University of Washington
CSE 331 Software Design & Implementation
Spring 2010

Midterm exam

Friday, April 23, 2010

Name: ***Solutions*** _____

UW Net ID: _____

This quiz is closed book, closed notes. You have **50 minutes** to complete it. It contains 28 questions and 8 pages (including this one), totaling 100 points. Before you start, please check your copy to make sure it is complete. Turn in all pages, together, when you are finished. **Write your initials on the top of *ALL* pages.**

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

Good luck!

Page	Max	Score
2	26	
3	16	
4	6	
5	8	
6	20	
7	24	
Total	100	

1 True/False

(2 points each) Circle the correct answer. T is true, F is false.

1. **T /** **F** When specification testing, it is good practice (but not required) to check that a `RuntimeException` is thrown when invalid input is passed to a method.

2. **T** **/ F** The representation invariant (RI) is guaranteed to hold for a correct implementation of an *immutable* abstraction, when no method in the class is executing.

3. **T** **/ F** The representation invariant (RI) is guaranteed to hold for a correct implementation of a *mutable* abstraction, when no method in the class is executing.

4. **T /** **F** It is a violation of the abstraction barrier for a class to directly use fields of its superclass.

5. **T /** **F** When the internal state of an object changes, its hash code must also change.

6. **T /** **F** A method may throw an exception if, and only if, the exception is listed in the throws clause of the method specification.

7. **T /** **F** A good black-box test should be designed to cover every branch of the code, because any untested code might harbor bugs.

   Black-box tests should not consider the implementation, only the specification.

8. **T /** **F** A class that represents a Cartesian point should include a method to compute the x and y coordinates, *or* the rho and theta coordinates, but *not* all 4 methods, because including all of these makes the implementation bulky, and clients can always do the transformation.

   These methods conceptually belong to the class, not to a client. Furthermore, the existence of these methods permits changing the implementation, and possibly enables more efficient operations.

9. **T /** **F** In a class that implements an *immutable* ADT, the representation should never change (after the constructor is exited).

   Benevolent side effects may change the concrete representation without affecting the abstract value.

10. **T /** **F** When the user passes an argument that violates the precondition, it is helpful to throw an exception and to document this behavior in the throws clause of the specification.

    It is helpful to throw the exception, but* not *to document it in the throws clause — that would make the specification inconsistent, as it would specify behavior when the precondition is not satisfied.

Recall that at a call site `x.f(y)`, an illegal value of `y` can violate the method precondition of `f`. Assume that there are no errors in the code that implements `x`'s class, and that `x` satisfies the rep invariant.

11. **T** **/ F** At a call site `x.f(y)`, it is possible for a value of `x` to violate the method precondition of `f`.

Suppose that there are two different implementations C1 and C2 of a given ADT, and the implementations have different representation invariants.

12. **T /** **F** It is a violation of the abstraction barrier for client code to intermix objects of type C1 and C2 in computations, or to intermix them in collections such as `List`.

13. **T /** **F** It is a violation of the abstraction barrier for the code of class C1 to make calls against an object of class C2.

2 Multiple choice (4 points each)

14. An `equals` method must satisfy which of the following properties? (Circle all that apply.)

    (a) atomicity
    (b) consistency with `hashCode`
    (c) consistency with `toString`
    (d) no side effects to the abstraction
    (e) no side effects to the rep
    (f) reflexivity
    (g) symmetry
    (h) transitivity

    consistency with hashCode, no side effects to the abstraction, reflexivity, symmetry, transitivity

15. Why is it valuable to formally compare two specifications to one another? (Circle all that apply.)

    (a) To determine whether the implementation is correct.
    (b) To determine whether one ADT is a true subtype of another.
    (c) To determine whether a procedure satisfying one can be substituted for a procedure satisfying the other.
    (d) To determine which one is more elegant.
    (e) To determine which one is more appropriate for use by a client.
    (f) As part of the process of weakening one of the specifications.

