (your local repo)  
> git pull origin                            ## Pull new files from the origin page
```

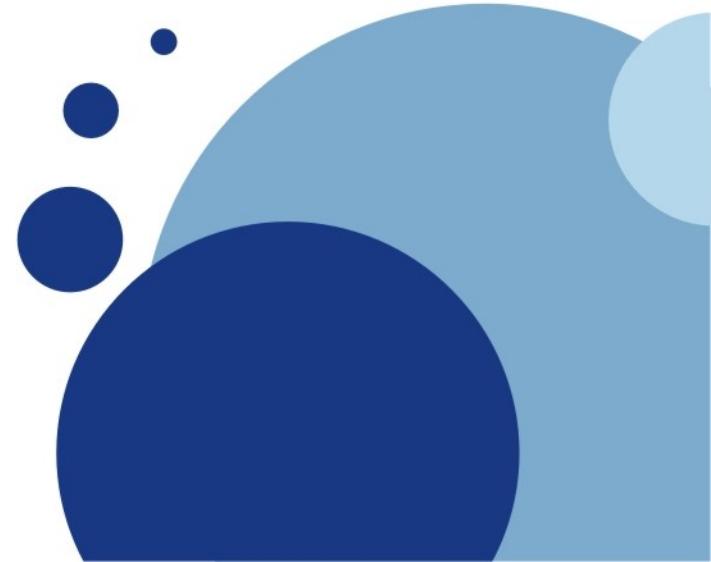
Option 2) Download the zip file from the course page

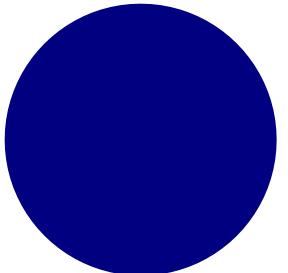
Week 2: OGC, Variables, Debugging, Loops

Section slides: OGC Simple Features

Section slides: Variable, Debugging, Loops

Example Code





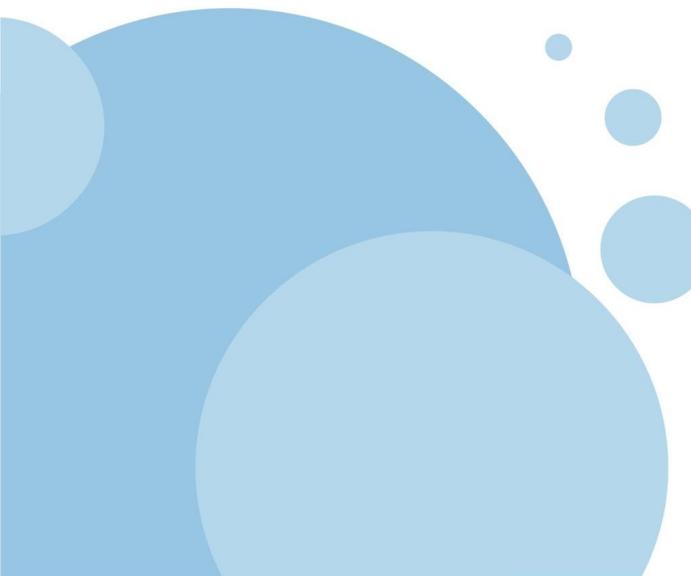
# Importing an Existing Project

Week

2



## Debugging



- Open a workspace on your flash drive or local desktop in Eclipse
- Go to: File → Import → General → Existing
- Select "Select root directory"
- Click 'Browse'
- Point it to the 'Week2\_examples' folder on your desktop
- Click 'Finish'

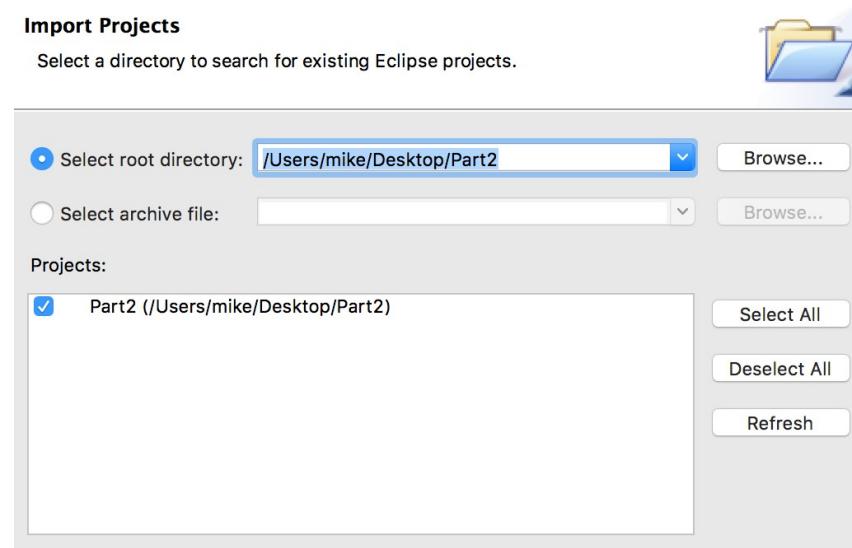
# Importing an Existing Project

Week

2

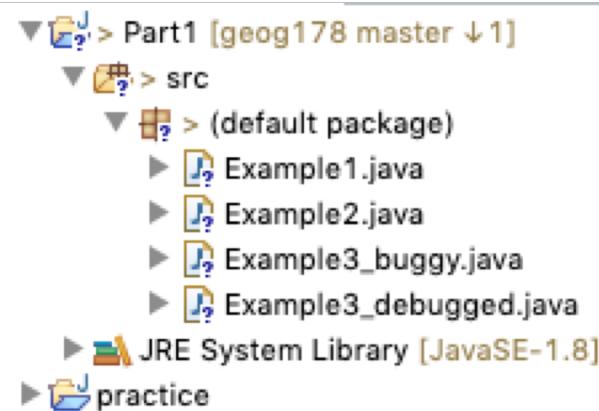
## Debugging

- Select "Select root directory"
- Click 'Browse'
- Point it to the downloaded folder on your desktop



- Click 'Finish'

- Under the src folder of the new project you should see the examples for today. Don't open them yet!!



# Where is UCSB (simple program)

Week

2

## Example #1

- Using what we now know about variables write a program that prints the following statement using variables and comments.

UCSB is located at 34.4139 degrees latitude and -119.8489 degrees longitude.

- In this program make location name, lat and long variables to be printed.
- (Answer on the next slide and in Example1.java)

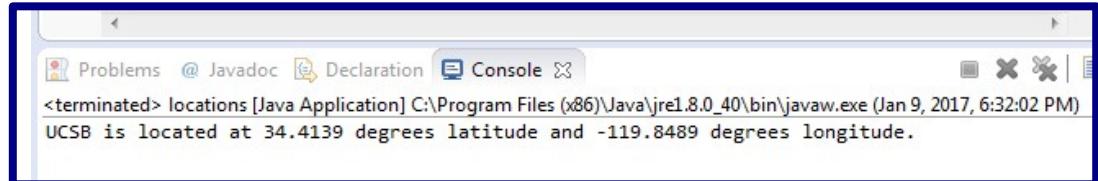
# Where is UCSB (simple program)

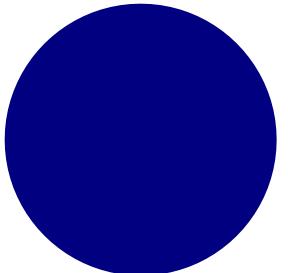
Week

2

## Example #1

