

As per Revised Syllabus of  
**SAVITRIBAI PHULE PUNE UNIVERSITY**  
Choice Based Credit System (CBCS)  
S.E. (Computer) Semester - IV

# **PRINCIPLES OF PROGRAMMING LANGUAGES**

**Anuradha A. Puntambekar**

M.E. (Computer)

Formerly Assistant Professor in  
P.E.S. Modern College of Engineering,  
Pune

**Dr. Jyoti Rao**

Ph. D. (Computer Engineering)

Professor in Computer Engineering Department  
Dr. D.Y. Patil Institute of Technology,  
Pimpri, Pune.



# PRINCIPLES OF PROGRAMMING LANGUAGES

Subject Code : 210255

S.E. (Computer) Semester - IV

© Copyright with Anuradha A. Puntambekar

All publishing rights (printed and ebook version) reserved with Technical Publications. No part of this book should be reproduced in any form, Electronic, Mechanical, Photocopy or any information storage and retrieval system without prior permission in writing, from Technical Publications, Pune.

Published by :

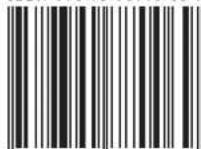


Amit Residency, Office No.1, 412, Shaniwar Peth,  
Pune - 411030, M.S. INDIA, Ph.: +91-020-24495496/97  
Email : sales@technicalpublications.org Website : www.technicalpublications.org

Printer :

Yogiraj Printers & Binders  
Sr.No. 10/1A,  
Ghule Industrial Estate, Nanded Village Road,  
Tal. - Haveli, Dist. - Pune - 411041.

ISBN 978-93-90770-06-9



9789390770069

SPPU 19

# PREFACE

The importance of **Principles of Programming Languages** is well known in various engineering fields. Overwhelming response to our books on various subjects inspired us to write this book. The book is structured to cover the key aspects of the subject **Principles of Programming Languages**.

The book uses plain, lucid language to explain fundamentals of this subject. The book provides logical method of explaining various complicated concepts and stepwise methods to explain the important topics. Each chapter is well supported with necessary illustrations, practical examples and solved problems. All chapters in this book are arranged in a proper sequence that permits each topic to build upon earlier studies. All care has been taken to make students comfortable in understanding the basic concepts of this subject.

Representative questions have been added at the end of each section to help the students in picking important points from that section.

The book not only covers the entire scope of the subject but explains the philosophy of the subject. This makes the understanding of this subject more clear and makes it more interesting. The book will be very useful not only to the students but also to the subject teachers. The students have to omit nothing and possibly have to cover nothing more.

We wish to express our profound thanks to all those who helped in making this book a reality. Much needed moral support and encouragement is provided on numerous occasions by our whole family. We wish to thank the **Publisher** and the entire team of **Technical Publications** who have taken immense pain to get this book in time with quality printing.

Any suggestion for the improvement of the book will be acknowledged and well appreciated.

*Authors*  
A. A. Puntambekar  
Dr. Jyoti Rao

*Dedicated to God.*

# SYLLABUS

## Principles of Programming Languages - 210255

Credit Scheme	Examination Scheme and Marks
03	Mid_Semester (TH) : 30 Marks End_Semester (TH): 70 Marks

### Unit I Fundamentals of Programming

Importance of Studying Programming Languages. History of Programming Languages. Impact of Programming Paradigms. Role of Programming Languages. Programming Environments. Impact of Machine Architectures: The operation of a computer, Virtual Computers and Binding Times.

**Programming paradigms** - Introduction to programming paradigms. Introduction to four main Programming paradigms- procedural, object oriented, functional, and logic and rule based. (Chapter - 1)

### Unit II Structuring the Data, Computations and Program

**Elementary Data Types** : Primitive data Types, Character String types, User Defined Ordinal Types, Array types, Associative Arrays, Record Types, Union Types, Pointer and reference Type.

**Expression and Assignment Statements** : Arithmetic expression, Overloaded Operators, Type conversions, Relational and Boolean Expressions, Short Circuit Evaluation, Assignment Statements, Mixed mode Assignment. **Statement level Control Statements** : Selection Statements, Iterative Statements, Unconditional Branching. **Subprograms** : Fundamentals of Sub Programs, Design Issues for Subprograms, Local referencing Environments, Parameter passing methods.

**Abstract Data Types and Encapsulation Construct** : Design issues for Abstraction, Parameterized Abstract Data types, Encapsulation Constructs, Naming Encapsulations. (Chapter - 2)

### Unit III Java as Object Oriented Programming Language-Overview

**Fundamentals of JAVA, Arrays** : one dimensional array, multi-dimensional array, alternative array declaration statements.

**String Handling** : String class methods, **Classes and Methods** : class fundamentals, declaring objects, assigning object reference variables, adding methods to a class, returning a value,

constructors, this keyword, garbage collection, finalize() method, overloading methods, argument passing, object as parameter, returning objects, access control, static, final, nested and inner classes, command line arguments, variable - length arguments. (**Chapter - 3**)

## **Unit IV Inheritance, Packages and Exception Handling using Java**

**Inheritances** : member access and inheritance, super class references, Using super, multilevel hierarchy, constructor call sequence, method overriding, dynamic method dispatch, abstract classes, Object class.

**Packages and Interfaces** : defining a package, finding packages and CLASSPATH, access protection, importing packages, interfaces (defining, implementation, nesting, applying), variables in interfaces, extending interfaces, instance of operator. Fundamental, exception types, uncaught exceptions, try, catch, throw, throws, finally, multiple catch clauses, nested try statements, built-in exceptions, custom exceptions (creating your own exception sub classes).

**Managing I/O** : Streams, Byte Streams and Character Streams, Predefined Streams, Reading console Input, Writing Console Output, Print Writer class. (**Chapter - 4**)

## **Unit V Multithreading in Java**

**Concurrency and Synchronization, Java Thread Model** : Thread priorities, Synchronization, Messaging, Main Thread, Creating thread: Implementing Thread using thread class and Runnable interface. Creating multiple threads using is Alive() and join().

**Web Based Application in Java** : Use of JavaScript for creating web based applications in Java, Introduction to Java script frameworks- ReactJS, VueJS, AngularJS (open source). (**Chapter - 5**)

## **Unit VI Logical and Functional Programming**

**Functional Programming Paradigm** : Understanding symbol manipulation, Basic LISP functions, definitions, predicates, conditionals and scoping, Recursion and iteration, Properties List array and access functions, Using lambda definitions, printing, reading and atom manipulation.

**Logic Programming Paradigm** : An Overview of Prolog, Syntax and Meaning of Prolog Programs, Lists, Operators, Arithmetic, Using Structures. (**Chapter - 6**)

# TABLE OF CONTENTS

## Unit - I

<b>Chapter - 1      Fundamentals of Programming</b>	<b>(1 - 1) to (1 - 26)</b>
1.1 Importance of Studying Programming Languages .....	1 - 2
1.2 History of Programming Languages.....	1 - 3
1.3 Role of Programming Languages .....	1 - 4
1.3.1 Evolution of Major Programming Languages .....	1 - 4
1.3.2 Attributes of Good Language.....	1 - 7
1.4 Programming Environments .....	1 - 8
1.4.1 Effect on Language Design.....	1 - 9
1.4.2 Environment Framework.....	1 - 10
1.5 Impact of Machine Architectures .....	1 - 10
1.5.1 Operation of a Computer .....	1 - 10
1.5.2 Translators and Virtual Architectures.....	1 - 13
1.5.3 Methods of Program Implementation .....	1 - 13
1.5.4 Virtual Computers .....	1 - 16
1.5.5 Binding and Binding Times .....	1 - 17
1.6 Programming Paradigms .....	1 - 19
1.6.1 Imperative Programming.....	1 - 19
1.6.2 Object Oriented Programming .....	1 - 20
1.6.3 Functional Programming .....	1 - 21
1.6.4 Logic Programming.....	1 - 21
1.7 Multiple Choice Questions with Answers .....	1 - 22

## Unit - II

---

### Chapter - 2 Structuring the Data, Computations and Program

**(2 - 1) to (2 - 52)**

2.1 Elementary Data Types.....	2 - 2
2.1.1 Primitive Data Types.....	2 - 2
2.1.2 Character String Types.....	2 - 6
2.1.2.1 Design Issues.....	2 - 6
2.1.2.2 String Operation.....	2 - 7
2.1.2.3 String Length Options.....	2 - 7
2.1.2.4 Implementation of Character String Type.....	2 - 7
2.1.3 User Defined Ordinal Types.....	2 - 8
2.1.4 Array Types.....	2 - 9
2.1.4.1 Design Issues.....	2 - 10
2.1.4.2 Subscript Binding for Arrays.....	2 - 10
2.1.4.3 Array Categories based on Subscript Binding.....	2 - 10
2.1.4.4 Heterogeneous Array .....	2 - 11
2.1.4.5 Array Initialization .....	2 - 11
2.1.4.6 Array Operations.....	2 - 11
2.1.4.7 Rectangular and Jagged Arrays .....	2 - 12
2.1.4.8 Slices.....	2 - 12
2.1.5 Associative Arrays.....	2 - 13
2.1.6 Record Types .....	2 - 14
2.1.7 Union Types.....	2 - 14
2.1.7.1 Design Issues .....	2 - 15
2.1.7.2 Discriminated and Free Unions .....	2 - 15
2.1.7.3 ADA Union Types.....	2 - 15
2.1.8 Pointer and Reference Types.....	2 - 16
2.1.8.1 Design Issues .....	2 - 17
2.1.8.2 Point Operations .....	2 - 17
2.1.8.3 Pointer Problems.....	2 - 19
2.1.8.4 Pointers in Various Languages.....	2 - 19

2.1.8.5 Reference Type.....	2 - 20
<b>2.2 Expression and Assignment Statements.....</b>	<b>2 - 22</b>
<b>2.2.1 Arithmetic Expressions .....</b>	<b>2 - 22</b>
2.2.1.1 Design Issues .....	2 - 22
2.2.1.2 Precedence and Associativity.....	2 - 22
2.2.1.3 Operand Evaluation Order .....	2 - 23
2.2.2 Overloaded Operators.....	2 - 24
2.2.3 Type Conversion .....	2 - 24
2.2.4 Relational and Boolean Expression .....	2 - 25
2.2.5 Short Circuit Evaluation .....	2 - 26
2.2.6 Assignment Statement .....	2 - 26
2.2.7 Mixed Mode Assignment.....	2 - 27
<b>2.3 Statement-level Control Statements.....</b>	<b>2 - 28</b>
2.3.1 Selection Statements.....	2 - 28
2.3.2 Iterative Statements .....	2 - 29
2.3.2.1 Counter Controlled Loops .....	2 - 30
2.3.2.2 Logically Controlled Loops.....	2 - 30
2.3.2.3 User-located Loop Control Mechanism.....	2 - 31
2.3.2.4 Examples of Looping in C.....	2 - 31
2.3.3 Unconditional Branching .....	2 - 32
<b>2.4 Subprograms.....</b>	<b>2 - 33</b>
2.4.1 Fundamentals of Subprograms .....	2 - 33
2.4.1.1 General Subprogram Characteristics.....	2 - 34
2.4.1.2 Basic Definitions .....	2 - 34
2.4.1.3 Procedure and Functions .....	2 - 34
2.4.2 Design Issues for Subprogram .....	2 - 36
2.4.3 Local Referencing Environments .....	2 - 36
2.4.3.1 Local Variables.....	2 - 37
2.4.3.2 Nested Subprograms.....	2 - 38
2.4.4 Parameter Passing Methods.....	2 - 38
<b>2.5 Abstract Data Types and Encapsulation Construct.....</b>	<b>2 - 41</b>

2.5.1 Design Issues For Abstraction.....	2 - 42
2.5.2 Parameterized Abstract Data Types .....	2 - 42
2.5.3 Encapsulation Constructs .....	2 - 44
2.5.4 Naming Encapsulations .....	2 - 45
<b>2.6 Multiple Choice Questions with Answers .....</b>	<b>2 - 46</b>

### **Unit - III**

## **Chapter - 3 Java as Object Oriented Programming Language - Overview (3 - 1) to (3 - 88)**

3.1 Fundamentals of Java .....	3 - 3
3.1.1 Features of Java .....	3 - 3
3.1.2 Difference between C, C++ and Java .....	3 - 5
3.2 Java Program Structure .....	3 - 7
3.3 How to Write a Java Program ? .....	3 - 8
3.4 Comments in Java .....	3 - 10
3.5 Variables .....	3 - 10
3.6 Keywords .....	3 - 10
3.7 Data Types and Size .....	3 - 11
3.8 Operators.....	3 - 13
3.9 User Defined Data Types .....	3 - 17
3.10 Control Statements.....	3 - 18
3.11 Jump Statements .....	3 - 29
3.12 Arrays.....	3 - 33
3.12.1 One Dimensional Array.....	3 - 34
3.12.2 Two Dimensional Array .....	3 - 36
3.13 String Handling.....	3 - 39
3.13.1 Finding Length of String.....	3 - 40
3.13.2 Concatenation .....	3 - 42
3.13.3 Character Extraction.....	3 - 42

3.13.4 String Comparison .....	3 - 43
3.13.5 Searching for the Substring .....	3 - 44
3.13.6 Replacing the Character from String .....	3 - 44
3.13.7 Upper Case and Lower Case .....	3 - 45
<b>3.14 Classes and Methods .....</b>	<b>3 - 45</b>
3.15 Objects .....	3 - 48
3.16 Adding Methods to Class .....	3 - 50
<b>3.17 Constructor .....</b>	<b>3 - 51</b>
3.17.1 Special Properties of Constructor.....	3 - 53
3.17.2 Types of Constructors.....	3 - 54
<b>3.18 this keyword .....</b>	<b>3 - 57</b>
3.19 Garbage Collection .....	3 - 60
3.20 finalize() Method .....	3 - 62
3.21 Overload Methods.....	3 - 63
3.22 Argument Passing .....	3 - 65
3.23 Object as Parameter .....	3 - 66
3.24 Returning Object.....	3 - 68
3.25 Access Control.....	3 - 68
3.26 Static Members.....	3 - 70
3.27 Nested and Inner Class .....	3 - 72
3.28 Command Line Argument.....	3 - 74
3.29 Variable Length Arguments .....	3 - 75
<b>3.30 Multiple Choice Questions with Answers.....</b>	<b>3 - 76</b>

