Queen Mary
University of London

Procedural Programming
ECS401

Workbook
20016-17

Part 2
Units 5 to 7

Paul Curzon

School of Electronic Engineering
and Computer Science

Unit 5: Arrays and Linear Search

Learning Outcomes

Once you have done the reading and the exercises you should be able to:

- write programs that process bulk data using arrays
- write programs that search arrays for information
- explain how arrays are declared and used

From last week …

Before moving on to this section you should be able to:

- write programs that follow instructions a fixed number of times
- explain what is meant by a for loop
- trace the execution of programs containing for loops

A program from last week:

You should be able to write code like the following

```
int sum = 0;
for (int i = 1; i<=5; i++)
{
    sum = sum + i;
}
```

You should also be able to work out what the above code does by doing a programmer's walkthrough.

Processing bulk data

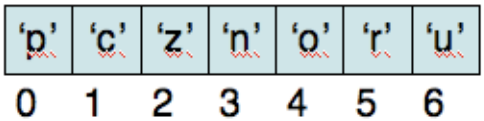
- We've covered the basic control structures: the ways of controlling what the computer does.
- We've also seen how to create records - essentially our own simple data types
- We still need to cover some basic data structures.
- In particular we have no way of processing "bulk" data in a program.
- The only storage we have is given by the variables in the program.
- They are all individual, and there aren't that many of them.

Lots of variables?

- We can put each separate piece of data in a separate variable.
- Each variable holds one piece of data
- This is no use if, say, we want to hold data for: 200 students, 1024x768 pixels or 1000,000,000 volume elements
- `int age1; int age2; ... int age200;`
- What we need is the **ARRAY**.

Arrays

- The array is the most basic of several different ways for holding bulk data.
- (Though lovers of list data structures would argue about that!)
- An array is a sequence of things, *all of the same type*, indexed (ie with position given) by numbers,.
- An array of characters:



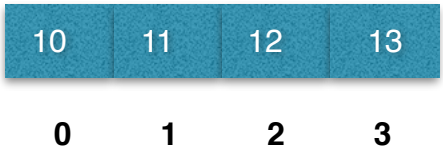
Arrays

Think of an array variable as a box (like any variable) but that has lots of identical compartments within it.

- You tell which array you mean from the name of the array. You tell which compartment by a number (the index) giving its position in the array.
- Can have arrays of Strings, booleans, integers…
- Don't get confused by the **value** stored in some position in the array and the **index** (the position - which compartment in the array)

Values versus positions

- This integer array called A has indexes 0, 1, 2, 3 but values stored 10,11,12,13



Array Indexing

- In Java arrays are always indexed by numbers starting at 0.
- You can't control the indexes used (though other languages are more flexible)
- An array of length 10 has indices 0. . . 9
- An array of length 4 has indices 0. . . 3
- An array of length 128 has indices 0. . . 127
- There is a reason for this. . .

Array indexing

- In the language C (from which Java is derived), the compiler always knows exactly the amount of storage required by any variable.
- So all the cells in an array have the same size.
- So the address of cell i is fst + i * size and this kind of thing can be done very efficiently by what is known as "offset addressing" in hardware.
- So its like that because its easier to convert it into fast machine code.

Array Declaration

- Arrays have to be declared like other variables.
- In Java arrays are declared like this:

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

- It declares an integer array of size 10 called myarray.
- int[] is the type of integer arrays.
- other types of array are similar
- In effect this gives you 10 integer variables called:
- myarray[0], myarray[1], . . . , myarray[9]
- It does NOT initialise those variables.

Manipulating array values

- So if you think of arrays like that as just a series of separate compartment variables where the names of each just have numbers in square brackets, you can use them like other variables:

```
int[] myarray = new int[10];
myarray[5] = 17;
System.out.println(myarray[5]);
int x = myarray[6]+myarray[7];
```

Variables for the index

- However we can now do something special with the index values
- Something we couldn't do if we used separate variables like age0,age1 etc
- We can use other variables for the index as we will see ..

Initializing Arrays

- If you want to initialise array variables, you could do it the hard way, by hand but that is really slow:

```
int [] myarray = new int[4];

myarray[0] = 0;
myarray[1] = 0;
myarray[2] = 0;
myarray[3] = 0;
```

- That fills each slot with a zero

Arrays mirror for loops

- We are doing the same thing over and over with a minor difference…we want a loop!
- The only difference between lines was the index

```
myarray[????] = 0;
```

- Replace it with a variable

```
myarray[i] = 0;
```

- And use a for loop to control the variable

Initializing Arrays: Loops

- A for loop allows us to process in bulk the whole array in one go

```
int [] myarray = new int[500];
for (int i=0; i<500; i++)
{
   myarray[i]=0;
}
```

- Here I filled an array of size 500 elements using a couple of lines of code!

A method to Initialize an Array

- This seems a useful thing to package in to a method
- Given the size of an array return an array full of 0s of that size
- Notice the return type of the method here is an array type

```
public static int[] initArray(int size)
{
      int[] newarray = new int[size];
      for (int i=0; i<size; i++)
      {
         newarray[i]=0;
      }
     return newarray;
}
```

As this creates the array too we can just declare an array variable then use this rather than new.

```
int ages[] = initArray(100)
```

Initialising arrays: shorthand

- If you know all the values you can fill an array using a shorthand notation:

```
int [] myarray = {0,0,0,0};
```

- Its still long-winded for big arrays though

- Apart from rare occasions for loops are better.

Initialising with other values

- Suppose you want to store the indices:

```
int [] myarray = new int[4];
for (int i=0; i<4; i++)
{
  myarray[i]=i;
}
```

- That stores 0 in position 0, 1 in position 1 etc

Exercises

- Write a program that initialises each entry of an array by double its index so it stores the 2 times table (position 2 holds 4, position 3 holds 6, etc
- Write a code fragment that fills the entries in a String array of size 10 by asking the person to type in 10 names.

Length of an Array

- The length of an array is fixed when it is created.

```
myarray.length
```

- tells you it.
- Notice the length is always one larger than the last subscript position.
- An array of length 10 has indexes from 0 to 9.
- Attempting to access position 10 of an array of size 10 would be a runtime error - there is no position 10!

Printing Arrays

- To print all entries in an array called wombat

```
int [] wombat = new int[10];
<code that fills the array>

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

Exercise

- Write a method that prints all the entries of a given array.

Printing some of an array

- To print every second element we just control the loop counter appropriately:

```
int [] wombat = new int[10];
<code that fills the array>

for (int i=0; i< wombat.length; i=i+2)
{
  System.out.println(wombat[i]);
}
```

Arrays in main

- In our standard header:

```
public static void main (String [] param)
```

- param[] is an array of strings,
- the parameters to the program. param[0] is the first parameter. param[1] is the next word, etc

Main parameters

- So in a call to java to run the program copy:

```
java copy infile copyfile
```

- The string "infile" is accessible in the program as param[0],and
- "copyfile" is accessible as param[1]

Array Uses

- Arrays are used to handle bulk data.
- Typical things you do with arrays are:
  - apply some operation to a single element
  - apply some operation to each element of the array
  - find some particular element
  - sort the elements into some order

Searching Arrays:

To "search" an array just means find the position of some particular value

- Search problems exist in all kinds of structures.
- For arrays there are two particular cases of interest.
  - Searching ordered arrays
  - Searching unordered arrays

Searching unordered arrays

- Simply look through the array element by element to find the goal (known as the search key).
- This is an algorithm known as **Linear Search**

Search: unordered arrays

```
for (int i=0; i<a.length; i++)
{
   if (a[i]==target)
   {
     result = i; // Store position found
   }
}
```

- This program will still keep going once its found it!

Search: unordered arrays

- We can make it stop by combining a for loop with a break statement
- Break just jumps you out of a loop straight away.

