CHAPTER 6 : PROBLEM SOLVING

A computer program is a solution to a problem in a computer language.

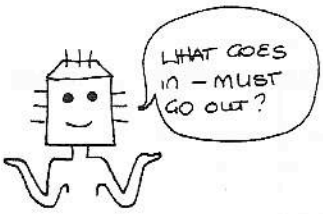
In this Chapter...

You will learn about:	You will learn how to:
IPO model to problem solving, methods from the Math class, casting / converting types, random numbers, how a program is run in Java, bytecode, errors - syntax / compilation, run time, logical errors including type mismatch and input errors, trace tables as a debugging tool, user friendly programs, readable programs, planning longer solutions	write programs using IPO, use square root, power, absolute value and round methods, convert between data types (cast), generate random numbers, identify and correct errors (syntax/run time/logical), use a tracetable to find logical errors in a program, create user friendly programs, create readable programs, plan solutions to longer programs

I **P** **O**	Most programs involve	**I**nput (data being input when the program is Run) **P**rocessing (the input data being worked on to produce results) **O**utput (the results being displayed so that they can be seen)

At this stage you are only coding programs to solve simple problems, so the Input, Processing and Output phases take place in 3 separate sequential steps. Later, in the development of solutions to more complex problems, you will see that these 3 phases can take place many times during a program, so it is not so easy to use the same format for solutions as you will use now.

6.1 A template for developing programs



Before developing a solution to a problem and coding this as a Java program, it is useful to identify exactly what Input, Processing and Output must take place, and also to decide what variables are needed, giving them identifiers and data types.

We will use the following **IPO** and **Variable** tables to do this, as they help in the coding of solutions to problems:

Input	Processing	Output	Variable	
			Identifier	**Data Type**

The steps in problem solving are outlined in the following "template".

1. Read the problem specification carefully, making sure that you understand exactly what is required.
2. Complete the IPO table, showing what Input, Processing and Output will be required when the program is Run.
3. Complete the table of variables, showing the identifiers and the data types of each variable that will be used when you code the program.
4. Start a new HSA Boilerplate Application, using the class name given.
5. Add your name and today's date as a comment at the top of the program.

6. Type in code to declare all the variables listed in the table of variables.
7. Type in code for the Input, Processing and Output sections, using the IPO table as a guide.
8. Leave a blank line (or put a comment) on the editor screen between each section of code, to clearly indicate what is taking place.
9. Save the program, using the same name as the class name.
10. Run the program, testing it with different input to make sure that it is correct.

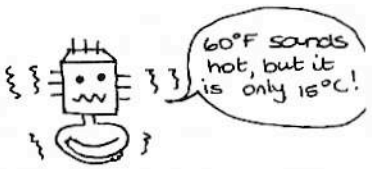
Two worked examples using this template follow.

You should apply this template to all problem solutions that you code after this.

Example 1: Converting temperatures

A temperature in Fahrenheit (F) is input and converted to its equivalent temperature in Centigrade (C), using the formula: `C = (F - 32)*5/9`. Both the Fahrenheit and the Centigrade temperatures must be displayed, to one decimal place. Use a suitable heading.

▶ Apply the steps in the template to solve this problem:



1. Read the problem specification carefully, making sure that you understand all parts of it.

2. Fill in the IPO table:

Input	Processing	Output
A temperature in Fahrenheit	Work out temp in Centigrade using `C = (F - 32)*5/9`	• Heading • Temps in F and in C (both to one decimal place)

3. Complete the table of variables:

Identifier	**Data Type**
temp_F	double
temp_C	double

double is a better choice of type than **int** for `temp_F` as it can cope with the possibility of the user entering real numbers; double is also a better choice for `temp_C` as it is possible that the calculation will produce a real result.

4. Start a new HSA Bolierplate Application, using the class name **Temperatures**.

5. Add your name and today's date as a comment at the top of the program.

6. Type in code to declare all the variables listed in the table of variables.

7. Type in code for the Input, Processing and Output sections. Use a meaningful prompt to explain clearly what input is required; code the processing statement; display the output with suitable text messages to show clearly what is displayed.

8. Leave a blank line (or put a comment) on the editor screen between each section of code, to clearly indicate what is taking place.

9. Save the program, using the same name as the class name.

10. Run the program, testing it with different input to make sure that it is correct.

☐ Check that your program looks similar to the one below:

```
//Variable declaration
double temp_F, temp_C;

// Input
c.println ("Type in a temperature in degrees Fahrenheit and <Enter>");
temp_F = c.readDouble ();

//Processing
temp_C = (temp_F - 32) * 5 / 9;

//Output
c.println ("TEMPERATURE CONVERSION");
c.print ("The Fahrenheit temperature ");
c.print (temp_F, 0, 1);
c.print (" is equivalent to ");
c.print (temp_C, 0, 1);
c.println (" in degrees Centigrade");
```

☐ Run your program, and input each of the following Fahrenheit temperatures. Write down the equivalent Centigrade temperatures produced by your program:

Fahrenheit temperature	Centigrade temperature
212	
39.4	
100	
132.8	

Example 2: Working out the cost of breakfasts

1. Read the following problem specification carefully, making sure that you understand exactly what is required:

A certain hotel offers a choice of 2 set menus for breakfast:

- Continental Breakfast, which costs $15.50 and
- Full English Breakfast, which costs $21.99

A group of housewives in a Book Club meet for breakfast at this hotel.

You are required to determine how many in the group want the Continental Breakfast, and how many want the Full English Breakfast. The computer must then calculate the costs for the two different types, and the total amount due. An invoice must be displayed, with the following layout:

```
INVOICE FOR BREAKFAST

                                                        AMOUNT
Number of Continental Breakfasts      : . . . at $15.50 each = $
Number of Full English Breakfasts     : . . . at $21.99 each = $

                                          TOTAL AMOUNT DUE = $
```

- Show the number and cost of each of the 2 types of breakfasts wanted, as well as the total cost. Display all amounts to 2 decimal places.

2. Complete the IPO table below, showing what Input, Processing and Output will be required when the program is Run:

Input	Processing	Output
• Number of continental breakfasts (num_c) • Number of	Multiply num_c by 15.50 to get cost_c. Multiply	Display headings. Display invoice details using num_c and . . . . and cost_c and . . . . and totalCost. Line up costs, and use 2 decimal places for all costs.
	Add cost_c and . . . . . to get totalCost	

3. Complete the table of variables below, showing the identifiers and the data types of each variable that will be used when you code the program:

Identifier	Data Type
num_c	int
. . . . .	int
cost_c	double
. . . . .	double
totalCost	double

4. Start a new HSA Application Boilerplate called **Breakfasts**.

5. Add your name and today's date as a comment at the top of the program.

6. Type in code to declare all the variables listed in the table of variables.

- Leave a blank line on the editor screen to separate the variable declarations from the next section of code.

