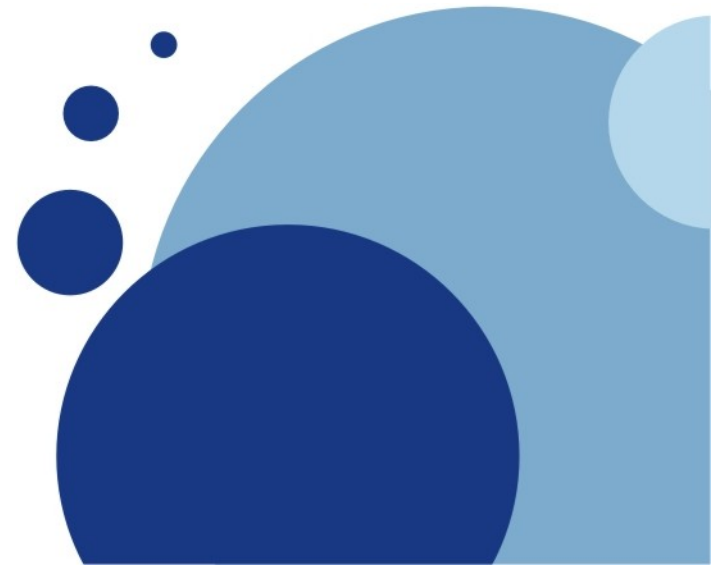


# **GEOG 178/258**

## **Week 3:**

**Arrays, Objects and Classes**

*mike johnson*



# Recap:

**Package:** collection of pieces of code that can be shared together

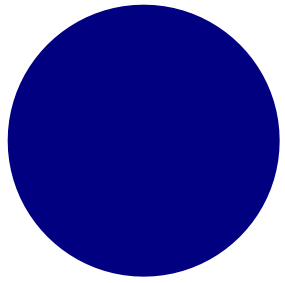
**Public:** This declares public access to a member across a package

**Void:** This means that a method has no return value

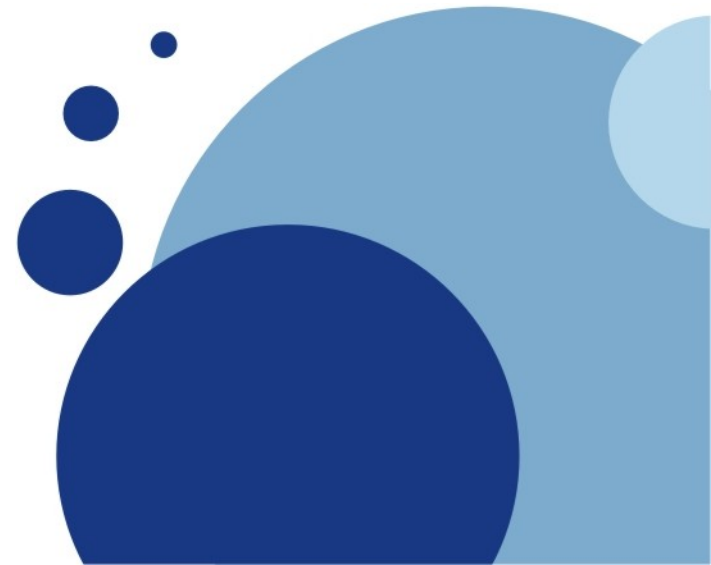
**Main:** the part of the code that will execute.

It is a static method meaning it is part of its class and not part of objects.





# 1. Arrays





# So far...

Week



3

## Arrays



- So far we have looked at initializing variables as single values:
  - For example `int x = 10`
  - This creates an integer called x that is equal to 10
- Sometimes we want a collection of data to be stored together.
  - Examples of this might be:
    1. Large data tables or matrices
    2. Lat, Long Pairs
    3. Yearly values (Jan, Feb, March values)
    4. A list of any sort
    5. What can you think of??

