

Massachusetts Institute of Technology  
6.170 Laboratory in Software Engineering  
Spring 2001

# Quiz 1

Wednesday, March 7, 2001

Name: \_\_\_\_\_

Athena username: \_\_\_\_\_

**Section (circle one):**

- |                             |                            |                           |
|-----------------------------|----------------------------|---------------------------|
| Section 1, Felix Klock      | Section 5, Jeffrey Sheldon | Section 9, Kenneth Lu     |
| Section 2, Allison Waingold | Section 6, Andreas Hofman  | Section 10, David Maze    |
| Section 3, Matt Deeds       | Section 7, Jeremy Nimmer   | Section 11, Delphine Nain |
| Section 4, Sameer Ajmani    | Section 8, John Holmes     |                           |
- 

This quiz is 50 minutes long. It contains 39 questions (you will answer 32 questions on this sheet, and will answer the remainder elsewhere) and 7 pages (excluding this one). Please check your copy to make sure it is complete before you start. You may separate this sheet from the rest of the test, but turn in all pages, together, when you are finished. Write your username and section on the top of pages 5–7.

Please write neatly; we cannot give credit for what we cannot understand.

The problems are weighted as follows:

true-false : multiple-choice : short-answer :: 1 : 2 : 6

Expect to spend (on average; some problems are easier, some are harder) about 30 seconds per true-false question, 1 minute per multiple-choice question, and 3 minutes per short-answer question.

Good luck!

**True/false:**

1	2	3	4	5	6	7	8	9	10	11	12

13	14	15	16	17	18	19	20	21	22	23	24

**Multiple choice:**

25	26	27	28	29	30	31	32

## True/False [1 point each]

1. Class type hierarchies (classes and subclasses, no interfaces) in Java form tree structures.
2. Interface type hierarchies in Java form tree structures.
3. A good test suite proves the absence of errors in a program.
4. The revealing subdomains for a given procedure are determined by the specification of that procedure.
5. Two inputs are in the same revealing subdomain if a program has the same behavior on both inputs.
6. Regression testing tests all modules together, to validate the overall program.
7. It is often possible to test individual methods of an ADT in isolation using unit testing.
8. The goal of debugging is to eliminate the current bug as quickly as possible.
9. Black-box testing of a procedure should usually include cases where the requires clause is not met.
10. Derived specification fields may change from implementation to implementation, so clients should not depend on them.
11. It is possible that the code of `A.foo` is executed during evaluation of `(new B()).foo()` in the following?

```
class A {  
    ...  
    public foo() { ... }  
}  
class B extends A {  
    ...  
    public foo() { ... }  
}
```

12. Java differs from Scheme in that Scheme objects are heap-allocated, whereas Java's are declared.
13. Java differs from Scheme in that Scheme objects have no runtime type, whereas Java's do.
14. After a program accesses an object for the last time, the object will (eventually) be garbage-collected.

Write your answers to these questions on the answer page, not on this page.

---

15. The Java compiler alerts programmers of potential errors by enforcing that compile-time types and run-time types are identical.
16. The Java compiler ensures the absence of array bounds errors at runtime.
17. Performance-critical code usually should not declare exceptions, for reasons of efficiency.
18. When a module is replaced by another module with the identical specification, integration tests should be repeated.
19. A class that overrides `equals` should also override `hashCode`.
20. An ADT does not always guarantee deterministic behavior.

For questions 21–24, assume

- B extends A,
  - a refers to an object whose compile-time and run-time type is A, and
  - b refers to an object whose compile-time and run-time type is B.
21. `b.apply(a)` can behave differently than `((B) b).apply(a)`.
  22. `b.apply(a)` can behave differently than `((A) b).apply(a)`.
  23. `b.apply(a)` can behave differently than `b.apply((B) a)`.
  24. `b.apply(a)` can behave differently than `b.apply((A) a)`.

## Multiple choice [2 points each]

25. Which of the following is the easiest requires clause to satisfy?
- (a) `requires: arguments are non-null`
  - (b) `requires: false`
  - (c) `requires: true`
  - (d) none of the above
26. Suppose that the specification of `Quux.foo()` is stronger than the specification of `Bar.foo()`. A program that works when it calls `Bar.foo()`
- (a) will work if it calls `Quux.foo()`
  - (b) will not work if it calls `Quux.foo()`
  - (c) may work if it calls `Quux.foo()`
27. Suppose that the specification of `Quux.foo()` is stronger than the specification of `Bar.foo()`. A program that works when it calls `Quux.foo()`
- (a) will work if it calls `Bar.foo()`
  - (b) will not work if it calls `Bar.foo()`
  - (c) may work if it calls `Bar.foo()`
28. Specification fields should usually be
- (a) Java classes such as `Vector`
  - (b) Java interfaces such as `List`
  - (c) mathematical types such as “sequence”
  - (d) none of the above
  - (e) a or b
  - (f) a, b, or c

29. Which of the following procedural specifications is stronger?

- (a) requires:  $x \geq 0$
- (b) requires:  $x > 0$
- (c) equally strong
- (d) incomparable

30. Which of the following procedural specifications is stronger?

- (a) modifies: nothing
- (b) modifies:  $x$
- (c) equally strong
- (d) incomparable

31. Which of the following procedural specifications is stronger?

- (a) returns: 5
- (b) returns: 5 if  $x \geq 0$   
throws: `NegativeException` if  $x < 0$
- (c) equally strong
- (d) incomparable

32. Which of the following procedural specifications is stronger?

- (a) requires:  $x < 0$   
returns: a value  $> 0$
- (b) requires:  $x \leq 0$   
returns: a value  $\geq 0$
- (c) equally strong
- (d) incomparable

**Short answer [approximately 6 points each]**

33. Can a deterministic procedure satisfy an underdetermined spec? If so, give an example. If not, explain (in one sentence) why not. [6 points]

34. Write a code fragment for which it is possible to write a test suite that achieves 100% statement coverage but not 100% decision (edge) coverage. (You need not write down the test suite.) [5 points]

35. Give an example of an immutable abstraction with a mutable representation. (The example may be in words or in code.) Briefly (one sentence) describe why an immutable representation would be inferior. [6 points]

36. The following program finds the index of an array element by recursively dividing the array in half.

```
public class Buggy {
    // returns: the first index, between left and right inclusive, of n in array a
    // throws: RuntimeException if n does not appear in a[left..right]
    private static int indexOf(int[] a, int n, int left, int right) {
        // ... recursive implementation elided ...
    }

    public static int indexOf(int[] a, int n) {
        return indexOf(a, n, 0, a.length-1);
    }

    public static void main(String argv[]) {
        int[] a = { 0, 5, 3, 2, 7, 6, 9, 1, 4, 8 };
        System.out.println("indexOf(0): " + indexOf(a,0));
        System.out.println("indexOf(1): " + indexOf(a,1));
    }
}
```

The program throws an exception after printing the index of 0. Give two hypotheses for the source of the error and briefly discuss how you would test those hypotheses, including what you would learn from your investigations. [8 points]

37. Give an example of classes **A** and **B** such that **B** is a behavioral (true) subtype of **A** but not a Java subtype (subclass) of **A**, or briefly (one sentence) explain why no such pair exists. [5 points]
38. Briefly (one sentence each) give two reasons why specifications are advantageous. [5 points]
39. Briefly (one sentence each) give two situations in which a require clauses is preferable to specifying that an exception is thrown. [7 points]