EECS2030Z Test 2

Version D

GETTING STARTED

1. Start eclipse; use the workspace suggested by eclipse (or remember the directory name of your workspace).
2. Import the test project by doing the following:
    1. Under the **File** menu choose **Import...**
    2. Under **General** choose **Existing Projects into Workspace** and press **Next**
    3. Click the **Select archive file** radio button, and click the **Browse...** button.
    4. Select the file **test2D.zip** and click **OK**
        - If you do not see the file named **test2D.zip** in your directory, then open a terminal and copy and paste the following command:

          ```
          cp /eecs/dept/www/course/2030/labtest/test2D.zip .
          ```

          and re-import the project.
    5. Click **Finish**.
3. All of the files you need for this test should now appear in eclipse.
4. Open a terminal. You will use this terminal to submit your work.
5. Copy and paste the command `cd workspace/Test2D/src/test2` into the terminal and press enter.

Question 1 (20 marks total)

SOLUTION

Implement the utility class described by this API. You do not have to include javadoc comments.

`submit 2030 test2D Utility2D.java`

Question 2 (10 marks total)

SOLUTION

Implement the constructors for the class described by this API. Use constructor chaining where possible when implementing your constructors. You do not have to include javadoc comments in your code, but see Question 3C below.

`submit 2030 test2D Fraction.java`

Question 3 (20 marks total)

A.

State the definition of a method precondition.

A method precondition is a condition that the client must ensure is true immediately before a method is invoked.

Grading scheme: 3 marks total.

B.

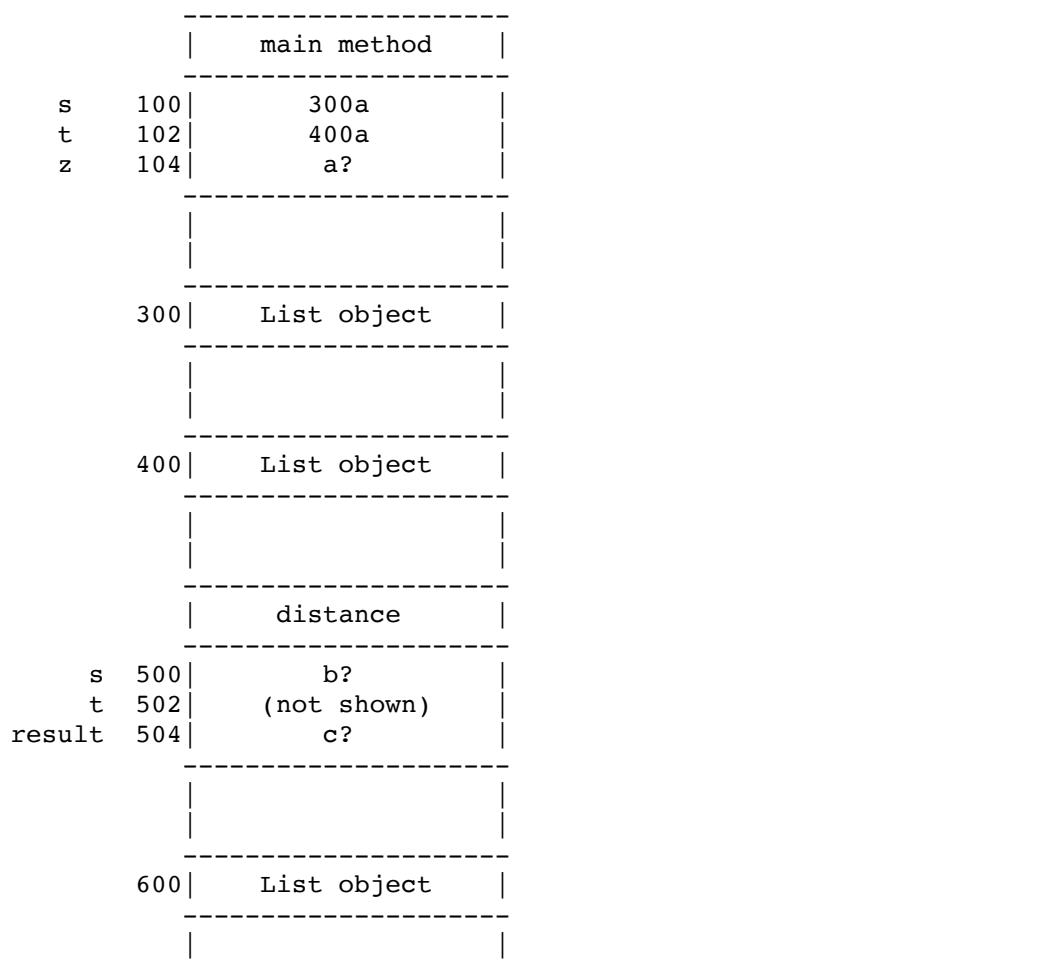
Suppose that the method name `zipper` from Question 1 was implemented like so:

```
public static List<Integer>(List<Integer> s, List<Integer> t) {
  List<Integer> result = new ArrayList<Integer>();
  /* code not shown here that assigns the correct value to result */
  return result;
}
```

