

Massachusetts Institute of Technology
6.170 Laboratory in Software Engineering
Spring 2005

Quiz 1

Wednesday, March 2, 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 27 questions and 12 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!

Section	Max	Score
1	26	
2	7	
3	28	
4	14	
5	25	
Total	100	

1 True/False

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

1. T/F The representation of an immutable object always includes at least one immutable field.
False. The fields can be mutable. They might simply never be changed, or observer methods might have benevolent side-effects.
2. T/F If A is a true subtype of B, then any instance of A satisfies B's specification.
True.
3. T/F Java guarantees that the run-time type of an object is a true subtype of the declared type of any references to that object.
False. It only guarantees that the run-time type is a Java subtype. The full specifications are not checked.
4. T/F One of the benefits of glass box testing is that test cases can be generated immediately after the specification is written and before any code is developed.
False. This is a benefit of black box testing, not glass box testing.
5. T/F A rep invariant should be part of the specification of every non-trivial class.
False. They are implementation dependent.
6. T/F The abstraction function for an ADT should be independent of its representation.
False. Abstraction functions map from representation to abstract value, and so are necessarily representation-dependent.
7. T/F Covariant return types in Java are incompatible with true subtyping.
False.
8. T/F If we design tests that exercise every line of code in a method, we might still fail to discover all bugs.
True.
9. T/F A test suite should never test the behavior of a method when its requires clause is unsatisfied.
True. If a valid but different implementation were substituted the tests might fail.
10. T/F In Java, both `String` and `StringBuilder` implement a Java interface called `CharSequence`. Therefore, `String` and `StringBuilder` objects can be freely substituted for each other anywhere in a program.
False.

11. **T/F** A change to the representation always changes the abstract value.
False.
12. **T/F** Anytime that the abstract value changes, the representation must have also changed.
True.
13. **T/F** An observer method does not affect the representation.
False. It may, but it does not affect the abstract value.

2 Multiple choice

14. (3 points) Consider the following code:

```
class A {  
    Number foo(Object) { ... }  
}  
class B extends A {  
    Integer foo(Object) { ... }  
}
```

How is it determined which version of `foo` gets invoked by a particular line of code? (Circle one.)

- (a) resolved by overloading
- (b) resolved by dispatching
- (c) none of the above: compile-time error
- (d) none of the above: run-time error
- (e) none of the above: other

Resolved by dispatching. This is an example of covariant return types (the return type is more specific in the more specific class). However, the two methods are considered to be implementations of the same signature (they override, not overload) for the purposes of resolving the call.

15. (4 points) Which of the following are benefits of using an abstract data type? (Circle **all** that apply.)
- (a) Facilitates changes in the abstraction.
 - (b) Facilitates changes in the implementation.
 - (c) Reduces the communication overhead between developers of different parts of a system.
 - (d) Improves run-time performance.

b,c

3 Short answer

16. (6 points) Ben Bitdiddle says, “Ordinarily, an implementation should check the representation invariant on entry to and exit from each method and constructor in a class.” In 1–2 clauses each, give three distinct exceptions to this rule. (There are at least five acceptable answers.)
- (a) *static methods (no this)*
 - (b) *constructor entry (rep invariant need not hold)*
 - (c) *private helper methods (e.g., checkRep itself)*
 - (d) *in deployed code, if profiling has indicated that checking the rep invariant degrades performance unacceptably*
 - (e) *at the exit of trivial accessor methods (still check on entry, as rep exposure might have broken the rep). No credit for “if it doesn’t change the rep” without an explanation of how to determine that fact. No credit for “observer methods”, since they can change the rep.*

An argument about returning an immutable datatype is irrelevant to whether a method should be checked. An argument about the representation already being checked elsewhere is not acceptable without an explanation of how to easily check that property — and is usually not compelling, since checking the representation invariant is typically intended to detect errors that reasoning failed to uncover.

