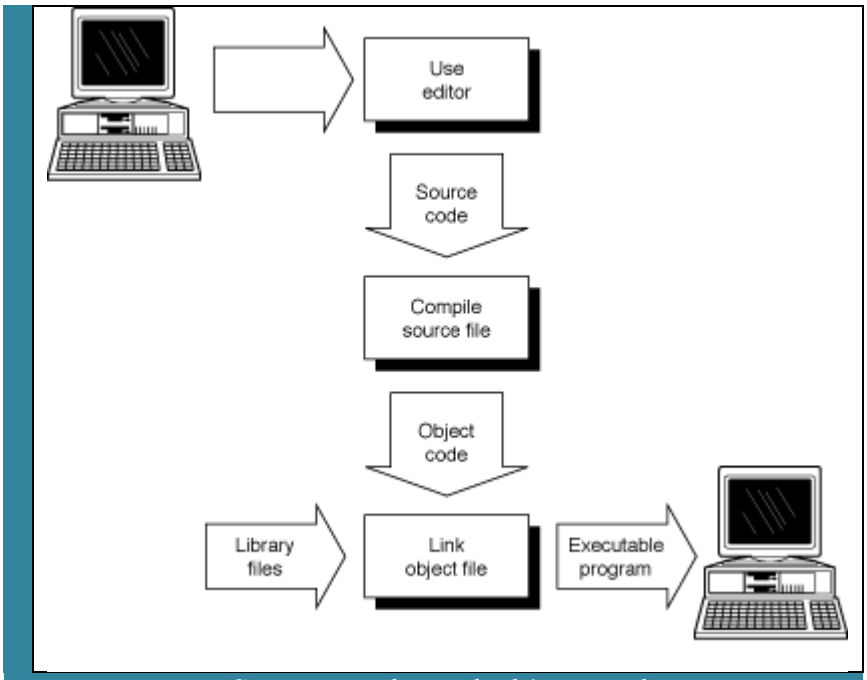
Java Notes for Computing Intermediate

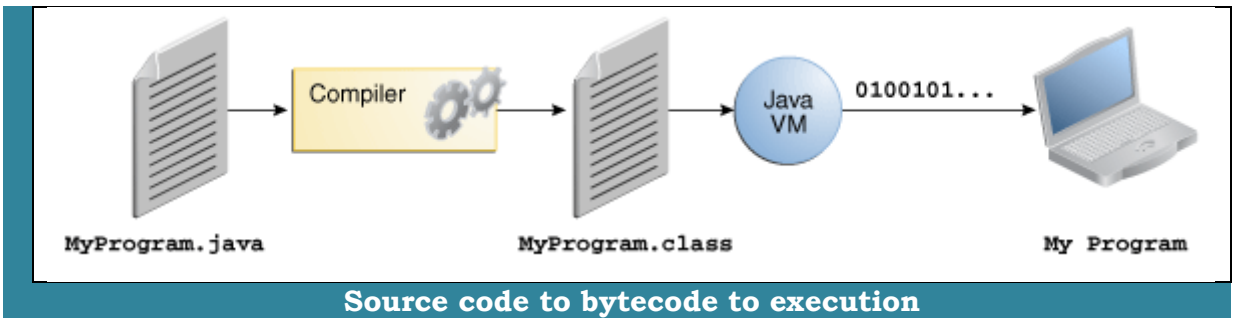
1 Source code and Object code



Source code and object code

Source code is the code as written by the programmer. **Object code** is the code (in binary) as presented to the computer. This conversion is done by a program called a **compiler**.

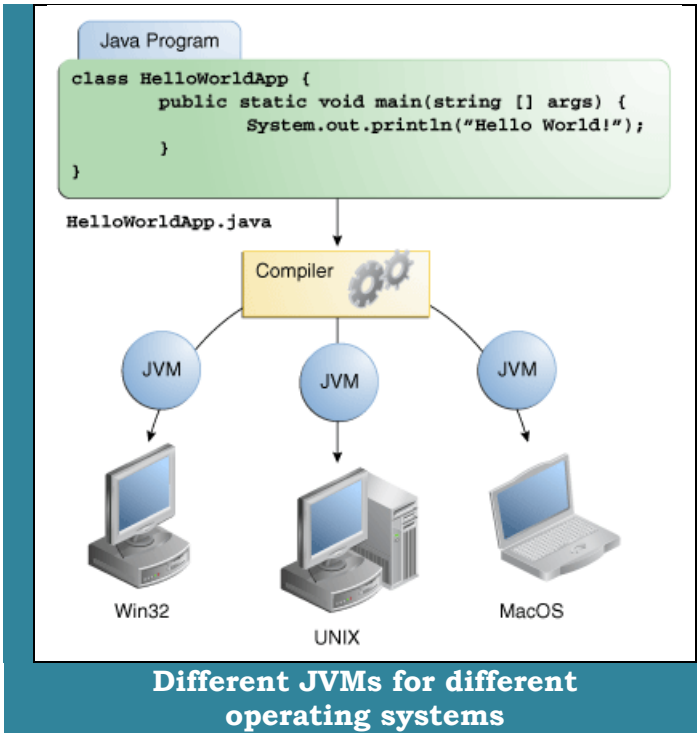
2 Bytecode



Source code to bytecode to execution

In the Java programming language, all source code is first written in plain text files ending with the **.java extension**. Those source files are then compiled into **.class files** by the **javac compiler**. A .class file does not contain code that is native to your processor; it instead contains **bytecodes** — the machine language of the **Java Virtual Machine** (Java VM).

Because the Java VM is available on many different operating systems, the same .class files are capable of running on Microsoft Windows, the Solaris™ Operating System (Solaris OS), Linux, or Mac OS.



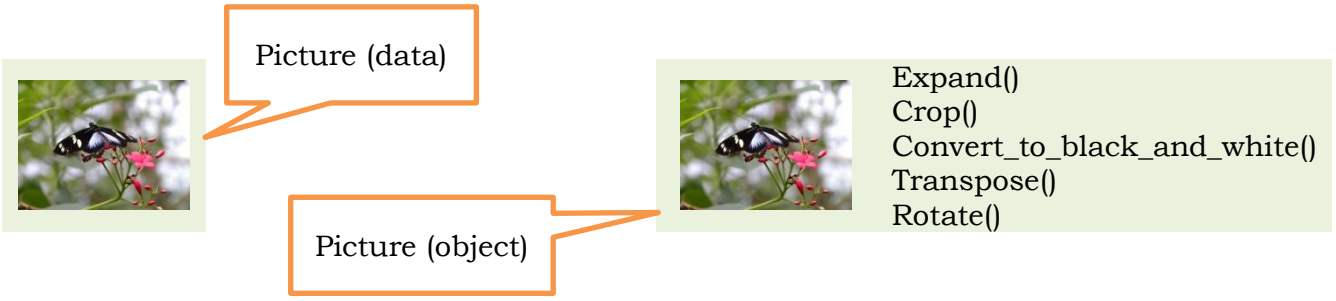
Different JVMs for different operating systems

3 BlueJ

BlueJ is a free **IDE** (**integrated development environment**) for Java! It is the IDE we will be using in the lab. The BlueJ environment was developed as part of a university research project about teaching object-orientation to beginners.

4 Java is object-oriented

The idea of languages being object-oriented made its first appearance in the late 1950s at MIT. The idea is that of joining data to procedures that perform some kind of manipulation on the data. Let us take the example of a picture. Previous to object-oriented ideas a picture was considered as a formation of pixels. In the object-oriented world a picture is considered as a formation of pixels plus a number of operations that can be performed on pictures like expand, crop, convert to black and white etc.



5 First Program

```
   Class HelloWorld
1  public class HelloWorld
2  {
3      public static void main(String[ ] args)
4      {
5          System.out.println("Hello world");
6      }
7  }
```

Line 1

- This is called the **header**.
- A program consists of a **class** or a group of classes.
- The keyword **public** shows that there is no limit on who can view and use this class.

Lines 3 to 6

- They form a **method** i.e. a block of instructions that are grouped together to form a mini-program.
- It is the **main** method which means that the program starts execution from this method.

Line 5

- "Hello world" is called a **string**.
- **System.out.println** is used to display strings and other values on the screen.

Line 3

- Other terms in this line will be explained later on.

6 Adding Comments

```
   Class HelloWorld with comments
1  //this class displays two words on the screen
2
3  public class HelloWorld
4  {
5      public static void main(String[ ] args)
6      {
7          System.out.println("Hello world");
8      }
9          /* this style is used for comments
10         that are longer than one line */
11 }
```

- **Comments** are used to make a program more **readable**.
- Comments are ignored by the compiler.
- There is more than one way to insert comments

7 Variables and Types

```
   Class Variables Types
1  public class VariablesTypes
2  {
3
4      public static void main (String[ ] args)
```

```
5       {
6           // the following are declarations
7           int i;
8           double d;
9           boolean b;
10          String s;
11
12          // the following are assignment statements
13          i = 86;
14          d = 34.6;
15          b = true;
16          s = "Java is object-oriented";
17
18          //The following statements display expressions
19          System.out.println (i - 6);
20          System.out.println (d - 8.5);
21          System.out.println (!b);
22          System.out.println (s + " and it was originated from C++.");
23      }
24  }
```

Refer to the above program.