```
public class locations {  
  
    public static void main(String[] args) {  
        // Location of interest given as a String variable  
        String loc1 = "UCSB";  
  
        // The latitude of Location 1 given as a double variable  
        double lat1 = 34.4139;  
  
        // The longitude of Location 1 given as a double variable  
        double lon1 = -119.8489;  
  
        //A print statement is used to combine our three variables  
        System.out.print(loc1 + " is located at " + lat1 +  
                          " degrees latitude and " + lon1 + " degrees longitude." );  
    }  
}
```





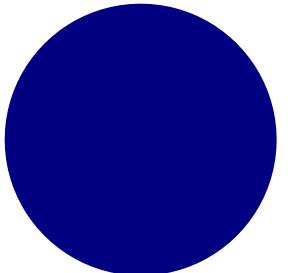
# How far is your home from UCSB? (more complex program)

Week

2

## Example #2

- If Example 1 was easy, try to calculate the distance between two points:  
Where you went to high school and UCSB:
- Look up the lat, long of your high school in decimal degrees
  - E.g.: I went to Cheyenne Mountain in Colorado Springs, Colorado
  - Lat: 38.8031 Lon: -104.8572
- We will use the Haversine formula to determine the distance between these locations. To do this we will need to find functions and/or do the following:
  - Convert decimal degrees to radians
  - Determine the differences in lat and long between locations
  - Apply the equation using the java math package (see hyperlink)
  - Print out your answer!



# How far is your home from UCSB??

Week

2

## Example #2

# Give it a try!

(Answer on the next slide and in  
Example2.java)



# How far is your home from UCSB? (Example Code)

Week

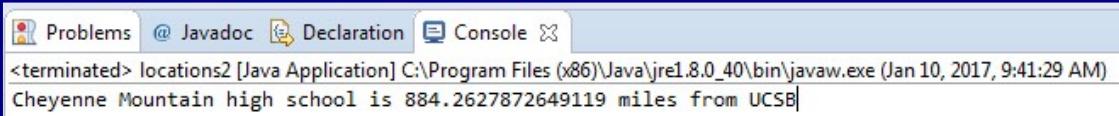
2

## Example #2

- Example Code:

```
public class locations2 {  
    public static void main(String[] args) {  
        // Locations of interest given as a String variables  
        String loc1 = "UCSB";  
        String loc2 = "Cheyenne Mountain";  
  
        // The latitudes given as a double variable in radians.  
        // This is done using the 'toRadians' tool in the 'Math' package  
        double lat1 = Math.toRadians(34.4139);  
        double lat2 = Math.toRadians(38.8031); // Enter your data!  
  
        // The longitudes given as a double variable in radians.  
        double lon1 = Math.toRadians(119.8489);  
        double lon2 = Math.toRadians(104.8572); // Enter your data!  
  
        // Determine change in lat and long between locations:  
        double d_lat = Math.abs(lat2 - lat1);  
        double d_lon = Math.abs(lon2 - lon1);  
  
        /* Apply the Haversine Formula  
         * The Math package is used again for sin, cos, arctan2, and square root operators  
         * The 'Math.pow(variable, 2)' is a method for squaring a number */  
  
        double a = Math.pow(Math.sin(d_lat/2),2) + (Math.cos(lat1) * Math.cos(lat2) * Math.pow(Math.sin(d_lon/2),2));  
        double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));  
  
        // To get the distance in miles we multiply by the radius of the earth - 3,961 miles  
        double d = 3961 * c;  
  
