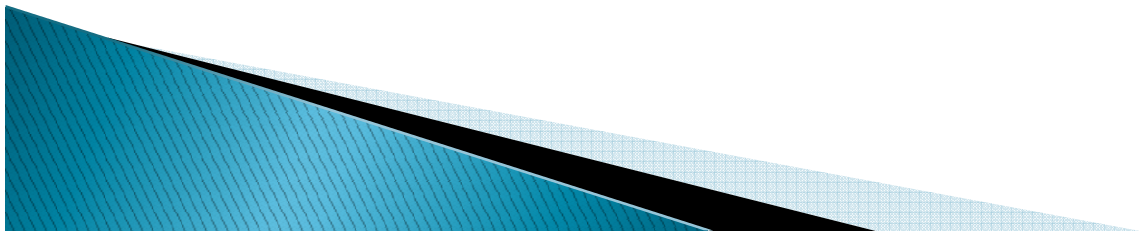
Arrays: Comparison Shopping

Arrays…	Java	C	Python
have fixed length	yes*	yes	no
are initialized to default values	yes	no	?
track their own length	yes	no	yes
trying to access "out of bounds" stops program before worse things happen * But an ArrayList – Java's "class like arrays" – does not have a fixed length; it grows by itself as needed	yes	no	yes

Array Types

- Syntax: *ElementType[] name*

- Examples:
  - A variable: `double[] averages;`
  - Parameters: `public int max(int[] values) {…}`
  - A field: `private Investment[] mutualFunds;`



Allocating Arrays

- Syntax: `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[totalVotesPossible];`
  - `Employee[] employees = new Employee[30];`

Don't forget this step!

All 30 are currently null. This statement is typically followed by a loop that initializes each element of the array to a new Employee. See next slide.

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;`

```
Dog[] dogs = new Dog[numberOfDogs];

for (int k = 0; k < dogs.length; ++k) {
    dogs[k] = new Dog(...);
}
```

Initializes an array of Dogs to actually hold Dog objects

- Index numbers run from 0 to array length – 1
- Getting array length: `elecVotes.length`

No parens, array length is (like) a field

Q3,4

Enhanced For Loop and Arrays

- Given:

```
double scores[] = …
double sum = 0.0;
```

- Old school

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

- New, enhanced for loop

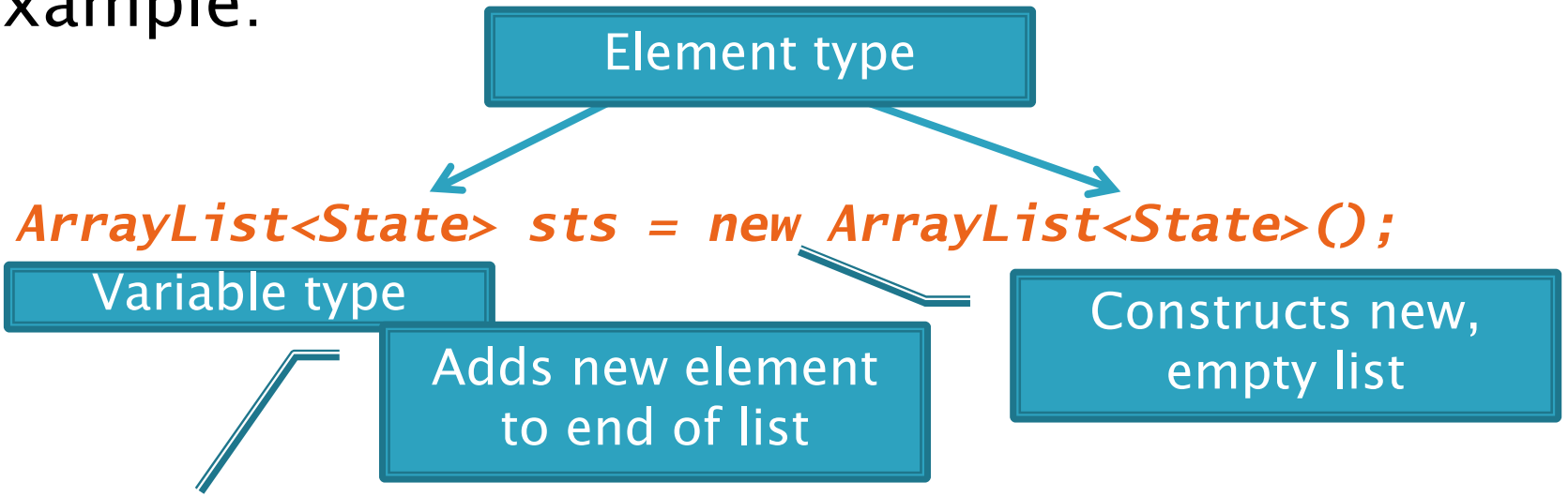
```
for (double sc : scores) {
    sum += sc;
}
```