INPUT:

- Type in code to prompt and read in to the variables **num_c** and . . . . . the quantity of each type of breakfast wanted.
- Leave a blank line on the editor screen to separate the Input in the previous section of code from the Processing in the next section.

PROCESSING:

- Use the 2 quantities that were input into the variables **num_c** and . . . . to work out the cost of each type of breakfast, storing the costs in the variables **cost_c** and . . . . .
- Assign the sum of the 2 costs to the variable **totalCost**.
- Leave a line blank on the editor screen to separate the Processing in the previous section from the Output in the next section of code.

OUTPUT:

- ▶ Type in code to display the headings "INVOICE FOR BREAKFAST" at the left of the screen, and then on the next line "AMOUNT" starting in column 60.
- ▶ Type in code to display "Number of Continental Breakfasts: " followed by the value stored in **num_c**, followed by " at $15.50 each = $ ", then the cost of these, stored in **cost_c** (to 2 decimal places).
- ▶ Do the same for the English breakfasts.
- ▶ Display a blank line, then display "TOTAL AMOUNT DUE = $ " followed by the value stored in the variable **totalCost**.

9. Save onto disk with the same name as the class name.

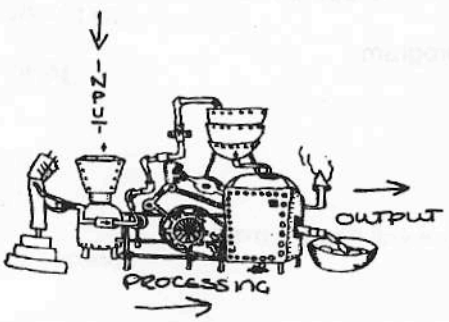
10. Run the program, typing in 7 for Continental and 4 for Full English Breakfasts when prompted.

▶ Copy down the output produced by <u>your</u> program, when it works without any errors:

EXERCISE 6A

For each of the following problems,

- Apply the steps in the problem solving template to develop a Java program solution.
- Use a separate piece of paper to draw up the IPO and Variable tables.
- Use the name next to the number of the problem as the class name.
- Run your program when it is complete, using the test data given as the input
- Copy down the output produced by <u>your</u> program in each case.
- Print each program when it works correctly, if you can.
- File your IPO and Variable tables with your program's printout.



1 **NameAge**

Ask for a person's name to be input. Store this in a String type variable. Use another prompt to get the person's year of birth (an integer). Get the computer to work out the person's age by subtracting their year of birth from the current year. Leave a blank line, then display a meaningful message with the person's name and age this year.

For example, if the name John and the year 1986 is input, the output should be similar to:

John, you will be 16 years old this year.

Output produced by your program:

2 **Chocs**

All chocolate bars today cost $0.99. Ask for the name of any chocolate bar to be entered. Also ask for the quantity of bars wanted. Leave a blank line, then use `"\t"` to display the name, quantity and total cost, with headings in capitals, as follows:

CHOC BAR	QUANTITY	COST	
KitKat	3	$2.97	(to 2 decimal places)

▶ Run your program with the name KitKat and the quantity 3. Check that the output is correct.

▶ Run your program again using the name Snickers and the quantity 2.

Output produced by your program:

3 **Supermarket**

Ask for the name of any food to be entered as well as the price of this food. Then ask for the name of another food to be entered as well as its price. Leave a blank line, then display a receipt, showing the shop's name, the names of the two foods on the left and their prices next to them. Also show the total of the two prices under the individual prices. Line up the prices using field widths so that they appear in line with each other. For example:

Sally's Supermarket

Tomatoes	7.99
Chips	4.75

TOTAL	12.74

▶ Run your program again using Apples as the first food and their price: 2.99 and Nectarines as the second food and their price: 2.50.

Output produced by your program:

4 **RoundedMarks**

Ask for the name and mark of a pupil to be entered from the keyboard. The mark is for a test out of 43. Convert the mark to a rounded percentage (ie. marks out of 100). Display the pupils' name, their mark out of 43 and the rounded percentage (to a whole number). Use columns with headings as shown on the next page.

NAME MARK OUT OF 43 ROUNDED %

▶ Run your program with the data: Jane 35, Sophie 28, Ismail 40

Output produced by your program:

5 **PlayCosts**

A school play is being held on 2 nights. Tickets for adults are $15 and for pupils $10. Read in the number of adults and the number of pupils who attended on each night and determine the total proceeds made. Also determine the profit made, if the total costs of the production were $9000.

▶ Run your program, entering the following numbers:

Night 1: Adults = 235 Pupils = 309
Night 2: Adults = 278 Pupils = 415

▶ Write down the results produced by your program, using the above numbers:

Total number of adults (both nights): . . . . . Cost for the adults: . . . . . .
Total number of pupils (both nights): . . . . . Cost for the pupils:. . . . . .
Total proceeds : . . . . . .
Profit made : . . . . . .

6 **Swimmers**

a. Ask for the names of 2 Olympic women swimmers to be input, as well as their times (in seconds) in the 100m breast stroke final. Leave a blank line after the input, then display the results in lined-up columns, with headings.

▶ Use the following test data when you Run your program:

100M BREASTSTROKE FINAL

(⇐ *Leave a blank line here)*

NAME TIME
HEYNS 67.55
POEWE 67.85

b. Also work out how much faster the first time is than the second and display this underneath the 2 times. Your output should look like this:

100M BREASTSTROKE FINAL

NAME TIME
HEYNS 67.55
POEWE 67.85

(⇐ Leave a blank line here)

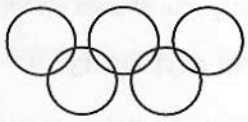
Difference in times 0.30 seconds

7 **Rings**

Code a new program which uses the HSA Console output window to display 5 interlocking Olympic rings (all of the same size, and the same distance apart), as shown below, in the following colors:

Top row (left to right) blue, black, red;

Bottom row (left to right) yellow, green.



8 **MarkAverage**

Enter a pupil's name and their marks for 2 tests (both integers). The first is a mark out of 25 and the second a mark out of 55. Add together the 2 marks and work out the percentage using the formula `perc = total*100/(25+55)`. (Use the identifiers `perc` and `total` in your program.)

If you first divide by (25+55) and then multiply by 100, you will get the wrong answer. Why?

.....

If you leave out the brackets around 25+55 the answer will be wrong. Why? .....

