

MASSACHUSETTS INSTITUTE OF TECHNOLOGY  
6.170 LABORATORY IN SOFTWARE ENGINEERING  
FALL 2002

Quiz 2

November 6, 2002

ANSWERS

## A Shorties

True or false?

Iterators should be used with care in Java, because they:

- A.1 may expose the rep of the underlying collection.  
*True.*
- A.2 may stop working if the underlying collection changes.  
*True.*
- A.3 may throw a checked exception.  
*False. A Java iterator may only throw unchecked exceptions, such as NoSuchElementException.*

Iterative design:

- A.4 is a pattern for traversing collections.  
*False. Iterative design is not the same as an iterator.*
- A.5 is compatible with the spiral model.  
*True. The spiral model is in fact a special case of iterative design.*
- A.6 cannot be done on paper.  
*False. Paper sketches, whether of user interfaces, object modules, or module dependencies, can be evaluated and then redesigned. Paper prototypes are a particularly handy method for iterative design of user interfaces.*
- A.7 never involves users until the last iteration.  
*False. Users can (and should) be brought in at various stages of an iterative design process, to test a prototype of almost any fidelity.*

A class A is a genuine subtype of a class B if:

- A.8 B can be substituted for A in client code.  
*False. Substitutability goes the other way: A is a subtype of B if A can be substituted for B in client code.*
- A.9 A is a subclass of B.  
*False. The subclass relationship is not sufficient for A to be a subtype of B; the behavior of A must be substitutable for B as well.*
- A.10 A and B are interfaces, and A extends B.  
*False. As in the previous answer, it isn't enough that A extends B. The specification of A must also be at least as strong as the specification of B.*

A conceptual model:

- A.11 describes the key elements of a problem domain.  
*True.*
- A.12 is not worth constructing unless done right at the start of a development.  
*False. Conceptual models can be constructed whenever additional clarity is called for.*
- A.13 can help improve an application from the user's perspective.  
*True. Basing an application on a clean and simple conceptual model can make it easier for a user to understand how to use its features effectively, and how to predict its behaviour.*

## B Usability

Each part in this section describes an aspect of a user interface design. For each part, select the one that is *most* relevant to (gives support for or evidence against) the stated design.

B.1 A pulldown menu has 20 menu items.

- A. Fitts's Law
- B. short-term memory
- C. perceptual fusion
- D. visual image store
- E. auditory image store

*A. By Fitts's Law, the time to move the mouse to an item at the end of the menu depends on the length of the menu. Short-term memory (B) and visual image store (D) are not relevant because the menu persists on the screen; the user doesn't need to recall it from memory. Reading speed would also be relevant to a long menu, but it wasn't one of the choices here.*

B.2 When the user clicks a button, some kind of feedback is given within 100 milliseconds.

- A. Fitts's Law
- B. short-term memory
- C. perceptual fusion
- D. visual image store
- E. auditory image store

*C. If the button click and the response occur within 100 msec (on average), then perceptual fusion will lead a user to view them as related by cause-and-effect.*

B.3 The Shift key is twice the size of a normal keyboard key.

- A. Fitts's Law
- B. short-term memory
- C. perceptual fusion
- D. visual image store
- E. auditory image store

*A. Fitts's Law indicates that a larger target is easier to hit.*

B.4 The Caps Lock key toggles caps lock mode on and off, with a light on the keyboard showing the current state of the mode.

- A. Help & Documentation
- B. Prevent Errors
- C. Minimize Memory Load
- D. Less is More

*B. Caps Lock is a mode, which may cause mode confusion errors. The Prevent Errors heuristic argues for either eliminating the mode or at least making it visible, by a light on the keyboard. The light alone is probably not strong enough to prevent errors, however. Even with the light, password entry errors are often due to Caps Lock being on. Answer C (Minimize Memory Load) received partial credit, 1 point, since the light saves users from remembering whether Caps Lock is on.*

B.5 The File menu includes a history of the last few files opened.

- A. Feedback

- B. Minimize Memory Load
- C. Less is More
- D. Help & Documentation

*B. The file history minimizes memory load by allowing the user to recognize the recent file they want to edit, rather than recall where it is and what it was named. An even more relevant heuristic is Shortcuts, but it wasn't included among the choices.*

## C Java Collections Framework

True or false?

- C.1 The abstract classes of the Java collection framework are skeletal implementations.  
*True.*
- C.2 Clients of collections should not refer to the abstract classes.  
*True. Clients should use interfaces or concrete classes.*
- C.3 Clients of collections should refer to collections by the strongest applicable interfaces.  
*False. The opposite is true: a use of any service by a client should always rely on the weakest possible specification.*
- C.4 Skeletal implementations of collections define default representations. *False.*

True or false?