17. (6 points) In one sentence each, give three distinct software engineering benefits of using a version control system such as CVS. (There are at least five acceptable answers.)
- (a) *The version control system provides the ability to obtain older versions of the software, for instance to compare to the current one, or to revert to in the case of problems with newer versions. The version control system also stores comments describing what was changed in each version of the code.*
 - (b) *A single individual can conveniently work from multiple locations.*
 - (c) *Multiple individuals can coordinate their work, simultaneously editing the code for a program, a class, or even a method.*
 - (d) *The repository provides a backup in the event of data loss in the checkout (and vice versa).*
 - (e) *It is possible to “branch” the code to try experiments, without affecting the main line of development.*

18. (6 points) Recall that `Integer` is a subtype of `Number`. (In this question, the Java and true subtyping relationships are the same.)

(a) Is `HashSet<Integer>` a subtype of `HashSet<Number>`? Justify your answer in 1–2 sentences.

No. A subtype must satisfy the specification of the supertype. `HashSet<Number>.put` can accept any `Number` as an argument, but `HashSet<Integer>.put` cannot (its argument must be an `Integer`); thus, `HashSet<Integer>` cannot be used in place of `HashSet<Number>`.

(b) Is `HashSet<Number>` a subtype of `HashSet<Integer>`? Justify your answer in 1–2 sentences.

No. A subtype must satisfy the specification of the supertype. `HashSet<Integer>.get` promises to return `Integers` from `get`, but `HashSet<Number>.get` does not necessarily do so; thus, `HashSet<Number>` cannot be used in place of `HashSet<Integer>`.

No credit for circular answers such as “it is not a (Java) subtype” or “cannot be used in circumstances where the other is expected”. Furthermore, note that an argument about the relationship between `Number` and `Integer` is irrelevant, except to support an argument about methods of `HashSet<Number>` and `HashSet<Integer>`.

19. (3 points) Consider the following specification for method `getMostFrequent`:

```
// returns: the character that appears most frequently in str
// throws: NullPointerException if str==null
char getMostFrequent(String str);
```

This specification is not well-defined for all possible input. It is not clear what the method should return when `str` is the empty string `""`.

Give another input string for which the behavior of the method is not well-specified. Give as short an answer as possible.

“ab”. More generally, any string where two or more characters occur with equal frequency, so that “most frequently” is not well defined. (However, not all these strings are “as short as possible”).

20. (2 points) We could fix the problem with the `getMostFrequent` specification by adding a `requires` clause. Is this use of a strengthened `requires` clause more appropriate when `getMostFrequent` is public or when it is private? Circle one.

- (a) `getMostFrequent` is public (and in a public class).
- (b) `getMostFrequent` is a private helper method.

It is more suited to private helper methods. The representation invariant or other facts about the representation may guarantee that the precondition is satisfied. In the more general case, a requires clause may be a burden, depending on how the method is intended to be used.

21. (5 points) We could also fix the problem with a better `returns` clause. Write one, so that the specification is meaningful for all possible inputs.

Solution 1: Returns a character `ch` such that the frequency of all characters in `str` is less than or equal to the frequency of `ch` in `str`. (This solution handles the situation where there are multiple possibilities, and states that if string is empty, any character will do.)

Solution 2: Returns any character that appears most frequently in `str`, if one exists. If none exists, return the empty string.

4 Substitutability

22. (8 points) This page describes a class we want you to write. Read the instructions on this page, then write the required class on the following page.

Consider this interface:

```
// Represents a mathematical matrix
public interface Matrix {
    // returns: value of cell at specified row and column
    public int get(int row, int col);

    // effects: sets value of cell at specified row and column to val
    public void set(int row, int col, int val);
}
```

Assume there are two implementations of this interface available, called `ArrayMatrix` and `SparseMatrix`. As their names suggest, `ArrayMatrix` uses a direct representation of a matrix where each cell corresponds to an area of memory, while `SparseMatrix` uses a more space-efficient but potentially slower representation. Matrices are created by passing the size of the matrix to the constructor:

```
// Create a small matrix with 10 rows and 5 columns.
Matrix m1 = new ArrayMatrix(10,5);
// Create a huge matrix with 100000 rows and 50000 columns.
Matrix m2 = new SparseMatrix(100000,50000);
```

