COMP-202
More Complex OOP

-Midterm next Tuesday, in class from 2:30-4:30

-Practice midterm Thursday in class (we'll spend about 30 minutes of class on it)

-Assignment 3 due Monday (officially)

-After lecture today, I will discuss a couple more hints on assignment 3.

-Assignment 4 will be posted soon

Defining your own types:

Remember that we can define our own types/classes.

These classes are *objects* and have *attributes* and *behaviors*

Good resource:

http://download.oracle.com/javase/tutorial/java/javaOO/index.html

You create an *object* or an instance of a class, by using the new operator.

type variablename = new *type([parameters])*

parameters can be there or not, depending on how you wrote the *constructor*

Motive to create our own type:

Let's say we have a desire in our program to use a type that isn't built in.

Example: Storing a fraction.

2nd Example: Storing an imaginary number.

Imaginary numbers

An imaginary number has 2 parts to it:

a “real” part (normal number)

and a “complex” (aka “imaginary”) part to it

Imaginary numbers appear whenever you take the square root of a negative number.

Cubic equation: If I have the equation y = ax^3 + bx^2 + cx + d, I can find the roots:

$$x = \sqrt[3]{\left(\frac{-b^3}{27a^3}+\frac{bc}{6a^2}-\frac{d}{2a}\right)+\sqrt{\left(\frac{-b^3}{27a^3}+\frac{bc}{6a^2}-\frac{d}{2a}\right)^2+\left(\frac{c}{3a}-\frac{b^2}{9a^2}\right)^3}}$$

$$+ \sqrt[3]{\left(\frac{-b^3}{27a^3}+\frac{bc}{6a^2}-\frac{d}{2a}\right)-\sqrt{\left(\frac{-b^3}{27a^3}+\frac{bc}{6a^2}-\frac{d}{2a}\right)^2+\left(\frac{c}{3a}-\frac{b^2}{9a^2}\right)^3}} - \frac{b}{3a}.$$

We know all cubic equations have 1 real solution, BUT sometimes the square root on the inside is a complex number!

$$x = \sqrt[3]{\left(\frac{-b^3}{27a^3}+\frac{bc}{6a^2}-\frac{d}{2a}\right)+\sqrt{\left(\frac{-b^3}{27a^3}+\frac{bc}{6a^2}-\frac{d}{2a}\right)^2+\left(\frac{c}{3a}-\frac{b^2}{9a^2}\right)^3}}$$

$$+ \sqrt[3]{\left(\frac{-b^3}{27a^3}+\frac{bc}{6a^2}-\frac{d}{2a}\right)-\sqrt{\left(\frac{-b^3}{27a^3}+\frac{bc}{6a^2}-\frac{d}{2a}\right)^2+\left(\frac{c}{3a}-\frac{b^2}{9a^2}\right)^3}} - \frac{b}{3a}.$$

If we want to define a type

ComplexNumber

what attributes should a ComplexNumber have?

```
public class ComplexNumber {
    double real;
    double complex;

....
}
```

Some of the useful behaviors:

add(ComplexNumber other)
subtract(ComplexNumber other)
multiply(ComplexNumber other)
equals(ComplexNumber other)
magnitude()

Using command line arguments

Ever wondered what the deal with String[] args is ?

For the most part, we have been running our programs by typing

java NameOfClassWithMainMethod

However, you can also type

java NameOfClassWithMainMethod some other words or 123 numbers

When you do this, the array args is initialized to values {“some”, “other”, “words”, “or”, “123”}

Using command line arguments

Why would I want to do this?

One potential motivation is to run your program automatically but with slightly different inputs each time.

Using command line arguments

```
public class CommandLineArgEcho {
        public static void main(String[] args) {
                System.out.println(args[0]);
        }
}

/*now when I run my program, I'll get different results*/

java CommandLineArgEcho 5 ---> prints 5
java CommandLineArgEcho hello -----> prints hello
```

Using command line arguments

This is better than reading from the keyboard because we can line up a whole bunch of calls to a program in a row.

It is possible, for example, using a *batch* or *bat* file to run your program hundreds of times in a row with different inputs

Using command line arguments

In the preceding example, what do you think would happen if you wrote:

```
java CommandLineArgsEcho
```

(without anything afterwards)?

Using command line arguments

Array out of bounds error!

