

4

Introducing Containers

Self-review Questions

4.1 Which of the following are valid array declarations?

```
int x = new int[10] ;
int[][] = new int[][10] ;
double []d = new double[10] ;
char[] s = "Hello" ;
```

The declaration:

```
double []d = new double[10] ;
```

is fine, although it would be much more usual to write it as:

```
double[] d = new double[10] ;
```

Whitespace is not significant in Java so the spacing between non-space elements is of no consequence. This does mean it is important to space things to aid human readability which is why the second form is preferred to the first form as it visually associates the [] with the **double** as the compiler does.

The declaration:

```
int x = new int[10] ;
```

causes a compilation error:

```
found   : int[]
required: int
    int x = new int[10] ;
           ^
2 errors
```

The problem here is that the variable is not of array type so the initialization is not possible.

```
int[][] = new int[][10] ;
```

The first problem here is that there is no variable name so the declaration is not well formed, it leads to an error like:

```
Trial.java:4: not a statement
int[][] = new int[][10] ;
    ^
Trial.java:4: ';' expected
int[][] = new int[][10] ;
    ^
2 errors
```

If we correct this to give:

```
int[][] xx = new int[][10];
```

then we still have an error:

```

Trial.java:4: ')' expected
    int[][] xx = new int[][10] ;
                        ^
Trial.java:4: array dimension missing
    int[][] xx = new int[][10] ;
                        ^
2 errors

```

The problem here is that we can only ever leave the last size blank. So:

```
int[][] xx = new int[10][] ;
```

compiles fine and on execution make xx a reference to a size 10 array of references to **int** arrays.

The declaration:

```
char[] s = "Hello" ;
```

results in the error message:

```

Trial.java:6: incompatible types
found   : java.lang.String
required: char[]
    char[] s = "Hello" ;
                ^
1 error

```

The problem is that the types on left and right hand side of the initialization are different. On the right we have a **String** on the left we have a **char[]** and the two are different. To correct this we have to use one of:

```
String s = "Hello" ;
char[] s = { 'H', 'e', 'l', 'l', 'o', '\0' } ;
```

4.2 What is array bounds checking?

It is a run time check that any index used in an index expression is greater than 0 and less than the maximum size of the array being indexed into.

4.3 What is the maximum index that can be used with an array of size 7?

6 because arrays are always 0-origin—a 7 item array has elements indexed by 0, 1, 2, 3, 4, 5 and 6.

4.4 How is a 3D array represented in terms of array objects?

4.5 Which of these are valid array indexing expressions for an array of size 10?

```
n[2.5]
n[0]
n[3-7]
n[2*3]
n[n[1]]
```

The expressions:

```
n[0]
n[2*3]
n[n[1]]
```

are all valid. The last expression however will lead to an `ArrayIndexOutOfBoundsException` if `n[1]` is negative or greater than 9.

The expression:

```
n[2.5]
```

is not valid as the index value is a **double** and all indexes must be positive and integral.

The expression:

```
n[3-7]
```

is not valid as the **int** expression results in -4 and negative indexes are not permitted.

4.6 When is it an advantage to use an `ArrayList` rather than an array?

Whenever there is any doubt about the exact size of the array at compile time.

4.7 Explain how array parameters work. What happens if an element of an array passed as a parameter to a method is changed by assignment? Do `ArrayLists` behave the same or differently?

The array elements are stored somewhere in memory and a reference to the array is passed from the caller to the callee.

As the caller has passed a reference to the array, the caller is using the same array elements which means the callee is changing the data that the caller had.

ArrayLists behave the same.

4.8 *What do opening and closing a file do?*

Opening a file causes the JVM and the underlying operating system to set up connections between the program and the file. These connections must always be set up before a program can read and/or write a file. Closing a file causes the connections to be terminated and all the JVM and operating system resources set up during opening to be released.

4.9 *How does eof work?*

Programming Exercises

4.1 *Write a program to read in 10 integers and store them in an array. Then display the contents of the array.*

```
public class E_4_1 {
    private void execute () {
        final int[] data = new int [10];
        final Input input = new Input ();
        System.out.print ( "Enter " + data.length + " int values : " );
        for ( int i = 0 ; i < data.length ; ++i ) { data[i] = input.nextInt ( ); }
        System.out.print ( "Values input were : " );
        for ( int v : data ) { System.out.print ( v + " " ); }
        System.out.println ( );
    }
    public static void main ( final String[] args ) {
        ( new E_4_1 ( ) ).execute ( );
    }
}
```

