Lab 6: Loops, ArrayLists, and Arrays

Arrays

As you know from lecture, an array is a data structure that holds a fixed number of elements of a certain type. When you initialize an array, you specify both its size and the type of element it stores. Arrays can be one-dimensional or multidimensional (a 2-dimensional array is just an array of arrays).

```java
int[] numbers = new int[10]; // initializes 1D int array of size 10
HeadTA[] headtas = new HeadTA[4]; // 1D array of HeadTAs (size 4)
Square[][] chessBoard = new Square[8][8]; // 2D array of Squares (8x8)
```

To access an element stored in an array, we access the array at a particular index. Arrays are 0-indexed: the first element is stored at index 0, and the n\textsuperscript{th} element is stored at index n-1.

![Array Access Diagram]

To access the fourth element in the array `dogs`, we say: `dogs[3]`. We can use this notation to call a method on an object stored at a particular index: for example, “dogs[3].bark();” tells the fourth Dog in the array to bark. We could use the same notation to store a particular Dog at that index: for example, “dogs[3] = new Husky();”. The notation for multi-dimensional arrays is similar. We could access a particular Square in the 2D `chessBoard` array we created above with “chessBoard[5][7].”

Confused about arrays? Check out the Oracle Java Tutorials!

Looping Through Arrays

In Java, when we initialize an array, each of its elements is initialized to the default value of the array’s element type. This means that when you initialize an array of ints with “int[] numbers = new int[10]; ”, each element starts out as 0. When you initialize an array of
booleans, each element starts out as false. When you initialize an array of objects, each element starts out as null.

Loops are a useful tool for navigating arrays. One use case is populating an array with the values we want to store. Here’s a short code example that creates a 1-dimensional array of booleans and uses a for loop to set each element.

```java
boolean[] myBoolArray = new boolean[5]; // declare, initialize array
for (int i = 0; i < myBoolArray.length; i++) { // for all array’s indices:
    myBoolArray[i] = true; // set element at current index to true
}
```

The result of this code is an array that looks like: {true, true, true, true, true}. If at a later point we need to change any of the elements in the array, we can do so easily: for example, “myBoolArray[1] = false;” changes the second element of myBoolArray, making it: {true, false, true, true, true}.

Check out the Loops lecture slides if you need a refresher on for loops before moving on!

---

**Coloring Arrays**

You’re going to initialize and populate a few `javafx.scene.paint.Color` arrays of specified dimensions, using different colors to form patterns. You’ll also have to write code to allow you to visualize the arrays you’ve created.

1. Run `cs015_install lab6` to install this lab’s stencil code. Open Eclipse and check out the contents of the `lab6` package (refresh Eclipse if you don’t see it). For the first exercise, you’ll be working within the file `ArrayBuilder.java`. Open it up now.

2. The `ArrayBuilder` class contains one method called `displayArray` followed by four methods which will each build a 2D array of colors. One method, `buildStripeArray`, has been written for you. Run the program by right-clicking on `lab6`’s `App.java` class and choosing “Run as >> Java Application”. Click the Stripes button once the window appears. Nothing shows up!

   This is because the `buildStripeArray` method only initializes/returns an array of colors. It does not actually create `Node` objects that can be added to a parent `Pane`, or add those `Nodes`. Remember - arrays are ways to logically organize objects within your program. They don’t have anything to do with JavaFX graphics.

3. To display this array we need to convert it into an array of rectangles with colors corresponding to the array created in `buildStripeArray`, and then add these rectangles to a `Pane`. You will need to complete the `displayArray` method in order to accomplish this. We have already handled converting an array of `Colors` into an...
array of Rectangles for you.

○ In displayArray:
  i. **Clear** all of the children of parentPane to get rid of any previously displayed arrays. (Hint: the Pane’s list of children is a List, just like an ArrayList... see below for descriptions of ArrayList’s methods!)

  ii. Loop through the array of Rectangles, adding each one to parentPane.

4. In order to check that your displayArray is working, run the program to check that the stripe array returned by buildStripeArray is displayed correctly. Once the window appears, click the “Stripes” button (under the “Arrays” tab). You should see this array:

5. Fill in the remaining three stencil methods, buildCheckerArray, buildTwoColorArray, and buildDiagonalStripeArray.

  ○ In each method:
    i. **Create an array of Colors** of the dimensions specified in the method comments.
    ii. Next, loop through the array you’ve created, figuring out the appropriate Color at each index and storing it in the array.
    iii. Finally, return the array you’ve created.

**Expected Results:**

Checkerboard  Two-color  Diagonal
6. Run the program and test each method by clicking the corresponding button in the GUI, which will call your method and visualize the array of Colors it returns. The images above show what each array should look like—when your arrays match them, move on to the next part of the lab.

