WRITTEN TEST 1G

This is a 45 minute test. The test is closed book (no aids are allowed).

Written question instructions

- There are 8 short answer questions, each worth 2 marks.
- There are 3 questions that require students to explain their answers, each worth 6 marks.
- Answer the written questions in a text file named `answers.txt`—a suitable file should open in a text editor when the test starts. Feel free to use a different text editor if you wish.
- Make sure to save your work before submitting it! Every year a small number of students end up submitting an empty file because they forgot to save their work.
- You may submit your work as many times as you wish.
- A few minutes before the end of the test you will receive a warning that the test is ending soon. The message will repeat every minute until the end of the test. Use this time to submit your work; it is difficult to work effectively with the message popping up every minute.

Submission instructions

- Open a terminal (if one is not already open)
- Find the directory where you have saved your `answers.txt` file; the file should be in your home directory.
- Type the following command:

  ```
  submit 2030 test1G answers.txt
  ```

  and press enter.

1. [2 marks] Consider the following memory diagram:

variable name	address	value
x	100	-1.0
y	102	5.0
s	104	270a
t	106	250a
	250	`Point2` object
x	255	-1.0
y	257	5.0
	270	`Point2` object
x	275	0.0
y	277	0.0

Complete the 4 incomplete lines of Java code below that would produce the given memory diagram:

```
double x =
double y =
Point2 s = new Point2(x, y); // this line is already complete
Point2 t =
s =
```

2. [2 marks] Recall the `Point2` class from the lecture slides. How is the state represented for each `Point2` object?

3. [2 marks] Consider the following Java expression:

   ```
   (a + b) / c
   ```

   How does the Java compiler determine the type of the final value of the expression?

4. [2 marks] Consider the following class:

```
/**
 * A line segment starting at one point and ending
 * at a second point.
 */
public class LineSegment {

   private Point2 p; // the start point
   private Point2 q; // the end point

   public LineSegment() {
      Point2 p = new Point2(-1.0, 0.0);
      Point2 q = new Point2( 1.0, 0.0);
   }

   public Point2 getStartPoint() {
      return this.p;
   }
}
```

   When testing the class you discover that the starting point returned by `getStartPoint` is always `null`. What went wrong in the constructor of `LineSegment`?

5. [2 marks] Consider the following class that has the class invariant `this.ohms >= 0`:

```
public class Resistor {

    // the resistance of this resistor
    double ohms;

    public Resistor(double ohms) {
        if (this.ohms < 0) {
            throw new IllegalArgumentException();
        }
        this.ohms = ohms;
    }
}
```

What is wrong with the implementation of the constructor?

6. [2 marks] Consider the following class:

```
public class Point2 {

    private double x;
    private double y;

    public Point2(double x, double y) {
        // implementation not shown
    }

    public Point2(Point2 other) {
        // implementation not shown
    }
}
```

What would you write in the second constructor if you implemented the constructor using chaining?

7. [2 marks] In this course, we have four requirements for implementing an `equals` method in a class. In the `equals` method of `Point2` shown below, why is the cast required?

```
@Override
public boolean equals(Object obj) {
    // some if statements not shown
    if (this.getClass() != obj.getClass()) {
        return false;
    }
    Point2 other = (Point2) obj; // why is the cast required?
    // rest of method not shown
}
```

8. [2 marks] When must you override the `hashCode` method?

9. (a) [4 marks] Suppose that you want to implement a class that represents the time of day in hours and minutes (seconds are not necessary for the purposes of this question); also, it is important that your class can distinguish between times before noon (AM) and times from noon onwards (PM).

   What *primitive* type fields would you use to implement the class? In your answer, explain how many fields you would use, their types, what the fields represent, and what invariants (if any) each field has.

   (b) [2 marks] Answer this part only if you used more than one field in part (a) of this question. You will receive all 6 marks for this question if you used a single field in part (a) and if your answer is correct.

   Using only a single primitive type field, how would you implement the class? In your answer, explain how many fields you would use, their types, what the fields represent, and what invariants (if any) each field has.

10. [6 marks] A `RangedValue` object represents an integer value that is *guaranteed* to be within a range of values. For example, the statement

```
RangedValue v = new RangedValue(0, -3, 5);
```

makes a `RangedValue` object having a value of 0 and lying in the range of -3 to 5.

The methods `min` and `max` return the minimum and maximum values of a range:

```
RangedValue v = new RangedValue(0, -3, 5);
int lo = v.min();               // lo is -3
int hi = v.max();               // hi is  5
```

The method `value` returns the value:

```
RangedValue v = new RangedValue(0, -3, 5);
int val = v.value();              // val is 0
```

The method `fraction` returns the value expressed as a percentage (a value between 0.0 and 1.0) of the width of the range:

```
RangedValue v = new RangedValue(0, -3, 5);
double f = v.fraction();              // f is 0.375
```

Finally, the method `overlaps` returns true if this range overlaps with a second range:

```
RangedValue v = new RangedValue(0, -3, 5);
RangedValue w = new RangedValue(1, 2, 10);
boolean b = v.overlaps(w); // true because both ranges include 2, 3, 4, 5
```

[questions on next page]

(a) [2 marks] Suppose that `fraction` is implemented like so:

```
public double fraction() {
    double f = (0.0 + this.value() - this.min()) / (this.max() - this.min());
    return f;
}
```

Is there anything wrong with the implementation of `fraction`? If so, suggest a way to correct the error.

For parts (b) and (c) of this question, assume that the method `fraction` has been implemented correctly.

(b) [1 mark] Suppose that `RangedValue` implements the `Comparable` interface and that ranged values are compared using the `fraction` method:

```
@Override
public int compareTo(RangedValue other) {
        return Double.compare(this.fraction(), other.fraction());
}
```

Do the values returned by `fraction` have a natural ordering? In other words, is there a smallest fractional value and a largest fractional value and can you sort all of the fractional values in a consistent way? A yes or no answer is sufficient.

(c) [2 marks] Suppose that `RangedValue` implements the `Comparable` interface. Ranged values are compared using the `fraction` method if the ranges do not overlap. Ranged values are compared using the actual values if the ranges do overlap:

```
@Override
public int compareTo(RangedValue other) {
    if (this.overlaps(other)) {
        return Double.compare(this.fraction(), other.fraction());
    }
    else {
        return Double.compare(this.value(), other.value());
    }
}
```

This version of `compareTo` does *not* define a natural ordering of ranged values. For example, it is possible to find three ranged values `x`, `y`, and `z` where:

- `x.compareTo(y)` returns a positive integer, and
- `y.compareTo(z)` returns a positive integer, but
- `x.compareTo(z)` returns a negative integer or zero

State such an example of three ranged values that show that the given implementation of `compareTo` does not define a natural ordering of ranged values.

11. (a) [3 marks] The `Nickel` class has an `equals` method that has documentation that says "a nickel is equal to all other nickels". A student implements the `equals` method in the `Nickel` class as follows:

```
@Override
public boolean equals(Nickel obj) {
    return true;
}
```

State what errors, if any, are in the implementation. In your answer, try to refer to the four requirements we have for the `equals` method in this course.

(b) [3 marks] The `Nickel` class has a `hashCode` method that has documentation that says the method "returns the issue year of the nickel". A student implements the `hashCode` method in the `Nickel` class as follows:

```
@Override
public int hashCode() {
    return this.year;
}
```

The implementation correctly does what the documentation says it should, but `hashCode` *should not* be implemented this way for `Nickel`. Explain why.