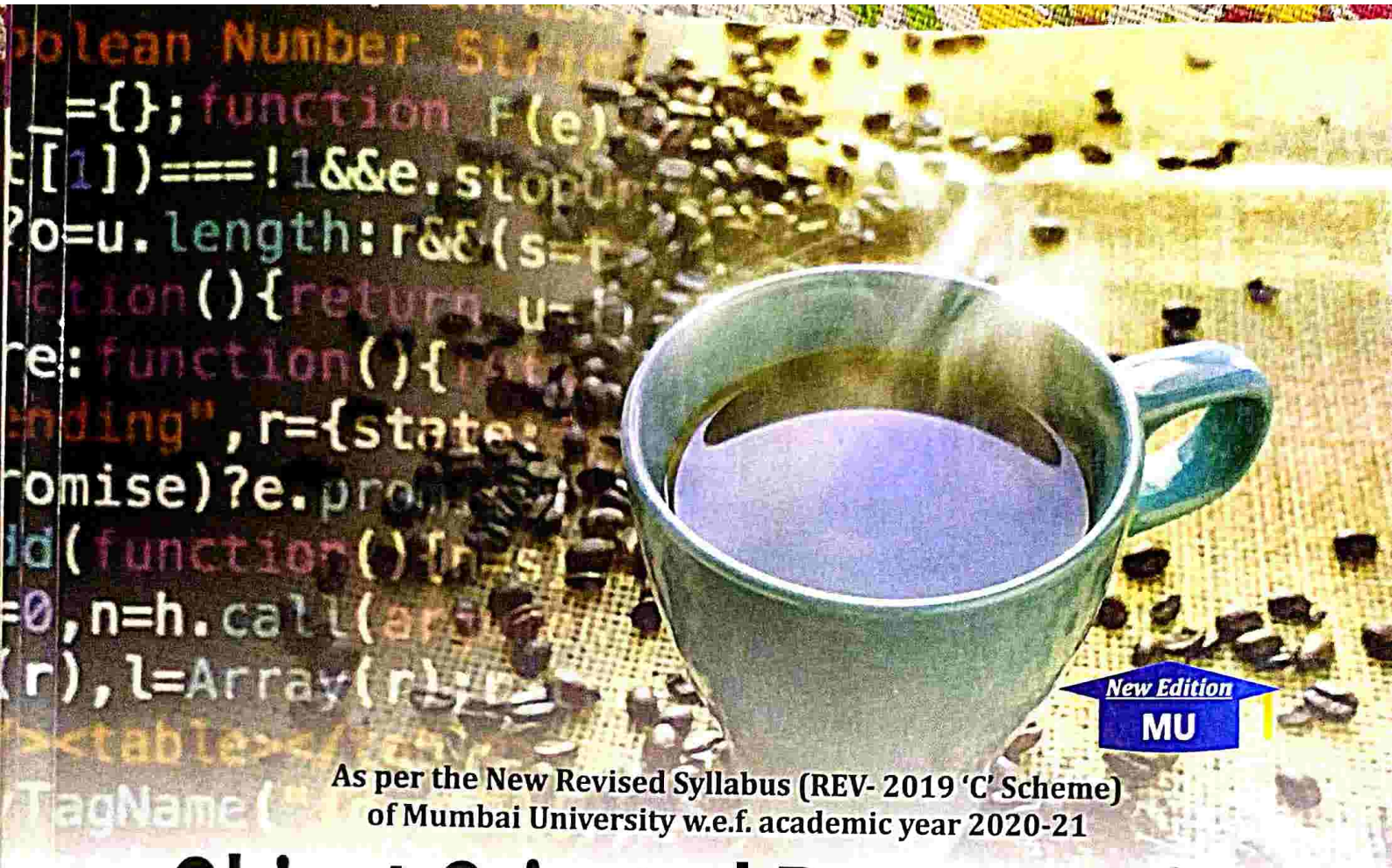


As per the New Revised Syllabus (REV- 2019 'C' Scheme)
of Mumbai University w.e.f. academic year 2020-21

Object Oriented Programming with Java

(Code : CSL304)

Semester III

Computer Engineering / Computer Science and Engineering /
Artificial Intelligence & Data Science / Machine Learning / Cyber Security /
Internet of Things (IoT) / Data Engineering / Data Science /
Internet of Things and Cyber Security Including Block Chain Technology /
Computer Science Design

Harish G. Narula Khushboo Shah

TechKnowledge™ Publications

Object Oriented Programming with Java (Code : CSL304)

Semester III - Computer Engineering / Information Technology
Artificial Intelligence and Data Science / Machine Learning
Cyber Security / Internet of Things (IOT) / Data Engineering / Data Science/
Internet of Things and Cyber Security including Block Chain Technology
(Mumbai University)

Harish G. Narula
Formerly, Assistant Professor (Senior),
D. J. Sanghvi College of Engineering,
Mumbai. Maharashtra, India.

Khushboo Shah
Senior Solution Architect,
Citus Tech, Mumbai.
Maharashtra, India.

TechKnowledge™ Publications

MO227B Price ₹ 395/-

Object Oriented Programming with Java (Code : CSL304)

Harish G. Narula , Khushboo Shah

Semester III – Computer Engineering / Information Technology
Artificial Intelligence and Data Science / Machine Learning
Cyber Security / Internet of Things (IOT) / Data Engineering / Data Science /
Internet of Things and Cyber Security including Block Chain Technology
(Mumbai University)

First Printed in India	:	January 2012
First Edition	:	September 2021 **(As per Rev - 2019 'C' Scheme)**
Second Revised Edition	:	July 2022

This edition is for sale in India, Bangladesh, Bhutan, Maldives, Nepal, Pakistan, Sri Lanka and designated countries in South-East Asia. Sale and purchase of this book outside of these countries is unauthorized by the publisher.

ISBN : 978-93-90694-73-0

Published By

TECHKNOWLEDGE PUBLICATIONS

Printed @

37/2, Ashtavinayak Industrial Estate,
Near Pari Company,
Narhe, Pune, Maharashtra State, India.
Pune - 411041

Head Office

B/5, First floor, Maniratna Complex, Taware Colony,
Aranyeshwar Corner, Pune - 411 009.
Maharashtra State, India
Ph : 91-20-24221234, 91-20-24225678.
Email : info@techknowledgebooks.com,
Website : www.techknowledgebooks.com

Subject Code : CSL304
Book Code : MO227B

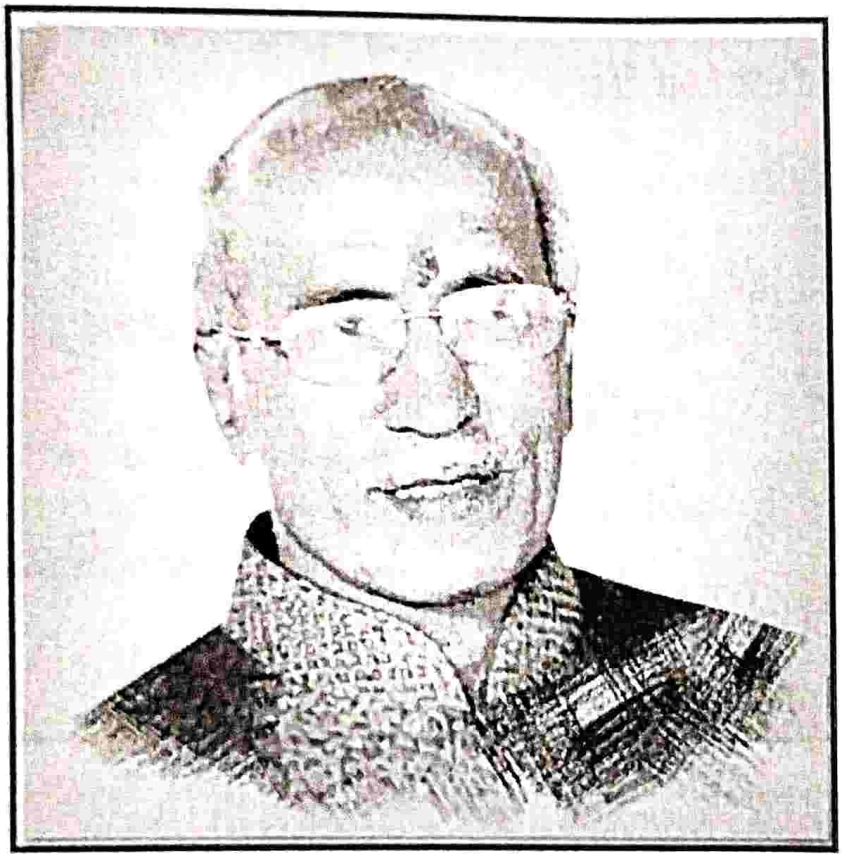
We dedicate this Publication soulfully and wholeheartedly,

in loving memory of our beloved founder director,

Late Shri. Pradeepji Lalchandji Lunawat,

who will always be an inspiration, a positive force and strong support

behind us.



"My work is my prayer to God"

- Late Shri. Pradeepji L. Lunawat

Soulful Tribute and Gratitude for all Your
Sacrifices, Hardwork and 40 years of Strong Vision...

Dedicated To .....

The Readers of this Book

Authors

Preface

Dear Students,

We are extremely happy to present the book of ***"Object Oriented Programming with Java"*** *for you. We have divided the subject into small chapters so that the topics can be arranged and understood properly. The topics within the chapters have been arranged in a proper sequence to ensure smooth flow of the subject.*

We present this book in the loving memory of ***Late. Shri. Pradeepji Lunawat****, our source of inspiration and a strong foundation of* ***"TechKnowledge Publications".*** *He will always be remembered in our hearts and motivate us to achieve our new milestone.*

We are thankful to Prof. J. S. Katre, Shri. Shital Bhandari, Prof. Arunoday Kumar and Shri. Chandroday Kumar for the encouragement and support that they have extended. We are also thankful to Seema Lunawat for technology enhanced reading, E-books support and the staff members of TechKnowledge Publications for their efforts to make this book as good as it is. We have jointly made every possible efforts to eliminate all the errors in this book. However if you find any, please let us know, because that will help us to improve further.

We are thankful to my family members and friends for their patience and encouragement. We would also like to thank Guruji Shree Swami Satyanandji Maharaj for his blessings

- Authors

❑❑❑

Syllabus

Mumbai University

Second Year of Computer Engineering (2019 Course)

CSL304 : Object Oriented Programming with Java

Lab Code	Lab Name	Credits
CSL304	Skill based Lab Course: Object Oriented Programming with Java	2

Prerequisite: Structured Programming Approach

Lab Objectives:

1	To learn the basic concepts of object-oriented programming
2	To study JAVA programming language
3	To study various concepts of JAVA programming like multithreading, exception Handling, packages, etc.
4	To explain components of GUI based programming.

Lab Outcomes : At the end of the course, the students should be able to

1	To apply fundamental programming constructs.
2	To illustrate the concept of packages, classes and objects.
3	To elaborate the concept of strings, arrays and vectors.
4	To implement the concept of inheritance and interfaces.
5	To implement the concept of exception handling and multithreading.
6	To develop GUI based application.

Module		Detailed Content	Hours
1	**Introduction to Object Oriented Programming**		2
	1.1	OOP concepts: Objects, class, Encapsulation, Abstraction, Inheritance, Polymorphism, message passing.	
	1.2	Java Virtual Machine	
	1.3	Basic programming constructs: variables, data types, operators, unsigned right shift operator, expressions, branching and looping.	

2	**Class, Object, Packages and Input/output**		6
	2.1	Class, object, data members, member functions Constructors, types, static members and functions Method overloading Packages in java, types, user defined packages Input and output functions in Java, Buffered reader class, scanner class	
3.	**Array, String and Vector**		3
	3.1	Array, Strings, String Buffer, Vectors	
4		**Inheritance**	4
	4.1	Types of inheritance, Method overriding, super, abstract class and abstract method, final, Multiple inheritance using interface, extends keyword	
5.		**Exception handling and Multithreading**	5
	5.1	Exception handling using try, catch, finally, throw and throws, Multiple try and catch blocks, user defined exception Thread lifecycle, thread class methods, creating threads using extends and implements keyword.	
6		**GUI programming in JAVA**	6
	6.1	Applet and applet life cycle, creating applets, graphics class functions, parameter passing to applet, Font and color class. Event handling using event class AWT: working with windows, using AWT controls for GUI design Swing class in JAVA. Introduction to JDBC, JDBC-ODBC connectivity, JDBC architecture.	

❑❑❑

Mumbai University

Second Year of Information Technology (2019 Course)

ITL304 : Java Lab (SBL)

		Teaching Scheme (Contact House)			Credits Assinged			
Lab Code	Lab Name	Theory	Practical	Tutorial	Theory	Practical	Tutorial	Total
ITL304	Java Lab (SBL)	-	04	-	-	02	-	02

Lab Code	Lab Name	Examination Scheme					
		Internal Assessment			End Sem. Exam		
		Test 1	Test 2	Avg.	Team work	Practical/oral	Total
ITL304	Java Lab (SBL)	-	-	-	-	25	25	50

Lab Objectives

Sr. No.	Lab Objectives
The Lab experiments aims :	
1	To understand the concepts of object-oriented paradigm in the Java programming language.
2	To understand the importance of Classes & objects along with constructors, Arrays ,Strings and vectors
3	To learn the principles of inheritance, interface and packages and demonstrate the concept of reusability for faster development.
4	To recognize usage of Exception Handling, Multithreading, Input Output streams in various applications
5	To learn designing, implementing, testing, and debugging graphical user interfaces in Java using Swings and AWT components that can react to different user events.
6	To develop graphical user interfaces using JavaFX controls.

Lab Outcomes

Sr. No.	Lab Outcomes	Cognitive levels of attainment as per Bloom's Taxonomy
On successful completion, of course, learner/student will be able to:		
1	Explain the fundamental concepts of Java Programing.	L1, L2
2	Use the concepts of classes, objects, members of a class and the relationships among them needed for a finding the solution to specific problem.	L3
3	Demonstrate how to extend Java classes and achieve reusability using Inheritance, Interface and Packages.	L3
4	Construct robust and faster programmed solutions to problems using concept of Multithreading, exceptions and file handling	L3
5	Design and develop Graphical User Interface using Abstract Window Toolkit and Swings along with response to the events.	L6
6	Develop Graphical User Interface by exploring JavaFX framework based on MVC architecture.	L6

Prerequisite : Basics of Computer Programming

Hardware & Software Requirements :

Hardware Requirements	Software Requirements	Other Requirements
PC With Following Configuration 1. Intel PIV Processor 2. 2 GB RAM 3. 500 GB Harddisk 4. Network interface card	1. Windows or Linux Desktop OS 2. JDK 1.8 or higher 3. Notepad ++ 4.JAVA IDEs like Netbeans or Eclipse	1. Internet Connection for installing additional packages if required

Detailed syllabus

Sr. No.	Module	Detailed Content	Hours
0	Prerequisite	**Basics of Computer Programming.**	2
I	Java Fundamentals	**Overview of procedure and object oriented Programming,** Java Designing Goals and Features of Java Language. **Introduction to the principles of object-oriented programming :** Classes, Objects, Abstraction, Encapsulation, Inheritance, Polymorphism. Keywords, Data types, Variables, Operators, Expressions, Types of variables and methods. **Control Statements :** If Statement, If-else, Nested if, switch Statement, break, continue. **Iteration Statements :** for loop, while loop, and do-while loop (Perform any 2 programs that covers Classes, Methods, Control structures and Looping statements) 1. Implement a java program to calculate gross salary & net salary taking the following data. Input : empno, empname, basic Process : DA = 70 % of basic HRA = 30 % of basic CCA = Rs. 240/- PF = 10 % of basic PT = Rs. 100/- 2. Five Bikers Compete in a race such that they drive at a constant speed which may or may not be the same as the other. To qualify the race, the speed of a racer must be more than the average speed of all 5 racers. Write a Java program to take as input the speed of each racer and print back the speed of qualifying racers. 3. Write a Java program that prints all real solutions to the quadratic equation $ax^2 + bx + c = 0$. Read in a, b, c and use the quadratic formula. If the discriminate $b^2 - 4ac$ is negative, display a message stating that there are no real solutions ?	7

Sr. No.	Module	Detailed Content	Hours
		4. Write a Menu driven program in java to implement simple banking application. Application should read the customer name, account number, initial balance, rate of interest, contact number and address field etc. Application should have following methods. 1. createAccount() 2. deposit() 3. withdraw() 4. computeInterest() 5. displayBalance() 5. Write a menu driven Java program which will read a number and should implement the following methods 1. factorial() 2. testArmstrong() 3. testPalindrome() 4. testPrime() 5. fibonacciSeries() 6. Create a Java based application to perform various ways of Method overloading.	
II	**Classes, objects, Arrays and Strings**	**Classes & Objects :** Reference Variables, Passing parameters to Methods and Returning parameters from the methods, Static members, Non-Static members Nested and Inner Classes. Static Initialization Block(SIB), Instance Initialization Block(IIB) **Constructors :** Parameterized Constructors, chaining of constructor, finalize() Method, Method overloading, Constructors Overloading. Recursion, Command-Line Arguments. Wrapper classes, InputBufferReader, OutputBufferReader, String Buffer classes, String functions. **Arrays & Vectors :** One and Two Dimensional arrays, Irregular arrays, dynamic arrays, Array List and Array of Object. (Perform any 3 programs that covers Classes & objects, Constructors, Command Line Arguments, Arrays/Vectors,String function and recursions). **Experiments :** 1. Write a program that would print the information (name, year of joining, salary, address) of three employees by creating a class named 'Employee'. The output should be as follows : Name Year of joining Address Robert 1994 64C-WallsStreat Sam 2000 68D-WallsStreat John 1999 26B-WallsStreat 2. Write a program to print the area of a rectangle by creating a class named 'Area' having two methods. First method named as 'setDim' takes length and breadth of rectangle as parameters and the second method named as 'getArea' returns the area of the rectangle. Length and breadth of rectangle are entered through keyboard.	07

Sr. No.	Module	Detailed Content	Hours
		3. Write a Java program to illustrate Constructor Chaining. 4. Create a class 'Student' with three data members which are name, age and address. The constructor of the class assigns default values name as "unknown", age as '0' and address as "not available". It has two members with the same name 'setInfo'. First method has two parameters for name and age and assigns the same whereas the second method takes has three parameters which are assigned to name, age and address respectively. Print the name, age and address of 10 students. Hint - Use array of objects. 5. Write a Java programs to add n strings in a vector array. Input new string and check whether it is present in the vector. If it is present delete it otherwise add it to the vector. 6. Print the sum, difference and product of two complex numbers by creating a class named 'Complex' with separate methods for each operation whose real and imaginary parts are entered by user. 7. Write menu driven program to implement recursive Functions for following tasks. (a) To find GCD and LCM (b) To print n Fibonacci numbers (c) To find reverse of number (d) To solve 1 + 2 + 3 + 4 +.....+ (n – 1) + n 8. Print Reverse Array list in java by writing our own function.	
III	**Inheritance, Packages and Interfaces.**	**Inheritance :** Inheritance Basics, Types of Inheritance in Java, member access, using Super- to call superclass Constructor, to access member of super class (variables and methods), creating multilevel hierarchy, Constructors in inheritance, method overriding, Abstract classes and methods, using final, Dynamic Method Dispatch **Packages :** Defining packages, creating packages and Importing and accessing packages **Interfaces :** Defining, implementing and extending interfaces, variables in interfaces, Default Method in Interface ,Static Method in interface, Abstract Classes vs Interfaces. (Perform any 3 programs covering Inheritance, Interfaces and Packages). **Experiments :** 1. Create a Teacher class and derive Professor/ Associate_Professor/Assistant_Professor class from Teacher class. Define appropriate constructor for all the classes. Also define a method to display information of Teacher. Make necessary assumptions as required. 2. Create a class Book and define a display method to display book information. Inherit Reference_Book and Magazine classes from Book class and override display method of Book class in Reference_Book and Magazine classes. Make necessary assumptions required. 3. A university has two types of students — graduate students and research students. The University maintains the record of name, age and programme of every student.	**10**

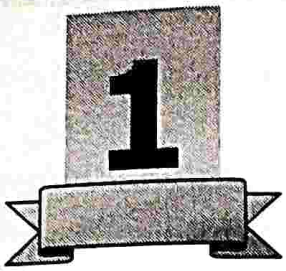
Sr. No.	Module	Detailed Content	Hours
		For graduate students, additional information like percentage of marks and stream, like science, commerce, etc. is recorded; whereas for research students, additionally, specialization and years of working experience, if any, is recorded. Each class has a constructor. The constructor of subclasses makes a call to constructor of the superclass. Assume that every constructor has the same number of parameters as the number of instance variables. In addition, every subclass has a method that may update the instance variable values of that subclass. All the classes have a function display_student_info( ), the subclasses must override this method of the base class. Every student is either a graduate student or a research student. Perform the following tasks for the description given above using Java : (I) Create the three classes with proper instance variables and methods, with suitable inheritance. (II) Create at least one parameterised constructor for each class. (III) Implement the display_student_info( ) method in each class. 4. An employee works in a particular department of an organization. Every employee has an employee number, name and draws a particular salary. Every department has a name and a head of department. The head of department is an employee. Every year a new head of department takes over. Also, every year an employee is given an annual salary enhancement. Identify and design the classes for the above description with suitable instance variables and methods. The classes should be such that they implement information hiding. You must give logic in support of your design. Also create two objects of each class. 5. Consider a hierarchy, where a sportsperson can either be an athlete or a hockey player. Every sportsperson has a unique name. An athlete is characterized by the event in which he/she participates; whereas a hockey player is characterised by the number of goals scored by him/her. Perform the following tasks using Java : (i) Create the class hierarchy with suitable instance variables and methods. (ii) Create a suitable constructor for each class. (iii) Create a method named display_all_info with suitable parameters. This method should display all the information about the object of a class. (iv) Write the main method that demonstrates polymorphism. 6. Create an interface vehicle and classes like bicycle, car, bike etc, having common functionalities and put all the common functionalities in the interface. Classes like Bicycle, Bike, car etc implement all these functionalities in their own class in their own way 7. Create a class "Amount In Words" within a user defined package to convert the amount into words. (Consider amount not to be more than 100000).	

Sr. No.	Module	Detailed Content	Hours
IV	**Exception Handling, Multithreading, Input Output streams**	**Exception Handling :** Exception-Handling Fundamentals, Exception Types, Exception class Hierarchy, Using try and catch, Multiple catch Clauses, Nested try Statements, throw, throws, finally , Java's Built-in Exceptions, Creating Your Own Exception Subclasses **Multithreaded Programming :** The Java Thread Model and Thread Life Cycle, Thread Priorities, Creating a Thread, Implementing Runnable, Extending Thread, Creating Multiple Threads, Synchronization: Using Synchronized Methods, The synchronized Statement **I/O Streams :** Streams, Byte Streams and Character, The Predefined Streams, Reading Console Input, Reading Characters, Reading Strings, Writing Console Output, Reading and Writing Files. (Perform any 3 programs that cover Exception Handling, Multithreading and I/O Streams). **Experiments :** 1. Write java program where user will enter loginid and password as input. The password should be 8 digit containing one digit and one special symbol. If user enter valid password satisfying above criteria then show "Login Successful Message". If user enter invalid Password then create InvalidPasswordException stating Please enter valid password of length 8 containing one digit and one Special Symbol. 2. Java Program to Create Account with 1000 Rs Minimum Balance, Deposit Amount, Withdraw Amount and Also Throws LessBalanceException. It has a Class Called LessBalanceException Which returns the Statement that Says WithDraw Amount(_Rs) is Not Valid. It has a Class Which Creates 2 Accounts, Both Account Deposite Money and One Account Tries to WithDraw more Money Which Generates a LessBalanceException Take Appropriate Action for the Same. 3. Create two threads such that one thread will print even number and another will print odd number in an ordered fashion. 4. Assume that two brothers, Joe and John, share a common bank account. They both can, independently, read the balance, make a deposit, and withdraw some money. Implement java application demonstrate how the transaction in a bank can be carried out concurrently. 5. You have been given the list of the names of the files in a directory. You have to select Java files from them. A file is a Java file if it's name ends with ".java". For e.g. File- "Names.java" is a Java file, "FileNames.java.pdf" is not. **Input :** test.java, ABC.doc, Demo.pdf, add.java, factorial.java, sum.txt **Output :** tset.java, add.java, factorial.java	10
V	**GUI programming- I (AWT, Event Handling, Swing)**	**Designing Graphical User Interfaces in Java :** Components and Containers, Basics of Components, Using Containers, Layout Managers, AWT Components, Adding a Menu to Window, Extending GUI Features. **Event-Driven Programming in Java :** Event-Handling Process, Event-Handling Mechanism, Delegation Modelof Event Handling, Event Classes, Event Sources, Event Listeners, Adapter Classes as Helper Classes in Event Handling.	12

Sr. No.	Module	Detailed Content	Hours
		Introducing Swing : AWT vs Swings, Components and Containers, Swing Packages, A Simple Swing Application, Painting in Swing, Designing Swing GUI Application using Buttons, JLabels, Checkboxes, Radio Buttons, JScrollPane, JList, JComboBox, Trees, TablesScroll pane Menus and Toolbar (Perform any 3 programs that contain AWT, Event handling and Swing to build GUI application). 1. Write a Java program to implement Swing components namely Buttons, ,JLabels, Checkboxes, Radio Buttons, JScrollPane, JList, JComboBox, Trees, Tables Scroll pane Menus and Toolbars to design interactive GUI. 2. Write a program to create a window with four text fields for the name, street, city and pincode with suitable labels. Also windows contains a button MyInfo. When the user types the name, his street, city and pincode and then clicks the button, the types details must appear in Arial Font with Size 32, Italics. 3. Write a Java program to create a simple calculator using java AWT elements. .Use a grid layout to arrange buttons for the digits and basic operation +, -, /, *. Add a text felid to display the results. 4. Write a Java Program to create a Student Profile form using AWT controls. 5. Write a Java Program to simulate traffic signal light using AWT and Swing Components. 6. Write a Java Program to create a color palette. Declare a grid of Buttons to set the color names. Change the background color by clicking on the color button. 7. Build a GUI program that allows the user to add objects to a collection and perform search and sort on that collection.(Hint. Use Swing components like JButton, JList, JFrame, JPanel and JOptionPane.)	
VI	**GUI Programming-II (JavaFX)**	JavaFX Basic Concepts, JavaFX application skeleton, Compiling and running JavaFX program,Simple JavaFX control:Label,Using Buttons and events, Drawing directly on Canvas. (Perform any one program that contains the concept of JavaFX). 1. Write a Java program to design a Login Form using JavaFX Controls. 2. Write Java program to draw various shapes on Canvas using JavaFX.	**04**

Chapter 1 : Java Fundamentals 1-1 to 1-84

❑❑❑



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.

