Table of Contents

Chapter One: Introduction – Thinking Procedurally

Java

Java is made up of many commands. In this tutorial we will cover the most important of them. We will not cover them all but those required by the IB syllabus.

A program is a sequence of commands or instructions. You enter commands into the editor to make a program and then compile the program (change it into a form that the computer understands) and then run the program.

Type

The information that a computer can hold can be many different type. There are complicated types like iMovie and spreadsheet files and there are basic types called primitive types of which all other information is made up from: boolean, byte, char, int, double.

Variables

A variable is the name of a memory just like the memory on a calculator. With Java you can have as many memories as you like. Variables come in different types depending on the nature of the information that is to be stored. The type of variable that we will first use is called int. This is used to store a whole number.

Any sequence of letters or numbers (providing it starts with a letter) can be used to denote a variable. Lower case letters are different from upper case letters. It is always best to use variable names that indicate the information that will be stored there. If a variable is used in a program then it stands for the information that was put in it.

Before a variable can be used it must be declared. (Note in the following examples the declaration of class has been omitted).

```
public class Example1
{
   public static void main(String args[])
   {  int x = 17;
      IBIO.output("the number was " + x);
   }
}
```

Operations

We use + for plus, - for subtract, * for multiplication, / for division and % for the modulo/remainder.

```
public class Example2
{
   public static void main(String args[])
   {  int a = 17;
      int b = 23;
      int c = a * b;
      IBIO.output ("the product of " + a + " and " + b + " is " + c);
   }
}
```

output() prints the contents of the brackets in one line. You can put several parts together using +. Strings must have double quotes around them.

Input

The command to input an integer is inputInt(prompt):

```
   public static void main(String args[])
   {  int   a = IBIO.inputInt("enter a number  ");
      int   b = IBIO.inputInt("and another one ");
      int   c = a * b;
      IBIO.output( "the product of " + a + " and " + b + " is " + c );
   }
```

Loops

Loops are very important in Java because they enable us to repeat something over and over again. The first loop we will learn is the **for** loop.

```
for (starting condition; ending condition; increment)
{     statements
}
```

```
   public static void main(String args[])
   {  for (int i = 0; i < 20; i++)
      {  IBIO.output("hello");
      }
   }
```

There are three parts to a **for** loop:

The starting condition	in example *i* starts at 0
the stopping condition	in example the condition is that *i* is less than 20
the increment	in example *i*++ means to increase *i* by 1

This loop works by first setting the starting situation, then doing the test, then does the body of the loop, then incrementing and repeating.

Pr 1.1 Change the program so that you enter in a number and then the program will print your name down the screen that number of times.

```
int   number = IBIO.inputInt(“enter starting number”);

for (int i = 0; i < 20; i++)
{  IBIO.output(number);
   number = number+ 7;
}
```

The program above will start from the number 3 and make a sequence by adding 7 each time.

Pr 1.2 Write a program that allows you to input the number of steps, the starting point and the increment and then prints out your sequence. So for example step = 4, start = 3, increment = 2. Then the sequence will be 3 5 7 9.

Pr 1.3 Write a program that will print out the first 10 numbers, their squares and their cubes.

Pr 1.4 Write a program that displays the first 100 terms of the triangular sequence. This is the sequence that goes 1,3,6,10,15,21,… The rule is that you add on 2, then add on 3, then add on 4, etc.

Pr 1.5 Write a program that displays the first 20 powers of 2. Number and then the power.

Pr 1.6 The Fibonacci sequence is obtained by adding together the two terms of a sequence to get the next term: 3 , 4 , 7 (because 3 + 4 = 7), 11, 18, etc. Write a program that allows you to input the number of terms of the Fibonacci sequence that should be calculated and output.

Chapter Two: Thinking Logically

if

In programming there are three main concepts. The first is sequence – going from one statement to the next; the second is looping – repeating something over and over again, and the third is decisions –changing what happens depending on the situation.

```
public static void main (String args[])
{
   int   n = IBIO.inputInt("input a number between 50 and 60 ");
   if (n > 60)
      IBIO.output("that number was too big");
}
```

The main statement that is used to control decisions is the **if statement**.

```
if (expression)          |  if (expression)
   one statement;        |     {  many statements  }
```

curly brackets { }

In Java programs you can either use one statement, or several statements enclosed by curly brackets. There are two rules for aligning up brackets. If the statement goes over one line then the brackets line up horizontally, if the statements go over many lines then the brackets align vertically like in the previous chapter.