## Unit - IV

### Chapter - 4 Inheritance, Packages and Exception Handling using Java (4 - 1) to (4 - 88)

#### Part I : Inheritance

4.1 Introduction to Inheritance .....	4 - 3
4.2 Types of Inheritance .....	4 - 4
4.3 Implementation of Single Inheritance.....	4 - 5
4.4 Member access and Inheritance.....	4 - 8
4.5 Super Class Reference.....	4 - 9
4.6 Using Super .....	4 - 11
4.7 Use of Final Keyword .....	4 - 13
4.7.1 Final Variables and Methods .....	4 - 13
4.7.2 Final Classes to Stop Inheritance.....	4 - 14
4.8 Multilevel Hierarchy .....	4 - 15
4.9 Constructor Call Sequence.....	4 - 17
4.10 Method Overriding .....	4 - 19
4.11 Polymorphism.....	4 - 21
4.11.1 Dynamic Method Dispatch .....	4 - 23
4.12 Abstract Classes .....	4 - 25
4.13 Object Class.....	4 - 28

#### Part II : Packages and Interfaces

4.14 Defining a Package.....	4 - 29
4.15 Finding Packages.....	4 - 30
4.16 Adding Class to a Package.....	4 - 31
4.17 CLASSPATH.....	4 - 32
4.18 Access Protection.....	4 - 33

4.19 Importing Packages.....	4 - 33
4.20 Nested Packages .....	4 - 33
4.21 JAVA API Package.....	4 - 35
4.22 Interfaces .....	4 - 44
4.23 Variables in Interfaces .....	4 - 45
4.24 Extending Interfaces .....	4 - 46
4.25 Creation and Implementation of Interface.....	4 - 47
4.26 Multiple Inheritance .....	4 - 48
4.27 The Instance of Operator.....	4 - 49

### **Part III : Exception Handling**

4.28 Fundamentals .....	4 - 51
4.28.1 Benefits of Exception Handling.....	4 - 52
4.28.2 Difference between Error and Exception .....	4 - 52
4.29 Exception Types.....	4 - 52
4.30 try, catch, throw, throws and finally.....	4 - 53
4.30.1 Handling try-catch Block .....	4 - 54
4.30.2 Using finally .....	4 - 56
4.30.3 Using throws .....	4 - 57
4.31 Uncaught Exception.....	4 - 58
4.32 Multiple Catch Clauses .....	4 - 60
4.33 Nested try Statements .....	4 - 62
4.34 Built-in Exceptions .....	4 - 63
4.35 Custom Exceptions .....	4 - 67

### **Part IV: Managing I/O**

4.36 Streams .....	4 - 69
4.37 Byte Streams and Character Streams .....	4 - 69
4.37.1 Byte Stream .....	4 - 69

4.37.2 Character Stream.....	4 - 70
4.37.3 Difference between Byte and Character Stream.....	4 - 70
<b>4.38 Predefined Streams .....</b>	<b>4 - 70</b>
4.39 Reading Console Input.....	4 - 71
4.40 Writing Console Output.....	4 - 73
4.41 Print Writer Class .....	4 - 74
4.42 FileInputStream/FileOutputStream .....	4 - 75
<b>4.43 Console I/O using Scanner Class .....</b>	<b>4 - 80</b>
4.43.1 Console Input.....	4 - 81
4.43.2 Console Output.....	4 - 81
<b>4.44 Multiple Choice Questions with Answers.....</b>	<b>4 - 82</b>

Unit - V

---

**Chapter - 5 Multithreading in Java**      **(5 - 1) to (5 - 56)**

**Part I : Concurrency and Synchronization, Java Thread Model**

5.1 Introduction to Thread .....	5 - 2
5.2 Creating Thread .....	5 - 2
5.2.1 Extending Thread Class.....	5 - 3
5.2.2 Using Runnable Interface .....	5 - 4
5.3 Thread Model .....	5 - 6
5.4 Thread Priorities .....	5 - 7
5.5 Synchronization Messaging .....	5 - 10
5.6 Creating Multiple Threads using Alive() and join() .....	5 - 15
5.6.1 The isAlive() Method .....	5 - 15
5.6.2 The join() Method.....	5 - 16

**Part II : Web Based Application in Java**

5.7 Introduction to JavaScript.....	5 - 18
5.7.1 Difference between Java and JavaScript .....	5 - 18

5.7.2 Features of JavaScript.....	5 - 18
5.8 Using JavaScript in HTML.....	5 - 19
5.9 Use of JavaScript for Creating Web Applications .....	5 - 22
5.9.1 Input and Output.....	5 - 22
5.9.2 Pop-Up Box .....	5 - 23
5.9.3 Function.....	5 - 25
5.9.4 Form Design.....	5 - 27
5.9.4.1 Text.....	5 - 27
5.9.4.2 Checkbox.....	5 - 29
5.9.4.3 Radio Button .....	5 - 30
5.9.4.4 Button.....	5 - 31
5.9.4.5 Select Element.....	5 - 33
<b>Part III : Introduction to JavaScript Framework</b>	
5.10 AngularJS.....	5 - 37
5.10.1 MVC Architecture .....	5 - 38
5.10.2 Directives.....	5 - 38
5.11 ReactJS.....	5 - 43
5.11.1 Installation of ReactJS.....	5 - 44
5.11.2 Writing Simple Web Applications.....	5 - 47
5.12 VueJS.....	5 - 48
5.12.1 Installation of VueJS .....	5 - 49
5.12.2 Writing Web Application .....	5 - 49
5.12.3 Basic Components of VueJS Script.....	5 - 51
<b>5.13 Multiple Choice Questions with Answers.....</b>	<b>5 - 54</b>

## Unit - VI

---

### **Chapter - 6     Logical and Functional Programming     (6 - 1) to (6 - 32)**

<b>Part I : Functional Programming Paradigm</b>
---

6.1 Introduction to Functional Programming.....	6 - 2
---	-------

6.2 Understanding Symbol Manipulation.....	6 - 3
6.3 Basic LISP Functions .....	6 - 4
6.3.1 Building Blocks.....	6 - 4
6.3.2 Lisp Manipulation Function .....	6 - 6
6.3.3 Mathematical Functions.....	6 - 7
6.3.4 Eval Function .....	6 - 7
6.3.5 Basic Lisp Primitives.....	6 - 7
6.4 Definitions, Predicates, Conditional and Scoping.....	6 - 9
6.4.1 Definitions .....	6 - 9
6.4.2 Predicates .....	6 - 10
6.4.3 Conditional Statements.....	6 - 11
6.4.4 Scoping .....	6 - 13
6.5 Recursion and Iteration .....	6 - 14
6.6 Properties, A- Lists, Arrays and Access Function.....	6 - 14
6.7 Using Lambda Definitions.....	6 - 16
6.8 Printing, Reading and Atom Manipulation .....	6 - 16

**Part II : Logic Programming Paradigm**

6.9 An overview of Prolog.....	6 - 17
6.10 Syntax and Meaning of Prolog Programs .....	6 - 19
6.10.1 Response to Query .....	6 - 22
6.11 Lists, Operators and Arithmetic.....	6 - 27
6.12 List Structures .....	6 - 28
6.13 Comparison between Functional Programming and Logical Programming and Object Oriented Programming Languages Functional .....	6 - 30
<b>6.14 Multiple Choice Questions with Answers.....</b>	<b>6 - 22</b>

## Notes

## **UNIT- I**

**1**

# **Fundamentals of Programming**

### **Syllabus**

*Importance of Studying Programming Languages, History of Programming Languages, Impact of Programming Paradigms, Role of Programming Languages, Programming Environments. Impact of Machine Architectures: The operation of a computer, Virtual Computers and Binding Times.*

*Programming paradigms - Introduction to programming paradigms, Introduction to four main Programming paradigms- procedural, object oriented, functional, and logic and rule based.*

### **Contents**

- 1.1 *Importance of Studying Programming Languages*
- 1.2 *History of Programming Languages*
- 1.3 *Role of Programming Languages*
- 1.4 *Programming Environments*
- 1.5 *Impact of Machine Architectures*
- 1.6 *Programming Paradigms.....Dec.-17, .....Marks 6*
- 1.7 *Multiple Choice Questions*

## 1.1 Importance of Studying Programming Languages

Following are the reasons for studying the programming language concepts -

**1) Increased capacity to express ideas :**

- The study of different programming languages makes the programmer with variety of programming constructs.
- In this study, programmer can understand various data structures, control structures abstractions and their ability.
- Then they can use suitable programming constructs to express the ideas they want to implement.

**2) Improved background for choosing appropriate language :**

- By the knowledge of various programming languages, one can be aware of its variety of features.
- Hence while developing a software application programmer can make appropriate selection of the language whose features are most applicable.
- For example – if application requirement is use of GUI then the programmer will choose Visual basic as a programming language.

**3) Increased ability to learn a new language :**

- Due to the knowledge of programming constructs and implementation techniques, programmer can learn new language easily.

**4) Better understanding of significance of implementation :**

- By learning and understanding different languages, programmer can know the reasons behind the language design.
- This knowledge leads to use the language more intelligently.

**5) Better use of languages that are already known :**

- By studying the concepts of programming languages, programmers can learn about previously unknown and unused parts of the languages they already use and begin to use those features.

**6) Overall advancement of computing :**

- By understanding variety of features of programming languages such as arrays, dynamic arrays, strings, dynamic memory allocation, generic programming - the programmer can use these features for advance level implementation and computing.

**Review Question**

1. Explain the importance of studying programming language.

## 1.2 History of Programming Languages

**1) Scientific applications :**

- The scientific applications require large numbers of floating point computations.
- It typically makes use of arrays.
- The programming language used is - Fortran.

**2) Business applications :**

- The business applications are characterized by facilities of producing variety of reports.
- It requires use of decimal numbers and characters data.
- It also needs an ability to specify decimal arithmetic operations.
- The programming language preferred for building the business applications is - COBOL.

**3) Artificial intelligence :**

- The artificial intelligence is an area in which the computations are symbolic rather than numeric. Symbolic computation means that symbols, consisting of names rather than numbers.
- Symbolic computation is more conveniently done with linked lists of data rather than arrays.
- This kind of programming sometimes requires more flexibility than other programming domains
- The programming language used in this area of applications is - LISP.

**4) Systems programming :**

- System programming is a system software which consists of collection of operating system and programming support tools.
- The system software is used continuously, hence it needs to be efficient.
- It should also have a support for interfacing with external devices to be written. (For example - some support programs for plug and play devices).
- The general programming languages such as C, C++ are usually preferred for system programming.

**5) Web software :**

- The world wide web software is used to represent some dynamic web contents, representation of text, audio or video data, contents containing some computations and so on.
- It need a support for variety of languages from markup languages (For example - HTML), scripting(For example - PHP), to general-purpose(For example -Java).

**1.3 Role of Programming Languages****1.3.1 Evolution of Major Programming Languages****FORTRAN I :**

- It is designed for new IBM 704 , which had index registers and floating point hardware.
- From this language onwards the idea of compilation came into existence.
- The first implemented version of Fortran has following features -
  - Names could have up to six characters.
  - Post-test counting loop such as DO was present.
  - It has formatted I/O
  - It allows user defined subprograms.
  - There were no data typing statements

**FORTRAN II :**

- It was distributed in 1958
- Independent compilation was supported for FORTRAN II

**FORTRAN IV,77,90 :**

- It is evolved during 1960-62
- It has explicit type declarations
- There was logical selection statement
- Subprogram names could be parameters
- It was standardized as FORTRAN 66 in 1966
- In FORTRAN 77 the character string handling features was introduced. IF-THEN-ELSE statement were introduced.
- Most significant changes took place in FORTRAN 90 by allowing use of modules, dynamic arrays, pointers, recursion,CASE statements, parameter type checking.

**Functional Programming : LISP :**

- It is a list processing language.
- This language is used in the research of AI.
- The syntax is based on lambda calculus.

**ALGOL 60 :**

- It supports the block structure.
- It has two parameter passing methods.
- It supports the subprogram recursion.

**COBOL :**

- The first macro facility in a high level language.
- It has a support for hierarchical data structures.
- It contains nested selection statements.
- Long names for variables up to 30 characters were allowed with hyphens.
- It has separate data division.

**BASIC :**

- It is easy to use and learn.
- The current popular version of BASIC is visual basic.

**PL/I :**

- It was designed by IBM and SHARE.
- Characterized by dynamic typing and dynamic storage allocation
- Variables are untyped i.e. A variable acquires a type when it is assigned a value
- Storage is allocated to a variable when it is assigned a value.

**PASCAL :**

- It is designed initially for teaching structured programming.
- It was small, simple and easy to learn.

**C :**

- It is developed in 1972 for system programming.
- It has powerful set of operators.
- Though designed as a systems language, it has been used in many application areas.

**PROLOG :**

- It is based on formal logic.
- It is a Non-procedural language.
- It is considered to be intelligent database system that uses an inferencing process to infer the truth of given queries.
- It is comparatively inefficient.

**ADA :**

- For design of this language required huge design effort, involving hundreds of people, much money, and about eight years.
- By allowing use of packages, ADA has a support for data abstraction.
- It supports the exception handling mechanism.
- It allows the use of generic units and concurrency mechanism.
- It contains more flexible libraries.

**C++ :**

- It is developed as Bell labs in 1980.
- It supports both procedural and object oriented features.

**JAVA :**

- Developed by Sun Microsystems in 1990.
- It is based on C++.
- It supports only OOP.
- It contains references but no pointers.
- It has a facility and support for applets and concurrency.
- It has libraries for applets, GUIs, database access.
- It is widely used for web programming.

**Scripting Languages for WEB :**

- Various scripting languages such as Perl, PHP, Python, JavaScript, Ruby and so on are used as a scripting languages for web programming.

**.NET Language : C# :**

- It is based on C++, Java, and Delphi.
- It includes pointers, delegates, properties, enumeration types, a limited kind of dynamic typing, and anonymous types.
- Is evolving rapidly.

**Markup/Programming Hybrid Languages :**

- XSLT: eXtensible Stylesheet Language Transformation (XSTL) transforms XML documents for display.
- Java Server Pages : a collection of technologies to support dynamic Web documents.

**1.3.2 Attributes of Good Language**

Various attributes of good languages are -

1. **Clarity, simplicity and unity** : The programming language should provide the **conceptual framework** for thinking about the algorithm. It should provide the clear, simple and unified set of concepts that can be **used as primitives** for developing the algorithms. The language should allow to use different concepts with the rules for their combinations with ease and simplicity. This attribute is called **conceptual integrity**.

The **syntax** of the language should be such that **readability** of the program can be increased.

2. **Orthogonality** : In context of programming languages, a programming construct is said to be **orthogonal** if these constructs can be **freely used** in combination with each other. The orthogonality features expects that the meaning is context independent. For instance - Pointer should be able to point to any type of variable or data structure.

This feature makes the language **easier to learn** and programs are easier to write.

3. **Naturalness for application** : The syntax of a programming language should be such that it should follow logical structure of the algorithm. Various algorithms such as sequential algorithms, concurrent algorithms, logic algorithms and so on have different natural structures which should be represented by the programming languages. The language should provide appropriate data structures, operations, and control structures for the problem to be solved.