Because of this, it is often useful to add checks, for example

```
if (args.length > 0 )
{
    //do something with args[0]
}
```

Java Packages

We have seen in Java that:

A group of commands makes a method
A group of methods makes a class

A **package** is a group of classes

Java Packages

There are many packages that come with Java standard library.

java.lang –general (includes String and System)
java.util – utilities (really! with that name!?) -- (includes Scanner)
java.io – input and output classes (for writing to files)
java.awt and java.swing – GUI classes

Java Packages

There are 2 ways you can use a class that is part of a package:

The first is to use the entire, “fully qualified” name

```
java.util.Scanner scan = new java.util.Scanner(System.in);
```

This works because the Scanner class is part of the java.util package.

If you do this, you do *not* have to write import java.util.Scanner !
(Yeah, dude it's totally awesome!)

Java Packages

When you import a package with an import statement, you tell Java to consider that package as well when looking for classes.

So if you write

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

Then when it sees the token “Scanner” it will scan the package java.util for it

Note: java.lang.* is automatically imported even if you don't write anything.

Analyzing System.out.println()

System is a class inside of the java.lang package

out is a *public static attribute* of the System class.

static ---> write the name of a class before the .
public --->means I can use it outside
attribute---> in general, you can only write an attribute or a behavior after the dot. Since we write a . AFTER “out” as well, then it must be an attribute

“out” is actually an object itself. This object, we are calling the println() method on.

Example: ArrayList

Are you sick of having to resize arrays by copying values one at a time?

Tired of manually searching for values in an array?

Well....then use ArrayList   !

Example: ArrayList

You can make an ArrayList out of any reference type. You do this by writing the following:

```
ArrayList<type> varName= new ArrayList<type>();
```

For example:

```
ArrayList<String> foo = new ArrayList<String>();
```

or

```
ArrayList<ComplexNumber>bar  = new
ArrayList<ComplexNumber>();
```

or

```
ArrayList<ArrayList<String> > name = new
ArrayList<ArrayList<string> >();
```

Example: ArrayList

If you want to make an ArrayList out of a primitive type, you have to use a *wrapper* class.

A wrapper class is a class that stores just 1 field--- a primitive type. *The entire purpose of it is to use any methods that require a reference type.*

Integer
Double
Character

are a few of the wrapper classes. There are wrapper class for every primitive type.

Using the Integer class

To use a wrapper class, you mainly need two things:

1) Converting the primitive type to the reference type
2) Converting the reference type to the primitive type

Both of these are very easy in Java.

```
Integer i = new Integer(1+3); //creates a reference from the int 4

int normal = i.intValue();

int pointlessExpression = new Integer(1).intValue();

int evenMorePointless = new Integer(new
                                        Integer(1).intValue()).intValue();
```

Using other wrapper classes is similar.

Example: ArrayList

```
ArrayList<Integer> foo = new ArrayList<Integer>();

foo.add(new Integer(3));

System.out.println(“The array size now is” + foo.size());

foo.add(new Integer(4));

System.out.println(“The array size now is” + foo.size());

System.out.println(“3 is located at position” + foo.indexOf(new
Integer(3)));  ----> doesn't work since it's a reference type!
```

Example: Finding the maximum index of an array (regular)

```
public static double maxValue(double[] array)
{
        double maxSoFar = array[0];
        for (int i=0; i < array.length; i++)
        {
                if (array[i] > maxSoFar)
                {
                        maxSoFar = array[i];
                }
        }

        return maxSoFar;
}
```

Example: Finding the maximum index of an array (regular)

```
public static double maxValue(double[] array)
{
        if (array == null || array.length == 0) return 0;

        double maxSoFar = array[0];
        for (int i=0; i < array.length; i++)
        {
                if (array[i] > maxSoFar)
                {
                        maxSoFar = array[i];
                }
        }

        return maxSoFar;
}
```

Example: Finding the maximum index of an ArrayList

```
public static double maxValue(ArrayList<Double> array)
{
        double maxSoFar = array.get(0).doubleValue();
        for (int i=0; i < array.size(); i++)
        {
                if (array.get(i).doubleValue()> maxSoFar)
                {
                        maxSoFar = array.get(i).doubleValue();
                }
        }

        return maxSoFar;
}
```

Example: Inserting an element into an array (at end)

```
public static int[] insertValue(int[] array, int newElement)
{
        int[] newArray = new int[array.length + 1];

        for (int i=0; i < array.length; i++)
        {
                newArray[i] = array[i];
        }

        newArray[array.length] = newElement;
        return newArray;
}
```

Example: Inserting an element into an ArrayList (at end)

```
public static void insertValue(ArrayList<Integer> array, int
newElement)
{
        array.add(new Integer(newElement));
}
```

Learning about ArrayList

http://download.oracle.com/javase/6/docs/api/java/util/ArrayList.html

(In class we went over a few methods at the website such as size(), insert(), get(), add())

Printing an ArrayList<String>

Printing an ArrayList will be almost identical to a regular array.

```
public static void print(ArrayList<String> list)
{
    for (int i=0; i < list.size(); i++) {
            System.out.println(list.get(i));
   }
}
```

We have to be very careful with reference types here!