Inside the brackets is an expression. These are called boolean expressions that work out to either true or false. If the statement is true then the next statement is done (if you need to include more than one statement, then it must be put into a block (ie surrounded by { and }). If the statement is false then the next line is skipped.

There are 6 relational operators – you must not leave a space between the symbols:

>	larger than	**>=**	larger or equal to
==	equal to	**!=**	not equal to
<	less than	**<=**	less than or equal to

Pr 2.1 Alter the above program so that it will comment if you entered a number less than 50, or a number larger than 60.

% Remainder

This is an operator that gives the remainder of two whole numbers. So 23 % 7 will be 2 because the remainder when you divide 23 by 7 is 2

Pr 2.2 Write a program that will let you enter a number and will reply with EVEN or ODD depending if the number you entered was even or odd.

else

This command allows us to choose an alternative, like in the following example.

```
if (expression)          |     if (expression)
   statement             |           statement
else                     |     else if (expression)
   statement             |           statement
```

```
int x = IBIO.inputInt(input a number ");
if (n > 50)
   IBIO.output("larger than 50");
else
   IBIO.output("smaller than 50");
```

Pr 2.3 Change your program for writing EVEN and ODD so that it uses the else command.

One of the problems when we print numbers they do not align up on the right as they should. The following program segment fixes this

```
   for (int i = 0; i < 20; i++)
   {  if (i < 10)
         IBIO.output("  " + i); // there are 2 spaces between the ""
      else
         IBIO.output(" " + i);     // there is only 1 space between the ""
   }
```

Pr 2.4 Write a program to print out the cubes of numbers from 1 to 10 so that they line up on the right using the same idea as above.

Pr 2.5 Write a program to print out the numbers from 1 to 100 but omit printing all the even numbers. Do this by using `for (int i = 1; i <= 100; i++)`

Pr 2.6 As above write a program to print out the numbers from 1 to 100 but omit printing all the even numbers and all the numbers divisible by 3.

&& AND , || OR

These commands allow us to combine two relations together.

a > 3 && b < 2. This is true when *a* is larger than 3 and at the same time *b* is smaller than 2.
a > 3 | | b < 2. This is true when *a* is larger than 3 or *b* is smaller than 2.

Pr 2.7 Change your last program so that it uses && instead of two if statements.

Pr 2.8 Write a program to count all the numbers from 1 to 1 000 000 which are not divisible by 2 or 3 or 5 or 7. Output the results (the answer is 228571)

Chapter Three: Thinking Concurrently

do – while Loop

Apart from the for – loop there are two other ways of repeating. One is the do – while loop. This tests its expression at the end of the loop and if it is true, then the loop will continue.

```
do
{  statements
   ...
} while ( expression );
```

```
public static void main(String args[])
{  int x;
   do
   {  x = IBIO.inputInt("enter a number less than 100 ");
   } while (x >= 100);
   IBIO.output("thank you");
}
```

In this example the program will continue to ask for a number if you type in a number larger than 100.

Pr 3.1 Change the program so that it only accepts numbers that are even and are larger than 0 and less than 100.

Primes

A prime is a number evenly divisible only by the number itself and one.

```
public static void main(String args[])
{  int i = 1;
   int x = IBIO.inputInt("Enter a number: ");

   do
   {  i++;
   } while (x % i != 0);

   IBIO.output(x + " is divisible by " + i);
}
```

This last program will accept a number and keep dividing it by 2,3,4,5,6, etc. until it finds one number that goes evenly into it. Note that it will always find one because the number goes into itself. So if a prime number was input into the program then the output would be that number itself, or else the output would be the smallest number that goes into it. Note that in the declaration of i it has also been initialised.

Pr 3.2 Change the last program so that it only accepts numbers that are greater than 1 and outputs the word prime if indeed the number is prime and otherwise outputs the smallest prime that goes into the number.

