GEOG 178/258
Week 2:

Variables, Debugging, and Loops

mike johnson

OVERVIEW

Week 2

Contents

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 reserve space in memory
  - 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 types 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 '<u>String</u>' 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 **pull** the new data

Complete Workflow:

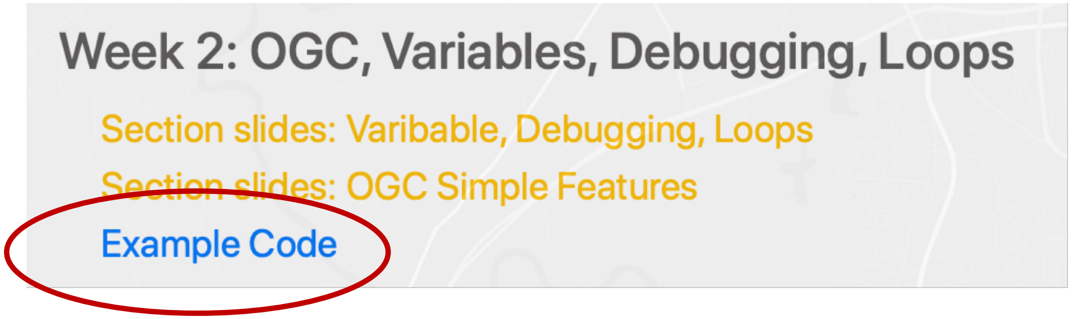
Do once:

```
> 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 ‘
```

To Update:

```
> 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: Varibable, Debugging, Loops

Section slides: OGC Simple Features

Example Code

Importing an Existing Project

Week 2

Debugging

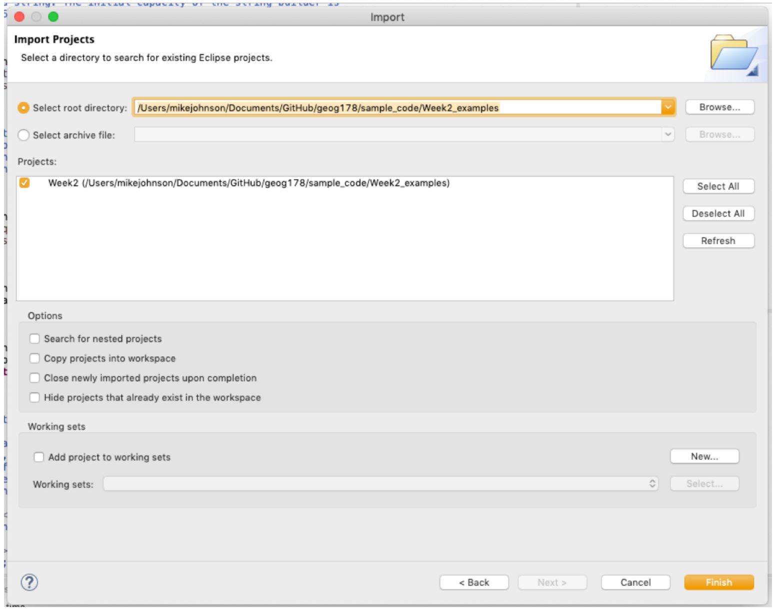
- Open an Eclipse workspace on your flash drive or local desktop
- Go to: File → Import → General → Existing
  - Select ”Select root directory”
  - Click ‘Browse’
  - Point it to the ‘Week2_examples’ folder
  - Click ‘Finish’

Importing an Existing Project

Week 2

Debugging

- Select ”Select root directory”
- Click ‘Browse’
- Point it to the downloaded folder on your desktop



Import

Import Projects

Select a directory to search for existing Eclipse projects.

● Select root directory: /Users/mikejohnson/Documents/GitHub/geog178/sample_code/Week2_examples

○ Select archive file:

Projects:

☑ Week2 (/Users/mikejohnson/Documents/GitHub/geog178/sample_code/Week2_examples)

Options

□ Search for nested projects
□ Copy projects into workspace
□ Close newly imported projects upon completion
□ Hide projects that already exist in the workspace

Working sets

□ Add project to working sets

Working sets:

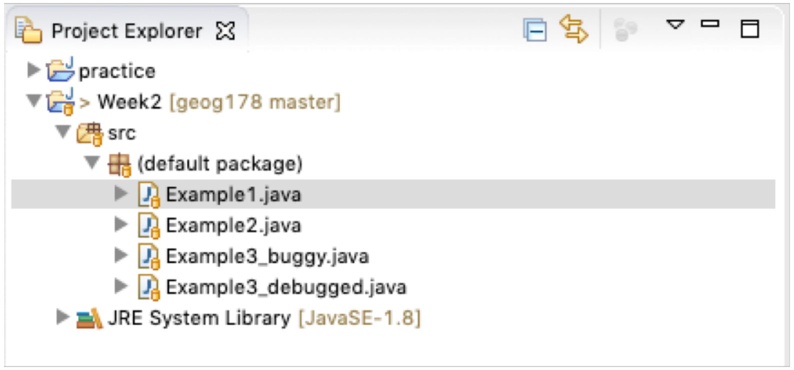
- Click ‘Finish’

Importing an Existing Project

Week 2

Debugging

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