        // A print statement is used to provide our answer  
        System.out.print(loc2 + " High School is " + d + " miles from " + loc1);  
    }  
}
```

- Output:



The screenshot shows a Java IDE interface with several tabs at the top: Problems, @ Javadoc, Declaration, Console, and another tab that is partially visible. The Console tab is active, displaying the command-line output of the Java application. The output text is:  
<terminated> locations2 [Java Application] C:\Program Files (x86)\Java\jre1.8.0\_40\bin\javaw.exe (Jan 10, 2017, 9:41:29 AM)  
Cheyenne Mountain high school is 884.2627872649119 miles from UCSB|

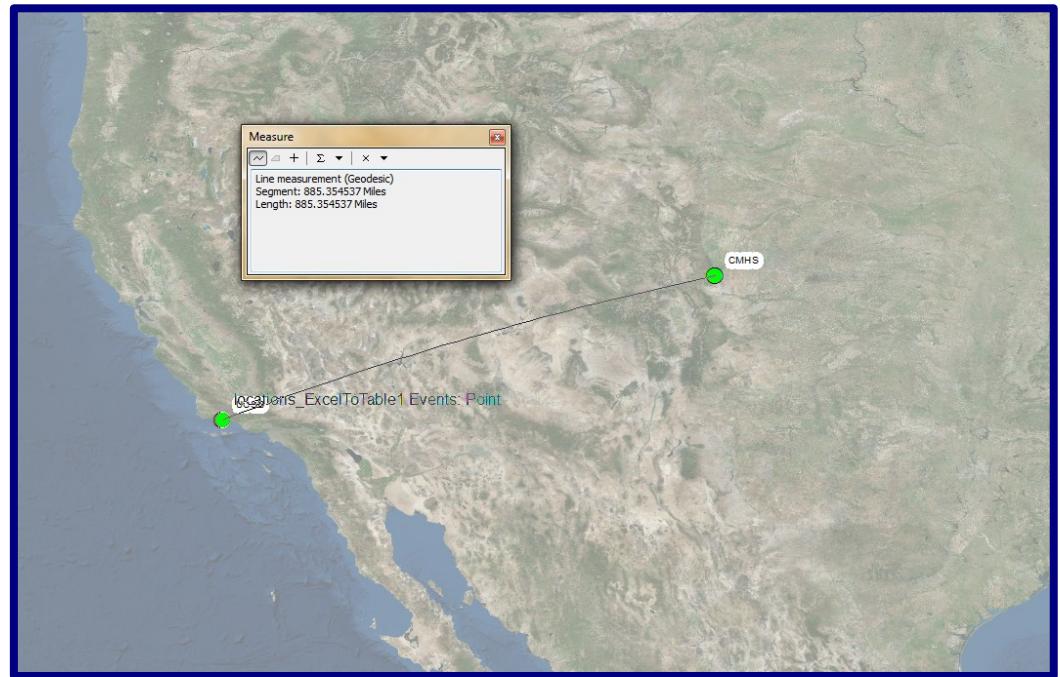
# How far is your home from UCSB? (more complex program)

Week

2

## Example #2

- Validation using ESRI ArcGIS



- Percent Difference:
  - $[885.3545 - 884.2627] / 885.3545 * 100 = .12\%$

# 3. Debugging

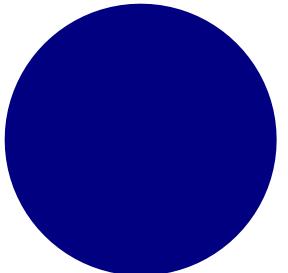
# Debugging

Week

2

## Debugging

- It is very easy, and natural, to make mistakes when programming
- There are a number of ways to find mistakes:
  1. Visually
  2. Working/reading the program backwards
  3. Debugging
- In Eclipse, debugging allows to run a program INTERACTIVELY while watching the source code and the variables as it executes
- Eclipse even provides a ‘Debug Perspective’ loaded with a pre-confined set of VIEWS to help do this
- It will also allow you to control the execution flow through embedded ‘debug’ commands.



# Common Mistakes to watch for:

Week

2



## Debugging



1. Missing Semicolons
2. Typos
3. Wrong Variable Types
4. Uneven brackets, parentheses, etc.
5. Missing package extensions (i.e 'Math.')

# Debugging Practice

Week

2

## Example #3

- In this example we do the following:
  1. Create breakpoints
  2. Open the “debugging perspective” (DP)
  3. Execute code in the DP
  4. Edit Variables and breakpoints in DP

# Problem:

- Open Example3\_buggy.java
- This code is written to:
  - A) select a random number of values (1-10)
  - B) determine how many coordinate pairs can be made from the values
  - C) determine what kind of geometry can be formed with that number of coordinate pairs, and
  - D) print out a pseudo WKT string
- Run the code a few times:

```
Number of Values = 8  
Number of pairs = 4  
Geometry Type = POLYGON  
POLYGON [87, 15, 64, 97, 70, 28, 93, 94]
```

Good !!