One useful way of testing your program is to put the main part into an infinite while loop.

```
do
{
   // main part of program to test here
} while ( true );
```

Digit sum

Given any number the following program will add up the digits in that number. So if 345 was entered, the program would calculate 3+4+5 = 12. It would start with n = 345, then it would divide 345 by 10 and write down the remainder, which is 5, then it would divide 345 by 10. Because we are dealing with whole numbers it would get a whole number answer and write down 34. This process continues until there are no more digits left in the number.

```
public static void main(String args[])
{
   int sum = 0;
   int n   = IBIO.inputInt(" enter a number ");

   do
   {  int digit = n % 10;    // get right most digit
      sum = sum + digit;     // add to units digits
      n = n / 10;            // make new number
   } while ( n != 0);
   IBIO.output("the sum of the digits of the number is " + sum);
}
```

Pr 3.3 Change this program so that it will add up the cubes of the digits of the number. So if the input number was 345 it would go $3^3 + 4^3 + 5^3$.

Pr 3.4 Consider the sequence. If a number was even then the next number would be half of that number, if the number was not even then the next number would be got by multiplying that number by 3 and then adding 1. eg if 7 was the starting number then that number is odd so it is multiplied by 3 and 1 added to get 22, 7, 22, 11, 34. This sequence continues until it eventually arrives at 1. Write a program that will allow you to input a number and then it continues this sequence until it eventually arrives at 1. I want to know how many steps it takes. Eg starting at 3 the sequence is 3, 10, 5, 16, 8, 4, 2, 1 and that takes 7 steps.

while Loop

This is an alternative way of expressing a loop. In this form, the test is done first, before the execution of its statements.

```
while ( expression )
{  statements
   ...
}
```

Chapter Four: Thinking Logically (Switch Statement)

Switch, case, break, default

This is a way of making decisions based on many alternatives. Notice carefully the syntax of the program below. The switch statement is made, then in brackets after it has the variable that is examined. Then we must have the alternatives enclosed in { }. Lay out your program the same as this. Each alternative starts with the word "case", a number and then the alternative followed by colon ":". At the end of each case is the word "break", which is to stop the program continuing into the next case statement.

```
switch ( expression )
{  case value1:
      statements
      break;
   case value2:
      statements
      break;
   etc etc etc
   default:
      statements
      break;
}
```

```
public static void main(String args[])
{
   int num = IBIO.inputInt("enter a number ");
   switch (num)
   {  case 1:
         IBIO.output("that number was 1");
         break;
      case 2:
         IBIO.output("that number was 2");
         break;
      default:
         IBIO.output("that number was neither 1 nor 2");
   }
}
```

Pr 4.1 Write a program that will let you enter two numbers. Then it will ask you to enter "1" for add, "2" for multiply, "3" for quit. This will be displayed on the screen like below:

```
Press:   [1]  for addition
         [2]  for multiplication
         [3]  for quit
```

```
public static void main(String args[])
{
   int   a = IBIO.inputInt("enter first number  ");
   int   b = IBIO.inputInt("enter second number ");
   int   num;
   do
   {
      //menu and switch in here
   } while (num != 3)
}
```

If none of these are entered then the program will announce an error and ask again for these possibilities. The program will keep doing this until quit is chosen. Only enter the two numbers once.

Pr 4.2 Write a program that will add up the sequence
1*7 + 2*2 - 3*5 + 4*7 + 5*2 - 6*5 + 7*7 + 8*2 - 9*5 + 10*7 + … 1000

Note that the there are three cases. First calculate the remainder when divided by 3. x%3. If the remainder is 0 then the number gets multiplied by -5, if the remainder is 1 then the number is multiplied by 7 and if the remainder is 2 then the number is multiplied by 2. (669004)

Labels

The break statement allows the program to exit the current situation (in this case the switch). It is possible to exit more layers. For example we might have a loop which test alternatives. The program below will continue to loop until the user has guessed the correct number (in this case 17)

```
mainLoop : do
{  int num = IBIO.inputInt("enter a number ");
   switch ( num )
   {  case 17:
         IBIO.output("correct answer");
         break mainLoop;
      default:
         IBIO.output("wrong");
         break;
   }
} while (true);
```

Pr 4.3 Write a program that allows the user to enter a number less than 1000. The program will search for two numbers that when squared and added together make the number that was input. The program has two loops. One loop going from 1 to *num* and the second loop also goes from 1 to *num* also. In the loop the program will square the numbers and then add them together to see if they are the same as the input number. If they are then use the break statement to break out of both loops. Your output will be the numbers or state that it is impossible.