4.2 *Write a program to read in a sequence of integers until 'stop' is entered. Store the integers in an array. Then display the average of the numbers entered.*

```
public class E_4_2 {
    private int[] doInput ( ) {
        final Input in = new Input ( );
        int[] buffer = new int[2000];
    }
}
```

```

System.out.println ( "Type in up to " + buffer.length + " integers, and the input with the word stop." );
for ( int i = 0 ; i < buffer.length ; ++i ) {
    final String s = in.next ( ) ;
    if ( s.compareTo ( "stop" ) == 0 ) {
        // buffer is too large for the amount of data input so create a new array of the right
        // size for the number of items input and replace the current buffer.
        final int[] data = new int[i] ;
        System.arraycopy ( buffer , 0 , data , 0 , i ) ;
        buffer = data ;
        break ;
    }
    buffer[i] = Integer.parseInt ( s ) ;
}
return buffer ;
}
private double calculateMean ( final int[] data ) {
    int total = 0 ;
    for ( int i = 0 ; i < data.length ; ++i ) { total += data[i] ; }
    return ( (double) total ) / data.length ;
}
public static void main ( final String[] args ) {
    final E_4_2 object = new E_4_2 ( ) ;
    System.out.println ( "Mean of entered data is " + object.calculateMean ( object.doInput ( ) ) ) ;
}
}

```

4.3 Repeat Exercise 4.2 but use an ArrayList instead of an array.

```

import java.util.ArrayList ;
public class E_4_3 {
    private ArrayList<Integer> doInput ( ) {
        final Input in = new Input ( ) ;
        final ArrayList<Integer> buffer = new ArrayList<Integer> ( ) ;
        System.out.println ( "Type in integers and the input with the word stop." ) ;
        while ( true ) {
            final String s = in.next ( ) ;
            if ( s.compareTo ( "stop" ) == 0 ) { break ; }
            buffer.add ( Integer.parseInt ( s ) ) ;
        }
        return buffer ;
    }
    private double calculateMean ( final ArrayList<Integer> data ) {
        int total = 0 ;
        for ( int i = 0 ; i < data.size ( ) ; ++i ) { total += data.get ( i ) ; }
        return ( (double) total ) / data.size ( ) ;
    }
    public static void main ( final String[] args ) {
        final E_4_3 object = new E_4_3 ( ) ;
        System.out.println ( "Mean of entered data is " + object.calculateMean ( object.doInput ( ) ) ) ;
    }
}

```

4.4 A matrix can be represented using a 2D array. Write methods to perform matrix addition, subtraction and

multiplication. Each method should take two 2D arrays as a parameter and return a new 2D array containing the result. Use the methods in a test program to verify that they work correctly.

The following is a potential solution:

```

/**
 * Some methods for doing things with 2D rectangular arrays representing matrices. The data has to be of some
 * form of numeric type so for now we make doubles, later we can do this better.
 *
 * @author Russel Winder
 * @version 2006-09-10T18:58
 */
public class E_4_4 {
    public static boolean isRectangular ( final double[][] a ) {
        for ( int i = 1 ; i < a.length ; ++i ) {
            if ( a[i].length != a[0].length ) { return false ; }
        }
        return true ;
    }
    public static void checkSameDimensions ( final double[][] a , final double[][] b ) {
        if ( ! isRectangular ( a ) ) { throw new RuntimeException ( "a is not a rectangular array." ) ; }
        if ( ! isRectangular ( b ) ) { throw new RuntimeException ( "b is not a rectangular array." ) ; }
        if ( a.length != b.length ) { throw new RuntimeException ( "a and b do not have the same number of rows." ) ; }
        if ( a[0].length != b[0].length ) { throw new RuntimeException ( "a and b do not have the same number of columns." ) ; }
    }
    public static double[][] add ( final double[][] a , final double[][] b ) {
        checkSameDimensions ( a , b ) ;
        final double[][] r = new double[a.length][a[0].length] ;
        for ( int i = 0 ; i < a.length ; ++i ) {
            for ( int j = 0 ; j < a[0].length ; ++j ) {
                r[i][j] = a[i][j] + b[i][j] ;
            }
        }
        return r ;
    }
    public static double[][] subtract ( final double[][] a , final double[][] b ) {
        checkSameDimensions ( a , b ) ;
        final double[][] r = new double[a.length][a[0].length] ;
        for ( int i = 0 ; i < a.length ; ++i ) {
            for ( int j = 0 ; j < a[0].length ; ++j ) {
                r[i][j] = a[i][j] - b[i][j] ;
            }
        }
        return r ;
    }
    public static double[][] multiply ( final double[][] a , final double[][] b ) {
        if ( ! isRectangular ( a ) ) { throw new RuntimeException ( "a is not a rectangular array." ) ; }
        if ( ! isRectangular ( b ) ) { throw new RuntimeException ( "b is not a rectangular array." ) ; }
        if ( a[0].length != b.length ) { throw new RuntimeException ( "a and b are not multiplication compatible." ) ; }
        final double[][] r = new double[a.length][b[0].length] ;
        for ( int i = 0 ; i < a.length ; ++i ) {
            for ( int j = 0 ; j < b[0].length ; ++j ) {
                for ( int k = 0 ; k < b.length ; ++k ) {
                    r[i][j] += a[i][k] * b[k][j] ;
                }
            }
        }
    }
}

