ARRAY LISTS

CS302 – Introduction to Programming
University of Wisconsin – Madison
Lecture 17

By Matthew Bernstein – matthewb@cs.wisc.edu

Array Lists

- **Array Lists** are objects, like arrays, that store multiple elements of the same data type
- Unlike arrays, array lists can grow and shrink as needed
- The array list class supplies methods for common tasks such as inserting and removing elements

Declaring and Creating Array Lists

- You create an array list as follows:

```
ArrayList<data_type> variable_name = new ArrayList<data_type>()
```

The data type that the array list will store

The name of the array list variable

The **new** operator is required to create the array list

The data type that the array list will store

Example Creating an Array List

- The following statement creates a new, empty, array list, called “names” that stores Strings:

```
ArrayList<String> names = new ArrayList<String>();
```

Adding Elements

- We add values using the array list's **add()** method
- Values you add to the array list must be compatible with the data type that the array list is capable of storing
- Example:

```
// Create the array list
ArrayList<String> names = new ArrayList<String>();

// Add the String "Harry" to the array list
names.add("Harry");
```

Find the Error

- What is wrong with the following code?

```
ArrayList<String> names;
names.add("Harry");
```

Array Lists Grow Automatically

- We can keep adding elements to the array list "indefinitely" and the array list will grow to accommodate the new elements
- Example:

```
ArrayList<String> names = new ArrayList<String>();

names.add("Harry");     // names now has 1 element
names.add("Jeremy");  // names now has 2 elements
names.add("Abe");       // names now has 3 elements
```

Accessing Elements

- Once we add elements to an array list, we can access elements using the array list's **get()** method
- The get method returns an element at a specified index of the array list
- Do not attempt to retrieve an element from an invalid index
- Example:

```
names.get(3);  // Will return the element at the 3rd
               // index of the array list
```

Example Accessing Elements

```
ArrayList<String> names = new ArrayList<String>();
names.add("Harry");
names.add("Emily");
names.add("Cindy");

String thirdName = names.get(2); // Will store "Cindy"
```

Array Lists are Reference Types

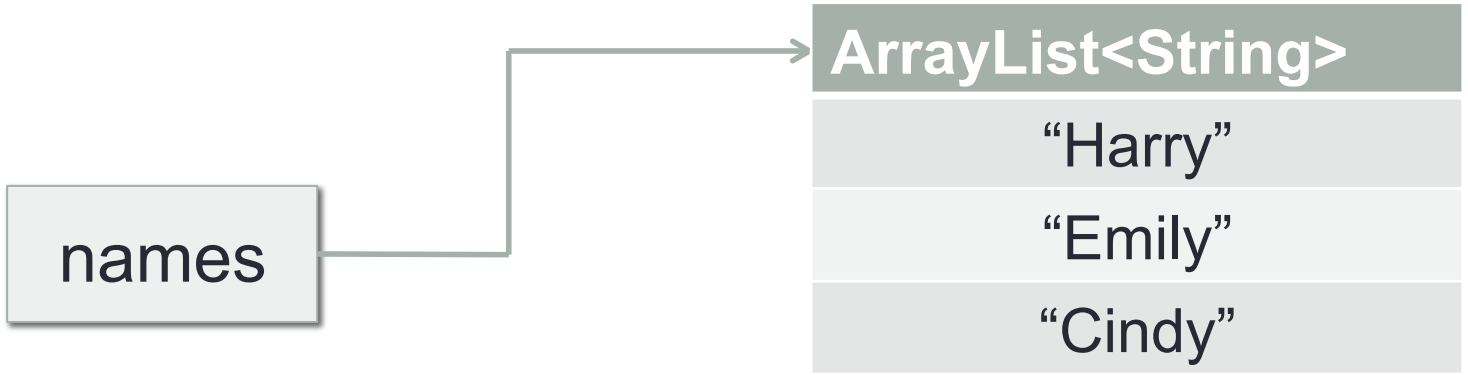
- When you declare an array list variable, the variable itself does not store any values. Rather, the variable points to the location in memory of the array list object.
- **In the Java programming language, all objects are reference types**

Memory Diagram of Empty Array List

```
ArrayList<String> names = new ArrayList<String>()
```

