14 Arrays

1 Index mapping

You are expected already to have a good understanding of arrays. We examine some array techniques in more detail, beginning with index mapping. Index mapping is a technique for compact array code by employing rich expressions in indices. Suppose, say, we want read a text and print the frequencies of word-lengths in the text. An example of output is:

```
1-letter words: 3
2-letter words: 5
3-letter words: 9
4-letter words: 4
6-letter words: 1
```

```
class WordCount {

    public static void main(String[] args) {
        int[] count = new int[20]; // word lengths up to 19, say
        // zero counts
        int i = 0;
        while (i<count.length) {
            count[i] = 0; i++;
        }
        // Read words & keep track of frequencies
        while (! Console.endOfFile()) {
            String word = Console.readToken();
            count[word.length()]++;
        }
        // Print result
```

```
            i = 1;
            while (i<count.length) {
               if (count[i]>0)
                  System.out.println(i + "-letter words: " + count[i]);
               i++;
            }
        }
}
```

Each word read is mapped to the index of the appropriate component of count[] – component i of count[] records the number of words of length i (component 0 is not used). Each word is read into string variable word, and hence the word's length is word.length(). If this is 4, say, then we must increment word[4]. Note that the length operator for arrays has no brackets, unlike that for arrays.

2 Accessing arrays using for-each loops

For-each loops provide a convenient way to access all the values in an array. For example, suppose a program contains the integer array `myArray`; the values in myArray are printed using the following for-each loop

```
    for (int k: myArray)
        System.out.println(k);
```

We could have also expressed this a for-loop as follows, although it is a little more cumbersome:

```
    for (int i=0; i<myArray.length; i++)
        System.out.println(myArray[i]);
```

The for-loop is in turn just a shorthand for the following while-loop

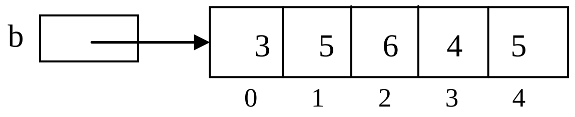
```
    int i = 0;
    while (i<myArray.length) {
        System.out.println(myArray[i]);
        i++;
    }
```

While loops are completely general, for-loops are more restrictive but a little more convenient when, and for-each loops are again more convenient but are even more limited in their applicability. For example, it is not possible to use a for-each loop to assign a value to every element of an array. None of the loops in the word count program above can be written using a for-each loop.

3 Arrays as objects

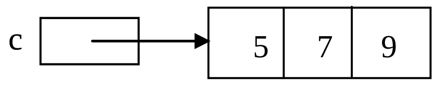
Java treats arrays as objects. Hence an array variable does not literally contain an array, but a reference to one. For example, the declaration below can be envisaged as shown:

```
int[] b = {3, 5, 6, 4, 5};
```



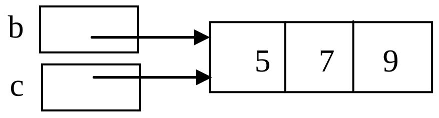
Suppose we have if addition to the declaration of b above, array c declared as follows:

```
int[] c = {5, 7, 9};
```



Then the assignment

```
b = c;
```



is effected by copying the reference to the array, as illustrated.

4 Arrays as parameters and return types

Arrays may occur as parameters of methods, or as return types, as in the following example to make a copy of an array.

```
static int[] copy(int[] b) {
        int[] r = new int[b.length];
        for (int i=0; i<r.length; i++)
                r[i] = b[i];
        return r;
}
        .....
        int[] c = {5, 7, 9};
        int[] d = copy(c); // note: no "= new int[...]" needed here
```

The array parameter in the following example is assigned to in the method body. As an exercise trace the execution of the program; state the output produced, justify it, and then test your answer (you may be surprised at first!)

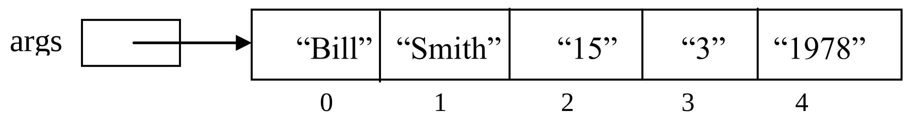
```
    static void incrementAll(int[] b) {
        for (int i=0; i<b.length; i++)
            b[i]++;
    }
    .....
        int[] c = {5, 7, 9};
        incrementAll(c);
        System.out.println(c[0] + " " + c[1] + " " + c[2]);
```

5 Command line arguments

You can pass arguments to a program from the command line. The arguments are formed into an array of strings and passed to the program. For example, if we issue the command

```
java MyProgram Bill Smith 15 3 1978
```

where class MyProgram has method `public static void main(String[] args)` the following array is made available in main()



Note that every item is passed as a string, even the integers (you can convert them back to type int using Integer.parseInt()). We illustrate with a program which sums a list of integers supplied at the command line. The following is a typical invocations of the program:

```
java AddInts 7  13  -4  10  13
```

```
class AddInts {
  public static void main(String[] args) {
    int sum = 0;
```

```
      for (String s: args) {
         sum = sum + Integer.parseInt(s);
      }
      System.out.println(sum);
   }
}
```

Warning: remember that command line arguments are *not* read from the keyboard by the executing program – they are supplied in the command line when the program is invoked.