Data Structures & Algorithms

Lecture 04

What is Data Structure?

A data structure is a collection of data organized in some fashion. The structure not only stores data, but also supports operations for accessing and manipulating the data.

- Basic data structures
  - Record
  - Array
  - Linked List
  - Binary Tree
  - Hash Table
  - Heap
  - Graph

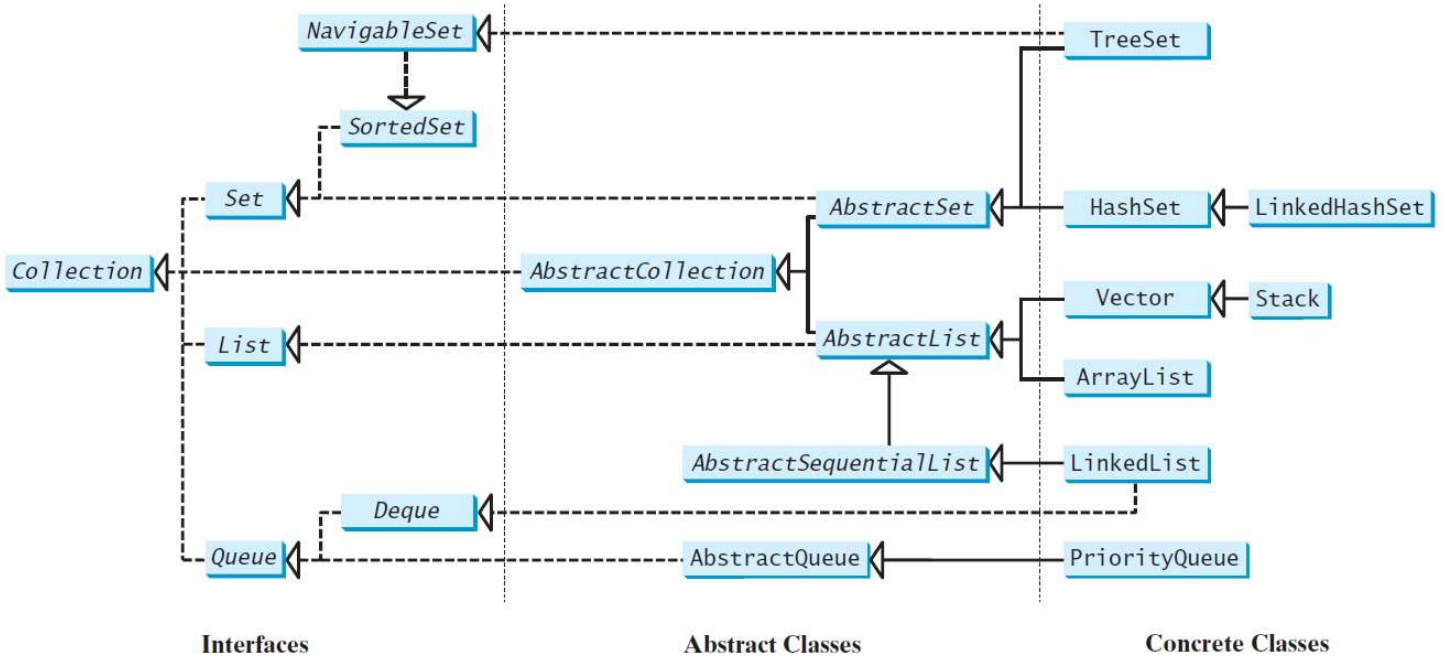
Java Collection Framework hierarchy

A *collection* is a container object that holds a group of objects, often referred to as *elements*. The Java Collections Framework supports two types of collections:

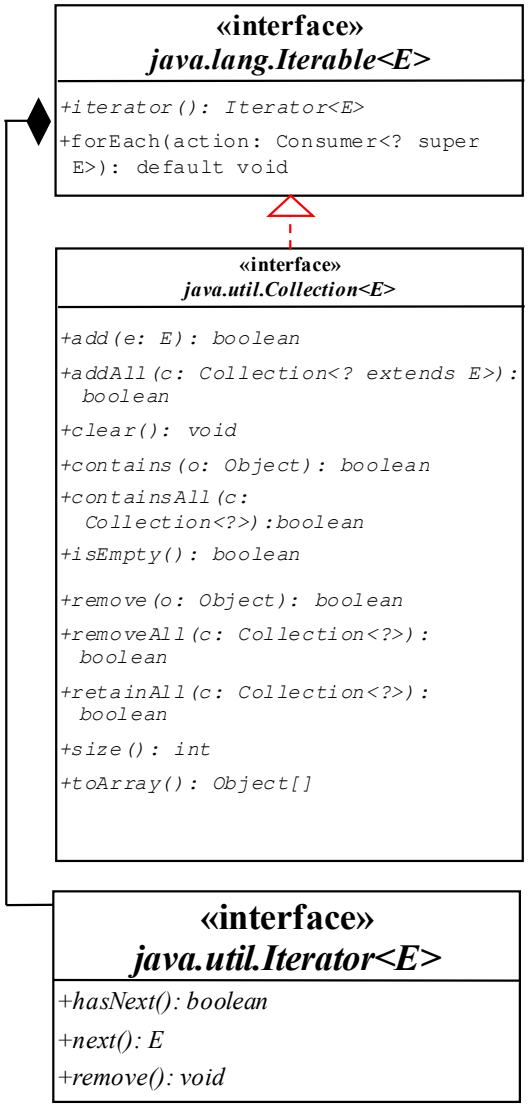
- One, for storing a collection , is simply called collection.
- The other, for storing key/value pairs, is called a map.

Java Collection Framework hierarchy, cont.

Set and List are subinterfaces of Collection.



The Collection Interface



Returns an iterator for the elements in this collection.

Performs an action for each element in this iterator.

The Collection interface is for manipulating a collection of objects.

Adds a new element `e` to this collection.

Adds all the elements in the collection `c` to this collection.

Removes all the elements from this collection.

Returns true if this collection contains the element `o`.

Returns true if this collection contains all the elements in `c`.

Returns true if this collection contains no elements.

Removes the element `o` from this collection.

Removes all the elements in `c` from this collection.

Retains the elements that are both in `c` and in this collection.

Returns the number of elements in this collection.

Returns an array of Object for the elements in this collection.

Returns a stream from this collection.

UML Diagram

TestCollection

Returns true if this iterator has more elements to traverse.

Returns the next element from this iterator.

Removes the last element obtained using the next method.

Iterators

Iterator is a classic design pattern for walking through a data structure without having to expose the details of how data is stored in the data structure.

TestIterator

Using the forEach Method

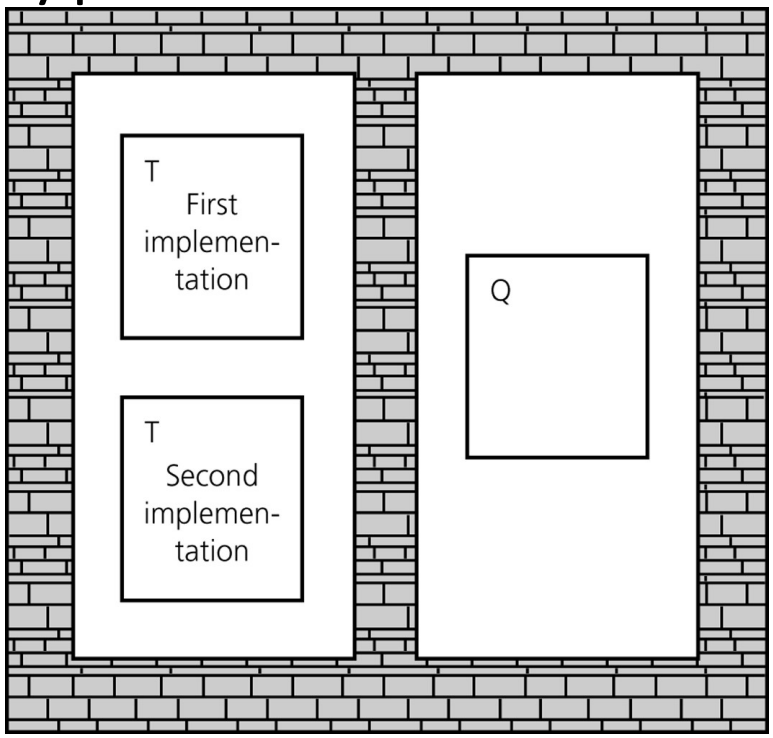
Java 8 added a new forEach method in the Iterable interface. It functions like a foreach loop.

