

True/False [1 point each]

1. Mutable objects cannot be interned.
2. Decentralized organization permits meetings to proceed effectively without a leader.
3. A good design does not rule out any future extension to a project.
4. It is typically preferable to allow a project to slip gradually rather than enduring a single large slip in the schedule.
5. The Composite pattern is useful for operations on hierarchical data structures.
6. When examining a data structure using the Procedural pattern, the examining class must do its own type dispatching on the possible types of the data structure.
7. A Java interface should include an abstraction function.
8. Introducing interfaces can increase the total number of dependences.
9. Introducing interfaces can help to eliminate dependences.
10. A weak dependence of class A on class B implies an arrow from class A to class B in the code object model.
11. A strong dependence of class A on class B implies an arrow from class A to class B in the code object model.

For questions 12–15, consider the following code fragment:

```
/** An IntSet is a mutable set of integers.
    Examples include: { } , { 1, 2, 3 }, { 4, 5, 6, 7, 100 }
 */
class IntSet {
    REPTYPE nums;

    // RI(c): ...
    // AF(c): { i | i is in nums }
}
```

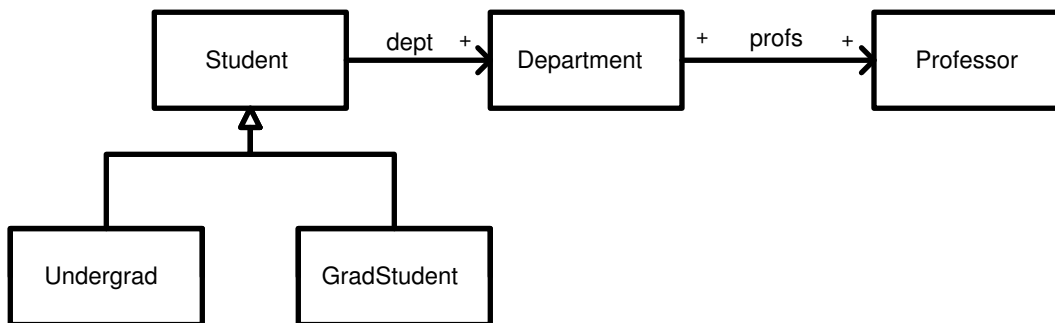
Note that questions 12–14 use the term “sensible”, while question 15 uses the term “necessary”.

12. If REPTYPE is HashSet, “No duplicate values in c.num” would be a *sensible* clause to include in the representation invariant.
13. If REPTYPE is HashSet, “o in c.num \Rightarrow (o instanceof Integer)” would be a *sensible* clause to include in the representation invariant:
14. If REPTYPE is Integer[], “o in c.num \Rightarrow (o instanceof Integer)” would be a *sensible* clause to include in the representation invariant.
15. If REPTYPE is Integer[], “No duplicate values in c.num” would be a *necessary* clause to include in the representation invariant.

Write your answers to these questions on the answer page, not on this page.

16. The representation of an ADT is exposed if one of its methods returns a mutable object.
17. A code object model describes the state of the heap at runtime.
18. A snapshot describes the state of the heap at runtime.
19. In a problem object model, the nodes represent classes and the edges represent relations or subtyping.
20. “Requirement specification 50% completed” is a good milestone.
21. “Module A coding completed and unit-tested” is a good milestone.
22. “Project design completed and reviewed” is a good milestone.
23. “Module A unit testing completed” is a good milestone.
24. “Module A specification completed” is a good milestone.

Questions 25–28 relate to the object model below, which describes a university (though not necessarily MIT).



25. Every student is in some department.
26. Students may switch departments.
27. A student cannot switch from undergrad to grad status.
28. Every department has students.

Multiple choice [2 points each]

29. A Factory can:
- (a) return a subtype of a class where a supertype is expected
 - (b) return pre-existing objects rather than new ones
 - (c) encapsulate the creation of an object or set of objects
 - (d) (a) and (b)
 - (e) (a) and (c)
 - (f) (b) and (c)
 - (g) (a), (b), and (c)
30. Which of these patterns does not control sharing?
- (a) Flyweight
 - (b) Interning
 - (c) Prototype
 - (d) Singleton
31. A Singleton class has:
- (a) a public constructor and a public accessor for the single instance
 - (b) a private constructor and a public accessor for the single instance
 - (c) a public constructor that throws an exception if called more than once
 - (d) a private constructor that throws an exception if called more than once
32. A Proxy:
- (a) changes the interface to an underlying class
 - (b) extends the functionality of an underlying class
 - (c) both changes the interface and extends the functionality
 - (d) neither changes the interface nor extends the functionality
33. Subjects and Observers/Listeners appear in:
- (a) the push model
 - (b) the pull model
 - (c) the publish/subscribe model
 - (d) all of the above
 - (e) none of the above

Write your answers to these questions on the answer page, not on this page.

