

Massachusetts Institute of Technology
6.170 Laboratory in Software Engineering

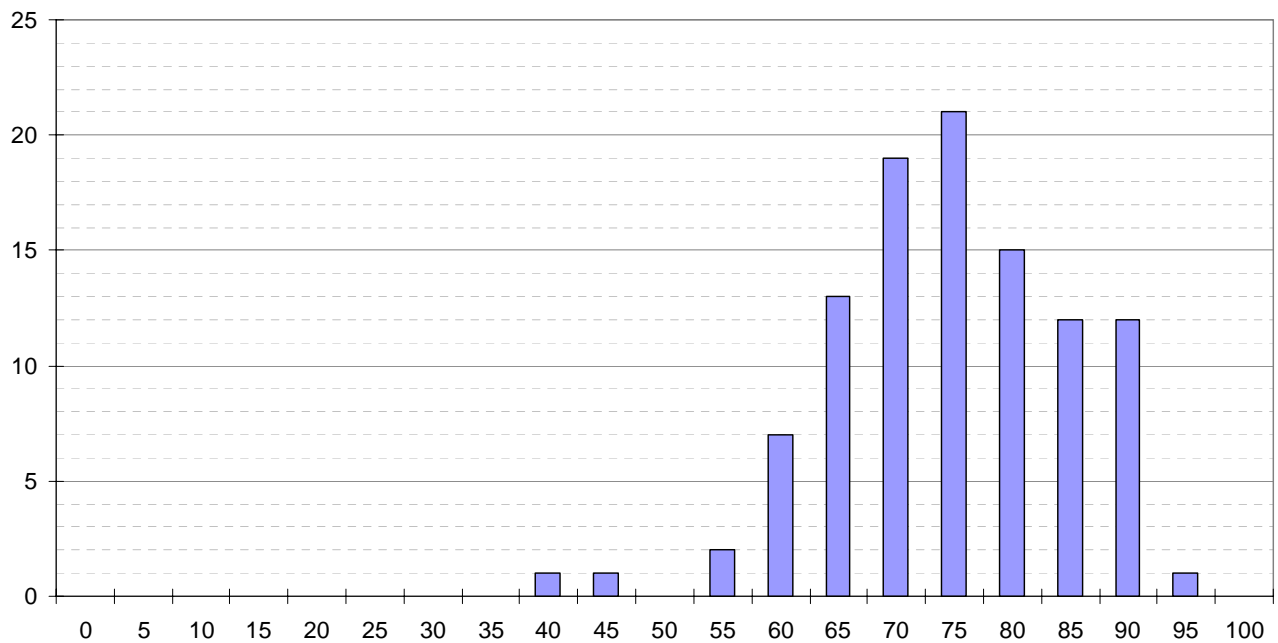
Spring 2006

Quiz 2

Friday, April 14, 2006

Number of Students	104
Mode	74
Median	73
Mean	72.596
Standard Deviation	10.275

Histogram



Massachusetts Institute of Technology
6.170 Laboratory in Software Engineering

Spring 2006

Quiz 2

Friday, April 14, 2006

Name: Ben Bitdiddle

Athena username: bit

Section (circle one):

- 1: Matthew Tschantz (10am) 2: Vincent Yeung (11am) 3: C. Scott Ananian (12pm)
4: Philip Guo (1pm) 5: Tucker Sylvestro (3pm) 6: Natan Cliffer (3pm)

This quiz is closed book, closed notes. You have 50 minutes to complete it. It contains 32 questions in 14 pages (including this one), totaling 100 points. Before you start, please check your copy to make sure it is complete.

Write your initials and section number on the top of ALL pages.

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

Good luck!

Question(s)	Grading Scheme	Total	Max
1 to 14	1 to 5 6 to 10 11 to 14	X 2 =	28 of 28
	5 5 4		
15 to 28	15, 16 17, 18 19, 20 21 22, 23 24, 25 26, 27 28	X 3 =	42 of 42
	2 2 2 1 2 2 2 1		
29	3	X 1 =	3 of 3
30 and 31	30 9 of 9 + 31 5 of 5	=	14 of 14
32 and 33	32 5 of 5 + 33 8 of 8	=	13 of 13
			100 of 100

True/False Questions (2 points each)**Question 1.**

Usability is important for simple programs, but for the majority of commercial software applications targeted towards home users, polls show that users value ease of making feature requests and bug reports over ease of use.

T / F

Question 2.

Fitt's law states that the user should never be to blame when (s)he cannot perform a routine task such as saving a file.

T / F

Question 3.

