Array and ArrayList
Introduction
In order to process large quantities of data, you need to collect values in a data structure. This module introduces arrays and array lists 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

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
```
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.
Wrappers and Auto-boxing

Wrapper classes can wrap primitive type values into object types. All eight primitive data types have their correspondence wrapper classes. When working with Java classes, you often need change the primitive data type into class type. Since Java 5, this is automatically done, as called “auto-boxing”.

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.

Making Parallel Arrays into Arrays of Objects

Avoid parallel arrays by changing them into arrays of objects. (You should adopt this idea in your assignment)

Reading

Text book:
   Chapter 7: Arrays and Array Lists (whole chapter)

Review questions

Review exercises:
   Page 324 - 327: Exercise R7.3, R7.9, R7.15, R7.17, R7.18

Programming exercises:
   Page 327: Exercise P7.1 – the Bank class is a collection of bank account. Write a Bank class using ArrayList data structure.

References

Cay Horstmann, CH 2007, Big Java, 3rd Edition
Lab session

1) Below is a Person class example which contains two attributes of a Person object. Fill in the implementation blanks and then attempt the task a), b), and c) followed. This exercise demonstrates how useful to override a toString() method and how to manage Objects in different data structures.

```java
public class Person
{
    private String name;
    private int age;

    public Person(String str, int n)
    {
        //implementation
    }

    public String getName()
    {
        //implementation
    }

    public int getAge()
    {
        //implementation
    }

    public String toString()
    {
        return ("Name is " + name +"\t"+ "Age is " + age +"\n\n");
    }
}
```

Task a). Create a class PersonTester1 which contains a method main(), in this method, create three Person objects p1, p2, and p3. Then display these three persons’ name and age information.

Task b). Create a class PersonTester2 which contains a method main(), in this method, create six Person objects p1, p2, p3, p4, p5 and p6. Put these six objects into an array variable pArray, display these six persons’ name and age information. Then, find a specific person (such as John) from the pArray and display the person’s information.

Task c). Create a class PersonTester3 (you may modify PersonTester2) which contains a method main(), in this method, create six Person objects p1, p2, p3, p4, p5 and p6. Put these six objects into an ArrayList object pArrayList, display these six persons’ name and age information. Then find the youngest person from this pArrayList collection.
2) Exercise P7.1, this exercise covers ArrayList data structure and few most common used array algorithms. Inspect these programs and try to understand them

**Bank.java**

```java
import java.util.ArrayList;

/**
 * This bank contains a collection of bank accounts.
 */
public class Bank
{
    /**
     * Constructs a bank with no bank accounts.
     */
    public Bank()
    {
        accounts = new ArrayList<BankAccount>();
    }

    /**
     * Adds an account to this bank.
     * @param a the account to add
     */
    public void addAccount(BankAccount a)
    {
        accounts.add(a);
    }

    /**
     * Gets the sum of the balances of all accounts in this bank.
     * @return the total balance
     */
    public double getTotalBalance()
    {
        double total = 0;
        for (BankAccount a : accounts)
        {
            total = total + a.getBalance();
        }
        return total;
    }

    /**
     * Counts the number of bank account whose balance is at least a given value.
     * @param atLeast the balance required to count an account
     * @return the number of accounts having least the given balance
     */
    public int count(double atLeast)
    {
        int matches = 0;
        for (BankAccount a : accounts)
        {
            if (a.getBalance() >= atLeast) matches++;// found a match
        }
        return matches;
    }