34. In a well-managed large project typically the least time is spent on

- (a) planning
- (b) coding
- (c) component testing
- (d) integration testing

35. What are the three properties (from the list below) that a loop invariant must satisfy? (Write down three answers, in alphabetic order, in the three boxes of the answer sheet.)

Assume that the loop is of the form

$$P \{ \text{while } (b) \ S \} Q$$

where P is the precondition and Q is the postcondition; let I be the loop invariant.

- (a) $P \Rightarrow I$
- (b) $P \Rightarrow Q$
- (c) $I \ \&\& \ !b \Rightarrow Q$
- (d) $I \Rightarrow Q$
- (e) $I \{ S \} Q$
- (f) $I \ \&\& \ b \{ S \} I$
- (g) $I \ \&\& \ P \{ S \} Q$
- (h) $I \{ S \} Q \Rightarrow I$
- (i) $I \{ S \} \ !b \Rightarrow Q$
- (j) $Q \Rightarrow !P$
- (k) $Q \Rightarrow !b$
- (l) $Q \Rightarrow I$

Short answer [approximately 6 points each]

36. (4 points) What is the weakest precondition for statement composition, i.e., $wp((S1 ; S2), Q)$?

37. (6 points) The *strongest postcondition* sp is defined analogously to the weakest precondition wp . The strongest postcondition is useful for forward reasoning about code. What is $sp((x = y + 1; y = 2 * x), (y > 10 \ \&\& \ x > 0))$?

38. (6 points) Give two guidelines for at what time a prototype should be discarded.

Username:

Section number:

39. (5 points) Which sorts of projects is the waterfall model good for?

40. (6 points) Give three reasons why adding people to a project is usually inefficient.

41. (7 points) Write a decrementing function (as a legal Java expression) and a loop invariant (which may be either formal or informal) for the following code fragment:

Precondition: `sbuff` is a character array possibly containing a 0 character (`'\0'`).

Postcondition: `s` is a `String` which is made up of the characters from `sbuff` up to, but not including, the first 0 character in `sbuff`, if any.

```
String s = "";
for (int i=0; sbuff[i] != '\0' && i < sbuff.length; i++) {
    s += sbuff[i];
}
```

You may find it more convenient to consider the following equivalent code; the answer is the same regardless of which version of the code you use.

```
String s = "";
int i=0;
while (sbuff[i] != '\0' && i < sbuff.length) {
    s += sbuff[i];
    i++;
}
```

(a) Loop invariant:

(b) Decrementing function:

42. (8 points) Consider a collection class C with a representation made up of two integer arrays:

```
class C {
    int[] a;
    int[] b;
    ...
}
```

One concrete value is this one:

$$\begin{aligned} a &= [-2, 4, 8, 3] \\ b &= [1, 4, 10, 6] \end{aligned}$$

Hint: think about pairs, such as $\langle a[i], b[i] \rangle$.

- (a) Assume that C can represent any set. Write a representation invariant and an abstraction function for an implementation in which the concrete value represents the set

$$\{-2, -1, 0, 1, 3, 4, 5, 6, 8, 9, 10\}.$$

(Hint: this is similar to something you saw on a problem set.) Your answer may be formal or informal, but must be clear.

representation invariant:

abstraction function:

- (b) Assume that C can represent any multiset. Write a representation invariant and an abstraction function for an implementation in which the concrete value represents the multiset (bag)

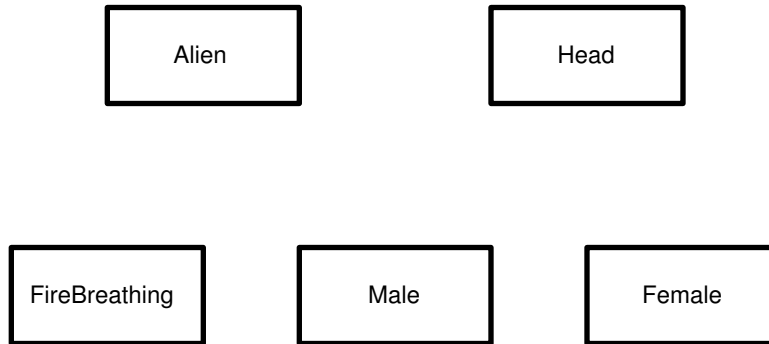
$$\{-2, 4, 4, 4, 4, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 3, 3, 3, 3, 3, 3\}.$$

Your answer may be formal or informal, but must be clear.