.....

▶ Leave a blank line, then display the pupil's name, the 2 marks and the final percentage.

Eg. Jane got 18 out of 25 and 43 out of 55, so her final percentage is 76%

9 **FoodStall**

A food stall at a flea market is selling cups of Coke at $3.00 each and packets of chips at $2.50 each.

a. Code a program which will work out the total amount each customer has to pay, depending on how many of each item is bought.

For example, if a customer wants 3 Cokes and 2 packets of chips, he will have to pay $14.00. (Format the amount to 2 decimal places.)

Save and Run your program with each of the following test data, and write down the total amount each customer has to pay:

Cokes	Packets of chips	Amount to be paid
2	2	.....
5	0	.....
1	1	.....

b. Extend your program so that you can input the amount of money the customer gives you, and your program will work out and display the change that you have to give.

For example, if the total cost is $5.50 and the customer gives you $20.00, you will have to give $14.50 in change.

10 **StarInBox**

Using graphics, a star can drawn using the fillStar method. This method, like the fillRect method, uses 4 integer parameters.

Draw a large red star which appears approximately in the centre of the output screen. Draw a blue border around the star, which must completely enclose the star, but not be larger than the screen's dimensions. The star must be approximately in the centre of the blue border.

6.2 Using methods from the Math class

Java has many built-in methods that make our life easier as we do not have to tell the computer the details of how to work things out (eg. Square roots).

- Start a new class called **UsingMathMethods**.
- Type in code which asks for any real number to be input. Store it in a variable called **num**.
- Add the following code which uses several methods from the Math class:

```
c.println( Math.round (num) );
c.println( Math.sqrt (num) );
c.println( Math.abs (num) );
c.println( Math.pow (num,2) );
```

- Save and Run the program and write down the output that is produced when each of the following values are input:

num = 16.75	num = 4.0	num = -23.45
Math.round(num)=	Math.round(num)=	Math.round(num)= . . . .
Math.sqrt(num)=	Math.sqrt(num)=	Math.sqrt(num)= . . . .
Math.abs(num)=	Math.abs(num)=	Math.abs(num)= . . . .
Math.pow(num,2)=	Math.pow(num,2)=	Math.pow(num,2)= . . . .
	Math.pow(num,num)=	Math.pow(num,3)= . . . .

Change these too

- Why does the last value (-23.45) give strange output for `Math.sqrt(num)`? . . . . . (`Nan` stands for `Not a number`!)
- Complete the following table which summarises what each of the methods does.

Method from Math class	WHAT IT DOES
round	
sqrt	Works out the square root of a number, IF the number is positive.
abs	Always gives a positive answer ie. if the number is negative, the abs method makes the number positive; if the number is positive, abs leaves the number positive.
pow	

▶ The methods `round`, `sqrt` and `abs` only have one parameter (which is the value in brackets). Why does the `pow` method have 2 parameters? Explain the purpose of each of these 2 parameters (eg the `num` and the `2` in `Math.pow(num,2)`)

. . . . . .

. . . . . .

EXERCISE 6B

1 Write down what answer the computer will determine for each of the following. Show working where applicable.

a. Math.pow(3.5, 2) . . . . . .

b. Math.sqrt(16) . . . . . .

c. Math.abs(-5) . . . . . .

d. Math.round(0.75) . . . . . .

e. Math.round(-3.8) . . . . . .

f. Math.pow(Math.round(4.75),2) . . . . . .

g. Math.pow(65 % 9, 2) . . . . . .

h. Math.round (Math.abs (14.75 - 20))

. . . . . .

. . . . . .

i. Math.round(Math.sqrt(150)) . . . . . .

j. 43 % 2 + 18 * Math.sqrt(36/4)

. . . . . .

. . . . . .

k. Math.sqrt(6 + 76 % 10 / 2)

. . . . . .

. . . . . .

l. Math.sqrt(7 + 18 / 7 - 23 % 10)

. . . . . .

. . . . . .

. . . . . .

. . . . . .

m. Math.abs (7.88 - Math.pow(12.5,2))

. . . . . .

. . . . . .

. . . . . .

n. Math.round(14.7 - 0.95) + 2.5 * Math.abs (-6-4)

. . . . . .

. . . . . .

. . . . . .

2 Work out the answer the computer will get for each of the following:

- Show all steps
- Remember the order of operations

a. Math.sqrt(16 * 4) + Math.pow(7, 3)

b. Math.round(86.75) - Math.abs(-86.75)

c. 75 / 14 + 75 % 14 + Math.round(75 / 14)

d. 18 / 5 / 3 - 67 % Math.round(Math.abs (-5.77))

e. Math.sqrt(Math.round (412.6 / 14.8)) + Math.pow(Math.round (44.67 / 2.5), 2)

▶ Type the above expressions in output statements in a new program and run the program. Compare the output to the answers you wrote down.

EXERCISE 6C

▶ Add code to programs for each of the following. Use the class name next to each question number. Save them, then Run them, and print the programs when they are correct.

1 **NumResults1**

a. Ask for any real number and any integer to be input. Display the real number, and its square root. Also determine and display the real number raised to the integer power. Use suitable headings, displaying the results across the screen.

Eg. If the real number 13.5 and the integer 2 are input, display:

REAL NUMBER SQUARE ROOT RAISED TO POWER **2** ← Display the power here.
13.5 3.6742 182.25

▶ Write down the results produced by your program when the real number 7.8985 and the integer 3 are input:

. . . . . .

. . . . . .

b. Change your program so that instead of inputting an integer power, a real number power can be input.

▶ Write down the results produced by your program when the following are input:

- the number 16 and the real number power 0.5 (1/2)

. . . . . .

- the number 8 and the real number power 0.33 (1/3)

. . . . . .

Does your program give a different answer if 0.333333 is input instead of 0.33 for the power?

2 **NumResults2**
Input any positive integer. Display the number, its square, cube and its square root (to 4 decimal places). Make sure that you state clearly what the numbers in your output mean.

3 **ColdDrinks**

a. You are selling cold drinks at $1.89 each. Ask a customer how many he/she wants to buy, input this amount, then display the number wanted, as well as the amount to be paid.

b. Extend your program so that you enter the amount (in whole dollars) given by the customer, and the computer determines and displays the change.

Line up the output well, so that all decimal points are one below the other.

6.3 Converting between variable types

You should already know that Java converts all items to Strings when the *print* and *println* methods are used to display output.

Java allows other data type conversions, such as converting from a **char** type to an **integer** or a **real** number, and converting between integer numbers and real numbers.

Refer to the table of data types in the Summary at the end of Chapter 4 to remind yourself of what can be stored in each data type.

