Java Fundamentals

1.1 Programming Approach from Procedural to Object Orientation (OO) Methodologies

- **Programming** is to give the machine a list of steps to perform a particular task.
- If the system to which the programming is done is a computer than it is called as **Computer Programming**.
- The programming of any system has to be done in the language understood by that system.
- A digital system like computer understands only binary language (which consists only of 0s and 1s), also called as **machine language**. But, programming in machine language is almost impossible for a human being. Hence, the manufacturers of the processor develop a language called as **assembly language**. Assembly language is simpler than the machine language, but making a huge software using this is again very difficult. It includes the following :
- An English language word that specifies the operation to be performed and
- The operands, which are the data on which the operation is to be performed.
- Machine language and assembly language are called as low level languages.
- To remove the complexity of programming, which was there because of low level languages, High level languages (HLL) were developed. HLL are simplest for programming the processors.
- Some of these languages are FORTRAN, BASIC, COBOL, C, C++, JAVA, .net, etc.
- HLL programming languages like C / C++ / Java are structured programming languages and hence are quite easy for the programmers. C / C++ are sometimes also referred to as middle level languages as they are not fully high level languages and neither are they low level languages.
- In this subject we have to learn the programming language Java. Also we have to make some small programs using this language.
- Procedure oriented programming gives significance to procedure i.e. how to do a task.
- The structure of a procedure oriented programming as shown in Fig. 1.1.1, is made up of functions. If a variable is to be accessed by multiple functions then such a variable is known as global variable.

- The structure of object oriented programs is shown in Fig. 1.1.2.

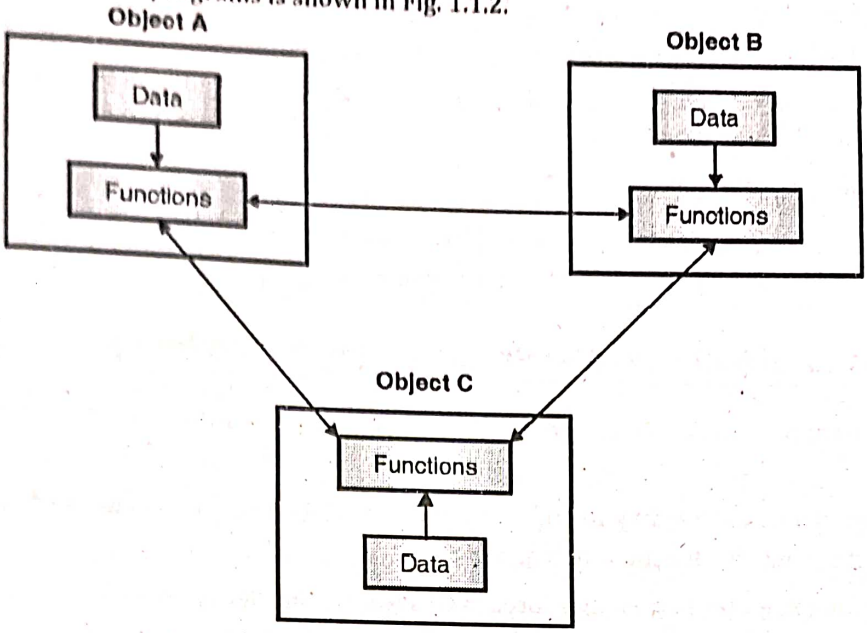


Fig. 1.1.2 : Structure of an object oriented program

1.2 Comparison of C++ and Java

Sr. No.	C++	Java
1.	C++ is object oriented programming but not a purely object oriented programming language. A program can be written in C++ even without using classes and objects.	Java is a purely object oriented programming; no program can be written without classes and objects.
2.	C++ has a goto statement. But this statement makes the program system dependent.	To avoid this dependency of system, goto statement is not available in Java.
3.	Pointers are also allowed in C++, and it also makes the program system dependent.	Pointers are not there in Java to avoid platform dependency.
4.	Multiple and Hybrid inheritance are possible in C++.	Multiple and Hybrid inheritance are not in Java. A slight implementation of the same can be done with a special tool called as Interface.
5.	Operator overloading is possible in C++ programming.	Operator overloading is not allowed in Java programming.
6.	We include a header file in C++ that consumes a lot of memory space for even those objects that are not used in the program.	In Java we import class files, which do not consume memory. As we know memory space is required for objects and not classes. Hence until we need an object, we will not create it and hence memory space is not wasted.
7.	In C++ we had three access specifiers namely: public, protected and private.	In Java we have five access specifiers namely: public, protected, private, default and private protected. Although private protected is not available in recent versions of Java.
8.	We have destructors in C++.	Java is said to be garbage collected i.e. the objects are automatically destroyed once their use is over.

Sr. No.	C++	Java
9.	C++ doesn't have exception handling. In case of any errors, the compiler cannot handle it.	Java has a powerful exception (error) handling system.
10.	C++ doesn't support multi threading.	Java supports multi threading.
11.	C++ cannot be used on internet for making applets.	Java can be used on internet by making applets and hence having dynamic applications.

1.3 Introduction to Object Oriented Programming Methodology

- Object oriented programming as the name says gives more significance to the objects which has data and functions built around it.
- The data of the object can be accessed by the functions associated with it. The functions of one object can access the data of another object through the functions of that object.
- Object oriented programming uses bottom up approach wherein the smaller tasks are first dealt in detail and gradually creating the entire huge system.
- Object oriented programming on the other hand gives more importance to data rather than a procedure.
- The structure of object oriented programs is shown in Fig. 1.3.1.

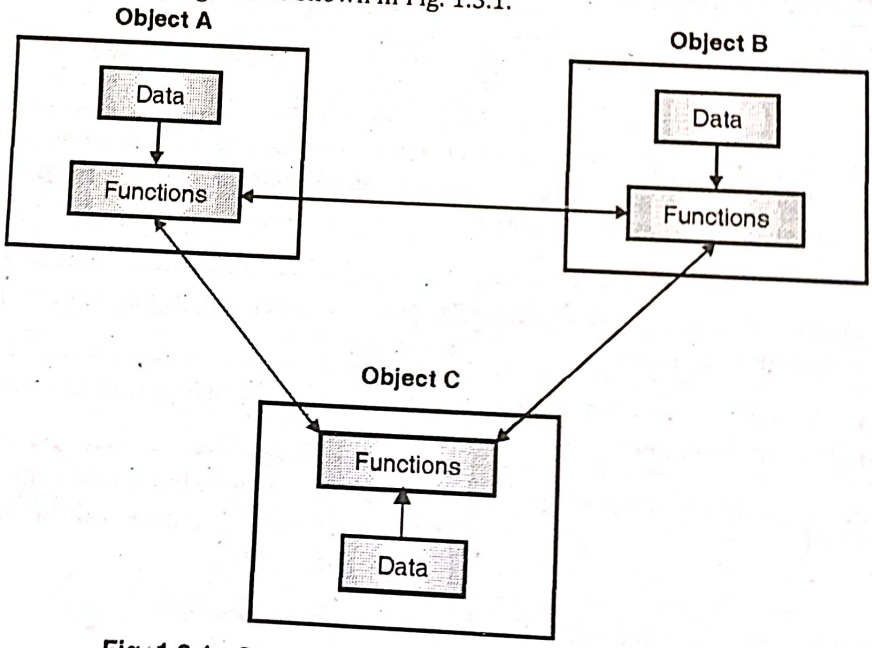


Fig. 1.3.1 : Structure of an object oriented program

1.4 Features of Object Oriented Programming (OOP)

Object oriented programming gives significance to objects or data rather than the procedure. Let us see the features of OOP.

1. More emphasis is given to data rather than procedure. It is seen in the real world problems that the data or the objects are more important than the procedure or the method to perform a task. Hence, in object oriented programming, object is given more importance compared to the procedure of doing it.

2. Programs are divided into objects as shown in Fig. 1.3.1. As shown in Fig. 1.3.1, the objects contain data and functions required to access them. Thus, in an object oriented program, you will notice the objects as shown in the structure.
3. Data structures are designed such that they characterize the object i.e. all the information required for an object are stored in the variables of that object.
4. Functions that operate on the data of an object are tied together in data structures i.e. the functions that operate on a data are associated with them as shown in Fig. 1.3.1.
5. Data is hidden or cannot be access by external function. The data of an object can be accessed only by the functions of the same object.
6. Objects communicate with each other through functions. If a function of an object wants the data of another object then it can be accessed only through the functions.
7. New data and functions are easily added when required. Whenever new data is to be added, all the functions need not be changed; only that functions are to be changed, which require to access the data.
8. It follows a bottom-up approach. In this type of approach, each element is first specified in detail and then the entire system is made using these elements. You will notice in the programming of C++, that this approach of bottom-up approach makes the programming very simple.

1.5 Important Terminologies for Object Oriented Programming

There are few important terms related to Object Oriented Programming that we need to understand. First the concept of objects and classes and then the specialties of OOPs viz. Data Abstraction, Encapsulation, Inheritance, Polymorphism, etc. Let us understand them one by one.

1. **Class and objects :** It is a type or a category of things. It is similar to a structure with the difference that it can also have functions besides data items. A structure, we have seen, can have only data variables but a class can have data members as well as function members.
2. **Object :** It is an instance or example of a class. You can imagine it to be similar to a variable of class like we have a variable of a structure.
3. **Data abstraction :** Data abstraction is like defining or abstracting the object according to the required parameters. For example; if there is a class for circle we need to just define the radius of the object of this class. We need not bother about anything else of that object.
4. **Data encapsulation or data hiding :** The data of an object is hidden from other objects. This is called as encapsulation. Encapsulation is achieved by putting data and functions associated with it into a class.
5. **Inheritance :** The mechanism of deriving the properties of one class into another class is known as inheritance. We will see in detail about this concept in a special section dedicated on Inheritance.
6. **Polymorphism :** Poly refers to multiple and morph refers to different forms. Hence, polymorphism means multiple forms of the same thing. This topic will also be covered in detail in the later sections
7. **Message Communication :** The communication of the objects to pass messages is done by the functions or the methods in the class. This is shown in the Fig. 1.3.1. The message or the data required by one object can be taken from the other using the function of that class.
8. **Reuse :** The main advantage of the object oriented (OO) system is the reusability of the code. The methods or functions written in a class can be used by all the objects of tat class. Hence there is no need to write separate functions for each of the objects.

9. **Coupling and Cohesion :** Coupling can be defined as the strength of the relationships between various modules in object oriented system, whereas cohesion can be defined as to how the elements making up a module are related. Cohesion measures the internal design while coupling measures the design interface. To minimize the complexity of an application, it is divided into modules such that there is high cohesion and low coupling.

   Examples of cohesion :

   1. Subtract 3 from 1st argument
   2. Type caste the 9th argument to char
   3. Reverse string of characters in another argument
   4. Print next line

   Examples of coupling

   1. A component directly modifies another's data
   2. A component modifies another's code, with the help of jumps (goto) into the middle of a routine.

10. **Sufficiency, Completeness and Primitiveness :** Sufficiency as the name refers to the satisfaction of the minimum requirements. In this case the user should be able to find the minimum required methods in the class. There should not be any extra methods. It can be said that sufficiency refers to keeping the class as simple and focused as possible. Completeness is a measure for the satisfaction of the necessary requirements. It can be said that it gives the users a class of services they expect. The users tend to assume the services required from a class based on the name and semantics of the class. Primitiveness is a check for completeness. The methods should be designed to offer a single primitive and unique operation. There shouldn't be multiple methods to perform the same operation. Hence there should be minimum and simple set of methods to implement the behavior of the class. According to Booch for well formed objects there should be:

    1. *High cohesion* 2. Low coupling
    3. *Sufficiency* 4. Completeness
    5. *Primitiveness*

11. **Meta Class :** A class whose instances are not objects instead they are again classes are called as Meta class. Hence the meta class has its instances as classes. A meta class is used to define the behavior of a group of classes and their instances.

1.6 Java Evolution : History

- A team from Sun Microsystems, Inc. was working on a project wherein a team member James Gosling was assigned a work of identifying the programming language for their project.
- They couldn't find a single programming language suited for their project. They decided to make a new programming language and called it as "oak".
- They later found that this name was a registered trademark of some other company, hence they decided to rename it and finally named it as "Java", which is a name of an island.
- The team had consumed a lot of coffee during the making of this programming language. The island "Java" cultivated a coffee named as "Javanese" which had gained global popularity in the 17th century. And hence the name "Java" was given as a synonym for coffee.

1.7 Features of Java

Java has various features as listed below :

1.	Simple	2.	Purely object-oriented
3.	Platform independent	4.	Distributed
5.	Dynamic	6.	Multi Threaded
7.	Garbage collected	8.	Robust
9.	Secure	10.	Interpreted
11.	Portable	12.	High performance
13.	Scalability		

Let us discuss in detail each of these features of Java

1. **Simple :** Java is very simple especially compared to languages like C, C++. Java doesn't allow pointers, which was a complicated concept in C/C++. Java also doesn't allow operator overloading, goto statement etc. Besides the syntax rules of Java are kept almost similar to that of C/C++, hence allowing easy migration for the C/C++ programmers.
2. **Purely object-oriented :** Java is a purely object oriented language means no program can be written without a class. An object oriented programming language needs to necessarily have classes and object. A class is a type of objects and an object is an instance of class. We will be seeing more about these in the subsequent chapters.
3. **Platform Independent :** This is the most important feature of Java. This was the main feature what James Gosling wanted from the programming language to be used in their software. Platform Independent means a program made for one platform will also work on any other platform. A platform is a combination of a system (computer), operating system and other packages with that system. For example, a Pentium based system with Windows, or a Pentium based system with Linux, or a AMD based system with Windows etc. The feature "Platform Independent" ensures that the program or software developed in the Java programming language can work on any of the above said or some other platform. This was not possible in earlier programming languages like C/C++. This is possible with Java because of a special tool called as Java Virtual Machine (JVM). We will discuss in detail about JVM in the subsequent section.
4. **Distributed :** This feature of Java was found to be very useful. Because of being platform independent, a java based program can be distributed over internet and hence run on any platform (system) in the world. This feature of being distributed of Java has made many things possible over the Internet.
5. **Dynamic :** We have seen what is meant by dynamic and static in C++. Those things that happen during the compile time are called as static while those things that happen during the execution time are called as dynamic. Java has more dynamic features compared to C++. We will see the dynamic features of Java during the later chapters.
6. **Multi-Threaded :** This is another new thing implemented in Java. A thread is one function in a program or a process. We can have multiple threads that can run concurrently with the same program. There is a time sharing implemented to share the processor resource for multiple threads on a time scale i.e. one thread runs for some time then another thread for some time and so on multiple threads can run concurrently wherein any one of the thread is being executed by the processor at any given time.
7. **Garbage collected :** An object which is no more in use is automatically thrown in garbage by JVM. Thus the memory allocated to an object is automatically made free and available for other variables or objects. This removes the requirement of destructor as it was required in C++. In C++ the memory allocated to an object was to be made free by a destructor. In Java it is automatically done, hence garbage collected.

8. **Robust :** Java has a very good inbuilt exception handling capacity that makes a Java program robust. Also, Java does not keeps the authority of memory allocation and de-allocation with the programmer. JVM automatically allocates and frees up the memory allocated for an object. Hence a software made on Java will not crash so easily as it would do on a C/C++ based software.
9. **Secure :** Security issues like virus attack are less possible on a Java based software.
10. **Interpreted :** Java programs are said to be interpreted. Java programs are converted into a special byte code. This byte code is interpreted by the JVM. JVM interprets the byte code generated by the Java compiler. By interprets, it means that the JVM converts the byte code into the machine understandable code.
11. **Portable :** A program written on Java is portable i.e. it can be carried over anywhere on any computer. This is because the Java program is platform independent and hence can work on any computer.
12. **High Performance :** Earlier there was a problem with JVM. The JVM has to interpret the Java program and then execute the same. This use to take more time i.e. first convert from byte code to machine code, then execute it. Later Java people introduced a special tool in JVM called as Just In Time (JIT) compiler. This increased the speed and hence giving high performance.
13. **Scalability :** Java based programs are scalable over a huge number of systems besides computer. The team of Sun Microsystems Inc. had mainly developed this programming language tool for programming the systems used in consumer electronics like washing machine, microwave oven, etc. Hence this programming language can be used even in such independent systems and not just computers. Thus Java is said to be scalable over all systems.

1.8 Java Virtual Machine (JVM)

- JVM is an interpreter. JVM is the main component of Java programming language that makes Java to be platform independent and gives the language so many advantages over other programming languages.
- The main feature of Java i.e. platform independent is because this byte code file can be carried onto any system and the JVM residing in the system will interpret or translate the byte code into the machine understandable code of that machine.
- The java program written has to be stored with the ".java" extension. The source code file is the program file where you have your code i.e. your program.
- When it says that the Java program is portable or architectural neutral, it is because of this byte code that can be carried on any system.

> **Note :** The source code is never distributed. If the source code is distributed then the customer can do the needful changes by himself and will never come back to the software developer. Hence a software developer should never give away the source code developed by him. In C++ programs, the developer had to go and install the software on the customer's system; and sometimes also make some changes in the code as required.

- When this code is compiled by the Java compiler i.e. "javac", it converts the java program into a special format called as byte code. The byte code as the name says is a coded language with each information in a packet of 8 binary digits i.e. byte. This byte code file is stored as a ".class" file i.e. a file with the extension of ".class".
- The components of JVM are shown in the block diagram of JVM (Fig. 1.8.1).

- The different tokens of Java are its

o character set	o keywords
o identifiers	o constants and variables
o data types	o operators

- We will see these tokens in the subsequent sub-sections.

1.9.1 Character Set of Java

- The character set of any programming language indicates the different characters the program can contain. Character set includes all the alphabets, digits and special symbols supported by the processor.
- When we will be writing Java programs, all these will be found in our program. Table 1.9.1 gives a list of Java character set.

Table 1.9.1 : Character Set of Java

Sr. No.	Characters	List included
1.	Alphabets (Upper case and lower case)	A, B, C ..... Z a, b, c, ... z
2.	Digits (numbers)	0, 1, 2, ... 9
3.	Special symbols (all those seen on a keyboard, nothing besides that)	< > { } ( ) [ ] , . ; : ! ? ' " / + * = % & # @ | \ ~ ` $.^ _ - (The names used for these symbols are given in the Table 1.9.2)
4.	Other special characters	Blank Space, Tab, Carriage Return (Enter Key)

Table 1.9.2 : Special Symbols and their names

Symbol	Name of Symbol	Symbol	Name of Symbol
,	Comma	&	Ampersand
.	Period or dot	*	Asterisk
;	Semicolon	-	Minus Sign
:	Colon	+	Plus Sign
?	Question Mark	<	Opening Angle (Less than sign)
!	Exclamation Mark	>	Closing Angle (Greater than sign)
|	Pipe	(	Left Parenthesis
/	Forward Slash	)	Right Parenthesis
\	Back Slash	[	Left Square Bracket
'	Single Quotes	]	Right Square Bracket
"	Double Quotes	{	Left Brace
~	Tilde	}	Right Brace
_	Underscore	#	Hash
^	Caret	$	Dollar
%	Percentage		

- *Each of these special symbols may have some significance which will be discussed in the forthcoming chapters.*

1.9.2 Keywords

- These are some special words that have a predefined meaning for the Java compiler. Hence, these words cannot be used as identifiers (identifiers are discussed in Section 1.9.3).
- These are a set of words which are reserved for the certain operations and hence are also sometimes referred as reserved words.
- All keywords are in lower case.
- The keywords used in Java are as given in Table 1.9.3.

Table 1.9.3

abstract	default	if	private	this
boolean	do	implements	protected	throw
break	double	import	public	throws
byte	else	instanceof	return	transient
case	extends	int	short	try
catch	final	interface	static	void
char	finally	long	strictfp	volatile
class	float	native	super	while
const	for	new	switch	
continue	goto	package	synchronized	

- These keywords will be used in different places in programming. Their significance and use will be studied wherever required in the forthcoming chapters.

1.9.3 Identifiers

Identifiers are names given to different user defined things like variables, constants, functions, classes, objects, structures, unions, etc. While making these identifiers we need to follow some rules. These rules are stated below :

1. The identifier can consist of alphabets, digits and two special symbols i.e. '_' (underscore) and '$' (dollar sign).
2. An identifier cannot start with a digit. It can start either with an alphabet or underscore or dollar sign.
3. It cannot contain any special symbol except underscore and dollar sign. Blank spaces are also not allowed.
4. It cannot be a keyword.
5. It is case sensitive i.e. an alphabet capital in one identifier with same name in another identifier with that alphabet small case will be considered different (for more details see examples in this section).

Ex. 1.9.1 : A list of valid and invalid identifiers is given below with reasons wherever required.

1. simple_interest
2. char
3. 3friends
4. _3$friends
5. Simple interest
6. #3friends
7. void
8. Void
9. $xyz

Soln. :

1. **simple_interest :** Valid
2. **char :** Invalid, because it is a keyword
3. **3friends :** Invalid, because starts with a digit
4. **_3$friends :** Valid
5. **Simple interest :** Invalid, because blank spaces are not allowed
6. **#3friends :** Invalid, because no special symbol except underscore is allowed.
7. **void :** Invalid, because keyword not allowed.
8. **Void :** Valid, case sensitive.
9. **$xyz :** Valid

1.9.4 Data Types

- The data type decides the type of data and the memory locations required for storing that type of data in the memory.
- The data types of Java can be divided into three types: Primitive, Derived and User defined data types.
- The different primitive types of data that can be used in Java are integer, character, fraction type numbers, etc.
- Table 1.9.4 shows the different primitive data types and the memory space required for storing them.