```
for (int i=0; i<a.length; i++)
{
   if (a[i]==target)
   {
     result = i;
     break;
   }
}
```

- This will stop once found

Search method: unordered arrays

- Searching is of course a natural thing to make a method.
- Given an array and a key to search for return the position it is found in, otherwise return -1 (to mean NOT FOUND as it cant be a valid position)

```
public static int search(int [] a, int key)
{
           for (int i=0; i<a.length; i++)
           {
              if (a[i]==key)
              {
                return i;
              }
           }
           return -1;
}
```

- The first return inside the loop acts like break, jumping out of the method and so out of the loop early
- Notice the return -1 is OUTSIDE the loop
- You only know the key is not there if you get to the **end** of the array

Exercise

- Modify the search method above so it searches String arrays.

Searching ordered arrays

- An ordered array is one that has been sorted
  - the elements are sorted by some property
  - eg into numerical order or alphabetical order
- You could still just use linear search but…
- There are cleverer ways to search ordered arrays.
- Try and work out a faster algorithm yourself
- We will return to this later when we cover "divide and conquer" algorithms
- Hint: do you check a telephone directory one by one searching for a name?
- Can you code a version similar to the way you do search a telephone directory?

Lookup Tables

- One simple but powerful use of an array is as a Lookup Table
- This is a data structure that holds precomputed values that can then be quickly found by just going straight to the appropriate entry in the table and getting the answer
- Lookup tables trade off spending time once to compute all possible answers before you start with the space needed to store all those values.
- You can create a search algorithm based on lookup tables that is different to linear search
- Have a lookup table whose indices are the keys you could search for
- It is a second array in addition to the array you intend to use it to search.
- Store in the table at the table the place in the array being searched where that key can be found.

Unit 5 Example Programs and Programming Exercises

Here is a typical program using arrays.

```
/* ***************************************
AUTHOR Paul Curzon
This program demonstrates
 - arrays.
 - lookup tables
  Fill an array using a for loop as the result of a calculation
  Lookup tables are data structures used to hold precomputed values
  that can quickly be obtained
****************************************** */

import java.util.*;

class lookupcube
{
    public static void main (String[] param)
    {
        int[] lookupTable; //create an unallocated array variable

        lookupTable = createLookup();
        useLookup(lookupTable);
        System.exit(0);

    } // END main

      // first fill an array with the answer
      // to a calculation for each value (cubing)
      //
    public static int[] createLookup()
    {
       int[] lookup = new int[10];

       for(int i = 0; i<lookup.length; i++)
       {
           lookup[i] = i*i*i;
       }

       return lookup;      // return the whole array
    } // END createLookup

       //get an answer quickly without calculating just looking it up
       //
    public static void useLookup(int [] lookup)
    {
       int query;

       query = inputInt("Give me a number to cube quickly");

       System.out.println(query + " cubed is " + lookup[query]);
    } // END createLookup

     // Input an integer
    public static int inputInt(String message)
    {
       return Integer.parseInt(input(message));
    } // END inputInt
```

```
    // Input strings
    public static String input(String message)
    {
       Scanner scanner = new Scanner(System.in);

       System.out.println(message);
       ans = scanner.nextLine();

       return scanner.nextLine();
    } // END input

} // END class lookupcube
```

Only try these exercises once you feel confident with those from the previous unit.

Array Example Programs

Compile and run the following programs from the ECS401 website – unit 5 area, experiment with them modifying them to do slightly different things.

- **Set up an array and print it** Modify this program so that it prints each element twice before moving on to the next. Try modifying it so that it prints the contents in reverse order (Hint you need a loop that counts down not up)
- **Store the same number in all positions of an array then print it out** Modify it so that it stores a number input by the user.
- **Create an array that stores cubes of numbers so they can be looked up when asked rather than calculated** Modify it so that it stores the result of a different calculation (eg the square root of a number. Then modify it so you can ask for values over and over, and so it doesn't crash if you ask for a too big value that goes off the end of the array.
- **Search an array** What happens if you search for "cat"? What happens if you search for "hat" - explain the answer. Modify it so it does something sensible if the thing searched for isn't there. Modify it so searches integer arrays

Fill and Print an array

Write a program that reads in 10 integers from the keyboard storing them in an array. Once they are all read in, print them to the screen. HINT This will need two for loops one after the other - the first reads them in, the second reads them out.

Reverse

Read in 10 integers from the keyboard storing them in an array. Then print out the same integers in reverse order of input.

More than 5

Write a fragment of code that asks for 12 numbers in to an array prints them all out and having done that for all then prints out each number only if that number is greater than 5.

PrettyPrint Part 1

Write a program that prints an array in a table with the array positions above the contents of the array.

```
        0    1    2    3    4    5    6    7    8    9
    |   1|   8|  10|   7|   4| 112|  43| 144|  18|  11|
```

HINT: This will need two for loops one after the other, the first printing the indexes, the second printing the array contents.

Prettyprint Part 2

Write a program that prints an array in a nicer "graphical" representation:

```
     0    1    2    3    4    5    6    7    8    9
-----------------------------------------------------
|   1|   8|  10|   7|   4| 112|  43| 144|  18|  11|
-----------------------------------------------------
```

Use your code to print out arrays when debugging other exercises.

Random array

Write a program that fills an array A of size 5 with random integers in the range 1 to 100 and prints it out (for testing purposes). Write programs that then after creating such a random array, do each of the following:

(a) Calculate the sum of the elements of A and print it out.

(b) Count the number of elements in A whose contents are less than 10.

(c) Create a new array B of size 5 such that for all entries B[i] (where i is between 0 and 19 ie 0<=i<=19), B[i] = A[0] + A[1] + ... + A[i]

Modify your code to have arrays of size 20 (so that you are filling an array A of size 20 with random integers...). If you cannot do this with a single edit, modify your program so that you can change the array size with a single edit.

Searching an array for a given number

Write a program that fills an array with 20 random integers. Print out the array. Then search the array to discover whether or not it contains an integer input from the keyboard.

Searching an array for a given string

Modify your program above to search for strings.

Once you have done these, try the next assessed exercise. The assessed exercises must be done on your own. If you have problems with them then go back to the non-assessed exercises.

Additional Exercises

Copy and double

Read integers from the keyboard until zero is read, storing them in input order in an array A. Then copy them to another array B doubling each integer (i.e. after the copying of n integers and assuming you are using item 0 of the arrays, then for all i such that 0 <= i < n, B[i] = A[i]*2). Then print B.

Find the maximum element of an array

Write a program that fills an array with 20 random integers. Print out the array. Then scan the array from one end to the other to find the largest element. Print out both the maximum element of the array and the index (position) of this element. HINT Keep a record of the largest value found so far and compare it with subsequent values.

Testing whether the contents of one array are a subset of the contents of another:

Fill two arrays A and B with 20 random integers in the range 1 to 10. Test whether or not all the numbers occurring in B also occur in A. HINT One way to do this is repeatedly search for the elements from one array in the other- do linear search lots of times.

Dice throwing

Write a program that simulates dice throwing- throwing two dice and recording the frequency of occurrence of each outcome. Use an array to store the frequency counts for each total.

Frequency table

Read an integer n from the keyboard. Generate 100,000 random integers in the range 1 to n. Use an array A to keep count of the frequency of occurrence of each integer (ie use A[1] to keep track of the number of occurrences of the integer 1, A[2] for the integer 2 and so on). Print the number of times each integer from 1 to n is generated.

Question: what happens if the number input from the keyboard is larger than the size of the array A?

The Sieve of Eratosthenes

This is a nice exercise that involves writing a program to generate all the prime numbers less than a given number n. (Remember that a prime number is a number whose only factors are 1 and itself or, put another way, is not exactly divisible by any numbers other than 1 and itself. So the first few primes are 2,3,5,7,11,13,17,19..)
Only attempt the program if you understand Eratosthenes' sieve method as follows:

1) Start with the first number not crossed off (i.e. 2). Call this k.

2) Cross off (or 'sieve out') all the multiples of k that are less than or equal to n (since these are obviously not prime).

3) Repeat steps 2 and 3 with the next number k that is not crossed off. Terminate when you have reached n.

The prime numbers are all the numbers not sieved out.

Using paper and pencil run Eratosthenes sieve by hand on the numbers 2..30.
To write a program to carry out this method, use a boolean array of size N with all elements initially set to false.
Crossing off a number n involves setting the nth array element to true.
Make your program print out the number of primes less than N as well as the primes themselves.

How many prime numbers are there less than 1 million?

Can you make your program more 'efficient' by stopping one of your loops at the square root of N, rather than N itself? Explain why.

Spell checker

Write a program that reads a list of words (the dictionary) and stores them in an array (note that this will be an array of Strings). The program then reads more words and for each word tests whether or not it appears in the dictionary. Print out each word from the input file that does not occur in the dictionary. (As an extra challenge read up from a text book or read ahead to unit 11 about files and read the words in from a file rather than having to type them in)

Implementing a search algorithm – a common mistake

The following code that is supposed to print out whether the given search key is in the array or not. It should print "Found" if it is there and "Not in array" if it is not in the array at all.

