GEOG 178/258
Week 2:

Variables, Loops, and debugging

mike johnson

OVERVIEW

Week 2

Contents

1. Variables

Week 2

Variables

What are 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.

Week 2

Variables

Variable Types

- 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

Week 2

Variables

Variable Types

- 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

- 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 **pull** the new data

Complete Workflow:

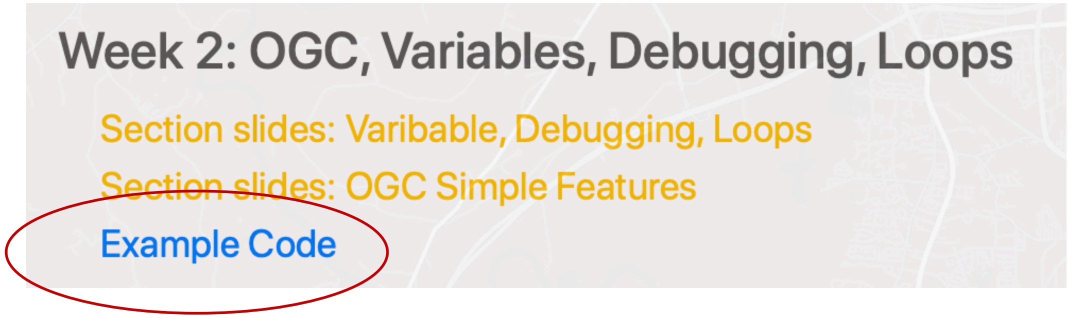
Do once:

```
> 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

2. Examples

Importing an Existing Project

Week 2

Importing

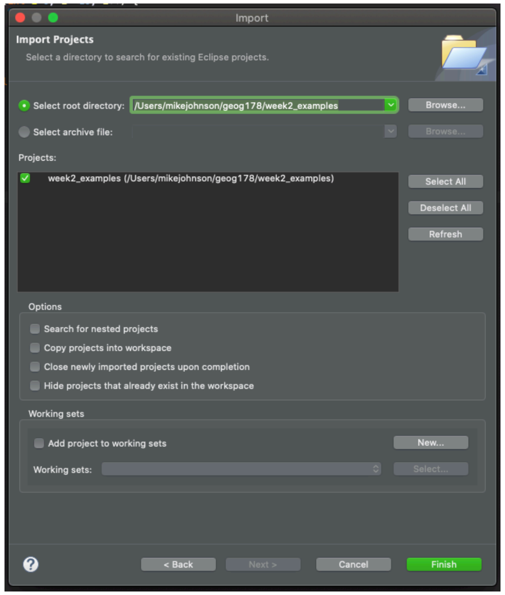
- Open an Eclipse workspace on your flash drive or local desktop
- Go to: File → Import → General → Existing
    - Select ”Select root directory”
    - Click ‘Browse’
    - Point it to the ‘Week2_examples’ folder
    - Click ‘Finish’

Importing an Existing Project

Week 2

Importing

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



Import

Import Projects

Select a directory to search for existing Eclipse projects.

● Select root directory: /Users/mikejohnson/geog178/week2_examples

○ Select archive file:

Projects:

✓ week2_examples (/Users/mikejohnson/geog178/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:

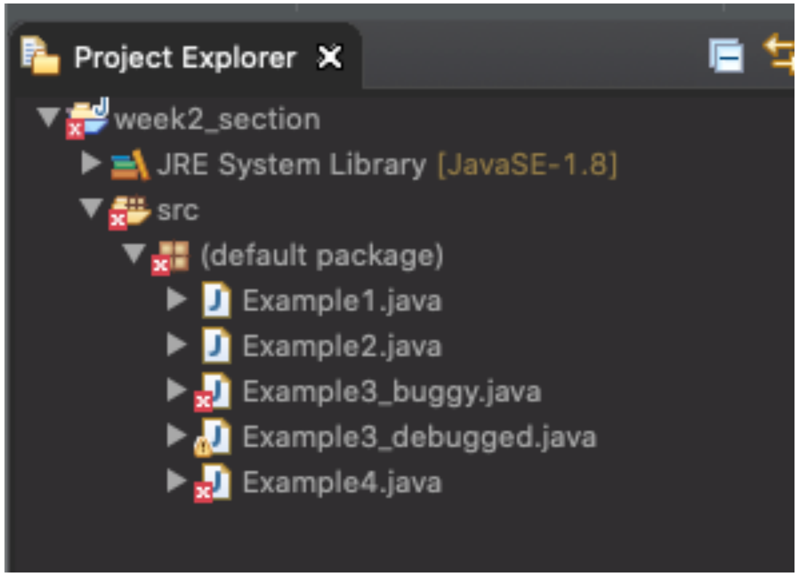
- Click 'Finish'

Importing an Existing Project

Week 2

Importing

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



Project Explorer

```
▼ week2_section
  ▶ JRE System Library [JavaSE-1.8]
  ▼ src
    ▼ (default package)
      ▶ Example1.java
      ▶ Example2.java
      ▶ Example3_buggy.java
      ▶ Example3_debugged.java
      ▶ Example4.java
```

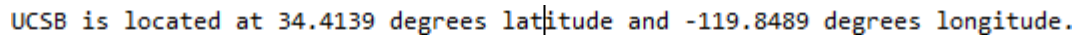
- **Create a new class called `My_Example1`**

Java…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.
```

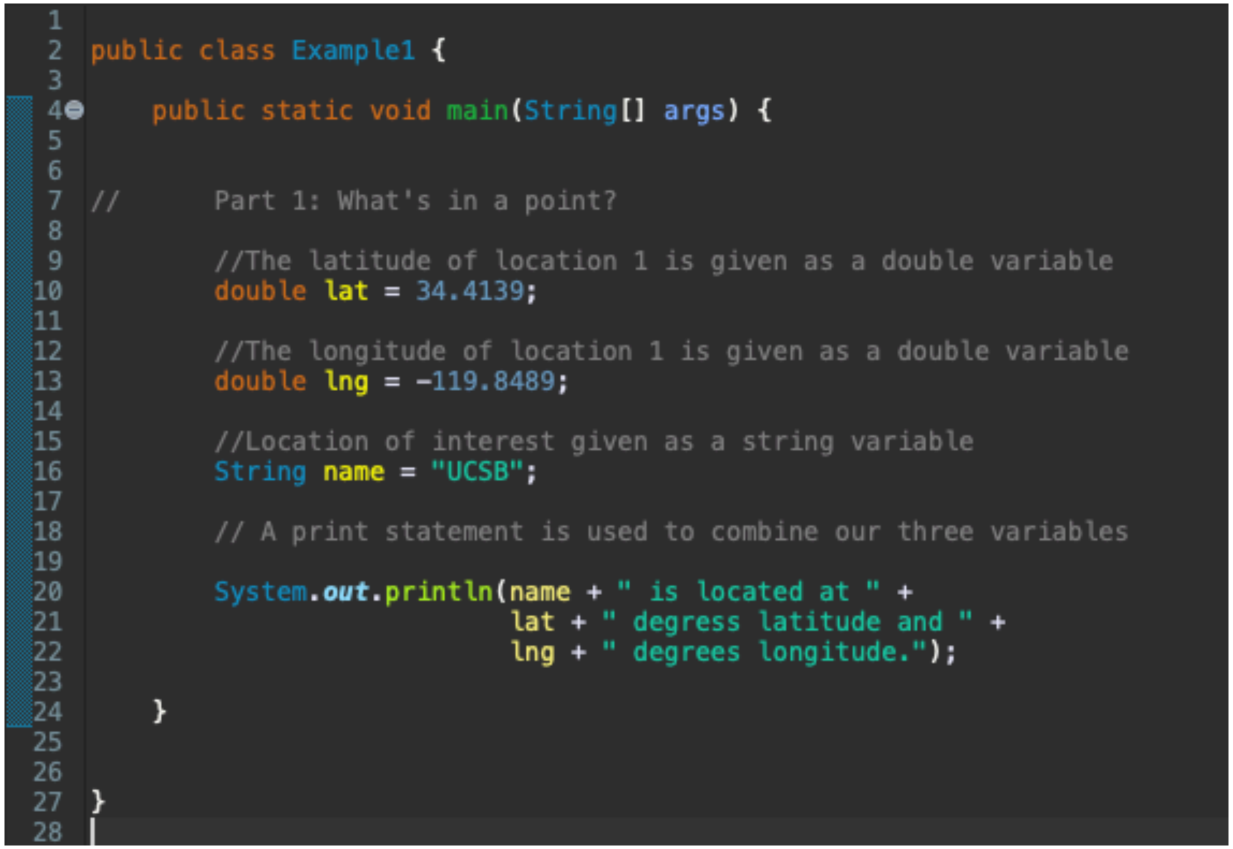
- In this program make location name, lat and long variables variables that can be changed