```
Number of Values = 2  
Number of pairs = 0  
Geometry Type = INVALID  
INVALID []
```

1  
POINT  
POINT [X, Y]

Bad !!

# Adding/Removing Breakpoints

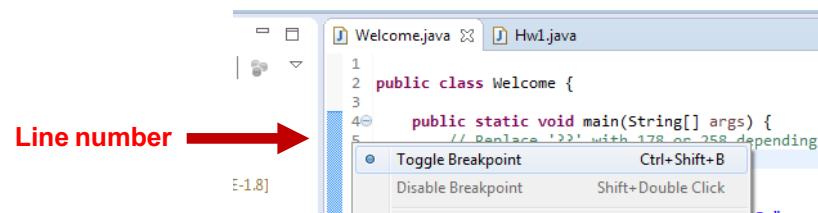
Week

2

## Debugging

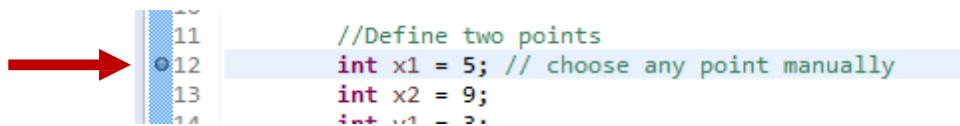
- Breakpoints are locations in the source code, created by you, where the program should stop during debugging.
- Once the program stops, you can examine variables, change their content, among other things.
- Break points can be added and removed in two ways:

1. Right clicking on a line number and selecting “Toggle Break Point”



2. Having your cursor on a line and holding down 'Ctrl +Shift + B'  
For MAC user anytime a shortcut is given, replace Ctrl with command

- When a break point is added successfully a ‘blue dot’ will appear



- Add a break point to lines 9, 14, 22, 34, 45

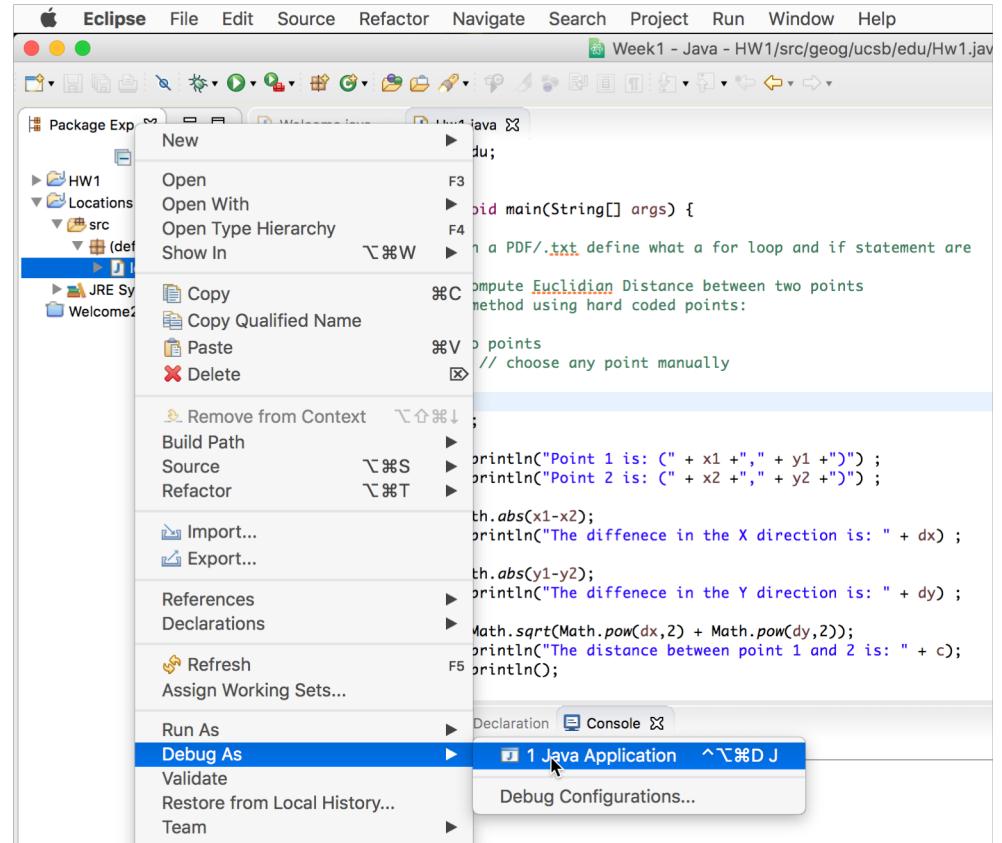
# Starting the debugger

Week

2

## Debugging

- To begin debugging a Java File Right click on the 'Example3\_buggy.java' file and select:
- Debug As → Java Application



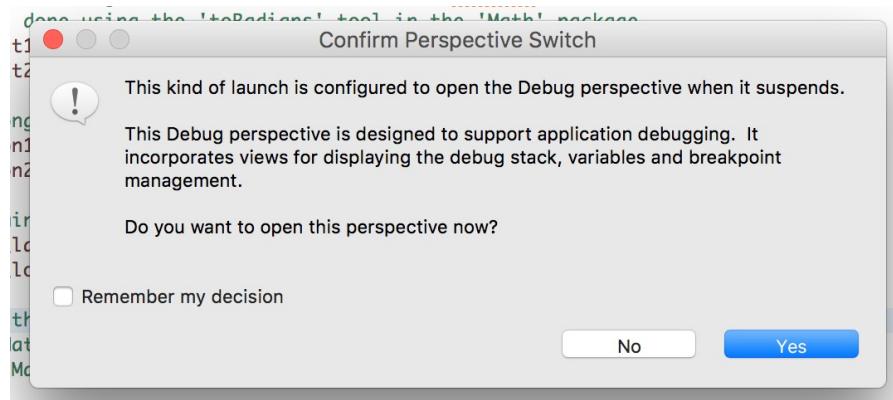
# Starting the debugger

Week

2

## Debugging

- If you have not defined any break points the continue programming normally. Remember that debugging will ONLY work if breakpoints have been assigned!
