WRITTEN TEST 2H

This is a 60 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 8 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 test2H answers.txt
  ```

  and press enter.

1. [2 marks] What is a `public static final` field normally used for?

Solution: To represent a constant value.

2. [2 marks] Consider the following class that represents a line segment connecting two points (its start point and its end point):

```
public final class LineSegment {
    private final Point2 start;
    private final Point2 end;

    public LineSegment(Point2 p1, Point2 p2) {
        this.start = p1;
        this.end = p2;
    }

    // remainder of class not shown
}
```

Which statement best describes the class `LineSegment`? Give your answer as A, B, C, or D.

A. `LineSegment` is an aggregation of two points

B. `LineSegment` is a composition of two points

C. `LineSegment` is immutable

D. `LineSegment` is a superclass

Solution: A

3. [2 marks] How could you use a Java `Map` to emulate the functionality of a Java `List<String>`? (In other words, what would the keys of the map represent and what would the values of the map represent?) Hopefully, you would never actually do this.

Solution: The keys would be the indices and the values would be strings.

4. [2 marks] What is the definition of the term defensive copy?

Solution: A new copy of an object (created to prevent aliasing).

5. [2 marks] When a class has fields that are not of primitive type, we often need to consider using composition instead of aggregation. What is the main disadvantage of using composition instead of aggregation?

Solution: The memory and time needed to create the defensive copies.

6. [2 marks] What does the keyword `final` mean when it is used as a modifier on a method?

Solution: The method cannot be overridden.

7. [2 marks] Consider the following Java statement:

```
Counter c = new AscendingCounter();
```

Which class is the superclass and which class is the subclass? Write your answer like:

superclass:
subclass:

and complete each line with a class name.

Solution: superclass: Counter
subclass: AscendingCounter

8. [2 marks] What should the first line of a subclass constructor do?

Solution: The first line of a subclass constructor should call another constructor (often the superclass constructor).

9. Suppose that you have the following class (which is very similar to the class from Lab 3):

```
public final class Complex {

	private static final Complex I = new Complex(0.0, 1.0);

	private final double real;
	private final double imag;

	public Complex(double real, double imag) {
		this.real = real;
		this.imag = imag;
	}

	public static Complex i() {
		return Complex.I;
	}

	public static Complex one() {
		return new Complex(1.0, 0.0);
	}

	public static Complex add(Complex x, Complex y) {
		double re = x.real + y.real;
		double im = x.imag + y.imag;
		return new Complex(re, im);
	}
}
```

(a) [2 marks] The class `Complex` is an example of an immutable class. There are five rules in the recipe for immutability. State any *three* of those rules in the recipe for immutability that the class `Complex` uses to ensure immutability.

Solution: (2 marks for any three of the four rules below; 1 mark for any one or two of the four rules below)

1. do not provide any methods that can alter the state of the object (not mutator methods)
2. prevent the class from being extended (make the class final)
3. make all fields `final`
4. make all fields `private`

(b) [2 marks] Consider the following program:

```
public class Test {
	public static void main(String[] args) {
		Complex a = Complex.i();
		Complex b = Complex.i();
		System.out.println(a.equals(b));

		Complex x = Complex.one();
		Complex y = Complex.one();
		System.out.println(x.equals(y));
	}
}
```

What does the program print? Use one or two sentences to explain each line of output that the program produces.

Solution: The program prints:
`true`
`false`

`equals` is not overridden so `a.equals(b)` and `x.equals(y)` are equivalent to `a == b` and `x == y`. `a` and `b` are aliases for `Complex.I` so `a == b` is `true`. `x` and `y` are references to new objects so `x == y` is `false`.

(c) [2 marks] Consider the following program:

```
public class Test {
	public static void main(String[] args) {
		Complex x = Complex.i();
		Complex y = Complex.one();
		Complex z = Complex.add(x, y);

		// HERE
	}
}
```

On the line with the comment `HERE` how many `Complex` objects are in memory? Briefly describe your answer using one or two sentences.

Solution: 3.
`Complex.i()` is an alias for `I` (1 object), `Complex.one()` returns a new object (2 objects), and `Complex.add(x, y)` returns a new object (3 objects).

(d) [2 marks] In the `add` method, the method uses the non-static fields `real` and `imag` even though the method is static. Why is the method allowed to do so in this case?

Solution: Because the method uses the fields via the parameters `x` and `y` (which have type `Complex`). (The method cannot use the fields using `this` because no object is needed to call the method.)

(2 marks if the answer states that the method is allowed to access the fields of the object using the parameters `x` and `y`)

10. Consider the following class that represents a transcript of grades:

```
/**
 * A transcript of grades. The transcript owns all of the grades
 * in the transcript.
 */
public class Transcript {
    private List<CourseGrade> grades;

    public Transcript(List<CourseGrade> grades) {
        /* IMPLEMENTATION NOT SHOWN */
    }

    public int numCourses() {
        /* IMPLEMENTATION NOT SHOWN */
        /* RETURNS THE NUMBER OF COURSES IN THIS TRANSCRIPT */
    }

    public double getGPA() {
        /* IMPLEMENTATION NOT SHOWN */
        /* RETURNS THE GPA OVER ALL COURSES IN THIS TRANSCRIPT */
    }

    public CourseGrade getGrade(String course) {
        /* IMPLEMENTATION NOT SHOWN */
        /* RETURNS THE COURSEGRADE FOR THE SPECIFIED COURSE */
    }
}
```

Now consider the following fragment of code:

```
// course completion date
LocalDate completed = LocalDate.of(2018, Month.APRIL, 30);

// grade for EECS2030 with gpa of 7 completed Apr 30, 2018
CourseGrade c = new CourseGrade("EECS2030", 7, completed);

// a list with one CourseGrade
List<CourseGrade> grades = new ArrayList<>();
grades.add(c);

// a transcript
Transcript t = new Transcript(grades);

// remove all elements from grades
grades.clear();

// change the grade of c
c.setGrade(9);

// get the grade for EECS2030
CourseGrade g = t.getGrade("EECS2030");
```

(a) [2 marks] After running the code fragment above, what values should `t.numCourses()` and `t.getGPA()` return if the constructor of `Transcript` is implemented correctly?

> **Solution:** There is only one course with a grade of 7 in the transcript so:
> `t.numCourses()` should return `1`
> `t.getGPA()` should return `7`

(b) [2 marks] Suppose that after running the code fragment above, the statement `t.numCourses()` returns the value `0`. Did the constructor make an alias, shallow copy, or deep copy of the argument `grades`? Briefly explain your answer.

> **Solution:** The constructor made an alias because clearing the list `t` also causes the transcript to clear its list of grades.

(c) [2 marks] Suppose that after running the code fragment above, the statement `t.getGPA()` returns the value `9`. Did the constructor make an alias, shallow copy, or deep copy of the argument `grades`? Briefly explain your answer.

> **Solution:** The constructor must have made a copy of some kind because clearing the list `t` did not cause the transcript to clear its list of grades. It must have made a shallow copy because modifying the grade of the course in the list `t` causes the transcript to have a modified grade for the same course.

(d) [2 marks] Suppose that after running the code fragment above, the statement `c == g` returns `true`. Did the constructor make an alias, shallow copy, or deep copy of the argument `grades`? Briefly explain your answer.

> **Solution:** The constructor must have made a copy of some kind because clearing the list `t` did not cause the transcript to clear its list of grades. It must have made a shallow copy because the `CourseGrade c` is the same object as the `CourseGrade g` from the transcript.

11. Consider the following two classes related by inheritance:

```
public class Lock {

	private boolean isLocked;

	protected Lock() {
		this.isLocked = false; // UNUSUAL BECAUSE WE LOCK THE LOCK ON THE NEXT LINE
		this.lock();
	}

	public boolean isLocked() {
		return this.isLocked;
	}

	public void lock() {
		this.isLocked = true;
	}

	protected void unlock() {
		this.isLocked = false;
	}
}

public class KeyedLock extends Lock {

	private boolean isLocked;
	private Key key;

	public KeyedLock(Key k) {
		[PART (a)]
		this.isLocked = true;
		this.key = k;
	}

	@Override
	public void lock() {
		this.isLocked = true;
	}

	public void unlock(Key tryMe) {
		if (this.key.equals(tryMe)) {
			this.isLocked = false;
		}
	}
}
```

(a) [2 marks] What would you write on the line labelled `[PART (a)]` to complete the `KeyedLock` constructor? Explain if the line that you would write is actually required in this case.

> **Solution:** You could write `super();` but the line is not required because the superclass no-argument constructor will be called automatically if you do not include the line.

(b) [2 marks] Consider the following fragment of Java code:

```
Key k = new Key();
Lock lock = new KeyedLock(k);
System.out.println(lock.isLocked());
```

Assume that the constructor has been modified according to your answer in Part (a). What is printed when the fragment of code is run? Explain your answer.

> **Solution:** Surprisingly, `false` is printed. When the `KeyedLock` constructor runs, the `Lock` constructor is called. The `Lock` constructor sets the field `isLocked` to `false` before calling the `lock();` method. `KeyedLock` overrides `lock()` so the `KeyedLock` version of `lock()` runs. The `KeyedLock` version of `lock()` sets the field `isLocked` to `true` in `KeyedLock` but it does not set `isLocked` in `Lock`. The `isLocked()` method in `Lock` uses the `isLocked` field in `Lock` which is `false`.
>
> (2 marks if the answer is false and the explanation states that the field `isLocked` is set to true in `KeyedLock` but not in `Lock`)

(c) [2 marks] What significant error has the implementer of `Lock` made?

> **Solution:** They called a non-final method in the constructor.

(d) [2 marks] What significant error has the implementer of `KeyedLock` made? How does this error affect the user of `KeyedLock` when they try to unlock the lock?

> **Solution:** Two errors have been made. The first error is that the implementer has added a field (`isLocked`) that that the superclass is responsible for. The second error is that they did not call the superclass version of the method `unlock` to set the `isLocked` field in `Lock`.
>
> (2 marks for stating either error)