Introduction to Data Structure
Introduction

In order to process large quantities of data, you need to collect values in a data structure. This module introduces some most commonly used data structures in programming. It covers arrays and array lists, stack, queue and linked list data structures. In next module, we will learn more about how Java programming language handles these data structures.

Objectives

- To become familiar with arrays and array lists data structure
- To learn Wrappers classes and auto-boxing
- To learn some simple array algorithms
- To become familiar with stacks data structure
- To become familiar with queue data structure
- To become familiar with linked lists data structure

Array and Array List

An array is a sequence of values of the same type. Arrays are a rather primitive construct. An array variable can hold any data types such as int, double or object. The number of elements in an array is called its length. Each element in the array is specified by an integer index representing its position in the list.

In Java, the new operator is used to construct an array. For example, the following statement constructs an array of 10 floating-point numbers:

```java
double[] data = new double[10];
```

With the above declaration, the length of data is 10, the elements stored in the data will be

```java
data[0],data[1],data[2],…data[9].
```

Remember the index values of an array range from 0 to length-1. Accessing a non-existent element will result in a bounds error.

You can initialise an array by allocating it and then filling each entry, for example

```java
int[] primes = new int[3];
primes[0] = 2;
primes[1] = 4;
primes[2] = 6;
```

However, if you already know all the elements that you want to place in the array, there is an easier way, i.e., list all elements that you want to include in the array, enclosed in braces and separated by commas:

```java
int [] primes = {2,4,6};
```
The ArrayList class manages a sequence of objects. Compared with an array, array lists offer two significant conveniences:

- Array lists can grow and shrink as needed (you do not need to worry about the boundary)
- The ArrayList class supplies methods for many common tasks, such as inserting and removing elements

ArrayList is a standard Java class defined in java.util package.

To construct an ArrayList object, use

```java
ArrayList<E> al = new ArrayList<E>(); // E is the object type
```

The most common methods defined in an ArrayList class are:

- To add an object to the end of the array list:
  ```java
  boolean add (Object obj)   e.g. al.add (“QLD”);
  ```
  After an add operation, the size of the ArrayList is extended by 1.

- To get the current size of the array list:
  ```java
  int size ( )      e.g. al.size ();
  ```

- To get an object at index i:
  ```java
  Object get (int i)     e.g. al.get (2);
  ```

- To set an object obj at index i, (note this will overwrite whatever value was there before)
  ```java
  Object set (int i, Object obj)    e.g. al.set (2, “NSW”);
  ```

- To insert an object obj at index i, (note this will move all elements up by one position, from the current element at position i, to the last element in the array list, and increase the size of the array list by 1)
  ```java
  void add (int i, Object obj)   e.g. al.add(2, “VIC=”);
  ```

- To remove an element at index i, (note this will move all element after the removed element down by one position, and reduce the size of array list by 1):
  ```java
  Object remove (int i)    e.g. al.remove(1);
  ```

When inserting / removing an element in the middle of an ArrayList, all the elements from the current position to the end of the array list must be moved by 1 position (you do not need to do the shifting, the ArrayList class will do this moving for you). This is really inefficient if there are large numbers of elements in the list. In this case, a LinkedList data structure should be considered.

Please read Chapter section 8.1 and Chapter section 8.2 for array and ArrayList explanations. Please also read Chapter section 8.3, which is about how to wrap primitive data types into objects by using Wrappers classes.
Simple Array Algorithms

Simple array algorithms include Counting Matches, Finding a Value and Finding the Maximum or Minimum values in the array lists. They all need to go through the entire collection. The chapter 8.4 and 8.5 have details. The enhanced for loop is another style “for” loop. It doesn’t matter if you still want to use the original for loop.

Stack and Queues

Stacks and Queues are among the simplest of all data structures, but are also among the most important. Stacks and queues are used in a host of different applications that include many more sophisticated data structures. In addition, stacks and queues are among the few kinds of data structures that are often implemented in the hardware microinstructions inside a CPU.

A stack is a container of objects that are inserted and removed according to the last-in-first-out (LIFO) principle. Objects can be inserted into a stack at any time, but only the most-recently inserted (that is, “last”) object can be removed at any time. The name “stack” is derived from the metaphor of a stack of cafeteria plate dispenser. In this case, the fundamental operations involve the “pushing” and “popping” of plates on the stack. When we need a new plate from the dispenser, we “pop” the top plate off the stack, and when we add a plate, we “push” it down on the stack to become the new top plate.