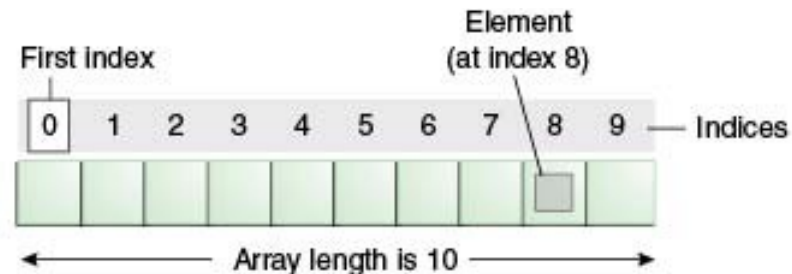
# What is an Array??

Week

3

## Arrays

- An array acts as a 'container' object to hold a fixed number of values of the same type!
- When an array is initialized the type of data it can contain AS WELL as the length must be established!
- In an array, each item is called an element, and the below diagram highlights how Java interprets an array



An array of 10 elements.

# Creating an array...

Week

3

## Arrays

1. Initializing an array is similar to initializing a variable with one difference:

```
Int[] testArray; // An array of integers is declared
```

2. The length of the array is then specified:

```
testArray = new int[10];
```

```
// memory reserved in testArray for 10 integers
```

```
// be sure to include the 'new', its significance will be covered soon...
```

3. Elements can then be set using the numerical index of the array:

```
testArray[0] = 10; // first element set to 10
```

```
testArray[1] = 20; // second element set to 20
```

```
...
```

4. Remember that Java is a 0 referenced language. This means the first entry of an array is indexed at **0 NOT 1**.



# Challenge!!

Week

3

## Arrays



1. Create an array that takes a pair of lat, long points for UCSB
  - The coordinates for UCSB are:
  - Latitude: 34.41
  - Longitude: -119.84
2. Print: **“The Latitude of UCSB is 34.41.”** using your array...
3. Print an empty line:
4. Print: **“The Longitude of UCSB is -119.84.”** again, using your array...

# Solution

Week

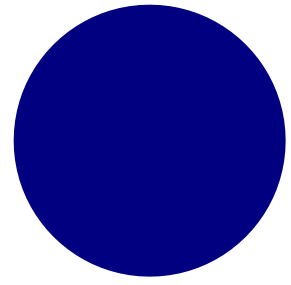
3

## Arrays

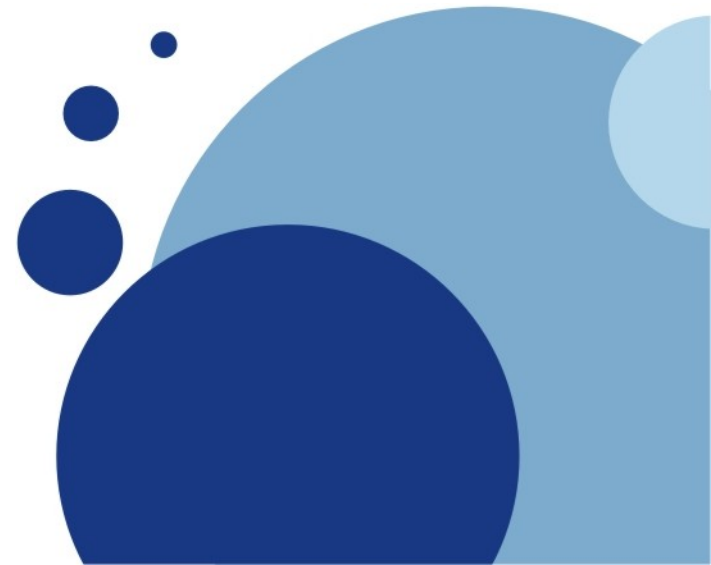
```
public class Arrays {  
    public static void main(String[] args) {  
        double [] UCSB = new double[2]; // Create a new array of 2 doubles...  
  
        UCSB[0] = 34.41; // Initialize the first element as the latitude  
        UCSB[1] = -119.84; // Initialize the second element as the longitude  
  
        System.out.print("The Latitude of UCSB is " + UCSB[0]+ ". "  
            + "\n\nThe Longitude of UCSB is " + UCSB[1] + ".");  
  
        /* note how each element can be called using the numerical index  
        also note that the \n sequence can be used to 'skip' lines  
        within the same print statement */  
    }  
}
```

```
terminated Arrays [C:\Program Files\Java\jdk1.6.0_02\bin>  
The Latitude of UCSB is 34.41.  
  
The Longitude of UCSB is -119.84.
```