```
Project Explorer
▶ practice
▼ > Week2 [geog178 master]
  ▼ src
    ▼ (default package)
      ▶ Example1.java
      ▶ Example2.java
      ▶ Example3_buggy.java
      ▶ Example3_debugged.java
  ▶ JRE System Library [JavaSE-1.8]
```

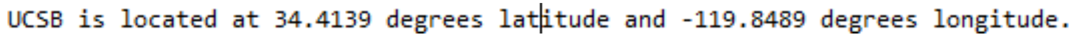
- **Create a new class called `My_Example1`**

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.



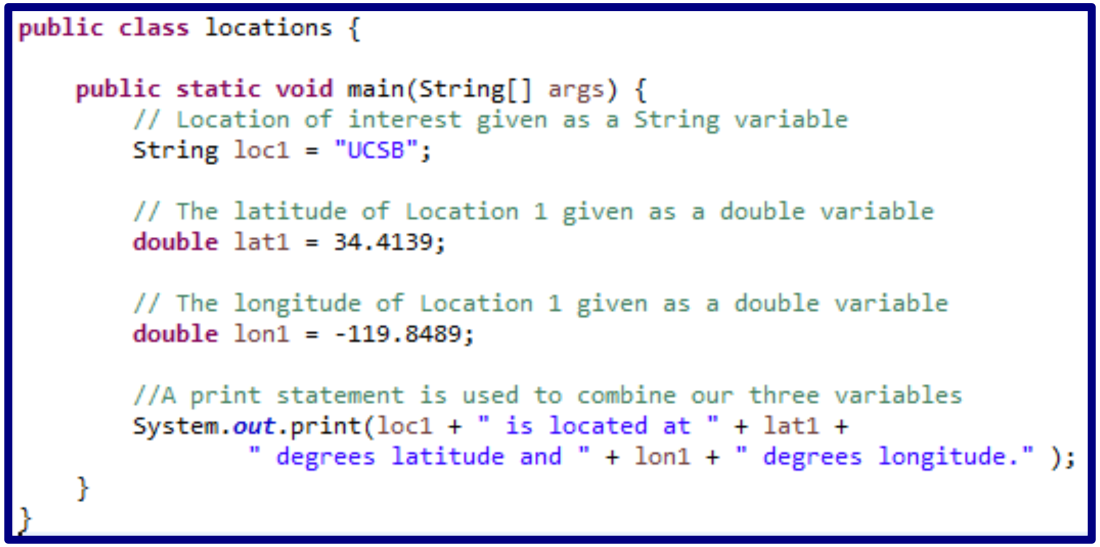
```
UCSB is located at 34.4139 degrees latitude and -119.8489 degrees longitude.
```

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

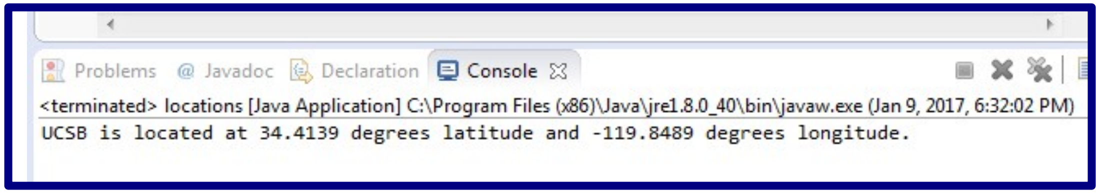
Where is UCSB (simple program)

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." );
    }
}
```



```
<terminated> locations [Java Application] C:\Program Files (x86)\Java\jre1.8.0_40\bin\javaw.exe (Jan 9, 2017, 6:32:02 PM)
UCSB is located at 34.4139 degrees latitude and -119.8489 degrees longitude.
```

How far is your high school 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 (1) high school and (2) 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:

- Create a new class (My_Example2) and copy the contents of My_Example1
- Convert decimal degrees to radians
- Determine the differences in lat and long between locations
- Apply the equation (see hyperlink) using the Java math package
- Print out your answer!

How far is your high schools from UCSB??

Week 2

Example #2

Give it a try!

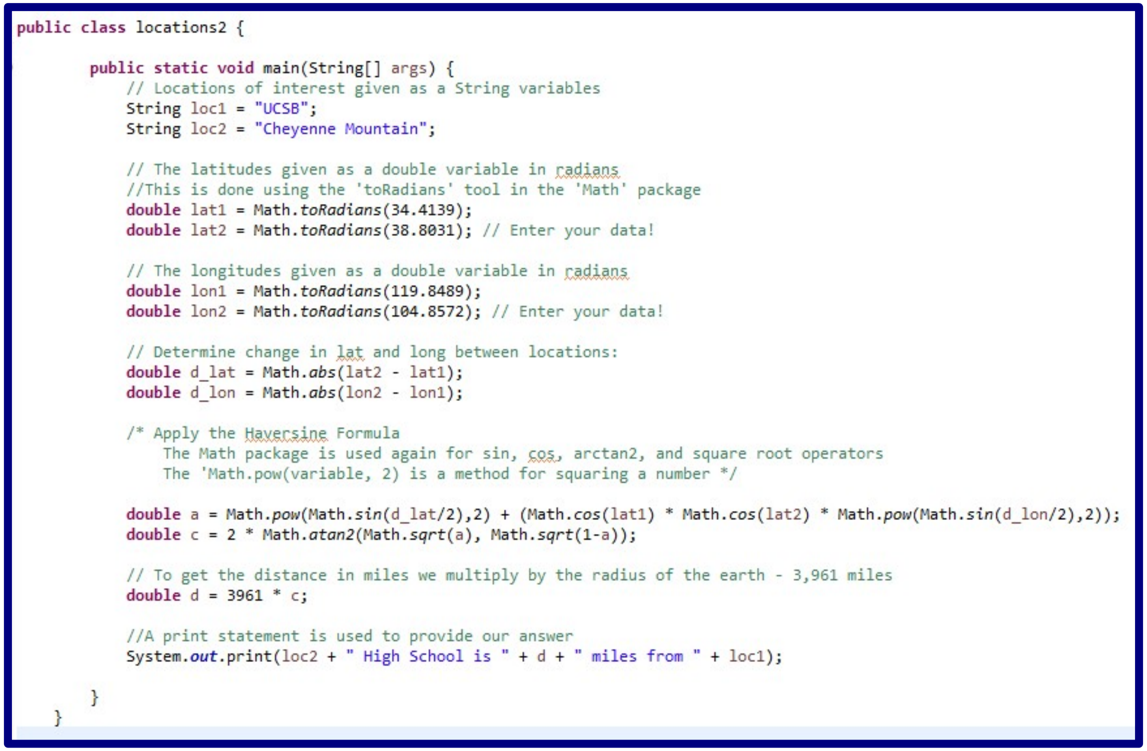
(Answer on the next slide and in Example2.java)