- C.5 Views are a powerful but dangerous programming idiom.  
*True.*
- C.6 Views can result in a form of 'aliasing' between references of different types.  
*True.*
- C.7 The use of views generally increases coupling between modules.  
*False. Views provide decoupling by allowing clients to assume weaker specifications.*
- C.8 A view can never be modified except through the underlying object.  
*False: consider the keySet view of Map, for example.*

Select one answer for the following question.

- C.9 Which design pattern is central to the skeletal collection implementations?
- A. Visitor.
  - B. Factory.
  - C. Composite.
  - D. Template.
  - E. State.
- D.*

Select one answer for the following question.

- C.10 Which aspect of the framework design is relevant to the question of whether subclasses and subinterfaces are subtypes?
- A. No instance variables in abstract classes.
  - B. Optional methods.
  - C. Provision of views.
  - D. Use of comparators in sorted collections.
- B.*

## D Conceptual Model Jigsaw

Your task, for this question, is to construct a conceptual model of the Java type system. It captures the relationships between classes and interfaces, and between the declared types of fields and the types of the objects they hold. It ignores primitive types and local and static variables.

The diagram, shown below, is missing its set and relation labels. For each box or arc, select the appropriate set or relation from the lists below, and enter the corresponding letter into the answer key.

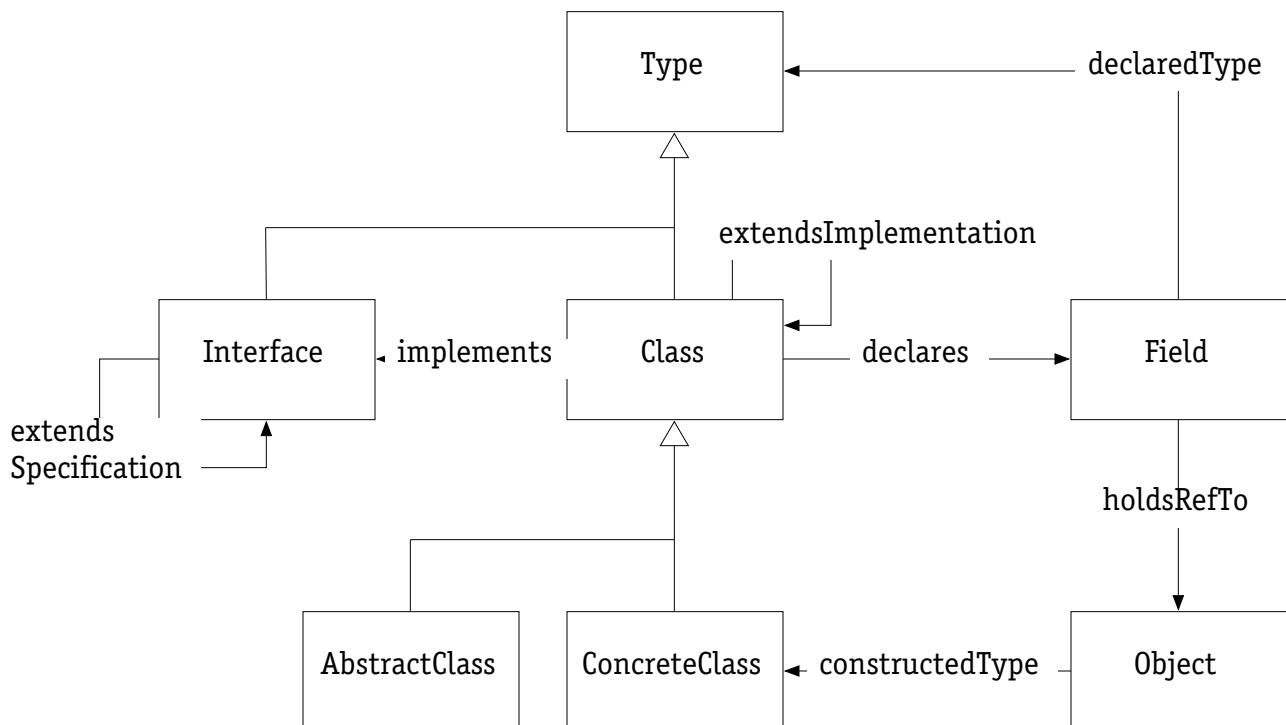
The sets are:

- A. AbstractClass
- B. Class
- C. ConcreteClass
- D. Field
- E. Interface
- F. Object
- G. Type

The relations are:

- A. implements
- B. extendsImplementation
- C. extendsSpecification
- D. declaredType
- E. constructedType
- F. holdsRefTo
- G. declares

*Many students were confused by Object, and assumed that, by analogy with the Object class in the Java type hierarchy, it should be the root of the classification. But here it refers to the set of objects (as distinct from types, fields, etc).*



