CSSE 220

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

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

Import *ArraysListPractice* from Git clone

Speed With Which Things Move

- Moving up a level in speed
- Anticipate:
  - Go through slides before class
  - Familiarize yourself with terminology
  - Read/Skim the Big Java chapters
  - Write down questions for instructor
  - Ask questions in class, or hand piece of paper with questions to instructor at beginning of class

Getting things done

- If something has a hard deadline, then set a reminder in your smart device
- Live by: “if I don’t do it now, it won’t get done”

HW1 + TwelveProblems

- We will be downloading the assignments from Moodle
- You will write the code to complete these assignments in Eclipse
- When you are done, you will upload a single .java file with your solution
- Moodle has an assignment for each of these
  - HW1.java
  - TwelveProblems.java

Review Loops: while & for Loops

While loop syntax: Similar to Python

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

For loop syntax: Different from Python

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

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

Comparing for vs. while

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


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

Important Reminder: Comparisons

- Fast rules for now:
- Use .equals() for comparing Strings

```
String alpha = "aaa";
if (alpha.equals("bbb") {
   System.out.println("Yes!")
} // end if
```

- 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 Friday night
  - First half you can probably do already

Syllabus Highlights

- Course policies: https://www.rose-hulman.edu/class/csse/csse220/201920/Docs/syllabus.html
- Late Assignments
  - Grading
  - Collegiality

Syllabus Highlights

- Schedule: https://www.rose-hulman.edu/class/csse/csse220/201920/Schedule/Schedule.htm

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 fixed 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

1. Form groups of 2
2. Look at the Array Examples Handout
   Steps 1 – 3 of handout – Built-in Java Arrays
3. Study how arrays are used and answer the questions in the quiz:
   FIRST PAGE OF QUIZ ONLY
4. Step 3 of handout: http://codingbat.com/java/Array-2
   - Work in your groups to solve:
     fizArray3, *bigDiff*, *shiftLeft*
   - If you finish early, try: *zeroFront*
   - Save your codingbat work by copy and paste
5. At bell: we move on to ArrayLists
   Steps 4 – 7 of handout

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
- Java automatically 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*)

Reading and Writing Array Elements

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

Accesses 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

Arrays: Comparison Shopping

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

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>();
```

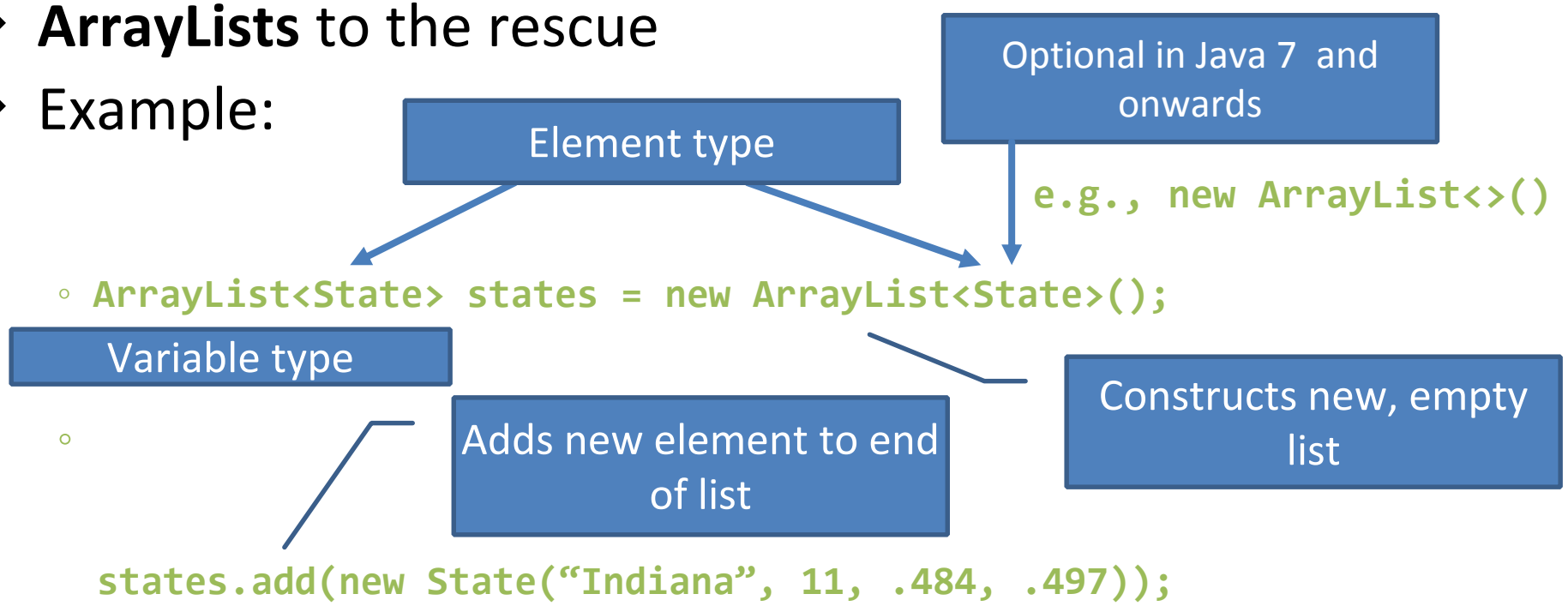
  - Creates a new ArrayList of type Type stored in variable *arl*

ArrayList Examples Handout

- Look at the ArrayList section of the examples handout
- Study how arrayLists are used and answer the questions in the quiz (page 2)
- Then solve the 3 problems in ArrayListPractice (you downloaded it from Git)

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 cannot be a primitive type
  - Not: `ArrayList<int> runs;`
  - But: `ArrayList<Integer> runs;`

- Use ***get*** method to access 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

Example List:

```
ArrayList<WorldSeries> victories =
    new ArrayList<WorldSeries>();
```

- 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)`
    this removes at the end

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***

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

- Remember to use wrapper class for array 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 k = 0; k < scores.length; k++) {
    sum += scores[k];
}
```

- 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