There are two data type conversions:

- **A *widening* conversion**, which occurs when a value of one type is converted to a data type that has a larger range of values. Java performs widening conversions automatically, such as when an integer value is assigned to a variable of the double data type.
  Eg.
```
double realNum;
realNum = 7;
```

- **A *narrowing* conversion**, which occurs when a value of one type is converted to a data type that has a smaller range of values (such as double to integer). Narrowing conversions are not always safe, because data can be lost if the value is outside the possible range of values in the new data type.

  Java can be forced to perform a narrowing conversion using a construct know as a **typecast**, or **cast** for short, by placing the name of the desired data type in round brackets before the value to be converted.
  Eg.
```
int num;
num = (int) 13.456;
```

INVESTIGATION

A. First we will look at **converting real numbers to integers**.

☐ Start a new HSA Application called **TypeCast**.

☐ Add code to declare one variable called **realNum** of data type double, and one integer variable called **intNum**.

☐ Assign the real number value 654.875 to **realNum**.

- Then add the statement: `intNum = (int) realNum;`

`realNum` is being cast to the integer `intNum` in an assignment statement

- Add 2 lines of output which should display, one line below the other, the values of the variables, together with suitable messages:

```
c.println ("The real number is " + realNum);
c.println ("This number, cast to an integer, is " + intNum);
```

- Save and Run your program.
- Write down the output that is produced: .....
  .....
- Change the assignment statement, so that any real number can be typed in and stored in the variable **realNum**.
- Run the program, entering the different real values given below. Write down the integer values into which each real number was cast:

Real Numbers:	Typecast integer values:
0.375	.....
76354.54	.....
-532.976	.....
15.0	.....
777	.....

A narrowing conversion **truncates** (chops off) the decimals of a real number when it converts it to an integer. Eg. 7.9 will be converted to a 7, and 7.1 will also be converted to a 7. (This explains why there is no trunc method as in some other programming languages.) Conversions should be avoided if accuracy is to be preserved.

B. We have seen that a real number can be converted to an integer with some loss in accuracy. Now we will investigate **converting characters to integers**:

- Start a new class called **TypeCastChar** and type in the following code:

A **character** is any symbol that can be displayed on the screen in a single space. For example 'p', '9', '*' and '?' are all characters.

```
int num;
char letter;
c.println ("Type in an integer value <Enter> ");
num = c.readInt ();
letter = (char) num;
c.println (letter);
```

- Enter the following integer values, and write down the corresponding characters.

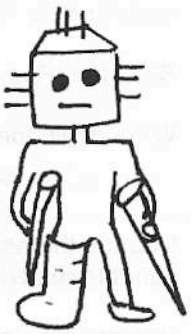
Integer Value	Character Output
63	.....
65	.....
90	.....
97	.....
122	.....
47	.....

Java converts the integer values to characters using ASCII code values. Each character found on the keyboard has a unique associated integer code value. The first 127 ASCII code values (0 to 126) can be cast into characters. Look at the Appendix for an ASCII table showing these 127 code numbers and their corresponding characters.

- Change the last three lines of the program **TypeCastChar** so that a character is entered, converted to the integer **num**, and then **num** is displayed.

- Check the above values to see if the conversion works in reverse. Use the table below to investigate other characters.

Character Entered	Integer Output
.....	.....
.....	.....
.....	.....
.....	.....
.....	.....
.....	.....
.....	.....
.....	.....
.....	.....
.....	.....



6.4 Random numbers

The computer can generate random numbers in a specific range, which can be useful in some problems, such as in simulating throws of a dice, in which a number between 1 and 6 is wanted.

The built-in **Math** class has a *random()* method which produces a real number between 0 and 1 (but not including 1). Multiplying this by an integer (N) produces a random real number in the range $0 \leq$ random number $<$ N. This is converted to an integer using the cast `(int)`.

For example, to produce one random integer between 0 and 9 (inclusive), use the code:

```
int randomNumber = (int) ( Math.random() * 10 );
```

The variable `randomNumber` will contain one integer from the set { 0;1;2;3;4;5;6;7;8;9 }.

How does this work?

`Math.random()`	produces a number between 0 and 1. Lets say it generates 0.453
`Math.random() * 10`	multiplies 0.453 by 10 = 4.53
`randomNumber = (int) (Math.random() * 10)`	casts (converts) 4.53 to an integer by dropping the decimals
`randomNumber`	will store 4 (in this example)

To produce one random integer between 1 and 10 (inclusive), use the code:

```
int randNum = (int)( Math.random() * 10 ) + 1;
```

The variable `randNum` will contain one integer from the set { 1;2;3;4;5;6;7;8;9;10 }.

In general,

to produce one random integer between 0 and N (inclusive), use the code:

```
int randomNumber = (int) ( Math.random() * (N+1) );
```

The variable `randomNumber` will contain one integer from the set { 0;1;2;3;. . . ; N }.

To produce one random integer between 1 and N (inclusive), use the code:

```
int randNum = (int)( Math.random() * N ) + 1;
```

The variable `randNum` will contain one integer from the set { 1;2;3; . . . ; N }.

KWIKWIZ

1. Write down the Java statement needed to produce one random integer in each of the following ranges:

   Between 0 and 99 (inclusive) .....

   Between 1 and 100 (inclusive) .....

   Between 0 and 100 (inclusive) .....

   { 0;1;2;3;4;5;6;7;8;9;10;11;12;13;14;15 } .....

   { 1;2;3;4;5;6;7;8;9;10;11;12;13;14;15;16 } .....

   { 0;1;2;3;4;5;6;7;8;9;10;11;12;13;14;15;16 } .....

   Between 1 and 6 (inclusive) .....

2. Write down the range from which a random number will be selected when each of the following Java statements is executed:

   ```
   int randomNumber = (int) ( Math.random() * 9 ); .....
   int randomNumber = (int) ( Math.random() * 9 ) + 1; .....
   int randomNumber = (int) ( Math.random() * 50 ); .....
   int randomNumber = (int) ( Math.random() * 50 ) + 1; .....
   int randomNumber = (int) ( Math.random() * 2 ); .....
   ```

3. Explain why a zero is never produced when the statement

   ```
   int randomNumber = (int) ( Math.random() * 20 ) + 1;
   ```

   is carried out .....

4. What is the purpose of the `(int)` when generating a random number in a given range?

   .....

EXERCISE 6D

1 Code a new HSA Application called **DiceThrow** which simulates a throw of a dice by producing one random integer in the range {1;2;3;4;5;6}.

▶ Save your program, then run it several times, checking that you do only get values in the specified range.

