Chapter 2: Abstract Data Types

Section 2.1: Abstraction

True / False

1. True or False? An abstraction of a system is a model of the system that includes all the details about the system.

Answer: False

2. True or False? UML diagrams are a form of abstraction, since they hide details and allow us to concentrate just on the major design components.

Answer: True

3. True or False? UML diagrams are not a form of abstraction, since they hide details and only allow us to concentrate on the major design components.

Answer: False

4. True or False? When you code a method that has preconditions the first thing you should have the code do is check that the preconditions are true.

Answer: False

5. True or False? An *abstract* method must not have any executable code.

Answer: True

6. True or False? An *abstract* method must not have any comments.

Answer: False

7. True or False? In order to be used, an *interface* must implement a *class*.

Answer: False

8. True or False? A *class* can extend an *interface*.

Answer: False

9. True or False? A class that implements an interface can include methods that are not required by the interface.

Answer: True

10. True or False? A class that implements an interface can leave out methods that are required by the interface.

Answer: False

11. True or False? An interface definition can include constant declarations.

Answer: True

12. True or False? An interface definition can include concrete methods.

Answer: False

13. True or False? A Java interface must have at least one defined constructor.

Answer: False

14. True or False? You can instantiate objects of an interface.

Answer: False

Multiple Choice

15. We deal with ADTs on three levels. On which level do we just need to know how to use the ADT?

A. application (or user, client)
B. machine (or assembly)
C. logical (or abstract)
D. primitive (or language dependent)
E. implementation (or concrete)

Answer: A

16. We deal with ADTs on three levels. On which level do we deal with the "how" questions, as in how we represent the attributes and fulfill the responsibilities of the ADT?

A. application (or user, client)
B. machine (or assembly)
C. logical (or abstract)
D. primitive (or language dependent)
E. implementation (or concrete)

Answer: E

17. We deal with ADTs on three levels. On which level do we just deal with the "what" questions, as in what does the ADT model, what are its responsibilities, what is its interface?

A. application (or user, client)
B. machine (or assembly)
C. logical (or abstract)
D. primitive (or language dependent)
E. implementation (or concrete)

Answer: C

Fill-in-the-Blank

18. A data type whose properties (domain and operations) are specified independently of any particular implementation is a(n) _____ .

Answer: Abstract Data Type or ADT

19. The Java _____ mechanism can be used by Java programmer's to create their own ADTs.

Answer: *class*

20. Assumptions that must be true upon entry into a method for it to work correctly are _____ .

Answer: preconditions

21. The Java _____ construct is used to formally specify the logical level of our ADTs.

Answer: interface

Short Answer

22. What does the acronym ADT stand for?

Answer: Abstract Data Type

23. What are the three perspectives, or levels, with which we deal with ADTs?

Answer: Application (or user or client) level. Logical (or abstract) level. Implementation (or concrete) level.

24. *Abstract* methods do not have any code, so what can they be used for?

Answer: An *abstract* method is used to define the interface (or signature) of its corresponding concrete methods.

25. What kinds of constructs can be declared in a Java interface?

Answer: constants (*final* variables) and *abstract* methods

26. What are the benefits we accrue by using a Java *interface* construct to formally specify of ADTs?

Answer: We can check the syntax of our specification by compiling the interface. We can verify that the interface "contract" is satisfied by a class that *implements* the interface. We can provide a consistent interface to applications from among alternative implementations of the ADT.

Section 2.2: The StringLog ADT Specification

[The questions in this section assume the student has access to the StringLog interface found on page 74 of the textbook.]:

```
//-------------------------------------------------------------------
// StringLogInterface.java     by Dale/Joyce/Weems           Chapter 2
//
// Interface for a class that implements a log of Strings.
// A log "remembers" the elements placed into it.
//
// A log must have a "name".
//-------------------------------------------------------------------

package ch02.stringLogs;

public interface StringLogInterface
{
  void insert(String element);
  // Precondition:   This StringLog is not full.
  //
  // Places element into this StringLog.

  boolean isFull();
  // Returns true if this StringLog is full, otherwise returns false.

  int size();
  // Returns the number of Strings in this StringLog.

  boolean contains(String element);
  // Returns true if element is in this StringLog,
  // otherwise returns false.
  // Ignores case differences when doing string comparison.

  void clear();
  // Makes this StringLog empty.

  String getName();
  // Returns the name of this StringLog.

  String toString();
  // Returns a nicely formatted string representing this StringLog.
}
```