- When BREAKPOINTS are assigned, and the DEBUGGER is run Eclipse will ask if you want to switch to the Debugger Perspective.
- Select 'YES'



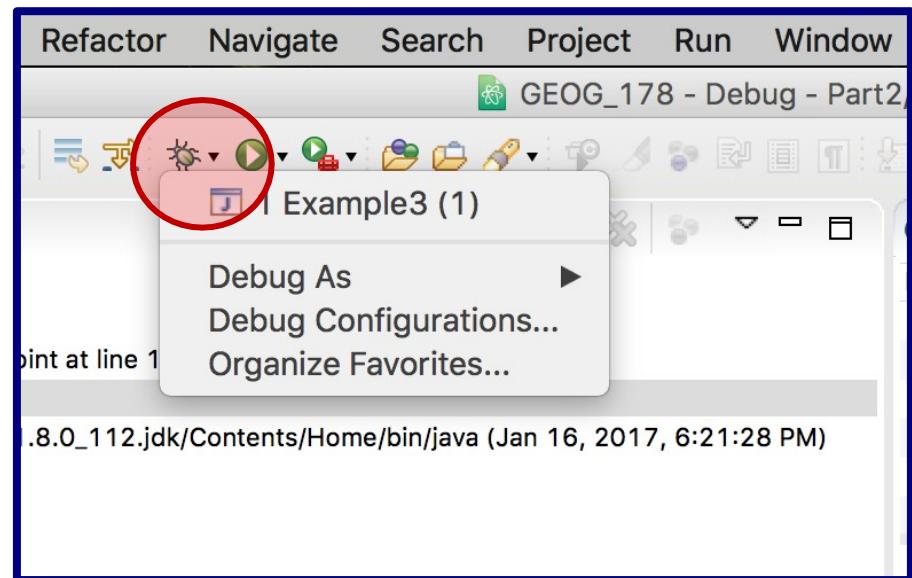
# Starting the debugger

Week

2

## Debugging

The Debugger can also be launched and executed from the Top Toolbar!

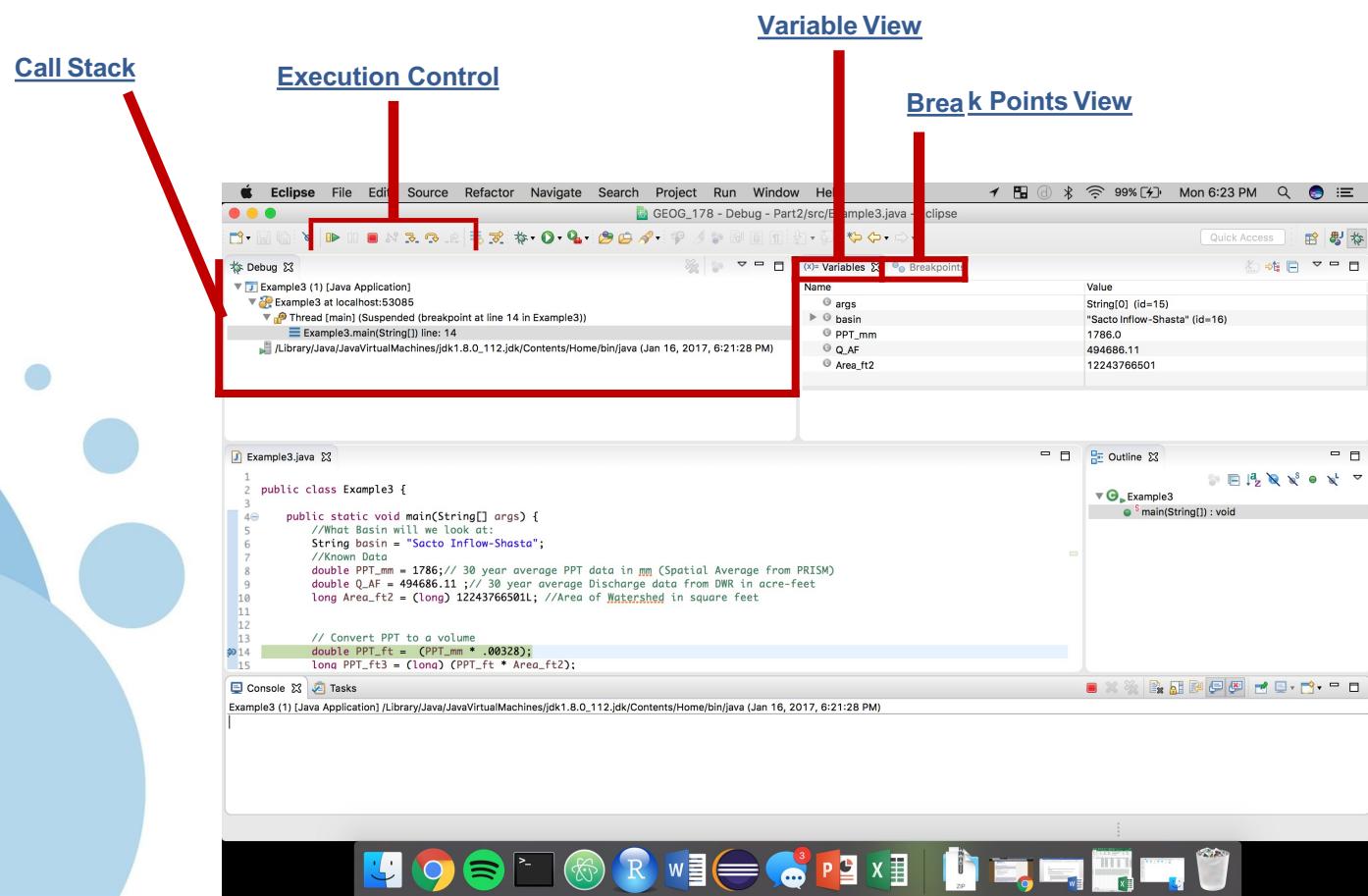


# The Debugger Perspective

Week

2

## Debugging



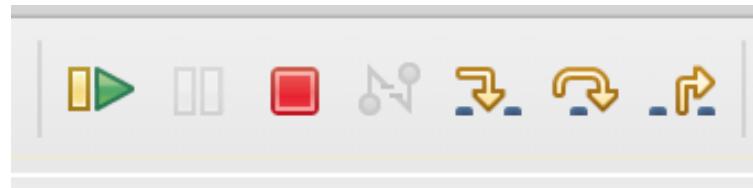
# Execution Control

Week

2

## Debugging

- In the “Debugging Perspective” Eclipse allows you to control the execution of a program.
- The Following shows how these commands work in addition to there keyboard shortcuts:



F8      STOP      F5    F6    F7

- F5 → Executes the currently selected line.
- F6 → Executes a method – or ‘steps-over’ a call without stepping into the debugger (MOST USEFULL!!)
- F7 → ‘Steps out’ to the caller of the currently executed method
- F8 → Tells the Debugger to resume the execution of the program code until it reaches the next break point.

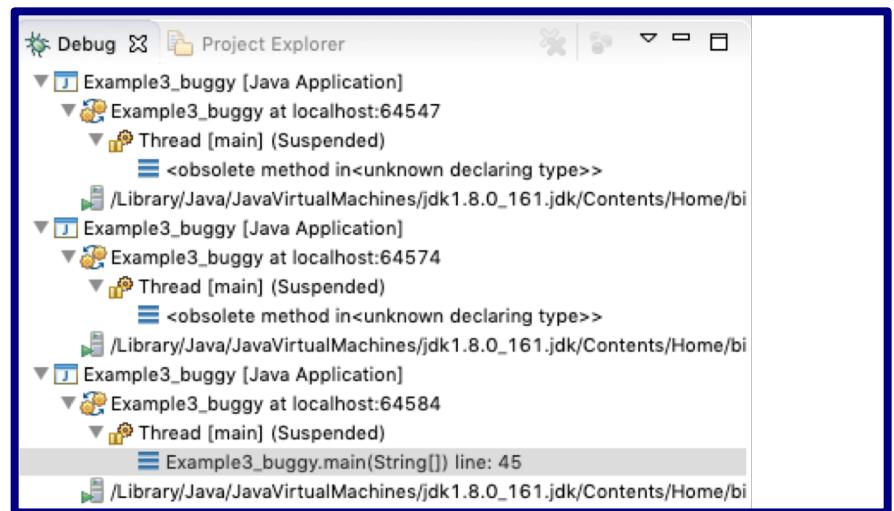
# The Call Stack

Week

2

## Call Stack

- The current stack is displayed in the DP
- The call stack shows the parts of your program which are currently executed and how they relate to each other
- Clicking on one element of this stack switches the editor view to display the corresponding class, and "variables" view will show variables of this stack element.