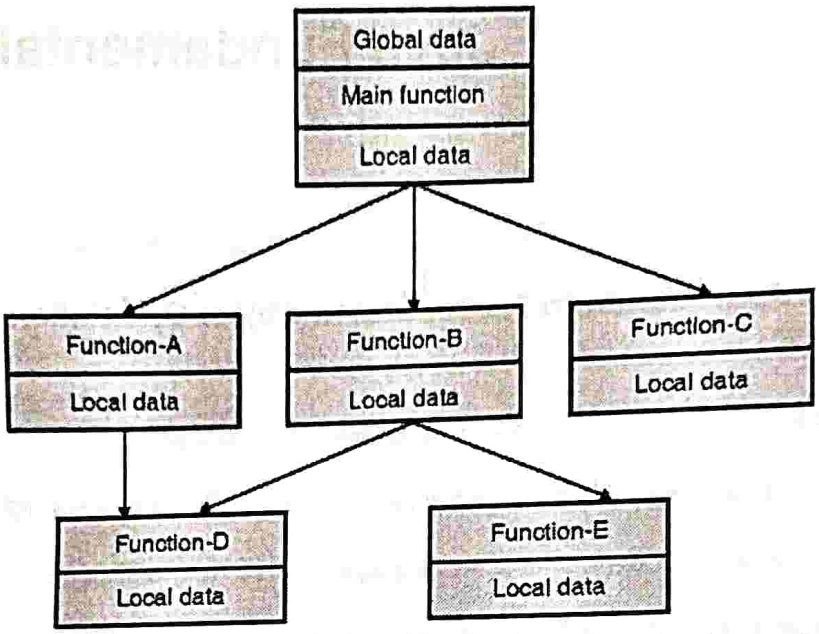


Fig. 1.1.1 : Structure of a procedure oriented programming language

- In this type of languages the task is divided into smaller tasks or sub tasks called as procedures or functions or methods. Any function can be called at any instant.
- There are global and local variables as seen in the Fig. 1.1.1. The global variables can be accessed by all the functions, while the local variables are local to the corresponding function.
- A procedure oriented programming language follows top bottom approach.
- Top bottom approach refers to approaching to a problem as a big task or as a whole. Then dividing this task into small instructions or operations. Hence, initially the task is broken into small tasks and then these smaller tasks are written in detail.
- The disadvantage of such type of programming is that it is difficult to trace which functions are using a data and also error correction is difficult.
- C programming language is an example of procedure oriented language.
- Disadvantages of procedure oriented programming language :
  1. Global data is vulnerable to inadvertent changes. In other words, the global variables may be changed unintentionally.
  2. If a data structure is changed then all the functions accessing this structure have to be altered i.e. if the data variables in the structure are changed, then all the functions that use this structure will have to be changed.
  3. Procedure oriented programs cannot model the real world problems very well.
- 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. 1.1.2.

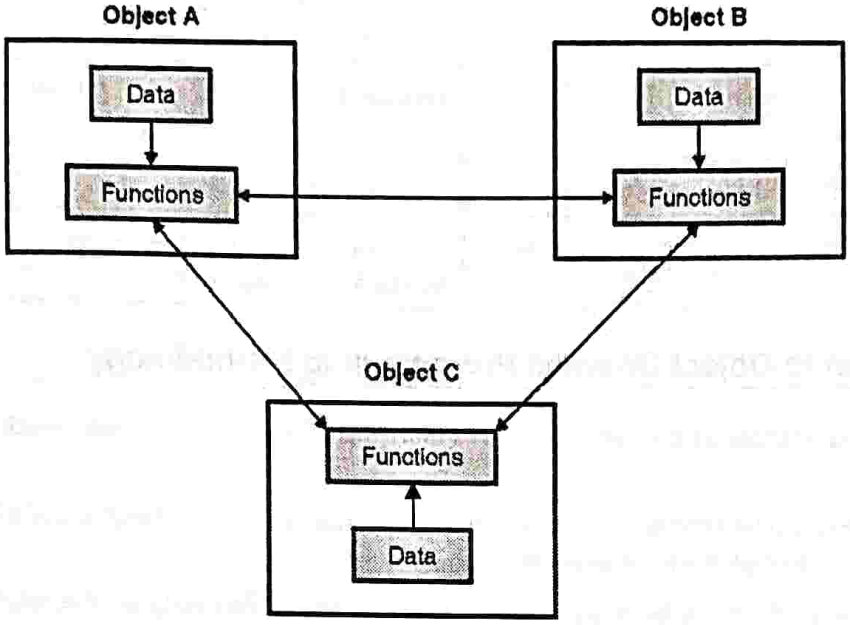


Fig. 1.1.2 : Structure of an object oriented program

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.

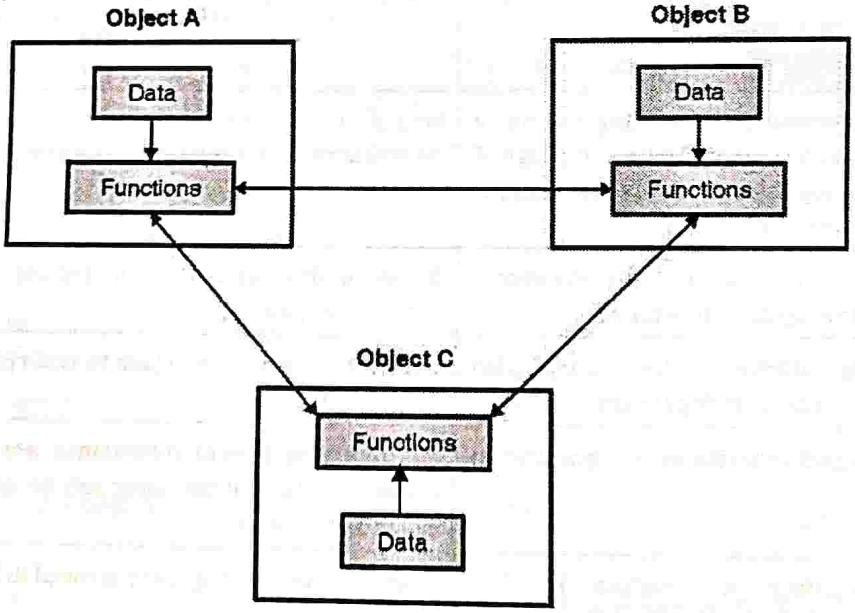


Fig. 1.3.1 : Structure of an object oriented program

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).

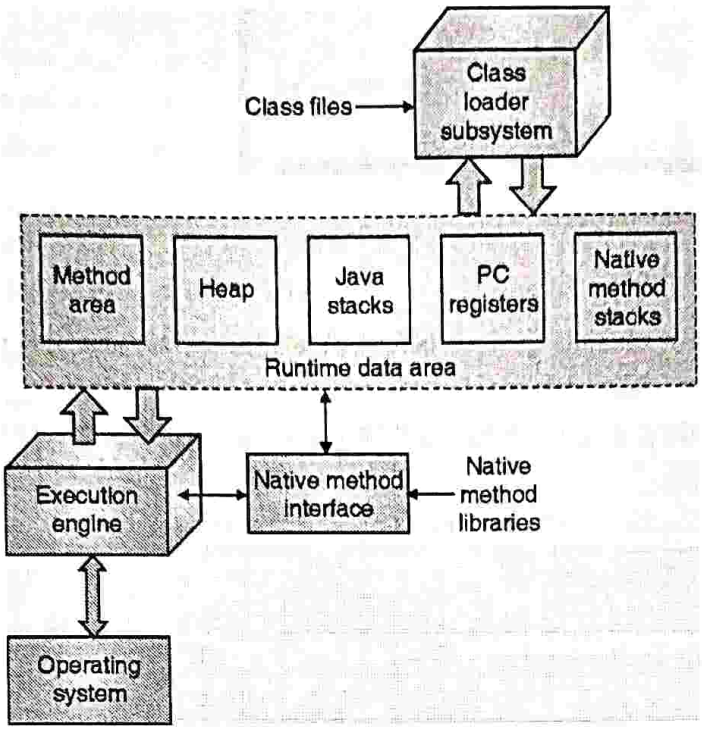


Fig. 1.8.1

- The Java compiler converts the .java file to .class file, which can be given to any system as discussed in the features of Java. The JVM first loads this .class file into the class loader sub system.
- **Run time Memory area**: The program is allocated memory for different data as shown in the Fig 1.8.1. Some area is allocated for method (In Java the word method means functions), heap, stacks, PC registers and stack for native method. The space for all these is allocated when the class file is loaded.
- The method area is used to store the code of the program that is currently being executed.
- Heap is used to store objects i.e. space for the variables of the objects created in the program during the run time.
- Stack is a special kind of memory used to store data when calling methods.
- PC (Program Counter) registers are registers that contain the address of the code of the program that is in execution.
- The library file methods that are executed are called as native methods. There is also a memory space allocated for these native methods to be executed. The native methods as shown, are loaded from the libraries and an interface is used to handle these methods.
- Execution engine of the JVM is a special block with an Interpreter and JIT (Just in Time). The interpreter as also discussed earlier is a unit that translates the byte code into the system dependent machine code. This process of translating the byte code to machine code and then executing this machine code took a long time earlier. This problem was resolved by the implementation of JIT in the execution engine. The JIT translates and immediately executes the machine code i.e. just in time. Hence increasing the speed of operation.

1.9 Tokens of Java

- Before going for programming, let us see a very important thing required for programming i.e. the tokens of a programming language.
- The Java programs are made up of different things termed as tokens.

- 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

6.	\\	Backslash
7.	\'	Single quote
8.	\"	Double quote
9.	\ddd	Octal character
10.	\uxxxx	Hexadecimal Unicode character
11.	\b	backspace

1.9.7 Operators

- Operators are different special symbols used for indicating an operation to be performed.
- The operators are one of the tokens of Java. There are different types of operators classified based on their functions.
- Many of these operators although listed here, but will be understood in the subsequent chapters as and when we use them. So you may just go through the operators and understand them later while programming.
- The data on which the operation is performed are called as operands. If an operator requires one operand, it is called as Unary operator. If an operator requires two operands, then it is called as Binary operator and if it requires three operands, it is called as Ternary operator.

1.9.7(A) Unary Operators

These are operators that require only one operand. Let us discuss these operators one by one.

1. Unary minus (–)

- The symbol shown in the bracket is used as unary minus operator.
- It returns the negative value of the variable to which it is preceded.
- For e.g.

  if x = 3 and y = 6 then

  y = – x;

  will make the value of y as – 3.

2. Casting Operator (( ))

- It is many times required to convert one data type to another.
- The casting operator is used for type conversion i.e. it can be used to convert a data of one type to another data type.
- For e.g. if we have

  int x = 3;

  float y = 5.6;

  then the statement, x= (int) y;

  will result in the value of x as 5.

3. Logical Not operator (!)

- The symbol shown in the brackets i.e. the exclamation mark is used as a logical not operator. It is used to check certain conditions in a condition statement. It performs the logical not of the result obtained from the expression on its right.

- For e.g.

  if x =1, then y = !x;

  will result with y having the value 0.

- We will see some more logical not operator based expressions and program followed by this section.

4. Bitwise not operator (~)

- This operator is used to perform bitwise NOT operation as discussed in the section 2.7.7.2(II). We have also seen some examples of NOT operations.
- The bitwise NOT operator can be used to perform binary NOT operation. This operation can be performed by using the operator given in brackets i.e. ~.
- For e.g.

  if x = 3,

  then y = ~x;

  will result in y having a value of -4

5. Increment Operator (++)

- It returns the value of the variable added with one and stores the result in the variable itself.
- For e.g.

  if x = 5,

  then x++;

  will make the value of x equal to 6.

- This "x++;" (also called as post increment operator) can also be written as "++x;" (also called as pre increment operator).
- It can also be used to store the result in another variable. But in this case the post increment and pre increment statements will have different behavior as explained below with examples.
- In post increment case, for e.g. if x = 5,

  then y = x++;

  will make the value of y equal to 5 and x equal to 6.

- In pre increment case, for e.g. if x = 5,

  then y = ++x;

  will make the value of y equal to 6 and x equal to 6.

- More such examples will be seen with programs.

6. Decrement Operator(--)

- It returns the value of the variable subtracted with one and stores the result in the variable itself.
- For e.g. if x = 5,

  then x--;

  will make the value of x equal to 4.

- This "x--;" (also called as post decrement operator) can also be written as "--x;" (also called as pre decrement operator).

- 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 the 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 th 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. $a = \dfrac{xy + z\left(\dfrac{x}{y}\right) + zy}{x + y + z}$

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

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

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

   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

- We will see the use of these input and output methods in the programs to be followed to this section.

1.12.3 Accepting Input using BufferedReader Class

- The string accepted is to be then converted to the required data type. We will see how this works for BufferedReader.
- There is a function called as readLine() to accept a string from user. This method is non-static and is in the class BufferedReader. Hence we need to make an object of this class to use the readLine() method. 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 BufferedReader is in the package "java.io". When making an object of the class BufferedReader, we need to pass an object of InputStreamReader to the constructor of this class. Hence we need to also create and pass an object to the constructor of this class. But to create an object of the class InputStreamReader, 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 BufferedReader is to be created as shown below :

  BufferedReader br = new BufferedReader (new InputStreamReader (System.in))

- Here the object named as "br" is created of the class BufferedReader. While creating this object an object of InputStreamReader is passed to the constructor, in the brackets meant for passing the parameters to an object. The new operator is used to create an object of a class as already discussed in this section. The class InputStreamReader also has a constructor that requires the variable to indicate the source of the input.
- Since this class "BufferedReader" is in the package "java.io", we need to import all the classes of this package by writing the statement "import java.io.*" in the beginning of the program.
- The readLine() method stated above can accept only a string. This string can be converted to int, float etc with the help of wrapper classes. For time being, we will just see how to convert a string to int, float etc. The wrapper class "Integer" has a static method called as "parseInt()". The string passed to this method is converted into an integer value (but the string should contain only digits). Hence this method can be called to parse (convert) a string to integer in the following way;

  int i = Integer.parseInt(str), where 'i' is an integer and "str" is a string.

- Similarly, to convert a string to float number, the wrapper class "Float" with its static method "parseFloat" is to be used.

  For e.g. float f=Float.parseFloat(str), where 'f' is a float number and "str" is a string.

- Another care to be taken when using readLine() method is that it throws an Exception called as IOException when you press the enter key after entering the value. This is a known exception i.e. it is known to the compiler. Hence compiler doesn't allow to compile the program if it is not said that the main function throws an IOException.

1.13 First Program of Java

- In Java, no program can be written without a class as already discussed. Hence we will start writing our program with the keyword "class" followed by a name for the class.
- Every program should have a main method which indicates the entry point of the program.
- Some important points to be noted for the **main ()** method of Java:
  1. The main() method should always be **public**. The main method is called by the JVM. JVM is an outsider. Hence the access specifier of this method should always be public so that the outsider i.e. JVM is able to call this function.
  2. There can be multiple classes in a program. Multiple classes can have the method main(). In such a case, the class that has the main method which is to be taken as the entry point should be the same as the name of the program file. And this function will have to be called without making an object of the class. Hence this main() function must be declared as **static**.

A static member of a class is accessed with the class name while a non-static member is accessed with the object name. The JVM hence calls the main () method of the program name (which is same as that of the class name) with the dot ('.') operator which is a member select operator.

3. The main() method which is expected to be considered as the entry point by the JVM should also be capable of **accepting an array of string arguments**. These arguments can be passed to the main() method from the command line. Hence these arguments are called as command line arguments. The command line arguments will be accepted in the array associated with the declaration of string in the place of parameter list of a function. In Java there is a special method of handling strings as against the complicated method of C++. Here there is a class called as "String". The array of strings in the parameter list of the main() method is nothing but an array of String objects, wherein each object can store a string. We will discuss about Strings in more detail in subsequent chapters.

- To begin with a very simple program in Java, to display the string "Hello Friend" on the standard output device i.e. the monitor of a computer.
- We will then understand the meaning of each word written in the program and also how to execute this program.

Program 1.13.1 : Write a program to display a statement "Hello Friends".

```
//This program displays Hello Friends.
class HelloFriends
{
public static void main (String args[ ])
   {
        System.out.println("Hello Friends");
   }
}
```

Output

```
C:\java programs\WORLD>javac HelloFriends.java

C:\java programs\WORLD>java HelloFriends
Hello Friends
C:\java programs\WORLD>
```

Explanation of the above program

- The first line that starts with "//", indicates that this is a comment line. The comments are written in a program for future reference of a programmer.
- The comments are neglected by the compiler (compiler is a software tool that converts the Java program to the byte code). In comments you can write anything, it may be helpful to understand your program by somebody else or by yourself after you see the program after a long period.
- The class HelloFriends begins with the declaration "class HelloFriends". The class definition indicates what data members and method members are there in the class.
- The main() method must be declared as public and static as discussed earlier. Also the main() method must be capable of accepting an array of string objects as discussed. The main() method has no return type, hence "void".

- The println() method is called with the string to be displayed "Hello Friends". This is a static method called with the syntax class_name.method_name(). But, in this case the method is called with the variable "out" as discussed in the previous section. We will see how to execute the above program in Section 1.14.

1.14 Installing and Implementing Java

- The question that should come to your mind after reading the above program is that how can you write this program on your PC and see the output.
- The system in which you run the program should have the Java Development Kit (JDK). To do programming in Java on your machine you need to download JDK (Java Development Toolkit) and install the same.
- The JDK is available on the website www.java.com. This software is downloaded and then installed by double clicking on it. The software once installed you can write and execute Java programs.
- This procedure of writing, compiling and executing the programs will be seen in this section. Also you need to set a path for the bin folder of you JDK using the following steps. The following steps are to be followed to do the same

Step 1 : Right click on the "My computer" icon and go to properties. Go to the "Advanced settings" tab and click on "Environment variables". In the "System variables" box select the variable "Path" and click the button "Edit" below that box.

Step 2 : A new window opens. In the "Variable value" field, type the path of the bin folder of the JDK followed by semicolon (;). For example if the JDK is installed in "C" drive itself the path to be given is ";C:\jdk1.5.0_02\bin".

1.14.1 Java Development Kit (JDK)

- JDK is developed by Sun Microsystems. Java Development Toolkit as the name says is a toolkit required on your system for programming in Java. This toolkit mainly has a compiler that can compile the java programs.
- Besides the compiler it also has some packages that can be imported by your program to use some of the in-built classes and their methods.
- The package java.lang which has the class System used to display a data; the package java.util etc. are all available in the JDK.
- You can execute the Java program on a Windows based system that has a JDK. The steps to be followed to execute a Java program on Windows are :

Step 1 : Type your program on notepad and store it as the class name.java. For example the above program must be stored as HelloFriends.java.

Step 2 : Compile the program in command prompt using the Java compiler i.e. javac. The path should be according to folder where your program is stored. The syntax for compiling the program is :

C:\....>javac class_file_name.java

for e.g. as shown in the output of the Program 1.13.1

C:\java programs\WORLD>javac HelloFriends.java

Step 3 : Run the program in command prompt with the following syntax:

C:\....>java class_file_name

for e.g. as shown in the output of Program 1.13.1

C:\java programs\WORLD>java HelloFriends

These steps are to be followed to run the program written by you.

1.15 Type Casting and Type Conversion in Java

- A data type can be automatically converted in case if the data to be copied in another variable can fit into the size for the destination data type. For e.g. int variable can be automatically given to a long type of variable.
- But this implicit conversion does not work for all type of conversions. For example a float variable cannot be given to a byte type of variable. In such case we need to go for explicit conversion.
- When a variable of one type is assigned to a variable of another type it is called as automatic conversion. This is possible as discussed only if the two data types as compatible and the destination variable's data type has more space in memory compared to the source variable's data type.
- In case, if the float type variable is to be assigned into a variable of byte type, it won't happen automatically; instead it will have to be done explicitly with the use of type casting operator. The syntax of such a conversion is :

  (*target_data_type*) source_data_type_variable

  For e.g. (byte) x;

  where 'x' may be a variable of float type.

  The Program 1.15.1 demonstrates explicit and implicit type castings.

Program 1.15.1 : Write a program to demonstrate automatic type conversion (implicit) and type casting (explicit).

```
class Conversion {
public static void main(String args[ ])
 {
 byte b;
 int i = 500;
 float f = 123.942f;
 System.out.println("\nConversion of float to byte.");
 b = (byte) f;
 System.out.println("float was " + f + " and byte is " + b);
 System.out.println("\nConversion of int to float.");
 f = i;
 System.out.println("integer was " + i + " and float is " + f);
 }
}
```

Output

```
C:\java programs\WORLD>java Conversion
Conversion of float to byte.
float was 123.942 and byte is 123
Conversion of int to float.
integer was 500 and float is 500.0
C:\java programs\WORLD>
```

Explanation

- A byte 'b' is declared; integer 'i' is declared and initialized to value 500; float 'f' is declared and initialized to 123.942.

Note : If a value is just declared with fraction part, then it is considered as a double value. If you want it to be considered as a float, then the letter 'f' must be written at the end as shown in the Program 1.15.1 i.e. float f=123.942f.

- The statement b= byte (f), is explicit casting of a float number into byte type using the type casting operator.
- To display multiple values or text using the println() method, you need to use the string concatenation operator i.e. '+' sign as seen in the Program 1.15.1. The statement :

  System.out.println("float was " + f + " and byte is " + b);
- This statement will display the string "float was" as it is, then it will display the value of the variable 'f', then the string "and byte is" will again be displayed as it is and then the value of b.
- The integer is implicitly casted to a float value as shown in the statement : f = i. The integer value of variable 'i' is copied into the float variable 'f'.
- These values are also displayed using the println() statement.
- These type casting rules are almost similar to that in C++

1.16 Solved Programs

Program 1.16.1 : Write a Java program to accept a number and display its square.

```
import java.util.*;
class Square
{
public static void main(String args[ ])
 {
 int i,res;
 Scanner sc = new Scanner (System.in);
 System.out.println("\nEnter a number:");
 i=sc.nextInt();
 res=i*i;
 System.out.println("The Square=" + res);
 }
}
```

Output

```
C:\java programs>java Square

Enter a number:
3
The Square=9
C:\java programs >
```

Explanation

- This program begins with a statement to import the package "java.util". This package is required for the class Scanner. The class Scanner as discussed earlier is required to accept the input from user with the help of the nextInt() method.

- An object of the class Scanner is made as discussed earlier.
- To find the square the number is simply multiplied by itself and the result is stored in the variable "res".
- Let us see this program using the BufferedReader class.

```
import java.io.*;
class Square
{
  public static void main(String args[ ]) throws IOException
  {
    int i,res;
    BufferedReader br = new BufferedReader (new InputStreamReader (System.in));
    String str;
    System.out.println("\nEnter a number:");
    str=br.readLine();
    i=Integer.parseInt(str);
    res=i*i;
    System.out.println("The Square=" + res);
  }
}
```

Output

```
C:\java programs >java Square
Enter a number:
3
The Square=9
C:\java programs >
```

Explanation

- This program begins with a statement to import the package "java.io". This package is required for the class BufferedReader. The class BufferedReader as discussed earlier is required to accept the input from user with the help of the readLine() method. The main() method is associated with the statement "throws IOException" indicates the compiler that this method will throw an IOException. As discussed earlier it is required for the readLine() method.
- An object of the class BufferedReader is made as discussed earlier and also an object is made of the class String. This object is used to store the string accepted from the user. We will see more details of this String object in chapter 4. For time being, we can assume that the "str" is as good as the variable of type string.
- The user entered number is accepted as a string in the String object "str". Then the string is parsed to an integer using the static method parseInt() in the wrapper class Integer. To find the square the number is simply multiplied by itself and the result is stored in the variable "res".
- As you must have noticed that using Scanner class to accept user input is very simple. Hence here onwards we will be using the Scanner class, unless it is required to demonstrate something related to the BufferedReader class.

Program 1.16.2 : Write a Java program to accept two numbers and display its product.

```
import java.util.*;
class Product
{
public static void main(String args[ ])
 {
   int a,b,res;
   Scanner sc = new Scanner (System.in);
   System.out.println("Enter two numbers:");
    a=sc.nextInt();
    b=sc.nextInt();
    res=a*b;
    System.out.println("The Product=" + res);
  }
}
```

Output

```
C:\java programs>javac Product.java
C:\java programs>java Product
Enter two numbers:
4
3
The Product=12
C:\java programs>
```

Program 1.16.3 : Write a program to calculate the simple interest taking principal, rate of interest and number of years as inputs from user.

```
import java.util.*;
class SimpleInterest
{
public static void main(String args[ ])
{
  float p,n,r,si;
  Scanner sc = new Scanner (System.in);
  System.out.println("Enter Principal amount, no. of years and rate of interest:");
  p=sc.nextFloat();
  n=sc.nextFloat();
  r=sc.nextFloat();
  si=p*n*r/100;
  System.out.println("The Simple Interest=" + si);
 }
}
```

Output

```
C:\java programs>javac SimpleInterest.java
C:\java programs>java SimpleInterest
Enter Principal amount, no. of years and rate of interest:
1000
3
10
The Simple Interest=300.0
C:\java programs>
```

Explanation

- All the variables are taken as float as these parameters are normally with fraction part included like principal has rupees and paisa; rate of interest is fraction etc.
- Hence the method to accept the float type is used i.e. nextFloat() in the class Scanner.