TestForEach

Abstract Data Types

- Procedural abstraction
  - Separates the purpose and use of a module from its implementation
  - A module's specifications should
    - Detail how the module behaves
    - Identify details that can be hidden within the module
- Information hiding
  - Hides certain implementation details within a module
  - Makes these details inaccessible from outside the module

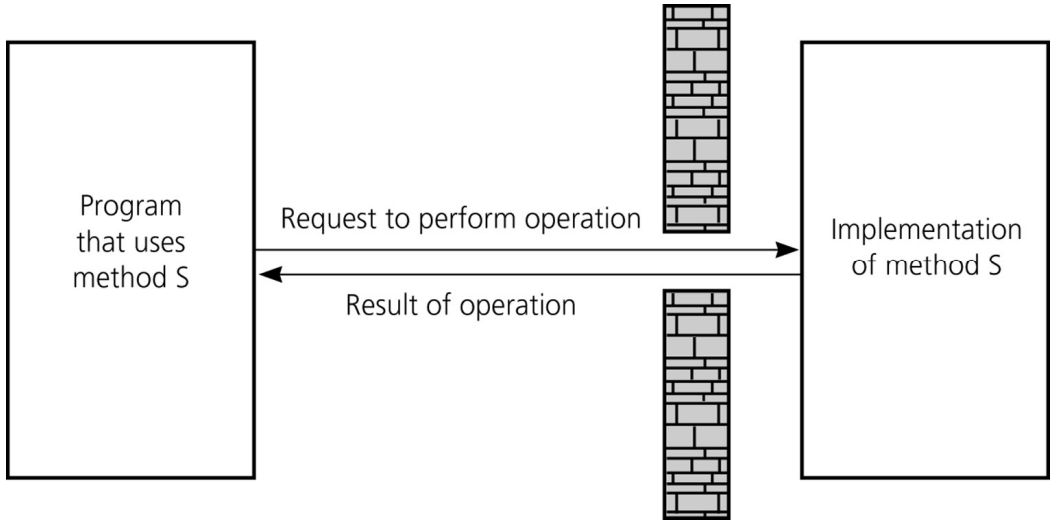
Abstract Data Types



Isolated tasks: the implementation of task *T* does not affect task *Q*

Abstract Data Types

- The isolation of modules is not total
  - Methods' specifications, or contracts, govern how they interact with each other



A slit in the wall

Abstract Data Types

- Typical operations on data
  - Add data to a data collection
  - Remove data from a data collection
  - Ask questions about the data in a data collection
- Data abstraction
  - Asks you to think *what* you can do to a collection of data independently of *how* you do it
  - Allows you to develop each data structure in relative isolation from the rest of the solution
  - A natural extension of procedural abstraction