```
int [] a = {1,2,3,4,5};                          //L1
int searchkey = 3;                               //L2

for (int i = 0; i < 5; i++)                      //L3
{
    if (searchkey == a [i])                      //L4
    {
        System.out.println("Found");             //L5
        break;
    }
    else
        System.out.println("Not in array");      //L6
}
```

a) Dry run the program fragment and explain what it does wrong. Give a corrected version of the code.

Answer and discussion

a)

Line L1: Initially we fill the array with values {1,2,3,4,5}

Line L2: The searchkey is set to 3.

searchkey	a[0]	a[1]	a[2]	a[3]	a[4]
7	1	2	3	4	5

Line L3: We then enter the for loop. Variable i is set to 0 which is less than 5 so we enter the body.

Line L4: The searchkey (value 3) is not the same as the value in a[0] which is 1 so we jump to the else case (line L6).

searchkey	a[0]	a[1]	a[2]	a[3]	a[4]	i	searckkey == a[i]?
7	1	2	3	4	5	0	false (3== 1?)

Line L6 says print "not in the array" so that is printed on the screen:

THE SCREEN

Not in array

Line L3: We go back round the loop to line L3 again. i is increased by 1 which is still less than 5 so we go back into the body.

Line L4: The searchkey (value 3) is not the same as the value in a[1] which is 2 so we jump to the else case (line L6).

searchkey	a[0]	a[1]	a[2]	a[3]	a[4]	i	searckkey == a[i]?

7	1	2	3	4	5	1	false (3 == 2?)

Line L6 says print “not in the array” so that is printed on the screen:

THE SCREEN

Not in array Not in array

Line L3: We go back round the loop to line L3 again. i is increased by 1 which is still less than 5 so we go back into the body.
Line L4: The searchkey (value 3) is the same as the value in a[2] which is 3 so we jump to the then case (line L5).

searchkey	a[0]	a[1]	a[2]	a[3]	a[4]	i	searckkey == a[i]?
7	1	2	3	4	5	1	true (3 == 3?)

Line L5 says print “Found” so that is printed on the screen and then the break jumps us out of the loop.

THE SCREEN

Not in array Not in array Found

Rather than just printing either “Found” or “Not in array” as intended, we have repeatedly printed “Not in array” when we shouldn’t have.
b) The following is a corrected version of the code:

```
int [] a = {1,2,3,4,5};
int searchkey = 7;
Boolean flag = false;
for (int i = 0; i < 5; i++)
{
    if (searchkey == a [i])
    {
        System.out.println(“Found”); flag = true;
        break;
    }
}
if (flag == false) System.out.println(“Not in array”);
```

What have we changed to fix the problem? The if statement in the loop no longer has an else part. Instead the print statement to say that the value we are looking for is not in the array is placed ***outside*** the loop. It gets executed only once ***after*** the loop has finished. If you think about it that has to be where it is. You cannot make a statement that a value is nowhere in the array until you have finished checking the whole array. If we have checked an individual entry and it is not in the place looked we do not want to do anything at all – just move on.
We also only want to print “Not in array” if it wasn’t found so we set a flag variable to true to indicate this and test it’s value before printing not found.

Avoiding Bugs

Its always better to avoid a bug in the first place, rather than cure it afterwards. That is where programming "style" comes in...Be a stylish programmer and make your programs easier to follow.

TIP 1 Always check your loops are processing the array between position 0 and one less than the length - otherwise you will run off the end. Virtually all array based errors are to do with running off the end of the array (or not getting all the way to the end).

TIP 2 Use print statements in the loop to print out the contents of an array and messages to tell you how far through an array it gets before it goes wrong.

TIP 3 Use the length variable (eg a.length for array a) rather than fixed numbers so that if you change the array size the program still works.

Some common compile time errors

Use square brackets

```
pc$ javac printarray.java
printarray.java:43: cannot find symbol
symbol  : method fourTimes(int)
location: class printarray
           System.out.println(i + ":" + fourTimes(i));
                                        ^
1 error
```

Oops should have used square brackets - its an array not a method call

```
System.out.println(i + ":" + fourTimes[i]);
```

Remember to give the array a size

```
 pc$ javac fivearray.java
fivearray.java:38: array dimension missing
       int[] fives = new int[];
                             ^
1 error
```

I forgot to tell it what size array I wanted

```
int[] fives = new int[12];
```

Length of an array

```
javac fivearray.java
fivearray.java:42: cannot find symbol
symbol  : method length()
location: class int[]
```

the length of an array is a variable not a method call - so I need fives.length not fives.length()

Common Run-time errors

Array indices start from 0

```
java printarray
0:4
1:8
2:12
3:16
4:20
5:24
```

Oops thats not the 4 times table - I forgot the array indexes count from 0 so the first element value is 0 not 4 if its supposed to be go tp position i to get 4*i

Make the loop count up to the end

```
java printarray
0:0
1:4
2:8
3:12
4:16
5:20
6:24
7:28
8:32
9:36
```

...and now I forgot having changed it to the 10 times table that I have to count till less than 11 not just put 10 as the size - the array has 11 entries.

Check the user cant enter a value that runs off the end of an array

```
java lookupcube
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException:
10
        at lookupcube.createLookup(lookupcube.java:54)
        at lookupcube.main(lookupcube.java:17)
```

The code doesn't stop the user typing in a query to search beyond the end of the array, so doing so crashes the program. It really needs an if statement checking that the value given is ok.

Unit 5 Exam Style Question: Arrays

a. [6 marks]
The following code fragments contain bugs. **Identify and correct** the bugs

i)
```
/* increment position 5 of array a */
a(5) = a(5) + 1;
```

ii)
```
/* print all elements of an array of size 10 */
for(int i=0; i <= 10; i++)
{
  System.out.println(a[i]);
}
```

iii)
```
/* declare an integer array with 4 elements*/
int [] a = new int[];
```

b. [9 marks]
Trace the execution of the following fragment of code

```
int [] a = new int[6];
for(int i=0; i <= 4; i++)
{
  a[i] = i;
}

for(int j=0; j <= 4; j++)
{
  a[j+1] = a[j+1]+a[j];
}
```

c. [10 marks]
Write a Java program that reads in 20 integers from the keyboard. It should then search the array printing the position in the array of *all* places where the integer was found.

Explaining concepts

Here are some more concepts to check you can write good explanations about. The list of concepts here is not exhaustive of course!

Explain what is meant by the following concepts, using Java programming examples to illustrate your answer:

a) an array
b) an array index
c) an array declaration
d) search algorithms
e) linear search

Unit 6 : While Loops

Lecture Slides/Notes

Learning Outcomes

Once you have done the reading and the exercises you should be able to:

- write programs that follow instructions repeatedly where the number of times is not known at the outset
- explain what is meant by for loops and while loops, their similarities and differences
- trace the execution of programs containing while loops

From last week …

Before moving on to this section you should be able to:

- write programs that process bulk data using arrays
- write programs that search arrays for information
- explain how arrays are declared and used

Code from last week

```
for (int i=0; i<a.length; i++)
{
   if (a[i]==target)
   {
     result = i;
     break;
   }
}
```

Repetition in general

Loops generally have two parts:

- a test to tell you whether to do it again
- a thing to be done repeatedly

When writing loops always work out these parts first before writing actual code.

- What is the test and the thing to be repeated for each of the dice games?
- There is also always something that is changed by the action of the loop.
    - Eg the running score

Dice games

Here are some games with different loop properties

- Roll a dice 5 times. Get as high a score as possible
- Roll a dice until you roll a 6 - get as high a score as possible
- Roll a dice until you decide to stop, with a question asking if you want to continue before each roll. Roll a 6 and your score is reset to zero. Get as high a score as possible.
- Roll a dice until you decide to stop, Roll a 6 and you get zero and the game also ends. Get as high a score as possible.
- Roll a dice repeatedly adding the scores. Keep going till you get a score that is more than 21. Do it in as few rolls as possible.

How do you do that?

Which of the above could you program with a pure for loop?

For some problems a for loop is not general enough.

Each of the above needs a slightly different kind of loop structure.

Exactly the same loop structures appear in lots of other programs.
Can we come up with a new kind of loop that covers ALL possibilities?

Unbounded loops

You don't always know before the loop starts how many times to repeat

- Need to know that with a for loop…
- it depends on what happens like what scores you roll on the dice
- A pure for loop is then little help.

Let us invent a new *general* loop construct...

- We need to tell the computer what is to be repeated,
- We also need a way of checking: do we stop yet?
- We also need a keyword to tell us this is our new kind of loop

A general form of loop

We want to say something like:

While a test says to go on
do these instructions over and over