Table 1.9.4 : Data types

Sr. No.	Data type	Default Value	Range	Space required in memory
1.	byte	0	– 128 to 127	1
2.	short	0	– 32768 to 32767	2
3.	int	0	– 2147483648 to 2147483647	4
4.	long	0L	– 9223372036854775808 to 9223372036854775807	8
5.	float	0.0f	– 3.4e38 to 3.4e38	4
6.	double	0.0d	– 1.8e308 to 1.8e308	8
7.	char	'\u0000'	0 to 65535	2
8.	String	Null		
9.	boolean	false		

Ex. 1.9.2 : *Some examples of data to be stored are listed below with the data types that will be best suited for them*

1. *Age in years*
2. *Rate of interest*
3. *Alphabet*
4. *Principal amount*
5. *Runs made by a batsman*
6. *Factorial of a number*
7. *Radius of a circle*
8. *Area of a circle*
9. *User input as 'true' or 'false'*

Soln. :

1. **Age in years :** int type data must be used, as years is an integer value.
2. **Rate of interest :** float data must be used, as rate of interest is a fraction type value.
3. **Alphabet :** char type of data is to be used, as a character is to be used.
4. **Principal amount :** float type data must be used, as principal is in rupee and paise.
5. **Runs made by a batsman :** int type data must be used, as no. of runs will always be integer.
6. **Factorial of a number :** int type data must be used, as the factorial is always an integer.
7. **Radius of a circle :** float type data must be used, as radius will mostly be a fraction number.
8. **Area of a circle :** float type data must be used, as area is $(radius)^2 \times 3.14$, which has to be fraction number.
9. **User input as 'true' or 'false' :** boolean type data must be used.

Note : The arithmetic operations can be done even on char type of data. This is because the processor stores ASCII (American Standard Code for Information Interchange) to store the characters. The ASCII value for capital 'A' is 65, 'B' is 66, 'C' is 67 and so on. While the ASCII values for small alphabets are 97 for 'a', 98 for 'b', 99 for 'c' and so on. The derived and user defined data types will be studied in the subsequent chapters wherever it is needed.

1.9.5 Constants and Variables

- Constants are values given to the identifiers that do not change their values throughout the execution of the program.
- Constants can be defined in Java by writing the keyword "final" before the data type.
- Constants are used to declare the values that remain constant, for e.g. value of pi.
- The use of constants in program will be discussed later wherever required.
- Variables are values given to identifiers that can change their values during the execution of the program.

Ex. 1.9.3 : A variable can be defined with the data type as shown in the examples below :

1. For declaring age as an integer type data the syntax (grammar or method of writing) is
2. For declaring rate of interest as float data type the syntax is
3. For declaring a character data type element the syntax is

Soln. :

1. For declaring age as an integer type data the syntax (grammar or method of writing) is : int age;
2. For declaring rate of interest as float data type the syntax is : float rate;
3. For declaring a character data type element the syntax is : char x;

1.9.6 Escape Sequences

- Escape sequence is a character followed by a backslash (\).
- They are used especially to perform some special operations like going to new line, providing a horizontal tab, vertical tab etc.
- The following is a list of escape sequences.

1.	\n	Newline
2.	\t	Horizontal Tab
3.	\v	Vertical Tab
4.	\r	Carriage Return
5.	\f	Form feed

- It can also be used to store the result in another variable. But in this case the post decrement and pre decrement statements will have different behavior as explained below with examples.
- In post decrement case, for e.g. if x = 5,

  then y = x--;

  will make the value of y equal to 5 and x equal to 4.
- In pre decrement case, for e.g. if x = 5,

  then y = --x;

  will make the value of y equal to 4 and x equal to 4.
- More such examples will be seen with programs.

1.9.7(B) Binary Operators

- Operators that require two operands are called as binary operators.
- These operators are further classified into various types namely the Arithmetic operators, Logical operators, Bitwise operators and Relational operators.
- We will see these operators one by one in this section.

(a) Arithmetic operators

- This set includes the basic arithmetic operators to perform basic arithmetic operations like addition, subtraction, multiplication and division. There are five operators in this set. They are:
  1. * to find the product
  2. / to find the quotient after division
  3. % to find the remainder after division
  4. + to find the sum
  5. - to find the difference
- One important thing to be noted here is that the '/ ' operator returns only the quotient, while the '%' operator (also called as MOD operator) returns the remainder after division.
- The sign of the remainder is always the same as that of the dividend.
- MOD operator is possible only for int or char type of data. It doesn't work on float and double type of data.
- For example of each of these operators
  1. 2 * 2 = 4;
  2. For int type of data, 5 / 3 =1;

     For float type of data, 5 / 3 = 1.67
  3. 5 % 3 =2;
  4. 2 + 2 = 4;
  5. 3 - 2 = 1;

(b) Bitwise operators

- These operators work bitwise on a data.
- They perform different operations on bits of a data like AND, OR, EXOR and NOT.
- The operators are listed below :
  1. ~ to perform bitwise NOT operation.
  2. & to perform bitwise AND operation.
  3. | to perform bitwise OR operation.
  4. ^ to perform bitwise EXOR operation.
  5. << to perform bitwise left shift operation.
  6. >> to perform bitwise right shift operation.
  7. >>>to perform bitwise right shift operation and fill zeroes in blank spaces
- These operators are use to perform bitwise binary operations on the data.
- As we have addition, subtraction operations in decimal data for performing arithmetic operations; similarly AND, OR, NOT are basic bitwise operations on the binary data.
- The result of these operations can be understood from the following examples and referring the Section 1.9.7(B) of binary operations :

1. 5 & 3 = 1

$(5)_{10}$ $(0\ 1\ 0\ 1)_2$

$(3)_{10}$ $(0\ 0\ 1\ 1)_2$

$(0\ 0\ 0\ 1)_2$ $= (1)_{10}$

2. 12 | 9 = 13

$(12)_{10}$ $(1\ 1\ 0\ 0)_2$

$(9)_{10}$ $(1\ 0\ 0\ 1)_2$

$(1\ 1\ 0\ 1)_2$ $= (13)_{10}$

3. 8 ^ 10 = 2

$(8)_{10}$ $(1\ 0\ 0\ 0)_2$

$(10)_{10}$ $(1\ 0\ 1\ 0)_2$

$(0\ 0\ 1\ 0)_2$ $= (2)_{10}$

4. ~7 = – 8

$(7)_{10}$ $(0\ 1\ 1\ 1)_2$

$(1\ 0\ 0\ 0)_2$ $= (8)_{10}$

(According to C/ C++ / Java, wherein it will take more no. of bits and hence -8 will be the result). Hence; in general the ~x is always = – (x + 1).

5. 10 << 2 = 40

Assuming the data to be char i.e. 8 bit data

$(10)_{10}$ $(0\,0\,0\,0\,1\,0\,1\,0)_2$

After shifting left once $(0\,0\,0\,1\,0\,1\,0\,0)_2$

After shifting left for the second time $(0\,0\,1\,0\,1\,0\,0\,0)_2 = (40)_{10}$

Note : When shifting to left, each of the bit is shifted left. The first bit is lost and the last bit is inserted as 0.

6. 13 >>> 3 = 1

Assuming the data to be char i.e. 8 bit data

$(13)_{10}$ $(0\,0\,0\,0\,1\,0\,1\,1)_2$

After shifting right once $(0\,0\,0\,0\,0\,1\,0\,1)_2$

After shifting right for the second time $(0\,0\,0\,0\,0\,0\,1\,0)_2$

After shifting for the third time $(0\,0\,0\,0\,0\,0\,0\,1)_2 = (1)_{10}$

Note : When shifting to right, each of the bit is shifted right. The first bit is inserted as 0 and the last bit is lost.

7. -128 >> 3 = -16

Assuming the data to be char i.e. 8 bit data

$-(128)_{10}$ $(1\,0\,0\,0\,0\,0\,0\,0)_2$

After shifting right once $(1\,1\,0\,0\,0\,0\,0\,0)_2$

After shifting right for the second time $(1\,1\,1\,0\,0\,0\,0\,0)_2$

After shifting for the third time $(1\,1\,1\,1\,0\,0\,0\,0)_2 = -(16)_{10}$

Note : When shifting to right, each of the bit is shifted right. The first bit is retained as it is as well as copied in the next position while the last bit is lost.

(c) Logical operators

- Logical operators follow the same truth table as for the bitwise operators as seen in Section (1.9.7(B)(b)); but they are used to check conditions instead of performing operations on a data.
- The binary logical operators are AND and OR. The symbols used for these operators in Java are && and || respectively.
- For example a statement

  y > 5 && y < 10;

  will result in "true" i.e. 1 is the value of y is greater than 5 AND less than 10, else it will result in false i.e. 0.
- Another example a statement

  y > 5 || y==2;

  will result in "true" i.e. 1 is the value of y is greater than 5 OR equal to 2, else it will result in false i.e. 0.
- Logical operators will be understood in more details with the expressions and program examples followed by this section.

(d) Relational operators

- The relational operators are used to test the relation between two variables or a variable and constant.
- The operators like '<' (less than) and '>' (greater than) used in the above examples are relational operators. We have seen the example also of the "==" (equality operator) in the above logical operators.
- A list of all the relational operators is given below :
  1. == used to check if the two things are equal.
  2. != used to check if the two things are not equal.
  3. < used to check if the first data is less than the second one.
  4. > used to check if the first data is greater than the second one.
  5. <= used to check if the first data is less than or equal to the second one.
  6. >= used to check if the first data is greater than or equal to the second one.
- The examples of these relational operators are as shown with the logical operators above.

1.9.7(C) Ternary Operator

- An operator that requires three operands is called as a ternary operator.
- There is only one ternary operator in Java. This operator is used to check a condition and accordingly do one of the two things based on the condition being true or false.
- The syntax (way of writing) of this operator is as given below :

  (condition) ? <value if condition is true> : <value if condition is false>;
- Hence as shown in the syntax, first the condition is to be written in brackets followed by a question mark (?). Then the operation that is to be performed if condition is true and then a colon (:) followed by the operation to be performed if the condition is false.
- For example :

  z = (x > y) ? x : y;

  This statement will put the value of x into z if the given condition i.e. x > y is true. Else the value of y will be put into z. Hence, the value of the greater variable will be put into z.
- A slightly complicated use of this operator is to find the greatest of three numbers as shown in the example

  g = (x > y) ? ((x > z) ? x : z)) : ((y > z) ? y: z);

  This statement will give the largest of x, y and z into the variable g.

1.9.7(D) Assignment Operators

- These operators are used to assign the value of the expression or variable on the right of the assignment operator to the variable on its left.
- The simple assignment operator is '='. But there are some more assignment operators called as composite assignment operators.
- The different assignment operators are as listed below :

1. = : This operator assigns the value of the expression or variable on its right to the variable on its left. For e.g. y = x + 2 ;

2. += : This operator adds the variable on its left and right and the result is put into the variable on its left. For e.g. y+=x; is same as y = y + x ;
3. -= : This operator subtracts the variable on its right from the variable on its left and the result is put into the variable on its left. For e.g. y-=x; is same as y=y-x;
4. *= : This operator multiplies the variable on its left and right and the result is put into the variable on its left. For e.g. y* = x; is same as y = y*x;
5. /= : This operator divides the variable on its right from the variable on its left and the result is put into the variable on its left. For e.g. y /=x; is same as y = y / x;
6. %= : This operator finds the remainder by dividing the variable on its right from the variable on its left and the result is put into the variable on its left. For e.g. y % = x; is same as y = y % x;
7. &= : This operator ANDs the variable on its left and right and the result is put into the variable on its left. For e.g. y& = x; is same as y = y & x;
8. |= : This operator ORs the variable on its left and right and the result is put into the variable on its left. For e.g. y|= x; is same as y = y|x;
9. ^= : This operator EXORs the variable on its left and right and the result is put into the variable on its left. For e.g. y ^ = x; is same as y = y ^ x;
10. <<= : This operator shifts in left direction the variable on its left for the number of times indicated by the variable or value on right and the result is put into the variable on its left. For e.g. y<<=x; is same as y=y<<x;
11. >>= : This operator shifts in right direction the variable on its right for the number of times indicated by the variable or value on right and the result is put into the variable on its left. The blank spaces are filled with the MSB. For e.g. y >> = x; is same as y = y>>x;
12. >>>= : This operator shifts in right direction the variable on its right for the number of times indicated by the variable or value on right and the result is put into the variable on its left. The blank spaces are filled with the zeroes. For e.g. y >>> = x; is same as y = y>>>x;

1.9.7(E) Selection Operators

- These operators are used to select certain element of a set of elements. Here, we will make a list of these operators and see the detailed use of these operators when we study the corresponding topic where these operators are required.
- The different operators in this set are listed below :

1. [] : This operator is used to select an element of Array. We will understand more about this in the chapter named Array.
2. . : This is called as period operator and is used to select an element of a object.
3. () : This is called as a function call operator and is used to call or select a function.
4. , : The comma (,) operator is used to separate the different values etc.

1.9.8 Precedence and Associativity of Operators

- The precedence of the operators means the sequence in which the operators will be operated on, in case of multiple operators in a statement i.e. which operator will be executed first and which operator will be executed later.

- The associativity of operators refers to the direction in which the operation will be performed in case of equal precedence operators i.e. if multiple additions are there in a statement then it will be performed from left to right. We will see more about this in the examples followed by this section.
- The precedence and associativity table is as given in Table 1.9.5.

Table 1.9.5 : Precedence and Associativity of operators

Precedence	Operator	Operator name	Associativity
1.	[] . ()	access array element access object member invoke a method	Left to right
2.	++ -- - ! ~	pre-increment pre-decrement unary minus logical NOT bitwise NOT	Right to left
3.	() new	Cast object creation	Right to left
4.	* / %	multiplicative	Left to right
5.	+ - +	Additive string concatenation	Left to right
6.	<< >> >>>	shift	Left to right
7.	< <= > >= instanceof	Relational type comparison	Left to right
8.	== !=	Equality Inequality	Left to right
9.	&	bitwise AND	Left to right
10.	^	bitwise XOR	Left to right
11.	|	bitwise OR	Left to right
12.	&&	logical AND	Left to right
13.	||	logical OR	Left to right
14.	?:	Ternary (conditional operator)	Right to left
15.	= += -= *= /= %= &= ^= |= <<= >> = >>>=	Assignment and compound assignment operators	Right to left

1.10 Expressions

Let us see some examples of how the processor performs operations when some statements are given to us in the following examples. We will also see some examples wherein an expression is to be written in Java, so how are they written.

(a) Determine the value of the following expressions if a = 7, b = 3 and c = 4;

1. a % b

   = 7 % 3

   = 1 (since the remainder after dividing 7 by 3 is 1)

2. a / c

   = 7 / 4

   = 1 (since the quotient after dividing 7 by 4 is 1)

3. a * b / c

   = 7 * 3 / 4

   = 21 / 4 (since the associativity of arithmetic operations is left to right first operation is *)

   = 5 (since; quotient is 5)

4. a * (c % b)

   = 7 * (4 % 3)

   = 7 * 1 (since bracket opening is to be done first, and remainder of 4 divided by 3 is 1)

   = 7

5. 2 * b+ 3* (a-c)

   = 2 * 3 + 3 * (7 – 4)

   = 6 + 3 * (3)

   = 6 + 9 (since precedence of multiply is more than add, first multiplication is done)

   = 15

6. a * c % b

   = 7 * 4 % 3

   = 28 % 3

   = 1

7. a + b – c

   = 7 + 3 – 4

   = 6

(b) Suppose the following statements are written :

```
int i = 9, j = 6;
float x = 0.5, y = 0.1;
char a = 'a', b = 'b';
```

Find the values of the following expressions

1. (3 * i - 2 * j) % (2 * a - b)

   = (27 - 12) % (2 * 97 - 98)

   ...(Since the ASCII values stored in char type variables is 97 for a & 98 for b)

   = (15) % (96)

   = 15

2. 2 * (j/5) + (4 * (j-3)) % (i + j - 2)

   = 2 * (6 / 5) + (4 * (6 - 3)) % (9 + 6 -2)

   = 2 * (1) + (4 * (3)) % (13)

   = 2 + (12) % 13

   = 14

3. (x > y) && (i > 0) && (j > 5)

   = 1 && 1 && 1 .....(if condition is true its result is 1 else it is 0)

   = 1

4. ((x < y) && (i > 0)) || (j > 3)

   = (0 && 1) || (1)

   = (0) || (1)

   = 1

5. a==99

   = 0 ....(since condition given is false, the value of a is 97 as discussed in section 2.12)

6. ++i

   = 10 ....(since it is pre increment it will return the value after incrementing i.e. 10)

7. i++

   = 9 ....(since this is post increment it will return the value before incrementing i.e. 9)

8. !(b==98)

   = ! (1) ....since the condition given is true)

   = 0 ....(since after performing NOT operation the value becomes 0)

(c) Write the Java assignment expressions for the following :

1. A is greater than B and greater than C

   Solution : (A > B) && (A > C)

2. A is either less than B or greater than C

   Solution : (A < B) || (A > C)

3. Side is equal to square root of $(a^2 + b^2 + 2ab)$

   Solution : Side = Math.pow ((a * a + b * b + 2 * a * b), 0.5)

Note :

1. pow(x,y) is an available static method in the class Math that can be used to perform x^y operation. x and y are the variables to be given in the brackets as shown in above example.
2. Only the round brackets i.e. () are allowed in expressions. Other brackets have different meaning in the Java programming languages.
3. All the operations including that of multiplication is to be specified; no operation is implied. For e.g. we cannot write x = ab, instead we need to write x= a* b.

4. $a = \frac{xy + z\left(\frac{x}{y}\right) + zy}{x + y + z}$

   Solution : a = (x*y + z*(x/y) + z*y) / (x + y + z);

5. $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

   Solution : x_1 = (-b + Math.pow ((b*b-4*a*c) ,0.5)) / (2*a);

   x_2 = (-b-Math.pow ((b*b-4*a*c),0.5)) / (2*a);

6. $z = a + \frac{1}{1 + \frac{k + a}{2}}$

   Solution : z = a+1 / (1+ (k+a) / 2);

1.11 Comments

- Description of the program statements are called as comments. Comments are useful for someone who reads the program later.
- Compiler ignores the comments and hence there is no standard syntax for comments.
- But the beginning of comments and sometimes the end of comments are to be indicated to the Compiler. Java supports three types of comments :

1. **Single line comments :** Statements that begin with double forward slash ("//") are comments upto the end of that line.
2. **Block Comments :** Statements beginning with forward slash and star sign ("/*") indicate the beginning of block comments, while the reverse (i.e. "*/") indicates the end of block comments. This type of comments can extend to multiple lines.
3. **Documentation Comments :** Statements that begin with forward slash and two star sign ("/**") indicates the beginning and a star sign followed by forward slash ("*/") indicates the end of documentation comments. Documentation comments have an advantage to automatically generate the documentation of a program created by documentation generation tool like "javadoc".

1.12 Input / Output in Java

In this section we will see how to accept the data from user and display some data on the monitor.

1.12.1 Displaying Output in Java

- In Java we have two methods namely print() and println() for displaying the output on standard output device i.e. monitor.

- These methods are available in the a package called as "java.lang". This file has many classes and is by default available in any java program. We will discuss in more detail about this package in chapter 10.
- One of the class of this package is called as "System". This class has a variable called as "out"; that corresponds to the standard output device i.e. monitor. The class has two methods namely print() and println() that are static i.e. these methods can be called without making the object of the class "System". The syntax for calling a static member method is:

```
class_name.method_name()
```

For e.g.: System.out.print() or System.out.println()

- The parameters passed to these methods are displayed on the monitor. The only difference between the two methods print() and println() is that the second function makes the cursor go to the next line after displaying the given values while the first one doesn't.

1.12.2 Accepting Input in Java

- In Java we have many methods to accept input from the user (or keyboard). Mostly used ones are functions of the classes like BufferedReader, DataInputStreamReader and Scanner.
- We will be using a very simple and important method of these i.e. Scanner. All the other classes have functions to accept string. The string accepted is to be then converted to the required data type. But for the class Scanner, we can directly accept the required data type value. We will also see a program using the BufferedReader class in this section.
- There are methods to accept various data type values in the class Scanner. These methods are non-static and are in the class Scanner. Hence we need to make an object of this class to use these methods. The syntax of making an object of a class in Java is as given below:

```
class_name object_name = new class_name(parameters_for_constructor);
```

- The class Scanner is in the package "java.util". When making an object of the class Scanner, we need to pass an object of , we need to pass to the constructor a variable of the class "System" called as "in". This variable refers to the standard input device i.e. keyboard. Hence the object of the class Scanner is to be created as shown below:

```
Scanner sc = new Scanner (System.in);
```

- Here the object named as "sc" is created of the class Scanner. While creating this object a variable is passed to the constructor that indicates the source of the input. The new operator is used to create an object of a class as already discussed in this section.
- Since this class "Scanner" is in the package "java.util", we need to import all the classes of this package by writing the statement "import java.util.*" in the beginning of the program.
- The nextInt() method can accept integers. Similarly nextFloat() for float type values. A list of these methods is given in the Table 1.12.1.

Table 1.12.1

Sr. No.	Method	Function
1.	nextInt()	Returns an integer value entered from the keyboard
2.	nextLong()	Returns an long value entered from the keyboard
3.	nextFloat()	Returns an float value entered from the keyboard
4.	nextDouble()	Returns an double value entered from the keyboard
5.	next()	Returns an string terminated with a blank space entered from the keyboard
6.	nextLine()	Returns an string value (that terminates with a new line or enter key) entered from the keyboard

Explanation

- The three arguments passed through the command line are received in the three elements of the array i.e. args[0], args[1] and args[2]. These three elements are then parsed to integers and stored in the variables namely no1, no2 and no3 respectively.
- This program uses ternary operator to find the larger of two numbers no1 and no2 and put that into the variable "large". Then again the same operator is used to find the largest of large and no3 and hence get the largest number in the variable large.

1.18 Introduction to Control Statements

- In programming we need to sometimes control the flow of operation other than just the sequential statements. In this case we need the control statements.
- Control statements are classified into two types viz. the **iterative statements** and **conditional statements**.
- **Iterative statements** are used to perform certain operations repetitively for certain number of times. In Java we have three iterative statements viz. for loop, while loop and do-while loop. Iterative statements are also called as repetitive statements as they repeat a set of statements for a given number of times.
- **Conditional statements** are used to perform the operations based on a particular condition i.e. if a condition is true perform one task else perform another task. In Java we have two conditional statements namely if-else statements and switch-case statements. Conditional statements are also called as selective statements i.e. these statement select a particular statement to be executed based on the condition.
- In the subsequent sections we will see all these statements one by one in detail.