Program 1.16.4 : Write a program to accept the length and breadth of a rectangle from the user. Calculate and display the area and perimeter.

```
import java.util.*;
class Rectangle
{
public static void main(String args[ ])
 {
  float l,b,area,perimeter;
  Scanner sc = new Scanner (System.in);
  System.out.println("Enter length and breadth of rectangle:");
  l=sc.nextFloat();
  b=sc.nextFloat();
  area=l*b;
  perimeter=2*(l+b);
  System.out.println("The Area=" + area + "\nThe Perimeter=" + perimeter);
 }
}
```

Output

```
C:\java programs>javac Rectangle.java
C:\java programs>java Rectangle
Enter length and breadth of rectangle:
5
6.3
The Area=31.5
The Perimeter=22.6
C:\java programs>
```

Program 1.16.5 : Write a program to accept two numbers from user and display the greatest of two using the conditional operator.

```
import java.util.*;
class Larger
{
public static void main(String args[ ])
{
  int a,b,large;
  Scanner sc = new Scanner (System.in);
  System.out.println("Enter two numbers:");
  a=sc.nextInt();
  b=sc.nextInt();
  large=(a>b)?a:b;
  System.out.println("The Larger no.=" + large);
 }
}
```

Output

```
C:\java programs>javac Larger.java
C:\java programs>java Larger
Enter two numbers:
5
7
The Larger no.=7
C:\java programs>
```

Program 1.16.6 : Write a program to find the area of a circle.

```
import java.util.*;
class Circle
{
public static void main(String args[ ])
{
  float r,area;
  Scanner sc = new Scanner (System.in);
  System.out.println("Enter radius:");
  r=sc.nextFloat();
  area=3.14f*r*r;
  System.out.println("The Area=" + area);
}
}
```

Output

```
C:\java programs>javac Circle.java
C:\java programs>java Circle
Enter radius:
5
The Area=78.5
C:\java programs>
```

Explanation :

- The value of pi i.e. 3.14 is to be written followed by a character 'f' to indicate the value is to be considered as a float number, else the fractional value as discussed earlier will be considered as a double value.
- Hence you will notice in the program, we have written area=3.14f*r*r i.e. the value is followed by 'f', since the destination operand is of float type and a double type value cannot be assigned directly to a float type variable.

Program 1.16.7 : Write a program to swap two integers.

```
import java.util.*;
class Swap
{
public static void main(String args[ ])
 {
  int a,b,temp;
  Scanner sc = new Scanner (System.in);
  System.out.println("Enter two numbers:");
  a=sc.nextInt();
  b=sc.nextInt();
  System.out.println("a=" + a + "\nb=" + b);
  temp=a;
  a=b;
  b=temp;
  System.out.println("After Swapping:\na=" + a + "\nb=" + b);
 }
}
```

Output

```
C:\java programs>javac Swap.java
C:\java programs>java Swap
Enter two numbers:
5
7
a=5
b=7
```

```
After Swapping:
a=7
b=5
C:\java programs>
```

Explanation

- For swapping two numbers we need to use a temp variable and store one of the values in that temp variable.
- The other value is to be then copied as shown in the program above.

Program 1.16.8 : Write a program to shift a number three bits left and display the result.

```
import java.util.*;
class Shift
{
public static void main(String args[ ])
 {
  int a,res;
  Scanner sc = new Scanner (System.in);
  System.out.println("Enter a number:");
  a=sc.nextInt();
  res=a<<3;
  System.out.println("After Shifting:\na=" + a + "\nResult=" + res);
 }
}
```

Output

```
C:\java programs>javac Shift.java
C:\java programs>java Shift
Enter a number:
4
After Shifting:
a=4
Result=32
C:\java programs>
```

Program 1.16.9 : Write a program to accept two numbers and display the result of their AND, OR, EXOR and NOT operations.

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

```
int a,b;
Scanner sc = new Scanner (System.in);
System.out.println("Enter 2 numbers:");
a=sc.nextInt();
b=sc.nextInt();
System.out.println("After ANDing:" + (a&b) + "\nAfter ORing:" + (a|b) + "\nAfter EXORing:" +(a^b) + "\nNOT of a:" + (~a) + "\nNOT of b:" + (~b));
}
}
```

Output

```
C:\java programs>javac BitwiseOperations.java
C:\java programs>java BitwiseOperations
Enter 2 numbers:
3
5
After ANDing:1
After ORing:7
After EXORing:6
NOT of a:-4
NOT of b:-6
C:\java programs>
```

1.17 Command Line Arguments

- As discussed in earlier sections, the main() method must be capable of accepting an array of strings from the command line. The parameters can be passed while executing the program i.e. while writing the statement and passing the parameters the syntax is :

  "java class_name parameter list"

- These arguments are normally used by the main() method as input from user.

Program 1.17.1 : Write a program to accept a two digit number as command line argument and display the two digits separately.

```
class TwoDigit
{
public static void main(String args[ ])
{
  int no,d1,d2;
  no=Integer.parseInt(args[0]);
  d2=no%10;
  d1=no/10;
  System.out.println("Digit 1:" + d1 + "\nDigit 2:" + d2);
 }
}
```

Output

```
C:\java programs>javac TwoDigit.java
C:\java programs>java TwoDigit 67
Digit 1:6
Digit 2:7
C:\java programs>
```

Explanation

- As seen the parameter is passed as an argument on the command line while executing the program i.e the statement

  C:\java programs>java TwoDigit 67

- The number "67" is passed here as a parameter to the main() method. The parameter passed as a string is received by the main() method in the string array i.e args[ ] in this case.
- The first string is received in args[0], second in args[1] and so on. The command line arguments as discussed earlier can be accepted only as strings.
- Hence these strings are to be converted to integers or in other words the integer is to be extracted from the string. This can be done by the static method parseInt() in the wrapper class Integer.
- Thus in this program we have parsed the string args[0] into integer using the static method parseInt() and passed the string from which the integer is to be extracted.

Note : If you don't pass an argument to args[0], then there will be an error generated i.e exception called as ArrayIndexOutOfBoundsException. We will see in more details about the exceptions in the chapter 8.

Program 1.17.2 : Write a program to find the largest of three integers accepted from command line.

```
class Largest
{
public static void main(String args[ ])
 {
  int no1,no2,no3,large;
  no1=Integer.parseInt(args[0]);
  no2=Integer.parseInt(args[1]);
  no3=Integer.parseInt(args[2]);
  large=(no1>no2)?no1:no2;
  large=(large>no3)?large:no3;
  System.out.println("Largest no:" + large);
 }
}
```

Output

```
C:\java programs>javac Largest.java
C:\java programs>java Largest 6 34 18
Largest no:34
C:\java programs>
```

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.

Program 1.19.2 : Write a program to display the first ten natural numbers.

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

Output

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

Explanation

- In the above program we have used for loop. The variable 'i' is initialized to 1 in the initialization statement. In the condition statement it is checked if the value has reached 10.
- If not reached then the control enters inside the loop i.e. the curly braces. In the curly braces we are printing the value of 'i' itself, hence displaying 1 to 10.

Program 1.19.3 : Write a program to display the first n natural numbers, where the value of n is taken from user.

```
import java.util.*;
class Natural
{
public static void main(String args[ ])
{
  int i,n;
  Scanner sc = new Scanner (System.in);
  System.out.print("Enter the value of n:");
  n=sc.nextInt();
```

```
for(i=1;i<=n;i++)
{
        System.out.println(i);
}
}
}
```

Output

```
Enter the value of n:7
1
2
3
4
5
6
7
```

Explanation

- In this program we accept the value from user, using the readLine() method and take the value of i from 1 to that number. We are displaying the value of i, and hence displaying the natural numbers from 1 to n.

Program 1.19.4 : Write a program to find the factorial of a number.

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

Output

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

Explanation

- In this program we accept a number from user in the variable n. We initialize the value of i to 1 and also the value of the variable fact to 1. Remember, initialization part of the for loop can have multiple initializations. Update or increment / decrement operations can be multiple. But the condition statement has to be only one, if required to be combined with AND / OR operator.
- Now we keep on incrementing the value of i and multiplying this to the variable fact. Hence first time the value of fact is multiplied by 1, then by 2, then by 3 and so on. Thus we implement the logic of fact = $1 \times 2 \times 3 \times 4 \times \ldots\ldots \times n$.
- Note the value of fact is initialized to 1, because it is to be multiplied by different numbers. Hence if we don't initialize it, some random number will be there in the variable fact which will be multiplied by these numbers i.e. 1, 2, 3... Also we cannot initialize the variable fact to 0, else it will always be multiplied with 0 and result will always be 0.

Program 1.19.5 : Write a program to display the multiplication table of a user entered number. The table must be upto 10.

```
import java.util.*;
class MultiplicationTable
{
public static void main(String args[ ])
{
 int i,n;
 Scanner sc = new Scanner (System.in);
 System.out.print("Enter the value of n:");
 n=sc.nextInt();
 for(i=1;i<=10;i++)
 {
  System.out.println(n + " X " +i + " = " + (n*i));
 }
}
}
```

Output

```
Enter the value of n:4
4 X 1 = 4
4 X 2 = 8
4 X 3 = 12
4 X 4 = 16
4 X 5 = 20
4 X 6 = 24
4 X 7 = 28
4 X 8 = 32
4 X 9 = 36
4 X 10 = 40
```

Explanation

In this program we accept a number from user in the variable n. We initialize the value of i to 1 and also then keep on calculating the value of n*i, until the value of i is equal to 10. These values are displayed in the multiplication table format.

Program 1.19.6 : Write a program to display first n odd numbers.

```
import java.util.*;
class OddNumbers
{
public static void main(String args[ ])
    {
        int i,n;
        Scanner sc = new Scanner (System.in);
        System.out.print("Enter the value of n:");
        n=sc.nextInt();
        for(i=1;i<=2*n;i+=2)
        {
        System.out.println(i);
        }
    }
}
```

Output

```
Enter the value of n:6
1
3
5
7
9
11
```

Explanation

- In this program we multiply the value of i by 2 and add to it 1. To start the odd numbers from 1, we have to make t value of i initialized to 0, and hence go upton-1.
- Since we need a total of n odd numbers, starting from 0 when we reach n-1, we have already covered n numbers.

Program 1.19.7 : Write a program to display odd numbers upto n.

```
import java.util.*;
class OddNumbers
{
public static void main(String args[ ])
{
 int i,n;
Scanner sc = new Scanner (System.in);
System.out.print("Enter the value of n:");
```

```
n=sc.nextInt();
for(i=1;i<=n;i+=2)
{
 System.out.println(i);
}
}
}
```

Output

```
Enter the value of n:7
1
3
5
7
```

Explanation

- In this program we increment the value of i by 2. As already discussed it is not necessary that only increment decrement statements can be written in the updating statements; we can also put some other operations like the addition statement written in this program.
- The condition checked over here is that i must be less than or equal to n. As it is specified that the odd numbers are to be printed upto n, we want the value of i to reach maximum upto n.

Program 1.19.8 : Write a program to calculate and display the sum of first n natural numbers.

```
import java.util.*;
class Sum {
public static void main(String args[ ])
{
 int i,n,sum=0;
 Scanner sc = new Scanner (System.in);
 System.out.print("Enter the value of n:");
 n=sc.nextInt();
 for(i=1;i<=n;i++)
 {
  sum+=i;
 }
 System.out.println("Sum="+sum);
}
}
```

Output

```
Enter the value of n:4
Sum=10
```

Explanation :

- In this program we are changing the values of I from 0 to n and in each case we are adding the value of I to sum.
- Hence at the end, the sum of all numbers from 1 to n is stored in the variable sum, which is displayed.

Program 1.19.9 : Write a program to calculate and display the first n even numbers.

```
import java.util.*;
class Even
{
public static void main(String args[ ])
{
 int i,n,sum=0;
 Scanner sc = new Scanner (System.in);
 System.out.print("Enter the value of n:");
 n=sc.nextInt();
 for(i=2;i <=2*n;i+=2)
 {
  System.out.println(i);
 }
 }
}
```

Output

```
Enter the value of n:5
2
4
6
8
10
```

Explanation

In this program we are changing the values of I from 2 to n and in each case we are adding the value of 2*i to sum.

Program 1.19.10 : Write a program to display first n elements of Fibonacci series.

```
import java.util.*;
class Fibonacci
{
public static void main(String args[ ])
{
 int i,n,a,b,c;
 a=0;
 b=1;
```

```
Scanner sc = new Scanner (System.in);
System.out.print("Enter the value of n:");
n=sc.nextInt();
System.out.println("Fibonacci Series:\n0\n1");
for(i=1;i<=n-2;i++)
{
  c =a+b;
  System.out.println(c);
  a=b;
  b=c;
}
}
}
```

Output

```
Enter the value of n:10
Fibonacci Series:
0
1
1
2
3
5
8
13
21
34
```

Note : Fibonacci series is a series wherein the first two elements are 0 and 1. Thereafter the next values are the sum of previous two elements. Hence it can be said that the series is 0,1,1,2,3,5,8,13,21,34,55 . . .

Explanation

- We first display the values 0 and 1 i.e. the first two elements of Fibonacci series. Also these two values are initialized in the variables a and b.
- In every iteration, the next value is calculated and stored in the variable c and is printed. Also the values of a and b are updated accordingly i.e. the latest two values calculated must always be in a and b.
- The condition statement of the for loop is important in this case. Since the first two elements are already printed the counting of printing the elements of Fibonacci series in the loop must go only upto n-2 i.e. two values less which are printed outside the loop.

Program 1.19.11 : Write a program to calculate the value of the following series : $1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \ldots + \frac{1}{n}$

```
import java.util.*;
class Sum
{
public static void main(String args[ ])
{
 int i,n;
 float sum=0;
 Scanner sc = new Scanner (System.in);
 System.out.print("Enter the value of n:");
 n=sc.nextInt();
 for(i=1;i<=n;i++)
 {
  sum=sum+1.0f/i;
 }
 System.out.println("Sum=" + sum);
}
}
```

Output

```
Enter the value of n:5
Sum=2.2833335
```

Explanation

- In this program we have taken the sum variable as float and initialized it to 0. This is because the result of this expression has to be a fraction value. And the terms are to be calculated and added to this sum, hence initially 0.
- Another important thing is we have written sum=sum+1.0f/i. Instead of using 1, we have used 1.0. If we would have written 1 over there, the result would always be 0. Because 1/i, for i as integer and even 1 as integer, the result is always 0 i.e. the quotient.

Program 1.19.12 : Write a program to calculate the value of the following series : $1 + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \ldots + \frac{1}{n!}$

```
import java.util.*;
class Sum
{
public static void main(String args[ ])
{
 int i,n,fact=1;
 float sum=0;
 Scanner sc = new Scanner (System.in);
```

```
System.out.print("Enter the value of n:");
n=sc.nextInt();
for(i=1;i<=n;i++)
{
 fact=fact*i;
 sum=sum+1.0f/fact;
}
System.out.println("Sum=" + sum);
}
}
```

Output

```
Enter the value of n:3
Sum=1.6666666
```

Program 1.19.13 : Write a program to calculate the sine of an angle using the following series for x as the angle in radians : $\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots \frac{x^n}{n!}$. Accept the angle from user in degrees and display the result correct upto 6 decimal places.

```
import java.util.*;
class Sine
{
public static void main(String args[ ])
{
 int i,sign=-1,fact=1;
 double x,sum,num,term;
 Scanner sc = new Scanner (System.in);
 System.out.print("Enter angle in degrees:");
 x=sc.nextDouble();
 x=x*3.14159/180;
 sum=term=x;
 for(i=3;term>=0.0000001;i+=2)
 {
  fact=fact*i*(i-1);
  num=Math.pow(x,i);
  term=num/fact;
  sum=sum+num/fact*sign;
  sign=sign*-1;
 }
 System.out.println("Sum=" + sum);
}
}
```

Output

```
Enter angle in degrees:90
Sum=1.0030828607606848
```

Explanation :

- The formula given in the question is for an angle in radians. Hence we first take the angle in degrees taken from user and then convert it in radians.
- The first term is x, hence the value of x is first put into the variable term and hence into the variable sum. The variable term is used to calculate the value of each term and then to be added in the variable sum, which is used to calculate the value of the entire series.
- In the for loop, the value of i is started from 3, as seen in the series. The condition is that term should be less than 0.0000001 i.e. term's weight must be less than the 6^{th} decimal place and hence allowing the change in the sum upto 6^{th} decimal place.
- The factorial is calculated for every term in the variable fact. The previous value of fact is just updated by multiplying the next two numbers to the previous factorial. For example if the variable fact has the factorial of 3, when multiplied by 5 and 4 (i and i-1), the resultant is the factorial of 5.
- The numerator is simply calculated by using the Math.pow() method.
- The term is calculated by dividing the numerator by factorial.
- Then this term is added with the sum with proper sign. The variable sign is used for this. The sign is initialized to -1 and in every iteration its sign is changed by multiplying it with -1.
- The value of i, is incremented by 2; according to the series the terms require the factorial and exponent incremented by two w.r.t. its previous value.

Program 1.19.14 : Write a program to determine the sum of the series : $1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + ... \frac{1}{n}$

```
import java.util.*;
class Sum
{
public static void main(String args[ ])
 {
 int i,n,sign=1;
 float sum=0;
 Scanner sc = new Scanner (System.in);
 System.out.print("Enter value of n:");
 n=sc.nextInt();
 for(i=1;i<=n;i++)
 {
  sum=sum+1.0f/i*sign;
  sign=sign*-1;
 }
 System.out.println("Sum=" + sum);
}
}
```

Output

```
Enter value of n:4
Sum=0.5833334
```

Program 1.19.15 : Write a program to display the value of s for t=1, 5, 10, 15, 20, ...100 : $s = s_0 + v_0 + \frac{1}{2} a t^2$

```
import java.util.*;
class Sum
{
public static void main(String args[ ])
 {
  int t;
  float s,s0,v0,a;
  Scanner sc = new Scanner (System.in);
  System.out.print("Enter values of s0, v0 and a:");
  s0=sc.nextFloat();
  v0=sc.nextFloat();
  a=sc.nextFloat();
  s=s0+v0+0.5f*a;
  System.out.println("t\ts\n1\t" + s);
  for(t=5;t<=100;t+=5)
  {
    s=s0+v0+0.5f*a*t*t;
    System.out.println(t+"\t" + s);
  }
 }
}
```

Output

```
Enter values of s0, v0 and a:0
0
9.8
t    s
1    4.9
5    122.5
10   490.0
15   1102.5
20   1960.0
25   3062.5
30   4410.0
35   6002.5
40   7840.0
```

```
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++)
```

```
 {
        System.out.print("Hi\t");
 }
 System.out.println();
 }
}
}
```

Output

```
Hi    Hi
Hi    Hi
Hi    Hi
Hi    Hi
Hi    Hi
```

Explanation

- The value of i is first initialized to 1, and the value of j is also initialized to 1. The statement in the inner for loop prints the string "Hi" for two times in the same line. Thereafter the value of j doesn't satisfy the condition and the control comes outside the inner for loop.
- The next operation in the outer for loop is to go to the next line i.e. println() method.
- After that the value of i is incremented. The value of i is less than 5, hence it repeats the same for the next line as seen in the output. For this line the value of j is again initialized to 1.

Note : Whenever the execution of a for loop repeats, the initialization statements are executed again at the beginning.

- This process is repeated for five times. Hence the same thing is displayed for five times as shown in the output.

Program 1.19.17 : Write a program to display the following

```
1
11
111
1111
11111
```

```
class Pattern
{
public static void main(String args[ ])
{
 int i,j;
 for(i=1;i<=5;i++)
 {
 for(j=1;j<=i;j++)
 {
   System.out.print("1");
```

```
 }
 System.out.println();
 }
 }
}
```

Output

```
1
11
111
1111
11111
```

Explanation :

- The inner loop should display the number of 1s equal to the line number. Hence the value of j is taken from 1 to i, where i keeps the record of line number.
- Hence you will notice in the last four programs we have written, i has always kept a track of line number. This is just to make it convenient for understanding.

Program 1.19.18 : Write a program to display the following :

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

```
class Pattern
{
public static void main(String args[ ])
{
 int i,j;
 for(i=1;i<=5;i++)
 {
  for(j=1;j<=i;j++)
  {
        System.out.print("*");
  }
  System.out.println();
 }
}
}
```

Output

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

Program 1.19.19 : Write a program to display the following for the user specified number of lines

```
*
**
***
****
*****
|
|
n lines
```

```
import java.util.*;
class Pattern
{
public static void main(String args[ ])
 {
  int i,j,n;
  Scanner sc = new Scanner (System.in);
  System.out.print("Enter number of lines:");
  n=sc.nextInt();
  for(i=1;i<=n;i++)
  {
   for(j=1;j<=i;j++)
   {
            System.out.print("*");
   }
   System.out.println();
  }
 }
}
```

Output

```
Enter number of lines:6
*
**
***
****
*****
******
```

Program 1.19.20 : Write a program to display the following for the user specified number of lines

```
*
**
***
****
*****
******
*******
********
|
|
n lines
```

```
import java.util.*;
class Pattern
{
public static void main(String args[ ])
{
 int i,j,n;
 Scanner sc = new Scanner (System.in);
 System.out.print("Enter number of lines:");
 n=sc.nextInt();
 for(i=1;i<=n;i++)
 {
  for(j=1;j<=n-i;j++)
  {
          System.out.print(" ");
  }
  for(j=1;j<=i;j++)
  {
          System.out.print("*");
  }
  System.out.println();
 }
}
}
```

Output

```
Enter number of lines:5
   *
  **
 ***
****
*****
```

Explanation

- In this program there are two operations to be performed in each line viz. print blank spaces and some stars based on the line number. Hence, we have two inner loops. Finally the third thing to be done is to go to the next line.
- The first inner loop is used to display the number of blank spaces. The blank spaces in first line will be five, if the number of lines is six. The next line number of blanks spaces will reduce by one and keep on reducing in every line. Hence in general the number of blank spaces in each line is n-i, where n is the number of lines and i is the current line number.
- The same variable i.e. j can be used in the next for loop also as done in the above program.
- The next for loop i.e. the inner second one is used to display the number of stars equal to the line number.
- Finally once everything is done for the line we use println() method to go to the next line for the next iteration. This process will continue for n lines, where the value of n is taken from user.

Program 1.19.21 : Write a program to display the following asking the user for the number of "*" in the largest line

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

```
import java.util.*;
class Pattern
{
public static void main(String args[ ])
{
 int i,j,n;
 Scanner sc = new Scanner (System.in);
 System.out.print("Enter number of * in the largest line:");
 n=sc.nextInt();
 for(i=1;i<=n;i++)
 {
  for(j=1;j<=n-i;j++)
  {
            System.out.print(" ");
  }
  for(j=1;j<=i;j++)
  {
```

```
            System.out.print("*");
  }
  System.out.println();
 }
 for(i=n-1;i>=1;i--)
 {
  for(j=1;j<=n-i;j++)
  {
            System.out.print(" ");
  }
  for(j=1;j<=i;j++)
  {
            System.out.print("*");
  }
  System.out.println();
 }
}
}
```

Output

```
Enter number of * in the largest line:5
    *
   **
  ***
 ****
*****
 ****
  ***
   **
    *
```

Explanation

- In this program the upper half is same as the previous program. The lower half is added to it, wherein the same things to be printed in the reverse method.
- The inner loops remain the same in this case. But the outerloop counts in the reverse order. Everything else remains the same.

Program 1.19.22 : Write a program to display the following asking the user for the number of lines.

```
1
121
12321
1234321
123454321
12345654321
```

```
import java.util.*;
class Pattern
{
public static void main(String args[ ])
{
 int i,j,n;
 Scanner sc = new Scanner (System.in);
 System.out.print("Enter number of lines:");
 n=sc.nextInt();
 for(i=1;i<=n;i++)
 {
  for(j=1;j<=n-i;j++)
  {
          System.out.print(" ");
  }
  for(j=1;j<=i;j++)
  {
          System.out.print(j);
  }
  for(j=i-1;j>=1;j--)
  {
          System.out.print(j);
  }
  System.out.println();
 }
}
}
```

Output

```
Enter number of lines:5
    1
   121
  12321
 1234321
123454321
```

Explanation

- Here we have to perform 3 operations in each line and then next line i.e. println().
- The first operation is blank spaces. The second operation is printing j i.e. numbers from 1 to line number. The third operation is to print the numbers from i-1 (where i is line number) to 1. For each of these three operations there is an inner for loop.