Short Answer

27. How many methods are specified in the *StringLogInterface*?

Answer: Seven

28. Which methods specified in the *StringLogInterface* have a *boolean* return value?

Answer: *isFull* and *contains*

29. Which methods specified in the *StringLogInterface* are observer methods.

Answer: *contains*, *size*, *isFull*, *getName*, and *toString*

30. Which methods specified in the *StringLogInterface* are transformer methods.

Answer: *insert* and *clear*

31. According to the *StringLogInterface* what happens if an application calls the *insert* method when a StingLog object is full?

Answer: Since “this StringLog is not full” is a precondition we assume that *insert* is never called when the StringLog object is full – so if it is called in that situation the result is not defined.

32. What package is the *StringLogInterface* part of?

Answer: ch02.stringLogs

33. What happens if you include the following code in an application that has access to the *StringLogInterface*?

StirngLogInterface newLog = new StringLogInterface();

Answer: You get a syntax error since you cannot instantiate interfaces.

Section 2.3: Array-Based StringLog ADT Implementation

True / False

34. The *ArrayStringLog* implementation provided in Chapter Two is an example of a bounded implementation of a StringLog.

Answer: True

Multiple Choice

35. The repeated use a method name with a different signature is an example of

A. an interface
B. a subclass
C. a transformer
D. overloading
E. none of these

Answer: D

Fill-in-the-Blank

36. A method's name, the number and type of parameters that are passed to it, and the arrangement of the different parameter types within the parameter list combine to form what Java calls the _____ of the method.

Answer: signature

37. Complete the following table by entering the Big-O execution time of the indicated operations when using the *ArrayStringLog* class, assuming a list size of N.

method	efficiency
constructor	_____
insert	_____
clear	_____
clear and reclaim	_____
isFull	_____
getName	_____
size	_____
toString	_____
contains	_____

Answer:

method	efficiency
constructor	O(maxsize)
insert	O(1)
clear	O(N)
isFull	O(1)
getName	O(1)
size	O(1)
toString	O(N)
contains	O(N)

Short Answer

Questions 30 to 32 assume the student has access to the beginning of the *ArrayStringLog* class, as developed in the textbook:

```
//--------------------------------------------------------------------
// ArrayStringLog.java      by Dale/Joyce/Weems               Chapter 2
//
// Implements StringLogInterface using an array to hold log strings.
//--------------------------------------------------------------------

package ch02.stringLogs;

public class ArrayStringLog implements StringLogInterface
{
  protected String name;              // name of this log
  protected String[] log;             // array that holds log strings
  protected int lastIndex = -1;       // index of last string in array
```

38. Complete the implementation of the following constructor:

```
public ArrayStringLog(String name, int maxSize)
// Precondition:   maxSize > 0
//
// Instantiates and returns a reference to an empty StringLog object
// with name "name" and room for maxSize strings.
{
  // complete the method body
}
```

Answer:

```
log = new String[maxSize];
this.name = name;
```

39. Complete the implementation of the *insert* method:

```
public void insert(String element)
// Precondition:   This StringLog is not full.
//
// Places element into this StringLog.
{
  // complete the method body
}
```

Answer:

```
lastIndex++;
log[lastIndex] = element;
```

40. Complete the implementation of the *isFull* method:

```
public boolean isFull()
// Returns true if this StringLog is full, otherwise returns false.
{
  // complete the method body
}
```

Answer:

```
  if (lastIndex == (log.length - 1))
  return true;
else
  return false;
```

Section 2.4: Software Testing

True / False

Multiple Choice

41. Testing a method by itself is called

A. Unit testing
B. Inspection
C. Desk checking
D. Black box testing
E. Walkthrough

Answer: A

42. A verification method in which the team performs a manual simulation of the program or design is called

A. Unit testing
B. Inspection
C. Desk checking
D. Black box testing
E. Walkthrough

Answer: E

Fill-in-the-Blank

43. A(n) _____ is a program that calls operations exported from a class, allowing us to test the results of the operations.