4. **Abstraction** : Abstraction means **hiding** the implementation details. This feature allows the programmer to define and use the complicated structure in such a way that the implementation details can be ignored. By this attribute - Programmer can concentrate only on abstract properties without bothering for their implementation details. ADA and C++ are the languages that support for abstraction feature.

5. **Program verification** : The **reliability** of programs written in a language is always a major issue. There are various techniques for program verification. One of the techniques of program verification is to use **test input data**. In this technique, the

program is executed with **test input data** and the **output** is checked **against the specifications**.

If the **semantic and syntactic structure** is simple then the program verification becomes simplified.

6. **Program environment :** The programming environment includes well-documented implementations, special editors and testing packages, version control services and so on. The technical structure of programming language has a great influence on use of programming environment appropriately.
7. **Portability :** Portability is a feature by which the program that work on one platform can be modified to work on other platform.
8. **Cost of use :** The cost is major element in evaluation of any programming language. The cost can be computed for various factors such as -
  - a) **Cost of program execution :** Cost of program execution is particularly important for **large production programs** that will be **executed repeatedly**. The program execution cost is greatly reduced due to use of **optimizing compilers, efficient register allocation**, and design of **efficient runtime support mechanism**.
  - b) **Cost of program translation :** For compiling the large programs/systems, the compilers take too much time. This increases the overall cost. In such a case, it is important to have a fast and efficient compiler rather than a compiler that produces optimized executable code.
  - c) **Cost of program creation, testing and use :** The cost is involved in program designing, coding, testing and modifying. The Smalltalk and Perl are cost effective languages in this aspect.
  - d) **Cost of program maintenance :** The maintenance cost can be **four times more than** that of **development cost**. The cost of maintenance depend on various characteristics of the language mainly it is dependent upon readability.

### Review Questions

1. Explain the Evolution of Major Programming Languages.
2. What are the attributes of Good language ?

## 1.4 Programming Environments

**Definition :** Programming environment is an environment in which the programs can be created and tested.

Programming Environment consists of a collection of support tools that are used in software development. Each support tool is basically one kind of program which is used by the programmer during the stages of creation of program. These tools are editors, debuggers, verifiers, data generators and so on.

### 1.4.1 Effect on Language Design

The programming environment influences on language design in **two major areas** namely **separate compilation** and **testing and debugging**.

- 1. Separate Compilation
- 2. Testing / Debugging

#### Separate compilation :

There are varying number of programmers working on design, code and test the parts of programs. This require the language to be structured so that individual subprograms or other parts can be separately compiled and executed without requirement of other part. Later on these parts are to be merged into a final program.

Separate compilation is difficult because compiling one subprogram may need some **information** which is present in some other sub program or sometimes **shared data** is also used in the program due to which separate compilation is not possible. To solve this problem, some languages have adopted certain features that aid separate compilation. These features are -

1. The statements such as **extern** can be used to indicate that particular data is used from other subprogram.
2. Languages use **scoping rules** to hide names. One subprogram can be contained within another subprogram so that the name of the outer subprograms are known to separately compiled subprogram.
3. In object oriented programming, the feature like **inheritance** can be used to use the shared data.

#### Testing and Debugging :

Many languages contain some features that help in testing and debugging. For example -

1. **Breakpoint Feature** : By this feature, the programmer can set the **breakpoints in the program**. When the breakpoint is reached during the execution of the program, the execution of the program is interrupted and control is given to the programmer. The programmer can then inspect the program for testing and debugging.

2. **Execution Trace Feature :** By this feature particular variable or statement can be tagged for tracing during the execution of the program.
3. **Assertions :** The assertions is a **conditional expression** which is inserted as a separate statement in a program. For example following is a Java Code fragment in which the assertion is used

```
System.out.print("Enter your age: ");
int age = reader.nextInt();
Assertion.assert(age<18, "You are not allowed to vote");
```

### 1.4.2 Environment Framework

- **Environment framework** is nothing but the collection of infrastructure services that can be used for program development purpose. These services are data repository, Graphical user interface, security and communication services.
- **Borland JBuilder** is a programming environment that provides integrated compiler, editor and debugger for Java development.
- The modern programming environment makes use of **Visual Studio .NET** for developing the applications using C#, Visual Basic, .NET, Jscript, J# and so on.
- **Unix** is an old programming environment and has a strong support for various powerful tools for software production and maintenance. The UNIX GUI runs on the top of basic UNIX kernel. Typical examples of such GUI are GNOME and KDE.

### Review Questions

1. *What is programming environment ? Explain the effect of programming environment on language design.*
2. *Explain the concept of environment framework in programming environment.*

## 1.5 Impact of Machine Architectures

### 1.5.1 Operation of a Computer

Working of any computer is based on the software present in it. The software contains collection of programs. These programs are created using suitable programming languages.

A computer consists of **six major components** that are associated with the programming languages. These components are –

#### 1. Data

- In any programming language there are variety of data items and data structures.

- The built in data types can be directly manipulated by hardware primitive operations. Such built in data types are – integer, floating point numbers, double precision floating point numbers, and strings
- The data is stored and processed within data storage components of a computer. The three major data storage components are –
  - Main memory:** Main memory consists of bits subdivided into fixed length words.
  - High-speed Registers:** The contents of registers represent either data or the memory address of main memory containing data.
  - External Files:** The external files is generally stored on hard disk or on CD ROM. It contains records.

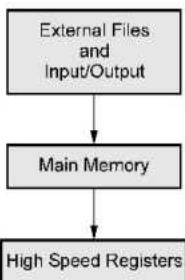


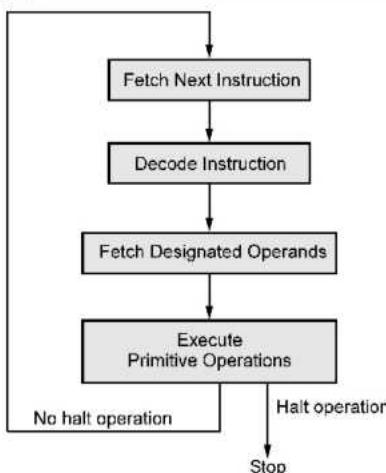
Fig. 1.5.1 Data storage components

## 2. Primitive Operations

- A computer contains a set of built-in primitive operations such as addition, multiplication, subtraction and division.
- The primitive operations are also used for controlling the sequence of execution of programming statements.

## 3. Sequence Control

- Computer must provide the mechanism for controlling the flow of execution of programming instruction. This mechanism is called as **sequence control**.
- Various primitive operations are used for sequence control. These operations transfer the control flow from one memory location to another.
- The next instruction to be executed is determined by the special purpose register called **program counter(PC)**.
- The mechanism of execution of programming instruction is illustrated by following figure.

**Fig. 1.5.2 Steps for Program Execution**

#### **4. Data Access**

Computer provides a controlling mechanism for supplying data for execution of programming instruction. This mechanism is called data access.

#### **5. Storage Management**

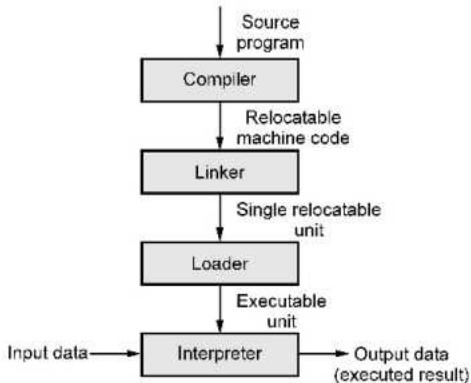
- Computer provides the mechanism for controlling the storage for programs and data resources.
- There are three types of storage components in computer and those are – (i) Main Memory (ii) High speed Registers and (iii) External Files.
- Sometimes the speed of CPU and main memory does not get synchronized. Hence a special memory called cache memory is used.
- A cache memory is high speed small memory situated in between the main memory and CPU. The data present in the cache memory is immediately available CPU.

#### **6. Operating Environment**

- Computer provides the mechanism by which external programs help in execution of application or user programs which constitutes the programming environment.
- The operating environment consists of peripheral storage and I/O devices.

### 1.5.2 Translators and Virtual Architectures

- The process of translation of a program from high level programming language to low level machine language is performed in following steps -
  - The program modules are separately translated into relocatable machine code. This translator is called as **compiler**.
  - These translated modules are linked together in a relocatable unit. This task is carried out by **linker** or **linkage editor**.
  - Finally the complete program is loaded into the memory as an executable machine code. This task is carried out by **loader**.



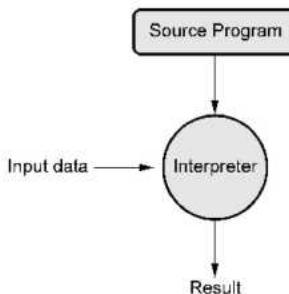
**Fig. 1.5.3 Language processing using translation**

The modern translation technique uses the combined two technique - translation and then interpretation. That means the source code is translated into **intermediate code** and then the intermediate code is interpreted.

### 1.5.3 Methods of Program Implementation

The three methods of language implementation are -

- Compilation** : This is a process in which the high level programs are translated into machine language.
- Pure interpretation** : In this process, programs are interpreted line by line to target program.

**Advantages :**

1. Modification of user program can be easily made and implemented as execution proceeds.
2. Type of object that denotes a variable may change dynamically.
3. Debugging a program and finding errors is simplified task for a program used for interpretation.
4. The interpreter for the language makes it machine independent.

**Disadvantages :**

1. The execution of the program is slower.
2. Memory consumption is more.

**3) Hybrid implementation system :** In this system the programs are translated using both interpreter and compiler.

- Using compilers the high level programs are translated into intermediate language. These intermediate languages are easy to interpret. The interpreter interprets the intermediate language to form the machine code.
- Examples - Perl is implemented using hybrid implementation system. Initially implementation of Java were all hybrid. The intermediate form of Java code is called as **byte code**. This byte code provides the facility of execution of the code on any platform. Thus portability to any machine(Platform independence) can be achieved using byte code.
- The Just-In-Time (JIT) compiler is a component of the Java Runtime Environment that improves the performance of Java applications at run time. Java programs consists of classes, which contain platform neutral bytecode that can be interpreted by a Java Virtual Machine(JVM) on many different computer architectures. At run time, the JVM loads the class files, determines the semantics of each individual bytecode, and performs the appropriate computation.

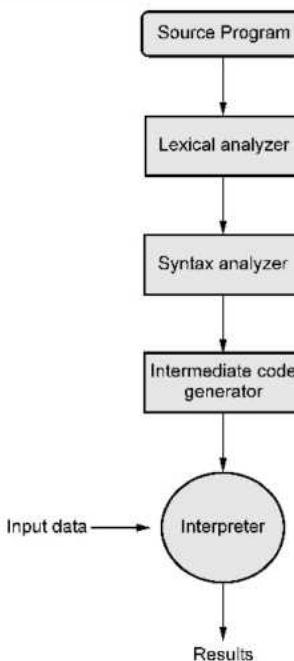


Fig. 1.5.4 Hybrid implementation system

**Difference between Compiler and Interpreter**

Sr. No.	Interpreter	Compiler
1.	<b>Demerit :</b> The source program gets interpreted every time it is to be executed, and every time the source program is analyzed. Hence interpretation is less efficient than Compiler.	<b>Merit :</b> In the process of compilation the program is analyzed only once and then the code is generated. Hence compiler is efficient than interpreter.
2.	The interpreters do not produce object code.	The compilers produce object code.
3.	<b>Merit :</b> The interpreters can be made portable because they do not produce object code.	<b>Demerit :</b> The compilers has to be present on the host machine when particular program needs to be compiled.
4.	<b>Merit :</b> Interpreters are simpler and give us improved debugging environment.	<b>Demerit :</b> The compiler is a complex program and it requires large amount of memory.

### Concept of Preprocessor

- **Definition :** Preprocessor is a program that processes a program just before the program is compiled.
- Preprocessor instructions are embedded in the program.
- Preprocessor macros (instructions) are commonly used to specify that code from another file is to be included.
- For example : The C preprocessor expands #include or #define in the following manner -
  - `#include "mytestfile.h"`  
causes the preprocessor to copy the contents of mytestfile.h into the program at the position of the #include.
  - `#define SIZE 5`  
In the program we can declare an array of size 5 in following manner -  
`int a[SIZE];`

### 1.5.4 Virtual Computers

**Definition of Virtual Machine :** Virtual machine is a piece of software which simulates the actual machine environment. An implementation of programming language requires that the programs in the language are analysed and then translated into a form that can be run by interpreter i.e. on virtual machine first and then on actual real machine.

- For a virtual machine environment the source program is translated into an intermediary abstract form which is then interpretively executed.
- In this case, the source program first goes through three stages such as lexical analyser, syntax analyser and type checker. Then the abstract syntax obtained from these three phases is interpreted to get the translated output.
- Due to this feature it is possible to execute the abstract code on any desired platform.
- For example – Java virtual machine is a kind of virtual machine which plays an important role in achieving the platform independence.
- The general structure of virtual machine is as shown in Fig. 1.5.5.

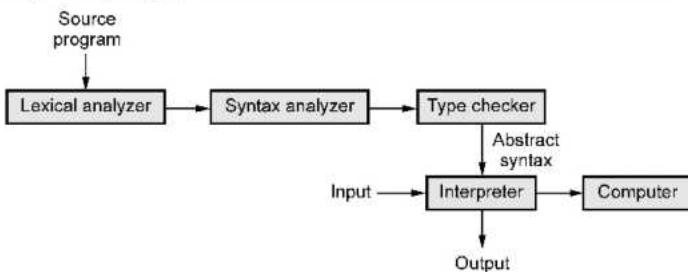


Fig. 1.5.5 Structure of virtual machine

### 1.5.5 Binding and Binding Times

- Any program contain various entities such as variables, routines, control statements and so on. These entities have special properties. These properties are called **attributes**. For example - the programming entity **routine** or **function** has number of parameters, type of parameters, parameter passing methods and so on. Specifying the nature of attributes is called **binding**.
- The time at which the choice for binding occurs is called **binding times**.
- There are various classes of Binding types and these are represented by following figure –

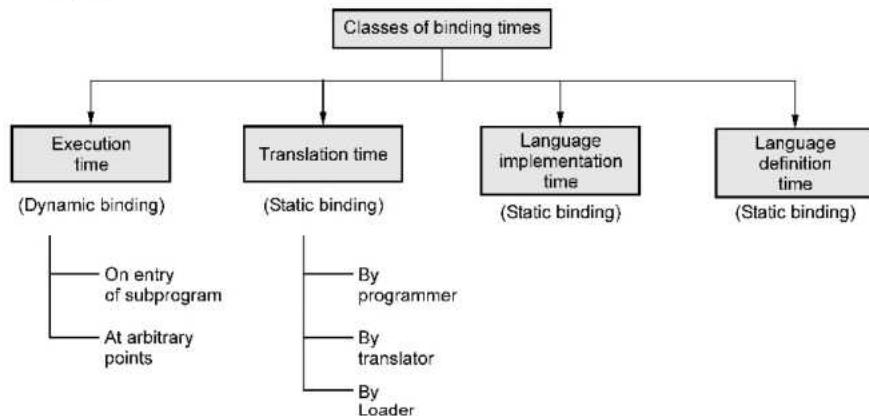


Fig. 1.5.6 Classes of binding times

#### 1. Execution Time

- There are many bindings that are performed during execution of the program. These type of binding are also called as **dynamic binding** or **Run time binding**.

