Object Oriented Programming

(Code : 214444)

Semester III – Information Technology

(Savitribai Phule Pune University)

Strictly as per the New Credit System Syllabus (2019 Course)
Savitribai Phule Pune University w.e.f. academic year 2020-2021

Harish G. Narula

Formerly, Assistant Professor (Senior),
Department of Computer Engineering
D. J. Sanghvi College of Engineering,
Mumbai, Maharashtra, India.

Ansari M. A.

Ph. D (Computer Engineering)
Asst. Professor,
Department of Computer Engineering,
Smt. Kashibai Navale College of Engineering,
Vadgaon Budruk, Pune-411041

Khushboo Shah

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

For Lab Manual visit our website https://techknowledgebooks.com/library/#student_downloads

TechKnowledge Publications

PO132A Price ₹ 235/-

(Book Code : PO132A)

Object Oriented Programming (Code : 214444)

Harish G. Narula, M. A. Ansari, Khushboo Shah

(Semester III – Information Technology, (Savitribai Phule Pune University))

First Printed in India : July 2013 (For Mumbai University)

First Edition : August 2020

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 : 978-93-89889-57-4

Published by

TechKnowledge Publications

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

[214444] (FID : PO132) (Book Code : PO132A)

(Book Code : PO132A)

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"

- Lt. Shri. Pradeepji L. Lunawat

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

Preface

My Dear Students,

We are extremely happy to come out with this book on **"Object Oriented Programming"** for you. 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 heart and motivate us to achieve our milestone.

We are thankful to Shri. J. S. Katre, Shri. Shital Bhandari, Shri. Arunoday Kumar and Shri. Chandroday Kumar for the encouragement and support that they have extended. We are also thankful to Seema Pradeep 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 also thankful to our family members and friends for their patience and encouragement.

- Authors

Syllabus

214444 : Object Oriented Programming

Teaching Scheme :	Credit	Examination Scheme :
Lectures : 03 Hours/Week	03	**In-Sem (Paper) :** 30 Marks **End-Sem (Paper) :** 70 Marks

Prerequisite Courses :

Principles of Programming Languages

Course Objectives :

1. Apply concepts of object oriented paradigm.
2. Design and implement models for real life problems by using object oriented programming.
3. Develop object oriented programming skills.

Course Outcomes :

CO1 : Differentiate various programming paradigms and apply basic concepts of OOP.

CO2 : Identify classes, objects, methods, and handle object creation, initialization, and destruction to model real-world problems.

CO3 : Identify relationship among objects using inheritance and polymorphism.

CO4 : Handle different types of exceptions and perform generic programming.

CO5 : Use file handling for real world application.

CO6 : Apply appropriate design patterns to provide object-oriented solutions.

UNIT - I : Foundations of Object Oriented Programming **(06 Hours)**

Introduction OOP : Software Evolution, Introduction to Procedural, Modular, Object-Oriented and Generic Programming Techniques, Limitations of Procedural Programming, Need of Object-Oriented Programming, Fundamentals of Object-Oriented Programming: Objects, Classes, Data Members, Methods, Messages, Data Encapsulation, Data Abstraction and Information Hiding, Inheritance, Polymorphism, Static and Dynamic Binding, Message Passing. **(Refer Chapter 1)**

UNIT - II : Classes, Objects and Methods **(06 Hours)**

Class : Creating a Class, Visibility/Access Modifiers, Encapsulation, Methods: Adding a Method to Class, Returning a Value, Adding a Method That Takes Parameters, The 'this' Keyword, Method Overloading, Object Creation, Using Object as a Parameters, Returning Object, Array of Objects, Memory Allocation: 'new', Memory Recovery: 'delete', Static Data Members, Static Methods, Forward Declaration, Class as Abstract Data Types (ADTs), Classes as Objects.

(Refer Chapter 2)

(Book Code : PO132A)

UNIT - III : Constructors and Destructors **(06 Hours)**

Constructors : Introduction, Use of Constructor, Characteristics of Constructors, Types of Constructor, Constructor Overloading, Constructor with Default Arguments, Symbolic Constants, Garbage Collection: Destructors and Finalizers. **(Refer Chapter 3)**

UNIT - IV : Inheritance and Polymorphism **(06 Hours)**

Inheritance : Introduction, Need of Inheritance, Types of Inheritance, Benefits of Inheritance, Cost of Inheritance, Constructors in derived Classes, Method Overriding, Abstract Classes and Interfaces. Polymorphism and Software Reuse: Introduction, Types of Polymorphism (Compile Time and Run Time Polymorphism), Mechanisms for Software Reuse, Efficiency and Polymorphism. **(Refer Chapter 4)**

UNIT - V : Exception Handling and Generic Programming **(06 Hours)**

Exception : Errors, Types of Errors, Exception and its Types, Exception-Handling Fundamentals, Uncaught Exception, Using try and Catch, Multiple Catch Clauses, Nested Try Statements, User Define Exception using Throw.

Generics : What are Generics? Introduction to Language Specific Collection Interface: List Interface and Set Interface, Collection Classes: ArrayList Class and LinkedList Class. **(Refer Chapter 5)**

UNIT - VI : File Handling and Design Patterns **(06 Hours)**

File Handling : Introduction, Concepts of Stream, Stream Classes, Byte Stream Classes, Character Stream, Classes, Using Stream, Other Useful I/O Classes, Using the File Class, Input/output Exceptions, Creation of Files, Reading/Writing Character, Reading/Writing Bytes, Handling Primitive Data Types, Concatenating and Buffering Files, Random Access Files.

Design Patterns : Introduction, Types of Design Patterns, Adapter, Singleton, Iterator. **(Refer Chapter 6)**

❑❑❑

(Book Code : PO132A)

Unit - I

Chapter 1 : Foundations of Object Oriented Programming 1-1 to 1-74

1.1 Computer 1-1
1.2 Binary Number System 1-1
1.3 Software Evolution 1-4
1.3.1 History of C / C++ Programming Languages 1-4
1.3.2 Comparison of C++ and Java 1-5
1.3.3 Java Evolution: History 1-6
1.4 Introduction to Procedural, Modular, Object-Oriented and Generic Programming Techniques 1-6
1.4.1 Introduction Object Oriented Programming (OOP) and OOP Languages 1-7
1.4.2 Limitations of Procedural Oriented Programming (POP) and Need of Object Oriented Programming (OOP) 1-7
1.4.3 Applications of Object Oriented Programming Language 1-8
1.5 Fundamentals of Object-Oriented Programming : Objects, Classes, Data Members, Methods 1-8
1.6 Static and Dynamic Binding, Message Passing 1-9
1.7 Java Virtual Machine (JVM) 1-9
1.8 Formatted and Unformatted IO Functions of C++ 1-11
1.8.1 Formatted IO Functions 1-11
1.8.2 Unformatted IO Functions 1-12
1.9 Basic C++ Program Examples 1-12
1.9.1 Type Casting 1-14
1.10 Input / Output in Java 1-15
1.10.1 Displaying Output in Java 1-15
1.10.2 Accepting Input in Java 1-15
1.10.3 Accepting Input using BufferedReader Class 1-16
1.11 First Program of Java 1-17
1.12 Installing and Implementing Java 1-22
1.12.1 Java Development Kit (JDK) 1-23
1.13 Solved Programs 1-23
1.14 Revisiting Loops Statements 1-27
1.14.1 Programs Based on for Loop 1-28
1.14.2 Nested for Loop 1-35
1.15 Revisiting Control Statements 1-43
1.15.1 Programs using if-else Statement 1-44
1.15.2 if-else Ladder or if-else if 1-46
1.16 Revisiting Arrays 1-47
1.16.1 Multi-dimensional Arrays 1-57
1.16.2 Arraycopy() 1-64
1.17 Revisiting Strings 1-65
1.17.1 Methods of String Class 1-66
1.17.2 Methods in String Buffer Class 1-72

Unit - II

Chapter 2 : Classes, Objects and Methods 2-1 to 2-39

2.1 Comparison of Procedure Oriented Programming and Object Oriented Programming 2-1
2.1.1 Class and Object : Introduction 2-2
2.1.2 Introduction to Objects 2-3
2.1.2(A) State and Behaviour of an Object 2-3
2.2 Visibility / Access Modifiers 2-4
2.2.1 Access Specifiers in C++ 2-4
2.2.2 Introduction to Java Access Modifiers 2-4
2.3 Encapsulation 2-5
2.4 Creating a Class and Adding Methods to a Class 2-5
2.4.1 Defining Member Functions of a Class 2-5
2.4.2 Internally Defined Functions 2-5
2.4.2(A) Nesting of Member Function 2-8
2.4.3 Externally Defined Functions 2-8
2.4.4 Inline Member Functions 2-10
2.4.5 Java Member Methods 2-11

2.5 Object Creation and Memory Allocation: 'new' Operator 2-15

2.5.1 Creating Objects and Memory Allocation of Objects 2-15

2.5.2 Memory Recover: 'delete' Operator 2-15

2.6 Returning a Value 2-16

2.7 Adding a Method that Takes Parameters 2-17

2.8 The "this" Keyword 2-18

2.8.1 "this" Keyword in C++ 2-18

2.8.2 The "this" Keyword in Java 2-19

2.9 Method Overloading 2-20

2.9.1 Function Overloading or Function Polymorphism 2-20

2.9.2 Constructor Overloading 2-22

2.10 Using Object as Parameter and Returning an Object 2-24

2.10.1 Objects as Function Parameter/Arguments 2-24

2.10.2 Passing Objects to a Method 2-24

2.10.3 Returning Objects from a Method 2-26

2.10.4 Call by Value and Call by Reference 2-27

2.10.4(A) Call by Value 2-27

2.10.4(B) Call by Reference 2-28

2.11 Array of Objects in C++ 2-28

2.11.2 Array of Objects in Java 2-31

2.13 Static Data Members and Methods 2-37

2.13.2 Static Class Members in Java 2-38

2.14 Forward Declaration 2-39

2.15 Class as "Abstract Data Type" (ADT) 2-39

- Model Question Paper (In Sem.) Q-1

Unit - III

Chapter 3 : Constructors and Destructors **3-1 to 3-20**

3.1 Introduction of Constructors : Use and Characteristics 3-1

3.1.1 Constructor 3-1

3.1.2 Constructors, Destructors, Modifiers, Iterators and Selectors 3-1

3.2 Types of Constructor 3-2

3.2.1 Default Constructor 3-2

3.2.2 Parameterized Constructor 3-3

3.2.3 Copy Constructor (Multiple Constructors/Constructor Overloading) 3-4

3.2.4 Constructors 3-7

3.2.4(A) Parameterized Constructor 3-7

3.2.4(B) Default Constructor 3-9

3.2.4(C) Copy Constructor 3-10

3.3 Constructor Overloading 3-12

3.4 Constructor with Default Arguments 3-13

3.5 Symbolic Constants 3-14

3.6 Garbage Collection : Destructors and Finalizers 3-15

3.6.1 Destructor 3-15

3.6.2 finalize() Method 3-16

3.7 A Book Shop Inventory 3-17

Unit - IV

Chapter 4 : Inheritance and Polymorphism **4-1 to 4-54**

4.1 Inheritance : Introduction and Need 4-1

4.1.1 Inheritance 4-1

4.1.1(A) Visibility Modes and Effects 4-1

4.1.2 Introduction to Inheritance 4-2

4.1.3 Cost of Inheritance 4-3

4.2 Types of Inheritance or Mechanism of Software Reuse in C++ 4-3

4.2.1 Single Inheritance 4-3

4.2.1(A) Constructors in Derived Class 4-7

4.2.2 Multi Level Inheritance 4-7

4.2.3 Multiple Inheritance 4-12

4.2.4 Hybrid Inheritance 4-13

4.2.5 Problem in Multiple and Hybrid Inheritance 4-15

4.2.6 Hierarchical Inheritance....4-16

4.3 Types of Inheritance or Mechanism of Software Reuse in Java....4-19

4.3.1 Single Inheritance....4-19

4.3.2 Multi Level Inheritance....4-21

4.3.3 Hierarchical Inheritance....4-23

4.4 Method Overriding....4-27

4.5 Abstract Classes and Interfaces....4-28

4.5.1 Abstract Classes....4-28

4.5.2 Interface....4-31

4.5.2(A) Extending an Interface....4-31

4.5.2(B) Variables in Interface....4-31

4.5.2(C) Difference between Interface and Abstract Class....4-32

4.6 Polymorphism....4-36

4.6.1 Polymorphism in C++....4-37

4.7 Types of Polymorphism: Compile Time and Run Time Polymorphism....4-37

4.7.1 Dynamic Binding using Virtual Function or Compile/Run Time Polymorphism....4-37

4.7.2 Rules for Virtual Functions....4-38

4.7.3 Pointer to Derived Class....4-39

4.7.4 Virtual Base Class and Abstract Class....4-42

4.7.5 Static Polymorphism in Java....4-43

4.7.5(A) Constructor Overloading....4-43

4.7.5(B) Method Overloading....4-44

4.7.5(C) Operator Overloading....4-46

4.7.6 Dynamic Polymorphism in Java....4-50

4.7.6(A) Dynamic Method Dispatch....4-50

4.8 Efficiency and Polymorphism....4-51

4.9 Case Study : A Bank Account System....4-51

Unit - V

Chapter 5 : Exception Handling and Generic Programming 5-1 to 5-21

5.1 Errors and Exception-handling Fundamentals....5-1

5.2 Types of Errors or Exceptions....5-1

5.2.1 Checked Exceptions....5-1

5.2.2 Unchecked or Uncaught Exceptions....5-2

5.3 Using try and Catch....5-4

5.4 Multiple Catch Clauses....5-6

5.5 Nested try Statements....5-7

5.5.1 Keyword “throws”....5-9

5.6 User Define Exception using Throw....5-9

5.7 Exception Handling in C++....5-11

5.8 Exception Handling Mechanism....5-11

5.8.1 Try-catch-throw, Multiple Catch and Catch All....5-11

5.8.2 Implementing User Defined Exception....5-12

5.8.3 Concept of Throw and Catch with Example....5-12

5.9 What are Generics?....5-13

5.9.1 Function Templates and Examples....5-13

5.9.2 Overloading a Template Function Using a Non-template Function....5-17

5.9.3 Overloading a Template Function Using a Template Function....5-17

5.9.4 Generic Classes, Class Template and Examples....5-18

5.9.5 Overview and Use of Standard Template Library (STL)....5-18

5.10 Introduction to Language Specific Collection Interface : List Interface and Set Interface....5-19

5.10.1 List Interface....5-19

5.10.2 Set Interface....5-19

5.11 Collection Classes : ArrayList Class and LinkedList Class....5-19

5.11.1 ArrayList Class....5-19

5.11.2 LinkedList Class 5-20

5.12 Exception Handling and Generic Programming using Array List (ArrayList Class) 5-20

Unit - VI

Chapter 6 : File Handling and Design Patterns **6-1 to 6-15**

6.1 File Handling : Introduction 6-1

6.2 Concepts of Stream 6-1

6.3 Stream Classes : Byte, Character 6-1

6.3.1 File Stream Classes 6-2

6.3.2 Advantages of Stream Classes 6-2

6.4 Other Useful I/O Classes 6-2

6.5 Input/Output Exceptions 6-3

6.6 Creation of Files and using the File Class 6-3

6.7 Using streams for reading from and writing to the files 6-5

6.7.1 Reading/Writing Bytes and Detecting End of File 6-6

6.8 Handling Primitive Data Types and Random Access Files 6-8

6.9 Concatenating and Buffering Files 6-9

6.10 Design Patterns : Introduction 6-11

6.11 Types of Design Patterns : Adaptor, Singleton and Iterator 6-11

6.11.1 Adaptor 6-11

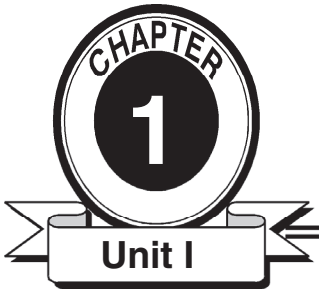
6.11.2 Singleton 6-12

6.11.3 Iterator 6-12

6.12 Student Management System 6-12

- Model Question Paper (End Sem.) ..... Q-1 to Q-2

❑❑❑



Foundations of Object Oriented Programming

Syllabus

Introduction OOP : Software Evolution, Introduction to Procedural, Modular, Object-Oriented and Generic Programming Techniques, Limitations of Procedural Programming, Need of Object-Oriented Programming, Fundamentals of Object-Oriented Programming: Objects, Classes, Data Members, Methods, Messages, Data Encapsulation, Data Abstraction and Information Hiding, Inheritance, Polymorphism, Static and Dynamic Binding, Message Passing.

1.1 Computer

- The computer as seen by you can be as illustrated in the Fig. 1.1.1.

Fig. 1.1.1 : Computer

- Besides other units, it is made of a monitor (standard display unit), CPU (Central Processing Unit) and keyboard (standard input device).
- A **computer** can be visualized as a system with Input/Output (I/O) devices, Central Processing Unit (CPU) and the memory interconnected as shown in the Fig. 1.1.2. This is the hardware view of a computer.

Fig. 1.1.2 : Hardware structure of a computer

- As a programmer, the **computer** can be seen as a programmable system. It is a system that can be programmed by the programmer

1.2 Binary Number System

- As seen above that a computer can understand only binary language, let us first see what is this binary language ?
- Binary number system can have only two representations namely '0' and '1'.
- All numbers and characters are to be represented using '0' or '1' only.
- The conversion from binary to decimal and decimal to binary can be done as shown below with examples.
- Decimal to binary : The steps to be followed for converting a decimal number to a binary number are :
  - Divide the decimal number by 2.
  - Note down the quotient and remainder as shown in the example below.
  - Repeat the above procedure till the quotient is zero.

- The last remainder is the MSB (Most Significant Bit i.e. the bit with highest weight) and the first remainder is the LSB (Least Significant Bit).

E.g. $(5)_8 = (?)_2$

$(5)_{10}$ means the number 5 in decimal (base 10) number system. Base 10 means 20 different representations 0 to 8.

$(?)_2$ means the number in binary (base 2) number system. Base 2 means 2 different representations 0 and 1.

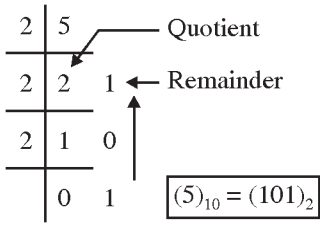


Fig. 1.2.1

– **Binary to decimal :** The steps to be followed for converting a binary number to a decimal number are :

- Multiply the binary digits (bits) with powers of 2 according to their positional weight. The position of the first bit (going from right to left) is 0 and it keeps on incrementing as you go towards left for every bit.

E.g. $(11100)_2 = (?)_{10}$

$= 1 \times 2^2 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^2 + 0 \times 2^0$

$= 16 + 4 + 2 + 0 + 0 = (28)_6$

– This is how your decimal data is stored in the computer. You store 5, but it stores 101 in binary form.

– Let us see some bitwise logical operations that are performed on binary data.

– There are various operations that are performed on binary data like AND, OR, EXOR and NOT. These operations are explained below.

1. **AND :** The output is 1 if and only **if A AND B** (A & B being the inputs) are 1.

A(INPUT)	B(INPUT)	Y(OUTPUT)
0	0	0
0	0	1
1	0	0
0	1	0

2. **OR :** The output is 1 if and only **if A OR B** (A & B being the inputs) is 1.

A(INPUT)	B(INPUT)	Y(OUTPUT)
0	0	0
0	1	0
1	0	1
1	0	1

3. **NOT :** The output **is NOT A** (A being the input) i.e. complement of A.

A(INPUT)	Y(OUTPUT)
0	1
1	0

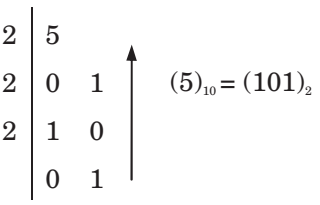
4. **EXOR :** The output is 1 if and only **if EXCLUSIVELY A OR B** (A & B being the inputs) are 1.

A(INPUT)	B(INPUT)	Y(OUTPUT)
0	0	0
0	1	1
1	0	1
1	1	0

Some examples of performing these operations on the binary data are given below :

1. 5 AND 3 :

In 'C' programming this is written as 5 & 3.



2	3
2	1 1 ↑
	0 1

$(3)_{10} = (011)_2$

Note : We can prefix 0's to a number in binary as in decimal. Here, $(011)_2$ is written to make it 3 digit data, as $(5)_{10} = (101)_2$ is 3 digit.

Now, 5 & 3

$$\begin{array}{r} (1\ 0\ 1)_2 \\ \&\ (0\ 1\ 1)_2 \\ \hline (0\ 0\ 1)_2 \end{array}$$

Note : & is bitwise AND operator i.e. it works on each bit as shown in the above example.

$$(001)_2 = 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$
$$= (1)_{10}$$
$$(5)_{10} \ \&\ (3)_{10} = (1)_{10}$$

2. 13 AND 10 :

2	13
2	6 1
1	2 0 ↑
2	1 1
	0 1

$(13)_{10} = (1101)_2$

2	10
2	3 0
2	2 1 ↑
2	1 0
	0 1

$(10)_{10} = (1010)_2$

Now; 13 and 10

$$\begin{array}{r} (1\ 0\ 0\ 1)_2 \\ \&\ (1\ 0\ 1\ 1)_2 \\ \hline (1\ 0\ 0\ 1)_2 \end{array}$$

$(1000)_2 = 1 \times 2^3 + 0 \times 2^4 + 0 \times 2^1 + 0 \times 2^2 = (8)_{10}$

$(13)_{10}$ & $(10)_{10} = (+)_{10}$

3. 7 OR 6 :

In 'C' programming, this is written as 7 | 6.

2	7
2	3 1 ↑
2	1 1
	0 1

$(7)_{10} = (111)_2$

2	6
2	3 0 ↑
2	1 1
	0 1

$(6)_{10} = (110)_2$

Now; 7 OR 6

$$\begin{array}{r} (1\ 1\ 1)_2 \\ |\ (1\ 1\ 0)_2 \\ \hline (1\ 1\ 1)_2 \end{array}$$

$(111)_2 = 1 \times 2^2 + 1 \times 2^2 + 1 \times 2^0$
$= (7)_{10}$

$\therefore \quad (7)_{10} \mid (6)_{10} = (7)_{10}$

4. 5 OR 2 :

2	5
2	2 1 ↑
2	1 0
	0 1

$(5)_{10} = (101)_2$

2	2
2	1 0 ↑
	0 1

$(2)_{10} = (010)_2$

Now, 5 OR 2

$$\begin{array}{r} (1\ 0\ 1)_2 \\ |\ (0\ 1\ 0)_2 \\ \hline (1\ 1\ 1)_2 \end{array}$$

$(111)_2 = 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$
$= (7)_{10}$

$\therefore \quad (4)_{10} \mid (2)_{10} = (7)_{10}$

5. 6 EXOR 3 :

In C, this is written as 6 ^ 3

2	6
2	3 0
2	2 1
	0 1

$(6)_{10} = (110)_2$

2	3
2	1 1
	0 1

$(3)_{10} = (011)_2$

Now; 6 EXOR 3

$$\begin{array}{r} (1\ 0\ 1)_2 \\ \wedge\ (0\ 1\ 1)_2 \\ \hline (1\ 0\ 1)_2 \end{array}$$

$(101)_2 = 1 \times 2^2 + 0 \times 2^4 + 1 \times 2^0$

$= (5)_{10}$

$\therefore \quad (6)_{10} \wedge (3)_{10} = (5)_{10}$

6) NOT 3 :

In C, this is written as ~ 3

2	3
2	1 1
	0 1

$(3)_{10} = (011)_2$

Now; ~ 3 is 0 1 0 1 1

1 0 1 0 0

Note :
1) The data may be of more than 2 bits and hence you will get the output depending on the size of data.
2) In computer binary language you will get this value as $(-4)_{10}$.
3) Hence if you say $\sim(10)_{10}$, you will get $-(11)_{10}$ and so on.

1.3 Software Evolution

- **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 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++ 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 these two programming languages viz. C and C++. Also we have to make some small programs using these languages.

1.3.1 History of C / C++ Programming Languages

- C programming language was developed by Dennis Ritchie at the Bell Laboratories of AT & T.

- He developed a complier that can convert the HLL namely C to the machine language that can be understood by the processor or the computer. But his compiler could understand only if certain rules were followed while writing the program.
- We need to understand and know these rules to program our computers using this programming language. C was the most popular programming language.
- Some other languages had become popular after some time of evolution of C. This was because these other programming languages were object oriented i.e. more significance was given to object.
- While 'C' was a procedure oriented language i.e. more significance was given to the procedure to perform a task.
- C++ which is an object oriented programming language was then developed again at Bell Laboratories by Bjarne Stroustrup. Earlier, this language was called as C with classes, but was later renamed as C++ in 1983.
- Object oriented and procedure oriented concepts of a programming language will be gradually seen in this book. Some of these concepts will be cleared in this chapter itself, while some will come as you reach towards the end of this book.

1.3.2 Comparison of C++ and Java

Table 1.3.1 : 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.

Sr. No.	C++	Java
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.
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.

1.3.3 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.

1.4 Introduction to Procedural, Modular, Object-Oriented and Generic Programming Techniques

Q. Write any two characteristics of procedure oriented programming. **(2 Marks)**

Q. List any four features of POP. **(4 Marks)**

Q. In procedure oriented programming all data are shared by all functions. Is this statement TRUE? Justify your answer. **(4 Marks)**

Q. What are the features of procedure oriented programming? **(4 Marks)**

- A procedure oriented programming language is one in which procedure i.e. method or functions are given more significance. Here, one function calls another.
- There can be global variables and local variables to each function.
- Fig 1.4.1 shows how a procedure oriented programming operation.
- 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. 1.4.1. 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.

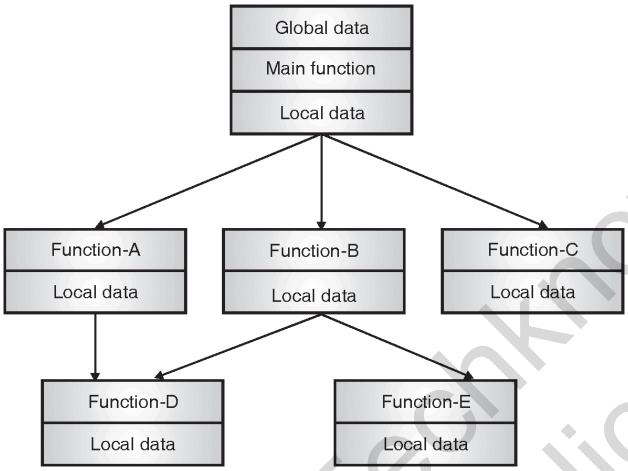


Fig. 1.4.1 : Structure of a procedure oriented programming language

- 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.

1.4.1 Introduction Object Oriented Programming (OOP) and OOP Languages

Q. Describe any four basic concepts of OOP. **(4 Marks)**

Q. List any four object oriented languages. **(2 Marks)**

- 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.

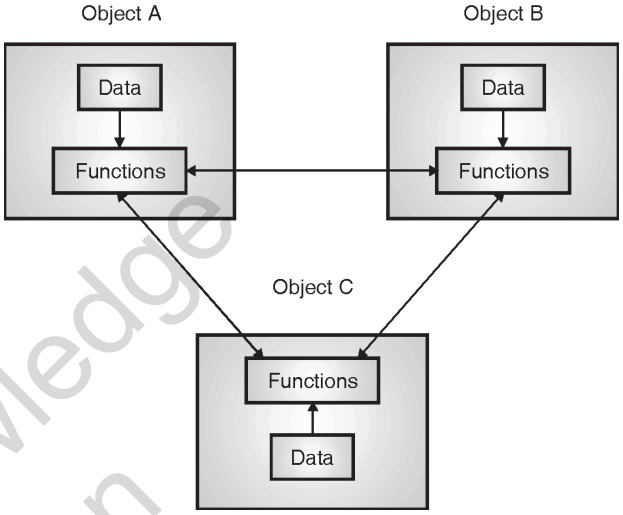


Fig. 1.4.2 : Structure of an Object Oriented Programming

- Fig. 1.4.2 shows the structure of a object oriented programming language.
- C++, Python, Ruby, C# and Java are examples of object oriented programming languages.

1.4.2 Limitations of Procedural Oriented Programming (POP) and Need of Object Oriented Programming (OOP)

SPPU - Dec. 16, Dec. 17

Q. Differentiate between Procedural Oriented Programming (POP) and Object Oriented Programming(OOP) (any four points) **(Dec. 16, 4 Marks, Dec. 17, 6 Marks)**

POP has certain issues that doesn't allow it to implement the real time problems. Some of the issues are limitations are:

1. Global data can be accessed and changed inadvertently.
2. Data structures cannot be secured from each other.
3. Changes to a data structure are to be implemented in all the functions that use the same.

4. Data can be accessed by all the functions and hence security of data from some users and access to others is not possible.

OOP finds wide applications in the IT industry as it models the real world problems quite preciecely. Also it resolves the above mentioned issues of POP. Let us see the differences of POP and OOP.

Table 1.4.1 : Differences between POP and OOP

Sr. No	Procedure Oriented Programming (POP) like 'C'	Object Oriented Programming (OOP) like 'Java'
1.	In this case emphasis is given on method or procedure.	In this case emphasis is on data or objects rather than procedure.
2.	Large programs are divided into smaller parts using functions.	Programs have classes and objects. Only the functions in a class can access the data in a given object.
3.	Data can be accessed by any function, and hence there is a scope of unexpected change of data	Data can be accessed only by functions of that class and hence it cannot be changed unexpectedly
4.	Global data is allowed to be accessed by all functions. Hence data is not hidden	No global data. Data is encapsulated
5.	If changes are done in some data, the corresponding changes are to be implemented in all the functions using the data.	Adding of new data or changes in the data requires to change only the functions of the class
6.	It follows top-down approach in program design.	It follows bottom-up approach in program design.
7.	It cannot model the real world problems precisely.	It is used to model the real world problems precisely

1.4.3 Applications of Object Oriented Programming Language

Q. Enlist the applications of OOP. **(4 Marks)**

OOP finds wide applications in the IT industry. OOP models the real world problems quite preciecely and hence it is more used today than the procedure oriented programming languages. The areas for application of OOP include :

- Real-time systems
- Simulation and modelling
- Object-oriented databases
- Hypertext, hypermedia and expertext
- AI and expert systems
- Neural networks and parallel programming
- Decision support and office automation systems
- CIM/CAM/CAD systems

1.5 Fundamentals of Object-Oriented Programming : Objects, Classes, Data Members, Methods

- 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	2. Object
3. Data abstraction	4. Data encapsulation
5. Inheritance	6. Polymorphism

1. Class

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

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 chapter 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 chapters.

1.6 Static and Dynamic Binding, Message Passing

- As discussed in the previous section, polymorphism is multiple functions or different things, but having same name.
- In this case how to decide, which of the function is to be called amongst the so many with same name?
- This process called as binding assigns a particular function to every call.
- If the binding is done during the compile time, it is called as static binding. All those operations happening during the compilation of a program are called as static.
- Similarly, if the binding is done during the execution, then it is called as dynamic binding or runtime binding.
- We will see this in detail in later chapters based on polymorphism.
- Message passing is used to pass information or data from one method or function to another.
- When we need to call another function or method to do a particular task, we need to pass the information of the data on which the operation is to be done.
- Also after performing the operation, the function may require to return some information.
- This is also called as parameters or argument passing from one method to another.
- In the subsequent chapters, we will also see how information is passed from one method to another.

1.7 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. 1.7.1).
- 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 1.7.1. 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.

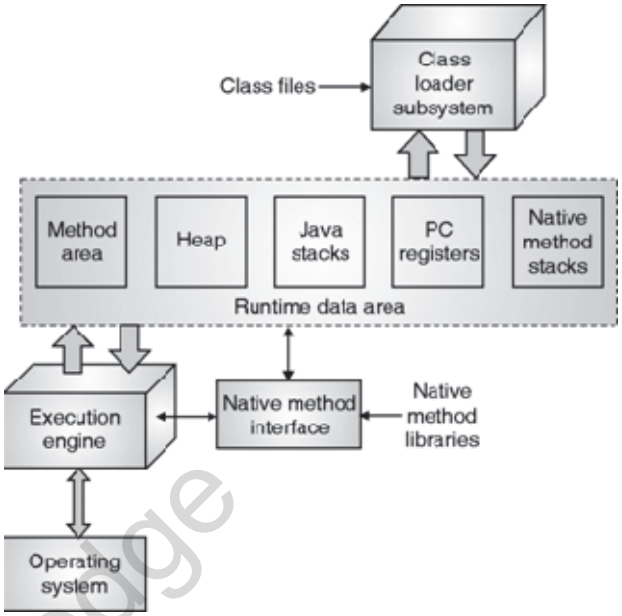


Fig. 1.7.1

- 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.

1.8 Formatted and Unformatted IO Functions of C++

– Inbuilt IO functions are available in C for displaying data on the standard output device i.e. monitor and accepting the input from the standard input device i.e. keyboard.

– These IO functions are classified as formatted and unformatted functions based on the formatting permitted or not. These are discussed in more details in the following sub sections.

1.8.1 Formatted IO Functions

– There are two formatted IO functions in C namely printf() to display a data on monitor and scanf() to accept data from keyboard.

1. printf()

Syntax: printf("Format string", list of variables
or expressions)

The format string can contain the following :

(a) Character set (A-Z, a-z, 0-9, all special symbols on the keyboard)

(b) Blank spaces

(c) Escape sequences

(d) Field width

The width of a value can be fixed by specifying the field width. Examples of such case are given below.

(e) Format specifiers

Format specifiers specify the format of the variable data whose value is to be displayed. The format specifiers for different types of data is given in the Table 1.8.1.

Table 1.8.1 : Format specifiers

Sr. No.	Format specifier	Data type
1.	%d or % i	For integer type of data
2.	%f	For float type of data
3.	%c	For char type of data
4.	%lf	For double type of data
5.	%Lf	For long double type of data

Examples:

1. printf("The number of buildings is %d",b);

   This statement will print the output with the format specifier replaced with the value of b.

2. printf("The simple interest is %f",si);

   This statement will print the output where the format specifier %f will be replaced with the float type value of the variable si.

3. printf("The simple interest is %5.2f",si);

   The format specifier in this statement is accompanied with a field width i.e. 5.2 (%5.2f), which indicates five digits before the fraction point and 2 digits after the fraction point. This statement will display the output wherein the value of si will be displayed with five digits before the fraction point and only two digits after the fraction point.

2. scanf()

Syntax: scanf("format String", address of variables);

Here, format string can contain format specifier, field width and assignment suppression character. The assignment suppression character (*) is used to discard one of the user entered value.

The address of the variable is obtained with the help of the address of operator (&).

Examples

1. scanf("%d",&x);

   This statement is used to accept aint type value from user in the variable x

2. scanf("%d %d %f",&x,&y,&z);

This statement is used to accept two int type data into variables x and y. It also accepts a float type data into the variable z.

1.8.2 Unformatted IO Functions

– The unformatted IO functions do not have any format specifier. They are mostly used to accept and display only one character or a string (string is a set of characters to make a word or sentence).

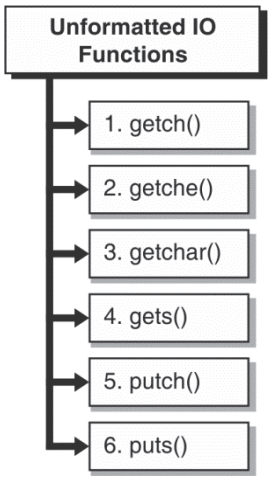


Fig. 1.8.1 :Unformatted IO Functions

– The different unformatted IO functions are listed below;

1. **getch():** This function is used accept one character from the user.
2. **getche():** This function is used accept one character from the user and echo it (display it on
the screen).
3. **getchar():** This function is used to accept one character from the user and echo it (display it on
the screen) and also wait after that for the user to press the enter key.
4. **gets():** This function is used to accept a string from the user. We will see in details of this when studying the chapter on strings.
5. **putch() :** These functions are used to display a character on the monitor. putchar().
6. **puts():** This function is used to display a string on the monitor.

1.9 Basic C++ Program Examples

Program 1.9.1 : Write a C++ program to accept a number and display its square.

```
#include<iostream.h>
#include<conio.h>
void main ()
{
   clrscr();
   int x, y;
   cout<<"Enter a number:";
   cin>>x;
   y=x*x;
   cout<<"The square of "<<x<<" is "<<y;
   getch();
}
```

Output

```
Enter a number:6
The square of 6 is 36
```

Explanation

– In this program another header file is included i.e. conio.h for the two functions used in the program namely clrscr() and getch().

