

Massachusetts Institute of Technology  
6.170 Laboratory in Software Engineering  
Spring 2003

# Quiz 1

Wednesday, March 12, 2003

Name: *Solutions* \_\_\_\_\_

Athena username: \_\_\_\_\_

**Section (circle one):**

- 1: Eugene Shih    2: Yuriy Brun    3: Connie Cheng    4: Naveen Goela  
5: Greg Harfst    6: Lee Lin    7: Jennifer Louie    8: Matt Notowidigdo  
9: Zeeshan Syed    10: Stefanie Tellex
- 

This quiz is 50 minutes long. It contains 22 questions and 9 pages (excluding this one). Please check your copy to make sure it is complete before you start. Turn in all pages, together, when you are finished. **Write your username and section on the top of *ALL* pages.**

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

## True/False [2 points each]

Circle the correct answer.

1. T/F The representation of an immutable object must not contain mutable objects.  
*False. The mutable objects merely need be inaccessible to external entities, such as clients.*
2. T/F Once it has been returned by the creator, the representation of an immutable object should not be changed.  
*False. Benevolent side effects are permitted as long as the abstract state of the class does not change.*
3. T/F If a field is declared `final`, the object referenced by the field is immutable.  
*False, the reference of the field cannot be changed, but the object itself may still be mutable, depending on the object type*
4. T/F There may be multiple implementations with different observable behavior that satisfy a given specification.  
*True. Imagine two implementations of a square root specification, one that returns the negative square root and one that returns the positive square root.*
5. T/F There may be multiple incomparable specifications that a given implementation satisfies.  
*True. For example, there could be two specs with incomparable `requires`, but an implementation which satisfies a stronger spec such as `requires true`.*
6. T/F The `modifies` clause of a specification specifies the objects (and/or specification fields) that an implementation is required to mutate.  
*False. The implementation is allowed to mutate them, but it is not required to.*
7. T/F A change to the abstract state of an ADT implies a change to its concrete state.  
*True. A concrete state must map to exactly one abstract state.*
8. T/F A change to the concrete state of an ADT implies a change to its abstract state.  
*False. An abstract state can have many concrete states.*
9. T/F Using immutable types increases the possibility of bugs caused by aliasing.  
*False. Aliasing depends on being able to tell one reference from another without using `==`. You can't do that with immutable types.*

Consider the following two specifications for procedure `foo(int)` for the next three questions.

SPECIFICATION A

```
/**
 * @requires true
 * @modifies nothing
 * @return a positive integer
 */
int foo(int arg);
```

SPECIFICATION B

```
/**
 * @requires arg > 0
 * @modifies nothing
 * @return arg
 */
int foo(int arg);
```

Answer the following true/false questions. Circle the correct answer.

10. **T/F** Any implementation that satisfies A also satisfies B.  
***False. A function that always returns "1" satisfies A but not B.***
11. **T/F** Any implementation that satisfies B also satisfies A.  
***False. The identity function satisfies B but not A.***
12. **T/F** Any program that works correctly using a correct implementation of A will also work correctly if a correct implementation of B is substituted.  
***False. As the answers to questions 10 and 11 show, A and B are incomparable.***

Consider the following three classes for questions 13–17. For brevity we do not list all methods and fields. You can assume that omitted fields and methods are the same for all classes. The only difference between ASet and CSet is CSet has the method isFull().

```
// A set that has a limit on its number of elements.
class ASet {
    /** @requires true
     * @modifies this
     * @effects if this.size < 10, this' = this U {e};
     *         otherwise, this' = this
     */
    void add(Object e);
}

// A finite set.
class BSet {
    /** @requires true
     * @modifies this
     * @effects this' = this U {e}
     */
    void add(Object e);
}

// A set that has a limit on its number of elements.
class CSet {
    /** @requires true
     * @modifies this
     * @effects if this.size < 10, this' = this U {e}
     *         otherwise, this' = this
     */
    void add(Object e);

    /** @requires true
     * @modifies nothing
     * @return true if the Set has nine elements; false otherwise.
     */
    boolean isFull();
}
```

Answer the following true/false questions. Circle the correct answer.

13. **T/F** ASet is a true subtype of BSet.  
*False. The specification for ASet is not stronger than the specification for BSet. In particular, ASet violates BSet's spec when it reaches 10 elements.*
14. **T/F** BSet is a true subtype of ASet.  
*False. The specification for BSet is not stronger than the specification for ASet. BSet violates ASet's spec of a bounded set.*

15. T/F BSet is a true subtype of CSet.  
*False. The specification for BSet is not stronger than the specification for CSet. BSet doesn't have an isFull method.*
16. T/F CSet is a true subtype of BSet.  
*False. The specification for CSet is not stronger than the specification for BSet. CSet is bounded, while BSet is infinite.*
17. T/F CSet is a true subtype of ASet.  
*True. The specification for CSet is stronger than the specification for ASet. The specification for add is the same, and it adds a method, making CSet's spec stronger.*

## Short Answer Questions

18. (8 points) Assume class A is a true subtype of class B. Under what circumstances will a program that works correctly using B also work correctly using A? Limit your answer to 20 words.

*The client program must depend only on the specification of B and A must also be a Java subtype of B.*

19. (16 points) Answer the following questions with no more than 3 sentences.

- (a) Describe a class of programming error that is more likely to be caught by black box test data than by glass box test data. Use an example if necessary.