Program 1.19.23 : Write a program to display the following asking the user for the number of lines.

```
A
ABA
ABCBA
ABCDCBA
ABCDEDCBA
ABCDEFEDCBA
```

```
import java.util.*;
class Pattern
{
public static void main(String args[ ])
{
 int i,j,n;
 Scanner sc = new Scanner (System.in);
 System.out.print("Enter number of lines:");
 n=sc.nextInt();
 for(i=1;i<=n;i++)
 {
  for(j=1;j<=n-i;j++)
  {
          System.out.print(" ");
  }
  for(j=1;j<=i;j++)
  {
          System.out.print((char)(j+64));
  }
  for(j=i-1;j>=1;j--)
  {
          System.out.print((char)(j+64));
  }
  System.out.println();
 }
}
}
```

Output

```
Enter number of lines:6
A
ABA
ABCBA
ABCDCBA
ABCDEDCBA
ABCDEFEDCBA
```

Explanation

- This program is exactly same as the previous one. The only difference being we have to type cast the number to char and add 64 to it to get the ASCII value of 'A'.
- Hence in the above program we have
  System.out.print((char)(j+64));
  instead of the statement of previous program
  System.out.print(j);

Program 1.19.24 : Write a program to display the following asking the user for the number of lines.

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

```
import java.util.*;
class Pattern
{
public static void main(String args[ ])
{
 int i,j,n,k;
 Scanner sc = new Scanner (System.in);
 System.out.print("Enter number of lines:");
 n=sc.nextInt();
 for(i=1,k=1;i<=n;i++)
 {
  for(j=1;j<=i;j++,k++)
  {
            System.out.print(k+" ");
  }
  System.out.println();
 }
 }
}
```

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.

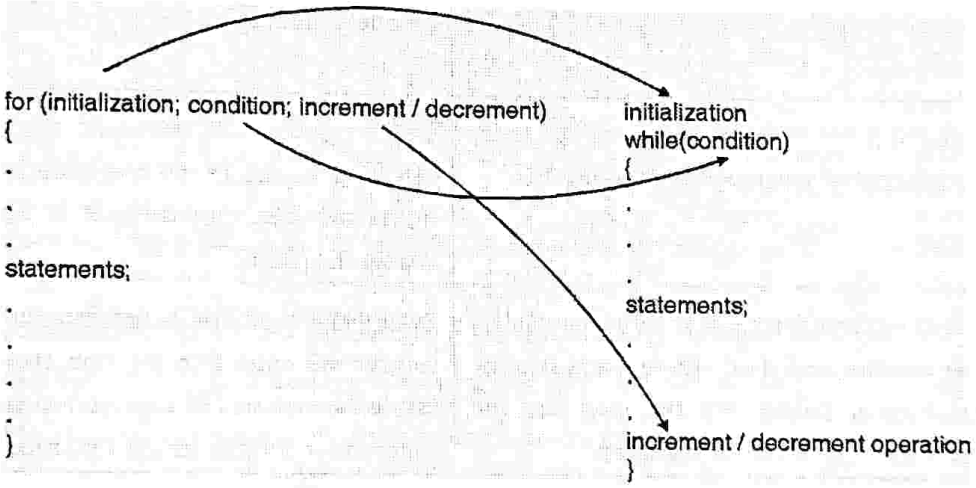


Fig. 1.20.1(a) : Implementation of a while loop related to for loop

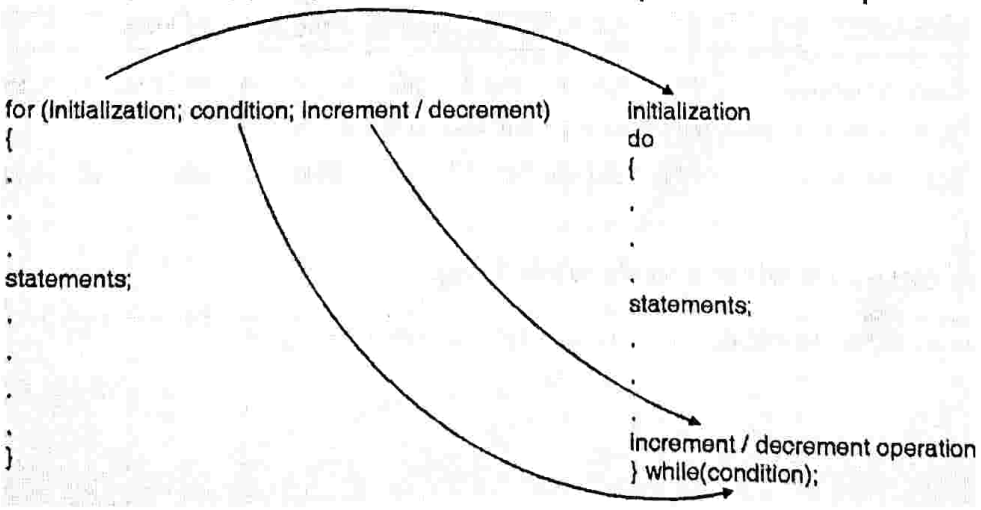


Fig. 1.20.1(b) : Implementation of a do-while loop related to for loop

- 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[ ])
{
```

```
 int i,n;
 Scanner sc = new Scanner (System.in);
 System.out.print("Enter number of lines:");
 n=sc.nextInt();
 i=1;
 while(i<=n)
 {
  System.out.println(i);
  i++;
 }
}
}
```

Output

```
Enter number of lines:5
1
2
3
4
5
```

Explanation

- You will notice here, that the initialization is brought outside the loop i.e. the value of i is initialized to 1 outside the loop.
- The condition is checked thereafter. The statements are executed if the condition is true.
- Then the increment decrement operation i.e. i++ is performed after that and then again the condition is to be checked.

Program 1.20.2 : Write a program to display first n natural numbers using the do-while loop.

```
import java.util.*;
class Natural
{
public static void main(String args[ ])
{
 int i,n;
 Scanner sc = new Scanner (System.in);
 System.out.print("Enter number of lines:");
 n=sc.nextInt();
 i=1;
 do
 {
  System.out.println(i);
```

```
 i++;
}while(i<=n);
}
}
```

Output

```
Enter number of lines:5
1
2
3
4
5
```

Explanation

- Here also you will notice that the initialization is brought outside the loop i.e. the value of i is initialized to 1 outside the loop.
- The statements are executed. Then the increment decrement operation i.e. i++ is performed.
- The condition is executed thereafter. And the statements are again executed if the condition is true.

Program 1.20.3 : Write a program to find the sum of all the digits of a user entered number using the while loop.

```
import java.util.*;
class Digits
{
public static void main(String args[ ])
 {
 int sum=0,n;
 Scanner sc = new Scanner (System.in);
 System.out.print("Enter a number:");
 n=sc.nextInt();
 while(n!=0)
 {
  sum+=n%10;
  n/=10;
 }
 System.out.println("Sum="+sum);
 }
}
```

Output

```
Enter a number:341
Sum=8
```

Explanation

- We have already used this logic of getting the digits of a number i.e. dividing the number by 10 and taking the remainder. And the number is changed by removing the digit considered i.e. divide by 10 and take the quotient.
- The sum is initialized to 0. The sum is added with the digit everytime a digit is found.
- This is repeated until the number becomes 0, i.e. all the digits are considered.

Program 1.20.4 : Write a program to count the number of digits in a user entered number.

```
import java.util.*;
class Digits
{
public static void main(String args[ ])
{
 int count=0,n;
 Scanner sc = new Scanner (System.in);
 System.out.print("Enter a number:");
 n=sc.nextInt();
 while(n!=0)
 {
  count++;
  n/=10;
 }
 System.out.println("Count="+count);
}
}
```

Output

```
Enter a number:3456
Count=4
```

Program 1.20.5 : Write a program to reverse the digits of a user entered number and store in another variable.

```
import java.util.*;
class Reverse
{
public static void main(String args[ ])
{
 int rev=0,n;
 Scanner sc = new Scanner (System.in);
 System.out.print("Enter a number:");
 n=sc.nextInt();
 while(n!=0)
```

```
{
 rev=rev*10+n%10;
 n/=10;
}
System.out.println("Reverse No="+rev);
}
}
```

Output

```
Enter a number:345
Reverse No=543
```

Explanation

- The digits are separated in the same way as seen in the previous programs.
- The variable reverse is initialized to 0. The digit is added to the previous reverse number multiplied by 10. This is done because we first get the lower digit first and we have to put it to the highest digit.

Program 1.20.6 : Write a program that reads a 4 digit integer and breaks it in a sequence of individual digits e.g. 1691 should be displayed as 1 6 9 1.

```
import java.util.*;
class Reverse
{
public static void main(String args[ ])
{
 int rev=0,n;
 Scanner sc = new Scanner (System.in);
 System.out.print("Enter a number:");
 n=sc.nextInt();
 while(n!=0)
 {
  rev=rev*10+n%10;
  n/=10;
 }
 while(rev!=0)
 {
  System.out.print(rev%10+ " ");
  rev/=10;
 }
}
}
```

Output

```
Enter a number:3421
3 4 2 1
```

Program 1.20.7 : Write a program to accept a no. from command line and print the sum of cube of individual digits.

```
import java.util.*;
class Sum
{
public static void main(String args[ ])
{
 int sum=0,n;
 n=Integer.parseInt(args[0]);
 while(n!=0)
 {
  sum=sum+(n%10)*(n%10)*(n%10);
  n/=10;
 }
 System.out.print("Sum="+sum);
}
}
```

Output

```
C:\java programs>java Sum 34
Sum=91
C:\java programs>
```

Explanation

- The digits are separated in the same way as seen in the previous programs.
- The cube is added to the sum after finding each digit.

1.21 The if-else Selective Statement

- This is a very important statement used to check the condition and accordingly execute a set of statements, based on whether the condition is true or false. Hence it is called as selective statement. It selectively executes some statements and doesn't execute some.
- The syntax of the if-else condition is as given below :

```
Syntax of the if-else statement
if(condition)
{
        -
        -
        statements1;
        -
        -
```

```
}
else
{
        -
        -
        statements2;
        -
        -
}
```

- The set of statements named as statements1 in the above syntax are executed if the condition given with the if statement is true. The set of statements statements2 are not executed in this case.
- If the condition specified in the if statement is false then the statements named as statements2 in the above syntax are executed. The set of statements statements1 are not executed in this case.

Note: The else part is optional i.e. we can have a if statement without the else statement. As seen in the above given syntax, only the first half will be there i.e.

```
if (condition)
{
        -
        -
        Statements
        -
        -
}
```

- The subsequent section deals with the programs using if-else statement.

1.21.1 Programs using if-else Statement

Program 1.21.1 : Write a program to check if the user entered number is divisible by 10 or not.

```
import java.util.*;
class Divisible
{
public static void main(String args[ ])
 {
 int sum=0,n;
 Scanner sc = new Scanner (System.in);
 System.out.print("Enter a number:");
 n=sc.nextInt();
 if(n%10==0)
 System.out.println("Divisible by 10");
 else
```

```
System.out.println("Not divisible by 10");
}
}
```

Output

```
Enter a number:56
Not divisible by 10
```

Explanation :

- We know that a number will be said to be divisible by 10 if and only if the remainder after dividing the number by 10 is equal to 0.
- Hence we check this condition using the if statement and accordingly display that the number is divisible by 0 or not.

Program 1.21.2 : Write a program to check if the entered number is prime number or not.

```
import java.util.*;
class Prime
{
public static void main(String args[ ])
{
 int sum=0,n,i=2;
 Scanner sc = new Scanner (System.in);
 System.out.print("Enter a number:");
 n=sc.nextInt();
 while(n%i!=0)
 {
  i++;
 }
 if(n==i)
 {
        System.out.println("Prime Number");
 }
 else
 {
        System.out.println("Not a prime number");
 }
}
}
```

Output

```
Enter a number:17
Prime Number
```

Explanation

- A simple logic is used to find out whether the entered number is prime or not. The logic is similar to the one used by us to check the number being prime or not.
- The number entered by user is checked for divisibility by 2, 3 and so on upto the entered number. If it is divisible then the further checking stops and the control comes out of the while loop.
- In worst case the control will come out of the while loop when the value of i is equal to n, because a number is definitely divisible by itself.
- Finally when the control comes out of the while loop, if n is equal to i, then the number is a prime number else it is not. This is checked using the if-else statement.

Program 1.21.3 : Write a program to display first n prime numbers where the value of n is taken from the user.

```
import java.util.*;
class Prime
{
public static void main(String args[ ])
{
 int n,i,x=1;
 Scanner sc = new Scanner (System.in);
 System.out.print("Enter a number:");
 n=sc.nextInt();
 while(n!=0)
 {
  x++;
  i=2;
  while(x%i!=0)
  {
          i++;
  }
  if(x==i)
  {
          System.out.println(x);
   n--;
  }
 }
}
}
```

Output

```
Enter a number:5
2
3
5
7
11
```

Explanation

- Another while loop is made outside the earlier while loop to keep a track of the number of prime numbers generated. In this case n is taken as a count to keep the track of these numbers.
- Another variable x is continuously incremented after being checked for prime number.
- The value of i is initialized to 2 after every outer loop iteration so that the checking should again start with the divisor being 2.

Program 1.21.4 : Write a program to check if the year entered is leap year or not.

```
import java.util.*;
class Leap
{
public static void main(String args[ ])
 {
 int year;
 Scanner sc = new Scanner (System.in);
 System.out.print("Enter year:");
 year=sc.nextInt();
 if(year%4==0 && year%100!=0 || year%100==0 && year%400==0)
 System.out.println("Leap Year");
 else
 System.out.println("Not Leap Year");
 }
}
```

Output

```
Enter year:2012
Leap Year
```

Explanation :

- This condition mentioned above is hence tested.

Note : A normal year is said to consist of 365 days. But the actual time required for Earth to revolve around the sun is 365.242199 days. Hence to average it out a day is added in every fourth year which gives average of 365.25 days per year. To reach more closer to the above mentioned time, every 100th year is not a leap year and every 400 years is leap year. This brings the average time for a year to be 365.2425 days almost closest.

1.21.2 If-else Ladder or if-else if

- In some cases we have to check multiple cases of a particular condition. In such cases we have to use an if-else ladder. A set of if-else statements as shown below is called as if-else ladder.

Syntax :

```
if(condition)
{
        statements;
}
else
{
        if (condition)
        {
                statements;
        }
        else
        {
                if (condition)
                {
                        statements;
                }
                else
                {
                        if(condition)
                        {
                                statements;
                        }
                        |
                        |
                        |
                        else
                        {
                                statements;
                        }
                }
        }
}
getch();
}
```

- In this case the first condition is checked, if it is true the statements inside the if statement are executed. But if the condition is false it goes to the else statement. Again there is a condition with if; the statements are executed if this second condition is true. Else it again goes to the else and again checks the condition associated with this if statement. Thus if one condition satisfies no other else is checked thereafter.
- A use of such a if-else statement is shown in the program below.

Program 1.21.5 : Write a program to display the class according to the marks scored by a student. The marks scored is taken as input and the class is displayed according to the following range :

Marks	Class
70-100	Distinction
60-69	First Class
50-59	Second Class
40-49	Pass Class
0-39	Fail

```
import java.util.*;
class Grade
{
public static void main(String args[ ])
{
 int marks;
 Scanner sc = new Scanner (System.in);
 System.out.print("Enter marks:");
 marks=sc.nextInt();
 if(marks>=70)
 {
        System.out.println("Distinction");
 }
        else
        {
              if (marks>60)
              {
                    System.out.println("First Class");
              }
              else
              {
                    if (marks>50)
                    {
                          System.out.println("Second Class");
```

```
                }
                else
                {
                        if(marks>40)
                        {
                                System.out.println("Pass Class");
                        }
                        else
                        {
                                System.out.println("Fail");
                        }
                }
        }
    }
}
```

Output

```
Enter marks:93
Distinction
```

Explanation

The above program is said to implement if else ladder with the operation as explained just before the program.

Program 1.21.6 : Write a program to accept a number from command line and to check whether it is armstrong number or not.

```
import java.util.*;
class Armstrong
{
        public static void main(String args[ ])
        {
                int sum=0,digit,n,copy;
                n=Integer.parseInt(args[0]);
                copy=n;
                while(n!=0)
                {
                        digit=n%10;
                        sum+=digit*digit*digit;
                        n/=10;
                }
```

```
            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

2. Default is not necessary. But in case if default is not written and some case occurs which is not considered then there will be no operation performed and the user will not be able to realize the problem in the program.
3. The label can only be value, it cannot be a condition or an expression.
4. The brackets associated with the switch statement can have either the variable or expression and no condition.

- We will see some program examples of this in the subsequent section.

Program 1.22.1 : Write a program to display the user entered single digit number in words.

```
import java.util.*;
class Digit
{
public static void main(String args[ ])
  {
 int n;
 Scanner sc = new Scanner (System.in);
 System.out.print("Enter a single digit no:");
 n=sc.nextInt();
 switch(n)
        {
              case 0:System.out.println("Zero");
              break;
              case 1:System.out.println("One");
              break;
              case 2:System.out.println("Two");
              break;
              case 3:System.out.println("Three");
              break;
              case 4:System.out.println("Four");
              break;
              case 5:System.out.println("Five");
              break;
              case 6:System.out.println("Six");
              break;
              case 7:System.out.println("Seven");
              break;
              case 8:System.out.println("Eight");
              break;
              case 9:System.out.println("Nine");
              break;
      }
  }
}
```

Output

```
Enter a single digit no:7
Seven
```

Explanation

The above program is said to implement the switch-case with the operation as explained just before the program.

Program 1.22.2 : Write a menu driven program to perform add / subtract / multiply / divide / modulus based on the users choice.

```
import java.util.*;
class Menu
{
public static void main(String args[ ])
{
 int choice,a,b;
 Scanner sc = new Scanner (System.in);
 System.out.print("1.Add\n2.Subtract\n3.Multiply\n4.Divide\n5.Modulus\nEnter your choice:");
 choice=sc.nextInt();
 System.out.print("Enter two numbers:");
 a=sc.nextInt();
 b=sc.nextInt();
 switch(choice)
 {
                case 1:System.out.println("Sum="+(a+b));
                break;
                case 2:System.out.println("Difference="+(a-b));
                break;
                case 3:System.out.println("Product="+(a*b));
                break;
                case 4:System.out.println("Quotient="+(a/b));
                break;
                case 5:System.out.println("Remainder="+(a%b));
                break;
                default:System.out.println("Invalid Choice");
 }
}
}
```

Output

```
1.Add
2.Subtract
3.Multiply
4.Divide
5.Modulus
Enter your choice:3
Enter two numbers:4
5
Product=20
```

Explanation

- Such a program wherein the user is given a choice to perform one of the many listed operations is called as a menu driven program.
- Here we have first taken from the user the two numbers on which the operation is to be performed.
- Then we have listed the menu, based on which the user has to enter his choice and the switch case performs the operation indicated by the users choice.

Program 1.22.3 : Write a program using switch-case to display the class according to the marks scored by a student. The marks scored is taken as input and the class is displayed according to the following range:

Marks	Class
70-100	Distinction
60-69	First Class
50-59	Second Class
40-49	Pass Class
0-39	Fail

```
import java.util.*;
class Grade
{
public static void main(String args[ ])
{
 int marks;
 Scanner sc = new Scanner (System.in);
 System.out.print("Enter marks out of 100:");
 marks=sc.nextInt();
 if(marks==100)
 System.out.println("Distinction");
 else
 switch(marks/10)
```

```
{
            case 0:
            case 1:
            case 2:
            case 3:System.out.println("Fail");
            break;
            case 4:System.out.println("Pass Class");
            break;
            case 5:System.out.println("Second Class");
            break;
            case 6:System.out.println("First Class");
            break;
            case 7:
            case 8:
            case 9:System.out.println("Distinction");
            break;
            default:System.out.println("Invalid Marks");
  }
 }
}
```

Output

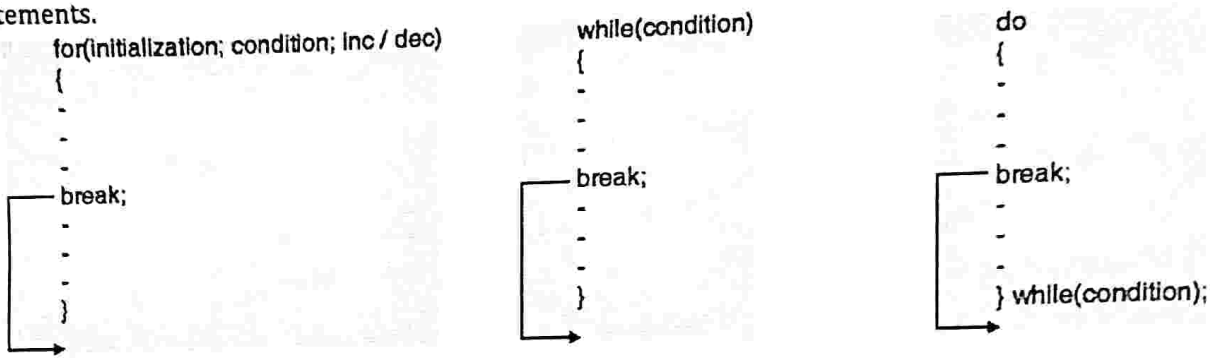
```
Enter marks out of 100:89
Distinction
```

Explanation

- Since the marks can be anything from 0 to 100, we will require 101 cases and a default case. Writing so many cases is almost impossible.
- Hence we have used a special logic based on the condition i.e. take the quotient of the marks divided by 10, and accordingly decide the grade. In this case we can have the quotient from 0 to 9, except for if the marks are 100.
- Hence to take care of this, we have taken the first condition outside the switch case i.e. if marks is 100, then display distinction else the entire switch case.
- We have taken the expression as marks % 10, as already discussed. Now, if the quotient is 0, 1, 2 or 3 it indicates fail. Hence for the cases 0,1 and 2 we haven't written anything.
- For example if the marks are 19, marks % 10 will be 1. Thus it will go to the case 2. Here nothing is written until case 3. Thereafter there is a statement, which displays the result as "Fail".
- Then the break statement transfers the control outside the switch-case statement and no other statements inside the switch are executed.
- Similarly for all other cases with the value of expression from 0 to 9.

1.23 Branching Statements (Break and Continue)

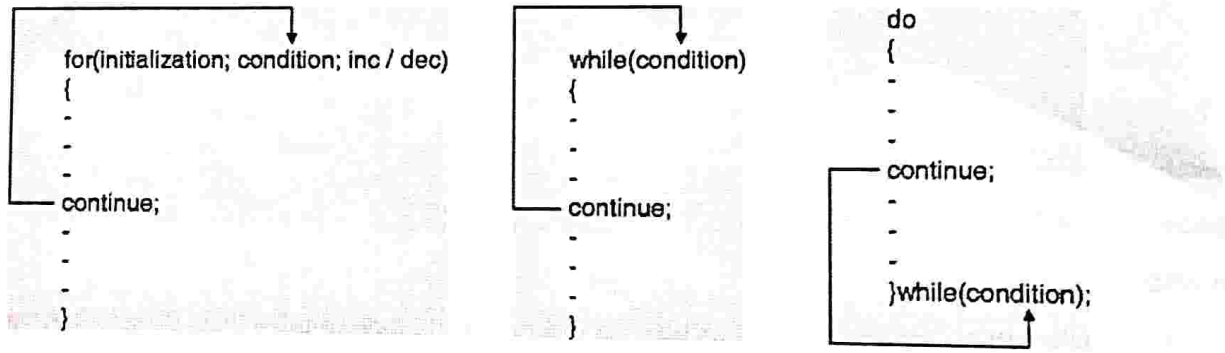
- These statements transfer the control to a different place in the program. We will see the operation of each of these. We have already seen the use of break statement in switch-case. Let us see the use of continue and break in other loop statements
- The break statement neglects the statements after 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;
```

```
Scanner sc = new Scanner (System.in);
for(i=1;i<=10;i++)
{
 System.out.print("Enter a no.:");
 n=sc.nextInt();
 if(n>99)
 break;
 total+=n;
}
System.out.println("Sum="+total);
}
}
```

Output

```
Enter a no.:4
Enter a no.:89
Enter a no.:23
Enter a no.:125
Sum=116
```

Explanation :

- In this program, the break statement is executed if a number greater than 99 (i.e. largest two digit number) is given. The sum is directly displayed without accepting the other numbers in this case.
- The break statement when executed the control is directly transferred outside the loop and the 'cout' statement is directly executed.

Program 1.23.2 : Write a program to demonstrate the use of continue statement.

OR Write a program to accept 5, 2-digit numbers from user and add them. If the user enters a three digit number, this number should be discarded and again accepted. Also an indication must be given to the user that he has entered a number greater than 99 and hence it is not accepted. The sum must be displayed at the end.

```
import java.util.*;
class Continue
{
public static void main(String args[ ])
{
 int n,total=0,i;
 Scanner sc = new Scanner (System.in);
 for(i=1;i<=5;i++)
 {
  System.out.print("Enter a no.:");
  n=sc.nextInt();
```

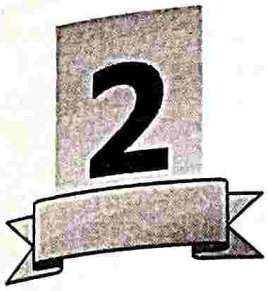
```
 if(n>99)
 {
  System.out.println("Number is greater than 99");
  i--;
  continue;
 }
 total+=n;
}
System.out.println("Sum="+total);
}
}
```

Output

```
Enter a no.:45
Enter a no.:123
Number is greater than 99
Enter a no.:2
Enter a no.:1
Enter a no.:12
Enter a no.:32
Sum=92
```

Explanation

- Here if the number entered by user is greater than 99, then this number is not considered and the user is asked to enter the number again.
- Also the value of i is decremented as this number is not to be considered as one of the five numbers to be added.



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.

Program 2.2.1 : Write a program to make a class called as Circle. It should have three methods namely : accept radius, calculate area and display the area.