- (Answer on the next slide and in Example1.java)

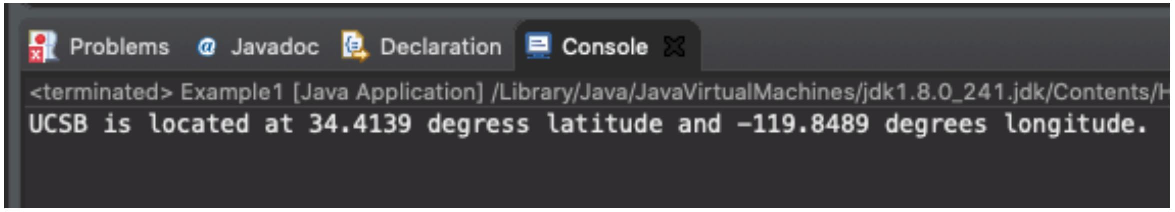
Where is UCSB (simple program)

Week 2

Example #1



```

public class Example1 {

    public static void main(String[] args) {


//      Part 1: What's in a point?

        //The latitude of location 1 is given as a double variable
        double lat = 34.4139;

        //The longitude of location 1 is given as a double variable
        double lng = -119.8489;

        //Location of interest given as a string variable
        String name = "UCSB";

        // A print statement is used to combine our three variables

        System.out.println(name + " is located at " +
                           lat + " degress latitude and " +
                           lng + " degrees longitude.");

    }


}

```



```
<terminated> Example1 [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_241.jdk/Contents/H
UCSB is located at 34.4139 degress 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 school from UCSB? (more complex program)

Week 2

Example #2

Angles must be in radians!!

Haversine formula:

$a = \sin^2(\Delta LAT/2) + \cos(LAT1) \cdot \cos(LAT2) \cdot \sin^2(\Delta LNG/2)$

$c = 2 \cdot \text{atan2}(\sqrt{a}, \sqrt{(1-a)})$

$d = 6{,}371 \cdot c$

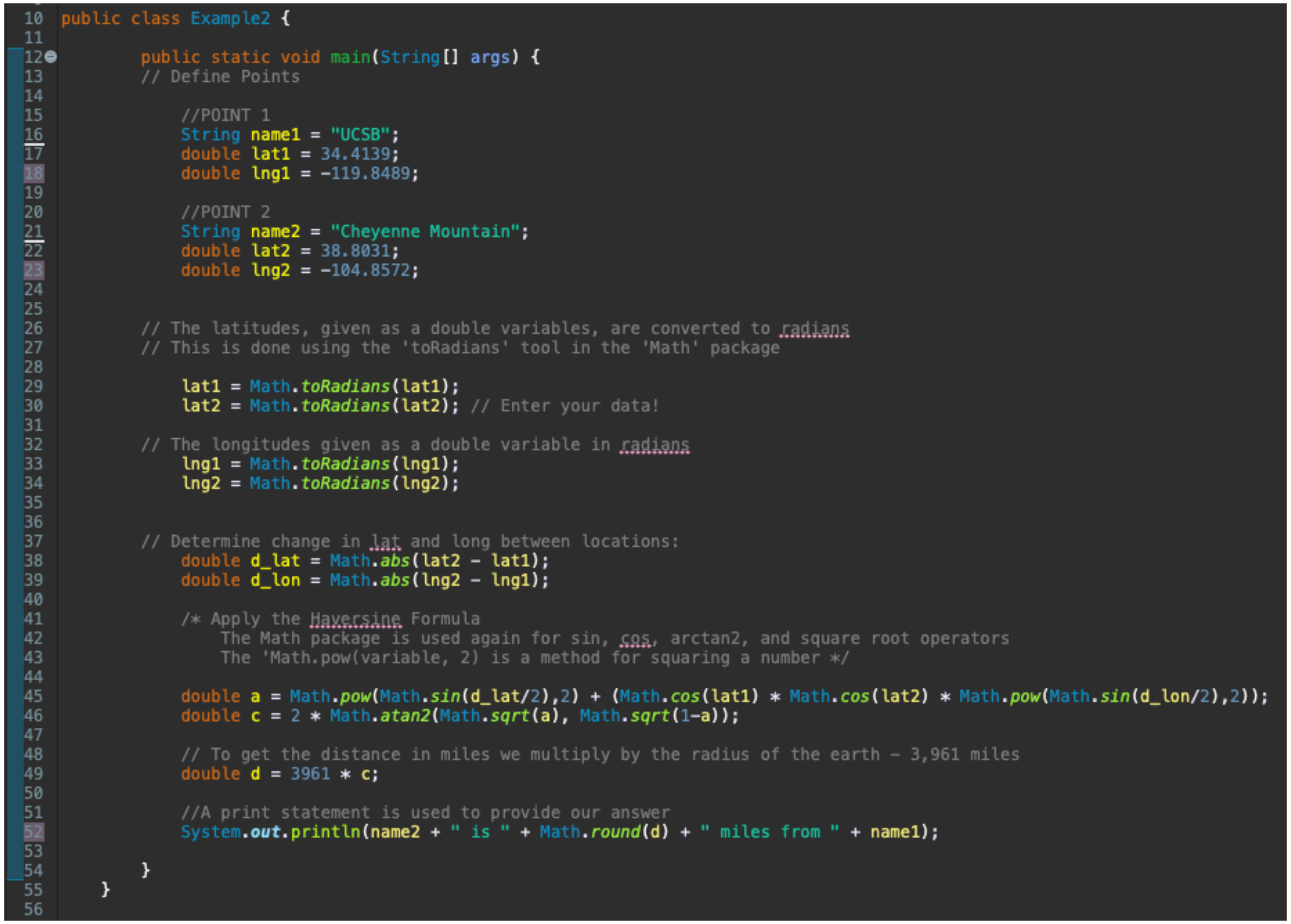
Give it a try!

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

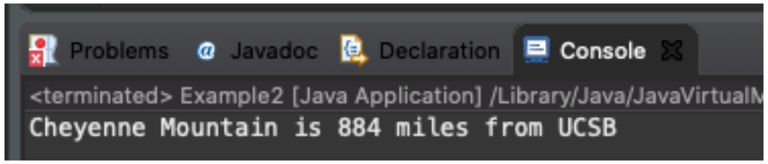
How far is your high school from UCSB? (more complex program)

Week 2

Example #2



```java
public class Example2 {

        public static void main(String[] args) {
        // Define Points

            //POINT 1
            String name1 = "UCSB";
            double lat1 = 34.4139;
            double lng1 = -119.8489;

            //POINT 2
            String name2 = "Cheyenne Mountain";
            double lat2 = 38.8031;
            double lng2 = -104.8572;


        // The latitudes, given as a double variables, are converted to radians
        // This is done using the 'toRadians' tool in the 'Math' package

            lat1 = Math.toRadians(lat1);
            lat2 = Math.toRadians(lat2); // Enter your data!

