WRITTEN TEST 2B

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 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 test2B 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):

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

    public Line(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:

```
public class Question {

        public static int a = 1;
        private static int b = 2;

        public double f;
        private double g;

        // constructors not shown

        public static void someMethod() {
                // implementation not shown
        }

        public static void someMethod(Question q) {
                // implementation not shown
        }

        public void anotherMethod(Question q) {
                // implementation not shown
        }
}
```

(a) [2 marks] What fields of `Question` can `someMethod()` use?

Solution: `someMethod` can only use the static fields (`a` and `b`).

(b) [4 marks] What fields of `Question` can `someMethod(Question q)` use? If your answer is different than your answer to part (a), explain why.

Solution: `someMethod(Question)` can use the static fields (`a` and `b`) and it can use the non-static fields as long as it uses the reference `q` to do so. The answer is different from (a) because there is a parameter of type `Question` in this method.

(4 marks if the answer says that all of the fields are usable and that the non-static fields are usable via the parameter.

2 marks if the answer says that only the static fields are usable)

(c) [2 marks] What fields of `Question` can `anotherMethod(Question q)` use? If your answer is different than your answer to part (b), explain why.

Solution: `anotherMethod` can use any field of the class because it is a non-static method.

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 static int numLocks = 0;
        private long id;
        private boolean isLocked;

        public Lock(boolean isLocked) {
               this.id = Lock.numLocks;
               if (isLocked) {
                      this.lock();
               } else {
                      this.unlock();
               }
               Lock.numLocks = Lock.numLocks + 1;
        }

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

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

public class CombinationLock extends Lock {

        private Combination combo;

        public CombinationLock(Combination combo) {
               [PART (a)]
               this.combo = new Combination(combo);
        }

        @Override
        protected void lock() {
               [PART (b)]
               this.combo.shuffle(); // randomly shuffles the dials on the lock
        }

        @Override
        public boolean equals(Object obj) {
               if (!super.equals(obj)) {
                      return false;
               }
               CombinationLock other = (CombinationLock) obj;
               if (!this.combo.equals(other.combo)) {
                      return false;
               }
               return true;
        }

}
```

(a) [2 marks] What would you write on the line labelled `[PART (a)]` to complete the `CombinationLock` constructor? Why is the line required?

> **Solution:** Either `super(true);` or `super(false);` (both answers are acceptable). One of the two is required because a `Lock` constructor must be called and there is no other constructor in `Lock`.

(b) [2 marks] What would you write on the line labelled `[PART (b)]` to complete the overridden version of the method `lock()`?

> **Solution:** `super.lock();` is required to actually lock the lock. The field `isLocked` is private in `Lock` so `CombinationLock` has no other way to access the field.

(c) [2 marks] What sequence of events occurs when a user of the two classes makes a new `CombinationLock`? Is the `CombinationLock` always successfully initialized?

> **Solution:** The `Lock` constructor is called which might call the method `lock` which is overridden by `CombinationLock`. If this happens then the overridden version of `lock` runs which will cause a `NullPointerException` because `this.combo` is `null` when `lock` is running. Therefore, the `CombinationLock` might not be successfully initialized.

(d) [2 marks] `CombinationLock` overrides the `equals` method; is this override required to check if two `CombinationLock` instances are equal? Justify your answer.

> **Solution:** The override is *not* required because `Lock` does not override `equals`. This means that `super.equals(obj)` is `true` if and only if `this` `CombinationLock` and `obj` have the same memory address, which means that `this.combo.equals(other.combo)` will always be `true`.
>
> (The explanation must be partly correct to receive part marks; answering yes with an incorrect answer should not receive any marks).