- i, d, b and s are **variables**. Each one of them holds a value. The values they hold are different. i holds a whole number, d holds a number which may contain a decimal point, b holds a Boolean value and its value can only be equal to *true* or *false*, and s is a string which means s is a sentence.
- Lines 7 to 10 are called **declarations**. This means that Java is informed by the programmer that he/she will be using those variables in the program. Before using a variable this must be declared.
- Lines 13 to 16 are called **assignment statements**. In these lines a value is assigned to each variable. Later in the program this value may change. In the above program the values of the variables do not change but in real programs they often do. We will see this in later programs.
- A **statement** is a command. It tells Java to perform some kind of operation. Some statements can be quite long. Consider the statement in line 22. This statement is telling Java to display on the screen, first the value of s and next to it the string " and it was originated from C++.".
- In line 21 '!b' stands for 'not b' which is equal to *false*.
- Note that the delimiters of strings are inverted commas. **Delimiters** are special characters at the left and right ends of strings.

7.1 Exercise

Write a program that implements the following steps:

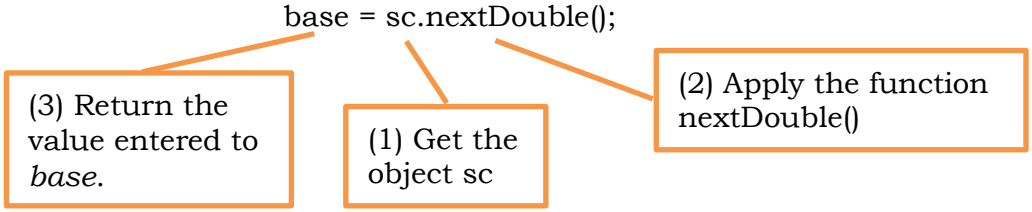
1. Create a class called Ex_7_1
2. Create the main method
3. In the main method declare three *double* variables called *length, breadth* and *area.*
4. Assign 5.3 to *length.*
5. Assign 6.1 to *breadth.*
6. Calculate *area* of the rectangle.
7. Display the *area* of the rectangle.

8 Input

```
    Class Input
1   import java.util.*;
2
3   public class Input
4   {
5      public static void main (String [ ] args)
6      {
7         Scanner sc = new Scanner (System.in);
8
9         double base, height, area;
10
11        System.out.println ("AREA OF A TRIANGLE");
12        System.out.println ("Input the length of the base");
13        base = sc.nextDouble();
14        System.out.println ("Input the height of the triangle");
15        height = sc.nextDouble();
16
17        area = (base * height) / 2;
18
19        System.out.println ("The area of the triangle is equal to " + area);
20     }
21  }
```

Refer to the above program:

- One way to input values is to use the **Scanner** class. This class is defined in a package that must be imported so that it can be used. This is the reason of line 1. A **package** is a collection of pre-defined classes.
- In line 7 the program is creating an object (called sc) from the class Scanner so that this object can be used in this class (Input). This is a fundamental idea in object-oriented languages i.e. creating objects from classes as necessary (more than one object can be created out of the same class). In fact we say that an **object** is an instance of a **class**.
- Lines 13 and 15 use the method nextDouble() to enter double values from the keyboard. nextDouble() is a method defined in Scanner. In fact line 13 can be explained in this way:



8.1 Exercise

1. Write a program that receives in input the three measurements of a cuboid and calculates the volume of the cuboid.
2. Write a program that accepts three marks and calculates the average.
3. Write a program that receives in input a temperature in degrees Celsius and converts this in degrees Fahrenheit. The formula is the following: $F = C\left(\frac{9}{5}\right) + 32$.

9 Taking a Decision

One of the most powerful concepts in programming is teaching a program how to take a decision. One construct to do this is called the 'if … then … else'. Some examples are shown in the following table.

if today is Sunday **then** no study **else** study
if you are younger than 18 **then** you cannot vote **else** you can vote
if it is raining **then** get an umbrella **else** do not get an umbrella

A few points:

1. In Java the keyword 'then' is not written down. In some other languages it is.
2. Sometimes the 'else' part is left out and we may write:

if it is raining **then** get an umbrella

3. The 'if .. then … else' statement is called a **conditional statement** or a **selective statement**. The expression between the keywords 'if' and 'then' is called a **condition**.

if condition is true **then** perform an action **else** perform some other action

Consider the following program.

Class using If .. then .. else

```
1  import java.util.*;
2
3  public class IfThenElse
4  {
5     public static void main (String [ ] args)
6     {
7        Scanner sc = new Scanner (System.in);
8
9        int age;
10
11       System.out.println ("Input your age");
12       age = sc.nextInt();
13
14       if (age < 18)
15          System.out.println ("You cannot vote");
16          else System.out.println ("You can vote");
17
18    }
19 }
```

9.1 Exercise

1. A program asks a user to enter the number of sides of a polygon. If the number 4 is entered then the program displays the message "the shape is a quadrilateral". Otherwise it displays the message "the program is not a quadrilateral".

2. This program is about whether profit was made from the sale of an object. The program asks the user "how much was the object bought?" and the user enters the price. Then the program asks how much it was sold. The program then displays either the message "a profit was made" or "no profit was made".

3. A club is organising a buffet. When one books for the buffet one will pay €12 per person if the number of persons being booked is more than 4. Otherwise the price is €15 per person. Write a program that asks the user to enter the number of persons to be booked for the buffet and then the program will display the total amount to be paid.

4. A person is organising a meal in a restaurant. Adults pay €10 and children €6. If the total price to be paid exceeds €30, a discount of 10% is given. Write a program that calculates the amount to be paid after being given the number of adults and children.

10 The 'for' loop

In Java (and other languages) the programmer has the structures to repeat a number of statements for a required number of times. In Java there are three kinds of **loops** (or repetitions). These are:

- for loop
- while loop
- do … while loop

In this section we will consider the **'for'** loop. Refer to the following program:

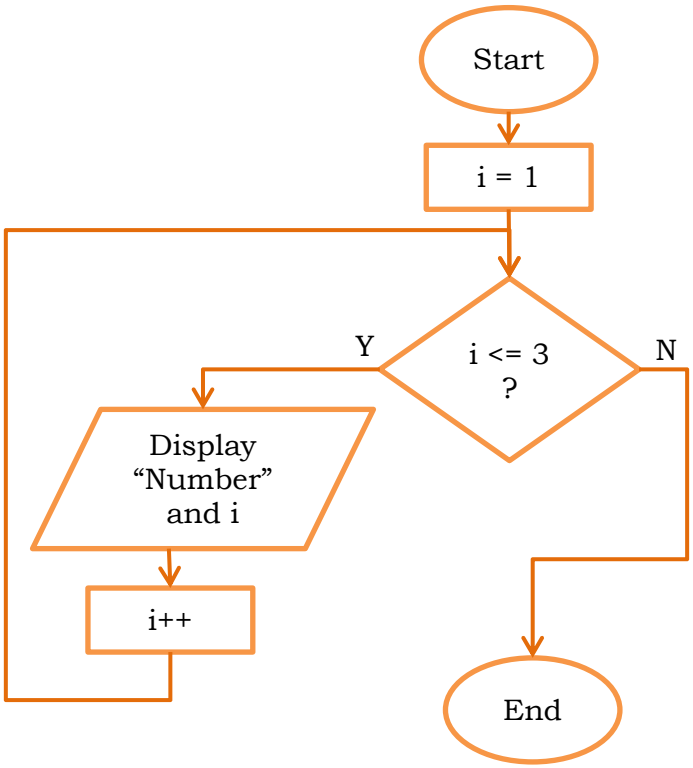
Class using the 'for' loop

```
1  public class forLoop
2  {
3      public static void main (String [ ] args)
4      {
5          for (int i=1; i<=3; i++)
6              System.out.println ("Number " + i);
7      }
8  }
```

- In line 5 the variable i is called a **counter**. The brackets following the 'for' keyword consists of 3 parts:
  - In the first part the counter is initialised. In this case i is started at value 1.
  - The middle part is called the **condition**. In this case the loop is repeated if i is less or equal to 5. If not the repetition stops.
  - The third part is a statement which indicates what must be done after each repetition (or **iteration**).
- Lines 5 and 6 do the following:
  - i is started with value 1
  - The condition is checked. Since 1 is less than 3 then line 6 is executed

- The value of i is incremented by 1. Now i is 2.
- The condition is checked. Since 2 is less than 3 then line 6 is executed.
- The value of i is incremented by 1. Now it is 3.
- The condition is checked. Since 3 is equal to 3 then line 6 is executed.
- The value of i is incremented by 1. Now it is 4.
- The condition is checked. Since 4 is not less or equal to 3 then the loop is stopped.

