COMP1406 – Winter 2019

Submit a single file called `assignment3.zip` to the submission server
`http://134.117.31.149:9091/`

Your zip file must contain a directory called `comp1406a3` and all of your `.java` files must be in this directory. Do not include your `.class` files.

This assignment has 50 marks.
All marks are based on your test cases finding bugs in code that has bugs.

1 FutureDate [10 marks]

In this problem you will implement a single static method that outputs an array of black-box test cases for the `futureDate()` method from Tutorial 5. The method you will implement, in the `TestFutureDate` class, is specified as follows:

```
/**  Generate black-box test cases for the futureDate() method.
  *  The API for the futureDate() method is provided in this Assignment document.
  *
  *  @return an array of test cases for the futureDate() method.
  */
  public static FutureDateTestCase[] makeTestCases(){...}
```

Each test case consists of a `Date` object, an integer (that is the number of days in the future), and another `Date` object that is the expected date in the future. For example, consider

```
Date day = new Date(2019, 1, 1);
int  daysInFuture = 2;
Date expected = new Date(2019, 1, 1+daysInFuture);
FutureDateTestCase testcase = new FutureDateTestCase(day, daysInFuture, expected);
```

This creates a test case to check that the expected Date is 2 days in the future from the day object. That is, it checks that January 3, 2019 is 2 days in the future from January 1, 2019. This is a **valid** test case because the expected output is correct.

Each of your test cases will first be run with a correct implementation of the `futureDate()` method. If the *actual* output matches the *expected* output (specified in the test case) then that test case is deemed to be `valid`. Each valid test case will then be run with several alternate versions of the method that have bugs in them. Test cases that are invalid will be discarded.

A bug is exposed each time a valid test case fails in an alternate version of the method. That is, when the outcome of the method does not match the expected outcome of the test case then something went wrong in the method (and hence a bug is exposed). A failed test case does not necessarily indicate what or where the bug is but it does let you know that a bug exists.

Your goal is to generate a collection of valid test cases that expose a bug in each of the alternate versions (buggy versions) that we will have on the server. Your grade will be determined by the number of alternate versions that you find a bug in.

The API for the `futureDate()` method is as follows:

```
/** Creates a new date object that is some specified number of days
  * in the future from date object calling it.
  *
  * Precondition: the current Date object calling this method will always correspond
  *         to a date that is NOT further in the past than August 26, 1735.
  *
  * @param daysInFuture is a non-negative integer (the number of days from
  *        this date that the new returned date object will be created).
  * @return a new Date object that is daysInFuture past this current date.
  */
  public Date futureDate(int daysInFuture){...}
```

2 Grades [20 marks]

In this problem you will implement a single static method that outputs an array of black-box test cases for two methods in the `Grades` class (`finalGrade()` and `canProgressTo2ndYear()`). As with Assignment 1, you must understand the marking scheme and rules as outlined in the course outline. The method you will implement, in the `TestGrades` class, is specified as follows:

```
/**  Generate black-box test cases for the Grades class.
  *  @return an array of test cases for the Grades class
  */
  public static GradesTestCase[] makeTestCases(){...}
```

Each test case consists of grades (assignments, midterms, tutorials, final exam) to make a `Grades` object and the expected output of both methods called from that object. For example, consider

```
      double[] a = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0};
      double[] m = {0.0, 0.0, 0.0};
      double[] t = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
      double   f = 0.0;
      GradesTestCase testcase = new GradesTestCase(a, m, t, f, "F", false);
```

This creates a valid test case. The first four parameters have all the actual grade information, the `"F"` is the expected output of the `finalGrade()` method, and `false` is the expected output of the `canProgressTo2ndYear()` method.

Each of your test cases will first be run with correct implementations of the given methods. If the output matches the expected output for **both** methods then that test case is deemed to be `valid`. Each valid test case will then be run with several alternate versions of the methods that have bugs in them. Test cases that are invalid will be discarded.

