Lab 2

Two (three) goals –

- identifying a set of test cases to thoroughly test code
  - what are the different behaviors? what aspects of the input lead to different results?
  - anticipate typical bugs
  - cover special cases – a common bug is not handling one or more correctly
- seeing a framework for implementing a tester
  - if it is easy to run tests, it is easy to do it each time the code is modified (fixing bugs, adding features) to verify that nothing broke
- seeing an example of how you might organize a program to facilitate testing
  - for Rock Paper Scissors, pull key pieces of the functionality – handling one round and handling the whole game – into methods

```java
public class SortTester {

  public static void main ( String[] args ) {
    // demo of running a test case
    runTestCase("demo",10,20,30,"10 20 30");
  }

  /**
   * Run a test case for sort3.
   *
   * @param testcase
   *          name of test case
   * @param a
   *          input to sort3 (a)
   * @param b
   *          input to sort3 (b)
   * @param c
   *          input to sort3 (c)
   * @param expected
   *          expected result (string printed)
   */
  public static void runTestCase ( String testcase, int a, int b, int c,
                                   String expected ) {
    System.out.println(" -- " + testcase);

    System.out.print("        got: ");
    Sort3Variant1.sort3(a,b,c);
    System.out.println("   expected: " + expected);
    System.out.println();
  }

}
```

it should be as easy as possible to tell what isn't working

when the thing being tested produces output, the best we can do is print the output generated and the expected output so it is easy to compare and see if they match

also print the name of the test case so it is easy to tell what failed

since this is done for every test case, define a method which takes the necessary info about the test case (name, input, expected result), runs it, and prints info for inspection

Lab 2

```java
public class TimeTester {

  public static void main ( String[] args ) {
    // demo of running a test case
    runTestCase("demo","0815Z","EDT","4:15 AM (EDT)");
  }

  public static void runTestCase ( String name, String zulu, String timezone,
                                   String result ) {
    System.out.println(" -- " + name);

    Time time = new TimeVariant1(zulu);
    String converted = time.convert(timezone);

    System.out.print(" got: " + converted + " / expected: " + result + " ... ");

    if ( result.equals(converted) ) {
      System.out.println("passed!");
    } else {
      System.out.println("failed!");
    }
  }
}
```

when the method being tested returns a value, that value can be checked against the correct answer – printing "passed" or "failed" makes it even easier to tell which tests worked and which didn't

Lab 2

- for Rock Paper Scissors, choose the run test case templates that match the methods being tested
  - do `getRoundWinner` and `getGameWinner` print or return?

```java
public class RPSTester {

  public static void main ( String[] args ) {}

}
```

- test cases should focus on each method's own job

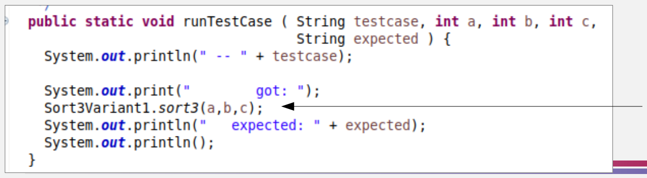
Based on the API description above, identify test cases for both `getRoundWinner` and `getGameWinner`. One wrinkle is that these methods aren't independent — `getGameWinner` uses `getRoundWinner` for each round within the game. Identify test cases for `getGameWinner` assuming that `getRoundWinner` works — that is, focus just on what could go wrong with `getGameWinner` itself.

Lab 2

Other variants –

- your goal is to create a thorough tester than can find any bug, not just the one(s) in variant 1
- trying implementations with different (or no) bugs helps you test your tester

The `lab2-variants.jar` file contains three additional implementations of `sort3` with different bugs (or perhaps no bugs at all). To try your tester with these variants, locate the line with `Sort3Variant1.sort3` in `runTestCase` and change it to `Sort3Variant2`, `Sort3Variant3`, or `Sort3Variant4`. You do not need to report on bugs found or not found in these variants, but there's at most one correct implementation provided so if all of your tests pass for several variants, you're missing some test cases...



```
public static void runTestCase ( String testcase, int a, int b, int c,
                                 String expected ) {
  System.out.println(" -- " + testcase);

  System.out.print("        got: ");
  Sort3Variant1.sort3(a,b,c);
  System.out.println("   expected: " + expected);
  System.out.println();
}
```