1.19 The for Loop

- For is a iterative statement. It is used to repeat a set of statements number of times. The syntax (method of writing) the **"for"** statement is given below :

```
for(initializations; condition; increment / decrement / updating)
{
        -
        -
        -
        statements;
        -
        -
        -
}
```

- The sequence of execution of the for loop is such that the **initialization statements are executed first.**
- These initialization statements are executed only once. They are used to initialize the values of the variables.
- The **second step is checking the condition** specified. There can be only one condition. If more than one conditions are required they can be combined using the logical AND, OR operators.
- If the condition is false the execution of the loop will be terminated i.e. the execution will go outside the braces of for loop, if the condition is not true.

- The **third step** is to **execute all the statements** inside the curly braces. These statements will be executed sequentially. The number of statements can be of any count. There can be another control statement if required inside one control statement.
- The **fourth step** is the **increment / decrement** or updating operations. These operations are not necessarily increment or decrement operations, but mostly these are increment decrement and hence called so. We can update the variables over here before starting the next iteration of the iterative statements.
- **Finally the control goes back to the second step.** As said the first step is executed only once, the steps that are repeated continuously are the second, third and fourth steps. After the fourth step the condition is checked again.
- If the condition is true the execution continues, else the control goes outside the for loop i.e. the curly braces.
- In the following sub section we will see some programs using the for loop.

1.19.1 Programs Based on for Loop

Program 1.19.1 : Write a program to display the word "Computer" five times using for loop.

```
class Display
{
public static void main(String args[ ])
{
 int i;
 for(i=1;i<=5;i++)
 {
        System.out.println("Computer\n");
 }
 }
}
```

Output

```
Computer
Computer
Computer
Computer
Computer
```

Explanation

- In the above program we have used the for loop. The variable 'i' is initialized to 1 in the initialization statement. The condition statement is checked if the value has reached 5. If not reached then the control enters inside the loop i.e. the curly braces.
- The condition we have put is i<=5, since the value of i will vary from 1 to 5. Hence the statements inside the loop must be executed in all cases when the value of i is less than or equal to 5. This makes the conditional statement to be i<=5.
- All the statements inside the loop are executed. In this case there is only one statement i.e. println() to display the required string. Finally the increment decrement operation statements or updating statements are executed. In this case the value of 'i' is incremented to 1. Thereafter the control goes back to the condition statement, and the loop continue until the value of 'i' reaches to five i.e. five times.

```
45    9922.5
50    12250.0
55    14822.5
60    17640.0
65    20702.5
70    24010.0
75    27562.5
80    31360.0
85    35402.5
90    39690.0
95    44222.5
100   49000.0
95    44227.500861
100   49005.000954
```

Explanation

- The values of s_0, v_0 and a are taken from the user.
- The output is displayed in a tabular form with the values of t and s separated by a horizontal tab (using the escape sequence "\t").
- The first value i.e. with t=1, is displayed outside the loop, as it doesn't match the series of 5, 10, 15, 20...100.
- The remaining values are calculated and displayed using a for loop wherein the value of t is incremented by 5 after every iteration. The new value of s is calculated and displayed again separated with a horizontal tab.
- This loop is repeated until the value of t reaches 100.

1.19.2 Nested for Loop

- A for loop inside another for loop is called as nested for loop.
- When a particular operation has two references, we require nested for loop. For example if we want to keep a reference of row number and column number, then we can use a nested for loop.
- Many programs based on nested for loop are given below. In this case the variable i keeps a track of row number and j keeps a track of column number.

Program 1.19.16 : Write a program to display "Hi" twice in a line and five such lines.

```
import java.util.*;
class Simple
{
public static void main(String args[ ])
{
 int i,j;
 for(i=1;i<=5;i++)
 {
 for(j=1;j<=2;j++)
```

Output

```
Enter number of lines:5
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
```

Explanation

- Here the numbers to be printed are not the same as the value of j i.e. column number. Instead the numbers are to be incremented every time a value is printed. Hence a separate variable i.e. k is used to keep a track of the numbers to be printed.

1.20 while and do-while Loops

- while and do-while loops are also used for repetitive operations.
- The operations are slightly different than the for loop, but the same operations can be implemented by for, while or do-while loops.
- Although there is one major difference in a do-while loop wherein one particular operation cannot be implemented. This will be discussed later in this section.
- The syntax of while loop is given below :

Syntax of while loop:

```
while(condition)
{
        -
        -
        statements;
        -
        -
        -
}
```

- The operation of the while loop is such that, first the condition is checked. If the condition is true, then the statements are executed.
- Once the statements are executed, the condition is again checked and this keeps on repeating, until the condition is false. If the condition is false, the statements inside the loop are not executed, instead the control directly comes out of the loop.
- The syntax of do-while loop is given below :

Syntax of do-while loop:

```
do
{
        -
```

```
    .
    statements;
    .
    .
    .
}while(condition);
```

- In this case the operation is slightly different i.e. first the statements are executed and then the condition is checked.
- If the condition is true the statements are executed again. If the condition is false, the statements are not executed again.
- One major point to be noted is that in case of do-while loop, the statements are executed atleast once even if the condition is not true for the first statement.
- On the other hand, in case of for loop and while loop, even for the first time the statements are executed only if the condition is true.

The implementation of a for loop converted to while and similarly do-while loop is shown in Fig. 1.20.1 (a) and 1.20.1 (b) respectively.

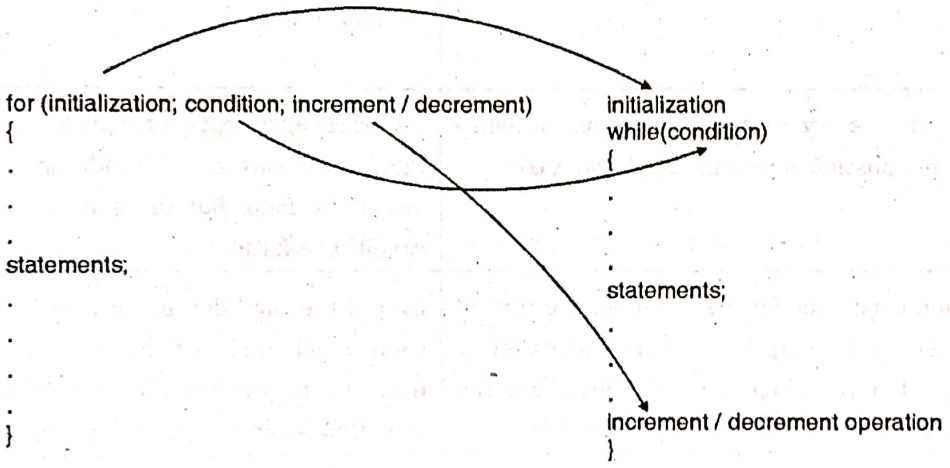


Fig. 1.20.1(a) : Implementation of a while loop related to for loop

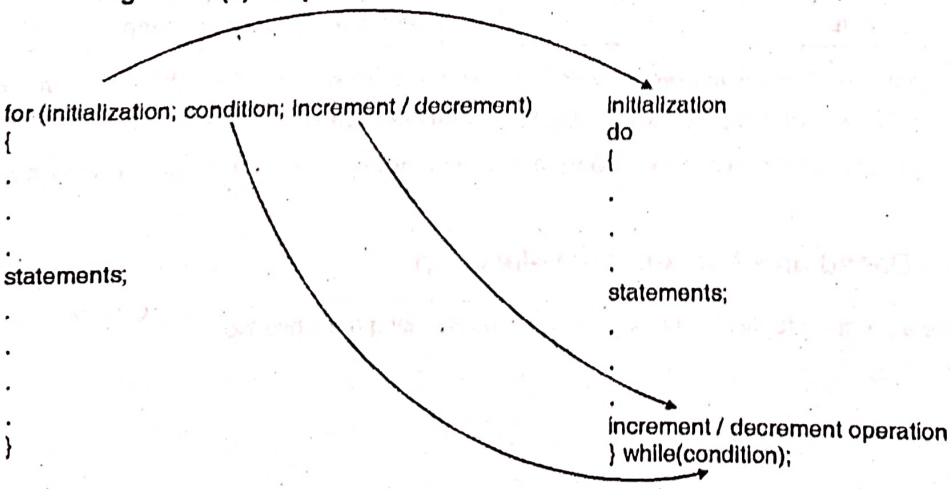


Fig. 1.20.1(b) : Implementation of a do-while loop related to for loop

- You will note (in Fig. 1.20.1) that the initialization statements are removed outside the loop, the condition is in the brackets associated with the while part and increment decrement are to be inside the loop at the end so as to implement a similar operation as that in the for loop.
- Differences between the while and do-while loop is given in Table 1.20.1.

Table 1.20.1 : Differences between while and do-while loop

Sr. No.	while loop	do-while loop

1	Syntax of while loop:
	while(condition)
	{
	-
	-
	statements;
	-
	-
	-
	}
		do
		{
		-
		-
		statements;
		-
		-
		-
		} while(condition);

2	This is called as a entry controlled loop, as the entry inside the loop is possible only if the condition is true.	This is called as exit controlled loop, as the entry inside this loop is sure i.e. no condition is checked to enter inside the loop. But the exit is possible only if the condition is false.
3	If the condition is not true for the first time, the control will never enter into the loop. Hence there is a possibility that the control never enters into the loop and the statements inside the loop are never executed.	Even if the condition is not true for the first time the control will enter into the loop. Hence the statements inside the loop will be executed atleast once even if the condition is not true for the first time.
4	There is no semicolon (;) after the condition in the syntax of the while loop.	There is a semicolon (;) after the condition in the syntax of the do-while loop.

- We will do some basic programs using these loops in the subsequent section. We will first see some programs already implemented using for loop now implemented using while and do-while loops.
- There after we will see some programs wherein we will notice that the while or do-while loop give better implementation.

1.20.1 Programs Based on while and do-while Loop

Program 1.20.1 : Write a program to display first n natural numbers using the while loop.

```
import java.util.*;
class Natural
{
public static void main(String args[ ])
{
```

```
            if(copy==sum)
            System.out.println("Armstrong Number");
            else
            System.out.println("Not Armstrong Number");
        }
}
```

1.22 Switch-Case Selective Statement

- In the previous section we have seen the if-else ladder. A better solution of this is switch-case. Using switch-case the if-else ladder can be implemented in a much better way.
- The syntax of switch-case is given below :

```
switch(expression / variable)
{
case label1: statements;
            break;
case label2:statements;
            break;
case label3:statements;
            break;
        |
        |
caselabeln:statements;
            break;
default : statements;
}
```

Note : Break statement transfers the control outside the current loop. We will see some more cases of break statement along with the for / while /do-while statements in section 3.6.

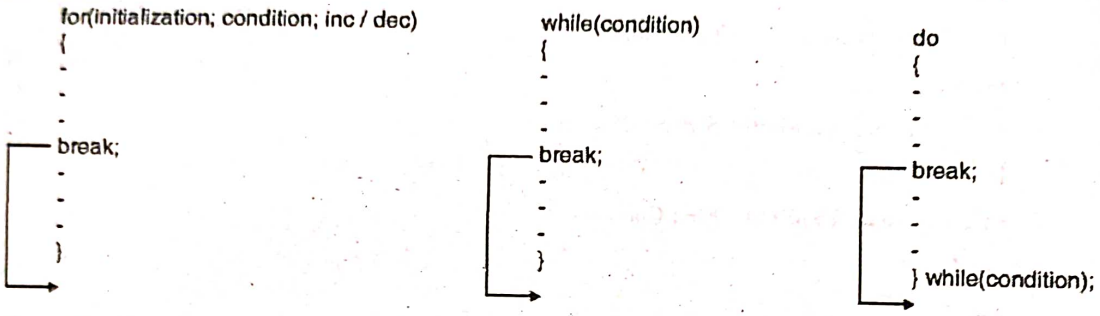
- The expression or variable can be given in the brackets associated with the switch statement. The values of this expression / variable are the labels associated with the cases inside the switch statement.
- The value of the expression / variable is first compared with the label1. If they are equal, then the statements followed by the corresponding case are executed. The break statement followed by these statements, transfers the control after the switch statement.
- If the expression / variable is not equal to label1, then it is directly compared to label2 without executing the statements followed by the label1.
- Hence the statements of only that case are executed with the correct label value of the expression / variable.

Note :

1. Break statement is not compulsory. But if the break statement is not written everything will be executed after the case statements

1.23 Branching Statements (Break and Continue)

- These statements transfer the control to a different place in the program. We will see the operation of each of these. We have already seen the use of break statement in switch-case. Let us see the use of continue and break in other loop statements
- The break statement neglects the statements after it in the loop and transfers the control outside the loop as shown in Fig. 1.23.1. The Fig. 1.23.1 shows the operation of break statement in each of the loop statements i.e. for, while and do-while statements.



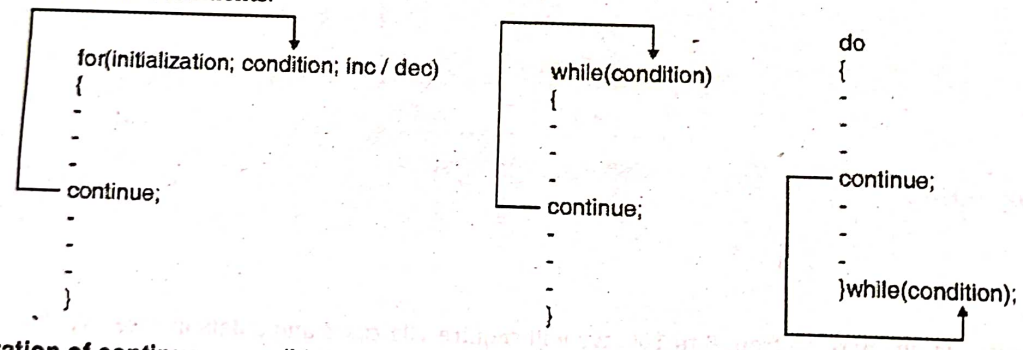
(a) Operation of break statement in a for loop

(b) Operation of break statement in a while loop

(c) Operation of break statement in a do-while loop

Fig. 1.23.1

- The continue statement also neglects the statements after it in the loop and transfers the control back to the starting of the loop for next iteration. The Fig. 1.23.2 shows the operation of continue statement in each of the loop statements i.e. for, while and do-while statements.



(a) Operation of continue statement in a for loop

(b) Operation of continue statement in loop

(c) Operation of continue statement in a while a do-while loop

Fig 1.23.2

Program 1.23.1 : Write a program to demonstrate the use of break statement.

OR Write a program to accept 10, 2-digit numbers from user and add them. If the user enters a three digit number stop accepting the numbers and display the sum.

```
import java.util.*;
class Break
{
public static void main(String args[ ])
{
 int n,total=0,i;
```

Classes and Objects

2.1 Introduction to Objects

- An object is an instance or an example of a class. For example if "Human Being" is a class, then you and me are examples or objects of this class. A real-world object or the object of a class has two characteristics:

  1. State 2. Behavior

- We will understand these concepts in the next sub-section.

2.1.1 State and Behaviour of an Object

- The state of an object can be related to the variables. For example if there is a class like "Student", then some of the states of the object of this class can be :

  1. Name of the student,
  2. Class and Division of the student,
  3. Roll number of the student, etc.

- The behavior refers to the different operations done by the object. The behavior of the object of the class "Student" can be

  1. Attending Lectures,
  2. Issue a book from Library,
  3. Completing Assignments, etc.

2.1.2 Introduction to Java Access Modifiers

- The access to the members of a class i.e. the constructors, methods and fields of class is controlled by access specifiers.
- Thus access specifiers indicate who can access the members of a class. For encapsulation, we should keep the data fields in minimal access while method members in maximal access.
- The access specifiers are as listed below :

1.	public	2.	protected
3.	default	4.	private
5.	private protected		

- Let us see these access specifiers in detail.

1. public

Those fields, methods and constructors that are declared public i.e. least restriction. The members that are declared public can be accessed by members of all the classes may be of same or different package.

2. **private**

The private member fields and methods are the most restricted ones i.e. they cannot be accessed by any methods except for the ones in the same class.

3. **protected**

The fields and methods declared "protected" can be accessed by every method except for the methods in the non sub classes of different package.

4. **private protected**

This gives a visibility level between the "protected" and "private". These members can be accessed only by the sub classes that can be of the same package or other package. This access specifier is not available in some later versions of JDK.

5. **default**

- Java also provides a default specifier which, as the name says is the access specifier for those members where no access modifier is present. All fields and methods that have no declared access specifier are accessible only by the methods of same class. This access specifier is also many times termed as "friend".
- The Table 2.1.1, shows the scope of access of the members of a class.

Table 2.1.1 : Access Specifiers/ Modifiers

AccessModifier → / Access Location ↓	Public	Protected	Default (friendly)	Private protected	Private
Same class	Yes	Yes	Yes	Yes	Yes
Sub class in same package	Yes	Yes	Yes	Yes	No
Other classes in same package	Yes	Yes	Yes	No	No
Subclass in other packages	Yes	Yes	No	Yes	No
Non-subclasses in other packages	Yes	No	No	No	No

2.2 Java Member Methods

- A class is a collection of fields and methods. We can make objects of that class and memory space will be allocated to the fields of that object. The methods of a class can be accessed using the objects using the period operator. The syntax of making an object of a class is as given below:

class_name object_name = new class_name(parameters_to_be_passed_to_the_constructor)

- We will learn some more concepts of Object Oriented Programming in later sections. We will see some simple program examples for making classes and its object.

2.3 Constructors, Destructors, Modifiers, Iterators and Selectors

- The operations or the methods in a class may be divided up into several types. According to Booch there are five types of operations, namely
  1. Constructor
  2. Destructor
  3. Modifier
  4. Selector
  5. Iterator
- The Constructors and destructors are used to create and destroy objects of a class, respectively. Basically the constructor initializes the values of the member variables of an object, while destructor destroys the memory space allocated for that object.
- The destructor work is implemented by the finalize() method in Java. We will see the constructors in detail in the next sub-sections while finalize() method later in next chapter.
- Some methods work as Modifiers i.e. change the values within the object. Selectors as the name says just read the values from an object without modifying them.
- Some methods are called as Iterators methods that provide orderly access to the components of an object. The iterator methods are most common with objects maintaining collections of other objects like vector class object.

2.3.1 Constructors

- Constructor is a special member method used to initialize the field members of an object.
- There are some special points to be noted for a constructor, as listed below:
  1. Constructor should always be in the public/default/protected visibility of a class.
  2. The name of the constructor should always be same as that of the class.
  3. Constructor should not have any return type, not even void.
  4. Constructor is automatically called whenever an object of that class is created.
  5. There can be more than one constructor, with different parameter list. This is called as constructor overloading.
  6. Parameters can be passed to the constructor, while creating the object in the brackets as discussed in the syntax of declaration of an object.
  7. Constructors can be classified based on parameters passed to it. If no parameters are passed to a constructor, such a constructor is called as Default constructor. If parameters are passed to constructor, such constructor is called as parameterised constructor. A constructor that accepts an object of same class as parameter is called as copy constructor.
- We will see the different types of constructors in the following program.

2.3.1(A) Parameterized Constructor

Program 2.3.1 : Write a program to make a class called as Circle. It should have a parameterized constructor to initialize the radius. It should have two methods namely : calculate area and display the area.

```
import java.util.*;
class Circle
{
    private float r,area;
    Circle(float x)
```

Output

```
Enter two numbers:20
12
GCD=4
```

Explanation

- The class Euclid is first made with the private data (field) members 'n1', 'n2' and "gcd". The class has three method members namely
  1. Euclid() i.e. the default constructor to accept the two numbers. This constructor accepts two inputs from user and initializes the variables n1 and n2.
  2. calculate() to calculate the GCD.
  3. display() to display the GCD.
- Then another class is made, that has the main() which is the beginning of the execution of the program. Hence the name of the program is as the name of this class. Also the main() method is declared public and static.
- In the main() method, we have created an object of the class Euclid named as 'e'.
- Then the calculate() method is called and finally the display() method.

2.3.1(C) Copy Constructor

- A parameterized constructor to which the parameter passed is an object is called as copy constructor.
- The parameters of the object passed to the constructor are used to initialize the parameters of the newly created object, hence the name "copy".
- Let us see some programs using this copy constructor.

Program 2.3.5 : Write a program to make a class called as Circle. It should have a default constructor to initialize the radius and a copy constructor. It should have two methods namely: calculate area and display the area.

```
import java.util.*;
class Circle
{
        private float r,area;
        Circle()
        {
                Scanner sc= new Scanner(System.in);
                System.out.print("Enter Radius:");
                r=sc.nextFloat();
        }
        Circle(Circle x)
        {
                r=x.r;
        }
    void calculate()
```

```
    {
        Euclid e=new Euclid();
        e.calculate();
        e.display();
        Euclid e1=new Euclid (e);
        e1.calculate();
        e1.display();
    }
}
```

Output

```
Enter two numbers:20
15
GCD=5
GCD=5
```

Explanation

- The class Euclid is first made with the private data (field) members 'n1', 'n2' and "gcd". The class has three method members namely :
  1. Euclid() i.e. the default constructor to accept the two numbers. This method accepts two integers, which is initialized in the variables n1 and n2. Another constructor is written with a parameter as an object of the same class. The two numbers of this object is initialized in the object of new constructor.
  2. calculate() to calculate the GCD.
  3. display() to display the GCD.
- Then another class is made, that has the main() method which is the beginning of the execution of the program. Hence the name of the program is as the name of this class. Also the main() method is declared public and static.
- In the main() method, an object is created of the class Euclid named as 'e'. This calls the default constructor.
- Then the calculate() method is called and finally the display() method.
- Another object is made i.e. "e1" of the same class i.e. "Euclid". To this constructor the previously created object is passed i.e. the object "e". Hence it calls the copy constructor.
- The calculate() and display() methods are then called for this object also.

2.4 Passing Objects to a Method

- We have seen that for copy constructors, we pass an object of the same class to the constructor. Objects can be passed to methods also. The objects are handled in the same manner by a method as that was done by constructors in the above program examples. We will see the following program example, wherein we pass an object to a method.
- When an object is passed to a method in Java, the values contained in that object can be changed. These values will remain changed even when the method execution ends and the control returns to the caller method. This will be seen in more details in the further sections. In this program given below, we are adding a complex number with another. The add() method will be called by one of the object and another object will be passed to this method.
- The method will add the values of the corresponding object and then the result will be displayed.

