WRITTEN TEST 1D

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 4 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 test1D answers.txt
  ```

  and press enter.

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

variable name	address	value
s	36	100a
t	38	120a
u	40	120a
v	42	120a
	100	`Point2` object
x	102	3.0
y	104	5.0
	120	`Point2` object
x	122	-2.0
y	124	-4.0

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

```
Point2 s = new Point2(-2.0, -4.0); // this line is already complete
Point2 t = new Point2(3.0, 5.0); // this line is already complete
Point2 u =
Point2 v =
s =
t =
```

Solution:

```
Point2 s = new Point2(-2.0, -4.0); // this line is already complete
Point2 t = new Point2(3.0, 5.0); // this line is already complete
Point2 u = s;
```

```
Point2 v = s; // or v = u;
s = t;
t = u; // or t = v;
```

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

Solution: The set of values of the fields of the object.

3. [2 marks] In an expression such as

```
double z = x * y;
```

how does the Java compiler determine if it should use integer multiplication or floating-point multiplication?

Solution: By looking at the types of x and y.

4. [2 marks] When making a new object (that is not an array) in Java you use the `new` operator. What comes immediately after the `new` operator?

Solution: A constructor call.

5. [2 marks] What is the definition of the term *class invariant*?

Solution: A condition that must be true immediately after every constructor and public method finishes.

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(Point other) {
        // implementation not shown
    }
}
```

What type of constructor is the second constructor?

Solution: copy constructor

7. [2 marks] The equals contract states that equals must be symmetric; what does symmetry mean for the equals method?

Solution: `x.equals(y)` must return the same value as `y.equals(x)`

8. [2 marks] How should you decide if a class that you are implementing should implement the `Comparable` interface?

Solution: You should implement the `Comparable` interface when it makes sense to say one object is smaller than, greater than, or equal to another object.

9. (a) [2 marks] In lecture halls at York University, the presenter can mute (turn off) and activate (turn on) the microphone. The muting and activating (and all the other audio-visual controls) is controlled via a small computer which means that someone wrote some software to control the microphone. Suppose that you had to write a class that represents whether the microphone is muted or active. In particular, the class needs methods that mutes the microphone and activates the microphone.

   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 microphone, 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] In lecture halls at York University, the presenter can set the microphone volume to one of twelve different volume levels (from silent to full volume). Suppose that you had to write a class that represents the volume level. In particular, the class needs methods that set the volume level to one of the twelve possible levels.

   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), from quietest to full volume (1–11)	must be in the range 0–11

10. [6 marks] 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 `int` 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
```

Suppose that you define a `compareTo` method for ranges using the following three rules:

1. `x.compareTo(y)` returns a negative integer if `x.min()` is less than `y.min()`
2. `x.compareTo(y)` returns a positive integer if `x.max()` is greater than `y.max()`
3. `x.compareTo(y)` returns zero if neither rule 1 nor rule 2 applies

When implementing `compareTo` the three rules shown above are applied in the order that they appear (i.e., first apply rule 1, then apply rule 2 if necessary, then apply rule 3 if necessary).

(a) [4 marks] Suppose that `equals` returns true if and only if two ranges have the same minimum and maximum values. Is `compareTo` consistent with `equals`?

Solution: No. Consider the two ranges `x = new Range(2, 3)` and `y = new Range(0, 5)`. Rules 1 and 2 do not apply; therefore, `x.compareTo(y)` returns 0 but `x.equals(y)` returns false.

(b) [2 marks] There is something wrong with defining `compareTo` for `Range` using the three rules above. Describe an example where the `Range` version of `compareTo` leads to a nonsensical result.

Solution: The rules do not define a total ordering of ranges. Consider the two ranges:

```
Range x = new Range(5, 6);
Range y = new Range(0, 10);
```

Then

`x.compareTo(y)` returns 0 (`x` "is equal to" `y`), but
`y.compareTo(x)` returns -1 (`y` "is less than" `x`)

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 one sentence to indicate what requirement is satisfied.

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) [2 marks] 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 to three 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).