Computer Science II
Handout 9

ArrayLists – arrays with more

- The ArrayList class is provided by the Java library as a more flexible alternative to arrays

```
import java.util.ArrayList;
```

- An ArrayList Object stores multiple Object references in a given sequence
  - Much like arrays!

ArrayLists – arrays with more

So why use ArrayLists?

- Regular arrays have a *fixed size*
- This can be a disadvantage in terms of execution time and space
  - You need to know the size of your array before-hand or else you risk wasting resources!
- ArrayList Objects can have a dynamic number of elements
  - The number of elements may change at run time

ArrayLists – arrays with more

- ArrayList elements may be added and removed, and the size will be automatically updated
  - Adding one element increases the size
  - Removing one element decreases the size

- ArrayList elements are *always* Object references
  - Never primitive types!

ArrayLists – constructor

- The ArrayList class has the following two constructors:

```
new ArrayList<T>()
```

This creates an ArrayList with a *default capacity* of 10 elements

```
new ArrayList<T>(cap)
```

This creates an ArrayList with a *capacity* of **`cap`** elements

- Here, **`T`** indicates a *generic type*, which is any type of Object
  - Such as a String, a Scanner, a Rectangle, etc.

ArrayLists – other methods

- The generic type **`T`** must match a known Object type in your code
  - The ArrayList will then be created to store references to *this* type of Object

- ArrayList Objects have various useful methods:

  - **`add(T element)`**
    Add the specified element to the *end* of the ArrayList

  - **`add(int index, T element)`**
    Add the specified element to the specified position in the ArrayList. The element in this position (and all subsequent elements) are shifted one position to the right

ArrayLists – other methods

- **`toString()`**

  Returns the contents as a formatted array: [element0, element 1, …, element n]

- **`remove(int i)`**

  Removes the element at the specified position. Shifts all subsequent elements one position to the left

- **`remove(T element)`**

  Removes the *first* occurrence of the specified element from this ArrayList, if it is present. (Returns true/false if it was/wasn't in this ArrayList)

ArrayLists – other methods

- **`get(int i)`**

  Returns the element (of type T) at the specified index.

- **`set(int i, T element)`**

  Replaces the element at the specified index with the specified element.

- **`clear()`**

  Removes *all* elements from this ArrayList; after calling this method, the ArrayList will be empty.

- **`removeRange(int fromIndex, int toIndex)`**

  Removes all elements with indices between **`fromIndex`** (inclusive) and **`toIndex`** (exclusive).

ArrayLists – other methods

- **`size()`**
  Returns the number of elements in the ArrayList.

- **`isEmpty()`**
  Returns true if the ArrayList contains no elements; false otherwise.

- **`contains(T element)`**
  Returns true if this ArrayList contains the specified element.

- **`indexOf(T element)`**
  Returns the index of the *first* occurrence of the specified element, or -1 if this ArrayList does not contain the element.

- **`lastIndexOf(T element)`**
  Returns the index of the *last* occurrence of the specified element, or -1 if this ArrayList does not contain the element.

ArrayLists – example

- Use the ArrayList class to create a list of Strings of musical performers, called musicians
  - Add three elements (Taylor Swift, Nickelback, Lady Gaga)
  - Print the list
  - Retrieve the first element (index 0)
  - Remove the item at index 1
  - Add a new element (Blink182) at index 1
  - Replace the item at index 2 with The Beatles

ArrayLists – example

- Declare an ArrayList with the appropriate type
- Add the three performers/groups:
- Print the elements
- Print item at index 1

ArrayLists – example

- Remove item at index 1
- Add Blink182 at index 1
- Replace with The Beatles at index 2

ArrayLists – example 2

- Use the ArrayList class to create a list of musical performers, called musicians
  1. Add the same three initial elements to the list, and a fourth element “Aerosmith”
  2. Print out the full list and its size
  3. Check whether “Aerosmith” is in the list; if so, give its index
  4. Print out the element at index 1
  5. Remove the element at index 1
     - (which element is this?)
  6. If the element “The Beatles” is not in the list, add it to the end of the list
  7. If the element “Oscar Peterson” is not in the list, add it to:
     - The third position, if there are at least three bands already in the list, or
     - The end of the list

```
import java.util.ArrayList;
public class ArrayListDemo2 {
        public static void main(String[] args) {
                ArrayList<String> musicians = new ArrayList<String>();









                // Continued ...
```

```
            // Continuing ...

            System.out.println("List is now: " + musicians);
        }
    }
```

ArrayLists – primitive types

- Since ArrayLists only store Object references, we cannot use any primitive types directly

- Java provides a way around this: the standard library includes *wrapper classes* that "wrap around" a primitive data type variable and provide object-oriented methods