Output

```
Enter real and imaginary part of a complex number:
2
3
Enter real and imaginary part of another complex number:
4
5
6+i8
```

Explanation

- The class Complex has a constructor to initialize the values of the object made of this class. The class has two method members namely
  1. add() to add the two complex numbers.
  2. display() to display the complex number.
- The add() method accepts an object of the class Complex. The method adds the real and imaginary parts of the object for which the method is called with the corresponding values of the object 'a' passed to it.
- In the main() method, we have accepted the real and imaginary parts of two complex numbers from user and accordingly two objects are made namely c1 and c2.
- Then the add() method is called by the object c1 and the object c2 is passed to this method. The method as discussed earlier adds the corresponding parts of the two complex members. Finally the display() method is called again by the object c1 as the result is in the object c1.
- The display() method has a special technique to be followed. If the imaginary part is greater than or equal to zero then "+i" must be displayed between the two values i.e. x and y. But if the imaginary part is less than zero, then the x value followed by the value of y with its sign and finally "i" as it is. The special care has to be taken to differentiate between the "+" sign as addition and the "+" sign as string concatenation (joining). You will notice that is done in the statement in the display() method.

2.5 Returning Objects from a Method

- The return type of a method indicates what type of data will a method return. If the method returns an integer type data, the return type is written as "int"; similarly for "float" type of data to be returned, the return type is "float" and so on. Similarly if an object of a class is to be returned the return type of that method will be the name of the class, whose object is to be returned.
- We will see an example of returning an object in the next program example. The same concept of adding two complex numbers will be considered, but the result will be returned to another object of the same class.

Program 2.5.1 : Write a program that demonstrates returning objects to a method.

```
import java.util.*;
class Complex
{
    private int x,y;
    Complex(int a,int b)
    {
```

Output

```
Enter real and imaginary part of a complex number:
1
2
Enter real and imaginary part of another complex number:
3
4
4+i6
```

Explanation

- The class Complex has a constructor to initialize the values of the object made of this class. There is a constructor added in this program that doesn't initialize the values of its variable 'x' and 'y'. This constructor is especially for the object c3 and the object c.
- The class has two method members namely
  1. add() to add the two complex numbers.
  2. display() to display the complex number.
- The add() method accepts an object of the class Complex. The method adds the real and imaginary parts of the object for which the method is called with the corresponding values of the object 'a' passed to it. The answer is stored in another object created in the method itself. Hence this object is returned by the add() method and accepted by the main() method into the object c3.
- In the main() method, we have accepted the real and imaginary parts of two complex numbers from user and accordingly two objects are made namely c1 and c2.
- Then the add() method is called by the object c1 and the object c2 is passed to this method. The method as discussed earlier adds the corresponding parts of the two complex members. Finally the display() method is called again by the object c3 as the result is in the object c3.
- The display() method has a special technique to be followed. If the imaginary part is greater than or equal to zero then "+i" must be displayed between the two values i.e. x and y. But if the imaginary part is less than zero, then the x value followed by the value of y with its sign and finally "i" as it is. The special care has to be taken to differentiate between the "+" sign as addition and the "+" sign as string concatenation (joining). You will notice that is done in the statement in the display() method.

2.6 Call by Value and Call by Reference

In the chapter 5, we saw we passed parameters to a method. The values of those variables were given to a set of another variable in the method.

2.6.1 Call by Value

- The variables passed by the method are called as **actual** parameters and the ones received by the called method are called as **formal** parameters.
- Since only the values of the variables are passed and not the actual parameters, the method cannot alter the actual parameters.
- The formal parameters are altered, but the actual parameters remain unchanged. This can be understood by a program on swapping two numbers using a method.

```
After swapping in method
Numbers are a=7 b=4
After returning from method
Numbers are a=7 b=4
```

Explanation

- In this case the object is passed to the method swap().
- The object is received in formal parameter namely x. The parameters of this object 'x' are swapped, but the actual parameters are also changed in this case.
- This is demonstrated in the output of the program. The numbers accepted were 4 and 7 respectively for 'a' and 'b'. In the method the numbers are swapped i.e. 'a' has 7 and 'b' has 4. And when the control returns back to the main() method the values are also found to be swapped i.e. changed.

2.7 Static Class Members

- We can declare a method or a field to be static. We may also have a block called as static. What is the effect of the keyword **"static"** on these members will be seen in the following sub-sections.
- In this chapter we have called methods by making the objects of the class. Such methods are called as **instance methods** i.e. they can be called by the instance (object) of a class.
- A static method is a method that works on a class and not on an object of that class. To call a static method of a class we need not make any object of that class.
- A static method has to be called with the syntax class_name.method_name(), as also discussed in the earlier chapters.
- A static field member is also having the similar properties like a static method. A static field or variable will be common for all the objects of the class.
- A common memory location will be allocated for a static variable for all the objects made of that class.
- One of the majoradvantages of a static variable is that we can make use of static variable to find the count of the objects made of a particular class. Since each object will access the same memory location for static variable, we can increment this variable in the constructor of the class. Whenever an object is created, this constructor will be automatically called and this static variable common to all the objects will be incremented. At any time if we want to know the number of objects created till that time we need to just display this static variable value. We will see this done in the program 2.7.1.

Program 2.7.1 : Write a program to count the number of objects made of a particular class using static variable and static method to display the same.

```
class Counter
{
    private static int count;
    Counter()
    {
        count++;
    }
static void display()
```

```
    {
      System.out.println("Count=" +count);
    }
  }
  class Main
  {
      public static void main(String args[])
      {
          Counter c1=new Counter();
          Counter.display();
          Counter c2=new Counter();
          Counter c3=new Counter();
          Counter.display();
          Counter c4=new Counter();
          Counter c5=new Counter();
          Counter.display();
      }
}
```

Output

```
Count=1
Count=3
Count=5
```

Explanation

- The class Counter has just one static variable and a static method. Besides static members, the class also has a constructor to increment the value of the static variable count.
- The variable is incremented whenever an object of the class Counter is created. We have called the static method by the syntax class_name.method_name(), after creating some objects.
- For the first time we have just created one object and we have called the display() method and hence the output shows Count=1.
- The next two times we have done the same thing but after making two objects each time and hence the outputs are Count=3 and Count=5 respectively.

2.8 The "this" Keyword

- The *"this"* keyword refers to the current object. When you are executing the statements in a method(), and you want to refer to the same object through which the method is called, then you need to use the "this" keyword.
- We have been using the parameters of the current object in the instance methods of the programs in this chapter. But we never required the keyword "this". We do require the keyword "this" in some cases. A very good example of the same is shown in Program 2.8.1 below.

Methods and Inheritance in JAVA

3.1 Arrays

- It is a collection of multiple data of same data type. For example we can have an array of int type data or float type data etc.
- Remember, array can have data of same type only i.e. all elements of an array have to be of same type only. We cannot have an array of combination of different data types.
- We need to note two very important points about array
  1. The starting index i.e. index of the first element of an array is always zero.
  2. The index of last element is n–1, where n is the size of the array.
- An array does not have static memory allocation in case of Java i.e. memory size can be allocated for an array during run time
- Syntax of declaring an array

```
data_type array_name [ ] = new data_type [array_size];
where,
data_type is the data type like int, float, double etc.
array_name is the identifier i.e. the name of the variable
```

new is an operator to allocate memory to an object

array_size is an integer value which determines size of the array or memory locations to be allocated to an array.

For e.g. int a[ ] = new int [10];

is an array of int type data named 'a' and can store upto 10 integers indexed from 0 to 9 i.e. 0 to n – 1, where n is the size

- The memory allocated to an array also depends on the data type i.e. the space required for each element

  For e.g. : int a[ ] = new int[10]; will require 40 byte memory locations, as each integer type data requires 4 byte memory locations double x[ ]=new double [20]; will require 1600 byte memory locations, as each double type data requires 8 bytes of memory.
- To access an element of an array we have to use the array selection operator i.e. [ ].
- For e.g. if we want to access the 3rd element of an array named 'a', then we need to access it as a[2]. Remember 3rd element will be indexed 2, as the index of first element is 0.
- Hence to access an element we need to use the array name or identifier and write the index number in brackets.
- Another e.g. to access the 9th element of an array 'x', we need to write x[8].

- If numbers from 1 to 10 are stored in an array of 10 elements then, the values will be stored in the array as shown in the Fig. 3.1.1 each of these locations shown in the Fig. 3.1.1 are of four bytes i.e. to store one integer type of data.

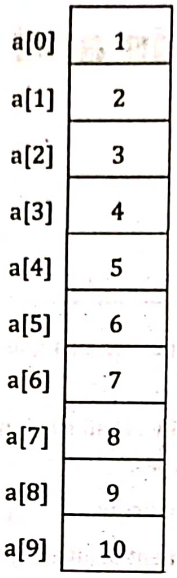


Fig. 3.1.1 : Memory allocation for an array 'a' of 10 integer elements

- We will see some very basic programs of accepting and displaying the elements of an array and then move on to some programs that use these operations.
- Later we will also see some programs of arrays using functions.

Program 3.1.1 : *Write a program to accept 'n' integers from user into an array and display them one in each line.*

```
import java.util.*;
class Array
{
  public static void main(String args[ ])
  {
        int n,i;
        Scanner sc = new Scanner (System.in);
        System.out.print("Enter no of elements:");
        n=sc.nextInt();
        int a[]=new int [n];
        for(i=0;i<=n-1;i++)
        {
         System.out.print("Enter a no:");
         a[i]=sc.nextInt();
        }
       for(i=0;i<=n-1;i++)
       {
         System.out.println(a[i]);
       }
  }
}
```

```
  {
   System.out.println(a[i]);
  }
 }
}
```

Output

```
Enter no of elements:5
Enter a no:4
Enter a no:6
Enter a no:1
Enter a no:85
Enter a no:9
After Sorting
1
4
6
9
85
```

Explanation

- Bubble sorting algorithm says that compare the consecutive elements from the beginning of array till the end. Wherever the proper sorting is not found, swap the numbers. And this entire set of comparisons is to be repeated for n-1 times. Let us take the example of numbers as given in the output of the program to understand this logic.
- The first two elements are compared i.e. 4 and 6, since they are already sorted i.e. 4 < 6, no swapping takes place. The next comparison is the next two elements i.e. 6 and 1; now these are not sorted i.e. 6 > 1, hence these are swapped. This continues until the last two elements are compared as shown in Fig. 3.1.2.
- You will notice in Fig. 3.1.2, after all the comparisons done in the first iteration; the largest number has reached to the last position. But the numbers are still not sorted properly. To sort these numbers we need to repeat the above process for n-1 times.

Note : Since the "j" and the "j + 1" elements are compared the value of "j" must go maximum upto n – 2. When the value of "j" is n – 2, we will be comparing the n – 2 and n-1 i.e. the last two elements. Also since the value of "i" is starting from 0, we have to go upto n – 2, as we have to perform n–1 comparisons.

	Initial values	After first comparison	After second comparison	After third comparison	After fourth comparison
Iteration 1		4>6, No, hence no swap	6>1, Yes, hence swap	6>85, No, hence no swap	85>9, Yes, hence swap
	4	4	4	4	4
	6	6	1	1	1
	1	1	6	6	6

	Initial values	After first comparison	After second comparison	After third comparison	After fourth comparison
	85	85	85	85	9
	9	9	9	9	85
Iteration 2		4>1, Yes, hence swap	4>6, No, hence no swap	6>9, No, hence no swap	9>85, No, hence no swap
	4	1	1	1	1
	1	4	4	4	4
	6	6	6	6	6
	9	9	9	9	9
	85	85	85	85	85
Iteration 3		1>4, No, hence no swap	4>6, No, hence no swap	6>9, No, hence no swap	9>85, No, hence no swap
	4	1	1	1	1
	1	4	4	4	4
	6	6	6	6	6
	9	9	9	9	9
	85	85	85	85	85
Iteration 4		1>4, No, hence no swap	4>6, No, hence no swap	6>9, No, hence no swap	9>85, No, hence no swap
	4	1	1	1	1
	1	4	4	4	4
	6	6	6	6	6
	9	9	9	9	9
	85	85	85	85	85

Fig. 3.1.2 : Bubble sorting example

- You may also notice in the Fig. 3.1.2, that the sorting is already completed after the second iteration. The remaining two iterations do no swapping. But, we have to consider the worst case condition i.e. if all the numbers entered by the user were in exactly reverse order. In this case we would require all the four iterations. This is explained in the Fig. 3.1.3.
- Here, you will notice that till the last iteration the swapping is required. The worst case condition is one in which the numbers are exactly in the reverse order as expected to be. You will notice in this case the numbers are in descending order and they are to be sorted in ascending order.

	Initial values	After first comparison	After second comparison	After third comparison	After fourth comparison
Iteration 1		5>4, Yes, hence swap	5>3, Yes, hence swap	5>2, Yes, hence swap	5>1, Yes, hence swap
	5	4	4	4	4
	4	5	3	3	3

	Initial values	After first comparison	After second comparison	After third comparison	After fourth comparison
	3	3	5	2	2
	2	2	2	5	1
	1	1	1	1	5
Iteration 2		4>3, Yes, hence swap	4>2, Yes, hence swap	4>1, Yes, hence swap	4>5, No, hence no swap
	4	3	3	3	3
	3	4	2	2	2
	2	2	4	1	1
	1	1	1	4	4
	5	5	5	5	5
Iteration 3		3>2, Yes, hence swap	3>1, Yes, hence swap	3>4, No, hence no swap	4>5, No, hence no swap
	3	2	2	2	2
	2	3	1	1	1
	1	1	3	3	3
	4	4	4	4	4
	5	5	5	5	5
Iteration 4		2>1, Yes, hence swap	2>3, No, hence no swap	3>4, No, hence no swap	4>5, No, hence no swap
	2	1	1	1	1
	1	2	2	2	2
	3	3	3	3	3
	4	4	4	4	4
	5	5	5	5	5

Fig. 3.1.3 : Worst case condition of sorting numbers using Bubble sorting

3.2 Multi-dimensional Arrays

- Multi dimensional arrays are used to store data that requires multiple references for e.g. a matrix requires two references namely row number and column number. Hence, "matrix" is a best example of two dimensional arrays.
- We can also have an array of more than two dimensions. But, we will not require an array of more than two dimensions as per our syllabus.
- If an two dimensional array i.e. matrix is to be declared of a size 3×3, then the declaration statement will be as below :
  `int a[][] = new int [3][3];`
- The size of the array will be 3×3, but the indices will be from 0 to 2 in both the rows and columns.

- The representation of the same can be as shown in the Fig. 3.2.1.

	0	1	2
0	a[0][0]	a[0][1]	a[0][2]
1	a[1][0]	a[1][1]	a[1][2]
2	a[2][0]	a[2][1]	a[2][2]

Fig. 3.2.1 : Arrangement of the elements in a two dimensional array

- As seen in the Fig. 3.2.1, the first brackets indicate the row number while the second brackets indicate the column number.
- To understand how a matrix element can be accessed Fig. 3.2.1 explains it all, but the elements of this matrix are stored in a different manner in the memory as shown in Fig. 3.2.2. This figure assumes the values in matrix as 1, 2, 3 ... 9.

	Memory
a[0][0]	1
a[0][1]	2
a[0][2]	3
a[1][0]	4
a[1][1]	5
a[1][2]	6
a[2][0]	7
a[2][1]	8
a[2][2]	9

Fig. 3.2.2 : Arrangement of the elements in a two dimensional array

- As shown in Fig. 3.2.2, the elements are stored in a sequential manner with all the elements of a row together one below the other, and then the next row and so on.
- The method of accepting elements of an m × n matrix and displaying it in natural form is shown in Program 3.2.1

  (**Note :** m is the number of rows and n is the number of columns).
- The values of m and n must be taken from user. Natural form display of a 2 × 4 matrix is as shown below.

  1 2 3 4

  5 6 7 8

Program 3.2.1 : Write a program to accept an m × n matrix and display it in natural form.

```
import java.util.*;
class Matrix
{
    public static void main(String args[])
    {
      int m,n,i,j;
     Scanner sc = new Scanner (System.in);
     System.out.print("Enter no of rows and columns:");
     m=sc.nextInt();
     n=sc.nextInt();
```

- Hence, in the Fig. 3.2.4 you will notice that to calculate each term of the resultant matrix, we need to add many product terms. This is what is implemented by the three level nested loops. Initially the element of the resultant matrix is made zero, then each of the products are added to this element and hence obtain the entire expression value.
- To find each element, you will notice that the column number of matrix1 is changed while the row number for matrix 2.
- For example in the first term, ($a_{0,0} \times b_{0,0}$) + ($a_{0,1} \times b_{1,0}$), as seen the column number of matrix 1 is 0 for the first term and 1 for the second term, while the row number of matrix 2 is 0 for the first term and 1 for the second term.
- Hence, the third nested loop varies the value of k from 0 to n-1 i.e. the number of columns for matrix 1 or number of rows for matrix 2.

3.3 Strings

- Strings used to be array of characters in C/C++. But in Java, we have been using them and we have seen many times that it is a class. And the variable of string is actually an object of the class String. The advantage is that, the class String has many member methods, that help making our programs very simple especially to handle the strings type data.
- There is also a method to convert a String object into an array of characters and then handle it in the same manner as was done in C/C++. We will also go through these methods and in some programs we will handle strings as an array of character type data.
- We will first go through some of the important methods supported by the class String and then make some programs using those methods.

3.3.1 Methods of String Class

- The String class has a number of methods for its objects. Some of them are listed in the table below with their operation.

Method	Operation
.length()	Returns an Integer value equal to the length of the string
.toLowerCase()	It returns the string object through which it is called with the string converted into lower case
.toUpperCase()	It returns the string object through which it is called with the string converted into upper case
.trim()	This method removes the blank spaces at the end of a string.
.replace(char, char)	This method replaces all the occurrences of the first character with the second character passed to the method.
.equals(String)	This method compares the string object through which it is called and the string object passed to this method. It returns true if the two strings are equal else returns false.
.equalsIgnoreCase(String)	This method compares the string object (ignoring their case i.e. upper case or lower case doesn't matters) through which it is called and the string object passed to this method. It returns true if the two strings are equal else returns false.
.compareTo(String)	This method compares the string object through which it is called and the string object passed to this method. It returns '0' if the two strings are equal. It returns positive value if the string object through which it is called is greater than the second else returns a negative value.
substring(int)	This method returns a sub-string of a string object starting from the index mentioned in the brackets.

Method	Operation
substring(int,int)	This method returns a sub-string of a string object starting and ending from the indices mentioned in the brackets. **Note :** the ending index given is always index of ending +1 instead of ending index.
.concat(String)	A string to be concatenated is passed to the method in the brackets. This method concatenates the two strings and returns the result as an object which is concatenated strings.
.charAt(int)	Returns the character at the index passed in the brackets to the method.
indexOf(char)	This method returns the index of the character passed in the brackets to the method
indexOf(char, int)	This method returns the index of the character passed in the brackets to the method after the index of the integer passed in the brackets.
.toCharArray()	This method converts the String object into an array of characters and returns this array of characters.

- We will use these methods in the following programs and understand their operation much better.

Program 3.3.1 : Write a program to convert a user entered string to upper case.

```
import java.util.*;
class UpperCase
{
  public static void main(String args[])
  {
    String str;
    Scanner sc = new Scanner (System.in);
    System.out.println("Enter a String:");
    str=sc.nextLine();
    str=str.toUpperCase();
    System.out.println(str);
  }
}
```

Output

```
Enter a String:
Hi how are you
HI HOW ARE YOU
```

Explanation

- The string is simply accepted from the user and then it is converted to upper case using the method discussed earlier.
- The returned string is put into the same object and then displayed.

3.4 Methods in Java

- When some task is to be executed many times in a program, a method must be written for the same; and this method must be called whenever that task is to be executed, instead of writing the same code again and again.
- We will see how to write methods in Java. The syntax for declaring a method or writing the method header line is as given
  below : return_type method_name(parameter_list_to_be_accepted_by_the_method_with_their_data_ types);
- Here the return_type indicates the data type of the data that will be returned by the method. A method is normally called to perform an operation and return an answer. The data type is of this data to be returned by the method. A method that doesn't return any data is called as a void method and its return type is void.
- The method_name can be anything as far as it satisfies the rules for identifiers seen in chapter 2.
- The parameters that are to be passed to the method will be accepted by the method in the variables listed in the brackets along with the method. The variables in which the parameter values are received are called as **formal parameters**. The parameter whose values are passed to the method, are called as **actual parameters**. The number of actual parameters and formal parameters must always be the same. The data type of the actual and formal parameters must also be the same. We will see the details of these formal and actual parameters in the first program itself of methods i.e. Program 3.4.1.

Program 3.4.1 : Write a program to add two numbers using a method.

```
import java.util.*;
class Add
{
  public static void main(String args[])
  {
    int a,b,sum;
    Scanner sc = new Scanner (System.in);
    System.out.println("Enter two numbers:");
    a.=sc.nextInt();
    b=sc.nextInt();
    sum=add(a,b);
    System.out.println("Sum="+sum);
  }
  static int add (int x, int y)
  {
    return(x+y);
  }
}
```

Output

```
Enter two numbers:
4
5
Sum=9
```

Explanation

- This program shows the real advantage of methods. The factorial is to be found of 5 different numbers. And instead of writing the same logic of factorial calculation five times, we have written it just once, and called it five times.
- First time we call the method facto() to calculate the factorial of 'n', then 'r' and then 'n-r' for the calculation of the value of ${}^{n}C_{r}$
- Similarly we call the method to calculate the factorial of new values of 'n' and 'r' for the calculation of ${}^{n}P_{r}$.

3.5 Recursive Methods

- A method that calls itself is called as recursive method.
- A recursive method is normally made of an if-else statement. This if-else statement has a condition to decide whether the method must be called again or it should exit from this loop of calling the method again and again.
- Thus the if-else condition may be implemented in such a manner that if the condition is satisfied then the method must be called again and if the condition is not satisfied then it must exit from this loop of calling the method again and again.
- The condition could also be reverse of what is given in the above statement i.e. if the condition is satisfied then it must exit from this loop of calling the method again and again while if the condition is not satisfied then the method must be called again.
- Let us see some program examples of recursive methods.

