CS 1113
LAB #7: ArrayLists, HashMaps and Java memory management
Dr. Cline

Lab Goals

1) To teach you about ArrayLists and give example uses
2) To compare the Java primitive types and Wrapper classes
3) To show you how to do associative indexing with HashMap

Motivation

Java arrays have a significant shortcoming. They cannot be resized after they have been instantiated. This can be a major problem, since you often don't know how many elements a list of items will need when creating it. One strategy is to start out with a small number of elements, then create a larger array when more space is needed, and copy the old array into the new one. This can be a hassle, and complicates the code, however. ArrayList performs this operation automatically, hiding the details inside a class, freeing up valuable "programmer cycles" for more important tasks.

Another drawback with arrays is that in many cases you have to search for the element that you want. HashMap provides a way to store and look up objects using a "key" value. For example, you might want to have a list of student records, and be able to look up a student by name. HashMap gives you this functionality.

ArrayLists

ArrayList is a "generic", also called a "parameterized type". This means that you can declare different kinds of ArrayList to hold different object types. Declaring a generic in Java is done with the angle bracket notation <>. For example, The following line of code creates a ArrayList of Strings:

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

There are a couple of things to notice here. One is that an ArrayList declaration has two types (1) ArrayList itself, and (2) the kind of objects that the ArrayList will hold. Also, the right side of the statement requires both angle brackets < >, and parentheses ( ). We can think of ArrayList<String> as the type of object we are creating. Just like any other object, to instantiate it, we call the constructor using "new" and parentheses (). Unlike an array, we don't have to give the ArrayList an initial size. It will expand as needed. However, you can declare an initial capacity by passing a parameter to the constructor. The following creates an ArrayList of strings with an initial capacity of 100:

```
ArrayList<String> stringList = new ArrayList<String>(100);
```

Write the answers to the numbered questions (Q1...) in a file called **lab7answers.txt**. At the top of this file, put your name and section number.

Q1) What code would you use to create an ArrayList called studentList that holds objects of type Student?

To eliminate some of the redundancy of this notation, Java allows you to omit the "String" inside the angle brackets on the right hand side (called type inference), so the following is equivalent to the above statement:

```
ArrayList<String> stringList = new ArrayList<>(100);
```

Notice that the angle brackets are still required. To add an element to the list, use the method "add":

```
stringList.add("Mis pantalones son azules");
```

Java does not allow operator overloading (like C++), so we cannot use square brackets to access elements of an ArrayList (C# allows this). Instead, we use the method "get".

```
String name = stringList.get(0);
```

You can remove an item using "remove" When this happens, everything from the given item index gets moved over. This has speed implications. If you remove things at the beginning of the list, it is slower than if you remove something at the end:

```
stringList.remove(2);
```

Q2) What code would you use to remove the last element of stringList? Write the answer in lab7answers.txt.

There is nothing magic about adding or removing things from an ArrayList. Underneath, it just holds an array. Some of the values in the array are not used, and the current "capacity" of the ArrayList is the size of this array. When you try to add another element to the ArrayList, if the underlying array does not have enough space, the ArrayList allocates a new array and copies the old values to it. Then the old array is discarded (nothing points to it anymore) so the garbage collector can recycle its memory.

An ArrayList (and other generics) cannot hold primitive types, so ArrayList<int> is not allowed. To remedy this shortcoming, the designers of Java created a set of wrapper classes, one for each primitive type. The names of wrappers are mostly the same as the primitive types, but distinguished from them by starting the name with a capital letter. The following table gives the set of primitive and wrapper types in Java:

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

Even though an ArrayList cannot hold primitive types, you can pretend it does using features called Autoboxing and Autounboxing. If a method takes a parameter that is a wrapper, Java allows you to pass the corresponding primitive (Autoboxing). On the other hand, if a method returns a wrapper type, you can put it into the corresponding primitive type without a cast (Autounboxing). Thus, we can do the following:

```
ArrayList<Integer> intList = new ArrayList<Integer>();
intList.add(1);
int n = intList.get(0);
```

This is much nicer than having to type intList.add(new Integer(1)), for example.

Q3) What is autoboxing?

Q4) What is the wrapper class for double?

Iterating through the elements of an ArrayList

You can use the standard or enhanced for loops to iterate through the elements of an ArrayList, using the size() method in place of the length variable.

Q5) What standard loop would you use to print the elements of myList, one string per line?

Q6) What enhanced loop would you use to print the elements of myList, one string per line?