- At entry to a block or subprogram:** At the entry of subprogram the binding takes place between actual to formal parameters. There can be binding of **variables with their values**.
- At arbitrary points during Program execution:** At arbitrary points during program execution the variables are assigned with values.

## 2. Translation Time

- Chosen By Programmer:** In this binding, declaration of bind types to variable name occurs. Similarly the binding between values and constants occur.
- Chosen by Loader:** Some bindings are chosen by loader. For example – global variables to location source program to object program representation.

## 3. Language Implementation Time

- In this type of binding, bind the values to the representation in computer.
- Also there is binding of operations and statements to semantics of operations.

## 4. Language Definition Time:

- Most of the structures of programming languages is fixed at the time the language is defined. Hence binding occurs as per the structure of the program that has been defined at the time of its definition. For example –  
“DO” is a reserved word in PASCAL but not in FORTRAN.

### Early and Late Binding

Early Binding	Late Binding
It happens at compile time.	It happens at run time.
The method definition and the method call are linked during the compile time. This happens when all information needed to call a method is available at the compile time.	The method definition and a method call are not bound until run time.
It is more efficient	It is more flexible.

### Review Questions

- What is the difference between compiler and interpreter ?
- Explain the concept of virtual computer.
- Explain various classes of binding times.
- What is the difference between early binding and late binding ?

**1.6 Programming Paradigms****SPPU : Dec.-17, Marks 6**

**Definition :** Programming paradigm can be defined as a method of problem solving and an approach to programming language design.

Various types of programming paradigm are -

1. Imperative or procedural programming
2. Object oriented programming
3. Functional programming
4. Logic programming

Examples of various programming languages based on the above mentioned paradigm is as shown by following table

Imperative programming	Object oriented	Functional programming	Logic programming
ALGOL	Smalltalk	LISP	Prolog
COBOL	Simula	Haskell	
ADA	C++	APL	
C	Java		
PASCAL			
FORTRAN			

Let us discuss them in brief.

**1.6.1 Imperative Programming**

The Latin word impetrare means to command. Hence this language is command driven or statement oriented language.

The imperative programming is also called as **procedural programming language**.

A program consists of sequence of statements. After execution of each statement the values are stored in the memory.

The central features of this language are variables, assignment statements, and iterations.

Examples of imperative programming are - C, Pascal, Ada, Fortran and so on.

**Merits :**

1. **Simple** to implement.
2. These languages have **low memory utilization**.

**Demerits :**

1. Large complex problems can not be implemented using this category of language.
2. Parallel programming is not possible in this language.
3. This language is **less productive** and at low level compared to other programming languages.

**1.6.2 Object Oriented Programming**

In this language everything is modeled as **object**. Hence is the name.

This language has a modular programming approach in which data and functions are bound in one entity called class.

This programming paradigm has gained a great popularity in recent years because of its characteristic features such as data abstraction, encapsulation, inheritance, polymorphism and so on.

Examples of object oriented programming languages are - Smalltalk, C++, Java,

**Merits :**

1. It provides a modular programming approach.
2. It provides abstract data type in which some of the implementation details can be hidden and protected.
3. Modifying the code for maintenance become easy, due to modules in the program. Modification in one module can not disturb the rest of the code.
4. Finding bugs become easy.
5. Object oriented programming provides good framework for code library from which the software components can be easily used and modified by the programmer.

**Demerits :**

1. Sometimes real world objects can not be realized easily. Hence it is **complex to implement**.
2. Some of the members(data or methods) of the class are **hidden**. Hence they can not be accessed by the objects of other class.
3. In object oriented programming everything is arranged in the form of classes and modules. For the **lower level** applications it is not desirable feature.

### 1.6.3 Functional Programming

In this paradigm the computations are expressed in the form of functions.

Functional programming paradigms treat values as single entities. Unlike variables, values are never modified. Instead, values are **transformed into new values**.

Computations of functional languages are performed largely through applying functions to values, i.e., (+ 10 20). This expression is interpreted as  $10 + 20$ . The result 30 will be returned.

For building more complex functions the previously developed functions are used. Thus program development proceeds by developing simple function development to complex function development.

Examples of function programming are LISP, Haskell, ML

**Merits :**

1. Due to use of functions the programs are **easy to understand**.
2. The functions are **reusable**.
3. It is possible to develop and **Maintain large programs** consisting of large number of functions.

**Demerits :**

1. Functional programming is **less efficient** than the other languages.
2. They consume **large amount of time and memory for execution**.
3. Purely functional programming is not a good option for commercial software development.

### 1.6.4 Logic Programming

In this paradigm we express computation in terms of mathematical logic only. It is also called as rule based programming approach.

Rules are specified in no special order.

It supports for declarative programming approach. In this approach how the computations take place is explained.

The logic paradigm focuses on predicate logic, in which the basic concept is a relation.

Example of Logic programming is Prolog.

**Merits :**

1. This programming paradigm is **reliable**.

2. The program can be **quickly developed** using this approach as it **makes use of true/false statements** rather than objects.
3. It is best suitable for the problems in which **knowledge base** can be established.

**Demerits :**

1. **Execution** of the program is **very slow**.
2. True/false statements **can not solve most of the problems**.
3. It can **solve only limited set of problems** efficiently.

**Review Questions**

1. Explain the programming paradigm in detail.
2. List the programming paradigms. For any three state which programming languages are based on them and how ?

SPPU : Dec.-17, Marks 6

**1.7 Multiple Choice Questions****Q.1 What is the relationship between reliability and failure ?**

- |  |  |
|--|--|
| <input type="checkbox"/> a Direct relation | <input type="checkbox"/> b Inverse relation                              |
| <input type="checkbox"/> c No relation     | <input type="checkbox"/> d Varying relation depending upon the situation |

**Explanation :** As reliability increases, the failure decreases. Hence it is an inverse relation.

**Q.2 The language qualities that support reliability are \_\_\_\_\_.**

- |  |   |
|--|---|
| <input type="checkbox"/> a writability | <input type="checkbox"/> b readability  |
| <input type="checkbox"/> c simplicity  | <input type="checkbox"/> d all of these |

**Q.3 Language efficiency can be achieved by \_\_\_\_\_.**

- |  |  |
|--|--|
| <input type="checkbox"/> a maximum speed | <input type="checkbox"/> b less memory consumption |
| <input type="checkbox"/> c reusability   | <input type="checkbox"/> d all of these            |

**Q.4 The feature by which the program that work on one platform can be modified on other platform is called as \_\_\_\_\_.**

- |  |  |
|--|--|
| <input type="checkbox"/> a reusability | <input type="checkbox"/> b portability |
| <input type="checkbox"/> c robustness  | <input type="checkbox"/> d locality    |

**Q.5 For specifying the syntax of a language \_\_\_\_\_ is used.**

- |  |   |
|--|---|
| <input type="checkbox"/> a context free language | <input type="checkbox"/> b regular expression |
| <input type="checkbox"/> c finite automata       | <input type="checkbox"/> d none of these      |

**Explanation :** The context free grammar consists of set of production rules using which it is possible to specify the syntactic rules of a language.

**Q.6 Syntax of a program means -**

- |  |   |
|--|---|
| <input type="checkbox"/> a Format of a program         | <input type="checkbox"/> b Meaning of a program |
| <input type="checkbox"/> c Simply content of a program | <input type="checkbox"/> d None of these        |

**Q.7 Semantic of a program means -**

- |  |   |
|--|---|
| <input type="checkbox"/> a Format of a program         | <input type="checkbox"/> b Meaning of a program |
| <input type="checkbox"/> c Simply content of a program | <input type="checkbox"/> d None of these        |

**Q.8 A program that interprets each line of high level program at the time of execution is called \_\_\_\_\_.**

- |                                       |  |
|---------------------------------------|--|
| <input type="checkbox"/> a translator | <input type="checkbox"/> b interpreter |
| <input type="checkbox"/> c instructor | <input type="checkbox"/> d executor    |

**Explanation :** Interpretation is a process in which every source language instruction is directly executed.

**Q.9 A program which translates high level programming language into machine code is called \_\_\_\_\_.**

- |                                       |  |
|---------------------------------------|--|
| <input type="checkbox"/> a translator | <input type="checkbox"/> b compiler      |
| <input type="checkbox"/> c assembler  | <input type="checkbox"/> d none of these |

**Q.10 Variable names resolving is based on :**

- |   |  |
|---|--|
| <input type="checkbox"/> a Source language                    | <input type="checkbox"/> b Assembler and loader implementation |
| <input type="checkbox"/> c Compiler and linker implementation |  |
| <input type="checkbox"/> d None of these                      |  |

**Q.11 The set of rules used in context free grammar are called as \_\_\_\_\_.**

- |   |  |
|---|--|
| <input type="checkbox"/> a terminals        | <input type="checkbox"/> b non terminals |
| <input type="checkbox"/> c production rules | <input type="checkbox"/> d start symbol  |

**Q.12 Pascal inventor Niklaus Wirth designed the language specially for :**

- |   |  |
|---|--|
| <input type="checkbox"/> a Use on any type of operating system          |  |
| <input type="checkbox"/> b Helping to promote use of personal computers |  |
| <input type="checkbox"/> c Teaching programming to students             |  |
| <input type="checkbox"/> d High speed computing                         |  |

**Q.13 Which was the first language for scientific applications ?**

- |                                     |                                    |
|-------------------------------------|------------------------------------|
| <input type="checkbox"/> a ALGOL 60 | <input type="checkbox"/> b FORTRAN |
| <input type="checkbox"/> c LISP     | <input type="checkbox"/> d COBOL   |

**Q.14 Which was the first high level language developed for business purpose ?**

- |                                     |                                    |
|-------------------------------------|------------------------------------|
| <input type="checkbox"/> a ALGOL 60 | <input type="checkbox"/> b LISP    |
| <input type="checkbox"/> c COBOL    | <input type="checkbox"/> d FORTRAN |

**Q.15 Which languages necessarily need heap allocation in the runtime environment ?**

- a Those that support recursion
- b Those that use dynamic scoping
- c Those that allow dynamic data structures
- d Those that use global variables

**Q.16 What is the name of the category of programming languages whose structure is dictated by the von Neuman computer architecture ?**

- |                                       |   |
|---------------------------------------|---|
| <input type="checkbox"/> a Imperative | <input type="checkbox"/> b Functional   |
| <input type="checkbox"/> c Constraint | <input type="checkbox"/> d Denotational |

**Q.17 Which one of the following is NOT performed during compilation ?**

- |  |   |
|--|---|
| <input type="checkbox"/> a Dynamic memory allocation | <input type="checkbox"/> b Type checking    |
| <input type="checkbox"/> c Symbol table management   | <input type="checkbox"/> d Inline expansion |

**Q.18 \_\_\_\_\_ is a piece of software which simulates the actual machine environment.**

- |  |  |
|--|--|
| <input type="checkbox"/> a Compiler        | <input type="checkbox"/> b Debugger    |
| <input type="checkbox"/> c Virtual machine | <input type="checkbox"/> d Interpreter |

**Q.19 The classe(s) of binding times is/are \_\_\_\_\_.**

- |   |   |
|---|---|
| <input type="checkbox"/> a Execution time | <input type="checkbox"/> b Translation time |
| <input type="checkbox"/> c both a and b   | <input type="checkbox"/> d None of these    |

**Q.20 Execution time is actually \_\_\_\_\_.**

- |  |   |
|--|---|
| <input type="checkbox"/> a dynamic binding | <input type="checkbox"/> b static binding |
| <input type="checkbox"/> c None of these   |   |

**Q.21 \_\_\_\_\_ is defined by Programmer, Loader or translator**

- |   |   |
|---|---|
| <input type="checkbox"/> a Execution time               | <input type="checkbox"/> b Translation time         |
| <input type="checkbox"/> c Language implementation time | <input type="checkbox"/> d Language definition time |

**Q.22 Static binding is \_\_\_\_.**

- |   |   |
|---|---|
| <input type="checkbox"/> a Translation time         | <input type="checkbox"/> b Language implementation time |
| <input type="checkbox"/> c Language definition time | <input type="checkbox"/> d All of the above             |

**Q.23 \_\_\_\_\_ is a program that processes a program just before the program is compiled.**

- |   |  |
|---|--|
| <input type="checkbox"/> a Preprocessor | <input type="checkbox"/> b Interpreter |
| <input type="checkbox"/> c Compiler     | <input type="checkbox"/> d Assembler   |

**Answer Keys for Multiple Choice Questions :**

Q.1	b	Q.2	d	Q.3	d	Q.4	b
Q.5	a	Q.6	a	Q.7	b	Q.8	b
Q.9	b	Q.10	c	Q.11	c	Q.12	c
Q.13	b	Q.14	c	Q.15	c	Q.16	a
Q.17	a	Q.18	c	Q.19	c	Q.20	a
Q.21	b	Q.22	d	Q.23	a		



## Notes

## **UNIT - II**

# **2**

# **Structuring the Data, Computations and Program**

### **Syllabus**

**Elementary Data Types :** Primitive data Types, Character String types, User Defined Ordinal Types, Array types, Associative Arrays, Record Types, Union Types, Pointer and reference Type.

**Expression and Assignment Statements :** Arithmetic expression, Overloaded Operators, Type conversions, Relational and Boolean Expressions, Short Circuit Evaluation, Assignment Statements, Mixed mode Assignment. **Statement level Control Statements :** Selection Statements, Iterative Statements, Unconditional Branching. **Subprograms :** Fundamentals of Sub Programs, Design Issues for Subprograms, Local referencing Environments, Parameter passing methods.

**Abstract Data Types and Encapsulation Construct :** Design issues for Abstraction, Parameterized Abstract Data types, Encapsulation Constructs, Naming Encapsulations.