Program 3.5.1 : Write a program to find the factorial of a number, using a recursive function.

```
import java.util.*;
class Factorial
{
 public static void main(String args[])
 {
  int n,fact;
  Scanner sc = new Scanner (System.in);
  System.out.println("Enter the value of n:");
  n=sc.nextInt();
  fact=facto(n);
  System.out.println("Factorial="+fact);
 }
 static int facto (int n)
 {
  if (n==1)
  return 1;
  else
  return(n * facto(n-1));
 }
}
```

Output

```
Enter the value of n:
5
Factorial=120
```

Explanation

- The number "n" is passed to the method facto() to find the factorial. If the value of the variable "n" received by the method is equal to 1, then the method returns 1, else it calls the same method itself again by passing the value of the variable "n" as "n"-1 and multiplying it by the variable "n".
- Let us understand this with an example. This is demonstrated in Fig. 3.5.1. The figure assumes that the number entered by the user is 5.

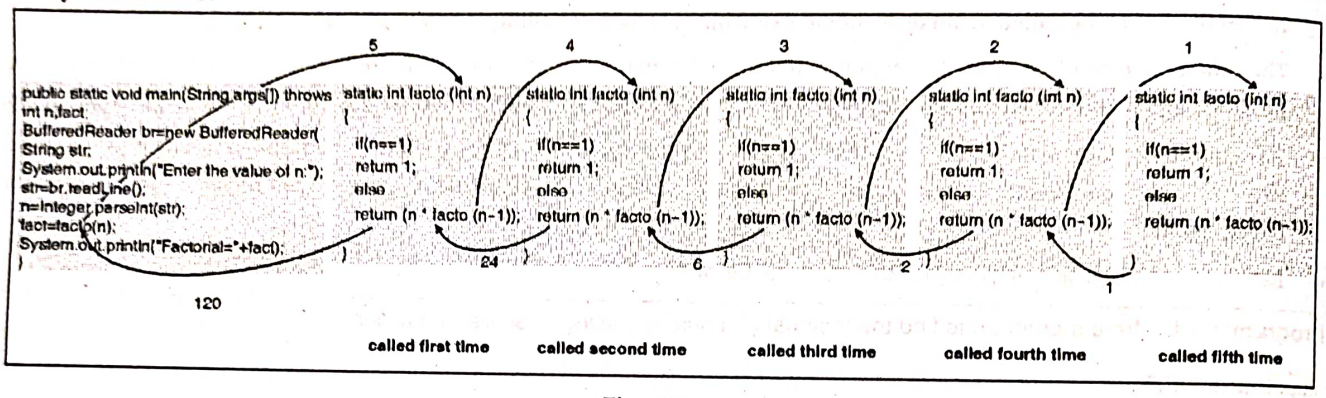


Fig. 3.5.1

- As said, we have assumed that the number entered by the user is 5. Now the method "main" calls the method "facto" for the first time with passing the value of the variable "no" as 5. This is shown in the left part of the figure. The method checks the value of "n" to be equal to 1. Since the condition is not true, the control goes to the else part of the if-else statement. Here the value to be returned is the value of "n" multiplied with the value to be obtained by calling the same method again to find the factorial of 4 (as the factorial of 5 can be written as 5 multiplied with the factorial of 4).
- Hence, the same method is called again (in the Fig. 3.5.1, another method is shown just for understanding purpose), with the value being "n" – 1 i.e. 5 – 1 = 4.
- This time the method again checks the value of "n" to be equal to 1. Since the condition is false, it again goes to the else part and calculates the value to be returned as "n" multiplied with the factorial of "n"– 1 i.e. 3. Hence again calls the same method again.
- The third time when the method is called, the value received in the variable "n" is 3. Again in this case it is not equal to 1, hence it goes to the else part. The same is repeated i.e. calling the method again with the value being passed as 2. This time again since the value of the variable "n" is not equal to 1, it goes to the else part calling itself again with the value of "n" as 1.
- This time the received value of the variable "no" is equal to 1, hence the condition is true and the statements under "if" are executed. The statement says to return the value 1.
- This value is received by the statement "return (n * facto (n – 1))" in the case where the method was called in the fourth time of Fig. 3.5.1. This method now multiplies this received value i.e. 1 with the value of "no" in this case i.e. 2. Hence, the value 2 is returned back to the case where this method was called in the third time of the Fig. 3.5.1. In this case the value to be returned is again calculated as the value received multiplied by the value of the variable "n" in this case i.e. 2 multiplied by 3, returning the result i.e. 6.

Program 3.6.3 : Write a program to find value of y using recursive method, where $y = x^n$.

```
import java.util.*;
class Power
{
   public static void main(String args[])
   {
     int x,n,y;
     Scanner sc = new Scanner (System.in);
     System.out.println("Enter the value of x and n:");
     x=sc.nextInt();
     n=sc.nextInt();
     y=pow(x,n);
     System.out.println("Ans=" +y);
   }
   static int pow(int x, int n)
   {
     if (n==1)
     return x;
     else
     return(x * pow(x,n-1));
   }
}
```

Output

```
Enter the value of x and n:
3
4
Ans=81
```

Explanation

- In this program to get the value of x^n, by just multiplying "x" with x^{n-1}. This is continued until we get x^1.
- This is done by the recursive method "exponential".
- It checks the value of the index, if it is equal to 1, then it returns just the value of "x"; else it returns the value of x^{n-1} by calling itself i.e. recursive method.

3.6 Introduction to Inheritance

- The mechanism of deriving a class from another class is known as inheritance.
- The class from which another class is derived is called as the base class or super class while the class which is derived is called as derived class or sub class.
- Although, we have already seen about the access specifiers in chapter 6, we will briefly go through the same once again.
- The members of a class may be public, protected or private and the class may also be derived in the following ways :

1.	public	2.	protected		
3.	private	4.	private protected	5.	default

- These keywords viz. public, protected, private, and default are called as "access specifiers", as they decide the access of a member.
- The visibility of class members within the program is as shown in Table 3.6.1.

Table 3.6.1 : Visibility of class members in the program

Access Modifier → / ↓ Access Location	public	protected	default (friendly)	private protected	private
Same class	Yes	Yes	Yes	Yes	Yes
Sub class in same package	Yes	Yes	Yes	Yes	No
Other classes in same package	Yes	Yes	Yes	No	No
Subclass in other packages	Yes	Yes	No	Yes	No
Non-subclasses in other packages	Yes	No	No	No	No

- Hence, you will notice in the Table 3.6.1, the public members are visible in the entire program and also in other packages.
- The protected members are accessible in the same package as well in the sub classes of other packages.
- The default or friendly members are accessible only in the same package, be it a derived or non-derived class.
- The private protected members are accessible in the same class or derived classes.
- The private members are accessible only in the same class.
- There are various types of inheritance namely:

1. Single Inheritance
2. Multilevel Inheritance
3. Hierarchical Inheritance

- To derive a class from another we need to use the following syntax :

```
class derived_class_name extends base_class_name
```

- Java doesn't supports Multiple and hence Hybrid Inheritances. These can be slightly made possible using interfaces seen in the further sections of this chapter.
- Let us see these types of inheritance in the subsequent sub-sections.

3.7 Single Inheritance

- In this case only one class is derived from another class.
- The class diagram and a program example is given in Fig. P.3.7.1.

Program 3.7.1 : Write a program to add two numbers using single inheritance such that the base class method must accept the two numbers from the user and the derived class method must add these numbers and display the sum.

Solution :

The class diagram of this requirement is as shown in Fig. P. 3.7.1. This diagram shows that the class "Sum" is derived from class "Data".

3.8 Multi Level Inheritance

- In this case one class is derived from a class which is also derived from another class. The class diagram of such a inheritance can be as shown in Fig. 3.8.1.
- In this case, class 'C' is derived from class 'B' which is derived from class 'A'. Let us see some program examples using multilevel inheritance.

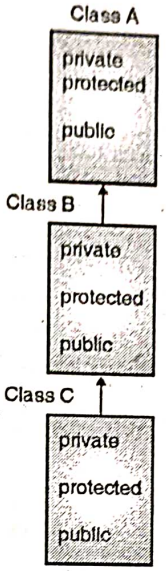


Fig. 3.8.1 : Multi level inheritance

Program 3.8.1 : Write a program to calculate volume of sphere using multi level inheritance. The base class method will accept the radius from user. A class will be derived from the above mentioned class that will have a method to find the area of a circle and another class derived from this will have methods to calculate and display the volume of the sphere.

Solution : The class diagram of this requirement is as shown in Fig. P. 3.8.1.

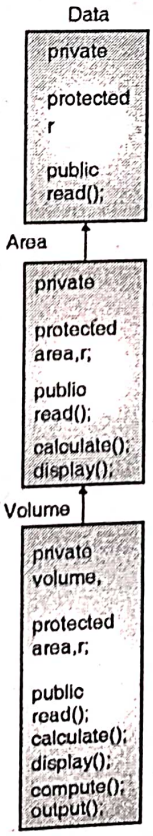


Fig. P. 3.8.1 : Class Diagram of Multi Level Inheritance for Program 3.8.1

3.9 Hierarchical Inheritance

- When multiple classes are derived from a class and further more classes are derived from these derived classes.
- It is like hierarchy father, his children, their children and so on.
- Hence, many a times the derived class is also called as child class while the base class is called as parent class
- Let us see a program example of hierarchical inheritance.

Program 3.9.1 : Write a program to define the following inheritance relationship.

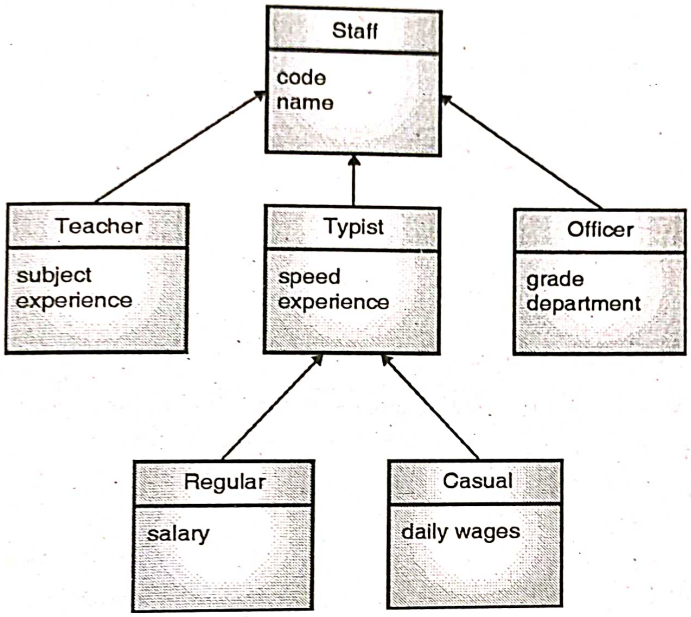


Fig. P. 3.9.1 : Class Diagram of Hierarchical Inheritance for Program 3.9.1

```
import java.util.*;
class Staff
{
        protected String name;
        protected int code;
}
class Teacher extends Staff
{
        private String subject;
        private int experience;
        public void read()
        {
                Scanner sc = new Scanner (System.in);
                System.out.println("Enter name, code, subject and experience of the teacher:");
                name=sc.next();
                code=sc.nextInt();
                subject=sc.next();
```

```
                    o.read();
                    o.display();
                    break;
                    case 3:Regular r=new Regular();
                    r.read();
                    r.display();
                    break;
                    case 4:Casual c=new Casual();
                    c.read();
                    c.display();
                    break;
                    default:System.out.println("Invalid choice");
                }
            }
}
```

Output

```
1. Teacher
2. Officer
3. Regular Typist
4. Casual Typist
Enter the choice, whose details you want to enter:
2
Enter name, code, department and grade of the officer:
Ajay
325
Accounts
1
Officer Details:
Name:Ajay
Code:325
Department:Accounts
Grade:1
```

3.10 Method Overriding

- If a class has multiple methods with same name but different parameter list, then it is called as method overloading.
- If a base class and the derived class have a method with the same name but different parameters, then also it is called as method overloading. But, if this member which has the same name in the base class (or super class) and the sub class has same parameter list, then it is called as method overriding.
- The method should have same name and same parameter list to be called as method overriding. Method overloading will be discussed in this chapter later in Section 3.14.2.
- Let us see a program example of method overriding.

```
        x=sc.nextFloat();
        Volume a=new Volume();
        a.read(x);
        a.calculate();
        a.compute();
        a.display();
    }
}
```

Output

```
Enter the radius:
10
Volume=4186.6665
```

Explanation

- The implementation Program 3.10.1 is done with same method display() in base class Area and in sub class Volume.
- Here the method display() of the derived class i.e. "Volume" overrides the method display() of the base class "Area".

3.11 Keyword "final" and Final class

- The keyword final can be preceded to any member of a class or to the class itself.
- Making anything final has following implications.
  - If a field member is declared as final then the variable value cannot be changed i.e. it becomes a constant.
  - If a method is declared as final then that method cannot be overridden.
  - If a class is declared as final then that class cannot have any sub class i.e. no class can be derived from the final class.
- Let us see a program example of a final method.

Program 3.11.1 : Write a program to display volume of sphere and hemisphere. Make use of final method to display the volume.

```
import java.util.*;
class Base
{
  protected float r,vol;
  public void read(float x)
  {
    r=x;
  }
  final public void display()
  {
   System.out.println("Volume="+vol);
  }
```

Explanation

- The implementation program is done with final method display() in base class and hence, cannot be again defined in sub classes Sphere and Hemisphere i.e. it cannot be overridden.
- The base class "Base" has the method display() declared as "final", hence the derived class members cannot override this method.

3.12 Java Abstract Class and Method

- Abstract classes are used to declare common characteristics of subclasses.
- Abstract classes are declared with the keyword **'abstract'** preceding the class definition. Abstract classes are used to provide a template for subclasses.
- No object can be made of an abstract class. It can be used as a base class for other classes that are derived from the abstract class.
- An abstract class can contain fields and methods.
- An abstract class can have methods with only declaration and no definition. These methods are called as abstract methods. The abstract method must be declared as shown in Program 3.12.1. If a class has any abstract method, the class becomes abstract and must be declared as abstract. Abstract methods provide a template for the classes that are derived from the abstract class.
- Any method definition can be overridden by subclasses, but the abstract methods must be overridden.

Program 3.12.1 : Write a program to display volume of sphere and hemisphere. Make use of abstract class.

```
import java.util.*;
abstract class Base
{
    protected float r,vol;
    public void read(float x)
    {
      r=x;
    }
    public abstract void calculate();
    public void display()
    {
       System.out.println("Volume="+vol);
    }
}
class Sphere extends Base
{
    public void calculate()
    {
       vol=3.14f*r*r*r*4/3;
    }
```

Explanation

- The implementation program of the abstract method is done with same methods read() and calculate() in base abstract class and in sub classes Circle, rectangle and triangle.
- The base class has the method read() and calculate() declared as "abstract" making the base class to be "abstract" and also making it compulsory for the derived classes to override these methods.

3.13 Polymorphism

- Polymorphism means multiple forms. We can have multiple forms of a same thing. We will understand it better with the subsequent sections and programs in these sections.
- Polymorphism has two types based on the time of resolution. If the resolution is done during the compiling of the program, then it is called as **static polymorphism**. But if the resolution is done during the run time, then it is called as **dynamic polymorphism**.
- Static polymorphism or early binding includes method overloading and constructor overloading.
- Dynamic polymorphism or late binding includes method overriding and dynamic method dispatch.
- We have already seen constructor overloading in chapter 6 and method overriding in Section 3.10. We will still see these concepts again and with the remaining topics also.

3.14 Static Polymorphism

- As already discussed, those multiple forms (polymorphism) whose resolution is done during the compile time are grouped under static polymorphism.
- Two cases come under static polymorphism namely constructor overloading and method overloading.

3.14.1 Constructor Overloading

Multiple constructors in the same class with different parameters is called as constructor overloading. Let us see some program examples of constructor overloading.

Program 3.14.1 : Write a program to demonstrate constructor overloading.

```
class Rectangle
{
        private int l,b;
        public Rectangle()
        {
                l=b=10;
        }
        public Rectangle(int x)
        {
                l=b=x;
        }
        public Rectangle(int x, int y)
        {
                l=x;
                b=y;
        }
        public void area()
        {
```

```
            System.out.println("Area="+l*b);
        }
}
class Main
{
        public static void main(String args[])
        {
            Rectangle r1=new Rectangle();
            Rectangle r2=new Rectangle(5);
            Rectangle r3=new Rectangle(3,3);
            r1.area();
            r2.area();
            r3.area();
        }
}
```

Output

```
Area=100
Area=25
Area=9
```

Explanation

- The class Rectangle has three constructors with no parameters, one parameter and two parameters respectively.
- There are three objects of the class Rectangle made in the main() method. The first object uses the default constructor, with no parameters. Hence the length and breadth are initialized to 10 each and the area is displayed as 100.
- The second object passes one parameter and hence both length and breadth are initialized to the passed value for the second constructor.
- The third object is made with passing 2 parameters to the constructor, and hence the passed values are initialized and the area displayed accordingly.
- There are multiple constructors in the same class (name has to be same) with different parameter list. This is called as constructor overloading.

3.14.2 Method Overloading

- Multiple Method with same name in the same class or in the base and derived class with different parameters is called as Method overloading.
- Let us see some program examples of Method overloading.

Comparison between overriding method and overloading method

Sr. No.	Method overloading	Method overriding
1.	It can occur in the same class as well as super and sub class.	It can occur only in the super and sub class.
2.	Inheritance is not necessary for method overloading.	Inheritance is necessary for method overriding.
3.	Return type of functions may not be same.	Return type must be same.
4.	Parameters of the overloading functions must be different.	Parameters of the overriding functions must be same.

Sr. No.	Method overloading	Method overriding
5.	Only static polymorphism can be implemented using method overloading.	Static as well as dynamic polymorphism can be implemented using method overriding.
6.	**Example :** Refer Program 3.14.2.	**Example :** Refer Program 3.10.1.

Program 3.14.2 : Write a program to demonstrate Method overloading by overloading the methods for calculating area of circle, rectangle and triangle.

```
class Area
{
        private float x,y,z;
        public float area(float r)
        {
                return(3.14f*r*r);
        }
        public float area(float l, float b)
        {
                return (l*b);
        }
        public float area(float a, float b, float c)
        {
                float s;
                s=(a+b+c)/2;
                s=s*(s-a)*(s-b)*(s-c);
                return(float)(Math.sqrt(s));
        }
}
class Main
{
        public static void main(String args[])
        {
                Area a=new Area();
                System.out.println("Area of circle="+a.area(10));
                System.out.println("Area of Rectangle="+a.area(10,10));
                System.out.println("Area of Triangle="+a.area(10,10,10));
        }
}
```

Output

```
Area of circle=314.0
Area of Rectangle=100.0
Area of Triangle=43.30127
```

Explanation

- The class Area has three methods with the same name i.e. area(). The methods have different number of parameters. This is called as method overloading.

Program 3.14.4 : Write a program to demonstrate Method overloading in a base and derived class by overloading the methods for displaying the value of a variable contained in the class.

```
class Parent
{
        public void display(int x)
        {
                System.out.println("x="+x);
        }
}
class Child extends Parent
{
        public void display(int x, int y)
        {
                System.out.println("x="+x+"\ny="+y);
        }
}
class Main
{
        public static void main(String args[])
        {
                Child c=new Child();
                c.display(10);
                c.display(5,5);
        }
}
```

Output

```
x=10
x=5
y=5
```

Explanation

- The class Parent has a method display() with one parameter.
- Class "child" extended from the class "Parent" has the same method display() with two parameters.
- When one parameter is passed you will notice the base class method is called while when two parameters are passed you will notice that the derived class method is called.
- Here the methods with same name and different parameter list are not in the same class but in the base and derived class. This is also called as Method overloading.

3.15 Dynamic Polymorphism

- Dynamic polymorphism as already discussed has two types, the method overriding and dynamic dispatch of methods.
- The method overriding concept is already discussed section 3.13. We will discuss here the concept of Dynamic Dispatch of methods.

3.15.1 Dynamic Method Dispatch

- In case of method overriding, when a method is called by an object of the class the method of the corresponding class is called.
- A base class reference object can be used to refer either a base class object or a derived class object.

3.16 The finalize() Method Instead of Destructor in Java

- The C++ programming language had the destructors. But in Java we do not have destructors. As discussed in the features of "Java" programming language in Chapter 1, "Java is garbage collected", i.e. the objects are automatically destroyed when their use is over. Hence there is no need of having destructors.
- But, there may be certain operations to be carried out when an object is destroyed. For example, if the object was using a particular resource of the system, the resource is to be surrendered. This can be done using the finalize() method.
- Let us see a program example of this method.

Program 3.16.1 : Write a program to demonstrate finalize() method.

```
class Demo
{
        public void display(int x)
        {
                System.out.println("x="+x);
        }
        protected void finalize()
        {
                System.out.println("In finalize method of class Demo");
        }
}
class Main
{
        public static void main(String args[])
        {
                Demo d=new Demo();
                d.display(10);
                d=null;
                System.gc();
        }
}
```

Output

```
x=10
In finalize method of class Demo
```

Explanation

- An object "d", is made of the class Demo. The object is used to call the display() method.
- Then the object is given "null" or made null. This is to indicate the JVM that the use of this object is over.
- The static method gc() of the class System is called. This method is called as garbage collection method.
- Although garbage collection happens at regular interval in a program, but since our program is very small and for demonstration purpose, we need to call this method.
- You will notice that the statement "In finalize method of class Demo", is automatically printed, without exclusively calling the finalize() method. Thus when the garbage collection happens, the finalize() method is automatically called.

3.17 The Super Keyword

- If you want to access a member of a base class from the derived class, then the keyword "**super**" is used.
- This is especially to access the constructors and method members of the super (or base) class.
- Let us see a program example that uses the super keyword to call the constructor and another method of the base class.

Program 3.17.1 : Write a program to demonstrate the use of "super" keyword.