– The function clrscr() is used to clear the user screen and the function getch() as already discussed accepts a character from user.

– Here, we use this function, so that the processor waits for us to enter the character before closing the output window after displaying the result.

– As the processor operation is very fast it displays the result on the output screen and closes the output screen, before we can see the output. Hence, to make the processor to wait for us to see the result we give a getch() function.

– The variables x and y are declared as integer type data.

– Using the first cout statement the user is asked to provide a number whose square is to be found.

- The cin operator is used to accept the input from the user. The statement cin is associated with the extraction operator (>>), which indicates that the cin statement extracts the input from the standard input device and provides it in the variable x. The standard input device is keyboard.
- The next statement assigns the value of x multiplied by itself to provide the square in the variable y.
- The next cout statement is said to have concatenated insertion operators. The insertion operator is used along with the cout statement to display a data on the monitor.

Program 1.9.2 : Write a C / C++ program to accept two numbers and display its product.

```
#include<iostream.h>
#include<conio.h>
void main ()
{
   clrscr();
   int x, y, z;
   cout<<"Enter two number:";
   cin>>x>>y;
   z=x*y;
   cout<<"The product is "<<z;
   getch();
}
```

Output

```
Enter two number:3
4
The product is 12
```

Explanation

- The two numbers are accepted in the variables x and y by concatenating the extraction operator in a single statement using cin.
- The product is calculated using the multiply operator (*).

Program 1.9.3 : Write a program to calculate the simple interest taking principal, rate of interest and number of years as inputs from user.

```
#include<iostream.h>
#include<conio.h>
void main ()
{
   clrscr();
   floatrateOfInterest, years, principal, simpleInterest;
   cout<<"Enter the principal, rate of interest and no. of years:";
   cin>>principal>>rateOfInterest>>years;
   simpleInterest = principal * rateOfInterest*years / 100;
   cout<<"The simple interest is "<<simpleInterest;
   getch();
}
```

Output

```
Enter the principal amount, rate of interest and no. of years:1000
9.5
10
The simple interest is 950
```

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.

Program 1.9.4 : Write a program to accept basic salary from the keyboard. Calculate the gross salary that includes basic salary, 50% DA and 40% HRA.

```
#include<iostream.h>
#include<conio.h>
void main ()
{
   clrscr();
   floatbasic,hra,da,gross;
   cout<<"Enter the basic salary:";
   cin>>basic;
   hra=40*basic/200;
   da = 50*basic/200;
   gross=basic+da+hra;
   cout<<"The total salary is "<<gross;
   getch();
}
```

Output:

```
Enter the basic salary:10000
The total salary is 19000
```

Explanation

- To calculate HRA we cannot write hra=40/100 * basic; as the division 40/100 will return the quotient of this division which is 0 and hence the hra will become 0.
- Once 40 is multiplied with basic the result is automatically type casted (upgraded) to the type of higher size i.e. float as one of the data was float (the variable basic is float type).

Program 1.9.5 : Write a program to swap two integers.

```
#include<iostream.h>
#include<conio.h>
void main ()
{
   clrscr();
   int a, b, temp;
   cout<<"Enter two numbers:";
   cin>>a>>b;
   temp=a;
   a=b;
   b=temp;
   cout<<"After the values are a="<<a<<" and b="<<b;
   getch();
}
```

Output

```
Enter two numbers:3
4
After swapping the values are a=4 and b=3
```

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

```
#include<iostream.h>
#include<conio.h>
void main ()
{
   clrscr();
   int a, b;
   cout<<"Enter a number to be shifted left:";
   cin>>a;
   b=a<<3;
   cout<<"After shifting times left the value is "<<b;
   getch();
}
```

Output:

```
Enter a number to be shifted left:5
```

After shifting three times left the value is 40

1.9.1 Type Casting

Program 1.9.7 : Write a program to accept one int type data and one float type data. Multiply the two numbers and display the result.

```
#include<iostream.h>
#include<conio.h>
void main ()
{
   clrscr();
   int n1;
   float n2, result;
   cout<<"Enter one integer and one float type each:";
   cin>>n1>>n2;
   result=n1*n2;
   cout<<"The product is :"<<result;
   getch();
}
```

Output

```
Enter one integer and one float type number each:2
3.9
The product is :7.8
```

Explanation

- The two number being multiplied are of int type and float type respectively.

– Hence the result is automatically type casted (upgraded) to the type of higher size i.e. float. Hence the result variable is taken as float.

Program 1.9.8 : Write a program to accept a float number and display the integer part using type casting operator.

```
#include<iostream.h>
#include<conio.h>
void main ()
{
   clrscr();
   int a;
   float b;
   cout<<"Enter a float number:";
   cin>>b;
   a=(int)b;
   cout<<"The integer part of the number is:"<<a;
   getch();
}
```

Output

```
Enter a float number:3.14
The integer part of the number is:3
```

Explanation

This program uses type casting operator to go to a lower data type size.

Program 1.9.9 : Write a program to accept a number and display its equivalent ASCII using type casting.

```
#include<iostream.h>
#include<conio.h>
void main ()
{
   clrscr();
   int a;
   cout<<"Enter a char:";
   cin>>a;
   cout<<"This is value of: "<<(char)a;
   getch();
}
```

Output

```
Enter a number:65
This is ASCII value of: A
```

Explanation

This program also uses type casting operator to go to a lower data type size.

1.10 Input / Output in Java

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

1.10.1 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.

– 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.

1.10.2 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 1.10.1.

Table 1.10.1 : List of Methods

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.

1.10.3 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. We will see more about wrapper classes in chapter 9 i.e. packages. 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.

1.11 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 1.11.1 : 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>javac HelloFriends.java

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

Explanation

- 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.

Program 1.11.2 : Define a Class Bank Account having data members and member functions as :

Data members :

1. Name of depositor
2. Account number
3. Type of account
4. Balance amount in the account

Members functions :

1. To assign initial values
2. To deposit an amount
3. To withdraw an amount after checking the balance
4. To display name and balance **May 17, 4 Marks**

```
import java.util.*;
class Account
{
   protected float bal,with,dep;
   protected int accno,areacode;
   public void enterBal()
   {
      Scanner sc=new Scanner(System.in);
      System.out.print("Enter your balance=");
      bal=sc.nextFloat();
   }
}
class Withdraw extends Account
{
   protected int dob,pin,mainpin;
   protected String name, surname,fullname;
   Scanner sc=new Scanner(System.in);
   public void accDetailsAccept()
   {
      System.out.println("Account Holder details:-");
      System.out.print("First name : ");
      name=sc.nextLine();
      System.out.print("Last name : ");
      surname=sc.nextLine();
      fullname=name+" "+surname;
      System.out.println("Account Number(5 digit)=");
      accno=sc.nextInt();
      System.out.print("Date Of Birth(DDMMYYYY):");
      dob=sc.nextInt();
      mainpin=dob;
      System.out.print("Area Code : ");
      areacode=sc.nextInt();
      System.out.println("\n\n");
   }
   public void dispData()
   {
      System.out.println("Account Holder details:-");
      System.out.print("Name : "+fullname);
      System.out.print("Account Number : "+accno);
      System.out.print("Date Of Birth  : "+dob);
      System.out.print("Area Code  : "+areacode);
   }
   public void checkBal()
   {
      System.out.println("Your Account balance is "+bal);
   }
   public void withdraw()
   {
      System.out.print("Enter amount to be withdrawn=");
      with=sc.nextFloat();
      if(with>bal)
      System.out.println("Insufficient funds.");
      else if(with>50000)
      System.out.println("Error: Prior Notice Not
                              Submitted.");
      else
      {
         System.out.println("Rupees"+with+"
                              withdrawn.\nYour new Account
                              Balance is "+(bal-with));
         bal=bal-with;
      }
   }
   public void deposit()
   {
      System.out.print("Enter amount to be deposited=");
      dep=sc.nextFloat();
      System.out.println("Your new Account Balance is
                              "+(bal+dep));
      bal=bal+dep;
   }
   public void enterPin()
   {
      do
      {
         System.out.print("Enter your PIN:");
         pin=sc.nextInt();
         if(pin==mainpin)
         {
            System.out.println("Welcome Back ");
            System.out.println();
            break;
         }
         else
         {
            System.out.println("Incorrect PIN. Try Again.");
            System.out.println();
            continue;
         }
      }while(pin!=mainpin);
```

```
    }
    public boolean search(int x)
    {
        if(accno==x)
        return true;
        else
        return false;
    }
}
class Deposit extends Account
{
    Scanner sc=new Scanner(System.in);
    public void Text()
    {
        System.out.println("******THAKUR WORLD
                            BANK******");
        System.out.println("WELCOME! PLEASE ENTER
                            YOUR DETAILS BELOW:");
        System.out.println("Enter 1 for if you are a manager,2 if
                            you are a Customer");
        System.out.println();
    }
    public void menuCustomer()
    {
        System.out.println("******___CUSTOMER MAIN
    MENU___******\n\nChoose One Option To Proceed.");
        System.out.println("---> 1. Add account\n---> 2.
Access Existing Account\n---> 3. Print Account Data\n--->
4. Exit Program");
        System.out.print("Enter your choice=");
        System.out.println();
    }
    public void menu2()
    {
        System.out.println("******___ACCOUNT
MENU___******\n\nChoose One Option To Proceed.");
        System.out.println("---> 1. Check Bank Balance\n--->
2. Withdraw Funds\n---> 3. Deposit Funds\n---> 4. Exit
Program");
    }
    public void menuManager()
    {
        System.out.println("******___MANAGER MAIN
MENU___******\n\nChoose One Option To Proceed.");
        System.out.println("---> 1. Delete account\n---> 2.
View Account Data \n---> 3. Exit Program");
        System.out.print("Enter your choice=");
        System.out.println();
    }
}
public class Bank
{
    public static void main(String args[])
    {
        Deposit d= new Deposit();
        Withdraw w[]= new Withdraw[100];
        Scanner sc=new Scanner(System.in);
        int
choice1,choice2,choice3,choice4,n=0,i=0,j=0,k=0,flag=
0,pin,mpin,an,del,disp;
        char ans1,ans2;
        do
        {
        d.Text();
        choice3=sc.nextInt();
        if(choice3==1)
        {
            do
            {
                System.out.println("Enter your pin:");
                mpin=sc.nextInt();
                if(mpin==123)
                {
                    System.out.println("Welcome Back ");
                    System.out.println();
                    break;
                }
                else
                {
                    System.out.println("Incorrect PIN. Try
                                        Again.");
                    System.out.println();
                    continue;
                }
            }while(mpin!=123);
            d.menuManager();
            choice4=sc.nextInt();
            switch(choice4)
            {
                case 1: if(n==0)
                    {
```

```
System.out.println("Add Account Holder Data First.");
                }
                else
                {
System.out.print("Enter account number of client to be deleted:");
                    del=sc.nextInt();
                    for(j=0;j<=n-1;j++)
                    {
                        if(w[j].search(del))
                        {
                            flag =1;
                            for(k=j;k<=n-2;k++)
                            {
                                w[k]=w[k+1];
                            }
                            n--;
                        }
                    }
                }
                if(flag==1)
                {
System.out.println("Account Holder Data has been deleted.");
                    flag=0;
                }
                else
System.out.println("Account Holder Data Does Not Exist.");
                break;
case 2 :System.out.println("Enter account number whose data is to be displayed:");
                disp=sc.nextInt();
                for(j=0;j<=n-1;j++)
                {
                    if(w[j].search(disp))
                    {
                        w[i].dispData();
                    }
                    else
System.out.println("Account Holder Data Does Not Exist.");
                }
                break;
case 3: System.out.println("Thank You! Have A Good Day!");
                break;
default:System.out.println("Incorrect option. Try Again.");
                break;
            }
        }
        if(choice3==2)
        {
            do
            {
                d.menuCustomer();
                choice1=sc.nextInt();
                switch(choice1)
                {
                    case 1: n++;
                    w[i]=new Withdraw();
                    w[i].accDetailsAccept();
                    i++;
                    break;
 case 2: System.out.println("Enter account number:");
                    an=sc.nextInt();
                    for(j=0;j<=n-1;j++)
                    {
                        if(w[j].search(an))
                        {
                            w[j].enterPin();
                            w[j].enterBal();
                            System.out.println();
                            do
                            {
                                d.menu2();
                                System.out.println();
System.out.print("Enter your choice=");
                                choice2=sc.nextInt();
                                System.out.println("\n");
                                switch(choice2)
                                {
                                    case 1: {
                                        w[j].checkBal();
                                        break;
                                    }

                                    case 2: {
                                        w[j].withdraw();
                                        break;
                                    }
```

```
                        case 3: {
                                w[j].deposit();
                                break;
                            }

                        case 4: {

  System.out.println("Thank You! Have A Good Day!");
                                break;
                            }

                        default:{

  System.out.println("Incorrect option. Try Again.");
                                break;
                            }
                    }
System.out.println("Do you wish to perform another operation?(Y/N)");
                    ans1=sc.next().charAt(0);
                    if(ans1!='y'||ans1!='Y')
                    System.out.println("Thank You For Using The System! Have A Great Day!");
                    }while(ans1=='y'||ans1=='Y');
                }
                else
                {
System.out.println("Account Does Not Exist.");
                    System.out.println();
                }
            }
            break;
case 3: System.out.println("Enter account number whose data is to be displayed:");
            disp=sc.nextInt();
            for(j=0;j<=n-1;j++)
            {
                if(w[j].search(disp))
                {
                    w[i].dispData();
                }
                else
System.out.println("Account Holder Data Does Not Exist.");
            }
            break;
case 4: System.out.println("Thank You! Have A Good Day!");
                    break;
default:System.out.println("Incorrect option. Try Again.");
                    break;
                }
            }while(choice1!=4);
        }
        else
System.out.println("Incorrect Option. The Program Has Been Terminated.");
System.out.println("Would you like to perform another operation?(Y/N)?");
        ans2=sc.next().charAt(0);
        if(ans2!='y'||ans2!='Y')
System.out.println("Thank You For Using The System! Have A Great Day!");
        }while(ans2=='y'||ans2=='Y');
    }
}
```

1.12 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".

1.12.1 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.

1.13 Solved Programs

Program 1.13.1 : 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\chapter 1\prog 1.13.1>java Square

Enter a number:
3
The Square=9
C:\java programs\chapter 1\prog 1.13.1>
```

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\chapter 1\prog 1.13.1>java Square

Enter a number:
3
The Square=9

C:\java programs\chapter 1\prog 1.13.1>
```

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.

– 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 1.13.2 : 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\chapter 1\prog 1.13.2>javac Product.java
C:\java programs\chapter 1\prog 1.13..2>java Product
Enter two numbers:
4
3
The Product=12
C:\java programs\chapter 1\prog 1.13.2>
```

Program 1.13.3 : 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\chapter 1\prog 1.13.3>javac SimpleInterest.java

C:\java programs\chapter 1\prog 1.13.3>java SimpleInterest
Enter Principal amount, no. of years and rate of interest:
1000
3
10
The Simple Interest=300.0
C:\java programs\chapter 2\prog  1.13.3>
```

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 1.13.4 : 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\chapter 1\prog 1.13.4>javac Rectangle.java

C:\java programs\chapter 1\prog 1.13.4>java Rectangle
Enter length and breadth of rectangle:
5
6.3
The Area=31.5
The Perimeter=22.6

C:\java programs\chapter 1\prog 1.13.4>
```

Program 1.13.5 : 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\chapter 1\prog  1.13.5>javac Larger.java
C:\java programs\chapter 1\prog 1.13.5>java Larger
Enter two numbers:
5
7
The Larger no.=7
C:\java programs\chapter 1\prog 1.13.5>
```

Program 1.13.6 : 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\chapter 1\prog 1.13.6>javac Circle.java

C:\java programs\chapter 1\prog 1.13.6>java Circle
Enter radius:
5
The Area=78.5

C:\java programs\chapter  1\prog 1.13.6>
```

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 1.13.7 : 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\chapter 1\prog 1.13.7>javac Swap.java
C:\java programs\chapter 1\prog 1.13.7>java Swap
Enter two numbers:
5
7
a=5
b=7
After Swapping:
a=7
b=5

C:\java programs\chapter 1\prog 1.13.7>
```

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 1.13.8 : 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\chapter 1\prog 1.13.8>javac Shift.java
C:\java programs\chapter 1\prog 1.13.8>java Shift
Enter a number:
4
After Shifting:
a=4
Result=32
C:\java programs\chapter 1\prog 1.13.8>
```

Program 1.13.9 : 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\chapter 1\prog 1.13.9>javac
BitwiseOperations.java

C:\java programs\chapter 1\prog 1.13.9>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\chapter 1\prog  1.13.9>
```

1.14 Revisiting Loops Statements

- 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.

1.14.1 Programs Based on for Loop

Program 1.14.1 : 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 hasreached 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 1.14.2 : 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 1.14.3 : 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 1.14.4 : 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 1.14.5 : 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 1.14.6 : 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 the value of i initialized to 0, and hence go upto n – 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 1.14.7 : 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 1.14.8 : 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 1.14.9 : 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 1.14.10 : 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 1.14.11 : 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 1.14.12 : 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 1.14.13 : Write a program to calculate the sine of an angle using the following series for x as the anglein 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 1.14.15 : 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 1.14.16 : 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.

1.14.2 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 1.14.17 : 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 1.14.18 : 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 1.14.19 : 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 1.14.20 : 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<=i;j++)
  {
   System.out.print("*");
  }
  System.out.println();
 }
}
}
```

Output

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

Program 1.14.21 : 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 1.14.22 : 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 thing is 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 1.14.23 : 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 just a blank space is to be added after every * of the previous program.
- Hence you will notice the previous program statement was

  System.out.print("*");

  while in this program the statement is replaced with

  System.out.print("* ");

Program 1.14.24 : Write a program to display the following asking the user for the number of lines in the upper half.

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

```
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 upper half:");
  n=sc.nextInt();
  for(i=1;i<=n;i++)
  {
    for(j=1;j<=n-i;j++)
    {
      System.out.print(" ");
    }
    for(j=1;j<=2*i-1;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<=2*i-1;j++)
    {
      System.out.print("*");
    }
    System.out.println();
  }
 }
}
```

Output

```
Enter number of * in the upper half:5
    *
   ***
  *****
 *******
*********
 *******
  *****
   ***
    *
```

Explanation

– In this program the operation is again similar to the program 3.2.22. The only difference being that here the number of stars (*) in each line is 2 * i -1, where i is the line number.

Program 1.14.25 : Write a program to display the following asking the user for the number of lines.

```
1
12
123
1234
12345
123456
```

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

Output

```
Enter number of lines:5
1
12
123
1234
12345
```

Explanation

– In this program the operation is almost similar to the pervious programs. Only the difference being that instead of stars, here we print the value of j itself. Hence the numbers are printed instead of stars.

Program 1.14.26 : 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 iis line number) to 1. For each of these three operations there is an inner for loop.

Program 1.14.27 : 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 1.14.28 : 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.

Program 1.14.29 : Write a program to display the following asking the user for the number of lines.

```
A
B C
D E F
G H I J
K L M N O
```

```
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((char)(k+64)+" ");
  }
  System.out.println();
 }
}
}
```

Output

```
Enter number of lines:6
A
B C
D E F
G H I J
K L M N O
P Q R S T U
```

Explanation

- There is only one change as compared to the previous program i.e. type casting. Here the numbers of previous program are type casted to char.
- Here the value of k is added to get the ASCII value of 'A' and also increment the value of k thereafter for the next character.

Program 1.14.30 : Write a program to display the following pattern.

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

```
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=n;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 lines:5
*****
 ****
  ***
   **
    *
```

Program 1.14.31 : Write a program to display the following pattern.

```
1
0 1
1 0 1
0 1 0 1
1 0 1 0 1
```

```
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;i<=n;i++)
  {
   for(j=0;j<=i-1;j++)
   {
    System.out.print((i+j)%2 + " ");
   }
   System.out.println();
  }
 }
}
```

Output

```
Enter number of lines:7
1
0 1
1 0 1
0 1 0 1
1 0 1 0 1
0 1 0 1 0 1
1 0 1 0 1 0 1
```

1.15 Revisiting Control Statements

- 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 :

```
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.

1.15.1 Programs using if-else Statement

Program 1.15.1 : 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 1.15.2 : 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 1.15.3 : 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 1.15.4 : Write a program to check if the entered number is Armstrong or not.

Note : A number is said to be Armstrong number if the sum of the cube of its digits is equal to the number itself. For e.g. 153, the sum of the cubes is $1^3 + 5^3 + 3^3 = 1+125 + 27 = 153$ i.e. the original number itself.

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

Output

```
Enter a number:153
Armstrong Number
```

Explanation

- The variable sum is initialized to 0, and then the cubes of the user entered number are added to this variable sum. The method of separating the digits is same as seen in quite a lot previous programs. But in this case the cube off the digit is taken and then added to the variable sum.
- Another variable called as copy is declared in this program to keep the copy of the user entered number as the number entered by the user will become 0 after dividing it by 10 in every iteration.

– Hence we would not have any copy of the original number. To avoid this problem the variable copy is declared and keeps the value of the user entered number.

Program 1.15.5 : 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 averagetime for a year to be 365.2425 days almost closest.

1.15.2 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 1.15.6 : 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.

1.16 Revisiting 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. 1.16.1 each of these locations shown in the Fig. 1.16.1 are of four bytes i.e. to store one integer type of data.

a[0]	1
a[1]	2
a[2]	3
a[3]	4
a[4]	5
a[5]	6
a[6]	7
a[7]	8
a[8]	9
a[9]	10

Fig. 1.16.1 : 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 1.16.1 : 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 1.16.2 : Write a program to accept 'n' integers from user into an array and display the average of these numbers.

```
import java.util.*;
class Average
{
public static void main(String args[])
{
  int n,i,sum=0;
  float avg;
  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=(float)sum/n;
  System.out.println("Average="+avg);
 }
}
```

Output

```
Enter no of elements:4
Enter a no:2
Enter a no:3
Enter a no:1
Enter a no:4
Average=2.5
```

Explanation

- The first loop as usual is to accept all the elements and the second for loop is to calculate the sum.
- The value of the variable sum is initially 0, and every element is added to it with the index of the array 'a' from 0 to n – 1.
- Finally this value is divided by the total number of elements i.e. 'n' to find the average. But before that the value of the integer variable 'sum' is type casted to float. This is done to take into consideration the fraction part after division.
- Finally the average is displayed.

Program 1.16.3 : Write a program to evaluate the value of the following series and display the result.

$$\sum_{i=1}^{n} x_i^2 - \left[\sum_{i=1}^{n} x_i\right]^2$$

```
import java.util.*;
class Sum
{
public static void main(String args[])
{
  int n,i,sum1=0,sum2=0,sum;
```

```
  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++)
  {
    sum1=sum1+a[i];
    sum2=sum2+a[i]*a[i];
  }
  sum=sum2-sum1*sum1;
  System.out.println("Sum="+sum);
 }
}
```

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
Sum=-170
```

Explanation

- The method to initialize and accept the elements from user is same as in previous programs.
- Note, the expression says elements from 1 to n, but we go from 0 to n – 1. This is because, the index of the elements in the array are from 0 to n-1 as already seen.
- The logic is very simple. The two term sums are calculated i.e. sum1 and sum2. These variables sum1 and sum2 are $\sum_{i=0}^{n} x_i^2$ and $\sum_{i=1}^{n} x_i$ respectively.
- These values are calculated in the "for" loop. Then the final result is calculated by subtracting the square of second term i.e. sum2 from the first term i.e. sum1. And this result is displayed.

Program 1.16.4 : Write a program to evaluate the value of the standard deviation (s.d.) and display the result.

$$\text{s.d.} = \sqrt{\frac{\sum_{i=i}^{n} (x_i - \bar{x})^2}{n}}$$

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.
- Also to calculate the average, we have already seen a program i.e. 1.16.2.
- 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 1.16.5 : Write a program in Java for fitting a straight line through a set of points (x_i, y_i), i = 1, 2, 3 .... n. The straight line equation is y = mx + c and the values of m and c are given by

$$m = \frac{n\sum (x_i\, y_i) - \left(\sum x_i\right)\left(\sum y_i\right)}{n\left(\sum x_i^{\,2}\right) - \left(\sum y_i\right)^2}$$

$$c = \frac{1}{n}\left(\sum y_i - m \sum x_i\right)$$

```
import java.util.*;
class Line
{
public static void main(String args[])
{
  int n,i,xi=0,yi=0,xiyi=0,xi2=0;
  float m,c;
  Scanner sc = new Scanner (System.in);
  System.out.print("Enter no of points:");
  n=sc.nextInt();
  int x[]=new int [n];
  int y[]=new int [n];
  for(i=0;i<=n-1;i++)
  {
    System.out.print("Enter x co-ordinate:");
    x[i]=sc.nextInt();
    System.out.print("Enter y co-ordinate:");
    y[i]=sc.nextInt();
  }
  for(i=0;i<=(n-1);i++)
  {
    xi=xi+x[i];
    yi=yi+y[i];
    xiyi=xiyi+x[i]*y[i];
    xi2=xi2+x[i]*x[i];
  }
  m =(float)(n*xiyi-xi*yi)/(n*xi2-yi*yi);
  c=(yi-m*xi)/n;
  System.out.println("Equation of straight line is
   y="+m+"x+"+c);
 }
}
```

Output

```
Enter no of points:3
Enter x co-ordinate:1
Enter y co-ordinate:1
Enter x co-ordinate:3
Enter y co-ordinate:3
Enter x co-ordinate:5
Enter y co-ordinate:5
Equation of straight line is y=1.0x+0.0
```

Program 1.16.6 : 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 1.16.7 : 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 1.16.8 : 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 1.16.9 : Write a program to find and display the reverse of an array into the same array.

```
import java.util.*;
class Reverse
{
public static void main(String args[])
{
  int n,i,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-1)/2;i++)
  {
    temp=a[n-i-1];
    a[n-1-i]=a[i];
    a[i]=temp;
  }
  for(i=0;i<=n-1;i++)
  {
    System.out.println(a[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 know how to swap two numbers. Here the first and last elements of the array are swapped first. Then the second and second last element and so on.
- The centre two elements are the last elements to be swapped, hence the value of "i" is taken only up to (n–1)/2.

Program 1.16.10 : 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 1.16.11 : 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. 1.16.2.
- You will notice in Fig. 1.16.2, after all the comparisons done in the first iteration; the largest number has reached to the last position. But thenumbers 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
	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. 1.16.2 : Bubble sorting example

– You may also notice in the Fig. 1.16.2, 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. 1.16.3.

– 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
	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. 1.16.3 : Worst case condition of sorting numbers using Bubble sorting

Program 1.16.12 : Write a program to sort numbers in descending order. OR Write a program to implement bubble sorting algorithm for sorting numbers in descending order.

```
import java.util.*;
class Descend
{
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
85
9
6
4
1
```

Explanation

This is implemented using the same logic as in the previous program i.e. bubble sorting. The only difference here is that the numbers are swapped if the first number is smaller than the consecutive next one. The remaining logic is entirely same.

1.16.1 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. 1.16.2.

	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. 1.16.4 : Arrangement of the elements in a two dimensional array

- As seen in the Fig. 1.16.4, 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. 1.16.5 explains it all, but the elements of this matrix are stored in a different manner in the memory as shown in Fig.1.16.3. 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. 1.16.5 : Arrangement of the elements in a two dimensional array

- As shown in Fig. 1.16.5, 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 1.16.13

  (**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 1.16.13 : 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 1.16.14 : Write a program to find and display the sum of the diagonal elements of a square matrix.

```
import java.util.*;
class Diagonal
{
public static void main(String args[])
{
  int m,n,i,j,sum=0;
  Scanner sc = new Scanner (System.in);
  System.out.print("Enter no of rows / columns:");
  m=sc.nextInt();
  n=m;
  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++)
   {
     if(i==j)
     sum=sum+a[i][j];
   }
  }
  System.out.println("Sum"+sum);
 }
}
```

Output

```
Enter no of rows and columns:3
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
Enter a no:7
Enter a no:8
Enter a no:9
Sum=15
```

Explanation

- The method to accept the elements of the matrix is same as the previous program.
- The next nested "for" loop is for finding the sum of diagonal elements of the matrix.
- To check if it is a diagonal element, we check whether the row number and column number are equal. Because row number and column number are equal only for diagonal elements.

Program 1.16.15 : Write a program to add two matrices of size m × n.

```
import java.util.*;
class Sum
{
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];
  int b[][]=new int [m][n];
  int c[][]=new int [m][n];
  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<=m-1;i++)
  {
   for(j=0;j<=n-1;j++)
   {
     System.out.print("Enter a no:");
     b[i][j]=sc.nextInt();
   }
  }
  for(i=0;i<=m-1;i++)
  {
   for(j=0;j<=n-1;j++)
   {
     c[i][j]=a[i][j]+b[i][j];
   }
  }
  System.out.println("Sum Matrix");
  for(i=0;i<=m-1;i++)
  {
   for(j=0;j<=n-1;j++)
   {
     System.out.print(c[i][j]+"\t");
   }
   System.out.println();
  }
 }
}
```

Output

```
Enter no of rows and columns: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
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
Sum Matrix
2    4    6
8    10   12
```

Explanation :

- The method to accept the elements of the matrix is same as the previous program.
- The corresponding elements of the matrices "a" and "b" are added together and the result is put into the matrix "c". This "c" matrix is then displayed in natural form.

Program 1.16.16 : 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:23
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 1.16.17 : Write a program to find the transpose of a square matrix.

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

Output

```
Enter value of m: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
Enter a no:7
Enter a no:8
Enter a no:9
Transpose Matrix
1    4    7
2    5    8
3    6    9
```

Explanation :

- The method to accept the elements of the matrix is same as the previous program.
- For a square matrix, you will notice that the transpose can be obtained by swapping certain elements as shown in Fig. 1.16.6.

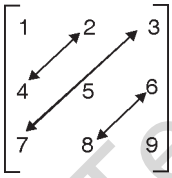


Fig. 1.16.6 : Transpose of a square matrix

- Here, the elements are swapped since it is a square matrix. Care is to be taken that for every new row; the swapping is to be started from the element after the swapped ones. For example, the first element of the second row is already swapped with the second element of first row while going along the first row. Hence, in the second row swapping has to begin from the second element and not the first one. Similarly, for the third row, we need to start from the third element, for the fourth row from fourth element and so on.

Program 1.16.18 : 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. 1.16.7 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. 1.16.7 : Multiplication of matrices

- Hence, in the Fig. 1.16.7 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.

Program 1.16.19 : The annual examination results of 5 students are tabulated as follows:

Roll No	Subject 1	Subject 2	Subject 3

Write a program to read the data and determine the following :

(i) Total Marks obtained by each student

(ii) The student who obtained the highest total marks.

```
import java.util.*;
class Exam
{
public static void main(String args[])
{
  int i,j,largeroll,largetotal;
  Scanner sc = new Scanner (System.in);
  int a[][]=new int [5][5];
  for(i=0;i<=4;i++)
  {
      System.out.print("Enter roll no and marks in three
                        subjects:");
      a[i][0]=sc.nextInt();
      a[i][1]=sc.nextInt();
      a[i][2]=sc.nextInt();
      a[i][3]=sc.nextInt();
      a[i][4]=a[i][1]+a[i][2]+a[i][3];
  }
  System.out.println("Roll NO\tSub 1\tSub 2\tSub
   3\tTotal\n");
  for(i=0;i<=4;i++)
  {
    for(j=0;j<=4;j++)
    {
       System.out.print(a[i][j]+"\t");
    }
    System.out.println();
  }
  largeroll=a[0][0];
  largetotal=a[0][4];
  for(i=0;i<=4;i++)
  {
       if(a[i][4]>largetotal)
       {
         largeroll=a[i][0];
         largetotal=a[i][4];
       }
  }
  System.out.println("Highest total marks is:" +largetotal
  +" and obtained by student with roll number:"+largeroll);
}
}
```

Output

```
Enter roll no and marks in three subjects:1
90
90
90
Enter roll no and marks in three subjects:2
90
98
99
Enter roll no and marks in three subjects:3
97
89
90
Enter roll no and marks in three subjects:4
90
90
97
Enter roll no and marks in three subjects:5
98
98
95
Roll NO Sub 1   Sub 2   Sub 3   Total
1       90      90      90      270
2       90      98      99      287
3       97      89      90      276
4       90      90      97      277
5       98      98      95      291
Highest total marks is:291 and obtained by student with roll
number:5
```


1.16.2 Arraycopy()

- This static method is available in the class "System" of the package java.lang. This method is used to copy an array from another.
- The speciality of this method is that you can select the starting location of source and destination array, and also the number of elements can be selected.
- The syntax of the method is as given below :

```
System.arraycopy(source_array_name,starting_element_ind
ex_of_source_array,
destination_array_name,
destination_array_starting_element,
number_of_elements_to_be_copied)
```

For e.g. System.arraycopy(a,0,b,1,a.length – 1);

will copy a total of n-1 elements, where n is the size of array 'a'.

- The elements will be copied beginning from the element number '0' of array 'a' into the array 'b' beginning from index '1' in the array 'b'.

Note : "length" is a variable of the array that returns the length of the array.

A program demonstrating the use of arraycopy() method is shown in Program 1.16.20.

Program 1.16.20 : Write a program to demonstrate the use of arraycopy() method.

```
import java.util.*;
class ArrayCopy
{
public static void main(String args[])
{
  int n,i;
  Scanner sc = new Scanner (System.in);
  System.out.print("Enter number 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();
  }
  System.arraycopy(a,0,b,0,a.length);
  for(i=0;i<=n-1;i++)
  {
    System.out.print(b[i]+"\t");
  }
  System.out.println();
  System.arraycopy(a,0,b,1,a.length-1);
  for(i=0;i<=n-1;i++)
  {
    System.out.print(b[i]+"\t");
  }
  System.out.println();
  System.arraycopy(a,1,b,0,a.length-1);
  for(i=0;i<=n-1;i++)
  {
    System.out.print(b[i]+"\t");
  }
 }
}
```

Output

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

Explanation

- The first time array 'a' is copied as it is in the array 'b' and then displayed
- The second case array 'a' is copied from element number '0' onwards into array 'b' from element number '1' onwards. The number of elements copied in this case is n-1 and then the array 'b' is displayed.
- The third case array 'a' is copied from element number '1' onwards into array 'b' from element number '0' onwards. The number of elements copied in this case is also n-1 and then the array 'b' is displayed.

1.17 Revisiting 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.

1.17.1 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.

Table 1.17.1 : Methods of string class

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.
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 1.17.1 : 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 1.17.2 : 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 1.17.3 : 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 1.17.4 : 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 1.17.5 : Write a Java program to convert a string to an array of characters and display each character with its index.

```
import java.util.*;
class CharArray
{
public static void main(String args[])
{
  int i,n;
  String str;
  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++)
  {
    System.out.println(i+"\t"+c[i]);
  }
 }
}
```

Output

```
Enter a String:
This is a chair.
0    T
1    h
2    i
3    s
4
5    i
6    s
7
8    a
9
10   c
11   h
12   a
13   i
14   r
15   .
```

Explanation

- The string is converted to an array of characters using the toCharArray() method.
- This array is then displayed with the index number i.e. the value of 'i' and the value of i^{th} index of the array in a tabular form.

Program 1.17.6 : Write a Java program to count the number of vowels in a string.

```
import java.util.*;
class Vowels
{
public static void main(String args[])
{
  int count=0,i,n;
  String str;
  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')
     count++;
  }
  System.out.println(count+" Vowels");
 }
}
```

Output

```
Enter a String:
There are 20 benches
6 Vowels
```

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. Finally the value of the count is displayed to indicate the number of vowels.

Program 1.17.7 : 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 1.17.8 : Write a Java program to count the number of upper case, lower case, blank spaces and digits in a string.

```
import java.util.*;
class Count
{
public static void main(String args[])
{
  String str;
  int countu=0,i,n,countd=0,counts=0,countl=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]<='Z')
    countu++;
    else if(c[i]==' ')
    counts++;
    else if(c[i]>='0' && c[i]<='9')
    countd++;
    else if(c[i]>='a' && c[i]<='z')
    countl++;
  }
  System.out.println(countu+" Upper case alphabets\n"+counts+" spaces\n"+countd+" Digits\n"+countl+" Lower case alphabets");
 }
}
```

Output

```
Enter a String:
There are 20 benches
1 Upper case alphabets
3 spaces
2 Digits
14 Lower case alphabets
```

Explanation

- The string is converted to an array of characters using the toCharArray() method.
- Each element of the array is then compared with lower case alphabets, spaces, digits and upper case alphabets as in the previous program. Every time one of the things is found the corresponding counter is incremented.
- Finally these counts are displayed.

