CMSC 206: Data Structures
Lab 4: Exceptions, `ArrayList`

This lab starts out with a review of exceptions and then continues to introduce `ArrayList`, a flexibly sized structure for ordered data.

As with all labs, complete this work in a fresh Eclipse project. Unlike other labs, **you will submit** this on Gradescope. It will not be graded, but it may be helpful for me to see what you're doing.

Part I: Exceptions

This exercise will give you further practice in throwing, catching, and testing exceptions. The exercise is structures around a square-root method that works only on integers (and thus can't work with, say, 8 or -4).

1. Using `ZipParseException` as a template, make a `SquareRootException` exception class. Have your constructor take an `int` which is the number that has no square root. Note that `ZipParseException`'s constructor calls a superclass constructor with a helpful message for users; yours should do the same.

   Note: Unlike `ZipParseException`'s constructor, yours will take only *one* parameter.

2. Write a `SquareRoot` class with the following method:

```
/** Takes the positive square root of an integer.
 *  @param n The integer to be rooted
 *  @return The square root of the parameter
 *  @throws SquareRootException when the parameter has no
 *                              integral square root
 */
public static int squareRoot(int n) throws SquareRootException
```

   You will find the `sqrt` method in the `Math` class in the `java.lang` package helpful; look it up in the Java online documentation. Note that `sqrt` works with `double`s, so you will have to do some work to make it work with `int`s.

3. Write a `main` method that allows you to interact with your `squareRoot` method. It should print a helpful message to the user if their number does not have a square root. Do *not* add a `throws` clause to your `main` method.

4. Create a JUnit test class `SquareRootTest` to test your method:

   a. One test method should test successful uses of `squareRoot` (by taking, say, the square root of numbers like 9 and 16). This test should fail if `squareRoot` throws an exception.

b. One test method should test what happens when you call `squareRoot` on a positive number that has no integral square root. It should succeed if `squareRoot` throws a `SquareRootException` and fail otherwise.
c. One test method should test what happens when you call `squareRoot` on a negative number. It should succeed if `squareRoot` throws a `SquareRootException` and fail otherwise.
d. One test method should test what happens when you call `squareRoot` on 0. By reading the specification above, you should figure out what the correct behavior should be.

Part II: Equality

This short interlude refreshes us on the difference between == and `equals` in Java.

5. Download the `Fraction` class posted next to yesterday's class and put it in your project.

6. In a new class `Equals`, write this `main` method:

```
public static void main(String[] args)
{
     Fraction f1 = new Fraction(3,4);
     Fraction f2 = new Fraction(3,4);
     Fraction f3 = new Fraction(6,8);
     Fraction f4 = f1;

     System.out.println("f1 == f2: " + (f1 == f2));
     System.out.println("f1 == f3: " + (f1 == f3));
     System.out.println("f1 == f4: " + (f1 == f4));
     System.out.println("f1.equals(f2): " + f1.equals(f2));
     System.out.println("f1.equals(f3): " + f1.equals(f3));
     System.out.println("f1.equals(f4): " + f1.equals(f4));
}
```

7. What behavior do you observe while running this program? Why do you think this happens? Write your answer in comments in the file. Test your hypotheses by writing more experiments.

Part II: Using `ArrayList`

8. Make a new class `ArrayListPractice`. Write this method in it:

```
/** Gets the prime factors of a number, without duplicates.
 *  @param n The number to factor
 *  @return An ArrayList of the prime factors. If the number
 *          has no prime factors (because it is less than 2),
 *          this returns an empty (but non-null) ArrayList.
 */
public static ArrayList<Integer> primeFactors(int n)
```

You should adapt the code you wrote for the first assignment to complete this. Yes, it's perfectly allowed to show this code to your partner, given that the first assignment is in the distant past.

Our book, sections 2.1-2.2, contains examples of using an `ArrayList`. It's also introduced on many blog posts, etc. One such post is at http://javarevisited.blogspot.com/2011/05/example-of-arraylist-in-java-tutorial.html. The writing isn't the best, but the Java is accurate.

Note that the return value is an `ArrayList<Integer>`, not an `ArrayList<int>`. There is a difference between these (for example, the latter is illegal in Java), as we'll discuss in class, but this difference does not matter in implementing this method. You can pretend that the elements in the `ArrayList` are all plain old `int`s, and everything should work.

9. Write a JUnit test class `ArrayListTest` that tests your method until you're confident that it works.

10. Lookup the documentation for `ArrayList` on the Java API web site. `ArrayList` is in the `java.util` package. Answer the following questions in comments in your *ArrayListPractice.java* file:

    a. One of the `ArrayList` constructors takes an `int` parameter. What does this constructor do? If I say
       ```
       ArrayList<String> list = new ArrayList<>(5);
       ```
       what will `list.size()` be? (Test your answer!)
    b. What exception is thrown if you try to `get` an element that's not in the `ArrayList`? (Test your answer!)
    c. One method on an `ArrayList` searches through the `ArrayList` for a specific value, returning `true` if the value is present and `false` otherwise. Which method is it?

11. The `ArrayList` method `indexOf` searches through an `ArrayList` for a value and returns the index of that value, or -1 if the value is not present. (This is *not* the answer to question (c) above.) How does it work to compare objects? Specifically, we've seen that the Java operator == and the method `equals` work differently. Which one does `indexOf` use? While the documentation answers this question (read it!), design an experiment (a `main` method in `ArrayListPractice` similar to the one in step 6) that demonstrates this choice. Include comments explaining how to interpret the results of your experiment. You may find it useful to have an `ArrayList` of `Fraction`s.

Part IV: Implementing `ArrayList`

12. Download the *KWArrayList.java* and *KWArrayListRaw.java* files from the syllabus page. These files are taken from the code written in section 2.3 of our textbook. The *KWArrayList.java* files includes an implementation of a *generic* list structure. Generic types have a *type parameter* (in this case, `E`) that allows the type to be specialized when you use it. When reading the code, treat `E` as an unknown reference type to be determined later. The book has more information about this, and we will discuss it in class.

13. Read through the code. In a new class `KWALExercises`, answer the following questions in comments:

    a. Why does the second `add` method have a loop?
    b. Why does that loop count backwards? Is it possible to rewrite this so that it counts up, instead of down?
    c. Why does the `remove` method have a loop?
    d. Why does that loop count up? Is it possible to rewrite this so that it counts down, instead of up?

14. The *KWArrayListRaw.java* file contains a non-generic list structure, which may feel more familiar. In it, `E` is replaced with `Object`. Otherwise, the files are the same. It has a `reallocate` method that makes the `theData` array double in size, using the method `copyOf`, a static method in the `java.util.Arrays` class. Rewrite `reallocate` not to call `copyOf` but instead to grow the array manually. Note that there is no way of directly growing an array. Instead, you will make a new array (with `new Object[…]`) and copy all the elements from the old array into the new one, using a loop. (This is exactly what `copyOf` does under the hood.)

15. Test your implementation, either using a JUnit test class or with a `main` method.

16. Submit your project on Gradescope.