The following is a flowchart for the program seen above.



10.1 Exercise

1. Write a program that displays the values 10, 11, 12 … 20.
2. Write a program that displays the values 2, 4, 6 … 20
3. Write a program that displays a table of values 10, 20, 30 … 100 (representing degrees Centigrade) and the equivalent temperature in Fahrenheit. The formula to convert Centigrade to Fahrenheit is $F = C\left(\frac{9}{5}\right) + 32$.
4. Write a program that asks the user to input 5 numbers. The program adds their total and displays it.
5. Write a program that asks the user to input 10 numbers. The program counts how many of these numbers are greater than zero.

11 The 'while' loop

The following program uses the '**while**' loop. It produces the same output as the previous 'for' program.

```
    Class using the 'while' loop
 1  public class WhileLoop
 2  {
 3     public static void main (String [ ] args)
 4     {
 5        int i=1;
 6
 7        while (i<=10)
 8        {
 9           System.out.println ("Number " + i);
10           i++;
11        }
12     }
13  }
```

Some points:

- The 'while' loop starts at line 7 and finishes at line 11.
- In line 7 the expression following the 'while' keyword is called a **condition**. When this condition is true (i.e. when i is less or equal to 10) the code inside the loop is executed. Otherwise the loop ends.
- The **counter** 'i' is initiated in line 7 (i.e. outside the loop). Inside the loop it is incremented (or decremented) as required. The counter controls the number of times a loop is executed.

11.1 A second example

```
    Another class using the 'while' loop
 1  public class WhileLoop2
 2  {
 3     public static void main (String [ ] args)
 4     {
 5        int i=1;
 6
 7        System.out.println ("Seven times table");
 8        System.out.println();
 9
10        while (i<=10)
11        {
12           System.out.println ("7*" + i + " = " + 7*i);
13           i++;
14        }
15     }
16  }
```

11.2 A third example

```
    Class using two 'while' loops and an 'if' statement
 1  import java.util.*;
 2
 3  public class WhileLoop3
 4  {
 5     public static void main (String [ ] args)
 6     {
```

```
7         final double euroToDollar = 1.34;
8         final double dollarToEuro = 0.75;
9
10        System.out.println ("Euros / US dollars convertor");
11        System.out.println();
12
13        Scanner sc = new Scanner (System.in);
14
15        System.out.println ("Input 1 or 2 as indicated");
16        System.out.println ("1: Euros to Dollars");
17        System.out.println ("2: Dollars to Euros");
18
19        int choice = sc.nextInt();
20
21        if (choice == 1)
22        {
23          int i=1;
24
25          while (i<=10)
26          {
27             System.out.println (i + " Euros = " + i*euroToDollar + " US Dollars");
28             i++;
29          }
30        }
31        else if (choice == 2)
32              {
33                 int i=1;
34
35                 while (i<=10)
36                 {
37                    System.out.println (i + " US Dollars = " + i*dollarToEuro + " Euros");
38                    i++;
39                 }
40              }
41              else System.out.println ("Wrong input. Please run the program again.");
42    }
43 }
```

11.3 An example using strings

Class to find longest word entered

```
1  import java.util.*;
2
3  public class LongestWord
4  {
5     public static void main (String [ ] args)
6     {
7        Scanner sc = new Scanner (System.in);
8
9        String longest_word = "";
10       int len = 0;
11       int l = 0;
12       String word = "continue";
```

```
13
14          while (!word.equals("stop"))
15          {
16              System.out.print ("Enter a word : ");
17              word = sc.nextLine();
18              l = word.length();
19              if (l > len)
20                  {len = l;
21                   longest_word = word;
22                  }
23          }
24
25          System.out.println ("The longest word entered was " + longest_word);
26      }
27  }
```

Comments on the program:

- In line 14 the expression 'word.equals("stop")' compares the value of the variable 'word' (which is a string) to the string "stop". If the values are equal (i.e. the strings are the same) the expression will be equal to 'true'. If not it will be equal to 'false'. The exclamation mark is equivalent to a 'not' and reverses the value of the expression just mentioned. So '!word.equals("stop")' is equivalent to 'if word is not equal to 'stop".
- The word entered can be a sentence as it can contain spaces.

11.4 Exercise

1) Write a program that produces the following sequence of numbers 2, 5, 8, … 47.
2) Write a program that given the input n (a positive whole number) it outputs 2^n.
3) Write a program that given inputs b and n (where n is a positive whole number) it outputs b^n.
4) Write a program that receives a number of names. When the input is 'stop' it means that there are no more inputs. The program counts the number of entries equal to 'Mario'.
5) Write a program that receives a number of names. When the input is 'stop' it means that there are no more inputs. The program calculates the average length of the names.
6) Write a program that receives names of two contestants for an election. The program adds the number of votes obtained by Mary and Jane. Then it declares the winner.
7) Write a program as above but now add the feature that the names of the contestants are inputted by the user.

12 Random numbers

Class to find 10 random integers

```
1   import java.util.*;
2
3   public class RandomNos
4   {
5       public static void main (String [ ] args)
6       {
7           Random rand = new Random();
```

```
 8
 9          for (int i=1; i<11; i++)
10          {
11              int r = rand.nextInt(100); // 0-99
12              System.out.print (r + " ");
13          }
14      }
15  }
```

- The above program makes use of the **Random class** which is found in the java.util package. It is used to generate random numbers.
- In line 7 an object named 'rand' is created from the class Random. This object is then used (in line 11) to create random integers. The use of the Random class is similar to the use of the class Scanner.
- rand.nextInt(100) calls the object 'rand' and applies the method 'nextInt' to it. The number 100 is called a parameter. This method gives a number between 0 and 99.

12.1 Exercise

1) Write a program that simulates a die. The program outputs a number from 1 to 6.
2) Write a program that outputs 20 random numbers from 1 to 50.
3) Write a program that outputs 3 numbers from 1 to 10. The numbers have to be different.
4) Write a program that implements a game called 'Guess the Number'. The program generates a random number from 1 to 100. It asks the user to guess the number. After each guess the program displays one of the following messages: (i) the number is bigger, (ii) the number is smaller, or (iii) you guessed the number. It is only when the user guesses the number that the program ends.
5) Write a program that rolls a die 1000 times. The program outputs the number of times each number (from 1 to 6) has appeared.

13 Methods

A class using methods

```
 1  public class Methods
 2  {
 3      public static int add (int a, int b)
 4      {
 5          return (a + b);
 6      }
 7
 8      public static void main (String args[])
 9      {
10          int r;
11          r = add (6, 4);
12          System.out.println ( r );
13
14          int p, q;
15          p = 12;
16          q = 7;
17          System.out.println ( add (p, q) );
18      }
19  }
```

- The class Methods seen above makes use of methods.
- A **method** is like a small program within a larger program. It is first declared (defined) and then used as often as required in the program. The method 'add' is **declared** in lines 3 to 6. It is used in lines 11 and 17.
- A method can **return** a value. It can only return one value not more. It can also be declared so that it does not return a value.
- The first line of a method is called the **signature**. So these are the signatures of the two methods of the above class:
    - public static int add (int a, int b)
    - public static void main (String args[])
- Let us explain line 11:
    - first Java calculates add (6, 4) (this is called a **method call**)
    - so it goes to line 3
    - a is assigned 6 and b is assigned 4
    - the method returns 6+4
    - the processing continues in line 11 where r is assigned the value returned by add (6, 4) so r is assigned the value of 10
- a and b are called **parameters** (or **formal parameters**) while the values 6 and 4 (and p and q in line 17) are called **arguments** (or **actual parameters**)
- note that in the program we have two method calls i.e. in lines 11 and 17
- in the signature, the term that precedes the name of the method is the type of the value returned; if no value is returned this term will be '**void**'.

13.1 Exercise

1) Write a program that consists of three methods. One method calculates the volume of a cylinder ($\pi r^2 h$), another calculates the volume of a sphere ($V = \frac{4}{3}\pi r^3$) and the third is the method main. The main method presents a simple menu so that the user can choose which volume to calculate.
2) Write a program with the following methods:
    a. A method that converts degrees Centigrande to Fahrenheit (F = C × 9/5 + 32).
    b. A method that converts degrees Centigrade to Kelvin (K = C + 273.15).
    c. The main method that asks the user which conversion she wants to calculate and then performs the required calculation.
3) Write a program that performs calculations on three numbers. The program has to contain the following methods:
    a. A method that decides whether the three numbers are all equal (it will return 'true' only when all three numbers are equal; in the other cases it will return 'false')
    b. A method that will return the maximum number.
    c. A method that calculates (and returns) the average.
    d. A method that displays the numbers in ascending order.
    e. The method 'main' that asks the user for three numbers and then uses all the other three methods on the numbers.