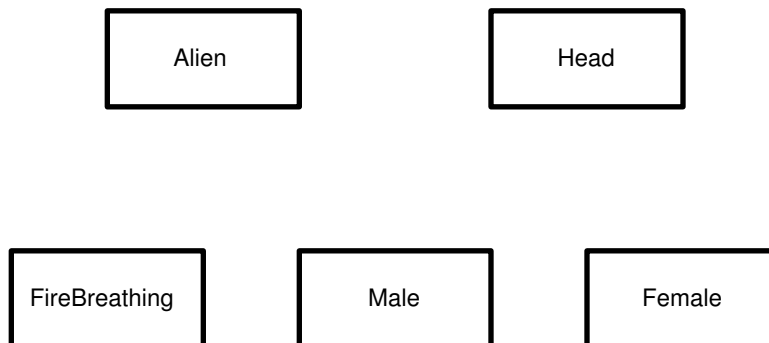
representation invariant:

abstraction function:

43. (6 points) Complete the object model below in order to describe alien life forms. Aliens are either male or female, and some aliens are fire-breathing. Aliens can have many heads, but each alien must have at least one head. Heads are never destroyed, nor are new ones created. After an alien dies, its head(s) detach and can at some later time become part of another alien that is being born. But once an alien is born, it cannot gain new heads or lose any heads. Aliens don't share heads. Your object model should not express any other properties about aliens.



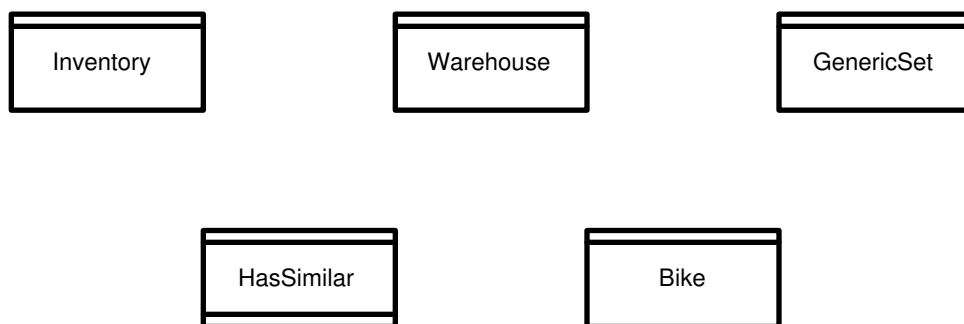
Here is another copy of the answer template, in case you make a mistake above. We will grade this copy *only* if the above copy is clearly crossed out.



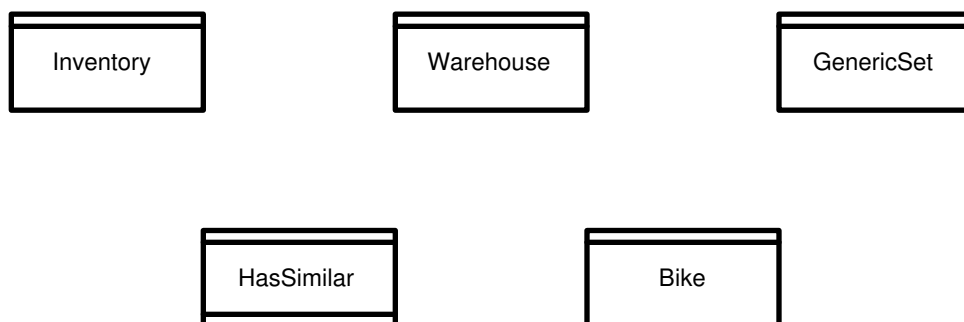
44. (6 points) You have been hired to review the design of the database for Mike's Bikes. To analyze the quality of the design, you decide to construct a module dependence diagram for this system.

Mike has outlined the code he plans to use; **see page 12**, which you may tear off for easier reference. Unfortunately for you, the outline is incomplete, and with hardly any specifications. Assuming the implementation is well-designed, with no unnecessary dependences, construct an MDD by completing the below figure.

You will have to make educated guesses as to how the classes are implemented and what the methods do. In order to answer this question, you need to understand the classes at a high level, but do not need to understand every nuance of the implementation.



Here is another copy of the answer template, in case you make a mistake above. We will grade this copy *only* if the above copy is clearly crossed out.



This is the code you will analyze for question 44, which is stated on page 11.

```
class Inventory { // mutable
    // implementation fields here

    public Inventory();
    public int addWarehouse(Warehouse r);
    public void addBike(int warehouseNumber, Bike b);
    public Warehouse getWarehouse(int warehouseNumber);
    public Warehouse findBike(Bike b);
}

class Warehouse { // mutable
    GenericSet bikes;

    public Warehouse();
    public void addBike(Bike b);
    public boolean hasBike(Bike b);
}

class GenericSet { // mutable
    // implemented via dynamically resized array.

    public GenericSet();
    public void addElement(HasSimilar obj);
    // returns true if this contains an element e such that e.similar(obj)
    public boolean hasElement(HasSimilar obj);
}

interface HasSimilar {
    // returns true if the 'this' and 'obj' cannot
    // be distinguished based on calls to their observers.
    public boolean similar(HasSimilar obj);
}

class Bike implements HasSimilar { // mutable
    // various fields here

    public Bike(/* various attributes here */);
    // getter and setter methods for attributes here
    public boolean similar(HasSimilar obj);
}
```