Memory Diagram After Adding Elements

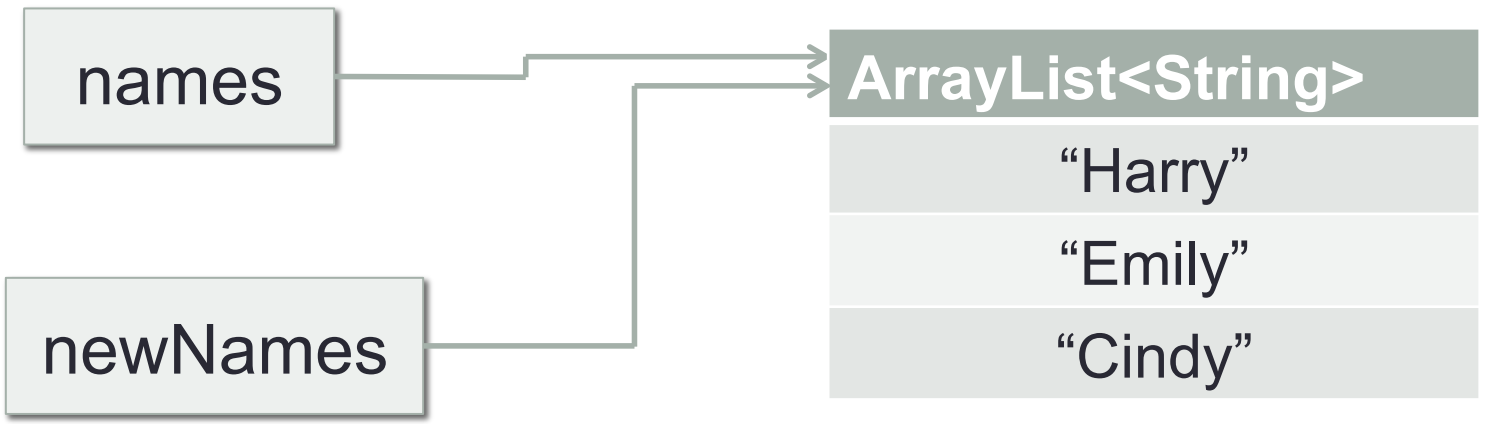
```
names.add("Harry");
names.add("Emily");
names.add("Cindy");
```



Copying Array List Variables

- If we copy the value of one array variable to another array variable, we are only copying the reference from the to the array list and not the array list itself:

```
ArrayList<String> newNames = names;
```



Retrieving Number of Elements

- To get the number of elements in an array list, use array list's **size()** method
- Example:

```
ArrayList<String> names = new ArrayList<String>();
names.add("Harry");
names.add("Emily");
names.add("Cindy");

int numNames = names.size(); // Will store 3
```

Over-writing Elements

- You can over-write the value of an element at a specific index of an array list using the **set()** method
- Example:

```
names.set(2, “Carolyn”); // The String at index 2
                         //is now “Carolyn”
```

Other Useful Array List Methods

- Read their Javadocs to get a better understanding of how they work
  - contains()
  - clear()
  - indexOf()
  - remove()
  - removeAll()
  - toString()
  - toArray()

IMPORTANT DETAIL

- You cannot store primitive data types inside an Array List.
- You can only store reference types
- What if you need to primitive types like ints, doubles, and booleans?
  - Use wrapper objects

Wrapper Classes

- Wrapper classes encapsulate a single primitive data type inside an object

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

Converting Between Wrapper Objects and Primitive Types

- Conversion between primitive types and the corresponding wrapper class is automatic.
- This process is called **auto-boxing**

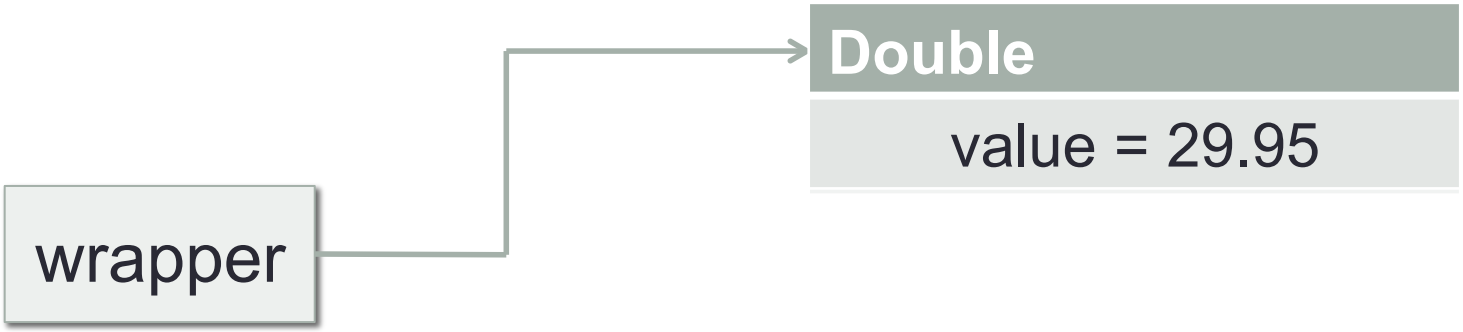
- Example:

```
// Primitive type assigned to wrapper object
Double wrapper = 29.95;

// Wrapper object assigned to primitive type
double x = wrapper;
```

Wrappers are Reference Types (All Objects are References)

```
Double wrapper = 29.95;
```



Array Lists vs. Arrays

- Use an Array if:
    1. The size of a collection never changes
    2. You collect a long sequence of primitive type values and you are concerned about efficiency
- Otherwise, use an Array List

Cool CS Link of the Day

- Top Technological Breakthroughs of 2013
- http://www.technologyreview.com/featuredstory/513696/deep-learning/