WRITTEN TEST 1C

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 test1C 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:

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

2. [2 marks] The domain name for the EECS department at York University is `eecs.yorku.ca`—what is the recommended package name for Java code produced in the EECS department?

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?

4. [2 marks] Why do we use constructor chaining?

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

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
    }
```

The constructor has parameters with the same name as the fields; what term is used when such a situation occurs?

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
}
```

8. [2 marks] What information does the `java.lang.Object` version of `hashCode` use to compute a hash code for an object?

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.

   (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.

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) [4 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(Object 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.

(b) [2 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.