Quiz

Please keep quizzes face down until class starts.

Please use conventions from class.
(underscore, camelCase)

You will have 15 minutes for the quiz.

When done, turn your quiz face down.
(if everyone is done early, we’ll stop early)
Quiz Review
What if we want to…

Create a shopping cart for coffee beans?
Track the gas mileage on our car?

Shopping carts should be able to hold more than one product.
Good way to track mileage is record how much you filled up each time you do.

Implication: we want to be able to store many objects in one place—an amount we cannot know when we are defining a class.
Arrays & Strings

COMP 110: Lecture 10
Midterm Study Guide

This is the study guide for last spring 2016 COMP110, things are (not surprisingly) pretty much the same…

http://comp110.com/topics/getting-started/exam-1-study-guide

Note: Where it says “PS2” that means “Problem Set 2”

In addition, you should be prepared for:
• a ‘%’ question (easier than part 7, so hopefully a freebie)
• converting integers to/from binary (between 0 and 63)
• a more complicated version of today’s quiz
• arrays (today’s lecture)
Loops Review

// guess until we gotIt
boolean gotIt = false;
while (!gotIt) {
    gotIt = guess();
}

// print "Hello!" 3 times
for (int i=0 ; i<3 ; ++i) {
    System.out.println("Hello!");
}
Loops at Moogfest
Loops at Moogfest
Arrays contain a fixed number of values of a single type.

Array length is 10

Index (starts at 0)

Value at index 5
Array Allocation and Length

The *length* of an array is set when an array is *allocated*.

Array allocation looks much like object allocation:

```
new int[10]
```

creates an array of 10 integers.
Arrays in RAM

When you allocate an array, you are allocating a single block of memory with enough space to hold all elements of the array.

*Where* the entire array resides in memory does not matter.
Array Type

Any type in Java can be the element type of an array. Adding `[]` after the base type makes an array type.

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

Declare a variable of type “array of ints” called “values”

Assign it to a newly allocated array of 10 ints.

Note: array type does not include length, whereas allocation does include length.
Array Type vs. Instance

```java
int[] values = new int[10];
```

An array’s **type** is based upon its element (aka component) type. It does not matter how many elements it has.

An **instance** of an array is a block of memory. Therefore it must have a size (and location) in memory.
Examples of Array Types

```java
int[] intArray;
long[] anArrayOfLongs;
float[] arrayOfFloats;
double[] someDoubles;
short[] aFewShorts;
char[] charArray;
byte[] byteArray;
boolean[] booleans;
String[] strings;
Address[] addresses;
```
Think of allocating an array as a shelf in a library. (The shelf’s size is fixed when it is created)

You get to the shelf by the variable (e.g., myBookshelf)

Then you find the book by counting from the left (starting at index 0)
Our Java Library

In our bookshelf analogy, when you allocate a shelf (array), you must specify what goes on that shelf.

For example, if you want to put toys in the library, you’d need to allocate a shelf (array) of toys.

Java doesn’t allow you to put toys on a bookshelf, or books on a toyshelf.
Arrays are accessed by an integer index.

```java
class IntArrayDemo {
    public static void main(String[] args) {
        int[] intArray = new int[10];
        intArray[0] = 100;
        intArray[1] = 200;
        intArray[2] = 300;
        intArray[3] = 400;
        intArray[4] = 500;
        intArray[5] = 600;
        intArray[6] = 700;
        intArray[7] = 800;
        intArray[8] = 900;
        intArray[9] = 1000;
        System.out.println("Element at index 0: " + intArray[0]);
        System.out.println("Element at index 1: " + intArray[1]);
        System.out.println("Element at index 2: " + intArray[2]);
        System.out.println("Element at index 3: " + intArray[3]);
        System.out.println("Element at index 4: " + intArray[4]);
        System.out.println("Element at index 5: " + intArray[5]);
        System.out.println("Element at index 6: " + intArray[6]);
        System.out.println("Element at index 7: " + intArray[7]);
        System.out.println("Element at index 8: " + intArray[8]);
        System.out.println("Element at index 9: " + intArray[9]);
    }
}
```
Iteration and Arrays

What makes arrays so powerful is that the index can be any integer expression.

```java
class IntArrayDemo {
    public static void main(String[] args) {
        int[] intArray = new int[10];
        for (int i=0 ; i<10 ; ++i) {
            intArray[i] = (i+1) * 100;
        }
        for (int i=0 ; i<10 ; ++i) {
            System.out.println("Element at index "+i+": "+intArray[i]);
        }
    }
}
```

Here we simplified the previous code using two for loops.
Array .length

Every array has a read-only field call “.length” that is the number of elements in the array.

class IntArrayDemo {
    public static void main(String[] args) {
        int[] intArray = new int[10];
        for (int i=0 ; i<intArray.length ; ++i) {
            intArray[i] = (i+1) * 100;
        }
        for (int i=0 ; i<intArray.length ; ++i) {
            System.out.println("Element at index "+i+: "+intArray[i]);
        }
    }
}

Changed (again) previous example to use .length.
Array Index is **Any** Integer Expression

(so powerful, its worth repeating)

class IntArrayDemo {

    
    ...  

    void printEveryOtherElement(int[] array) {
        for (int i=0 ; i*2<array.length ; ++i) {
            System.out.println(array[i*2]);
        }
    }

    
    void shuffle(int[] array) {
        for (int i=0 ; i<array.length ; ++i) {
            int j = randomInRange(i, array.length);
            int tmp = array[i];
            array[i] = array[j];
            array[j] = tmp;
        }
    }

    
    ...