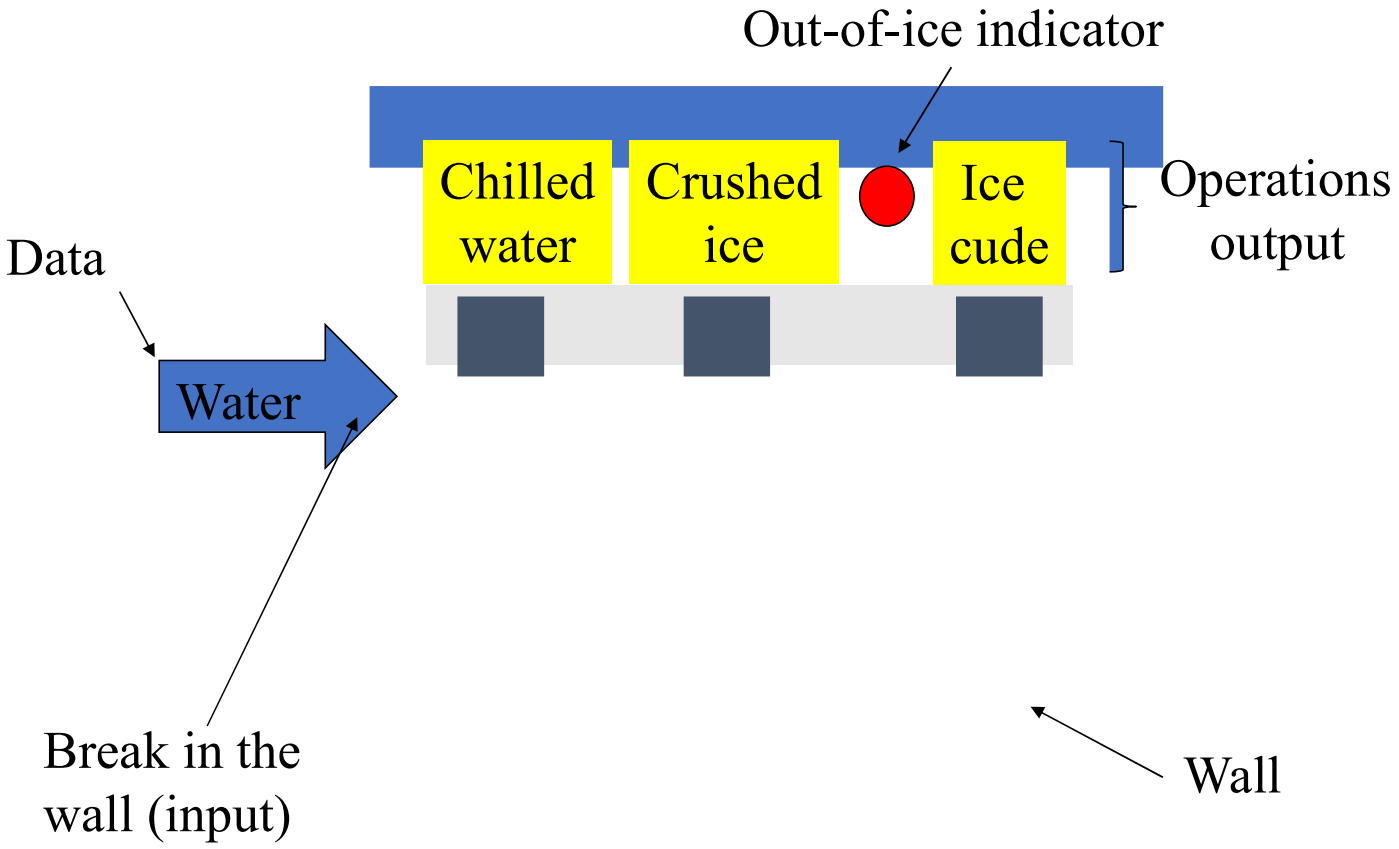
Abstract Data Types

- Abstract data type (ADT)
  - An ADT is composed of
    - A collection of data
    - A set of operations on that data
  - Specifications of an ADT indicate
    - What the ADT operations do, not how to implement them
  - Implementation of an ADT
    - Includes choosing a particular data structure

