WRITTEN TEST 1B

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 test1B 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
	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 lines of Java code below that would produce the given memory diagram:

```
double x =
double y =
Point2 s =
Point2 t =
```

Solution:

```
double x = -1.0;
double y = 5.0;
Point2 s = new Point2(x, y); // or new Point2(-1.0, 5.0);
Point2 t = new Point2(); // or new Point2(0.0, 0.0);
```

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

Solution: The 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] Why do we use constructor chaining?

Solution: To reduce code duplication.

5. [2 marks] A constructor that has a precondition involving one or more of its parameters usually validates the input arguments to ensure that the precondition is true. Why is it important for the constructor to validate the input arguments if the constructor has a precondition?

Solution: To avoid initializing the object to an invalid state. Also acceptable: To ensure that the class invariants are true.

6. [2 marks] When does *shadowing* occur in a method?

Solution: When the method has a parameter that has the same name as a field.

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

Solution: if x.equals(y) and y.equals(z) then x.equals(z)

8. [2 marks] A student attempting to implement the `compareTo` method for the `Die` class in Lab 2 writes the following in their class:

```
@Override
public int compareTo(Die other) {
    if (this.value < other.value) {
        return -1; // this Die is smaller than the other Die
    }
    else if (this.value > other.value) {
        return 1; // this Die is greater than the other Die
    }
    else if (this.value == other.value) {
        return 0; // this Die is equal to the other Die
    }
}
```

Their eclipse editor indicates that there is a compilation error in their method. What is wrong with what the student has written?

Solution: The method does not return a value if none of the conditional statements in the `if` statements are true. Also acceptable: The last `else if` statement should simply be an `else` statement (or the `else if` statement can be entirely removed and the method can simply return 0).

9. [6 marks] A PIN for a bank card is a sequence of 4 to 8 digits where any digit is allowed to be a zero (for example, `0009` is a valid 4-digit PIN and `01020304` is a valid 8-digit PIN). Suppose that you would like to implement a class that represents a PIN.

   Because PINs are used to authenticate a user, it is important that your class has the following method:

   - `matches(PIN other)` : returns true if this PIN and the other PIN have the same sequeunce of digits

   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: At least two primitive type fields are required if the PIN is allowed to have leading zeros. Two possible solutions are given below:

name	number	type	represents	invariants
`nZeros`	1	int	number of leading zeros	value must be between 0 and 7
`digits`	1	int	digits of the PIN not including the leading zeros	`length(digits) + nZeros` must be between 4 and 8
`nDigits`	1	int	total number of digits in the PIN (including leading zeros)	value must be between 4 and 8
`digits`	1	int	digits of the PIN not including the leading zeros	`length(digits) <= nDigits`

`length(x)` is the number of digits in the `int` variable `x`

10. [6 marks] Consider the following class that represents a temperature in degrees Celcius or degrees Fahrenheit:

```
public class Temperature {
    private double degrees;
    private String units; // INVARIANT: this.units is equal to "C" or "F"

    public Temperature() {
        this.degrees = 0.0;
        this.units = "C";
    }

    /**
     * Changes the units of this temperature to the specified units
     * if the specified units is equal to "C" or "F", otherwise leaves
     * the current units of this temperature unchanged.
     *
     * @param the desired units of this temperature
     */
    public void setUnits(String units) {
        if (units == "C" || units == "F") {
            this.units = units;
        }
    }

    public void getUnits() {
        return this.units;
    }
}
```

(a) [4 marks] List the test cases that you would use to test the `setUnits` method (i.e., list the strings that you would use and the expected results for each string). For each test case, use *no more than one sentence* to explain why you chose the test case.

Solution: DO NOT MARK. QUESTION INVOLVES MATERIAL NOT COVERED BY THIS TEST.

(b) [2 marks] One of your test cases should reveal an error in the `setUnits` method; what is the error in the method?

Solution: DO NOT MARK. QUESTION INVOLVES MATERIAL NOT COVERED BY THIS TEST.

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

(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(0, 5)` and `y = new Range(3, 8)`. 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 three ranges:

```
Range x = new Range(0, 5);
Range y = new Range(3, 8);
Range z = new Range(6, 11);
```

Then

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

12. (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.

Solution: The main error is that the method has the wrong signature; `Nickel` should be `Object`. As for the four requirements, the method fails to satisfy the requirement that **an object is never equal to null** because it returns `true` if `obj` is null

(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.

Solution: `hashCode` must return the same value for two objects that are equal. Because a nickel is equal to every other nickel, the field `year` must not be used to compute the hash code.