```
while(<a test on continuing is true>)
{
      <do these instructions over and over>
}
```

What does it mean…

```
while (<condition>)
    <body>
```

Does the following…

Check <condition>;
if true do <body>,
go back, check <condition>;
if true do <body>,
go back, check <condition>;
if true do <body>,. . .
But if <condition> is false then stop and jump to whatever is immediately after the body

Non-starting loops

If <condition> is false initially then the <body> is never executed.

- It just jumps straight to beyond the loop

Non-terminating loops

While loops can go on for ever:

```
while (true)
{}
```

will never stop on its own

- Stop it by typing Ctrl-C (Control-C).
- Note a property of an *algorithm* is that if followed it should terminate.
- Otherwise its not strictly a correct algorithm

Do it again?

Of course you can wrap a while loop round the code of lots of the previous exercises, so that you can do that previous thing a number of times.

Common patterns for while loops

There are a whole series of patterns of while loops (eg corresponding to the different kind of dice games mentioned)
We will preview some common ones.
Match the problem you have to the kind of loop pattern you need

Just ask

- The first pattern is to have the program to just ask if the person wants to do it again
- You do it the first time, then ask at the end of the loop if they want to do it again

```
String ans="y";
while (ans.equals("y"))
{
  // <real work to be repeated goes here>

  ans = input("Again? (y/n)");
}
```

- There are lots of variations, eg allowing several different ways of saying continue.
- Then the test just has to OR the alternatives

```
String ans="y";
while ((ans.equals("y") || (ans.equals("Y"))
{
  // <real work to be repeated goes here>

  ans = input("Again? (y/n)");
}
```

Check for a sentinel

- If you are inputing a whole series of values you can look for a special END or "sentinal" value amongst them, the person can type in to say they want to end.
- Its a good idea to store the sentinel as a final variable as it doesn't change

```
final String END = "QUIT";
String ans = input(First value"); // set to first value

while ( ! (ans.equals(END) ) // QUIT is NOT typed
{
  // <work to process value input goes here>

  ans = input("Next value");
}
```

- A variation is the values might be random ones generated from eg a dice roll method rather than the user inputing them

While loops and accumulators

- Often use an accumulator variable with while loops.
- It is just a variable that builds up the answer bit by bit each time round the loop.

```
int count=0;
String ans = "y";

while (ans.equals("y"))
{
  count = count+1;
```

```
  ans = input("Shall I go on?(y/n)");
}
System.out.println("That was " + count + times");
```

An accumulator stopping on a sentinel value

The following is a pattern for a loop with a slightly different kind of accumulator. You will find yourself using loops like this all the time to build up answers as new data is considered.

```
final int END = 0;
int sum = 0; //Must remember to initialise accumulators!
next = <first value eg from keyboard>
while (next != END)
{
  sum = sum+next;
  next = <next value eg from keyboard>
}
```

An accumulator stopping when the accumulator gets to some value

The following is a pattern for a loop where the value of the accumulator determines when you stop.

```
final int END_TOTAL = <some value>;
int sum = 0; //Must remember to initialise accumulators!

while (sum < END)
{
  next = <next value eg from keyboard>
  sum = sum+next;
}
```

Exercise

Write a program fragment using a while loop to do the dice game from earlier

- Roll a dice until you roll a 6 - get as high a score as possible.
- Assume you already have written a method that rolls a dice giving a number between 1 and 6
- `roll = diceroll();`

While loops and for loops

Any pure for loop can be programmed up using a while loop:

```
for (int i=<start>; i <= <finish>; i++)
{
   <body>
}
```

does the same thing as

```
int i=<start>;
while (i <= <finish>)
{
  <body>
  i++;
}
```

While loops and for loops

- The reverse is not true.
- While loops cannot be programmed up as pure for loops.

Evidence:

- It's easy to write non-terminating programs using while loops.
- You can't do this if you only use standard for loops.

While loops and for loops

- In fact the C/Java for loop is more general. It is equivalent to a while loop.

While loops and for loops

- A general Java for loop:

```
for (<init>;<test>;<reinit>) <body>
```

means:

do <init>,
evaluate <test>
if <test> is true
do <body>
do <reinit>
return to stage 2
otherwise end the loop and carry on with the rest of the program.

While loops and for loops

So in java

```
for (<init>;<test>;<reinit>) <body>
```

is equivalent to

```
<init>
while (<test>)
{
  <body>
  <reinit>
}
```

A general while loop:

```
while (<test>) <body>
```

is equivalent to

```
for ( ; <test>; ) <body>
```

So in Java (and C/C++) anything you can write with while you can write with for and vice versa – but only because the Java for loop is not a pure one. We have not given the parts that control the counter!

Search: unordered arrays

- Another way to code linear search is with a while loop

```
boolean found = false;
int i=0; int result = -1;
while (!found & i<a.length)
{
   if (a[i]==target)
   {
     found=true;
     i=i+1;
   }
}
result = i; // result is the position found
```

- This uses a Boolean flag variable as part of the test to say when to stop
- It is set to false before the loop start
- and set to true in the loop body, but only if the termination thing happens
- That is a common pattern that is used often with while loops, not just search

Structured programming

- We have been looking at the “structured” approach to programming.
- Be clear about what the body of a loop is supposed to do.

- Develop it as an entity in its own right.
- Program is a sequence of (named) subtasks.
- Develop the subtasks by refining them towards code.
- As you do this put in if statements and loops. Branches of if 's and bodies of loops are subtasks.

The basic control constructs: there are just 3 needed

Sequence: do this, then do that.

In Java this is done just by writing one command after another.

Branching: choose between different things

In Java this is

```
if (<test>) <case> else <case>
```

(the switch construct is another way to do something similar though less general).

Looping: keep repeating something until some exit condition is met.

There's a difference between "for loops" (do something a fixed number of times) and "while loops" but the latter are all you need.

In Java:

```
for (int i=<start>; i<=<finish>; i++)
            <body>
while (<test>) <body>
for (<init>;<test>;<reinit>) <body>
```

THAT'S IT.

THERE ARE NO MORE BASIC CONTROL CONSTRUCTS NEEDED!!!!

A language that has these control constructs is "*Turing complete*".

You can write any computable operation in it.

Why have anything else?

Everything else (like Methods, Object-oriented programming, other kinds of loops, etc) is there for other reasons

- to make the language clearer
- to allow you to break up code better
- to make code easier to reuse
- to make programs easier to develop
- to make programs more secure in their running . . .

Debugging Tips

Debugging: Step 1 INDENT YOUR CODE

Execution follows the grammatical structure of code.

Indentation

- makes that structure visible.
- lets you see where choice points are.
- lets you break the program up into phases.
- lets you see what bits you can comment out.
- often makes it obvious if you've put something simple in the wrong place
- It shows eg whether code is inside or outside a loop

Proper indentation

The two crucial points:

- indent blocks a fixed amount
- make ends line up with beginnings

Debugging: Step 2 LOCALISE THE PROBLEM

- Put in print statements to tell you how you are going through the code.
    - System.out.println("We got here");
- Print out values of variables,
    - so that you can compare them with what you expect.
- Use these to find out where in the program the problem is.

Debugging: Step 3 FIX THE PROBLEM

- Once you have the problem localised, put in more print statements systematically so that you can trace the code.
- Use your knowledge of what the program is actually doing versus what it should be doing to fix it.

Unit 6 Example Programs and Programming Exercises

Here is a typical program using a while loop

```
/* ***************************************
AUTHOR Paul Curzon
  A program to print the averages of a series of
  pairs of ages, stopping when the user wishes it.
****************************************** */
import java.util.*;
class ages2
{
   // Ask for a series pairs of ages printing their average each time
    public static void main (String[] param)
    {
      String again = "yes";

       while (again.equals("yes"))
       {
          int age1 = inputInt("Give me the first age");
          int age2  = inputInt("Give me the second age");

          int averageAge = calculateAverage(age1, age2);
          System.out.println("Average: " + averageAge);

          again = input("Continue? (yes/no)");
        } ?? END while loop

        System.exit(0);
    } // END main

//   A method that given two numbersreturns their average
    public static int calculateAverage(int value1, int value2)
     {
        int average = (value1 + value2) / 2;
        return average;
     } // END calculateAverage

//    Ask for a string with given message
//    Return the string typed in by the user
    public static String input(String message)
    {
       Scanner scanner = new Scanner(System.in);