```
import java.util.*;
class Circle
{
   private float r,area;
   public void accept(float x)
   {
     r=x;
   }
   public void calculate()
   {
     area=3.14f*r*r;
   }
   public void display()
   {
     System.out.println("Area="+area);
   }
}
class Main
{
   public static void main(String args[])
   {
      float x;
      Scanner sc = new Scanner (System.in);
      System.out.print("Enter Radius:");
      x=sc.nextFloat();
      Circle c=new Circle();
      c.accept(x);
      c.calculate();
      c.display();
   }
}
```

Output

```
Enter Radius:10
Area=314.0
```

Explanation

- The class Circle is first made with the private data (field) members 'r' and "area". The class has three method members namely
  1. accept() to accept the radius. This method is passed with the radius, which is initialized in the variable r.
  2. calculate() to calculate the area.
  3. display() to display the area.
- 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, we have accepted the radius from user. Then an object is created of the class Circle named as 'c'.
- The accept() method for the object 'c' is called passing the radius to the method.
- Then the calculate() method is called and finally the display() method.

Program 2.2.2 : Write an object oriented program in Java that uses Euclid's Algorithm to display the greatest common divisor of two integers.

```
import java.util.*;
class Euclid
{
   private int n1,n2,gcd;
   void accept(int x, int y)
  {
    n1=x;
   n2=y;
  }
 void calculate()
 {
   int temp;
    while(n1%n2!=0)
   {
     n1=n1%n2;
     temp=n1;
     n1=n2;
     n2=temp;
   }
  gcd=n2;
 }
  void display()
 {
  System.out.println("GCD="+gcd);
 }
```

```
}
class Main
{
public static void main(String args[])
 {
  int x,y;
  Scanner sc= new Scanner(System.in);
  System.out.print("Enter two numbers:");
  x=sc.nextInt();
  y=sc.nextInt();
  Euclid e=new Euclid();
   e.accept(x,y);
   e.calculate();
   e.display();
 }
}
```

Output

```
Enter two numbers:15
20
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. accept() to accept the two numbers. This method is passed with the two integers, which is initialized in the variables n1 and n2.
  2. calculate() to calculate the gcd.
  3. display() to display the gcd.
- Since no access specifier is mentioned with the methods they will be taken as default while the field members have an access specifier i.e. private. Hence the field members can be accessed only by the member methods of that class, while the method members can be accessed by all the methods in the same package.
- 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, we have accepted the two numbers from user. Then an object is created of the class Euclid named as 'e'. The accept() method is called by passing the two numbers accepted in the main() method
- Then the calculate() method is called and finally the display() method.

Program 2.2.3 : Create a class employee with data members empid, empname, designation and salary. Write methods get_employee()- to take user input, show_grade() - to display grade of employee based on salary. show_employee() to display employee details.

Let the employee be graded according to salary as follows :

Salary range	Grade
<10000	D
10000-24999	C
25000-49999	B
>50000	A

```
import java.util.*;
class Employee
{
    private int empid;
    private float salary;
    private String empname,designation;
    void get_employee()
    {
     Scanner sc=new Scanner(System.in);
     System.out.println("Enter employee name, designation, ID and salary");
     empname=sc.nextLine();
     designation=sc.nextLine();
     empid=sc.nextInt();
     salary=sc.nextFloat();
    }
  void show_grade()
  {
    if(salary<10000)
    System.out.println("Grade D");
    else if(salary<25000)
    System.out.println("Grade C");
    else if(salary<50000)
    System.out.println("Grade B");
    else
    System.out.println("Grade A");
  }
 void show_employee()
 {
   System.out.println("Name:"+empname+"\nDesignation:"+designation+"\nID:"+empid+"\nSalary:"+salary);
 }
}
```

```
class Main
{
   public static void main(String args[])
   {
      Employee e= new Employee();
      e.get_employee();
      e.show_grade();
      e.show_employee();
   }
}
```

Output

```
Enter employee name, designation, ID and salary
Ajay
Professor
325
51000
Grade A
Name:Ajay
Designation:Professor
ID:325
Salary:51000.0
```

Explanation

- The class Employee is first made with the private data (field) members as given in the problem statement. The class has three method members namely
  1. get_employee() to accept the information of employee. This method has to accept input from user and R hence has the nextLine() method. String data are directly accepted while the others are accepted as string and then converted into required data types using the wrapper class methods.
  2. show_grade() to find and display the grade.
  3. show_employee() to display all the information about the employee.
- 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 the object is created of the class Employee named as 'e'. And the three methods are called.

Program 2.2.4 : Create a class student to store their name, ID no., Marks of Maths, Physics, Chemistry.

```
import java.util.*;
class Student
{
      private int id,p,c,m,t;
      private String name;
```

```
    void accept()
    {
        Scanner sc= new Scanner(System.in);
        String str;
        System.out.println("Enter name, ID and marks in P, C & M");
        name=sc.nextLine();
        id=sc.nextInt();
        p=sc.nextInt();
        c=sc.nextInt();
        m=sc.nextInt();
        t=p+c+m;
    }
    void display()
    {
     System.out.println("Name:"+name+"\nID:"+id+"\nP:"+p+"\nC:"+c+"\nM:"+m+"\nTotal:"+t);
    }
}
class Main
{
    public static void main(String args[])
    {
        Student s= new Student();
        s.accept();
        s.display();
    }
}
```

Output

```
Enter name, ID and marks in P, C & M
Ajay
325
90
90
90
Name:Ajay
ID:325
P:90
C:90
M:90
Total:270
```

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)
```

```
    {
        r=x;
    }
    void calculate()
    {
        area=3.14f*r*r;
    }
    void display()
    {
        System.out.println("Area="+area);
    }
}
c lass Main
{
    public static void main(String args[])
    {
        float x;
        Scanner sc= new Scanner(System.in);
        System.out.print("Enter Radius:");
        x=sc.nextFloat();
        Circle c=new Circle(x);
        c.calculate();
        c.display();
    }
}
```

Output

```
Enter Radius:10
Area=314.0
```

Explanation

- The class Circle is first made with the private data (field) members 'r' and "area". The class has three method members namely
  1. Circle() i.e. parameterized constructor to accept the radius. This constructor is passed with the value of radius (hence parameterized constructor), which is initialized in the variable r.
  2. calculate() to calculate the area.
  3. display() to display the area.
- 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, we have accepted the radius from user. Then an object is created of the class Circle named as 'c'.
- The radius is passed to the constructor while making the object.
- Then the calculate() method is called and finally the display() method.

Program 2.3.2 : Write an object oriented program in Java that uses Euclid's Algorithm to display the greatest common divisor of two integers. The parameterized constructor is to be used to initialize the two numbers. Two methods to calculate() and display().

```
import java.util.*;
class Euclid
{
    private int n1,n2,gcd;
    Euclid(int x, int y)
    {
       n1=x;
       n2=y;
    }
void calculate()
{
   int temp;
   while(n1%n2!=0)
   {
       n1=n1%n2;
       temp=n1;
       n1=n2;
       n2=temp;
   }
 gcd=n2;
}
 void display()
 {
       System.out.println("GCD="+gcd);
 }
}
 class Main
 {
   public static void main(String args[])
   {
    int x,y;
    Scanner sc= new Scanner(System.in);
```

```
    System.out.print("Enter two numbers:");
    x=sc.nextInt();
    y=sc.nextInt();
    Euclid e=new Euclid(x,y);
    e.calculate();
    e.display();
  }
}
```

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 parameterized constructor to accept the two numbers. This constructor is passed with the two integers, which is initialized in 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 accepted the two numbers from user. Then an object is created of the class Euclid named as 'e' by passing the two parameters to the constructor.
- Then the calculate() method is called and finally the display() method.

2.3.1(B) Default Constructor

Program 2.3.3 : Write a program to make a class called as Circle. It should have a default 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()
    {
      Scanner sc= new Scanner(System.in);
       System.out.print("Enter Radius:");
      r=sc.nextFloat();
    }
```

```
void calculate()
{
   area=3.14f*r*r;
}
void display()
{
 System.out.println("Area="+area);
}
}
class Main
{
    public static void main(String args[])
    {
       Circle c=new Circle();
       c.calculate();
       c.display();
    }
}
```

Output

```
Enter Radius:10
Area=314.0
```

Explanation

- The class Circle is first made with the private data (field) members 'r' and "area". The class has three method members namely
  1. Circle() i.e. default constructor to accept the radius. This constructor accepts the value of radius, which is initialized in the variable r.
  2. calculate() to calculate the area.
  3. display() to display the area.
- 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 Circle named as 'c'.
- The constructor is automatically called, which accepts the radius from the user.
- Then the calculate() method is called and finally the display() method.

Program 2.3.4 : Write an object oriented program in Java that uses Euclid's Algorithm to display the greatest common divisor of two integers. Use a default constructor to initialize the two numbers. The calculate() method to calculate the GCD and display() method to display the same.

```
import java.util.*;
class Euclid
{
   private int n1,n2,gcd;
   Euclid()
   {
      Scanner sc= new Scanner(System.in);
      System.out.print("Enter two numbers:");
      n1=sc.nextInt();
      n2=sc.nextInt();
    }
  void calculate()
    {
        int temp;
       while(n1%n2!=0)
       {
          n1=n1%n2;
          temp=n1;
          n1=n2;
         n2=temp;
       }
      gcd=n2;
   }
   void display()
   {
      System.out.println("GCD="+gcd);
   }
}
class Main
{
   public static void main(String args[])
   {
    Euclid e=new Euclid();
    e.calculate();
    e.display();
  }
}
```

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()
```

```
        {
                area=3.14f*r*r;
        }
void display()
        {
                System.out.println("Area="+area);
        }
    }
 class Main
  {
        public static void main(String args[])
        {
         Circle c=new Circle();
         c.calculate();
         c.display();
         Circle c1=new Circle(c);
         c1.calculate();
         c1.display();
        }
  }
```

Output

```
Enter Radius:10
Area=314.0
Area=314.0
```

Explanation

- The class Circle is first made with the private data (field) members 'r' and "area". The class has three method members namely.
  1. Circle() i.e. default constructor to accept the radius. This constructor accepts the value of radius, which is initialized in the variable r. Another constructor is written with a parameter as an object of the same class. The radius of this object is initialized in the object of new constructor.
  2. calculate() to calculate the area.
  3. display() to display the area.
- 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 Circle named as 'c'.
- The constructor is automatically called.
- Then the calculate() method is called and finally the display() method.
- Another object called "c1" is made with the earlier object "c", whose radius is initialized into the radius of the object c1. Similarly the calculate() and display() method are called, hence the radius and area for both the circles is the same.

Program 2.3.6 : Write an object oriented program in Java that uses Euclid's Algorithm to display the greatest common divisor of two integers. Use a default constructor and copy constructor to initialize the two numbers. The calculate() method to calculate the GCD and display() method to display the same.

```
import java.util.*;
class Euclid
{
  private int n1,n2,gcd;
  Euclid()
  {
    Scanner sc = new Scanner (System.in);
    System.out.print("Enter two numbers:");
    n1=sc.nextInt();
    n2=sc.nextInt();
  }
  Euclid(Euclid x)
  {
    n1=x.n1;
    n2=x.n2;
  }

void calculate()
  {
    int temp;
    while(n1%n2!=0)
    {
      n1=n1%n2;
      temp=n1;
      n1=n2;
      n2=temp;
    }
  gcd=n2;
}
  void display()
{
  System.out.println("GCD="+gcd);
}
}
  class Main
{
  public static void main(String args[])
```

```
{
    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.

Program 2.4.1 : Write a program that demonstrates passing objects to a method.

```
import java.util.*;
class Complex
{
  private int x,y;
  Complex(int a,int b)
  {
    x=a;
    y=b;
  }
  void add (Complex a)
  {
    x =x+a.x;
    y =y+a.y;
  }
  void display()
  {
     if(y>=0)
     System.out.println(x+"+i"+y);
     else
     System.out.println(x + "" + y +"i");
   }
}
  class Main
{
   public static void main(String args[])
   {
      int a,b;
      Scanner sc=new Scanner(System.in);
      System.out.println("Enter real and imaginary part of a complex number:");
      a =sc.nextInt();
      b =sc.nextInt();
     Complex c1=new Complex(a,b);
     System.out.println("Enter real and imaginary part of another complex number:");
      a=sc.nextInt();
      b=sc.nextInt();
     Complex c2=new Complex(a,b);
     c1.add(c2);
     c1.display();
  }
}
```

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)
     {
```

```
        x=a;
        y=b;
    }
    Complex()
    {
    }
    Complex add (Complex a)
    {
        Complex c=new Complex();
        c.x=x+a.x;
        c.y=y+a.y;
        return c;
    }
    void display()
    {
      if(y>=0)
      System.out.println(x+"+i"+y);
       else
       System.out.println(x + "" + y +"i");
    }
}
    class Main
    {
      public static void main(String args[])
       {
          int a,b;
          Scanner sc=new Scanner(System.in);
          System.out.println("Enter real and imaginary part of a complex number:");
          a=sc.nextInt();
          b=sc.nextInt();
          Complex c1=new Complex(a,b);
          System.out.println("Enter real and imaginary part of another complex number:");
          a=sc.nextInt();
          b=sc.nextInt();
          Complex c2=new Complex(a,b);
          Complex c3 = new Complex();
          c3=c1.add(c2);
          c3.display();
       }
}
```

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.

Program 2.6.1 : Write a program that demonstrates call by value to swap two numbers.

```
import java.util.*;
class Main
{
   int a,b;
   public static void main(String args[])
   {
         Scanner sc=new Scanner(System.in);
         Main m=new Main();
         System.out.println("Enter two numbers:");
          m.a=sc.nextInt();
          m.b=sc.nextInt();
         System.out.println("Numbers are a="+m.a+"\tb="+m.b);
         swap(m.a,m.b);
         System.out.println("After returning from method\nNumbers are  a="+m.a+"\tb="+m.b);
   }
  static void swap(int a, int b)
  {
          int temp;
          temp=a;
          a=b;
          b=temp;
          System.out.println("After swapping in method\nNumbers are a="+a+"\tb="+b);
   }
}
```

Output

```
Enter two numbers:
4
7
Numbers are a=4 b=7
After swapping in method
Numbers are a=7 b=4
After returning from method
Numbers are a=4 b=7
```

Explanation

- In this case the values of the two variables i.e. 'a' and 'b' are passed to the method swap().
- The values are received in formal parameters namely a and b. These parameters are swapped, but the actual parameters remain unchanged.
- 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. But when the control returns back to the main() method the values are found to be same i.e. not changed.

2.6.2 Call by Reference

- The variables passed by the method are called as actual parameters and the ones received by the called method are called as formal parameters.
- In case of Java, call by reference is possible by passing the object to the method. If we pass the object to a method, we can access the actual parameters of that object according to the access specifiers.
- The object may be received in a different formal parameter name, but still we can access the values of that object.
- We will see how the parameters of the object can be swapped using call by reference or by passing the object in program 2.6.2.

Program 2.6.2 : Write a program that demonstrates call by reference to swap two numbers.

```
import java.util.*;
class Main
{
  int a,b;
  public static void main(String args[])
  {
        Scanner sc=new Scanner(System.in);
        Main m=new Main();
        System.out.println("Enter two numbers:");
        m.a=sc.nextInt();
        m.b=sc.nextInt();
        System.out.println("Numbers are a="+m.a+"\tb="+m.b);
        swap(m);
        System.out.println("After returning from method\nNumbers are a="+m.a+"\tb="+m.b);
  }
  static void swap(Main x)
  {
        int temp;
        temp=x.a;
        x.a=x.b;
        x.b=temp;
        System.out.println("After swapping in method\nNumbers are a="+x.a+"\tb="+x.b);
  }
}
```

Output

```
Enter two numbers:
4
7
Numbers are a=4 b=7
```

```
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.

Program 2.8.1 : Write a program to compare the variable contained in two objects and the method should return the object with the greater value.

```
import java.util.*;
class Compare
{
    private int x;
   Compare(int a)
   {
    x=a;
   }
  Compare compare (Compare c)
  {
     if(x>c.x)
     return this;
     else
     return c;
  }
  void display()
  {
   System.out.println("x="+x);
  }
}
class Main
{
  public static void main(String args[])
  {
    int a;
    Scanner sc= new Scanner(System.in);
    System.out.println("Enter a no:");
     a=sc.nextInt();
     Compare c1=new Compare(a);
     System.out.println("Enter another no:");
      a=sc.nextInt();
     Compare c2=new Compare(a);
     Compare c3;
     c3=c1.compare(c2);
     c3.display();
  }
}
```

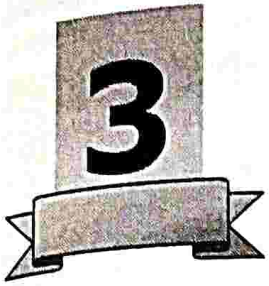
Output

```
Enter a no:
4
Enter another no:
8
x=8
```

Explanation

- The class Compare has just one variable and two methods. Besides these members, the class also has a constructor to initialize the value of the variable x.
- The compare method accepts an object of the class Compare and also returns an object of the same class.
- The method is called by the object c1 and the object c2 is passed to the method. If the variable 'x' of the object c2 is greater, the object c2 is returned by the method and is accepted in the object c3. While if the 'x' of c1 is greater than the current object is to be returned by the method compare(), which is done with the help of keyword "this" i.e. the current object i.e. "this" is returned and accepted in the object c3.

❑❑❑



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.

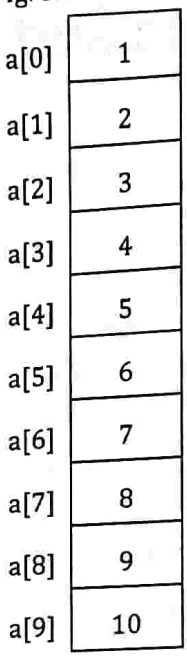


Fig. 3.1.1 : Memory allocation for an array 'a' of 10 integer elements

- 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]);
         }
   }
}
```

Output

```
Enter no of elements:4
Enter a no:1
Enter a no:2
Enter a no:3
Enter a no:4
1
2
3
4
```

Explanation

- The array named 'a' is declared with a size of n elements. The size is taken to be 'n' as the user wants n elements to be stored.
- The first "for" loop is to accept 'n' integers from the user in different index locations numbered from 0 to n – 1. In the loop the value of 'i' is incremented after every iteration and hence the value entered by the user is stored into the next index element.
- The second "for" loop is to display these integers one after the other in different lines. Again in this loop the value of 'I' varies from 0 to n – 1. Here also the value of 'i' is incremented after every iteration and hence the element displayed in every iteration is the consecutive next one in the array.

Program 3.1.2 : Write a program to evaluate the value of the standard deviation (s.d.) and display the result : s.d. = $\sqrt{\frac{\sum_{i=1}^{n}(x_i-\bar{x})^2}{n}}$ where, $\bar{x}$ is the average of all the numbers.

```
import java.util.*;
class SD
{
    public static void main(String args[ ])
    {
      int n,i;
      float avg,sum=0,sd;
      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++)
        {
          sum=sum+a[i];
        }
        avg=sum/n;
        sum=0;
        for(i=0;i<=n-1;i++)
        {
            sum=sum+(a[i]-avg)*(a[i]-avg);
        }
        sum=sum/n;
        sd=(float)Math.pow(sum,0.5);
        System.out.println("SD="+sd);
    }
}
```

Output

```
Enter no of elements:4
Enter a no:2
Enter a no:3
Enter a no:4
Enter a no:3
SD=0.70710677
```

Explanation

- The method to initialize and accept the elements from user is same as in previous programs.
- The average is calculated in the same manner so as to get the average in "float" type data format.
- Then the new sum is calculated to find the value of the term $(x_i - \bar{x})^2$ in the variable sum1.
- Finally the above calculated term is divided by "n" i.e. the total number of elements and its square root is calculated. This is the value of standard deviation which is then displayed.

Program 3.1.3 : Write a program to find the largest of 'n' numbers taken from user.

```
import java.util.*;
class Large
{
   public static void main(String args[])
   {
     int n,i,large;
     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();
    }
    large=a[0];
    for(i=0;i<=n-1;i++)
    {
      if(large<a[i])
      large=a[i];
    }
   System.out.println("Largest no="+large);
  }
}
```

Output

```
Enter no of elements:5
Enter a no:34
Enter a no:1
Enter a no:54
Enter a no:45
Enter a no:50
Largest no=54
```

Explanation

- The method to initialize and accept the elements from user is same as in previous programs.
- The first element is initially assumed to be the largest number.
- Thereafter in the "for" loop, each of the other elements i.e. starting from the second element is compared with the value of the variable "large". Whenever a larger value is found, this new value is copied into the variable "large" using the if statement.

Program 3.1.4 : Write a program to find the smallest of 'n' numbers taken from user.

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

```
        int n,i,small;
        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();
        }
        small=a[0];
        for(i=0;i<=n-1;i++)
        {
          if(small>a[i])
          small=a[i];
        }
      System.out.println("Smallest no="+small);
    }
}
```

Output

```
Enter no of elements:4
Enter a no:23
Enter a no:54
Enter a no:1
Enter a no:34
Smallest no=1
```

Explanation

To find the smallest number, a similar logic is used. Initially, the first element is taken as the smallest number and copied into the variable "small". Then, this variable "small" is compared with each of the remaining elements in the array. If an element found is smaller, then the value of the variable "small", then this number is copied into the variable "small". Finally the value of this variable "small" is displayed.

Program 3.1.5 : Write a program to find and display the reverse of an array.

```
import java.util.*;
class Reverse
{
    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];
    int b[]=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++)
    {
       b[n-i-1]=a[i];
    }
    for(i=0;i<=n-1;i++)
    {
      System.out.println(b[i]);
    }
  }
}
```

Output

```
Enter no of elements:5
Enter a no:1
Enter a no:2
Enter a no:3
Enter a no:4
Enter a no:5
5
4
3
2
1
```

Explanation

- We have accepted the array in the same manner as in the previous program. Here the array is reversed and put into another array. The logic to reverse is very simple in this case, the first element of the array "a" is copied into the last element of array "rev". Initially, the value of "i" is 0, hence the value of the expression n-i-1, will be n–1 i.e. the last element. Hence the element number 0 of the array "a" is copied into the n-1 element of the array "rev".
- When the value of "i" is incremented, the element number 1 of the array "a" is copied into the n-i-1 i.e. second last element of the array "rev". This continues until the value of i, reaches "n".

Program 3.1.6 : Write a program to find an element in an array and display the index of the element. **OR** Write a program to implement sequential search algorithm.

```
import java.util.*;
class Search
{
    public static void main(String args[])
    {
        int n,i,search;
        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();
        }
        System.out.print("Enter the no to be searched:");
        search=sc.nextInt();
        for(i=0;i<=(n-1);i++)
        {
          if(search==a[i])
          break;
        }
        if(i==n)
        System.out.println("No. not found");
        else
        System.out.println("Index =" + i);
    }
}
```

Output

```
Enter no of elements:5
Enter a no:1
Enter a no:2
Enter a no:3
Enter a no:4
Enter a no:5
Enter the no to be searched:4
Index =3
```

Explanation

- The element to be searched is compared with each of the elements. Whenever the element to be searched is found, the "if" condition is satisfied and the break statement transfers the control outside the "for" loop. This is also called as sequential search algorithm i.e. the element is searched sequentially in the list i.e. array of elements.
- The value of "i" is compared with the value of "n"; it they are equal then it indicates that the control has come out of the "for" loop because the number was not found in the array. Hence it displays "Not Found", else it displays the index "i" where the element was found.

Program 3.1.7 : Write a program to sort numbers in ascending order. OR Write a program to implement bubble sorting algorithm for sorting numbers in ascending order.

```
import java.util.*;
class Ascend
{
   public static void main(String args[])
   {
      int n,i,j,temp;
      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-2;i++)
     {
      for(j=0;j<=n-2;j++)
      {
        if(a[j]>a[j+1])
        {
          temp=a[j];
          a[j]=a[j+1];
          a[j+1]=temp;
        }
      }
   }
System.out.println("After Sorting");
for(i=0;i<=n-1;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();
```

```
    int a[ ][ ]=new int [m][n];
    for(i=0;i<=m-1;i++)
    {
      for(j=0;j<=n-1;j++)
      {
        System.out.print("Enter a no:");
        a[i][j]=sc.nextInt();
      }
    }
      for(i=0;i<=m-1;i++)
    {
      for(j=0;j<=n-1;j++)
      {
            System.out.print(a[i][j]+"\t");
      }
      System.out.println();
    }
  }
}
```

Output

```
Enter no of rows and columns:2
3
Enter a no:1
Enter a no:2
Enter a no:3
Enter a no:4
Enter a no:5
Enter a no:6
1    2    3
4    5    6
```

Explanation

- First, the user is asked for number of rows and columns. To accept the elements of the matrix, we need a nested "for" loop as shown. The outer loop has "i" as a counter for the row number and hence is counted from 0 to m-1; while the inner loop has "j" as a counter for the column number and hence is counted from 0 to n-1.
- The displaying of the elements of the matrix also requires a nested for loop as required for accepting the matrix elements from the user.

Program 3.2.2 : Write a program to find the transpose of a matrix of size m × n.

