University of Washington
CSE 331 Software Design & Implementation
Winter 2011

Final exam

Wednesday, March 16, 2011

Name: ***Solutions***

CSE Net ID (username):

UW Net ID (username):

This exam is closed book, closed notes. You have **110 minutes** to complete it. It contains 38 questions and 15 pages (including this one), totaling 220 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	24	
3	28	
4	28	
5	30	
6	26	
7	14	
8	6	
10	18	
11	10	
12	8	
13	12	
14	20	
Total	224	

1 True/False

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

1. **T** / F When reasoning about uses of an ADT, it is permissible to ignore the code.
2. **T** / F When reasoning about uses of an ADT, it is permissible to ignore observers.
3. T / **F** Given two MDDs (module dependency diagrams), the one with fewer edges is better. ***This is generally true, but is not guaranteed.***
4. T / **F** `List<Integer>` is a true subtype of `List<Number>`.
5. T / **F** Regular specification fields are used for values represented by a concrete field in the implementation, and derived specification fields are used for values with no representation (as a concrete field) in the implementation.
6. T / **F** For each concrete field in the implementation, there should be a specification field (of some variety) in the abstraction.

2 Multiple choice

Mark each of the following that can be true.

7. (3 points) When is overloading resolved?
   (a) at compile time
   (b) at run time
   (c) none of the above
   a. at compile time
8. (3 points) Overloading is resolved based on what information?
   (a) declared types
   (b) run-time types (object classes)
   (c) none of the above
   a. declared types
9. (3 points) When is overriding resolved?
   (a) at compile time
   (b) at run time
   (c) none of the above
   b. at run time
10. (3 points) Overriding is resolved based on what information?
    (a) declared types
    (b) run-time types (object classes)
    (c) none of the above
    b. run-time types, which are object classes

3 Short answer

11. (3 points) The representation invariant maps ***an object*** to ***a boolean***. (Fill in the blanks.)

12. (3 points) The abstraction function maps ***an object*** to ***an abstract value***. (Fill in the blanks.)

13. (10 points) In each of the following situations, which is better, a checked or an unchecked exception?

    (a) `Map.getExisting` when the key is not in the map ***unchecked***

    (b) `Rational.divide` when the key is not in the map ***unchecked***

    (c) `Object.clone` when the system is out of memory ***unchecked***

    (d) `File.load` when the file does not exist ***checked***

    (e) `File.load` when there is a disk failure ***unchecked***

    Recall that a library should use a checked exception when it is essential that the client handle the condition. Use an unchecked exception if the client can prevent the exception (due to knowledge about the application's logic or due to a check), or if the exception is so severe that the client can't do anything about it.

14. (12 points) In one sentence each, give three distinct reasons that you might *not* want to check the rep invariant at the entry and/or exit of a given method. Consider only instance methods that are not a constructor.

    (a) ***You are done with debugging and have deployed the application, and the rep invariant is very expensive and slows down the application. For instance, it degrades the asymptotic time complexity of the operation.***

    (b) ***The method is a private method, for which the rep invariant does not hold on entry and/or exit.***

    (c) ***The method cannot not change the rep, so there is no need to check the rep invariant at exit from the method. One reason is that the rep is immutable, and another reason is that the method's implementation does not modify the rep. (These justifications require a proof, but you are usually testing because you do not trust your proof! So, these are not particularly good reasons.)***

15. (12 points) Suppose that you have observed an error in an execution of your program. Give three ways to localize the defect to a small part of the program *while debugging*, and explain (in 1 phrase or sentence each). Give ways that are as different from one another as possible.

    The key idea is divide and conquer, and each answer is a technique for that.

    - ***assertions: if it fails, then the error must have occurred earlier in execution***
    - ***unit testing: if it fails, then the error must occur in the small part of the program that it executes. This could be expressed in terms of doing the testing top-down or bottom-up, which are just approaches to unit testing.***
    - ***regression testing: if it fails, then the error must be related to recently-changed code***

    Just "binary search" is too vague an answer. Any answer that is about minimizing the* input *is incorrect.

16. (10 points) In one sentence each, give an advantage of debugging using a debugger, and an advantage of debugging using print statements. If there are multiple answers, choose the best or most important one. Assume the bug is not timing-related. (Hint: don't give the answer, "you have to learn to use the debugger"; assume you are already proficient with it.)

    Advantage of using a debugger: ***Using a debugger is quicker: you don't need to edit, recompile your program, and re-run the test case.***
    Using a debugger lets you quickly get information about values you didn't think of putting in the log, without going through the above process.
    Using a debugger does not clutter your code with debugging statements.