A user interface that looks aesthetically beautiful may actually be bad in terms of usability.

T / F

Question 4.

The spiral model of user interface design involves spiraling between implementing a design, releasing and selling it to consumers, receiving bug reports, re-designing and re-implementing a new version, releasing an upgraded version to consumers, and repeating.

T / F

Question 5.

A rule of thumb for complex projects is that the majority of your team's time between inception and release should be spent writing code to add new features.

T / F

Question 6.

For large projects, test drivers requiring human intervention are always preferable over automated test frameworks because they can often test subtle features that are difficult to automate.

T / F

Question 7.

One advantage of running regression tests every night is that when new code is checked-in with some subtle bug that affects another unrelated part of the system, the programmer can be notified the next day when the change he/she just made is still fresh in his/her mind.

T / F

Question 8.

The person who reports a bug should always be the person who fixes it because, after all, he/she knows the bug the best.

T / F

Question 9.

According to the Liskov text, The equals() method should be used for "eternal" equality, while "similar()" should be used to determine if objects are equal "at this moment".

T / F

Question 10.

Mutable types should always override the default equals() method.

T / F

Question 11.

For mutable classes, behavioral equivalence and observational equivalence are the same thing.

T / F

Question 12.

A weakly coupled system will have more edges in its module dependency graph.

T / F

Question 13.

A callback, as used by the Observer Pattern, is a way to reverse a dependency edge.

T / F

Question 14.

The usability guideline of consistency suggests that a program should use only one of the words "delete", "erase", and "rename". The usability guide of matching the real world suggests that all of these should be acceptable, since any of them may be the word that is most natural to the user or the problem domain. True or false: the consistency guideline should take precedence, in most cases.

~~**T / F**~~

Due to the ambiguity of this question, it was not graded.

Multiple Choice Questions (circle all that apply, 3 points each, no partial grade):**Question 15.** (circle all that apply, no partial grade)

Thinking about Fitt's Law, please rate these mouse movement and click tasks from **slowest** to **fastest**, assuming that the mouse starts at the center of the screen:

1. Moving the mouse to the top of the screen and clicking on a menu entry to activate a pull-down menu (e.g., Mac OS menu bar)
2. Right clicking the mouse to pop-up a one-level menu, selecting the menu entry and clicking it.
3. Moving the mouse to the upper-right corner of a window that has been almost fully-maximized but does not stretch completely to the corners of the screen and clicking a button located there (e.g., Microsoft Windows 'Close Application' button on the upper-right corner of every window)

Answer is:

- (A) 1, 2, 3 **1 is clearly faster (infinite depth rectangle). However, it is not clear right clicking a single level menu (rectangle enclosing the menu name) is faster or slower than 3 (another rectangle enclosing the menu name).**
- (B) 1, 3, 2
- (C) 2, 3, 1
- (D) 3, 2, 1

Full points given to: C or D or C & D

Question 16. (circle all that apply, no partial grade)

Which are examples of defensive programming?:

- (A) Ben Bitdiddle makes sure his specifications are clear and explicit before he starts writing code.
- (B) Alyssa P. Hacker writes assert statements into her code when her code relies on nontrivial assumptions.
- (C) Java the Hutt, when possible, creates copies of mutable arguments passed to his constructors before storing them to fields.
- ~~(D) Lisa N. Gender sets up a system that runs a regression test suite on the project's code every hour, and emails the developer list about any test that fails when it once passed.~~

While it can be considered that regression testing help defensive programming, there was no explicit discussion about this relationship.

Question 17. (circle all that apply, no partial grade)

Which are possible pitfalls of the "Prototyping/Incremental" model of system development?

- (A) You might be locked in to a design that is not appropriate for the customer
- (B) Acceptance testing might not be able to be started until too late in the development cycle
- (C) It might be too tempting not to throw away the prototypes.
- (D) The "second system effect" may lead to inaccurate conclusions about the final system.

Question 18. (circle all that apply, no partial grade)

Suppose S is the statement:

`if b S1 else S2`

How can we write $wp(S, Q)$ in terms of $wp(S1, Q)$ and $wp(S2, Q)$?

- (A) $(b \rightarrow wp(S1, Q)) \wedge (\neg b \rightarrow wp(S2, Q))$
- (B) $(b \rightarrow wp(S1, Q)) \vee (\neg b \rightarrow wp(S2, Q))$
- (C) $(b \wedge wp(S1, Q)) \wedge (\neg b \wedge wp(S2, Q))$
- (D) $(b \wedge wp(S1, Q)) \vee (\neg b \wedge wp(S2, Q))$
- (E) None of the above

Full points given to: A or D or A & D

where \neg , \rightarrow , \wedge and \vee denote logical complementation, implication, conjunction, and disjunction, respectively.