▶ Run the program another 10 times, and write down the random number produced each time:

....

2 a. Write code in a program called **RandomShapes** that will generate four integers which should be used as parameters to display a blue rectangle anywhere in your output screen. The rectangle can be any valid size. You will need to work out the approximate size of the graphics screen (see Chapter 3).

b. Extend the program in 2a. to draw a yellow circle in a random position and of a random size.

c. Adapt the program so that the circle is always on top of the rectangle.

d. Now add more variables to randomly generate a new color for the rectangle and the circle.

6.5 What happens when Run is clicked?

Java programs consist of classes made up of Java statements, which are saved on disk with the **".java"** extension. The Java statements (which follow the syntax (rules) of the Java language) are fairly close to English, and are not understood by the computer's hardware.

When the **Run** command is given, the *Ready* environment attempts to **compile** (convert) the Java statements in the active window.

The following process occurs:

1 These Java statements are first checked for errors, and

2 if there are no errors, all the statements in the class are compiled (converted) into **bytecode** (which is a binary code understood by the computer's hardware), and stored in a file with the extension **".class",** using the same name as the **".java"** program.

3 This bytecode program can be run on any platform that has a Java **interpreter**. The interpreter has to convert the bytecode into the native machine code of the operating system being used.

One bytecode interpreter, the IBM Java Virtual Machine, is built into *Ready*. However, the bytecode files produced by *Ready* can be executed using any Java interpreter.

Java is a portable language. It is one of the few languages that is both compiled and interpreted.

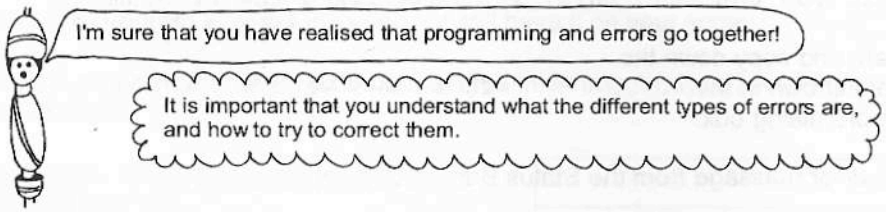
▶ Click Open on the File menu in Ready, and change the File Type (near the bottom of the dialog box) to "All Files". After almost every **.java** file, there should be a **.class** file with the same file name.

SUMMARY

When **Run** is clicked, the computer

- Compiles the Java code, which means it
  - Checks the Java code for errors; and, if error-free,
  - Converts the Java code into binary bytecode
- Converts the bytecode into the machine code of the computer, using an interpreter
- Carries out each step in the machine code program

6.6 Errors



6.6.1 Syntax or compilation errors

Syntax or **compilation** errors are errors that occur when you Run a program and the computer tries to compile (convert) it into bytecode.

Syntax errors occur when you have not obeyed the rules of Java, such as leaving out a semicolon at the end of the line.

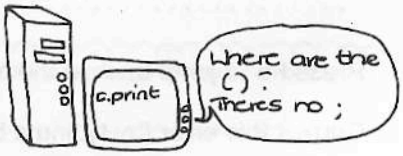
If there are syntax errors in your Java code, your program will not Run (because it will not compile). Instead, a dialog box indicating the number of errors appears on your screen.

The cursor is positioned in the line at which the computer thinks the first error occurs.

A message explaining each syntax error is shown in the Status Bar at the bottom of the screen.

Pressing F5 allows you to jump from one error to the next.

It is good to become familiar with the errors you encounter, so that you are able to fix them when they crop up again.



INVESTIGATION

Try to write down what is wrong with a Java statement when the following error messages appear in the Status Bar:

1. `Syntax : ";" inserted to complete ...Statement`

   .....

   .....

2. `Type string was not found`

   .....

   .....

3. `No field named "readInt" was found in type "hsa.Console"`

   .....

   .....

- Start a new HSA Boilerplate Application called **Errors1**.
- Type in the following code exactly as it is here, with the errors in it:

```
int number
string word;

c.println("Type in any integer and <Enter> ");
number = c.readInt;

c.println("If you can read this, you have fixed the program!");
```

Leave off the semi-colon ;

Use a lowercase 's' instead of a capital 'S'

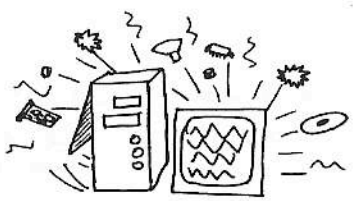
Leave off the () that should be after any method ie readInt();

- Run the program and copy down the number of errors shown in the Compilation Errors dialog box: `There was/were . . . error(s)`
- Copy down the error message from the Status Bar: .....
  .....
- Click OK, then correct the error (by adding the semi-colon at the end of the line).
- Run the program again and copy down the number of errors shown in the Compilation Errors dialog box: `There was/were . . . error(s)`
- Copy down the first error message from the Status Bar: .....
  .....
- Click OK, then press F5 to jump to the next error.
- Copy down this next error message from the Status Bar: .....
  .....
- Press F5 to jump back to the previous error.
- Correct this error (by typing a capital 'S' for String).
- Press F5 to jump to the next error.
- Correct this error (by adding the () after readInt to make it become `readInt()`).
- Run the program, typing in any integer value when prompted.

If your program runs and you get output, then you have successfully fixed the errors. Well done!

6.6.2 Run-time errors

A program that compiles successfully may still not work when you run it. Even if it has no compiler errors, it may have **run-time errors**.



A run-time error occurs when a compiled program tries to execute an illegal statement, such as:

- assigning a real number to an integer variable
- assigning a character (such as a letter) to a variable that has been declared as integer or real
- dividing by zero
- finding the square root of a negative number

6.6.2.1 Type mismatch errors

- Open the program **Errors1** if you do not still have it on your screen.
- Run the program and when prompted to type in an integer, type in a real number such as 3.5 and <Enter>.
- Copy down the contents of the dialog box that appears:

- Run the program again and when prompted to type in an integer, type in any non-numeric character on the keyboard such as a letter (k) or a symbol (#) and <Enter>.

The same type of error occurs.

This is called a **type mismatch**.

Different data types are stored differently in memory, so they cannot be assigned to each other.

An integer can be stored in a real, however, because integers are a subset of real numbers and when an integer is stored in a real variable it becomes a real number. (A *widening conversion* takes place automatically.)

A real number can only be stored in an integer variable if the computer is explicitly told to **typecast** the real number to an integer, using **(int)**.