```

```

    }
    return r;
}
}
}

```

The following is a short program to test some of the features of the program, there needs to be a lot more to test it properly.

```

public class E_4_4_Test {
    private void assertEquals ( final double[][] expected , final double[][] actual ) {
        E_4_4.checkSameDimensions ( expected , actual );
        for ( int i = 0 ; i < expected.length ; ++i ) {
            for ( int j = 0 ; j < expected[0].length ; ++j ) {
                if ( expected[i][j] != actual[i][j] ) {
                    System.err.println ( "Got a " + actual[i][j] + " expected a " + expected[i][j] );
                    System.exit ( 1 );
                }
            }
        }
    }
}

public void addTest () {
    final double[][] a = new double[][] { { 1 , 1 } , { 1 , 1 } , { 1 , 1 } };
    final double[][] b = new double[][] { { 1 , 1 } , { 1 , 1 } , { 1 , 1 } };
    final double[][] r = new double[][] { { 2 , 2 } , { 2 , 2 } , { 2 , 2 } };
    assertEquals ( r , E_4_4.add ( a , b ) );
}

public void subtractTest () {
    final double[][] a = new double[][] { { 1 , 1 } , { 1 , 1 } , { 1 , 1 } };
    final double[][] b = new double[][] { { 1 , 1 } , { 1 , 1 } , { 1 , 1 } };
    final double[][] r = new double[][] { { 0 , 0 } , { 0 , 0 } , { 0 , 0 } };
    assertEquals ( r , E_4_4.subtract ( a , b ) );
}

public void multiplyTest () {
    double[][] a = { { 1 , 1 } };
    double[][] b = { { 1 } , { 1 } };
    double[][] r = { { 2 } };
    assertEquals ( r , E_4_4.multiply ( a , b ) );
    a = new double[][] { { 1 , 1 } , { 1 , 1 } };
    b = new double[][] { { 1 , 1 } };
    r = new double[][] { { 1 , 1 } , { 1 , 1 } };
    assertEquals ( r , E_4_4.multiply ( a , b ) );
    a = new double[][] { { 1 , 1 } , { 1 , 1 } , { 1 , 1 } };
    b = new double[][] { { 1 , 1 , 1 } , { 1 , 1 , 1 } };
    r = new double[][] { { 2 , 2 , 2 } , { 2 , 2 , 2 } , { 2 , 2 , 2 } };
    assertEquals ( r , E_4_4.multiply ( a , b ) );
}

public static void main ( final String[] args ) {
    final E_4_4_Test tester = new E_4_4_Test ();
    tester.addTest ();
    tester.subtractTest ();
    tester.multiplyTest ();
}
}

```

4.5 Extend program `AddUpColumns` in Section 4.2.10, page 110, to add a method that ensures the 2D array is a square array and employs this new method to ensure adding up the columns actually makes sense.

Although the question as printed said 'square', it should have said 'rectangular'. In the following we have created a method `ensureRectangular` that terminates execution if the array is not rectangular. The method is called as the first action in `totalColumns` since that is the most appropriate place of responsibility, i.e. it is the responsibility of the `totalColumns` method to ensure the array that it is given is a reasonable array to work with. We have added an extra test case to ensure that the method is called and traps a non-rectangular array. It has to be the last test case though as it terminates execution.

```
public class E_4_5 {
    private void ensureRectangular ( final double[][] array ) {
        for ( int i = 1 ; i < array.length ; ++i ) {
            if ( array[i].length != array[0].length ) {
                System.out.println ( "Array is not rectangular." );
                System.exit ( 1 ) ;
            }
        }
    }
}

private double[] totalColumns ( final double[][] array ) {
    ensureRectangular ( array ) ;
    final int nColumns = array[0].length ;
    final double[] totals = new double[nColumns] ;
    for ( int column = 0 ; column < nColumns ; ++column ) {
        totals[column] = 0.0 ;
        for ( int row = 0 ; row < array.length ; ++row ) {
            totals[column] += array[row][column] ;
        }
    }
    return totals ;
}

private void displayTotals ( final double[] totals ) {
    System.out.print ( "Totals are: " ) ;
    for ( int column = 0 ; column < totals.length ; ++column ) {
        System.out.print ( totals[column] + " " ) ;
    }
    System.out.println ( ) ;
}

public static void main ( final String[] args ) {
    final E_4_5 object = new E_4_5 ( ) ;
    double[][] testArray = {
        { 2.1 , 3.4 , 5.6 , 9.6 , 5.5 } ,
        { 4.2 , 3.5 , 6.6 , 1.9 , 3.2 } ,
        { 1.1 , 5.8 , 8.2 , 4.5 , 2.8 }
    } ;
    object.displayTotals ( object.totalColumns ( testArray ) ) ;
    testArray = new double[][] {
        { 2.1 , 3.4 , 5.6 , 9.6 , 5.5 } ,
        { 4.2 , 3.5 , 6.6 , 1.9 }
    } ;
    object.displayTotals ( object.totalColumns ( testArray ) ) ;
}
}
```