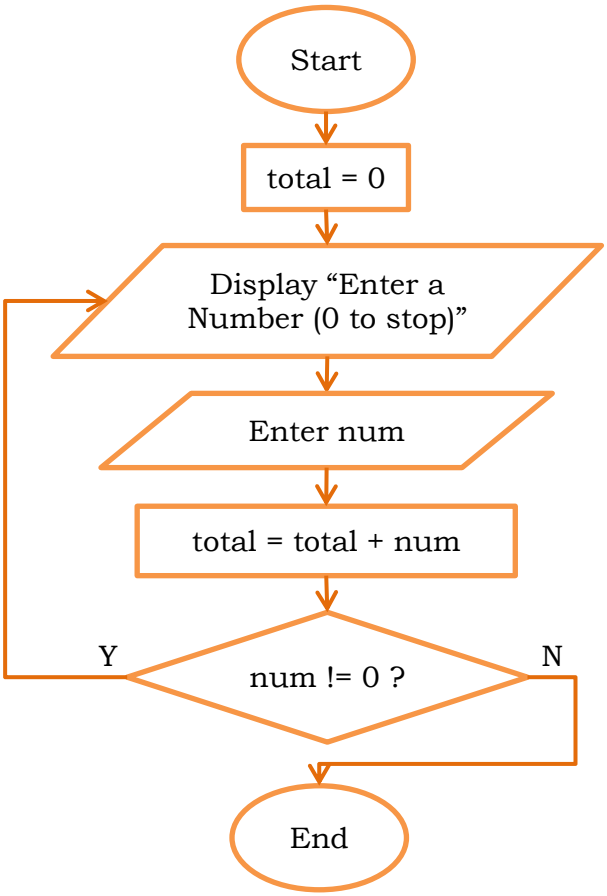
14 The do ... while loop

Another loop (besides the 'for' loop and the 'while' loop) is the 'do...while' loop. This is similar to the 'while' loop but has the condition for checking whether to repeat the loop or not, at the bottom of the loop. Here is an example:

Class to add a sequence of numbers

```
1  import java.util.*;
2
3  public class DoWhile
4  {
5     public static void main (String [ ] args)
6     {
7        double num, total;
8
9        Scanner sc = new Scanner (System.in);
10
11       total = 0;
12
13       do
14         {
15           System.out.print ("Enter a number (0 to stop): ");
16           num = sc.nextDouble();
17           total = total + num;
18         }
19       while (num != 0);
20
21       System.out.println ("The total is " + total);
22    }
23 }
```

The flowchart of the above program is shown hereunder:



14.1 Exercise

1) Modify the above program so that it calculates the average of the numbers entered. As in the above program:
   - The programmer does not know beforehand how many numbers the user will enter.
   - When the user enters the number 0 this would indicate that there are no more numbers to enter.
2) Write a program that asks the user to enter the number 7. If the user does not enter the number 7 the program will keep asking the user to enter a number until the number 7 is entered.
3) Write a program that asks the user to enter a number between 1 and 6. The program will keep asking the user to enter a number until a number in this range is entered.
4) Write a program that accepts numbers until the zero is entered. The program will add the positive numbers and will discard the negative numbers.
5) Write a program that receives in input a positive integer and it outputs an integer with the digits in reverse e.g. 86 will become 68, 9052 will become 2509.

15 Primitive data types

Primitive data types have two characteristics:

1) They are defined in the language.
2) They are **atomic** i.e. they cannot be divided into smaller parts. An integer is atomic but a string is not. A string can be divided into the characters that it is made up of.

The eight primitive data types supported by the Java programming language are:

- **byte**: The byte data type is an 8-bit signed two's complement integer. It has a minimum value of -128 and a maximum value of 127 (inclusive).
- **short**: The short data type is a 16-bit signed two's complement integer. It has a minimum value of -32,768 and a maximum value of 32,767 (inclusive).
- **int**: The int data type is a 32-bit signed two's complement integer. It has a minimum value of -2,147,483,648 and a maximum value of 2,147,483,647 (inclusive).
- **long**: The long data type is a 64-bit signed two's complement integer. It has a minimum value of -9,223,372,036,854,775,808 and a maximum value of 9,223,372,036,854,775,807 (inclusive).
- **float**: The float data type is a real number that ranges from approximately -3.40E+38 to 3.40E+38. It is a single-precision 32-bit floating point.
- **double**: The double data type is a double-precision 64-bit floating point. Its range of values is approximately from -1.80E+308 to 1.80E+308.
- **boolean**: The boolean data type has only two possible values: true and false. This data type represents one bit of information, but its "size" isn't something that's precisely defined.

- char: The char data type is a single 16-bit Unicode character. It has a minimum value of '\u0000' (or 0) and a maximum value of '\uffff' (or 65,535 inclusive).

16 Arithmetic operators

They are the following:

+	Additive operator
-	Subtraction operator
*	Multiplication operator
/	Division operator (gives a different result for integers and real numbers)
%	Remainder operator

Execute the following program:

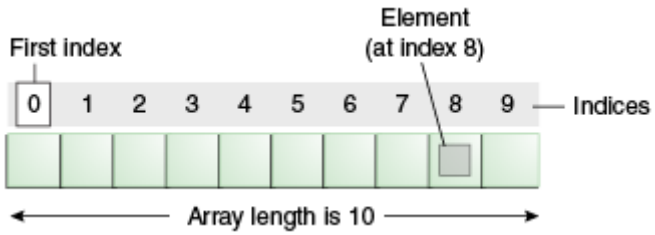
Class that performs arithmetic operations

```
1  public class ArithOper
2  {
3     public static void main ( String args[] )
4     {
5        System.out.println ("25/4 = " + 25/4); //division of two integers
6        System.out.println ("25.0/4.0 = " + 25.0/4.0); //division of two real numbers
7        System.out.println ("25%4 = " + 25%4); //operator mod
8     }
9  }
```

17 Arrays

An **array** is a container object that holds a fixed number of values of the same type. The length of an array is established when the array is created. After creation, its length is fixed.

17.1 Element and index



Each item in an array is called an **element**, and each element is accessed by its numerical **index**. As shown in the preceding illustration, numbering begins with 0. The 9th element, for example, would therefore be accessed at index 8.

The following program, ArrayDemo, creates an array of integers, puts some values in the array, and prints each value to standard output.

```
    Class that uses arrays
1   class ArrayDemo
2   {
3       public static void main(String[] args)
4       {
5       // declares an array of integers
6         int[] anArray;
7
8         // allocates memory for 10 integers
9         anArray = new int[10];
10
11        // initialize first element
12        anArray[0] = 100;
13        // initialize second element
14        anArray[1] = 200;
15        // and so forth
16        anArray[2] = 300;
17        anArray[3] = 400;
18        anArray[4] = 500;
19        anArray[5] = 600;
20        anArray[6] = 700;
21        anArray[7] = 800;
22        anArray[8] = 900;
23        anArray[9] = 1000;
24
25        System.out.println("Element at index 0: " + anArray[0]);
26        System.out.println("Element at index 1: " + anArray[1]);
27        System.out.println("Element at index 2: " + anArray[2]);
28        System.out.println("Element at index 3: " + anArray[3]);
29        System.out.println("Element at index 4: " + anArray[4]);
30        System.out.println("Element at index 5: " + anArray[5]);
31        System.out.println("Element at index 6: " + anArray[6]);
32        System.out.println("Element at index 7: " + anArray[7]);
33        System.out.println("Element at index 8: " + anArray[8]);
34        System.out.println("Element at index 9: " + anArray[9]);
35    }
36  }
```

The output from this program is:

```
Element at index 0: 100
Element at index 1: 200
Element at index 2: 300
Element at index 3: 400
Element at index 4: 500
Element at index 5: 600
Element at index 6: 700
Element at index 7: 800
Element at index 8: 900
Element at index 9: 1000
```

In a real-world programming situation, you would probably use one of the supported looping constructs to iterate through each element of the array, rather than write each line individually as in the preceding example.

17.2 Declaring a Variable to Refer to an Array

You can declare arrays of other types:

```
byte[] anArrayOfBytes;
short[] anArrayOfShorts;
long[] anArrayOfLongs;
float[] anArrayOfFloats;
double[] anArrayOfDoubles;
boolean[] anArrayOfBooleans;
char[] anArrayOfChars;
String[] anArrayOfStrings;
```

You can also place the brackets after the array's name:

```
// this form is discouraged
float anArrayOfFloats[];
```

However, convention discourages this form; the brackets identify the array type and should appear with the type designation.

