

Massachusetts Institute of Technology
6.170 Laboratory in Software Engineering
Spring 2005

Quiz 2

Tuesday, April 12, 2005

Name: Solutions _____

Athena username: _____

Recitation section (circle one):

1: Rui Viana 2: Joy Forsythe 3: Ben Leong

4: Rohit Rao 5: Jesse Smithnosky 6: Amy Williams

This quiz is closed book, closed notes. You have **50 minutes** to complete it. It contains 25 questions and 10 pages (including this one), totalling 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 and recitation section number on the top of ALL pages.**

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

Page	Max	Score
2	14	
3	12	
4	18	
5	23	
6	6	
7	16	
8	11	
Total	100	

1 True/False

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

1. T/F Decentralized organizational models work best for large groups. **False**
2. T/F When a project starts to slip behind schedule, adding more people is usually your best hope for recovery. **False**
3. T/F Bottom-up testing is not an effective way to reveal architectural design flaws. **True. It may reveal such flaws, but isn't an effective way to do so. Top-down testing is an effective way to reveal architectural design flaws.**
4. T/F Top-down testing can be time-consuming, because you need to write stubs for many modules. **True. Top-down testing tends to be more time-consuming than bottom-up, given that you need to write unit tests anyway, and they effectively perform bottom-up testing on their own.**
5. T/F One drawback of bottom-up system testing (as compared to top-down system testing) is that when a test fails, there are more places you have to look for the bug. **True. Unless unit testing was perfect, the bug might appear in any class that was executed as part of the test.**
6. T/F A prototype is an initial implementation of partial functionality that can later be expanded into a full product. **False. Prototypes are intended to answer a specific question, then to be thrown away.**
7. T/F Design patterns typically increase the amount of code that needs to be written to accomplish a specific purpose. **True. They more often increase the generality or reusability of the code, not its brevity.**

2 Multiple choice

8. (4 points) Ben Bitdiddle wrote an ADT called `Period` that represents time intervals. He is feeling overworked, and so would like to have a more relaxed attitude to time. He makes an `equals` method for `Period` so that two `Periods` are considered equal if they are within 1 hour of each other. Circle any true implications of this:

- (a) Equality is not reflexive for `Period`.
- (b) Equality is not symmetric for `Period`.
- (c) Equality is not transitive for `Period`.
- (d) `Period`'s `hashCode` method would have to return a constant.
- (e) None of the above.

c, d

9. (4 points) Ben changes his `equals` method for `Period` so that two `Periods` are equal if they round to the same number of hours. Circle any true implications of this:

- (a) Equality is not reflexive for `Period`.
- (b) Equality is not symmetric for `Period`.
- (c) Equality is not transitive for `Period`.
- (d) `Period`'s `hashCode` method would have to return a constant.
- (e) None of the above.

e. This equals method works fine.

10. (4 points) Recall that `ArrayList`, `LinkedList`, and `Vector` all implement `List` but are not related to one another by subclassing or inheritance. Further recall that Java specifies `List`'s `equals` method to use observational, not behavioral, equivalence. Consider the following code:

```
ArrayList<Integer> al = new ArrayList<Integer>();
LinkedList<Integer> ll = new LinkedList<Integer>();
... // arbitrary code (with no compile-time or run-time errors)
Vector<List<Integer>> v = new Vector<List<Integer>>();
v.add(al);
v.contains(ll);
```

What result is returned by the last method call, “`v.contains(ll)`”? Choose the best answer.

- (a) Compile-time error
- (b) Run-time error
- (c) True
- (d) False
- (e) Either true or false

e. If the elided code adds non-equals elements to `al` and `ll`, then the result is false. If the elided code is empty, then the result is true.

3 Short answer

11. (3 points) How fast does a computer response need to be in order to feel instantaneous?
100 ms (or less)

12. (3 points) Java 5 eliminates much of the need for casting in Java programs. Name one use of casting that remains acceptable and necessary in Java 5, according to 6.170.

Casting the Object argument of the equals method. Less commonly, resolving overloading (including of numeric operations).

13. (4 points) Reasoning by structural induction about **uses** (clients) of an **immutable** ADT requires use of a base case and an inductive case. State the typical base case, and the typical inductive case.

(a) base: *constructors*

(b) inductive: *producers*

An example of such reasoning is a proof that, in client code, all visible Lists are sorted.

14. (4 points) Inductive reasoning about the **implementation** of a **mutable** ADT must consider all possible changes to the representation. Mutators are one way that the representation can change. Give two other conceptually distinct ways that it could change.

(a) *observers, producers via benevolent side effects*

(b) *representation exposure*

15. (4 points) Name the two key advantages of factory methods, when compared to constructors.

(a) *Can return an existing object*

(b) *Can return a subtype of the declared type*

Being able to choose a name for the factory method (whereas all constructors are required to have the same name) is an advantage, but a minor one compared to the two noted above.

16. (5 points) Place one X in each column of the table below, indicating which of the two toolkits is superior with respect to the given metric, according to Michael Bolin's lecture.

	completeness	API ease of use	performance	look-and-feel consistency	portability
Swing/AWT		X			X
SWT	X		X	X	

17. (4 points) For physical objects, maintenance is required to repair the effects of wear and tear. For non-buggy software, what is the most frequent cause that requires "maintenance"? Answer with no more than one sentence.

Use of the software in a new environment for which it was not originally designed, but in which it is desired to be used.