4.6 Rewrite program AddUpColumns in Exercise 4.5, to use ArrayLists instead of arrays.

```
import java.util.Arrays ;
import java.util.ArrayList ;
public class E_4_6 {
    private ArrayList<Double> totalColumns ( final ArrayList<ArrayList<Double>> array ) {
        final int nColumns = array.get ( 0 ).size ( ) ;
        final ArrayList<Double> totals = new ArrayList<Double> ( nColumns ) ;
        for ( int column = 0 ; column < nColumns ; ++column ) {
            totals.add ( 0.0 ) ;
            for ( int row = 0 ; row < array.size ( ) ; ++row ) {
                totals.set ( column , totals.get ( column ) + array.get ( row ).get ( column ) ) ;
            }
        }
        return totals ;
    }
    private void displayTotals ( final ArrayList<Double> totals ) {
        System.out.print ( "Totals are: " ) ;
        for ( Double d : totals ) { System.out.print ( d + " " ) ; }
        System.out.println ( ) ;
    }
    public static void main ( final String[] args ) {
        final E_4_6 object = new E_4_6 ( ) ;
        final ArrayList<ArrayList<Double>> testArray = new ArrayList<ArrayList<Double>> ( ) ;
        testArray.add ( new ArrayList<Double> ( Arrays.asList ( 2.1 , 3.4 , 5.6 , 9.6 , 5.5 ) ) ) ;
        testArray.add ( new ArrayList<Double> ( Arrays.asList ( 4.2 , 3.5 , 6.6 , 1.9 , 3.2 ) ) ) ;
        testArray.add ( new ArrayList<Double> ( Arrays.asList ( 1.1 , 5.8 , 8.2 , 4.5 , 2.8 ) ) ) ;
        object.displayTotals ( object.totalColumns ( testArray ) ) ;
    }
}
```

4.7 Rewrite program DisplayTextFile to read and display strings instead of characters.

It seems that this question is a bit redundant in that the version of DisplayTextFile on page 132 is actually the answer to this question. It appears that in updating the book, the original point behind this question was taken into the text itself, and we did not update the question.

```
/**
 * A program to display the contents of a text file.
 *
 * @author Graham Roberts
 * @author Russel Winder
 * @version 2004-11-03
 */
public class DisplayTextFile {
    public String getFileName ( ) {
        final Input in = new Input ( ) ;
        System.out.print ( "Enter name of file to display: " ) ;
        final String name = in.nextLine ( ) ;
        return name ;
    }
    public void display ( final String fileName ) {
        final FileInput in = new FileInput ( fileName ) ;
```

```

while ( in.hasNextLine () ) { System.out.println ( in.nextLine () ); }
in.close ();
}
public static void main ( final String[] args ) {
    final DisplayTextFile object = new DisplayTextFile ();
    object.display ( object.getFileName () );
}
}

```

4.8 Write a program to find the largest sized array you can use on your computer system.

The 'brute force' method might seem to be:

```

public class E_4_8_brute {
    private void findLargestArray () {
        int i = 1;
        while ( true ) {
            System.out.print ( "Trying " + ( i ) + "... " );
            final byte[] array = new byte[ i ];
            System.out.println ( "done." );
            ++i;
        }
    }
    public static void main ( final String[] args ) {
        final E_4_8_brute object = new E_4_8_brute ();
        object.findLargestArray ();
    }
}

```

but this is not actually a solution because it results (on a 3GHz, 1GB, Ubuntu machine under normal workstation usage) in a program that is indistinguishable from an infinite loop. It is not, in fact, an infinite loop, but it would take many, many, many, many hours to get to the point where the array size was too big.

The most obvious way of actually answering this question is to use an interactive program. By involving a human being we can avoid our program having to make any guesses about the maximum size of array. The user can apply sophisticated algorithms and heuristics to make the search relatively quick – the hint here is to employ *binary chop search*. So, using the program:

```

public class E_4_8 {
    private void findLargestArray () {
        final Input input = new Input ();
        while ( true ) {
            System.out.print ( "Enter a value to try: " );
            final int i = input.nextInt ();
            final byte[] array = new byte[i];
            System.out.println ( "Succeeded." );
        }
    }
    public static void main ( final String[] args ) {
        final E_4_8 object = new E_4_8 ();
    }
}

```

```

        object.findLargestArray ( ) ;
    }
}

```

we found that, on the aforementioned 3GHz, 1GB, Ubuntu machine under normal workstation usage, using JDK 1.5.0_06, the largest array was around 61860980 bytes. There was a certain amount of non-determinism – sometimes numbers would succeed and sometimes they would fail. Almost certainly something to do with the exact memory available at the instant the JVM was allocating the array.

Actually we didn't use the above program, we used:

```

public class E_4_8_exceptions {
    private void findLargestArray ( ) {
        final Input input = new Input ( ) ;
        while ( true ) {
            System.out.print ( "Enter a value to try: " ) ;
            final int i = input.nextInt ( ) ;
            try { final byte[] array = new byte[i] ; }
            catch ( OutOfMemoryError oome ) {
                System.out.println ( "Failed." ) ;
                continue ;
            }
            System.out.println ( "Succeeded." ) ;
        }
    }
    public static void main ( final String[] args ) {
        final E_4_8_exceptions object = new E_4_8_exceptions ( ) ;
        object.findLargestArray ( ) ;
    }
}

```

so as to avoid having to restart the Java system for every failure. However at this point we haven't covered exceptions so we you may not have come up with this. This reflects the difficulties of sequencing material. It would have been good to put exceptions earlier but . . .