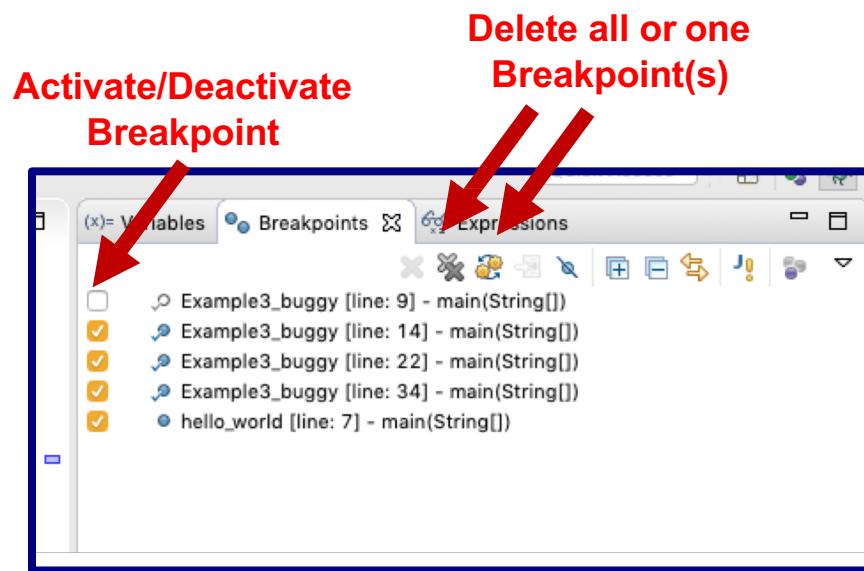
# The Breakpoint View

Week

2

## Breakpoint View

- This view port allows you to delete, deactivate and modify properties of breakpoints.
- You can deactivate a breakpoint by unselecting the check box next to each or....
- You can delete them using the corresponding buttons in the toolbar.



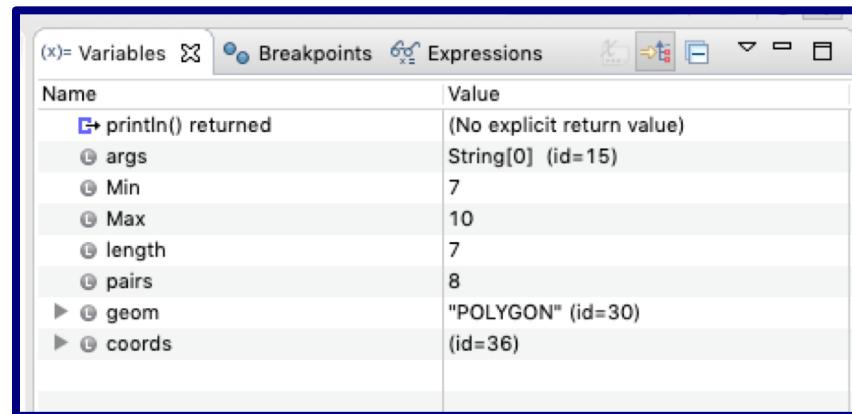
# Variable View

Week

2

## Variable View

- The Variables Viewport shows the fields and local variables from the current executing stack.
- You must run the Debugger (click on the little bug in the toolbar) to see the variables in the view!
- This is a good place to make sure all variable are initializing and are representing what you think they should...



The screenshot shows the Eclipse IDE's Variables View window. The window title is '(x)= Variables'. It contains a table with two columns: 'Name' and 'Value'. The 'Name' column lists local variables and their types, while the 'Value' column shows their current values or states. The variables listed are:

Name	Value
↳ println() returned	(No explicit return value)
↳ args	String[0] (id=15)
↳ Min	7
↳ Max	10
↳ length	7
↳ pairs	8
▶ ↳ geom	"POLYGON" (id=30)
▶ ↳ coords	(id=36)

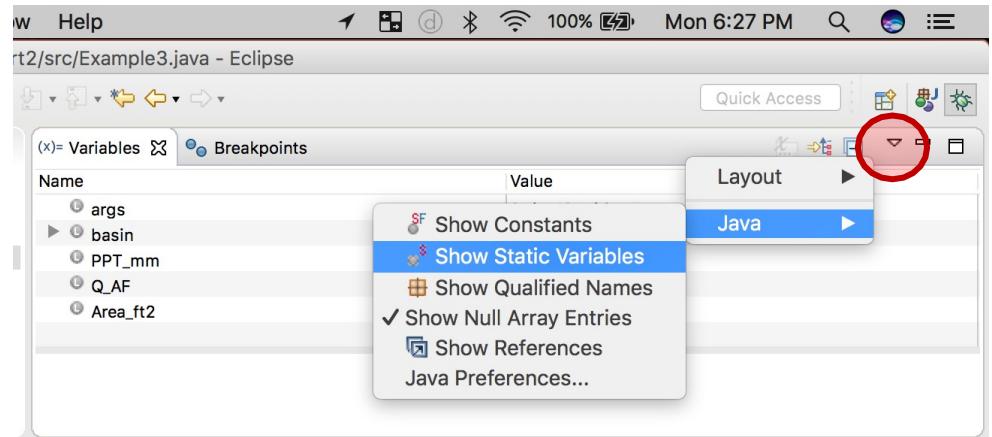
# Variable View

Week

2