    Advantage of using print statements: ***Using a trace log lets you see multiple moments of time at once, and directly compare them in your text editor or otherwise.***
    Using a trace log permits saving debugging information from the field or for later examination, especially by a different person.
    Print statements may impact timing less in multi-threaded programs (since most debuggers only pause the current thread).
    A less compelling reason is that using a trace log may permit more expressive formatting of data.

17. (6 points) Lecture said that the set of all non-negative real numbers is not a valid choice for a decrementing function. Would a closed set of real numbers work? (An example is all numbers x such that $0 \leq x \leq 1$.) This set has a minimum, and you can compare any 2 real numbers. Answer, and give one sentence of explanation.

    No. It is possible to reduce the value indefinitely many times without ever reaching the minimum.

18. (6 points) Why did Fred Brooks call the man-month "mythical" in his eponymous book and essay? Answer in one sentence.

    Man-months are not fungible. Measuring a project's cost in terms of them does not permit an accurate determination of the time to completion, given a certain number of workers.

19. (6 points) In one sentence, what is the scientific method?

    Formulate a hypothesis, then perform a systematic, repeatable experiment that has the potential to refute it.

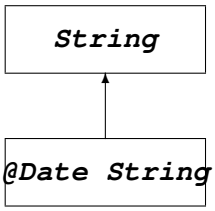
20. (6 points) Which is more important to clients of an abstraction? The abstraction function, the representation invariant, or are they of equal importance? Explain in one sentence.

    Neither: both are irrelevant to the client, because both depend on the implementation.

21. (6 points) Which gives more flexibility to a client (a user) of an ADT: an ADT that uses checked or unchecked exceptions? Explain why, in 1 sentence.

    Unchecked, because the client can choose to catch it or to ignore it if the client knows that it cannot happen.

22. (6 points) Consider two types, `String` and `@Date String`, where the latter is a String that can be interpreted as a date, such as `"July 4, 1776"` or `"3/16/2011"`. Draw the type hierarchy for these two types. The hierarchy may include a third type, if needed.



23. (6 points) Three ways for module A to depend on module B are for A to be a subclass of B, for A to have a field of type B, or for A to take an argument (or return a value) of type B. Which of these, if any, is the most severe or strongest dependence? Explain in no more than 2 sentences.

    They all have the potential to be equally severe. In each case, a change to the spec of one requires revisiting/rewriting the other. In each case, there are both specification and implementation changes that would be inconsequential, and others that would cause terrible problems. In each case, A might depend on every public method and field of B.

    We also gave credit for saying that the subclassing relationship is most severe, because then A depends on both the specification **and the implementation** ***of B (recall the*** `add` ***vs.*** `addAll` ***example of lecture).***

24. (12 points) Explain the difference between synchronous and asynchronous callbacks. When is each one appropriate? Give an example of each. (Use no more than one sentence for each part.)

    A synchronous callback is a method call from a library into client code, which occurs during the execution of a library method. It is appropriate when the library method needs more information from the client, in order to return a value. An example is a call from a collection class to `equals` ***or*** `hashCode`***.***
    An asynchronous callback is a method call from a library into client code, that occurs at an arbitrary time when the library becomes aware of some event of interest to the client — not necessarily as a result of some action by the client code. The client typically **registers** ***interest in receiving the callback. An example is Swing listeners.***

    Vague answers like "at the same time" (as what?) are not sufficient.

25. (8 points) A Macintosh-style menu item (at the top of the screen, not the top of the window as in a Windows-style menu) has effectively infinite height, because it can be thought of as extending infinitely upward. This infinite size does not affect the "reaction time" part of Fitts's law, but ought to make the "movement time" part of Fitts's law nearly zero (limited only by the fastest possible muscle movement, which is very fast). Explain, in 1-2 sentences, why that is an oversimplification of the benefit.

    The fact that a target has infinite size does not mean that it is infinitely easy to hit; the menu still has a finite **width** ***that must be hit, so only one of the two dimensions has been made smaller. Effectively, the smallest dimension of the target has been increased from the height of the menu item to its width, which is generally larger.***

26. (6 points) Would it be possible for an ADT to contain both producer and mutator methods? Why, or why not, would this make sense? Answer in no more than 2 sentences.

    It would be perfectly legal, but it is not encouraged by the CSE 331 methodology. Typically an object is put into a particular desired state either by being constructed in that state, or by being mutated into that state. If there are two ways to get the same result, then the interface may be excessively large and might be confusing to users.