# 2. Objects





# What is Java??... OOP

Week

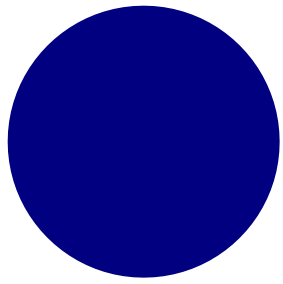


3

## Objects



- Java is an **Object-Oriented** language, or Object-Oriented Program (OOP)
- As such, it has the following characteristics:
  1. Organized around **objects** rather than 'actions'
  2. Organized around **data** rather than 'logic'
- OOP's care about the **objects** we want to manipulate rather than the logic that manipulates them.
- Other languages are often viewed as a logical sequence to process input data and produce output data. (Think R or instances of MatLab)



# What is an OBJECT??

Week

3

## Objects

- An OBJECT has **states** and **behaviors**
- **States are conditions** (think long-term/identity)
  - The apple is (red, yellow, green, big, 1/2, whole, ...)
  - The door is (wood, metal, glass, heavy, thin, ...)
  - The chair is (recliner, rocker, leather, soft, hard, ...)
- **Behaviors are 'actions'** (think short-term)
  - The apple is (hanging, falling, ripe, rotten, wet, ...)
  - The door is (open, closed, broken, working, ...)
  - The chair is (reclined, empty, taken, ...)
- **Objects are like the things we interact with everyday, there are millions and millions of objects and trying to code that into a computer language would prove very difficult...**



# 3. Classes



# What is a CLASS??

Week



3

## Classes



- One of the first jobs of a OOP is to identify the objects that need in a model, what defines them, and how they relate to each other.
  - This is known as **data modeling**
  - Once an OBJECT is identified it can be generalized into its core components...
- In Java this generalized form is the CLASS.
  - It describes the relevant **states** and **behaviors**
  - Defines the kind of data it contains and the logical sequences that can **manipulate** it
  - It provides methods for **getting** and **changing** these states and behaviors
  - Each logic sequence in a class is known as a **method**.

# The Blueprints within Java

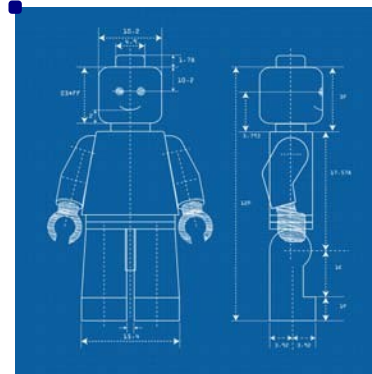
Week

3

## Classes

### Ways to think of Classes....

- In Java classes are templates for describing an object....
- It is the generic form of an object and describes what it “means to be that object”
- It is an object with “Open Variables”
- It is a form that needs to be filled out ...



# Plato's 'Doctrine of the Forms'...

Week

3

**Classes**

- Consider a chair...
- If I ask you to think of a chair, each of you knows what I am talking about.
- If I asked you to sit in a chair you won't mistakenly sit on a table.
- If I asked you to build me a chair you would all be able to.
- Given there are so many type of chairs, how do you all know what I am taking about???