6.6.2.2 Arithmetic Exceptions

- Add another integer variable called **number2** and assign to it the value zero (0).
- Change the output statement to become

```
c.println("The answer is " + number / number2);
```

- Run the program and type in any integer value (eg 7) when prompted.
- Copy down the error message that appears on the output screen: .....

.....

.....

In Java, an error is referred to as an **exception**.

In this case, the Arithmetic Exception (or error) is division by zero (/ by zero). The error message also states that the error has occurred in the main method and gives the line number at which the error occurred.

- Click the close button to return to the editor window. There should be a grey bar highlighting the line with the error, and if you look at the right of the Status Bar, you will see that the line number is given.
- Copy down the line numbering given: `Line ... of ...`
- Close the program.

A similar error will occur if other illegal arithmetic operations are attempted (such as finding the square root of a negative number).

You will learn at a later stage how to cope with errors / exceptions so that your programs will not crash if "illegal" values are entered.

6.6.3 Logical errors

Logical errors are the most difficult type of errors to fix as no error message is given by the computer. Your program will compile and run, but it won't do what you expect it to do. This happens when there is an error in the logic of the program. Remember, a computer only does what you tell it to do.

- Take a look at this segment of a Java program which has been designed to calculate the average of 3 numbers.

```
int m1,m2,m3;
double ave;

...
ave = m1 + m2 + m3 /3;
c.println(ave,5,2);
```

- Can you see the logical error? If so, what is it and how do we fix it?

If you haven't seen the error, type in these lines of code into a program and add statements to input values for the 3 variables. Run the program, using 10, 20 and 30 as data - then if you check your answer, you should see the problem!

It is important that you understand the problem completely and then are able to read and follow the program code to determine where a logical error has occurred.

6.7 Trace tables - a debugging tool

One of the best ways to try to find a logical error, is to go through your program line by line, checking what actually happens in each line of the program. We can use a trace table to do this.

Trace tables are done on paper to work out the values of the variables in a program. Not only does this **help to find logical errors** but it also helps to **understand the logic** of the program.

Errors in programs are often called **bugs**.

To **debug** a program means to remove all the errors.

Long ago computers were so big that actual bugs crept into them and caused problems. In those days you could debug by calling the exterminators!!!!

Study the following section of a Java program which will be used to show how to set out a trace table:

```
public static void main (String [] args)
{
      c = new Console ();

      int num1, num2, num3, num4, num5;

      num1 = 9;

      c.println ("Type in 2 integers, press <Enter> after each");
      num2 = c.readInt ();
      num3 = c.readInt ();

      num5 = num1 - (num2 - num3) % 5;
      num4 = 2 * num2;
      num2 = num4 + num5;
      num3 = num2 % 6;
      num1 = (num5 / 2) + (num3 * num3);

      c.print("12345678901234567890");
      c.print (num1, 3);
      c.print (num2, 3);
      c.println (num3, 3);
      c.print (num4, 6);
      c.print (num5, 6);

} // main method
```

The values 4 and 2 will be input

It is useful to display the value of variables during a program's execution to see what is happening.

Number the statements in the program from 1 to 15, starting with the statement `num1 = 9;` That is, write the number of each statement to the left of each statement in your book.

eg.
```
1     num1 = 9;
2     c.println ("Type in 2 integers, press <Enter> after each");
3     num2 = c.readInt ();
```
and so on ...

Fill in each variable's value in the correct column, or show what will appear on the screen, as you "carry out" each statement, in order. (The first few values have been filled in for you.)

TRACE TABLE:

VARIABLES IN MEMORY

Line No.	num1	num2	num3	num4	num5
1	9				
2					
3		4			
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					

SCREEN OUTPUT

```
Type in 2 integers,
press <Enter> after each
4
2

12345678901234567890
```

The values of the variables in a program are put into the columns of a table. We write down the value of each variable in the correct column, as each statement is carried out.

The current value of the variable is its last value; all the other values are **overwritten**.

We have put the variables and the screen output sections next to each other, but it is often best to put the screen output section underneath the variables section, because trace tables can become very wide.

STEPS TO MAKE A TRACETABLE:

1. Divide a page into 2 sections. One for all the **VARIABLES IN MEMORY**, and one for **SCREEN OUTPUT**.

2. Make one column in the memory section for line numbers and a column for each variable.

3. Number each line / statement of the program starting with the line after the variable declarations.

4. Start from the statement you numbered 1, and in the correct column in the trace table, write down the line number and the value of each of the variables, as you go from statement to statement.

5. If there are any **input** statements in the program, use the given data values.

6. If there are any **output** statements, write down exactly what will be displayed in the screen output section.

EXERCISE 6E

▶ Draw up trace tables on paper to show what will be stored in memory and what will be output in each of the following programs :

1 This program calculates the total amount due after doing some grocery shopping.

```
int num;
double price,cost,total;
String item;

total = 0;
item = "Baked beans";
num = 3;
price = 3.29;
cost = num * price;
total = total + cost;
item = "Kewl drink";
num = 5;
price = 1.95;
cost = num * price;
total = cost;
c.print("Total amount to pay: $ ");
c.println( total,8,2 );
```

▶ The program has a logical error. Use your trace table to help you find it and write down a short description of the problem:.

. . . . . .

▶ Correct the error in the program above. (Cross out the incorrect statement and write the correct statement next to it)

2

```
int a,c;
double b;

a = 17 - 3 * 4;
b = 7 - a*a;
c = (a + b) - b / a;
b = c / 4;
a = a % c;
c.print(a,5);
c.print(b,7,2);
c.print(c,5);
```

3

```
char character;
int len;

c.println("12345678901234567890");
len = 3;
character = '#';
c.print(character,len);
c.println(character,len+1);
len = len + 1;
c.print(character,len);
c.println(character,len-2);
character = '%';
c.println();
c.println(character,len+1);
len = len - 3;
c.println(character,len+4);
```

▶ Show clearly in which columns the output of this program will be on the screen.

6.8 User Friendliness

A **user friendly** program should use clear prompts to tell the user exactly what data to input. It should also display well lined up output, in an uncluttered way, with meaningful messages.

A **user friendly** program can help to avoid run-time errors because the user will know exactly what to type in.

The following program calculates the average of a pupil's 2 marks. It is **NOT user friendly.**

```
String name;
int mark1, mark2, average;

c.println("Type in a person's name and marks");
name = c.readString();
mark1 = c.readInt();
mark2 = c.readInt();
average = (mark1 + mark2) / 2;
c.println(average);
```

▶ What output will this program display on the screen if the following are input:

Jo 56 63

.....

▶ What is the purpose of this program? .....

.....

The program has at least 2 problems :

- it does not clearly tell the user how the data should be input
- the average of the 2 marks is displayed without any text.

