```
public class June16 {
public static void main(String args[]){
    arrayList();
    recursion();
}


public static void recursion() {
     recursion(); //class goes on forever:(
}
}
```

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> funnyPeople;

funnyPeople.add("John Stuart");

What's the problem?

ArrayList of String

```
ArrayList<String> funnyPeople;

funnyPeople.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> funnyPeople;
funnyPeople = new ArrayList<String>();
funnyPeople.add("John Stuart");
```

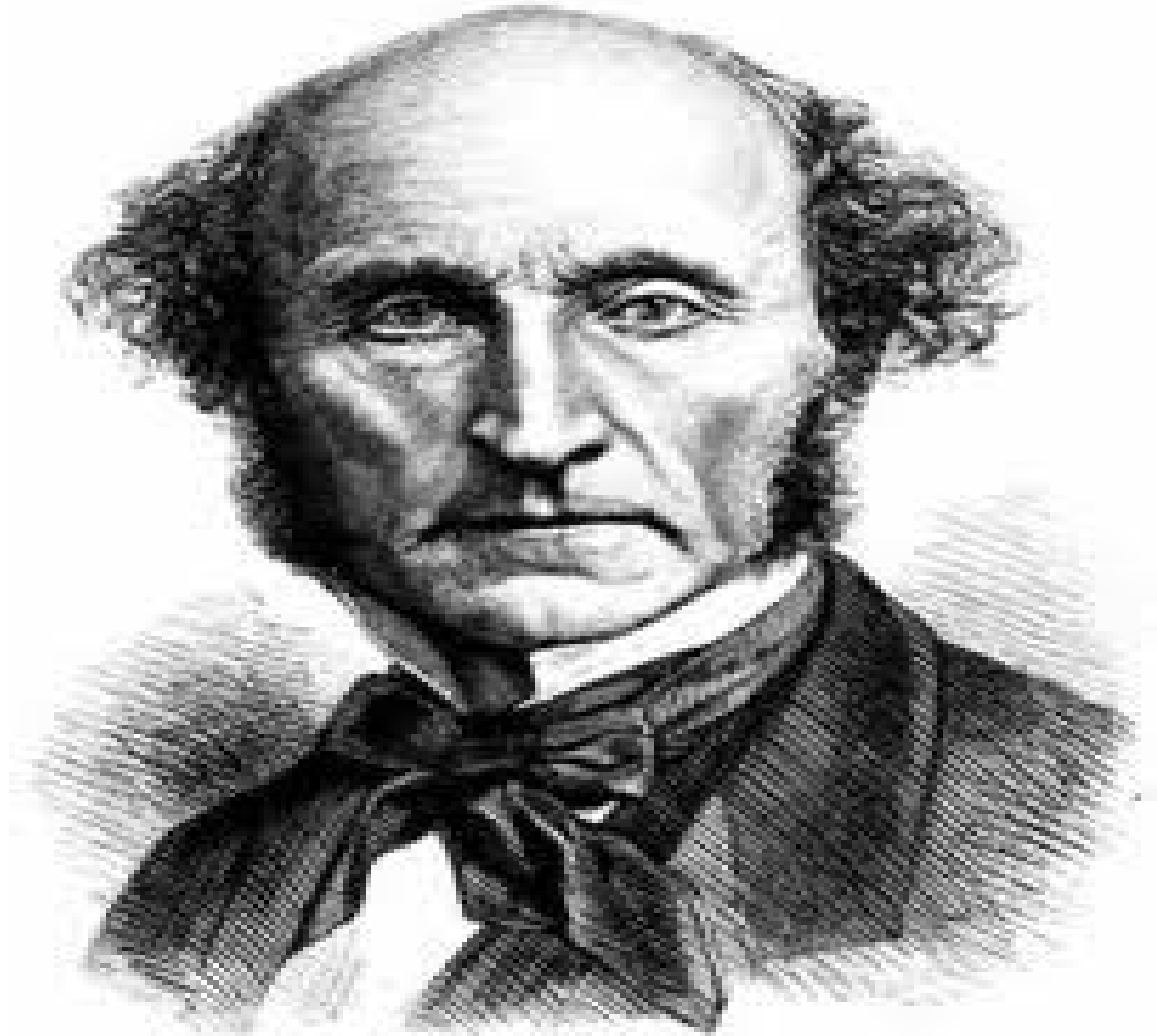
ArrayList of String

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

Now there is a different, non-Java related problem.