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

Week 2

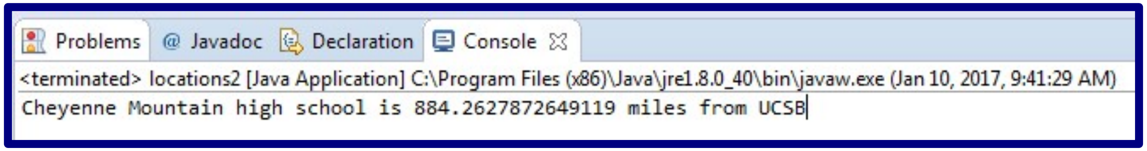
Example #2

- 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:



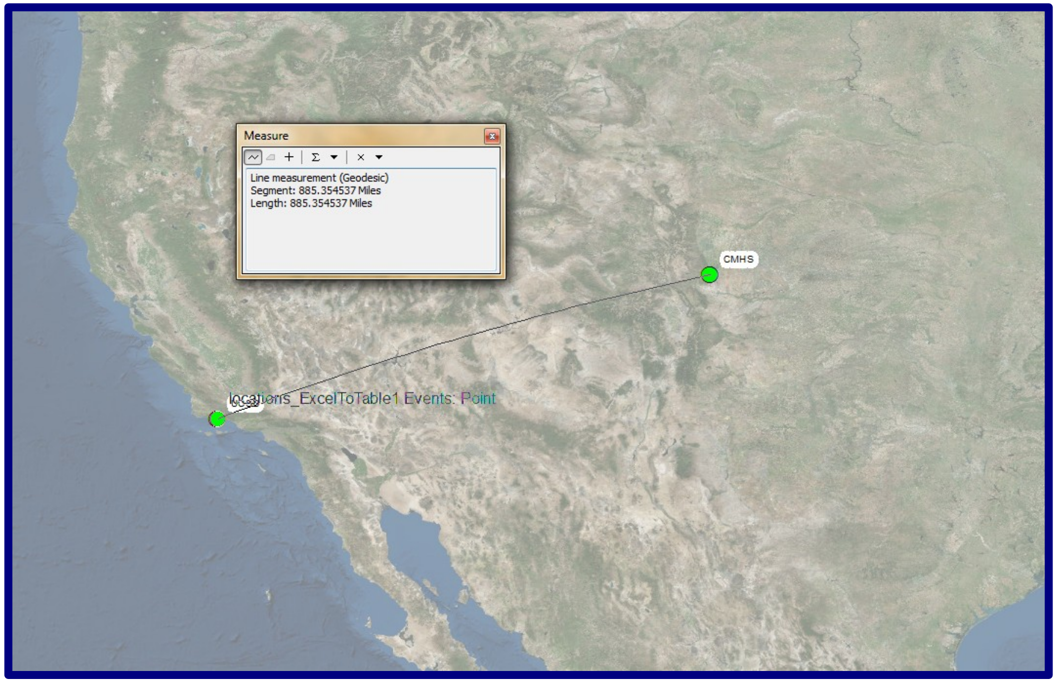
```
<terminated> locations2 [Java Application] C:\Program Files (x86)\Java\jre1.8.0_40\bin\javaw.exe (Jan 10, 2017, 9:41:29 AM)
Cheyenne Mountain high school is 884.2627872649119 miles from UCSB
```

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

Week 2

Example #2

- Validation using ESRI ArcGIS



Line measurement (Geodesic)
Segment: 885.354537 Miles
Length: 885.354537 Miles

- 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 programing
- 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 INTERACTIVLY 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

- Open Example3_buggy.java
- 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:

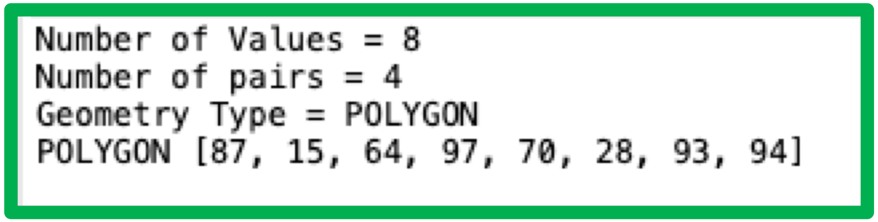
Week 2

Example #3

- Open Example3_buggy.java
- This code is written to:

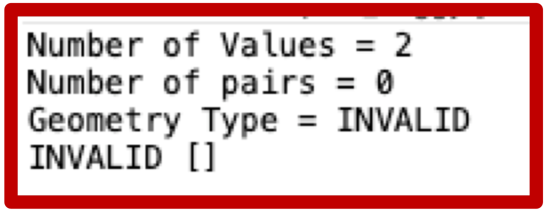
  A) select a random number of values (1-10)
  B) determine how many coordinate pairs can be made (P)
  C) determine what kind of geometry can be formed by P
  D) print out a pseudo WKT string

- 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

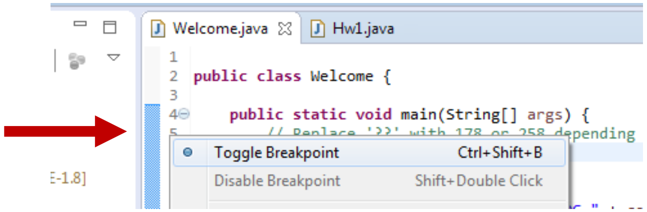
Week 2

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”

Line number

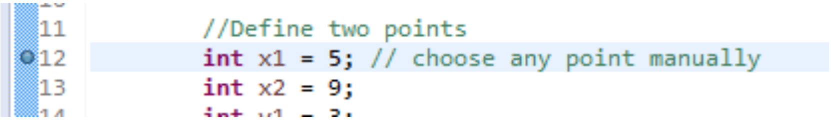


```
1
2  public class Welcome {
3
4      public static void main(String[] args) {
5          // Replace '??' with 178 or 258 depending
```

Welcome.java
Hw1.java
Toggle Breakpoint Ctrl+Shift+B
Disable Breakpoint Shift+Double Click
E-1.8]

2. Having you 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



```
11          //Define two points
12          int x1 = 5; // choose any point manually
13          int x2 = 9;
14          int y1 = 3;
```