Answer: test driver

Short Answer

44. Explain the difference between software validation and verification.

Answer: Software validation is the process of determining the degree to which software fulfills its intended purpose, whereas software verification is the process of determining the degree to which the software fulfills its specifications. In other words validation asks “are we doing the right job?” and verification asks “are we doing the job right?”.

Section 2.5: Introduction to Linked Lists

True / False

Multiple Choice

Fill-in-the-Blank

45. A(n) _____ class includes an instance variable or variables that can hold a reference to an object of the same class.

Answer: self-referential

Short Answer

46. In our linked list implementation what is the value of the *link* attribute of the last node on a list?

Answer: null

47. Write pseudocode for traversing a linked list and displaying the information contained in its nodes.

Answer:

```
Set currentNode to the first node on the list
While (currentNode is not null)
  display the information at currentNode
  change currentNode to point to the next node on the list
```

48. What are the three general cases of insertion into a linked list that we must consider?

Answer: Insertion into the beginning of a list, insertion into the end of a list, and insertion into the middle of a list.

49. Write code to insert the *LLStringNode firstNode* into the front of the *myList* linked list of nodes.

Answer:

```
firstNode.setLink(myList);
myList = firstNode;
```

50. Suppose we have a linked list of Strings, as defined in the textbook, named *presidents*. Suppose it contains three nodes, with the first node holding "Adams", the second node "Washington", and the third node "Kennedy". What would be output by the following code:

```
LLStringNode temp = presidents;
System.out.println(temp.getInfo());
```

Answer: Adams

51. Suppose we have a linked list of Strings, as defined in the textbook, named *presidents*. Suppose it contains three nodes, with the first node holding "Adams", the second node "Washington", and the third node "Kennedy". What would be output by the following code:

```
LLStringNode temp = presidents;
temp = temp.getLink();
System.out.println(temp.getInfo());
```

Answer: Washington

52. Suppose we have a linked list of Strings, as defined in the textbook, named *presidents*. Suppose it contains three nodes, with the first node holding "Adams", the second node "Washington", and the third node "Kennedy". What would be output by the following code:

```
LLStringNode temp = presidents;
while (temp != null)
{
  temp = temp.getLink();
}
System.out.println(temp.getInfo());
```

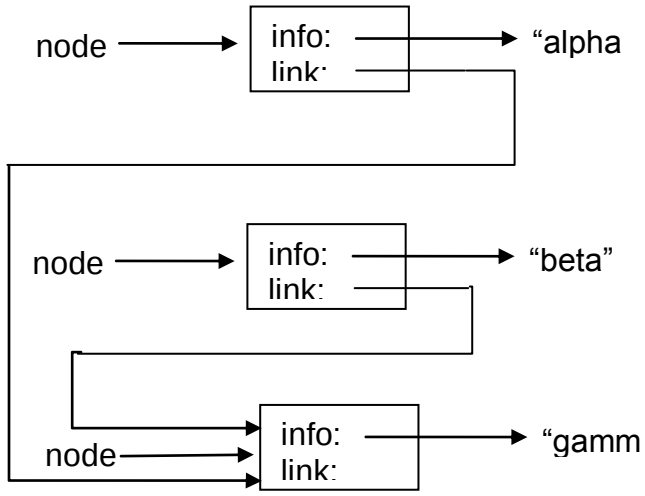
Answer: There will be an error message printed since we are attempting to access an object through a null reference.

53. Draw a figure representing our abstract view of the structures created by the following code sequence.

```
LLStringNode node1 = new LLStringNode("alpha");
LLStringNode node2 = new LLStringNode("beta");
```

```
LLStringNode node3 = new LLStringNode("gamma");
node1.setLink(node3);
node2.setLink(node3);
```