Program 1.17.9 : 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".

1.17.2 Methods in String Buffer Class

- We have already seen two methods of making a string in Java. The first method was an object of class String and the other method is an array of char type data.
- Another way of making a string in Java is making an object of StringBuffer class. One major speciality of the object of this class is that it is mutable.
- The size of this object can be changed again and again during runtime i.e. we can insert, append, delete etc a character from the string.
- An object of StringBuffer has some advantages of extra methods provided in the class. Let us see the methods supported in this class.

Method	Operation
capacity()	This method returns the current capacity of the StringBuffer object.
length()	This method returns an integer value equal to the length of the string
setCharAt(int,char)	The character passed is set at the index specified in the brackets
charAt(int)	The method returns the character at the index mentioned in the brackets.
toString()	Converts the string buffer data into a string
insert(int,char)	Inserts the character passed at the index passed.
delete(int,int)	Deletes the substring from the starting to ending indices passed to the method
replace(int,int,String)	Replaces the string passed into the original string from the starting to ending indices passed to the method.
reverse()	Reverses the original string
append(String)	Appends the string passed to the original string.

– In the string buffer class we have the reverse method. Let us see a program to check palindrome using string buffer methods.

Program 1.17.10 : Write a Java program to check if the string is palindrome or not using StringBuffer object.

```
import java.util.*;
class Palindrome
{
public static void main(String args[])
{
  String str,rev;
  StringBuffer str1=new StringBuffer();
  Scanner sc = new Scanner (System.in);
  System.out.println("Enter a String: ");
  str=sc.nextLine();
  str1.append(str);
  str1.reverse();
  rev=str1.toString();
  if(str.equalsIgnoreCase(rev))
  System.out.println("Palindrome");
  else
  System.out.println("Not Palindrome");
 }
}
```

Output

```
Enter a String:
nitIN
Palindrome
```

Explanation

- The string is converted to an object of class StringBuffer using the append() method i.e. a string is appended to an object of StringBuffer.
- Then this object of StringBuffer is reversed and converted back to an object of class String using the toString() method which returns a string object of the string in a StringBuffer object.
- The two strings are compared ignoring the case and if they are equal it displays “Palindrome” else it displays “Palindrome”.

❑❑❑

Classes, Objects and Methods

Syllabus

Class : Creating a Class, Visibility/Access Modifiers, Encapsulation, Methods: Adding a Method to Class, Returning a Value, Adding a Method That Takes Parameters, The 'this' Keyword, Method Overloading, Object Creation, Using Object as a Parameters, Returning Object, Array of Objects, Memory Allocation: 'new', Memory Recovery: 'delete', Static Data Members, Static Methods, Forward Declaration, Class as Abstract Data Types (ADTs), Classes as Objects.

2.1 Comparison of Procedure Oriented Programming and Object Oriented Programming

Q. What is object oriented programming? How it is different from procedure oriented programming? **(4 Marks)**

Q. Differentiate between structure and class. **(4 Marks)**

Q. What are the differences between structure and class ? **(4 Marks)**

Q. What is difference between procedure oriented programming and object oriented programming ? **(4 Marks)**

- As we have discussed in Chapter 1, procedure oriented programming gives significance to procedure i.e. how to do a task.
- The structure of a procedure oriented programming as discussed in chapter 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.
- 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 on the other hand gives more importance to data rather than a procedure.

– The structure of object oriented programs as discussed in chapter 1, is also shown in Fig. 2.1.1.

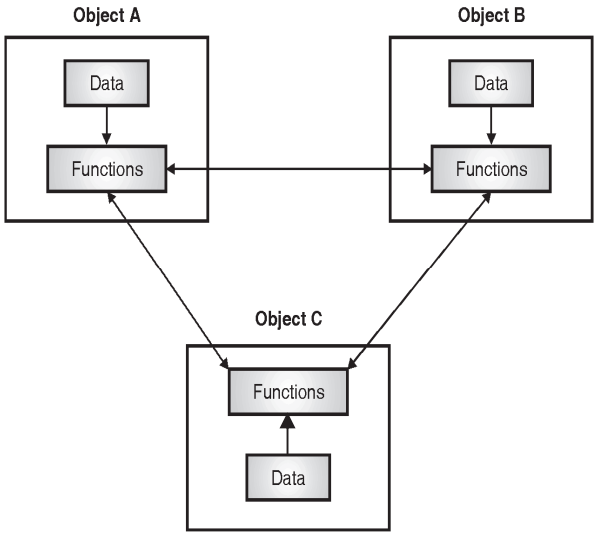


Fig. 2.1.1 : Structure of an object oriented program

Difference between structure and class

Sr. No.	Class	Structure
1.	Class is a reference type and its object is created on the heap memory.	Structure is a value type that is why its variable is created on the stack memory.
2.	Class can inherit the another class.	Structure does not support the inheritance.
3.	Class can have the all types of constructor and destructor.	Structure doesn't have constructors or destructors
	The member variable of class can be initialized directly.	The member variable of structure can not be initialized directly.

2.1.1 Class and Object : Introduction

SPPU - Dec. 16, May 17, Dec. 18

Q. What is object-oriented programming ? List basic concepts of it ? Explain each concept. **(10 Marks)**

Q. Explain the benefits of object oriented programming. **(8 Marks)**

Q. Define the following terms :
(1) Object (2) Class **(2 Marks)**

Q. Define class with it's syntax. **(2 Marks)**

Q. Write down characteristics of object oriented programming. **(4 Marks)**

Q. Define the terms polymorphism, data abstraction. **(Dec. 16, 4 Marks)**

Q. Explain various features of Object Oriented Programming. **(May 17, 4 Marks, Dec. 18, 6 Marks, May 19, 6 Marks)**

– 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. 2.1.1. As shown in Fig. 2.1.1, 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. 2.1.1.

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.

– There are few important features 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
2. Object
3. Data abstraction
4. Data encapsulation
5. Inheritance
6. Polymorphism

1. Class

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

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

2.1.2 Introduction to Objects

– An object is an instance or an example of a class. For example if "HumanBeing" 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.

2.1.2(A) 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.

2.2 Visibility / Access Modifiers

2.2.1 Access Specifiers in C++

- In C++, we have 3 access specifiers viz. public, protected and private.
- These access specifiers are used to specify the access of the data and function members of a class
- As the name says, access specifier will indicate who can access the data or function.
- A "public" access specifier indicates that every function can access this member.
- A "protected" access specifier indicates that only classes derieved can use this member. A derieved class concept will be dealt later in the topic called as Inheritance.
- A "private" access specifier indicates that only the functions of the same class can access the member.

2.2.2 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 2.2.1, shows the scope of access of the members of a class.

Table 2.2.1 : 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

2.3 Encapsulation

- The feature of keeping the data secured from functions outside the class is called as encapsulation.
- The data when declared as private, cannot be accessed by any method outside the class.
- The external method if requires access to the data members of a class, needs to take the help of internal methods of the class.
- This ensures encapsulation of the data of a class.
- In the following sections we will see the implementation of the same.

2.4 Creating a Class and Adding Methods to a Class

2.4.1 Defining Member Functions of a Class

SPPU - Dec. 17

Q. How many ways we can define member function in class ? Give it's syntax. **(4 Marks)**

Q. What do you mean by dynamic initialization of object ? **(Dec. 17, 4 Marks)**

- To define the data members of a class the logic is similar to that of the data members of a structure. But the new thing as discussed some time back is the member functions of a class. Let us see how this can be done and what are the different methods to do this.
- There are three methods of defining member functions of a class. They are :

1. Internally defined Functions
2. Externally defined functions
3. Inline functions

- Let us see each of these in details.

2.4.2 Internally Defined Functions

Q. Explain syntax for declaring the function inside the class and outside the class with example. **(4 Marks)**

- When the member functions of a class are defined inside the class itself they are called as internally defined member functions.
- Let us see some examples of object oriented programs with internally defined functions. We will see these examples for most of the concepts so as to make the understanding of the concept simple.
- Dynamic initialiazation means initializing the values of data members of the object during runtime or during the execution time. Program 2.4.1 shows dynamic initialization of object

Program 2.4.1 : Write a C++ program to find area of circle using Object Oriented Programming such that the class circle must have three member functions namely : (a) read() to accept the radius from the user (b) compute() for calculating the area (c) display() for displaying the result.

```
# include<iostream.h>
# include<conio.h>
class circle
{
   float r,a;
   public:
   void read()
   {
      cout<<"Enter radius:";
      cin>>r;
   }
   void compute()
   {
```

```
    a=3.14*r*r;
  }
  void display()
  {
    cout<<"Area="<<a;
  }
};
void main()
{
  clrscr();
  circle c;
  c.read();
  c.compute();
  c.display();
  getch();
}
```

Output

```
Enter radius:5
Area=78.5
```

Explanation

- A class is created with the name 'circle'. Two float type variables are declared in it namely 'r' and 'a' for the radius and area respectively.
- The three functions are written inside the class 'circle' namely read(), compute() and display().
- The read() function accepts the value of radius from the user. The compute() function calculates the area of the circle. The display() function displays the calculated area.
- But all these functions will perform these tasks only when they are called. These functions can be called by the period operator used with an object of this class.
- Hence, an object of this class is created in the main function. The syntax of making an object of a class is very simple.
- Syntax of declaring an object of a class:

```
Class_name Object_name;
```

- For example, in the above case we have made an object of the class "circle" by the statement,

```
circle c;
```

- Thereafter; we have called one by one each of the functions of the class 'circle' in the required sequence. To call the function of the class we have used the period operator('.'). The syntax of accessing a member function of a class is as shown below:

```
Object_name.Function_name();
```

- For example, in the above case to call the function read(), we have the statement,

```
c.read();
```

- Similarly; the compute() function and the display() function are called.
- Hence, the read() function accepts the input from user as seen in the output and the display() function displays the result calculated by the compute() function.

Program 2.4.2 : Write a C++ program to calculate the value of the following series using internal member function :

S = 12 + 22 + 32 + 42..... + n2

```
# include<iostream.h>
# include<conio.h>
class series
{
  int n,i,sum;
  public:
  void read()
  {
    cout<<"Enter the value of n:";
    cin>>n;
  }
  void compute()
  {
    for(i=1, sum = 0;i<=n;i++)
    {
      sum=sum+i*i;
    }
  }
  void display()
  {
    cout<<"Value of the series="<<sum;
  }
};
void main()
{
```

```
    clrscr();
    series s;
    s.read();
    s.compute();
    s.display();
    getch();
}
```

Output

```
Enter the value of n:5
Value of the series=55
```

Explanation

Note : A variable cannot be initialized in the declaration of a class member variable. It has to be initialized in a function only. Hence, the variable "sum" is not initialized in the declaration statement; instead it is initialized to zero in the compute function.

- The logic and way of writing the program is same as the previous one.

Program 2.4.3 : Write a C++ program to define a class student having data members name and roll no. Accept and display data for one object.

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class Student
{
   private:
   char name[20];
   int roll;
   public:
   void accept()
   {
      cout<<"Enter name and roll number:";
      cin>>name>>roll;
   }
   void display()
   {
      cout<<"Name:"<<name<<"\nRoll
number:"<<roll<<endl;
   }
};
void main()
{
   clrscr();
   Student s;
   s.accept();
   s.display();
   getch();
}
```

Output

```
Enter name and roll number:Ajay
24
Name:Ajay
Roll number:24
```

Program 2.4.4 : Write a C++ program to declare a class 'student' having data members as name and percentage. Write a constructor to initialize these data members accept and display data for one student.

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class Student
{
   private:
   char name[20];
   int percentage;
   public:
   void accept()
   {
      cout<<"Enter name and percentage:";
      cin>>name>>percentage;
   }
   void display()
   {

   cout<<"Name:"<<name<<"\nPercentage:"<<percent
age<<endl;
   }
};
void main()
{
   clrscr();
   Student s;
   s.accept();
   s.display();
   getch();
}
```

Output

```
Enter name and percentage:Ajay
85
Name:Ajay
Percentage:85
```

2.4.2(A) Nesting of Member Function

Q. What is nesting of member function ? Give one example. **(4 Marks)**

Q. Give syntax for defining a member function nesting of functions in a class with example. **(4 Marks)**

Program 2.4.5 : A member function of a class calling another member function of the same class is called as nesting of member function. The following C++ program example demonstrates this concept.

```
# include<iostream.h>
# include<conio.h>
class series
{
   int n,i,sum;
   public:
   void read()
   {
      cout<<"Enter the value of n:";
      cin>>n;
   }
   int compute()
   {
      for(i=1, sum = 0;i<=n;i++)
      {
         sum=sum+i*i;
      }
      return sum;
   }
   void display()
   {
      compute();
      cout<<"Value of the series="<<sum;
   }
};
void main()
{
   clrscr();
   series s;
   s.read();
   s.display();
   getch();
}
```

Output

```
Enter the value of n:5
Value of the series=55
```

2.4.3 Externally Defined Functions

Q. How function is defined outside of class, write general syntax and example of same. **(4 Marks)**

Q. Write general form of member function, definition out of class. **(2 Marks)**

Q. How user can declare member function outside the class ? **(2 Marks)**

- When the member functions of a class are defined outside the class they are called as externally defined member functions.
- The functions when defined outside must use scope resolution operator to define the scope of the function to be within the class. The syntax of defining a function outside the class using the scope resolution operator is as given below:

```
Return_type Class_name :: Function_name( argument list )
{
   -
   Statements;
   -
}
```

- Also in this case the prototype declaration has to be done inside the class for the externally defined functions.
- Let us see these in details.

Program 2.4.6 : Write a C++ program to find area of circle using Object Oriented Programming such that the class circle must have three externally defined member functions namely : (a) read() to accept the radius from the user (b) compute() for calculating the area (c) display() for displaying the result.

```
# include<iostream.h>
# include<conio.h>
class circle
{
```

```
    float r,a;
    public:
    void read();
    void compute();
    void display();
};
void circle::read()
{
    cout<<"Enter radius:";
    cin>>r;
}
void circle::compute()
{
    a=3.14*r*r;
}
void circle::display()
{
    cout<<"Area="<<a;
}
void main()
{
    clrscr();
    circle c;
    c.read();
    c.compute();
    c.display();
    getch();
}
```

Output

```
Enter radius:5
Area=78.5
```

Explanation

- A class is created with the name 'circle'. Two float type variables are declared in it namely 'r' and 'a' for the radius and area respectively.
- The prototype of the three functions are declared inside the class "circle" namely read(), compute() and display().
- The definition of these functions are written outside the class. But, with the help of the scope resolution operator the scope of the functions is declared in the class "circle".
- The read() function accepts the value of radius from the user. The compute() function calculates the area of the circle. The display() function displays the calculated area.
- But all these functions will perform these tasks only when they are called. These functions can be called by the period operator used with an object of this class.
- Hence, an object of this class is made in the main function.
- Thereafter we have called one by one each of the functions of the class "circle in the required sequence.

Program 2.4.7 : Write a C++ program to calculate the value of the following series using external member function :

S = 12 + 22 + 32 + 42 ..... + n2

```
# include<iostream.h>
# include<conio.h>
class series
{
    int n,i,sum;
    public:
    void read();
    void compute();
    void display();
};
void series::read()
{
    cout<<"Enter the value of n:";
    cin>>n;
}
void series::compute()
{
    for(i=1, sum = 0;i<=n;i++)
    {
        sum=sum+i*i;
    }
}
void series::display()
{
```

```
    cout<<"Value of the series="<<sum;
}
void main()
{
    clrscr();
    series s;
    s.read();
    s.compute();
    s.display();
    getch();
}
```

Output

```
Enter the value of n:5
Value of the series=55
```

Explanation

Note : A variable cannot be initialized in the declaration of a class member variable. It has to be initialized in a function only. Hence, the variable 'sum' is not initialized in the declaration statement; instead it is initialized to zero in the compute function.

- The logic and way of writing the program is same as the previous one.

2.4.4 Inline Member Functions

Q. What are inline functions ? Write a program to demonstrate inline function. **(10 Marks)**

Q. Write short note on : Inline function. **(5 Marks)**

Q. Explain inline function with an example. **(10 Marks)**

Q. What do you mean by inline function? Write its syntax and example. **(4 Marks)**

- We have seen that a function can be defined inside a class or outside a class.
- If a function is to be defined outside the class, but still be treated as a internally defined function then such a function has to be made "inline" function.
- To make a function inline function the keyword 'inline' before the header line of the function.
- The syntax of defining a inline function of a class is as given below:

```
inline Return_type Class_name :: Function_name(
argument list )
{
    -
    Statements;
    -
}
```

- Also in this case the prototype declaration has to be done inside the class for the inline functions.
- The advantage of inline function is that the execution is faster even though the functions are defined outside. Also another advantage is a member can be declared inline for two classes and hence access members of two classes.
- Let us see the same examples as seen inline defined functions.

Program 2.4.8 : Write a C++ program to find area of circle using Object Oriented Programming such that the class circle must have three inline functions namely : (a) read() to accept the radius from the user (b) compute() for calculating the area (c) display() for displaying the result.

```
# include<iostream.h>
# include<conio.h>
class circle
{
    float r,a;
    public:
    void read();
    void compute();
    void display();
};
inline void circle::read()
{
    cout<<"Enter radius:";
    cin>>r;
}
inline void circle::compute()
{
    a=3.14*r*r;
}
inline void circle::display()
{
```

```
    cout<<"Area="<<a;
}
void main()
{
    clrscr();
    circle c;
    c.read();
    c.compute();
    c.display();
    getch();
}
```

Output

```
Enter radius:5
Area=78.5
```

Program 2.4.9 : Write a C++ program to calculate the value of the following series using inline member function : S = 12 + 22 + 32 + 42 ..... + n2

```
# include<iostream.h>
# include<conio.h>
class series
{
    int n,i,sum;
    public:
    void read();
    void compute();
    void display();
};
inline void series::read()
{
    cout<<"Enter the value of n:";
    cin>>n;
}
inline void series::compute()
{
    for(i=1, sum = 0;i<=n;i++)
    {
        sum=sum+i*i;
    }
}
inline void series::display()
{
    cout<<"Value of the series="<<sum;
}
void main()
{
    clrscr();
    series s;
    s.read();
    s.compute();
    s.display();
    getch();
}
```

Output

```
Enter the value of n:5
Value of the series=55
```

2.4.5 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 2.4.10 : Write a C++ 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 2.4.11 : Write C++ 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 2.4.12 : Using C++ 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 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 2.4.13 : Using C++ 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();
  str=sc.nextLine();
  id=sc.nextInt();
  str=sc.nextLine();
  p=sc.nextInt();
  str=sc.nextLine();
  c=sc.nextInt();
  str=sc.nextLine();
  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
```

2.5 Object Creation and Memory Allocation: 'new' Operator

2.5.1 Creating Objects and Memory Allocation of Objects

Q. Explain how memory is allocated to an object of a class with diagram. **(4 Marks)**

Q. How memory is allocated when multiple object of class are created ? Explain with example. **(4 Marks)**

Q. Explain memory allocation for object with example. **(4 Marks)**

- An object of a class will be allocated the memory space as required by the data members specified in the class. The object is an instance (example) of the class.
- For example, if human being is a class then we all are objects (instances or examples) of this class.

Syntax for creating an object

class_name object_name;

Note : The memory space is allocated to an object and not to a class. Hence, by writing the statement, circle c; we are actually allocating the memory space required for the data members of the class 'circle'. This statement allocates space as required for two float type elements i.e. for the variables 'r' and 'a'.

Each object will be allocated separate space in memory. Hence, if we declare another object, the memory space for the same will be different than the first object.

Example: Calculate the size of object B1 defined in following class :

```
class Book
{
char B_name[15]; int B_id;
int Price;
};
Book B1;
```

Output

```
B_name: 15 bytes
B_id: 2 bytes
Price: 2 bytes
Total : 19 bytes
```

- We will see another example that uses the 'new' operator along with destructor in the subsequent section.

2.5.2 Memory Recover: 'delete' Operator

- A destructor is used to destroy the memory allocated by the constructor.
- The following points are to be noted w.r.t. destructor :
  1. It is defined only in the public visibility of a class.
  2. The name of the destructor must be same as that of class prefixed with a tilde (~) sign. The name of the constructor and destructor are same as that of the class. Hence to differentiate between the two, the tilde sign is used.
  3. It cannot return or accept any value.

4. It is automatically called when the compiler exits from the main program.

- The destructor as discussed can only destroy the memory locations allocated by the constructor and the constructor can allocate memory only using the 'new' operator.
- 'new' operator allocates memory to a pointer. Hence, a pointer is to be declared and using the new operator a memory location can be allocated by the following syntax:

```
Pointer_name = new Data_type;
```

- To destroy this memory allocated by the new operator, we require delete operator. In this case the delete operator must be in the destructor. Hence, the destructor will be destroying the memory allocated. The syntax for the delete operator to destroy the memory space allocated is as shown below :

```
delete Pointer_name;
```

Program 2.5.1 : Write a C++ program to demonstrate the destructor.

```
# include<iostream.h>
# include<conio.h>
class test
{
   int *p;
   public:
   test()
   {
      p=new int;
   }
   void read()
   {
      cout<<"Enter a number";
      cin>>*p;
   }
   void display()
   {
      cout<<"Value="<<*p<<endl;
   }
   ~test()
   {
      delete p;
      cout<<"Destroyed";
   }
};
void main()
{
   clrscr();
   test t;
   t.read();
   t.display();
   getch();
}
```

Output

```
Enter a number5
Value=5
Destroyed
```

Explanation

- The destructor is the ~test() function. This destructor is automatically called after the end of the execution of the main() function. Hence, in our case after the statement getch(); the destructor is automatically called.
- When the destructor is called, the memory allocated by the constructor is deleted using the delete operator. The statement;

```
cout<<"Destroyed";
```

is then executed.

- Hence, similar to constructor even the destructor is not to be called explicitly. It is automatically called.

2.6 Returning a Value

- A method or a function can return a value.
- The data type of the returned value is to be mentioned in the beginning of the method definition i.e. after its access specifier.
- If a method doesn't return anything, it can be declares as a void function.
- The Java program given below shows the calculate() method returning the answer

```
import java.util.*;
class Euclid
{
private int n1,n2;
```

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

Output

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

2.7 Adding a Method that Takes Parameters

- We can also pass parameters to the function of a class.
- The parameter passing is same as it used to be in procedure oriented programming.
- The following is a Java program example where the accept() method accepts the parameters or arguments passed to it.

```
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
```

2.8 The "this" Keyword

2.8.1 "this" Keyword in C++

Q. Explain the concept of 'this' pointer. **(4 Marks)**

Q. Describe 'this' pointer with respect to its use only. **(2 Marks)**

Q. Describe the concept of' this' pointer. Give one example. **(4 Marks)**

Q. What is 'this' pointer ? Give suitable example. **(4 Marks)**

- "this" is a keyword used to indicate the current object. It is used to access the members of the current object.
- It is not much used until now because of the ease of access of variables and assigning values to variables.
- Let us see an example wherein we return an object using the keyword "this".

Program 2.8.1 : Write a program to demonstrate the use of "this" pointer.

```
#include<iostream.h>
#include<conio.h>
class Compare
{
   protected:
   int a;
   public:
   void read()
   {
      cout<<"Enter a value:";
      cin>>a;
   }
   Compare compare (Compare c)
   {
      if(a>c.a)
      return *this;
      else
      return c;
   }
   void display()
   {
      cout<<"The value is:"<<a;
   }
};
 void main()
 {
   clrscr();
   Compare c1;
   Compare c2;
   c1.read();
   c2.read();
   Compare c3;
   c3=c1.compare(c2);
   c3.display();
   getch();
}
```

Output

```
Enter a value:3
Enter a value:5
The value is:5
```

Explanation

- The variable "counter" is declared static and of type integer. The value of this variable is incremented in the constructor i.e. whenever the object is made. Hence the variable keeps a track of the number of objects created.
- The statement declaring the static variable "counter" as a member of the class "Count" is stated after the class.

– The syntax of the statement is “data_type class_name::variable_name;”. This statement is necessary for the compiler to resolve the scope of the static variables. This statement is used to initialize the value of variable "counter" to zero.

2.8.2 The “this” Keyword in Java

SPPU - Dec. 16, Dec. 19

Q.	What is ‘this’ pointer ?	**(Dec. 16, 3 Marks)**
Q.	Write a note on ‘this’ pointer.	**(Dec. 19, 3 Marks)**

– 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 6.9.1 below.

Program 2.8.2 : Write a Java 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.

2.9 Method Overloading

2.9.1 Function Overloading or Function Polymorphism

Q. Write short notes on : Function overloading **(5 Marks)**

- Function overloading refers to creating multiple functions with the same name but different parameter list.
- Different parameter list can be with reference to the number of parameters or the type of parameters.
- Since, for overloaded functions the functions have same name, a function can be called with the common name but the function that will be invoked is based on the parameter type and number of parameters.
- This concept of function overloading is also called as function polymorphism.
- Let us see some program examples of function overloading.

Program 2.9.1 : Write a C++ program to add two numbers using function overloading such that one function adds two integers, second function adds two float numbers and the third function adds a float number with an integer.

OR Explain need of function overloading. Write C++ program to demonstrate function overloading.

SPPU - Dec. 18, 6 Marks, May 19, 6 Marks

```
# include<iostream.h>
# include<conio.h>
float add(float a, int b)
{
   float c;
   c=a+b;
   return c;
}
int add(int a, int b)
{
   int c;
   c=a+b;
   return c;
}
float add(float a, float b)
{
   float c;
   c=a+b;
   return c;
}
void main()
{
   clrscr();
   int x,a=5,b=6;
   float y,p=3.5,q=6.6;
   x=add(a,b);
   cout<<"Sum="<<x<<endl;
   y=add(p,q);
   cout<<"Sum="<<y<<endl;
   y=add(p,a);
   cout<<"Sum="<<y<<endl;
   getch();
}
```

Output

```
Sum=11
Sum=10.1
Sum=8.5
```

Explanation

- The first call to function add() in the main() function, has both the integer elements 'a' and 'b'; hence the second add() function is called which has both the integer parameters.
- The second call to function add() in the main() function, has both the float type elements 'p' and 'q'; hence the third add() function is called which has both the float type parameters.
- The third call to function add() in the main() function, has a float type element 'p' and an integer type element 'a'; hence the first add() function is called with a similar type of parameters.
- This is a program example wherein the number of parameters for all the overloaded functions is same but their data types are different.

Program 2.9.2 : Write a C++ program to calculate the area of triangle, rectangle and circle using function overloading. The program should be menu-driven.

SPPU - Dec. 17, 4 Marks

OR Explain need of function overloading. Write C++ program to demonstrate function overloading.

SPPU - Dec. 18, 6 Marks, May 19, 6 Marks

```
# include<iostream.h>
# include<conio.h>
# include<math.h>
float area(float r)
{
   return (3.14*r*r);
}
float area(float l, float b)
{
   return (l*b);
}
float area(float a, float b,float c)
{
   float s,ar;
   s=(a+b+c)/2;
   ar=s*(s-a)*(s-b)*(s-c);
   ar=pow(ar,0.5);
   return ar;
}
void main()
{
   clrscr();
   int choice;
   float x,y,z,a;
   cout<<"1.Area of Circle\n2.Area of Rectangle\n3.Area of Triangle\nEnter your choice:";
   cin>>choice;
   switch(choice)
   {
      case 1:cout<<"Enter the radius of the circle:";
      cin>>x;
      a=area(x);
      cout<<"The area of the circle="<<a;
      break;
      case 2:cout<<"Enter the length and breadth of the rectangle:";
      cin>>x>>y;
      a=area(x,y);
      cout<<"The area of the rectangle="<<a;
      break;
      case 3:cout<<"Enter the length of the three sides of the triangle:";
      cin>>x>>y>>z;
      a=area(x,y,z);
      cout<<"The area of the triangle="<<a;
      break;
      default:cout<<"Invalid Choice";
   }
   getch();
}
```

Output

```
1.Area of Circle
2.Area of Rectangle
3.Area of Triangle
Enter your choice:2
Enter the length and breadth of the rectangle:3.5
2
The area of the rectangle=7
```

Explanation

- The user is asked for the choice of the shape whose area is to be found.
- The first choice is that of circle that calls to function area() that has only one parameter 'x'; hence the first area() function is called which has only one parameter.
- The second choice is for rectangle that calls to function area() that has two float type elements 'x' and 'y'; hence the second area() function is called which has two float type parameters.
- The third choice is for triangle that calls to function area() that has three float type elements 'x', 'y' and 'z'; hence the third area() function is called with a similar number of parameters.
- This is a program example wherein the number of parameters for all the overloaded functions is different while their data types are same.

Program 2.9.3 : Write a C++ program using function overloading to swap 2 integer numbers and swap 2 float numbers.

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
void swap(int *a, int *b)
{
   int temp;
   temp=*a;
   *a=*b;
   *b=temp;
}
void swap(float *a, float *b)
{
   float temp;
   temp=*a;
   *a=*b;
   *b=temp;
}
void main()
{
   int a,b;
   float c,d;
   clrscr();
   cout<<"Enter two integers:";
   cin>>a>>b;
   cout<<"Numbers entered are: a="<<a<<" and b="<<b<<endl;
   swap(&a,&b);
   cout<<"After swapping, a="<<a<<" and b="<<b<<endl;
   cout<<"Enter two float numbers:";
   cin>>c>>d;
   cout<<"Numbers entered are: c="<<c<<" and d="<<d<<endl;
   swap(&c,&d);
   cout<<"After swapping, c="<<c<<" and d="<<d<<endl;
   getch();
}
```

Output

```
Enter two integers:3
4
Numbers entered are: a=3 and b=4
After swapping, a=4 and b=3
Enter two float numbers:3.5  2.75
Numbers entered are: c=3.5 and d=2.75
After swapping, c=2.75 and d=3.5
```

Program 2.9.4 : Write a C++ program to calculate area of circle and area of rectangle using function overloading.

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
float area(float r)
{
   return (3.14*r*r);
}
float area(float l,float b)
{
   return(l*b);
}
void main()
{
   clrscr();
   cout<<"Area of circle with radius 10 units is "<<area(10)<<endl;
   cout<<"Area of rectangle with sides 5 units and 7 units is "<<area(5,7)<<endl;
   getch();
}
```

Output

```
Area of circle with radius 10 units is 314
Area of rectangle with sides 5 units and 7 units is 35
```

2.9.2 Constructor Overloading

Q.	Explain overloaded constructor in a class with suitable example.	**(8 Marks)**
Q.	Define constructor overloading.	**(2 Marks)**
Q.	Write syntax for overloaded constructor.	**(2 Marks)**
Q.	Write a program which implement the concept of overloaded constructor.	**(4 Marks)**

Although we have seen examples of constructor overloading in previous section, let us see one more example of constructor overloading.

Program 2.9.5 : Write a C++ program to calculate the area of triangle, rectangle and circle using constructor overloading. The program should be menu-driven.

```
# include<iostream.h>
# include<conio.h>
# include<math.h>
class Area
{
   float area;
   public:
   Area(float r)
   {
      area=3.14*r*r;
   }
   Area(float l, float b)
   {
      area=l*b;
   }
   Area(float a, float b,float c)
   {
      float s;
      s=(a+b+c)/2;
      area=s*(s-a)*(s-b)*(s-c);
      area=pow(area,0.5);
   }
   void display()
   {
      cout<<"Area="<<area;
   }
};
void main()
{
   clrscr();
   int choice;
   float x,y,z;
   cout<<"1.Area of Circle\n2.Area of Rectangle\n3.Area of Triangle\nEnter your choice:";
   cin>>choice;
   switch(choice)
   {
      case 1:cout<<"Enter the radius of the circle:";
      cin>>x;
      Area a(x);
      a.display();

   break;
      case 2:cout<<"Enter the length and breadth of the rectangle:";
      cin>>x>>y;
      Area a1(x,y);
      a1.display();
      break;
      case 3:cout<<"Enter the length of the three sides of the triangle:";
      cin>>x>>y>>z;
      Area a2(x,y,z);
      a2.display();
      break;
      default:cout<<"Invalid Choice";
   }
   getch();
}
```

Output

```
1.Area of Circle
2.Area of Rectangle
3.Area of Triangle
Enter your choice:1
Enter the radius of the circle:5
Area=78.5
```

Explanation

- The user is asked for the choice of the shape whose area is to be found and the object of the class is made accordingly with the passing of those variables to the constructor.
- Then the display function is called with respect to the object.

2.10 Using Object as Parameter and Returning an Object

2.10.1 Objects as Function Parameter/Arguments

Q. Explain object as function argument. **(4 Marks)**

Q. Explain with example object as member function argument. **(4 Marks)**

- Objects can also be passed and returned by a function. The syntax of passing the arguments is same as primitive data types.
- In the following example an object of complex number is passed as well as returned by the function. The function adds two complex numbers and returns the result

Program 2.10.1 : Write a C++ program to add two complex numbers using objects of a class Complex.

```
#include<iostream.h>
#include<conio.h>
class Complex
{
   int x,y;
   public:
   void read()
   {
      cout<<"Enter the real and imaginary parts of a complex number:";
      cin>>x>>y;
   }
   Complex add (Complex c)
   {
      Complex c1;
      c1.x=x+c.x;
      c1.y=y+c.y;
      return c1;

   }
   void display()
   {
      if(y<0)
      cout<<x<<y<<"i";
      else
      cout<<x<<"+i"<<y;
   }
};
void main()
{
   clrscr();
   Complex c1;
   Complex c2;
   Complex c3;
   c1.read();
   c2.read();
   c3=c1.add(c2);
   c3.display();
   getch();
}
```

Output

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

2.10.2 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 2.10.2 : Write a Java 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.

2.10.3 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 2.10.3 : Write a Java 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.

2.10.4 Call by Value and Call by Reference

We know how parameters are passed to a method. The values of those variables were given to a set of another variable in the method.

2.10.4(A) 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 2.10.4 : Write a Java 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.

2.10.4(B) 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 2.10.5.

Program 2.10.5 : Write a Java 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.

2.11 Array of Objects in C++

- When multiple objects of a class are to be made we can use array of objects.
- Creating array of objects is quite similar to creating array of any other data type.
- Syntax of creating array of objects
- class_name object_array_name[size];

- Each object in the array can then be accessed using the indices from 0 to n-1, in the same manner as we have done for integer arrays.
- Let us take an example of array of objects.

Program 2.11.1 : Write a C++ program to declare a class 'staff' having data members as name and department. Accept this data for 10 staffs and display names of staff that are in cm department.

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class Staff
{
   private:
   char name[20],dept[20];
   public:
   void accept()
   {
      cout<<"Enter name and department:";
      cin>>name>>dept;
   }
   void display()
   {
      cout<<name<<"\t"<<dept<<"\n";
   }
};
void main()
{
   int i;
   clrscr();
   Staff s[10];
   for(i=0;i<=9;i++)
   {
      s[i].accept();
   }
   cout<<"Name\tDepartment\n";
   for(i=0;i<=9;i++)
   {
      s[i].display();
   }
   getch();
}
```

Output

```
Enter name and department:a
finance
Enter name and department:Ajay
HR
Enter name and department:Vijay
HR
Enter name and department:Jay
Finance
Enter name and department:Digvijay
Finance
Enter name and department:Manoj
Sales
Enter name and department:Manish
Sales
Enter name and department:Suresh
Finance
Enter name and department:Dinesh
HR
Enter name and department:Saurav
HR
Name        Department
a           finance
Ajay        HR
Vijay       HR
Jay         Finance
Digvijay    Finance
Manoj       Sales
Manish      Sales
Suresh      Finance
Dinesh      HR
Saurav      HR
```

Program 2.11.2 : Write a C++ program to declare a class staff having data members as name and post. Accept and display data for five staff members. (Using array of object)

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class Staff
{
```

```
    private:
    char name[20],post[20];
    public:
    void accept()
    {
       cout<<"Enter name and post:";
       cin>>name>>post;
    }
    void display()
    {
       cout<<name<<"\t"<<post<<"\n";
    }
};
void main()
{
    int i;
    clrscr();
    Staff s[5];
    for(i=0;i<=4;i++)
    {
       s[i].accept();
    }
    cout<<"Name\tPsot\n";
    for(i=0;i<=4;i++)
    {
       s[i].display();
    }
    getch();
}
```

Output

```
Enter name and post:Jay
Manager
Enter name and post:Ajay
Officer
Enter name and post:Vijay
Officer
Enter name and post:Suresh
Clerk
Enter name and post:Dinesh
Clerk
Name      Psot
Jay       Manager
Ajay      Officer
Vijay     Officer
Suresh    Clerk
Dinesh    Clerk
```

Program 2.11.3 : Write a C++ program to declare class Account having data member as acc_no and balance. Accept and display data for five object using pointer to array of object.

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class Account
{
    private:
    int acc_no,balance;
    public:
    void get_data()
    {
       cout<<"Enter account number and balance:";
       cin>>acc_no>>balance;
    }
    void display()
    {
       cout<<acc_no<<"\t"<<balance<<endl;
    }
};
void main()
{
    Account b[5];
    int i;
    clrscr();
    for(i=0;i<=4;i++)
    {
       b[i].get_data();
    }
    cout<<"Acc No\t\Balance\n";
    for(i=0;i<=4;i++)
    {
       b[i].display();
    }
    getch();
}
```

Output

```
Enter account number and balance:2
3
Enter account number and balance:1
2
Enter account number and balance:123
5000
Enter account number and balance:456
```

```
10000
Enter account number and balance:789
1000
Acc No   Balance
2          3
1          2
123       5000
456      10000
789       1000
```

Program 2.11.4 : Write a C++ program to declare a class mobile having data members as price and model number. Accept and display this data for ten objects.