Chapter Five: Thinking Abstractly (Decimals)

double

This is a new data type. Up to now we have only been dealing with whole numbers. Any variable we have used must be declared before we use it.

```
   int n;   //n is a variable to contain an integer number
```

Now we introduce a new data type – "double". This can be used to represent a decimal number.

```
public static void main(String args[])
{
   double a = inputDouble("enter first number  ");
   double b = inputDouble("enter second number ");
   double num = a / b;

   output("division gives " + num);
}
```

"output" will print the number. You can see that the accuracy is very high.

Sequence

The next program will add the numbers together 1 + 1/3 + 1/9 + 1/27 + 1/81 + … It will add 100 of these together. Notice important things about this program. "i" must be declared as an integer as it is used in a "for" statement. The variable "sum" and "term" have both been given descriptive names. In the program we use each term to calculate the next term.

```
public static void main(String args[])
{
   double   term = 1;
   double   sum  = 0;

   for (int i = 0; i < 100; i++)
   {  sum = sum + term;
      term = term / 3;
   }

   IBIO.output("total is " + sum);
}
```

Pr 5.1 Write a program that will add up the sequence
1/5 + 1/25 + 1/125 + 1/625 + … for 100 terms. (0.25)

In the example above we used one term to create the next term. For some sequences this is difficult and it is best to create each new term from scratch as in the next example.

Pr 5.2 Write a program that it adds up the sequence
1/1 + 1/4 + 1/9 + 1/16 + 1/25 + … for 100 terms (1.6348839001848923)

Alternating Sequence

If we have an alternating sequence then the problem is more complicated. An alternating sequence is one that the terms alternately add then subtract.

```
double   term = 1;
for (int i = 1 ; i < 10 ; i++)
{  IBIO.output(term);
   term = term + 3;
}
```

This creates a sequence of numbers. 1 , 4 , 7 , 10 , 13 ,…

```
double   term = 1;
int      sign = 1;
for (i = 1 ; i < 10 ; i++)
{  IBIO.output(sign * term);
   term = term + 3;
   sign = sign * -1;
}
```

This creates a sequence of numbers. 1 , -4 , 7 , -10 , 13

Pi

Pi can be calculated using the formula:

$$\frac{\pi}{4}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\frac{1}{11}+\ldots$$

Pr 5.3 Write a program that will add up the sequence discussed above to 10,000 terms. Output 4 times the answer to get pi. (3.1414926535900345)

$$\frac{\pi}{2}=1+\frac{1}{3}+\frac{1\times2}{3\times5}+\frac{1\times2\times3}{3\times5\times7}+\frac{1\times2\times3\times4}{3\times5\times7\times9}+\ldots$$

Pr 5.4 The sequence above is a much quicker way of calculating pi Write a program that will add up the sequence above to 100 terms. Output double the answer (3.1415926535897922)

Chapter Six: Thinking Procedurally (Strings & User Functions)

String

This is a new data type. A string is a sequence of characters. Any variable we have used must be declared before we use it.

```
String   text = “this is a string”;
```

One of the most important things that we can do with strings is to stick them together (concatenation). We do this with the + operator. We have already seen this in the output statements.

Functions

A function is a self contained program that can be called from another part of your program. It can be used over and over again. Functions can be given information to work with and they can give back answers. We want a program to print out the square according to the number that you enter.