}
Donald Ervin Knuth is an American computer scientist, mathematician, and professor emeritus at Stanford University. He is the author of the multi-volume work *The Art of Computer Programming*. Knuth has been called the “father of the analysis of algorithms”. He contributed to the development of the rigorous analysis of the computational complexity of algorithms and systematized formal mathematical techniques for it. …
Array indexes start at 0
Not true in all languages, some start at 1.
At this point, you might be saying, “haven’t we seen that before somewhere?”

```java
class ArrayExample {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Our main() method takes an argument of type: “array of Strings”
main arguments

When you run your java program, everything after the class name is a String argument to main.

```java
class Echo {
    public static void main(String[] args) {
        for (int i=0 ; i<args.length ; ++i) {
            System.out.println("args["+i+"] = "+args[i]);
        }
    }
}
```

These are called “command line arguments”

```
~/comp110/arrays $ java Echo
Hello World, here are some strings
args[0] = Hello
args[1] = World,
args[2] = here
args[3] = are
args[4] = some
args[5] = strings
```

If you try this, use alpha-numeric command line arguments only. Some characters like '!', '<', '>', '|' have special meaning on the command line.
Initializing Arrays

When an array is allocated, all elements are initialized to the default value for the element.

(the same default value as class fields)
Array Initializer

Arrays can be created using an array initializer.

```java
class IntArrayDemo {
    public static void main(String[] args) {
        int[] intArray = {
            100, 200, 300,
            400, 500, 600,
            700, 800, 900,
            1000,
        };
        ...
    }
}
```

Each element is comma separated. final comma is optional

A minor disadvantage is that you cannot specify the array length explicitly (e.g., in an attempt to only initialize the first elements of a larger array)
Array Initializer

Array initializers do exactly the same thing as allocating then assigning. With much less typing.

```java
class IntArrayDemo {
    public static void main(String[] args) {
        int[] intArray = {
            100, 200, 300,
            400, 500, 600,
            700, 800, 900,
            1000,
        };
    }
    ...
}

// Same as

class IntArrayDemo {
    public static void main(String[] args) {
        int[] intArray = new int[10];
        intArray[0] = 100;
        intArray[1] = 200;
        intArray[2] = 300;
        intArray[3] = 400;
        intArray[4] = 500;
        intArray[5] = 600;
        intArray[6] = 700;
        intArray[7] = 800;
        intArray[8] = 900;
        intArray[9] = 1000;
        ...
    }
}
```
Strings are arrays of chars

Here is a simplified snippet of the actual source for the String class.

String is a wrapper for char[], with extensive functionality added.
Strings are immutable

The designers of Java decided to make `String immutable`. Since `arrays are mutable`, the `String` class hides the `char[]` by marking it as private. We will get to more on access control later.

String equivalent of `array[i]` is `string.charAt(i)`

`array[i]` is a location in memory, and is thus assignable, `string.charAt(i)` is not.
class StringExample {
    public static void main(String[] args) {
        String hello = "Hello, World!";
        for (int i=0; i<hello.length(); ++i) {
            char ch = hello.charAt(i);
            for (int j=0; j<i; ++j) {
                System.out.print(' ');
            }
            System.out.println(ch);
        }
        System.out.println();
    }
}

Try it out!
Can you make it put a space between every letter?
Double every letter?
Multidimensional Arrays

Any type can be used as the component type of an array. Thus we can make an array of arrays:

```java
int[][] values = new int[10][10];
```

And array of arrays of arrays:

```java
int[][][] values = new int[10][10][10];
```

And so on…

Don’t get too carried away… a 10x10x10x10x10x10x10x10x10x10 array of ints requires more than 40 GB!
Checkers Revisited

Now that we have arrays, we have a way to track our checkers board. We can create a Checkers class with an 8x8 array of chars.

class Checkers {
    Console _console = System.console();
    char[][] _board = new char[8][8];

    Checkers() {
        for (int i=0 ; i<8 ; ++i) {
            for (int j=0 ; j<8 ; ++j) {
                if ((i+j)%2 == 1) {
                    if (i < 3) {
                        _board[i][j] = 'r';
                    } else if (i > 4) {
                        _board[i][j] = 'b';
                    }
                }
            }
        }
    }
}

With System.out.println() and console.readLine(), we could have a game.
java.util.Arrays

To get a wealth of static methods for manipulating arrays. Before your class definition, add:

```java
import java.util.Arrays;
```

These slides are just an overview of java.util.Arrays, see:

http://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html

The following slides are in a java file similar to this:

```java
import java.util.Arrays;

class ArrayExamples {
    public static void main(String[] args) {
        ...
    }
}
```
Copying Arrays

```java
int[] anArray = ...;

// manual copy
int[] copy0 = new int[anArray.length];
for (int i=0 ; i<anArray.length ; ++i) {
    copy0[i] = anArray[i];
}

// copy array to new array
int[] copy1 = anArray.clone();
// copy array to a new array of different size
int[] copy2 = Arrays.copyOf(anArray, anArray.length);
// copy portion of array from index 5 to 9 to new array
int[] copy3 = Arrays.copyOfRange(anArray, 5, 9);
// copy elements from one array to another
int[] copy4 = new int[anArray.length];
System.arraycopy(anArray, 0, copy4, 0, anArray.length);
```
Filling Arrays

```java
int[] anArray = ...;

// fill entire array with -3
Arrays.fill(anArray, -3);

// fill array from index 5 to 9 with -1;
Arrays.fill(anArray, 5, 9, -1);
```
int[] anArray = ...

// sort the entire array in ascending order
Arrays.sort(anArray);

// sort part of the array in ascending order
Arrays.sort(anArray, 5, 10);
Printing Arrays

System.out.println(Arrays.toString(anArray));

Will output:
[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]
Comparing Arrays

```java
int[] array1 = ...;
int[] array2 = ...;

if (Arrays.equals(array1, array2)) {
    // size and all elements of arrays are the same
}
```

(array1 == array2) tests if the arrays are the same arrays that is, the same block of memory allocated via a `new int[]`. 

(continue)
Chess

~ /comp110/arrays $ java Chess

Unicode has chess characters!

(and Java uses Unicode!)

(but unfortunately Microsoft is a little late to the game... this takes some effort to get running in Windows.)
Images

Images are 2D arrays of colors.

The human eye perceives in red, green and blue (RGB).

Pixels are stored as RGB values. Typically as 3 bytes. (1 byte per R, G, B).

Some images also store an alpha (A), which is how transparent it is.

Normalized responsivity spectra of human cone cells, S, M, and L types
Hexadecimal

Remember binary?
(Another way of saying “base 2”)  
Hexadecimal is base 16.

Wait… what?!? How do you have digits (hexit?) after 9?

0 1 2 3 4 5 6 7 8 9 a b c d e f
a = 10, b = 11, c = 12, d = 13, e = 14, f = 15

In Java we can write hex in code using the prefix “0x”,
thus $0x10 == 16$, $0x2a == 42$, $0x40 == 64$

(capitalization does not matter $0x2a == 0x2A$)
Hexadecimal

When you work with colors, you’ll often see things like:

#7BAFD4

which means (base 16):
red = 0x7B, green = 0xAF, blue = 0xD4
which means in base 10:
red = 123, green = 175, blue = 212

Each component is in the range 0 to 255 (0 to 0xff), and thus we get a some red, a little more green, and a lot of blue. In other words: **Carolina blue.**
Try at Home: Array Reverse

import java.util.Arrays;

class Reverse {
    static void reverse(int[] array) {
        // TODO: implement this
    }

    public static void main(String[] args) {
        int[] array = new int[10];
        for (int i=0 ; i<array.length ; ++i) {
            array[i] = i;
        }
        reverse(array);
        System.out.println(Arrays.toString(array));
    }
}

Without searching for the solution (because that would be too easy), implement reverse(int[] array) so that it reverses the elements of any integer array. The above test code should print out:

[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
Array Element Swapping

A common task in many algorithms is to swap two elements of an array.

Given the setup:

```java
String[] array = (the array, already created);
int i = (an index to swap);
int j = (an index to swap);
```

Why does this not work?

```java
array[i] = array[j];
array[j] = array[i];
```
### Array Element Swapping

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>“go”</td>
<td>“duke”</td>
<td>“!”</td>
<td>“beat”</td>
<td>“unc”</td>
<td>“!”</td>
<td></td>
</tr>
</tbody>
</table>

While correct for our friends in Durham, something’s wrong with this array in Chapel Hill…

*Let’s swap elements at i=1, j=4.*

```plaintext
array[1] = array[4];
array[4] = array[1];
```

Wrong!
Array Element Swapping

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>“go”</td>
<td>“duke”</td>
<td>“!”</td>
<td>“beat”</td>
<td>“unc”</td>
<td>“!”</td>
</tr>
</tbody>
</table>

Let’s try again, we lost “duke” last time. We’ll keep them in a variable so we keep them.

```java
String tmp = array[1];
array[1] = array[4];
```

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>“go”</td>
<td>“unc”</td>
<td>“!”</td>
<td>“beat”</td>
<td>“duke”</td>
<td>“!”</td>
</tr>
</tbody>
</table>
Reading

http://comp110.com/topics/getting-started/exam-1-study-guide

Chapter 12

Chapter 13

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html