Solution :

```
class Parent
{
        public Parent(int a)
        {
                System.out.println("In constructor of Parent class: "+a);
        }
        public void display(int x)
        {
                System.out.println("x="+x);
        }
}
class Child extends Parent
{
        public Child(int a)
        {
                super(a);
        System.out.println("In constructor of Child class: "+a);
        }
        public void display(int y)
        {
                super.display(y);
                System.out.println("y="+y);
        }
}
class Main
{
        public static void main(String args[])
        {
                Child c=new Child(3);
                c.display(5);
        }
}
```

Output

```
In constructor of Parent class: 3
In constructor of Child class: 3
x=5
y=5
```

Explanation

- An object "c" is created of the child class. The constructor as we know is automatically called. In the child class, we have also called the constructor of the "Parent" class, by simply using the keyword super(). When the keyword super, is written, without specifying the method name, then the constructor of the base class is called. The parameterized constructors of the parent and child classes are passed with the parameters
- The display() method is then called for the child class. Again using the keyword "super", the display() method of the parent class is called. But, in this case, we have specified the method name of the super class to be called in the statement super.display().

□□□

Interfaces and Packages

4.1 Interface

4.1.1 Introduction

- The multiple inheritance problem is solved in Java with the help of Interface.
- Multiple Inheritance is possible when extending interfaces i.e. one interface can extend as many interfaces as required. Java does not allow multiple inheritance, but it allows to extend one class and implement as many interfaces as required. A class that implements an interface has to either define all the methods of the interface or declare abstract methods and the method definitions may be provided in the subclass.

4.1.2 Extending an Interface

- Interface are used to define a general template and then classes can implement the interface and can hence inherit the properties of the "interface".
- Interfaces just specify the method declaration and can also contain fields.
- The methods are implicitly public and abstract.

4.1.3 Variables in Interface

- The fields are implicitly public static final.
- The definition of an interface begins with a keyword interface.
- No object can be made of an interface i.e. it cannot be instantiated.

4.1.4 Difference between Interface and Abstract Class

Sr. No.	Abstract class	Interface
1.	Abstract class is a class which contains one or more abstract methods, which are to be defined by sub classes.	An Interface can contain only method declarations and no definition.
2.	Abstract class definition begins with the keyword "**abstract**" keyword followed by Class definition.	An Interface definition begins with the keyword "**interface**" followed by the interface definition.
3.	Abstract classes can have general methods defined and some methods with only declaration defined in the subclasses.	Interfaces have all methods with only declarations defined in subclasses i.e. classes implemented from the interface.
4.	Variables in Abstract class need not be public, static and final.	All variables in an Interface are by default public, static and final.

Sr. No.	Abstract class	Interface
5.	Abstract class does not support Multiple Inheritance.	Interface supports multiple Inheritance.
6.	An abstract class can contain private as well as protected members.	An Interface can only have public members.
7.	When a class extends an abstract class, it need not implement any of the methods defined in the abstract class.	A class implementing an interface must implement all of the methods declared in the interface.
8.	Abstract classes are fast.	Interfaces are slow as it requires extra indirection to find corresponding method in the actual class.

Program 4.1.1 : Write a program to display volume of sphere and hemisphere. Make use of interface to define the template of methods to be there in the derived classes.

```
import java.util.*;
interface Base
{
    public void read(float x);
    public void calculate();
    public void display();
}
class Sphere implements Base
{
   protected float r,vol;
   public void read(float x)
   {
     r=x;
   }
   public void calculate()
   {
    vol=3.14f*r*r*r*4/3;
   }
   public void display()
   {
    System.out.println("Volume="+vol);
   }
}
class Hemisphere implements Base
{
```

```
}
class Main
{
  public static void main (String args[ ])
  {
     float x,y;
     er sc = new Scanner (System.in);
     println("Enter the length and breadth:");
     ();
     t();
     Square s=new Square();
     s .read(x);
     s.calculate();
     System.out.println("Square:");
     s.display();
     Rectangle h=new Rectangle();
     h.read(x,y);
     h.calculate();
     System.out.println("Rectangle:");
     h.display();
  }
}
```

Output

```
Enter the length and breadth:
10
20
Square:
Area=100.000
Rectangle:
Area=200.000
```

4.2 Introduction to Packages

- We have been writing small program until now. If we need to make a huge software, we must not put all the classes in the same package. We must have a package with a set of classes and then we can import the package to use the members of those classes.
- Package is a way to organize the code i.e. the similar function or related classes must be in a package.
- Thus one folder or package will have all the classes of similar functions while another folder or package for another type of classes. For e.g. all printer related classes in one package, all user input based classes in one package and so on.

Advantages of Packages

1. Since the classes of similar type are organized together, it becomes easy to access them. It also makes it easy for debugging to locate error if any.
2. It becomes easy for the coding purposes also.
3. Packages also avoid the clashes of using the same name classes. You can have classes with same name in different packages. Let us see the use of packages and some in built packages in JDK.

4.3 Creating a Package

- To create a package we need to use the keyword "package". We need to begin the program with this keyword and the name of the package. We need to also make a folder to store this package.
- Let us see the steps to be followed to make a package.

Step 1 : We need to specify the key word "package" followed by the name for the package.

Step 2 : We need to make a class and write the members of the class.

Step 3 : Store the program file in a folder with the same name as that of the package and store the file as the name of the class that has the main() method. For example if the package name is "world", then the folder in which the program will be stored should also be "world". If the name of the class having the main() method is "Hello", then the program should be stored as "Hello.java"

Step 4 : Compile the program as usual i.e. using "javac" followed by the class name. This has to be done in the folder of the package. In case of above example, in the folder "world" you need to compile the program with the statement "javac Hello.java"

Step 5 : Come out of the package folder and execute the program with the syntax: **java Package_name.class_name.** For example in the above case: java world.Hello

Let us see these steps in more details with a program example.

Program 4.3.1 : Write a program to create a package and show the execution of the same.

```
package World;
class Hello
{
        public static void main(String args[ ])
        {
                System.out.println("Hello Friends");
        }
}
```

Output

```
C:\java programs>cd world
C:\java programs>javac Hello.java
C:\java programs>cd..
C:\java programs>java World.Hello
Hello Friends
C:\java programs >
```

Explanation

- The output is shown with the entire process to be followed for executing the program. The program is written as explained in the steps 1 & 2 above. The program is to be stored in a proper folder as explained in step 3.
- Then the steps 4 and 5 can be seen in the output. First the program is compiled in the folder "world" i.e. the name of the package.
- Then the execution is done outside the folder with a special syntax as shown in step 5.

Program 4.3.2 : Write a program to create a package and show the execution of the same. The program should display numbers from 1 to 10.

```
package numbers;
class Integers
{
        public static void main(String args[])
        {
                int i;
                for(i=1;i<=10;i++)
                {
                        System.out.println(i);
                }
        }
}
```

Output

```
C:\java programs>cd numbers
C:\java programs>javac Integers.java
C:\java programs>cd..
C:\java programs>java numbers.Integers
1
2
3
4
5
6
7
8
9
10
C:\java programs>
```

Explanation

Similar program with a slight changes in the name and the main() method operation. Here the main() method displays the numbers from 1 to 10 using a simple "for" loop.

4.4 Creating a Sub-Package

- The same process is to be used to create a sub package. You need to just specify that it is a package inside a package by the statement as given in the Program 4.4.1 i.e. "package world.india".
- Also another care to be taken is that the folder of the sub package name has to be stored inside the folder with the package name. This is demonstrated in Program 4.4.1

Output

```
C:\java programs>cd prog 4.4.2
C:\java programs>cd world
C:\java programs >cd india
C:\java programs>javac Hello.java
C:\java programs >cd..
C:\java programs >cd..
C:\java programs>java world.india.Hello
Hello Indians
C:\java programs>
```

Explanation

Here an interface is created and a class is inherited or implemented from this interface. The display() method is executed according to the operation specified in the program.

4.5 Importing a Package

- We imported some packages in all our programs like java.io, java.util etc. These were inbuilt packages. Let us see how to import our self made package.
- The package is to be made in the same manner. This package is to be imported by another program outside the folder with the package name.
- For example we can make another program and call the display method from there. Let us see a program for the same.

Program 4.5.1 : Write a program to create a package, import it in another program and call the method in the package.

Note : Here we need to make two programs, one the package and another that imports the package. The package is to be stored as studied earlier in this chapter. Let us see these two programs.

Program 1 : Package

```
package world.india;
public class Hello
{
        public void display()
        {
                System.out.println("Hello Indians");
        }
}
```

Program 2 : Importing package

```
import world.india.*;
class Hello1
{
        public static void main(String args[ ])
        {
                Hello h=new Hello();
                h.display();
        }
}
```

Output

```
C:\java programs>javac Hello1.java
C:\java programs>java Hello1
Hello Indians
C:\java programs>
```

Explanation

- Two programs as seen above are to be written. The first one has to be stored in the folder according to the name of the package as seen in the above sections. Thus, since it is a sub-package the program is to be stored in the "india" folder inside the "world" folder of the main program folder.
- The main program is to be written in the outside folder i.e. the folder that contains the folder "world".
- Here we have imported the package made by us. We have made an object of the class "Hello" and called the member method display(). Since the object is made outside the package, the class is to be declared as "public"; else it will not be accessible in another package.

4.6 The java.lang Package

- We have many in built packages available with the JDK (Java Development Toolkit). Some of the packages used by us quite frequently are "java.lang" and "java.util". In the subsequent sections we will see the classes and some methods supported by these packages.
- The java.lang package has again many classes available in it. This class is so important, that it is found that you can almost have no program without importing this package. Hence this package is imported by default for any program written in Java. Hence you must have noticed in all the programs in this book, we have used the classes and their methods from this package without importing this package.
- One of the types of classes of this package is the wrapper classes. Let us see this in detail.

4.6.1 Wrapper Classes

- The base class is the Number class. It has many sub classes like Byte, Short, Integer, Long, Double and Float.
- All these sub classes of "Number", have methods to convert a primitive data to and from string. For example we have .parseInt() method in the class Integer to convert a string to "int" type data. Similarly in the class "Float" we have the method parseFloat() to convert the string to "float" type data. We also have toString() method in these classes to convert a data into string type object.

- These classes have static methods to convert string class objects into primitive data types. We have been using these methods in almost all our programs i.e. parseInt(), parseFloat() etc.
- Also these classes have methods that are used to convert the primitive data types to an object of a class.
- For example you can convert and "int" value to an object of the wrapper class "Integer". Similarly a "float" type value to an object of class "Float" and so on.
- This is required in the following cases:
  1. As seen in the chapter 6, we cannot change the actual value by passing the variable value, instead we need to pass the object. Objects of primitive data types can implement call by reference
  2. Many classes you need to pass an object of a class as a parameter, you cannot pass a primitive data type, for example in Vector class.
  3. Using objects of primitive data type you can have multiple methods using the object and its value can be accessed across many methods.
  4. Primitive data types cannot be made as a part of the object hierarchy.
  5. When a primitive data becomes an object various methods are available with it.
- The toString() method is used to convert an object to string. When you want to display the values of an object and you just say display the object, the toString() method of that class is automatically called and the operation in that method is performed.
- Let us see a program example for the use of toString() method

Program 4.6.1 : Write a program to make an object of a class and display the object using toString() method.

```
import java.io.*;
class Demo
{
        int x;
        public Demo(int a)
        {
                x=a;
        }
        public String toString()
        {
                return("x="+x);
        }
}
class Main
{
        public static void main(String args[])
        {
                Demo d=new Demo(5);
                System.out.println(d);
        }
}
```

```
        }
        public String toString()
        {
                return("Volume="+vol+"\n");
        }
}
class Size
{
        public String size(Rectangle z)
        {
                return(z.toString());
        }
        public String size(Cube z)
        {
                return(z.toString());
        }
        public String size()
        {
                return("-1");
        }
        public static void main(String args[])
        {
                Cube c=new Cube();
                Rectangle p=new Rectangle();
                Size s=new Size();
                p.Area();
                c.volume();
                System.out.print(s.size(c));
                System.out.print(s.size(p));
        }
}
```

4.6.2 Other Classes in java.lang

- The package java.lang includes the following classes:

Boolean	Long	String
Byte	Math	StringBuffer
Character	Number	System
Class	Object	Thread
ClassLoader	Package	ThreadGroup
Compiler	Process	ThreadLocal
Double	Runtime	Throwable
SecurityManager	Integer	Short
Float	RuntimePermission	
Void	InheritableThreadLocal	

- There are some more classes also in the java.lang package. Anyways, we have been using quite a few classes of these.
- Wrapper classes like Integer, Float, Double etc are been used in our programming.
- The "Thread" class has been widely used in chapter 8. We have seen many methods of this class like setName(), getName(), setPriority(), getPriority(), isAlive(), join(), yield(), sleep() etc.
- We have also used another class i.e. "String" in the chapter 3, with many methods of this class like toUpperCase(), toLowerCase(), compare(), substring() etc
- We have also used a method pow() of the class "Math". Let us see some more methods in this class in the subsequent sub-section.

4.6.3 Math

- This class contains various methods. Some of the important methods required can be grouped as below:
  1. Transcendental Methods
  2. Exponential Methods
- Table 4.6.1 shows the methods in these groups and their operation:

Table 4.6.1

Method	Operation
Transcendental Methods	
double sin(double)	It is a static method that returns the sine of the variable passed to it in radians.
double cos(double)	It is a static method that returns the cosine of the variable passed to it in radians.
double tan(double)	It is a static method that returns the tan of the variable passed to it in radians.
double asin(double)	It is a static method that returns the angle in radians whose sine is passed to the method.
double acos(double)	It is a static method that returns the angle in radians whose cosine is passed to the method.
double atan(double)	It is a static method that returns the angle in radians whose tan is passed to the method.
Exponential Functions	
double exp(double)	It is a static method and it returns the value of 'e' raised to the parameter passed.
double log(double)	It is a static method and it returns the natural logarithm of the parameter passed.
double pow(double, double)	It is a static method and it returns the value of the first parameter passed, raised to the second parameter passed.
double sqrt(double)	It is a static method and it returns the square root of the parameter passed.

- There is another widely used method i.e. the double random(), method. This is also a static method and returns a pseudorandom value. The random value returned by this method is between 0 and 1.

4.7 The java.util Package

- This is another very widely used package besides the "java.lang" package.
- One of the classes of this package used by us is "Vector". A list of all the classes in the package java.util are listed in the Table 4.7.1.

Table 4.7.1

AbstractCollection	Hashtable	TreeMap
EventObject	Scanner	Calendar
PropertyResourceBundleAbstractList	Stack	Observable
GregorianCalendar	ArrayList	TreeSet
Random	LinkedList	Collections
AbstractMap	StringTokenizer	Properties
HashMap	Arrays	Vector
ResourceBundleAbstractSequentialList	ListResourceBundle	Date
HashSet	TimeZone	PropertyPermission
SimpleTimeZone	BitSet	WeakHashMap
AbstractSet	Local	Dictionary

Let us see some of these classes and their methods.

Array list and Linked list

ArrayList

- The class ArrayList supports dynamic arrays i.e. arrays that can grow as and when required.
- The normal arrays in Java are fixed in length. Once they are created, they cannot grow bigger or shrink to smaller, while this can be done on ArrayList class objects.
- It has three constructors viz. a default constructor that builds an empty array, a constructor that takes the collection as arguments and another takes an integer as the size of the list.
- It has various methods like add() to add an element, clone() to make a clone of the list, clear() to remove all the elements etc.

LinkedList :

- The LinkedList class as the name says is used to create a linked list i.e. a list of objects linked together.
- It has two constructors viz. default constructor and a constructor that accepts a collection of objects.
- It has various methods add() to add a new object, addLast() to add at the end, addFirst() to add at the beginning, clone() to create a clone, clear() to clear all the elements, etc

---

Program 4.7.1 : Each year, sleepy Hollow Elementary school holds a "Principal for a Day" lottery. A student can participate by entering his/her name and ID into a pool of candidates. The winner is selected randomly from all entries. Each student is allowed one entry. Implement a student class that encapsulates a student. Implement StudentLottery class with methods addStudents ( ) and pickwinner ( ) and main ( ). **Hint :** Use Random class to pick winner.

```
        StudentLottery sl=new StudentLottery();
        sl.addStudents();
        sl.addStudents();
        sl.addStudents();
        sl.addStudents();
        sl.addStudents();
        sl.addStudents();
        sl.pickWinner();
    }
}
```

4.7.1 Date

- The class "Date" has a field that keeps a track of milliseconds that have elapsed from midnight January 1, 1970. The class has methods that can give the time and date using the calculation done on this field.
- Let us see a program example to demonstrate the use of the "Date" class.

Program 4.7.2 : Write a program to make an object of the class Date and display the date and time.

```
import java.util.*;
class DateTime
{
    public static void main(String args[ ])
    {
        Date d=new Date();
        System.out.println(d);
    }
}
```

Output

```
Fri Dec 16 07:00:55 IST 2011
```

Explanation

- A very simple program. We have simply made an object of the class "Date" and displayed that object using the println() method.
- The output shows the week day, month name, date, time in hours, minutes and seconds. Followed to this the output also shows the time zone and the year.

4.7.2 Calendar

- The calendar class is considered to be better in some cases, because it gives the access to all the fields for date, month, year, hours etc to get the time in our own required format.
- Let us see a program example to demonstrate the same.

Program 4.7.3 : Write a program to make an object of the class Calendar and display the date and time.

```
import java.util.*;
class DateTime
{
        public static void main(String args[ ])
        {
        String monthName[ ] = { "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"};
                Calendar c = Calendar.getInstance();
                System.out.println("Date: "+c.get(Calendar.DATE)
                        +"/"+monthName[c.get(Calendar.MONTH)]+"/"+c.get(Calendar.YEAR));
                System.out.println("Time: "+c.get(Calendar.HOUR) +
                        ":"+c.get(Calendar.MINUTE) + ":"+c.get(Calendar.SECOND));
        }
}
```

Output

```
Date: 16/Dec/2011
Time: 7:12:53
```

Explanation

- The class "Calendar" is an abstract class. Hence we cannot create an object of this class.
- We have used the getInstance() method that returns the object of the sub-class of an abstract class.
- Then we have accessed the static fields of the abstract class "Calendar". The fields like "MONTH", "YEAR", "HOUR", "MINUTE","SECOND","DATE" etc.
- The get() method is used to get the values of these variables.

4.7.3 Vector

- Vector is a class in the package java.util. Vector is a dynamic array. It is almost similar to an array, but with the major difference that it contains legacy methods to help programmer to insert, add or delete an element. It also has methods to search an element, remove an element etc. The size of the vector can be incremented during the execution of the program.
- The vector class has four constructors: a default one and three parameterized constructors. The prototype of these constructors is listed below:

1. Vector()
2. Vector (int size)
3. Vector(int size, int incr)
4. Vector(Collection c)

- The first constructor is the default constructor that takes an initial size of the Vector as 10 elements. The second constructor creates a vector with initial capacity as specified by size variable passed to the constructor.
- The third one has a size as well as variable incr. This variable "incr" indicates the number by which the size should increment in case if a new element is added to the vector and the capacity of the Vector was full. In case of any constructor where the incr. is not specified the incr. value is taken to be same as the size variable value. The fourth constructor accepts an array or collection of elements which it directly initializes in the vector.
- We have seen this in detail in Chapter 3 in the Section 3.5.

4.7.4 Hashtable

- Hash table is used in data structures. It is a kind of linked list.
- When you store a data and you have to link a detailed data using a key.
- The best example of use of Hash table usage is the phone book of your mobile. You select a name and that selects the detailed information of that person. You can store the phone number, date of birth, email id etc of a person.
- A hash table maps keys to values. For example in the above case the name may work as a key and point to the values i.e. the detailed information of that person.

4.7.5 Collection Classes

- The java.util.Collections class consists of static methods that operate on collections. It contains polymorphic algorithms that operate on collections, wrappers and return a new collection.
- The methods throw a NullPointerException in case if collection passed is NULL. Some of the methods in this class are listed below :
  1. Static Boolean addAll (Collection, < list >) : This method adds all the elements passed to the specified collection.
  2. Static int binarySearch (List, T key) : This method returns the element member of the "T key" passed in the specified list using binary search algorithm.
  3. Static void copy (List, List): This method copies the second list passed into the first list.

□□□

Multithreading and Exception Handling

5.1 Introduction to Exception Handling

- Exception handling refers to handling of abnormal or unexpected events.
- Some of these abnormal conditions are known to the java compiler while some occur during run time and are unknown to the compiler.
- 'Exception' is a class and it has sub classes. The different sub classes are the according to exceptions possible in Java. The list of the names of sub classes of the base class Exception is given in the Table 5.2.1. This table lists all the checked and unchecked exceptions of Java and the condition for their occurrences.
- Exception as discussed is an error condition occurrence. For example if a integer is expected and a character is entered, then while parsing the character to integer, an error will occur. This error or exception is called as NumberFormatException. It is one of the sub class of the base class 'Exception'. Since it is name of a class, there is no space between the words.
- Exceptions are classified into two types namely : Checked Exceptions and Unchecked Exceptions.

5.2 Checked and Unchecked Exceptions

5.2.1 Checked Exceptions

- All checked exceptions are sub-classes of the class 'Exception'
- Checked Exceptions makes the programmers to handle the exception that may be thrown.
- For e.g. I/O exception caused because of readLine() method is a checked exception.
- There are two ways of dealing with an exception :
  1. Indicate that the method throws an exception (as we have been doing till now i.e. indicating that the method throws IOException) or
  2. Method must catch the exception and take the appropriate action using the try catch block (will be studied in later sections of this chapter).

5.2.2 Unchecked Exceptions

- Unchecked Exceptions are RuntimeException and its subclasses. These are not sub class of the class Exception.
- The compiler doesn't check for the unchecked exceptions and hence the compiler doesn't make the programmers handle them.
- In fact, the programmers may not even know that such an exception would be thrown.