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class Mobile
{
   private:
   char model[20];
   int price;
   public:
   void accept()
   {
      cout<<"Enter model name and price:";
      cin>>model>>price;
   }
   void display()
   {
      cout<<model<<"\t"<<price<<"\n";
   }
};
void main()
{
   int i;
   clrscr();
   Mobile s[10];
   for(i=0;i<=9;i++)
   {
      s[i].accept();
   }
   cout<<"Model\tPrice\n";
   for(i=0;i<=9;i++)
   {
      s[i].display();
   }
   getch();
}
```

Output

```
Enter model name and price:s1
2000
Enter model name and price:s2
3000
Enter model name and price:s3
4000
Enter model name and price:s4
5000
Enter model name and price:s5
6000
Enter model name and price:s6
7000
Enter model name and price:s7
8000
Enter model name and price:s8
9000
Enter model name and price:s9
10000
Enter model name and price:s10
11000
Model   Price
s1      2000
s2      3000
s3      4000
s4      5000
s5      6000
s6      7000
s7      8000
s8      9000
s9      10000
s10     11000
```

2.11.2 Array of Objects in Java

- To create an array of objects of a class, the following syntax is to be used :

  class_name object_name[] = new class_name[size_of_array].

- You will notice this is exactly similar to creating an array of any in-built data type i.e. “int”, “float” etc.

Note : Memory is not allocated to each object with the above declaration. We need to use the "new" operator separately for each object of the array for allocating memory to it. We will see how this is to be done in the Program 2.11.5.

Program 2.11.5 : Write a Java program to create an array of objects of a class student. The class should have field members name, id, total marks and marks in three subjects namely Physics, Chemistry and Maths. Accept information of 'n' students and display them in tabular form.

```
import java.util.*;
class Student{
private String name;
private int id,p,c,m,t;
void accept()
{
 String str;
 Scanner sc = new Scanner (System.in);
 System.out.println("Enter name, ID and marks in Physics, Chemistry and Maths:");
 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+"\t"+id+"\t"+p+"\t"+c+"\t"+m+"\t"+t);
}
}
class Main
{
public static void main(String args[])
{
 int n,i;
 Scanner sc=new Scanner(System.in);
 System.out.println("Enter number of students:");
 n=sc.nextInt();
 Student s[]= new Student[n];
 for(i=0;i<=n-1;i++)
 {
  s[i]=new Student();
  s[i].accept();
 }
 System.out.println("Name\tID\tPhy\tChem\tMaths\tTotal");
 for(i=0;i<=n-1;i++)
 {
  s[i].display();
 }
}
}
```

Output

```
Enter number of students:
3
Enter name, ID and marks in Physics, Chemistry and Maths:
Ajay
325
90
90
90
Enter name, ID and marks in Physics, Chemistry and Maths:
Vijay
425
90
99
99
Enter name, ID and marks in Physics, Chemistry and Maths:
Jay
525
99
99
99
Name   ID    Phy    Chem    Maths    Total
Ajay   325   90     90      90       270
Vijay  425   90     99      99       288
Jay    525   99     99      99       297
```

Explanation

- You will notice that the declaration statement shown above is used to declare an array of objects of the class Student.
- The class Student has the field members as given in the problem statement. The class also has methods to accept and display the information of a student.

- The array of objects created is not allocated with the memory. Hence special memory is allocated to each object using the new operator. This is done by the statement

  s[i]=new Student()

- This statement is required to each object, hence it is put in the for loop. If this statement is not written, then an error will be generated during the execution of the program called as NullPointerException i.e. data is stored into an object that is not yet initialized or no memory is allocated to the object i.e. it is null.
- The information is accepted for each student by calling the accept() method for each object in the for loop and then displayed in tabular form using another for loop.

Program 2.11.6 : Create a class employee with data members empid, empname, designation and salary. Write methods get_employee() to take user input, and show_employee() to display employee details. Make an array of objects of this class and accept & display information of multiple employees.

```
import java.util.*;
class Employee
{
private String empname, designation;
private int empid;
private float salary;
void get_employee()
{
 Scanner sc = new Scanner (System.in);
 System.out.println("Enter name, ID , designation and salary:");
 empname=sc.next();
 empid=sc.nextInt();
 designation=sc.next();
 salary=sc.nextFloat();
}
void show_employee()
{
System.out.println(empname+"\t"+empid+"\t"+designation+"\t"+salary);
}
}
```

```
class Main
{
public static void main(String args[])
{
 int n,i;
 Scanner sc=new Scanner(System.in);
 System.out.println("Enter number of employees:");
 n=sc.nextInt();
 Employee s[]= new Employee[n];
 for(i=0;i<=n-1;i++)
 {
  s[i]=new Employee();
  s[i].get_employee();
 }
 System.out.println("Name\tID\tdesig.\t\tsalary");
 for(i=0;i<=n-1;i++)
 {
  s[i].show_employee();
 }
}
}
```

Output

```
Enter number of employees:
3
Enter name, ID , designation and salary:
Ajay
325
Accountant
50000
Enter name, ID , designation and salary:
Vijay
425
Registrar
30000
Enter name, ID , designation and salary:
Jay
525
Operator
12000
Name    ID      desig.          salary
Ajay    325     Accountant      50000.0
Vijay   425     Registrar       30000.0
Jay     525     Operator        12000.0
```

Explanation

- The class Employee is first made with the private data (field) members as given in the problem statement. The class has two method members namely
  1. get_employee() to accept the information of employee. This method has to accept input from user and 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_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 array of objects is created of the class Employee named as 's'. And the memory is allocated for each object in the array using a separate "new" operator for each of them as discussed in the previous program. This is done in the for loop while accepting the information of the employees.
- Similarly the show_employee() method is called to display the information in tabular form as seen in the output.

Program 2.11.7 : Write a Java program to create an array of objects of a class student. The class should have field members name, id, total marks and marks in three subjects namely Physics, Chemistry and Maths. Accept information of 'n' students and display them in tabular form in descending order of the total marks obtained by the student.

```
import java.util.*;
class Student
{
private String name;
private int id,p,c,m,t;
void accept()
{
 Scanner sc = new Scanner (System.in);
 System.out.println("Enter name, ID and marks in Physics, Chemistry and Maths:");
 name=sc.nextLine();
 id=sc.nextInt();
 p=sc.nextInt();
 c=sc.nextInt();
 m=sc.nextInt();
 t=p+c+m;
}
boolean compare (Student s)
{
 if(t<s.t)
 return true;
 else
 return false;
}
void display()
{

System.out.println(name+"\t"+id+"\t"+p+"\t"+c+"\t"+m+"\t"+t);
}
}
class Main
{
public static void main(String args[])
{
 int n,i,j;
 Scanner sc=new Scanner(System.in);
 System.out.println("Enter number of students:");
 n=sc.nextInt();
 Student s[]= new Student[n];
 for(i=0;i<=n-1;i++)
 {
  s[i]=new Student();
  s[i].accept();
 }
 Student temp=new Student();
 for(i=0;i<=n-2;i++)
 {
  for(j=0;j<=n-2;j++)
  {
   if(s[j].compare(s[j+1]))
   {
    temp=s[j];
```

```
      s[j]=s[j+1];
      s[j+1]=temp;
     }
    }
  }
  System.out.println("Name\tID\tPhy\tChem\tMaths\tTotal");
  for(i=0;i<=n-1;i++)
  {
   s[i].display();
  }
}
}
```

Output

```
Enter number of students:
3
Enter name, ID and marks in Physics, Chemistry and Maths:
Ajay
325
90
90
90
Enter name, ID and marks in Physics, Chemistry and Maths:
Vijay
425
90
90
99
Enter name, ID and marks in Physics, Chemistry and Maths:
Jay
525
99
99
99
Name  ID    Phy   Chem  Maths   Total
Jay   525   99    99    99      297
Vijay 425   90    90    99      279
Ajay  325   90    90    90      270
```

Explanation

- You will notice that the declaration statement shown above is used to declare an array of objects of the class Student.
- The class Student has the field members as given in the problem statement.
- The class also has methods to accept student information, compare total and display the information of a student.
- compare() method to compare the total of the object through which the method is called and the total of the consecutive next object which is passed to the method as a parameter. This method returns a Boolean value as true or false based on whether swapping will be required or not for arranging the two objects in descending order of their total.
- The array of objects created is not allocated with the memory. Hence special memory is allocated to each object using the new operator. This is done by the statement

  s[i]=new Student()

- This statement is required to each object, hence it is put in the for loop. If this statement is not written, then an error will be generated during the execution of the program called as NullPointerException i.e. data is stored into an object that is not yet initialized or no memory is allocated to the object i.e. it is null.
- The information is accepted for each student by calling the accept() method for each object in the for loop and then displayed in tabular form using another for loop.
- Another nested for loop is added for sorting the student details according to their total marks obtained before displaying the data in tabular form.

Program 2.11.8 : Create a class employee with data members empid, empname, designation and salary. Write methods get_employee()- to take user input, and show_employee() to display employee details. Accept and Display the information of multiple employees in descending order of their salary.

```
import java.util.*;
class Employee {
private String empname, designation;
private int empid;
private float salary;
void get_employee()
{
```

```
 Scanner sc = new Scanner (System.in);
 System.out.println("Enter name, ID , designation and salary:");
 empname=sc.next();
 empid=sc.nextInt();
 designation=sc.next();
 salary=sc.nextFloat();
}
boolean compare(Employee e)
{
 if(salary<e.salary)
 return true;
 else
 return false;
}
void show_employee()
{

System.out.println(empname+"\t"+empid+"\t"+designation+"\t"+salary);
}
}
class Main
{
public static void main(String args[])
{
 int n,i,j;
 Scanner sc=new Scanner(System.in);
 System.out.println("Enter number of employees:");
 n=sc.nextInt();
 Employee s[]= new Employee[n];
 for(i=0;i<=n-1;i++)
 {
  s[i]=new Employee();
  s[i].get_employee();
 }
 Employee temp=new Employee();
 for(i=0;i<=n-2;i++)
 {
  for(j=0;j<=n-2;j++)
  {
   if(s[j].compare(s[j+1]))
   {
    temp=s[j];
    s[j]=s[j+1];
    s[j+1]=temp;
   }
  }
 }
 System.out.println("Name\tID\tdesig.\t\tsalary");
 for(i=0;i<=n-1;i++)
 {
  s[i].show_employee();
 }
}
}
```

Output

```
Enter number of employees:
3
Enter name, ID , designation and salary:
Ajay
325
Accountant
50000
Enter name, ID , designation and salary:
Jay
525
Operator
12000
Enter name, ID , designation and salary:
Vijay
425
Registrar
30000
Name    ID      desig.          salary
Ajay    325     Accountant      50000.0
Vijay   425     Registrar       30000.0
Jay     525     Operator        12000.0
```

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 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_employee() to display all the information about the employee.
3. compare() method to compare the salary of the object through which the method is called and the salary of the consecutive next object which is passed to the method as a parameter. This method returns a Boolean value as true or false based on whether swapping will be required or not for arranging the two objects in descending order of their salary.

- 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 array of objects is created of the class Employee named as 's'. And the memory is allocated for each object in the array using a separate "new" operator for each of them as discussed in the previous program. This is done in the for loop while accepting the information of the employees.
- Another nested for loop is added for sorting the employee details according to their salary.
- Similarly the show_employee() method is called to display the information in tabular form as seen in the output.

2.13 Static Data Members and Methods

SPPU - Dec. 16, May 19, Dec. 19

Q.	Write any two characteristics of static member function.	**(4 Marks)**
Q.	Explain the need of static member function with example.	**(4 Marks)**
Q.	What is static data member and static member function ?	**(Dec. 16, 4 Marks)**
Q.	Enlist the properties of static data members and static member function.	**(May 19, 6 Marks)**
Q.	What is Static variable and static function? Explain with example.	**(Dec. 19, 6 Marks)**

- Static members are common for the entire class and not for each object of the class.
- We can have static data members as well as static function members.
- We will see the static data member in this class and static function members will be dealt with in chapter 7.
- Static data member will have same value for all the objects. Hence if one object alters the value of static data member, its altered value will be available for all other objects
- Let us see a program example of static data member.

Program 2.13.1 : Write a C++ program to define a class having data members principle, duration and rate-of-interest. Declare rate-of-interest as static member variable. Calculate the simple interest and display it for one object.

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class Interest
{
  private:
  static float rate;
  float principal,duration,sim_int;
  public:
  void accept()
  {
    cout<<"Enter principal amount and duration:";
    cin>>principal>>duration;
  }
  void calculate()
  {
    sim_int=principal*rate*duration/100;
  }
  void display()
  {
    cout<<"Simple Interest="<<sim_int;
  }
};
float Interest::rate=10;
void main()
```

```
{
   Interest i;
   clrscr();
   i.accept();
   i.calculate();
   i.display();
   getch();
}
```

Output

```
Enter principal amount and duration:1000
5
Simple Interest=500
```

2.13.1 Static Class Members in Java

- 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 2.13.2.

Program 2.13.1 : 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.

2.14 Forward Declaration

- If the object of one class is to be used in another which is defined before the other, then we need to declare the class name before the class where it is used. This is called as forward declaration.
- **For example:**

```
class Student
{
   int roll;
   Exam e;
}
class Exam
{
   int p,c,m;
}
```

- In the above case, an object of class Exam is declared in the class Student. Hence, it is necessary to declare the class Exam, although the details of this class can be defined later. Hence the correct definition will be as shown below :

```
class Exam;
class Student
{
   int roll;
   Exam e;
}
class Exam
{
   int p,c,m;
}
```

2.15 Class as "Abstract Data Type" (ADT)

- We can have abstract classes. These classes cannot be instantiated i.e. no objects of abstract class can be created.
- We will see Abstract class in further chapters. The abstract data type are used create a template for data.

❑❑❑

Model Question Paper (In Sem.)

Object Oriented Programming

Semester - III (Information Technology)

Time : 1 Hour **Maximum Marks : 30**

Instruction to the candidates :

1) **Answer Q. 1 or Q. 2, Q. 3, or Q. 4.**
2) **Neat diagrams must be drawn wherever necessary.**
3) **Figure to the right indicates full marks.**
4) **Make suitable assumptions if necessary.**

Q. 1 **(a)** Compare Object Oriented Programming with Procedure oriented programming.

(Refer Table 1.4.1) **(5 Marks)**

(b) Explain the terms: class, object, encapsulation, abstraction, inheritance and polymorphism

(Refer Section 1.5) **(5 Marks)**

(c) Write a Java program to accept two numbers and display its product.

(Refer Program 1.13.2) **(5 Marks)**

OR

Q. 2 **(a)** Explain the limitations of procedure oriented programming and structure of object oriented programming.

(Refer Sections 1.4.1 and 1.4.2) **(5 Marks)**

(b) Explain Java Virtual Machine with block diagram. **(Refer Section 1.7)** **(5 Marks)**

(c) Write a program to check if the entered number is prime number or not.

(Program 1.15.2) **(5 Marks)**

Q. 3 **(a)** Explain the methods of adding functions to a class in C++. **(Refer Section 2.4.1)** **(5 Marks)**

(b) Explain the use of “new”and “delete”operators

(Refer Section 2.5.1 and 2.5.2) **(5 Marks)**

(c) Write a program to demonstrate passing of object to a method

(Refer Section 2.10.1 and Program 2.10.1) **(5 Marks)**

OR

Q. 4 **(a)** Explain nesting of member function with a program example

(Refer Section 2.4.2(A)) **(5 Marks)**

(b) What is method overloading? Write a C++ program to add two numbers using function overloading such that one function adds two integers, second function adds two float numbers and the third function adds a float number with an integer. **(Refer Section 2.9.1 and Program 2.9.1)** **(5 Marks)**

(c) Write a C++ program to declare class Account having data member as acc_no and balance. Accept and display data for five object using pointer to array of object. **(Refer Program 2.11.3)** **(5 Marks)**

❑❑❑

Constructors and Destructors

Syllabus

Constructors : Introduction, Use of Constructor, Characteristics of Constructors, Types of Constructor, Constructor Overloading, Constructor with Default Arguments, Symbolic Constants, Garbage Collection: Destructors and Finalizers.

3.1 Introduction of Constructors : Use and Characteristics

3.1.1 Constructor

SPPU - May 17

Q. Define constructors. **(May 17, 4 Marks)**

– They are special member functions used for initializing the data elements of the object. There are some important points we need to understand about the constructor.

– These points are listed below:

1. The name of constructor must always be same as that of the class.
2. It has to be defined only in the public visibility of a class.
3. It should not have any return type; not even void.
4. Any number of constructors can be defined in a class and such a set of constructors implements a concept called as constructor overloading.
5. The constructor is executed the moment an object of that class is created.
6. A constructor cannot be inherited but a derived class can call the constructor of the base class.
7. A constructor cannot be static or virtual.

Note : Some of the terms used above may not be cleared now, but they will be cleared as you study the further sections

– There are three types of constructors based on the parameter list. The types of constructors are:

1. Default Constructor
2. Parameterized Constructor
3. Copy Constructor

– Let us see the implementation of these constructors in the subsequent sub-sections.

3.1.2 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.

3.2 Types of Constructor

3.2.1 Default Constructor

- If a constructor does not have any parameter list i.e. it cannot accept any parameters than it is called as a default constructor.
- Thus, the default constructor has the blank brackets indicating no parameters can be accepted.
- Let us initialize the values of the variable radius of the class circle, using default constructor and the value of the variable 'n' for the below programs.

Program 3.2.1 : Write a program to find area of circle using Object Oriented Programming. The value of the radius must be accepted from the user in the constructor and the class circle must have two inline functions namely : (a) compute() for calculating the area. (b) display() for displaying the result.

```
# include<iostream.h>
# include<conio.h>
class circle
{
   float r,a;
   public:
   circle()
   {
      cout<<"Enter the value of radius:";
      cin>>r;
   }
   void compute();
   void display();
};
inline void circle::compute()
{
   a=3.14*r*r;
}
inline void circle::display()
{
   cout<<"Area="<<a;
}
void main()
{
   clrscr();
   circle c;
   c.compute();
   c.display();
   getch();
}
```

Output

```
Enter radius:5
Area=78.5
```

Explanation

- The special function with the name of the class i.e. 'circle' is a constructor. This constructor is not called in the main function, but as discussed it is automatically called when the object of the class is made.
- When compared to the previous program, the task of initializing the value of the variable 'r' is done in the constructor, while the remaining logic is similar.

Program 3.2.2 : Write a program to calculate the value of the following series using default constructor and inline member function : $S = 1^2 + 2^2 + 3^2 + 4^2 \ldots\ldots + n^2$

```
# include<iostream.h>
# include<conio.h>
class series
{
   int n,i,sum;
   public:
   series()
   {
      cout<<"Enter the value of n:";
      cin>>n;
      sum=0;
   }
   void compute();
   void display();
};
inline void series::compute()
{
   for(i=1;i<=n;i++)
   {
      sum=sum+i*i;
   }
}
inline void series::display()
{
   cout<<"Value of the series="<<sum;
}
void main()
{
   clrscr();
   series s;
   s.compute();
   s.display();
   getch();
}
```

Output

```
Enter the value of n:5
Value of the series=55
```

Explanation

- Here, the variables i.e. 'n' and 'sum' are initialized in the constructor. The value of sum is initialized to zero while the value of 'n' is taken from the user. The remaining logic is similar.

3.2.2 Parameterized Constructor

- If a constructor does have a parameter list i.e. it accepts one or more parameters than it is called as a parameterized constructor.
- Thus, the parameterized constructor has an argument list written in the brackets associated with the function definition header line.
- To pass the variable to the constructor, it has to be mentioned in the brackets along with the object name while making the object. The syntax of creating an object is then:

```
Class_name Object_name (arguments);
```

- Let us see the same examples of programs again with parameterized constructor.

Program 3.2.3 : Write a program to find area of circle using Object Oriented Programming. The value of the radius must be accepted from the user in the main program and passed to the parameterized constructor and the class circle must have two inline functions namely : (a) compute() for calculating the area. (b) display() for displaying the result.

```
# include<iostream.h>
# include<conio.h>
class circle
{
   float r,a;
   public:
   circle(float x)
   {
      r=x;
   }
   void compute();
   void display();
};
inline void circle::compute()
{
   a=3.14*r*r;
}
inline void circle::display()
{
   cout<<"Area="<<a;
}
void main()
{
   clrscr();
```

```
    float p;
    cout<<"Enter the radius of the circle";
    cin>>p;
    circle c(p);
    c.compute();
    c.display();
    getch();
}
```

Output

```
Enter radius:5
Area=78.5
```

Explanation

- Here, the constructor has an argument to be accepted i.e. a float variable 'x'. This variable is to be passed to the constructor when the object is made.
- To pass the variable to the constructor, it has to be mentioned in the brackets along with the object name while making the object. You will notice that the variable is passed while making the object in the brackets along with the object i.e. in the statement:

```
circle c(p);
```

- Since the variable 'p' is to be passed while making the object, the value of variable 'p' is to be accepted from user first and then the object can be created.

Program 3.2.4 : Write a program to calculate the value of the following series using parameterized constructor and inline member function : $S = 1^2 + 2^2 + 3^2 + 4^2 \ldots\ldots + n^2$

```
# include<iostream.h>
# include<conio.h>
class series
{
    int n,i,sum;
    public:
    series(int x)
    {
        n=x;
        sum=0;
    }
    void compute();
    void display();
};
inline void series::compute()
{
    for(i=1;i<=n;i++)
    {
        sum=sum+i*i;
    }
}
inline void series::display()
{
    cout<<"Value of the series="<<sum;
}
void main()
{
    clrscr();
    int x;
    cout<<"Enter the value of n:";
    cin>>x;
    series s(x);
    s.compute();
    s.display();
    getch();
}
```

Output

```
Enter the value of n:5
Value of the series=55
```

Explanation

- Similar; to the previous program, here also the value of the variable 'n' is accepted from the user in the main program and this is passed to the constructor while making the object in the statement:

```
series s(x);
```

3.2.3 Copy Constructor (Multiple Constructors/Constructor Overloading)

- If a constructor has an object of the same class as a parameter in the argument list than it is called as a copy constructor.
- The copy constructor uses an object of the same class to initialize another object of the class.

- To pass the object to the constructor, it has to be mentioned in the brackets along with another object name while making this object. The syntax of creating an object is similar to parameterized constructor with the only difference that the object is passed as the argument. The syntax is shown again for your reference:

```
Class_name Object_name (arguments);
```

- In short, a copy constructor is used to initialize the variables of an object using another object.
- Let us see the same examples of programs again with copy constructor.

Program 3.2.5 : Write a program to find area of circle using Object Oriented Programming. The value of the radius must be accepted from the user in the main program and passed to the copy constructor and the class circle must have two inline functions namely : (a) compute() for calculating the area. (b) display() for displaying the result.

```
# include<iostream.h>
# include<conio.h>
class circle
{
   float r,a;
   public:
   circle(float x)
   {
      r=x;
   }
   circle(circle &c)
   {
      r=c.r;
   }
   void compute();
   void display();
};
inline void circle::compute()
{
   a=3.14*r*r;
}
inline void circle::display()
{
   cout<<"Area="<<a<<endl;
}
void main()
{
   clrscr();
   float p;
   cout<<"Enter the radius of the circle";
   cin>>p;
   circle c(p);
   c.compute();
   c.display();
   circle c1(c);
   c1.compute();
   c1.display();
   getch();
}
```

Output

```
Enter the radius of the circle5
Area=78.5
Area=78.5
```

Explanation

- There are two constructors, hence this concept is also called as constructor overloading.
- The first constructor has an argument to be accepted i.e. a float variable 'x'. This variable is to be passed to the constructor when the object is made.
- The second constructor has an argument of type 'circle'. This type is the object of the same class i.e. 'circle'. This is the copy constructor. We have passed the earlier created object 'c' to the constructor of the second object 'c1' to initialize its variable 'r'.
- Both the objects are created in the main() function with the respective constructors being called as discussed in the previous point. The other functions are called individually for each object.Remember memory space allocated for each object are different.

Program 3.2.6 : Write a program to calculate the value of the following series using copy constructor and inline member function : $S = 1^2 + 2^2 + 3^2 + 4^2 \ldots\ldots + n^2$

```
# include<iostream.h>
# include<conio.h>
class series
{
   int n,i,sum;
   public:
   series(int x)
   {
      n=x;
      sum=0;
   }
   series(series &x)
   {
      n=x.n;
      sum=0;
   }
   void compute();
   void display();
};
inline void series::compute()
{
   for(i=1;i<=n;i++)
   {
      sum=sum+i*i;
   }
}
inline void series::display()
{
   cout<<"Value of the series="<<sum<<endl;
}
void main()
{
   clrscr();
   int x;
   cout<<"Enter the value of n:";
   cin>>x;
   series s(x);
   s.compute();
   s.display();
   series s2(s);
   s2.compute();
   s2.display();
   getch();
}
```

Output

```
Enter the value of n:5
Value of the series=55
Value of the series=55
```

Explanation

- The copy constructor is similarly used for creating the second object as in the previous program.

Program 3.2.7 : Consider the definition of the following class :

```
class Sample
{
private
 int x;
 double y;
public :
 Sample(); //constructor 1
 Sample (int, double); //constructor 2
 Sample(Sample &P); Constructor 3
};
```

(i) Write the definition of the constructor 1 so that the private member variables are initialized to 0.

(ii) Write the definition of the constructor 2 so that the private member variable x and y is initialized according to the value of the parameter.

(iii) Write the definition of the constructors 3, where copy one object to another. **SPPU - Dec. 19, 6 Marks**

```
class Sample
{
private
   int x;
   double y;
public :
```

```
    Sample(); //constructor 1
    {
       x=y=0;
    }
    Sample (int a, double b);//constructor 2
    {
       x=a;
       y=b;
    }
    Sample(Sample &P); Constructor 3
    {
       x=P.x;
       y=P.y;
    }
};
```

3.2.4 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.

3.2.4(A) Parameterized Constructor

Program 3.2.8 : 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);
}
}
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(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 3.2.9 : 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.

3.2.4(B) Default Constructor

Program 3.2.10 : 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 3.2.11 : 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.

3.2.4(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 3.2.12 : 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 3.2.13 : 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.

3.3 Constructor Overloading

Q. Explain overloaded constructor in a class with suitable example. **(8 Marks)**

Q. Define constructor overloading. **(2 Marks)**

Q. Write syntax for overloaded constructor. **(2 Marks)**

Q. What is overloaded constructor in a class? Explain with example. **(4 Marks)**

Q. Write a program which implement the concept of overloaded constructor. **(4 Marks)**

Although we have seen examples of constructor overloading in previous section, let us see one more example of constructor overloading.

Program 3.3.1 : Write a program to calculate the area of triangle, rectangle and circle using constructor overloading. The program should be menu-driven.

```
# include<iostream.h>
# include<conio.h>
# include<math.h>
class Area
{
   float area;
   public:
   Area(float r)
   {
      area=3.14*r*r;
   }
   Area(float l, float b)
   {
      area=l*b;
   }
```

```
    Area(float a, float b,float c)
    {
      float s;
      s=(a+b+c)/2;
      area=s*(s-a)*(s-b)*(s-c);
      area=pow(area,0.5);
    }
    void display()
    {
      cout<<"Area="<<area;
    }
};
void main()
{
    clrscr();
    int choice;
    float x,y,z;
    cout<<"1.Area of Circle\n2.Area of Rectangle\n3.Area
of Triangle\nEnter your choice:";
    cin>>choice;
    switch(choice)
    {
      case 1:cout<<"Enter the radius of the circle:";
      cin>>x;
      Area a(x);
      a.display();

    break;
      case 2:cout<<"Enter the length and breadth of the
rectangle:";
      cin>>x>>y;
      Area a1(x,y);
      a1.display();
      break;
      case 3:cout<<"Enter the length of the three sides of
the triangle:";
      cin>>x>>y>>z;
      Area a2(x,y,z);
      a2.display();
      break;
      default:cout<<"Invalid Choice";
    }
    getch();
}
```

Output

```
1.Area of Circle
2.Area of Rectangle
3.Area of Triangle
Enter your choice:1
Enter the radius of the circle:5
Area=78.5
```

Explanation

- The user is asked for the choice of the shape whose area is to be found and the object of the class is made accordingly with the passing of those variables to the constructor.
- Then the display function is called with respect to the object.

3.4 Constructor with Default Arguments

Q. State the use of default parameters in constructor with example. **(4 Marks)**

Q. What do you mean by default argument ? Illustrate concept of constructor with default argument using suitable example. **(4 Marks)**

Q. Explain constructor with default argument. **(4 Marks)**

- If a constructor has value passed to it, the given values are initialised to the data members.
- But in case if data value is not passed, then some default value should be initialised in the data members by the objects. To achieve this we require constructors with default parameters.

Program 3.4.1 : Illustrate the concept of constructor with default argument with suitable example.

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class Interest
{
  private:
  static float rate;
  float principal,duration,sim_int;
  public:
```

```
    Interest(float x=1000,float y=10)
    {
      principal=x;
      duration=y;
    }
    void calculate()
    {
      sim_int=principal*rate*duration/100;
    }
    void display()
    {
      cout<<"Simple Interest="<<sim_int;
    }
};
float Interest::rate=10;
void main()
{
    float x,y;
    clrscr();
    cout<<"Enter principal amount:";
    cin>>x;
    Interest i(x);
    i.calculate();
    i.display();
    getch();
}
```

Output

```
Enter principal amount:2000
Simple Interest=2000
```

Explanation

- In this case, the values of principal and duration, if passed then the same will be initialised. If no values are passed then the values 1000 and 10 are initialised.

3.5 Symbolic Constants

- The keyword final can be preceded to any member of a class or to the class itself.
- Making anything final has following implications.
  1. If a field member is declared as final then the variable value cannot be changed i.e. it becomes a constant. This is called as symbolic constant.
  2. If a method is declared as final then that method cannot be overridden.
  3. 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 variable i.e. symbolic constant.

Program 3.5.1 : Write a program to display volume of sphere. Make use of symbolic constant to display the volume.

```
import java.util.*;
class Sphere{
protected float r,vol;
final float pi=3.14f;
public void read(float x)
{
 r=x;
}
public void calculate()
{
 vol=pi*r*r*r*4/3;
}
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();
 }
}
```

Output :

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

Explanation :

- The implementation program is done with final variable "pi" in class Sphere.
- This will become a constant and its value cannot be changed.

3.6 Garbage Collection : Destructors and Finalizers

3.6.1 Destructor

SPPU - May 17

Q	Explain the concept of destructor in C++.	**(5 Marks)**
Q.	State the characteristics of destructor.	**(2 Marks)**
Q.	What is destructor? Give its syntax and example.	**(4 Marks)**
Q.	What is destructor ? How destructor is declared ? When destructor is invoked ?	**(4 Marks)**
Q.	Define destructors.	**(May 17, 4 Marks)**

- A destructor is used to destroy the memory allocated by the constructor.
- The following points are to be noted w.r.t. destructor :
  1. It is defined only in the public visibility of a class.
  2. The name of the destructor must be same as that of class prefixed with a tilde (~) sign. The name of the constructor and destructor are same as that of the class. Hence to differentiate between the two, the tilde sign is used.
  3. It cannot return or accept any value.
  4. It is automatically called when the compiler exits from the main program.
- The destructor as discussed can only destroy the memory locations allocated by the constructor and the constructor can allocate memory only using the 'new' operator.
- 'new' operator allocates memory to a pointer. Hence, a pointer is to be declared and using the new operator a memory location can be allocated by the following syntax:

```
Pointer_name = new Data_type;
```

- To destroy this memory allocated by the new operator, we require delete operator. In this case the delete operator must be in the destructor. Hence, the destructor will be destroying the memory allocated. The syntax for the delete operator to destroy the memory space allocated is as shown below :

```
delete Pointer_name;
```

Program 3.6.1 : Write a program to demonstrate the destructor.

