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 does the modifier `final` mean when applied to a field of a class?

> **Solution:** The field can be assigned a value only once.

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 Line(Point2 p1, Point2 p2) {
		this.start = new Point2(p1);
		this.end = new Point2(p2);
	}

	public Point2 getStart() {
		return new Point2(this.start);
	}

	public Point2 getStop() {
		return new Point2(this.stop);
	}
}
```

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` has a privacy leak

> **Solution:** B

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 a Java interface?

Solution: A set of method declarations (and their contracts).

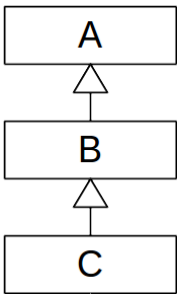
5. [2 marks] When a class has fields that are not of primitive type, we often need to consider using composition instead of aggregation. Why is composition often required instead of aggregation?

Solution: To maintain class invariants, or to prevent changes in state from outside of the class.

6. [2 marks] Is an overridden method also an overloaded method? Explain your answer using one sentence.

Solution: No, an overridden method has the same signature as the superclass method whereas overloaded methods have different signatures.

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



Which class or classes are substitutable for the class named `B`?

Solution: `B` and `C`

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

Solution: Call another constructor (often in the superclass).

9. Consider the following implementation of a class named `Counter`:

```
public final class Counter {

	private static final Map<String, Counter> counters = new HashMap<String, Counter>();
	private String name;
	private int value;

	private Counter(String name) {
		this.name = name;
		this.value = 0;
	}

	public String name() {
		return this.name;
	}

	public int value() {
		return this.value;
	}

	public void incr() {
		this.value++;
	}

	public void reset() {
		this.value = 0;
	}

	public static getCounter(String name) {
		if (Counter.counters.contains(name)) {
			return Counter.counters.get(name);
		}
		else {
			Counter c = new Counter(name);
			Counter.counters.put(name, c);
			return c;
		}
	}
}
```

The `Counter` class is an example of a well-known design pattern used in software engineering.

(a) [2 marks] The method `getCounter(String)` is a `static` method that returns a reference to a new object; what is the name given to such kinds of methods?

> **Solution:** static factory method (or just factory method)

(b) [2 marks] Consider the following fragment of code that uses the `Counter` class:

```
public class UseCounter {

	public static void strike(Counter strikes, Counter outs) {
		strikes.incr();
		if (strikes.value() == 3) {
			outs.incr();
			strikes.reset();
		}
	}

	public static void main(String[] args) {
		Counter c1 = Counter.getCounter("strikes");
		Counter c2 = Counter.getCounter("outs");
		Counter c3 = Counter.getCounter("strikes");

		strike(c1, c2);
		strike(c1, c2);
		strike(c3, c2);

		// HERE
	}
}
```

On the line with the comment `HERE` how many `Counter` objects are in memory?

> **Solution:** 2 (one counter with the name `strikes` and a second with the name `outs`)

(c) [2 marks] See part (b). On the line with the comment `HERE` what is the count held by `c1`, `c2`, and `c3`?

> **Solution:**
> `c1` 0
> `c2` 1
> `c3` 0
>
> `c1` and `c3` are aliases; thus, they have the same value.

(d) [2 marks] See part (b). Is it possible to change the name of any of the `Counters` referenced by `c1`, `c2`, or `c3`? Explain your answer using one or two sentences.

> **Solution:** No because strings are immutable (and the class does not provide a method to change the name).

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

```
/**
 * An integer value having 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 */
	}

	// Multiplies this number by 10 to the power x; x may be negative.
	public void times10toPower(int x) {
		times10toPower(x, this.digits);
	}

	// RECURSIVELY MULTIPLIES THE BIGINT REPRESENTED BY t BY
	// 10 RAISED TO THE POWER x USING INT ARITHMETIC;
	// x MAY BE NEGATIVE
	private static void times10toPower(int x, List<Digit> t) {
		/* BASE CASE(S) NOT SHOWN */

		if (x < 0) {
			t.remove(t.size() - 1);
			times10toPower(x + 1, t);
		}
		else {
			t.add(new Digit(0));
			times10toPower(x - 1, t);
		}
	}

}
```

(a) [2 marks] Assuming that `Digit` is *immutable*, what would you do to complete the constructor of `BigInt`? You can answer this question by writing Java code or you can briefly describe what is required.

Your answer for part (a) should have an important difference compared to your answer for part (b).

Solution: Make a shallow copy of `d`; a copy is required for the `BigInt` to own its digits, but a deep copy is not required if `Digit` is immutable.

```
this.digits = new ArrayList<>(d);
```

(2 marks for a suitable explanation or for the correct Java code)
(1 mark if the answer states that a copy or a defensive copy is required but does not state that a shallow copy is sufficient)

(b) [2 marks] Assuming that `Digit` is *mutable*, what would you do to complete the constructor of `BigInt`? You can answer this question by writing Java code or you can briefly describe what is required.

Your answer for part (b) should have an important difference compared to your answer for part (a).

Solution: Make a deep copy of `d`; a copy is required for the `BigInt` to own its digits, and a deep copy is required if `Digit` is mutable otherwise changing a digit in `d` will change the digit in the number.

```
this.digits = new ArrayList<>();
for (Digit dig : d) {
      this.digits.add(dig);
}
```

(2 marks for a suitable explanation or for the correct Java code)

(c) [2 marks] Three base cases are required for the recursive method `times10toPower`.

What is one of the suitable base cases for the recursive method `times10toPower`? The method should not throw an exception for any argument list `t` unless the size of the list exceeds `Integer.MAX_VALUE` or if the Java Virtual Machine runs out of memory.

If you need to make a new `Digit` assume that `Digit` has a constructor that accepts the value of the digit.

Solution: See part (c) for all 3 base cases.

(2 marks for any of the base cases)

(d) [2 marks] What is a second of the suitable base cases for the recursive method `times10toPower`? The method should not throw an exception for any argument list `t` unless the size of the list exceeds `Integer.MAX_VALUE` or if the Java Virtual Machine runs out of memory.

If you need to make a new `Digit` assume that `Digit` has a constructor that accepts the value of the digit.

Solution: (2 marks for any of the base cases below)

One base case occurs if `x` is equal to 0 which is equivalent to multiplying the number by 1.

```
if (x == 0) {
        return;
}
```

(2 marks for a suitable explanation or for the correct Java code)

A second base occurs when `t` has one digit and `x` is negative; this is equivalent to dividing by a power of 10 which would cause the value of the number to become 0.

```
if (x < 0 && t.size() == 1) {
        t.remove(0);
        t.add(new Digit(0));
        return;
}
```

(2 marks for a suitable explanation or for the correct Java code)

The third base case occurs when `t` has `Integer.MAX_VALUE` digits and `x` is positive. It is impossible to add digits to the number (because the list cannot hold any more digits) so the method should throw an exception.

```
if (x > 0 && t.size() == Integer.MAX_VALUE) {
        throw new IllegalArgumentException(); // or any other exception
}
```

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.
>
> Alternatively, you could write `super.lock();` which is also not required because the superclass no-argument constructor will be called automatically if you do not include the line.
> (2 marks if a suitable answer and explanation is given)

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

```
Lock k = new KeyedLock();
System.out.println(k.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 if you call the superclass constructor in the subclass constructor. 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`.
>
> If you answered `super.lock();` in part (a) then `true` is printed because the superclass version of `lock()` is called (instead of the overridden version) and the superclass field `isLocked` will be set correctly.
> (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)