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

  {  int num   = IBIO.inputInt("enter number of lines ");
     String aa = stars(num);

     for (int i = 0; i < num; i++)
        IBIO.output(aa);
  }

  static String stars(int n)
  {  String xx = "";
     for (int i = 0; i < n; i++)
        xx = xx + "*";
     return xx;
  }
```

The important thing in this piece of code is the function `stars()`:

`static`	This would be explained in depth later, but now you should always include it.
`String`	This is the return value. That is the routine creates a String.
`stars`	The name of the function.
`()`	The argument list, the data that the function needs as input.
`int`	The type of the data.
`n`	The name of the data.

The function makes a String of stars by first making an empty string and then gradually adding on more and more stars until we have enough.

In the function ‘i’ is used and in the main program ‘i’ is also used. These two do not conflict because the ‘i’ in the function only has existence during the time the program is running. This is called the scope of the variable.

Pr 6.1 Change the program so that it prints the square 10 spaces out. Do this by changing only the function and not changing the main routine.

Problem Solving

Below is another shape that your program is to print. This time there are a different number of stars to print in each line. As before you enter the number of lines in the triangle.

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

Do not erase your subroutine. You will use it again in this program. The first step is to write a program that will print the number of stars in each line.

```
  int   n = input("how many lines ");
  for (int i = 0; i < n; i++)
     IBIO.output(i + 1);
```

This program will not print out the stars but will print out the number of stars needed for each line. Now replace the output in this program with the star routine as before to get the shape as above.

Pr 6.2 Write a program that will print out the following shape. The width depends on the number you first input.

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

Pr 6.3 Write a program that will print out the following shape. The width depends on the number you first input. This is when you input 3 and there are 3 stars in each line, separated by spaces and there are 3 lines, a middle line and then more lines. To do this you can create another routine that makes a blank String.

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

Chapter Seven: Thinking Ahead

You will notice that it is hard to print numbers nicely on the screen. Nothing seems to line up. In the first work sheet you printed out the numbers and squares and cubes of the number from 1 to 20. It would have been nice if they had lined up like

```
1          1          1
2          4          8
3          9         27
4         16         81
```

In the second work sheet you went to a lot of effort using the **if** statement to print out extra space so that the numbers would line up.

Examine the program below which prints numbers lined up on the right.

```
public static void main(String args[])
{
   for (int i = 0; i < 20; i++)
   {  String xx = pad(i, 10)+pad(i*i, 10)+pad(i*i*i, 10);
      IBIO.output(xx);
   }
}

static String pad(int n, int tab)
{  String xx = "" + n;          // make a string of the number
   do
      xx = " " + xx;            // add spaces in front of the number
   while (xx.length() < tab);   // xx.length() is the length of xx
   return xx;
}
```

The method is to turn the number into a string and add spaces to the left side of the number.

Casting

To change a decimal into an integer we precede it with (int). This rounds down.

```
   int xx = 9.63 * 3.73;          // error
   int xx = (int)9.63 * 3.73;     // error only 9.63 changed
   int xx = (int)(9.63 * 3.73);   // is correct = 35
   int xx = (int)9.63 * (int)3.73;// also is correct = 27
```

Pr 7.1 Consider the program below. It computes powers of the number 3.732 and prints them out. Change the program so that all the answers are printed out to 2 decimal places only. Do this by multiplying by 100, change to integer and then dividing by 100;

```
   public static void main(String args[])
   {  double  xx = 1;
      for (int i = 0; i < 10; i++)
      {  xx = xx * 3.732;
         IBIO.output(xx);
      }
   }
```

Pr 7.2 Change the program above so that decimal places line up. To do this you must change the decimal answer xx into a string (*String yy = "" + xx;*). Then use the command *yy.indexOf('.')* to find the position the decimal place is in the string. eg if *String yy = "47.29";* then *yy.indexOf('.')* will be 2. Remember that counting starts from 0. Then add enough spaces at the start to line up the number.

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.

Chapter Eleven: Thinking Abstractly (Classes)

class

A class is a collection of data (attributes) and methods (actions, behaviours) that act on the data. We are going to create a class to model a fraction. To do this you must create a new text file and type in the first line:

```
public class Fraction
```

Save the file in your working folder as Fraction.java. Note that classes must begin with a capital letter.

```
import java.util.Scanner;//for later use

public class Fraction
{  int num;            //the fields/data for the fraction
   int den;

   Fraction(int a, int b) //constructor that creates a fraction
   {  num = a;
      den = b;
   }
}
```

The main method will look like

```
   public static void main(String args[])
   {  Fraction f = new Fraction(3, 4);
      IBIO.output(f.num + "/" + f.den);
   }
```

Run this program. Understand what is happening: a fraction is created (`f` is a new object instantiated from the `Fraction` class) and it is printed out. Now change the class fraction so that its attributes (data) are labelled private.