Table 5.2.1

Exception name	Case in which the exception occurs
Java's Unchecked Exceptions	
ArithmeticException	In case of arithmetic error, such as divide by zero.
ArrayIndexOutOfBoundsException	Accessing an element out of the size of an array i.e. outside its boundary
ArrayStoreException	Assigning an incompatible type element to an array
ClassCastException	Casting not possible or invalid casting
IllegalArgumentException	The formal parameters and actual parameters do not match
NegativeArraySizeException	The variable used to create the array has a negative value hence negative array size.
NullPointerException	Trying to access the members of an object which is not allocated with any memory space. Memory space is allocated to an object using the new operator.
NumberFormatException	The value expected was a number and the data given is not a number.
StringIndexOutOfBounds	Similar to array index out of bounds exception, but for an array of strings.
IllegalMonitorStateException	In case of multi threading, if a thread is waiting for another thread, which is locked.
UnsupportedOperationException	Operation is not supported by Java
Java's Checked Exceptions	
ClassNotFoundException	The class whose object is attempted to be made or static member is tried to be accessed is not found.
IllegalAccessException	Access to the member is illegal
InstantiationException	This exception is generated if you try to create an object of an abstract classs or an interface
InterruptedException	A thread has been interrupted by another
NoSuchFieldException	Attempt to access a field that doesn't exists
NoSuchMethodException	Attempt to access a method that doesn't exists

- Let us see a simple program that generates an exception. We will accept two numbers from user, the dividend and the divisor. The divisor entered by user must be zero, so that it generates the divide by zero error.

Program 5.2.1 : Write a program to perform division of two numbers accepted from user to demonstrate ArithmeticException and also NumberFormatException.

```
import java.io.*;
class Divide
{
public static void main(String args[ ]) throws IOException
```

Output 3

```
Enter two numbers:
5
2
The Quotient=2
```

- We will see in the subsequent sections, the different methods to handle these exceptions.
- There are different keywords to handle the Exceptions. The three ways are
  1. try-catch-finally
  2. throws
  3. throw
- We have been using the second method for many times by now. The first method i.e. try-catch-finally is used to handle an exception generated due to abnormal behavior.
- The second i.e. throws is used to indicate that the method generates an exception.
- The third i.e. throw is used to generate an exception of your own. Let us see them in details with programming example.

5.3 try-catch-finally

Exceptions can be handled using the **try-catch-finally**. The syntax of try-catch-finally is as given below :

```
try
{
statements;
}
catch (Exception_class_name object_name)
{
statements;
}
finally
{
statements;
}
```

try Block

- The statements that you think can generate an error or exception must be placed in this block.
- If an exception occurs then the catch block will be executed and then the finally block will be executed.
- Else if no exception occurs, then the control will directly go to the finally block i.e. the catch block will not be executed. Java code that you think may produce an exception is placed within a try block for a suitable catch block to handle the error.
- If an exception occurs, but a matching catch block is not found, then it reaches to the finally block.
- In any case, when the exception occurs in a particular statement of try block, the remaining statements after this statement of the try block are not executed i.e. no statements of the try block are executed after the exception generating statement.

catch Block

- The exception thrown by the try block are caught by the catch block.
- The type of the exception occurred must match with the exception mentioned in the brackets of the catch block.

finally Block

- A finally block is always executed irrespective of whether the exception occurred or not.
- It is an optional block. It is not necessary to have a finally block i.e. you may just have the try and the catch blocks.
- Let us see the program seen in 5.2.1 again, using try catch block.

Program 5.3.1 : Write a program to perform division of two numbers accepted from user. Handle the divide by zero exception using the try-catch block.

```
import java.io.*;
class Divide
{
   public static void main(String args[ ]) throws IOException
   {
      int a,b,res;
      BufferedReader br = new BufferedReader (new InputStreamReader (System.in));
      String str;
      System.out.println("Enter two numbers:");
      str=br.readLine();
      a=Integer.parseInt(str);
      str=br.readLine();
      b=Integer.parseInt(str);
      try
      {
        res=a/b;
        System.out.println("The Quotient=" + res);
      }
      catch(ArithmeticException ae)
      {
        System.out.println("Exception has occurred. You have entered the divisor as zero");
      }
   }
}
```

Output 1

```
Enter two numbers:
4
0
Exception has occurred. You have entered the divisor as zero
```

Explanation

- In this case when you compare with the Program 5.2.1, the exception is not some garbage, which will not be understo by an end user. You can make a proper statement of what you want to be displayed as an error.

```
    b=Integer.parseInt(str);
  }
  catch (NumberFormatException ne)
  {
    System.out.println("Invalid number");
  }
  try
  {
    res=a/b;
    System.out.println("The Quotient=" + res);
  }
    catch(ArithmeticException ae)
    {
      System.out.println("Exception has occurred. You have entered the divsior as zero");
    }
  }
}
```

Output

```
Enter two numbers:
5
a
Invalid number
Exception has occurred. You have entered the divisor as zero
```

Explanation

- You will notice in this case the character is entered, but a proper error is displayed saying "Invalid number".
- The values of 'a' and 'b' are to be initialized. As the statements in which the values are put into the variables 'a' and 'b' may not be executed if the exception takes place in a statement before them in the try block.
- Now one more exception is displayed, saying divisor given is zero. But you have entered divisor as a character. But the initial values of 'a' and 'b' were zero. Since the statement that generated the first exception did not allow the new value to be given to the variable "b", "b" remained 0. Hence we get this exception.
- We will see in the next section how should we handle multiple exceptions.

5.3.1 Multiple Try Catch Block

When multiple exceptions are to be caught, we must use multiple try catch block. But we must have all the statements that can generate an exception in one try block and the catch blocks for all exceptions that can be generated. Let us see this in the subsequent programs.

Program 5.3.3 : Write a program to perform division of two numbers accepted from user. Handle the Number format exception and also handle the divide by zero exception using multiple try catch block.

Explanation

- A similar implementation as in previous program. Only that here three exceptions are caught.
- In this case, we have removed the "throws IOException" for the main method. This is because the IOException is also now handled by the try-catch block instead of the throws keyword.

5.3.2 Nested Try Catch Block

- When multiple exceptions are to be caught there is one more method called as nested try catch block.
- The nested try catch block as the name says, has a try catch block inside another try block.
- Let us see a program example to understand this concept.

Program 5.3.5 : Write a program to perform division of two numbers accepted from user. Handle the Number format exception and also handle the divide by zero exception using nested try catch block.

```
import java.io.*;
class Divide
{
    public static void main(String args[ ]) throws IOException
    {
        int a=0,b=0,res;
        BufferedReader br = new BufferedReader (new InputStreamReader (System.in));
        String str;
        System.out.println("Enter two numbers:");
        try
        {
          str=br.readLine();
          a=Integer.parseInt(str);
          str=br.readLine();
          b=Integer.parseInt(str);
          try
          {
             res=a/b;
             System.out.println("The Quotient=" + res);
          }
          catch(ArithmeticException ae)
          {
           System.out.println("Exception has occurred. You have entered the divisor as zero");
          }
        }
        catch (NumberFormatException ne)
        {
```

```
    str=br.readLine();
    b=Integer.parseInt(str);
    try
    {
     res=a/b;
     System.out.println("The Quotient=" + res);
    }
    catch(ArithmeticException ae)
    {
      System.out.println("Exception has occurred. You have entered the divisor as zero");
     }
   finally
    {
       System.out.println("In Finally Block");
    }
  }
}
```

Output 1

```
Enter two numbers:
4
0
Exception has occurred. You have entered the divisor as zero
In Finally Block
```

Output 2

```
Enter two numbers:
4
2
The Quotient=2
In Finally Block
```

Explanation

- Here you will notice for both the cases the finally block is executed.
- This can be confirmed because in both the cases i.e. exception generated or not, catch block executed or not but the "In Finally Block" is executed.

5.4 Keyword "throws"

- Although we have been using this keyword in almost all the programs of our syllabus.
- We will still see a simple program as a demonstration purpose for "throws" keyword.

Program 5.4.1 : Write a program to perform division of two numbers accepted from user. Handle the IOException using the keyword "throws".

```
import java.io.*;
class Divide
{
   public static void main(String args[ ]) throws IOException
   {
      int a,b,res;
      BufferedReader br = new BufferedReader (new InputStreamReader (System.in));
      String str;
      System.out.println("Enter two numbers:");
      str=br.readLine();
      a=Integer.parseInt(str);
      str=br.readLine();
      b=Integer.parseInt(str);
      res=a/b;
      System.out.println("The Quotient=" + res);
   }
}
```

Output 1

```
Enter two numbers:
3
4
The Quotient=0
```

Explanation

In this case we have simply indicated that the main() method throws an IOException

5.5 Keyword "throw"

- Using the keyword 'throw', you can make your own conditions to throw the exceptions.
- This means till now, the statements of your program generated exceptions. But now you will purposely throw an exception.
- It will not be an in-built exception; it will be your own created exception
- For example, you accept an integer from user as the month number. The user enters 25. This is invalid month number, but not invalid integer. Hence the number format exception will not take place according to the rules. But you can throw your own exception for this.

Let us see this program example.

Program 5.5.1 : Write a program to accept and display the month number. Throw an number format exception if improper month number is entered.

- We have defined a constructor in the public visibility of this class. The constructor accepts the string type object and displays it.
- In the main program we throw an exception when the month number entered is invalid.
- To throw an exception, we have seen in the previous program that we need to make an object and throw the object.
- Hence we have made the object and while making the object we have passed a parameter of string type to the constructor. In the constructor we have simply displayed the received string.
- The object is caught in the catch block. Here there are no statements to be executed.
- If the month number is given in proper range no exception is thrown and the month number is again displayed as shown below in output 2.

Output 2

```
Enter month number:
12
Month number entered is 12
```

5.6 Introduction to Threads

- A thread can be said to be a set of statements or a small code or a task.
- The major advantage of having threads is that multiple threads can run concurrently on a time sharing basis i.e. each thread is given the computer and the processor for some time.
- Threads are required when multiple tasks are to be executed simultaneously.
- We will learn in this chapter how to make threads and what is the effect of threads.
- A thread is always being executed in the Java program. This thread is called as the "main" thread under the group "main" and has a priority of 5. We will see these terminologies in subsequent sections.
- Let us see a very simple program to understand what is a "Thread" by displaying the current thread parameters.

Program 5.6.1 : Write a program to display the current thread.

```
class Main
{
        public static void main(String args[])
        {
                Thread t1=Thread.currentThread( );
                System.out.println(t1);
        }
}
```

Output

```
Thread[main,5,main]
```

Explanation

- The static method currentThread( ) of the class Thread is used here that returns the current thread in execution to the newly created object of Thread class.
- This Thread is then displayed.

- When the "Thread" object is displayed the standard display is as shown in the output i.e. the class name "Thread", followed by some parameters in the square brackets.
- The parameters displayed are the "Thread" name (as discussed the basic thread is always in execution called as "main"), the priority of that "Thread" (priority is as the name says the importance given to that thread, we will see more about this concept later) and the group under which the "Thread" is (It is a group name).
- In this case the corresponding values are "main", 5 and "main" respectively.
- The priorities vary from 1 to 10. 1 is considered as minimum priority while 10 is considered as maximum priority.

5.7 Making Thread

- There are two ways of creating threads. We have an interface named as **Runnable** and a class named as **Thread.** Based on this, there are two ways to create thread:
  - By Implementing the Runnable interface.
  - By Extending the Thread class.
- Let us see these methods in following sub sections

5.7.1 Implementing the Runnable Interface

- As discussed one way to create threads in java is to make your class implementing the Interface Runnable Interface and make an object of this class.
- You have to override the run( ) method into your class.
- The run( ) method contains the code that is to be executed when the thread runs. The main task should be in this method.
- Let us see the stepwise procedure for creating thread using the Runnable interface :

Step 1 : Make a class that implements Runnable and write a run( ) method in that class. Also make an object of this class that implements "Runnable".

Step 2 : An object of the Thread class is to be created by passing an object (to the constructor of the Thread class) of the class which implements Runnable created in step 1.

Step 3 : The start( ) method is to be invoked for the object created of Thread class. The start( ) method actually makes the run( ) method of that class to be executed.

Step 4 : When the run( ) method ends, the thread ends.

Let us see a program for the demonstration of this method of creating a thread.

Program 5.7.1 : Write a program demonstrate the making of a thread to print numbers from 1 to 10.

```
class Numbers implements Runnable
{

    public void run( )
    {
        int i;
        for(i=1;i<=10;i++)
        {
            System.out.println(i);
        }
```

```
        }
}
class Main
{
        public static void main(String args[])
        {
                Numbers n=new Numbers( );
                Thread t1=new Thread(n);
                t1.start( );
        }
}
```

Output

```
1
2
3
4
5
6
7
8
9
10
```

Explanation

- The same steps are followed as discussed in this sub section. The class "Numbers" is created implementing the interface "Runnable".
- A method named as run( ) is written in this class. This method will be executed when the thread starts. A "for" loop is written in this method to display the numbers from 1 to 10.
- An object of the "Numbers" class is made in the main( ) method and passed to the constructor of another object made of class Thread.
- Finally the start( ) method is called with the object of the "Thread" class. The output displays as per the code written in the run( ) method.

5.7.2 Extending Thread Class

The second method of creating a thread as discussed earlier is by extending the class to the "Thread" class. To make a thread using this method, follow the steps below :

Step 1 : Make a class that extends the class "Thread" and write a run( ) method in that class. Also make an object of this class that extends the class "Thread".

Step 2 : An object of the Thread class is to be created by passing an object (to the constructor of the Thread class) of the class which extends Thread created in step 1.

Step 3 : The start( ) method is to be invoked for the object created of Thread class. The start( ) method actually makes the run( ) method of that class to be executed.

Step 4 : When the run( ) method ends, the thread ends.

Let us see a program for the demonstration of this method of creating a thread.

Program 5.7.2 : Write a program demonstrate the making of a thread to print numbers from 1 to 10 by extending the "Thread" class.

```
class Numbers extends Thread
{
        public void run( )
        {
                int i;
                for(i=1;i<=10;i++)
                {
                        System.out.println(i);
                }
        }
}
class Main
{
        public static void main(String args[])
        {
                Numbers n=new Numbers( );
                Thread t1=new Thread(n);
                t1.start( );
        }
}
```

Output

```
1
2
3
4
5
6
7
8
9
10
```

Explanation

- The same steps are followed as discussed in this sub section. The class "Numbers" is created extending the class "Thread".
- A method named as run( ) is written in this class. This method will be executed when the thread starts. A "for" loop is written in this method to display the numbers from 1 to 10.
- An object of the "Numbers" class is made in the main( ) method and passed to the constructor of another object made of class Thread.
- Finally the start( ) method is called with the object of the "Thread" class. The output displays as per the code written in the run( ) method.

Note : Use of method 1 (discussed in Section 5.7.1) is preferable because we can extend the class to another class besides implementing from Runnable, while this is not possible in method 2 (discussed in Section 5.7.2). In method 2 once you extend your class to "Thread", you cannot extend it to any other class, as Java doesn't support multiple inheritance.

5.8 Life Cycle of a Thread

A thread may have to go through the following states during its life time:

1. Newborn state
2. Runnable state
3. Running state
4. Blocked state
5. Dead state

- The Fig. 5.8.1 shows the transition from and to different states of the life cycle.

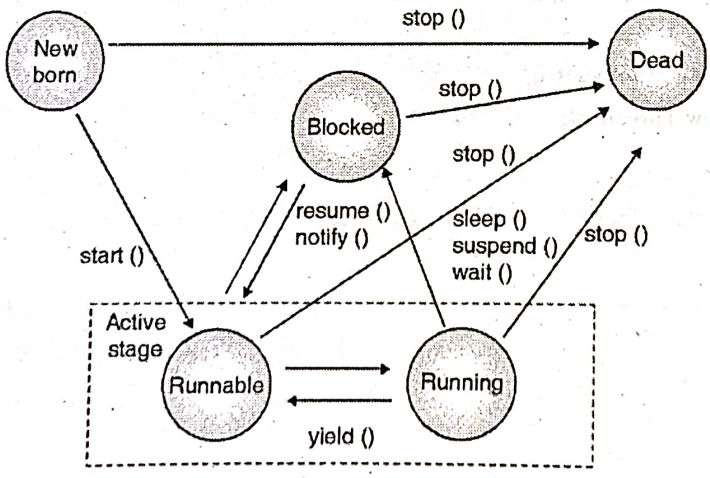


Fig. 5.8.1

- You will notice besides the different states, there are also many methods involved in the life cycle. We will discuss all these methods along with the states below.

5.8.1 New Born State

- A thread is in this state when it is created or made.
- When we call the start( ) method for a thread it enters into the runnable state. The thread will remain in this state until its turn comes to be in the running state.
- If a stop( ) method is executed for a thread then it goes into the dead state.

5.8.2 Active State

- This is the most important state of a thread.
- When the thread is started, it comes in this state. The thread is in active state means, that it can be in either runnable state or running state.
- Only one thread can be in running state at any given time. All other threads can be in runnable, if they are ready for execution. Based on the priorities time slices are allocated to each thread in the runnable state. Each thread gets a time slice, the time slice depends on its priority (importance).
- If a stop( ) method is executed for a thread then it goes into the dead state.
- If sleep( ), suspend( ) or block( ) method is called, the thread enters into the blocked state.

5.8.3 Blocked State

- A thread is in this state when it is blocked because of some resource or if a delay (sleeping) is required. The thread enters into this state by the sleep( ), suspend( ) or block( ) methods.
- The thread enters into this state from the active state. The thread exits from this state when the resume( ) or notify( ) method is called.
- In case if the thread is blocked for some event to occur using the block( ) method, then it is to be notified about the occurrence by the notify( ) method.
- In case if the thread is suspended using the suspend( ) method( ), then it is to be resumed using the resume( ) method( ).
- In case if the thread is made to go to sleep by the sleep( ) method, then the thread automatically wakes up after the specified milliseconds in the brackets while calling the sleep( ) method. This indicates that when calling the sleep( ) method, the delay for which the thread is expected to sleep, must be passed in the brackets. For e.g. Thread.sleep(5000), will make the current thread sleep for 5000 milliseconds.

5.8.4 Dead State

The thread goes into this state when it is stopped by the stop( ) method.

5.9 Creating Multiple Threads

- We can create multiple threads in a program wherein each task is used to perform different operations.
- Each task will perform operation as given in the start method of the corresponding class.
- Let us see some program examples for the same.

Program 5.9 1 : Write a program that has two threads. One of the threads displays the odd numbers from 1 to 10 while the other displays even numbers from 1 to 10.

```
class OddNumbers extends Thread
{
        public void run( )
        {
                int i;
                for(i=1;i<=10;i+=2)
                {
                        System.out.println(i);
```

Note : Each time you execute this (infact most of the programs related to Thread) program on your computer and when you execute on a different computer, the output will be different. This is because the computer assigns different time slice in each computer and each time you execute it. You will also notice that the output in this case is not in sequence from 1 to 10.

5.10 Thread Methods

- The "Thread" class has many methods. We will discuss the some of these methods available in the class "Thread" in this section.
- We have used one of the static methods in the first program of this chapter in Program 5.6.1 i.e. **currentThread()**. This method returns the current thread in executions i.e. running state.
- Another static method is the **sleep**(int) method, to which we pass a integer value, that makes the thread to sleep (enter into blocked state) for the specified milliseconds
- Another static method **yield()**, makes the current thread in running state to move into the runnable state.
- Some other methods like **setName**(String), **getName()** are used to name a thread and get the name of a thread respectively.
- Similarly the methods like **setPriority**(int) and **getPriority()** are used to set the priority of a thread and get the priority of a thread respectively.
- The **join()** and **isAlive()** are another very important methods.
- The **isAlive()** method says whether the thread is alive or in the dead state.
- The **join()** method actually waits for the thread to complete, for the object on which it is called. The **join()** method can be used to join one thread to another i.e. when a thread is joined the joined thread will begin only after the completion of the thread. For e.g. if we write t1.**join()** and then**t2.start()**, then thread t2 will begin after the end of t1, where t1 and t2 are thread objects.
- Let us see programs to demonstrate the use of the methods **join( )** and **isAlive( )**.

Program 5.10.1 : Write a program to demonstrate isAlive( ) and join( ) method.

```
class Alphabets extends Thread
{
    public void run( )
    {
        int i;
        for(i=1;i<=5;i++)
        {
            System.out.println((char)(i+64));
            try
            {
                Thread.sleep(100);
            }
            catch(Exception e)
            {
```

5.11 Thread Synchronization

- A resource can be simultaneously used by multiple threads. In some cases, it is required that only one thread must use the common resource at any given time. In such cases we require to use Synchronization.
- Synchronization refers to controlling the access of a shared resource for multiple threads such that only one thread can access this resource at any given time.
- The "Producer-Consumer model" is a widely known system that requires Synchronization. In this model, "Producer" is a task that produces data while "Consumer" is task that consumes the data created by the "Producer". The "Producer" produces the data and stores it in a memory or variable from where the "Consumer" takes it. Now this variable must be locked such that either the "Producer" is writing on it or the "Consumer" is reading from it. This can be done by synchronizing the statements that are accessing the variable.
- Fig. 5.11.1 shows how a Synchronized resource works.

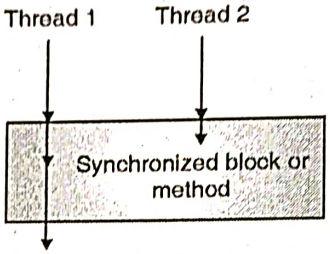
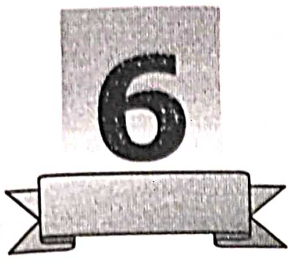


Fig. 5.11.1: Working of Synchronized block or method

- When a block or method is synchronized, only one thread can enter that block. You will notice in the Fig.5.11.1, Thread 1 enters the synchronized block. Since no other thread was in this block, it is allowed to enter. When Thread 2 comes, it has to wait since the thread 1 is already inside the synchronized block.
- Once the thread 1 leaves the block, the thread 2 is allowed to enter into the synchronized block.
- It is a kind of lock to the synchronized block. Only one thread can enter the block and the block is again locked until it leaves the block.
- Synchronizing can be attained in Java in two ways i.e. synchronized methods or synchronized blocks.
- For method synchronization the method should be preceded by the keyword "synchronized" and then only one thread of the object will be able to enter into this method.
- Similarly to make a synchronized block, the following syntax is to be used

```
synchronized (this)
{
        :
        statements;
        :
}
```

- The statements inside the above block will be synchronized in the same manner as a thread is synchronized. The keyword "this" in the brackets corresponds to the current object, hence only one thread of the current object can enter into this synchronized block.