       System.out.println(message);
       String answer = scanner.nextLine();

       return answer;
    } // END input

//  Return the integer typed in by the user
   public static int inputInt(String message)
    {
       int answer = Integer.parseInt(input(message));

       return answer;
    } // END input
} // END class ages2
```

Only try these exercises once you feel confident with those from the previous week/ unit.

While statement Example Programs

Compile and run the following programs from the ECS401 website – unit 6 area, experiment with them modifying them to do slightly different things. What controls how many times the program goes round the loop each time?

- A simple loop that does nothing more than ask if you want to continue and stops when you say no Modify it so that it prints an extra message of your own each time by adding another print statement to the loop
- Keeps a count of how many times round the loop, asking to go on
- Rather than explicitly asking the user each time if they want to stop watch out for a special value that is a non-legal input (not one you might want to add)
- A loop that never stops, whatever you do. Make sure you know how to stop a non-terminating program before you run this one. Try editing back in the line commented out towards the end - explain what happens when you compile it.

Read and Output

Repetitively read and output a series of numbers (non-zero integers) typed on the keyboard. Stop when the user types a zero.

HINT Get code working that reads a single number from the keyboard and prints it out first. Once that works, think about what test tells you that you must continue and put a while loop containing it round the code. Make sure you have set initial values of variables so the test is true to start with (otherwise it will never go into the loop).

Repeatedly adding

Repetitively read non-zero integers from the keyboard until the user types a zero. Then output the sum of the integers typed (excluding the zero). HINT You will need an accumulator in the loop.

Average

Repetitively read non-zero integers from the keyboard until the user types a zero. Then output the sum and average of the integers typed (excluding the zero).

Subtracting

Repetitively read non-zero integers from the keyboard until the user types a zero. As you read each integer subtract it from a running score that started at 501.

"Stop"

Repetitively read and output the lines typed on the keyboard. Stop when the user types a line consisting just of the string "stop". Do not output this line.

Once you have done these, try the next assessed exercise. The assessed exercises must be done on your own. If you have problems with them then go back to the non-assessed exercises.

Additional Exercises

The following are additional, harder exercises. The more programs you write this week the easier you will find next week's work and the assessments and the quicker you will learn to program. If you complete these exercises easily, you may wish to read ahead and start thinking about next week's exercises.

Rectangle

Write a program which inputs two positive integers m and n from the keyboard and outputs an m by n rectangle giving coordinates:

(1,1) (1,2) (1,3) (1,4)
(2,1) (2,2) (2,3) (2,4)
(3,1) (3,2) (3,3) (3,4)

Do not bother too much with the alignment, your code need only give a neat square for numbers up to 10.

Addition Square

Write a program which outputs an 'addition square' to 10; i.e. shows the sum of all the pairs of numbers from 1 to 10 as follows:

```
     1  2  3  4  5  6  7  8  9 10
    ----------------------------
1 |  2  3  4  5  6  7  8  9 10 11
2 |  3  4  5  6  7  8  9 10 11 12
   ...
```

(Alignment is a problem here.)

Explaining concepts

Here are some more concepts to check you can write good explanations about. The list of concepts here is not exhaustive of course!

Explain what is meant by the following concepts, using Java programming examples to illustrate your answer:

a) a control structure
b) sequencing
c) branching
d) if statements
e) loops
f) for loops
g) while loops
h) non-terminating loops
i) structured programming
j) the body of a loop
k) the condition of a loop

Unit 6 Exam Style Question

This question concerns programs acting repeatedly.

a. [6 marks]

Write a program fragment that uses a while loop rather than a for loop that does exactly the same as the following for loop in all circumstances. **Justify** the equivalence of the two fragments.

```
for(int i = 1; i < 4; i++)
{
   System.out.println(i);
}
```

b. [9 marks]

Trace the following program fragment showing the values of all variables as it executes.

```
int t = 50;
int c = 5;

while(c <= 10)
{
  t = t-c;
  c = c+2;
}

t = t+4;
```

c. [10 marks]

Write a Java program that uses a while loop to simulate a dice game. The person throws a real dice and types in the number they throw, with the program keeping score. The person aims to gain the highest score they can by repeatedly throwing the dice. If they roll a six they get score 0 and the game ends, but if they decide to stop before rolling a six their score is the sum of all their throws.

An example game might be:

```
Would you like to throw? y
What number did you throw? 4

Your total is 4.

Would you like to throw? y
What number did you throw? 2

Your total is 6.

Would you like to throw? n

You scored 6.
```

Unit 7 : Defining Methods, Abstract Data Types

This unit reviews and extends what we've seen about methods and also introduces the idea of an abstract data type.

Learning Outcomes

Once you have done the reading and the exercises you should be able to:

- write programs that are split into methods
- write programs that pass information to methods using parameters
- explain what is meant by methods, procedures, functions and parameters
- explain what is meant by an abstract data type
- explain how abstract data types can be implemented from existing types
- write programs that create and use abstract data types
- discuss the importance of abstract data types

From last week …

Before moving on to this section you should be able to:

- write programs that follow instructions repeatedly where the number of times is not known at the outset
- explain what is meant by for loops and while loops, their similarities and differences
- trace the execution of programs containing while loops

A program from last week:

You should be able to write code like the following

```
int count=0; String ans = "y";

while (ans.equals("y"))
{
  count = count+1;
  ans = input("Shall I go on? (y/n)");
}
System.out.println("That was " + count +  " times");
```

Big programs, big problems

- Writing small simple programs is easy!
- The real problem is big programs.
- If you are trying to write a big program:
    - how can you get it to work in the first place?
    - how can you write it so that anybody can understand it?
    - how can you write it so that somebody else can change it?
- (NB Software lasts **much** longer than hardware)

Chunking it

- The answer to all these is to break your program up into small chunks that you keep as separate as possible.
- There are several techniques and language structures that help.
- One modern approach is to use objects. We're going to be telling you about that in detail in a follow-on module.
- An essential component of the object-based approach is the notion of method.
- It is also a core of procedural programming

Methods

- The name "method" is object-oriented jargon.

- A **method** is a self-contained block of code that depends on parameters that are passed to it when the method is invoked ("***called***").
- Methods can be used to compute values (eg do calculations) in which case they're often called "functions".
- Or they can be used just to do things, without returning anything, in which case they're often called "procedures".
- You **used** both types of method from the start - ones written by others:
    - scanner.nextLine() is a function (its point is to return a value)
    - System.out.println() is a procedure (it doesnt return anything)

Methods: Write your own

- So you have been using methods written by others from the start

The next step up that is the basis of writing big programs and staying in control is to:

- write your own (easy bit)
    - the simplest do the same thing every time
    - you need to be able to pass information to them so they can do different things each time they are called.
- learn **when** to write your own (skill and judgement bit)

Program as a recipe book

A program is made up of a series of methods just like a recipe book is made of a series of recipes

```
class foo // recipe book
{
   method1 // recipe 1
   method2 // recipe 2
   method3  …
}
```

A little bit of grammar

- A phrase that computes a value is called an **expression**.
- Examples:
    - x+y
    - scanner.nextLine()
    - 0 <= x & x <= 10
- The whole point of them is to give some other part of the program a value
    - ans = x + y; // the right hand side of an assignment is an expression
    - return (scanner.nextLine()); the part saying what is returned is an expression
    - if (0 <= x & x <= 10) … // the test is an expression
- A **function** is a *parameterized* expression.
- That just means is an expression that is given information to work on as arguments.
- It processes the information given to it when it is called get a result

A little bit of grammar, continued

- A phrase that just does something is called a **statement** or **command**.
- Examples:

```
x = x+1; //The whole assignment is a statement
System.out.println(…);
if (0 <= x & x <= 10) {y = y+1;}

else {y = y-1;} //The whole if is a statement
```

```
while (x>0) do { ... } //The whole while is a statement
```

- A **procedure** is a *parameterized* statement.
- It is a statement that is missing some information that you give to it when you want it to execute – different each time

A little bit of grammar, continued

- If you are going to parameterize expressions and statements it makes sense to do both in the same way…Keep things uniform.
- The way this is done in programming languages is based on a theoretical study of functions by mathematicians, called the λ-calculus.
- A lot of the work on applying those ideas to programming languages was done by Peter Landin, who worked here for many years.
- Java is still not as flexible in the way it lets you do things as most λ -calculi (or as many other languages).

A Simple function

```
public static int f_to_c (int f)
{
  return ((f - 32) * 5) / 9;
}
```

A Simple function

There are two bits to this:

Header (also sometimes called the *signature* of the method)