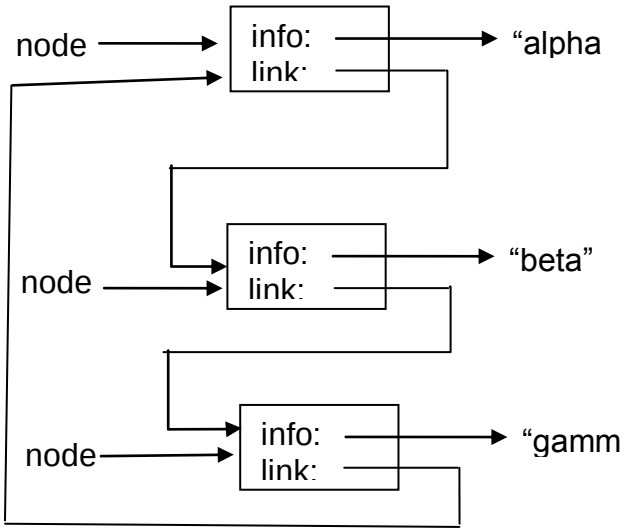
Answer:



54. Draw a figure representing our abstract view of the structures created by the following code sequence.

```
LLStringNode node1 = new LLStringNode("alpha");
LLStringNode node2 = new LLStringNode("beta");
LLStringNode node3 = new LLStringNode("gamma");
node1.setLink(node2);
node2.setLink(node3);
node3.setLink(node1);
```

Answer:


node
info:
link:
“alpha
node
info:
link:
“beta”
node
info:
link:
“gamm

Section 2.6: Linked List StringLog ADT Implementation

True / False

55. The *LinkedStringLog* implementation provided in Chapter Two is an example of a bounded implementation of a StringLog.

Answer: False

56. The *LinkedStringLog* implementation provided in Chapter Two is an example of an unbounded implementation of a StringLog.

Answer: True

Multiple Choice

57. A StringLog is

A. An array
B. A linked list
C. A container that holds strings
D. An interface
E. None of these

Answer: C

Fill-in-the-Blank

58. Complete the following table by entering the Big-O execution time of the indicated operations when using the *LinkedStringLog* class, assuming a list size of N.

method	efficiency
constructor	_____
insert	_____
clear	_____
clear and reclaim	_____
isFull	_____
getName	_____
size	_____
toString	_____
contains	_____

Answer:

method	efficiency
constructor	O(1)
insert	O(1)
clear	O(1)
clear and reclaim	O(N)
isFull	O(1)
getName	O(1)
size	O(N)
toString	O(N)
contains	O(N)

Short Answer

Questions 59 to 62 assume the student has access to the beginning of the *LinkedStringLog* class, as developed in the textbook:

```
//-------------------------------------------------------------------
// LinkedStringLog.java       by Dale/Joyce/Weems             Chapter 2
//
// Implements StringLogInterface using a linked list
// of LLStringNodes to hold the log strings.
//-------------------------------------------------------------------

package ch02.stringLogs;

public class LinkedStringLog implements StringLogInterface
{
  protected LLStringNode log; // reference to first node of linked
                              // list that holds the StringLog strings
  protected String name;      // name of this StringLog
```

59. Complete the implementation of the following constructor:

```
  public LinkedStringLog(String name)
  // Instantiates and returns a reference to an empty
  // StringLog object with name "name".
  {
    // complete this code
  }
```

Answer:

log = null;
this.name = name;

60. Complete the implementation of the *insert* method:

```
public void insert(String element)
// Precondition:   This StringLog is not full.
//
// Places element into this StringLog.
{
  // complete the method body
}
```

Answer:

```
LLStringNode newNode = new LLStringNode(element);
newNode.setLink(log);
log = newNode;
```

61. Complete the implementation of the *isFull* method:

```
public boolean isFull()
// Returns true if this StringLog is full, otherwise returns false.
{
  // complete the method body
}
```

Answer:

```
return false;
```

62. Complete the implementation of the *size* method:

```
public int size()
// Returns the number of Strings in this StringLog.
{
    // complete the method body
}
```

Answer:

```
int count = 0;
LLStringNode node;
node = log;
while (node != null)
{
  count = count + 1;
  node = node.getLink();
}
return count;
```

Section 2.7: Software Design: Identification of Classes

Fill-in-the-Blank

63. A standard technique for identifying classes and their attributes and responsibilities is to look for ideas in the problem statement. Candidate objects and their attributes are sometimes found in the _____ . And the _____ are often associated with responsibilities and actions.

Answer: nouns, verbs

Short Answer

64. What step in the design process usually follows brainstorming of the identification of classes?

Answer: Filtering the results of the brainstorming