WRITTEN TEST 1F

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 test1F 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	250a
t	106	270a
u	108	270a
	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:

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

Solution:

```java
double x = -1.0;
double y = 5.0;
```

```
Point2 s = new Point2(x, y); // this line is already complete
Point2 t = new Point2(); // or new Point2(0.0, 0.0);
Point2 u = t; // NOT new Point2();
```

2. [2 marks] The three main elements of a class are *constructors*, *methods*, and what?

Solution: fields

3. [2 marks] In the following line of Java code:

```
String s = "hello";
```

what value does the variable `s` store?

Solution: The address of the string ”hello”

4. [2 marks] If a class defines no constructors (public or otherwise) how many constructors does the class end up with?

Solution: One (the compiler generates an empty no-argument constructor).

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

What is wrong with the implementation of `setResistance`?

Solution: The class invariant is not true after the method finishes running. Also acceptable: The method should perform the validation before assigning the field a value.

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?

Solution: this(other.x, other.y);

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

```
@Override
public boolean equals(Object obj) {
        // some if statements not shown
    if (this.getClass() != obj.getClass()) {
        return false;
    }
    // rest of method not shown
}
```

Solution: Only two objects of the same type can be equal

8. [2 marks] If two objects of the same type are *not* equal can they return the same value for `hashCode`?

Solution: Yes.

9. (a) [2 marks] Most lights are controlled by turning them either on or off. Suppose that you wanted to implement a class that represents such a light. In particular, the class needs methods that turn the light on and off.

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.

Solution: The simplest solution is to use one `boolean` field to represent the state of the light, but any primitive could be used. Two possible solutions are given below:

number	type	represents	invariants
1	boolean	on (true) or off (false)	no invariants
1	int	on (1) or off (0)	must be equal to 0 or 1

(b) [4 marks] A tri-light can be off, on with low brightness, on with medium brightness, or on with full brightness. Suppose that you wanted to implement a class that represents such a light. In particular, the class needs methods that can turn the light on to each level of brightness and a method that turns the light off.

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.

Solution: One `int` field makes the most sense here, but any primitive type could be made to work. One possible solution is given below:

number	type	represents	invariants
1	int	off (0), low (1), medium (2), high (3)	must be equal to 0, 1, 2, or 3

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.

> **Solution:** No. The class invariant may not be true after the method finishes running. Also acceptable: The method should perform the validation before assigning the field a value.

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

> **Solution:** Yes. The class invariant ensures that the difference is always in the range of `int` (i.e., both `(Integer.MAX_VALUE - 1858)` and `(1858 - Integer.MAX_VALUE)` are valid `int` values)

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

> **Solution:** No. All nickels are equal (i.e., `x.equals(y)` returns `true` for all nickels) but `x.compareTo(y)` returns 0 only for nickels having equal issue years.

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

> **Solution:** No. The sign of the returned value is opposite to what it should be (the method should return `this.year - other.year`)

11. The game rock-paper-scissors is a game played between two people. Both players form a shape with their hand at the same time; the hand shapes are:

- a closed fist or *rock*
- a flat hand or *paper*
- a two finger V or *scissors*

The winner is determined by applying the following rules:

- rock beats scissors
- scissors beats paper
- paper beats rock
- if both hands are the same then the result is a draw (neither player wins)

Suppose that you have the following class that represents the hand shapes for the game rock-paper-scissors:

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

    private String shape;

    // precondition: shape is one of "rock", "paper", or "scissors"
    public Hand(String shape) {
        this.shape = shape;
    }

    public String getShape() {
        return new String(this.shape);
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (this.getClass() != obj.getClass()) {
            return false;
        }
        Hand other = (Hand) obj;
        if (this.getShape() == other.getShape()) {
            return true;
        }
        return false;
    }

    // compareTo and hashCode not shown
}
```

(a) [3 marks] In this course, we have four requirements for every `equals` method. The first three if statements in the `equals` method shown above ensure that three of the four requirements are satisfied. For each of the first three if statements in the method, use sentence to indicate what requirement each if statement satisfies.

Solution:

- the first if statement ensures that an object is equal to itself
- the second if statement ensures that an object is never equal to null
- the third if statement ensures that only objects of the same type can be equal

(b) [1 mark] There is one error in the given implementation of `equals`. What is the error and how should it be fixed? [There are technically no errors elsewhere in the class.]

Solution: `this.getShape() == other.getShape()` is never true because `getShape` always returns a reference to a new string. The solution is to use `equals` instead of == to compare the two strings.

(c) [1 mark] The class says that it implements the `Comparable` interface. Should `compareTo` be implemented using the rules that determine which hand wins a round of rock-paper-scissors? Use one or two sentences to explain your answer.

Solution: No. There is no total ordering of hand shapes (every shape is "less than" some other shape and "greater than" some other shape).