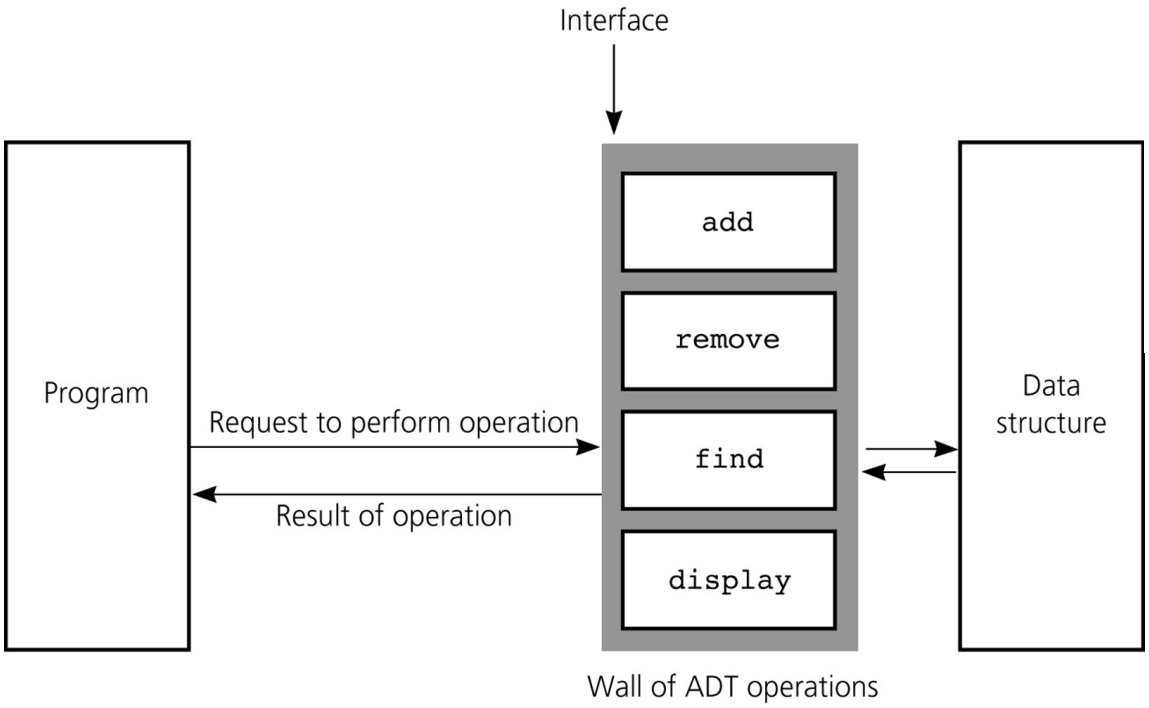
Abstract Data Types

- Data structure
  - A construct that is defined within a programming language to store a collection of data
  - Example: arrays
- ADTs and data structures are not the same
- Data abstraction
  - Results in a wall of ADT operations between data structures and the program that accesses the data within these data structures

ADT (Refrigerator ice dispenser)

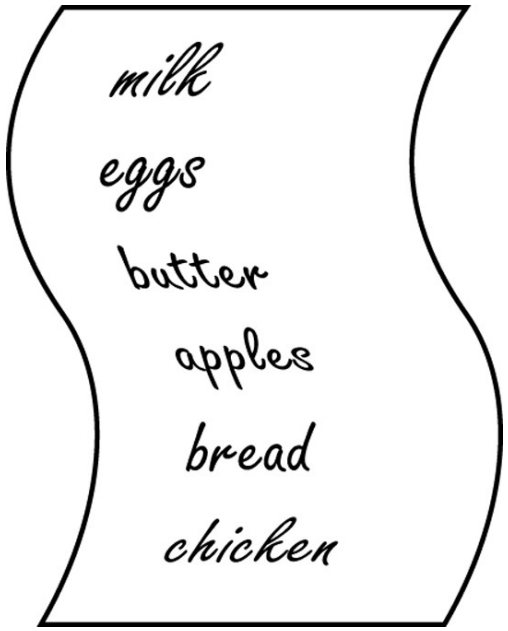


Abstract Data Types



A wall of ADT operations isolates a data structure from the program that uses it

Specifying ADTs



- In a list
  - Except for the first and last items, each item has
    - A unique predecessor
    - A unique successor
  - Head or front
    - Does not have a predecessor
  - Tail or end
    - Does not have a successor

The List Interface

A list stores elements in a sequential order, and allows the user to specify where the element is stored. The user can access the elements by index.

