CSSE 220 Day 8

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

Check out *ArraysAndLists* from SVN

Questions?

Exam 1 Coming!

- Tuesday **after** break, in-class
- Over chapters 1-7
- We'll review on our first day back

If there's anything that you're confused about, get it straight this week. Come see me for help!

Q1

Array Types

- What it is for:
  - Bundling a collection of objects under a single name,
  - With elements in the collection referred to by their **position**, or ***index***, in the collection (0, 1, 2, …)
- Syntax for declaring: *ElementType*`[]` *name*
- 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 Dog's. It just allocates space for referring to Dog's (all the Dog's start out as *null*)

Q2

Reading and Writing Array Elements

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

Reads the element with index 42.

Sets the value in slot 37.

- 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

Q3-Q4

Arrays: Comparison Shopping

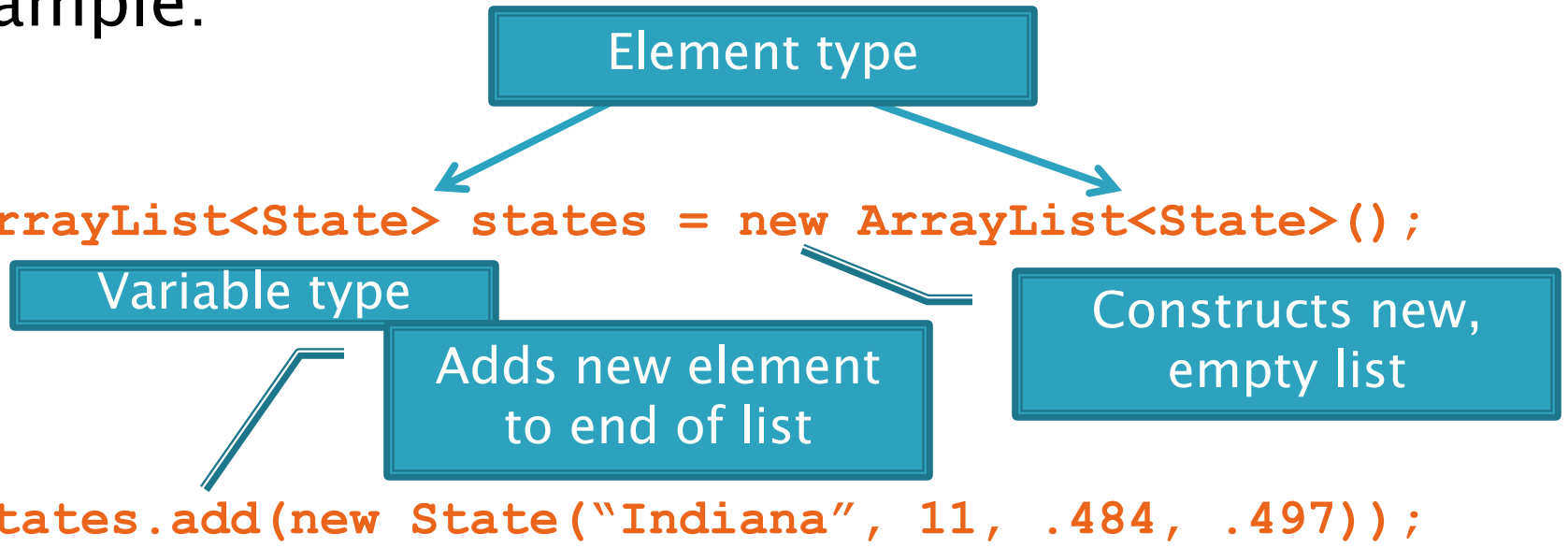
Arrays…	Java	C	Python
have fixed length	yes	yes	no
are initialized to default values	yes	no	n/a
track their own length	yes	no	yes
trying to access “out of bounds” stops program before worse things happen	yes	no	yes

Live Coding

- Investigating the Law of Large Numbers
  - A simulation using dice
- Design
- Implementation (together)
- Begin the **RollingDice** program for HW8 (in **ArraysAndLists**)

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*

Q5-Q6

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 2 and higher up one
- Can also remove:
  - `victories.remove(victories.size() - 1)`

Live Coding

- Continue **RollingDice**

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
byte	Byte
boolean	Boolean
char	Character
double	Double
float	Float
int	Integer
long	Long
short	Short

Q7

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 ArrayList's 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();
}
```

Q8

Live Coding

- Finish **RollingDice**, then continue on HW 10.

Q9–Q10