```
import java.util.*;
class Transpose
{
```

```
public static void main(String args[])
{
   int m,n,i,j;
   Scanner sc = new Scanner (System.in);
   System.out.print("Enter values of m and n:");
   m=sc.nextInt();
   n=sc.nextInt();
   int a[ ][ ]=new int [m][n];
   int b[ ][ ]=new int [n][m];
   System.out.println("Matrix A");
   for(i=0;i<=m-1;i++)
   {
      for(j=0;j<=n-1;j++)
      {
       System.out.print("Enter a no:");
       a[i][j]=sc.nextInt();
      }
   }
   for(i=0;i<=m-1;i++)
  {
   for(j=0;j<=n-1;j++)
   {
            b[j][i]=a[i][j];
   }
}
   System.out.println("Transpose Matrix");
  for(i=0;i<=n-1;i++)
  {
   for(j=0;j<=m-1;j++)
   {
     System.out.print(b[i][j]+"\t");
   }
    System.out.println();
  }
}
}
```

Output

```
Enter values of m and n:2  3
Matrix A
Enter a no:1
Enter a no:2
Enter a no:3
Enter a no:4
Enter a no:5
Enter a no:6
Transpose Matrix
1    4
2    5
3    6
```

Explanation

- The method to accept the elements of the matrix is same as the previous program.
- The transpose of a matrix is shown below.

  If matrix $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$, the transpose of this matrix will be $\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$

  i.e. the column 1 of matrix A, will be the row 1 of its transpose; similarly column 2 will become row 2 and so on.
- The transpose is found by just placing the a[i][j] term in b[j][i], using the same type of nested "for" loop.
- Also while displaying the transpose of the matrix i.e. the matrix "b", we need to take care that the number of rows and columns are reversed. Hence, the value of outer "for" loop i.e. "i" varies from 0 to n-1, while that of the inner loop i.e. "j" varies from 0 to m-1.

***Program* 3.2.3** : Write a program to multiply two matrices.

```
import java.util.*;
class Product
{
  public static void main(String args[])
  {
    int m,n,p,i,j,k;
    Scanner sc = new Scanner (System.in);
    System.out.print("Enter values of m, n and p:");
    m=sc.nextInt();
    n=sc.nextInt();
    p=sc.nextInt();
    int a[ ][ ]=new int [m][n];
    int b[ ][ ]=new int [n][p];
```

```
int c[ ][ ]=new int [m][p];
System.out.println("Matrix A");
for(i=0;i<=m-1;i++)
{
  for(j=0;j<=n-1;j++)
  {
   System.out.print("Enter a no:");
   a[i][j]=sc.nextInt();
  }
}
 System.out.println("Matrix B");
 for(i=0;i<=n-1;i++)
 {
  for(j=0;j<=p-1;j++)
  {
   System.out.print("Enter a no:");
   b[i][j]=sc.nextInt();
  }
}
  for(i=0;i<=m-1;i++)
  {
  for(j=0;j<=p-1;j++)
  {
    c[i][j]=0;
    for(k=0;k<=n-1;k++)
    {
     c[i][j]+=a[i][k]*b[k][j];
    }
  }
}
  System.out.println("Product Matrix");
  for(i=0;i<=m-1;i++)
  {
  for(j=0;j<=p-1;j++)
  {
    System.out.print(c[i][j]+"\t");
  }
```

```
    System.out.println();
  }
 }
}
```

Output

```
Enter values of m, n and p:2
3
3
Matrix A
Enter a no:1
Enter a no:2
Enter a no:3
Enter a no:4
Enter a no:5
Enter a no:6
Matrix B
Enter a no:1
Enter a no:2
Enter a no:3
Enter a no:4
Enter a no:5
Enter a no:6
Enter a no:7
Enter a no:8
Enter a no:9
Product Matrix
18   21   24
27   30   33
```

Explanation

- The method to accept the elements of the matrices is same as the previous program. To multiply two matrices their sizes must be m × n and n × p respectively. Hence the number of rows of the matrix 2 is kept same as the number of columns of matrix 1. The size of the resultant matrix will be m × p.
- Three level of nested "for" loops are required for multiplication. The outer two loops are used to select an element of the resultant matrix whose value is to be calculated. Initially the value is made zero. Then, the corresponding elements of matrix 1 and 2 are multiplied and added to get the terms in an element of the resultant matrix. The terms required to get an element of the resultant matrix are shown in the Fig. 3.2.4 considering the example of the matrices to be multiplied to be of sizes m × n and n × p respectively.

$$\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \\ a_{2,0} & a_{2,1} \end{bmatrix} \times \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} \\ b_{1,0} & b_{1,1} & b_{1,2} \end{bmatrix} = \begin{bmatrix} (a_{0,0} \times b_{0,0}) + (a_{0,1} \times b_{1,0}) & (a_{0,0} \times b_{0,1}) + (a_{0,1} \times b_{1,1}) & (a_{0,0} \times b_{0,2}) + (a_{0,1} \times b_{1,2}) \\ (a_{1,0} \times b_{0,0}) + (a_{1,1} \times b_{1,0}) & (a_{1,0} \times b_{0,1}) + (a_{1,1} \times b_{1,1}) & (a_{1,0} \times b_{0,2}) + (a_{1,1} \times b_{1,2}) \\ (a_{2,0} \times b_{0,0}) + (a_{2,1} \times b_{1,0}) & (a_{2,0} \times b_{0,1}) + (a_{2,1} \times b_{1,1}) & (a_{2,0} \times b_{0,2}) + (a_{2,1} \times b_{1,2}) \end{bmatrix}$$

Fig. 3.2.4 : Multiplication of matrices

- 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.

Program 3.3.2 : Write a program to capitalize the first character of each word from a sentence taken from user i.e. convert a string to title case.

```
import java.util.*;
class TitleCase
{
  public static void main(String args[])
  {
      String str,str1,str2;
      int i,n;
      Scanner sc = new Scanner (System.in);
      System.out.println("Enter a String:");
      str=sc.nextLine();
      str=str.toLowerCase();
      str1=str.substring(1);
      str=str.substring(0,1);
      str=str.toUpperCase();
      str=str.concat(str1);
      n=str.length();
     for(i=0;i<=n-1;i++)
     {
       if(str.charAt(i)==' ')
       {
         str1=str.substring(0,i+1);
         str2=str.substring(i+2);
         str=str.substring(i+1,i+2);
          str=str.toUpperCase();
         str=str1.concat(str);
         str=str.concat(str2);
       }
  }
    System.out.println(str);
  }
}
```

Output

```
Enter a String:
alL tHe beSt
All The Best
```

Explanation

- The string is simply accepted from the user and then it is converted to lower case using the method discussed earlier. The returned string is put into the same object.
- The string is then divided into two parts viz. the first character and the remaining string. The first character is then converted to upper case and then concatenated again to get the new string.
- The remaining characters are converted to upper case in the for loop. The charAt() method is used to find the blank space and then the substring() methods to separate the strings into three parts viz. the string up to space, the first character of the word and the remaining string. The second part of the string is converted to upper case and then the strings are again concatenated.
- Finally the string is displayed. This kind of string with first alphabets capital of each word is called as title case.

Program 3.3.3 : Write a program to capitalize the first character of first word from a sentence taken from user i.e. convert a string to sentence case.

```
import java.util.*;
class SentenceCase
{
   public static void main(String args[])
   {
        String str,str1,str2;
        int i,n;
        Scanner sc = new Scanner (System.in);
        System.out.println("Enter a String:");
        str=sc.nextLine();
        str=str.toLowerCase();
        str1=str.substring(1);
        str=str.substring(0,1);
        str=str.toUpperCase();
       str=str.concat(str1);
       n=str.length();
       System.out.println(str);
   }
}
```

Output

```
Enter a String:
all thE besT
All the best
```

Explanation

- The string is simply accepted from the user and then it is converted to lower case using the method discussed earlier. The returned string is put into the same object.
- The string is then divided into two parts viz. the first character and the remaining string. The first character is then converted to upper case and then concatenated again to get the new string.
- Finally the string is displayed. This kind of string with first alphabets capital of each word is called as title case.

Program 3.3.4 : Develop a Java program that determines the number of days in a given semester. Input to the program is year, month and day information of the first and the last days of a semester. The program should accept the date information as a single string instead of accepting year, month and day information separately. The input string must be in the MM/DD/YYYY format. (Assuming the semester starts and ends in the same year)

```
import java.util.*;
class Days
{
  public static void main(String args[])
  {
    String str,str1;
    int sd,sm,sy,ed,em,ey,i,total;
    int days[ ]={31,28,31,30,31,30,31,31,30,31,30,31};
    Scanner sc = new Scanner (System.in);
    System.out.println("Enter the Starting date of the semester (MM/DD/YYYY):");
    str=sc.nextLine();
    str1=str.substring(0,2);
    sm=Integer.parseInt(str1);
    str1=str.substring(3,5);
    sd=Integer.parseInt(str1);
    str1=str.substring(6,10);
    sy=Integer.parseInt(str1);
    System.out.println("Enter the ending date of the semester (MM/DD/YYYY):");
    str=sc.nextLine();
    str1=str.substring(0,2);
    em=Integer.parseInt(str1);
    str1=str.substring(3,5);
    ed=Integer.parseInt(str1);
    str1=str.substring(6,10);
    ey=Integer.parseInt(str1);
    total=ed;
    for(i=sm;i<=em-2;i++)
    {
     total+=days[i];
```

```
    }
    total+=days[sm-1]-sd+1;
    if((sy%4==0 && sy%100!=0 && sm<=2 && em>2) || (sy%100==0 && sy%400==0 && sm<=2 && em>2))
    total++;
    System.out.println("Total Days="+total);
  }
}
```

Output

```
Enter the Starting date of the semester (MM/DD/YYYY):
01/16/2012
Enter the ending date of the semester (MM/DD/YYYY):
04/20/2012
Total Days=96
```

Explanation

- The string is divided into parts to get the month, day and year separately. Each time after conversion the string is parsed into integer to get the values of starting date, month and year.
- The same procedure is repeated for the ending date of the semester.
- Then the days of the last month are directly added as the ending date is the number of days in that month for which the semester is on.
- The remaining months except for the first month, the total days are directly added from the array that stores the number of days in every month. The days in the first month are total days in that month minus the starting date plus 1. This value is then added to the total days.
- Finally if the year is a leap year and the month February is a part of the semester then the total days is incremented by 1.
- And then this value total is displayed that indicates the total number of days.

Program 3.3.5 : Write a Java program to count the number of vowels, blank spaces, digits and consonants in a string.

```
import java.util.*;
class Count
{
  public static void main(String args[])
  {
    String str;
    int countv=0,i,n,countd=0,counts=0,countc=0;
    Scanner sc = new Scanner (System.in);
    System.out.println("Enter a String:");
    str=sc.nextLine();
```

```
    n=str.length();
    char c[ ]=new char[n];
    c=str.toCharArray();
    for(i=0;i<=n-1;i++)
    {
    if(c[i]=='a'||c[i]=='e'||c[i]=='i'||c[i]=='o'||c[i]=='u'||c[i]=='A'||c[i]=='E'||c[i]=='I'||c[i]=='O'||c[i]=='U')
     countv++;
     else if(c[i]==' ')
     counts++;
     else if(c[i]>='0' && c[i]<='9')
     countd++;
     else if((c[i]>='a' && c[i]<='z') || (c[i]>='A' && c[i]<='Z'))
      countc++;
    }
    System.out.println(countv+" Vowels\n"+counts+" spaces\n"+countd+"Digits\n"+countc+" Consonants");
  }
}
```

Output

```
Enter a String:
There are 20 benches
6 Vowels
3 spaces
2 Digits
9 Consonants
```

Explanation

- The string is converted to an array of characters using the toCharArray() method.
- Each element of the array is then compared with each of the vowels in lower case and upper case.
- Every time a vowel is found the count is increased. If it is not a vowel, then it is checked for a blank space then another counter is incremented. Similarly for digits it checks in the range of 0 to 9 and for alphabets from the range from a to z, capital and small case. This is because the alphabets and digits are stored in ASCII form in sequence.
- Finally these counts are displayed.

Program 3.3.6 : Write a Java program to check if the string is palindrome or not.

```
import java.util.*;
class Palindrome
{
```

```
public static void main(String args[])
{
    String str;
    int i,n;
    Scanner sc = new Scanner (System.in);
    System.out.println("Enter a String:");
    str=sc.nextLine();
    n=str.length();
    char c[ ]=new char[n];
    char rev[ ]=new char[n];
    c=str.toCharArray();
    for(i=0;i<=n-1;i++)
    {
        rev[n-i-1]=c[i];
    }
    for(i=0;i<=n-1;i++)
    {
      if(rev[i]!=c[i])
      break;
    }
  if(n==i)
  System.out.println("Palindrome");
  else
  System.out.println("Not Palindrome");
 }
}
```

Output

```
Enter a String:
bench
Not Palindrome
```

Explanation

- The string is converted to an array of characters using the toCharArray() method.
- The char array is reversed in another array named rev. To reverse the array, the i^{th} element of array is stored in the reverse array's n-i-1 element.
- The reverse array is then compared element by element with the original array. If the comparison is found to be not equal then the control breaks out of the loop and in this case the value of 'n' and 'i' are not equal, it display "Not Palindrome". Else if the control comes out of the loop then it indicates all elements are same hence it is a "Palindrome".

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

- The method add() is declared static, so that this method can be called without making an object of this class.
- The parameters passed to this method are the values of variables 'a' and 'b'. These parameters are called as actual parameters.
- These parameters are received in the variables 'x' and 'y', by the method add(). These variables are called as formal parameters.
- The return type of the add() method is "int". Hence it has to return an "int" type data. The method returns the sum of the two numbers. This value returned by the add() method is accepted by the variable "sum" in the main() method.

Program 3.4.2 : Write a program to add two numbers using a void method.

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

Output

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

Explanation

- The method add() is declared static, so that this method can be called without making an object of this class.
- The parameters passed to this method are the values of variables 'a' and 'b'. These parameters are called as actual parameters.
- These parameters are received in the variables 'x' and 'y', by the method add(). These variables are called as formal

- The return type of the add() method is "void". Hence it doesn't return any value. The method calculates and displays the sum of the two numbers.
- Finally the value of this variable is displayed.

Program 3.4.3 : Write a program to find the factorial of a number using a method.

```
import java.util.*;
class Factorial
{
    public static void main(String args[])
    {
      int n,fact;
      Scanner sc = new Scanner (System.in);
      System.out.println("Enter a number:");
      n=sc.nextInt();
      fact=facto(n);
      System.out.println("Factorial="+fact);
    }
    static int facto (int n)
    {
     int i,fact=1;
     for(i=1;i<=n;i++)
     {
       fact=fact*i;
     }
       return fact;
    }
}
```

Output

```
Enter a number:
5
Factorial=120
```

Explanation

- The method facto() is declared static, so that this method can be called without making an object of this class.
- The number whose factorial is to be found is passed as a parameter to this method.
- The facto() method calculates the factorial and returns the result.
- The result returned by the facto() method is accepted in the variable "fact" of the main() method.
- Finally the value of this variable fact is displayed.

Program 3.4.4 : Write a program to find the values of ${}^{n}C_{r}$ and ${}^{n}P_{r}$

```
import java.util.*;
class Factorial
{
    public static void main(String args[])
    {
     int n,r,nCr,nPr;
     Scanner sc = new Scanner (System.in);
     System.out.println("Enter the values of n and r to find nCr:");
     n=sc.nextInt();
     r=sc.nextInt();
     nCr=facto(n)/(facto(r)*facto(n-r));
     System.out.println("nCr="+nCr);
     System.out.println("Enter the values of n and r to find nPr:");
     n=sc.nextInt();
     r=sc.nextInt();
    nPr=facto(n)/facto(n-r);
   System.out.println("nPr="+nPr);
  }
   static int facto (int n)
  {
    int i,fact=1;
   for(i=1;i<=n;i++)
   {
    fact=fact*i;
   }
  return fact;
 }
}
```

Output

```
Enter the values of n and r to find nCr:
4
2
nCr=6
Enter the values of n and r to find nPr:
4
2
nPr=12
```

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.

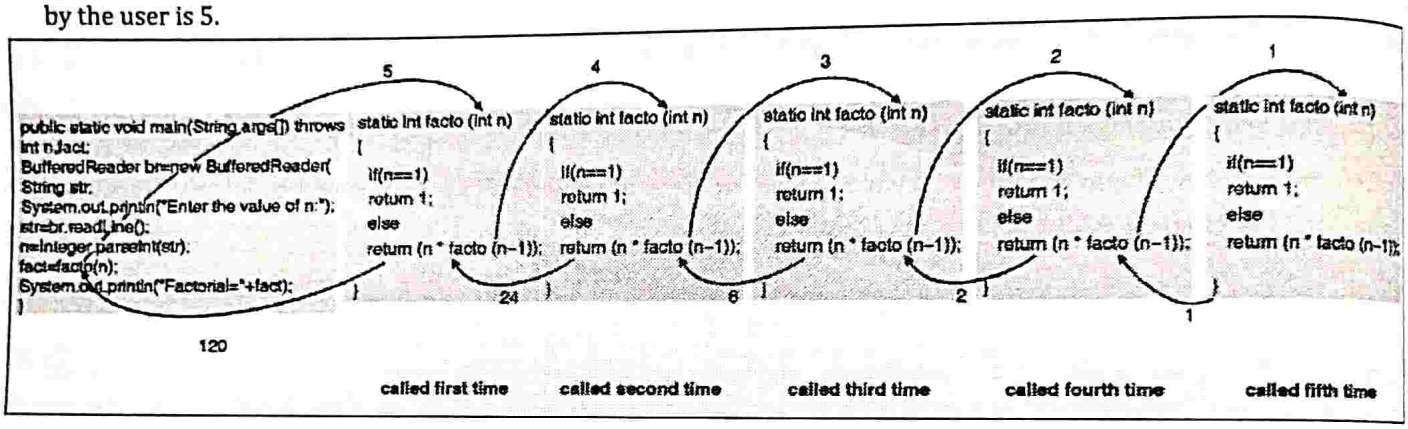


Fig. 3.5.1

- 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.

- The value 6 is received by the previous case called method and multiplied by the value of the variable "n" in this case i.e. 6 multiplied by 4, returning the result i.e. 24. This value is received by the first called method.
- This method again calculates the multiplication of the received value with the value of the variable "n" in this case i.e. 24 multiplied by 5, returning the final result i.e. 120, to the "main" method. The "main" method displays this result.

Note :

1. The value returned and received in each case is shown in the Fig. 3.5.1.
2. For understanding the concept, in this figure we have shown the method named "facto" for five times. Actually, the method is only once. It is written only once in the program and also it is stored only once in the memory. But it is executed five times in this case, hence shown for five times.
3. The variables "n" are different for the method every time the same method is called again i.e. different memory spaces are allocated to these variables every time the method is called.

Program 3.5.2 : Write a recursion method to compute the sum of first 'n' number beginning with 1.

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

Output

```
Enter the value of n:
5
Sum=15
```

Explanation

- The logic is same as the program for factorial using recursive method except that instead of multiplying we have to add.
- For example 5! is 1*2*3*4*5; while for this program we have to perform 1+2+3+4+5.

Program 3.5.3 : 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".

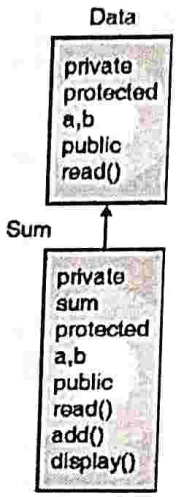


Fig. P. 3.7.1 : Class Diagram of Single Inheritance

```
import java.util.*;
class Data
{
    protected int a, b;
    public void read(int x, int y)
    {
            a=x;
            b=y;
    }
}
class Sum extends Data
{
   private int sum;
   public void add()
   {
     sum=a+b;
   }
   public void display()
   {
    System.out.println("Sum="+sum);
   }
}

class Main
{
  public static void main (String args[ ])
  {
   int x,y;
   Scanner sc = new Scanner (System.in);
   System.out.println("Enter two numbers");
   x =sc.nextInt();
   y=sc.nextInt();
```

```
Sum s=new Sum();
s.read(x,y);
s.add();
s.display();
}
}
```

Output

```
Enter two numbers
3
4
Sum=7
```

Explanation

- As shown in the class diagram, the base class data has two protected member variables namely 'a' and 'b'. These members are kept protected, so that they are accessible from the derived class method to calculate the sum.
- The base class also has a method in public visibility to read the inputs from user. The derived class also has public member methods add() and display() to find the sum of the two numbers taken and to display this sum respectively.
- The class "Sum" is derived from the class "Data". This derivation is implemented using the already discussed syntax in the statement:

  class Sum extends Data
- The members of the base class are shown available in the derived class in the class diagram in Fig. P. 3.7.1 but these member methods and member variables are not to be defined again in the derived class when actually writing the program. This concept of gaining access to the members of the base class is called as **code reusability.**
- The main() method has first a creation of the object of the class "Sum". The object of the class sum can access all the methods i.e. read(), add() and display(); because the derived class derives the protected members in the same visibility when derived publicly as discussed in Fig. 3.7.1. The methods read(), add() and display() are thereafter called one by one.

Program 3.7.2 : Write a program to find the area of circle using single inheritance such that the base class method must accept the radius from the user and the derived class method must calculate and display the area.

Solution : The class diagram of this requirement is as shown in Fig. P. 3.7.2. This diagram shows that the class "Area" is derived from class "Data".

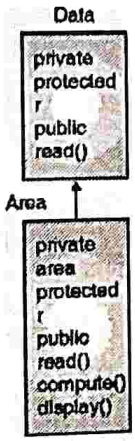


Fig. P. 3.7.2 : Class Diagram of Single Inheritance for Program 3.7.2

```
import java.util.*;
class Data
{
    protected float r;
    public void read(float x)
    {
      r=x;
    }
}
class Area extends Data
{
   private float area;
   public void calculate()
   {
       area=3.14f*r*r;
   }
  public void display()
  {
     System.out.println("Area="+area);
  }
}
class Main
{
    public static void main (String args[ ])
    {
     float x;
     Scanner sc = new Scanner (System.in);
     System.out.println("Enter the radius:");
     x=sc.nextFloat();
    Area a=new Area();
   a.read(x);
   a.calculate();
   a.display();
 }
}
```

Output

```
Enter the radius:
10
Area=314.0
```

Explanation

A program with similar logic just that instead of calculating the sum we have to calculate the area of a circle.

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.

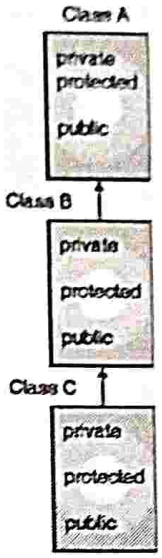


Fig. 3.8.1 : Multi level inheritance

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.

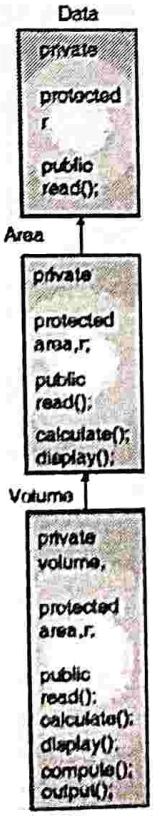


Fig. P. 3.8.1 : Class Diagram of Multi Level Inheritance for Program 3.8.1

```
import java.util.*;
class Data
{
   protected float r;
   public void read(float x)
   {
    r=x;
   }
}
class Area extends Data
{
   protected float area;
   public void calculate()
   {
    area=3.14f*r*r;
   }
  public void display()
  {
   System.out.println("Area="+area);
  }
}
class Volume extends Area
{
   private float volume;
    public void compute()
   {
      volume=area*r*4/3;
   }
   public void output()
  {
    System.out.println("Volume="+volume);
  }
}
class Main
{
   public static void main (String args[ ])
   {
      float x;
      Scanner sc = new Scanner (System.in);
      System.out.println("Enter the radius:");
```

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

Output

```
Enter the radius:
10
Area=314.0
Volume=4186.6665
```

Explanation

- The implementation of the class diagram shown in Fig. P. 3.8.1 is done in similar manner as for the previous program.

Program 3.8.2 : Consider a class network given. The class 'Admin' derives information from the class 'Account' which in turn derives information from 'Person' class. Write a program to display Admin object.

Fig. P. 3.8.2 : Class Diagram of Multi Level Inheritance for Program 3.8.2

Solution :

```
import java.util.*;
class Person
{
  protected int code;
  String name;
}
class Account extends Person
{
  protected float pay;
}
class Admin extends Account
{
```

```
  private int exp;
  public void accept(String s, int c, float p, int e)
  {
    name=s;
    code=c;
    pay=p;
    exp=e;
  }
  public void display()
  {
   System.out.println("Name:"+name+"\nCode:"+code+"\nPay:"+pay+"\nExp:"+exp);
  }
}
class Main
{
  public static void main (String args[ ])
  {
     float pay;
     int c,e;
    String str,str1;
    Scanner sc = new Scanner (System.in);
    System.out.println("Enter name, code, pay and exp:");
    str1=sc.nextLine();
    c=sc.nextInt();
   pay=sc.nextFloat();
   e=sc.nextInt();
    Admin a=new Admin();
    a.accept(str1,c,pay,e);
   a.display();
  }
}
```

Output

```
Enter name, code, pay and exp:
Ajay
325
50000
10
Name:Ajay
Code:325
Pay:50000.0
Exp:10
```

Explanation

The implementation of the class diagram shown in Fig. P. 3.8.2 is done in similar manner as for the previous program.

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.

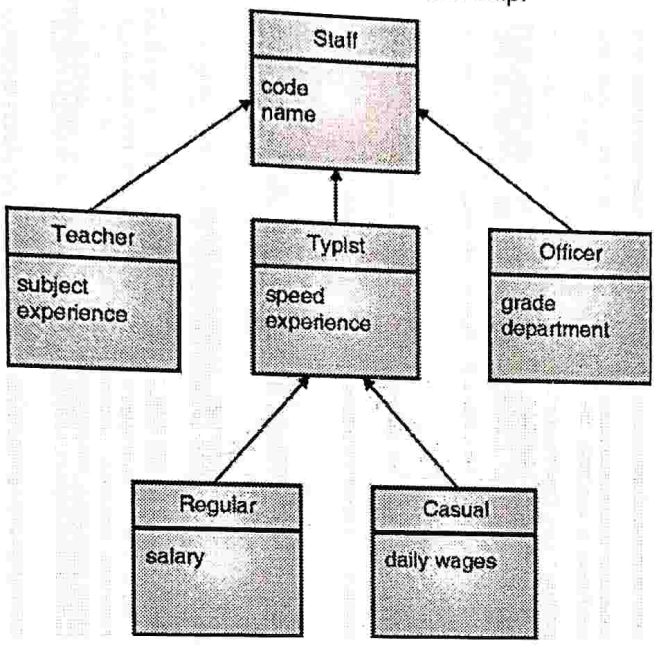


Fig. P. 3.9.1 : Class Diagram of Hierarchical Inheritance for Program 3.9.1

```
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();
```

```
                experience=sc.nextInt();
        }
        public void display()
        {
    System.out.println("TeacherDetails:\nName:"+name+"\nCode:"+code+"\nSubject:"+subject+"\nExperience:"+experience);
        }
}
class Officer extends Staff
{
        private String dept;
        private int grade;
        public  void read()
        {
                Scanner sc= new Scanner(System.in);
                System.out.println("Enter name, code, department and grade of the officer:");
                name =sc.next();
                code =sc.nextInt();
                dept =sc.next();
                grade =sc.nextInt();
        }
        public void display()
        {
                System.out.println("Officer Details:\nName:"+name+"\nCode:"+code+
        "\nDepartment:"+dept+"\nGrade:"+grade);
        }
}
class Typist extends Staff
{
        protected int speed,experience;

}
class Regular extends Typist
{
        private float salary;
        public void read()
        {
                Scanner sc = new Scanner (System.in);
                System.out.println("Enter name, code, speed, experience and salary of the regular typist:");
                name=sc.next();
                code=sc.nextInt();
```

```
            speed=sc.nextInt();
            experience=sc.nextInt();
            salary=sc.nextFloat();
        }
        public void display()
        {
        System.out.println("Regular Typist
Details:\nName:"+name+"\nCode:"+code+"\nSpeed:"+speed+"\nExperience:"+experience+"\nSalary:"+salary);
    }
}
class Casual extends Typist
{
        private float dailywages;
        public void read()
        {
            Scanner sc = new Scanner (System.in);
            System.out.println("Enter name, code, speed, experience and daily wages of the Casual typist:");
            name=sc.next();
            code=sc.nextInt();
            speed=sc.nextInt();
            experience=sc.nextInt();
            dailywages=sc.nextFloat();
        }
        public void display()
        {
            System.out.println("Casual Typist Details:\nName:"+name+"\nCode:"+code+"
        \nSpeed:"+speed+" \nExperience:"+experience+"\nDaily Wages:"+dailywages);
        }
}
class Main
{
        public static void main(String args[])
        {
            Scanner sc = new Scanner (System.in);
            int choice;
            System.out.println("1. Teacher\n2. Officer\n3. Regular Typist\n4. Casual Typist\nEnter the choice, whose
            details you want to enter:");
            choice=sc.nextInt();
            switch(choice)
            {
                case 1:Teacher t=new Teacher();
                t.read();
                t.display();
                break;
                case 2:Officer o=new Officer();
```

```
                    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.