Lists

A list is a popular data structure to store data in sequential order. For example, a list of students, a list of available rooms, a list of cities, and a list of books, etc. can be stored using lists. The common operations on a list are usually the following:

- Retrieve an element from this list.
- Insert a new element to this list.
- Delete an element from this list.
- Find how many elements are in this list.
- Find if an element is in this list.
- Find if this list is empty.

Two Ways to Implement Lists

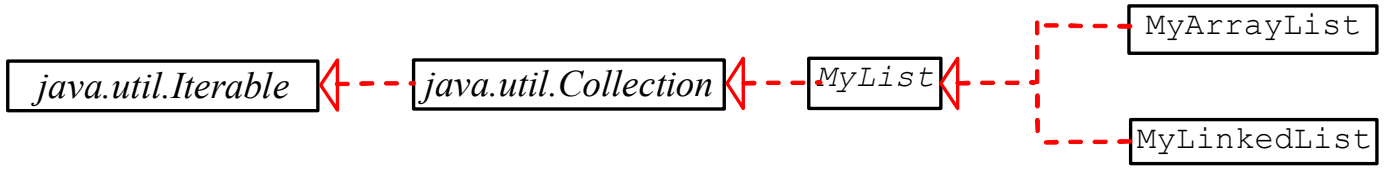
There are two ways to implement a list.

Using arrays. One is to use an array to store the elements. The array is dynamically created. If the capacity of the array is exceeded, create a new larger array and copy all the elements from the current array to the new array.

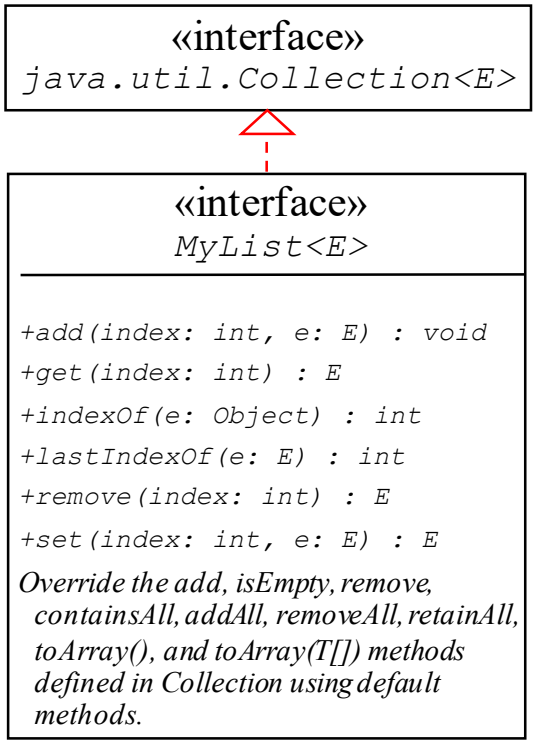
Using linked list. The other approach is to use a linked structure. A linked structure consists of nodes. Each node is dynamically created to hold an element. All the nodes are linked together to form a list.

Design of ArrayList and LinkedList

For convenience, let's name these two classes: MyArrayList and MyLinkedList. These two classes have common operations, but different data fields. The common operations can be generalized in an interface or an abstract class. Prior to Java 8, a popular design strategy is to define common operations in an interface and provide an abstract class for partially implementing the interface. So, the concrete class can simply extend the abstract class without implementing the full interface. Java 8 enables you to define default methods. You can provide default implementation for some of the methods in the interface rather than in an abstract class.



MyList Interface



Inserts a new element at the specified index in this list.

Returns the element from this list at the specified index.

Returns the index of the first matching element in this list.

Returns the index of the last matching element in this list.

Removes the element at the specified index and returns the removed element.

Sets the element at the specified index and returns the element being replaced.