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

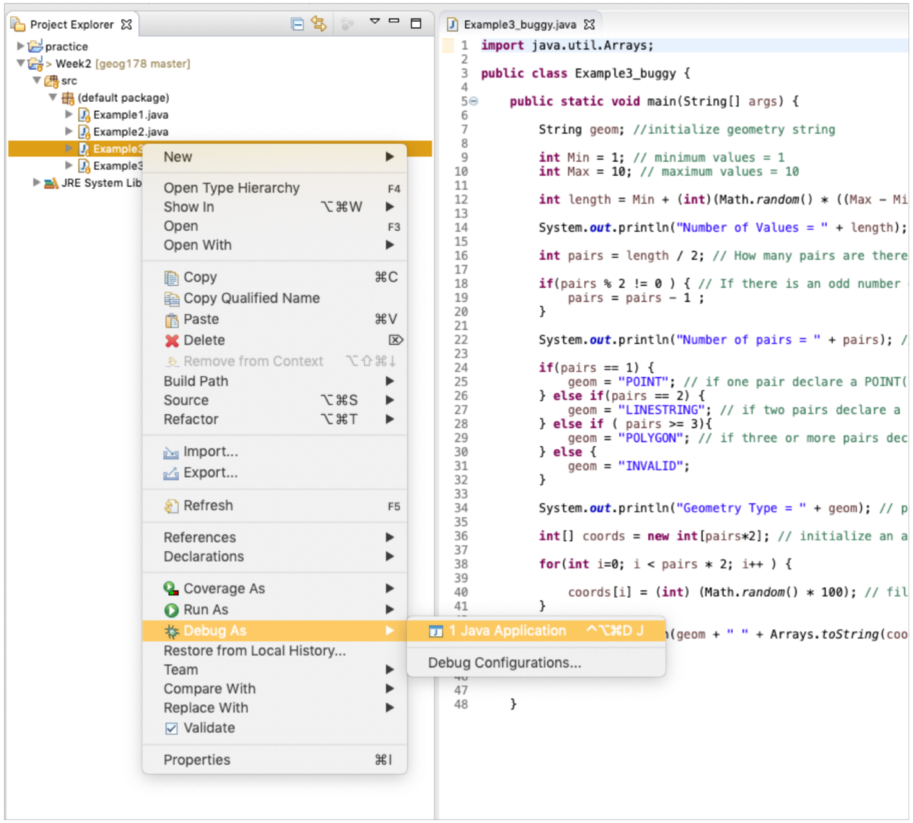
Starting the debugger

Week 2

Debugging

- To begin debugging a Java File Right click on the ‘Example3_buggy.java’ file and select:

- Debug As → Java Application



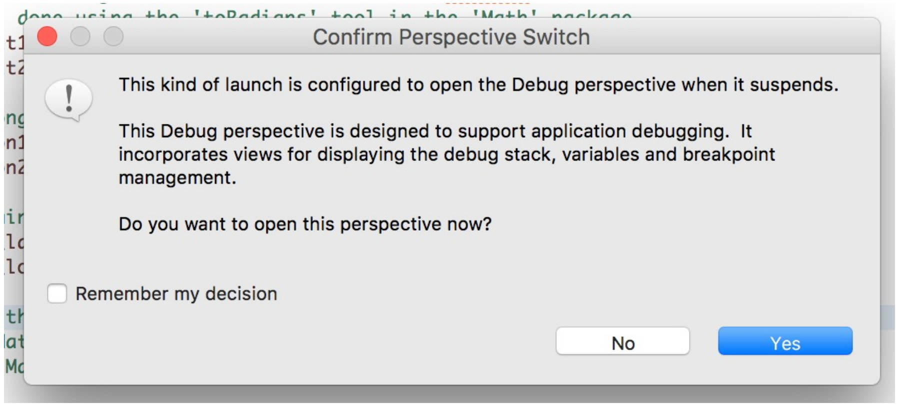
```java
import java.util.Arrays;

public class Example3_buggy {

    public static void main(String[] args) {

        String geom; //initialize geometry string

        int Min = 1; // minimum values = 1
        int Max = 10; // maximum values = 10

        int length = Min + (int)(Math.random() * ((Max - Mi

        System.out.println("Number of Values = " + length);

        int pairs = length / 2; // How many pairs are there

        if(pairs % 2 != 0 ) { // If there is an odd number
            pairs = pairs - 1 ;
        }

        System.out.println("Number of pairs = " + pairs); /

        if(pairs == 1) {
            geom = "POINT"; // if one pair declare a POINT(
        } else if(pairs == 2) {
            geom = "LINESTRING"; // if two pairs declare a
        } else if ( pairs >= 3){
            geom = "POLYGON"; // if three or more pairs dec
        } else {
            geom = "INVALID";
        }

        System.out.println("Geometry Type = " + geom); // p

        int[] coords = new int[pairs*2]; // initialize an a

        for(int i=0; i < pairs * 2; i++ ) {

            coords[i] = (int) (Math.random() * 100); // fil
        }

        n(geom + " " + Arrays.toString(coo


    }
```

Starting the debugger

Week 2

Debugging

- If you have not defined any break points the continue programing 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’



Confirm Perspective Switch

This kind of launch is configured to open the Debug perspective when it suspends.

This Debug perspective is designed to support application debugging. It incorporates views for displaying the debug stack, variables and breakpoint management.

Do you want to open this perspective now?