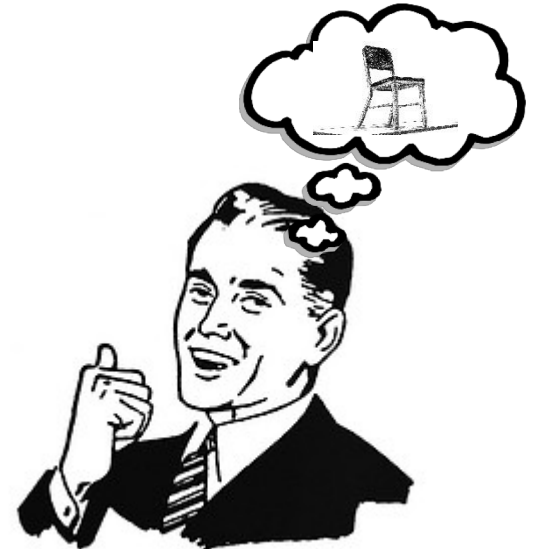
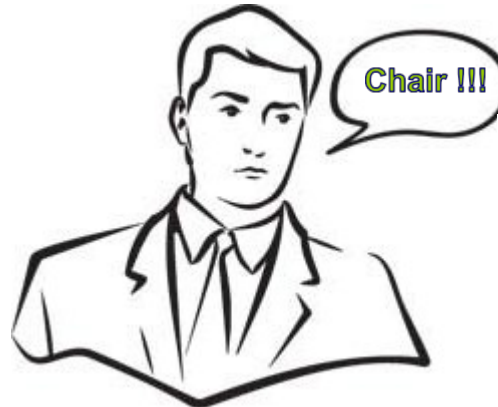
# Plato's 'Doctrine of the Forms'...

Week

3

Classes

- If 'chair' is to have a definite meaning there must be something common to **ALL** chairs...
- The thing that is common to all chairs is considered "the form of the chair"
- The form of the 'chair' allows people to communicate:







# So...

Week

3

## Classes



An object is a specific instance of a more general idea deconstructed into a sum of parts

If you look around the class you see that all your peers are 'people'. None of us are 'aliens', 'dogs' or 'cats'.

However your brain is also able to detect characteristics that make each of us unique, and specific 'objects' of the class 'people'.

For example we all have a gender, hair color, age, race, and occupation, ect that make us unique.

All 'people' have these things (this is the class people), and each object (individual) has a unique make up of these general qualities.



# So...

Week

3

## Classes



### Class/ the “Form of a Person”:

*person (“gender”, “age”, “hair color”, “occupation”, “eye color”)*

### Object/ Individuals:

*person mike = new person (male, 26, brown, grad student, green)*

Mike is a **person** who is a male grad student with brown hair and green eyes...

*person carolina = new person (female, 24, black, teacher, brown)*

Carolina is a **person** who is a female teacher with black hair and brown eyes...



# Why are CLASSES important?

Week



3

## Classes



- Classes make it possible to define subclasses of data objects that share some or all class characteristics. This feature is known as **inheritance**.
- The concept of data classes allow programmer's to **create new data types that are not pre-defined in the language**.
- Class definitions are reusable by both the program it was created for and other OOP's. This allows them to be distributed more easily across networks.
- Classes define only the data it is concerned with. Thus when an instance of that class (an OBJECT) is run, the code will not accidentally access other program data. This avoids unintended data corruptions AND enhances system security.



# 3. Practice





# Example:

In this example we will do the following:

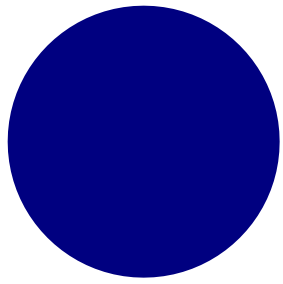
1. Create a class called point
2. Construct the general form of point
3. Create a method in the point class
4. Create two points using the point class
5. Call on the point method to perform an operation

Week

3

**Practice  
Problem**





## Example:

Week

3

## Practice Problem

1. Open a new project and create a class called 'Point'
2. Do NOT elected to create a 'main' string.

# 1. Creating a point class

Week

3

## Practice Problem

```
1
2 public class Point {
3
4     // Attributes
5
6     // Constructors
7
8     // Getters
9
10    // Setters
11
12    // Methods
13
14
15 }
```

