WRITTEN TEST 1E

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 test1E 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:

```java
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:

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

2. [2 marks] What is the difference betweeen an *object* and a *reference*?

Solution: A reference is the memory address of an object.

3. [2 marks] What is the definition of the term *type* (in the context of the type of a variable)?

Solution: A type represents a set of values and the operations that can be performed using those values.

4. [2 marks] What is the purpose of the no-argument constructor?

Solution: To initialize the state of an object to some default state.

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 `setOhms`?

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 type of constructor is the second constructor?

Solution: copy constructor

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

Solution: An object is equal to itself

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

Solution: When `equals` has been overridden.

9. [6 marks] York University uses a letter grade to represent a course grade but many professors compute the grade using a percentage value (i.e., a floating-point value between 0.0 and 100.0). The percentage grade must be converted to a letter grade but there is no standard way of doing so. For simplicity, assume that there are only five letter grades: A, B, C, D, F. One professor might use the following conversion of percent grades to letter grades:

percent grade	letter grade
90 to 100	A
75 to 90	B
60 to 75	C
50 to 60	D
less than 50	F

Another professor might use the following conversion of percent grades to letter grades:

percent grade	letter grade
80 to 100	A
65 to 80	B
55 to 65	C
40 to 55	D
less than 40	F

For the purposes of this question, it is unimportant to what happens to a numeric grade exactly on the boundary of two letter grades.

Suppose that you had to write a class that lets a professor create an object to map percent grades to letter grades. The constructor allows the user to specify what range of percentage grades map to which letter grade. The class also has a method that maps a specifed percentage grade to a letter grade.

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 question says that the percent grades are floating-point values so either `float` or `double` fields are probably required. Four fields are required one each for the boundary between grades:

name	number	type	represents	invariants
ab	1	double	boundary between A and B	must be between 0.0 and 100.0
bc	1	double	boundary between B and C	must be between 0.0 and ab
cd	1	double	boundary between C and D	must be between 0.0 and bc
df	1	double	boundary between D and F	must be between 0.0 and cd

This particular solution allows one or more letter grades to have a percent range of 0.0 which is acceptable for this question.

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
```

[questions on next page]

(a) [2 marks] What class invariants should `RangedValue` have?

Solution: Two invariants are require:

1. the minimum value must be less than or equal to the maximum value of the range
2. the value must be between the minimum and maximum values of the range

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

```
public double fraction() {
   double f = (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.

Solution: The error is that `f` is computed using integer arithmetic which means that the quotient will always be an integer value and one or both differences might overflow. The solution is to make sure both differences are computed using floating-point arithmetic; for example:

```
double f = (0.0 + this.value() - this.min()) / (0.0 + this.max() - this.min());
```

(c) [2 marks] Suppose that `RangedValue` implements the `Comparable` interface and that ranged values are compared using the `fraction` method:

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

Suppose that `equals` is defined using the rule that two ranged values are equal if and only if their values are equal. Is `compareTo` consistent with `equals`? Explain your answer by using the conditions required for `compareTo` to be consistent with `equals`.

Solution: No. Consider the following two ranged values:

```
RangedValue x = new RangedValue(5, 0, 10);
RangedValue y = new RangedValue(50, 0, 100);
```

Both ranged values have a fractional value of 0.5; therefore, `x.compareTo(y)` returns 0. The ranged values are not equal because their values differ; therefore, `x.equals(y)` returns `false`. Because `compareTo` returns 0 when `equals` returns `false` we can conclude that `compareTo` is not consistent with `equals`.

11. (a) [1 mark] 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(Object obj) {
        if (obj == null) {
                return false;
        }
        if (this.getClass() != obj.getClass()) {
                return false;
        }
        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.

Solution: There are no errors.

(b) [2 marks] The `getClass` method used in `equals` returns a reference to a `Class` object. You should be aware that we normally use `equals` to compare two references for equality; however, the if statement:

```
if (this.getClass() != obj.getClass())
```

is guaranteed to work correctly. What can you conclude about the reference returned by `getClass`?

Solution: `getClass` always returns a reference to the same object for each class. In other words, for each class there is exactly one `Class` object and `getClass` returns a reference to this object.

(c) [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`. Give a correct (one line) implementation of `hashCode` for `Nickel`.

Solution: The requirement for `hashCode` is that it must return the same hash code for two equal objects. Because all nickels are equal, `hashCode` must return a constant value:

```java
@Override
public int hashCode() {
    return 1;
}
```