MyList.java

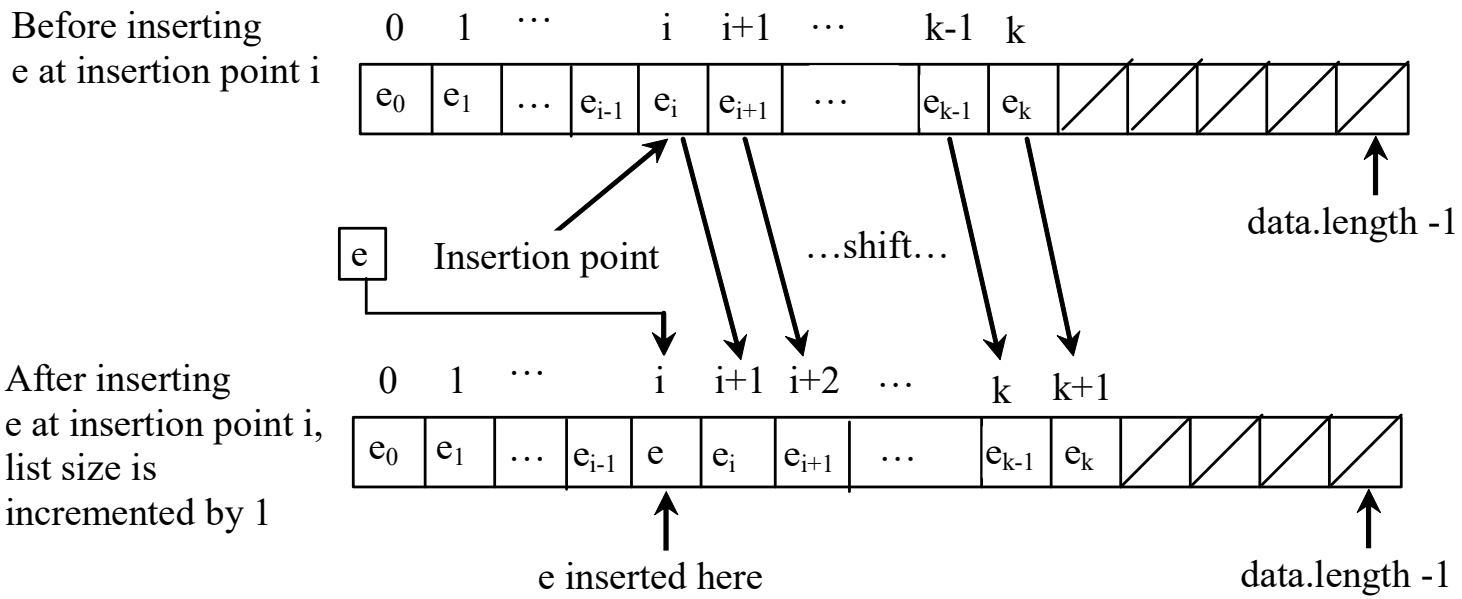
Array Lists

Array is a fixed-size data structure. Once an array is created, its size cannot be changed. Nevertheless, you can still use array to implement dynamic data structures. The trick is to create a new larger array to replace the current array if the current array cannot hold new elements in the list.

Initially, an array, say data of Object[] type, is created with a default size. When inserting a new element into the array, first ensure there is enough room in the array. If not, create a new array with the size as twice as the current one. Copy the elements from the current array to the new array. The new array now becomes the current array.