4.9 *Create a program that writes a data file containing 1000 random integers between 1 and 25. Use the JDK class `Random` to create random numbers, or implement your own random number algorithm. Then write a second program that reads the integers from the file and counts how many of each integer is in the file.*

Writing our own random number generator is almost certainly a bad idea. Generating random numbers on a computer is a complicated issue – both in terms of the mathematics and in terms of the actual programming – and the standard Java library has some good solutions. We will just use them. From the documentation we see there is a class `Random` that we could use. Alternatively, we could use the method `Math.random`, but this is just a utility method for using `Random.nextDouble` in a certain way and as the question asks about random integers, and there is a method `Random.nextInt (int n)` which seems to do more of what we want, let us use that.

So now we are in a position to create the writer of numbers:

```

import java.util.Random ;
public class E_4_9_writer {
    private void writeNumbers () {
        final Random generator = new Random () ;
        final FileOutputStream output = new FileOutputStream ( "random_numbers" ) ;
        for ( int i = 0 ; i < 1000 ; ++i ) {
            // NB Random.nextInt generates in the range 0 <= x < n and the question requires the numbers to be in
            // the range 1 <= x <= 25.
            output.writeInt ( generator.nextInt ( 25 ) + 1 ) ;
            output.writeEndOfLine () ;
        }
        output.close () ;
    }
    public static void main ( final String[] args ) {
        final E_4_9_writer object = new E_4_9_writer () ;
        object.writeNumbers () ;
    }
}

```

Creating the reader is a question of reading in the numbers from the file and then counting them. We do this by having an array where the index of the array is the number being counted and the value stored in that element of the array is the count. The hard bit is that arrays have indexes starting with 0 but the data does not use that value. We cheat a bit and subtract 1 from each data item in order to know which element of the array to use as the counter. The really hard bit is, of course, to remember to add one when we print things out!

```

public class E_4_9_reader {
    private void displayCounts () {
        final int maxNumber = 25 ;
        final FileInputStream input = new FileInputStream ( "random_numbers" ) ;
        final int[] counts = new int[maxNumber] ;
        while ( input.hasNextInt () ) { ++counts[input.nextInt () - 1] ; }
        input.close () ;
        for ( int i = 0 ; i < maxNumber ; ++i ) { System.out.println ( "counts [ " + ( i + 1 ) + " ] = " + counts[i] ) ; }
    }
    public static void main ( final String[] args ) {
        final E_4_9_reader object = new E_4_9_reader () ;
        object.displayCounts () ;
    }
}

```

You will see when you run this, that although the counts of the numbers are all reasonably close together they are not the same. Herein lies the problem with computer-generated random numbers: they aren't actually random and it is hard with only a small number (1000 is small in this sort of situation) to get a good distribution. However, to go further is to diverge far from the purpose of this book. There is a mass of literature on pseudo-random and quasi-random numbers. If you are interested it is well worth hunting out.

4.10 Using the data file created in the last question, write a drawing program that plots a bar chart showing how many of each integer between 1 and 25 appears in the file.

- 4.11** Write a method that takes two character array parameters and returns true if the sequence of characters stored in the second array is a subsequence of those characters stored in the first array.

We can create an answer to this question using indexing and iteration:

```
import java.util.Arrays ;
public class E_4_11 {
    private boolean containsSequence ( final char[] a , final char[] b ) {
        if ( a.length >= b.length ) {
            for ( int i = 0 ; i <= a.length - b.length ; ++i ) {
                boolean areTheSame = true ;
                for ( int j = 0 ; j < b.length ; ++j ) {
                    if ( a[ i + j ] != b[ j ] ) { areTheSame = false ; break ; }
                }
                if ( areTheSame ) { return true ; }
            }
        }
        return false ;
    }
    private void assertTrue ( final boolean b , final String message ) {
        if ( ! b ) { System.out.println ( "Test failed: " + message ) ; }
    }
    private void runTest ( ) {
        char[] a = { 'a' , 'b' , 'c' , 'd' , 'e' , 'f' } ;
        char[] b = { 'a' , 'b' , 'c' , 'd' } ;
        assertTrue ( containsSequence ( a , a ) , "a == a" ) ;
        assertTrue ( containsSequence ( b , b ) , "b == b" ) ;
        assertTrue ( containsSequence ( a , b ) , "b is in a" ) ;
        assertTrue ( ! containsSequence ( b , a ) , "a is longer than b" ) ;
        char[] c = { 'c' , 'd' } ;
        assertTrue ( containsSequence ( c , c ) , "c == c" ) ;
        assertTrue ( containsSequence ( a , c ) , "c is in a" ) ;
        assertTrue ( ! containsSequence ( c , a ) , "a is longer than c" ) ;
        assertTrue ( containsSequence ( b , c ) , "c is in b" ) ;
        assertTrue ( ! containsSequence ( c , b ) , "b is longer than c" ) ;
        char[] d = { 'b' , 'z' } ;
        assertTrue ( containsSequence ( d , d ) , "d == d" ) ;
        assertTrue ( ! containsSequence ( a , d ) , "d is not in a" ) ;
        assertTrue ( ! containsSequence ( b , d ) , "d is not in b" ) ;
        assertTrue ( ! containsSequence ( c , d ) , "d is not in c" ) ;
    }
    public static void main ( final String[] args ) {
        ( new E_4_11 ( ) ).runTest ( ) ;
    }
}
```