```
   private int   num;  //the attributes/fields for the fraction
   private int   den;
```

The program now refuses to run because the main program is not allowed to access the data inside a fraction. This is the usual way that programmers write classes. They do not want other programmers to directly access the data inside their class. This is called **information hiding**. So the class fraction must provide its own print routine.

```
   void  print()
   {  IBIO.output(num + "/" + den); }
```

Now the main program looks much simpler

```
      Fraction f = new Fraction(3, 4);   //create a fraction
      f.print();                         //print itself
```

A better way of doing this is to use a `toString()` method on a fraction. Replace the print routine in the Fraction with a `toString` routine. This is because we do not want a class to do printing (a class should have no direct input or output – that should be done by the controlling class) instead we have the class provide a string to represent what it looks like

```
   public String toString()
   {  String ss = num + "/" + den);
      return ss;
   }
```

Now the main program looks like this

```
      Fraction f = new Fraction(3, 4);   //create a fraction
      System.out.println(f);             //print itself
```

Note that we must use `System.out.println()` because the IBIO output function is not very clever and expects a string. You would have to explicitly use it like `IBIO.output( f.toString() );`

input

The next stage is to have an input method so that we can enter a fraction with the keyboard. In the main program this will look like the below.

```
public class FractionTest
{
   public static void main(String args[])
   {  Fraction f = new Fraction();
      f.enter();
      System.out.println(f);
   }
}
```

The enter method is defined back in the Fraction class:

```
   public void enter()
   {  String strFraction = IBIO.input("Enter fraction (a/b format): ");
      strFraction = strFraction.replace("/", " "); // Changes slash to space
      Scanner parse = new Scanner(strFraction);    // to parse fraction
      num = parse.nextInt();    // so we can extract the numerator &
      den = parse.nextInt();    // denominator; also this.num & this.den
      //this.simplify();     // <<< uncomment after you complete problem 11.2
   }                           // simplifies the fraction object we just entered
```

This routine acts upon a fraction object (that should have been previously created). Here it takes a string containing a fraction in x/y format, replaces the slash for a space, and extracts each number, converting them to integers. Then it sets the numerator and denominator to the given values and simplifies the fraction. The keyword **`this`** refers to the current object that is being used (`f` in our example above) and **`this.simplify()`** means to execute the simlpify method on the current object or “cancel this fraction that called the enter method/being entered”.

add

Pr 11.1 Write a routine that adds together two fractions. This routine is declared (non-static) in the Fraction class, as we want each fraction to be able to input its own data. The main method will look like the code below.

```
Fraction a = new Fraction()
a.enter();          //get the first fraction
Fraction b = new Fraction();
b.enter();          //get the second one
Fraction c = new Fraction();
c = a.add(b);       //add b to a and put result into c
System.out.println(c);
```

simplify

When we create (instantiate and initialise) or enter a fraction, and after any calculation, we would like the fraction to be simplified as much as possible. The way to do this is to calculate the greatest common divisor (GCD) of the numerator and denominator and divide both the numbers by this.

```
private static int gcd();
private void simplify();
```

Pr 11.2 Write a routine that simplifies the fraction in the constructor and the enter methods.

decimal

Next we want to change our fractions into decimals. This is not a static routine because it acts on a fraction that is already created.

```
Fraction a = new Fraction();
a.enter();    //get the fraction
a.print();    //print it
double x = a.toDecimal();
IBIO.output(a + "change to decimal = " + x);
```

Pr 11.3 Design a routine that changes the given fraction to decimal.

Test your last routine by entering 0 as the denominator. Note that we can never have 0 as the denominator of a fraction. We have to add error checking features into our code to avoid this. This is done by using `throw-catch` statements as we will see next.

Exceptions (Thinking Ahead)

There are two ways that a subroutine can return. The first is the normal way and the other is by an exception. When the return is by exception, the return value does not matter and is unnecessary.

An exception is an unusual situation that has been caused in the running of the program. Common causes are using a file and the filename cannot be found, input cannot be sent or received from a file. To understand exceptions we will write our own.