The key thing to remember is whenever you set a reference variable EQUAL to something, you are changing the address of the variable---this means the variable refers to an entirely different object altogether.

Anything else causes you to change the DATA of a particular object

We have to be very careful with reference types here!

If x is a reference variable:

x.anything(); //can change the data referred to by x

x.something = somethingelse; // changes the data referred to by x

x.somemethod() = something; //compiler error. somemethod returns a value and you can't set a value equal to something

x = something; //x now refers to a different object than before

ArrayList of String

```
ArrayList<String> people;

people.add(“John Stuart”);
```

What's the problem?

ArrayList of String

```
ArrayList<String> people;

people.add(“John Stuart”);
```

What's the problem? NullPointerException

The variable people is null . You need to set it equal to an Object in order to add something to it.

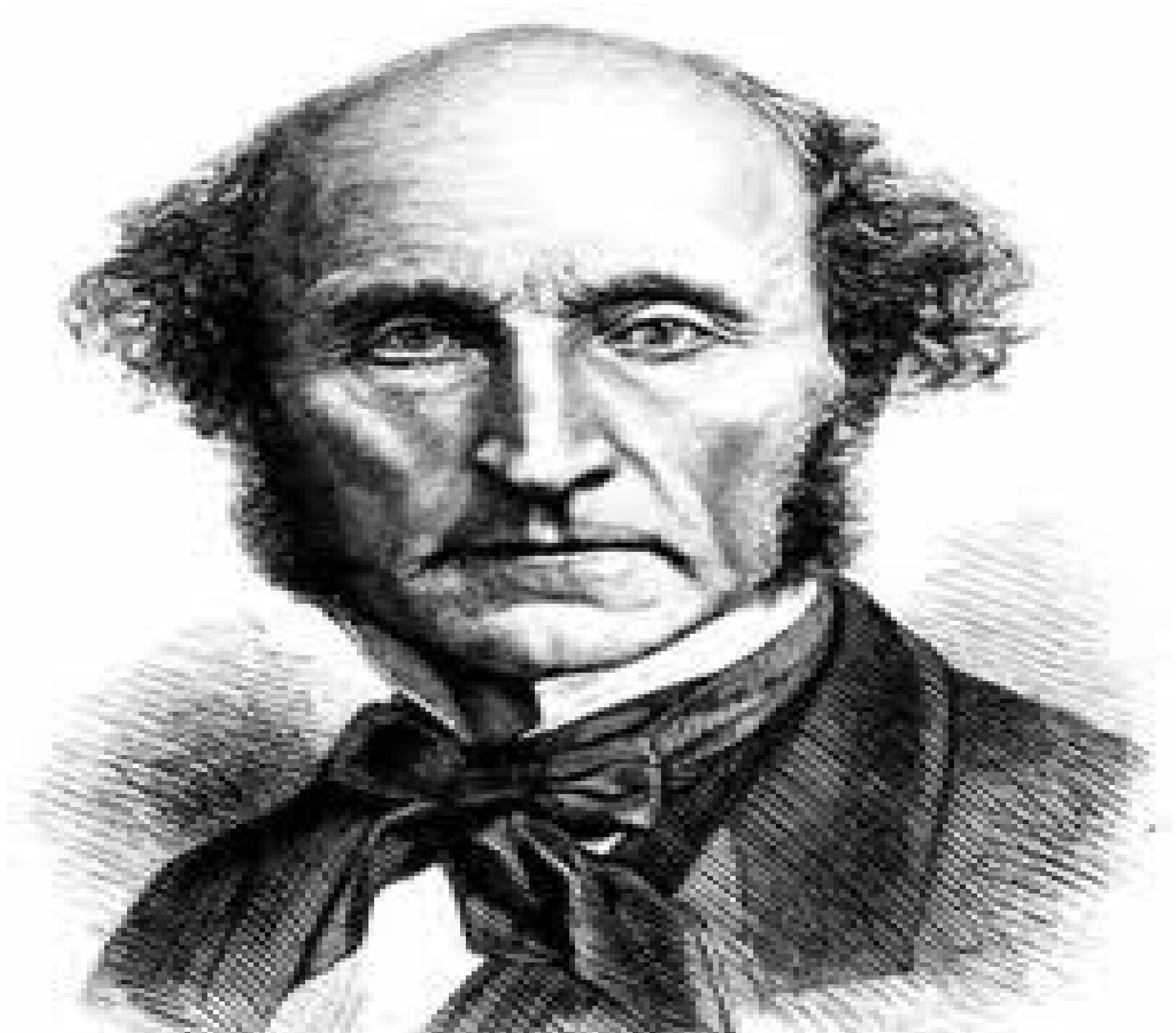
ArrayList of String

```
ArrayList<String> people;
people = new ArrayList<String>();
people.add(“John Stuart”);
```