        // The longitudes given as a double variable in radians
            lng1 = Math.toRadians(lng1);
            lng2 = Math.toRadians(lng2);


        // Determine change in lat and long between locations:
            double d_lat = Math.abs(lat2 - lat1);
            double d_lon = Math.abs(lng2 - lng1);

            /* 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.println(name2 + " is " + Math.round(d) + " miles from " + name1);

        }
    }
```



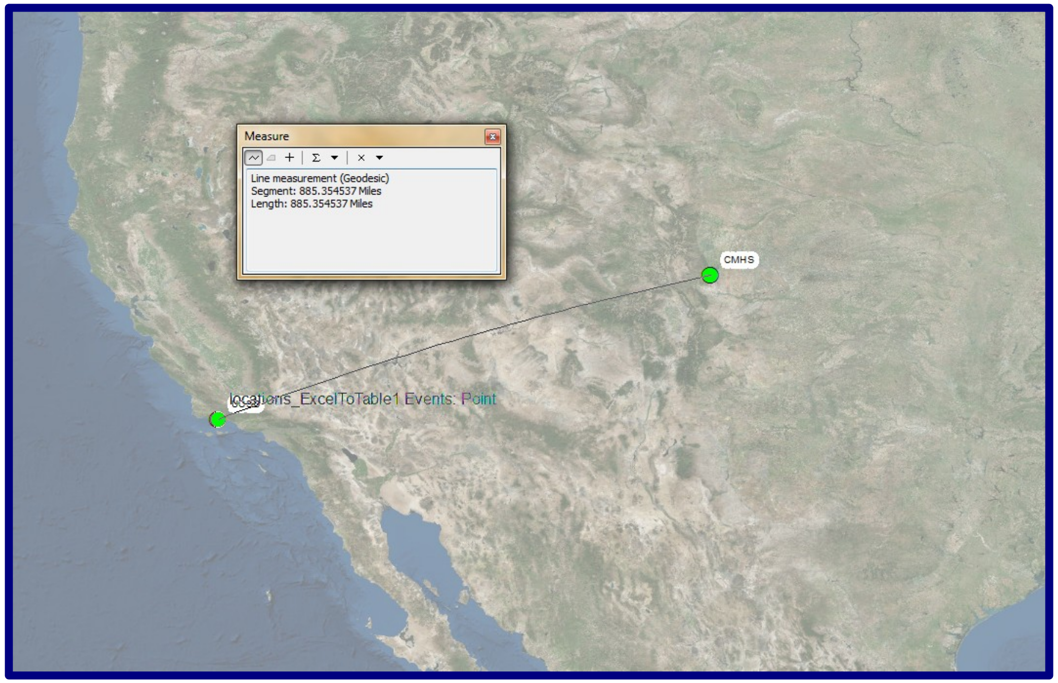
```
<terminated> Example2 [Java Application] /Library/Java/JavaVirtualM
Cheyenne Mountain is 884 miles from UCSB
```

Validation?

Week 2

Example #2

- Validation using ArcMap



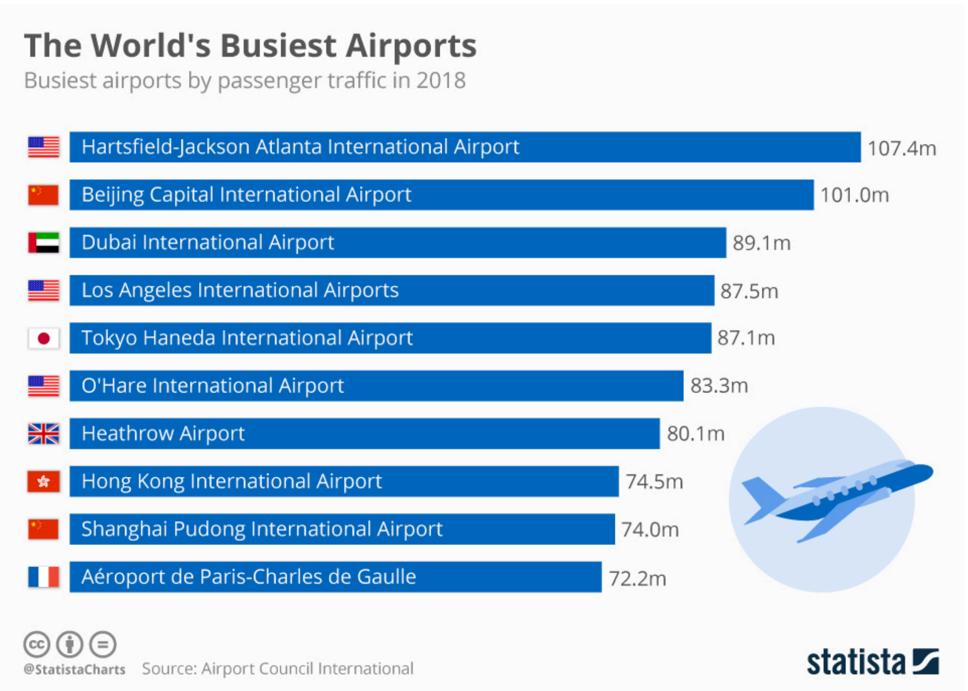
```
Line measurement (Geodesic)
Segment: 885.354537 Miles
Length: 885.354537 Miles
```

- Percent Difference:
  - [885.3545 – 884.2627) / 885.3545] * 100 = .12%

What's the minimum??

Week 2

Example #3



Create a new class called My_Example3.java

Create four double variables to store the latitudes of the four busiest airports:

ATL = 33.6407
BEJ = 40.0799
DXB = 25.2532
LAX = 33.9416

Using Math.min()…. Produce the statement:

“The lowest latitude of the worlds four busiest airports is _____”

3. Loops

What are Loops??

Week 2

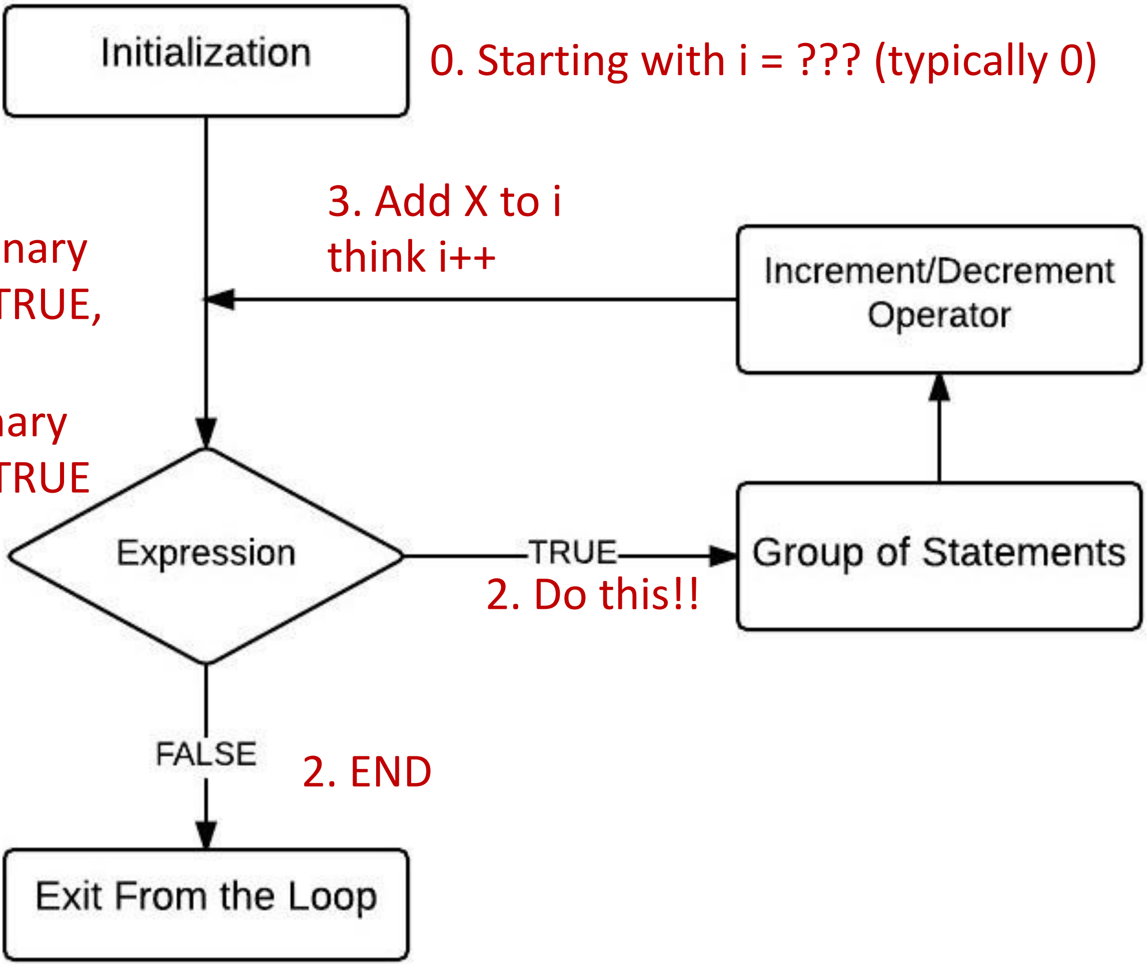
Loops

- 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!

Loop Logical Flowchart

Week 2

Loops



0. Starting with i = ??? (typically 0)

1. Check **if** binary condition is TRUE,
- Or –
do **while** Binary condition is TRUE

2. Do this!!

2. END

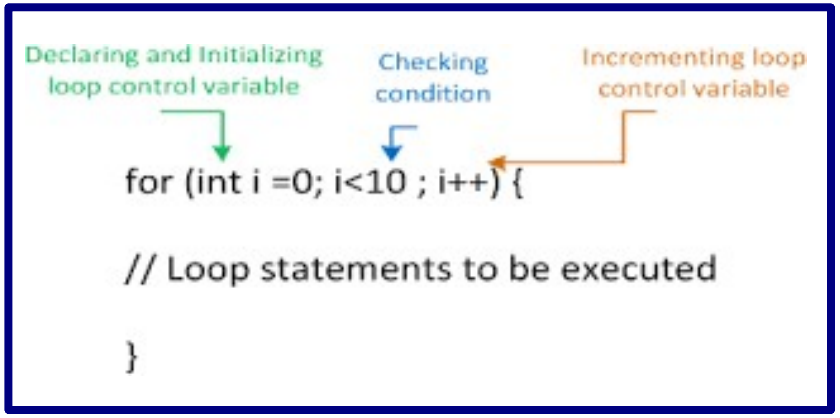
3. Add X to i
think i++

For Loops and While Loops

Week **2**

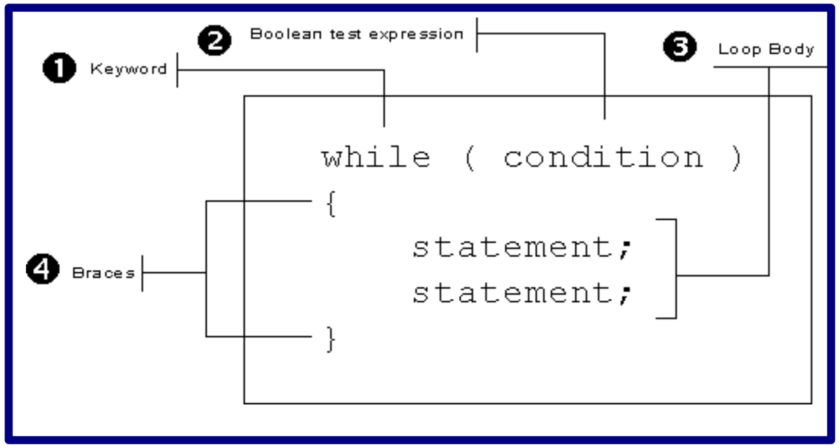
Loops

- FOR LOOP SYNTAX


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

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

// Loop statements to be executed

}
```