27. (8 points) Suppose that a program calls method M, which satisfies specification S. You have not examined nor verified the program's source code, but it passes an *exhaustive* test suite (the suite tests every legal input and verifies all postconditions). Suppose that you replace the program's use of M by a use of method M', which satisfies specification S', and S' is stronger than S. Does the program still pass its test suite? Explain why or why not in no more than two sentences.

    The program does not necessarily still pass its test suite. The program may* accidentally *depend on properties of the* implementation *of M, that are not satisfied by M'. If we had verified the program source code, using only S in our proof, then the program would still be guaranteed to pass its test suite when using M'.

4 Reasoning about code

28. (6 points)

Give the weakest precondition for the following code, with respect to the postcondition $x > y$. Assume that `p` is `boolean` and `x` and `y` are `int`.

```
p = x>y;
if (p) {
  x++;
} else {
  y = x + y;
}
```

Answer: $wp(\textbf{“if ...”}, x > y)$

= $p \Rightarrow wp(\textbf{“x=x+1”}, x > y)$ ***AND*** $\neg p \Rightarrow wp(\textbf{“y=x+y”}, x > y)$

= $p \Rightarrow x+1 > y$ ***AND*** $\neg p \Rightarrow x > x+y$

$wp(\textbf{“p = x>y”}, p \Rightarrow x+1 > y$ ***AND*** $\neg p \Rightarrow x > x+y)$

= $x > y \Rightarrow x+1 > y$ ***AND*** $\neg(x > y) \Rightarrow (x > x+y)$

= *true* ***AND*** $\neg(x > y) \Rightarrow y < 0$

= $\neg(x > y) \Rightarrow y < 0$

= $x > y$ ***OR*** $y < 0$

Questions 29 and 30 ask you to prove that this this routine is correct. In other words, you will prove that it terminates with a correct answer.

There is another copy of the code on page 15 that you may tear off.

```
// requires: a is non-null, non-empty, and sorted in increasing order
// requires: val is an element in a
// returns: the index of val in a
int binarySearch(int[] a, int val) {
  int min = 0;
  int max = a.length - 1
  while (min < max) {
    int mid = (min + max) / 2;
    if (val == a[mid]) {
      min = mid;
      max = mid;
    } else if (val > a[mid]) {
      min := mid + 1;
    } else { // val < a[mid]
      max := mid - 1;
    }
  }
  return max;
}
```

29. (6 points) Prove that the `binarySearch` routine (which appears on pages 9 and 15) terminates.

    Use $\texttt{max} - \texttt{min}$ as the decrementing function.
    If it becomes zero or less, then the loop exits.
    Note that $\texttt{min} \leq \texttt{mid} \leq \texttt{max}$.
    Each of the branches reduces the decrementing function:

    - The first branch reduces it from non-zero to zero
    - The second branch reduces it by increasing `min`.
    - The third branch reduces it by decreasing `max`.

    Therefore, the loop exits.

30. (12 points) Prove that the `binarySearch` routine (which appears on pages 9 and 15) gives a correct answer, if it terminates. This property is called "partial correctness".

    The loop invariant LI has two parts:

    - $\texttt{val} \in \texttt{a[min..max]}$; that is, `val` is an element of the subarray of `a` from indices `min` to `max`, inclusive. Equivalently, $\texttt{a[min]} \leq \texttt{val} \leq \texttt{a[max]}$
    - $\texttt{min} \leq \texttt{max}$

    Another way to state the LI is: $\exists i.\texttt{min} \leq i \leq \texttt{max} \wedge \texttt{a}[i] = \texttt{val}$

    A partial correctness proof has three parts:

    PRE $\Rightarrow$ LI
    - The first part of the LI is true because $\texttt{val} \in \texttt{a}$ and $\texttt{a} = \texttt{a[min..max]}$.
    - The second part of the LI is true because `a` is non-empty.

    LI $\wedge$ pred {loop body} LI
    - The first part of the LI still holds, even though `min` and `max` have changed, because $\texttt{a[min]} \leq \texttt{val}$ and $\texttt{val} \leq \texttt{a[max]}$ and therefore $\texttt{val} \in \texttt{a[min..max]}$. (Note that `a` has not changed.)
    - The second part of the LI still holds, because $\texttt{min} \leq \texttt{mid} \leq \texttt{max}$.