```
   public static void main (String args[])
   {  while (true)
      {  try
         {  int num = inputNumber();
            IBIO.output("that was correct");
         } catch (Exception e)
         {  IBIO.output(e.getMessage() );  }
      }
   }
//=====================================================
   public static int inputNumber() throws Exception
   {  int num = IBIO.inputInt("enter a number between 40 and 50 ");
      if (num < 40)
         throw new Exception("number too small");
      else if (num > 50)
         throw new Exception("number too big");
      return num;
   }
```

This is a small program that expects the user to enter a number between 40 and 50. The code that can cause the exception is in the routine inputNumber if there is an error in the input then it uses the word `throw`. The routine itself is declared as throws Exception.

The calling routine has `try` and `catch` statements. The statements after the try are ones that can cause the error even subroutines of this. The catch part just reports back on the error. When the routine inputNumber encounters an error, it throws and the previous routine catches it.

Any time that you are using any built in classes that contain exceptions then we must be prepared to catch them. See an example in the `IBIO.java` file for `inputInt`.

The next example shows that the exception can propagate through several levels until it is caught. It will be a compile error if the exception is not caught by any calling routines:

```
   public static void main(String args[])
   {
      while (true)
      {  try
         {  int num = getThreeNumbers();
            IBIO.output("that was correct");
         } catch (Exception e)
         {  IBIO.output( e.getMessage() ); }
      }
   }
//=====================================================
   public static int getThreeNumbers() throws Exception
   {  int num1 = IBIO.inputNumber("first number  ");
      int num2 = IBIO.inputNumber("second number ");
      int num3 = IBIO.inputNumber("third number  ");
      return num1 + num2 + num3;
   }
//=====================================================
   public static int inputNumber(String ss) throws Exception
   {  int num = IBIO.inputInt(ss);
      if (num < 40)
         throw new Exception("number too small");
      else if (num> 50)
         throw new Exception("number too big");
      return num;
   }
```

Pr 11.4 Modify your Fraction class so it that will only you to enter a fraction with a denominator greater than zero.

Pr 11.5 Write a program that will all you to enter only a four digit number that has all digits different. Input the number as a string.

Chapter Twelve: Thinking Ahead (Text Files)

String

The program below creates an array of strings. It allows you to enter words into this array and when you have finished just press return without entering anything and the names will be printed again.

```
   public static void main(String args[])
   {  int count = 0;
      String[] names = new String[20];

      do
      {  names[count] = IBIO.input("enter some words: ");
         if (names[count].length() == 0)
         {  break;
         }
         count++;
      } while (true && count < 20);
   }
```

Pr 12.1 Change the last program so that the words are printed out.

File Output

We want to save this information into a file on the disc so that it can be loaded at a later time.

```
import java.io.*;      //allow the program to use file routines
public class FileIO
{
   public static void main(String args[]) throws IOException
   {
      // >>> make the strings here

      File  ff = new File("temp.txt");         //create the file
      FileWriter fw = new FileWriter(ff);      //set it for writing/saving data
      PrintWriter save = new PrintWriter(fw);
      for (int i = 0; i < count; i++)
         save.println(names[i]);               //save to the file
      save.close();
   }
}
```

There are several important things in the above code:

1) The import line at the beginning of the program allows your program access to all the file input / output routines.
2) On the main line there is the addition of throws `IOException`. This is because the input routines could cause an error. Eg the file not being present on the disc, getting a disc error because the disc is bad.
3) The three lines with `File`, `FileWriter`, `PrintWriter` you do not need to understand except that they must be present whenever you write to a file. The `PrintWriter` is called *`save`*. This name can be anything you like but must be used whenever you output stuff to the file.
4) At the end of dealing with the file, it must be closed.

Run the example above. Use the previous program to input the names into the string array. Press return and instead of printing out the names the names will be printed into the file. The file may be in your project folder.

File Input

Now to load in the previous file we use the code below. It also has three lines to create the object that can read in from a file. When this routine has finished count will be the number of strings in the array.