- WHILE LOOP SYNTAX

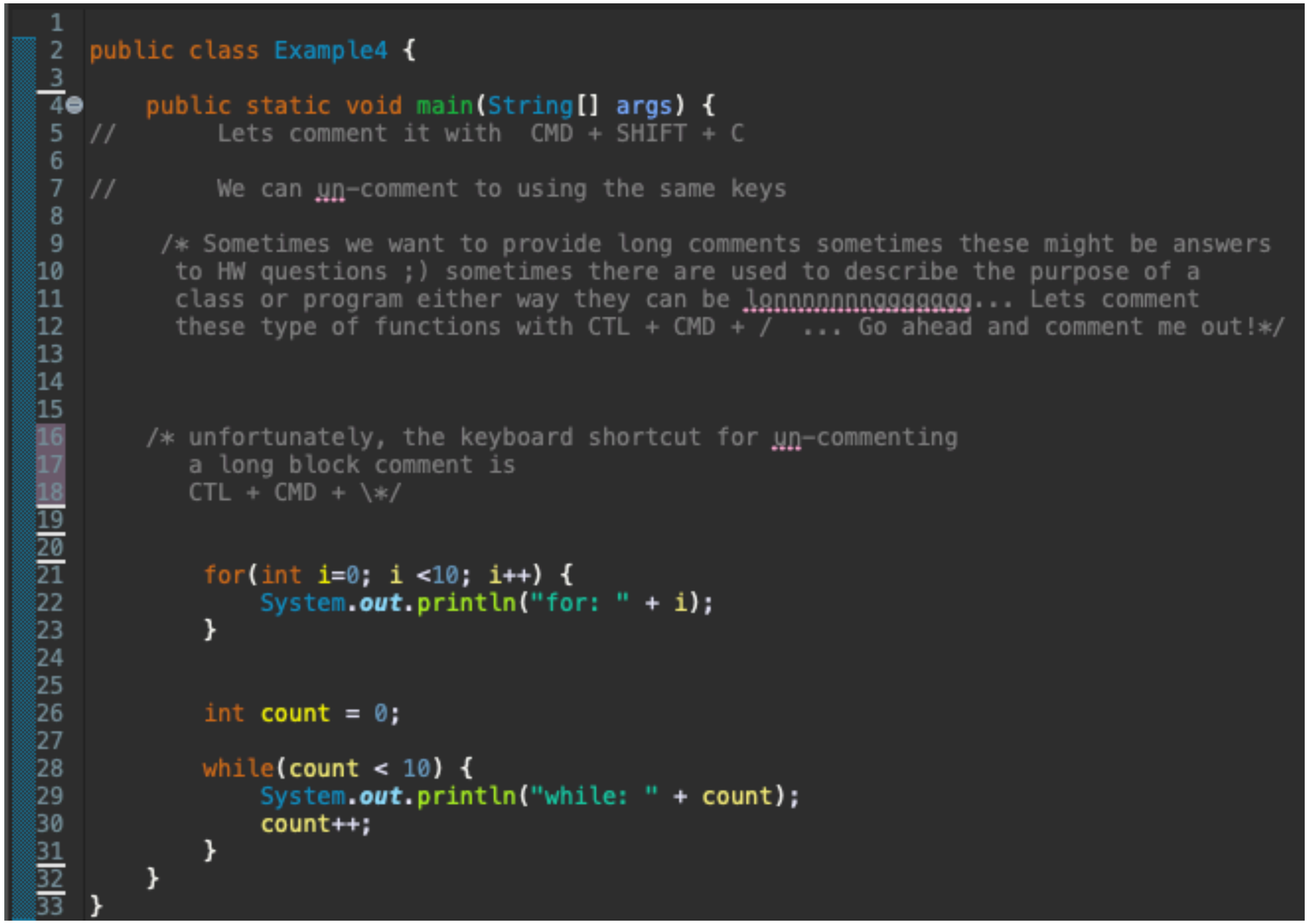


```
while ( condition )
{
      statement;
      statement;
}
```