    LI $\wedge$ $\neg$pred $\Rightarrow$ POST $\texttt{min} = \texttt{max} \wedge \texttt{val} \in \texttt{a[min..max]} \Rightarrow \texttt{val} = \texttt{a[max]}$

31. (10 points) Consider the `PriorityQueue.peek` method:

```
class PriorityQueue<E>
  ...
  // Retrieves, but does not remove, the head of this queue,
  // or returns null if this queue is empty.
  public @Nullable E peek() { ... }
}
```

Suppose you want to use it in a fashion like this:

```
int myMethod(PriorityQueue<Date> myQueue) {
  return myQueue.peek().getMonth();
}
```

and you want a compile-time guarantee that no `NullPointerException` will be thrown — despite the fact that the documentation of `peek` states that an exception *might* be thrown, it will not in your circumstance.

Describe how you could augment the type system to give this guarantee. Choose two relevant methods of `PriorityQueue`, and state how the modified type-checker would treat them.

You need to introduce a parallel type qualifier hierarchy, exactly analogous to the `@KeyFor` hierarchy that is used to augment the Nullness Checker: while `@Nullable` and `@NonNull` are related to one another, they are not related to `@KeyFor`.

The new annotation could be named `@NonEmpty`, indicating that the queue is non-empty.

Type-checking of `peek` would know that if the argument is `@NonEmpty`, then the result is `@NonNull`.

Type-checking of `add` would know that the result is `@NonEmpty`.

5 Code examples

32. (8 points) In Java, `Integer[]` is a subtype of `Number[]`. Assume the following declarations:

```
Number n;
Number[] na;
Integer i;
Integer[] ia;
```

Give code that passes the type-checker but is type-incorrect with respect to true subtyping. Use a maximum of 4 statements (fewer is possible and preferable).

```
na = ia;
na[2] = 3.14;
i = ia[2];
```

In one sentence, what does your code do at run time?

Throws a run-time exception when* storing *a type-incorrect value into the array. No check happens when* accessing *an array element.

For the next 4 questions, assume you have this code:

```
class ListNode {
  Object data;
  ListNode next;
  void m(ListNode arg) {
    ...
  }
}
```

33. (2 points) Write a method body for `m` that performs assignment but not mutation to `arg`. (2 lines of code max; the code does not have to do anything sensible, but should not use bad style.)

    `arg = null;`

34. (4 points) How could you prevent such an assignment from occurring, in standard Java? What would happen if a programmer wrote, compiled, and ran such code? (1 sentence each)

    Add `final` to the declaration, and the code will be flagged as a compile-time error.

35. (2 points) Write a method body for `m` that performs mutation but not assignment to `arg`. (2 lines of code max; the code does not have to do anything sensible, but should not use bad style.)

    `arg.data = null;`

36. (4 points) How could you prevent such mutation from occurring, in standard Java? What would happen if a programmer wrote, compiled, and ran such code? (1 sentence each)

    Pass in an immutable object that throws an exception when a mutating method is called. An exception would occur at run time.

    Or, change the type to an immutable type so the method call is flagged as a compile-time error.

6 Design patterns

37. (10 points) Why can interning be applied only to immutable classes? (Answer in no more than 3 sentences.)

    The* implementation *of interning requires that the equals method uses abstract value equality; otherwise, a previously-constructed value would not be found. But,* clients *of interning require that the equals method uses behavioral, or eternal, equality, so that substitution of an interned value for a non-interned one is guaranteed to give the same behavior. These are compatible only for immutable classes.

38. (10 points) The procedural pattern contains uses of `instanceof`. For example, we saw this in lecture:

```
// Example of procedural pattern:  typechecking a programming language expression
Type tcExpression(Expression e) {
  if (e instanceof PlusOp) {
    return tcPlusOp((PlusOp)e);
  } else if (e instanceof VarRef) {
    return tcVarRef((VarRef)e);
  } else ...
```

    The visitor pattern achieves the same purpose, without any uses of instanceof. Why not? That is, what happened to them?

    They were replaced by dynamic dispatch, which selects the appropriate implementation of the `accept` method based on the type of the currently-visited element.

This is a duplicate of the code that appears on page 9 and is used in questions 29 and 30. You may tear it off if you find that convenient. You do not need to hand it in.

```
// requires: a is non-null, non-empty, and sorted in increasing order
// requires: val is an element in a
// returns: the index of val in a
int binarySearch(int[] a, int val) {
  int min = 0;
  int max = a.length - 1
  while (min < max) {
    int mid = (min + max) / 2;
    if (val == a[mid]) {
      min = mid;
      max = mid;
    } else if (x > a[mid]) {
      min := mid + 1;
    } else { // x < a[mid]
      max := mid - 1;
    }
  }
  return max;
}
```