A program can be made to be more **user friendly** by adding a prompt before **every** statement that reads data into a variable, so that a user knows exactly how what data to enter.

Output should be displayed on the output screen with **suitable messages** to make it meaningful.

The output should also be well laid out, and lined up if necessary.

- ▶ Type in the code shown above, exactly as it is, into a class called **UserFriendly**.
- ▶ Run it, inputting the data Jo 56 63. Check the output you wrote down.
- ▶ Fix the code so that it is **user friendly**. Run the program again to check that it really is user friendly.
- ▶ Save it and print it, filing it with your other printed programs.

6.9 Readability

A program is user friendly if a **user** can use it easily. A program is readable if a **programmer** can understand it easily.

A program is considered to be **readable** if it is easy for a programmer to understand. In reality more than one person is involved with developing a program or a set of programs. Often the person who originally wrote the code has left the company. In this case another programmer has to maintain the code by correcting errors or adding enhancements. It will take a long time for a programmer to understand the code if it is badly written, and time costs money!!!

At this level there are some simple techniques that make your program more readable.

1 Use descriptive variable names.
If a variable is going to store a sum call it `sum`. It can be misleading when a variable determines the average, but is called `sum`.

2 Include comments
If the variable name does not make the code easy to understand, include comments. At this stage the method names are fairly explanatory, but if you are doing something new that you doubt you will understand later, add in some explanations in the form of comments.

3 Separate code
To separate sections of code use blank lines. Try to group code that performs a similar function together. For example declare variables in one section. Perform calculations in another and output in another. Later on we will write our own methods to break the code into sections.

4 Follow conventions

- By convention, identifiers in Java start with lowercase letters.
- The identifier should reflect the contents of the variable.

Your teacher may have other conventions that should be followed. Use them as it makes marking quicker for your teacher, and it's easier to understand your friends programs.

KWIKWIZ

1. Explain the difference between **compiler, runtime** and **logical** errors. .....
   .....
   .....
   .....

2. Write down all the causes of a run time error ? .....
   .....
   .....

3. Consider the following piece of Java code:

```
int num;
...
num = c.readInt();
```

The user types in <Shift> + 5 when the program is Run. What type of error will occur ?
Explain. .....
.....

4. Find all the syntax, run time and logical errors in the following program segment:

The user entered 25 and 0 as input.

```
String name;
Int mark, OutOf;
Real percentage

//Input

println('Enter your mark out of 40');
mark = c.readDouble;
println('Enter what the mark was out of');
outof = c.readDouble;

//Processing
perc = mark/outof*100;

//Output
println('The mark converted to a percentage is + mark);
```

5. Circle TRUE (T) or FALSE (F) for each of the following assignment rules.
(Try these out on the computer if you are not sure of them.)

a. A real value cannot be assigned to an integer variable unless a cast statement is used. T / F

b. An integer value cannot be assigned to a real variable. T / F

c. A String cannot be assigned to a char variable, but it can be assigned to another String variable. T / F

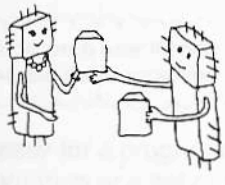
d. A character can be assigned to a String variable, but not to an integer variable. T / F

6. What is the purpose of a trace table? .....
.....

7. What should be used for the column headings in any trace table? .....
.....
.....

8. The program segment below has been written as a solution to swopping the values in the variables **num1** and **num2**.

```
num1 = num2;
num2 = num1;
```



Explain why the code is logically incorrect, with the aid of a trace table.

In order to swop the values in two variables successfully, a third variable is necessary. Try to work out the code to swop the values in num1 and num2 with the use of a third variable called **temp** .....
.....
.....

9. a. What is the difference between a program being **user friendly** and **readable**? .....
.....

b. What factors make a program user friendly? .....
.....

c. What factors make a program readable? .....
.....
.....
.....

10. Explain what takes place when a program is Run in Java. . . . . .

. . . . . .

. . . . . .

11. What is **bytecode**? . . . . . .

. . . . . .

6.10 Planning for more complicated programs

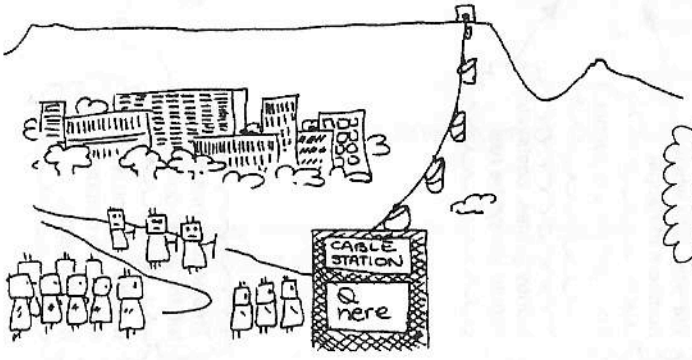
As problems become more complex, it becomes more and more difficult to get your programs to work correctly. Most people hardly ever get a program to work correctly the first time.

HOWEVER... **if you spend some time planning your solutions, you will have a better chance of getting them to work sooner.**

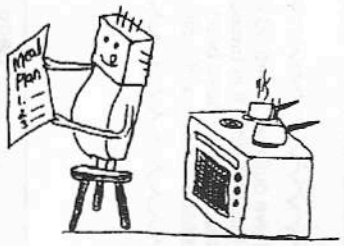
Take a look at this example:

MOUNTAIN TRIP
A group of 30 pupils, who are going to Cape Town on a sports tour, want to book a cable-car ride up Table Mountain. Some of the group want to walk down the mountain, while others want to ride back down in the cable car. If the price of a single trip is $40.00 and the price of return trips is $60.00, and a discount of 5% is given for groups of over 25 people, determine the total cost of this outing. (The discount is given regardless of whether people are riding both ways or not)

▶ Try to work out a solution to this problem in the space below. Do not use your computer at all!



On the next page we have outlined a method of planning a solution to this problem. It is based on the steps in the template which we have been using. Pay careful attention to each of the steps.



STEP 1

Read the problem details carefully and make rough notes to help you identify all the parts of the problem:

- We need to input how many pupils want single tickets.
- Then the computer can work out the number of return tickets, because there are a total of 30 pupils in the group.
- Then the total cost can be worked out, and, finally, the discount.

What **type** should the variables be ?

We work with whole numbers of people.
You can't get 2½ people.

Number of single tickets - integer
Number of return tickets - integer

Money (prices, costs, etc) should always be real.

Prices of single and return tickets - real
Total cost, discount and final cost - real.