Say "in"

- No index variable
  - Good! Abstracts away the beginning/end/index in the collection
- Gives a name (***sc*** here) to each element
  - Note: Can NOT use it to MODIFY the current slot in the array

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

- ArrayLists to the rescue
- Example:
  - `ArrayList<State> sts = new ArrayList<State>();`



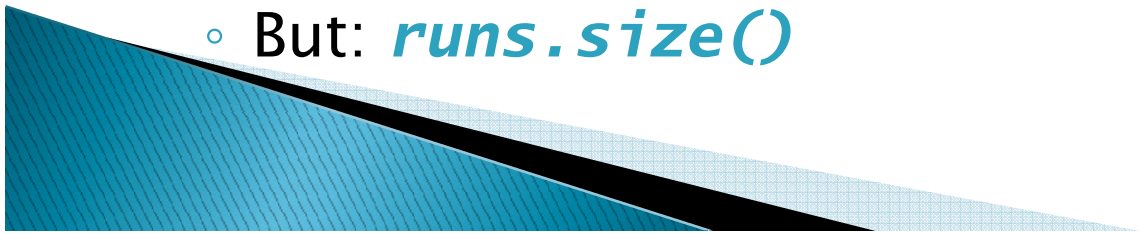

```
sts.add(new State("Indiana", 11, .484, .497));
```

- `ArrayList` is a *generic class*
  - Type in <brackets> is called a *type parameter*

Q5,6

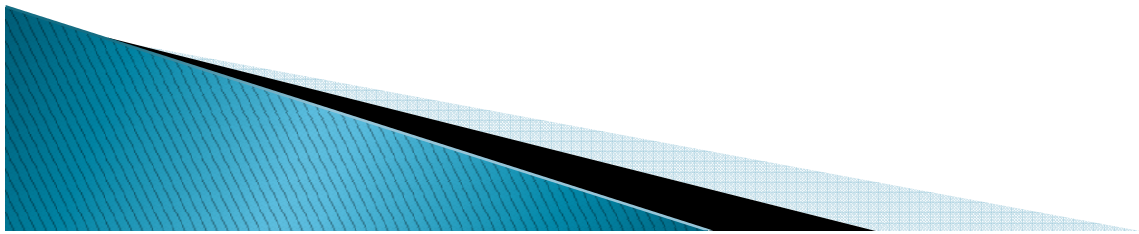
Array List Gotchas

- 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(2008));`
- 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)`



Enhanced For and ArrayLists

- Same notation as for arrays:

- ```
  ArrayList<State> states = …

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

Say "in"

- No index variable
  - Good! Abstracts away the beginning/end/index in the collection
- Gives a name (***sc*** here) to each element
  - Note: Can NOT use it to MODIFY the current slot in the array
  ```

Live Coding

- Do ElectionSimulator program

So, what's the deal with primitive types?

- Problem:
  - ArrayLists only hold objects
  - Primitive types aren't objects

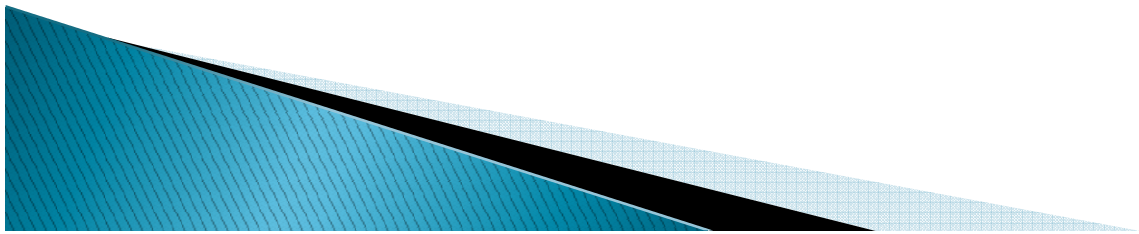
- Solution:
  - *Wrapper classes*—instances are used to "turn" primitive types into objects
  - Primitive value is stored in a field inside the object

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 ans= m * 7;`
  - Java does:
```
int temp = m.intValue() * 7;
Integer ans = 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
    ```