```
# include<iostream.h>
# include<conio.h>
class test
{
  int *p;
  public:
  test()
  {
    p=new int;
  }
  void read()
  {
    cout<<"Enter a number";
    cin>>*p;
  }
  void display()
  {
    cout<<"Value="<<*p<<endl;
  }
  ~test()
  {
    delete p;
    cout<<"Destroyed";
  }
};
void main()
{
  clrscr();
```

```
    test t;
    t.read();
    t.display();
    getch();
}
```

Output

```
Enter a number5
Value=5
Destroyed
```

Explanation

- The destructor is the ~test() function. This destructor is automatically called after the end of the execution of the main() function. Hence, in our case after the statement getch(); the destructor is automatically called.
- When the destructor is called, the memory allocated by the constructor is deleted using the delete operator. The statement;

```
cout<<"Destroyed";
```

is then executed.

- Hence, similar to constructor even the destructor is not to be called explicitly. It is automatically called.

3.6.2 finalize() Method

- The C++ programming language had the destructors. But in Java we do not have destructors.
- "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 3.6.2 : 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.

3.7 A Book Shop Inventory

- A book shop maintains his inventory, as in how many books of which subject and publication is purchased, how many are sold, how many is stock etc.
- The following program implements the same.

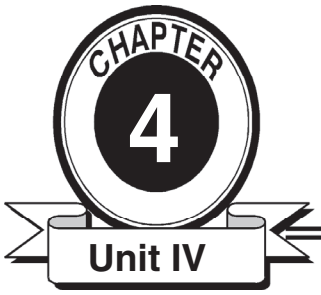
```
import java.util.*;
class Shopping
{
   protected float totalpay,amt,epay,cpay;
   protected int n,input;
   Scanner sc=new Scanner(System.in);
   public void echeck()
   {
      System.out.println("Your total amount to be paid for engineering books="+epay);
   }
   public void ccheck()
   {
      System.out.println("Your total amount to be paid for commerce books="+cpay);
   }
   public void payment(Engineering x)
   {
      totalpay=x.epay+cpay;
      System.out.println("Your grand total="+totalpay);
      System.out.println("Enter the amount:");
      amt=sc.nextFloat();
      if(amt<totalpay)
System.out.println("Transaction failed...Please enter again");
      else if(amt==totalpay)
System.out.println("Transaction successful,Delivery would be done in 5-10 Business days....Balance to be returned="+(amt-totalpay));
   else
{
System.out.println("Transaction successful,delivery would be done in 5-10 business days....Balance to be returned="+(amt-totalpay));
   }
}
class Engineering extends Shopping
{
   private float epay1,epay2,epay3;
   private int ecount = 0, ecount1 = 0, ecount 2 = 0, estock1, estock2, estock3;
   Scanner sc=new Scanner(System.in);
   public void buyengineer()
   {
      do
      {
   System.out.println("Subject\tPublications\tCost");
System.out.println("1.Programming Combo(C,C++,Java,Python,HTML)\tTechnowledge\t3000 INR\n2.Engineering Mathematics Combo(Sem1,,Sem2,Sem3)\tTechknowledge\t1500 INR\n3.Engineering Graphics\tN.H.Dubey\t2500 INR\n4.Process payment\n5.exit");
System.out.println("Enter your choice:");
         input=sc.nextInt();
         switch(input)
         {
            case 1:ecount++;
System.out.println("Book- Programming Combo");
               if(ecount==1)
               {
   System.out.println("Stock left=5");
   System.out.println("Enter the quantity:");
   n=sc.nextInt();
if(n>5) System.out.println("Sorry,not much left in stock");
                  else if(n<=5)
                  {
                     epay1=3000*n;
                     estock1=5-n;
                  }
               }
               else if(ecount>1)
               {
                  if(estock1<=0)
System.out.println("Sorry,out of stock");
                  else
                  {
   System.out.println("Stock left="+estock1);
   System.out.println("Enter the quantity:");
                     n=sc.nextInt();
                     if(n>estock1)
System.out.println("Sorry,not much left in stock");
                     else if(n<=estock1)
                     {
```

```
                    epay1=3000*n;
                    estock1=estock1-n;
                }
            }
        }
        break;
        case 2:ecount1++;
System.out.println("Book-Engineering Mathematics-1");
            if(ecount1==1)
            {
  System.out.println("Stock left=5");
System.out.println("Enter the quantity:");
                n=sc.nextInt();
if(n>5) System.out.println("Sorry,not much left in stock");
                else if(n<=5)
                {
                    epay2=1500*n;
                    estock2=5-n;
                }
            }
            else if(ecount1>1)
            {
                if(estock2==0)
System.out.println("Sorry,out of stock");
                else
                {
System.out.println("Stock left="+estock2);
System.out.println("Enter the quantity:");
                    n=sc.nextInt();
                    if(n>estock2)
System.out.println("Sorry,not much left in stock");
                    else if(n<=estock2)
                    {
                        epay2=1500*n;
                        estock2=estock2-n;
                    }
                }
            }
        break;
        case 3:ecount2++;
System.out.println("Book-Engineering Graphics");
            if(ecount2==1)
            {
  System.out.println("Stock left=5");
  System.out.println("Enter the quantity:");
                    n=sc.nextInt();
                    if(n>5)
System.out.println("Sorry,not much left in stock");
                    else if(n<=5)
                    {
                        epay3=2500*n;
                        estock3=5-n;
                    }
                }
                else if(ecount2>1)
                {
                    if(estock3==0)
System.out.println("Sorry,out of stock");
                    else
                    {
System.out.println("Stock left="+estock3);
                        System.out.println("Enter the quantity:");
                        n=sc.nextInt();
                        if(n>estock3)
System.out.println("Sorry,not much left in stock");
                        else if(n<=estock3)
                        {
                            epay3=2500*n;
                            estock3=estock3-n;
                        }
                    }
                }
            break;
            case 4:epay=epay1+epay2+epay3;
                epay1=0;
                epay2=0;
                epay3=0;
   System.out.println("Payment has been processed successfully(Press appropriate key to check and make payment)");
            break;
case 5:System.out.println("Thank you for shopping");
            break;
            default:System.out.println("Invalid input");
            break;
          }
      }while(input!=5);
   }
}
class Commerce extends Shopping
```

```
{
   private float cpay1,cpay2,cpay3;
   private int cstock1 ,cstock2, cstock3, ccount = 0,ccount1
= 0,ccount2 = 0;
   Scanner sc=new Scanner(System.in);
   public void buycomm()
   {
     do
     {
   System.out.println("Subject\tPublications\tCost");
   System.out.println("1.Business & service sector\tmanan
public\t140 INR\n2.Marketing & Human Resource
development\tSheth Publics\t120 INR\n3.Advertising\tVipul
publics\t110 INR\n4.Process payment\n5.exit");
        System.out.println("Enter your choice:");
        input=sc.nextInt();
        switch(input)
        {
           case 1:ccount++;
   System.out.println("Book-Business & service sector";
              if(ccount==1)
              {
        System.out.println("Stock left=5");
           System.out.println("Enter the quantity:");
                    n=sc.nextInt();
     if(n>5)
System.out.println("Sorry,not much left in stock");
                    else if(n<=5)
                    {
                       cpay1=140*n;
                       cstock1=5-n;
                    }
              }
              else if(ccount1>1)
              {
                 if(cstock1==0)
System.out.println("Sorry,out of stock");
                 else
                 {
   System.out.println("Stock left="+cstock1);
   System.out.println("Enter the quantity:");
                    n=sc.nextInt();
                    if(n>cstock1)
System.out.println("Sorry,not much left in stock");
                    else if(n<=cstock1)
                    {
                       cpay1=140*n;
                       cstock1=cstock1-n;
                    }
                 }
              }
           break;
           case 2:ccount1++;
System.out.println("Book-Marketing & Human Resource
development");
              if(ccount1==1)
              {
   System.out.println("Stock left=5");
   System.out.println("Enter the quantity:");
                    n=sc.nextInt();
                    if(n>5)
System.out.println("Sorry,not much left in stock");
                    else if(n<=5)
                    {
                       cpay2=120*n;
                       cstock2=5-n;
                    }
              }
              else if(ccount1>1)
              {
                 if(cstock2==0)
System.out.println("Sorry,out of stock");
                 else
                 {
   System.out.println("Stock left="+cstock2);
   System.out.println("Enter the quantity:");
                    n=sc.nextInt();
                    if(n>cstock2)
System.out.println("Sorry,not much left in stock");
                    else if(n<=cstock2)
                    {
                       cpay2=120*n;
                       cstock2=cstock2-n;
                    }
                 }
              }
           break;
           case 3:ccount2++;
              System.out.println("Book-Advertising");
              if(ccount2==1)
              {
```

```
    System.out.println("Stock left=5");
    System.out.println("Enter the quantity:");
                    n=sc.nextInt();
                    if(n>5)
System.out.println("Sorry,not much left in stock");
                    else if(n<=5)
                    {
                        cpay3=110*n;
                        cstock3=5-n;
                    }
                }
                else if(ccount2>1)
                {
                    if(cstock3==0)
System.out.println("Sorry,out of stock");
                    else
                    {
    System.out.println("Stock left="+cstock3);
    System.out.println("Enter the quantity:");
                        n=sc.nextInt();
                        if(n>cstock3)
System.out.println("Sorry,not much left in stock");
                        else if(n<=cstock3)
                        {
                            cpay3=110*n;
                            cstock3=cstock3-n;
                        }
                    }
                }
            break;
            case 4:cpay=cpay1+cpay2+cpay3;
                cpay1=0;
                cpay2=0;
                cpay3=0;
                Engineering x=new Engineering();
                epay=x.epay;
System.out.println("Payment has been processed successfully(Press appripriate key to check and make payment)");
            break;
            case 5:
System.out.println("Thank you for shopping");
                break;
                default:System.out.println("Invalid input");
                break;
            }
        }while(input!=5);
    }
}
public class BookInventory
{
    public static void main(String[] args)
    {
        int choice;
        Scanner sc=new Scanner(System.in);
        Engineering e=new Engineering();
        Commerce c=new Commerce();
        System.out.println("Welcome to our shop");
        do
        {
System.out.println("1.Engineering books\n2.Commerce books\n3.check payment history\n4.Make payment\n5.exit");
            System.out.println("Enter your choice:");
            choice=sc.nextInt();
            switch(choice)
            {
                case 1:e.buyengineer();
                break;
                case 2:c.buycomm();
                break;
                case 3:e.echeck();
                    c.ccheck();
                break;
                case 4:c.payment(e);
                break;
case 5:System.out.println("Your happiness is our happiness....#AtmaNirbharBharat");
                break;
                default:System.out.println("Invalid input");
                break;
            }
        }while(choice!=5);
    }
}
```


❑❑❑



Inheritance and Polymorphism

Syllabus

Inheritance : Introduction, Need of Inheritance, Types of Inheritance, Benefits of Inheritance, Cost of Inheritance, Constructors in derived Classes, Method Overriding, Abstract Classes and Interfaces. Polymorphism and Software Reuse: Introduction, Types of Polymorphism (Compile Time and Run Time Polymorphism), Mechanisms for Software Reuse, Efficiency and Polymorphism.

4.1 Inheritance : Introduction and Need

4.1.1 Inheritance

SPPU - May 17, Dec. 17, Dec. 18

Q. Define inheritance and enlist it's types.
Q. What is inheritance ?
(May 17, 3 Marks, Dec. 17, 3 Marks, Dec. 18, 7 Marks)

- 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 while the class which is derived is called as derived class.
- Benefits or need of Ineritance
  1. The main advantage of inheritance is that a code written for a particular class is also accessible to the objects of the classes derieved from this class. This concept is called as code-reusability.
  2. Inheritance implements a structured system of an organization or any system where the programming is to be implemented for.
  3. The attributes (i.e. data members and method members) that are common for many classes, can be written in a base class and accessible to all the classes derieved from the base class.

4.1.1(A) Visibility Modes and Effects

SPPU - May 17, Dec. 17, Dec. 18

Q. Explain different visibility modes used in inheritance.
Q. State different types of inheritance with diagram.
Q. State general format of defining derived class.
Q. State and explain various visibility modifiers in inheritance.
Q. Define derived class. Give one example.
Q. State any two access specifier with example.
Q. Explain various types of inheritance with example
Q. How protected access specifier is different from private?
Q. What are different types of inheritance ?
(May 17, 3 Marks, Dec. 17, 3 Marks, Dec. 18, 7 Marks)

- 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
- These keywords viz. public, protected and private are called as “access specifiers”, as they decide the access of a member.

– The visibility of base class members within the derived class is as shown in Table 4.1.1

Table 4.1.1 : Visibility of base class members in the derived class

Base class member visibility	Derived class visibility		
	Public derivation	Protected derivation	Private derivation
Public members	Public	Protected	Private
Protected members	Protected	Protected	Private
Private members	Not inherited	Not inherited	Not inherited

– Hence, you will notice in the Table 4.1.1, in case of public derivation the public and protected members of the base class remain in the same visibility even in the derived class. The private members are never derived.

– In case of protected derivation the public and protected members of the base class become protected in the derived class. The private members are never derived.

– In case of private derivation the public and protected members of the base class become private in the derived class. The private members are never derived.

– There are various types of inheritance namely :

1. Single Inheritance
2. Multilevel Inheritance
3. Multiple Inheritance
4. Hybrid Inheritance
5. Hierarchical Inheritance

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

```
class derived_class_name : access_specifier
base_class_name
```

– Let us see these types of inheritance in the subsequent sub-sections.

4.1.2 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 5, 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 4.1.2.

Table 4.1.2 : Visibility of Class Members

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 4.1.2, 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.

4.1.3 Cost of Inheritance

- In terms of time, the cost of inheritance is nothing extra. This means that no extra time is required due to inheritance.
- The only thing is that it takes extra memory space as the data members of the base class as well as the derived class.

4.2 Types of Inheritance or Mechanism of Software Reuse in C++

4.2.1 Single Inheritance

- In this case only one class is derived from another class.
- The class diagram and a program example is given below.

Program 4.2.1 : Write a program to add two numbers using single inheritance such that the base class function must accept the two numbers from the user and the derived class function must add these numbers and display the sum.

Solution :

The class diagram of this requirement is as shown in Fig. P. 4.2.1.

This diagram shows that the class "Sum" is derived from class "Data" in public visibility.

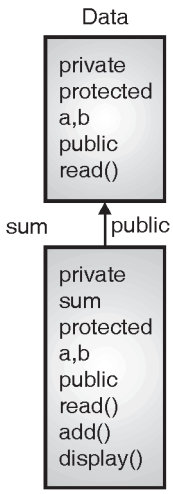


Fig. P. 4.2.1 : Class Diagram of Single Inheritance

for Program 4.2.1

```
# include<iostream.h>
# include<conio.h>
class Data
{
   protected:
   int a,b;
   public:
   void read()
   {
      cout<<"Enter two numbers";
      cin>>a>>b;
   }
};
class Sum: public Data
{
   private:
   int sum;
   public:
   void add()
   {
      sum=a+b;
   }
   void display()
   {
      cout<<"The sum is "<<sum;
   }
};
void main()
{
   clrscr();
   Sum s;
   s.read();
   s.add();
   s.display();
   getch();
}
```

Output

```
Enter two numbers4
5
The sum is 9
```

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 function to calculate the sum.
- The base class also has a function in public visibility to read the inputs from user. The derived class also has public member functions 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: public Data
```

- The members of the base class are shown available in the derived class in the class diagram in Fig. P.4.2.1 but these member functions 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() function has first a creation of the object of the class "Sum". The object of the class sum can access all the functions i.e. read(), add() and display(); because the derived class derives the protected members in the same visibility when derived publicly as discussed in Table 4.1.2.
- The functions to read, calculate the sum and display the sum are thereafter called one by one.

Program 4.2.2 : Write a program to find the area of circle using single inheritance such that the base class function must accept the radius from the user and the derived class function must calculate and display the area.

Solution :

The class diagram of this requirement is as shown in Fig. P. 4.2.2. This diagram shows that the class "Area" is derived from class "Data" in public visibility.

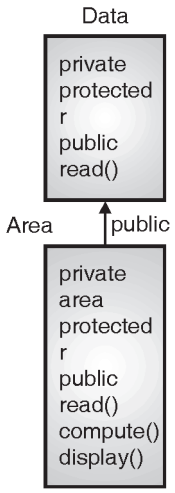


Fig. P. 4.2.2 : Class Diagram of Single Inheritance for Program 4.2.2

```
# include<iostream.h>
# include<conio.h>
class Data
{
   protected:
   int r;
   public:
   void read()
   {
      cout<<"Enter the radius";
      cin>>r;
   }
};
class Area: public Data
{
   private:
   float result;
   public:
   void compute()
   {
      result=3.14*r*r;
   }
   void display()
   {
      cout<<"The area of the circle is "<<result;
   }
};
void main()
{
   clrscr();
   Area a;
   a.read();
   a.compute();
   a.display();
   getch();
}
```

Output

```
Enter the radius5
The area of the circle is 78.5
```

Explanation

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

Program 4.2.3 : Write a program to implement single inheritance from following Fig. P. 4.2.3 accept and display the data for one table.

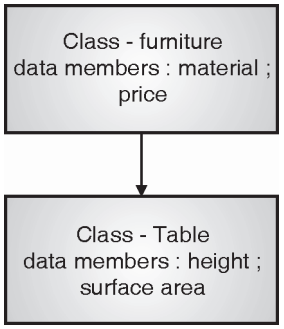


Fig. P. 4.2.3 : Class diagram of single inheritance for Program 4.2.3

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class Furniture
{
   protected:
   char material[20];
```

```
    int price;
};
class Table:public Furniture
{
    private:
    int height,area;
    public:
    void accept()
    {
        cout<<"Enter material, price, height and area:";
        cin>>material>>price>>height>>area;
    }
    void display()
    {

    cout<<"Material:"<<material<<"\nPrice:"<<price<
<"\nHeight:"<<height<<"\nArea:"<<area;
    }
};
void main()
{
    Table t;
    clrscr();
    t.accept();
    t.display();
    getch();
}
```

Output

```
Enter material, price, height and area:Plywood
1500
35
50
Material:Plywood
Price:1500
Height:35
Area:50
```

Program 4.2.4 : Write a program to implement single inheritance. Declare base class 'Employee' with emp_no and emp_name. Declare derived class 'Fitness' with height and weight. Accept and display data for one employee.

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class Employee
{
    protected:
    int no;
    char name[20];
};
class Fitness:public Employee
{
    private:
    int height,weight;
    public:
    void get_data()
    {
        cout<<"Enter name, number height and weight:";
        cin>>name>>no>>height>>weight;
    }
    void display()
    {

    cout<<"Name:"<<name<<"\nNumber:"<<no<<"\n
Height:"<<height<<"\nWeight:"<<weight;
    }
};
void main()
{
    Fitness b;
    clrscr();
    b.get_data();
    b.display();
    getch();
}
```

Output

```
Enter name, number height and weight:Ajay
24
175
67
Name:Ajay
Number:24
Height:175
Weight:67
```

4.2.1(A) Constructors in Derived Class

Program 4.2.5 : Write a program to demonstrate constructor in derived class with respect to order of calling constructor and passing parameters to base class constructor.

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class Base
{
   public:
   Base(int a)
   {
      cout<<"Base class constructor:"<<a<<endl;
   }
};
class Sub:public Base
{
   public:
   Sub(int b):Base(b)
   {
      cout<<"Sub class constructor:"<<b<<endl;
   }
};
void main()
{
   clrscr();
   Sub s(5);
   getch();
}
```

Output

```
Base class constructor:5
Sub class constructor:5
```


4.2.2 Multi Level Inheritance

Q. Differentiate between multiple inheritance and multilevel inheritance.

	Multiple inheritance	Multi level inheritance
Definition	Multiple inheritance is an inheritance type were a class inherits from more than one base class.	Multi level inheritance is an inheritance type that inherits from derived class making that derived class a base class for a new class.
Usage	Multiple inheritance is not widely used because it makes the system more complex.	Multi level inheritance is widely used.
Class level	Multiple inheritance has two class levels namely base class and derived class.	Multi level inheritance has three class levels namely base class intermediate class and derived class.

- 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. 4.2.1(b)

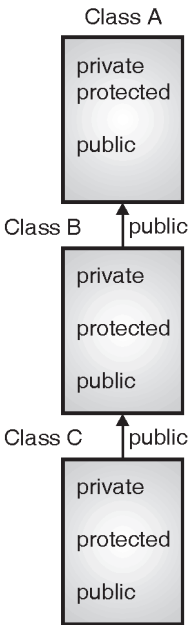


Fig. 4.2.1(a) : Multi level inheritance

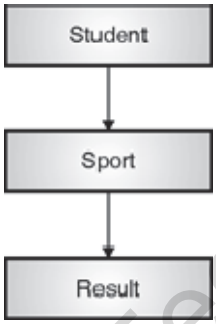


Fig. 4.2.1(b) : Multi level inheritance

- 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.

Program 4.2.6 : Write a program to calculate percentage of a student using multi level inheritance. The base class function will accept the marks in three subjects from user. A class will be derived from the above mentioned class that will have a function to find the total marks obtained and another class derived from this will have functions to calculate and display the percentage scored.

OR Write a C++ program to demonstrate concept of multilevel and hierarchical inheritance.

SPPU - May 19, 6 Marks

Solution :

The class diagram of this requirement is as shown in Fig. P. 4.2.6.

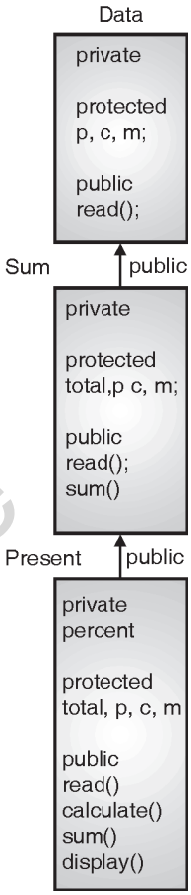


Fig. P. 4.2.6 : Class Diagram of Multi Level Inheritance for Program 4.2.6

```
# include<iostream.h>
# include<conio.h>
class Data
{
  protected:
  int p,c,m;
  public:
  void read()
  {
    cout<<"Enter the marks obtained in Physics, Chemistry and Maths";
    cin>>p>>c>>m;
  }
};
class Sum: public Data
{
  protected:
  int total;
  public:
  void sum()
```

```
    {
      total=p+c+m;
    }
};
class Percent: public Sum
{
   private:
   float percent;
   public:
   void calculate()
   {
      percent=total/300.0*100;
   }
   void display()
   {
      cout<<"The percentage is "<<percent;
   }
};
void main()
{
   clrscr();
   Percent a;
   a.read();
   a.sum();
   a.calculate();
   a.display();
   getch();
}
```

Output

```
Enter the marks obtained in Physics, Chemistry and Maths90
98
99
The percentage is 95.666664
```

Explanation

- The most important part of a program based on inheritance is the class diagram. Once you draw the class diagram then writing the program to implement the diagram is very simple.
- This program is also just an implementation of the class diagram drawn in the Fig. P. 4.2.6. The implementation is similar to that as seen in single inheritance, only that there is one more level of inheritance.

Program 4.2.7 : Write a program to calculate volume of sphere using multi level inheritance. The base class function will accept the radius from user. A class will be derived from the above mentioned class that will have a function to find the area of a circle and another class derived from this will have functions to calculate and display the volume of the sphere.

Solution :

The class diagram of this requirement is as shown in Fig. P. 4.2.7

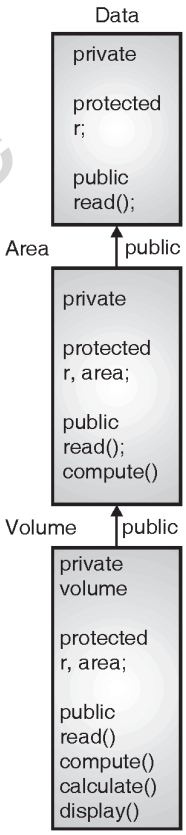


Fig. P. 4.2.7 : Class Diagram of Multi Level Inheritance for Program 4.2.7

```
# include<iostream.h>
# include<conio.h>
class Data
{
   protected:
   float r;
   public:
   void read()
   {
      cout<<"Enter the radius of the circle: ";
      cin>>r;
   }
```

```
};
class Area: public Data
{
   protected:
   float area;
   public:
   void compute()
   {
      area=3.14*r*r;
   }
};
class Volume: public Area
{
   private:
   float volume;
   public:
   void calculate()
   {
      volume=4*area*r/3;
   }
   void display()
   {
      cout<<"The volume of the sphere is "<<volume;
   }
};
void main()
{
   clrscr();
   Volume v;
   v.read();
   v.compute();
   v.calculate();
   v.display();
   getch();
}
```

Output

```
Enter the radius of the circle: 5
The volume of the sphere is 523.333313
```

Explanation

The implementation of the class diagram shown in Fig. P. 4.2.7 is done in similar manner as for the previous program.

Program 4.2.8 : Identify the type of inheritance and implement it by writing a program for the following Fig. P. 4.2.8. Assume suitable member functions.

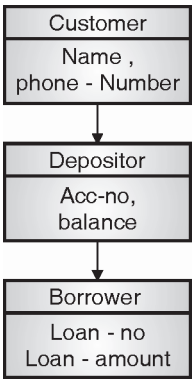


Fig. P. 4.2.8 : Class diagram of multi level inheritance for Program 4.2.8

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class Customer
{
   protected:
   char name[20];
   float phone;
};
class Depositor:public Customer
{
   protected:
   float acc_no,balance;
};
class Burrower:public Depositor
{
   private:
   float loan_no,amount;
   public:
   void accept()
   {
      cout<<"Enter name, phone number,
account number, balance, loan number and loan amount:";
      cin>>name>>phone>>acc_no>>balance
>>loan_no>>
      amount;
   }
   void display()
   {
   cout<<"Name:"<<name<<"\nPhone
```

```
Number:"<<phone
<<"\nAccount
number:"<<acc_no<<"\nBalance:"<<balance
<<"\nLoan number:"<<loan_no<<"\nLoan amount:"
<<amount;
   }
};
void main()
{
   clrscr();
   Burrower b;
   b.accept();
   b.display();
   getch();
}
```

Output

```
Enter name, phone number, account number, balance, loan
number and loan amount:a
bc
123456
111222
100000
45678
200000
Name:abc
Phone Number:123456
Account number:111222
Balance:100000
Loan number:45678
Loan amount:200000
```

Program 4.2.9 : Identify the type of inheritance shown in following Fig. P. 4.2.9. Implement it by using suitable member function.

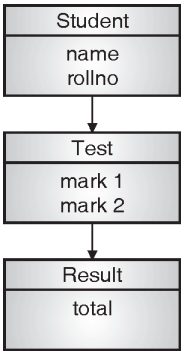


Fig. P. 4.2.9 : Class diagram of multi level inheritance for Program 4.2.9

```
#include<iostream.h>
#include<conio.h>
class student
{
   protected :
      int roll_no ;
   public :
      void get_no (int) ;
      void put_no (void) ;
} ;
void student :: get_no ( int a)
{
   roll_no = a ;
}
void student :: put_no ( )
{
   cout << "Roll Number :" << roll_no << "\n" ;
}
class test : public student       // 1st level derivation
{
   protected :
      float sub1 ;
      float sub2;
   public :
      void get_marks (float, float);
      void put_marks (void);
} ;
void test :: get_marks (float x, float y)
{
   sub1 = x ;
   sub2 = y ;
}
void test :: put_marks ( )
{
   cout << "Marks in subject 1 =" << sub1 << "\n" ;
   cout << "Marks is subject 2 =" << sub2 << "\n" ;
}
class result : public test              // 2nd level derivation
{
      float total ;
   public :
      void display (void) ;
} ;
void result :: display ( )
```

```
{
   total = sub1 + sub2 ;
   put_no ( ) ;
   put_marks ( ) ;
   cout << "Total :" << total  << "\n" ;
}
main ( )
{
   result student1 ;          // student is created
   clrscr( );
   student1.get_no (10) ;
   student1.get_marks (75.0 , 59.5) ;
   student1.display ( ) ;
   getch ( );
   return 0;
}
```

Output

```
Roll Number :10
Marks in subject 1 =75
Marks is subject 2 =59.5
Total :134.5
```

4.2.3 Multiple Inheritance

Q. Describe multiple inheritance with suitable example. **(4 Marks)**

Q. Differentiate between multiple inheritance and multilevel inheritance. **(4 Marks)**

Q. Explain Multiple inheritance. **(4 Marks)**

- In this case one class is derived from multiple classes.
- The class diagram of such a inheritance can be as shown in Fig. 4.2.2.

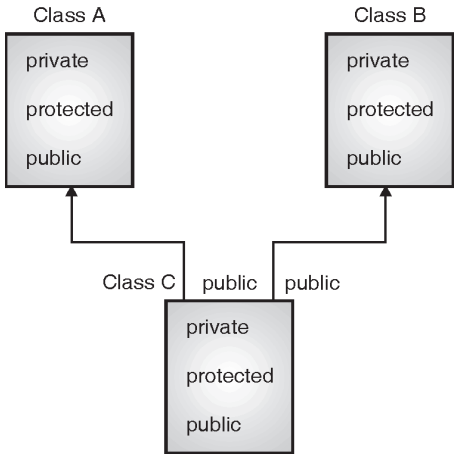


Fig. 4.2.2 : Multiple inheritance

- In this case, class 'C' is derived from class 'B' as well as from class 'A'.
- To derive a class from multiple classes we need to use the following syntax :

```
class derived_class_name : access_specifier
base_class_name,access_specifier
base_class_name2,access_specifier base_class_name3.
```

- Hence, as seen from the above syntax, a class can be derived from whatever number of classes as required.
- Let us see some program examples using multiple inheritance.

Program 4.2.10 : Write a program to define the following relationship using multiple inheritance.

SPPU - May 17, 4 Marks

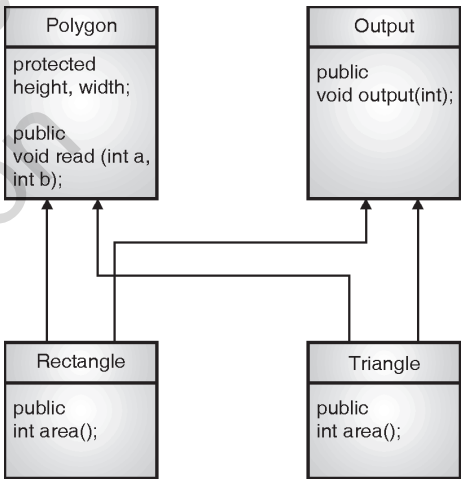


Fig. P. 4.2.10 : Class Diagram of Multiple Inheritance for Program 4.2.10

```
# include<iostream.h>
# include<conio.h>
class Polygon
{
   protected:
   int height, width;
   public:
   void read(int a, int b)
   {
      height=a;
      width=b;
   }
};
class Output
{
```

```
    public:
    void output(int x)
    {
       cout<<"Area is "<<x;
    }
};
class Rectangle: public Polygon, public Output
{
    public:
    int area()
    {
       return(height*width);
    }
};
class Triangle: public Polygon, public Output
{
    public:
    int area()
    {
       return(height*width/2);
    }
};
void main()
{
    clrscr();
    int h,w,choice,a;
    cout<<"1. Area of Rectangle
      \n2. Area of Triangle\nEnter your choice:";
    cin>>choice;
    cout<<"Enter height and width:";
    cin>>h>>w;
    switch(choice)
    {
       case 1:
       Rectangle r;
       r.read(h,w);
       a=r.area();
       r.output(a);
       break;
       case 2:
       Triangle t;
       t.read(h,w);
       a=t.area();
       t.output(a);
       break;
       default: cout<<"Invalid choice";
    }
    getch();
}
```

Output

```
1. Area of Rectangle
2. Area of Triangle
Enter your choice:2
Enter height and width:5
4
Area is 10
```

Explanation

- The most important part of a program based on inheritance is the class diagram. Once you draw the class diagram then writing the program based on the diagram is very simple.
- Here, also we have just implemented the class diagram drawn in the Fig. P. 4.2.10.

4.2.4 Hybrid Inheritance

Q. What is hybrid inheritance ? Give one example. **(4 Marks)**

- In this case there is a mixture of multi level and multiple inheritances. Hence the name is hybrid inheritance.
- The class diagram of such a inheritance can be as shown in Fig. 4.2.3.

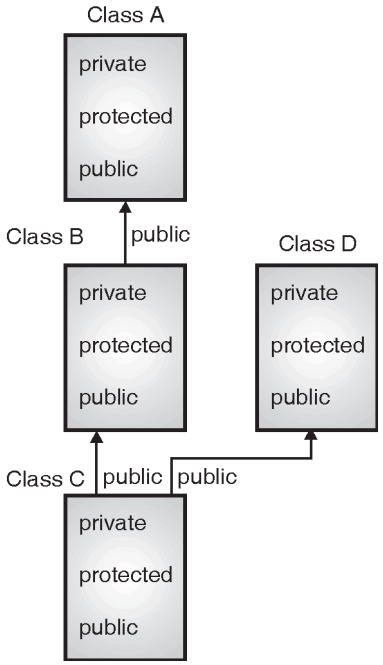


Fig. 4.2.3 : Hybrid inheritance

- In this case, class 'C' is derived from class 'B' which is derived from class 'A'; hence, multi level inheritance. Class 'C' is also derived from class 'D'; hence, multiple inheritance.
- To derive a class from multiple classes we need to use the following syntax:

```
class derived_class_name : access_specifier
base_class_name,access_specifier base_class_name,
access_specifier base_class_name…
```

- Let us see some program examples using hybrid inheritance.

Program 4.2.11 : Write a program to define the following relationship using hybrid inheritance.

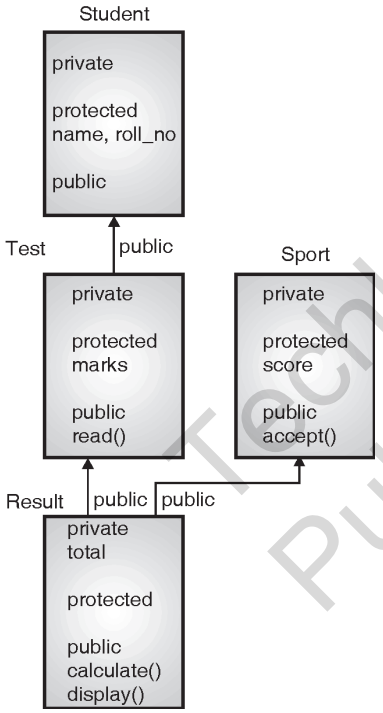


Fig. P. 4.2.11 : Class Diagram of Hybrid Inheritance for Program 4.2.11

```
# include<iostream.h>
# include<conio.h>
# include<stdio.h>
class Student
{
   protected:
   char name[20];
   int roll_no;
};
class Test: public Student
{
   protected:
   int marks;
   public:
   void read()
   {
      cout<<"Enter name, roll number and marks
   obtained:";
      gets(name);
      cin>>roll_no>>marks;
   }
};
class Sports
{
   protected:
   int score;
   public:
   void accept()
   {
      cout<<"1. Student has won in national sports
event\n2. Student has not won in any national sports
event\nEnter your choice:";
      cin>>score;
   }
};
class Result: public Test, public Sports
{
   int total;
   public:
   void calculate()
   {
      if(score==1)
      total=marks+15;
      else
      total=marks;
   }
   void display()
   {
      cout<<"The total is "<<total;
   }
};
void main()
{
   clrscr();
   Result r;
   r.read();
```

```
    r.accept();
    r.calculate();
    r.display();
    getch();
}
```

Output

```
Enter name, roll number and marks obtained:Sandeep
24
81
1. Student has won in national sports event
2. Student has not won in any national sports event
Enter your choice:1
The total is 96
```

4.2.5 Problem in Multiple and Hybrid Inheritance

- When deriving a class from multiple classes i.e. in Multiple Inheritance or Hybrid Inheritance, there is a problem. The members of the two base classes from which a class is derived, may be same. Hence, there is an ambiguity to which class function is called by the object of the derived class.
- This ambiguity is resolved by using the class name along with the function name when calling it.
- This problem and its resolution is shown in the Program 4.2.12 below.

Program 4.2.12 : Write a program to define the following relationship using hybrid inheritance.

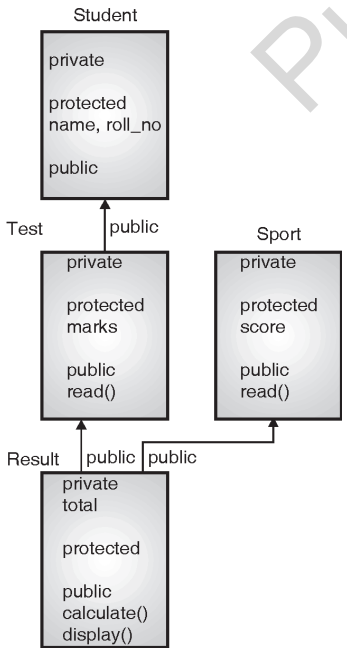


Fig. P. 4.2.12 : Class Diagram of Hybrid Inheritance for Program 4.2.12

```
# include<iostream.h>
# include<conio.h>
# include<stdio.h>
class Student
{
   protected:
   char name[20];
   int roll_no;
};
class Test: public Student
{
   protected:
   int marks;
   public:
   void read()
   {
      cout<<"Enter name, roll number and marks obtained:";
      gets(name);
      cin>>roll_no>>marks;
   }
};
class Sports
{
   protected:
   int score;
   public:
   void read()
   {
      cout<<"1. Student has won in national sports event\n2. Student has not won in any national sports event\nEnter your choice:";
      cin>>score;
   }
};
class Result: public Test, public Sports
{
   int total;
   public:
   void calculate()
   {
      if(score==1)
      total=marks+15;
      else
      total=marks;
```

```
    }
    void display()
    {
       cout<<"The total is "<<total;
    }
};
void main()
{
    clrscr();
    Result r;
    r.Test::read();
    r.Sports::read();
    r.calculate();
    r.display();
    getch();
}
```

Output

```
Enter name, roll number and marks obtained:Satish
24
96
1. Student has won in national sports event
2. Student has not won in any national sports event
Enter your choice:2
The total is 96
```

Explanation

- You will notice in this program both the classes namely "test" and "Sports" have a member function with the same name i.e. read().
- The resolution to the problem can be seen in the main() function where the two functions are called with specifying the class name along with the function name.

4.2.6 Hierarchical Inheritance

Q. Illustrate the hierarchical inheritance. **(4 Marks)**

Q. How hierarchical inheritance is achieved, explain with example. **(4 Marks)**

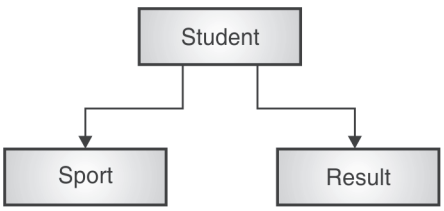


Fig. 4.2.4 : Hierarchical Inheritance

- 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 4.2.13 : Write a program to implement inheritance as shown in Fig. P. 4.2.13. Assume suitable member function.

OR Write a C++ program to demonstrate concept of multilevel and hierarchical inheritance.