Note how we are written lots of tests for the method. In fact, these proved crucial since the first version of the method that was written failed the test labelled "Position3". The problem was that we had written `i < a.length - b.length` instead of `i <= a.length - b.length`. A minor slip, but a major error nonetheless. The tests found this and so stopped us issuing a program with a bug in it.

There is also a 'trick' solution for this question:

```
private boolean containsSequence ( final char[] a , final char[] b ) {
```

```

    return ( new String ( a ) ).contains ( new String ( b ) );
}

```

This works by converting each character array to a `String` and then using the method `String.contains` which tests to see if the parameter `String` is a subsequence of the `String` the method is called on.

Which of these two answers is better is a moot point. Without a context in which to compare and contrast the solutions, asking the question 'Which is better?' is fairly meaningless.

4.12 Write a method that takes two character array parameters and returns true if both arrays contain the same characters but not necessarily in the same order.

If the arrays contain the same characters then it implies that they are the same length and that one array is just a permutation of the other other array. For two sequences to be permutations of one another, the count of each value present in either sequences must be the same in both sequences. The 'brute force' method is to loop over one of the sequences checking the counts in both sequences of the each value in the chosen sequence. This involved an inner loop to pass over each item in both sequences creating a count.

```

import java.util.Arrays ;
public class E_4_12 {
    private boolean isPermutation ( final char[] a , final char[] b ) {
        if ( a.length != b.length ) { return false ; }
        for( int i = 0 ; i < a.length ; ++i ) {
            int countA = 0 ;
            int countB = 0 ;
            char c = a[i] ;
            for ( int j = 0 ; j < a.length ; ++j ) {
                if ( c == a[j] ) { ++countA ; }
                if ( c == b[j] ) { ++countB ; }
            }
            if ( countA != countB ) { return false ; }
        }
        return true ;
    }
    private void assertTrue ( final boolean b , final String message ) {
        if ( ! b ) { System.out.println ( "Test failed: " + message ) ; }
    }
    private void runTest ( ) {
        char[] a = { 'a' , 'b' , 'c' , 'd' , 'e' , 'f' } ;
        char[] b = { 'a' , 'b' , 'c' , 'd' } ;
        assertTrue ( isPermutation ( a , a ) , "a is permutation of a" ) ;
        assertTrue ( isPermutation ( b , b ) , "b is permutation of b" ) ;
        assertTrue ( ! isPermutation ( a , b ) , "a is different from b" ) ;
        assertTrue ( ! isPermutation ( b , a ) , "b is different from a" ) ;
        char[] c = { 'f' , 'e' , 'd' , 'c' , 'b' , 'a' } ;
        assertTrue ( isPermutation ( c , c ) , "c is permutation of c" ) ;
        assertTrue ( isPermutation ( a , c ) , "a is permutation of c" ) ;
        assertTrue ( isPermutation ( c , a ) , "c is permutation of a" ) ;
        char[] d = { 'a' , 'a' , 'a' , 'b' } ;
        char[] e = { 'a' , 'b' , 'b' , 'b' } ;
    }
}

```

```

    assertTrue ( isPermutation ( d , d ) , "d is permutation of d" );
    assertTrue ( isPermutation ( e , e ) , "e is permutation of e" );
    assertTrue ( ! isPermutation ( d , e ) , "d is different from e" );
    assertTrue ( ! isPermutation ( e , d ) , "e is different from d" );
}
public static void main ( final String[] args ) {
    ( new E_4_12 ( ) ).runTest ( );
}
}

```

This is actually quite inefficient if there are repeated values since it implies recalculating the count for the same value.

An alternative approach if the two sequences are permutations is to sort them in which case they should now be equal. Well we can't sort the originals since that would disturb the original data, and that is unacceptable. So, we have to create copies (clone) the original and then we can sort the clones.

```

private boolean isPermutation ( final char[] a , final char[] b ) {
    if ( a.length != b.length ) { return false ; }
    char[] aa = (char[]) a.clone ( );
    Arrays.sort ( aa ) ;
    char[] bb = (char[]) b.clone ( );
    Arrays.sort ( bb ) ;
    return Arrays.equals ( aa , bb ) ;
}

```

Many people would consider this less good because of the resources required to make the clones and sort them. In fact, this may be a false position. The counting method is an $O(n^2)$ algorithm, we have two nested loops both of which loop over the entire sequence. Cloning an array is $O(n)$, comparing arrays is $O(n)$, and sorting them is $O(n \log_2(n))$ for a total of $O(n(2 + \log_2(n)))$ which is $O(n \log_2(n))$. So this method may actually be more efficient! For small arrays other factors dominate, but for very large arrays the difference could well be significant.