    B,C

16. An `IllegalArgumentException` must be thrown when (circle all that apply):

    (a) A non-`null` parameter violates the @requires clause of the method
    (b) A `null` parameter violates the @requires clause of the method
    (c) The object state is inappropriate for method invocation
    (d) None of the above

    None of the above

17. Suppose that specification S1 differs from specification S2 in that S1 has a strictly stronger precondition than S2, and S1 has a strictly stronger postcondition. What relations between S1 and S2 are possible? (Circle all that apply.)

    (a) S1 may be stronger than S2
    (b) S1 may be weaker than S2
    (c) S1 may have the same strength as S2
    (d) None of the above.

    None of the above.

3 Comparing implementations and specifications

18. (6 points) Consider the following specifications for a procedure that takes an integer as an argument:

    (a) returns an integer $\geq$ its argument

    (b) returns a non-negative integer $\geq$ its argument

    (c) returns argument + 1

    (d) returns argument^2

    (e) returns `Integer.MAX_VALUE`

    Consider these implementations:

    (i) `return arg * 2;`

    (ii) `return Math.abs(arg);`

    (iii) `return arg + 5;`

    (iv) `return arg * arg;`

    (v) `return Integer.MAX_VALUE;`

    Place a check mark in each box for which the implementation satisfies the specification. If the implementation does not satisfy the invariant, leave the box blank.

    Ignore overflow.

	Specification				
Impl.	(a)	(b)	(c)	(d)	(e)
(i)					
(ii)	***yes***	***yes***			
(iii)	***yes***				
(iv)	***yes***	***yes***		***yes***	
(v)	***yes***	***yes***			***yes***

19. (8 points) Consider the following four specifications for `double log(double x)`, a method that returns the natural logarithm of the input `x`:

```
A       @requires x > 0
        @return y such that |e^y - x| <= 0.1

B       @return y such that |e^y - x| <= 0.001
        @throws IllegalArgumentException if x <= 0

C       @requires x > 0
        @return y such that |e^y - x| <= 0.001

D       @return y such that |e^y - x| <= 0.001 if x > 0
        and Double.NEGATIVE_INFINITY if x <= 0
```

For each of the following pairs of specifications, circle the stronger specification, or circle "neither" if the two specifications are either equivalent or incomparable.

(i)	A	[B]	neither
(ii)	A	[C]	neither
(iii)	A	[D]	neither
(iv)	[B]	C	neither
(v)	B	D	[neither]
(vi)	C	[D]	neither

4 Short answer

20. (3 points) Write two words that describe when (that is, under what circumstances) an implementation should check preconditions.

    (a) ***inexpensive***

    (b) ***convenient***

    (c) debugging

    "Before" and "after" a method body could be acceptable answers for a* rep invariant*, but "after" does not make sense for a precondition.

21. (4 points) How, if at all, are the RI and the AF related? Answer in 1 sentence.

    The domain of the AF is objects that satisfy the RI.

22. (4 points) Write the full transition relation for the following specification:

    requires $x > 2$
    returns y such that $y \geq x$
    `int anyGte(int x)`

    ...
    ..., $\langle 1,-1\rangle$, $\langle 1,0\rangle$, $\langle 1,1\rangle$, ..., $\langle 1,$ ***`RuntimeException`***$\rangle$, ..., $\langle 1,$ ***infinite loop***$\rangle$, ...
    ..., $\langle 2,-1\rangle$, $\langle 2,0\rangle$, $\langle 2,1\rangle$, ..., $\langle 2,$ ***`RuntimeException`***$\rangle$, ..., $\langle 2,$ ***infinite loop***$\rangle$, ...
    $\langle 3,3\rangle$, $\langle 3,4\rangle$, $\langle 3,5\rangle$, ...
    $\langle 4,4\rangle$, $\langle 4,5\rangle$, $\langle 4,6\rangle$, ...
    ...

23. (4 points) In no more than 2 sentences, what is the difference between verification and validation?

    Verification is the process of determining whether the implementation satisfies the specification. Validation is the process of determining whether the specification meets the functional (user) requirements of the system — whether it is fit for use.

24. (5 points) In 1 sentence each, state the two key limitations of Java constructors and why each are a limitation.