SPPU - May 19, 6 Marks

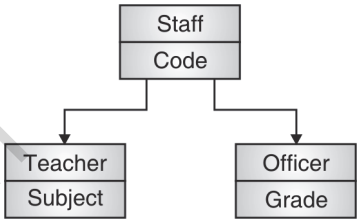


Fig. P. 4.2.13

```
# include<iostream.h>
# include<conio.h>
# include<stdio.h>
class Staff
{
   protected:
   char name[20];
   int code;
};
class Teacher: public Staff
{
   private:
   char subject[20];
   int experience;
   public:
   void read()
   {
      cout<<"Enter name, code, subject and experience of the teacher:";
      gets(name);
      cin>>code;
      gets(subject);
```

```
    cin>>experience;
  }
  void display()
  {
    cout<<"Teacher
Details:\nName:"<<name<<"\nCode:"<<code
<<"\nSubject:"<<subject<<"\nExperience:"
<<experience;
}
};
class Officer: public Staff
{
  private:
  char dept[20];
  int grade;
  public:
  void read()
  {
    cout<<"Enter name, code, department and grade of
the officer:";
    gets(name);
    cin>>code;
    gets(dept);
    cin>>grade;
  }
  void display()
  {
    cout<<"Officer
    Details:\nName:"<<name<<"\nCode:"<<code
  <<"\nDepartment:"<<dept<<"\nGrade:"<<grade;
  }
};
void main()
{
  clrscr();
  int choice;
  cout<<"1. Teacher\n2. Officer\nEnter the choice, whose
details you want to enter:";
  cin>>choice;
  switch(choice)
  {
    case 1:Teacher t;
    t.read();
    t.display();
    break;
    case 2:Officer o;
    o.read();
    o.display();
    break;
    default:cout<<"Invalid choice";
  }
  getch();
}
```

Output

```
1. Teacher
2. Officer
Enter the choice, whose details you want to enter:1
Enter name, code, subject and experience of the
teacher:Satish
234
Maths
13
Teacher Details:
Name:Satish
Code:234
Subject:Maths
Experience:13
```

Program 4.2.14 : Write a program to define the following inheritance relationship.

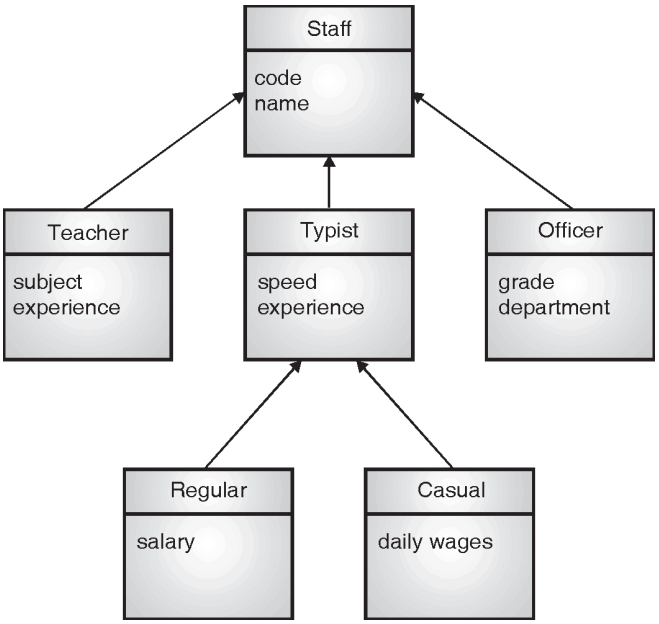


Fig. P. 4.2.14 : Class Diagram of Hierarchical Inheritance for Program 4.2.14

```
# include<iostream.h>
# include<conio.h>
# include<stdio.h>
class Staff
{
   protected:
   char name[20];
   int code;
};
class Teacher: public Staff
{
   private:
   char subject[20];
   int experience;
   public:
   void read()
   {
      cout<<"Enter name, code, subject and experience of
the teacher:";
      gets(name);
      cin>>code;
      gets(subject);
      cin>>experience;
   }
   void display()
   {
      cout<<"Teacher
Details:\nName:"<<name<<"\nCode:"<<code
<<"\nSubject:"<<subject<<"\nExperience:"
<<experience;
   }
};
class Officer: public Staff
{
   private:
   char dept[20];
   int grade;
   public:
   void read()
   {
      cout<<"Enter name, code, department and grade of
the officer:";
      gets(name);
      cin>>code;
      gets(dept);
      cin>>grade;
   }
   void display()
   {
      cout<<"Officer
      Details:\nName:"<<name<<"\nCode:"<<code
   <<"\nDepartment:"<<dept<<"\nGrade:"<<grade;
   }
};
class Typist: public Staff
{
   protected:
   int speed,experience;
};
class Regular: public Typist
{
   private:
   int salary;
   public:
   void read()
   {
      cout<<"Enter name, code, speed, experience and
salary of the regular typist:";
      gets(name);
      cin>>code>>speed>>experience>>salary;
   }
   void display()
   {
      cout<<"Regular Typist
Details:\nName:"<<name<<"\nCode:"<<code<<"\nSpe
ed:"<<speed<<"\nExperience:"<<experience<<"\nSal
ary:"<<salary;
   }
};
class Casual: public Typist
{
   private:
   int dailywages;
   public:
   void read()
```

```
    {
        cout<<"Enter name, code, speed, experience and dailywages of the Casual typist:";
        gets(name);
        cin>>code>>speed>>experience>>dailywages;
    }
    void display()
    {
        cout<<"Casual Typist Details:\nName:"<<name<<"\nCode:"
<<code<<"\nSpeed:"<<speed<<"\nExperience:"
<<experience<<"\nDaily Wages:"<<dailywages;
    }
};
void main()
{
    clrscr();
    int choice;
    cout<<"1. Teacher\n2. Officer\n3. Regular Typist \n4. Casual Typist\nEnter the choice, whose details you want to enter:";
    cin>>choice;
    switch(choice)
    {
        case 1:Teacher t;
        t.read();
        t.display();
        break;
        case 2:Officer o;
        o.read();
        o.display();
        break;
        case 3:Regular r;
        r.read();
        r.display();
        break;
        case 4:Casual c;
        c.read();
        c.display();
        break;
        default:cout<<"Invalid choice";
    }
    getch();
}
```

Output

```
1. Teacher
2. Officer
3. Regular Typist
4. Casual Typist
Enter the choice, whose details you want to enter:1
Enter name, code, subject and experience of the teacher:Satish
234
Maths
13
Teacher Details:
Name:Satish
Code:234
Subject:Maths
Experience:13
```

4.3 Types of Inheritance or Mechanism of Software Reuse in Java

4.3.1 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.4.3.1.

Program 4.3.1 : 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. 4.3.1. This diagram shows that the class “Sum” is derived from class “Data”.

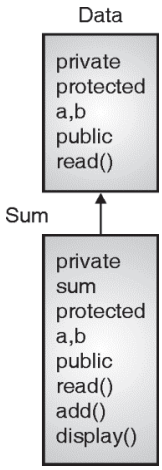


Fig. P. 4.3.1 : Class Diagram of Single Inheritance

```
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. 4.3.1 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 Table 4.1.2
- The methods read(), add() and display() are thereafter called one by one.

Program 4.3.2 : 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. 4.3.2. This diagram shows that the class "Area" is derived from class "Data".

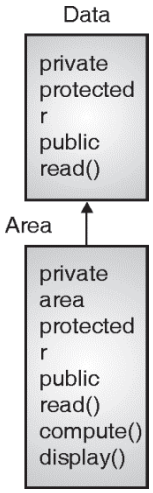


Fig. P. 4.3.2 : Class Diagram of Single Inheritance for Program 4.3.2

```
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.

4.3.2 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. 4.3.1.
- 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.

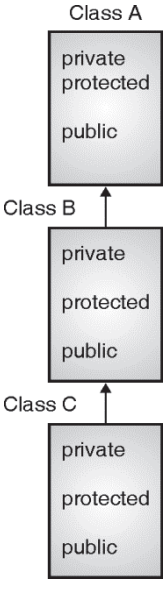


Fig. 4.3.1 : Multi level inheritance

Program 4.3.3 : 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. 4.3.3

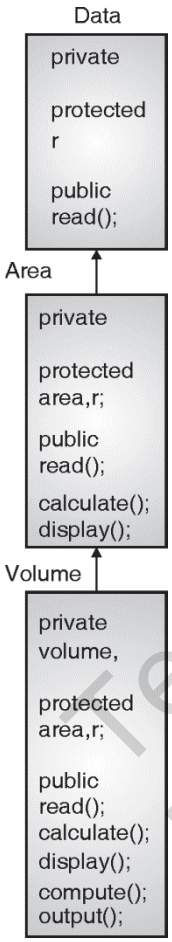


Fig. P. 4.3.3 : Class Diagram of Multi Level Inheritance for Program 4.3.3

```
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. 4.3.3 is done in similar manner as for the previous program.

Program 4.3.4 : 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.

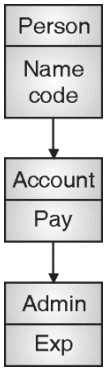


Fig. P. 4.3.4 : Class Diagram of Multi Level Inheritance for Program 4.3.4

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+"\nP
ay:"+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. 4.3.4 is done in similar manner as for the previous program.

4.3.3 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 4.3.5 : Write a program to define the following inheritance relationship.

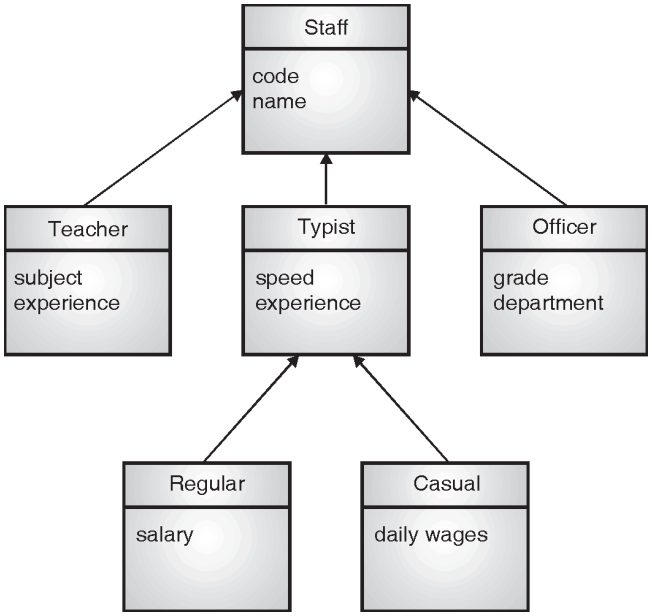


Fig. P. 4.3.5 : Class Diagram of Hierarchical Inheritance for Program 4.3.5

```
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("Teacher Details: \nName: "+name + "\nCode: "+code + "\nSubject:" + subject + "\n Experience:" + 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
```

Program 4.3.6 : Assume that a bank maintains two kinds of accounts for its customers. The saving account provides compound interest and withdrawal facilities but no cheque book facility. The current account provides cheque book facility but no interest. Current account holders should also maintain a minimum balance and if the balance falls below this level a service charge is imposed. Create a class account that stores customer name, account number and type of account. From this derive the classes current-acct and savings-acct to make them more specific to their requirements include the necessary methods in order to achieve the following tasks : (i) Accept deposit from a customer and update the balance (ii) Display the balance (iii) Compute the deposit interest (iv) Permit withdrawal and update the balance (v) Check the minimum balance, impose penalty if necessary, and update the balance.

```
import java.util.*;
class Bank
{
   protected String name;
   protected int acc_no;
   protected String type;
   protected boolean cheque;
   protected float penalty;
   protected float min_bal;
   protected float balance;
   protected float interest;
}
class Current_acc extends Bank
{
   public void read()
   {
      Scanner sc = new Scanner (System.in);
      System.out.println("Enter name, account number, penalty, minimum balance and opening balance of the current account holder:");
      name=sc.next();
      acc_no=sc.nextInt();
      penalty=sc.nextInt();
      min_bal=sc.nextInt();
      balance=sc.nextInt();
      type="Current";
      cheque=true;
      interest=0;
   }
   public void deposit(float x)
   {
      balance+=x;
   }
   public void display()
   {
System.out.println("Account Details: \nName:" + name + "\nAccount no:\" + acc_nom + "\nBalance:" + balance + "\nCheque book facility:"+cheque);
   }
   public void withdraw(float x)
   {
      balance-=x;
   }
   public void minimum()
   {
      if(balance<min_bal)
      balance-=penalty;
      System.out.println("Penalty imposed");
   }
}
class Savings_acc extends Bank
{
   public void read()
   {
      Scanner sc = new Scanner (System.in);
System.out.println("Enter name, account number, interest rate and opening balance of the current account holder:");
      name=sc.next();
      acc_no=sc.nextInt();
      penalty=0;
      min_bal=0;
      interest=sc.nextFloat();
      balance=sc.nextInt();
      type="Savings";
      cheque=false;
   }
   public void deposit(float x)
   {
      balance+=x;
   }
   public void display()
   {
      System.out.println("Account Details:\nName:"+name+"\nAccount

   no:"+acc_no+"\nBalance:"+balance+"\nCheque book facility:"+cheque);
   }
   public void withdraw(float x)
   {
      balance-=x;
   }
   public void interestAdd()
   {
      balance=balance+balance*interest;
      System.out.println("Interest Added");
   }
}
```

```
class Main
{
   public static void main(String args[])
   {
      Current_acc c=new Current_acc();
      c.read();
      c.deposit(10000);
      c.withdraw(5000);
      c.display();
      Savings_acc s=new Savings_acc();
      s.read();
      s.deposit(10000);
      s.withdraw(5000);
      s.display();
   }
}
```

Output

```
Enter name, account number, penalty, minimum balance and opening balance of the
current account holder:
Ajay
325
2
10000
25000
Account Details:
Name:Ajay
Account no:325
Balance:30000.0
Cheque book facility:true
Enter name, account number, interest rate and opening balance of the current
account holder:
Vijay
425
6
10000
Account Details:
Name:Vijay
Account no:425
Balance:15000.0
Cheque book facility:false
```

4.4 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
- Let us see a program example of method overriding.

Program 4.4.1 : 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 4.4.1 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".

4.5 Abstract Classes and Interfaces

4.5.1 Abstract Classes

- 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 4.5.1. 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 4.5.1 : Write a program to display volume of sphere and hemisphere. Make use of abstract class

Solution :

```
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 4.5.2 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 4.5.2 : 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.

Program 4.5.3 : Write a abstract class program to calculate area of square and triangle.

Solution :

```
import java.util.*;
abstract class Base{
protected float l,b,area;
abstract void calculate();
public void display()
{
 System.out.println("Area="+area);
}
}
class Square extends Base
{
public void read(float x)
{
 l=x;
}
public void calculate()
{
  area=l*l;
}
}
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("Square:");
 System.out.println("Enter the length of a side:");
 x=sc.nextFloat();
 Square s=new Square();
 s.read(x);
 s.calculate();
 s.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

```
Square:
Enter the length of a side:
10
Area=100.0
Triangle:
Enter height and breadth:
10
10
Area=50.0
```

Explanation

- The implementation program of the is done with same methods read() and calculate() in base abstract class and in sub classes Square 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.

4.5.2 Interface

- 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.

4.5.2(A) 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.

4.5.2(B) 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.

4.5.2(C) 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.
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 4.5.4 : 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.

Solution :

```
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 4.5.5 : Interface Matrix

```
{
final static int M=5,N=5;
void readMatrix();
void displayMatrix();
void addMatrix();
void multMatrix();
void transposeMatrix();
}
```

Implement the above interface using suitable JAVA class program and also develop the main program.

Solution :

```
import java.util.*;
interface Matrix
{
   final static int M=5,N=5;
   void readMatrix() ;
   void displayMatrix();
   void addMatrix();
   void multMatrix();
   void transposeMatrix();
}
class Sub implements Matrix
{
   protected int i,j,k;
      int a[ ][ ]=new int[M][N];
      int b[ ][ ]=new int[M][N];
      int sum[ ][] =new int[M][N];
     int product[ ][ ]=new int[M][N];
      int ta[ ][ ]=new int[M][N];
      int tb[ ][ ]= new int[M][N];
   public void readMatrix()
   {
      Scanner sc = new Scanner(System.in);
      System.out.println("Matrix a:");
      for(i=0;i<=M-1;i++)
      {
         for(j=0;j<=N-1;j++)
         {
```

```
            System.out.print("Enter the element:");
            a[i][j]=sc.nextInt();
        }
    }
    System.out.println("Matrix b:");
    for(i=0;i<=M-1;i++)
    {
        for(j=0;j<=N-1;j++)
        {
            System.out.print("Enter the element:");
            b[i][j]=sc.nextInt();
        }
    }
}
public void displayMatrix()
{
    System.out.println("Matrix A");
    for(i=0;i<=M-1;i++)
    {
        for(j=0;j<=N-1;j++)
        {
            System.out.print(a[i][j]+"\t");
        }
        System.out.println();
    }
    System.out.println("Matrix B");
    for(i=0;i<=M-1;i++)
    {
        for(j=0;j<=N-1;j++)
        {
            System.out.print(b[i][j]+"\t");
        }
        System.out.println();
    }
}
public void addMatrix()
{
    for(i=0;i<=M-1;i++)
    {
        for(j=0;j<=N-1;j++)
        {
            sum[i][j]=a[i][j]+b[i][j];
        }
    }
    System.out.println("The sum of two matrices is");
    for(i=0;i<=M-1;i++)
    {
        for(j=0;j<=N-1;j++)
        {
            System.out.print(sum[i][j]+"\t");
        }
        System.out.println();
    }
}
public void multMatrix()
{
    for(i=0;i<=M-1;i++)
    {
        for(j=0;j<=N-1;j++)
        {
            product[i][j]=0;
            for(k=0;k<=N-1;k++)
            {
                product[i][j]+=a[i][k]*b[k][j];
            }
        }
    }
    System.out.println("The product of two matrices is");
    for(i=0;i<=M-1;i++)
    {
        for(j=0;j<=N-1;j++)
        {
            System.out.print(product[i][j]+"\t");
        }
        System.out.println();
    }
}
public void transposeMatrix()
{
    for(i=0;i<=M-1;i++)
    {
        for(j=0;j<=N-1;j++)
        {
            ta[j][i]=a[i][j];
        }
    }
    for(i=0;i<=M-1;i++)
    {
        for(j=0;j<=N-1;j++)
        {
            tb[j][i]=b[i][j];
```

```
        }
      }
      System.out.println("Transpose of Matrix 1");
      for(i=0;i<=M-1;i++)
      {
        for(j=0;j<=N-1;j++)
        {
          System.out.print(ta[i][j]+"\t");
        }
        System.out.println();
      }
      System.out.println("Transpose of Matrix 2");
      for(i=0;i<=M-1;i++)
      {
        for(j=0;j<=N-1;j++)
        {
          System.out.print(tb[i][j]+ "\t");
        }
        System.out.println();
      }
   }
}
class Main
{
   public static void main(String args[])
   {
      Sub m=new Sub();
      m.readMatrix();
      m.addMatrix();
      m.multMatrix();
      m.transposeMatrix();
   }
}
```

Output

```
Matrix a:
Enter the element:1
Enter the element:2
Enter the element:3
Enter the element:4
Enter the element:5
Enter the element:6
Enter the element:7
Enter the element:8
Enter the element:9
Enter the element:10
Enter the element:11
Enter the element:12
Enter the element:13
Enter the element:14
Enter the element:15
Enter the element:16
Enter the element:17
Enter the element:18
Enter the element:19
Enter the element:20
Enter the element:21
Enter the element:22
Enter the element:23
Enter the element:24
Enter the element:25
Matrix b:
Enter the element:1
Enter the element:2
Enter the element:3
Enter the element:4
Enter the element:5
Enter the element:6
Enter the element:7
Enter the element:8
Enter the element:9
Enter the element:10
Enter the element:11
Enter the element:12
Enter the element:13
Enter the element:14
Enter the element:15
Enter the element:16
Enter the element:17
Enter the element:18
Enter the element:19
Enter the element:20
Enter the element:21
Enter the element:22
Enter the element:23
Enter the element:24
Enter the element:25
The sum of two matrices is
2    4    6    8    10
12   14   16   18   20
```

```
22    24    26    28    30
32    34    36    38    40
42    44    46    48    50
The product of two matrices is
215   230   245   260   275
490   530   570   610   650
765   830   895   960   1025
1040  1130  1220  1310  1400
1315  1430  1545  1660  1775
Transpose of Matrix 1
1     6     11    16    21
2     7     12    17    22
3     8     13    18    23
4     9     14    19    24
5     10    15    20    25
Transpose of Matrix 2
1     6     11    16    21
2     7     12    17    22
3     8     13    18    23
4     9     14    19    24
5     10    15    20    25
```

Explanation :

– The interface "Matrix" is implemented with all its methods defined.

4.6 Polymorphism

Q. Write short notes on: Polymorphism. **(5 Marks)**

Q. Explain with examples how is polymorphism achieved at compile time and run time. **(10 Marks)**

Q. Describe the following features of object oriented programming : Polymorphism. **(10 Marks)**

Q. Distinguish between run-time polymorphism and compile-time polymorphism. **(4 Marks)**

Q. Define polymorphism. List types of polymorphism. **(2 Marks)**

Q. Define polymorphism. Explain any two types with example. **(4 Marks)**

Q. What is static polymorphism? **(2 Marks)**

Q. What is polymorphism ? Enlist different types of polymorphism. What are the differences between them ? **(4 Marks)**

– As discussed in chapter 1 and, polymorphism refers to multiple forms of same thing i.e. function, constructors or operators.

– The chart in Fig. 4.6.1 shows the different types of polymorphism. The chart shows two types of polymorphism viz. static and dynamic. Those operations that take place during the compile time are called as "static". Those operations that take place during the run time or execution time are called as "dynamic".

– Static polymorphism is also referred to as early binding while dynamic polymorphism is referred to as late binding.

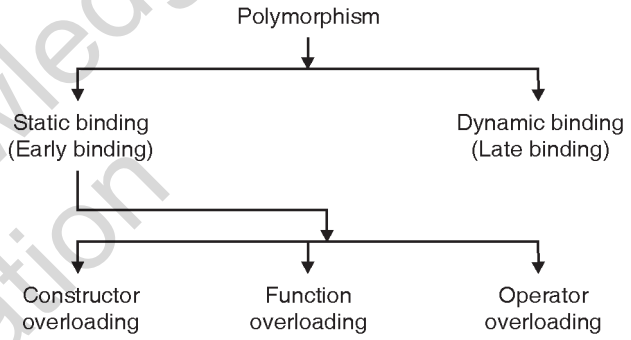


Fig. 4.6.1 : Types of polymorphism

– Static binding or static polymorphism includes Constructor overloading, function overloading and operator overloading i.e. in static polymorphism operators, functions or constructors can take multiple forms or behave differently at different places in the program.

– As already stated, static means compile time; the compiler resolves the reference in this case. Hence, when a function is called and there are multiple functions of the same name, then the compiler decides which function is to be called based on the parameters.

– Thus, the compiler during the compile times resolves the reference. Similarly; it is done for constructor and operator overloading.

- We have already seen the static polymorphism types i.e. constructor overloading, function overloading and operator overloading.
- We will see dynamic polymorphism in the subsequent sub section.

Sr. No.	Compiler Time Polymorphism	Run Time Polymorphism
1.	In compile time polymorphism call is resolved by the compiler.	In run time polymorphism call is not resolved by the compiler.
2.	It is also known as static binding, Early binding and overloading as well.	It is also known as dynamic binding late binding and over riding as well.
3.	Overloading is compile time polymorphism where more than one methods share the same name with different parameters or signature and different return type.	Over riding is run time polymorphism having same method with same parameters or signature but associated in a class and its subclass.
4.	It is achieved by function over loading and operator over loading.	It is achieved by virtual functions and pointers.
5.	It provides fast execution because known early compile time.	It provides slow execution as compare to early binding because it is known at runtime.
6.	Compile time polymorphism is less flexible as all things execute at compile time.	Run time polymorphism is more flexible as all things execute at run time.

4.6.1 Polymorphism in C++

- 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 earlier chapter and method overriding in this chapter earlier. We will still see these concepts again and with the remaining topics also.

4.7 Types of Polymorphism: Compile Time and Run Time Polymorphism

4.7.1 Dynamic Binding using Virtual Function or Compile/Run Time Polymorphism

Q. Write short notes on : Virtual function. **(5 Marks)**

Q. What is function overriding? Give example. **(10 Marks)**

Q. Explain pure virtual function with example. **(5 Marks)**

Q. Explain virtual function with suitable example. **(4 Marks)**

Q. What is virtual function ? Why we need virtual function ? **(4 Marks)**

Q. What is pure virtual functions ? Write down the rules used for pure virtual function. **(4 Marks)**

Q. How to define virtual function ? Give example. **(4 Marks)**

- Dynamic binding of a member function is achieved in C++ with the use of the keyword “virtual”.

– Those member functions that are preceded by the keyword "virtual" are known as virtual functions.

4.7.2 Rules for Virtual Functions

Q.	Why do we need virtual functions ?	**(4 Marks)**
Q.	What is pure virtual function ?	**(2 Marks)**
Q.	Write short note on: virtual function.	**(5 Marks)**
Q.	Explain any four rules for virtual function.	**(4 Marks)**
Q.	What is need of virtual function? Explain with example.	**(4 Marks**

– If a base class function has the same name as that of the derived class then the function called depends on the parameters, this is called as function overloading. But if the base class function and derived class function have same name and same parameter list then it is said to be function overriding.

– If the keyword "virtual" is preceded to the header line of the function in the base class then the binding of such a function with its call becomes dynamic i.e. runtime. Thus, if this function is called then whether the base class function is to be invoked or the subclass function is to be invoked is decided during the runtime. This is called as dynamic binding or late binding.

– The question arises, why pure virtual functions? The reason is that, in practical applications the base class member functions are rarely used to perform any function; in such cases the functions are to be declared as pure virtual function and their definitions in the derived classes.

– Virtual function may be a friend function of another class.

– The virtual functions must be non-static members of same class accessed by using object pointers only. The object pointer must be of the base class i.e. the object pointer of the base class can access the object of derived class but vice versa is not true.

– If a function in a base class is left blank i.e. without any definition and only declaration then;

1. The function must be declared "virtual".
2. This function is called as "pure virtual function" or "do nothing function" or "dummy function".
3. This function must be overridden in the sub class i.e. the function with same name and parameters must be declared and defined in the sub class of the virtual class.
4. A pure virtual function can be defined with the following syntax:

   virtual return_type function_name () =0;

   For e.g.

   virtual void read() = 0;

– Let us see examples for each of these concepts i.e.

1. Function overriding
2. Virtual function
3. Pure virtual function

Program 4.7.1 : Write a program to demonstrate function overriding.

```
#include<iostream.h>
#include<conio.h>
class Base
{
   protected:
   int a,b;
   public:
   void read()
   {
      cout<<"Enter two values:";
      cin>>a>>b;
   }
   void display()
   {
      cout<<"The values are:"<<a<<"\n"<<b;
   }
};
class Sub: public Base
{
```

```
    protected:
    int c,d;
    public:
    void read()
    {
       cout<<"Enter 4 values:";
       cin>>a>>b>>c>>d;
    }
    void display()
    {
cout<<"The values are:"
    <<a<<"\n"<<b<<"\n"<<c<<"\n"<< d;
    }
};
void main()
{
    clrscr();
    Sub s;
    s.read();
    s.display();
    getch();
}
```

Output

```
Enter 4 values:4
3
2
1
The values are:4
3
2
1
```

Explanation

- The class "Sub" is derived from the class "Base". The functions read() and display() are declared and defined in the base class and derived class.
- The name and parameter list of the functions in base class and derived class are same. This is called as function overriding.
- In this case, the functions in the class "Sub" override the functions read() and display() of the class "Base".
- Hence when the object of the class "Sub", created in the main() function will call the respective functions of the sub class and not the base class.

4.7.3 Pointer to Derived Class

Program 4.7.2 : Write a program to demonstrate dynamic binding using virtual function.

```
#include<iostream.h>
#include<conio.h>
class Base
{
   protected:
   int a,b;
   public:
   virtual void read()
   {
      cout<<"Enter two values:";
      cin>>a>>b;
   }
   virtual void display()
   {
      cout<<"\nThe values
are:"<<a<<"\n"<<b<<endl;
   }
};
class Sub: public Base
{
   protected:
   int c,d;
   public:
   virtual void read()
   {
      cout<<"Enter 4 values:";
      cin>>a>>b>>c>>d;
   }
   virtual void display()
   {
      cout<<"\nThe values are:"
   <<a<<"\n"<<b<<"\n"<<c<<"\n"<<d<<endl;
   }
};
void main()
{
```

```
    clrscr();
    Base *ptr;
    Base b;
    Sub s;
    ptr=&b;
    ptr->read();
    ptr->display();
    ptr=&s;
    ptr->read();
    ptr->display();

    getch();
}
```

Output

```
Enter 2 values:1
2

The values are:1
2
Enter 4 values:1
2
3
4

The values are:1
2
3
4
```

Explanation

- The class "Sub" is derived from the class "Base". The functions read() and display() are declared and defined in the base class and derived class.
- The name and parameter list of the functions in base class and derived class are same. This is called as function overriding.
- But, here the functions are preceded by the keyword "virtual", hence the binding with the call to these functions are done during the runtime. This is called as dynamic binding or late binding.
- In the main() function, we have made objects for both the classes and a pointer for the base class.
- The base class pointer initially points to the object of the base class and hence when the functions are called, the base class functions are invoked.
- For the second time the same pointer is pointed to the object of the sub class and hence at this time when the functions are called, the derived class functions are invoked.
- This can also be seen from the output and comparing it with the program. The same pointer when calls the function at different places calls different functions decided during the run time.

Program 4.7.3 : Write a program to demonstrate pure virtual function.

```
#include<iostream.h>
#include<conio.h>
class Base
{
   protected:
   int a,b;
   public:
   virtual void read()
   {
   }
   virtual void display() = 0;
};
class Sub: public Base
{
   protected:
   int c,d;
   public:
   void read()
   {
      cout<<"Enter 4 values:";
      cin>>a>>b>>c>>d;
   }
   void display()
   {
      cout<<"The values are:"
         <<a<<"\n"<<b<<"\n"<<c<<"\n"<<d;
   }
};
void main()
```

```
{
   clrscr();
   Sub s;
   Base *ptr;
   ptr=&s;
   ptr->read();
   ptr->display();
   getch();
}
```

Output

```
Enter 4 values:1
2
3
4
The values are:1
2
3
4
```

Explanation

- Here the functions in the "Base" class are defined without any body and are preceded by the keyword "virtual", hence the binding with the call to these functions are done during the runtime. This is called as dynamic binding or late binding. Such functions without any body and overridden in the derived class are called as "pure virtual function".
- As already discussed pure virtual functions are used to create a template wherein all the classes derived from the class that has a pure virtual function must have the same function defined with its body. If not defined than again it has to be declared as virtual and defined with a blank body as shown in program 4.10.3.
- The remaining things are similar to that of the previous program.

Program 4.7.4 : Create class shape. Derive two classes Triangle and Rectangle. Accept dimensions of Triangle and Rectangle with appropriate functions. Make area() function virtual which is common to all classes. With area function calculate area of triangle and rectangle. Display the result.

```
#include<iostream.h>
#include<conio.h>
class Shape
{
   protected:
   int a,b,area;
   public:
   virtual void read()
   {
   }
   virtual void calc_area()
   {
   }
   void display()
   {
      cout<<"Area="<<area<<endl;
   }
};
class Triangle: public Shape
{
   public:
   void read()
   {
      cout<<"Enter base and height";
      cin>>a>>b;
   }
   void calc_area()
   {
      area=a*b/2;
   }
};
class Rectangle: public Shape
{
   public:
   void read()
   {
      cout<<"Enter length and breadth";
      cin>>a>>b;
   }
   void calc_area()
   {
      area=a * b;
   }
};
```

```
void main()
{
   clrscr();
   Triangle t;
   Rectangle r;
   Shape *ptr;
   ptr=&t;
   ptr->read();
   ptr->calc_area();
   ptr->display();
   ptr=&r;
   ptr->read();
   ptr->calc_area();
   ptr->display();
   getch();
}
```

Output

```
Enter base and height2
10
Area=10
Enter length and breadth5
4
Area=20
```

4.7.4 Virtual Base Class and Abstract Class

- A class can be inherited as a virtual class by another class. In this case when the base class is inherited by another class as virtual class, then the base class is called a virtual base class.
- To inherit a class as virtual, we need to mention the keyword "virtual" while deriving the class. The syntax for virtual derivation of a class is as given below:

```
class sub_class : public virtual base_class;
```

- For example in the program 4.7.3, we can write class Sub: virtual public Base;
- In the syntax mentioned above the keywords "virtual" and public" are interchangeable. Hence the syntax can also be written as:

```
class sub_class : virtual public base_class;
```

- A class which is used only as a virtual derivation and no object is made of that class is called as abstract class i.e. it is not used to create objects.
- Abstract classes are used as a template for the designing purposes.

Program 4.7.5 : Write a program to demonstrate virtual class or abstract class.

```
#include<iostream.h>
#include<conio.h>
class Base
{
   protected:
   int a,b;
   public:
   void read()
   {
      cout<<"Enter two values:";
      cin>>a>>b;
   }
   void display()
   {
      cout<<"The values are:"<<a<<"\n"<<b;
   }
};
class Sub: public virtual Base
{
   protected:
   int c,d;
   public:
   void read()
   {
      cout<<"Enter 4 values:";
      cin>>a>>b>>c>>d;
   }
   void display()
   {
```

```
    cout<<"The values are:"
        <<a<<"\n"<<b<<"\n"<<c<<"\n"<< d;
  }
};
void main()
{
  clrscr();
  Sub s;
  s.read();
  s.display();
  getch();
}
```

Output

```
Enter 4 values:4
3
2
1
The values are:4
3
2
1
```

Explanation

- The class “Sub” is derived from the class “Base”. The functions read() and display() are declared and defined in the base class and derived class. The name and parameter list of the functions in base class and derived class are same. This is called as function overriding.
- In this case, the functions in the class “Sub” override the functions read() and display() of the class “Base”.
- The keyword "virtual" while deriving the class "sub", makes the class "base" as abstract.

4.7.5 Static Polymorphism in Java

- 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.

4.7.5(A) 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 4.7.6 : Write a program to demonstrate constructor overloading.

Solution :

```
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.

4.7.5(B) 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.

Program 4.7.7 : Write a program to demonstrate Method overloading by overloading the methods for calculating area of circle, rectangle and triangle.

Solution :

```
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 4.7.8 : 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 4.7.9 : 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.

Solution :

```
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

4.7.5(C) Operator Overloading

SPPU - May 17, Dec. 18

Q. What are rules of operator overloading ? **(May 17, 3 Marks)**

Q. Explain need of operator overloading. Write C++ program to demonstrate any unary operator overloading. **(Dec. 18, 6 Marks)**

- The mechanism of assigning a new meaning to an already existing operator is called as operator overloading.
- There are some rules necessary to be known for operator overloading. Below is a list of these rules.

1. Only existing operators can be given a new meaning i.e. only existing operators can be overloaded.
2. Even after overloading the basic meaning of the operator remains the same.
3. Overloaded operators follow the same syntax as that of the original operators.
4. Unary operators overloaded by means of member function can accept no parameters.
5. Unary operators overloaded by means of friend function can accept up to one parameter.
6. Binary operators overloaded by means of member function can accept up to one parameter.
7. Binary operators overloaded by means of friend function can accept up to two parameters.
8. Some of the operators cannot be overloaded viz.
   (a) sizeof()
   (b) member operator (i.e. '.')
   (c) pointer to member operator (i.e. ".*")
   (d) scope resolution operator (i.e. "::")
   (e) conditional operator (i.e. "?:")
9. Some of the operators cannot be overloaded by friend functions viz.
   (a) assignment operator (i.e. '=')
   (b) function call operator (i.e. "( )")
   (c) subscripting operator (i.e. "[ ]")
   (d) class member access operator (i.e. "->")
10. Some of the rules are very simple, but some you may not understand now. We will understand these rules with program examples.
11. The declaration of the function that is meant for operator overloading has a special syntax as given below :

    return_type operator op (argument list);

    where "op" is to be replaced by the symbol of the operator to be overloaded.

Program 4.7.10 : Write a program to negate the values of two variables contained in an object.

```
#include<iostream.h>
#include<conio.h>
class Negate
{
   int x,y;
   public:
   void read()
   {
      cout<<"Enter two numbers";
      cin>>x>>y;
   }
   void compute()
   {
      x=-x;
```

```
    y=-y;
  }
  void display()
  {
    cout<<"x="<<x<<endl<<"y="<<y;
  }
};
void main()
{
  clrscr();
  Negate n;
  n.read();
  n.compute();
  n.display();
  getch();
}
```

Output

```
Enter two numbers4
6
x=-4
y=-6
```

Explanation

- A simple program with three functions respectively to accept the numbers, compute the negative and to display the result.
- An object of the class is made and the three functions are called one after the other.
- Let us see the same operation by overloading the unary minus ('-') operator in the following program.

Program 4.7.11: Write a program to negate the values of two variables contained in an object by overloading the operator unary negate i.e. '-'.