Your goal is to generate a collection of valid test cases that expose a bug in each of the alternate versions (buggy versions) that we will have on the server. Your grade will be determined by the number of alternate versions that you find a bug in.

The API for the `Grades` class is provided on the next page.

```
/** Creates a Grades object with the specified grades.
  *
  * @param assignments is a list of 9 assignments each in the range [0.0, 100.0].
  *       The first 6 are the normal assignments and the last 3 are the study assignments.
  * @param midterms is a list of 3 midterm grades each in the range [0.0, 100.0]
  * @param tutorials is a list of 10 tutorial grades each in the range [0.0, 100.0]
  * @param finalExam is the final exam grade in the range [0.0, 100.0]
  */
  public Grades(double[] assignments, double[] midterms,
                double[] tutorials, double finalExam){...}


/** Computes the letter grade based on the grades stored in this
  * object using the description given in the course outline and using
  * the letter grade table given in Assignment 1.
  *
  * @return the letter grade obtained with the grades in this object.
  */
  public String finalGrade(){...}


/** Decides if the current grades are sufficient for a student to
  * progress to the 2nd year programming courses in SCS. That is,
  * do these current grades allow the student register in any of
  * COMP2401, 2402, 2404 or 2406?
  *
  * @return true if the current grades are sufficient for a student to enroll in second
  *        year SCS coding course; returns false otherwise. The rule for this can be found
  *        in the course outline.
  */
  public boolean canProgressTo2ndYear(){...}
```

3 Find/FindAgain [20 marks]

In this problem, you will be creating black-box test cases for the static methods in the Find and FindAgain classes from Assignment 1. For this assignment, howver, both methods (locateSequence() and locateAllSequenceLocations()) will be in a modified Find class to simplify things. The API for the two methods is shown later in this document.

You will complete the makeTestCases() method in the TestFind class.

```
/** Generate test cases for the (modified) Find class.
  * @return an array of test cases for the (modified) Find class.
  */
  public static FindTestCase[] makeTestCases(){...}
```

Each of the FindTestCase objects is a test case for both methods. A test case will consist of a target sequence, an array and the expected output for both methods. A test case is **valid** if it is correct for **both** methods. All **valid** test cases will be used to try and expose bugs in various implementations of each of the methods. Test cases that are invalid will be discarded.

```
package comp1406a3;

public class Find{

  /** Finds the last occurrence of the target sequence in the array or
    * indicates that the target sequence is not present.
    *
    * @param target is an array of one or more integers.
    *        It is the target sequence we are looking for.
    *
    * @param array is an array of zero or more integers.
    *
    * @return the starting index position of the last occurrence of the
    *         target sequence in the array if it exists.
    *         Returns -1 otherwise.
    */
    public static int locateSequence(int[] target, int[] array){...}


  /** Finds ALL occurrences of the target sequence in the array and
    * returns an array indicating how many times the target sequence appeared
    * in the array and all of their starting position (in increasing order).
    *
    * @param target is an array of one or more integers.
    *        It is the target sequence we are looking for.
    *
    * @param array is an array of zero or more integers.
    *
    * @return an array of integers of size at least one. The first element
    *         in the array is the number of times the target sequence was
    *         found in the array. The next elements, if any, are the
    *         starting index positions of each occurrence of the
    *         target sequence in the array listed in increasing order.
    *         The returned array should be big enough to hold this data and not
    *         any larger.
    */
    public static int[] locateAllSequenceLocations(int[] target, int[] array){...}
```

Submission Recap

A complete assignment will consist of a single file (`assignment3.zip`) that has a single folder/directory called `comp1406a3`. The `comp1406a3` folder will have the following three files included:

`TestFutureDate.java`
`TestGrades.java`
`TestFind.java`

All classes must be in the `comp1406a3` package. That is, all files must have the `package comp1406a3;` directive as the first line. Your code will NOT compile if it does not have this and you will receive zero correctness marks if your code does not compile.