This is whom we wanted to add

This is whom we actually added



Changing an element in an ArrayList

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

String john = funnyPeople.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> funnyPeople;
funnyPeople = new ArrayList<String>();
funnyPeople.add("John Stuart");
funnyPeople.add("Samantha Bee");
funnyPeople.add("John Oliver");

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

funnyPeople.remove(john);
funnyPeople.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> funnyPeople;
funnyPeople = new ArrayList<MutableString>();
funnyPeople.add(new MutableString("John Stuart"));
funnyPeople.add(new MutableString("Samantha Bee"));
funnyPeople.add(new MutableString("John Oliver"));

MutableString john = funnyPeople.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!!
```

Recursion

THE

IDIOT CARD

If you'd like to know how to keep an idiot busy for hours, turn this card over...

THE

IOT CA

'd like to know how to

idiot busy for hours, t

this card over...

“To understand recursion,
you must first understand
recursion.”

Silly Definition

recursion (n):

See recursion

Recursion

Recursion is when a method calls itself.

Infinite Recursion

```
public static void forever() {
    forever();
}
```

Recursion

Every time you call a method, remember that the computer keeps track of **where** you were before the method call.

This includes if a method calls itself.

Recursion

This list of methods is known as a "Stack"

We could add an extra column to our variable table to include stackdepth if we wanted to include:

variable, value, method, stack level and type

Recursion

A good criteria for using recursion is when we can write a function with input X in terms of the same function with input Y where $Y < X$

For example: factorial(X) = X * factorial(X-1) if X >= 1

Recursion

```
public static int factorial(int n){
    if (n == 0) { //could make < 0
        return 1;
    }

    return n * factorial(n-1);
}
```

factorial(3)

When you call the method factorial with input 3, the computer will go to the line

```
return n * factorial(n-1);
```

As it performs the computation it gets:

```
return 3 * factorial(2)
```

factorial(3)

The computer then remembers "I am going to call the method factorial with input 2. Whenever I figure out the result of that method call, I will use the result as part of a computation 3 * factorial(2), which will then be returned"

factorial(2) and factorial(1) then have similar effects.

factorial(1) will return

1 * factorial(0);

factorial(0) returns 0 since n == 0

Base Case

If we write our problem with larger input in terms of a problem with smaller input, it is always critical to write a *base* case or a stopping point.

In this case, the stopping point is when n == 0

If we don't do this, we have infinite recursion

Fixing the Silly Definition

recursion (n):

See recursion

This has no base case. Thus it's an infinite loop.

Fixing the Silly Definition

recursion (n):

If you still don't get it, see recursion

Now the base case happens when the user gets it.

Add with recursion (positive #s)

```
public static int add(int x, int y) {
    if (y == 0) {
        return x;
    }

    return x + 1 + add(y-1);
}
```

Fibonacci #s, 2 base cases

The fibonacci numbers are defined as follows:

$f(1) = 1;$
$f(2) = 1;$
$f(n) = f(n-1) + f(n-2);$

We get the sequence
1,1,2,3,5,8,13,21,34,55,......

Fibonacci #s

How can we write this using recursion?

Bonus question: How can we write this using a for loop

Fibonacci #s

```
public static int fib(int n) {
    if (n==1) return 1;
    if (n==2) return 1;

    return fib(n-1) + fib(n-2);
}
```

Searching for a file on a computer

We could use recursion to search for a file on a computer. Suppose you were given as input a String filename to search for and a Folder folder. Folders can contain directories and other folders.

For starters, we will have our method return a boolean whether it finds it or not