The answer is A and D. A is the direct answer. D is equivalent to A. However, 6.170 is not about testing your ability to manipulate boolean formulas. Furthermore, some of the staff did not get D. So points were given to any combination of A and D.

Question 19. (circle all that apply, no partial grade)

Suppose S is the following while loop statement:

```
// P: assert x >= 0 && y = 0
while (x != y) {
    x = x - 1;
}
// Q: assert x=y
```

Which of the following is (are) suitable loop invariants for showing $P \{S\} Q$? We are only interested in partial correctness in this question.

- (A) $x = y$
- (B) $x \geq y$
- (C) $x \leq y$
- (D) $y = 0$
- (E) None of the above

Full points given to: B or D or B & D or E

The bug in this question is "partial correctness", which is ill-defined. A and C are not loop invariants, thus wrong. However, B and/or D can be used to prove loop termination. And E on some interpretations of "partial correctness"

Question 20. (circle all that apply, no partial grade)

Which of the following statements are true regarding A* search, such as the one you implemented for Problem Set 4?

- (A) Adding an A* heuristic produces more optimal (i.e. shorter) paths than without it.
- (B) Adding an A* heuristic can reduce the computation time of finding a path.
- (C) To guarantee that an optimal (shortest) path is found, A* search requires that the heuristic used never overestimates the actual remaining shortest path distance to the goal.
- (D) To guarantee that an optimal (shortest) path is found, A* search requires that the heuristic used never underestimates the actual remaining shortest path distance to the goal.

Gill Bates is attempting to prove that the rep invariant holds for the following code:

```
/**
 * BugLocations represents the set of line numbers containing
 * bugs in a release of the Doors operating system.
 */
public class BugLocations {

    public Set<Number> elts;

    // RI: all elements of elts are > 0.

    public BugLocations() {
        elts = new Set<Number>();
    }

    public boolean contains(Number n) {
        return elts.contains(n);
    }

    public BugLocations copy() {
        BugLocations other = new BugLocations();
        other.addAll(this);
        return other;
    }

    public void add(Number n) {
        elts.add(n);
    }

    public void addAll(BugLocations bl) {
        for (Number n : bl.elts) {
            add(n);
        }
    }
}
```

Question 21. (circle all that apply, no partial grade)

Using the approach taught in lecture, which of the following methods/constructors would be treated as base cases in an inductive proof of rep invariant:

- (A) BugLocations
- (B) contains
- (C) copy
- (D) add

Question 22. (circle all that apply, no partial grade)

Can rep invariant be proved to hold?

- (A) Yes
- (B) No, due to copy
- (C) No, due to add
- (D) No, due to rep exposure

Question 23. (circle all that apply, no partial grade)

Gill Bates is being flooded with bug reports, so he decides to add concurrency to his code. The following is code executed from two threads. The order of execution is not known. The variable "bugs" points to the same instance of `BugLocations` in both threads.

```
Thread 1          Thread 2
bugs.add(3);      bugs.add(3);
bugs.remove(3);   bugs.remove(3);
bugs.contains(3);
```

Which of the following are possible outcomes of executing thread 1 and thread 2 concurrently:

- (A) Thread 1 and 2 enter deadlock
- (B) Thread 1 experience starvation
- (C) `bugs.contains(3)` returns **true**
- (D) `bugs.contains(3)` returns **false**

Question 24. (circle all that apply, no partial grade)

Under increasing pressure, Gill Bates decides that he should use synchronization and two instances of BugLocation, bugs1 and bugs2. He rewrites the threads as follows:

```
Thread 1          Thread 2
synchronized(bugs1) {      synchronized(bugs2) {
  synchronized(bugs2) {    synchronized(bugs1) {
    bugs1.add(3);           bugs2.add(3);
    bugs1.remove(3);       bugs2.remove(3);
  }                         }
}                            }
bugs1.contains(3);
bugs2.contains(3);
```

Which of the following are possible outcomes of executing thread 1 and thread 2 concurrently:

- (A) Thread 1 and 2 enter deadlock
- (B) Thread 1 experience starvation
- (C) bugs1.contains(3) returns **true**
- (D) bugs2.contains(3) returns **false**

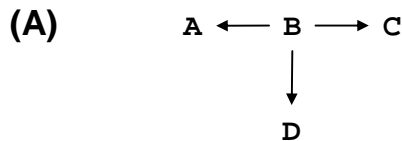
Question 25. (circle all that apply, no partial grade)