- Each primitive type has its own wrapper: **`int`** has Integer, **`double`** has Double, etc.
  - Each class name starts with a *capital letter*
  - Each class name is the "full name" of the primitive type (e.g., Integer for **`int`**)

ArrayLists – primitive types

- As a convenience, Java will automatically convert between primitive data types and their respective wrapper class
  - This is called *autoboxing* – the primitive data type is automatically "boxed" inside a wrapper Object

```
Scanner kb = new Scanner(System.in);
Integer num = kb.nextInt();  // Returns an int
```

- This uses the Scanner to read an **`int`** into the Object referenced by **`num`**

ArrayLists – primitive types

- We can use this to make working with primitives possible (and convenient) with an ArrayList:

```
ArrayList<Integer> list = new ArrayList<Integer>();
Scanner kb = new Scanner(System.in);
Integer num = kb.nextInt();  // reads an int
list.add(num);  // stores an autoboxed int
```

ArrayLists – example 3

- Create an ArrayList of Integers, then:
  - Populate the list with twenty random values between 1-10
  - Print the contents of the list
  - Search the list to find the first and last index of a number given by user input

```
import java.util.*;
public class ArrayListDemo3 {
        public static void main(String[] args) {
                ArrayList<Integer> list = new ArrayList<Integer>();
                Scanner kb = new Scanner(System.in);

                for(int i=0; i<20; i++) {



                }
                System.out.println(list);
                System.out.print("\nEnter number to search: ");
                Integer searchVal = kb.nextInt();









        }
}
```

ArrayLists – example 4

- Create an ArrayList that stores Rectangle Objects (assume you already have the Rectangle class written from earlier)
  - Populate the list by reading in length/width measurements from the user, as integers; continue reading until -1 is entered for either the length or width
  - Print the full ArrayList

```
import java.util.*;
public class ArrayListDemo4 {
        public static void main(String[] args) {
                ArrayList<Rectangle> list = new ArrayList<Rectangle>();
                Scanner kb = new Scanner(System.in);

                System.out.print("\nEnter length and width: ");
                int length = kb.nextInt();
                int width = kb.nextInt();









                System.out.println(list);  // toString() called on Rectangle
        }
}
```

ArrayLists – example 4

- Notice that the Rectangle toString method is called by the ArrayList class!
  - Another good case for giving meaningful output in your toString method

- If Rectangle were written without a toString method, what sort of output would we expect to see?

ArrayLists – example 5

- Create an ArrayList that stores positive double values
  - Solve each task by writing a static method in the same class (for simplicity)

1. Accept input from the user to populate the list (-1 to terminate input)
2. Compute the average of the values
3. Swap the first and last elements of the list
4. Find the index of the largest value in the list
5. Find the index of the smallest value in the list
6. Swap the largest and smallest values

```
import java.util.*;
public class ArrayListDemo5 {
        public static void main(String[] args) {
                ArrayList<Double> scores = new ArrayList<Double>();

                fillArrayList(scores);  // #1
                System.out.println(scores);

                System.out.println("Average = " + computeAverage(scores));  // #2

                swapFirstLast(scores);  // #3
                System.out.println("Swapped first/list: " + scores);

                System.out.println("Index of largest = " + findLargest(scores));  // #4

                System.out.println("Index of smallest = " + findSmallest(scores));  // #5

                swapLargeSmall(scores);  // #6
                System.out.println("Swapped largest/smallest: " + scores);
        }
        // Continued ...
```

```
// Continuing ...

// Method #1
public static void fillArrayList(ArrayList<Double> a)
{
    Scanner kb = new Scanner(System.in);
    System.out.println("Enter scores, end with -1: ");
    Double num = kb.nextDouble();
    while(num != -1) {
       a.add(num);
       num = kb.nextDouble();
   }
}


public static double computeAverage(ArrayList<Double> a)
{
    // TODO: add code
}

public static void swapFirstLast(ArrayList<Double> a)
{
    // TODO: add code
}

public static int findLargest(ArrayList<Double> a)
{
    // TODO: add code
}
```

```
   public static int findLargest(ArrayList<Double> a)
   {
      // TODO: add code
   }

   public static void swapLargeSmall(ArrayList<Double> a)
   {
      // TODO: add code
   }

}
```

Example 5: Method #2

```
public static double computeAve(ArrayList<Double> a) {
    double total = 0;

    for(int i=0; i<a.size(); i++)
        total += a.get(i);




}
```

Example 5: Method #3

```
public static void swapFirstLast(ArrayList<Double> a) {
    double first = a.get(0);  // Copy first element









}
```

Example 5: Method #4

```
public static int findLargest(ArrayList<Double> a) {
    double largest = a.get(0);
    double index = 0;









    return index;
}
```

Example 5: Method #5

```
public static int findSmallest(ArrayList<Double> a) {



}
```

Example 5: Method #6

```
public static void swapLargeSmall(ArrayList<Double> a) {

}
```