```
public static int f_to_c (int f)
```

Body

```
{
  int c = ((f - 32) * 5) / 9;
  return c;
}
```

Header

The header tells you:

- the name of the function: **f_to_c**
- the arguments it takes – essentially a series of variable declarations here: **int f**
- what type of result the method can produce: **int**
- some other OOP linked information you don't need to worry about now for now: **static**

Body

- The body is the actual code to execute whenever its called
- The body produces the result by doing a calculation on the information given to it as put in the variable that was declared in the header line.
- The information is put there when the method is called
- It must include a return statement. This says that the value the expression evaluates to that follows the word **return** is the result of the function.

Calling methods

- Having defined a method as we just did for f_to_c you can now call it just like the in-built methods you have used from the start
- In declaring the above we told the compiler when you defined it:
  - it should always be given an integer argument
  - And return an integer result

- Thats part of the point. We are giving the compiler information so it can check we do what we intended - that the call and declaration match
- So you must use ("call") it like that
  - `temperature = f_to_c(200);`
  - This is called as an expression as its on the right side of an assignment so it must be a function
  - `System.out.println( f_to_c(a));`
  - This is called as a statement so it must be a procedure
- temperature and a must be declared as integers

Return types

- In Java the distinction between a procedure and function is indicated in the return type. A return type of void means "nothing is returned" - its a procedure!
- Functions have some other return type

Exercise

- Write a program that squares numbers by calling a method called square.

Location of methods

- You put method declarations after the class header:
  - `class conversion  {`
- and before the header for main:
  - `public static int main (String param[]) {`
- Or after the closing bracket of main
- You cannot put them inside main or inside other methods
- You cannot "nest" method declarations.
  - some other languages do let you localise methods
- You can have as many methods as you need in a program

Procedures

So there are just two differences between a procedure and a function.

- the return type of a procedure must be void:

```
static void my_procedure(...)
 { … }
```

- the procedure does not have an expression as part of its return statement
  - (or possibly no return statement)
- A simple procedure

```
static void myprintinbox (String s)
{
  JOptionPane.showMessageDialog(null,s);
}
```

- With this method I could use a call

```
myprintinbox(s);
```

- Instead of

```
JOptionPane.showMessageDialog(null,s);
```

- Ive defined it to do the same thing (now I don't need to forget to include the null argument!)
  - The new method's body now sorts that out for me

A more complicated Procedure

A general procedure for printing arrays:

```
public static void print_array (int[] myarray)
{
```

```
    int length = myarray.length;
    for (int i=0; i<length; i++)
    {
      myprintinbox(myarray[i]+" ");
    }
}
```

Methods and Variables

- Methods have to be self-contained.
- The only variables you can use are
- 1) the formal parameters of the method
- 2) any variables you declare inside the procedure (**local variables**)
- 3) global variables for the class as a whole (pretend you didn't see this - dont use globals)
- A variable declared in main is **local** to main
- So you can't use it in print_array
- In the above print_array method, length is a local variable.
  - it only exists inside print_array
  - there may be other variables declared in other methods with the same name
  - just because they have the same name doesn't make them the same variable!
- In the print_array method, myarray is aformal parameter.
  - it only exists inside print_array too
  - but it's initialised with whatever value is passed each time print_array is called.

Methods

- If you can write something using methods, you could in principle write it without (though it can become very messy!)
- So one way to approach methods is to ask which bits of the programs you are writing are suitable to be turned into methods.

When should it be a method?

Part of a program is a good candidate for being turned into a method if:

- it performs a clear task
  - if you cant easily write a clear description of what it does for a comment (eg `// inputs an integer`) then its probably not a method
- it does not use very many variables
- you find yourself writing this part more than once

Methods

- Sometimes you have to modify your program a bit to make it possible to extract methods from it.
- For example: you may need extra variables

Making a method: **In-line code:**

```
...... // some code
int length = myarray.length;
for (int i=0; i<length; i++)
{
  System.out.println(myarray[i]+" ");
}

...... // some more code
```

Pull out a method - add a header:

```
static void print_array (int[] myarray)
```

```
{
  int length = myarray.length;
  for (int i=0; i<length; i++)
  {
    System.out.println(myarray[i]+" ");
  }
}
```

And left in the program:

```
...... // some code
print_array(myarray);
...... // some more code
```

Notice can change the name of the argument:

```
static void print_array (int[] a)
{
  int length = a.length;
  for (int i=0; i<length; i++)
  {
    System.out.println(a[i]+" ");
  }
}
```

In-line code:

```
...... // some code
print_array(myarray);
Print_array(anotherarray);
...... // some more code
```

Make method return a string:

```
static void print_array (int[] a)
{
  String array_string = "";
  int length = a.length;
  for (int i=0; i<length; i++)
  { array_string = array_string + a[i] + " ";
  }
  return array_string;
}
```

And then in program:

```
...... // some code
System.out.println(print_array(myarray));
...... // some more code
```

..which has big advantages

This approach has advantages:

- function is more flexible in use
- only solution where method cannot throw exception
- simple interface

So best way may not be most obvious

Procedures A Warning

- Parameter passing in Java is call-by-value.
- This means that: the (formal) parameter of a procedure (or function) acts like a local variable when the procedure is called it is initialised with the **value** of the actual parameter
- this means that if you call a procedure with a variable as its actual parameter, it's the **value** of the variable that gets used, not the variable itself
- this means that procedures cannot change the values of variables

- So a call wombat1 (a); does not change the variable a
- (except…see next lecture)
- functions cant either but they can at least return a value to be used in an assignment so that can change the value of the variable back where the method was called eg a = wombat2(a);

Procedure calls

- Suppose we have a procedure:

```
static void proc (int n)
{ .... body ....
}
```

which is called later in the context:

```
.... stuff ....
proc(sink, z);
.... more stuff ....
```

Procedure calls…equivalent to

Then we get exactly the same results as if we had:

- first renamed all the variables in the procedure and all it's parameters
- so that they did not clash with variables in scope at the point of call.

```
.... Stuff....
// start block corresponding to procedure call
{ // declare (formal) procedure parameters as variables
  // and bind actual parameters (in call) to them

  int n = z;
  // execute body of procedure
  .... body ....
}
// end block corresponding to procedure call
.... more stuff ....
```

Example

```
static void proc (int n)
{
  int m = n*n;
  System.out.println("Result is ");
  System.out.println(n*n);
}

static void main (String [] p)
{
  String s = input("Enter number: ");
  int z = Integer.parseInt(s);
  proc(out, z);
  System.out.println(”Bye");
}
```

…is equivalent to

```
static void proc (int n)
{
  int m = n*n;
  System.out.println("Result is ");
  System.out.println(n*n);
}

static void main (String[] p)
{
```

```
    String s = input("Enter number: ");
    int z = Integer.parseInt(s);
    // proc(out, z);
    {  int n = z; // set up parameter
       int m = n*n;
       System.out.println("Result is ");
       System.out.println(n*n);
     }
     System.out.println(”Bye");
}
```

Function calls

- A similar expansion holds for function calls.
- But it is not quite so clean:
- We need a variable (let's call it return though you wouldn't really be able to call a variable that as it's a keyword) to hold the result of the call. And we need to replace `return expression;`
- by `return = expression;`
- And there's a complication that function calls can come in places where we can't put blocks. . .

Function calls: example

```
static int square (int n)
{
 int m = n*n;
 return m;
}

static void main (String[] param)
{
 String s = input("Enter number: ");
 int z = Integer.parseInt(s);
 System.out.println("Result is: ” + square(z));
}
```

this is equivalent to