Program 3.10.1 : Write a program to calculate volume of sphere using multi level inheritance demonstrating method overriding. 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 and display the area of a circle and another class derived from this will have methods to calculate and display the volume of the sphere.

Solution :

```
import java.util.*;
class Data
{
   protected float r;
   public void read(float x)
   {
      r=x;
   }
}
class Area extends Data
{
   protected float area;
   public void calculate()
   {
     area=3.14f*r*r;
   }
   public void display()
   {
     System.out.println("Area="+area);
   }
}
class Volume extends Area
{
   private float volume;
   public void compute()
   {
    volume=area*r*4/3;
   }
   public void display()
   {
     System.out.println("Volume="+volume);
   }
}
class Main
{
   public static void main (String args[ ])
   {
     float x;
     Scanner sc = new Scanner (System.in);
     System.out.println("Enter the radius:");
```

```
        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);
  }
```

```
}
class Sphere extends Base
{
  public void calculate()
  {
      vol=3.14f*r*r*r*4/3;
  }
}
class Hemisphere extends Base
{
   public void calculate()
   {
      vol=3.14f*r*r*r*2/3;
   }
}
class Main
{
  public static void main (String args[ ])
  {
     float x;
     Scanner sc = new Scanner (System.in);
     System.out.println("Enter the radius:");
     x=sc.nextFloat();
     Sphere s=new Sphere();
     s.read(x);
     s.calculate();
     System.out.println("Sphere:");
     s.display();
     Hemisphere h=new Hemisphere();
     h.read(x);
     h.calculate();
     System.out.println("Hemisphere:");
     h.display();
  }
}
```

Output

```
Enter the radius:
10
Sphere:
Volume=4186.6665
Hemisphere:
Volume=2093.3333
```

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;
   }
```

```
}
class Hemisphere extends Base
{
  public void calculate()
  {
    vol=3.14f*r*r*r*2/3;
  }
}
class Main
{
    public static void main (String args[ ])
    {
      float x;
      Scanner sc = new Scanner (System.in);
      System.out.println("Enter the radius:");
      x=sc.nextFloat();
      Sphere s=new Sphere();
      s.read(x);
      s.calculate();
      System.out.println("Sphere:");
      s.display();
      Hemisphere h=new Hemisphere();
      h.read(x);
      h.calculate();
      System.out.println("Hemisphere:");
      h.display();
    }
}
```

Output

```
Enter the radius:
10
Sphere:
Volume=4186.6665
Hemisphere:
Volume=2093.3333
```

Explanation

- The method calculate() is declared to be "abstract" in the base class. This makes it compulsory for the derived class to override this method. Also it makes the base class to be compulsorily be declared as "abstract"
- The implementation Program 3.12.2 is done with method calculate() in base abstract class and in sub classes Sphere and Hemisphere. The derived classes have the definition of the method calculate(), which is an abstract method of the base class.

Program 3.12.2 : Write a abstract class program to calculate area of circle, rectangle and triangle.

Solution :

```
import java.util.*;
abstract class Base
{
   protected float r,l,b,area;
   public abstract void calculate();
   public void display()
   {
     System.out.println("Area="+area);
   }
}
class Circle extends Base
{
   public void read(float x)
   {
     r=x;
   }
  public void calculate()
  {
   area=3.14f*r*r;
  }
}
 class Rectangle extends Base
 {
   public void read(float x, float y)
   {
     l=x;
     b=y;
   }
  public void calculate()
  {
   area=l*b;
  }
}
 class Triangle extends Base
 {
  public void read(float x, float y)
  {
    l=x;
    b=y;
  }
   public void calculate()
   {
   area=0.5f*l*b;
  }
```

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

Output

```
Circle:
Enter the radius:
10
Area=314.0
Rectangle:
Enter length and breadth:
10
10
Area=100.0
Triangle:
Enter height and breadth:
10
10
Area=50.0
```

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.

- When one parameter is passed in the first println() method, the area of circle is calculated and returned by the first method.
- In the second case the area of rectangle is printed as two parameters are passed.
- In the third case, area of triangle is calculated and printed as three parameters are passed.

Program 3.14.3 : Write a program to demonstrate Method overloading by overloading the methods for calculating volume of cylinder, cube and cuboid.

Solution :

```
class Volume
{
        private int x;;
        public float volume(float l)
        {
                return(l*l*l);
        }
        public float volume(float r, float h)
        {
                return (3.14f*r*r*h);
        }
        public float volume(float l, float b, float h)
        {
                return(l*b*h);
        }
}
class Main
{
        public static void main(String args[])
        {
                Volume a=new Volume();
                System.out.println("Volume of cube="+a.volume(10));
                System.out.println("Volume of cylinder="+a.volume(10,10));
                System.out.println("Volume of cuboid="+a.volume(10,10,10));
        }
}
```

Output

```
Volume of cube=1000.0
Volume of cylinder=3140.0
Volume of cuboid=1000.0
```

Explanation

- The class Volume has three methods with the same name i.e. volume(). The methods have different number of parameters. This is called as method overloading.
- When one parameter is passed in the first println() method, the volume of cube is calculated and returned by the first method.
- In the second case the volume of cylinder is printed as two parameters are passed.
- In the third case, volume of cuboid is calculated and printed as three parameters are passed.

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.

- In this case the overridden method to be called depends on the object to which the reference object is referring at that moment.
- This can be realized only during the run time that the reference object is pointing to which class object.
- Hence the method to be called depends during the run time. Hence it is called as Dynamic dispatch of methods. Let us see this concept with a program example.

Note : One thing that should be clear over here is that Dynamic method dispatch is possible only for overridden methods.

Program 3.15.1 : Write a program to demonstrate Dynamic Method dispatch.

Solution :

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

Output

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

Explanation

- The class Parent has a method display() with one parameter. The Child extended from the class Parent has the same method display() also with one parameter.
- In the main() method one object of each the Parent and Child class are made. A reference object of the Parent class is made. Initially the reference object is given the Child class object and the display() method is called. Hence the Child class display() method is called.
- The second time reference object is given the object of the base class. Hence in this case the display() method of the Parent class is called.

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);
        }
}
```

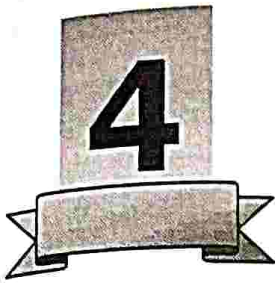
Output

```
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
{
```

```
protected float r,vol;
public void read(float x)
{
  r=x;
}
public void calculate()
{
  vol=3.14f*r*r*r*2/3;
 }
public void display()
{
  System.out.println("Volume="+vol);
}
}
class Main
{
 public static void main (String args[ ])
 {
    float x;
    Scanner sc = new Scanner (System.in);
    System.out.println("Enter the radius:");
    x=sc.nextFloat();
   Sphere s=new Sphere();
   s.read(x);
   s.calculate();
   System.out.println("Sphere:");
   s.display();
   Hemisphere h=new Hemisphere();
   h.read(x);
   h.calculate();
   System.out.println("Hemisphere:");
   h.display();
 }
}
```

Output

```
Enter the radius:
10
Sphere:
Volume=4186.6665
Hemisphere:
Volume=2093.3333
```

Explanation

- Here you will notice that the interface "Base" is made with the declaration of methods read(), calculate() and display(). The classes "Hemisphere" and "Sphere", are derived from the interface using the keyword "implements". These implemented classes have overridden the methods declared in the interface, with proper definition of the methods.
- Thus is the use of the "interface" for making templates for the programming.

Program 4.1.2 : Write a program to display area of square and rectangle. Make use of interface to define templates of methods to be there in the derive classes.

```
import sjava.util.*;
interface Base
{
    public void calculate();
    public void display();
}
class Square implements Base
{
   protected float side, area;
   public void read(float x)
   {
     side=x;
   }
   public void calculate()
  {
   area=side*side;
  }
  public void display()
  {
   System.out.println("Area="+area);
  }
}
class Rectangle implements Base
{
  protected float l,b,area;
  public void read(float x, float y)
  {
   l=x;
   b=y;
  }
  public void calculate()
 {
  area=l*b;
 }
 public void display()
{
  System.out.println("Area="+area);
}
```

```
}
class Main
{
public static void main (String args[ ])
{
    float x,y;
    Scanner sc = new Scanner (System.in);
    System.out.println("Enter the length and breadth:");
    x=sc.nextFloat();
    y=sc.nextFloat();
    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

Program 4.4.1 : Write a program to create a sub-package "india" inside the package "world" and show the execution of the same.

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

Output

```
C:\java programs>cd prog 4.4.1
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

Similar program with a slight change in the method of storing the program and executing it as discussed in the above output.

Program 4.4.2 : Write a program to create a package having an interface and show the execution of the same.

```
package world.india;
interface Greetings
{
        void display();
}
class Hello implements Greetings
{
        public static void main(String args[])
        {
                Hello h=new Hello();
                h.display();
        }
        public void display()
        {
                System.out.println("Hello Indians");
        }
}
```

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);
        }
}
```

Output

```
x=5
```

Explanation

- You will notice that we haven't displayed the value of "x" anywhere. But it is seen in the output.
- When we try to display the object of the class "Demo", as discussed earlier, the toString() method of that class is automatically called.
- This method has a return type of "String", hence we return the concatenated string to display the value of x.

Program 4.6.2 : Create rectangle and cube class that encapsulates the properties of a rectangle and cube i.e. rectangle has default and parameterised constructor and area ( ) method. Cube has default and parameterised constructor and volume ( ) method. They share no ancestor other than object. Implement a class size with size() method. This method accepts a single reference argument z. If z refers to a rectangle then size (z) returns its area and if z is a reference to a cube, then size (z) returns its volume. If z refers to an object of any other class, then size (z) returns - 1. Use main ( ) method in size class to call size (..) method.

```
class Rectangle
{
        private int l,b,area;
        public Rectangle()
        {
                l=5;
                b=10;
        }
        public Rectangle(int x, int y)
        {
                l=x;
                b=y;
        }
        public void Area()
        {
                area=l*b;
        }
        public String toString()
        {
                return("Area="+area+"\n");
        }
}
class Cube
{
        private int l,vol;
        public Cube()
        {
                l=5;
        }
        public Cube(int x)
        {
                l=x;
        }
        public void volume()
        {
                vol=l*l*l;
```

```
    }
    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.

```
import java.util.*;
class Student
{
        private int id;
        String name;
        public Student(int x, String y)
        {
                id=x;
                name=y;
        }
        public String toString()
        {
                return("Principal for a day is "+name+" with id "+id);
        }
}
class StudentLottery
{
        Student s[ ]=new Student[100];
        int n=0;
        public void addStudents()
        {
                int x;
                String y;
                Scanner sc=new Scanner(System.in);
                System.out.print("Enter name and id:");
                y=sc.next();
                x=sc.nextInt();
                s[n]=new Student(x,y);
                n++;
        }
        public void pickWinner()
        {
                int x;
                Random rno=new Random();
                x=rno.nextInt(5000);
                System.out.print(s[x%n]);
        }
        public static void main(String args[ ])
        {
```

```
        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.

❑❑❑

Note

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
```

```
{
 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:
4
0
Exception in thread "main" java.lang.ArithmeticException: / by zeroat Divide.main(Divide.java:12)
```

Explanation

- The divisor entered is zero. Hence an error called as ArithmeticException is generated.
- This is not known during the compiling of the program, that user will enter the divisor as zero, hence it is an unchecked exception.
- Similarly, in this case if the number entered was a character, we will get NumberFormatException, i.e. not a proper number is entered.
- Let us see the output in this case.

Output 2

```
Enter two numbers:
a
Exception in thread "main" java.lang.NumberFormatException: For input string: "a"
    at java.lang.NumberFormatException.forInputString(Unknown Source)
    at java.lang.Integer.parseInt(Unknown Source)
    at java.lang.Integer.parseInt(Unknown Source)
    at Divide.main(Divide.java:9)
```

- If the numbers are entered properly then you will not get any error. You will get a proper output as shown in Output 3 below.

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 understood by an end user. You can make a proper statement of what you want to be displayed as an error.

- You must have noted that the second statement in the try block is not executed, no quotient is displayed. This is as discussed earlier, whenever one statement in the try block generates an exception the remaining statements after that statement in the try block are not executed.
- Let us see how the execution takes place if no exception is generated.

Output 2

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

- In case the data is given properly, the quotient is displayed. Let us also see how will this program behave when a character is given instead of a number

Output 3

```
Enter two numbers:
2
a
Exception in thread "main" java.lang.NumberFormatException: For input string: "a"
    at java.lang.NumberFormatException.forInputString(Unknown Source)
    at java.lang.Integer.parseInt(Unknown Source)
    at java.lang.Integer.parseInt(Unknown Source)
    at Divide.main(Divide.java:11)
```

- We have caught the arithmetic exception in the catch block. We haven't caught the number format exception, hence the exception will again show something not understandable to end user. Let us see in the next program how to handle this exception.

Program 5.3.2 : Write a program to perform division of two numbers accepted from user. Handle the Number format exception using the try-catch block. Also handle the divide by zero exception using 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);
  }
  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.

```
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);
     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)
   {
     System.out.println("Invalid number");
   }
  }
}
```

Output 1

```
Enter two numbers:
5
0
Exception has occurred. You have entered the divisor as zero
```

Output 2

```
Enter two numbers:
4
a
Invalid number
```

Output 3

```
Enter two numbers:
5
3
The Quotient=1
```

- The first output shows only divide by zero exception as the divisor entered was zero.
- The second output shows only invalid number as there was a number format exception
- The third output shows the correct data entered and hence no exception

Program 5.3.4 : Write a program to perform division of two numbers accepted from user. Handle the IO Exception, Number format exception and also handle the divide by zero exception using multiple try catch block.

```
import java.io.*;
class Divide
{
   public static void main(String args[])
   {
      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);
         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(IOException ioe)
      {
         System.out.println("IOException occurred");
      }
      catch (NumberFormatException ne)
      {
         System.out.println("Invalid number");
      }
   }
}
```

Output

```
Enter two numbers:
5
3
The Quotient=1
```

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)
      {
```

```
        System.out.println("Invalid number");
      }
    }
}
```

Output 1

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

Output 2

```
Enter two numbers:
4
a
Invalid number
```

Output 3

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

- The first output shows only divide by zero exception as the divisor entered was zero.
- The second output shows invalid number as there was a number format exception
- The third output shows the correct data entered and hence no exception

Now let us also see a program example of using try catch as well as finally block in a program.

Program 5.3.6 : Write a program to perform division of two numbers accepted from user. Handle the divide by zero exception using the try-catch-finally 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");
    }
  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.

```
import java.io.*;
class Month
{
 public static void main(String args[ ]) throws IOException
 {
  int m;
  BufferedReader br = new BufferedReader (new InputStreamReader (System.in));
  String str;
  System.out.println("Enter month number:");
  str=br.readLine();
  m=Integer.parseInt(str);
  try
  {
   if(m>12 || m<1)
   throw new NumberFormatException();
   System.out.println("Month number entered is "+m);
  }
  catch(NumberFormatException ne)
  {
   System.out.println("Invalid month number");
  }
 }
}
```

Output 1

```
Enter month number:
15
Invalid month number
```

Explanation

- You will notice in this case we check a condition i.e. is the month number greater than 12 or less than 1. In such case we throw an exception. How do we throw this exception? We make a new object of the exception class for which we want to generate the exception and throw that object. The object is created using the "new" operator.
- The object is caught in the catch block.
- 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:
3
Month number entered is 3
```

- This can also be done by making a new exception class say for example MonthNumberException. We will see this concept in the next program.

Program 5.5.2 : Write a program to accept and display the month number. Throw an exception if improper month number is entered. Make your own exception class to handle this exception.

```
import java.io.*;
class MonthNumberException extends Exception
{
  public MonthNumberException(String str)
  {
    System.out.println(str);
  }
}
class Month
{
  public static void main(String args[ ]) throws IOException
  {
    int m;
    BufferedReader br = new BufferedReader (new InputStreamReader (System.in));
    String str;
    System.out.println("Enter month number:");
    str=br.readLine();
    m=Integer.parseInt(str);
    try
    {
      if(m>12 || m<1)
      throw new MonthNumberException("Invalid Month Number");
      System.out.println("Month number entered is "+m);
    }
    catch(MonthNumberException me)
    {
    }
  }
}
```

Output 1

```
Enter month number:
13
Invalid Month Number
```

Explanation

- We have made our own class named as MonthNumberException. This class must extend i.e. derived from the class Exception.

- 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.

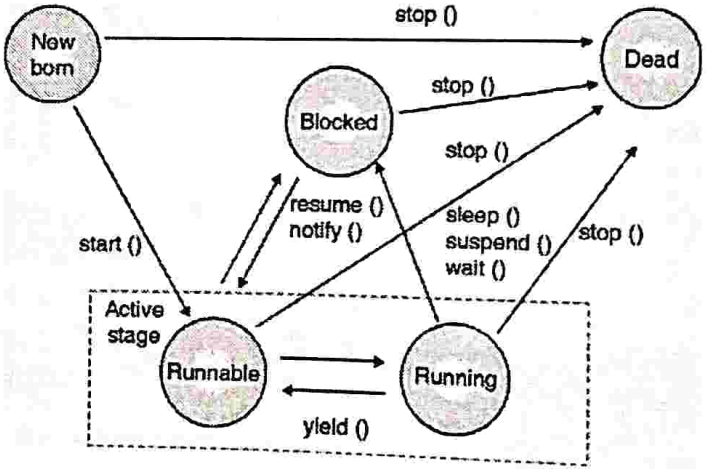


Fig. 5.8.1

- 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);
```

```
                }
            }
}
class EvenNumbers extends Thread
{
        public void run( )
        {
                int i;
                for(i=2;i<=10;i+=2)
                {
                        System.out.println(i);
                }
        }
}
class Main
{
        public static void main(String args[])
        {
                OddNumbers n=new OddNumbers( );
                Thread t1=new Thread(n);
                EvenNumbers n1=new EvenNumbers( );
                Thread t2=new Thread(n1);
                t1.start( );
                t2.start( );
        }
}
```

Output

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

Explanation

We have made two classes in this case and similar procedure is followed for each thread as discussed in the previous sections.

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)
                        {
```

```
                }
            }
        }
}
class Numbers extends Thread
{
        public void run( )
        {
            int i;
            for(i=1;i<=5;i++)
            {
                System.out.println(i);
                try
                {
                    Thread.sleep(100);
                }
                catch(Exception e)
                {
                }
            }
        }
}
class Main
{
        public static void main(String args[])
        {
            Alphabets n=new Alphabets( );
            Thread t1=new Thread(n);
            Numbers n1=new Numbers( );
            Thread t2=new Thread(n1);
            t1.start( );
            t2.start( );
            System.out.println("Thread t1 is alive:"+t1.isAlive( ));
            System.out.println("Thread t2 is alive:"+t2.isAlive( ));
            try
            {
                t1.join( );
```

```
                t2.join( );
            }
            catch(InterruptedException ie)
            {
            }
            System.out.println("Thread t1 is alive:"+t1.isAlive( ));
            System.out.println("Thread t2 is alive:"+t2.isAlive( ));
        }
}
```

Output

```
A
1
Thread t1 is alive:true
Thread t2 is alive:true
2
B
3
C
D
4
5
E
Thread t1 is alive:false
Thread t2 is alive:false
```

Explanation

- In this case we are printing the alphabets from 'A' to 'E' and numbers from 1 to 5 using two child threads i.e. the derived threads of class "Thread".
- Initially when the threads are made and started, their status of alive or dead is displayed using the is Alive() method. The **join( )** method throws an exception called as Interrupted Exception.
- Hence the statement is put in the try block and the catch block is put according to the syntax.
- You will notice that when the threads are started, one value in each thread is displayed i.e. '1' and 'A'. Then the status is displayed where the main( ) method has its time slice, hence it shows that the threads are alive.
- After that the join( ) method is executed, for both the threads. The main( ) method control doesn't proceeds until the two threads complete their execution. Again the status is displayed.
- Now the status shows that the threads are not alive.
- If the thread t2 was started after the t1.join( ) method, then the thread t2 will start after the completion of thread t1.

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.

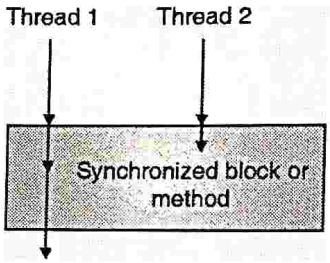


Fig. 5.11.1: Working of Synchronized block or method

- 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.

Program 5.11.1 : Write a program that has two threads. Each of the threads displays the numbers from 1 to 10. Use the sleep( ) method to display the numbers sequentially. Make the synchronized method.

```
class Numbers extends Thread
{
        public synchronized void run( )
        {
                int i;
                for(i=1;i<=10;i++)
                {
                        System.out.println(i);
                        try
                        {
                                Thread.sleep(100);
                        }
                        catch(Exception e)
                        {
                        }
                }
        }
}
class Main
{
        public static void main(String args[])
        {
                Numbers n=new Numbers( );
                Thread t1=new Thread(n);
                Thread t2=new Thread(n);
                t1.start( );
                t2.start( );
        }
}
```

Output

```
1
2
3
4
5
6
```

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

Explanation

- In this case you will notice, since the keyword "synchronized" is written before the method run( ), there is a change in the output compared to a similar program but without synchronized method i.e. Program 5.9.7. Here the entire method run( ) is first executed for the first thread i.e. t1 hence the output from 1 to 10. Then the same run( ) method is executed for the thread t2, hence again the output from 1 to 10.

Program 5.11.2 : Write a program to book the last available ticket of a class Movie. Create two threads of the object of this class, created at two counters and book the ticket for one of the customer, while display "House Full" for the other customer.

```
class Movie extends Thread
{
        int vacant=1,required;
        Movie(int x)
        {
                required=x;
        }
        public synchronized void run( )
        {
                if(vacant>=required)
                {
                        System.out.println(required+" tickets booked for"+Thread .currentThread( ).getName( ));
                        try
                        {
                                Thread.sleep(100);
                        }
                        catch(Exception e)
```

```
                    {
                    }
                    vacant-=required;
                }
                else
                {
                    System.out.println("House Full "+Thread.currentThread( ).getName( ));
                }
        }
}
class Main
{
        public static void main(String args[])
        {
            Movie n=new Movie(1);
            Thread t1=new Thread(n);
            Thread t2=new Thread(n);
            t1.setName("Ajay");
            t2.setName("Vijay");
            t1.start( );
            t2.start( );
        }
}
```

Output

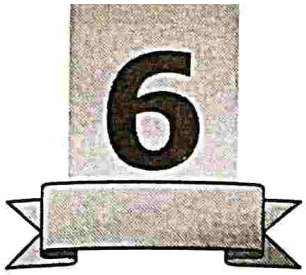
```
1 tickets booked for Ajay
House Full Vijay
```

Explanation

- There are two variables declared i.e. vacant seats and required seats. The variable vacant seats are initialized to 1, as specified only one seat is available.
- The variable required is initialized to 1 by the use of a constructor.
- In the run( ) method, a check is performed to find whether the seats are available, then the thread is made to go to sleep for printing the ticket.
- Finally once the ticket is printed, the vacant seats status is updated. Before completing this entire task, another thread to book the seat must not enter this block of statements.
- This is taken care by using the synchronized method, but it can also be implemented using the synchronized block.

