

# CSSE 220

Arrays, ArrayLists,  
Wrapper Classes, Auto-boxing,  
Enhanced *for* loop

Check out *ArrayListPractice* from SVN

# Help with Peers

- Having a peer help you with some strange bug or specific problem – Great Idea!
- Discussing your approach to a problem with a peer – still OK
- Letting a peer see your code/Emailing code to a peer – NEVER OK
- Every person has a unique code style, it's easy to tell when two sets of code are too similar

# Arrays- What, When, Why, & How?

- What
  - A special **type** used to hold a set number of items of a specified type
- When
  - Use when you need to store multiple items of the same type
  - Number of items is known and will not change

# Arrays- What, When, Why, & How?

- Why
  - Avoids things like int1, int2, int3, int4
  - Avoids repetitive code and frequent updates
- How
  - `Type[] arr = new Type[num];` ← Creates a new array of type Type stored in variable arr
  - An array of 5 Strings (stored in the variable fiveStrings) would look like this:
    - `String[] fiveStrings = new String[5];`

# Array Examples Handout

- Form groups of 2
- Look at the Array Examples Handout
- Study how arrays are used and answer the questions in the quiz
  - FIRST PAGE OF QUIZ ONLY

Go to <http://codingbat.com/java/Array-2>

- Work in your groups to solve `fizArray3`, `bigDiff`, `shiftLeft`
- When you finish all 3, call me over to take a look
- If you finish early, try `zeroFront`

# Array Types

- ▶ Group a collection of objects under a single name
- ▶ Elements are referred to by their **position**, or *index*, in the collection (0, 1, 2, ...)
- ▶ Syntax for declaring: *ElementType[] name*
- ▶ Declaration examples:
  - A local variable: `double[] averages;`
  - Parameters: `public int max(int[] values) {...}`
  - A field: `private Investment[] mutualFunds;`

# Allocating Arrays

- ▶ Syntax for allocating:

`new ElementType[Length]`

- ▶ Creates space to hold values

- ▶ Sets values to defaults

- `0` for number types
- `false` for boolean type
- `null` for object types

- ▶ Examples:

- `double[] polls = new double[50];`
- `int[] elecVotes = new int[50];`
- `Dog[] dogs = new Dog[50];`

Don't forget this step!

This does NOT construct any **Dogs**. It just allocates space for referring to **Dogs** (all the **Dogs** start out as *null* )



# Reading and Writing Array Elements

## ▶ Reading:

- `double exp = polls[42] * elecVotes[42];`

Sets the value in  
slot 37.

Reads the element with  
index 42.

## ▶ Writing:

- `elecVotes[37] = 11;`

▶ Index numbers run from 0 to array length – 1

▶ Getting array length: `elecVotes.length`

No parentheses, array length  
is (like) a field

# Arrays: Comparison Shopping

Arrays...	Java	Python lists
<i>have fixed length</i>	<i>yes</i>	<i>no</i>
<i>are initialized to default values</i>	<i>yes</i>	<i>n/a</i>
<i>track their own length</i>	<i>yes</i>	<i>yes</i>
<i>trying to access "out of bounds" stops program before worse things happen</i>	<i>yes</i>	<i>yes</i>

# ArrayList- What, When, Why, & How?

- What
  - A class in a Java library used to hold a collection of items of a specified type
  - Allows variable number of items
  - Fast random access
- When
  - Use when you need to store multiple items of the same type
  - Number of items is not known/will change

# ArrayList- What, When, Why, & How?

- Why
  - Fast random access
  - Allows length changes, cannot do this with an array
- How
  - `ArrayList<Type> arl = new ArrayList<Type>();`
    - Creates a new ArrayList of type Type stored in variable arl

# ArrayList Examples Handout

- Look at the `ArrayList` section of the examples handout
- Study how `ArrayLists` are used and answer the questions in the quiz
- Then solve the 3 problems in `ArrayListPractice` (you downloaded it from SVN)
- When you finish, call me over to take a look

# What if we don't know how many elements there will be?

## ▶ **ArrayLists** to the rescue

## ▶ Example:

Element type

```
◦ ArrayList<State> states = new ArrayList<State>();
```

Variable type

Constructs new, empty list

```
◦ states.add(new State("Indiana", 11, .484, .497));
```

Adds new element to end of list

## ▶ **ArrayList** is a *generic class*

- Type in <brackets> is called a *type parameter*

# ArrayList Gotchas

- Type parameter can't be a primitive type
  - Not: `ArrayList<int> runs;`
  - But: `ArrayList<Integer> runs;`
- Use ***get*** method to read elements
  - Not: `runs[12]`
  - But: `runs.get(12)`
- Use **`size()`** not **`length`**
  - Not: `runs.length`
  - But: `runs.size()`

# Lots of Ways to Add to List

- ▶ Add to end:
  - `victories.add(new WorldSeries(2011));`
- ▶ Overwrite existing element:
  - `victories.set(0, new WorldSeries(1907));`
- ▶ Insert in the middle:
  - `victories.add(1, new WorldSeries(1908));`
  - Pushes elements at indexes 1 and higher up one
- ▶ Can also remove:
  - `victories.remove(victories.size() - 1)`



# So, what's the deal with primitive types?

## ► Problem:

- ArrayList's only hold objects
- Primitive types aren't objects

## ► Solution:

- *Wrapper classes*—instances are used to “turn” primitive types into objects
- Primitive value is stored in a field inside the object

Primitive	Wrapper
<i>byte</i>	<i>Byte</i>
<i>boolean</i>	<i>Boolean</i>
<i>char</i>	<i>Character</i>
<i>double</i>	<i>Double</i>
<i>float</i>	<i>Float</i>
<i>int</i>	<i>Integer</i>
<i>long</i>	<i>Long</i>
<i>short</i>	<i>Short</i>

# Auto-boxing Makes Wrappers Easy

- ▶ Auto-boxing: automatically enclosing a primitive type in a wrapper object when needed
- ▶ Example:
  - You write: `Integer m = 6;`
  - Java does: `Integer m = new Integer(6);`
  - You write: `Integer answer = m * 7;`
  - Java does: `int temp = m.intValue() * 7;`  
`Integer answer = new Integer(temp);`

# Auto-boxing Lets Us Use ArrayLists with Primitive Types

- ▶ Just have to remember to use wrapper class for list element type
- ▶ Example:
  - `ArrayList<Integer> runs =  
 new ArrayList<Integer>();  
 runs.add(9); // 9 is auto-boxed`
  - `int r = runs.get(0); // result is  
 unboxed`

# Enhanced For Loop and Arrays

## ▶ Old school

```
double scores[] = ...  
double sum = 0.0;  
for (int i=0; i < scores.length; i++) {  
    sum += scores[i];  
}
```

## ▶ New, whiz-bang, enhanced for loop

```
double scores[] = ...  
double sum = 0.0;  
for (double score : scores) {  
    sum += score;  
}
```

Say "in"

- No index variable (easy, but limited in 2 respects)
- Gives a name (score here) to each element

# Enhanced For and ArrayList's

```
▶ ArrayList<State> states = ...  
int total = 0;  
for (State state : states) {  
    total += state.getElectoralVotes();  
}
```

# Debugging—Key Concepts

- Breakpoint
- Single stepping
- Inspecting variables

# Debugging—Demo

- ▶ Debugging Java programs in Eclipse:
  - Launch using the debugger
  - Setting a breakpoint
  - Single stepping: *step over* and *step into*
  - Inspecting variables
- ▶ Complete **WhackABug** exercise