WRITTEN TEST 2E

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 test2E 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 = new Point2(p1);
        this.end = new Point2(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 mutable

   D. `LineSegment` is a superclass

   Solution: B

3. [2 marks] What must be true about the keys in a Java `Map`?

   Solution: The keys must be unique.

4. [2 marks] What is the definition of the term *privacy leak*?

   Solution: A privacy leak occurs when an object exposes a reference to a mutable field.

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] Which features of the superclass does a subclass inherit?

Solution: All of the non-private fields and methods (but not the constructors).

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.

(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

```
public class DeckOfCards {
    private Set<Card> cards;

    public DeckOfCards() {
        /* IMPLEMENTATION NOT SHOWN but sets this.cards to be equal to
           standard deck of 52 playing cards */
    }

    public DeckOfCards(DeckOfCards other) {
        this.cards = new HashSet<>();
        for (Card card : other.cards) {
            this.cards.add(card);
        }
    }

    public Set<Card> getCards() {
        /* IMPLEMENTATION NOT SHOWN */
    }

}
```

(a) [2 marks] Consider only the copy constructor of `DeckOfCards`; what is the name of the relationship between `DeckOfCards` and its set `this.cards`?

> **Solution:** Composition (not aggregation because the copy constructor makes a new set).

(b) [2 marks] The copy constructor of `DeckOfCards` makes a copy of of `other.cards`. What kind of copy does the copy constructor make?

> **Solution:** A shallow copy (no new `Card` objects are made).

(c) [3 marks] Consider the following program that uses `DeckOfCards`:

```
public class Test {

    public static void main(String[] args) {
        DeckOfCards deck = new DeckOfCards();
        Set<Card> deckCards = deck.getCards();

        System.out.println(deckCards == deck.getCards());
    }

}
```

Assuming that no privacy leak occurs and that there are *no* additional fields in `DeckOfCards` what does the program print? Explain your answer.

> **Solution:** The program must print `false` because `getCards` must return a new set each time if no privacy leak occurs.

(d) [1 mark] Assuming that no privacy leak occurs and that there *are* additional fields in `DeckOfCards` what does the program print? Explain your answer.

Solution: The program can print either `false` or `true` if additional fields are allowed. The additional field could be a copy of the set `this.cards`

The above is sufficient to receive the 1 mark for the question; to illustrate how `getCards` might be implemented consider the following:

```
// this.copy is a Set<Card>
// DeckOfCards never uses this.copy except in getCards

public Set<Card> getCards() {
    if (!this.cards.equals(this.copy)) {
       this.copy = new ArrayList<>(this.cards);
    }
    return this.copy;
}
```

The above implementation of `getCards` has no privacy leak even though it returns a reference to the field `this.copy` because the class does not use the field for anything except this method. If a user modifies `this.copy` so that it is no longer equal to `this.cards` then the method makes a new copy of `this.cards` and returns a new copy.

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() {
		this.id = Lock.numLocks;
		this.lock();
		Lock.numLocks = Lock.numLocks + 1;
		// [PART (a)]
	}

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

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

public class CombinationLock extends Lock {

	private Combination combo;

	public CombinationLock(Combination combo) {
		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; // [PART (c)]
		if (!this.combo.equals(other.combo)) {
			return false;
		}
		return true;
	}

}
```

(a) [2 marks] For this part of the question, assume that there are no subclasses of `Lock`.

When the `Lock` constructor reaches the line labelled `[PART (a)]` you know that the lock will be locked. What else do you know is true about every `Lock` object when the line labelled `[PART (a)]` is reached?

Solution: That every lock will have a unique id.

For the remaining parts of this question, assume that there may be subclasses of `Lock`.

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

Solution: `super.lock();`

The field `isLocked` is `private` so it cannot be accessed directly by `CombinationLock`.

(c) [2 marks] On the line with the comment `[PART (c)]` is the cast safe (in other words, is `obj` guaranteed to be a `CombinationLock` reference)? Explain why you answered 'yes' or 'no'.

Solution: The cast is safe 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 the cast only occurs if `this` `CombinationLock` and `obj` refer to the same `CombinationLock`.

(The explanation must be partly correct to receive part marks; answering yes with an incorrect answer should not receive any marks).

(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).