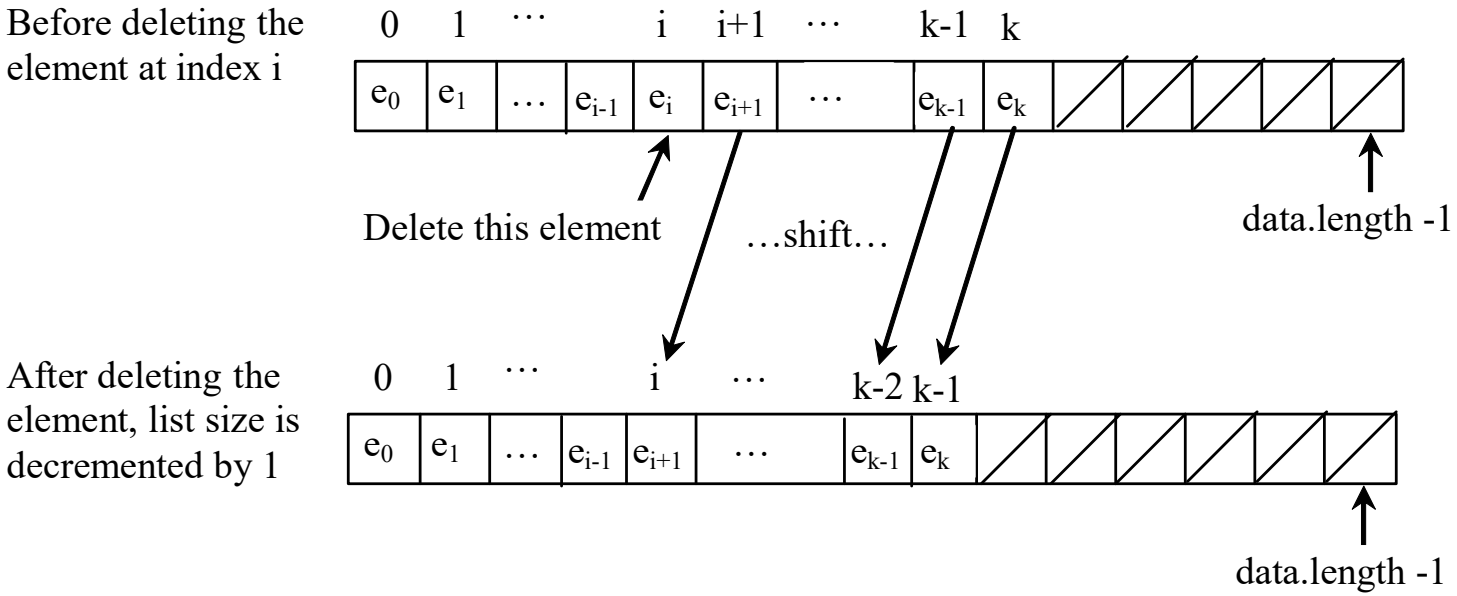
Insertion

Before inserting a new element at a specified index, shift all the elements after the index to the right and increase the list size by 1.

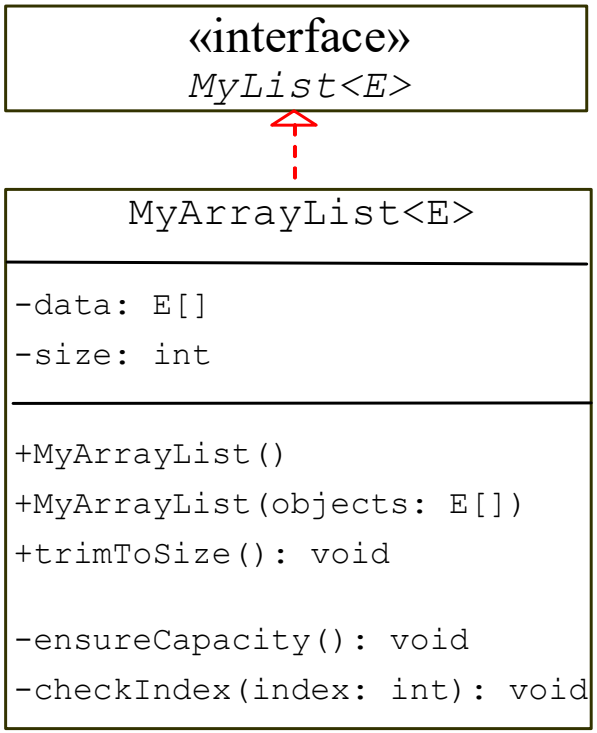


Deletion

To remove an element at a specified index, shift all the elements after the index to the left by one position and decrease the list size by 1.



Implementing MyArrayList



Array for storing elements in this array list.

The number of elements in the array list.

Creates a default array list.

Creates an array list from an array of objects.

Trims the capacity of this array list to the list's current size.

Doubles the current array size if needed.

Throws an exception if the index is out of bounds in the list.

MyArrayList.java