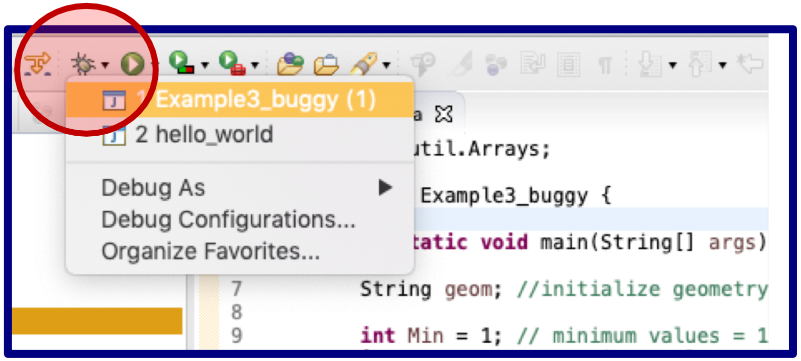
☐ Remember my decision

Starting the debugger

Week 2

Debugging

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



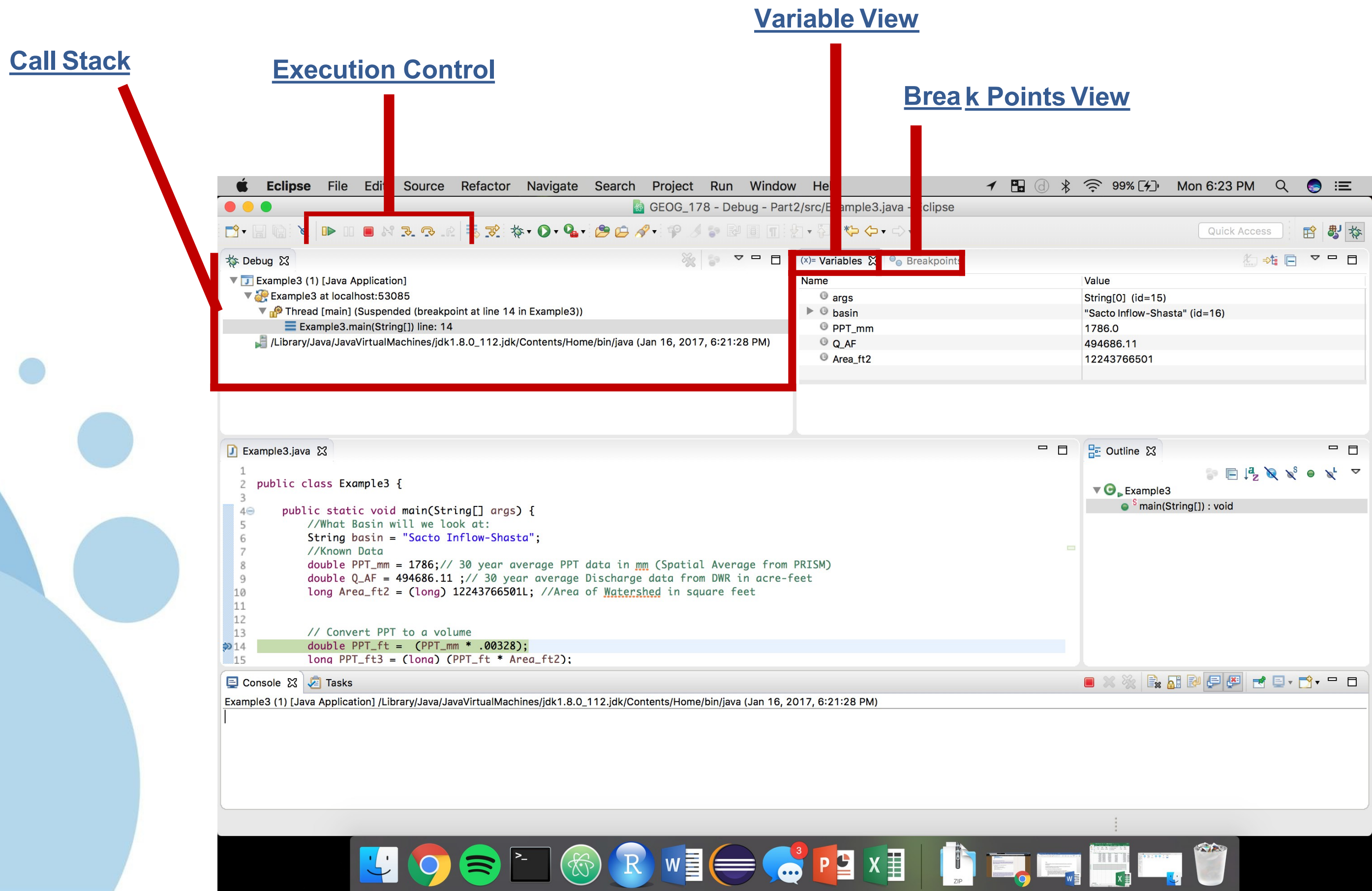
```
util.Arrays;
 Example3_buggy {
tatic void main(String[] args)
   String geom; //initialize geometry

   int Min = 1; // minimum values = 1
```

The Debugger Perspective

Week 2

Debugging

Once you enter the Debugger Perspective you will see the following:



Call Stack

Execution Control

Variable View

Brea k Points View

Debug:

```
Example3 (1) [Java Application]
  Example3 at localhost:53085
    Thread [main] (Suspended (breakpoint at line 14 in Example3))
      Example3.main(String[]) line: 14
  /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Jan 16, 2017, 6:21:28 PM)
```

Name	Value
args	String[0] (id=15)
basin	"Sacto Inflow-Shasta" (id=16)
PPT_mm	1786.0
Q_AF	494686.11
Area_ft2	12243766501

Example3.java

```java

public class Example3 {

    public static void main(String[] args) {
        //What Basin will we look at:
        String basin = "Sacto Inflow-Shasta";
        //Known Data
        double PPT_mm = 1786;// 30 year average PPT data in mm (Spatial Average from PRISM)
        double Q_AF = 494686.11 ;// 30 year average Discharge data from DWR in acre-feet
        long Area_ft2 = (long) 12243766501L; //Area of Watershed in square feet


        // Convert PPT to a volume
        double PPT_ft =  (PPT_mm * .00328);
        long PPT_ft3 = (long) (PPT_ft * Area_ft2);
```

Outline:

```
Example3
  main(String[]) : void
```

Console:

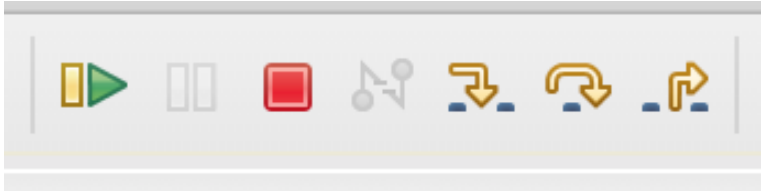
```
Example3 (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Jan 16, 2017, 6:21:28 PM)
```