Partial credit for specific examples of this. The best such example is new user requirements. (That does not cover all cases, because users do not usually think of the software they depend on, such as the format of results from a given website, as part of their requirements.) New features are another good example. Operating system and programming language upgrades are rare, and they do not necessarily require software to be changed.

18. (4 points) What is "mythical" about the "mythical man-month"? Answer with no more than one sentence.

Some acceptable answers are as follows. When people are added to a project, the completion time does not go down proportionately. Man-months are not fungible (interchangeable), so it is misleading to treat them as such by measuring a task only in man-months. It may be possible for one person to accomplish in a year what 12 people cannot accomplish in one month, and 365 people cannot accomplish in one day.

This question was not about the fact that different people have different productivity, or about the difficulty of estimating the time that a project will require.

19. (6 points) Indicate for each of the following types of wrappers whether the functionality is the same or different, and whether the interface is the same or different (when compared to the wrapped class). That is, put "same" or "different" in each box, choosing the best answer for each box.

	functionality	interface
adapter	<i>same</i>	<i>different</i>
decorator	<i>different</i>	<i>same</i>
proxy	<i>same</i>	<i>same</i>

20. (4 points) Suppose that you have (correctly) proved that the implementation of an ADT properly establishes and maintains its representation invariant. Give two distinct reasons that it is still a good idea to use a `checkRep` method.
- (a) *The code may be modified in the future.*
 - (b) *Representation exposure might cause the representation invariant to be violated.*
 - (c) *A method might be given code that does not satisfy its requires clause, which in turn causes that method to break the rep invariant. (To get any credit, you had to specify that the invariant could be broken. Simply saying “Unforeseen input” was not enough.)*
 - (d) *“You told us we always had to” was amusing enough for one point.*

4 Design

21. (6 points) Consider an immutable `Point` class, where two `Points` are equal if their `x` and `y` coordinates match.

```
class Point {
    private final int x;
    private final int y;
    ...
}
```

In this question, you will write three `hashCode` methods.

- (a) Give a `hashCode` implementation that can result in incorrect behavior for clients of `Point`.

```
int hashCode() {
    return return super.hashCode();
}
Or: int hashCode() { return System.identityHashCode(this); }
Or: int hashCode() { return (int) Math.random(); }
```

- (b) Give a `hashCode` implementation that can result in inefficient, but not incorrect, behavior for clients of `Point`.

```
int hashCode() {
    return 0;
}
Or: int hashCode() { return x + y; }
```

- (c) Give a good `hashCode` implementation—for instance, one such as that recommended by Bloch in *Effective Java*.

```
int hashCode() {
    return x + 37 * y;
}
```

22. (8 points) Recall the Visitor design pattern, which was discussed in lecture.
- (a) What sort of program change does the Visitor pattern make easy? Why is it easy?
The Visitor pattern makes it easy to add new operations, because all the code for handling all types goes in a single new class.
 - (b) What sort of program change does the Visitor pattern make difficult? Why is it difficult?
The Visitor pattern makes it difficult to add new types (objects), because each visitor (of which there may be many) must be modified to handle the new type of object. The amount of code to be added is the same as if some other design pattern such as Interpreter were used, but many different classes must be modified.
23. (8 points) Some applications supply access to common operations via context-sensitive menus (right-click menus).
- (a) State a reason that context menus may be superior to standard menus.
Users may have faster access (less mouse movement) to access the menu items, especially if the regular menu bar is not at the very top of the screen.
 - (b) State a reason that context menus may be inferior to standard menus.
Context-sensitive menus may be unintuitive to users who do not expect to right-click in order to reveal possible operations; regular menus are more standard and thus may be more intuitive.
Context menus may pop up in unexpected places, when the mouse is clicked near the edge of a window or the screen so that the usual location is not available.
(Menus that force the user to move the mouse through a “tunnel” in order to get to submenu items are problematic, but that can be just as much of a problem with regular menus. It is not specific to context menus.)

5 Reasoning

24. (5 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: $wp(\text{"if ..."}, x > y)$
 $= p \Rightarrow wp(\text{"x=x+1"}, x > y) \text{ AND } \neg p \Rightarrow wp(\text{"y=x+y"}, x > y)$
 $= p \Rightarrow x + 1 > y \text{ AND } \neg p \Rightarrow x > x + y$
 $wp(\text{"p = x>y"}, p \Rightarrow x + 1 > y \text{ AND } \neg p \Rightarrow x > x + y)$
 $= x > y \Rightarrow x + 1 > y \text{ AND } \neg(x > y) \Rightarrow (x > x + y)$
 $= \text{true AND } \neg(x > y) \Rightarrow y < 0$
 $= \neg(x > y) \Rightarrow y < 0$
 $= x > y \text{ OR } y < 0$

25. (6 points) Consider the following brief routine.

```
/** Requires:  in != null.
 * Returns:   the reverse of its argument.
 **/
List reverseList(List in) {
    List remaining = new ArrayList(in);
    List out = new ArrayList();
    while (! remaining.isEmpty()) {
        // removes first element of remaining, add it to the beginning of out
        out.add(remaining.remove(0), 0);
    }
    return out;
}
```

Give a loop invariant and a decrementing function for its loop, assuming the goal is to prove total correctness.

Loop invariant: $in = \text{concat}(\text{reverse}(\text{out}), \text{remaining})$, where **“concat” is list concatenation and “reverse” is list reversal**

Decrementing function: $\text{length}(\text{remaining})$