17.3 Creating, Initializing, and Accessing an Array

The following statement in the ArrayDemo program allocates an array with enough memory for 10 integer elements and assigns the array to the anArray variable.

```
// create an array of integers
anArray = new int[10];
```

Alternatively, you can use the shortcut syntax to create and initialize an array:

```
int[] anArray = {
    100, 200, 300,
    400, 500, 600,
    700, 800, 900, 1000
};
```

Here the length of the array is determined by the number of values provided between braces and separated by commas.

The array that we saw in the example is called a **one-dimensional array**. You can also declare an array of arrays (also known as a **multidimensional array**) by using two or more sets of brackets, such as String[][] names. Each element, therefore, must be accessed by a corresponding number of index values.

17.4 Example: average age

Class that uses arrays and finds average age

```
1  public class Arrays
2  {
3      static String names [] = new String [5];
4      static int ages [] = new int [5];
5
```

```
6       public static void main (String args[])
7       {
8          int aveAge;
9
10         enterValues();
11         aveAge = averageAge();
12         System.out.println ("The average age is " + aveAge);
13      }
14
15      public static void enterValues ()
16      {
17         for (int i=0; i<5; i++)
18         {
19            System.out.print ("Enter name: ");
20            names[i] = Keyboard.readString();
21            System.out.print ("Enter age of " + names[i] + ": ");
22            ages[i] = Keyboard.readInt();
23         }
24      }
25
26      public static int averageAge ()
27      {
28         int ave;
29         int tot = 0;
30
31         for (int i = 0; i<5; i++)
32            tot = tot + ages[i];
33
34         ave = tot / 5;
35
36         return ave;
37      }
38   }
```

17.5 Exercise

1. Consider the array below and then answer the questions.

	0	1	2	3	4	5
numbers	78	-32	9	66	-85	12

   a. What is the value of numbers[3]?
   b. What is the value of 4*numbers[2] + 8?
   c. What is the value of 5*numbers[2 + 3]?
   d. How will the array change after the following modifications?
      i. numbers[3] = 7;
      ii. numbers[0] = numbers[2] – numbers[5]
      iii. for (i=0, i<5, i++)
                numbers[i] = numbers[i]*2 – i;

2. Look at the following array and answer the questions below.

   String[] skiResorts = {

```
        "Whistler Blackcomb", "Squaw Valley", "Brighton",
        "Snowmass", "Sun Valley", "Taos"
    };
```

a. What is the index of Brighton?
b. Write an expression that refers to the string “Brighton” within the array.
c. What is the index of the last item in the array?
d. What is the value of the expression skiResorts[4]?

3. An array BinNum holds 8 integers. The value of each element is either 1 or 0 and they represent an 8 bit number. Write a program that first asks you to insert a binary number (bit by bit) in the array and then it converts the number to decimal.

4. Write a program that inserts random numbers from 1 to 10 in a 12-element array. The program then calculates how many odd and even numbers there are in the array.

5. Write a program that creates a 10-element (one-dimensional) array of integers. The program will present the following options for the user to choose:

    a. Fill array with numbers (inputted by the user)
    b. Display the array
    c. Find the total of the numbers in the array
    d. Find the average of the numbers in the array
    e. Add a number (given by the user) to all the elements of the array
    f. Shift the elements to the left
    g. Shift the elements to the right
    h. Rotate the elements (i.e. the last element swaps with the first etc.)
    i. Quit program

18 Compound Assignment Operators

Java allows you to combine assignment and other operators using a shorthand operator. For example i = i + 8 can be written as i += 8. This is called the addition assignment operator. Other shorthand operators are shown in the table below.

Operator	Name	Example	Equivalent
+=	Addition assignment	i += 5;	i = i + 5
-=	Subtraction assignment	j -= 10;	j = j -10;
*=	Multiplication assignment	k *= 2;	k = k * 2;
/=	Division assignment	x /= 10;	x =x / 10;
%=	Remainder assignment	a % =4;	a = a % 4;

19 The Unary Operators

The unary operators require only one operand; they perform various operations such as incrementing/decrementing a value by one, negating an expression, or inverting the value of a boolean.

+	Unary plus operator; indicates positive value (numbers are positive without this, however)
-	Unary minus operator; negates

an expression

++	Increment operator; increments a value by 1
--	Decrement operator; decrements a value by 1
!	Logical complement operator; inverts the value of a Boolean

20 The Math Class

The Math class contains a number of methods for calculations. The following table gives a description of some of the functions found in the Math class.

Method	Description
abs(d)	Returns the absolute value of the argument.
ceil(d)	Returns the smallest integer that is greater than or equal to the argument.
floor(d)	Returns the largest integer that is less than or equal to the argument.
rint(d)	Returns the integer that is closest in value to the argument.
round(d)	Returns the closest long or int, as indicated by the method's return type, to the argument.
min(arg1, arg2)	Returns the smaller of the two arguments.
max(arg1, arg2)	Returns the larger of the two arguments.
exp(d)	Returns the base of the natural logarithms, e, to the power of the argument.
log(d)	Returns the natural logarithm of the argument.
pow(base, exponent)	Returns the value of the first argument raised to the power of the second argument.
sqrt(d)	Returns the square root of the argument.
sin(d)	Returns the sine of the specified double value.
cos(d)	Returns the cosine of the specified double value.
tan(d)	Returns the tangent of the specified double value.
asin(d)	Returns the arcsine of the specified double value.
acos(d)	Returns the arccosine of the specified double value.
atan(d)	Returns the arctangent of the specified double value.
toDegrees(d) toRadians(d)	Converts the argument to degrees or radians.

The following program makes use of a number of functions found in the Math class.

Class that uses the Math class

```
1  public class UseMath
2  {
3      public static void main (String args[])
4      {
```

```
5         System.out.println ( Math.pow (2, 4) );
6         System.out.println ( (int) Math.pow (2, 4) );
7         System.out.println ( Math.random() );
8         System.out.println ( Math.random()*6 );
9         System.out.println ( Math.random()*6 + 1);
10        System.out.println ( (int) Math.random()*6 + 1);
11        System.out.println ( Math.ceil (7.3) );
12        System.out.println ( Math.ceil ( -7.3) );
13        System.out.println ( Math.floor (7.3) );
14        System.out.println ( Math.floor ( -7.3) );
15        System.out.println ( Math.round (7.3) );
16        System.out.println ( Math.round ( -7.3) );
17    }
18 }
```

21 Ternary Function

The Java ternary operator lets you assign a value to a variable based on a Boolean expression.

Example: minVal = (a < b) ? a : b;

In this code, if the variable a is less than b, minVal is assigned the value of a; otherwise, minVal is assigned the value of b. Note that the parentheses in this example are optional, so you can write that same statement like this: minVal = a < b ? a : b;

I think the parentheses make the code a little easier to read, but again, they're completely optional.

The general syntax of the ternary operator looks like this:

result = testCondition ? value1 : value2

22 Nested Statements

Nested statements are statements inside each other. We can have nested 'if statements' and also nested loops. The following program shows that we can also have nested ternary statements.

Class that uses nested ternary functions

```
1  public class TernaryFunction
2  {
3     public static void main (String args[])
4     {
5        int a = 5;
6        int b = 10;
7        int c = 8;
8
9        int max_a_b = (a < b) ? a : b ;
10
11       System.out.println (max_a_b);
12
13       int max_a_b_c = (a>b) ? (a>c) ? a : c : (b>c) ? b : c;
14
```

```
15          System.out.println (max_a_b_c);
16      }
17  }
```

23 Operators are not only Arithmetic

The following table shows different types of expressions and operators. Apart from the arithmetic operators there are character, string, Boolean and inequality operators.

Expression	Type of Expression	Operators	Operands	Evaluation
3 + 53 mod 4	integer	+, mod	3, 53, 4	6 (int)
4.2 + 3.1 * 4.0	float	+, *	4.2, 3.1, 4.0	16.6 (float)
c++ (where c='p')	character	++	c	'q' (char)
"wind" + "mill"	string	+ (concatenation not plus)	+	"windmill"
True And False	Boolean	And	True, False	False (Boolean)
9>8	inequality	>	9, 8	True (Boolean)

The following program makes use of different types operators.

Class that uses various operators and expressions

```
1   public class Operators
2   {
3       public static void main (String args[])
4       {
5           //integer expression
6           int x = 4 + 20 / 3;
7           System.out.println (x);
8
9           //real-number expression
10          double y = 7.8 + 9.2 * 8.3;
11          System.out.println (y);
12
13          //character expressions
14          char c = 'p' + 7;
15          System.out.println (c);
16          c = 'f' - 32;
17          System.out.println (c);
18          c = 'f' + 'g' - 50;
19          System.out.println (c);
20
21          //string expressions
22          String s = "fort" + "night";
23          System.out.println (s);
24          s = s + " is a period of 14 consecutive days";
25          System.out.println (s);
26
27          //Boolean expressions
28          Boolean r = true && false;
29          System.out.println (r);
30          r = true || true && false;
31          System.out.println (r);
32          r = true && (5 == 4 + 1);
```

```
33          System.out.println (r);
34
35          //inequality expressions
36          Boolean b = 10 > 58;
37          System.out.println (b);
38          b = (4 == 8 - 5) || ('f' > 't') && (false == true);
39          System.out.println (b);
40      }
41  }
```

24 The Switch Statement

The switch statement is used to avoid a number of nested if..then..else. Look at the following program.