Your job is to write `SmartMatrix`, a third implementation of `Matrix`, which uses a direct representation if the number cells in the matrix is less than 10000, and uses a sparse representation otherwise. This class saves the client from having to make this decision:

```
// Create a small matrix with 10 rows and 5 columns.
// Uses an array representation.
Matrix m1 = new SmartMatrix(10,5);
// Create a huge matrix with 100000 rows and 50000 columns.
// Uses a sparse representation.
Matrix m2 = new SmartMatrix(100000,50000);
```

Write your code for `SmartMatrix` here. You need not provide specifications or comments. However, if writing `SmartMatrix` requires modification to the implementation of `ArrayMatrix` and/or `SparseMatrix`, you must note and justify this. Your solution will be evaluated on both correctness and style. Please think before you write, because revising your answer will make your exam less legible.

```
public class SmartMatrix implements Matrix {
    private Matrix m;
    public SmartMatrix(int row, int col) {
        if (row*col<10000) {
            m = new ArrayMatrix(row,col);
        } else {
            m = new SparseMatrix(row,col);
        }
    }
    public int get(int row, int col) {
        return m.get(row,col);
    }
    public void set(int row, int col, int val) {
        m.set(row,col,val);
    }
}
```

23. (6 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 |ey - x| <= 0.1`
- B `@return y such that |ey - x| <= 0.001`
 `@throws IllegalArgumentException if x <= 0`
- C `@requires x > 0`
 `@return y such that |ey - x| <= 0.001`
- D `@return y such that |ey - 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

5 Representations

Consider the following specification:

```
// CharSets are finite sets of lowercase letters
public class CharSet {
    // effects: creates a fresh, empty CharSet
    public CharSet();

    // modifies: this
    // effects: this_post = this_pre + {c}
    // throws: OutOfRangeException if Character is not a lowercase letter
    public void insert(Character c);

    // returns: c is an element of this
    boolean member(Character c);
}
```

Suppose we choose to represent CharSets with a string field `String str`. For efficiency, each pair of characters in `str` represent a character range. Here are some examples:

representation	abstract value
<code>str="adxz"</code>	<code>{a,b,c,d,x,y,z}</code>
<code>str="aacceg"</code>	<code>{a,c,e,f,g}</code>
<code>str=""</code>	<code>{}</code>

The representation for a particular set is chosen so that `str` is as short as possible, and the characters in the string are in increasing alphabetical order.

24. (8 points) Write a rep invariant based on this description. You may use math, clear English, Java code, or any mixture.

str != null

all chars should be lower case

length(str) is even

character codes at even indices are at least two greater than previous character (if exists)

each character code is at least as great as one before it

25. (8 points) Write an abstraction function for CharSet:

$$\{ch \mid \exists i . str[2 * i] \leq ch \leq str[2 * i + 1]\}$$

26. (4 points) Ben Bitdiddle has a `Signal` class for representing values coming from sensors. He wants to use it with a class that expects `Numbers`. So he changes his `Signal` class to be a subclass of `Number`:

```
class Signal extends Number {
    private double v = 0;
    public void update(double v) {
        this.v = v;
    }
    public double doubleValue() { return v; }
    public float floatValue() { return (float)v; }
    public int intValue() { return (int)v; }
    public long longValue() { return (long)v; }
    // No need to override byteValue or shortValue; they are not abstract.
}
```

This code compiles without a problem. The `Signal` class is a Java subtype of `Number`, but not a true subtype. Why?

Signal is mutable, while the specification of Number states that it is immutable. Thus, an instance of Signal cannot be used to replace an instance of Number. In general, a mutable type cannot be a true subtype of an immutable type.

27. (5 points) The existence of the `Signal` class can cause rep exposure. Briefly explain how, referring to the following code fragment:

```
class History {
    private List<Number> history = new LinkedList<Number>();
    private double total = 0;
    public void add(Number n) {
        history.add(n);
        total += n.doubleValue();
    }
    public double getAverage() {
        return total/history.size();
    }
}
```

Rep exposure can occur if a client adds a Signal to History and then modifies the Signal by calling update(double) since it still has that reference. This will cause the History.getAverage() to return the wrong value since History is implemented on the assumption that Number is immutable.

End of exam