## Variable View

- In the Variable Viewport, you can use the Drop-Down Menu to display static variables



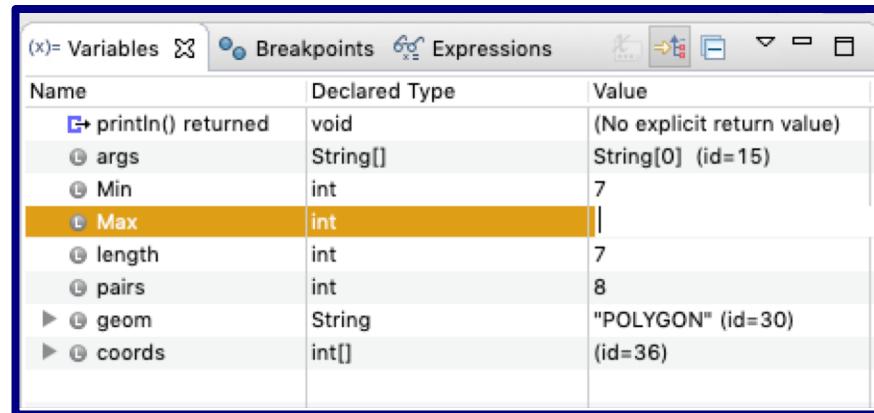
# Variable View

Week

2

## Variable View

- The Variables Viewport also allows you to change the value of each static variable before resuming!
- Do this by double clicking (or right clicking on the value box)



Name	Declared Type	Value
↳ println() returned	void	(No explicit return value)
↳ args	String[]	String[0] (id=15)
↳ Min	int	7
↳ Max	int	
↳ length	int	7
↳ pairs	int	8
▶ ↳ geom	String	"POLYGON" (id=30)
▶ ↳ coords	int[]	(id=36)

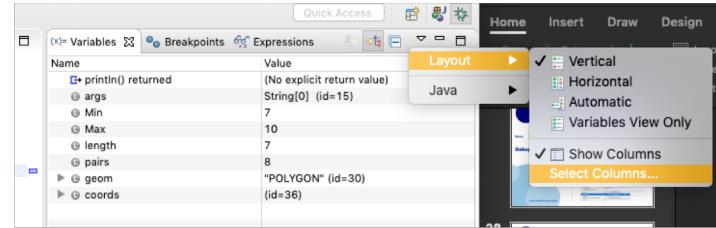
# Variable View

Week

2

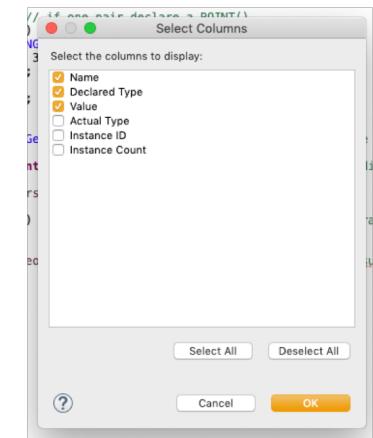
## Variable View

(1)



(2)

Go: Layout → Select Columns → Type



(3)

Name	Declared Type	Value
println() returned	void	No explicit return value)
args	String[]	String[0] (id=15)
Min	int	7
Max	int	10
length	int	7
pairs	int	3
geom	String	"POLYGON" (id=30)
coords	int[]	(id=36)

# Why did we do this??

Week

2

## Big Picture

- In this example try to figure out where the logic in the buggy script is failing and correct it ...
- The idea is to be comfortable exploring a new program (or your own) in the debugger to both find errors AND familiarize yourself with it.
- Even though you did not write this the sample code you should have a good understanding of the variables and steps executed after using the debugger....