Class that uses the switch statement

```
1   import java.util.*;
2
3   public class Digits
4   {
5       public static void main (String args[])
6       {
7           Scanner sc = new Scanner (System.in);
8           int dig;
9           System.out.println ("Enter a digit");
10          dig = sc.nextInt();
11
12          switch (dig) {
13              case 0: System.out.println ("ZERO");
14                      break;
15              case 1: System.out.println ("ONE");
16                      break;
17              case 2: System.out.println ("TWO");
18                      break;
19              case 3: System.out.println ("THREE");
20                      break;
21              case 4: System.out.println ("FOUR");
22                      break;
23              case 5: System.out.println ("FIVE");
24                      break;
25              case 6: System.out.println ("SIX");
26                      break;
27              case 7: System.out.println ("SEVEN");
28                      break;
29              case 8: System.out.println ("EIGHT");
30                      break;
31              case 9: System.out.println ("NINE");
32                      break;
33              default: System.out.println ("Invalid input");
34                       break;
35          }
36      }
37  }
```

The switch statement in the above program bases its choice on an integer but this type could also be a byte, short, char or string.

Note that after each choice there is a 'break' statement. If this is not included, all the options that follow the selected choice will be performed for example if 5 is entered then 6, 7, 8 and 9 will also be displayed.

25 Classes and Objects

The following program makes use of three classes. Have a look at it and then read the comments that follow it.

Class that defines a Customer

```
1  import java.util.*;
2
3  public class Customer
4  {
5      String ID;
6      String Name;
7      String Address;
8      String DOB;
9
10     Scanner sc = new Scanner (System.in);
11
12     void EnterCustomerData()
13     {
14         System.out.print ("Enter Customer ID: ");
15         ID = sc.next();
16         System.out.print ("Enter Customer Name: ");
17         Name = sc.next();
18         System.out.print ("Enter Customer Address: ");
19         Address = sc.next();
20         System.out.print ("Enter Customer Date of Birth (dd/mm/yyyy): ");
21         DOB = sc.next();
22     }
23
24     void ModifyCustomerData()
25     {
26         System.out.println ("The Customer ID is " + ID);
27         System.out.print ("Enter the new ID or press enter to leave unchanged: ");
28         String newID = sc.next();
29         if (!newID.equals(""))    ID = newID;
30
31         System.out.println ("The Customer Name is " + Name);
32         System.out.print ("Enter the new Name or press enter to leave unchanged: ");
33         String newName = sc.next();
34         if (!newName.equals("")) Name = newName;
35
36         System.out.println ("The Customer Address is " + Address);
37         System.out.print ("Enter the new Address or press enter to leave unchanged: ");
38         String newAddress = sc.next();
39         if (!newAddress.equals("")) Address = newAddress;
40
```

```
41          System.out.println ("The Customer DOB is " + DOB);
42          System.out.print ("Enter the new DOB (dd/mm/yyyy)or press enter to leave
                                                                    unchanged: ");
43          String newDOB = sc.next();
44          if (!newDOB.equals("")) DOB = newDOB;
45      }
46
47      void DisplayCustomerData()
48      {
49          System.out.println ("Customer ID: " + ID);
50          System.out.println ("Customer Name: " + Name);
51          System.out.println ("Customer Address: " + Address);
52          System.out.println ("Customer DOB: " + DOB);
53      }
54  }
```

Class that defines an Account

```
1   import java.util.*;
2
3   public class Account
4   {
5       Double Sum;
6       Double InterestRate;
7       String CustomerID;
8
9       void EnterAccountData()
10      {
11          Scanner sc = new Scanner (System.in);
12          Sum = 0.0;
13          System.out.print ("Enter Interest Rate: ");
14          InterestRate = sc.nextDouble();
15          System.out.print ("Enter Customer ID: ");
16          CustomerID = sc.next();
17      }
18
19      void Deposit(Double Amount)
20      {
21          Sum = Sum + Amount;
22          System.out.print ("Sum has been updated");
23      }
24
25      void Withdrawal(Double Amount)
26      {
27          Sum = Sum - Amount;
28          System.out.print ("Sum has been updated");
29      }
30
31      void AddInterest()
32      {
33          Sum = Sum * (1 + InterestRate/100);
34          System.out.print ("Interest has been added");
35      }
36  }
```

Class (for you to complete) called Bank that makes use of the classes Customer and Account

```
1  import java.util.*;
2
3  public class Bank
4  {
5     static Customer Customer1 = new Customer();
6     static Customer Customer2 = new Customer();
7     static Customer Customer3 = new Customer();
8     static Account Account1 = new Account();
9     static Account Account2 = new Account();
10    static Account Account3 = new Account();
11
12    public static void main(String[] args)
13    {
14       Scanner sc = new Scanner (System.in);
15       int choice;
16
17       do
18       {
19          DisplayMenu();
20          System.out.print ("Enter your choice: ");
21          choice = sc.nextInt();
22          switch (choice)
23          {
24          case 1: EnterInfo();
25                  break;
26          case 2: ModifyCustomer();
27                  break;
28          case 3: DepositMoney();
29                  break;
30          case 4: WithdrawMoney();
31                  break;
32          case 5: DispCustData();
33                  break;
34          case 6: DispAccData();
35                  break;
36          case 7: DispSumInBank ();
37                  break;
38          }
39       }
40       while (choice != 8);
41    }
42
43    public static void DisplayMenu()
44    {
45       System.out.println ("\f");
46       System.out.println ("MENU");
47       System.out.println ("1. Enter Information for Customers and Accounts");
48       System.out.println ("2. Modify data of a Customer");
49       System.out.println ("3. Deposit");
50       System.out.println ("4. Withdrawal");
51       System.out.println ("5. Display Customers Data");
```

```
52          System.out.println ("6. Display Accounts Data");
53          System.out.println ("7. Display the Sum of Money in the Bank");
54          System.out.println ("8. Quit the Program");
55          System.out.println ();
56      }
57
58      public static void EnterInfo()
59      {
60          System.out.println ("ENTERING DATA FOR FIRST CUSTOMER");
61          Customer1.EnterCustomerData();
62          System.out.println ();
63          System.out.println ("ENTERING DATA FOR FIRST ACCOUNT");
64          Account1.EnterAccountData();
65          System.out.println ();
66
67          System.out.println ("ENTERING DATA FOR SECOND CUSTOMER");
68          Customer2.EnterCustomerData();
69          System.out.println ();
70          System.out.println ("ENTERING DATA FOR SECOND ACCOUNT");
71          Account2.EnterAccountData();
72          System.out.println ();
73
74          System.out.println ("ENTERING DATA FOR THIRD CUSTOMER");
75          Customer3.EnterCustomerData();
76          System.out.println ();
77          System.out.println ("ENTERING DATA FOR THIRD ACCOUNT");
78          Account3.EnterAccountData();
79      }
80
81      public static void ModifyCustomer()
82      {
83      }
84
85      public static void DepositMoney()
86      {
87      }
88
89      public static void WithdrawMoney ()
90      {
91      }
92
93      public static void DispCustData ()
94      {
95      }
96
97      public static void DispAccData ()
98      {
99      }
100
101     public static void DispSumInBank ()
102     {
103     }
104 }
```

An object is made up of (1) Data, and (2) Functions (data and procedures; variables and methods; attributes and functions etc.). An object is defined by means of a class. The class Customer defines the bank user thus:

	Customer
Data	ID; Name; Address; DOB;
Functions	EnterCustomerData() ModifyCustomerData() void DisplayCustomerData()

The class Account can be represented thus:

	Account
Data	Sum; InterestRate; CustomerID;
Functions	EnterAccountData() Deposit(Double Amount) Withdrawal(Double Amount) AddInterest()

And finally the class that defines a Bank:

	Bank
Data	Customer1 Customer2 Customer3 Account1 Account2 Account3
Functions	main(String[] args) DisplayMenu() public static void EnterInfo() public static void ModifyCustomer() public static void DepositMoney() public static void WithdrawMoney () public static void DispCustData () public static void DispAccData () public static void DispSumInBank ()

26 Arraylists

An **Arraylist** is a dynamic data structure. It is dynamic because its size is not fixed and it changes according to the number of elements it holds. The array is a static data structure. A second important difference between an array and an arraylist is that the array holds elements of only one type while the arraylist can hold elements of different types.