Check Point 1: Call over a TA to check your arrays!

Additional Uses for Arrays

Recall how we assign an initial value to variables of type `int` using the following syntax:

```java
int x = 3;
```

We can also store `ints` as static in the Constant class using the syntax:

```java
public static final int X = 3;
```

Arrays are also able to take on an initial value! The following lines of code are both valid:

```java
int[] myArray1 = {1, 2, 3};
int[][] myArray2 = {{4,5,7}, {9,0,3}, {2,6,9}};
```

Likewise, we can store arrays as constants in the `Constants` class:

```java
public static final int[][] MY_FAVORITE_COORDINATES = {{3,7},{9,1}};
```

This has applications in programs where you want to store something as a constant that can’t be expressed as just one integer (for example: coordinates to build a shape or positions in a 2D space).

**ArrayLists**

Arrays are handy when you know exactly how many elements you’re going to be dealing with. But what if you want to model a collection of objects whose size may change?

Luckily, the core Java library provides several implementations of “collections” of objects, whose size may change as elements are added and removed. One such implementation is the class `java.util.ArrayList`. Like an array, an `ArrayList` stores elements at “indices”, and allows you to access and modify the element stored at a particular index. However, an `ArrayList` provides convenient methods for adding, removing, and modifying elements, and changes size automatically as elements are added and removed. We’ll give a quick refresher on `ArrayLists` below—check out the lecture slides for the full scoop.

When declaring an `ArrayList`, we need to specify what type of object it stores. We would declare and instantiate an `ArrayList` of `Aliens` like this:

```java
ArrayList<Alien> aliens = new ArrayList<Alien>();
```
We put the type of object our ArrayList will store within the angle brackets. “ArrayList<Alien>” just means “an ArrayList of Aliens”. When we initialize an array, it takes on the size we tell it to -- but when we initialize an ArrayList, we don’t give it a size. Every ArrayList starts out empty (size 0). To add an element to the ArrayList, call the add method. Let’s add a few Aliens to our ArrayList:

```java
aliens.add(new PurpleAlien());
aliens.add(new GreenAlien());
aliens.add(new BlueAlien());
```

To access the element at a specific index in the ArrayList, use the get method. For example, to tell the second alien in the list to do something, we would write:

```java
aliens.get(1).doSomething();
```

**Note:** This is different from the array syntax, where we would write:

```java
arrayName[1].doSomething();
```

To remove the element at a specific index in the ArrayList, use the remove method. To remove the first Alien in the list, we would write:

```java
aliens.remove(0);
```

Note that since we removed the purple alien, the green alien is now the first in the list, meaning it is at index 0. The blue alien is now the second in the list, meaning it is at index 1.

To replace the element at a specific index in the list with another element, use the set method:

```java
aliens.set(1, new PurpleAlien());
```
Looping through an `ArrayList` is pretty similar to looping through an array. For example, the following code replaces every element in the list with a `PurpleAlien`:

```java
for (int i = 0; i < aliens.size(); i++) {
    aliens.set(i, new PurpleAlien());
}
```

To get the number of elements in your `ArrayList`, use the `size` method.

```java
aliens.size();
```

To remove all elements from an `ArrayList`, use the `clear` method:

```java
aliens.clear();
```

Check out the `ArrayList` Javadocs for more useful methods!

---

**Practice with ArrayLists**

You're going to fill in a series of methods, each of which performs a particular operation on an `ArrayList<Color>` called `_colors`. Again, you'll first write code to visualize the `ArrayList`!

1. Open up the file `ArrayListBuilder.java`. The `ArrayListBuilder` class contains several stencil methods, each of which should perform an operation on the `ArrayList<Color>` instance variable called `_colors`.

2. Again you will need to fill in the method `displayArrayList` before anything will show up on the screen. Clear the `parentPane` and add all the elements of `rects` to the `parentPane`. (HINT: Does `ArrayList` have any methods which would save you from needing to loop through every index?) [ArrayList Javadocs](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html)

3. Run the app and navigate to the “ArrayLists” tab. Each of the buttons at the bottom of the interface calls the corresponding method in the stencil code. Play around with the “Add Pink” and “Clear” buttons (we’ve filled in these methods for you-- the other buttons won’t do anything until you fill in their methods)! If you’ve implemented `displayArrayList` correctly, you should see the visualization of `_colors` update as elements are added and removed.

4. Fill in the remaining stencil methods according to the descriptions in the method
comments. You can reference the Javadocs for a full description of the `ArrayList`'s methods. Run the app and play around to test your work.

**Check Point 2:** Call over a TA to check you off!