Building Loops
(Example Code with comments…)

Week **2**

Example #4



```
public class Example4 {

    public static void main(String[] args) {
//      Lets comment it with  CMD + SHIFT + C

//      We can un-comment to using the same keys

        /* Sometimes we want to provide long comments sometimes these might be answers
         to HW questions ;) sometimes there are used to describe the purpose of a
         class or program either way they can be lonnnnnnnnggggggg... Lets comment
         these type of functions with CTL + CMD + /   ... Go ahead and comment me out!*/



        /* unfortunately, the keyboard shortcut for un-commenting
           a long block comment is
           CTL + CMD + \*/


            for(int i=0; i <10; i++) {
                System.out.println("for: " + i);
            }


            int count = 0;

            while(count < 10) {
                System.out.println("while: " + count);
                count++;
            }
        }
}
```

3. Debugging

Debugging

Week 2

Debugging

- It is very easy, and natural, to make mistakes when programing
- There are several ways to find mistakes:
  1. Visually
  2. Working/reading the program backwards
  3. Debugging
- In Eclipse, debugging allows you to run a program INTERACTIVLY (much like R or python) while watching the source code and the variables as it executes
- Eclipse 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

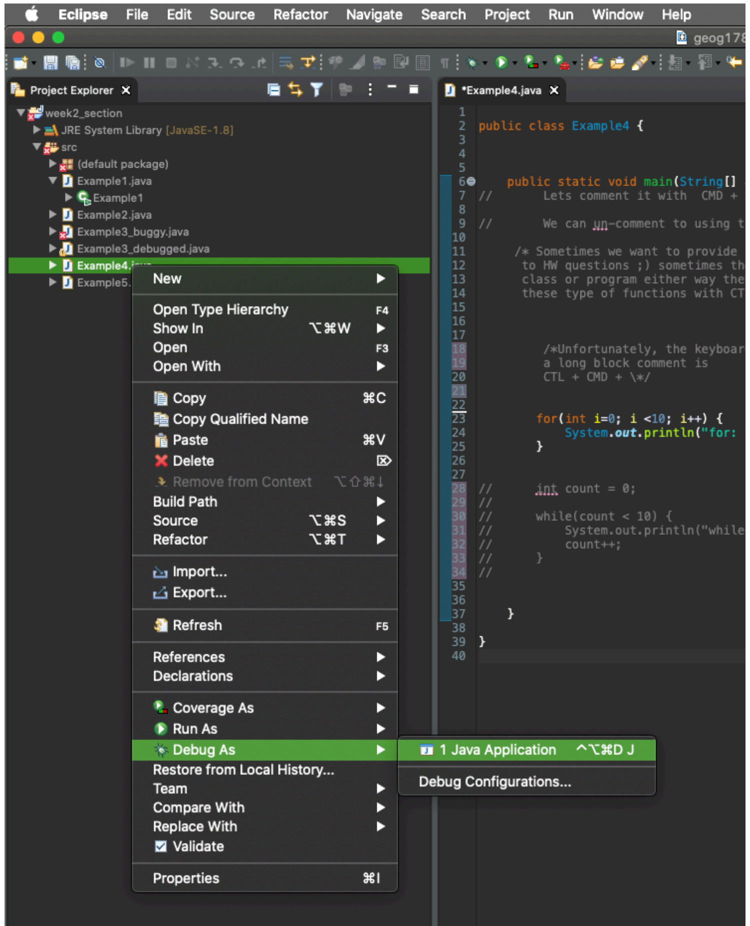
1. Missing Semicolons
2. Typos
3. Wrong Variable Types
4. Uneven brackets, parentheses, etc.
5. Missing package extensions

Week 2

Debugging

Starting the debugger

- To begin debugging a Java File Right click on the ‘Example4.java’ file and select:
- Debug As → Java Application



```

public class Example4 {



    public static void main(String[]
//          Lets comment it with  CMD +

//          We can un-comment to using t

        /* Sometimes we want to provide
         to HW questions ;) sometimes th
         class or program either way the
         these type of functions with CT



            /*Unfortunately, the keyboar
            a long block comment is
            CTL + CMD + \*/


            for(int i=0; i <10; i++) {
                System.out.println("for:
            }


//          int count = 0;
//
//          while(count < 10) {
//              System.out.println("while
//              count++;
//          }
//


    }

}

```

Adding/Removing Breakpoints

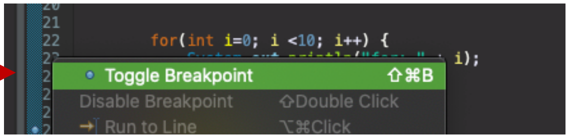
Week **2**

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"

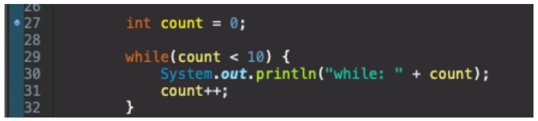
Line number



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

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



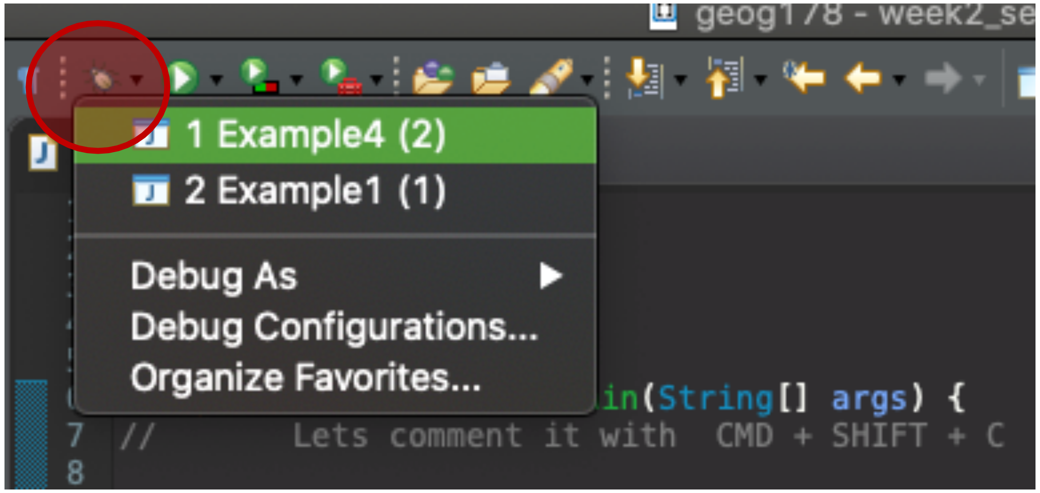
```
        int count = 0;

        while(count < 10) {
            System.out.println("while: " + count);
            count++;
        }
```

Week 2

Debugging

Starting the debugger

- 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 (add a breakpoint to the for loop print message by RIGHT clicking on the respective line number)
- Run the debugger (clicking on the 'bug') for the desired file…