**Line 2:** Creates a new public class called 'Point'

**Line 4:** Sets aside space to define the **attributes** of a point

**Line 6:** Sets aside space to define the **constructors** of a point

**Line 8:** Sets aside space to define the **getters** of a point

**Line 10:** Sets aside space to define the **setters** of a point

**Line 12:** Sets aside space to define the **methods** of a point

**Line 15:** Closes the point class

## 2. Defining the attributes

Week

3

### Practice Problem

```
4 // Attributes  
5  
6 double x, y;  
7
```

**Line 12:** Declares that all points are comprised of values X and Y (doubles)



## 2. Defining the constructor

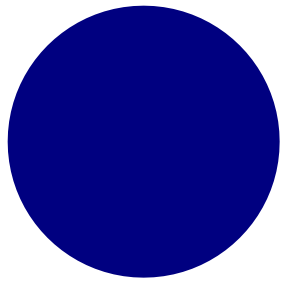
Week

3

### Practice Problem

```
4 // Attributes
5
6 double x, y;
7
8 // Constructors
9 public Point(double x, double y) {
10     this.x = x;
11     this.y = y;
12 }
13
```

**Line 9:** Defines a new public method called **Point** that takes an input x and y value (doubles) and assigns them to an internal value to that point.



## 2. Defining the getters

Getters allow you (and the program) to access elements of a class

Week

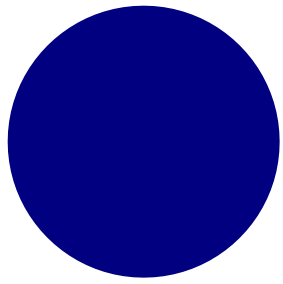
3

### Practice Problem

```
14 // Getters
15
16 public double getX() {
17     return x;
18 }
19
20 public double getY() {
21     return y;
22 }
23
```

**Line 16:** Defines a new **public** method, that can be applied to any object of class **point**, that returns a **double**, called **getX()**. When run, **getX()** returns the 'x' value associated with the object it is called on.

**Line 20:** Does the same for y.



## 2. Defining the setters

Setters allow you (or the program) to modify the attributes of an object

Week

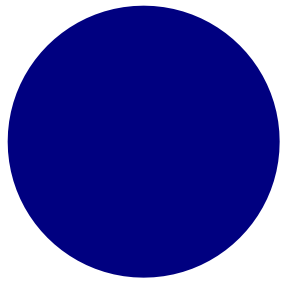
3

### Practice Problem

```
24 // Setters
25
26 public void setX(double x) {
27     this.x = x;
28 }
29
30 public void setY(double y) {
31     this.y = y;
32 }
```

**Line 26:** Creates a **public** method, that has **no return value** called **setX()**. SetX() requires an input **double 'x'** and sets that values within the point constructor to the new x value, **'this.x'**.

**Line 30:** Does the same for y



## 3. Add an method to the point class

\* Return a point as a WKT string

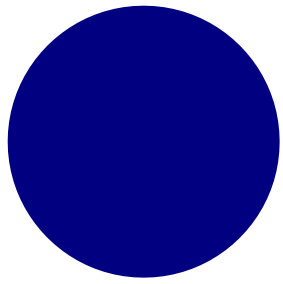
Week

3

## Practice Problem

```
34 // Methods
35
36 public String toWKT() {
37     return "POINT (" + x + " " + y + ")";
38 }
```

**Line 36:** Creates a **public** method called **toWKT()** that **returns a String**. This method uses the 'x' and 'y' values of the Point object it is called on to populate the returned string.



## 3. Add another method

\* Calculate the area between two points

Week

3

## Practice Problem

```
34 // Methods
35
36 public String toWKT() {
37     return "POINT (" + x + " " + y + ")";
38 }
39
40 public double area(double x, double y) {
41     return Math.abs(this.x - x) * Math.abs(this.y - y);
42 }
43
```

**Line 40:** Creates a **public** method called **area()** that **returns a double**. This method requires a user supplied x and y (doubles) and computes the area between them and the **x (this.x)** and **y (this.y)** values of the Point object it is called on.