If it turns out that the number of items is not actually an issue then there is a 'trick' solution. If the number of times a character appears is not relevant, then we can use a set data structure (we will use `HashSet`) and ask if the two sets are equal:

```

import java.util.Arrays ;
import java.util.HashSet ;
public class E_4_12_trick {
    private boolean containsTheSame ( final Character[] a , final Character[] b ) {
        final HashSet<Character> hA = new HashSet<Character> ( Arrays.asList ( a ) ) ;
        final HashSet<Character> hB = new HashSet<Character> ( Arrays.asList ( b ) ) ;
        return hA.equals ( hB ) ;
    }
    private void assertTrue ( final boolean b , final String message ) {
        if ( ! b ) { System.out.println ( "Test failed: " + message ) ; }
    }
    private void runTest ( ) {
        Character[] a = { 'a' , 'b' , 'c' , 'd' , 'e' , 'f' } ;

```

```

Character[] b = { 'a', 'b', 'c', 'd' };
assertTrue ( containsTheSame ( a , a ) , "a contains the same as a" );
assertTrue ( containsTheSame ( b , b ) , "b contains the same as b" );
assertTrue ( ! containsTheSame ( a , b ) , "a does not contain the same as b" );
assertTrue ( ! containsTheSame ( b , a ) , "b does not contain the same as a" );
Character[] c = { 'P', 'e', 'd', 'c', 'b', 'a' };
assertTrue ( containsTheSame ( c , c ) , "c contains the same as c" );
assertTrue ( containsTheSame ( a , c ) , "a contains the same as c" );
assertTrue ( containsTheSame ( c , a ) , "c is permutation of a" );
Character[] d = { 'a', 'a', 'a', 'b' };
Character[] e = { 'a', 'b', 'b', 'b' };
assertTrue ( containsTheSame ( d , d ) , "d contains the same as d" );
assertTrue ( containsTheSame ( e , e ) , "e contains the same as e" );
assertTrue ( containsTheSame ( d , e ) , "d does not contain the same as e" );
assertTrue ( containsTheSame ( e , d ) , "e does not contain the same as d" );
}
public static void main ( final String[] args ) {
    ( new E_4_12_trick ( ) ).runTest ( );
}
}

```

Notice that we have made the data `Character` arrays in this version. We do this because we are making use of `Collections` classes and these cannot use primitive types. The trick here is to use the method `Arrays.asList` to create a `Collections` view of the array so that we can use them as the initializing data for the newly created `HashSets`. This creates two sets that can be compared. Sets have a single instance of any value so creating the sets from the arrays removes any duplicates. Comparing the sets for equality asks the question 'Do the two sets contain one or more instances of the same values?' which is close the question we were asked to answer.

- 4.13** Amend the `AverageUsingArrayList` program (Section 4.3.4, page 119) to take its input from a file instead of the keyboard. (It may well be best to start with the final version of the program discussed in that section!)
- 4.14** Amend the `AddUpColumns` program (Section 4.2.10, page 110) and the solution to Exercise 4.6 so as to read the numbers to total from a file.
- 4.15** Write a program that reads a text file and counts the number of times a selected word appears in the file.
- 4.16** Write a program that reads a text file containing Java source code and verifies that there is a matching number of opening and closing braces (curly brackets). Beware of braces appearing in comments, or in character literals and constants.

- 4.1** Write a program that sorts an array of integers into ascending order. Include a sort method that you write yourself, rather than using one from the JDK.
- 4.2** Write a program to act as a simple address book, storing names and addresses in a data file.

The following program is a basic one-class program implementing an address book storing a list of names and addresses. It provides a core set of operations to add, remove and search entries, along with the ability to load and save the entries from and to a text file. A simple text based user interface is implemented.

Two `ArrayList<String>`s are used to store the name and addresses, relying on the corresponding element in each `ArrayList` to represent the same address book entry. An entry is stored in a data file as two lines, the first containing the name string and the second the address string. Loading a data file appends the contents of the file to the current address book.

```

/**
 * A simple address book program that allows a collection of names and addresses to be added,
 * deleted, searched, stored in a file and appended from a file.
 *
 * @author Graham Roberts
 * @version 2006-11-02T21:42
 */

import java.util.ArrayList ;

public class C_4_2 {
    private static final int SEARCH = 1 ;
    private static final int LIST = 2 ;
    private static final int ADD = 3 ;
    private static final int REMOVE = 4 ;
    private static final int SAVE = 5 ;
    private static final int APPEND = 6 ;
    private static final int QUIT = 7 ;

    private Input in = new Input ( ) ;
    private ArrayList<String> names = new ArrayList<String> ( ) ;
    private ArrayList<String> addresses = new ArrayList<String> ( ) ;

    public void go ( ) {
        boolean quit = false ;
        while ( !quit ) {
            displayMenu ( ) ;
            int item = getMenuOption ( ) ;
            if ( item == QUIT ) { quit = true ; }
            else { doOption ( item ) ; }
        }
    }

    public void displayMenu ( ) {
        System.out.println ( "\nAddress Book" ) ;
        System.out.println ( SEARCH + ". Search for an address" ) ;
        System.out.println ( LIST + ". List all entries in the address book" ) ;
    }
}