Week 2

Debugging

Execution Control

- 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

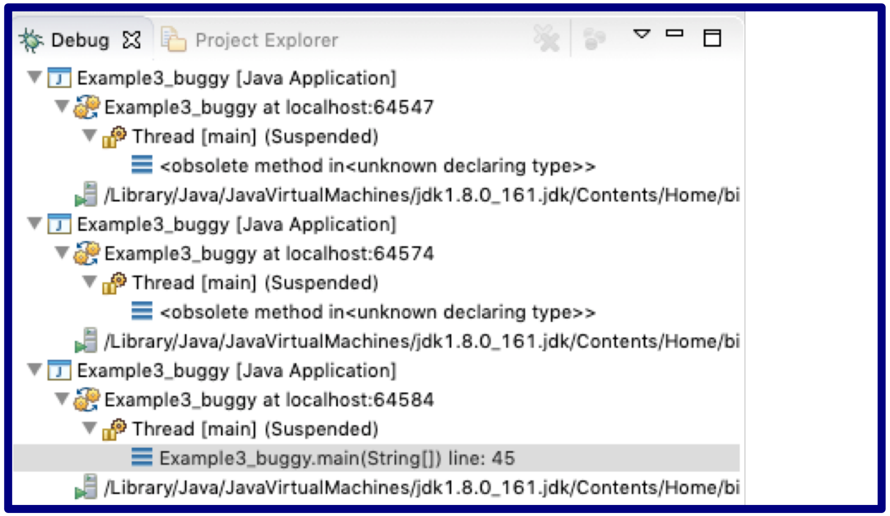
- F5 → Executes the currently selected line.
- F6 → Executes a method – or ‘steps-over’ a call without stepping into the debugger (MOST USEFULL!!)
- F7 → ‘Steps out’ to the caller of the currently executed method
- F8 → Tells the Debugger to resume the execution of the program code until it reaches the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

The Call Stack

Week 2

Call Stack

- The call 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 the "variables" view will show variables of this stack element.



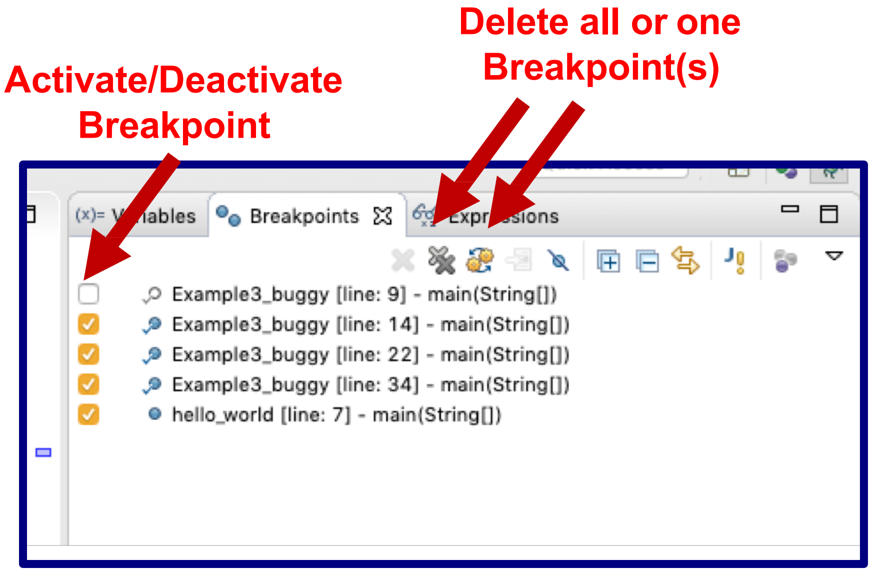
```
▼ Example3_buggy [Java Application]
  ▼ Example3_buggy at localhost:64547
    ▼ Thread [main] (Suspended)
        <obsolete method in<unknown declaring type>>
  /Library/Java/JavaVirtualMachines/jdk1.8.0_161.jdk/Contents/Home/bi
▼ Example3_buggy [Java Application]
  ▼ Example3_buggy at localhost:64574
    ▼ Thread [main] (Suspended)
        <obsolete method in<unknown declaring type>>
  /Library/Java/JavaVirtualMachines/jdk1.8.0_161.jdk/Contents/Home/bi
▼ Example3_buggy [Java Application]
  ▼ Example3_buggy at localhost:64584
    ▼ Thread [main] (Suspended)
        Example3_buggy.main(String[]) line: 45
  /Library/Java/JavaVirtualMachines/jdk1.8.0_161.jdk/Contents/Home/bi
```

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.



Delete all or one Breakpoint(s)

Activate/Deactivate Breakpoint

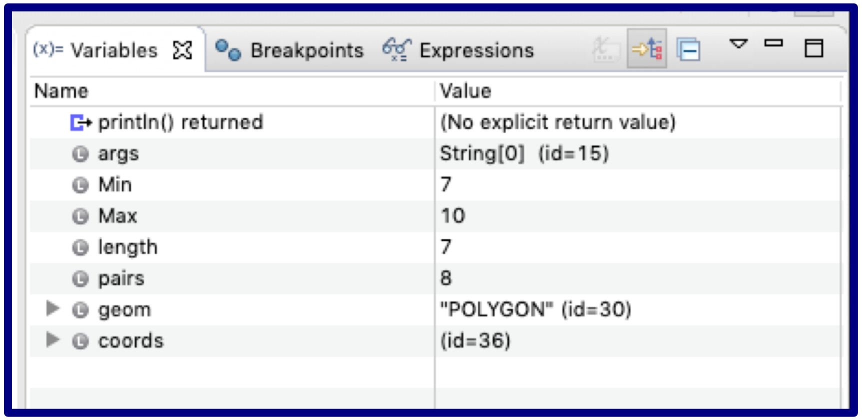
□ Example3_buggy [line: 9] - main(String[])
☑ Example3_buggy [line: 14] - main(String[])
☑ Example3_buggy [line: 22] - main(String[])
☑ Example3_buggy [line: 34] - main(String[])
☑ hello_world [line: 7] - main(String[])