ArrayList of String

```
ArrayList<String> people;
people = new ArrayList<String>();
people.add(“John Stuart”);
people.add(“Samantha Bee”);
people.add(“John Oliver”);
```

COMEDY CENTRAL



Changing an element in an ArrayList

```
ArrayList<String> people;
people = new ArrayList<String>();
people.add(“John Stuart”);
people.add(“Samantha Bee”);
people.add(“John Oliver”);

String john = people.get(0); //stores into john the address of a String

john = “Jon Stewart”;
```

Changing an element in an ArrayList

When we print the ArrayList though, we get the same results as before.

Why?

Changing an element in an ArrayList

When we print the ArrayList though, we get the same results as before.

Why?

since we set the variable john EQUAL to “Jon Stewart” it means the variable john refers to an entirely different String. Thus there is no link to the original list.

Changing an element in an ArrayList

What we'd like to be able to do is call

john.setValue(“Jon Stewart”);

but no such method exists since we know that Strings are immutable.

So we are stuck deleting the element and adding it back in.

Changing an element in an ArrayList

```
ArrayList<String> people;
people = new ArrayList<String>();
people.add(“John Stuart”);
people.add(“Samantha Bee”);
people.add(“John Oliver”);

String john = people.get(0); //stores into john the address of a String

people.remove(john);
people.add(0, new String(“Jon Stewart”));
```

Making a *mutable* String wrapper class

```
public class MutableString {
        private String data;

        public MutableString(String initialValue) {
                data = initialValue;
        }
        public String getValue() {
                return data;
        }
        public void setValue(String newValue) {
                data = newValue;
        }
}
```

Changing an element in an ArrayList

```
ArrayList<MutableString> people;
people = new ArrayList<MutableString>();
people.add(new MutableString(“John Stuart”));
people.add(new MutableString(“Samantha Bee”));
people.add(new MutableString(“John Oliver”));

MutableString john = people.get(0); //stores into john the address of a
MutableString, which in turn stores an address of a String

//john = new MutableString(“Jon Stuart”); //doesn't work!
john.setValue(“Jon Stuart”); //Success!!
```

Storing a 2d array in a 1d array

A 2d array can be stored inside a 1d array. We basically "flatten" each row of the array:

{ {1,2,3}, {4,5,6}, {7,8,9} }

would become

{1,2,3,4,5,6,7,8,9}

Storing a 2d array in a 1d array

Sometimes, this is more convenient than storing it in a 2d array.

Converting a 2d array to 1d

In general, the value at location [i][j] of my 2d array will go to location [i * numberColumns + j] of the 1d array:

```
public static int[] flatten(int[][] original) {
        int[] flatArray = new int[original.length * original[0].length];
        for (int i=0; i < original.length;i++) {
                for (int j=0; j < original[0].length; j++) {
                        flatArray[i * original[0].length +j] =
                                original[i][j];
                }
        }
        return flatArray;
}
```

Converting a 1d array back to 2d

In general, the value at location [i][j] of my 2d array will go to location [i * numberColumns + j] of the 1d array:

```
public static int[][] flatten(int[] flatArray, int width) {
        int[][] matrix = new int[flatArray.length / width][width];
        for (int i=0; i < flatArray.length / width;i++) {
                for (int j=0; j < width; j++) {
                        matrix[i][j] = original[i * width + j];
                }
        }
        return flatArray;
}
```

Assignment 3

In Assignment 3, when you load an image, you load a 2d array.
However, the 2d array is actually storing 4 “flattened” 2d arrays.

The first array inside the 2d array is the alpha array (transparency)
The 2nd array inside the 2d array is the red array
The 3rd array inside the 2d array is the green array
The 4th array inside the 2d array is the blue array

Assignment 3

In part one, you write a bunch of a methods that operate on 2d arrays. In order to use these methods, you will have to do the following:

1)Take the original 2d array and unflatten each of the 4 arrays
2)Now you will have FOUR 2d arrays
3)For each of these arrays, you should apply the transformation specified
4)Now convert the 2d arrays BACK to flattened 1d arrays.
5)Take each of these FOUR now 1d arrays, and put them into one array.