```
public static boolean search(String
filename, Folder f) {
        for every file in f {
                if (filename = file.Name) return
true
        }
        for every directory d in f {
                result= search(filename, d);
                if (result = true) return true
        }
      return false;
}
```

Aside: Representing a folder or file in Java

There is a class called File inside the package java.io

http://download.oracle.com/javase/1.4.2/docs/api/java/io/File.html

This has methods such as the following:

isDirectory() : tests if it is a directory or not
isFile() : tests if it is a file or not
listFiles() : lists all the files (and directories) in the directory. Returns a File[] . It returns null if not found
getName()

Based on the methods given to us, we actually want to rewrite our algorithm a bit.

```
public static boolean searchFile(File f,
String target) {
    if (target == f.getName()) return true;

    File[] allFiles = f.listFiles();
    for every File f2 in allFiles
        if ( searchFile(f2, target) == true)
            return true
    end for
   return false
}
```

The base case here is when a file has no “children” --- in other words it is not a directory.

Depending on how we code it, we may not actually have to explicitly test for this

```
public static boolean searchFile(File f, String
target) {
    if (f.getName == target) {
        return true;
    }

    File[] allFiles = f.listFiles();
    for (int i=0; i<allFiles.length; i++) {
        if( searchFile(allFiles[i], target))
            return true;
    }
    return false;
}
```

```
public static boolean searchFile(File f, String
target) {
    if (f.getName == target) {
        return true;
    }
    File[] allFiles = f.listFiles();
    if (allFiles == null) return false;
    for (int i=0; i<allFiles.length; i++) {
        if( searchFile(allFiles[i], target))
            return true;
    }
    return false;
}
```

Now, suppose we want to return an ArrayList of File representing all files found.

Most of this will be similar, but we'll have to either return an ArrayList or pass an ArrayList as an argument

Option 1: Return an ArrayList

We can have the method return an ArrayList<File>

Then, everytime we call a method, we'll add it's returned results to the ArrayList

```
public static ArrayList<File> searchFile(File f, String target)
{
    ArrayList<File> ret = new ArrayList<File>();
    if (f.getName == target) {
        ret.add(f);
    }

    File[] allFiles = f.listFiles();
    if (allFiles == null) return ret;
    for (int i=0; i<allFiles.length; i++) {
        ArrayList<File> temp= searchFile(allFiles[i], target));
          ret.addAll(temp);
    }
      return ret;
}
```

Option 2: Pass an ArrayList as an argument

We will pass an ArrayList to all the various recursive calls. At the end, the ArrayList will be filled with all the entries we need.

```
public static ArrayList<File> searchFile(File f, String target) {
    ArrayList<File> files = new ArrayList<File>();
     searchFileHelper(f, target, files);
     return files;
}
public static void searchFileHelper(File f, String target,
ArrayList<File> list) {
    if (f.getName == target) {
        list.add(f);;
    }

    File[] allFiles = f.listFiles();
    if (allFiles == null) return;
    for (int i=0; i<allFiles.length; i++) {
        searchFileHelper(allFiles[i], target, list);
    }
}
```

Sorting an ArrayList<Integer> recursively:

One way to sort an ArrayList<Integer> is recursively:

We can think of sorting an array of n numbers as
1)Putting the minimum value into the first spot
2)Sorting the rest of the array (of n-1 numbers)

```
public static void sort(ArrayList<Integer> nums) {
    sortHelper(0, nums)
}
public static void sortHelper(int start, ArrayList<Integer>
nums) {
        if (start == nums.size()) return;
        int minIndex = findMinimumInteger(start, nums);
        swapElements(start, minIndex);
        sortHelper(start+1, nums);
}
```

Exercise (take a few minutes):

Using recursion, write the method

public static calculateSum(int[] numbers)

```
public static int calculateSum(int[] numbers) {
	return calculateSumHelper(numbers, 0);
}

public static int calculateSumHelper(int[]
numbers, int start) {
	if (start == numbers.length){
		return 0;
	}
	return numbers[start] +
calculateSumHelper(numbers, start+1);

}
```