```
#include<iostream.h>
#include<conio.h>
class Negate
{
  int x,y;
  public:
  void read()
  {
    cout<<"Enter two numbers";
    cin>>x>>y;
  }
  void operator -()
  {
    x=-x;
    y=-y;
  }
  void display()
  {
    cout<<"x="<<x<<endl<<"y="<<y;
  }
};
void main()
{
  clrscr();
  Negate n;
  n.read();
  -n;
  n.display();
  getch();
}
```

Output

```
Enter two numbers4
6
x=-4
y=-6
```

Explanation

- In this case we have implemented the operator overloading operation by overloading the operator unary minus i.e. '-'.
- Compare to previous program where we had to call the compute() function, here we have to just use the negate operator with the object of the class i.e. "–n;".
- Actually operators can work only with variables of basic data type. The operators cannot be used with the objects of a class. But operator overloading allows those operators that are overloaded in a class to be used with the objects of that class.
- Remember, we cannot use an operator with an object unless we have written an operator overloading function for that operator in the corresponding class.

Program 4.7.12 : Write a program to overload unary operators i.e. increment and decrement.

```
#include<iostream.h>
#include<conio.h>
class IncDec
{
   int x,y;
   public:
   void read()
   {
      cout<<"Enter two numbers";
      cin>>x>>y;
   }
   void operator --()
   {
      x--;
      y--;
   }
   void operator ++ ()
   {
      x++;
      y++;
   }
   void display()
   {
      cout<<"x="<<x<<endl<<"y="<<y<<endl;
   }
};
void main()
{
   clrscr();
   IncDec n;
   n.read();
   --n;
   cout<<"After decrementing the object one time\n";
   n.display();
   ++n;
   ++n;
   cout<<"After incrementing the object twice\n";
   n.display();
   getch();
}
```

Output

```
Enter two numbers4
5
After decrementing the object one time
x=3
y=4
After incrementing the object twice
x=5
y=6
```

Explanation

– In this case we have implemented the operator overloading operation by overloading the operators unary increment and decrement ("++" and "--").

Program 4.7.13 : Write a program to overload binary operator "+" to add two complex numbers.

```
#include<iostream.h>
#include<conio.h>
class Complex
{
   int x,y;
   public:
   void read()
   {
      cout<<"Enter the real and imaginary parts of a complex number:";
      cin>>x>>y;
   }
   Complex operator + (Complex c)
   {
      Complex c1;
      c1.x=x+c.x;
      c1.y=y+c.y;
      return c1;

   }
   void display()
   {
      if(y<0)
      cout<<x<<y<<"i";
      else
      cout<<x<<"+i"<<y;
   }
};
void main()
{
   clrscr();
   Complex c1;
   Complex c2;
```

```
    Complex c3;
    c1.read();
    c2.read();
    c3=c1+c2;
    c3.display();
    getch();
}
```

Output

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

Explanation

- Here we have overloaded the binary operator "+" by the operator overloading function Complex operator + (Complex c).
- Two objects of the class complex are created. The values of these complex variables are read.
- The two objects of the class "complex" are added using the same syntax of the addition operator i.e. '+'. The result is returned to another object of the same class.

Program 4.7.14 : Write a program to overload binary operator "+=" to add two complex numbers.

```
#include<iostream.h>
#include<conio.h>
class Complex
{
   int x,y;
   public:
   void read()
   {
      cout<<"Enter the real and imaginary parts of a complex number:";
      cin>>x>>y;
   }
   void operator += (Complex c)
   {
      x=x+c.x;
      y=y+c.y;
   }
   void display()
   {
      if(y<0)
      cout<<x<<y<<"i";
      else
      cout<<x<<"+i"<<y;
   }
};
void main()
{
   clrscr();
   Complex c1;
   Complex c2;
   c1.read();
   c2.read();
   c1+=c2;
   c1.display();
   getch();
}
```

Output

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

Explanation

- Similar implementation like the previous program, but the original object of the class "complex" i.e. 'c1' is changed instead of having another object.

Program 4.7.15 : Write a program to add two distances entered by the user in feet and inches using overload binary operator "+".

```
#include<iostream.h>
#include<conio.h>
class Distance
{
   int ft,in;
   public:
   void read()
   {
      cout<<"Enter the distance in feet and inches:";
      cin>>ft>>in;
   }
   Distance operator + (Distance d)
   {
```

```
    Distance dr;
    dr.ft=ft+d.ft;
    dr.in=in+d.in;
    if(dr.in>=12)
    {
      dr.ft++;
      dr.in=dr.in-12;
    }
    return dr;
  }
  void display()
  {
    cout<<"Distance is "<<ft<<" feet and "<<in<<"
inches.";
  }
};
void main()
{
  clrscr();
  Distance d1;
  Distance d2;
  Distance d3;
  d1.read();
  d2.read();
  d3=d1+d2;
  d3.display();
  getch();
}
```

Output

```
Enter the distance in feet and inches:4
3
Enter the distance in feet and inches:5
11
Distance is 10 feet and 2 inches.
```

Explanation

- Similar operation as in program 6.4.
- After adding the inches part, if the total inches is greater than or equal to 12, then the feet is to be incremented by one and inches are to be deducted by 12. This is implemented in the "if" statement of the operator overloading function in the operator overloading function of the class "Distance".

4.7.6 Dynamic Polymorphism in Java

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

4.7.6(A) 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 4.7.16 : 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.

4.8 Efficiency and Polymorphism

Polymorphism compromises with certain issues including efficiency. For example, as seen in the dynamic method dispatch, if the field members would have been known in advance, the decision of the method to be executed would have been faster and the system would be efficient.

Hence, the efficiency is compromised with the use of polymorphism.

4.9 Case Study : A Bank Account System

- A bank may have current account as well as savings account.
- The account opening, withdrawing, depositing etc operations may be required for the bank.
- A sample of such kind of system is implemented in the following software.

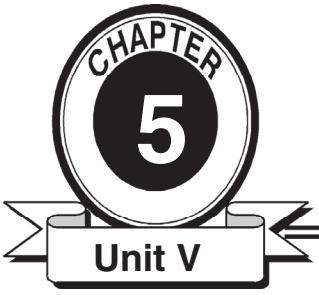
```
import java.util.*;
class Bank
{
    private int accountNumber;
    private float accountBalance;
    private String accountHolder;
    private String name,surname,middleName;
    public Bank(int count)
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("Account Holder details:-");
        System.out.print("First name : ");
        name=sc.next();
        System.out.print("Middle name : ");
        middleName=sc.next();
        System.out.print("Last name : ");
        surname=sc.next();
        accountHolder=name+""+middleName+"
"+surname;
        accountHolder=accountHolder.toUpperCase();
        accountNumber=count;
        System.out.println("Minimum balance to be kept =
1000");
        accountBalance=0;
        do
        {
            System.out.print("Deposit amount = ");
            accountBalance=sc.nextFloat();
        }while(accountBalance<1000);
        System.out.println("Account "+accountNumber+"
Successfully created for "+accountHolder);
        System.out.println("Account balance =
"+accountBalance);
```

```
    }
    public void updateName()
    {
        Scanner sc = new Scanner(System.in);
        System.out.println(accountNumber+
"\t"+accountHolder);
        System.out.println("Account Holder details:-");
        System.out.print("First name : ");
        name=sc.nextLine();
        System.out.print("Middle name : ");
        middleName=sc.nextLine();
        System.out.print("Last name : ");
        surname=sc.nextLine();
        accountHolder=name+" "+middleName+"
"+surname;
        accountHolder=accountHolder.toUpperCase();
        System.out.println("Account "+accountNumber+"
Name updated to "+accountHolder);
    }
    public void deposit()
    {
        float amount;
        System.out.print("Enter amount to deposit : ");
        Scanner sc = new Scanner(System.in);
        amount=sc.nextFloat();
        accountBalance = accountBalance + amount;
        System.out.println(amount+ " credited successfully to
account number "+accountNumber+", updated account
balance is "+accountBalance);
    }
    public boolean checkName(String str)
    {
        if(accountHolder.equals(str)) return true;
        else return false;
    }
    public void withdraw()
    {
        float amount;
        System.out.println("Balance =
"+accountBalance+"\tMaintain minimum balance of
1000");
        Scanner sc = new Scanner(System.in);
        do
        {
            System.out.print("Enter amount to withdraw : ");
            amount=sc.nextFloat();
            if(amount>accountBalance-1000)
System.out.println("Minimum balance cannot be maintained
if you withdraw this amount");
        }while(amount>accountBalance-1000);
        accountBalance = accountBalance - amount;
        System.out.println(amount+ " debited successfully
from account number "+accountNumber+", updated
account balance is "+accountBalance);
    }
    public void checkBalance()
    {
        System.out.println("A/c
"+accountNumber+"\n"+accountHolder+"\nAccount
Balance : "+accountBalance+"\n");
    }
}
public class Banking
{
    public static void main(String args[])
    {
        int i,choice,count,copy;
        String str,str1;
        count=0;
        Scanner sc = new Scanner(System.in);
        Bank b[] = new Bank[100];
        do
        {
            System.out.println("\n1 : Add account\n2 : Change
account holder name\n3 : Deposit\n4 : Withdraw\n5 : Check
account balance\n6 : exit");
            choice = sc.nextInt();
            switch(choice)
            {
                case 1: count=count+1;
                    b[count-1] = new Bank(count);
                    break;
                case 2: System.out.println("Enter account number
or account holder's first name");
                    str=sc.next();
                    if(str.charAt(0)>='1'&&str.charAt(0)<='9')
                    {
                        copy=Integer.parseInt(str);
                        if(copy>count)
                        {
                            System.out.println("Account not found");
                            break;
```

```
                }
                b[copy-1].updateName();
            }
            else
            {
                System.out.print("Middle name : ");
                str1=sc.next();
                str=str+" "+str1;

                System.out.print("Last name : ");
                str1=sc.next();
                str=str+" "+str1;
                str=str.toUpperCase();
                copy=count+1;
                for(i=0;i<=count-1;i++)
                {
                    if(b[i].checkName(str))
                    {
                        copy=i;
                        break;
                    }
                }
                if(copy>count)
                {
                    System.out.println("Account not found");
                    break;
                }
                b[copy].updateName();
            }
            break;
        case 3: System.out.println("Enter account number
or account holder's first name");
            str=sc.next();
            if(str.charAt(0)>='1'&&str.charAt(0)<='9')
            {
                copy=Integer.parseInt(str);
                if(copy>count)
                {
                    System.out.println("Account not found");
                    break;
                }
                b[copy-1].deposit();
            }
            else
            {
                System.out.print("Middle name : ");
                str1=sc.next();
                str=str+" "+str1;

                System.out.print("Last name : ");
                str1=sc.next();
                str=str+" "+str1;
                str=str.toUpperCase();
                copy=count+1;
                for(i=0;i<=count-1;i++)
                {
                    if(b[i].checkName(str))
                    {
                        copy=i;
                        break;
                    }
                }
                if(copy>count)
                {
                    System.out.println("Account not found");
                    break;
                }
                b[copy].deposit();
            }
            break;
        case 4: System.out.println("Enter account number
or account holder's first name");
            str=sc.next();
            if(str.charAt(0)>='1'&&str.charAt(0)<='9')
            {
                copy=Integer.parseInt(str);
                if(copy>count)
                {
                    System.out.println("Account not found");
                    break;
                }
                b[copy-1].withdraw();
            }
            else
            {
                System.out.print("Middle name : ");
                str1=sc.next();
                str=str+" "+str1;

                System.out.print("Last name : ");
```

```
                str1=sc.next();
                str=str+" "+str1;
                str=str.toUpperCase();
                copy=count+1;
                for(i=0;i<=count-1;i++)
                {
                    if(b[i].checkName(str))
                    {
                        copy=i;
                        break;
                    }
                }
                if(copy>count)
                {
                    System.out.println("Account not found");
                    break;
                }
                b[copy].withdraw();
            }
            break;
        case 5: System.out.println("Enter account number
or account holder's first name");
            str=sc.next();
            if(str.charAt(0)>='1'&&str.charAt(0)<='9')
            {
                copy=Integer.parseInt(str);
                if(copy>count)
                {
                    System.out.println("Account not found");
                    break;
                }
                copy=copy-1;
                b[copy].checkBalance();
            }
            else
            {
                System.out.print("Middle name : ");
                str1=sc.next();
                str=str+" "+str1;

                System.out.print("Last name : ");
                str1=sc.next();
                str=str+" "+str1;
                str=str.toUpperCase();
                copy=count+1;
                for(i=0;i<=count-1;i++)
                {
                    if(b[i].checkName(str))
                    {
                        copy=i;
                        break;
                    }
                }
                if(copy>count)
                {
                    System.out.println("Account not found");
                    break;
                }
                b[copy].checkBalance();
            }
            break;
        case 6: break;
        }
    }while(choice!=6);
  }
}
```


❑❑❑



Exception Handling and Generic Programming

Syllabus

Exception : Errors, Types of Errors, Exception and its Types, Exception-Handling Fundamentals, Uncaught Exception, Using try and Catch, Multiple Catch Clauses, Nested Try Statements, User Define Exception using Throw.

Generics : What are Generics? Introduction to Language Specific Collection Interface: List Interface and Set Interface, Collection Classes: ArrayList Class and LinkedList Class.

5.1 Errors and Exception-Handling Fundamentals

- 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 Types of Errors or Exceptions

5.2.1 Checked Exceptions

- All checked exceptions are subclasses 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 or Uncaught 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 divideby 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 Number Format Exception, 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 Using try and catch

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 earlier, 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 5.3.1, 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.

5.4 Multiple Catch Clauses

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 5.4.1 : 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 5.4.2 : 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.

5.5 Nested try Statements

- 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 5.5.1 : 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 5.5.2 : 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.

5.5.1 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 5.5.3 : 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.

5.6 User Define Exception using 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 5.6.1 : 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 5.6.2 : 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
```

5.7 Exception Handling in C++

SPPU - May 17, May 19, Dec. 19

Q. Explain how the exception is handled in C++. **(May 17, 3 Marks)**

Q. How exception handling mechanism works in C++? **(May 19, 7 Marks)**

Q. Explain need of exception handling. Explain it with example. **(Dec. 19, 6 Marks)**

- Exception handling refers to handling of abnormal or unexpected events.
- Some of these abnormal conditions are known to the 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.
- Exception as discussed is an error condition occurrence. For example if a integer is expected and a character is entered, then while

5.8 Exception Handling Mechanism

5.8.1 Try-catch-throw, Multiple Catch and Catch All

SPPU - Dec. 16, Dec. 17, May 19, Dec. 19

Q. Explain try-catch-throw with a program example.

Q. Explain exception handling mechanism in C++. **(Dec. 16, 4 Marks)**

Q. How to catch multiple exceptions. **(Dec. 16, 3 Marks, Dec. 17, May 19. Dec. 19)**

Q. What is exception handling mechanism in C++ ? Write a program in C++ to handle "divide by zero" exception. **(Dec. 17, 4 Marks)**

Q. What is the advantage of using exception handling mechanism in a program? **(May 19, 4 Marks)**

- The C++ exception handling is done by the keywords "try", "catch" and "throw".
- The general syntax of the try catch block is as shown below :

```
try {
    //statements that can cause exception;
  }
  catch(data_type arg)
  {
    //statements to be executed in case of the exception is generated.
  }
  catch(data_type arg)
  {
    //statements to be executed in case of the exception is generated.
  }
  catch(data_type arg)
  {
    //statements to be executed in case of the exception is generated.
  }
```

- As shown in the syntax, there can be multiple catch blocks associated with one try block.
- Thus using the exception handling i.e. the try-catch block, the programmer can generate the error or exception statements as per the requirement of the programmer.
- Another keyword used in the exception handling of the C++ is the "throw" keyword. The exception can be generated only by the use of the throw keyword in the try block.
- The control enters into the try block. If an exception occurs, it compares the data type of the parameter with the data type of the argument in the catch block. the control then enters into that catch block, wherever the data type matches. Once the statements in the catch block are executed, the control proceeds further and doesn't enter into the try block again.
- Thus if the exception occurs in a particular statement in the try block, the remaining statements followed by this statement in the try block are not executed.

- If the exception does not occur, the try block is completely executed and thereafter the control is transferred to the statements followed by the catch block.
- Let us see some program examples for the exception handling in C++.

Program 5.8.1 : Write a program to demonstrate the use of try catch block with the arguments as an integer and a string using multiple catch blocks. **SPPU - Dec. 19**

```
#include<iostream.h>
#include<conio.h>
void test(int x)
{
   try
   {
      if(x>=0 && x<=9) throw x;
      else throw "not a single digit number";
   }
   catch(int i)
   {
      cout<<"The number is single digit";
   }
   catch(char a[100])
   {
      cout<<a;
   }
}
void main()
{
   int p;
   cout<<"Enter a number:";
   cin>>p;
   test(p);
   getch();
}
```

Output

```
Enter a number:7
The number is single digit
```

Note : This program will not work in Turbo C 3.0 version. You will require a higher version for the same.

5.8.2 Implementing User Defined Exception

Q. Implement a user defined exception and explain the same using program example.

Program 5.8.2 : Write a program to accept password and throw an exception if the password has less than 6 characters or does not contain a digit.

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
void main()
{
   char x[20];
   int i;
   cout<<"Enter a password:";
   gets(x);
   try
   {
      for(i=0;x[i]!='\0';i++)
      {
         if(x[i]>='0' && x[i]<='9') throw "Incorrect password format";
      }
      if(i<=5) throw "Incorrect password format";
   }
   catch(char a[100])
   {
      cout<<a;
   }
   getch();
}
```

Output

```
Enter a password:not
Incorrect password format
```

5.8.3 Concept of Throw and Catch with Example

- If an exception is thrown in the catch block, it is called as rethrowing of an exception.
- In this case only if the statement in the try block, throws an exception, the control will enter into the catch block which will rethrow an exception that will be caught by another catch block. A program to demonstrate the same is shown below.

Program 5.8.3 : Write a program to accept password and throw an exception if the password has less than 6 characters or does not contain a digit. Give another chance to enter a correct password and rethrow an exception if the password is again incorrect.

```
#include <iostream>
# include <conio.h>
#include<stdio.h>
using namespace std;
void check(char x[20])
{
  int i;
  try
   {
      for(i=0;x[i]!='\0';i++)
      {
         if(x[i]>='0'  &&  x[i]<='9')  throw  "Incorrect password format";
      }
      if(i<=5) throw "Incorrect password format";
   }
   catch(char a[100])
   {
      cout<<"Incorrect password format";
      cout<<"\nEnter password again:";
      cin>>x;
      for(i=0;x[i]!='\0';i++)
      {
         if(x[i]>='0'  &&  x[i]<='9')  throw  "Incorrect password format";
      }
      if(i<=5) throw "Incorrect password format";
   }
}
void main()
{
   char x[20];
   int i;
   cout<<"Enter a password:";
   cin>>x;
   try
   {
      check(x);
   }
   catch(char a[100])
   {
      cout<<"Incorrect again...sorry";
   }
   getch();
}
```

Output

```
Enter a password:hello
Incorrect password format
Enter password again:hi
Incorrect again...sorry
```

5.9 What are Generics?

- The generic function defines a set of statements that are common for the different types of data.
- Hence instead of writing multiple functions for various data types, we can use generic functions if the set of statements that are to be executed are same for all the data types.
- For example when data is to be added, the function will have same statements for any type of data. Similarly for sorting the statements to sort data will be same irrespective of the type of data. Hence this can be implemented by the use of generic functions.

5.9.1 Function Templates and Examples

- To create a generic function can be created using the keyword "template". The syntax for creating a template or a generic function is as given below :

  template <class type> ret_data_type function_name (list of arguments).

```
{
      --
      --
      statements of the function
      --
      --
}
```

- The class type can be any random class name that will be used as a generic data type in the function. This will be more clear in the Program 5.9.1.
- The main advantage of the generic function as discussed earlier is that instead of writing multiple functions and increasing the code size, one can use templates and the task of writing the program will also be simple for the programmer.
- Let us see some simple program examples using the generic function or the template.

Program 5.9.1 : Write a program to write a generic function or template and demonstrate addition of multiple types of data using the same.

```
#include<conio.h>
#include<iostream.h>
template <class x> x add(x a, x b)
{
   return (a+b);
}
void main()
{
   cout<<add(3,5)<<endl;
   cout<<add(3.5,6.2)<<endl;
   cout<<add(2.0,4.5)<<endl;
   getch();
}
```

Output

```
8
9.7
6.5
```

Explanation

- Thus in the first case the data is passed as integers. In the second case it is passed as float numbers or double numbers. In the third case, we cannot give the data as 2 and 4.5, else it will be integer and double, which are different; while the generic function claims both the parameters to be of same type.

Program 5.9.2 : Write a program to write a generic function or template and demonstrate swapping of multiple types of data using the same

```
#include<conio.h>
#include<iostream.h>
template <class x> void swap(x &a, x &b)
{
   x temp;
   temp=a;
   a=b;
   b=temp;
}
void main()
{
   int a=4,b=5;
   float p=3.5,q=4.2;
   cout<<"a="<<a<<"\tb="<<b<<"  before  calling
swap function"<<endl;
   swap(a,b);
   cout<<"a="<<a<<"\tb="<<b<<"   after   calling
swap function"<<endl;
   cout<<"p="<<p<<"\tq="<<q<<"  before  calling
swap function"<<endl;
   swap(p,q);
   cout<<"p="<<p<<"\tq="<<q<<"   after   calling
swap function"<<endl;
   getch();
}
```

Output

```
a=4   b=5 before calling swap function
a=5   b=4 after calling swap function
p=3.5  q=4.2 before calling swap function
p=4.2  q=3.5 after calling swap function
```

Program 5.9.3 : Write a C++ program using a class template to read any five parameterized data type such as float and integer and print the average.

```
#include<conio.h>
#include<iostream.h>
template <class x> float avg(x a, x b, x c, x d, x e)
{
   float average;
   average=(a+b+c+d+e)/5.0;
   return average;
```

```
}
void main()
{
   int p,q,r,s,t;
   float v,w,x,y,z;
   cout<<"Enter five integers:";
   cin>>p>>q>>r>>s>>t;
   cout<<"Average="<<avg(p,q,r,s,t)<<endl;
   cout<<"Enter five float numbers:";
   cin>>v>>w>>x>>y>>z;
   cout<<"Average="<<avg(v,w,x,y,z)<<endl;
   getch();
}
```

Output

```
Enter five integers : 1
2
3
4
5
Average=3
Enter five float numbers : 1.2
2.3
3.4
4.5
5.6
Average=3.4
```

Program 5.9.4 : Write a program to create a vector class template to add, delete and display values from the vector.

```
#include<conio.h>
#include<iostream.h>
template <class vector> int insert(vector data, vector array[], int size)
{
   array[size]=data;
   return(size+1);
}
template <class vector> int remove(vector data, vector array[], int size)
{
   int i;
   for(i=0;i<=size-1;i++)
   {
      if(data==array[i]) break;
   }
   while(i<=size-2)
   {
      array[i]=array[i+1];
      i++;
   }
   return(size-1);
}
template <class vector> void display(vector array[], int size)
{
   int i;
   for(i=0;i<=size-1;i++)
   {
      cout<<array[i]<<"\t";
   }
   cout<<endl;
}
void main()
{
   int m,n,i,a[100];
   float b[100],x;
   cout<<"Enter size of integer array:";
   cin>>n;
   for(i=0;i<=n-1;i++)
   {
      cout<<"Enter an integer:";
      cin>>a[i];
   }
   cout<<"1. Add an element\n2. Delete an element\n3. Display\nEnter your choice:";
   cin>>i;
   switch(i)
   {
      case 1:  cout<<"Enter element to insert:";
         cin>>i;
         n=insert(i,a,n);
         cout<<"After insertion:\n";
         display(a,n);
         break;
      case 2: cout<<"Enter element to be deleted:";
         cin>>i;
         n=remove(i,a,n);
```

```
        cout<<"After deletion:\n";
        display(a,n);
        break;
    case 3: display(a,n);
        break;
    default:cout<<"Invalid choice\n";
  }
  cout<<"Enter size of float array:";
  cin>>m;
  for(i=0;i<=m-1;i++)
  {
    cout<<"Enter a float number:";
    cin>>b[i];
  }
  cout<<"1. Add an element\n2. Delete an element\n3. Display\nEnter your choice:";
  cin>>i;
  switch(i)
  {
    case 1:  cout<<"Enter element to insert:";
        cin>>x;
        m=insert(x,b,m);
        cout<<"After insertion:\n";
        display(b,m);
        break;
    case 2: cout<<"Enter element to be deleted:";
        cin>>x;
        m=remove(x,b,m);
        cout<<"After deletion:\n";
        display(b,m);
        break;
    case 3: display(b,m);
        break;
    default:cout<<"Invalid choice\n";
  }
  getch();
}
```

Output

```
Enter size of integer array : 3
Enter an integer : 1
Enter an integer : 2
Enter an integer : 3
1. Add an element
2. Delete an element
3. Display
Enter your choice : 1
Enter element to insert  : 4
After insertion:
1    2    3    4
Enter size of float array : 3
Enter a float number : 1.5
Enter a float number : 2.25
Enter a float number : 3.75
1. Add an element
2. Delete an element
3. Display
Enter your choice : 2
Enter element to be deleted : 2.25
After deletion:
1.5   3.75
```

Program 5.9.5 : Write a C++ program using a class template to read any parameterized data type such as float and integer and print them in sorted form.

```
#include<conio.h>
#include<iostream.h>
template <class x> void sort(x a[], int n)
{
  int i,j;
  x temp;
  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;
      }
    }
  }

}
void main()
{
  int a[100],m,n,i;
```

```
    float b[100];
    cout<<"Enter number of integers:";
    cin>>n;
    for(i=0;i<=n-1;i++)
    {
       cout<<"Enter an integer:";
       cin>>a[i];
    }
    sort(a,n);
    cout<<"Sorted array:\n";
    for(i=0;i<=n-1;i++)
    {
       cout<<a[i]<<"\t";
    }
    cout<<endl;
    cout<<"Enter number of float numbers:";
    cin>>m;
    for(i=0;i<=m-1;i++)
    {
       cout<<"Enter a float number:";
       cin>>b[i];
    }
    sort(b,m);
    cout<<"Sorted array:\n";
    for(i=0;i<=m-1;i++)
    {
       cout<<b[i]<<"\t";
    }
    cout<<endl;
    getch();
}
```

Output

```
Enter number of integers : 3
Enter an integer : 4
Enter an integer : 2
Enter an integer : 6
Sorted array :
2    4    6
Enter number of float numbers : 3
Enter a float number : 2.25
Enter a float number : 3.25
Enter a float number : 1.5
Sorted array :
1.5    2.25    3.25
```

5.9.2 Overloading a Template Function Using a Non-template Function

- A template function can be overloaded for a particular type of data by writing a separate function for that data type.
- An example program is shown below for addition of a special case char type data.

Program 5.9.6 : Write a program to write a generic function or template and demonstrate addition of multiple types of data using the same. Write overloaded function for showing addition of char type data.

```
#include<conio.h>
#include<iostream.h>
template <class x> x add(x a, x b)
{
   return (a+b);
}
char add(char x, char y)
{
   return(x+y-64);
}
void main()
{
   cout<<add(3,5)<<endl;
   cout<<add(3.5,6.2)<<endl;
   cout<<add('A','B')<<endl;
   getch();
}
```

Output

```
8
9.7
C
```

5.9.3 Overloading a Template Function Using a Template Function

Q. Explain overloading a template function using a template function with a program example.

- A template function can be overloaded also using template for a special case of data.
- An example program is shown below for addition of a special case char type data with integer type data.

Program 5.9.7 : Write a program to write a generic function or template and demonstrate addition of multiple types of data using the same. Write overloaded template for showing addition of char type data and integer type data.

```
#include<conio.h>
#include<iostream.h>
template <class x> x add(x a, x b)
{
   return (a+b);
}
template <class x>x add(char a,x  y)
{
   return(a+y-64);
}
void main()
{
   cout<<add(3,5)<<endl;
   cout<<add(3.5,6.2)<<endl;
   cout<<add('A',3)<<endl;
   getch();
}
```

Output

```
8
9.7
4
```

5.9.4 Generic Classes, Class Template and Examples

- Similar to generic functions we can also have generic classes.
- In this case the actual type of data can be manipulated as per the requirement.
- The syntax of a generic class is as given
- below :

```
template <class type> class class_name
{
   --
   --
   data and function members of the class;
   --
   --
};
```

- An example of such a generic class can be as given below :

```
template <class x> class Test
{
   x a,b,c;
   public:
   void get_data();
   void put_data();
   void display();
};
```

5.9.5 Overview and Use of Standard Template Library (STL)

Q. List the use of Standard Template Library.

- It is a collection of many libraries to help in C++ programming. It consists of four major libraries :
  1. Algorithms
  2. Containers
  3. Functional or Functors
  4. Iterators
- The algorithms section consists of a large number of algorithms to perform basic operations like search, sort, etc. There are various algorithms available in each of these i.e. various search algorithms implemented, various sort algorithms implemented and so on.
- The container section as the name says has various types of data containers like lists, vectors, queues, priority queues, stacks, map, hash map, etc.
- The functors consists of various call operators, predicate etc. The iterator section consists of various iterators like,
  1. Input iterator : Used to read a sequence of input values.
  2. Output iterator : Used to display a sequence of values.
  3. Forward iterators : Used to read or write but only in forward direction.
  4. Bidirectional iterators : Used to read and write in either of the directions.

5. Random access iterate : Used to access a data from the array randomly, to be read or written.

5.10 Introduction to Language Specific Collection Interface : List Interface and Set Interface

- In Java, the collection interface defines many types of collection. It enables to work with group of data or objects.
- The interfaces "List", "Set" and "SortedSet" are extended from the interface "Collection". Besides these three, there are many other interfaces also extended from the interface "Collection".
- We will see the two interfaces in detail namely: the "List" and "Set" interfaces

5.10.1 List Interface

- The "List" interface extends the interface"Collection".
- The list interface has methods for inserting, accessing and deleting a particular element by the index starting from "0".
- If a collection cannot be modied, the exception named "UnsupportedOperationException" will be thrown. The "ClassCastException" will be thrown if one object is not compatible with the other.
- To add elements to the list, we have methods like add(), addAll() defined by the class that implements the interface List.
- The method get() is used to obtain an object at specific location.
- Tje method set() is used to assign a value to the element in the list.
- To search an element, we have objects like indexOf() and lastIndexOf(), required in case of multiple existence of same value objects.
- The remove() method is used to delete an element in the list with the specified index.

5.10.2 Set Interface

- The "Set" interface also extends the interface"Collection".
- It has almost the same methods as that of the "List" interface.
- The main difference is that the add() method of class implementing "Set" interface has to add the object only if its new and not a duplicate element, as it is making a set.

5.11 Collection Classes : ArrayList Class and LinkedList Class

- Similar to the collection interfaces, Java also has collection classes.
- Some of the collection classes are "AbstractCollection", "AbstractList", "LinkedList", "ArrayList", etc.
- We will see the "ArrayList" and "LinkedList" collection classes in the subsequent sub-sections

5.11.1 ArrayList Class

- The "ArrayList" class extends the classs "AbstractList" and also implements the interface "List", seen in the previos section.
- The general arrays in case of Java are of fixed size and cannot grow or shrink in size. This problem is handled by ArrayList class objects.
- Thus the ArrayList can increase or decrease in size dynamically i.e. during the program execution.
- The constructors of ArrayList class are :
  - Default constructor i.e. no parameters
  - Constructor with an integer parameter defining initial size
  - Constructor with the parameter as object of collection
- Let us see a simple example based on ArrayList class

```
import java.util.*;
class DemoArrayList
{
   public static void main(String args[])
   {
      ArrayList a=new ArrayList();
      System.out.println("Size of ArrayList object on initialization="+a.size());
```

```
        a.add("T");
        a.add("e");
        a.add("c");
        a.add("h");
        a.add("K");
        a.add("n");
        a.add("o");
        a.add("w");
        a.add("l");
        a.add("e");
        a.add("d");
        a.add("g");
        a.add("e");
        System.out.println("Size of ArrayList object after adding elements="+a.size());
        System.out.println("Contents of ArrayList object are: "+a);
    }
}
```

Output

```
Size of ArrayList object on initialization=0
Size of ArrayList object after adding elements=13
Contents of ArrayList object are: [T, e, c, h, K, n, o, w, l, e, d, g, e]
```

5.11.2 LinkedList Class

- The "LinkedList" class extends the classs "AbstractSequentialList" and also implements the "List" interface.
- The constructors are :
  - Default constructor
  - Constructor with the parameter as object of collection
- It also has methods the add() and remove() elemet from the list. Besides it has additional methods like addFirst(), addLast(), getFirst(), getLast(), removeFirst(), remoeLast() etc.
- This helps in creating queues of various types, stacks etc. using thelinked list.
- Let us see a simple example based on LinkedList class

```
import java.util.*;
class DemoLinkedList
{
   public static void main(String args[])
   {
      LinkedList a=new LinkedList();
      a.add("T");
      a.add("e");
      a.add("c");
      a.add("h");
      a.add("K");
      a.add("n");
      a.add("o");
      a.add("w");
      a.add("l");
      a.add("e");
      a.add("d");
      a.add("g");
      a.add("e");
      System.out.println("Contents of LinkedList object are: "+a);
   }
}
```

Output

```
Contents of LinkedList object are: [T, e, c, h, K, n, o, w, l, e, d, g, e]
```

5.12 Exception Handling and Generic Programming using Array List (ArrayList Class)

- There are various exceptions generated by the ArrayList class.
- Some of the exceptions generated are listed below:
  - "UnsupportedOperationException" will be generated if the collection cannot be modified.
  - Exception"ClassCastException"is generated when an object is not compatible with the other

- Exception "NullPointerException" will be generated if the reference made is to a null object.

– We can also perform generic programming using ArrayList class.

– The following example shows the use of ArrayList class for generic programming.