However the elements of an arraylist are objects. We cannot store a primitive value in an arraylist. For example if we add an integer to an arraylist the added element will be of type Integer. An Integer is an object containing only one value which is of type int.

26.1 Some Basic Arraylist Functions

The following makes use of the following arraylist functions:

- **add** examples:
  - Assume al is an arraylist having the elements (“Mark”, 8, 3, “Stephanie”) then the statement al.add(-5) will add an element at the end of the arraylist. The new al is (“Mark”, 8, 3, “Stephanie”, -5)
  - Suppose we now perform al.add(1, “George”). This will add “George” at index 1 moving all the necessary elements by one position. The new arraylist al is now (“Mark”, “George”, 8, 3, “Stephanie”, -5).

- **remove** examples:
  - continuing on the previous arraylist, al.remove(2) will eliminate the element with index 2. So the new formed al will consist of (“Mark”, “George”, 3, “Stephanie”, -5).
  - al.remove(“Stephanie”) will remove the first occurance of “Stephanie” (i.e. it will only remove the value with the least index). So now al will now be (“Mark”, “George”, 3, -5).

- **size**
  - This is a function that gives you the number of elements in an arraylist. See line 7 in the following program.

- **get**
  - This function is used to read an element from an arraylist. See line 10.

Class that uses and ArrayList

```
1  import java.util.ArrayList;
2
3  public class AboutArraylists
4  {
5     static void ViewArraylist (ArrayList al)
6     {
7        int l = al.size();
8
9        for (int i = 0; i<l; i++)
10          System.out.print (al.get(i) + " ");
11       System.out.println();
12    }
13
14    static void main (String[ ] args)
15    {
16       ArrayList ListTest = new ArrayList();
17
18       ListTest.add ("James");
19       ListTest.add ("Anne");
20       ListTest.add ("Isabel");
21       ListTest.add (7);
```

```
22
23          ViewArraylist (ListTest);
24
25          ListTest.remove(2);
26          ViewArraylist (ListTest);
27
28          ListTest.add ("Paul");
29          ViewArraylist (ListTest);
30
31          ListTest.add (1, "Paul");
32          ViewArraylist (ListTest);
33
34          ListTest.remove ("Anne");
35          ViewArraylist (ListTest);
36
37          ListTest.remove ("Paul");
38          ViewArraylist (ListTest);
39
40          int x;
41          x = (int) ListTest.get(1);
42          System.out.println (x+5);
43      }
44  }
```

26.2 Some Other Functions

- void **clear**(): Removes all of the elements from this list.
- boolean **contains**(Object o): Returns true if this list contains the specified element. More formally, returns true if and only if this list contains at least one element e such that (o==null ? e==null : o.equals(e)).
- int **indexOf**(Object o): Returns the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.
- int **lastIndexOf**(Object o): Returns the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.
- Object **set**(int index, Object element): Replaces the element at the specified position in this list with the specified element. Throws IndexOutOfBoundsException if the specified index is is out of range (index < 0 || index >= size()).

26.3 Uni-type Arraylists

The following program shows how it is possible to create an arraylist that accepts only elements of one type.

Class that uses arraylists with elements of the same type

```
1  import java.util.ArrayList;
2
3  public class ArrayListOther
4  {
5      public static void main (String[] args)
6      {
7          ArrayList<String> members = new ArrayList<String>();
```

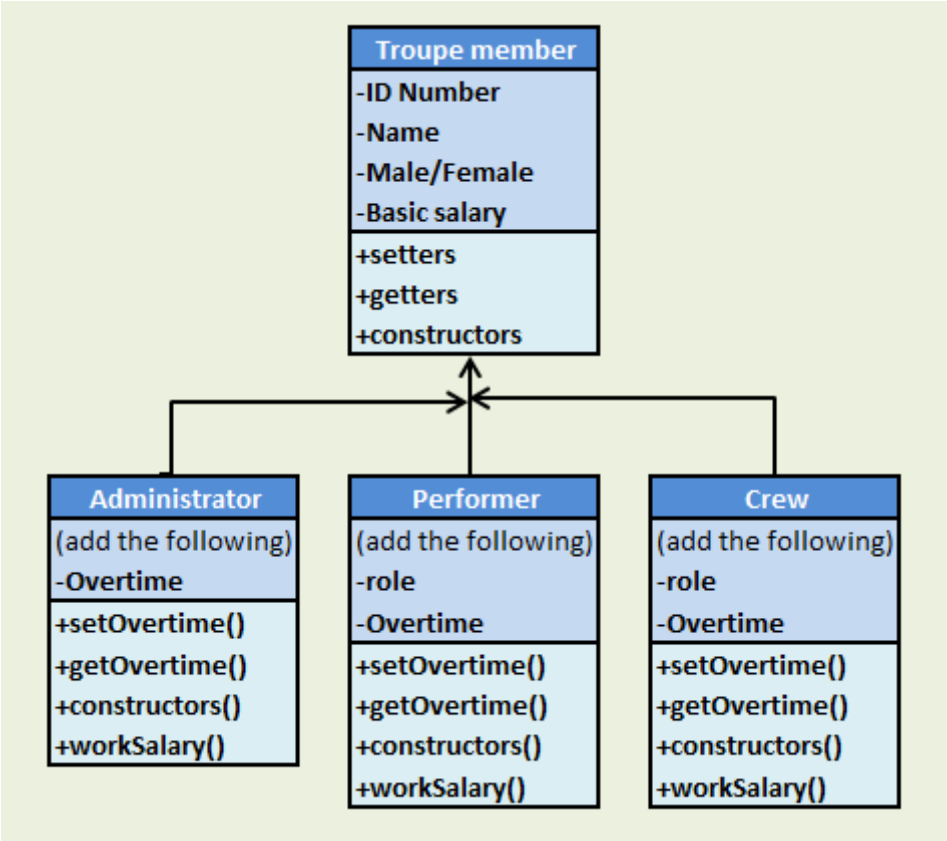
```
8
9           members.add("Graham");
10          // the statement members.add(8) is not accepted
11
12          ArrayList<Integer> memNum = new ArrayList<Integer>();
13          double r = 3.6;
14          memNum.add(7);
15          // the statement memNum.add("Java") is not accepted
16          // the statement memNum.add(3.6) is not accepted
17          memNum.add( (int) r );
18
19          int i;
20          i = memNum.get(0);
21
22          ArrayList<Double> ald = new ArrayList<Double>();
23          ArrayList<Boolean> alb = new ArrayList <Boolean>();
24      }
25  }
```

26.4 Exercise

1) Write a method that given a string and a string arraylist as parameters it adds the number of occurancies of the string in the arraylist.

2) Write a method that given a string and a string arraylist as parameters deletes all occurancies of the string in the arraylist.

27 Inheritance



Inheritance occurs when a class is derived from another class. As an example consider the class diagram shown above. This diagram is implemented in the following four classes.

Class TroupeMember

```
1  public class TroupeMember
2  {
3     private int IDnum;
4     private String name;
5     private Boolean female;
6     private double basicSal;
7
8     public void TroupeMember (int id, String n, Boolean f, double basSal)
9     {
10       IDnum = id;
11       name = n;
12       female = f;
13       basicSal = basSal;
14    }
15
16    public void setIDnum (int ide)
17    {
18       IDnum = ide;
19    }
20
21    public void setName (String nm)
22    {
23       name = nm;
24    }
25
26    public void setFemale (Boolean fm)
27    {
28       female = fm;
29    }
30
31    public void setBasicSalary (double bs)
32    {
33       basicSal = bs;
34    }
35
36    public int getIDnum()
37    {
38       return IDnum;
39    }
40
41    public String getName()
42    {
43       return name;
44    }
45
46    public Boolean getFemale()
47    {
48       return female;
49    }
```

```
50
51      public double getBasicSalary()
52      {
53          return basicSal;
54      }
55
56      public double calcSalary()
57      {
58          //method to be overridden;
59          return 1.3;
60      }
61  }
```

Class Administrator

```
1   public class Administrator extends TroupeMember
2   {
3      private int overtime;
4
5      public void Administrator (int id, String n, Boolean fem, double bs, int ov)
6      {
7          overtime = ov;
8          super.setIDnum (id);
9          super.setName (n);
10         super.setFemale (fem);
11         super.setBasicSalary (bs);
12     }
13
14     public void setOvertime (int ot)
15     {
16         overtime = ot;
17     }
18
19     public int getOvertime()
20     {
21         return overtime;
22     }
23
24     public double calcSalary ()
25     {
26         return (super.getBasicSalary()*(1 + overtime/160.0));
27     }
28  }
```