    /**
     * Finds a bank account with a given number.
     * @param accountNumber the number to find
     * @return the account with the given number, or null if there is no such account
     */
    public BankAccount find(int accountNumber)
    {
        for (BankAccount a : accounts)
        {
            if (a.getAccountNumber() == accountNumber) // found a match
                return a;
        }
        return null; // no match in the entire array list
    }
}
```
/**
 * Gets the bank account with the largest balance.
 * @return the account with the largest balance, or null if the bank has no accounts
 */
public BankAccount getMaximum()
{
    if (accounts.size() == 0) return null;
    BankAccount largestYet = accounts.get(0);

    for (int i = 1; i < accounts.size(); i++)
    {
        BankAccount a = accounts.get(i);
        if (a.getBalance() > largestYet.getBalance())
            largestYet = a;
    }
    return largestYet;
}

/**
 * Add an account to the bank.
 * @param accountNumber the account number of this account
 * @param initialBalance the initial balance of this account
 */
public void addAccount(int accountNumber, double initialBalance)
{
    accounts.add(new BankAccount(accountNumber, initialBalance));
}

/**
 * Deposit money into an account.
 * @param accountNumber the account number
 * @param amount the amount to be deposited
 */
public void deposit(int accountNumber, double amount)
{
    find(accountNumber).deposit(amount);
}

/**
 * Withdraw money from an account.
 * @param accountNumber the account number
 * @param amount the amount to be withdrawn
 */
public void withdraw(int accountNumber, double amount)
{
    find(accountNumber).withdraw(amount);
}

/**
 * Get an account balance
 * @param accountNumber the account number
 * @return the account balance
 */
public double getBalance(int accountNumber)
{
    return find(accountNumber).getBalance();
}

private ArrayList<BankAccount> accounts;
BankAccount.java

/**
 * A bank account has a balance that can be changed by deposits and withdrawals.
 */

class BankAccount
{
    /**
     * Constructs a bank account with a zero balance.
     * @param anAccountNumber the account number for this account
     */
    public BankAccount(int anAccountNumber)
    {
        accountNumber = anAccountNumber;
        balance = 0;
    }

    /**
     * Constructs a bank account with a given balance.
     * @param anAccountNumber the account number for this account
     * @param initialBalance the initial balance
     */
    public BankAccount(int anAccountNumber, double initialBalance)
    {
        accountNumber = anAccountNumber;
        balance = initialBalance;
    }

    /**
     * Gets the account number of this bank account.
     * @return the account number
     */
    public int getAccountNumber()
    {
        return accountNumber;
    }

    /**
     * Deposits money into the bank account.
     * @param amount the amount to deposit
     */
    public void deposit(double amount)
    {
        double newBalance = balance + amount;
        balance = newBalance;
    }

    /**
     * Withdraws money from the bank account.
     * @param amount the amount to withdraw
     */
    public void withdraw(double amount)
    {
        double newBalance = balance - amount;
        balance = newBalance;
    }

    /**
     * Gets the current balance of the bank account.
     * @return the current balance
     */
    public double getBalance()
    {
        return balance;
    }

    private int accountNumber;
    private double balance;
}
BankTester.java

/**
 * This program tests the Bank class.
 */
public class BankTester
{
    public static void main(String[] args)
    {
        Bank bank = new Bank();

        int dannysAccount = 0;
        int sallysAccount = 1;
        int harrysAccount = 2;
        int jerrysAccount = 3;

        bank.addAccount(dannysAccount, 1000);
        bank.addAccount(sallysAccount, 2000);
        bank.addAccount(harrysAccount, 3000);
        bank.addAccount(jerrysAccount, 10000);

        bank.deposit(dannysAccount, 200);
        bank.withdraw(sallysAccount, 500);
        bank.deposit(harrysAccount, 1000);
        bank.withdraw(jerrysAccount, 7000);

        System.out.println(
            "Danny's Account Balance: " + bank.getBalance(dannysAccount));
        System.out.println("Expected: 1200");
        System.out.println(
            "Sally's Account Balance: " + bank.getBalance(sallysAccount));
        System.out.println("Expected: 1500");
        System.out.println(
            "Harry's Account Balance: " + bank.getBalance(harrysAccount));
        System.out.println("Expected: 4000");
        System.out.println(
            "Jerry's Account Balance: " + bank.getBalance(jerrysAccount));
        System.out.println("Expected: 3000");
    }
}