Common algorithms for ArrayLists and arrays

There are a number of algorithms that come up again and again when using arrays or ArrayLists. Some of these are: filling in the elements of the list based on a computation, computing the sum and average value, computing the minimum and maximum, adding and removing elements, printing the list list, swapping elements, and performing a linear search. Most of these can be done with a simple loop.

Part 1) Copy and finish the program below, filling in the code for the many methods.

```
// Lab 7a
// <Your name>
// <Your section>

import java.util.*;

public class Lab7a
{
	public static void main(String args[])
	{
		ArrayList<Double> list = createSquaresList(10);
		printList(list);
		removeElement(list, 4);
		printList(list);
		swapElements(list, 2, 6);
		printList(list);

		double max = getMaxValue(list);
		double ave = getAverage(list);

		// Print the max and average
		// [Your code here]

		int idx1 = linearSearch(list, 4);
		int idx2 = linearSearch(list, 75);

		// Print the two indices
		// [Your code here]
	}

	public static ArrayList<Double> createSquaresList(int n)
	{
		// Create an ArrayList with the squares of n numbers, 0 to n-1
		// Return the ArrayList
		// [Your code here]
		return null;
	}

	public static double getMaxValue(ArrayList<Double> list)
	{
		// [Your code here]
		return 0.0;
	}
```

```
	public static double getAverage(ArrayList<Double> list)
	{
		// [Your code here]
		return 0.0;
	}

	public static void removeElement(ArrayList<Double> list, int index)
	{
		// Remove the specified element.
		// [Your code here]
	}

	public static void swapElements(ArrayList<Double> list, int a, int b)
	{
		// Write code that swaps elements a and b of the ArrayList
		// Hint: you need a temporary variable.
		// [Your code here]
	}

	public static int linearSearch(ArrayList<Double> list, double val)
	{
		// Use a linear search to find the index of a particular value.
		// Return that index, or -1 if the value is not found.
		// Do not use list.indexOf(val)
		// [Your code here]
		return -1;
	}

	public static void printList(ArrayList<Double> list)
	{
		// Print out the number of the list on one line, separated by
		// a comma and space, with a newline at the end.
		// [Your code here]
	}
}
```

Part 2) Once it is working, run your program, redirecting the output to **lab7aout.txt.**

HashMaps

A hashtable is a data structure that allows the user to look up things by a key value. Java implements a hashtable called "HashMap". The way it works is to specify two types of classes. One is the "key", and another is the "value". You can put data values into the HashMap at an "index" specified by the key.

Like ArrayList, HashMap is a generic class, so you declare it using angle brackets. For HashMap, you must include both the key type and the value type in this declaration, so the declaration looks like HashMap<Key,Value>. As an example, the following declares a HashMap with a String key and an Integer value:

```
HashMap<String, Integer> h = new HashMap<String, Integer>();
```

HashMap has quite a few useful methods. A couple of the most important are "put", to put (key,value) pairs into the HashMap, and "get" to get a value based on a key. For example:

```
h.put("Hello", 1);
int helloVal = h.get("Hello");
```

Other methods of HashMap clear the table, determine if a certain key or value is stored in the table, remove a

value, and get a list of all keys in the table.

Q7) Suppose that your program has a HashMap called plants with keys that are Strings and values that are Integers. What code would you use to print the value associated with the string "grass"?

Part 3) Copy and complete the following program that does a little bit with HashMap.

```
// Lab 7b
// <Name>
// <Section>

import java.util.*;

public class Lab7b
{
	public static void main(String args[])
	{
		// Create a new HashMap called "songStars"
		HashMap<String, Integer> songStars = // [Your code here]

		// Yes, you should listen to all these on Youtube.
		songStars.put("The ballad of Bilbo Baggins", 5);
		songStars.put("A still more glorious dawn", 4);
		songStars.put("A finite simple group of order two", 4);
		songStars.put("Code monkey", 4);
		songStars.put("Fish heads", 2);
		songStars.put("I'll form the head", 3);
		songStars.put("Honeybee", 5);
		songStars.put("Silver apples of the moon", 4);

		// Get the number of stars for "Code monkey", and print the # of stars
		// [Your code here]

		// Remove the song "Honeybee"
		// [Your code here]

		// Determine if "Honeybee" is still in the table.
		// If it is in the table, print "Honeybee still in table".
		// Otherwise print "Successfully removed Honeybee".
		// [Your code here]
	}
}
```

Part 4) Make sure that your code compiles and runs correctly.

Part 5) Turn in your lab files to D2L, including

```
Lab7a.java
Lab7b.java
lab7aout.txt
lab7answers.txt
```