Chapter Eight: Thinking Procedurally and Concurrently

User defined methods

A *method* is a subroutine that may or may not return a value. When they do return a value, they work like a *function*; when they do not, they work like simple *procedures*. In Java there is no distinction between them. Procedures are declared using `void` to indicate no return value.

```
public static void main(String args[])
{
   for (int i = 1; i < 100; i++)
   {  for (int j = 1; j < 100; j++)
      {  for (int k = 1; k < 100; k++)
         {  if ( good(i,j) && good(j,k) && good(i,k) )
               IBIO.output( I + "   " + j + "   " + k );
         }
      }
   }
}

static boolean good(int a, int b)
{  int     x = a * b + 1;
   int     y = (int) (Math.sqrt(x)+.5);
   return ( y * y == x );
}
```

Diophantine

Equations that have whole numbers as solutions are called diophantine. The above program attempts to find all numbers that are less than 100 and have the property that when they are multiplied together they are one less than a perfect square. The simplest example is 1, 3, 8 because $1*3 = 2^2-1$, $3*8 = 5^2-1$, $1*8 = 3^2-1$. The program above uses a subroutine to test each pair of numbers to see if they meet such condition.

Pr 8.1 Change the program so that no duplicates will be printed out.

Pr 8.2 Change the program so that it will find 4 numbers with the above property – that any two of them multiply together to make a number one less than a perfect square. (need to loop to 200 to find one answer)

Pr 8.3 Write a program that will find all the numbers less than 100 that have the property that $a^2 + b^2 = c^2$. (100 possible answers)

Pr 8.4 Write a function gcd which calculates the greatest common divisor of two numbers a and b. It does this by subtracting the smaller number from the larger and continues to do this until the numbers are the same. e.g. start {36, 27} then next stage {9, 27} then {9, 18} then {9, 9} now stops because both numbers are the same. 9 is the gcd of 36 and 27.

```
static int gcd(int a, int b)
{
   return x;
}
```

Pr 8.5 Change the program in 8.3 so that all the duplicates are removed (now 50 answers) and also remove all multiples. 3, 4, 5 is one of the answers and we do not want 6, 8, 10 to also be an answer (16 answers). Use the function `gcd()` created in Pr 8.4

Chapter Nine: Thinking Abstractly (Arrays)

Random Numbers

In Java random numbers are decimal numbers between 0 and 1. These are very useful for simulation experiments on a computer. If we want a random whole number like throwing a dice we multiply this number by 6 add 1 then convert to an integer. In the next example we use the computer to calculate 20 random numbers simulating the throw of a dice.

```
for (int i = 0; i < 100; i++)
{  double xx = Math.random() * 6;     //Math.random - decimal
   int    yy = (int)(xx + 1);         //change to number 1 to 6
   IBIO.output(yy);
}
```

Pr 9.1 Write a program that will generate 100 random numbers from 1 to 6 as in the above program and find the average of them.

Arrays

If we wanted to investigate the 100 numbers that we created above then we must have a way of saving them. The way of doing this is by using an array. This is a list of memories that use the same name. If we decide that the name of the array was "num" then the memories would be labelled "num[0], num[1], num[2], etc ". Notice that the first one is num[0] and not as expected num[1]. Like all variables an array must be declared before it is used. With an array the size that you want it to be must also be stated. This size is called its "dimension". The first line of the following program is the line that creates the array.

```
   int[ ] array = new int[size];
```

```
int[] num = new int[100];    // create the array

for (int i = 0; i < 100; i++)
{  double xx = Math.random() * 6;
   num[i] = (int)(xx + 1);
}

for (int i = 0; i < 100; i++)
   IBIO.output(num[i]);
```

This program will create 100 numbers in an array called "num" and then print them out.

Pr 9.2 Change the program above so that it uses a function called random(int) to make the random number. The finished program will look like the one below.

```
int[] num = new int[100]; //create the array

for (int i = 0; i < 100; i++)
   num[i] = random();     // your built in function

for (int i = 0; i < 100; i++)
   IBIO.output(num[i]);
```

Pr 9.3 Write a program that will generate 100 random numbers from 1 to 6 as in the above program. Then it will print them out. First all the 1's then all the 2's etc.

```
11111111111
22222
33333333
4444444444
555555
666666666666
```

To do this you will need to have one loop that counts from 1 to 6. Inside that loop another loop counts from 0 to 99, printing out the values. To print out a number without going to the next line use the command out() instead of output().

Pr 9.4 Write a program that will generate 100 pairs of random numbers from 1 to 6. It will save the sum of these numbers in an array. So at this stage every element of the array will have a number from 2 to 12. Then to print out a bar graph showing how many 2's, 3's there are. Use the method above to print the numbers from 2 to 12 in the margin lined up. This is like throwing two dice and adding the numbers together.

```
2  XXXXXXXX
3  XXXXXX
4  XXXXXXXXXX
5  XXXXXXXXXXX
6  XXXXXXXXX
```

Use the segment of the program below to put the numbers into the array.

```
   for (i = 0; i < 100; i++)
      num[i] = random(6) + random(6);
```

To do this the numbers on the left must be lined up so you use the Pad function that was used in Sheet 7.

Final Static

Often we need to define a constant that is the same throughout the program but we may want to change later. In the above example we use 100 and we may want later to change it to 1000 but do not want to go through the program and change it everywhere in the program.

```
public class Simple
{  public final static int SIZE = 100;

   public static void main(String[] args)
   {  int[] x = new int[SIZE];
      ...
```

The above segment shows how to add a constant called SIZE that is set to 100 at the beginning of the program and retains that value throughout. Constants are always capitalised, final, and static.

Pr 9.5 Change the last program so is uses an `int` constant called `SIZE`. Then run the program for 1000.

Chapter Ten: Thinking Ahead (Strings)

String & char

We came across strings in Chapter 7. Now we will learn how to input a string.

```
String xx = IBIO.input("enter your name ");
IBIO.output(xx);
```

This is a new type of variables that we can have. Strings are not one of the primitive data types, but an actual class, so it comes with several built-in methods.

```
public static void main(String args[])
{
   String ss = IBIO.input("enter your name ");   //input your name
   char[] xx = ss.toCharArray();                 //make into an array

   for (int i = 0; i < ss.length(); i++)
      IBIO.out(xx[i]);
}
```

This program will read in a sequence of letters and print them out again. The program will save the letters first in a String and then into an array.

Remember that the size of the array is exactly the right number for the number of characters that you typed in.

Pr 10.1 Write a program that will read in a sequence of letters and print out the word and then the word reversed and then the combination of both (note that the last letter is not repeated).

```
Input a word: TRAIN
TRAIN
NIART
TRAINIART
```

Pr 10.2 Write a program that will read in a sequence of 0's and 1's as a binary number and change the input form from binary to decimal. Remember that what is being read in are characters and not numbers. Your program must test if only 0's and 1's are in the input. Otherwise output error message. To test if the digit is a '0' or a '1' you must do the following

```
char x;
if (x == '0') // is x the digit 0
if (x == '1') // is x the digit 1
```

Refer back to your notes on binary to decimal conversion. Take the number so far, multiply it by 2, and then add next digit to get the next number.