```
static void main (String[] param)
{
 String s = input("Enter number: ");
 int z = Integer.parseInt(s);

 int return;
 { int n = z;
   int m = n*n;
   return = m;
 }
 // System.out.println("Result is: ” + square(z));
 System.out.println("Result is: ” + return);
}
```

Method calls and blocks

- The person most strongly associated with the idea that equivalences like these should be systematic and uniform is Peter Landin (who was Emeritus Prof of Computer Science here at Queen Mary).
- Landin observed that if such equivalences really work then they are very useful in understanding and restructuring programs.
- If they don't, then they are very useful for introducing bugs!

Abstract Data Types

- We've seen how methods give us a way to abstract away from (ie hide) implementation details from the rest of the program.
- When writing other methods you only need to know the name and what the method does to use it not how it is implemented.
- We used getter and setter methods to do a similar thing for records
    - we hid the details of the record structure
    - we just had a new type to use, and methods to access it.
    - Changes to the implementation then dont need changes elsewhere
- We can use the same trick much more generally to create a version of what are known theoretically as abstract data types (ADTs) invented by Barbara Liskov and Stephen Zilles
- We use the type system with methods to hide the details of how data structures are implemented.

Strings as Abstract Data Types

- You've actually been using abstract data types from the start!
- For example how is a String value actually implemented?
- You just think of it as a string of characters like "abc" and know methods like concatenation to manipulate it.
- But in all the programs you've written you've never had to worry about how it is actually implemented.
    - Is it just an array of characters underneath?
    - Is it something else?

When does a string stop

- One issue to deal with is how their length is recorded ie when do they stop
- Here are two ways the inventors of the language could have dealt with that:
    - make a string a sequence of characters in memory followed by a terminating character to say where they end
    - as a length (a number) followed by that many characters.
- Then there is a separate issue of how characters are represented …
- You didn't need to know which though to write programs with strings!
- It is all hidden by the abstraction..and that is a good way to write programs!

Abstract Data Type

An Abstract data type is a model for a data type where the actual details of how the data type is really implemented are hidden. The data type is defined in terms only of how it is seen to someone (the programmer) using it via the operations that can be applied to it and values that are visible.

Breaking the abstraction

In practice many uses of abstract data types are not true ones as details of the implementation leak (eg the implementation of integers are not perfect in programming languages due to the fixed amount of space they are stored in - you can make the integer overflow to something to big to be stored and break the abstraction!)

Abstract Data Types in Programming Languages

When programming we can use the idea of an abstract data type to make code easier to write, modify and understand

Getter and Setter Methods

This is what getter and setter methods combined with using the class mechanism to create new types gave us.

We hid the details of the record fields and what exactly was stored and what calculated when asked for.

We can use to to hide the implementation of more complicated data structures too

We make them into abstract data types

Example: A queue

Suppose we want to write a program that keeps track of olympians who are being interviewed. They will be interviewed in the same order as they arrive (first-in-first-out).

- We need to implement a Queue data structure to use int he program.
  What does that mean?
- Well think of it as an abstract data type.
- Don't worry about how it will work we just ask:
- What operations can you do on a queue?
- You can
    - have someone (or something) join the queue
    - have someone leave the queue
    - create a new queue
    - see who is currently at the front of the queue
- We need a method for each:
    - joinQueue is given a queue and returns an updated queue with a new name added
    - leaveQueue is given a Queue and returns a new Queue with the front of the queue removed
    - createEmptyQueue creates and returns an empty queue of a given size.
    - firstInQueue returns the front of the queue but leaves the queue unchanged

The following simple sequence of commands shows how we might want to use these methods to add and remove people from the queue:

```
Queue q = createEmptyQueue(10);
System.out.println(firstInQueue(q));
q = joinQueue(q,"Alistair Brownlee");
System.out.println(firstInQueue(q));
q = leaveQueue(q);
q = joinQueue(q,"Mo Farah");
System.out.println(firstInQueue(q));
q = joinQueue(q,"Laura Trott");
q = joinQueue(q,"Nicola Adams");
q = joinQueue(q,"Amir Khan");
System.out.println(firstInQueue(q));
q = leaveQueue(q);
System.out.println(firstInQueue(q));
q = leaveQueue(q);
System.out.println(firstInQueue(q));
```

This way of interacting shouldn't change even if we change how the queue is implemented! The above should always work

So how is it implemented?

Here is one simple (and slightly naive as it would be slow) way to implement our Queue data type with the above operations.

Creating a Queue type

- We can use a record with fields of an array to hold the entries.
- We also need some way of keeping track of how full the array is.
- We will therefore use a field that tells us how many are in the queue

```
class Queue
{
   String [] entries; // Where the queued values are stored
   int numberqueueing;      // The number of entries in the queue
} // END class Queue
```

- We need to decide how we will use it.
- There are lots of options (and the abstract data type approach means we can switch between them without affecting the rest of the program!)
- One easy way is to have position 0 of the array always the front of the queue.
- We will shuffle the entries up whenever someone leaves
- Like having a line of chairs and when the first person gets up, everyone moves up a chair

Front of the queue

- We find out who is at the front of the queue by checking position 0.
- Notice we need a way to deal with empty queues!

```
public static String firstInQueue(Queue q)
 {
      if (q.numberqueueing == 0)
          return "Queue Empty";
      else
      {
          String firstentry = q.entries[0];
          return firstentry;
      }
 }
```

Joining the queue

- To join the queue we need to add an entry at the next free position
    - this is the number in the queue as the array index counts from 0
- Then increase the number queueing

```
public static Queue joinQueue(Queue q, String newentry)
 {
     if (q.numberqueueing < q.entries.length)  // Still space
     {
          q.entries[q.numberqueueing] = newentry;
          q.numberqueueing = q.numberqueueing + 1;
      }
      return q;
 }
```

Leaving the queue

- To have someone leave the queue we need to add an entry at the next free position
- Notice only the first person in the queue can leave at any time
- We use a for loop to shuffle everyone up
- Then change the entry stored as the next free one
- We need to check its not empty first though

```
public static Queue leaveQueue(Queue q)
{
    if (q.numberqueueing != 0) // queue is not empty
    {
        for (int i = 0; i < q.numberqueueing; i++)
        {
           q.entries[i] = q.entries[i + 1];
                                  //Shuffle all entries down
        }
        q.numberqueueing = q.numberqueueing - 1;
    }
    return q;
}
```

Creating an empty queue

- We need to create a new record of type Queue,
- We need set the number queueing to start with to 0
- We also need to create a new array of the right size to store the data

```
public static Queue createEmptyQueue(int size)
{
    Queue q = new Queue ();
    String [] a = new String[size];
    q.entries = a;
    q.numberqueueing = 0;
    return q;
 }
```

- Notice its quite tricky to get the detail right here
- Having created these methods though we dont have to worry about making mistakes elsewhere in the program
- the details are always right as long as we only use the methods to access our Queue.

Object-oriented programming

- Object-oriented programming takes these ideas further and gives new and convenient ways to implement data types like this
- It gives a way that the compiler can enforce you only using the methods that provide the interface so you cant make that kind of mistake
- However the syntax needed is different and there are a bunch of new concepts to learn to do it properly - you will see how to do it next semester if you do the object-oriented programming module

Unit 7 Example Programs and Programming Exercises

Here is a typical program that has been split into lots of methods that pass information around as arguments and by returning results.

```
/* ***************************************
AUTHOR Paul Curzon
  Giving status of endangered animals up to a number
  indicated first by the user.
****************************************** */

import java.util.*;

class endangered
{
  public static void main (String [] param)
  {
    int numberAnimals = 0;

    // First get the number of queries
    numberAnimals = howmanyanimals();

    //Then ask for that many animals and give status
    aretheyendangered(numberAnimals);

    System.exit(0);
  }

  /* ************************************
  Give endangered status of a sequence of animals given the number of
animals */
  public static void aretheyendangered (int numberAnimals)
  {
     //Then ask for that many animals and give status
    for(int i = 1; i <= numberAnimals; i++)
    {
      String testAnimal;
      String animalStatus;

      testAnimal = input("Give me the name of animal " + i);

      animalStatus = statusofanimal(testAnimal);

      System.out.println("The " + testAnimal  + " is " +
                         animalStatus);
    }

    return;
  }

  /* ************************************
  For a given animal return its status as a string message  */
  public static String statusofanimal (String animal)
  {
     String status = "";

     /* Now set message based on status of animal */

      if (extinct(animal))
```

```
          {
            status = "unfortunately now extinct.";
          }
      else if (endangered(animal))

          {
            status = "endangered and needs protection.";
          }
      else if (vulnerable(animal))
          {
            status = "vulnerable";
          }
      else if (notindanger(animal))
          {
            status = "not in danger.";
          }
      else
          {
            status = "one I've not heard of.";
          }

      return status;
  }

  /* ************************************
     Get the number of animals to check from user  */
  public static int howmanyanimals ()
  {
      int numberOfAnimals = inputInt("How many animals do you want to
check?");
      return numberOfAnimals;
  }

  /* ************************************
     Is the given animal extinct?          */
  public static boolean extinct (String animal)
  {
       if ((animal.equals("Dodo") ||
           (animal.equals("Passenger Pigeon"))))
          {
            return true;
          }
      else
          {
            return false;
          }
  }

  /* ************************************
     Is the given animal endangered?       */
  public static boolean endangered (String animal)
  {
       if ((animal.equals("Blue Whale") ||
            (animal.equals("Albatross"))))
          {
            return true;
          }
      else
```

