Image Credit: Julius Schorzman

Java Lesson 9

Concept Review

Sorting an Array

- In Java, there are a variety of ways that you can sort information. One of the most popular methods is to use the `Arrays` class, as shown below:

```
import java.util.Arrays;

public class DemoArrays
{
   public static void main(String[] args)
   {
      int[] scores = {7, 10, 8, 5, 3, 9};

      System.out.println("The sorted scores are: ");
      Arrays.sort(scores);     // Does the sorting.

      for (int x: scores)
         System.out.print(x + " ");
   }
}
```

Imports the Arrays class into the program.

The sort() method performs the sorting.

Sorting an Array

- Sometimes, you might encounter a situation where you want to tell Java exactly how to sort your information. Consider the following example with the product class. We'll use a bubble sort to sort the various grocery products by their price.

```
public class Product
{
   private String name;
   private double price;

   // Constructor that receives the product name and price.
   public Product(String productName, double thePrice)
   {
      name = productName;
      price = thePrice;
   }

   // Get methods
   public String getName()
   {
      return name;
   }

   public double getPrice()
   {
      return price;
   }
}
```

Receives the grocery product's name and price.

Sorting an Array

- To demonstrate the product class, we'll create an array that holds five grocery products, as you can see here:

```
Product[] groceries = new Product[5];

groceries[0] = new Product("Carrots", 1.25);
groceries[1] = new Product("Apples", 2.15);
groceries[2] = new Product("Pineapple", 4.55);
groceries[3] = new Product("Celery", 1.75);
groceries[4] = new Product("Figs", 3.15);
```

Sorting an Array

- To sort the various grocery products, we'll need to create a bubble sort.

```
Product temp;
for (int x=0; x < groceries.length - 1; ++x)
    for (int y=0; y < groceries.length - 1; ++y)
    {
        if (groceries[y].getPrice() > groceries[y+1].getPrice())
        {
            temp = groceries[y];
            groceries[y] = groceries[y+1];
            groceries[y+1] = temp;
        }
    }
```

Loop Body

The two for() loops!

- Notice the two for() loops above? This helps to ensure that all of the items in the loops are sorted, as the sorting is performed in the loop body.

Sorting an Array

- When the program is run, the results of the sorting is the following:

Original

```
Product[] groceries = new Product[5];

groceries[0] = new Product("Carrots", 1.25);
groceries[1] = new Product("Apples", 2.15);
groceries[2] = new Product("Pineapple", 4.55);
groceries[3] = new Product("Celery", 1.75);
groceries[4] = new Product("Figs", 3.15);
```

Sorted Result

```
Carrots costs 1.25
Celery costs 1.75
Apples costs 2.15
Figs costs 3.15
Pineapple costs 4.55
```

An ArrayList

- The ArrayList is a powerful version of the array that is resizable while the program is running. Consider the following ShoppingList example:

```
ArrayList list = new ArrayList();

// Add some items.
list.add("Bread");
list.add("Milk");
list.add("Peanut Butter");
list.add("Salad");
list.add("Cookies");
```

Creates the ArrayList, so we can add() items to it!

An ArrayList

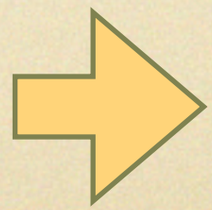
- Since ArrayLists are dynamic, you can easily remove() items too, as shown below:

```
// Remove some items.
list.remove("Milk");
list.remove("Cookies");
```

Removes these items from the ArrayList.

- To display the items in the list, you can use the size() and get() methods. The size() method returns how many elements are currently in the array, where as the get() method returns the specific, desired element.

```
// Display the items in the list
for (int x=0; x < list.size(); ++x)
   System.out.println(list.get(x));
```



```
Bread
Peanut Butter
Salad
```