- A debugged solution can be found in Example3\_debugged.java

# 4. Loops

# What are Loops??

Week

2

## Loops

- Loops are a sequence of instructions to be continually repeated until a specific condition is reached.
- They are helpful when checking for a condition or when repeating the same process over a large amount of data points...
- Anytime you want to do something many times a loop will be helpful!

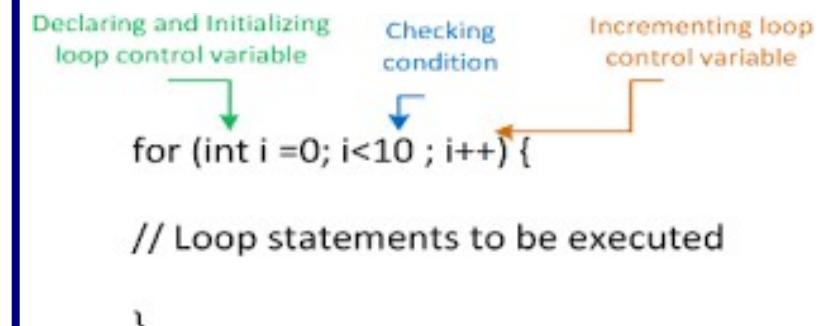
# For Loops and While Loops

Week

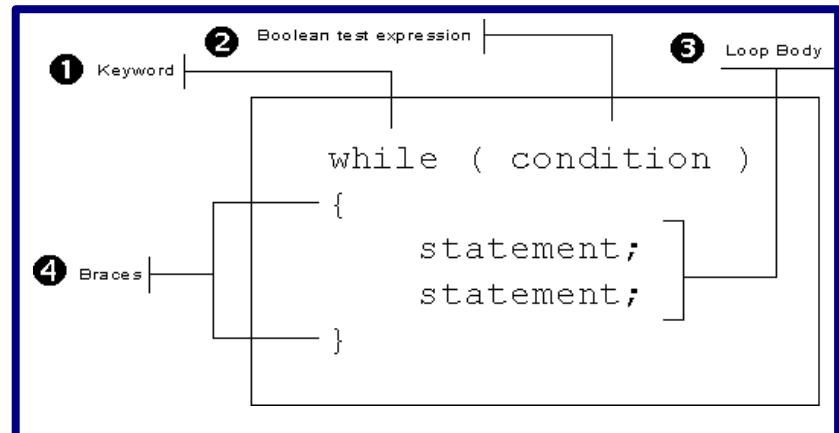
2

## Loops

- FOR LOOP SYNTAX



- WHILE LOOP SYNTAX

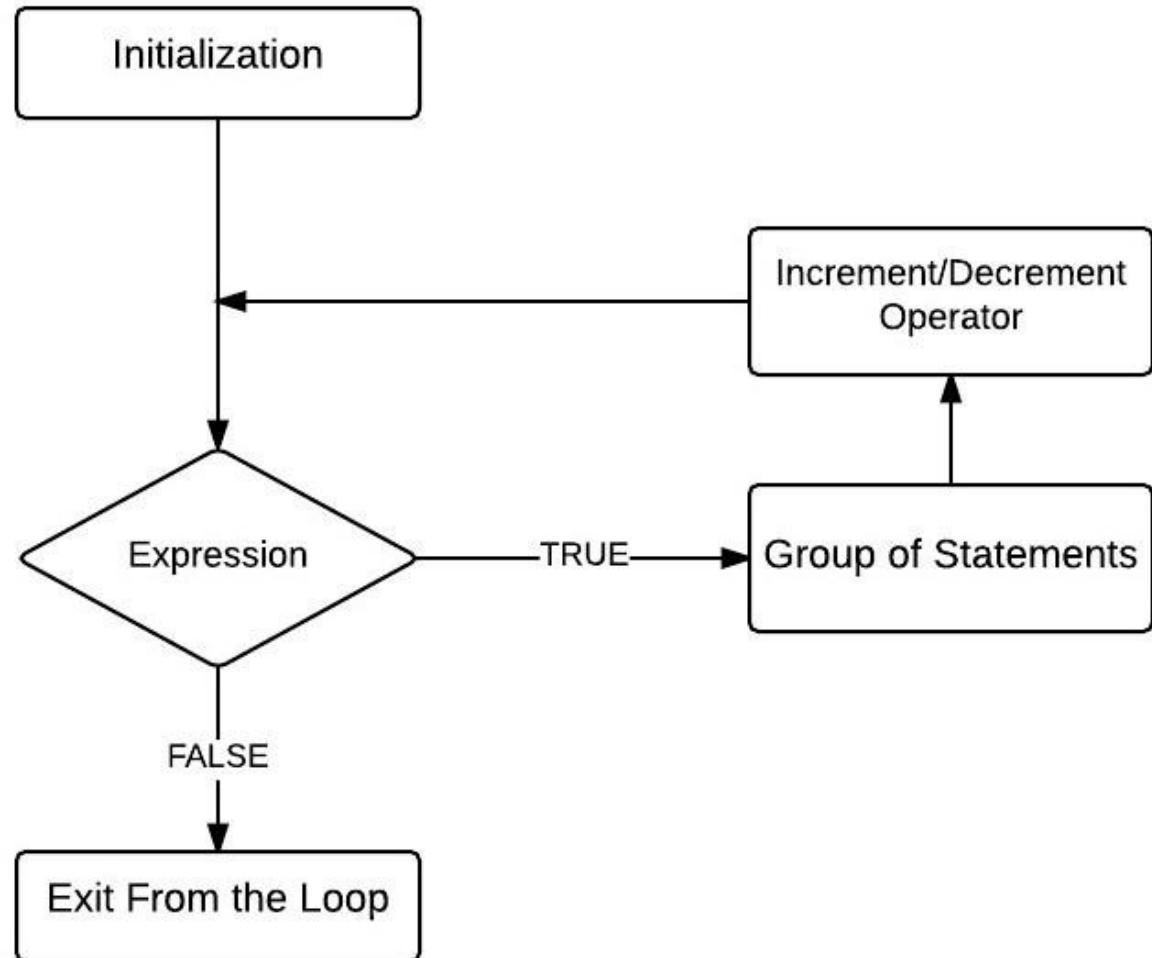


# Loop Logical Flowchart

Week

2

Loops



# Summary:

Week

2

**END:**

**At this point you should be comfortable:**

1. Launching a workspace and creating a Java Project in Eclipse on both your machine AND a lab machine
2. Importing a program from the class website, github, your flash, ora partners flash
3. The different types of variables, their uses, and how to declare them
4. Manipulating variables with the 'Math' package and print statements
5. Writing, and reading, for and while loops in your program and others
6. Opening and navigating the Debugger (this will become valuable when our programs get more complicated)

**If you have any questions please don't hesitate to email or visit office hours!**