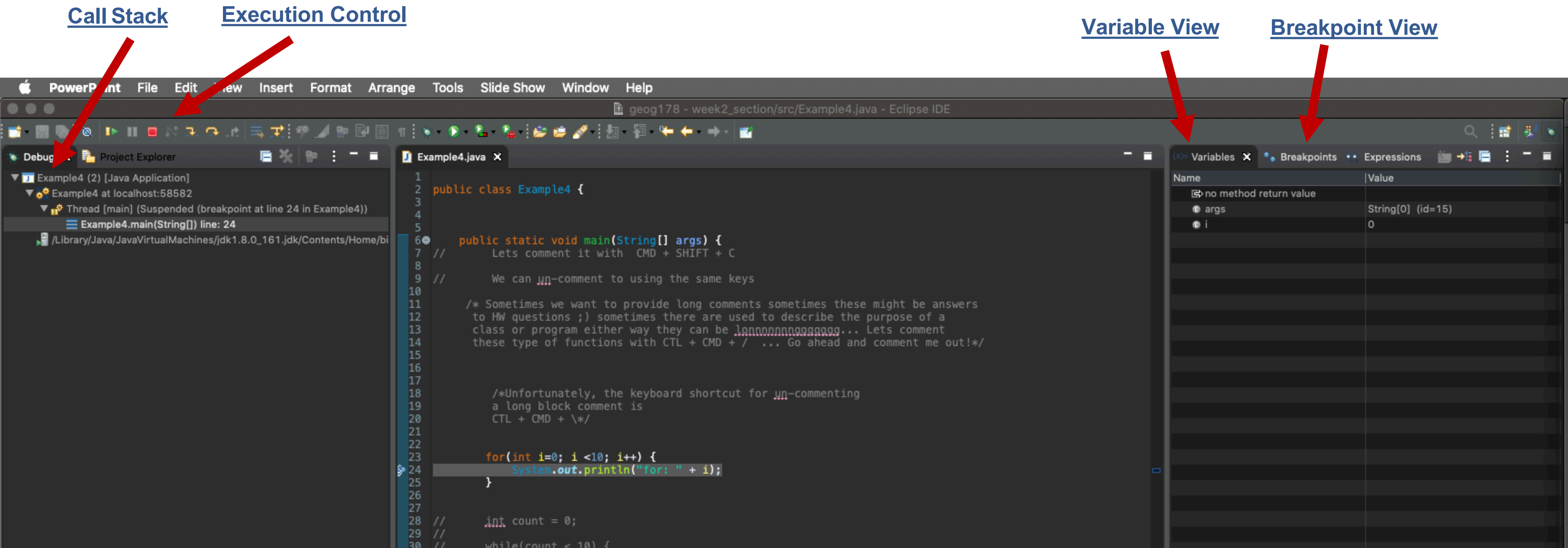
```
in(String[] args) {
//        Lets comment it with  CMD + SHIFT + C
```

The Debugger Perspective

Week 2

Debugging

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


Call Stack
Execution Control
Variable View
Breakpoint View
PowerPoint File Edit View Insert Format Arrange Tools Slide Show Window Help
geog178 - week2_section/src/Example4.java - Eclipse IDE
Debug
Project Explorer
Example4.java
Variables
Breakpoints
Expressions

```
Example4 (2) [Java Application]
  Example4 at localhost:58582
    Thread [main] (Suspended (breakpoint at line 24 in Example4))
      Example4.main(String[]) line: 24
  /Library/Java/JavaVirtualMachines/jdk1.8.0_161.jdk/Contents/Home/bi
```

```

public class Example4 {



    public static void main(String[] args) {
//        Lets comment it with  CMD + SHIFT + C

//        We can un-comment to using the same keys

     /* Sometimes we want to provide long comments sometimes these might be answers
      to HW questions ;) sometimes there are used to describe the purpose of a
      class or program either way they can be lonnnnnnnggggggg... Lets comment
      these type of functions with CTL + CMD + /  ... Go ahead and comment me out!*/



        /*Unfortunately, the keyboard shortcut for un-commenting
        a long block comment is
        CTL + CMD + \*/


        for(int i=0; i <10; i++) {
            System.out.println("for: " + i);
        }


//      int count = 0;
//
//      while(count < 10) {
```

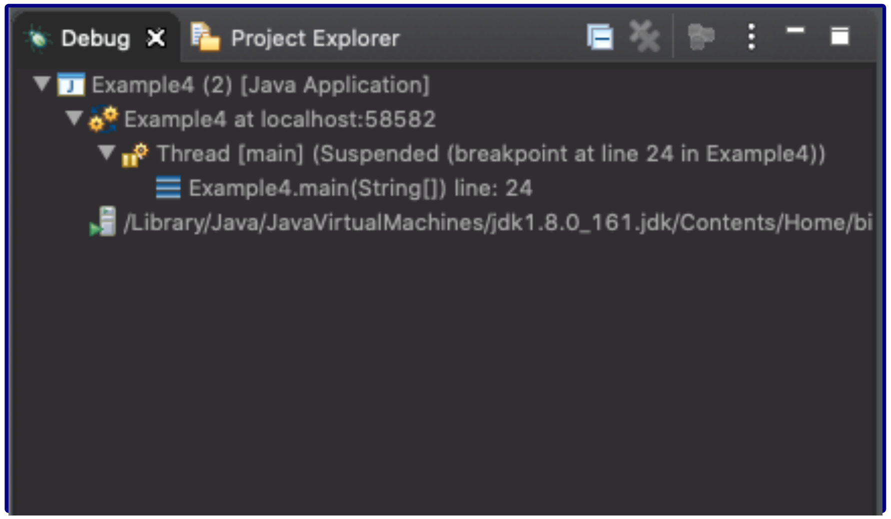
Name	Value
no method return value	
args	String[0] (id=15)
i	0

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.



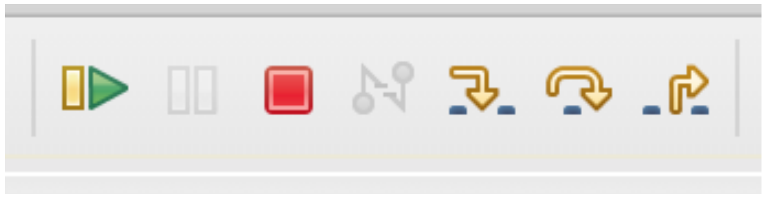
```
Example4 (2) [Java Application]
  Example4 at localhost:58582
    Thread [main] (Suspended (breakpoint at line 24 in Example4))
      Example4.main(String[]) line: 24
  /Library/Java/JavaVirtualMachines/jdk1.8.0_161.jdk/Contents/Home/bi
```

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.

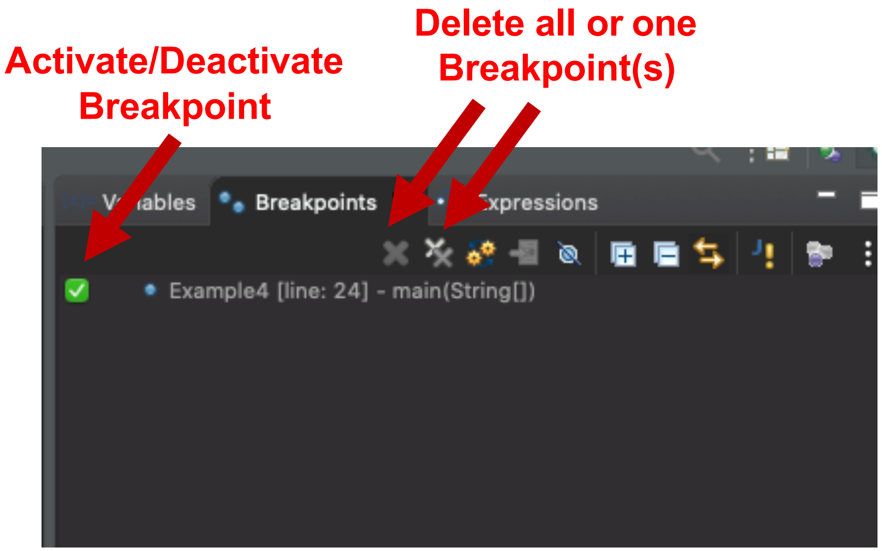
Always Terminate your debugger when done!!