Class Performer

```
1   public class Performer extends TroupeMember
2   {
3      private String role;
4      private int overtime;
5
6      public void Performer (int id, String n, Boolean fem, double bs, int ov, String rl)
7      {
8         role = rl;
```

```
9         overtime = ov;
10        super.setIDnum (id);
11        super.setName (n);
12        super.setFemale (fem);
13        super.setBasicSalary (bs);
14     }
15
16     public void setRole (String rl)
17     {
18        role = rl;
19     }
20
21     public void setOvertime (int ot)
22     {
23        overtime = ot;
24     }
25
26     public String getRole()
27     {
28        return role;
29     }
30
31     public int getOvertime ()
32     {
33        return overtime;
34     }
35
36     public double calcSalary ()
37     {
38        double sal = 0;
39        if (role.equals("actor"))sal = (super.getBasicSalary()*(1 + overtime/160.0));
40        if (role.equals("musician")) sal = (super.getBasicSalary()*(1 + overtime/140.0));
41        return sal;
42     }
43  }
```

Class Crew

```
1   public class Crew extends TroupeMember
2   {
3      private String role;
4      private int overtime;
5
6      public void Crew (int id, String n, Boolean fem, double bs, int ov, String rl)
7      {
8        role = rl;
9        overtime = ov;
10       super.setIDnum (id);
11       super.setName (n);
12       super.setFemale (fem);
13       super.setBasicSalary (bs);
14     }
15
16     public void setRole (String rl)
```

```
17      {
18          role = rl;
19      }
20
21      public void setOvertime (int ot)
22      {
23          overtime = ot;
24      }
25
26      public String getRole()
27      {
28          return role;
29      }
30
31      public int getOvertime ()
32      {
33          return overtime;
34      }
35
36      public double calcSalary ()
37      {
38          double sal = 0;
39
40          if (role.equals("costume")) sal = (super.getBasicSalary()*(1 + overtime/140.0));
41          else if (role.equals("light")||role.equals("sound")) sal = (super.getBasicSalary()*(1
42  + overtime/150.0));
43          else if (role.equals("stage")) sal = (super.getBasicSalary()*(1 + overtime/160.0));
44
45          return sal;
46      }
47  }
```

28 Polymorphism

In the classes above the method calcSalary() is defined differently in each class. This is called polymorphism which means that this particular method takes different forms in different objects.

The following class makes use of the four classes seen above.

Class TroupeManagement

```
1   import java.util.ArrayList;
2
3   public class TroupeManagement
4   {
5       public static void main (String[] args)
6       {
7           TroupeMember troupe[] = new TroupeMember [5];
8
9           Administrator admin = new Administrator();
10          admin.Administrator (658, "George", false, 1365.5, 8);
11          troupe[0] = admin;
12
```

```
13          Performer actor1 = new Performer();
14          actor1.Performer (12, "Jacqueline", true, 1220, 2, "actor");
15          troupe[1] = actor1;
16
17          Performer mus1 = new Performer();
18          mus1.Performer (158, "Anne", true, 1055, 3, "musician");
19          troupe[2] = mus1;
20
21          Crew costume1 = new Crew();
22          costume1.Crew (78, "James", false, 965, 3, "costume");
23          troupe[3] = costume1;
24
25          for (int i=0; i<4; i++)
26          {
27              System.out.println (troupe[i].calcSalary());
28          }
29      }
30  }
```

29 Text files

A text file is one in which one can only write text; no numbers, no Boolean values etc.

Class TextFileWritingTo

```
1   import java.io.*;
2
3   public class TextFileWritingTo
4   {
5       public static void main ( String[] args)
6       {
7         try
8           {
9             FileWriter sen_fw = new FileWriter ("sentences.txt");
10            PrintWriter sen_pw = new PrintWriter (sen_fw);
11            sen_pw.println ("first sentence");
12            sen_pw.println ("second sentence");
13            sen_pw.println ("third sentence");
14            sen_pw.close();
15          }
16        catch (IOException e)
17          {
18             System.out.println ("There was a problem writing in the file");
19          }
20      }
21  }
```

Class TextFileReadingFrom

```
1   import java.io.*;
2
3   public class TextFileReadingFrom
4   {
5       public static void main ( String[] args)
```

```
6     {
7       try
8         {
9           FileReader sen_fr = new FileReader ("sentences.txt");
10          BufferedReader sen_br = new BufferedReader (sen_fr);
11          String s = sen_br.readLine();
12          while (s != null)
13          {
14            System.out.println (s);
15            s = sen_br.readLine();
16          }
17          sen_br.close();
18        }
19      catch (IOException e)
20        {
21           System.out.println ("There was a problem reading from the file");
22        }
23    }
24  }
```

30 Binary files

Class BinaryFileWritingTo

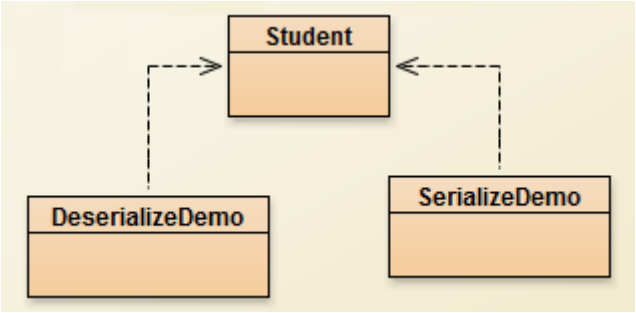
```
1  import java.io.*;
2
3  public class BinaryFileWritingTo
4  {
5    public static void main (String[] args)
6    {
7      try
8      {
9        FileOutputStream fos_var = new FileOutputStream ("binfile.bin");
10       DataOutputStream dos_var = new DataOutputStream (fos_var);
11       dos_var.writeInt (8);
12       dos_var.writeDouble (3.58);
13       dos_var.writeChar ('k');
14       dos_var.writeUTF ("Java");
15       dos_var.close();
16     }
17     catch (IOException e)
18     {
19       System.out.println ("There was a problem writing in the file");
20     }
21   }
22 }
```

Class BinaryFileReadingFrom

```
1  import java.io.*;
2
3  public class BinaryFileReadingFrom
4  {
5    public static void main (String[] args)
```

```
6     {
7        int i;
8        double d;
9        char c;
10       String s;
11
12       try
13       {
14          FileInputStream fis_var = new FileInputStream ("binfile.bin");
15          DataInputStream dis_var = new DataInputStream (fis_var);
16          i = dis_var.readInt();
17          d = dis_var.readDouble();
18          c = dis_var.readChar();
19          s = dis_var.readUTF();
20          dis_var.close();
21          System.out.println (i);
22          System.out.println (d);
23          System.out.println (c);
24          System.out.println (s);
25       }
26       catch (IOException e)
27       {
28          System.out.println ("There was a problem writing in the file");
29       }
30    }
31 }
```

31 Object serialization



Class Student

```
1  public class Student implements java.io.Serializable
2  {
3     public int IdNo;
4     public String Name;
5     public String Address;
6
7     int[] Marks = new int[3];
8  }
```

Class SerializeDemo

```
1  import java.io.*;
2
```

```
 3  public class SerializeDemo
 4  {
 5     public static void main(String [] args)
 6    {
 7       Student stud = new Student();
 8       stud.IdNo = 2586;
 9       stud.Name = "Sandy Bright";
10       stud.Address = "55, Top Hill Road, Zurrieq";
11       stud.Marks[0] = 87;
12       stud.Marks[1] = 57;
13       stud.Marks[2] = 66;
14
15       try
16       {
17          FileOutputStream fileOut =
18             new FileOutputStream("employee.ser");
19          ObjectOutputStream out =
20             new ObjectOutputStream(fileOut);
21          out.writeObject(stud);
22          out.close();
23          fileOut.close();
24       }catch(IOException i)
25       {
26           i.printStackTrace();
27       }
28    }
29 }
```

Class DeserializeDemo

```
 1  import java.io.*;
 2
 3  public class DeserializeDemo
 4  {
 5    public static void main(String [] args)
 6    {
 7       Student stu = null;
 8       try
 9       {
10          FileInputStream fileIn =
11             new FileInputStream("student.ser");
12          ObjectInputStream in = new ObjectInputStream(fileIn);
13          stu = (Student) in.readObject();
14          in.close();
15          fileIn.close();
16       }catch(IOException i)
17       {
18          i.printStackTrace();
19          return;
20       }catch(ClassNotFoundException c)
21       {
22          System.out.println("Student class not found");
23          c.printStackTrace();
24          return;
```

```
25        }
26
27        System.out.println ("Deserialized Student...");
28        System.out.println ("Id No: " + stu.IdNo);
29        System.out.println ("Name: " + stu.Name);
30        System.out.println ("Address: " + stu.Address);
31        System.out.println ("Mark 1: " + stu.Marks[0]);
32        System.out.println ("Mark 2: " + stu.Marks[1]);
33        System.out.println ("Mark 3: " + stu.Marks[2]);
34    }
35 }
```