Suppose a client writes a `main` method that includes the following two lines of Java code:

```
List<Integer> s = Arrays.asList(5, 7);    // the list [5, 7]
List<Integer> t = Arrays.asList(6, 8);    // the list [6, 8]
List<Integer> z = Utility2D.zipper(s, t);
```

The memory diagram illustrating the state of memory for the three lines of client code is shown below. What suitable values of a, b, and c would complete the memory diagram?



a = 600a, b = 300a, c = 600a

a and c are both equal to the address of the returned list. b is equal to the address of the first list passed to the method.

Grading scheme: 3 marks total (1 for each of a, b, and c).

C.

Provide the Javadoc necessary to *exactly* reproduce the API documentation for the constructor `Rotation(double angle, String units)` from Question 2.

```
/**
 * Initializes the rotation to the given angle using the given
 * units.
 *
 * @param angle the angle of the rotation
 * @param units the units of the angle
 * @throws IllegalArgumentException if units is not equal
 * to Rotation.DEGREES or Rotation.RADIANS
 */
```

D.

Provide 3 test cases for the method `Utility2D.interval`. Make sure that each test case tests a different feature of the method; try to include one boundary test case. For each test case, provide a one sentence explanation of what the test case is testing.

```
There are many possible test cases; below are 3 examples:

a                    : 1
b                    : 1
expected return value: [1]
explanation          : boundary case where a == b

a                    : 1
b                    : 5
expected return value: [1, 2, 3, 4, 5]
explanation          : tests a typical case where a < b

a                    : 5
b                    : 1
expected return value: [1, 2, 3, 4, 5]
explanation          : tests a typical case where b < a
```

E.

Consider the following class and its implementation of `equals(Object)`:

```
public class Bool {

  private boolean value;
  private String name;
```

```
  @Override
  public boolean equals(Object obj) {
    if (this == obj) {
      return true;
    }
    if (obj == null) {
      return false;
    }
    if (this.getClass() != obj.getClass()) {
      return false;
    }
    Bool other = (Bool) obj;
    if (this.name != other.name) {
      return false;
    }
    if (this.value && other.value == false) {
      return false;
    }
    return true;
  }

}
```

Explain whether or not the implementation shown above satisfies the equals contract. Does it contain any errors not related to the equals contract?

The implementation does not satisfy the equals contract because `x.equals(x) == true` is not always true! This occurs when `this.value == false` because `x.equals(x)` returns in the following `if` statement:

```
    if (this.value && other.value == false) {
      return false;
    }
```

Grading scheme: 4 marks total.

2 marks for stating `x.equals(x) == true` is not always true. 2 marks for the explanation.

`submit 2030 test2D answers.txt`