Graphics Programming and File Handling

6.1 Introduction to AWT, Graphics and Swings Packages

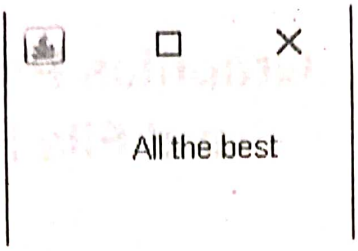
- For graphics programming in Java, we need the "Graphics" class in the package "awt". This class has many methods to draw various shapes and hence these function can be called by making an object of the "Graphics" class.
- The Graphics implemented needs to be displayed in a separate window. To create this frame (similar to window) we need the "JFrame" class in the "swing" package. In this frame we need to place a panel in which the graphics drawn will be displayed. The panel can be created using the "JPanel" class, also in the package "swing".
- Thus, we need to create the class that extends to "JPanel"
- Then, we need to create a class that extends the "JFrame" class and add a panel instance to the same in the constructor of this class as shown in the Program below.

Program 6.1.1 : Write a graphics Java program to display "All the Best" using a frame.

```
import javax.swing.JFrame;
import java.awt.Graphics;
import javax.swing.JPanel;
class MyPanel extends JPanel
{
        public void paint(Graphics g)
        {
                g.drawString("All the best", 50, 30);
        }
}
public class Graphi extends JFrame
{
        Graphi()
        {
                setSize(150,100);
                MyPanel mp=new MyPanel();
                add(mp);
        }

        public static void main(String args[])
        {
                new Graphi().setVisible(true);
        }
}
```

Output



Explanation

- The AWT (Abstract Window Toolkit) is a package required to support the graphical user interface of windows.
- The second package imported is the java.swing. The class made by you must extend the class JPanel. Hence we have extended our class "MyPanel" to the class "JPanel".
- Every class for JPanel should have a paint() method, as in application based programs we use to have the main() method. This paint() method should be capable of accepting an object of the class "Graphics" as in case of main() method we use to have an array of objects of the "String" class.
- This class has many methods. We will be studying these methods in this chapter. One of the method i.e. drawString() is used here to display a text.
- The drawString() method has three arguments. The first argument is a string to be displayed. The second and third arguments are the x and y co-ordinates respectively of the start point of the string to be displayed.

6.2 Graphics Class and its Methods

This class has many methods to draw various shapes on the applets. The drawString() method seen in the above sections is also a method is this class. There are many other methods which are discussed with program in the following sub-sections.

6.2.1 Drawing Lines

- The following method is used to draw a line in an applet:

```
void drawLine (int startX, int startY, int endX, int endY)
```

- This method has parameters startX and startY which indicate the x and y co-ordinates of the starting point of the line. The parameters endX and endY are the x and y co-ordinates of ending point of the line.
- Let us see a Program 6.2.1 to draw some lines.

Program 6.2.1 : Write a Java graphics program to draw horizontal and vertical parallel lines.

```
import javax.swing.JFrame;
import java.awt.Graphics;
import javax.swing.JPanel;
class MyPanel extends JPanel
{
        public void paint(Graphics g)
        {
                g.drawLine(10,10,50,10);
                g.drawLine(10,20,50,20);
                g.drawLine(15,15,15,55);
```

- In this program and in all the programs in the subsequent sections where the methods of the class Graphics are used, the co-ordinate system shown in Fig 6.2.1 is followed.
- Hence you will notice that the lines drawn are on the left top corner of the applet, as per the co-ordinates passed to the drawLine() method.

6.2.2 Drawing Rectangles

- The methods available for drawing the rectangles are given below :
  1. void drawRect(int top, int left, int width, int height)
  2. void fillRect(int top, int left, int width, int height)
  3. void drawRoundRect(int top, int left, int width, int height, int xDiam, int yDiam)
  4. void fillRoundRect(int top, int left, int width, int height, int xDiam, int yDiam)
- There are four methods to draw rectangles as listed above.
- The first method has variables top and left which are the co-ordinates of the starting point on the left top of the rectangle. The parameters width and height are the width and height of the rectangle.
- In the second method the parameters are same. But this method fills the rectangle with the current colour.
- In the third method, you will notice there are two extra parameters. This method also draws rectangle but with rounded corners. The parameters xDiam and yDiam are the diameters of the round (or oval) at the corners of the rectangle.
- The fourth method is again same as the third method, but that it fills the rectangle with the current colour.

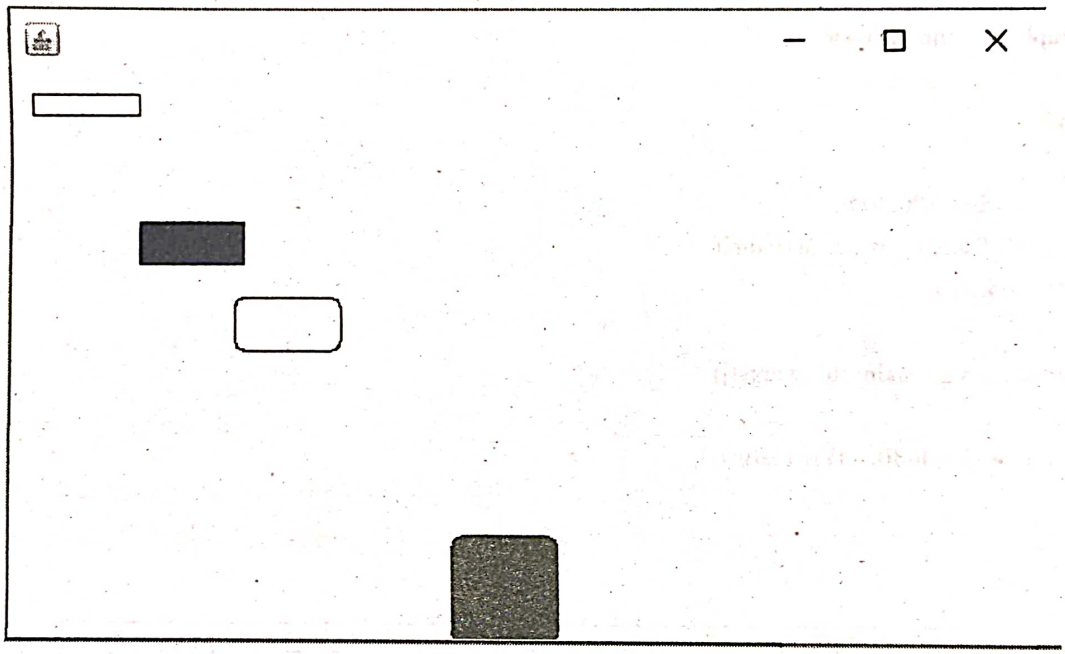
Let us see a Program 6.2.2 to draw some rectangles.

Program 6.2.2 : Write a graphics Java program to draw all four types of rectangles i.e. normal rectangle, filled rectangle, round corners rectangle and filled round corners rectangle.

```
import javax.swing.JFrame;
import java.awt.Graphics;
import javax.swing.JPanel;
class MyPanel extends JPanel
{
        public void paint(Graphics g)
        {
                g.drawRect(10,10,50,10);
                g.fillRect(60,70,50,20);
                g.drawRoundRect(105,105,50,25,10,10);
                g.fillRoundRect(205,215,50,50,10,10);
        }
}
public class Graphi2 extends JFrame
{
        Graphi2()
        {
```

```
        setSize(500,300);
        MyPanel mp=new MyPanel();
        add(mp);
    }
    public static void main(String args[])
    {
        new Graphi2().setVisible(true);
    }
}
```

Output



Explanation

The rectangles are drawn of different sizes filled and not filled, round corner rectangle and filled round corner rectangle.

6.2.3 Drawing Ovals and Circles

- The methods available for drawing the ovals are given below:
  1. void drawOval(int top, int left, int width, int height)
  2. void fillOval(int top, int left, int width, int height)
- There are two methods to draw ovals as listed above. The first method has variables top and left which are the co-ordinates of the starting point on the left top of the rectangle in which the oval will fit. The parameters width and height are the width and height of the same rectangle in which the oval has to fit.
- In the second method the parameters are same. But this method fills the oval with the current colour.
- If the height and width are same, we will get a circle.

Let us see a Program 6.2.3 to draw some ovals.

6.2.4 Drawing Arcs

- The methods available for drawing the arcs are given below :
  1. void drawArc(int top, int left, int width, int height, int startAngle, int sweepAngle)
  2. void fillArc(int top, int left, int width, int height, int startAngle, int sweepAngle)
- There are two methods to draw arcs as listed above.
- The first method has variables top and left which are the co-ordinates of the starting point on the left top of the rectangle in which the oval will fit, whose part is the arc to be drawn. The parameters width and height are the width and height of the same rectangle. The parameters startAngle and sweepAngle are the angles which gives the starting position of the curve in the given oval in anti-clockwise direction. The sweep angle is the angle of how much the arc is to be made. We will understand more about this in the program in this sub-section.
- In the second method the parameters are same. But this method fills the arc with the current colour.

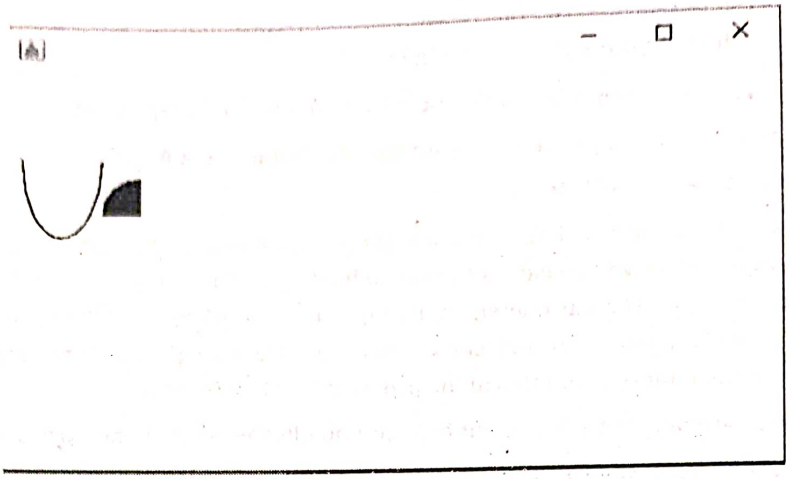
Let us see a Program 6.2.4 to draw some arcs.

Program 6.2.4 : Write a Java graphics program to draw an arc and a filled arc.

```
import javax.swing.JFrame;
import java.awt.Graphics;
import javax.swing.JPanel;
class MyPanel extends JPanel
{
        public void paint(Graphics g)
        {
                g.drawArc(10,10,50,100,180,180);
                g.fillArc(60,70,50,50,90,90);

        }
}
public class Graphi4 extends JFrame
{
        Graphi4()
        {
                setSize(500,300);
                MyPanel mp=new MyPanel();
                add(mp);
        }
        public static void main(String args[])
        {
                new Graphi4().setVisible(true);
        }
}
```

Output



Explanation

Compare the starting angle and sweep angle given in the methods with the output. The sweep angle indicates the total angle of which the curve is to be drawn.

6.2.5 Drawing Polygons

- The methods available for drawing the polygons are given below:
  1. void drawPolygon(int x[ ], int y[ ], int numPoints)
  2. void fillPolygon(int x[ ], int y[ ], int numPoints)
- There are two methods to draw polygons as listed above.
- The first method has parameters as an array of integer which are the co-ordinates of the points to be joined to make the polygon. There are two integer arrays, one for the x co-ordinates of all the points while the other array is the y co-ordinates of all the points. The third parameter is an integer which is the number of points or the number of elements to be taken from the array.
- In the second method the parameters are same. But this method fills the polygon with the current colour.
- Let us see a Program 6.2.5 to draw some polygons.

Program 6.2.5 : Write a Java graphics program to draw a pentagon.

```
import javax.swing.JFrame;
import java.awt.Graphics;
import javax.swing.JPanel;
class MyPanel extends JPanel
{
        public void paint(Graphics g)
        {
                int i;
                int x[]={25,5,5,45,45,25};
                int y[]={25,45,65,65,45,25};
                g.drawPolygon(x,y,6);
```

6.2.6. Changing Colors

- The shapes drawn in all the above graphics in the above sub-sections is by default black in color. But sometimes we need to change the colors.
- In this sub section, we will see how can we change colors for the shapes using some methods.
- There are quite a few methods of changing colors. There are many ways of defining colors for example by the values of R(ed), G(reen) and B(lue). Another method is by defining the values for Hue, Saturation and Brightness. We will use the first one i.e. defining the values for RGB (red, green and blue).
- We have a class called as **Color**. We need to make an object of this class. There are three constructors in this class as listed below.
  1. Color (int red, int green, int blue)
  2. Color (int rgb)
  3. Color (float red, float green, float blue)
- Using the color combination of red, green and blue; we can make the required color. For example, if we need red color, we need to make first variable maximum value and the remaining two are to be made zero in the first constructor. For yellow, we can mix red, green and blue in the right quantity, and so on to get different colors. The maximum value that can be given for the first constructor to each parameter is 255.
- We will be using this way of changing the color i.e. make an object of the class Color with the right values of R, G and B. Hence while making the object of the class Color itself, we need to pass the values of R, G and B to the constructor.
- Then using the setColor() method we can set the color object to be the current color for all the graphics operation carried out in the applets. We can also set the background color and foreground color using the setForeground( ) and setBackground( ) methods. The getColor() method gives the object of the class Color, with the current color combination. The setColor() method is used to set a color.
- We will see more use of these methods in the Program 6.2.6 that demonstrates the color changing in Java applets.

Program 6.2.6 : Write a Java graphics program to draw different shapes of different colors.

```
import javax.swing.JFrame;
import java.awt.*;
import javax.swing.JPanel;
class MyPanel extends JPanel
{
        public void paint(Graphics g)
        {
            Color red=new Color(255,0,0);
            Color green=new Color(0,255,0);
            Color blue=new Color(0,0,255);
            Color yellow=new Color(255,255,0);
            g.setColor(red);
            g.drawLine(10,10,40,40);
            g.setColor(green);
            g.drawRect(10,60,20,20);
```

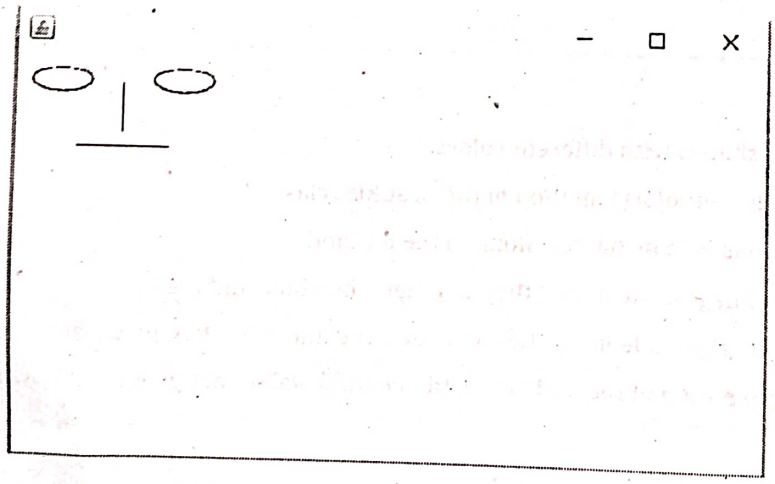
6.3 Miscellaneous Graphics Programs

Program 6.3.1 : Write a Java graphics program to display the following.

Solution :

```
import javax.swing.JFrame;
import java.awt.*;
import javax.swing.JPanel;
class MyPanel extends JPanel
{
        public void paint(Graphics g)
        {
                g.drawOval(10,10,40,15);
                g.drawOval(90,10,40,15);
                g.drawLine(70,20,70,50);
                g.drawLine(40,60,100,60);
        }
}
public class Graphi7 extends JFrame
{
        Graphi7()
        {
                setSize(500,300);
                MyPanel mp=new MyPanel();
                add(mp);
        }
        public static void main(String args[])
        {
                new Graphi7().setVisible(true);
        }
}
```

Output



Explanation

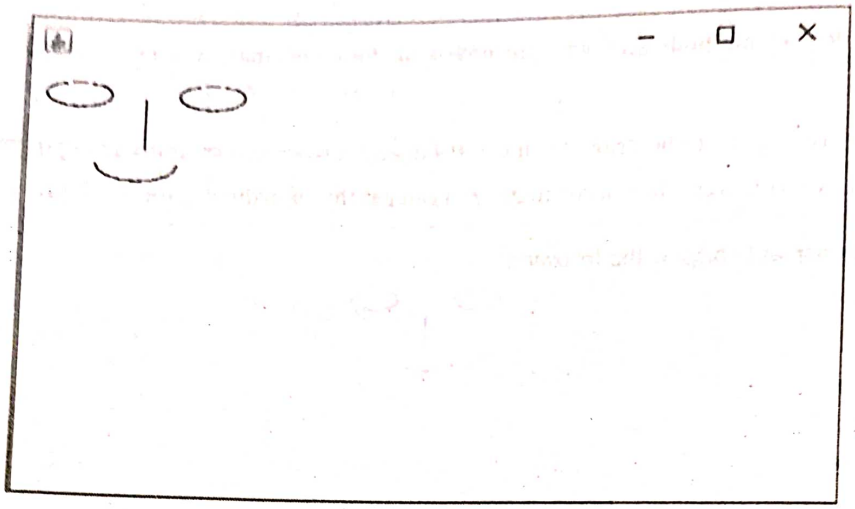
- Just by using the different methods studied upto now and the co-ordinate system of Graphics, we have made this program.
- Whenever such a figure is given to be drawn, you can try drawing rows and columns giving them co-ordinates or draw on graph paper with co-ordinates. Then accordingly you can get the co-ordinates for the methods to be called.

Program 6.3.2 : Write an applet to display the following.

Solution :

```
import javax.swing.JFrame;
import java.awt.*;
import javax.swing.JPanel;
class MyPanel extends JPanel
{
        public void paint(Graphics g)
        {
                g.drawOval(10,10,40,15);
                g.drawOval(90,10,40,15);
                g.drawLine(70,20,70,50);
                g.drawArc(40,50,50,20,180,180);
        }
}
public class Graphi8 extends JFrame
{
      Graphi8()
      {
              setSize(500,300);
              MyPanel mp=new MyPanel();
              add(mp);
       }
      public static void main(String args[])
      {
            new Graphi8().setVisible(true);
      }
```

Output



Explanation :

Just by using the different methods studied upto now and the co-ordinate system of Graphics, we have made this program.

6.4 File Handling in Java

- File handling mainly refers to accessing a file i.e. to read or write a file through the Java program. It is a very important part of any programming language as every time we execute a program its output is displayed and gone. If we want to store the output or use it as input for another program or give input to any program through a file, it is necessary to learn file handling.
- For file handling in Java, we have a class called as "File" in the java.io package. We need to create an object of this class and then perform various operations using the in-built methods of this class.
- Table 6.4.1 below gives a list of few of the useful methods in the class "File". There are many methods in the "File" class, but we will see some of them as required for us. We will see the use of these methods in the subsequent programs in this section.

Table 6.4.1 : Methods in "File" class of Java

Sr. No.	Method name	Return type	Description
1.	createNewFile()	Boolean	It creates the file with the given name and path as a parameter to the method. It returns true if a file was created else returns a false
2.	exists()	Boolean	This method checks whether a particular file exists or not. If it exists, it returns a true, else returns false
3.	getName()	String	It returns the name of the file along with the path
4.	getAbsolutePath()	String	It returns the exact path of the file
5.	canWrite()	Boolean	It returns true or false based on whether the file is writable or not
6.	canRead()	Boolean	It returns true or false based on whether the file is readable or not

Sr. No.	Method name	Return type	Description
7.	length()	int	It returns the size of the file in bytes
8.	write()	void	It writes the data (parameters) passed to it into the file. This is a method in the class FileWriter.
9.	hasNextLine()	Boolean	It returns true or false, based on whether the given file has next line or not
10.	delete()	Boolean	This method deletes the specified file and returns a true if it could delete the file else returns a false
11.	close()	void	This method is in the Scanner class and is used to close the connection established by a Scanner class object with a given File class object. Hence closes the file opened with the given object.

Let us see some program examples of file handling

Program 6.4.1 : Write a Java program to create a file "Data.txt".

```
import java.io.*;
class CreateFile
{
    public static void main(String args[])
    {
      File f0 = new File("C:Data.txt");
      try
      {
          if (f0.createNewFile())
                                    {
          System.out.println("Created sucessfuly file: " + f0.getName());
       }
      else
      {
          System.out.println("File already exists in the given directory.");
      }
    }
    catch (Exception e)
    {
        System.out.println("An unexpected error has occurred.");
    }
}
```

```
            {
                System.out.println(f0.getName()+ " file is successfully deleted .");
            }
            else
            {
                System.out.println("An unexpected error is occurred.");
            }
        }
}
```

Output

```
C:\Java>javac DeleteFile.java
C:\Java>java DeleteFile
Data.txt file is successfully deleted .
```

Explanation

- We have used the various methods discussed in the table in this section to display the information of the file.

6.5 Concept of Streams, Stream Classes And Random File Access

- Streams refer to an array or set of data like a stream of characters, integers etc.
- We have seen BufferedReader class which reads data from the standard input device as a stream. Similarly using streams, we can read and write from the file randomly.
- Let us first see some stream classes and then we will access files randomly (from any given position in the file) using the streams.
- Java has two types of streams namely, byte stream and character stream.
- The byte streams are used to read or write binary data while the character streams are used to read the characters.
- InputStream and OutputStream named abstract classes are the base classes for the byte stream classes in Java. These classes hove many methods with the most useful ones being read() and write() methods.
- Reader and Writer named abstract classes are the base classes for the character stream classes in Java. These also have many methods with the most useful ones being read() and write() methods.
- Some of the byte and character stream classes are listed in Table below.

Table 6.5.1 : Stream classes

Stream Class Name	Stream Class Type	Description
BufferedInputStream	ByteStream	Used for buffered input
BufferedOutputStream	ByteStream	Used for buffered output
DataInputStream	ByteStream	Has methods to read standard data types
DataOutputStream	ByteStream	Has methods to write standard data types
FileInputStream	ByteStream	Has methods to read from file