WRITTEN TEST 2G

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 test2G answers.txt
  ```

  and press enter.

1. [2 marks] What is the main difference between a non-static field and a static field?

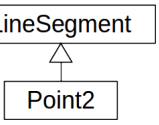
Solution: Every object has its own copy of a non-static field but shares the one copy of the static field. (Or, the class owns the static fields whereas objects own their own non-static fields.)

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

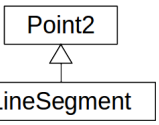
```
public class LineSegment {
    private Point2 start;
    private Point2 end;

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

Which UML diagram best describes the relationship between `LineSegment` and `Point2`? Give your answer as A, B, C, or D.

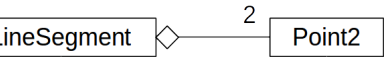
A.



B.



C.



D.

Solution: C

3. [2 marks] A Java `Map` has a collection of keys and a collection of values. What kind of collection is used to store the keys of a map?

Solution: A Set
HashSet is acceptable but not necessarily how the Java maps actually store the keys.
TreeSet receives 1 of 2 marks

4. [2 marks] Under what circumstances can a class that has one or more class invariants use aggregation to manage its fields?

Solution: When the fields are of immutable types.

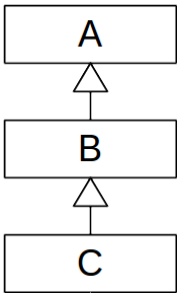
5. [2 marks] What is the main difference between an interface and a regular class?

Solution: The methods in an interface have no implementations (except for default methods which were not discussed in class).
An interface has no fields is also acceptable (technically, an interface allows `public static final` fields but this was never discussed in class).

6. [2 marks] What does the access modifier `protected` mean?

Solution: That the feature is accessible to subclasses.

7. [2 marks] Consider the following inheritance hierarchy:



If class `C` has a `protected` field, which classes are allowed to access that field directly by name?

Solution: `C`

8. [2 marks] Consider the inheritance hierarchy from Question 7 and the following method:

```
public static void someMethod(B b) {
        // IMPLEMENTATION NOT SHOWN
}
```

What variable types can be used as an argument to the method?

Solution: `B` and `C`

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 multiply(Complex x, Complex y) {
		// IMPLEMENTATION NOT SHOWN
		// COMPUTES THE COMPLEX PRODUCT x times y AND RETURNS A
		// REFERENCE TO A NEW COMPLEX NUMBER
	}
}
```

(a) [2 marks] The methods `one` and `multiply` are static methods that return an object reference; what is the name used to describe such methods?

> **Solution:** Static factory method (factory method is acceptable).

(b) [2 marks] Is a static method allowed to use the keyword `this`? Explain why or why not using one or two sentences.

> **Solution:** No. In a method the keyword `this` is a reference used to call the method and a static method is not called using an object reference.

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

```
public class Test {
	public static void main(String[] args) {
		Complex x = Complex.i();
		Complex y = Complex.i();
		Complex z = Complex.multiply(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:** 2. `x` and `y` are both references to the same static field and `z` is a reference to the new object returned by `multiply`.

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

```
public class Test {
	public static void main(String[] args) {
		Complex x = Complex.one();
		Complex y = Complex.one();
		Complex z = Complex.multiply(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. `x` and `y` are references to new objects returned by `one` and `z` is a reference to the new object returned by `multiply`.

10. Consider the following class that represents an integer number having up to `Integer.MAX_VALUE` digits:

```
/**
 * An integer value having at least one and up to as many digits as will
 * fit in a list. A BigInt object owns all of its digits.
 */
public class BigInt {
	private List<Digit> digits;

	public BigInt(List<Digit> d) {
		/* IMPLEMENTATION NOT SHOWN */
	}

	public int numDigits() {
		return this.digits.size();
	}

	public Digit getDigit(int index) {
		/* IMPLEMENTATION NOT SHOWN */
		/* RETURNS THE DIGIT AT THE SPECIFIED INDEX IN THE NUMBER */
	}

	// Reverses the order of the digits in this number
	public void reverse() {
		reverse(this.digits);
	}

	// RECURSIVELY REVERSES A LIST OF DIGITS BY SWAPPING THE FIRST
	// AND LAST DIGITS OF t
	private static void reverse(List<Digit> t) {
		if (t.size() == 1) { // PART (c)
			return;
		}
		String first = t.get(0);
		String last = t.get(t.size() - 1);
		t.set(0, last);
		t.set(t.size() - 1, first);
		reverse( /* [PART (d)] */ );
	}

}
```

(a) [2 marks] Suppose that you make a `BigInt` object using an existing list of digits `t` and then clear the list:

```
BigInt huge = new BigInt(t);
t.clear();
int n = huge.numDigits();
```

If a `NullPointerException` is thrown on the last line of the code fragment shown above, what can you conclude about the constructor of `BigInt` (try to be as specific as possible in your answer; for example, the answer "The constructor is implemented incorrectly" would not receive any marks).

Solution: `this.digits` must be equal to `null` for `numDigits()` to cause a `NullPointerException`; therefore, the constructor probably failed to assign a new list to the field `this.digits`

(2 marks for stating the constructor did not initialize `this.digits`)

(b) [2 marks] Suppose that you make a `BigInt` object using an existing list of digits `t` and then examine the first digit in the number:

```
BigInt huge = new BigInt(t);
System.out.println(t.get(0).equals(huge.getDigit(0)));
System.out.println(t.get(0) == huge.getDigit(0));
```

The program shown above prints two lines of output. What should the program shown above print if the constructor is implemented correctly? Explain your answer using one sentence each to explain each line of output.

Solution: The first line should print `true` because the first digit of `huge` should be equal to the first digit of `t`.

The second line should print `false` because the first digit of `huge` should be a new copy of the first digit of `t` (otherwise the `BigInt` does not own all of its digits).

(c) [2 marks] Is the base case of the method `reverse` (marked with the comment `PART (c)` correct? Explain your answer using one or two sentences.

Solution: No, the base case is incorrect because the method will not terminate if `t` has an even number of digits.

(Note that the correct base case is not `if (t.size() == 0)` because then the method will terminate if `t` has an odd number of digits. The correct base case is `if (t.size() < 2)`)

(d) [2 marks] What is the argument (marked with the comment `PART (d)`) to the recursive invocation?

Solution: The argument is `t.subList(1, t.size() - 1)`

`reverse(t.subList(1, t.size() - 1));` is also acceptable.

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: Everb `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();`

(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: Yes, the cast is safe. `Lock` does not override `equals` so `super.equals(obj)` will be `true` if and only if `this CombinationLock` and `obj` refer to the same object. If `this` and `obj` refer to the same object then they both 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: No, the override is not required. `Lock` does not override `equals` so `super.equals(obj)` will be `true` if and only if `this CombinationLock` and `obj` refer to the same object. If `this` and `obj` refer to the same object then they both have the same combination.

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