```
          {
            return false;
          }
  }

  /* ************************************
     Is the given animal vulnerable?       */
  public static boolean vulnerable (String animal)
  {
       if (animal.equals("Cheetah"))
          {
            return true;
          }
      else
          {
            return false;
          }
  }

  /* ************************************
     Is the given animal fine?             */
  public static boolean notindanger (String animal)
  {
       if (animal.equals("Rabbit"))
          {
            return true;
          }
      else
          {
            return false;
          }
  }

  // Input Integer
    public static int inputInt(String message)
    {
       return Integer.parseInt(input(message));
    } // END inputInt


   // Input strings
    public static String input(String message)
    {
       Scanner scanner = new Scanner(System.in);
       String ans;

       System.out.println(message);
       ans = scanner.nextLine();

       return ans;
    } // END input

}
```

Only try the following exercises once you feel confident with those from the previous week/unit. By now you should be using methods as standard, but if not the early exercises will help.

Method call and abstract data type Example Programs

Compile and run the following programs ECS401 website – week 7 area, experiment with them modifying them to do slightly different things.

- **PrintP8P.java** Print big letters using methods to do the bits. Modify it to include other letters.
- **PrintP8Pagain.java** Print big letters using methods to do the bits but this time pass around the character to use to do the printing
- **sum3.java** A simple example of a function that takes 3 arguments. Add your own calls with different values and as method arguments and work out what they do
- **QueueProgram.java**: A simple example of an abstract data type. Add olympians to a queue.

Simple exercises on methods

Hello procedure - doing hte same thing multiple times by calling methods

Write a program that prints "Hello " followed by your name to the screen twice. It should call a procedure (twice) called hello that actually does the printing.

Hello procedure - can you pass arguments

Write a program that inputs two string names from the user, then passes them in turn to a procedure called hello. The procedure should print "Hello " followed by the given name each time it is called. The name should therefore be a parameter to the procedure.

Simple calculation - can you return results

Write a program that inputs an integer a, and outputs a+1. Use a function called plusone to calculate a+1 returning the result .

sum to n

The result of adding the first n numbers can be calculated using the formula: sum = n*(n+1)/2. Write a function to calculate this value and write a program that tests it for test calls sum(1), sum(5) and sum(10).

Repeated sum to n

Modify your sum test program from the above exercise so that a special test method uses a for loop to call the sum method for all values from 1 to 10.

Printing lots of characters

(a) Write a program that inputs two integers m and n and outputs an m-by-n rectangle of *'s. Define a procedure

```
static void printchar(char c, int n) …
```

which outputs the character c n times. Use it to output each row of *'s.

(b) Write a program that inputs one integer m and outputs an m-line high triangle of *'s with the right angle at the bottom-right. Use the same procedure

```
static void printchar(char c, int n)
```

to output the spaces and the *'s.

Once you have done these, try the next assessed exercise. The assessed exercises must be done on your own. If you have problems with them then go back to the non-assessed exercises.

Simple exercises on abstract data types

Queues

Write a user interface to the queue - a program that runs a loop giving a menu of options of adding to the queue and leaving the queue or quitting. Always printing who is at the front of the queue. You should use the existing unmodified queue methods.

Medal winners

Create a new abstract data type of Olympians that stores details of lots of Olympians - storing for each their name and the kind of medal they won from Gold, Silver and Bronze (assume it stores the highest medal only) . One or more arrays may be used as the underlying record to store the details. Operations are to create an empty database, to add a record to the database, and to return the highest medal of a named olympian. Write a simple test program that illustrates the use of the abstract data type.

Endangered Animals

Modify the endangered animals program provided to use an abstract data type as a database to store the vulnerability levels of animals. It should provide a method to allow animal data to be stored (implemented behind the scenes in a hidden array), and then methods that check if a given animal is extinct, endangered, vulnerable, not in danger, or unknown.

Queues Again

1) Modify the queue data structure in the example queueprogram.java to have a different implementation. The use of the queue (main method) should not change at all In particular change the implementation of the record so that rather than storing the number of entries currently in the queue, it has a field that stores the position of the last person in the queue (ie one less than the number of entries)

2) Now modify the original queue implementation so it no longer has to shuffle entries up. It should have a field holding the position of the front of the queue and one that stores the number of entries in the queue. The first position should move as entries leave the queue wrapping round from the end to the beginning again.

Stacks

Implement a stack data structure (i.e. a last in first out queue). The entry to remove is always the most recent entry added.

Additional Exercises on methods

The following are additional, harder exercises. The more programs you write this week the easier you will find next week's work and the assessments and the quicker you will learn to program. If you complete these exercises easily, you may wish to read ahead and start thinking about next week's exercises.

Rotate vowels

Write a program that takes a string sentence typed by the user and rotates the lower-case vowels. That is, change all occurrences of

'a' to 'e' 'e' to 'i' 'i' to 'o' 'o' to 'u' 'u' to 'a'

Use functions you write to decide if a letter is upper case or lower case, and if it's a vowel or not. Use another function to rotate the vowels.

Quadratics

Write a program that inputs three integers, a, b and c and prints a message describing the roots of the quadratic ax2+bx+c. (This quadratic has two real roots if the

discriminant b2-4ac > 0, one repeated real root if b2-4ac = 0, and two imaginary roots if b2-4ac < 0). Use a function to calculate b2-4ac.

RASTER triangles

'RASTER' printing of triangles

Write an application that outputs rows of large triangles across the screen such as

```
*     *     *     *
**    **    **    **
***   ***   ***   ***
****  ****  ****  ****
********************
*     *     *     *
**    **    **    **
***   ***   ***   ***
****  ****  ****  ****
********************
```

The program should input 3 numbers from the keyboard giving

(a) the size of each triangle (i.e. width in characters and height in lines)

(b) the number of triangles to be output in each row

(c) the number of rows of triangles to be output.

In the example output above the triangle size is 5, the number of triangles per row is 4 and the number of rows is 2.

Hint: You have to realise that your program can draw just one component line of each triangle at a time. In doing this it will output a sequence of *'s and then a sequence of spaces. Define a method printchar to do this.

Redo existing programs

If you finish all of the above exercises then go back to some other earlier programs of yours and see how they could be improved by splitting them into methods more elegantly. Each method should do a single well-defined and generally useful task. Are there methods you can write once and then use in several programs?

A program to print tomorrow's date, for example, might be improved with a function

```
static int days_in_month(int month, int year)
```

that returns the number of days in the given month and year. This might itself make use of a function

```
static boolean is_leap_year(int year)
```

that returns true if the given year is a leap year and false otherwise.

Avoiding Bugs

TIP 1 : Write a comment for every method making clear what it does, how it is expected to work and what each argument is for. Note situations it is not designed to be used for.

Finding Bugs

TIP 1 : One reason for using methods is that they can each be tested thoroughly separately – there is no point testing (or even writing) the rest of the program until the individual methods work – so write methods one at a time and make sure they work one at a time before moving on to the next or to the full program that uses them.

Unit 7 Exam Style Question: Methods / Abstract Data Types

Question 1 [25 marks]

a. [6 marks]
Compare and contrast the following, **illustrating** your answer with examples:
i) function call
ii) procedure call

b. [9 marks]
Write an equivalent code fragment to the following that does not use a call to a method, **explaining** the steps involved in the conversion.

```
public static void main(String [] params)
{
  int temp = inputInt("Give me a temperature in F: ");

  convert(temp);
}

static void convert(int f)
{
   int newtemp = (f - 32) * 5 / 9;
   System.out.println("That is " + newtemp + "C");
   return;
}
```

c. [10 marks]
The Garage, Mile End Marvellous Motors sells used cars at discount prices. Given the full price of a car, the salespeople always knock off 10% plus give a further discount of the agreed price of any existing car that the customer traded-in. **Write** a Java method that calculates the value of a car given its full price and the trade-in price of the car being exchanged, both passed as arguments to the method. In addition write a main method that calls this method repeatedly with values input from the keyboard until the user indicates they wish to quit.

Question 2 [25 marks]

a. [9 marks]
Using the example of a priority queue, a queue where entries leave in order of a priority given to them at the point when joined the queue, **explain** how methods can be used to create abstract data types, hiding the details of a data structure from the rest of the program.

b. [6 marks]
Justify the statement "Abstract data types help make programs easier to write, understand and change."

c. [10 marks]
Write a program to answer queries about the zones of overground stations. It should provide a menu running in a loop that gives options of adding a new station, looking up the zone of a named station, or quitting. The underlying information about stations (pairs of names and zones) should be stored as an array of station records. However it should use an abstract data type approach to implement this, ensuring that the underlying representation can easily be changed.