## E Design Patterns

Study the following fragments of a system for organizing shipments, then answer the questions that follow.

```
interface Container {
    // represents items that can contain other items

    void add (Containable c);
    Iterator contentsIterator ();
}
interface Containable {
    // represents items that can be contained in other items

    int getWeight ();
    int getVolume ();
}
class Box implements Container, Containable {
    ...
}

class Fruit implements Containable {
    Fruit (String kind) {...}
    ...
}
class Package implements Container {
    Package (TrackingLabel label) {...}

    void addTracker (Tracker tracker) {...}
        // modifies this
        // effects: associates a tracker with this package so that subsequent location changes can be tracked

    void removeTracker (Tracker tracker) {...}
        // modifies this
        // effects: removes the association of a tracker with this package
    ...
    int getWeight () {...};
    int getVolume () {...};
    Label getLabel () {...}
}
class TrackingLabel {
    static TrackingLabel generate () {...}
    ...
}
interface Tracker {
    // represents an object interested in knowing where a package is.
    void arrivedAt (TrackingLabel package, Location waypoint);
}

class Location implements Location {
    // represents the location of a package
    ...
}
```

Which of the following patterns occur in the code above? Select all that apply.

- E.1 composite  
*True. All the classes shown participate in a composite pattern, with Fruit as a leaf, and Box and Package as composites.*
- E.2 prototype  
*False.*
- E.3 observer  
*True. Tracker is an observer.*
- E.4 iterator  
*True. Containers can return an iterator over their contents.*
- E.5 adapter  
*False.*
- E.6 proxy  
*False.*
- E.7 factory method  
*True. TrackingLabel has a factory method.*
- E.8 typesafe enumeration  
*False.*

Ben Bitdiddle proposes using the **typesafe enumeration pattern** for Fruit. Which of the following would be relevant arguments for or against Ben's proposal? Select all that apply.

- E.9 You wouldn't be able to ask a particular banana what box it's in.  
*True. The typesafe enumeration pattern would prevent this, because there would be only one object representing each kind of fruit, not an object for each particular piece of fruit.*
- E.10 There's no natural ordering of fruits like there is of integers.  
*False. This isn't relevant to the decision, because a typesafe enumeration can represent either ordered or unordered types.*
- E.11 The user must be able to add new kinds of fruit at runtime.  
*True. A typesafe enumeration is fixed at compile time, so the user can't add new kinds of fruit at run time. The interning pattern might be preferred in this case.*
- E.12 Every piece of fruit has its own expiration date.  
*True. See E.9.*

Ben notices that the performance of his program is poor, and he thinks using an **interning pattern** for Fruit might help. Which of the following issues should he take into account in making his decision? Select all that apply.

- E.13 Fruits of the same weight and kind are indistinguishable.  
*True.*
- E.14 A large number of Fruit objects are being allocated.  
*True.*
- E.15 In a future version of the software, Ben expects to add a feature that marks all the fruit in a shipment as safe when the shipment is approved by the FDA.  
*True.*

E.16 Fruits are not themselves containers of other items.  
*True.*

Alyssa Hacker proposes adding the **visitor pattern** to the design. Which of these operations could be reasonably implemented as visitors?

E.17 Calculating the total weight of a shipment.  
*True.*

E.18 Marking fruits in a shipment that have past their expiration date.  
*True.*

E.19 Moving items from one container to another.  
*False. Moving items from one container to another does not require looking deeply into each item.*

E.20 Finding a package in a collection of packages with a given tracking label.  
*False. Packages can only be top-level (never found in a Container), so a visitor is unnecessary. An iterator over the collection of packages would be more appropriate.*

Alyssa points out to Ben that because Package has the methods getWeight and getVolume, it should be declared as implementing Containable. Ben disagrees, and they have a heated argument in which many points are made. Which of the following points are correct about having Package implement Containable? Select all that apply.

E.21 It would mean that an object model constraint now checked by the compiler would need to be checked explicitly at runtime.  
*True. In the present design, the compiler won't allow you to add a Package to another Container. This is a reasonable object model constraint, since if a Package is inside another, its tracking label is invisible to delivery workers. If Package implemented Containable, then the compiler would no longer check this constraint at compile time.*

E.22 It would allow for more code sharing in the bodies of methods in Package.  
*False. Interfaces do not include any code, so Package would not be able to inherit any of its method bodies from Containable.*

E.23 It would allow the code that computes the total weight of a collection of packaged and un-packaged items to be implemented more uniformly.  
*True. In the present design, code must cast each item in the collection to its appropriate class, either Package or Containable, before calling getWeight. If Package implemented Containable, then the code could treat all items as Containable, whether packaged or not.*

E.24 It would not be possible anyway because there is no multiple inheritance in Java.  
*False.*