STEP 2

Draw up the IPO and Variable tables on a sheet of paper so that it can be filed with the printed program once it has been completed.

This section has been left for you to do ...
Decide how you want to set out the output and fill it in alongside.

STEP 4

Plan what the **output** should look like.

STEP 3

Develop an **algorithm** or **rough plan** indicating what statements will be needed in the program:

An **algorithm** is a sequence of steps, in point form, which shows a solution to the problem. It is a rough plan and normally written in some form of English - **not** in Java code.

1. Declare all the variables.
2. Prompt the user to enter the number of single tickets required.
3. Read in num_single.
4. Calculate num_return by subtracting num_single from 30.
5. Work out the total_cost:
   = (num_single * single_price) +
   (num_return * return_price)
6. Work out the discount amount:
   = total_cost*5.00/100
7. Work out the final_cost:
   = total_cost - discount
8. Display the number of single tickets and the number of return tickets.
9. Display the total_cost.
10. Display the value of final_cost with a suitable message.

In general, the order of an algorithm is the same as a program (IPO):

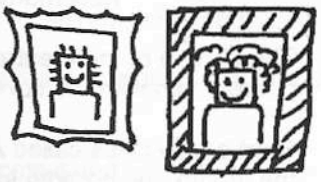
- input data
- do calculations (process)
- output answers

STEP 5

Code the program

- Start a new class called **Mountain**.
- Using the algorithm as a guide, code a program to solve this problem.
- Run your program 3 times, using different input values.
- When you are satisfied that it works properly, print it and file it with the IPO and Variable tables.

Here is another example for you to practice planning a solution to a more complicated problem, using the 5 steps that were detailed in the previous problem.



PICTURE FRAMES

You have decided to sell painted wooden picture frames at flea markets and so you are investigating the options. There are several calculations that have to be done.

Wooden frames can be bought for $14.00 per frame, but these have to be bought in packs of 50. A small tin of paint will cost you $12.00 and it will be enough to paint 27 frames. You need to work out what it will cost you to produce 50 painted frames and how much you will need to charge in order to make 40% profit.

- Write down the details of steps 1,2 3, and 4 on a piece of paper.

STEP 1 - Make sure that you understand all parts of the problem, then write down a rough solution.

Some of the things you need to work out are:

- How much does the paint for each frame cost?
- Are you going to include a "labor" cost?
- How do you calculate 40% profit?
  - Remember, the computer doesn't understand '%' - you must multiply by 40 and then divide by 100.

STEP 2 - Draw up the IPO and Variable tables.

STEP 3 - Write an algorithm showing the logical steps in your solution.

STEP 4 - Plan what the output will look like.

In this particular example, the layout is not necessarily important but make sure that you work accurately with the money figures, otherwise your profit might be reduced.

STEP 5 - Code a Java program, using your planning to guide you.

- When you have checked that your program works correctly, print it and file it, together with your planning steps.

EXERCISE 6F

Use paper to plan solutions to each of the following problems and then code them into Java programs. Make up your own class names, if they are not given.

1 The prices charged to attend a certain theatre show are $16.50 for adults and $12.50 for children. Write user-friendly code in a new class called **Theatre** which asks for the number of adults and children to be input from the keyboard. Calculate the total price of the theatre tickets. Design a simple ticket that contains:

- the name of the theatre
- the name of the show (make these up)
- how many adults and how many children in the group

The person at the theatre door must be able to check all these details when the people present him with the ticket. Print the ticket on paper.

2 Start a new class called **AreaPerim**. Enter the length and breadth of a rectangle. Store the length and breadth in variables called LEN and BREADTH respectively. Calculate the area and the perimeter and store them in variables called AREA and PERIM. Calculate the length of the diagonal of the rectangle. Display all the results.

3 At a restaurant, a group at a table orders drinks. Cold drinks cost $4.85 each, wine costs $6.50 per glass and a beer costs $5.00. Input the number of each type of drink ordered and display an invoice showing items ordered with individual subtotals, and the total cost. Work out a tip of 10% for the waitron and include this on the invoice, showing the total including the tip.

4 Input 3 positive real numbers which are the lengths of the sides of a triangle. Use the following to determine the perimeter and area of the triangle:

Perimeter = side1 + side2 + side3
S = perimeter/2

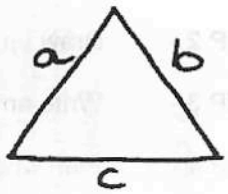
$$Area = \sqrt{S(S - side1)(S - side2)(S - side3)}$$

Display the lengths of the 3 sides, as well as the perimeter and area of the triangle.

Your output should look like this:

The lengths of the 3 sides of a triangle are:
3.85 cm 4.13 cm 2.53 cm

The perimeter of the triangle is 10.51 cm
The area of the triangle is 4.76 square cm



5 The walls of a rectangular room need painting.

a. Determine the area that must be painted if the room has the following dimensions: Length: 5.95 m, breadth: 3.6 m, height: 3.2 m.

b. If the cost of paint is $4.50 per square metre. Determine how much it will cost to paint the room.

c. Design a statement that will give the dimensions of the room, the cost of the paint and the total cost of the job. Print out a statement on hard copy.

6 Write a program that displays a monthly statement for cell D cellular phone service provider. A person who uses this service is charged a basic rate of $135.00 per month, regardless of whether or not he / she makes any phone calls. The cost of a call is $0.98 per minute, but this company charges per second so a call that lasts for 1½ minutes will cost $1.47.

Display a monthly statement which includes the following:

- Name of service provider
- Name of user
- Month and year
- Rate per minute and basic monthly fee
- Total cost of the service for the month, including the basic rate
- Total cost of calls
- Total number of hours, minutes and seconds that the service was used

The user must enter his/her name, the month and the total number of seconds (of calls) for the month.

7 The speed at which a dropped ball is travelling at, can be calculated using:

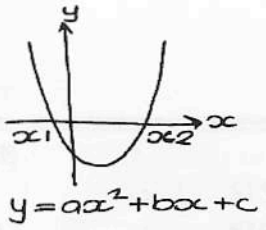
$$s = \frac{1}{2}gt^2$$

where t is the time in seconds the ball has been falling, and g is the gravitational constant which is approximately 10 m/s².

Write a program that will input the time t and determine s the speed at which the ball is travelling.

In reality the speed is actually called the velocity.

8 A parabola has the general form: $ax^2 + bx + c$



To find where this graph cuts the x axis, the equation must be solved equal to 0. This produces the following formula to find both the x values:

$$x1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \quad \text{and} \quad x2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

Write a program to input values for a, b and c and calculate values for x1 and x2 to two decimal places.