*When the code fails to deal at all with some class of possible inputs. For example, if there were two branches to the requires clause: requires  $x < 10$  or  $x > 20$ , and the code only dealt with the case where  $x < 10$ .*

- (b) Describe a class of programming error that is more likely to be caught by glass box test data than by black box test data. Use an example if necessary.

*When two inputs that are treated the same way in the specification are treated differently in the code. For example, imagine an implementation that computes elements of the Fibonacci sequence, which caches the first 100 elements but computes the rest.*

20. (15 points) Consider the following two specifications for a method that operates on a set. Assume that the set contains only natural numbers (non-negative integers).

## SPECIFICATION A

```
/**
 * @requires true
 * @modifies nothing
 * @return   if set.size > 0, return the smallest number in the set
 * @throws   EmptySetException if set.size == 0
 */
int getSmallest() throws EmptySetException;
```

## SPECIFICATION B

```
/**
 * @requires true
 * @modifies nothing
 * @return   if set.size > 0, return the smallest number in the set
 *           otherwise return -1.
 */
int getSmallest();
```

Answer the following questions. Use no more than 5 sentences for each answer.

- (a) Which specification defines a better abstraction? Explain.

*First, note that the notion of better specifications is unrelated to stronger specifications.*

*Specification A is better because it throws an exception instead of returning a value of the same type as valid results. Clients could forget to check the return value, causing it to be mistaken for a valid answer; by contrast, it is impossible for clients to ignore an exception at runtime — and if the exception is checked, clients cannot ignore it at compile time, either. (In this particular case, since the set never contains non-negative numbers, the return value of -1 can be distinguished from any valid result.) A secondary advantage to using an exception is that it aids debugging: getSmallest on an empty set fails quickly with a comprehensible error message.*

- (b) For specification A, should `EmptySetException` be a checked exception or an unchecked exception? Explain.

*Either answer is acceptable, given sufficient justification. In particular, the best answer depends on how you expect the abstraction to be used, and your answer must so state (not just make an assertion about how you claim the specification is used).*

*A checked exception may be desirable to ensure that the client never overlooks, at compile time, the case when the set is empty; the client must explicitly catch the exception or declare the enclosing method to throw it. This approach is less error-prone and is usually the right approach.*

*An unchecked exception may be desirable if two conditions hold. First, the client is able to determine, or is expected to know, the size of the set. Second, it is desired to simplify the client's use of the abstraction by not requiring writing many `try...catch` blocks for a situation that reasoning has indicated should never occur in practice.*

*Many students stated that a checked exception could be caught by the client, but an unchecked one could not. This is not true: a catch clause may name any exception.*

*Many students said that the frequency of the exception determines whether it should be checked or unchecked. This is not true: it is whether the exception is ever expected to happen that determines this.*

21. (15 points) Answer the following questions using no more than 3 sentences each.

Java has both run-time and compile-time type checking.

(a) Why have type checking at all?

*Type checking lets you catch type errors early and in a predictable way that does not cause harmful side-effects during failure. Java fails fast with a `ClassCastException` instead of trying to access a field that doesn't exist in the object, and returning a random bit of memory.*

*Other acceptable answers included mentioning that type checking prevents subtle, unrecognizable, or unpredictable errors.*

(b) Why not have just compile-time type checking?

*It is not possible to do complete type checking at compile time. Type checking performed at compile time cannot resolve the runtime type of an object. For example, the following code will compile legally (with no type checking failures) since at compile time we will not know the runtime type of the object returned by `.get()` in line 3 and thus, whether the runtime type of the object returned can be cast to a `String`. Therefore, the compiler will report no errors.*

```
ArrayList streetNumberList = new ArrayList();           // 1
streetNumberList.add(1, new Integer(5));               // 2
String myString = (String)streetNumberList.get(1);     // 3
System.out.println("Length of string is: " +           // 4
                  myString.length());                  // 5
```

*Runtime type checking determines whether the casting of an object from one type to another is permissible.*

(c) Why not have just runtime type checking?

*Compile time type checking lets you catch errors sooner, so it makes sense to do what you can at compile time. Catching errors sooner can speed up debugging and testing. For example, in the following code, statement 2 will be caught by compiler-time type checking since the `floatValue()` method is not defined for `String` objects.*

```
String myInputString = "15";                           // 1
float convertedNum = myInputString.floatValue();        // 2
```

*Also, using only runtime type checking can be expensive to debug, since you will not know that you have used incorrect types until after the code has been executed.*

22. (12 points) Specify (*do not implement*) an iterator class that yields the elements of a collection of `GeoSegments` in order of ascending length. The class should have only two methods.

*The specification should contain an overview, and should specify `hasNext()` and `next()`. It should also mention how the iterator is related to the original collection, in particular, what happens if the collection changes.*

```
/**
 * Iterates over a collection of GeoSegments in order of
 * ascending length. The underlying collection must not
 * be modified while using the iterator.
 */
public class GsIterator implements Iterator {

    /**
     * @requires true
     * @modifies nothing
     * @return true if the iterator has more GeoSegments, false otherwise.
     */
    public boolean hasNext();

    /**
     * @requires true
     * @modifies this
     * @return the next GeoSegment in the iterator, in order of
     * ascending length.
     * @throws NoSuchElementException if there are no more GeoSegments.
     */
    public Object next();
}
```