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…



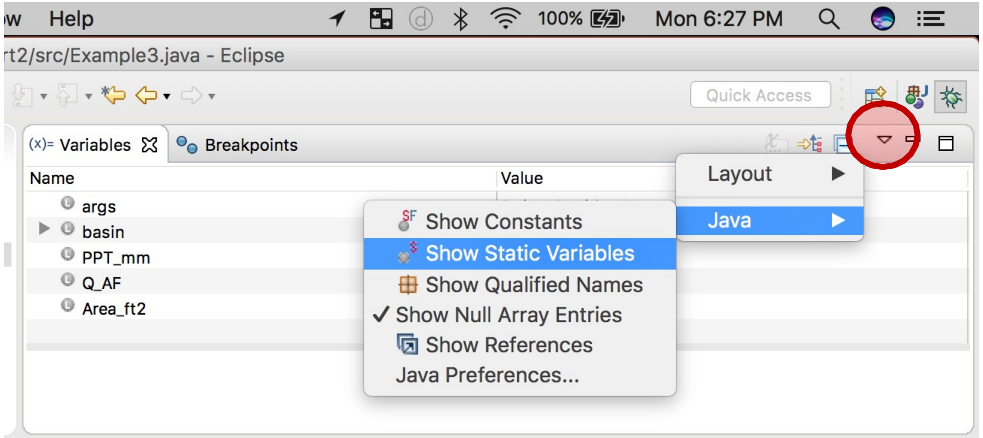
Name	Value
println() returned	(No explicit return value)
args	String[0] (id=15)
Min	7
Max	10
length	7
pairs	8
geom	"POLYGON" (id=30)
coords	(id=36)

Variable View

Week 2

Variable View

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

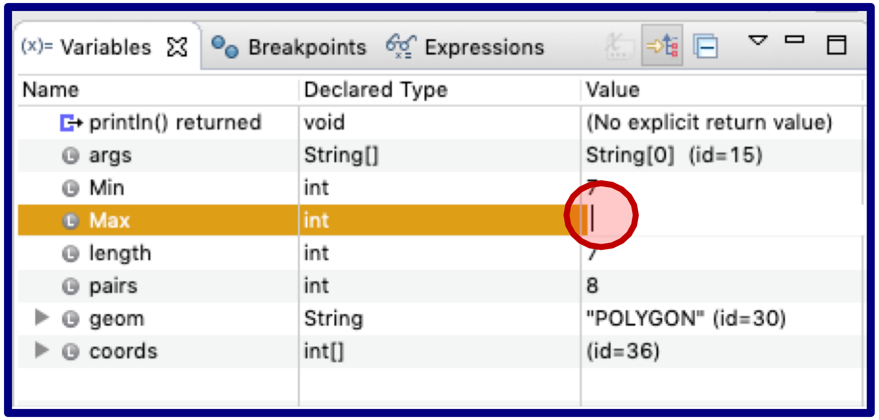


Variable View

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

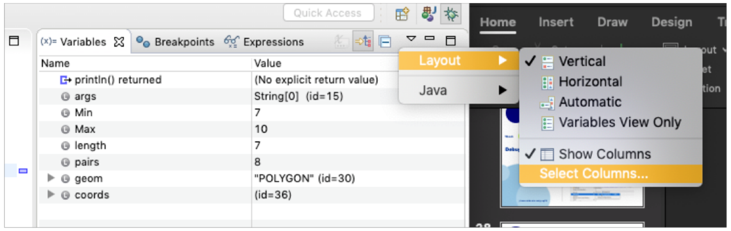
Week 2

Variable View

The viewport also allows you to customize what is displayed for each variable. For example say you wanted to know the TYPE:

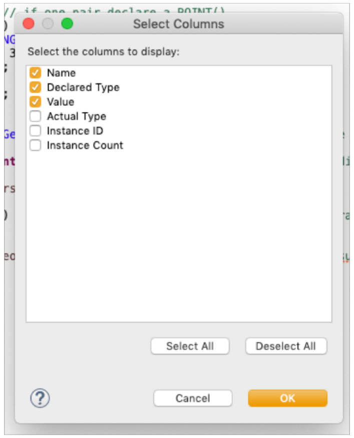
Go: Layout → Select Columns → Type

(1)



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)

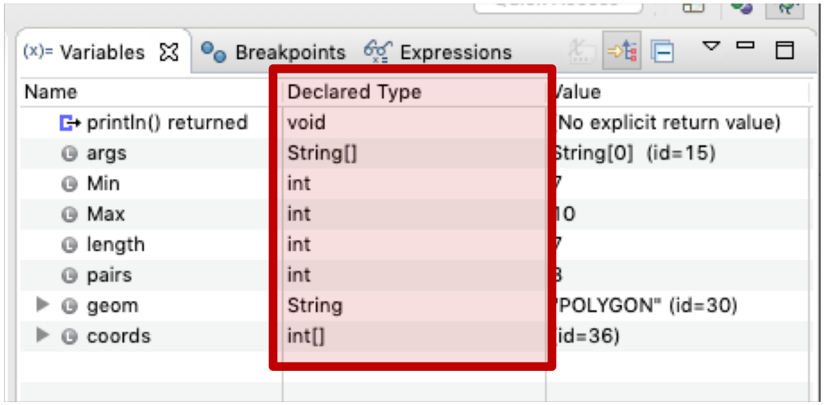
(2)



Select the columns to display:

☑ Name
☑ Declared Type
☑ Value
□ Actual Type
□ Instance ID
□ Instance Count

(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	8
geom	String	"POLYGON" (id=30)
coords	int[]	id=36)

Your Turn

Week 2

Example 3

- Take some time to fix the broken logic in Example3_buggy.java

- You can do this:

  1. Visually
  2. With the debugger
  3. By hand
  4. ???

Why did we do this??

Week 2

Big Picture

- In this example you worked to correct **WORKING** by **BUGGY** code…
- 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

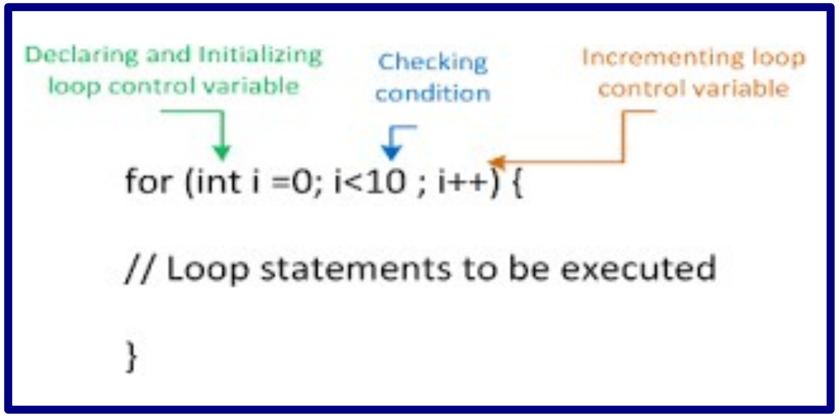
- Loops are sequences of instructions to be continually repeated until a specific condition is reached.
- They are helpful when checking for a condition or when repeating the same process over a large amount of data points…
- Anytime you want to do something many times a loop will be helpful!