```

```
System.out.println ( ADD + ". Add a new name and address" );
System.out.println ( REMOVE + ". Remove a name and address" );
System.out.println ( SAVE + ". Save address book to a file" );
System.out.println ( APPEND + ". Append an address book from a file" );
System.out.println ( QUIT + ". Quit" );
}

public int getMenuOption ( ) {
    while ( true ) {
        System.out.print ( "\nEnter selection: " );
        if ( in.hasNextInt ( ) ) {
            int temp = in.nextInt ( );
            in.nextLine ( );
            return temp ;
        }
        in.nextLine ( );
        System.out.println( "Menu option not recognised, please try again" );
    }
}

public void doOption ( int item ) {
    switch ( item ) {
        case SEARCH :
            searchForEntry ( );
            break ;
        case LIST :
            listAllEntries ( );
            break ;
        case ADD :
            addEntry ( );
            break ;
        case REMOVE :
            removeEntry ( );
            break ;
        case SAVE :
            saveAddressBook ( );
            break ;
        case APPEND :
            appendAddressBook ( );
            break ;
        default :
            System.out.println( "\nSorry - don't recognise that selection, try again" );
    }
}

public void searchForEntry ( ) {
    System.out.println( "\nSearch address book" );
    System.out.print ( "Enter a name: " );
    String name = in.nextLine ( );
    int index = names.indexOf(name) ;
    if (index != -1) { System.out.println( "The address is: " + addresses.get ( index ) ); }
    else { System.out.println( "Sorry - nothing found" ); }
}

public void listAllEntries ( ) {
    if ( names.size ( ) == 0 ) {
```

```

        System.out.println( "There are no entries in the address book" );
        return ;
    }
    for ( int i = 0 ; i < names.size ( ) ; i++ ) {
        String name = names.get ( i ) ;
        String number = addresses.get ( i ) ;
        System.out.println( "Entry " + ( i + 1 ) + ": " ) ;
        System.out.println( " Name: " + name ) ;
        System.out.println( " Address: " + number ) ;
    }
}

public void addEntry ( ) {
    System.out.println ( "\nAdd entry to address book" ) ;
    System.out.print ( "Enter a name: " ) ;
    String name = in.nextLine ( ) ;
    System.out.print ( "Enter an address: " ) ;
    String address = in.nextLine ( ) ;
    names.add ( name ) ;
    addresses.add ( address ) ;
}

public void removeEntry ( ) {
    System.out.println ( "\nRemove entry from address book" ) ;
    System.out.print ( "Enter a name: " ) ;
    String name = in.nextLine ( ) ;
    remove( name ) ;
}

public void remove ( String name ) {
    int index = names.indexOf ( name ) ;
    if ( index != -1 ) {
        names.remove ( index ) ;
        addresses.remove ( index ) ;
        System.out.println ( "The entry for: " + name + " has been removed" ) ;
    }
    else { System.out.println ( "Name not found - nothing removed" ) ;
    }
}

public void saveAddressBook ( ) {
    System.out.println ( "\nSave address book to a file" ) ;
    System.out.print ( "Enter the file name: " ) ;
    String fileName = in.nextLine ( ) ;
    FileOutputStream out = new FileOutputStream ( fileName ) ;
    writeToFile ( out ) ;
    out.close ( ) ;
}

public void writeToFile ( FileOutputStream file ) {
    for ( int i = 0 ; i < names.size ( ) ; i++ ) {
        file.writeString ( names.get ( i ) ) ;
        file.writeEndOfLine ( ) ;
        file.writeString ( addresses.get ( i ) ) ;
        file.writeEndOfLine ( ) ;
    }
}

```

```
}

public void appendAddressBook ( ) {
    System.out.println ( "\nAppend address book from a file" );
    System.out.print ( "Enter the file name: " );
    String fileName = in.nextLine ( );
    FileInput fileIn = new FileInput ( fileName );
    readFromFile ( fileIn );
    fileIn.close ( );
}

public void readFromFile ( FileInput file ) {
    while ( file.hasNextLine ( ) ) {
        String name = file.nextLine ( );
        String address = file.nextLine ( );
        names.add ( name );
        addresses.add ( address );
    }
}

public static void main(String[] args) {
    new C_4_2 ( ).go ( );
}
}
```

- 4.3** Write a program that reads a text file and builds an `ArrayList` containing each distinct word found in the file, in sorted order, along with a count of how many times each word appears.