

## Chapter 17 Object-Oriented Data Structures

1. A data structure is a collection of data organized in some fashion.

In object-oriented thinking, a data structure is an object that stores other objects, referred to as data or elements. So some people refer a data structure as a *container object* or a *collection object*. To define a data structure is essentially to declare a class.

2. Two limitations: (1) once an array is created, its size cannot be altered; (2) it does not provide adequate support for insertion and deletion operations.

3. MyArrayList is implemented using an array and an array is a fixed-size data structure. But MyArrayList is considered as a dynamic data structure, because its storage size changes behind the scene and hidden from the user.

4. Providing both interface and abstract class in the design makes it easy for the user to use the code. The user can use either the interface or the abstract class whichever is convenient.

The abstract class `MyAbstractList` is provided for convenience. For this reason, it is called a convenience class.

5. You have to add an object to list, not a primitive data value.

6. data will be null.

7. If the number of elements is fixed in the program, use array is more efficient. If the number of elements changes in the program, you may use `ArrayList` or `LinkedList`.

8. If you have to add or delete the elements anywhere in a list, use `LinkedList`.

9. Using inheritance: You can declare the stack class by extending the array list class, and the queue class by extending the linked list class. Using composition: You can declare an array list as a data field in the stack class, and a linked list as a data field in the queue class. Both designs are fine, but using composition is better because it enables you to declare a complete new stack class and queue class without inheriting the unnecessary and inappropriate methods from the array list and linked list.

10. Line 5, `new MyStack()` is not an instance of `MyList`.

11. If a set of the same elements is inserted into a binary tree in two different orders, will the two corresponding binary trees look the same? No. Will the inorder traversal be the same? Yes. Will the postorder traversal be the same? No. Will the preorder traversal be the same? No.