### **Contents**

2.1 Elementary Data Types .....	<b>May-19,</b>	Marks 6
2.2 Expression and Assignment Statements		
2.3 Statement-level Control Statements		
2.4 Subprograms		
2.5 Abstract Data Types and Encapsulation Construct.....	<b>Dec.-17,</b>	Marks 7
2.6 Multiple Choice Questions		

## 2.1 Elementary Data Types

SPPU : May-19, Marks 6

### 2.1.1 Primitive Data Types

- The primitive data types are elementary data types which are not built from some other types.
- The values of primitive data types are **atomic** and cannot be decomposed into simpler constituents.
- Many times the built in data types can be interchangeably used with built in data type but there are exceptions. **For example** - enum in C is a primitive data type which is used to define new constants.
- Any programming language has a predefined set of built in data types. This built in data type reflects the behavior of underlying hardware.
- Due to built in data type, the values are interpreted differently using hardware instructions.
- Following are the **types of built in** types of programming languages.

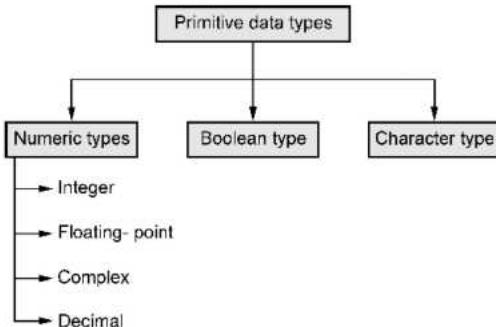


Fig. 2.1.1 Data types

#### 1) Numeric Types :

- Integer :**
  - This is the most commonly used data type.
  - The integer data types can be signed or unsigned data types.
  - In Java there are singed integer data types such as byte, short, int and long.
  - In C++ unsigned integer data types are simply the integer values without signs. The unsigned integer data types are typically used for binary data.

- **Floating Point :**

- The floating point data type is used for representing the real numbers.
- On most of the computers the floating point numbers are stored in binary number format.
- Floating point values are represented as fractions and exponents.
- In most of the languages, there are two types of floating point values - **float** and **double**.

- **Complex :**

- Complex values are represented as pair of real and imaginary part.
- The imaginary part of complex number is represented as literal such as i or j.
- For example  $5 + 7i$  is a complex number.
- Python is a programming language that supports the complex data type.

- **Decimal**

- The decimal data type is used to support business systems applications.
- This is a type of data type that store fixed number of decimal digits with decimal point at a fixed position in the value.
- The languages like COBOL and C# makes use of decimal data type.
- The decimal data type values can be stored using a special representation called **Binary Coded Decimal(BCD)**

**2) Boolean Types :**

- This data type define only two values TRUE and FALSE.
- In Boolean algebra the Boolean data type is used.
- The basic operations performed are and, not , or.

**3) Character Type :**

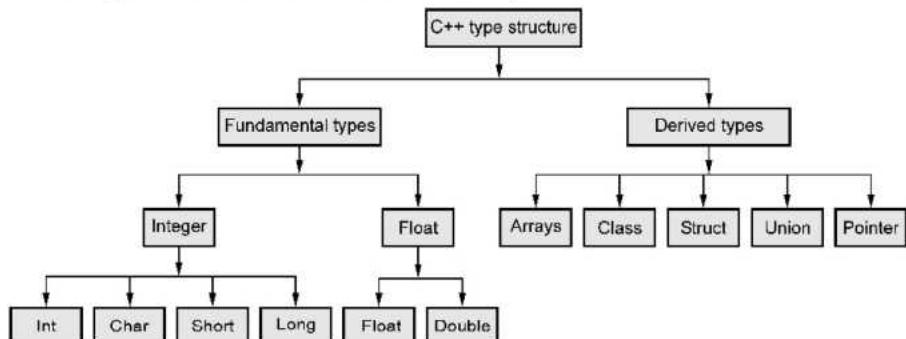
- The single character value is represented using char data type.
- In C the character type variable is

char choice;

**Type Structure of C++**

- The type structure of C++ is divided into two main categories - Fundamental types and derived types.
- The fundamental type is - integer and float. The integer data type is classified into int, char, short, long. The float data type is classified into float and double.

- The derived type of actually derived from the fundamental types - For example, arrays, class, structure, union, functions and pointers.



**Fig. 2.1.2 Type structure of C++**

- Arrays is collection of similar data type elements. For example

```
int a[10]; //It will create an array of 10 integer elements
```

- The class, structure and union allows the programmer to create user defined data types. The class is well known entity in C++ by which one can encapsulate data members and member functions in C++. For example

```
class stack
{
private:
    int s[10];
    int top;
public:
    void push(int item)
    int pop();
}
```

The above class creates a **stack** data structure using class.

The pointer is different type of data type which stores the address of the variable. Just similar to pointer there is a concept of **reference** which holds the address of a variable. Reference variable is initialized by another variable.

### Type Structure of Java

The type structure of Java is classified in two categories

- Built in Data Types
- Derived Types.

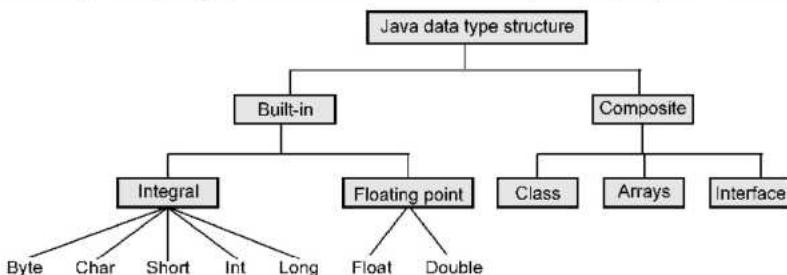


Fig. 2.1.3 Types structure of Java

**The built in data types :**

Various data types used in Java are byte, short, int, long, char, float, double and boolean.

**byte**

This is in fact smallest integer type of data type. Its width is of 8-bits with the range -128 to 127. The variable can be declared as byte type as

```
byte i,j;
```

**short**

This data type is also used for defining the signed numerical variables with a width of 16 - bits and having a range from -32,768 to 32,767. The variable can be declared as short as

```
short a,b;
```

**int**

This is the most commonly used data type for defining the numerical data. The width of this data type is 32-bit having a range 2,147,483,648 to 2,147,483,647. The declaration can be

```
int p,q;
```

**long**

Sometimes when *int* is not sufficient for declaring some data then *long* is used. The range of *long* is really very long and it is -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. The declaration can be

```
long x,y;
```

**float**

To represent the real number(i.e. number that may have decimal point) float data type can be used. The width is 32-bit and range of this data type is  $3.40282347 \times 10^{38}$ ,  $1.40239846 \times 10^{-45}$ .

**double**

To represent the real numbers of large range the double data type is used. Its width is 64-bit having the range  $1.8 \times 10^{308}$ ,  $4.9 \times 10^{-324}$

**char**

This data type is used to represent the character type of data. The width of this data type is 16-bit and its range is 0 to 65,536.

**boolean**

Boolean is a simple data type which denotes a value to be either true or false.

## 2.1.2 Character String Types

A character string type is one in which values are **sequences of characters**.

### 2.1.2.1 Design Issues

The two most important **design issues** for character string types are -

#### 1) Is it a primitive type or just a special kind of character array ?

- C and C++ use char arrays to store char strings. These char strings are terminated by a special character **null**.
- In Java strings are supported by **String** class whose value are constant string, and the **String Buffer** class whose value are changeable and are more like arrays of single characters.
- C# and Ruby include string classes that are similar to those of Java.
- Python strings are immutable, similar to the String class objects of Java.

#### 2) Is the length of objects static or dynamic ?

- The length can be static and set when the string is created. For example - in C, C++, Java we can have static length string.
- When the string length is dynamic then it has no maximum limit. For example - Perl and JavaScript provide this kind of facility.

### 2.1.2.2 String Operation

Various types of string operations include

- 1) **Finding Length of string** : Finding length means counting number of characters within a string.
- 2) **Comparing two string** : In this operation, the match between characters from each string is evaluated.
- 3) **Finding substring** : In this operation, the substring of the original string returns.
- 4) **String concatenation** : Concatenation is an operation in which second string is appended to the first string.

### 2.1.2.3 String Length Options

- The string length can be static or can be dynamic. There are three kind of design choices for string length -

- |           |                    |            |
|-----------|--------------------|------------|
| 1. Static | 2. Limited Dynamic | 3. Dynamic |
|-----------|--------------------|------------|

- 1) **Static String Length** : The length can be static and set when the string is created. For example - in C,C++, Java we can have static length string. This is a choice for immutable objects.
- 2) **Limited Dynamic String Length** : This option allows strings to have varying length up to a declared and fixed maximum set by variable's definition.
- 3) **Dynamic String Length** : When the string length is dynamic then it has no maximum limit. For example
  - Perl and JavaScript provide this kind of facility.

### 2.1.2.4 Implementation of Character String Type

For each of the following type of strings the implementation methods vary -

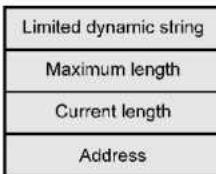
- **Static Length Strings** : A descriptor for a static character string type, which is required only during compilation, has three fields.
  - Name of the type
  - Type's length
  - Address of first char

It is as shown in following Fig. 2.1.4 -



**Fig. 2.1.4 Compile time descriptor**

- **Limited Dynamic Length Strings :** It needs a run-time descriptor for length to store both the fixed maximum length and the current length. The limited dynamic strings of C and C++ do not require run-time descriptors, because the end of a string is marked with the null character. They do not need the maximum length, because index values in array references are not range-checked in these languages. It is as shown by following Fig. 2.1.5 -



**Fig. 2.1.5 Run time descriptor**

- **Dynamic Length strings :** Dynamic length strings require more complex storage management. The length of a string is not fixed hence storage may grow or shrink dynamically.

### 2.1.3 User Defined Ordinal Types

These are the data types that are defined by the programmer with the help of primitive data types. For example - In C structure is an user defined data type created using the keyword **struct**

```

struct student {
    int roll;
    char name[10];
}
  
```

An ordinal type is one in which the range of possible values can be easily associated with the set of positive integers. There are two user-defined ordinal types -

- |                     |             |
|---------------------|-------------|
| 1) Enumeration Type | 2) Subrange |
|---------------------|-------------|

### 1) Enumeration Type

Enumeration types provide a way of defining and grouping collections of named constants, which are called **enumeration constants**.

For example in C# we can define the enum data type as follows -

```
enum days {mon, tue, wed, thu, fri, sat, sun};
```

The enumeration constants are typically implicitly assigned the integer values, 0, 1, ..., but can explicitly assign any integer literal in the type's definition. That means, in following example we can explicitly assign the enumeration constants

```
enum suit {  
    club = 0,  
    diamonds = 10,  
    hearts = 20,  
    spades = 3,  
};
```

### Design Issues for Enumeration Types

Following are design issues for enumeration types -

- Is an enumeration constant allowed to appear in more than one type definition ?
- If enumeration constants appear in more than one type definition, then how the type of an occurrence of that constant is checked ?
- Are enumeration values coerced to integer ?
- Any other type coerced to an enumeration type ?

### 2. Subrange

- Subrange data types are used in Pascal and ADA. It is basically a continuous sequence of an ordinal types. For defining a subset of values the subrange
  - For example -

```
type Color is(RED,ORANGE,YELLOW,GREEN,BLUE,INDIGO,VIOLET)  
subtype myfavcolor is Color range ORANGE..BLUE;
```

#### 2.1.4 Array Types

- Array is collection **similar data type elements**.
- It is **homogeneous** aggregate of data elements in which an individual element is identified by its position in the aggregate, relative to the first element.
- Individual array element is specified using **subscription expression**.
- **For example** - In C, C++ array can be declared as

```
int a[10];
```

#### 2.1.4.1 Design Issues

The primary design issues specific to arrays are the following :

- 1) What types are legal for subscripts ?
- 2) Are subscripting expressions in element references range checked ?
- 3) When are subscript ranges bound ?
- 4) When does allocation take place ?
- 5) Are ragged or rectangular multidimensional arrays allowed, or both ?
- 6) Can arrays be initialized when they have their storage allocated ?
- 7) What kinds of slices are allowed, if any ?

#### 2.1.4.2 Subscript Binding for Arrays

- The binding of subscript type to an array variable is usually **static**, but the subscript value ranges are sometimes **dynamically bound**.
- In C, C++ the lower bound of all index ranges is fixed at 0; In Fortran 95, it defaults to 1.

#### 2.1.4.3 Array Categories based on Subscript Binding

There are basically four categories -

- 1) **Static Array** : It is one in which the subscript ranges are statically bound and storage allocation is static that means it is done before run time. Advantage of this type of arrays is - Its working is efficient as there is no need to allocate and deallocate memory.
- 2) **Fixed Stack Dynamic Array** : A fixed stack-dynamic array is one in which the subscript ranges are statically bound, but the allocation is done at elaboration time during execution. The advantage of this type is - it is space efficient. A large array in one subprogram can use the same space as a large array in different subprograms.
- 3) **Fixed Heap Dynamic Array** : It is similar to fixed stack-dynamic. The storage binding is dynamic but fixed after allocation (i.e., binding is done when requested and storage is allocated from heap, not stack).
- 4) **Heap Dynamic Array** : In this type, binding of subscript ranges and storage allocation is dynamic and can change any number of times. The advantage of this type of array is - it is very flexible.

#### 2.1.4.4 Heterogeneous Array

- Heterogeneous array is a type of array in which the **elements need not be of the same data type**.
- Heterogeneous arrays are supported by the programming languages such as Perl, JavaScript, Python and Ruby.
- In Perl, the heterogeneous array consists of numbers, strings and references.
- In Python and Ruby array elements are references to the objects of any type.

#### 2.1.4.5 Array Initialization

- Array initialization is a process in which a list of values that are put in the array in the order in which the array elements are stored in memory.
- Some language allow initialization at the time of storage allocation.
- For example - Following is an array initialization in C
  - int a[] = {10,20,30};
- Character Strings in C & C++ are implemented as arrays of char. These arrays can be initialized to string constants -

```
char name[] = "ISRO"  
//Note that this array will contain 5 elements and this string is terminated by null
```

- In Java the array of string can be initialized as follows -

```
String[] names = {"Aditya", "Aakash", "Amit"};
```

#### 2.1.4.6 Array Operations

- The common array operations are - **assignment, catenation, comparison** for equality and inequality, and **slices**.
- The C-based languages do not provide any array operations, except through the methods of Java, C++, and C#.
- Perl supports array assignments but does not support comparisons.
- Python's arrays' are called **lists**, they behave like **dynamic arrays**. In this lists the objects can be of any datatype. Hence arrays are heterogeneous. Python also supports array catenation and element membership operations.
- In Python, assignment with an = on lists does not make a copy. Instead, assignment makes the two variables point to the one list in memory. For example
  - color=[10,20,30]
  - a=color;

The list will point to the same memory location as that of color.

- Ruby also provides array catenation.
- APL provides the most powerful array processing operations for vectors and matrixes as well as unary operators.

