Lab 3
Array Lists

Although arrays are handy to group values of the same type, they are limited when it comes to change their contents dynamically, that is, when adding, deleting or moving items around. Fortunately, Java comes with a flexible data structure that can handle these cases: the **ArrayList** class.

This lab begins by explaining that array lists can only store class types and not primitive types, like int and double. To overcome this limitation Java implements a wrapper class for each of the primitive types in the language. As it will be explained, wrapper classes and a feature called “auto-boxing” simplify handling array lists of primitive types.

1. Wrapper Classes & Auto-boxing

To hold values, Java has **primitive types** (such as *int*) and **class types** (for objects, such as *String*).

Primitive types store values efficiently are stored differently than class types are, which means that primitive types and class types are incompatible. To overcome that, Java has created classes that allow primitives to be stored as objects. For example, there is an **Integer** class that stores an **int** value and a **Boolean** class that stores a **boolean** value. These classes are known as **wrapper** classes because each implements a class to hold (or wrap) a primitive value. There is a “wrapper” class for each primitive type.

In earlier versions of Java, using wrapper objects required using a constructor (to set a value) and a get method (to retrieve a value). This approach follows object-oriented conventions but makes code that is awkward to write and read. For example, the table below shows code that could be written with primitives (on the left, 33 characters) and wrappers (on the right, 91 characters).

Primitives	Wrappers

`int x, y, z;`
`x = 3;`
`y = 5;`
`z = x+y;`
`System.out.println("z is " + z);`
	`Integer x, y, z;`
	`x = new Integer(3);`
	`y = new Integer(5);`
	`z = new Integer(x.intValue() + y.intValue());`
	`System.out.println("z is " + z.intValue());`

Realizing that programmers would not be happy using wrappers, the creators of Java modified the language to automate setting and getting values. This feature, which is called **auto-boxing**, allows using wrapper objects in the same way that primitive types are used. As shown below, the only difference between using primitives and wrappers is the type definition of the variables (*int* vs. *Integer*)

Primitives	Wrappers (with auto-boxing)

`int x, y, z;`
`x = 3;`
`y = 5;`
`z = x+y;`
`System.out.println("z is " + z);`
	`Integer x, y, z;`
	`x = 3;`
	`y = 5;`
	`z = x+y;`
	`System.out.println("z is " + z);`

Exercise 1

Create a new Eclipse project named **Lab03** with a **Lab03One** class in which to write the methods below:

- `public static Double` **`getMedian`**`( Integer[] array )`
- `public static Character[]` **`getDigits`**`( Character[] array )`

Follow the steps below to implement them:

1. Write the implementation of the method **getMedian**.
   It receives an (non-ordered) array of Integer objects and returns the median value (or 0 if the array is empty). The median of an ordered list of numbers is the number in the middle (if the number of values is odd) or the average of the 2 numbers in the middle (if the number of values is even). For example, the median in the list {19, 20, 36} is 20; the median in the list {19, 20, 21, 22} is 20.5.
2. Write the implementation of the method **getDigits**.
   It receives an array of Character objects and returns a new array with all characters in the array (including duplicates) that are digits (that is, '1', '2', '3', …, '8', '9', '0'). The order of digits from the original array should be kept in the new one.

Download the JUnit file **Lab03OneTest.java** and import it into the project; then modify the Build Path to add JUnit4, and use it to test your implementation.

Exercise 1 - Get Instructor's Signature

2. Java's ArrayList Class

An array list is similar to an array in that it holds a group of values in order. However, this is where similarities end and their differences begin. Their main differences are:

- To use array lists, you should **import java.util.ArrayList** into your programs.
- Array lists are classes created using **new ArrayList<*type*>()**, where *type* is the type of elements the list will contain. For example, to create an array list of strings named *list* we should use the statement "ArrayList<String> list = new ArrayList<String>();"

- You do not need to specify the number of elements in an array list. The size of a list is flexible and will grow and shrink when you add and remove elements from it.
- Whereas plain arrays used square brackets ("[ ]") to access their elements, array lists use **methods**. For example, given an array list named *list*,
    - To see what the element is located at position *i*, you should use **list.get(i)**.
    - To add an element to position *i*, you should use **list.add(i, element)**; if position doesn't matter, you could use **list.add(element)** to add the element at the end.
    - To learn how many elements are in the list, you should use **list.size()**.
    - There are many other useful methods to handle array lists. To find out, you will need to look under "ArrayList" in the **Java API**.
- Since the values in array lists are objects, you should always use their **equals** method to compare them (unless they are wrapper classes for primitives). **Remember**: to find out whether 2 primitive type values are equal you would use "==" but to find out whether 2 object type values are equal you would use their "equals" method). For example, to find out whether 2 values in an array list of strings are the same (let's say the first and second strings), you could use code similar to the one below:

```
ArrayList<String> aList = new ArrayList<String>();
…
String first  = aList.get( 0 );
String second = aList.get( 1 );
if (first.equals( second )) {
      // do something if they are the same
}
```

In this section we will practice writing methods that use array lists.

