

CSSE 220

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

**Please sit in the first four rows!
(not the back row if possible 😊)**

Check out *ArrayListPractice* from SVN

SVN re-cap

- Make sure to refresh to see new folders
- Make sure you commit when complete
- Black * box should go away after committing
- If any ongoing issues, please email me!

Review Loops: while & for Loops

While loop syntax:

Similar to Python

```
while (condition) {  
    statements  
}
```

For loop syntax:

Different from Python

```
for (initialization ; condition ; update) {  
    statements  
}
```

In both cases, curly braces optional if only one statement in body; but be careful!

Comparing for vs. while

```
int i =0;           ← extra line
while ( i<10) {
    System.out.println( i );
    i++;           ← extra line
}
```

```
for (int i=0 ; i<10; i++) {
    System.out.println( i );
}
```

Important Reminder: Comparisons

- Fast rules for now:
- Use `.equals()` for comparing Strings
String alpha = "aaa";
if (alpha.equals("bbb") {
 System.out.println("Yes!")
}
}
- Use `==` comparing numbers or char (primitives)
boolean a = (5 == 6);
boolean b = ('T' == 'F');

JavaIntro, HW1, TwelveProblems

- Any questions: feel free to ask individually
- JavaIntro will not be collected and graded
 - Intended to help you learn
 - Not intended as busy work
- TwelveProblems
 - Due tomorrow night
 - First half you can probably do already

Syllabus Highlights

- You should read the whole thing
- But pay special attention to the grading policies of the course
- <https://www.rose-hulman.edu/class/csse/csse220/201830/syllabus.html>

Review of types

- Primitives
 - int, double, char, boolean, long, ...
- Objects
 - String, ...
- Gotchas:

What is $7/2$?

Alternatives?

What is x/y if x and y are both ints?

Alternatives?

What is s after these 2 lines?

```
String s = "computer";
```

```
s.substring(0,3);
```

Alternatives?

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
- 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

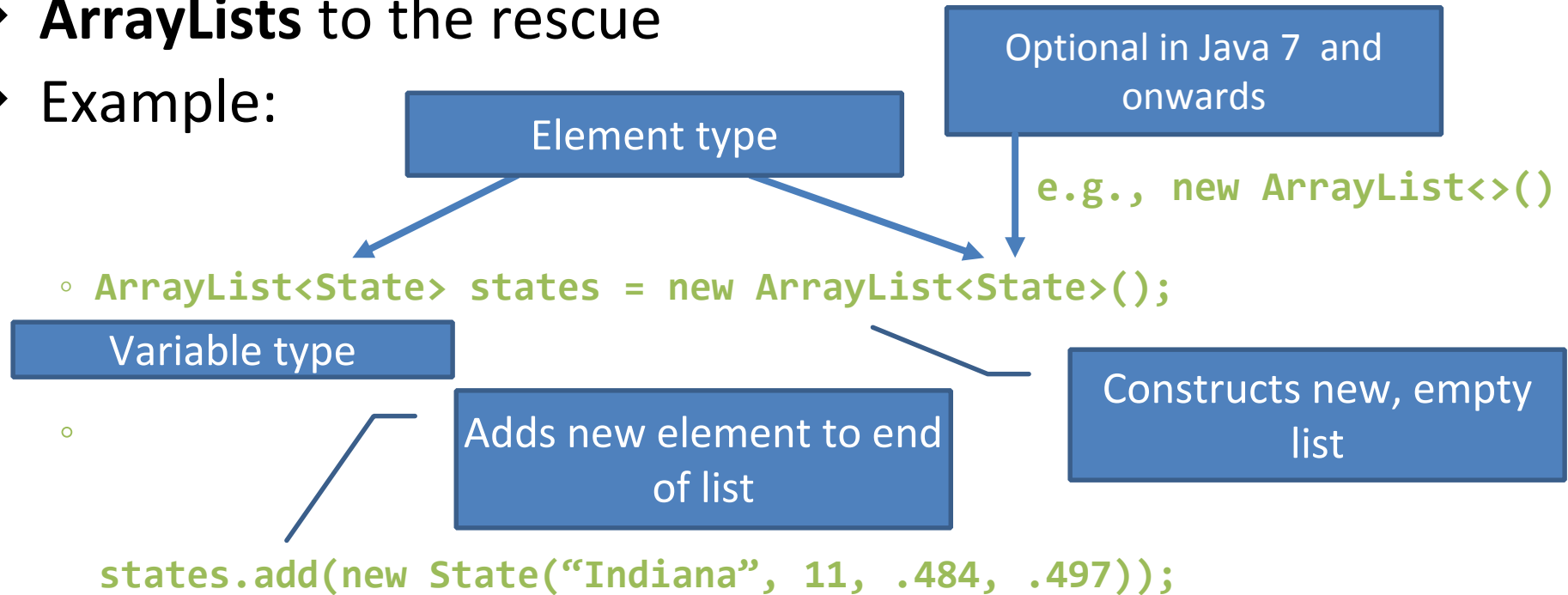
ArrayList Examples Handout

- 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)

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

▶ **ArrayLists** to the rescue

▶ Example:



▶ **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();  
}
```

Work Time

- Finish all the in-class material exercises if you haven't yet
- Work on TwelveProblems