A stack data structure example that you familiar with is Internet Web browser, which store the addresses of recently visited sites on a stack. Each time a user visits a new site, that site’s address is “pushed” onto the stack of addresses. The browser then allows the user to “pop” back to previously visited sites using the “back” button.

Another stack example is “undo” mechanism in text editors, which keep text changes in a stack. Java has a standard Stack class defined in java.util library. We will learn how to use this class in next module.

Queue is another fundamental data structure. It is a close “cousin” of the stack, as a queue is a container of objects that are inserted and removed according to the first-in-first-out (FIFO) principle. That is, elements can be inserted at any time, but only the element that has been in the queue the longest can be removed at any time. We usually say that elements enter the queue at the rear and are removed from the front. The metaphor for this terminology is a line of people waiting to get on an amusement park ride. People enter at the rear of the line and get on the ride from the front of the line.

Queue data structure can be used in transaction processing, incoming calls handling, printing jobs handling etc. The common operations to a queue data structure include en-queue and de-queue. En-queue insert object at the rear of the queue. De-queue remove and return object from the queue at the front.

There is no Queue class in the Java library, but it can be easily implemented by using a LinkedList class. We will see how to do it in next module.
Linked List

Link list is another important data structure used for collecting a sequence of objects. A linked list consists of a number of nodes, each of which has a reference to the next node. In a linked list structure, each node stores a value and a reference to the next node in the sequence. When you insert/remove a new node into/from a linked list, only the neighbouring node references need to be updated. This makes adding and removing elements in the middle of a linked list efficient.

Visiting the elements of a linked list in sequential order is efficient, but random access is not because you need to reach the preceding node first.

Reading

Text book:
  Chapter 8: Arrays and Array Lists
  Chapter 20: An introduction to Data structure

Review questions

Review exercises:
  Page 312 - 315: Exercise R8.1, R8.9, R8.16, R8.17, R8.18

Programming exercises:
  Page 315: Exercise P8.2, P8.3

References

Solution to Programming Exercises

P8-2
/**
 * This class tests the Purse class.
 */
public class ExP8_2
{
    public static void main(String[] args)
    {
        Purse p = new Purse();
        p.add("Quarter");
        p.add("Dime");
        p.add("Nickel");
        p.add("Dime");

        System.out.println(p.toString());
    }
}

import java.util.ArrayList;

/**
 * A purse holds a collection of coins.
 */
public class Purse
{
    /**
     * Constructs an empty purse.
     */
    public Purse()
    {
        coins = new ArrayList();
    }

    /**
     * Add a coin to the purse.
     * @param coinName the coin to add
     */
    public void add(String coinName)
    {
        coins.add(coinName);
    }

    /**
     * Displays the string representation of the object.
     * @return output the string representation of coins
     */
    public String toString()
if (coins.size() == 0)
    return "Purse[]";

String output = "Purse[";
for (String coin : coins)
{
    output = output + coin + ",";
}
output = output.substring(0, output.length() - 1); // remove the last ","
    return output + "]";

private ArrayList coins;

P8-3
/**
 * This class tests the Purse class.
 */
public class ExP8_3
{
    public static void main(String[] args)
    {
        Purse p = new Purse();
        p.add("Quarter");
        p.add("Dime");
        p.add("Nickel");
        p.add("Dime");

        System.out.println("Original purse: " + p.toString());
        p.reverse();
        System.out.println("Reversed purse: " + p.toString());
    }
}

import java.util.ArrayList;
/**
 * A purse holds a collection of coins.
 */
public class Purse
{
    /**
     * Constructs an empty purse.
     */
    public Purse()
```java
/**
 * Add a coin to the purse.
 * @param coinName the coin to add
 */
public void add(String coinName) {
    coins.add(coinName);
}

/**
 * Displays the string representation of the object.
 * @return output the string representation of coins
 */
public String toString() {
    if (coins.size() == 0)
        return "Purse[]";

    String output = "Purse["
    for (String coin : coins) {
        output = output + coin + ",";
    }
    output = output.substring(0, output.length() - 1); // remove the last ","
    return output + "]";
}

/**
 * Reverses the elements in the purse.
 */
public void reverse() {
    int i = 0;
    int j = coins.size() - 1;
    while (i < j)
    {
        String temp = coins.get(i);
        coins.set(i, coins.get(j));
        coins.set(j, temp);
        i++;
        j--;
    }
}

private ArrayList coins;
```