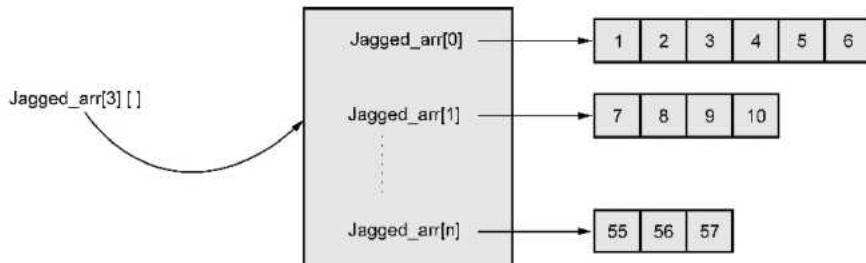
### 2.1.4.7 Rectangular and Jagged Arrays

**Rectangular Array :** A rectangular array is a multidimensional array in which all of the rows have the same number of elements, all of the columns have the same number of elements and so on.

For example -

rect_array[3][3]		
11	12	13
14	15	16
17	18	19

**Jagged Array:** A jagged array is one in which the length of the rows need not be the same. For example -



It can be represented as

```

int arr[][] = new int[][] {
    new int[] {1,2,3,4,5,6};
    new int[] {7,8,9,10};
    new int[] {11,12};
};
  
```

### 2.1.4.8 Slices

Slice of an array is nothing but the substructure of that array. The **: Operator** used within the square bracket to indicate that it is a list slice and not the index of the list.

For example in Python -

```

>>> a=[10,20,30,40,50,60]
>>> a[1:4]
[20, 30, 40]
>>> a[:5]
[10, 20, 30, 40, 50]
  
```

```
>>> a[4:]  
[50, 60]  
>>> a[:]  
[10, 20, 30, 40, 50, 60]
```

### Review Questions

1. Mention the primary design issues specific to arrays.
2. Explain the subscript binding and array categories.
3. Describe the process of array initialization.
4. Define and explain rectangular and jagged array.

### 2.1.5 Associative Arrays

- An associative array is an unordered collection of data elements that are indexed by an equal number of values called keys.
- Each element of an associative array is a pair of **a key** and **a value**.
- Associative arrays are supported by the standard class libraries of Java, C++, C#, and F#.
- For Example : In Perl, associative arrays are often called **hashes**. Names begin with %; literals are delimited by parentheses. Hashes can be set to literal values with assignment statement as shown below -

```
%ages = ("AAA" => 25, "BBB" => 10, "CCC" => 55);
```

- Subscripting is done using braces and keys. For example -

```
$ages{"CCC"}=48;
```

- In Python, the associative arrays are called as dictionaries. For example -

```
thisdict = {  
    "brand": "Honda",  
    "model": "Honda-Civic",  
    "year": 2019  
}
```

- PHP's arrays are both normal arrays and associative array.
- C# and F# support associative arrays through a .NET class

### Review Question

1. Explain the concept of associative arrays with example.

### 2.1.6 Record Types

- A data structure which is collection of components of different data types is termed as record.
- **Specification :** The attributes of record are
  1. The **number** of components
  2. The **data type** of each component
  3. The **selector** used to name each component.
- In C the syntax for record declaration is denoted by **struct**.
- **For example -** In C, C++, and C#, records are supported with the struct data type

```
struct student{  
    int roll_no;  
    char name[20];  
}
```

#### Access to Record Fields

Most of the programming languages make use of dot operator to access or refer to different fields of the record. For example -

```
struct student.roll_no
```

### 2.1.7 Union Types

- A union is a type whose variables are allowed to store different type values at different times during execution.
- **For example -** In C we can declare union as follows

```
union Student  
{  
    char name[30];  
    int rollno;  
}s1,s2;
```

Here s1 and s2 are the variables of union.

- We can access the elements of union using dot operator. For instance **s1.rollno**
- Note that only one member of union is accessed at a time. For example if –

```
s1.name="AAA";  
s1.rollno=10;
```

Then as an output we get either **s1.rollno = 10** or **s1.name = "AAA"**;

### 2.1.7.1 Design Issues

The primary design union types are as follows -

- 1) Should unions be embedded in records ?
- 2) Should type checking is needed ? If so, then such type checking must be dynamic.

### 2.1.7.2 Discriminated and Free Unions

Two types of unions are :

- 1) Free Union :** The union constructs in which there is no language support for type - checking is called free union. C, C++ provide such type of unions. For example -

```
union test {
    int a;
    float b;
}
union test u;
float x;
...
u.a=10;
x=u.b;
```

Note that the integer value is assigned to the float variable in the last assignment and there is no type checking mechanism.

- 2) Discriminated Union :** Discriminated union is a union structure in which each union include the type indicator. The discriminated unions are supported by ALGOL 68, ML, HASKELL.

For example – the shape base class posses three derived classes namely Circle, Triangle and Square classes.

```
type Shape =
    // The value here is the radius.
    | Circle of float
    // The value here is the side length.
    | Equilateral Triangle of double
    // The value here is the side length.
    | Square of double
    // The values here are the height and width.
    | Rectangle of double * double
```

### 2.1.7.3 ADA Union Types

The syntax is as follows -

```
type Typ (Choice : Discrete_Type) is record
    case Choice is
```

```
when Choice_1 =>
  N1 : Typ1;
  ...
when Choice_2 | Choice_3 =>
  ...
when others =>
  ...
end case;
end record;
```

For example -

```
type Traffic_Light is (Red, Yellow, Green);
type Union (Option : Traffic_Light := Traffic_LightFirst) is
record
  -- common components

  case Option is
    when Red =>
      -- components for red
    when Yellow =>
      -- components for yellow
    when Green =>
      -- components for green
  end case;
end record;
```

#### **Example 2.1.1** Distinguish between PASCAL and ADA Union Types.

**Solution :** The ADA union is similar to Pascal union types but there is a difference between the two -

- 1) In ADA, there are no free unions. That means the tag is specified with union declaration.
- 2) When tag is changed, all appropriate fields must be set.

#### **Review Question**

1. Explain discriminated and free union variables.

#### **2.1.8 Pointer and Reference Types**

- **Pointer :** A pointer is a variable that stores a memory address, for the purpose of acting as an alias to what is stored at that address.
- A pointer can be used to access a location in the area where storage is dynamically

allocated which is called as **heap**.

- Variables that are dynamically allocated from the heap are called **heap dynamic variables**.
- Variables without names are called **anonymous variables**.

### Uses of Pointers

- 1) Provide the power of indirect addressing.
- 2) Provide a way to manage dynamic memory. A pointer can be used to access a location in the area where storage is dynamically created usually called a heap.

#### 2.1.8.1 Design Issues

The primary design issues are

- 1) Should a language support a pointer type or reference type or both ?
- 2) What are the scope and lifetime of a pointer variable ?
- 3) Are pointers used for dynamic storage management, indirect addressing or both ?
- 4) Are pointers restricted as to type of value to which they can point ?
- 5) What is the life time of dynamic variable ?

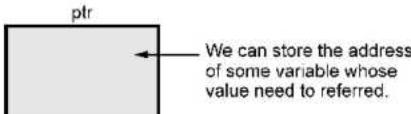
#### 2.1.8.2 Point Operations

Consider the variable declaration

```
int *ptr;
```

**ptr** is the name of our variable. The \* informs the compiler that we want a pointer variable, the int says that we are using our pointer variable which will actually store the address of an integer. Such a pointer is said to be **integer pointer**.

Thus **ptr** is now ready to store an address of the value which is of integer type.



**Fig. 2.1.6 Pointer variable**

The pointer variable is basically used to store some address of the variable which is holding some value.

Consider,

```
Line 1-> int *ptr;  
Line 2-> int a,b;
```

```

Line 3-> a=10; /*storing some value in a*/
Line 4-> ptr=&a; /*storing address of a in ptr*/
Line 5-> b=*ptr; /*getting value from address in ptr and storing it in b*/

```

Here we have used two important operators \* and &. The \* means 'contents at the specified address' and & means the 'address at'.

On Line 1 and Line 2 we have declared the required variables out of which **ptr** is a pointer variable and variables **a** and **b** are our normal variables. On Line 3 we have assigned value 10 to variable **a**. The Line 4 tells us that address of variable **a** is stored in a pointer variable **ptr**. And on Line 4 we have written that in **ptr** variable we have stored some address and at that address whatever value is stored, store that value in variable **b**. That means at **ptr** we have stored address of variable **a**. Then at the address of **a** whatever is a value we have to store that value in variable **b**. This can be illustrated by following Fig. 2.1.7.

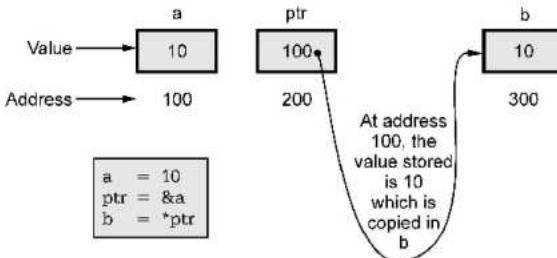


Fig. 2.1.7 Use of \* and & operator

The dynamic memory allocation is done using an operator **new**. The syntax of dynamic memory allocation using **new** is

new data type;

For example :

```

int *p;
p=new int;

```

We can allocate the memory for more than one element. For instance if we want to allocate memory of size in for 5 elements we can declare.

```

int *p;
p=new int[5];

```

In this case, the system dynamically assigns space for five elements of type **int** and returns a pointer to the first element of the sequence, which is assigned to **p**. Therefore, now, **p** points to a valid block of memory with space for five elements of type **int**.



The memory can be deallocated using the **delete** operator. The syntax is  
**delete variable\_name;**

For example

```
delete p
```

### 2.1.8.3 Pointer Problems

Following are **implementation problems** when pointers are used -

- Management of heap storage area :** Due to creation of objects of different sizes during execution time requires management of general heap storage area.
- The garbage problem :** Sometimes the contents of the pointers are destroyed and object still exists which is actually not at all accessible.
- Dangling references :** The object is destroyed however the pointer still contains the address of the used location, and can be wrongly used by the program.

**Example 2.1.2** For a language that provides a pointer type for programmer- constructed data

objects and operations such as new and dispose that allocate and free storage for data objects, write a program segment that generates a dangling reference. If one or the other program segment cannot be written, explain why.

**Solution :** The dangling reference is a live pointer that no longer points to a valid object.

```
var q : integer;
var p : integer;
begin
  new(p);
  q = p;
  dispose(p);
  ...
end
```

The live pointer p has created reference for q and then p is deleted. This creates dangling reference for q.

### 2.1.8.4 Pointers in Various Languages

#### C and C++

Pointers are basically the variables that contain the location of other data objects. It allows to construct complex data objects. In C or C++ pointer are data objects that can be manipulated by the programmer.

For example -

```
int *ptr;  
ptr=malloc(sizeof(int));
```

The \* is used to denote the pointer variable.

## FORTRAN 90

The first step in using Fortran pointers is to determine the variables to which you will need to associate pointers. They must be given the TARGET attribute in a type statement.

For example

```
real, target :: a, b(1000), c(10,10)  
integer, target :: i, j(100), k(20,20)
```

Then we define some pointers as -

```
real, pointer :: pa, aprt, pb(:), pc1(:, :), pc2(:, :)
```

The type of the pointer must match the type of the intended target.

## ADA

Pointers in ADA are known as **access types**. There are four kinds of access types in Ada : **pool access types**, **general access types**, **anonymous access types**, **access to subprogram types**.

For example -

```
type Int    is range -100 .. +500;  
type Acc_Int is access Int;
```

## PASCAL

Pascal support use of pointers. Pointers are the variables that hold the address of another variable.

For example -

```
Program pointers;  
type  
  Buffer = String[255];  
  BufPtr = ^ Buffer;  
Var B : Buffer;  
  BP : BufPtr;  
  PP : Pointer;
```

In this example, BP is a pointer to a Buffer type; while B is a variable of type Buffer.

### 2.1.8.5 Reference Type

- **Reference :** A reference is a variable that refers to something else and can be used as an alias for that something else.

- A pointer is a reference, but a reference is not necessarily a pointer.
- **Difference between Reference and Pointer Variable**

Sr. No.	Reference	Pointer
1.	References must be initialized when created.	Pointer can be initialized at any time.
2.	Once reference is assigned with some object it can not be changed.	Pointers can point to another object at any time.
3.	One can not have NULL references.	The pointer can be assigned with the value NULL.

- In pointers to access the value of actual variable we need to explicitly dereference the pointer variable by using value at address or using \* operator. In references, to access the value of actual variable we do not need to explicitly dereference the reference variable.
- In C++, the reference variable is a better choice for formal parameter than pointer. It must be initialized with the address of some variable in its definition and after initialization a reference type variable can never be set to reference any other variable,
- Reference can never point to NULL value whereas pointer can point to NULL. Thus with the use of reference there can not be NULL pointer assignment problem.
- **Creating Reference Variable**

The reference variables are created using the &. For example we can create a reference variable x for an integer variable a. It is as follows

```
int i=10;
int &x=i;      // x is a reference
```

Here variable x acts as a reference variable for variable i. Hence if value of i is changed then the value of x changes automatically. This is because variable x is a reference of variable i.

### Review Question

1. What is built-in and primitive data types? State the primitive data types in C++ and Java.

SPPU : May-19, Marks 6

## 2.2 Expression and Assignment Statements

### 2.2.1 Arithmetic Expressions

- Arithmetic expressions consist of operators, operands, parentheses, and function calls.
- For example  $x=y+2*\sqrt{25}$ ;
- The purpose of an arithmetic expression is to specify an arithmetic computation.

#### 2.2.1.1 Design Issues

Design issues for arithmetic expressions are -

- 1) What are the operator precedence rules ?
- 2) What are the operator associativity rules ?
- 3) What is the order of operand evaluation ?
- 4) Are there restrictions on operand evaluation side effects ?
- 5) Does the language allow user-defined operator overloading ?
- 6) What mode mixing is allowed in expressions ?

#### 2.2.1.2 Precedence and Associativity

##### Precedence :

- The operator precedence rules for expression evaluation define the order in which the operators of different precedence levels are evaluated.
- Many languages also include unary versions of addition and subtraction.
- Unary addition (+) is called the identity operator because it usually has no associated operation and thus has no effect on its operand.
- In all of the common imperative languages, the unary minus operator can appear in an expression either at the beginning or anywhere inside the expression, as long as it is parenthesized to prevent it from being next to another operator. For example, unary minus operator (-) :

$x + (- y) * z$       // is legal

$x + - y * z$       // is illegal

- Exponentiation has higher precedence than unary minus.