```
File           ff = new File("temp.txt");
FileReader     fr = new FileReader(ff);
BufferedReader load = new BufferedReader(fr);

count = 0;
while ( load.ready() )
{  names[count] = load.readLine();
   count++;
}
load.close();
```

Pr 12.2 Write a new program that will read in the previous file into an array. Then print the names to the screen. Using the code above just add extra lines to print the names to the screen.

Pr 12.3 Write a new program that will read in the previous file into an array. Then allow you to add one extra word (it is added at the end of the array) and then write it all back to the same file. The extra word is added on the end of the array. The schema for that is below.

```
import java.io.*;
public class FileIO
{
   public static void main(String args[]) throws IOException
   {
      // read in the strings (count is now fixed)
      // print them to the screen
      // add one more string to the end of the file (increase count)
      // write all the strings to the file
   }
}
```

Pr 12.4 Open the file that you have created above. Write a program to delete a word from the file. At the beginning of the program enter one word and the program deletes that word from the file. If that word is not in the list, then it is added to the file.

```
   public static void main(String args[]) throws IOException
   {
      // read in the strings
      // print them to the screen
      // enter one word - remember in a string - not the array
      // write all the strings except that one to the file
      // if that string is not in the list write that also.
   }
```

Note that given `String aa, bb` then `aa.equals(bb)` will return true if strings `aa` and `bb` are exactly the same.

Chapter Thirteen: Thinking Ahead (Number Files Input)

Input from a File

Study the example below carefully. In our class resources there are three files for you to practice on. They are called "Num10", "Num100" and "Num1000". Put these files in your working folder. These are files of random numbers and the only difference between them is that one has 10 numbers, the second has 100 numbers and the last has 1000 numbers. For the examples below:- use the easy example of 10 numbers and then when your program is working you can change it to work with the 100 or 1000 number file.

```
public class Simple extends IBIO
{
   public static void main(String args[]) throws IOException
   {
      File       f  = new File("Num100");     //same as before
      FileReader  fr = new FileReader(f);
      BufferedReader load = new BufferedReader(fr);

      while (load.ready())
      {  String ss = load.readLine();   // read one line of data as a string
         int num = Integer.parseInt(ss);//change to a number (integer)
         IBIO.out(num + " ");           //print out the number
      }
      load.close();
      IBIO.output("\nEnd of file.");
   }
}
```

Pr 13.1 Change the program above so that it counts the numbers in the file.

Pr 13.2 Write a program that allows you to enter in a number and the program will find how many times that number occurs in the given file. (Test: 37 occurs 12 times in the 1000 file).

Pr 13.3 Write a program that will output the average of the numbers in the number file. (Test: the 1000 number file has an average of 49.418).

Pr 13.4 Make a count of the number in the ranges 0-9, 10-19, 20-29, …, 90-99. Your print out should look like the below. Notice how all the numbers are lined up. To do this program you have to have an array that will keep count of the numbers in each group. The answers below are correct for the 1000 file.

```
 0- 9       88
10-19      106
20-29      104
…
90-99       90
```

Chapter Fourteen: Thinking Ahead (Number File Output)

Output to a File

The program below will let you type some numbers and then save them to a file. Note in doing these example writing to a file can destroy your original so only do this on a copy.

```
public static void main(String args[]) throws IOException
{
   File       f    = new File("temp");
   FileWriter fw   = new FileWriter(f);
   PrintWriter save = new PrintWriter(fw);

   do
   {  int num = IBIO.inputInt("enter a number (99 to finish) ");
      if (num == 99)
         break;
      String ss = "" + num;  //change to string
      save.println(ss);      //save to the file
   } while (true);
   save.close();
}
```

Check by loading the file.

Pr 14.1 Using the example of num10. Read in the data, add 10 to each number, write the numbers out to a different file.

Pr 14.2 Find the file “sort”. This is a list of numbers that are in order from smallest to biggest. Allow the user to type in a number and then read in the file and write the file out to another file inserting the new number in the correct place. You do the reading and the writing at the same time. Take care if the number entered is smaller than the first or larger than the last.

Pr 14.3 Change the last program so that when finished. You read the new file in and write it out to the original array so that it looks like the number you inserted was into the original file.

***** END *****