Which if these hash codes would be best for a Date object:

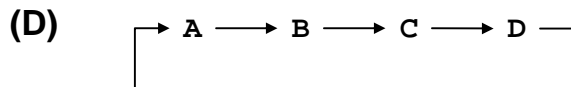
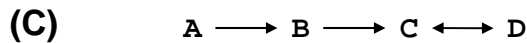
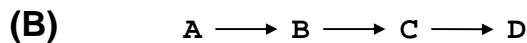
- (A) return 1
- (B) return year + 7*month + 23*day_of_the_month
- (C) return (year + 7*month + 23*day_of_the_month) % 11
- (D) return new Random().nextInt()

Question 26. (circle all that apply, no partial grade)

Which of these module dependency diagrams most likely depicts the best design?



The best answer is B. However, A is also close because modules A, C and D are independent.



Full points given to A or B

Question 27. (circle all that apply, no partial grade)

Circle all of the below that are advantages of sketching a proposed user interface (as opposed to using a drawing program or building a prototype).

- (A) It is easier to make small changes to a large drawing
- (B) It is faster
- (C) It looks less polished to users, encouraging them to suggest changes
- (D) It makes it easier to ignore irrelevant details

Question 28. (circle the best choice)

Ordinarily, the code "myString1 == myString2" is a likely bug. Which design pattern makes this a performance optimization, without any chance of error?.

(A) flyweight

(B) interning

(C) prototype

~~(D) singleton~~

The answer is B and D. However, the use of the variable name myString implies that the variable is a string, which is unlikely to be a singleton. Thus D was not graded.

Match the Words Question**Question 29.** (3 points)

```
public boolean transfer(Account fr, Account to, int val) {
    <BODY ?>
}
```

BODY 1

```
synchronized(fr) {
    if(fr.getbal() > val) {
        fr.post(-val);
    } else {
        throw new Exception();
    }
}
synchronized(to) {
    to.post(val);
}
```

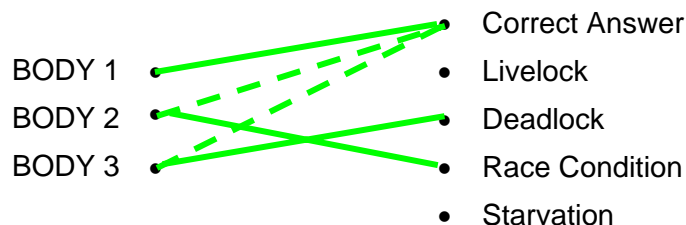
BODY 2

```
int fv;
synchronized(fr) {
    fv = from.getbal();
}
if(fv > val) {
    synchronized(fr) {
        fr.post(-val);
    }
} else {
    throw new Exception();
}
synchronized(to) {
    to.post(val);
}
```

BODY 3

```
synchronized(fr) {
    if(fr.getbal() > val) {
        fr.post(-val);
    } else {
        throw new Exception();
    }
}
synchronized(to) {
    to.post(val);
}
```

Draw lines connecting the dots for the three different candidate bodies to indicate which property holds for that body. Note that the connectivity can be one-to-many.



The bug in the question is "one-to-many", which should have been "many-to-one". A lot of students got confused by this and assumed that since some executions produce a correct answer even with deadlock and race conditions it should also be included.

Short Answer Questions

Question 30. (9 points)

In no more than 10 words each, list three distinct advantages of factory methods over constructors.

Many valid answers including:

- * It can return objects of any subtype, not just the declared type
- * It can return null
- * It can return existing objects, not just new ones
- * It can be easily replaced by another factory, at run time or compile
- * It can be used to intelligently choose which subtype to return

Question 31. (5 points)

Is `List<Integer>` a behavioral subtype of `List<Number>`? YES / **NO**

In 1 sentence, explain why or why not.

`List<Integer>` does not support the operation `put(Float)`, but `List<Number>` does.

Question 32. (5 points)

Is List<Number> a behavioral subtype of List<Integer>?

YES / **NO**

In 1 sentence, explain why or why not.

List<Integer> does not support the operation put(Float), but List<Number> does.

Question 33. (10 points)

The path interfaces of the problem set sequence used immutable lists. In no more than 2 sentences each, give two advantages of using an immutable, rather than a mutable, datatype for paths.

The extend method is efficient (and it gets called a lot!): there is no need to copy the path that is being extended.

Changing an element in a Priority Queue (or certain other collection data structures) can make it impossible to find that object again. If the object is immutable, it is easy to reason that it was not changed while in the collection.

END OF QUIZ 2!