- The precedence of Ruby and C language operators is as given in following table

Precedence	Ruby	C/C++
Highest	**	postfix ++,-
	unary +, -	prefix ++,-, unary +,-
	* , /, %	* , /, %
Lowest	binary +, -	binary +, -

### Associativity :

- The operator associativity rules for expression evaluation define the order in which adjacent operators with the same precedence level are evaluated. An operator can be either left or right associative.
- Typical associativity rules :
  - Left to right, except exponentiation \*\*, which is right to left.
  - For example –  $a - b + c$  // left to right
  - Sometimes unary operators associate right to left (Fortran)

`A ** B ** C` // right to left  
`(A ** B) ** C` // in Ada it must be parenthesized

- The associativity rules for a few common languages are given here :

Language	Associativity Rule
Ruby, FORTRAN	Left : *, /, +, - Right : **
C-Based Languages	Left : *, /, %, binary +, binary - Right : ++, --, unary -, unary +

### 2.2.1.3 Operand Evaluation Order

The operand evaluation order is as given below -

- Variables** : Fetch the value from memory.
- Constants** : Sometimes a fetch from memory; sometimes the constant in the machine language instruction and not require a memory fetch.
- Parenthesized expression** : Evaluate all operands and operators first.

### 2.2.2 Overloaded Operators

- Overloading the operators is an activity for using the operator for more than one purpose. For example - the + operator can be used for addition of two integers, two complex numbers or for concatenating two strings.
- Both binary and unary operators can be overloaded.
- C++ has few operators that can not be overloaded such as dot operator(.), scope resolution(::) operator.
- User-defined operator overloading can harm the readability of a program because the built in operator has the precision and compiler knows all the precision between the operators, and it works on that precision. User can also create its own operator but the compiler does not come to know how to make precision of this operator. This will be the cause of the overloading harm.

### 2.2.3 Type Conversion

- Type conversion is a technique in which element present in one data type can be converted into another data type.
- There are two type of conversions –
  - 1) Narrowing conversion and 2) Widening conversion.
- A narrowing conversion** is one that converts an object to a type that cannot include all of the values of the original type. For example - conversion from double to float.
- A widening conversion** is one in which an object is converted to a type that can include at least approximations to all of the values of the original type. For example - from int to float.

There are two approaches of type conversions -

- 1) Implicit type conversion :** This type of type conversion is also called as coercion. The one data type element is automatically converted into another data type by widening conversion.

For example : In the following code the character data type of variable c is converted implicitly to integer data type.

```
int num = 10;
char c = 'm'; /* ASCII value is 109 */
int sum;
sum = num + c;      // output:119
In Ada, there are virtually no coercions in expressions
In ML and F#, there are no coercions in expressions
```

**2) Explicit type conversion :** It is also called as type casting in C based languages.

Syntax is

(type-name) expression

For example –

```
float a = 1.5;
int b = (int)a + 1;      //output will be 2
```

## 2.2.4 Relational and Boolean Expression

### Relational Expression

- Any relational expression is comprised of relational operator and two operands. A relational operator is an operator that compares the values of its two operands
- The value of a relational expression is Boolean, unless it is not a type included in the language
- Operator symbols used vary somewhat among languages ( $\neq$ ,  $\neq$ , .NE.,  $\diamond$ , #)
- The syntax of the relational operators available in some common languages is as follows :

Operation	Ada	C-Based Languages	Fortran 95
Equal	=	==	.EQ. or ==
Not Equal	/=	!=	.NE. or $\diamond$
Greater than	>	>	.GT. or >
Less than	<	<	.LT. or <
Greater than or equal	>=	>=	.GE. or >=
Less than or equal	<=	<=	.LE. or >=

- JavaScript and PHP have two additional relational operator, === and !==

### Boolean Expression

- In boolean expression, the operands are Boolean and result of the expression is Boolean.
- Boolean operators used in various languages are -

FORTRAN 90	C	Ada
and	&&	and
or		or
not	!	not

During evaluation of Boolean expression, the leftmost operator is evaluated first because in C language the relational operators are left associative producing either 0 or 1.

### 2.2.5 Short Circuit Evaluation

Short circuit evaluation is a kind of evaluation in which the expression is evaluated only if it is necessary.

MODULA-2 uses short circuit evaluation for OR and AND operators. The short circuit evaluation is also allowed in C as well.

For example -

```
while( p>=0 && q<=10)
    i=i+1;
If p>=10 is true then only the control reaches to test q<=10
```

With short circuit evaluation,

- While evaluating E1 or E2, if E1 is true then the whole expression is true and in that case E2 is not evaluated.
- Similarly, while evaluating E1 and E2, if E1 is false then E2 is not evaluated.

### 2.2.6 Assignment Statement

The purpose of assignment statement is to assign the value to a variable.

#### Types of Assignment Statements

- Simple Assignments :** The language like C, C++, JAVA makes use of = operator for assignment statement. The ALGOL, PASCAL, ADA makes use of := operator for assignment purpose.
- Conditional Targets :** In Perl the conditional target can be specified as -

```
($count ? $total : $subtotal) = 0
```

The above code is equivalent to

```
if ($count){
    $total = 0
} else {
    $subtotal = 0
}
```

- Compound Assignment Operator :** This is commonly used form of assignment statement in which shorthand method of using assignment statement.

For example -

```
a=a+b
can be written as
a+=b;
```

- 4) Assignment as an Expression :** In the C-based languages, Perl, and JavaScript, the assignment statement produces a result and can be used as an operand.

```
while ((ch = getchar())!= EOF) {  
...  
}
```

In above code, `ch = getchar()` is carried out; the result (assigned to `ch`) is used as a conditional value for the while statement.

- 5) Unary Assignment Operator :** It is possible to combine increment or decrement operator with assignments. The operators `++` and `--` can be used either in expression or to form stand-alone single-operator assignment statements. They can appear as prefix operators.

For example -

```
sum = ++ count; is equivalent to count=count+1; sum=count;  
If the same operator appears as postfix operator then  
sum=count++; sum=count; count=count+1
```

- 6) Multiple Assignments :** It is possible to perform multiple assignments at a time. For example - the Perl, Ruby, and Lua provide multiple-target multiple-source assignments.

```
($one, $two, $three) = (100, 200, 300);
```

## 2.2.7 Mixed Mode Assignment

- Assignment statements can also be mixed-mode.
- In Fortran, C, and C++, any numeric value can be assigned to any numeric scalar variable; whatever conversion is necessary is done.
- In Java and C#, only widening assignment coercions are done.
- In Ada, there is no assignment coercion.
- In all languages that allow mixed-mode assignment, the coercion takes place only after the right-side expression has been evaluated. For example, consider the following code :

```
int a, b;  
float c;  
...  
c = a / b;
```

In above code, `c` is float, the values of `a` and `b` could be coerced to float before the division.

**Review Questions**

1. Explain implicit and explicit type conversion process.
2. What is short circuit evaluation? Explain.
3. Explain various types of assignment statements.
4. Explain mix mode operation.

**2.3 Statement-level Control Statements**

A control structure is a control statement and the statements whose execution it controls.

**Design Issue :** There is only one design issue for control structure -  
" should a control structure have multiple entries ?"

**2.3.1 Selection Statements**

For choosing one correct path from two or more paths of execution, the selection statement is used.

There are two general categories of selection statements

1) Two Way Selectors

2) Multiple-way Selectors

**1) Two way selectors :** The general form of two way selector statement is -

```
if control_expression  
then clause  
else clause
```

Here

**Control expression :** These are specified in parenthesis. Some languages like C, C++ or JAVA do not use the reserved word **then**. The Java, C, C++ makes use of boolean expressions for control expressions.

**Clause :** In most contemporary languages, the **then** and **else** clauses either appear as single statements or compound statements. C-based languages use **braces** to form **compound statements**. One **exception** is Perl, in which all the and else clauses must be compound statements, even if they contain single statements. In Python and Ruby, clauses are statement sequences. **Python** uses **indentation** to define clauses. For example -

```
if a> b :  
    a = b  
    print "a is greater than b"
```

### Design Issues for Two-way Selection

- 1) How are then and else clauses specified ?
- 2) What is the form and type of expressions used for selection ?
- 3) How to specify the meaning of nested selectors ?

**2) Multiple-way Selectors :** It allows the selection of one of any number of statements or statement groups. For example - In C we can use **switch case** statement as

```
switch (choice) {  
    case 1 : c=a+b;  
    break;  
    case 2 : c=a-b;  
    break;  
    case 3 : c=a*b;  
    break;  
    case 4 : c=a/b;  
    break;  
    default : printf("exit");  
}
```

### Design Issues for Multi-way Selection Statement

1. What is the form and type of the control expression ?
2. How are the selectable segments specified ?
3. Is execution flow through the structure restricted to include just a single selectable segment ?
4. How are case values specified ?
5. What is done about unrepresented expression values ?

## 2.3.2 Iterative Statements

**Definition :** Iterative statements is one that cause a statement or collection of statements to be executed **zero one or more times**.

- The repeated execution of statements can be achieved either by iterative statements or by using recursion.
- The iterative statements are normally called as **loop**.
- The general **design issues** for iterative statements are,
  - How is iteration controlled ?
  - Where should the control mechanism appear in the loop statement ?
- The loop has **body** which is basically collection of statements whose execution is

- controlled by iterative statements.
- The iteration statement and the associated loop body together form an iteration statement.
  - There are two **important terms** associated with loop and those are -
    - Pretest** : The term pretest means that the loop completion occurs before the loop body is executed.
    - Posttest** : The term posttest means that the loop completion occurs after the loop body is executed.
  - Example of Loop** : Following is an example of for loop used in C

```
for(i=0;i<10;i++)
{
    printf("i = %d",i);
}
```

### 2.3.2.1 Counter Controlled Loops

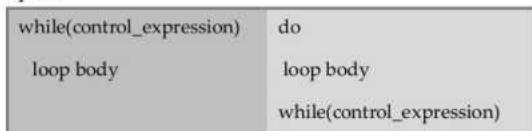
- The **counter-controlled loops** are those loop that possess a variable called the loop variables. This loop variable is used to maintain the count value.
- The loop variables of counter controlled loops include the means of specifying the initial and **terminal** values of the loop variable, and the difference between sequential loop variable values, is called the **stepsize**.
- The **initial, terminal** and **stepsize** are called the loop **parameters**.
- Design issues** for the counter controlled loops are -
  - What are the type and scope of the loop variable ?
  - Should it be legal for the loop variable or loop parameters to be changed in the loop body, and if so, does the change affect loop control ?
  - Should the loop parameters be evaluated only once, or once for every iteration ?
- Example of counter controlled loop** : Following code in Java that displays the character and its' ASCII value for all lower case characters.

```
for (char x = 'a'; x <= 'z'; x++)
System.out.println(x + " = " + (int) x);
```

### 2.3.2.2 Logically Controlled Loops

- The logically controlled loop is a kind of loop statement in which repetition control is based on a Boolean expression rather than a counter.
- C and C++ have both pretest and posttest forms, in which the control expression can be arithmetic :

- For example :



- In both C and C++ it is legal to branch into the body of a logically-controlled loop.
- Java is like C and C++, except the **control expression** must be **Boolean**.
- **Design Issues are**
  - 1) Should the control be pretest or posttest ?
  - 2) Should the logically controlled loop be a special form of a counting loop or a separate statement ?

### 2.3.2.3 User-located Loop Control Mechanism

- This is a control structure in which programmer choose a location for loop control. This location generally other than the top or bottom of the loop.
- In C or C++ the user located loop control mechanism is accomplished using unlabeled exits such as **break**.
- Following is an example user located loop control using C#

```
UP:  

for (row = 0; row < n; row++)  

for (col = 0; col < m; col++) {  

sum += a[row][col];  

if (sum > 1000.0)  

break UP;  

}
```

- The motivation for user-located **loop exits** is simple.
- They fulfill a common need for goto statements through a highly restricted branch statement.
- The target of a goto can be many places in the program, both above and below the goto itself .

### 2.3.2.4 Examples of Looping in C

The iteration means repeated execution of statements based on some condition. In C the **for**, **while**, **do-while** statements are used for iterations.

1. **for loop :** The general form of for loop is

```
for(initialization;condition;step)
{
    ...
    //statements
}
```

2. **while statement :** The general form of while statement is

```
while(condition)
{
    ...
    //statements
}
```

3. **do-while :** The general form of do-while is

```
do
{
    ...
    //statements
}while(condition);
```

For example - Following code will store numbers 10,11,12,13,14,15 in an array and display them.

```
#include<stdio.h>
void main()
{
    int i;
    for(i=0;i<=5;i++)
    {
        a[i]=i+10;
        printf("%d",a[i]);
    }
}
```

### **2.3.3 Unconditional Branching**

- An unconditional branch statement transfers execution control to a specified place in the program.
- The unconditional branch, or goto, is the most powerful statement for controlling the flow of execution of a program's statements.
- For example -

```
sum=0;
for(int i = 0; i<=10; i++)
{
```

```
sum = sum+i;
if(i==5)
{
    goto addition;
}
}
addition:
printf("%d", sum);
```

- Java, Python, and Ruby do not have a **goto**. C# uses **goto** in switch statement.

#### Problems associated with unconditional branching

- The unconditional branch, or **goto**, is the most powerful statement for controlling the flow of execution of a program's statements.
- Without restrictions on use, imposed by either language design or programming standards, **goto** statements can make programs very difficult to read, and as a result, code becomes highly unreliable and costly to maintain.
- The **gotos** have ability to force any program statement to follow any other in execution sequence, regardless of whether the statement proceeds or follows previously executed statement in textual order.

## 2.4 Subprograms

### 2.4.1 Fundamentals of Subprograms

- Subprograms can be viewed as abstract operations on a predefined data set.
- A **subprogram definition** describes the interface to and the actions of the subprogram abstraction.
- A **subprogram call** is an explicit request that the subprogram be executed. For example - In Python the header of subprogram is as follows -

```
def swap(parameters):
```

- There are two categories of sub-program –

1. Procedure and      2. Functions.

- **Procedure** - Procedure is set of commands executed in order.
- **Function** - Function is set of instructions for used for some computation.

**2.4.1.1 General Subprogram Characteristics**

- Each subprogram has a **single entry point**.
- The calling program is suspended during execution of the called subprogram.
- Control always returns to the caller when the called subprogram's execution terminates.

**2.4.1.2 Basic Definitions**

- The **subprogram call** is an explicit request that subprogram be executed.
- The **subprogram header** is first part of the definition, including the name, the kind of subprogram and the formal parameters.
- The **signature** is also called as **parameter profile** which is nothing but the number, order and types of its parameters.

