

Massachusetts Institute of Technology  
6.170 Laboratory in Software Engineering  
Spring 2005

## Quiz 2

Tuesday, April 12, 2005

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

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 8 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.
2. **T/F** When a project starts to slip behind schedule, adding more people is usually your best hope for recovery.
3. **T/F** Bottom-up testing is not 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.
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.
6. **T/F** A prototype is an initial implementation of partial functionality that can later be expanded into a full product.
7. **T/F** Design patterns typically increase the amount of code that needs to be written to accomplish a specific purpose.

## 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.
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.
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

### 3 Short answer

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

\_\_\_\_\_

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.

\_\_\_\_\_

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: \_\_\_\_\_

(b) inductive: \_\_\_\_\_

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) \_\_\_\_\_

(b) \_\_\_\_\_

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

(a) \_\_\_\_\_

(b) \_\_\_\_\_

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<br>ease of use | performance | look-and-feel<br>consistency | portability |
|-----------|--------------|--------------------|-------------|------------------------------|-------------|
| Swing/AWT |              |                    |             |                              |             |
| SWT       |              |                    |             |                              |             |

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.

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

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

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

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   |               |           |
| decorator |               |           |
| proxy     |               |           |

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) \_\_\_\_\_
- (b) \_\_\_\_\_
- (c)
- (d)

## 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() {  
    _____  
  
}
```

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

```
int hashCode() {  
    _____  
  
}
```

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

```
int hashCode() {  
    _____  
  
}
```

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?

---

---

---

(b) What sort of program change does the Visitor pattern make difficult? Why is it difficult?

---

---

---

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.

---

---

---

(b) State a reason that context menus may be inferior to standard 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: \_\_\_\_\_

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: \_\_\_\_\_

Decrementing function: \_\_\_\_\_