```
import java.util.*;
class DemoGenericsArrayList
{
   public static void main(String args[])
   {
      ArrayList<String> a=new ArrayList<String>();
      a.add("Jay");
      a.add("Ajay");
      a.add("Vijay");
      a.add("Sanjay");
      a.add("Sujay");
      String str=a.get(3);
      System.out.println("Element at 3 is: "+str);

      System.out.println("Entire list is: ");
      Iterator<String> i=a.iterator();
      while(i.hasNext())
      {
         System.out.println(i.next());
      }
   }
}
```

Output

```
Element at 3 is: Sanjay
Entire list is:
Jay
Ajay
Vijay
Sanjay
Sujay
```

❑❑❑

File Handling and Design Patterns

Syllabus

File Handling : Introduction, Concepts of Stream, Stream Classes, Byte Stream Classes, Character Stream, Classes, Using Stream, Other Useful I/O Classes, Using the File Class, Input/output Exceptions, Creation of Files, Reading/Writing Character, Reading/Writing Bytes, Handling Primitive Data Types, Concatenating and Buffering Files, Random Access Files. **Design Patterns :** Introduction, Types of Design Patterns, Adapter, Singleton, Iterator.

6.1 File Handling : Introduction

- File handling implies reading from a file or writing to a file. C++ has streams to read from or write onto a file. The streams are similar to the iostreams used to read from the keyboard and write to the monitor.
- The file handling is mainly required to access large data and also to store the data of one execution of a program to be used when the same is executed another time.
- In the following sections we will various streams available to access file in C++.

6.2 Concepts of Stream

Q. What is a stream ? **(2 Marks)**

Q. Explain various constructors of of stream. **(4 Marks)**

- The earlier C++ programs used the "stream.h" header files for accessing the files, but the newer versions have the "iostream.h" which has better support.
- The file handling is done using the stream objects predefined in the header file "iostream.h".
- Each of the stream has an associated class that contains various functions for dealing a particular type of data.

Constructors of stream

There are two constructors in the class of stream.

They are :

1. **Default constructor :** This creates an object of the stream class which is not associated with any file.
2. **Initialization constructor :** The first argument passed is the file name with which the object is to be associated. The second parameter indicates the file open mode.

6.3 Stream Classes : Byte, Character

SPPU - Dec. 16

Q. What are C++ stream classes for console operations ? Explain it in detail with the help of diagram. **(Dec. 16, 6 Marks)**

- The data can be read as bytes or characters. The stream of data in case of C++ is byte stream or character stream. Since for C++, the size of a character is a single byte, there is no difference in the two when it comes to C++. Although in case of Java there are separate byte and character streams.

6.3.1 File Stream Classes

SPPU - May 19

Q.	Explain the any two file stream classes needed for the file manipulation.	**(4 Marks)**
Q.	What are stream extraction and stream insertion operators ?	**(4 Marks)**
Q.	Which classes are used in file stream operations explain it in details.	**(May 19, 6 Marks)**

- The three classes for handling files are :

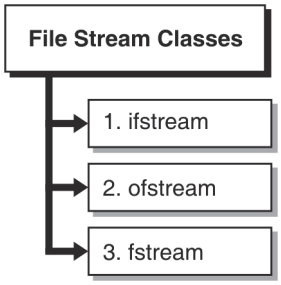


Fig. 6.3.1 : Classes for handling files

1. **ifstream :** This class is used to handle the input files.
2. **ofstream :** This class is used to handle output files.
3. **fstream :** This class is used for files in which both input as well as output is to be done.

- A stream is a sequence of bytes or characters. The iostream as the name says has input stream and output stream to read a stream or sequence of bytes and similarly write a stream i.e. sequence of bytes.
- The stream class has an hierarchy as shown in the Fig. 6.3.2.

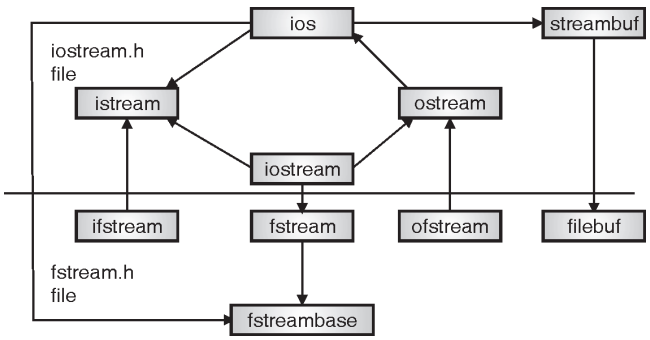


Fig. 6.3.2 : Stream class hierarchy

- We have been using some of these in this book with the help of the extraction (>>) and insertion (<<) operators i.e. with the "cin" and "cout" statements. These are derived from the "ios" class.
- The ios class is virtual class for the "istream" and "ostream" i.e. input stream and output stream respectively. The "istream" is a class derived from "ios" for input while "ostream" is for output.
- The class "iostream" has a multiple inheritance i.e. it is derived from classes "istream" and "ostream", hence used for both input as well as output.
- Similarly the classes "ifstream" is for input from files and "ofstream" is for output to files. The "fstream" class is for both input and output.

6.3.2 Advantages of Stream Classes

- Stream classes have various functions that make it easy for the programmer for reading or writing a sequence of bytes.
- These classes have good error handling capabilities
- The classes work as an abstraction for the user i.e. the internal operation is encapsulated from the user.
- The stream classes are buffered and hence do not use the memory disk space.

6.4 Other Useful I/O Classes

- As seen in the Fig. 6.3.1, there are many classes that are useful in I/O Stream handling.
- Although the main classes used are “ifstream” for reading from a file and “ofstream” for writing on to a file, there are other useful IO classes
- The base class is ios as seen in the Fig. 6.3.1.
- The classes derieved from this class are ifstream, ofstream, streambuf and other classes like ostream, istream, filebuf, fstream and fstreambase. Each of these I/O classes have their own significance.
- Some of those are discussed here:
  1. **filebuf :** The main purpose of this class is to set the file buffers as input or output as reqiored.

2. **fstreambase :** It is mainly used to provide the operations that are common to all the classes, like opening and closing a file.
3. **ifstream :** This class is used to read as discussed earlier and will also be seen in quite a lot detail in later sections
4. **ofstream :** This class is used to write on to the file and will also be seen in detail in the subsequent sections
5. **fstream :** It is mainly used to perform simultaneous read and write operations onto a file. Hence it has functions to do both read as well as write operations.

6.5 Input/Output Exceptions

There are various cases wherein there can be problems while accessing files, such as :

1. Attempting to access a non-existing file.
2. Some disk error.
3. Opening a read-only file in write mode.
4. Reading beyond the end of the file.
5. Opening a file with an invalid name.

There are functions in "ios" class to handle some of these errors are listed in the Table 6.5.1 :

Table 6.5.1: functions in "ios" class

Sr. No.	Function	Operation
1.	eof()	This function returns "true" if it reaches the end of file while accessing the file; else it returns "false".
2.	bad()	This functions returns "true" if attempted unrecoverable errors, else it returns "false".
3.	good()	This function returs "true" if the last operation was sucessful else returns "false".
4.	clear()	This function clears all the previous error states.
5.	fail()	This function returns "true" if the last read or write operation has failed. else returns "false"".
6.	rdstate()	This function returns the status data of the class ios.

6.6 Creation of Files and Using the File Class

SPPU - Dec. 16

Q. List file mode operation. **(3 Marks)**

Q. Explain flags ios::binary and ios::ate. **(4 Marks)**

Q. Explain tellg and seekp functions. **(4 Marks)**

Q. Explain various file mode parameters in C++. Write a program to copy the contents of a source file **student1.txt** to a destination file **student 2.txt** character by character. **(7 Marks)**

Q. Explain *ios::app* and *ios::ate* flags. **(3 Marks)**

Q. Explain the functions used to read and write data in binary file. **(4 Marks)**

Q. Explain the following file mode ios::trunk, ios::app, ios::ate, ios::binary. **(Dec. 16, 4 Marks)**

Tellg and Seekp functions

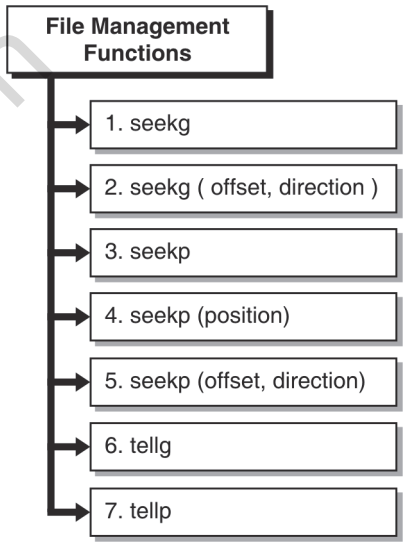


Fig. 6.6.1 : Tellg and seekp functions

1. seekg

– Sets the position of the *get pointer*. The *get pointer* determines the next location to be read in the source associated to the stream.

Syntax

```
seekg (position)
```

– Using this function the stream pointer is changed to the absolute position (counting from the beginning of the file).

2. seekg (offset, direction)

– Using this function, the position of the get pointer is set to an offset value relative to some specific point determined by the parameter direction. offset is of the member type off_type, which is also an integer type.

– And direction is of type seekdir, which is an enumerated type (enum) that determines the point from where offset is counted from.

3. seekp

– The *seekp* method changes the location of a stream object's file pointer for output (put or write.) In most cases, seekp also changes the location of a stream object's file pointer for input (get or read.)

4. seekp (position)

– Using this function the stream pointer is changed to the absolute position (counting from the beginning of the file).

5. seekp (offset, direction)

– Using this function, the position of the put pointer is set to an offset value relative to some specific point determined by the parameter direction.

– Offset is of the member type off_type, which is also an integer type. And direction is of type seekdir, which is an enumerated type (enum) that determines the point from where offset is counted from.

6. tellg

– The tellg() function is used with input streams, and returns the current "get" position of the pointer in the stream.

Syntax :

```
pos_type tellg()
```

– It has no parameters and return a value of the member type pos_type, which is an integer data type representing the current position of the get stream pointer.

7. tellp

– Returns the absolute position of the *put pointer*. The put pointer determines the location in the output sequence where the next output operation is going to take place.

Syntax

```
pos_type tellp()
```

– The tellp() function is used with output streams, and returns the current "put" position of the pointer in the stream. It has no parameters and return a value of the member type pos_type, which is an integer data type representing the current position of the put stream pointer.

– To work with files as discussed in the earlier section we need to include the header file "fstream.h".

– We need to first create an object of the stream class before opening a file. This stream object will be used to open the file as well as to perform various operations like read and write on the file.

– The object of that class may be created based on the operation required to be done on the file. Thus if input operation is to be done then an object of the class "ifstream" is to be created.

– Thus, the object can be created as below :

```
ifstream i;
ofstream o;
fstream f;
```

– Once the object of stream class is created, the file can be opened using the open() function. The prototype of open() function is as given below :

```
void open(const char *filename, int mode, int access
= filebuf::openprot);
```

– The filename is the name of the file to be opened, which can be given along the path. The mode can be one of the following based on the operation to be performed :

Sr. No.	Mode	Function
1.	ios::app	Append to a file
2.	ios::ate	Move the pointer/cursor to the end of file
3.	ios::binary	File open in binary mode

Sr. No.	Mode	Function
4.	ios::in	Open file in read mode
5.	ios::nocreate	This will cause to fail the open function if file doesn't exists
6.	ios::norepace	This will cause function to fail open function if file exists
7.	ios::out	Open file in write mode
8.	ios::trunc	This will cause earlier files (if any) with same name to be destroyed.

- Thus, for example if a file "test" is to be opened in out mode, it will be done by the following two statements :

```
ofstream o;
o.open("test",ios::out);
```

- Another example to open a file "test" in input mode, the statements can be as given below :

```
ifstream i;
i.open("test",ios::in);
```

- For opening a file in input and output mode, the statements can be as given below:

```
ifstream i;
i.open("test",ios::in | ios::out);
```

- To close a file the function close() is to be used. This can be simply done as shown below:

```
i.close();
```

6.7 Using Streams for Reading from and Writing to the Files

SPPU - Dec. 18

Q. Which classes are used in file stream operations? Explain in detail. **(Dec. 18, 7 Marks)**

- The functions to read and write the data from a file can be done by fprintf() and fscanf() or simply using the insertion operator (<<) and extraction operator (>>).
- For the insertion and extraction operators the objects created are required for reading or writing data from/to file; instead of the objects "cout" and "cin" that are required for the reading or writing data from/to the console.
- Let us see some programs to access a file.

Program 6.7.1 : Write a program to write and read a string from/to a file.

```
#include<iostream.h>
#include<fstream.h>
#include<conio.h>
void main()
{
   char msg[20];
   ofstream o;
   o.open("test0",ios::in);
   o<<"HelloFriends!!"<<endl;
   o<<"Bye!!"<<endl;
   o.close();
   ifstream i;
   i.open("test0",ios::out);
   i>>msg;
   cout<<msg<<endl;
   i>>msg;
   cout<<msg<<endl;
   getch();
}
```

Output

```
HelloFriends!!
Bye!!
```

Program 6.7.2 : Write a program to write and read string, integer and float from/to a file.

```
#include<iostream.h>
#include<fstream.h>
#include<conio.h>
void main()
{
   char name[20];
   int roll;
   float fee;
   ofstream o;
   o.open("test0",ios::in);
   o<<"Ajay"<<endl;
   o<<24<<endl;
   o<<55988.45<<endl;
   o.close();
   ifstream i;
   i.open("test0",ios::out);
   i>>name;
```

```
    i>>roll;
    i>>fee;
    cout<<"Name:"<<name<<endl;
    cout<<"Roll No."<<roll<<endl;
    cout<<"Fee:"<<fee<<endl;
    getch();
}
```

Output

```
Name:Ajay
Roll No.24
Fee:55988.449219
```

6.7.1 Reading/Writing Bytes and Detecting End of File

- To detect an end of file we have the function eof() that can be called by the ifstream class object. This function returns a true of false based on the pointer pointing to the end of file or not respectively.
- Let us see a program example for the same.

Program 6.7.3 : Write a program that reads a text file and creates another text file that is identical except that every letter must be converted to lower case irrespective of its original case (e.g 'a' or 'A' will become 'a').

```
#include<conio.h>
#include<iostream.h>
#include<fstream.h>
void main()
{
    clrscr();
    char msg;
    ofstream o;
    o.open("test123",ios::in);
    ifstream in;
    in.open("test0",ios::out);
    while(!in.eof())
    {
        in.get(msg);
        if(msg>='A' && msg<='Z')
        msg=msg+'a'-'A';
        o<<msg;
    }
    o.close();
    in.close();
    in.open("test123",ios::out);
    while(!in.eof())
    {
        in.get(msg);
        cout<<msg;
    }
    getch();
}
```

Output

```
this is a bench.
```

Program 6.7.4 : Write a program that opens two text files for reading data. It creates a third file that contains the text of first file and then that of second file (text of second file to be appended after text of the first file, to produce the third file).

```
#include<iostream.h>
#include<fstream.h>
#include<conio.h>
void main()
{
    char msg;
    int i;
    ofstream o;
    o.open("test11",ios::in);
    ifstream in;
    in.open("test0",ios::out);
    while(!in.eof())
    {
        in.get(msg);
        o<<msg;
    }
    o.close();
    in.close();
    o.open("test11",ios::app);
    in.open("test00",ios::out);
    while(!in.eof())
    {
        in.get(msg);
        o<<msg;
    }
```

```
    in.close();
    o.close();
    cout<<"\nData from concatenated file:\n";
    in.open("test11",ios::out);
    while(!in.eof())
    {
       in.get(msg);
       cout<<(msg);
    }
    getch();
}
```

Output

```
Data from concatenated file:
This is a bench.  This is a car.
```

Program 6.7.5 : Write a program to copy the contents of a source file student1.txt to a destination file student2.txt character by character.

```
#include<conio.h>
#include<iostream.h>
#include<fstream.h>
void main()
{
   clrscr();
   char msg;
   ofstream o;
   o.open("student1.txt",ios::in);
   ifstream in;
   in.open("student2.txt",ios::out);
   while(!in.eof())
   {
      in.get(msg);
      o<<msg;
   }
   o.close();
   in.close();
   in.open("student2.txt",ios::out);
   while(!in.eof())
   {
      in.get(msg);
      cout<<msg;
}
   getch();
}
```

Program 6.7.6 : Explain the output of the following programs.

```
#include<iostream.h>
#include<conio.h>
  typedef void(*Fptr)(int,int);
  void Add(int i,int j)
  {
     cout<<i<<"+"<<j<<"="<<i+j;
  }
  void subtract(int i,int j)
  {
     cout<<i<<"-"<<j<<"="<<i-j;
  }
  void main()
  {
     clrscr();
     Fptr ptr;
     ptr=& Add;
     ptr(1,2);
     cout<<endl;
     ptr=&subtract;
     ptr(3,2);
     getch();
  }
```

Output

```
1+2 = 3
3-2 = 1
```

The function pointer ptr, initially points to the Add function and then to the subtract function. Accordingly the parameters are passed to those functions and the function is called and executed.

Sr. No.	Manipulators	ios member function
1.	Manipulators do not return value.	ios member functions can return value.
2.	We can create our own manipulators.	We cannot create our own functions for ios.

Sr. No.	Manipulators	ios member function
3.	Multiple manipulators can be combined.	ios member functions cannot be combined.
4.	For manipulators we need to include the "iomanip.h" header file.	For ios functions we need to include the "iostream.h" header file.
5.	These are non-member functions.	These are member functions.

6.8 Handling Primitive Data Types and Random Access Files

Q. Explain the use of binary files in C++ with a program. **(7 Marks)**

- In the previous sections we have seen how to read or write integer, float numbers and string into the file. In this section we will see how to read or write a binary data from/to the file.
- To read and write the binary data we have the functions get() and put() respectively.

Program 6.8.1 : Write a program to write string, integer and float to a file and read it as binary data.

```
#include<iostream.h>
#include<fstream.h>
#include<conio.h>
void main()
{
   char name[20],ch;
   int roll;
   float fee;
   ofstream o;
   o.open("test0",ios::in);
   o<<"Ajay"<<endl;
   o<<24<<endl;
   o<<55988.45<<endl;
   o.close();
   ifstream i;
   i.open("test0",ios::out);
   while(i)
   {
      i.get(ch);
      cout<<ch;
   }
   getch();
}
```

Output

```
Ajay
24
55988.45
!
```

Program 6.8.2 : Write a program to store employee names with their designation and net pay to a file on console. Also display employee details from file. Create a class 'employee'.

```
#include<iostream.h>
#include<fstream.h>
#include<conio.h>
#include<stdio.h>
class employee
{
   char name[20], designation[20];
   int pay;
   public:
   void accept()
   {
      cout<<"Enter name, designation and net pay:";
      gets(name);
      gets(designation);
      cin>>pay;
      ofstream o;
      o.open("test0",ios::in);
      o<<name<<endl<<designation<<endl<<pay;
      o.close();
   }
   void display()
   {
      char n[20],d[20];
      int p;
      ifstream i;
      i.open("test0",ios::out);
      i>>n;
      i>>d;
      i>>p;
      cout<<n<<endl<<d<<endl<<p;
      i.close();
```

```
    }
};
void main()
{
    employee e;
    e.accept();
    e.display();
    getch();
}
```

Output

```
Enter name, designation and net pay:Ajay
Manager
20000
Ajay
Manager
20000
```

Program 6.8.3 : Write a program to display the contents of a text file in the reverse order (use pointer manipulation).

```
#include<iostream.h>
#include<fstream.h>
#include<conio.h>
void main()
{
    char text[200],c;
    int i=0;
    ifstream in;
    in.open("test0",ios::out);
    while(!in.eof())
    {
        in.get(c);
        text[i]=c;
        i++;
    }
    i--;
    while(i>=0)
    {
        cout<<text[i];
        i--;
    }
    in.close();
    getch();
}
```

Output

```
!00002
reganaM
yajA
```

6.9 Concatenating and Buffering Files

- In some cases, we need to access multiple files. The files may need to be concatenated or separated for doing certain operations.
- We will see some program exampled in case where we will use multiple files by buffering them or concatenating them

Program 6.9.1 : Write a C++ program to read character data from a file. Create one file to store all capital alphabets and another file to store all small case alphabets. Also display contents of both files.

```
#include<iostream.h>
#include<fstream.h>
#include<conio.h>
void main()
{
    char msg;
    int i;
    ofstream o;
    o.open("test0",ios::in);
    for(i=0;i<=25;i++)
    {
        o<<char('A'+i)<<endl;
    }
    o.close();
    o.open("test01",ios::in);
    for(i=0;i<=25;i++)
    {
        o<<char('a'+i)<<endl;
    }
    o.close();
    ifstream in;
in.open("test0",ios::out);
    cout<<"Reading from file 1:\n";
    for(i=1;i<=26;i++)
    {
        in>>msg;
```

```
    cout<<msg<<"\t";
  }
  cout<<"\n\n\nReading from file 2:"<<endl;
  in.close();
  in.open("test01",ios::out);
  for(i=1;i<=26;i++)
  {
    in>>msg;
    cout<<msg<<"\t";
  }
  in.close();;
  getch();
}
```

Output :

```
Reading from file 1:
A B C D E F G H I J K
L M N O P Q R S T U V
W X Y Z
Reading from file 2:
a b cde f g h i j k
l m n o p q r s t u v
w xy z
```

Program 6.9.2 : Write a c++ program which opens two text files (only if they exist) and concatenates the contents of the second file to the first. Display an error message if the file do not exist.

```
#include<iostream.h>
#include<fstream.h>
#include<conio.h>
void main()
{
  char msg;
  int i;
  ofstream o;
  o.open("test0",ios::app);
  ifstream in;
  in.open("test01",ios::out);
  for(i=1;i<=26;i++)
  {
    in>>msg;
    o<<msg<<"\t";
  }
  in.close();
  o.close();
  in.open("test0",ios::out);
  cout<<"\n\n\nReading from concatenated file "<<endl;
  for(i=1;i<=52;i++)
  {
    in>>msg;
    cout<<msg<<"\t";
  }
  in.close();;
  getch();
}
```

Output :

```
Reading from concatenated file
A B C D E F G H I J
K L M N O P Q R S T
U V W X Y Z a b c d
e f g h I j k l m n
o p q r s t u v wx
y z
```

Program 6.9.3 : A file "student.txt" contains roll numbers and names. Write a C++ program to read the contents of this file and search for a student having a specific roll number.

```
#include<iostream.h>
#include<fstream.h>
#include<conio.h>
void main()
{
  char name[20];
  int roll,search,i;
  ifstream in;
  in.open("test02",ios::out);
  for(i=1;i<=4;i++)
  {
    in>>roll;
    cout<<roll<<"\t";
    in>>name;
    cout<<name<<endl;
  }
  in.close();
  cout<<"Enter roll number of student to search:";
  cin>>search;
```

```
    in.open("test02",ios::out);
    do
    {
       in>>roll;
       in>>name;
    }while(search!=roll);
    cout<<"roll number "<<search<<" is of student with
name "<<name;
    getch();
}
```

Output

```
24      Ajay
16      Jay
48      Vijay
64      Digvijay
Enter roll number of student to search:16
roll number 16 is of student with name Jay
```

Program 6.9.4 : A file contains a list of telephone numbers in the following form :

John	23406
Ahmed	56789
…	…

The names contain only one word and the names and telephone numbers are separated by white spaces. Write a program to read the file and output the list in two columns. The names should be left justified and numbers should be right-justified in a suitable field width.

```
#include<iostream.h>
#include<fstream.h>
#include<conio.h>
void main()
{
char name[20];
int phone,i;
ifstream in;
in.open("test02.txt",ios::out);
for(i=1;i<=4;i++)
{
   in>>name;
   cout<<name<<"\t";
   in>>phone;
   cout<<phone<<endl;
}
in.close();
getch();
}
```

Output

```
Ajay 12345
Jay     01267
```

6.10 Design Patterns : Introduction

- There are various design patterns used in the making of softwares. The design pattern to be used depends on the case.
- Design patterns are basically used to give the method for designing the software or application based on the class diagram.
- We need to decide the best design pattern that is suitable to our application. The design pattern that fits best for our application is to be selected.
- It may also be possible that multiple design patterns may be required at different parts in our software.
- There are various design patterns like adaptor, decorator, state, composite, observer, singleton, template method, factory method, abstract factory, strategy, momento, command, proxy, iterator, visitor, interpretor, builder, prototype, bridge, facade, mediator and many more
- We will discuss some of the widely used design patterns in the consecutive next section.

6.11 Types of Design Patterns: Adaptor, Singleton and Iterator

- We will see the three design patterns viz. Adaptor, singleton and iterator in this section

6.11.1 Adaptor

- In this case, we make a new adaptor class. This is used when we have to use an existing class in our application through a new class with another interface required.

- For example, if we have an existing class “A” in a particular application, but it doesn’t have a required interface “B”; then we will make another class called as adaptor class “C” that is extended from the class “A” and implements the interface “B”
- This can be as shown in the Fig. 6.11.1.

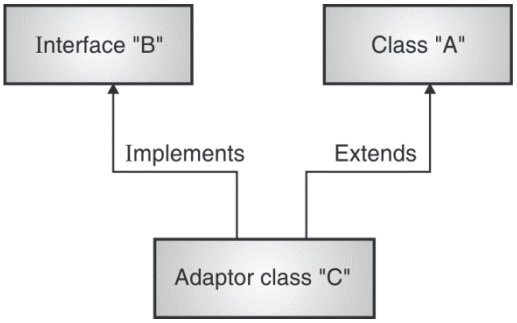


Fig. 6.11.1 :Adaptor

6.11.2 Singleton

- This is a special design case wherein only one instance or object of each class is made and the object is made accessible globally.
- Hence in this case, the client or the user made class need not create an object. The object can be obtained by some method like getInstance().
- The field members of the instance or object may be static and private.

6.11.3 Iterator

- This gives the option for the user made class to access the members of the parent and child classes without understanding the representation used internally.
- In this case, we need not have to take care of implementing a separate interface and another class getting extended. We need to bother of only one class that will give all the method accessible to us.
- The iterator can be implemented externally or internally as per requirement.
- For example:If there is a class “A” and an interface “B” required by an user class “D”. There may be implementation of an Iterator class “C” that implements the interface “B” and the class “A”.
- The user therefor need not bother about the various classes or interfaces. Instead the user class “D” needs to extend only the Iterator class “C”

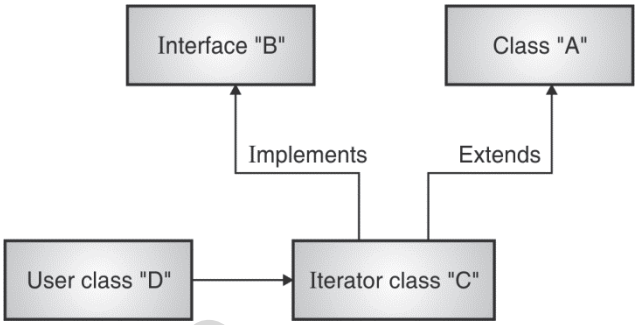


Fig. 6.11.2 : Iterator

6.12 Student Management System

- Storing the information of students is a very important requirement in colleges.
- Information like name, fee, exam marks etc need to be stored.
- The following program example, demonstrates some of these .

```
import java.util.*;
class Department
{
   private String dept_name;
   private int dept_number;
   private int
code_number,active=1,student_count=0,intake,flag;
   private String temp;
   String student[] = new String[400];
   public Department(int x,String str)
   {
      Scanner sc = new Scanner(System.in);
      dept_name = str;
      code_number = 2000000+1000*x;
      dept_number=x;
      System.out.print("Enter intake capacity : ");
      intake=sc.nextInt();
      System.out.println("Branch Added : "+dept_name);
   }
   public void addStudent()
   {
      String first_name,mid_name,last_name;
```

```
    int i,j;
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter first name");
    first_name=sc.next();
    System.out.println("Enter middle name");
    mid_name=sc.next();
    System.out.println("Enter last name");
    last_name=sc.next();
    student[student_count] = first_name+"
    "+mid_name+" "+last_name;
    student[student_count] =
    student[student_count].toUpperCase();
    temp=student[student_count];
    student_count++;
    sortStudents();
    System.out.println("Student added, press 7 to know roll
    numbers");
}
public void sortStudents()
{
    int i,j;
    for(i=0;i<=student_count-2;i++)
    {
        for(j=i;j<=student_count-2;j++)
        {
            if(student[j].compareTo(student[j+1])>0)
            {
                temp=student[j];
                student[j]=student[j+1];
                student[j+1]=temp;
            }
        }
    }
}
public int returnDeptNunber()
{
    return dept_number;
}
public boolean activity()
{
    if(active==0) return false;
    else return true;
}
public boolean searchDepartmentByName(String str)
{
    if(str.equalsIgnoreCase(dept_name)) return true;
    else return false;
}
public boolean searchDepartmentByNumber(int x)
{
    if(x==dept_number) return true;
    else return false;
}
public void removeDepartment()
{
    active=0;
}
public void restartDepartment()
{
    if(active==0)
    System.out.println("Branch reactivated");
    else System.out.println("Branch already exists");
    active=1;
}
public int checkIfActivatedJustNow(int x)
{
    if(active==0) return (x+1);
    else return x;
}
public boolean searchStudentByName(String str)
{
    int i;
    flag=-1;
    for(i=0;i<=student_count-1;i++)
    {
        if(str.equalsIgnoreCase(student[i])) flag=i;
    }
    if(flag!=-1) return true;
    else return false;
}
public int getStudentNumber()
{
    return flag;
}
public void removeStudent(int x)
{
    int i;
    showStudentInfo(x);
    for(i=x;i<=student_count-2;i++)
    {
        student[i]=student[i+1];
    }
    student_count--;
    System.out.println("Removed");
}
public boolean checkVacancy()
```

```
    {
      if(student_count<intake) return true;
      else return false;
    }
    public void showStudentInfo(int x)
    {
System.out.println("Name : "+student[x]+"\nRoll no : "+(code_number+x+1)+"\nBranch : "+dept_name);
    }
    public void showAllStudents()
    {
      int i;
      System.out.println("Roll\tStudent name\n");
      for(i=0;i<=student_count-1;i++)
      {
System.out.println((code_number+i+1)+"\t"+student[i]);
      }
    }
    public void updateIntake()
    {
      int copy=intake;
      System.out.println("Present Intake = "+intake);
      Scanner sc = new Scanner(System.in);
      do
      {
System.out.print("Enter new intake(must be greater than or equal to old) : ");
        intake=sc.nextInt();
      }while(intake<copy);
    }
    public int getRollNumber(int x)
    {
      return (code_number+x+1);
    }
    public boolean numberExists(int x)
    {
      if(x%1000<student_count) return true;
      else return false;
    }
}
public class College
{
    public static void main(String args[])
    {
int count = 0, dept_count = 0, dummy_count, choice, i, j, copy, data = 0;
      String test_string="";
      Department d[] = new Department[100];
      Scanner sc = new Scanner(System.in);
      do
      {
        copy=0;
      if(dept_count==0)
   System.out.println("1.Add Branch");
        else
System.out.println("\n1.Add branch\n2.Remove branch\n3.Add student\n4.Remove student\n5.Get student information\n6.Update Intake\n7.Display all students in a branch\n8.Exit\n");
        choice=sc.nextInt();
        sc.nextLine();

   if(choice!=5&&choice!=4&&choice>=1&&choice<8)
        {
          System.out.println("Enter branch name");
          test_string=sc.nextLine();
          test_string=test_string.toUpperCase();
          for(i=0;i<=count-1;i++)
          {
            if(d[i].searchDepartmentByName(test_string))
            {
              copy=1;
              data=i;
              break;
            }
          }
   if(count>0&&copy==0&&choice>1)
   System.out.println("Branch not found");
        }
        switch(choice)
        {
          case 1:  if(copy==0)
            {
              d[count]=new
   Department(count+1,test_string);
              count++;
              dept_count++;
            }
            else
            {
     dept_count=d[data].checkIfActivatedJustNow
     (dept_count);
              d[data].restartDepartment();
            }
            break;
case 2: if(dept_count==0) System. out. Println ("Add a branch first");
```

```
                else if(copy!=0)
                {
                    d[data].removeDepartment();
                    System.out.println("Branch removed");
                    dept_count--;
                }
                break;
case 3: if(dept_count==0)
System.out.println("Add a branch first");
                else if(copy!=0)
                {

   if(d[data].checkVacancy()&&d[data].activity())
                    {
                        d[data].addStudent();
                    }
                    else
System.out.println("Branch isn't accepting more students");
                }
                break;
            case 4: if(dept_count==0)
                {
                    System.out.println("Add a branch first");
                    break;
                }
System.out.print("Enter full name of student OR roll number
: ");
                dummy_count=0;
                test_string=sc.nextLine();
                if(test_string.charAt(0)=='2')
dummy_count=Integer.parseInt(test_string);
                for(i=0;i<=count-1;i++)
                {
                    if(d[i].searchStudentByName(test_string))
                    {

dummy_count=d[i].getRollNumber(d[i].getStudentNumber());
                        break;
                    }
                }

   if(dummy_count/100000!=20||!d[((dummy_count-
2000000)/1000)-1].numberExists(dummy_count-1))
System.out.println("Invalid name OR number");
                else
d[((dummy_count-2000000)/1000)-
1].removeStudent(dummy_count%1000-1);
                break;
```

```
            case 5: if(dept_count==0)
                {
                    System.out.println("Add a branch first");
                    break;
                }
System.out.print("Enter full name of student OR roll number
: ");
                dummy_count=0;
                test_string=sc.nextLine();
                if(test_string.charAt(0)=='2')
dummy_count=Integer.parseInt(test_string);
                for(i=0;i<=count-1;i++)
                {
                    if(d[i].searchStudentByName(test_string))
                    {

dummy_count=d[i].getRollNumber(d[i].getStudentNumber());
                        break;
                    }
                }

   if(dummy_count/100000!=20||!d[((dummy_count-
2000000)/1000)-1].numberExists(dummy_count-1))
System.out.println("Invalid name OR number");
                else
d[((dummy_count-2000000)/1000)-
1].showStudentInfo(dummy_count%1000-1);
                break;
            case 6: if(dept_count==0)
                {
                    System.out.println("Add a branch first");
                    break;
                }
                else if(copy!=0) d[data].updateIntake();
                break;
            case 7: if(dept_count==0)
                {
                    System.out.println("Add a branch first");
                    break;
                }
                else if(copy!=0) d[data].showAllStudents();
                break;
        }
    }while(choice!=8);
  }
}
```


❑❑❑

Model Question Paper (End Sem.)

Object Oriented Programming

Semester - III (Information Technology)

Time : 2 ½ Hour **Maximum Marks : 70**

Instruction to the candidates :

1) **Answer Q. 1 or Q. 2, Q. 3 or Q. 4 , Q. 5 or Q. 6, Q. 7 or Q. 8**

2) **Neat diagrams must be drawn wherever necessary.**

3) **Figure to the right indicate full marks.**

4) **Make suitable assumptions, if necessary.**

Q. 1 **(a)** What is constructor? List different types of constructors and write program example for any one.

(Refer Sections 3.1.1 and 3.2) **(6 Marks)**

(b) Explain destructor with a program example. **(Refer Section 3.6)** **(6 Marks)**

(c) 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(). **(Refer Program 3.2.8)** **(6 Marks)**

OR

Q. 2 **(a)** Explain copy constructor with a program example. **(Refer Section 3.2.3)** **(6 Marks)**

(b) Write a program to calculate the area of triangle, rectangle and circle using constructor overloading. The program should be menu-driven. **(Refer Program 3.3.1)** **(6 Marks)**

(c) Write a program to find area of circle using C++. The value of the radius must be accepted from the user in the main program and passed to the parameterized constructor and the class circle must have two inline functions namely : (a) compute() for calculating the area. (b) display() for displaying the result.

(Refer Program 3.2.3) **(6 Marks)**

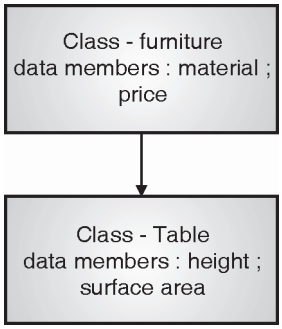
Q. 3 **(a)** List the different types of Inheritance in C++ and Write a program to add two numbers using single inheritance such that the base class function must accept the two numbers from the user and the derived class function must add these numbers and display the sum. **(Refer Section 4.1.1(A) and Program 4.2.1)** **(6 Marks)**

(b) Write a program to calculate percentage of a student using multi level inheritance. The base class function will accept the marks in three subjects from user. A class will be derived from the above mentioned class that will have a function to find the total marks obtained and another class derived from this will have functions to calculate and display the percentage scored.**(Refer Program 4.2.6)** **(6 Marks)**

(c) Differentiate between Interface and Abstract class. **(Refer Section 4.5.2(C))** **(5 Marks)**

OR

Q. 4 **(a)** Write a program to implement single inheritance from following Fig. P. 4.2.3 accept and display the data for one table. **(Refer Program 4.2.3)** **(6 Marks)**



(b) Differentiate between multiple inheritance and multilevel inheritance. **(Refer Section 4.2.2)** **(5 Marks)**

(c) List rules of virtual function. Write a program to demonstrate dynamic binding using virtual function.

(Refer Section 4.7.2 and Program 4.7.2) **(6 Marks)**

Q. 5 **(a)** Explain the concept of checked and unchecked exceptions with examples.

(Refer Section 5.2.1 and 5.2.2) **(6 Marks)**

(b) Write a C++ program to demonstrate the use of try catch block with the arguments as an integer and a string using multiple catch blocks. **(Refer Program 5.8.1)** **(6 Marks)**

(c) Explain LinkedList class with a program example. **(Refer Section 5.11.2)** **(6 Marks)**

OR

Q. 6 **(a)** Explain the syntax of try-catch-finally in detail. **(Refer Section 5.3)** **(6 Marks)**

(b) Write a Java 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 (Program 5.4.2)(6 Marks)

(c) What are generics? Write a C++ program using a class template to read any parameterized data type such as float and integer and print them in sorted form. **(Refer Section 5.9 and Program 5.9.5)** **(6 Marks)**

Q. 7 **(a)** Which classes are used in file stream operations explain it in details. **(Refer Section 6.3.1)** **(6 Marks)**

(b) Write a program to store employee names with their designation and net pay to a file on console. Also display employee details from file. Create a class 'employee'. **(Refer Program 6.8.2)** **(6 Marks)**

(c) Explain the following file mode ios::trunk, ios::app, ios::ate, ios::binary. **(Refer Section 6.6)** **(5 Marks)**

OR

Q. 8 **(a)** List and explain the IO Exceptions. **(Refer Section 6.5)** **(6 Marks)**

(b) Explain the design patterns: Adaptor, Singleton and Iterator **(Refer Sections 6.10 and 6.11)** **(6 Marks)**

(c) Write a program to write and read a string from/to a file. **(Refer Program 6.7.1)** **(5 Marks)**

❑❑❑