❑❑❑

Note



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.

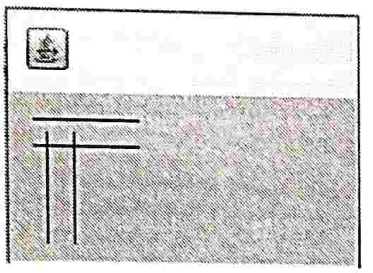
```
import javax.swing.JFrame;
import java.awt.Graphics;
import javax.swing.JPanel;
class MyPanel extends JPanel
{
        public void paint(Graphics g)
        {
                g.drawLine(10,10,50,10);
                g.drawLine(10,20,50,20);
                g.drawLine(15,15,15,55);
```

```
            g.drawLine(25,15,25,55);
        }


}
public class Graphi1 extends JFrame
{
        Graphi1()
        {
                setSize(150,100);
                MyPanel mp=new MyPanel();
                add(mp);
        }

        public static void main(String args[])
        {
                new Graphi1().setVisible(true);
        }
}
```

Output



Explanation

- The use of drawLine() method of the class "Graphics" is used in this program. The concept of co-ordinates is very important here.
- The Fig. 6.2.1 below shows the co-ordinate system used in methods of class Graphics.

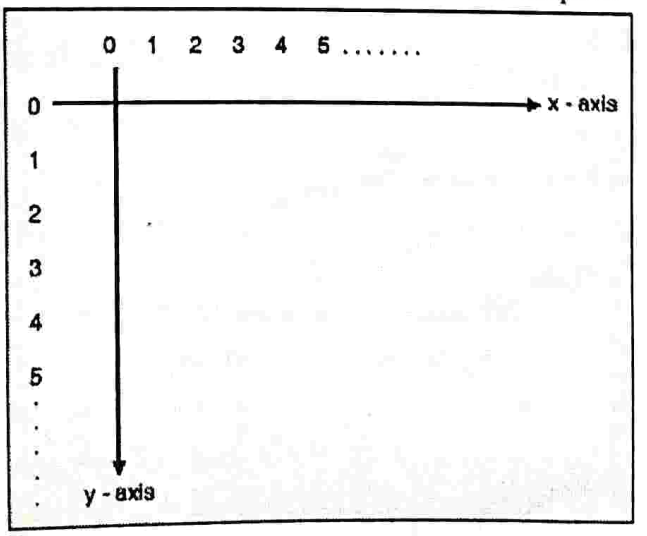


Fig. 6.2.1 : Co-ordinate system followed by the graphics class methods

- 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.

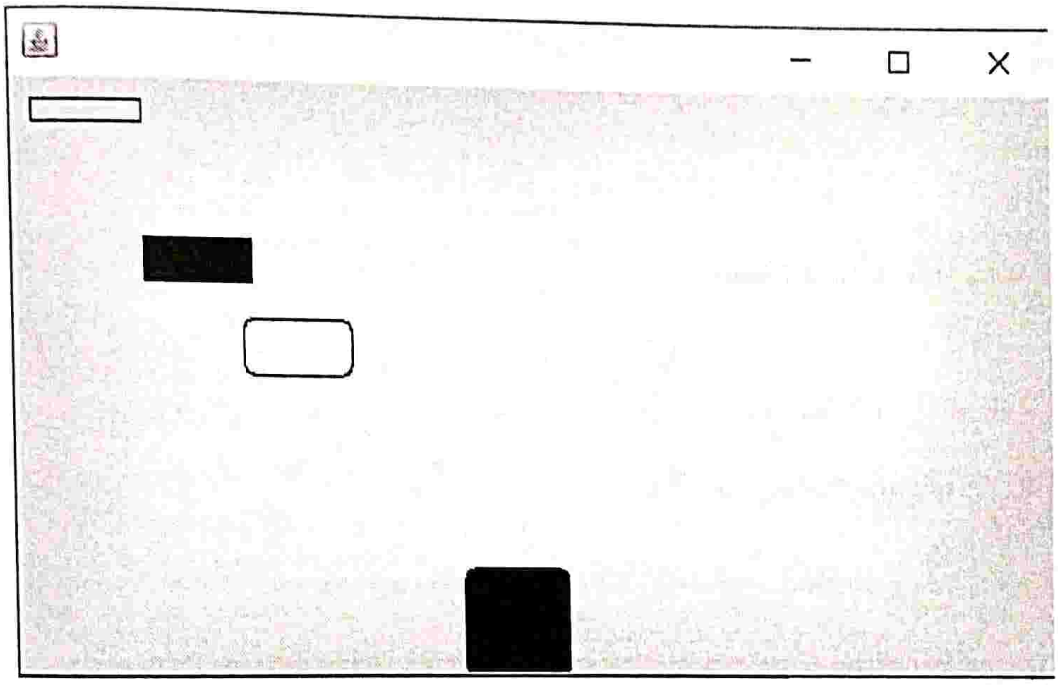
Let us see a Program 6.2.2 to draw some rectangles.

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

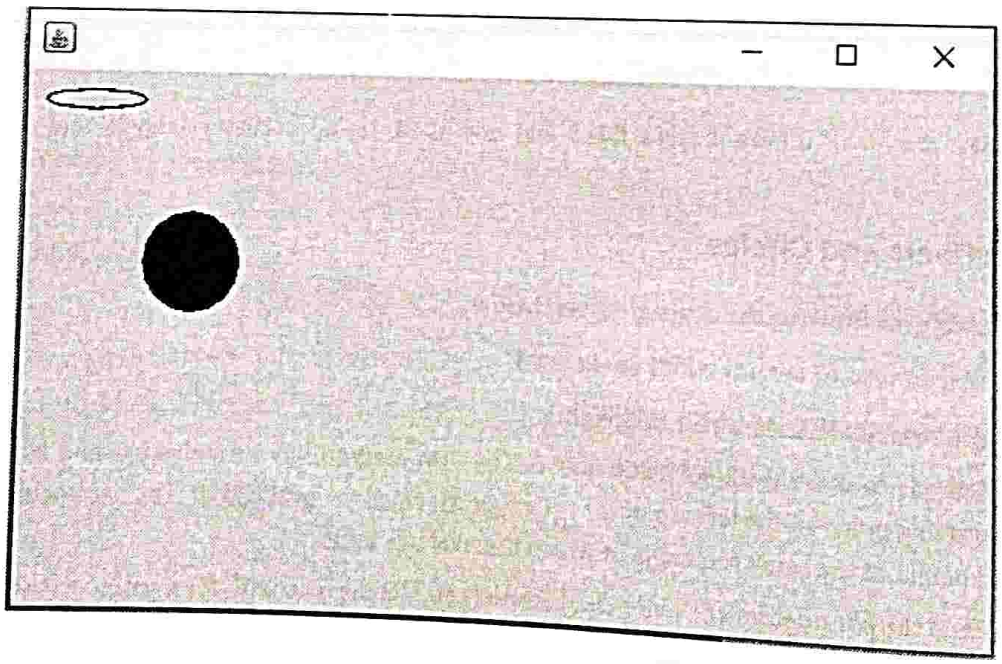
- The methods available for drawing the ovals are given below:
  1. void drawOval(int top, int left, int width, int height)
  2. void fillOval(int top, int left, int width, int height)
- There are two methods to draw ovals as listed above. The first method has variables top and left which are the co-ordinates 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.

Program 6.2.3 : Write a graphics Java program to draw Oval and a filled Oval of different and equal diameters.

```
import javax.swing.JFrame;
import java.awt.Graphics;
import javax.swing.JPanel;
class MyPanel extends JPanel
{
        public void paint(Graphics g)
        {
                g.drawOval(10,10,50,10);
                g.fillOval(60,70,50,50);
        }
}
public class Graphi3 extends JFrame
{
        Graphi3()
        {
                setSize(500,300);
                MyPanel mp=new MyPanel();
                add(mp);
        }
        public static void main(String args[])
        {
                new Graphi3().setVisible(true);
        }
}
```

Output



Explanation

Similarly the ovals are drawn as other shapes in the previous programs.

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.

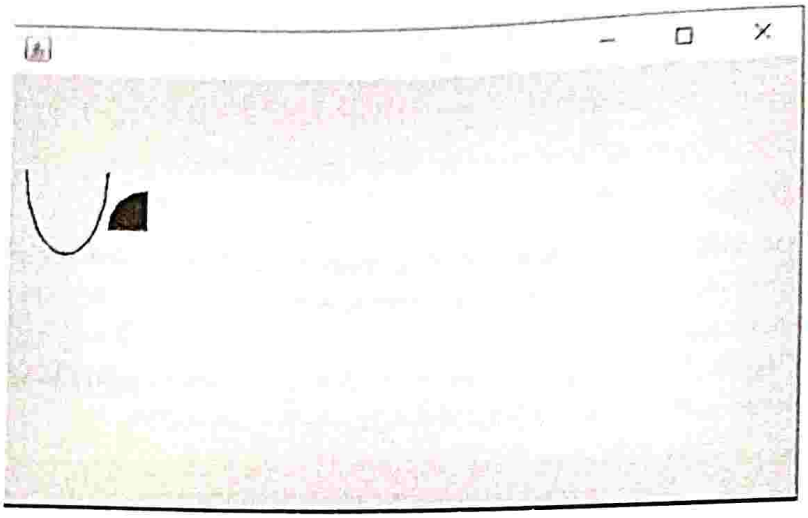
Let us see a Program 6.2.4 to draw some arcs.

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

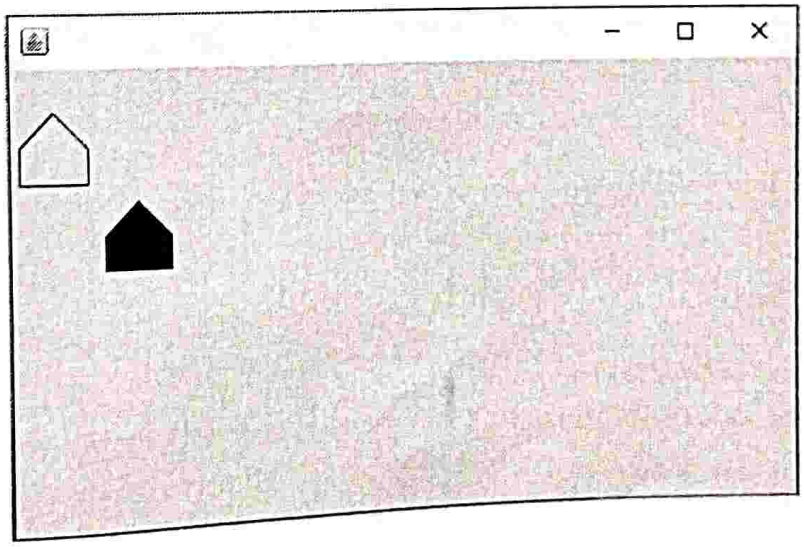
- The methods available for drawing the polygons are given below:
  1. void drawPolygon(int x[ ], int y[ ], int numPoints)
  2. void fillPolygon(int x[ ], int y[ ], int numPoints)
- There are two methods to draw polygons as listed above.
- The first method has parameters as an array of integer which 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);
```

```
            for(i=0;i<=5;i++)
            {
                    x[i]+=50;
                    y[i]+=50;
            }
            g.fillPolygon(x,y,6);
        }
}
public class Graphi5 extends JFrame
{
        Graphi5()
        {
                setSize(500,300);
                MyPanel mp=new MyPanel();
                add(mp);
        }
        public static void main(String args[])
        {
                new Graphi5().setVisible(true);
        }
}
```

Output



Explanation

- Two arrays are made of x and y co-ordinates for the required polygon i.e. Pentagon. The arrays and number of points i.e. 6 is passed to make the pentagon.
- A "for" loop is used to change the values of all the co-ordinates and draw a filled polygon.

6.2.6 Changing Colors

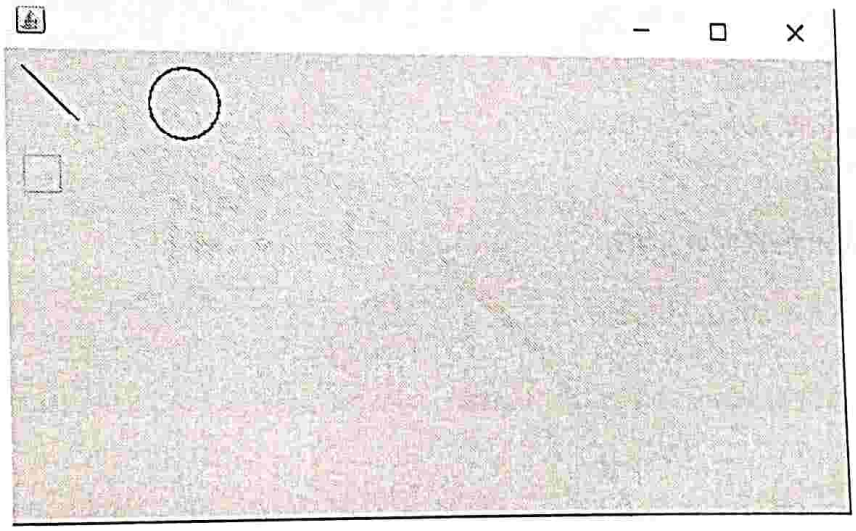
- The shapes drawn in all the above graphics in the above sub-sections is by default black in color. But sometimes we need to change the colors.
- In this sub section, we will see how can we change colors for the shapes using some methods.
- There are quite a few methods of changing colors. There are 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);
```

```
            g.setColor(blue);
            g.drawOval(80,10,40,40);
            g.setColor(yellow);
            g.fillArc(80,80,40,40,90,270);
        }
}
public class Graphi6 extends JFrame
{
        Graphi6()
        {
            setSize(500,300);
            MyPanel mp=new MyPanel();
            add(mp);
        }
        public static void main(String args[])
        {
            new Graphi6().setVisible(true);
        }
}
```

Output



Explanation

- We have drawn different shapes with different colors.
- The colors are set using the setColor() method of the Graphics class.
- The object of the Color circle is to be passed along to the method.
- We have made four color images, one in red, then in green, then blue and green.
- The first three colors are easily made using the color required and other two parameters as zero to the constructor.
- The yellow color is a combination of red and green. Hence these values are given as 255 while blue as zero.

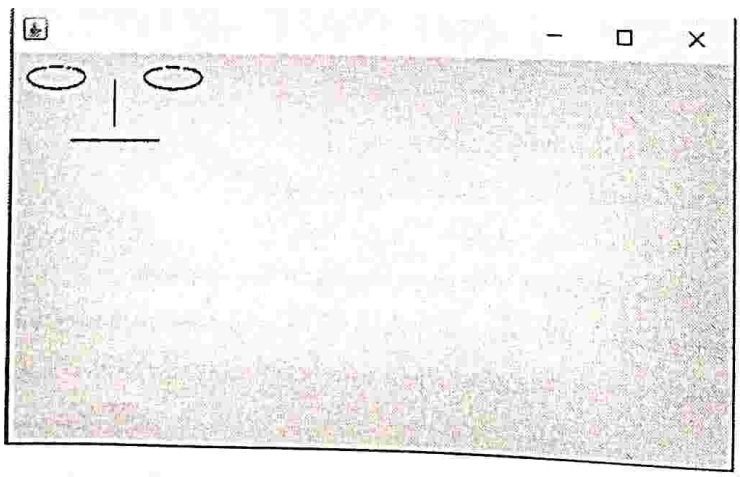
6.3 Miscellaneous Graphics Programs

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

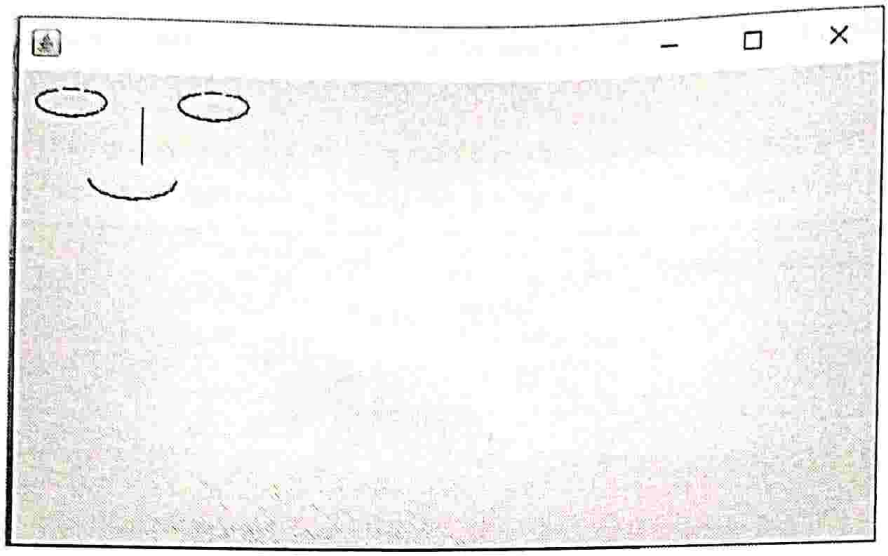
- Just by using the different methods studied upto now and the co-ordinate system of Graphics, we have made this program.
- Whenever such a figure is given to be drawn, you can try drawing rows and columns giving them co-ordinates or draw on graph paper with co-ordinates. Then accordingly you can get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

Output

```
C:\Java>javac CreateFile.java
C:\Java>java CreateFile
Created sucessfuly file: Data.txt
```

Explanation

- By importing java.io, all the classes of the package including the class "File" are imported.
- We have made an object f0 of the class "File" and used the createNewFile() method to create the required file.
- If the file already exists, the method returns a "False" and hence we display that the file already exists in the else part.
- A very important thing is that the method createNewFile() may throw an exception if it is not able to create a file (even if the given file doesn't exists). There could be various reasons for that like the permissions are not given to create a file in a given folder, etc. Hence, we need to use this method in a try-catch block. The exception thrown is of type IOException, but as seen in earlier chapters we can catch it using the base class i.e. Exception.

Program 6.4.2 : Write a Java program to display the info of a file "Data.txt".

```
import java.io.*;
class FileInfo
{
  public static void main(String[ ] args)
  {
        File f0 = new File("C:Data.txt");
        if (f0.exists())
        {
              System.out.println("File Name: " + f0.getName());
              System.out.println("Absolute Path: " + f0.getAbsolutePath());
              System.out.println("Writeable?: " + f0.canWrite());
              System.out.println("Readable?: " + f0.canRead());
              System.out.println("Size of the file(in bytes): " + f0.length());
        }
        else
        {
              System.out.println("The file does not exist.");
        }
  }
}
```

Output

```
C:\Java>javac FileInfo.java
C:\Java>java FileInfo
File Name: Data.txt
```

```
Absolute Path: C:\Java\Data.txt
Writeable?: true
Readable?; true
Size of the file(in bytes): 0
```

Explanation

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

Program 6.4.3 : Write a Java program to write into the file "Data.txt".

```
import java.io.*;
class WriteToFile
{
  public static void main(String[ ] args)
        {
              try
              {
                FileWriter f0 = new FileWriter("C:Data.txt");
                f0.write("This is a book on Java.");
                f0.close();
                System.out.println("Content is successfully written to the file.");
              }
              catch (Exception e)
              {
               System.out.println("An unexpected error has occurred.");
              }
        }
}
```

Output

```
C:\Java>javac WriteToFile.java
C:\Java>java WriteToFile
Content is successfully written to the file.
```

Explanation

- We have created an object of the class "FileWriter" and its instance method write() to write into the file.
- We have used the various methods discussed in the table in this section to display the information of the file.

Program 6.4.4 : Write a Java program to read from the file "Data.txt".

```
import java.io.*;
import java.util.Scanner;
class ReadFromFile
{
```

```
    public static void main(String[] args)
    {
        File f0 = new File("C:Data.txt");
        try
        {
                Scanner sc= new Scanner(f0);
                while (sc.hasNextLine())
                {
                  String str = sc.nextLine();
                  System.out.println(str);
                }
                sc.close();
      }
       catch (Exception e)
       {
          System.out.println("An unexpected error is occurred.");
       }
    }
}
```

Output

```
C:\Java>javac ReadFromFile.java
C:\Java>java ReadFromFile
This is a book on Java.
```

Explanation

- We have created an object of the class "Scanner" attached the "File" class object to it instead of System.in as in earlier chapters. Thus the Scanner class object "sc" will read from the given file, instead of reading from the standard input device.
- At the end we have closed the same, so as to avoid any unexpected access to the file.
- We have used the various methods discussed in the table in this section to display the information of the file.

Program 6.4.5 : Write a Java program to delete the file "Data.txt".

```
import java.io.*;
class DeleteFile
{
        public static void main(String[ ] args)
        {
                File f0 = new File("C:Data.txt");
                if (f0.delete())
```

```
        {
                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

Stream Class Name	Stream Class Type	Description
FileOutputStream	ByteStream	Has methods to write onto the file
InputStream	ByteStream	Abstract input stream class
OutputStream	ByteStream	Abstract output stream class
PrintStream	ByteStream	Has methods like print() and println()
ByteArrayInputeStream	ByteStream	Can read from a byte array
ByteArrayOutputStream	ByteStream	Can write into a byte array
BufferedReader	CharStream	Used for buffered input
BufferedWriter	CharStream	Used for buffered output
FileReader	CharStream	Can read character array from file
FileWriter	CharStream	Can write character array to file
CharArrayReader	CharStream	Can read from character array
CharArrayWriter	CharStream	Can write onto character array
PrintWriter	CharStream	Has methods like print() and println()

Let us see a file handling program using the FileReader and FileWriter stream classes.

Program 6.5.1 : Write a Java program to read and write into a file using random access.

```
import java.io.*;
public class RandomAccess
{
        public static void main(String[] args)
        {
                try
                {
                  writeToFile("Data.TXT", "This is a book on Java", 0);
                  System.out.println(new String(readFromFile("Data.TXT", 0, 22)));
                }
                catch (IOException e)
                {
                        System.out.println("An unexpected error is occurred.");
                }
        }
        private static void writeToFile(String path, String dat, int posi) throws IOException
        {
```

```
        RandomAccessFile f0 = new RandomAccessFile(path, "rw");
        f0.seek(posi);
        f0.write(dat.getBytes());
        f0.close();
    }
    private static byte[ ] readFromFile(String path, int posi, int size)  throws IOException
    {
        RandomAccessFile f0 = new RandomAccessFile(path, "r");
        f0.seek(posi);
        byte[] b = new byte[size];
        f0.read(b);
        f0.close();
        return b;
    }
}
```

Output

```
C:\Java>javac RandomAccess.java
C:\Java>java RandomAccess
This is a book on Java
```

Explanation

- We have written two methods writeToFile() and readFromFile(). These two methods are called with the required parameters in the main() method. The call for these methods is written in a try-catch block so that if any error is generated, it can be notified accordingly.
- Both the methods are defined with the "throws IOException" as discussed in the earlier chapter.
- Since the methods are called directly without making any object, we have defined the methods to be static.
- The parameter passed to the writeToFile() method are name of the file, the data to be written and the position from where it must be written in the file. An object "f0" is created in the writeToFile() method with "rw" mode (i.e. read or write mode) of the class RandomAccessFile. This class has instance methods used like the seek(), write() and close() methods. The seek() method makes the cursor reach to the given location/position in the file. The write() method writes the stream of characters into the file. The close() method as the name states closes the corresponding file.
- The parameters passed to the readFromFile() method are name of the file, position from where to read and the size of the return buffer of byte type array. This indicates that the readFromFile() method will return a byte array with the data read from the specified file. An object "f0" is created in the writeToFile() method with "r" mode(i.e. read mode) of the class RandomAccessFile. This class has instance methods used like the seek(), read() and close() methods. The seek() method makes the cursor reach to the given location/position in the file. The read() method reads the stream of characters from the file and returns the array/stream of bytes. The close() method as the name states closes the corresponding file.

❑❑❑

Note

About the Authors

Harish G. Narula

Author has done his degree of bachelors in Electronics and Telecommunication and Master degree in Electronics Engineering with specialization in Digital Electronics. He had been teaching in various renowned Engineering colleges like D. J. Sanghvi for almost 16 years. He has authored 60+ books for engineering students (for various streams like Computer, Electronis, Mechanical, Telecommunication, Information Technology etc.) as well as for CS (Bifocal) subject of XII.

"For online batches, on 9892 177 166"

Tech Knowledge Publications

Head Office :
B/5, First Floor, Maniratna Complex, Taware Colony,
Aranyeshwar Corner, Pune - 411009. Maharashtra, India.
Tel. : 91-20-24221234, 91-20-24225678

PRINTED ON RECYCLED PAPER • PRINTED ON RECYCLED PAPER •

ISBN : 978-93-90694-73-0

Price ₹ 395/-

MO227B

Distributors

Vidyarthi Sales Agencies
T. : (022) 23867279, 98197 76110

Ved Book Distributors
80975 71421 / 92208 77214 / 80973 75002

Bharat Sales Agency
T. : (022) 23819359, 86572 92797

Our Branches : Pune | Mumbai | Kolhapur | Nagpur | Solapur | Nashik

For Library Orders Contact - Ved Book Distributors M : 80975 71421 / 80973 75002

Email : info@techknowledgebooks.com | Website : www.techknowledgebooks.com

Like us at: TechknowledgePublications