Exercise 2

Create a new class named **Lab03Two** (within project **Lab03**) in which to write the methods below:

- `public static int` **`getTally`**`(ArrayList<Integer> list, int number)`
- `public static int` **`getFirstIndex`**`(ArrayList<Integer> list, int number)`
- `public static void` **`doReverse`**`(ArrayList<Integer> list)`
- `public static ArrayList<Integer>` **`getSorted`**`(ArrayList<Integer> one, ArrayList<Integer> two, boolean ascendingly)`
- `public static ArrayList<Integer>` **`getUnion`**`( ArrayList<Integer> one, ArrayList<Integer> two )`
- `public static ArrayList<Double>` **`getIntersection`**`( ArrayList<Double> one, ArrayList<Double> two )`
- `public static ArrayList<String>` **`getDifference`**`( ArrayList<String> one, ArrayList<String> two )`

Follow the steps below to implement them (union, intersection & difference are shown in Figure 1):

1. Write the implementation of the method **getTally**.
   It receives an array list of Integer objects and a number to tally, and returns the number of times

that this given number is found in the array list. For example, given an array list with the numbers {1, 2, 3, 4, 3, 2, 1, 2 and 2}, and passing 3 as the number to tally, the method should return 2.

2. Write the implementation of the method **getFirstIndex**.
   It receives an array list of Integer objects and a number to find, and returns the index in which this number is first found in the array list (from the beginning of the array up) or -1 if the number is not found. For example, given the array list {1, 2, 3, 4, 3, 2, 1}, and passing 3 as the number to find, the method should return 2.
3. Write the implementation of the method **doReverse**.
   It receives an array list of Integer objects and modifies it so that it holds all its numbers in reversed index order. For example, given the array list {7, 6, 10, 3, 2, 9, 14, 12}, the method should modify the list so that it contains {12, 14, 9, 2, 3, 10, 6, 7}. The method doesn't have a return value.
4. Write the implementation of the method **getSorted**.
   It receives 2 array lists of Integer objects and a Boolean indicating the sorting order (ascending, if true; descending, if false), and returns an array list with all the elements from both arrays sorted. For example, given the array lists {1, 2, 3, 4} and {3, 2, 1}, and sorting in ascending order, the method should return {1, 1, 2, 2, 3, 3, 4}; and given the array lists {5, 3, 1} and {2, 4}, and sorting in descending order, the method should return {5, 4, 3, 2, 1}.
5. Write the implementation of the method **getUnion**.
   It receives 2 array lists of Integer objects and returns a new array list with all values from both array lists (with no duplicates). For example, given the array lists {2, 3, 4} and {3, 5, 0}, the method should return {2, 3, 4, 5, 0} (order is not important). The original array lists should not be modified.
6. Write the implementation of the method **getIntersection**.
   It receives 2 array lists of Double objects and returns a new array list with all values that are in both array lists (with no duplicates). For example, given the array lists {0, 3, 3, 0} and {3, 5, 0}, the method should return {3, 0} (order is not important). The original array lists should not be modified
7. Write the implementation of the method **getDifference**.
   It receives 2 array lists of String objects and returns a new array list with all values that are in any array list but not in both (duplicates are not allowed). For example, given the array lists {2, 3, 3, 4} and {3, 5}, the method should return {2, 4, 5} (order is not important). The original array lists should not be modified.

Download the JUnit file **Lab03TwoTest.java** and use it to test your implementation.

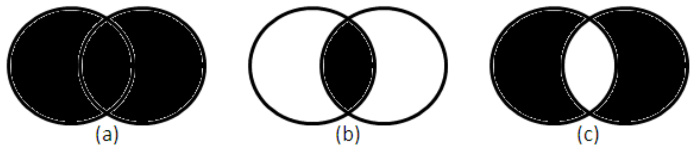

Figure 1. Union (a), intersection (b) and difference (c) of 2 sets.

Exercise 2 - Get Instructor's Signature

3. Counting Letters

In this last section, you will write a solution given the problem description below:

"To find out which letters are used the most, a publishing company wants to automate the counting of letters on the books they print. To that end, they ask you to write a program that receives book paragraphs and returns the number of letters (thus ignoring any other non-alphabetic characters) in these paragraphs. The program should be case-insensitive (which means that upper- and lower-case letters count as the same letter; for example, both 'A' and 'a' count towards the letter A).

Exercise 3

Create a new class named **Lab03Three** (within project **Lab03**) in which to write the method below:

- `public static int[] getLetters( ArrayList<String> list )`

Follow the steps below to implement it:

1. Write the implementation of the method **getLetters**.
   It receives an array list of String objects and returns an array of (primitive) integers with the count of all letters in the array list of strings. Each string in the array list can contain any number of characters (alphabetic or otherwise). The returned array of integers should always have 26 values, where each value represents a letter: the first value (at index 0) is the number of letters 'A', the second value (at index 1) is the number of letters 'B', and so on, until the last value with the number of letters 'Z'.

   Look for useful string methods under the "String" entry in the Java API.

Download the JUnit file **Lab03ThreeTest.java** and use it to test your implementation.

Exercise 3 - Get Instructor's Signature

Lab 3 Instructor's Signature Page

Student's Name:	____	Student ID	__________________

Exercise 1:	_____________________________	JUnit tests passed:	_____________
Exercise 2:	_____________________________	JUnit tests passed:	_____________
Exercise 3:	_____________________________	JUnit tests passed:	_____________