    (a) ***Constructors always return a new object, but for both space and time efficiency it can be desirable to re-use objects, as in the interning design pattern.***

    (b) ***Constructors always return an object of the requested type, never a subtype, which is inconvenient when clients need a particular type.***

25. (8 points) A general way to generate tests is to divide the input domain into distinct partitions, then choose one input from each domain.

    (a) In one sentence, what property should be true of *each* partition?

    Each partition is uniform with respect to failure; that is, the program should either fail for every input in the partition, or succeed for every input in the partition.

    Partial credit was given for mentioning heuristics, such as code coverage — but recall that those heuristics are intended to approximate the real answer given here. The theory behind partition testing explains why those heuristics tend to work.

    Note that this question asked for a property of each partition, not a property of the* set *of partitions. Also, saying that the partitions should be distinct and exhaustive is merely restating the definition.

    (b) In one sentence, explain why violating that property can mean that the test suite fails to detect (some) errors.

    If a failing input exists, but every partition contains at least one succeeding input (which means that some partition contains both types of input), then the arbitrary choice could choose only succeeding tests and miss errors.

    (c) In one sentence, state how the property can be violated but the suite is still guaranteed to find all errors.

    If, for every distinct defect in the program, some partition contains only inputs that fail because of that defect, then the test suite will catch all defects.

    A common incorrect answer is to test all inputs. This is guaranteed to find all errors, but it redefines the partitions to have one input per partition, in which case the uniformity condition on each partition is satisfied.

    It isn't right to answer that you could over-partition, so that the partitions you use are sub-partitions of the coarsest (most efficient) partitions that satisfy the uniformity requirement.

    Another incorrect answer was boundary testing. This is a heuristic that aims to over-partition, but it provides no guarantees.

26. (6 points) In one sentence each, give the two most important reasons that throwing an exception is preferable to returning a special value.

    (a) ***A special value may be hard to distinguish from a real result.***

    (b) ***Special values are error-prone: the programmer may forget to check them.***

    (c) An exception can carry arbitrary data that may be useful in describing the problem.

    (d) It can simplify the program logic.

    Many people said an exception implements the "fail fast" paradigm. But special values can do this just as easily: the library returns a special value as soon as there is a problem, and the client checks for it. Likewise, exceptions do not prevent any more (or fewer) problems than special values, if each is used correctly.

    By contrast, exceptions give you "guaranteed fail", which is different but also valuable.

27. (4 points) In the PS3 and PS4 testing file format, a single test file can contain multiple differently-named graphs. In one sentence, give a reason why the staff chose to have multiple distinct graphs in a single test file. (Hint: the answer is not "to reduce the number of files in the test suite.")

This enables testing whether the class uses static variables, suffers representation exposure, or has other undesired dependences/interference between distinct `Graph` ***objects.***

An inadequate answer is that it tests whether the same node is in multiple graphs. Nodes associated with the same name might be in multiple graphs, but that is a different matter.

The test file format does not permit comparison of Graph objects, so that is not the reason either.

28. (6 points) PS4 uses, as its example graph, a map of roads on the UW campus. In one sentence each, give a reason that this is a good idea for testing, and a reason it is a bad idea.

    Obviously, you are going to do testing with some kind of small node and edge data; the question here is about the appropriateness of the UW campus data.

    (a) Good: ***It leverages our intuitions about what paths are the right ones between any two points.***

    An incorrect answer is that it is like the real-world data that the system will eventually use. The purpose of testing is to create test cases that reveal errors, not to simulate use.

    (b) Bad: ***Reality may not contain (or it may be difficult to find) the corner cases that you wish to test, such as multiple paths of similar length, longer paths with a smaller number of segments, each type of turn, each type of road, multiple ZIP codes, etc.***

    Another acceptable answer is that creating a realistic graph requires a lot of work to have it correspond closely enough to reality. If the graph is only somewhat like reality, then testers' intuitions may mislead them.

    An incorrect answer is that a larger and more complex graph is needed. Testing on complex data just makes it harder for you to understand the errors! It is better for you to carefully craft specific test cases, rather than to throw heaps of data at your program.