# 3. Duplicate Methods for Robust Classes

\* Calculate distance between two points

Week

3

## Practice Problem

```
34 // Methods
35
36 public String toWKT() {
37     return "POINT (" + x + " " + y + ")";
38 }
39
40 public double area(double x, double y) {
41     return Math.abs(this.x - x) * Math.abs(this.y - y);
42 }
43
44 public double area(Point p) {
45     return Math.abs(this.x - p.x) * Math.abs(this.y - p.y);
46 }
```

**Line 44:** Creates a **public** method called **area()** that **returns a double**. This method requires a user supplied Point and computes the area between it (**p.x, p.y**) and the **x (this.x)** and **y (this.y)** values of the Point object it is called on.

# 4. Create Points using the point Class:

Week

3

## Practice Problem

```
2 public class Test {
3
4     public static void main(String[] args) {
5         // TODO Auto-generated method stub
6
7         Point point1 = new Point(1,3);
8
9         System.out.println("Point 1 is: " + point1.toWKT());
10        System.out.println("Point 1 is: " + point1.x + ", " + point1.y);
11        System.out.println("Point 1 is: " + point1.getX() + ", " + point1.getY());
12
13    }
14 }
15 }
```

**Line 7:** Creates a new point object in the same way as a principle type in the main() method:

**Type(point) name(P1) = new value (x,y)**

The addition is that the value needed is a set of integers as defined by the class and the addition of the 'new' statement. By doing this Java is creating a new object and attaching the reference to that object within the variable.

**Line 9:** Prints a description of point1 using the toWKT() method

**Line 10:** Prints a description of point1 using direct access

**Line 11:** Prints a description of point1 using our getters

# Test our code, apply our methods:

Week

3

## Practice Problem

```
2 public class Test {
3
4 public static void main(String[] args) {
5     // TODO Auto-generated method stub
6
7     Point point1 = new Point(1,3);
8
9     System.out.println("The X value of point 1 is: " + point1.getX());
10    System.out.println("The Y value of point 1 is: " + point1.getY());
11    System.out.println(" ");
12
13    point1.setX(5);
14    point1.setY(10);
15
16    System.out.println("The new X value of point 1 is: " + point1.getX());
17    System.out.println("The new Y value of point 1 is: " + point1.getY());
18    System.out.println(point1.toWKT());
19    System.out.println(" ");
20
21    Point point2 = new Point(1, 3);
22
23    System.out.println(point2.toWKT());
24
25    System.out.println(point1.area(1,3));
26    System.out.println(point1.area(point2));
27
28
29 }
30 }
```

```
Problems @ Javadoc Declaration Console
<terminated> Test [Java Application] /Library/Java/JavaVirtualMachin
The X value of point 1 is: 1.0
The Y value of point 1 is: 3.0

The new X value of point 1 is: 5.0
The new Y value of point 1 is: 10.0
POINT (5.0 10.0)

POINT (1.0 3.0)
28.0
28.0
```





# HW 2 Hints

Week

3

**HW #2**



1. Define a Point class (look at these notes)
2. Define a BoundingBox class
  - > Think of the attributes, constructors, getters, setters needed
3. Add a distance method to the Point class
  - > Make this robust!
4. Add an isInside method to BoundingBox
  - > Make this robust!
5. Test your code !!
  - > Be sure to think of unique instances that might break your code and test to make sure they don't



# Summary

Week



3

End



1. Variables are able to store single values
2. Java offers you a set of Primitive variable types such as int, long, double, char, float, short...
3. If you want to store a collection of values you can create an array of length x where each value is an numerically indexed element.
  - String is an example of an array of char elements
4. Java is an object oriented program (OOP) thus is is focused of the states and behaviors of objects (people, places and things...)
5. Objects are specific examples of a more generic template.
6. These templates are classes!
7. Data modeling is essentially about defining the states and behaviors of the needed classes to answer a problem!!