For Loops and While Loops

Week 2

Loops

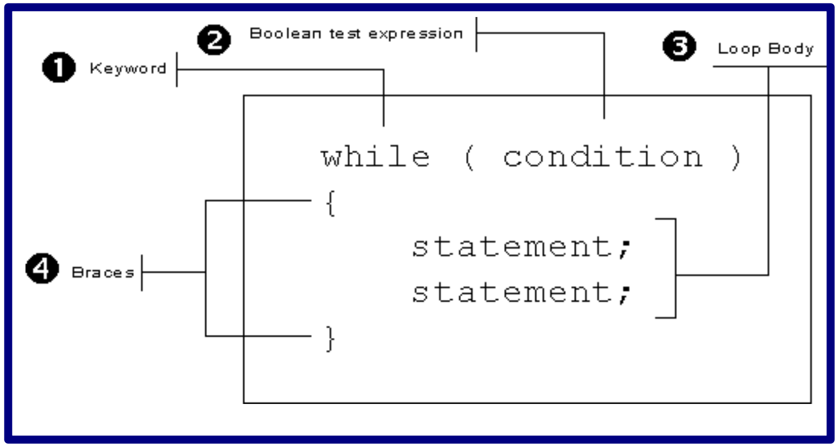
- FOR LOOP SYNTAX



```
for (int i =0; i<10 ; i++) {

// Loop statements to be executed

}
```

Declaring and Initializing loop control variable
Checking condition
Incrementing loop control variable

- WHILE LOOP SYNTAX



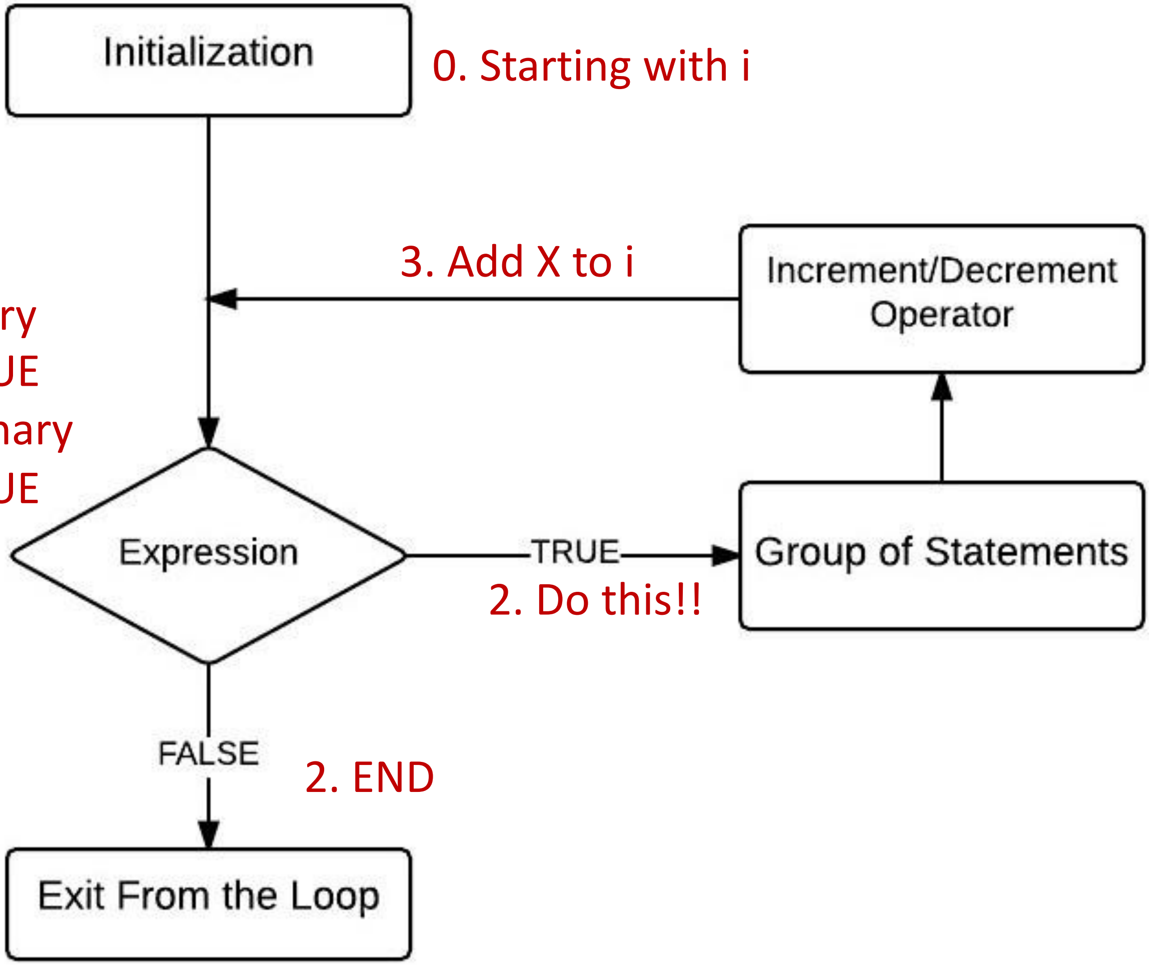
```
while ( condition )
{
        statement;
        statement;
}
```

Loop Logical Flowchart

Week 2

Loops

1. Check **if** binary condition is TRUE (or do **while** Binary condition is TRUE



0. Starting with i

3. Add X to i

2. Do this!!

2. END

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, or a 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 of visit office hours!