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.


Activate/Deactivate Breakpoint
Delete all or one Breakpoint(s)
Variables
Breakpoints
Expressions

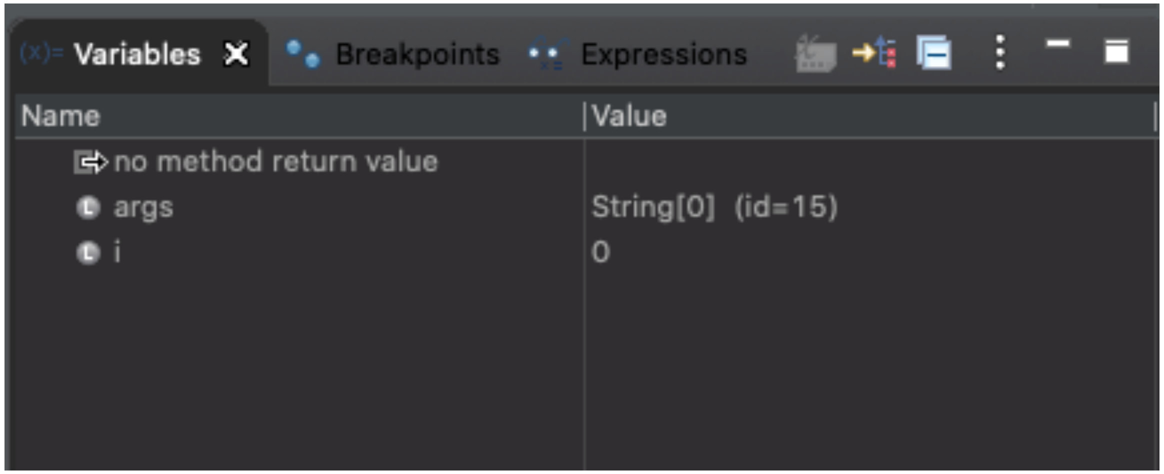
Example4 [line: 24] - main(String[])

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…



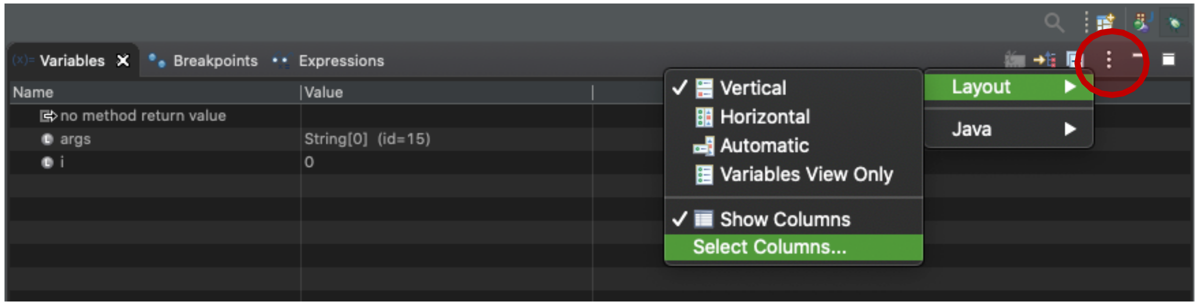
Name	Value
no method return value	
args	String[0] (id=15)
i	0

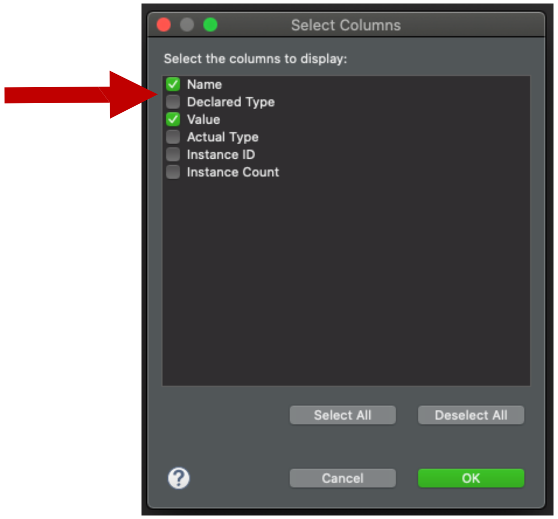
Variable View

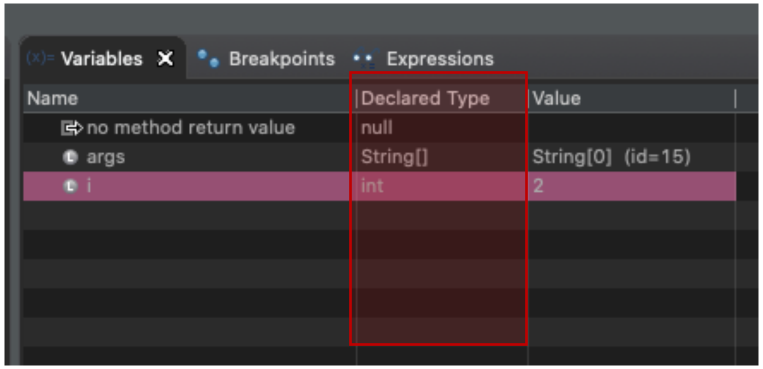
Week 2

Variable View

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







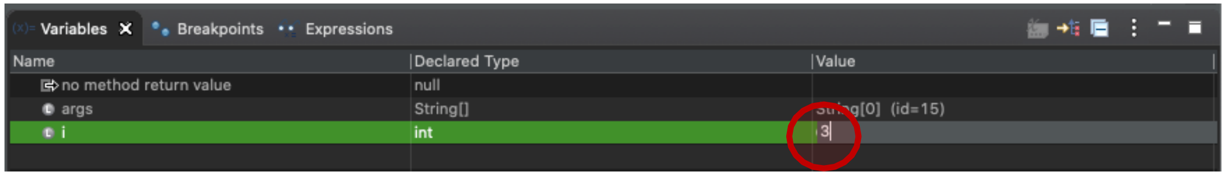
Name	Declared Type	Value
no method return value	null	
args	String[]	String[0] (id=15)
i	int	2

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)



Variables | Breakpoints | Expressions

Name	Declared Type	Value
no method return value	null	
args	String[]	String[0] (id=15)
i	int	3

Go ahead and use the execution control to get a sense of the debugger and the for loop:

1. Step into methods and back out
2. Execute lines
3. When done, TERMINATE!

Your Turn

Week 2

Example 5

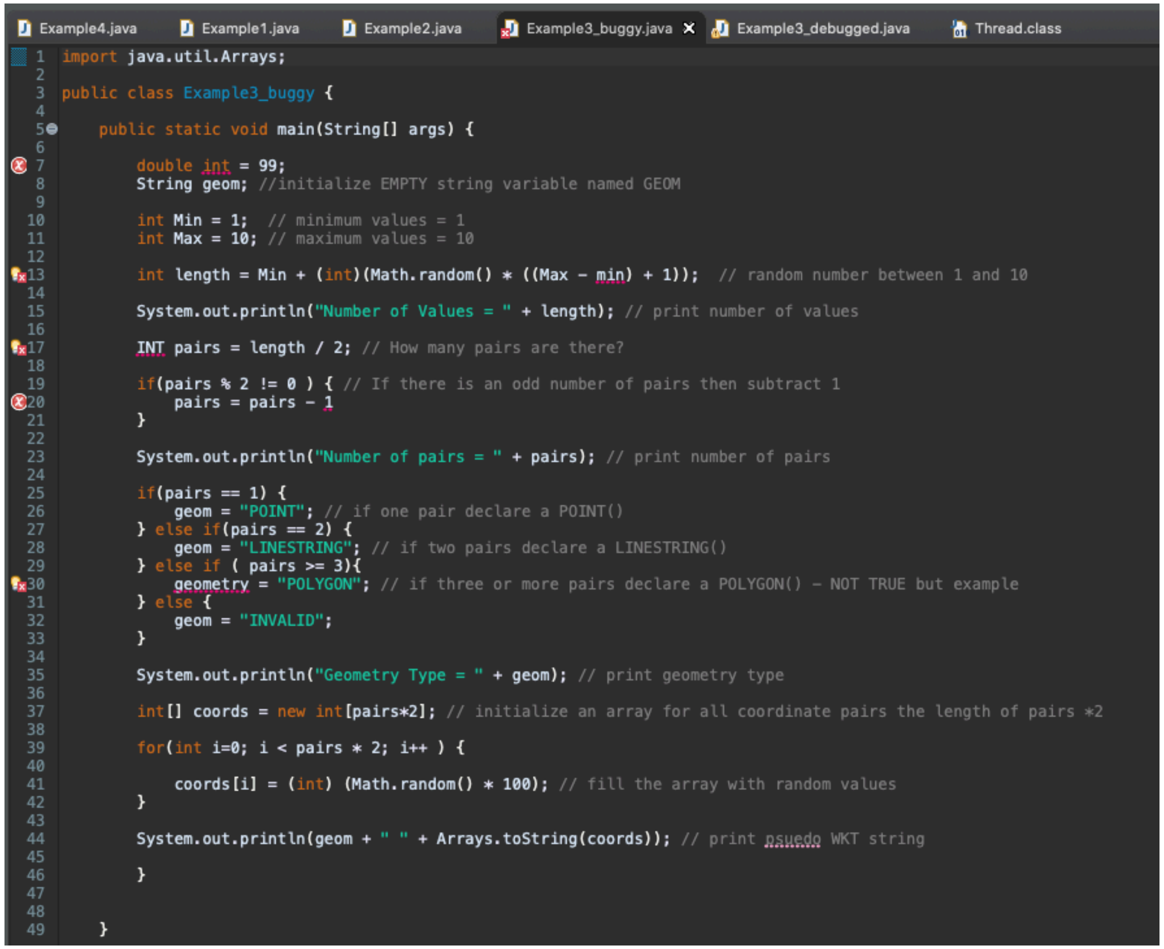
- Together, lets take some time to fix the Example5_buggy file. Do not open the debugged file yet!!

Week 2

Example 5

Debugging Practice

First, get the program to run by fixing the issues indicated by red X’s….



```java
import java.util.Arrays;

public class Example3_buggy {

    public static void main(String[] args) {

        double int = 99;
        String geom; //initialize EMPTY string variable named GEOM

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

        int length = Min + (int)(Math.random() * ((Max - min) + 1));  // random number between 1 and 10

        System.out.println("Number of Values = " + length); // print number of values

        INT pairs = length / 2; // How many pairs are there?

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

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

        if(pairs == 1) {
            geom = "POINT"; // if one pair declare a POINT()
        } else if(pairs == 2) {
            geom = "LINESTRING"; // if two pairs declare a LINESTRING()
        } else if ( pairs >= 3){
            geometry = "POLYGON"; // if three or more pairs declare a POLYGON() - NOT TRUE but example
        } else {
            geom = "INVALID";
        }

        System.out.println("Geometry Type = " + geom); // print geometry type

        int[] coords = new int[pairs*2]; // initialize an array for all coordinate pairs the length of pairs *2

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

            coords[i] = (int) (Math.random() * 100); // fill the array with random values
        }

        System.out.println(geom + " " + Arrays.toString(coords)); // print psuedo WKT string

        }


    }
```

Problems once its running ☹

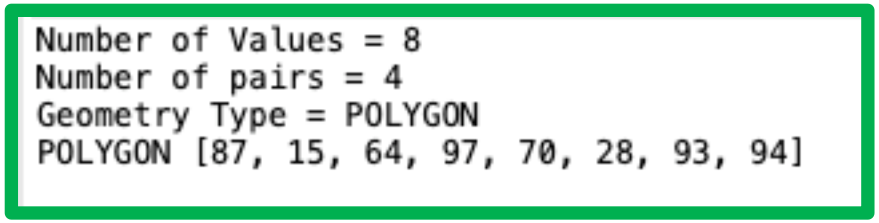
Week 2

Example #3

- Once Example3_buggy.java is running lets look at what is is suppose to do!
- 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

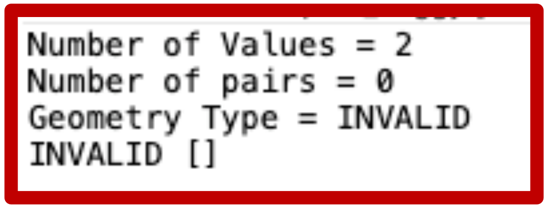
- Run the code a few times:



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

Good !!

Why does this happen??



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

1
POINT
POINT [X, Y]

Bad !!

Problems once its running ☹

Week 2

Example #5

Where does the logic fall apart?

Use any of your tricks to find the error(s?)

Why did we do this??

Week 2

Big Picture

- In this example you first fixed broken code and then work 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

Summary:

Week 2

END:

At this point you should be comfortable:

1. Launching a workspace and creating a Java Projectin Eclipse
2. Importing a program from the class website, github, or your flash
3. With the different types of variables, their uses, and howto declare them
4. Manipulating variables with the ‘Math’ package and print statements
5. Writing, and reading, ***for*** and ***while*** loops in your program
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!

Homework hints …

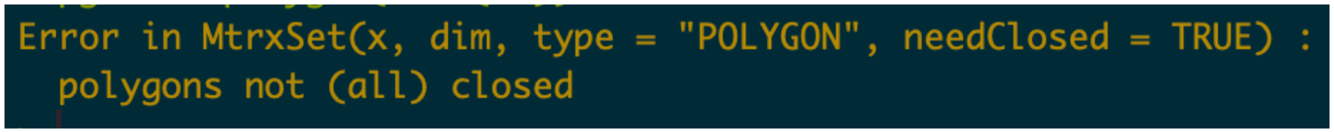
- **Part 2: Define POINT()s and determine the distance between them?**
  - How is a POINT() defined?
    - What components are needed?
  - How many POINT()s do you need to compute a distance operation?
  - How is distance in Cartesian space determined?

Homework hints …

- **Part 3a: Define a LINESTRING()…**
  - What are the minimum number of POINTS()s needed to define a LINESTRING() ?
- **Part 3b: Define a POLYGON()…**
  - What is the minimum number of unique POINT()s needed to define a POLYGON() ?
  - What is the minimum number of POINT()s needed to initialize a POLYGON() ?
- How many unique POINT()s are needed to define a valid LINESTRING() and POLYGON() ?

Homework hints …

- **Part 4: Is the POLYGON() closed … aka … is it a valid geometry??**
  - Think of this is building your own error handling
  - What conditional statement is needed to generate a warning like:



```
Error in MtrxSet(x, dim, type = "POLYGON", needClosed = TRUE) :
  polygons not (all) closed
```