WRITTEN TEST 1H

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 test1H answers.txt
  ```

  and press enter.

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

variable name	address	value
x	32	100
s	36	100a
t	38	100a
u	40	120a
	100	`Point2` object
	120	`Point2` object

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

```
int x =
Point2 s =
Point2 t =
Point2 u =
```

The actual coordinates of the points that you create are unimportant and are not shown in the memory diagram.

2. [2 marks] What is the definition of the term *state of an object*?

3. [2 marks] What is the purpose of a no-argument constructor?

4. [2 marks] Consider the following class:

```
/**
 * A line segment starting at one point and ending
 * at a second point. A line segment has a length
 * (the distance between the two points) that is
 * always greater than zero.
 */
public class LineSegment {

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

   public LineSegment(Point2 startPoint, Point2 endPoint) {
      // HOW DO YOU VALIDATE startPoint and endPoint?

      this.p = startPoint;
      this.q = endPoint;
   }

   // methods not shown
}
```

The class invariant for `LineSegment` is that "A line segment has a length (the distance between the two points) that is always greater than zero." How do you validate the constructor parameters `startPoint` and `endPoint` so that the class invariant is ensured?

5. [2 marks] What is the definition of the term *method postcondition*?

6. [2 marks] In a well designed class, a copy constructor never needs to validate its parameter. Use one sentence to explain why this true.

7. [2 marks] In this course, we have four requirements for implementing an `equals` method in a class. In the `equals` method below, which requirement does the `if` statement satisfy?

```
@Override
public boolean equals(Object obj) {
    // some if statements not shown

    if (obj == null) {
        // not shown
    }
    // rest of method not shown
}
```

8. [2 marks] Suppose that you have a class that implements the `Comparable` interface and that you have two references `x` and `y` to objects of your class' type. What does it mean if `x.compareTo(y)` returns a negative integer?

9. The winning numbers for a Lotto 6/49 lottery draw consist of seven integer values and all seven values are unique (no repeated values). All of the values are between 1 and 49, inclusive. The first six values are sorted from smallest to largest. The seventh value is a bonus number.

   Suppose that you wanted to create a class that represents a Lotto 6/49 lottery draw. 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.

   You do not need to express the invariants as a mathematical condition (although you may if you wish).

10. [6 marks] An almost complete implementation of the `Nickel` class from Lab 2 is shown below:

```
public class Nickel implements Comparable<Nickel> {

	private int year; // CLASS INVARIANT: year >= 1858

	/**
	 * Initializes this nickel to have the specified issue year.
	 *
	 * @param year the year this coin was issued in
	 * @throws IllegalArgumentException if year is less than 1858
	 */
	public Nickel(int year) {
		this.year = year;
		if (year <= 1858) {
			throw new IllegalArgumentException();
		}
	}

	/**
	 * Compares this nickel to another nickel by their issue year. The result is a
	 * negative integer if this nickel has an earlier issue year than the other
	 * nickel, a positive integer if this nickel has a later issue year than the the
	 * other nickel, and zero otherwise.
	 *
	 * @return a negative integer, zero, or a positive integer
	 *
	 */
	@Override
	public int compareTo(Nickel other) {
		return other.year - this.year;
	}

	/**
	 * Compares this nickel to the specified object for equality. The result is true
	 * if obj is a nickel. The issue year is not considered when comparing two
	 * nickels for equality.
	 *
	 * @return true if obj is a nickel
	 */
	@Override
	public boolean equals(Object obj) {
		// implementation not shown
	}
}
```

(a) [1 mark] Is the constructor implemented correctly? Explain why or why not.

(b) [2 marks] Is the implementation of `compareTo` safe from `int` overflow? Explain why or why not.

(c) [2 marks] Is `compareTo` consistent with `equals`? Explain why or why not.

(d) [1 mark] Is `compareTo` correctly implemented (ignoring the possibility of overflow)? Explain why or why not.

11. A `Range` object represents a range of integer values. For example, the statement

```
Range r = new Range(-3, 5);
```

makes a `Range` object having a minimum value of -3 and a maximum value of 5; i.e., the object represents the range of integer values `-3, -2, -1, 0, 1, 2, 3, 4, 5`. The class invariant for `Range` is that the minimum value of the range is less than or equal to the maximum value of the range.

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

```
Range r = new Range(-3, 5);
int lo = r.min();               // lo is -3
int hi = r.max();               // hi is  5
```

Consider the following implementation of `equals` for `Range`:

```
/*
   Two ranges are equal if and only if they have the same
   minimum and maximum values.
*/
@Override
public boolean equals(Object obj) {
   if (this == obj) {
      return false;
   }
   if (this.getClass() != obj.getClass()) {
      return false;
   }
   Range other = (Range) obj;
   if (this.min() != other.min() &&
         this.max() != other.max()) {
      return false;
   }
   return true;
}
```

where the method `min` returns the minimum value of a range and the method `max` returns the maximum value of a range.

(a) [2 marks] State what errors, if any, are in the implementation.

(b) [4 marks] Are all parts of the `equals` contract provided by the implementation? If your answer is yes, explain how the implementation provides each item of the `equals` contract. If your answer is no, state which parts of the `equals` contract are not supported and explain why those parts of the contract are not supported.