**2.4.1.3 Procedure and Functions**

Sr.No.	Procedure	Function
1.	Procedure may or may not return a value.	Function returns a value.
2.	Procedure is set of commands executed in order.	Function is set of instructions for used for some computation.
3.	Function can be called from procedure.	Procedures can not be called from function.
4.	Example - Pascal, ADA are some programming languages that make use of procedures.	Example - C, C++, Java are some programming languages that make use of functions.

**Example of Procedure in Pascal**

```
Procedure MaxNum (x, y : integer; var z : integer)
begin
  if      x > y then
    z: = x
  else
    z: = y;
end;
```

### Example of Function in Pascal

```
Function MaxNum (x, y : integer) : integer; ( returning maximum between two numbers )
var
    result : integer;
begin
    if (x > y) then
        result := x
    else
        result := y;
    MaxNum := result;
end;
```

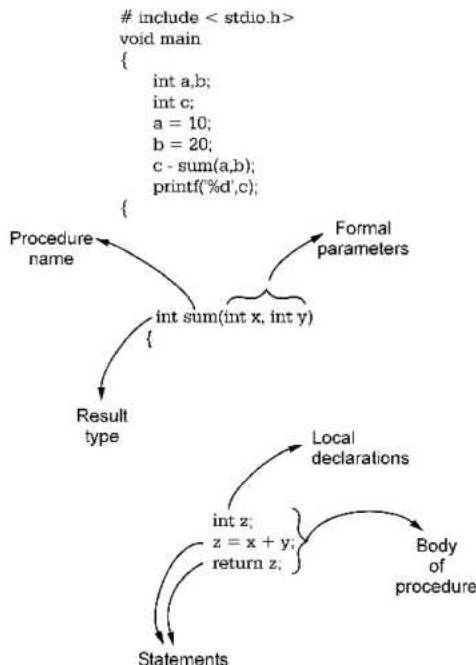
### Different Elements of Procedure

Various elements of procedure are -

1. Name for declaration of procedure
2. Body consisting of local declarations and statements
3. Formal parameters which are the placeholders of actuals
4. Optional result type.

### For example

Consider following C code



### Benefits of Procedure

Following are some benefits of using procedures

1. The implementation of certain piece of code can be **separated out** from the main implementation. Suitable names used for such procedure represents its purpose as well. For instance - If we call swap(a,b) function in the main implementation then it will indicate interchanging of the values a and b.
2. Writing a separate procedure allows to hide **implementation** details. The main procedure looks abstract with simply the call to the procedure.
3. Procedures can be used to partition a program into smaller modules. These **modules** are useful to maintain a large and complex code.
4. **Finding errors** from a large complex program becomes simplified when procedures or functions are written.

### 2.4.2 Design Issues for Subprogram

The design issues are as follows -

- Are local variables static or dynamic ?
- Can subprogram definitions appear in other subprogram definitions ?
- What parameter passing methods are provided ?
- Are parameter types checked ?
- If subprograms can be passed as parameters and subprograms can be nested, what is the referencing environment of a passed subprogram ?
- Can subprograms be overloaded ?
- Can subprogram be generic ?
- If the language allows nested subprograms, are closures supported ?

### 2.4.3 Local Referencing Environments

**What is the meaning of referencing environment ?**

- Each program or subprogram has a set of identifiers that has associations for the use in referencing during its execution. Thus referencing environment is a complete set of bindings active at a certain point in a program.
- **Local referencing environment :** This kind of referencing environment is denoted by the **set of identifiers** that are **created on the entry of the subprogram** and are used within the subprogram. The meaning of reference to the name can be found within the local environment of the subprogram definition. For example **local**

**variables**, and formal parameters represent the set of identifiers that are associated with local referencing environment. This kind of referencing environment is also called as **local environment**.

#### 2.4.3.1 Local Variables

- Variables that are defined inside subprograms are called local variables. There are two types of local variables –

1. Static Local Variables

2. Stack Dynamic Variables

**1. Static local variable :** The static local variables are variables whose scope finishes when the subprogram within which it is defined gets terminated and they retain their value once they are initialized.

##### Advantages :

- 1) The static local variables are efficient than stack-dynamic local variables as they require no run time overhead for allocation and deallocation.
- 2) It allows direct accessing.
- 3) They allow subprograms to be history sensitive.

##### Disadvantages :

- 1) They do not support for recursion.
- 2) The storage for locals cannot be shared among some programs.

**2. Stack dynamic variable :** The stack dynamic local variables are kind of local variables which are bound to storage when the **subprogram** begins **execution** and are unbound from storage when the execution terminates. Stack-dynamic variables are allocated from the run-time stack.

##### Advantages

- 1) The stack dynamic local variables support for **recursion**.
- 2) The storage for locals is **shared** among some subprograms.

##### Disadvantages

- 1) The **cost** is required to allocate, initialize, and deallocate the variables for each call to subprogram.
- 2) **Accesses** to stack dynamic local variables must be indirect.

- 3) When all the local variables are stack dynamic, subprograms cannot be history sensitive. That means they **can not retain data values** of local variables between calls.

#### 2.4.3.2 Nested Subprograms

- Some non-C-based static-scoped languages such as Fortran 95, Ada, Python, JavaScript, and so on use stack-dynamic local variables and allow subprograms to be nested.
- All variables that can be non-locally accessed reside in some activation record instance in the stack.
- The process of locating a non-local reference is as follows -
  - Find the correct activation record instance
  - Determine the correct offset within that activation record instance
- Locating a non-local reference
  - Finding the offset is easy
  - Finding the correct activation record instance
- Static semantic rules guarantee that all non-local variables that can be referenced have been allocated in some activation record instance that is on the stack when the reference is made.

#### 2.4.4 Parameter Passing Methods

- Parameter passing is the mechanism used to pass parameters to a procedure (subroutine) or function.
- The most common methods are to pass the parameters are call by value and call by reference.
- When the value of the actual parameter is passed then it is called **call by value method**.
- When we pass the address of the memory location where the actual parameter is stored to the function or procedure then it is called as **call by reference method**.

##### Various Parameter Passing Methods

1. **Call by value :** This is the simplest method of parameter passing.
  - The actual parameters are evaluated and their r-values are passed to called procedure.
  - The operations on formal parameters do not changes the values of actual parameter.

**Example :** Languages like C, C++ use actual parameter passing method. In PASCAL the non-var parameters.

**2. Call by reference :** This method is also called as **call by address or call by location**.

- The L-value, the address of actual parameter is passed to the called routines activation record.
- The values of actual parameters can be changed.
- The actual parameters should have an L-value.

**Example :** Reference parameters in C++, PASCAL'S var parameters.

**3. Copy restore :** This method is a hybrid between call by value and call by reference. This method is also known as **copy-in-copy-out or values result**.

- The calling procedure calculates the value of actual parameter and it is then copied to activation record for the called procedure.
- During execution of called procedure, the actual parameters value is not affected.
- If the actual parameter has L-value then at return the value of formal parameter is copied to actual parameter.

**Example :** In Ada this parameter passing method is used.

**4. Call by name :** This is less popular method of parameter passing.

- Procedure is treated like macro. The procedure body is substituted for call in caller with actual parameters substituted for formals.
- The actual parameters can be surrounded by parenthesis to preserve their integrity.
- The locals names of called procedure and names of calling procedure are distinct.

**Example :** ALGOL uses call by name method.

#### Difference between Parameter Pass By Value and Parameter Pass By Reference

Call By Value	Call By Reference
When a function is called, then in that function actual values of the parameters are passed.	When a function is called, then in that function addresses of the parameters are passed.
The parameters are simple variables.	The parameters are pointer variables.

If the values of the parameters get changed in the function then that change is not permanent. That means the values of the parameter get restored when the control returns back to the caller function.	If the values of the parameters get changed in the function then that change is permanent. That means the values of the parameter get changed when the control returns back to the caller function.
<b>Example :</b> <pre>main() {     x=10;     fun(x);     printf("%d",x); } void fun(x) {     x=15; }</pre> <b>Output</b> will be 10 and not 15	<b>Example</b> <pre>main() {     *x=10;     fun(x);     printf("%d", *x); } void fun(x) {     x=15; }</pre> <b>Output</b> will be 15 and not 10
Compiler executes this type function slowly as values get copied to formal parameters.	Compiler executes this type of function faster as addresses get copied to the formal parameters.

### Parameter Passing Methods of Common Languages :

#### 1) C

- It supports pass by value method.
- The pass by reference is achieved by using pointers as parameter.

#### 2) C++ :

- A special pointer type called reference type is used for pass-by-reference.

#### 3) Java :

- All parameters are passed by value
- Object parameters are passed by reference

#### 4) ADA :

- There are three semantics modes of parameter transmission: in, out, in out; in is the default mode.
- Formal parameters declared out can be assigned but not referenced; those declared in can be referenced but not assigned; in out parameters can be referenced and assigned.

**5) C# :**

- Default method of parameter passing is pass by value.
- Pass-by-reference is specified by preceding both a formal parameter and its actual parameter with `ref` as keyword.

**2.5 Abstract Data Types and Encapsulation Construct****SPPU : Dec.-17, Marks 7****Concept of Abstraction**

- An abstraction is a view or representation of an entity that includes only the most significant attributes.
- It is fundamental aspect of programming.
- Almost all the languages support process of abstraction with subprograms.
- Abstraction is a process of hiding the implementation details from the user, only the functionality will be provided to the user. In other words, the user will have the information on what the object does instead of how it does it.
- In Java, abstraction is achieved using **Abstract classes and interfaces**.

**Benefits of Abstraction**

- 1) It allow organization of data and corresponding operations in a single unit which can be considered as a data structure together.
- 2) Modifications can be made easily without affecting rest of the code.
- 3) By hiding the data representations, user code cannot directly accessible. This helps in making the programs reliable.

**Data Abstraction**

- The concept of data abstraction is represented by means of **abstract data type**.
- An abstract data type is a user-defined data type that satisfies the following **two conditions** :
  - 1) The representation of objects of the type is hidden from the program units that use these objects.
  - 2) The declarations of the type and the protocols of the operations on objects of the type are contained in a single syntactic unit. Other program units are allowed to create variables of the defined type.

**Procedural Abstraction**

- It specifies what a procedure does and ignores how it does. Following are advantages of procedural abstraction -

- 1) **Locality:** Programmers don't need to know the implementation details.
- 2) **Modifiability:** Replacing of one code does not affect another code.
- 3) **Language Independence :** Due to procedural abstraction implementation could be done in any programming language.
- Procedural abstractions are normally characterized in a programming language as "function/sub-function" or "procedure" abstraction.

#### Difference between Data Abstraction and Procedural Abstraction

Data Abstraction	Procedure Abstraction
The focus is primarily on data and then on procedure for abstraction.	The focus is only on procedures for abstraction.
Data abstraction is characterized as a data structure unit.	Procedural abstractions are normally characterized in a programming language as "function/sub-function" or "procedure" abstraction.
The advantage of data abstraction over procedural abstraction is that the data and the associated operations get specified together and hence it is easy to modify the code when data changes.	Due to procedural abstraction implementation could be done in any programming language.

#### 2.5.1 Design Issues For Abstraction

Following are the design issues of abstract data types -

- 1) What is the form of the container for the interface to the type ?
- 2) Can abstract types be parameterized ?
- 3) What access controls are provided ?

#### 2.5.2 Parameterized Abstract Data Types

- Parameterized ADTs allow designing an ADT that can store any type elements - only an issue for static typed languages.
- It is also known as **generic classes**.
- C++, Ada, Java 5.0, and C# 2005 provide support for parameterized ADTs.

### Parameterized ADT in ADA

The generic package can be used for implementing the stack of any type of elements is as follows -

```
generic
  Max: Positive;
  type Element_T is private;
  package Generic_Stack is
    procedure Push (E: Element_T);
    function Pop return Element_T;
  end Generic_Stack;
  package body Generic_Stack is
    Stack: array (1 .. Max) of Element_T;
    Top : Integer range 0 .. Max := 0; -- initialise to empty
    -- ...
  end Generic_Stack;
```

A stack of a given size and type could be defined in this way :

```
declare
  package Float_100_Stack is new Generic_Stack (100, Float);
  use Float_100_Stack;
begin
  Push (45.8);
  -- ...
end;
```

### Parameterized ADT in C++

**Templates** are considered to be the generic abstract data type. In case of templates, the data components can be of different types however, the operations are the same.

Using class template we can write a class whose members use template parameters as types.

The complete program using class template is as given below.

```
#include <iostream.h>
template <class T>
class Compare { //writing the class as usual
    T a, b; //note we have used data type as T
public:
    Compare (T first, T second)
    {
        a=first;
        b=second;
    }
    T max (); //finds the maximum element among two
};
```

```

//template class member function definition
//here the member function of template class is max
template <class T>
T Compare <T>:: max ()
{
    T val;
    if(a>b)
        val=a;
    else
        val=b;
    return val;
}
void main ()
{
    Compare <int> obj1 (100, 60);           //comparing two integers
    Compare <char> obj2('p','t');          //comparing two characters
    cout << "\n maximum(100,60) = "<obj1.max();
    cout << "\n maximum('p','t') = "<obj2.max();
}

```

**Output**

```

maximum(100,60) = 100
maximum('p','t') = t

```

### 2.5.3 Encapsulation Constructs

- Encapsulation is a mechanism for wrapping the data (variables) and code acting on the data (methods) together as a single unit.
- The logical module of C++ is class. The class defines the encapsulated unit. It encapsulates the **data** members and the operations that can be performed on these data members.

For example

```

class Test
{
private:
    int a,b,c;
public:
    void add();
    void display();
};

```

- Thus encapsulation approach is useful in two ways -
  - It binds the data members with operations or methods.
  - It imposes the level of abstraction in a program.

**Review Question**

1. What are abstract data types ? How C++ implements abstract data types ? Give example.

**SPPU : Dec.-17, Marks 7**

**2.5.4 Naming Encapsulations**

- Large programs define many global names need a way to divide into logical groupings. A naming encapsulation is used to create a new scope for names.
- C++, C#, Java, Ada, and Ruby provide naming encapsulations.
- Namespaces are used to group the entities like class, variables, objects, function under a name. The namespaces help to divide global scope into sub-scopes, where each sub-scope has its own name.
- For example - Following C++ program shows how to create namespaces

```
namespace ns1
{
    int a = 5;
}

namespace ns2
{
    char a[] = "Hello";
}

int main ()
{
    cout << ns1::a << endl;
    cout << ns2::a << endl;
    return 0;
}
```

**Output**

5

Hello

**Program Explanation :**

In above program,

- There are two different namespaces containing the variable **a**. The variable **a** in the first namespace **ns1** is of type **int** but the variable **a** in the second namespace **ns2** is of array of characters. The both the entities are treated separately. There is no re-declaration error for variable **a**.

## 2.6 Multiple Choice Questions

**Q.1 Which of the following is not valid variable name?**

- |   |  |
|---|--|
| <input type="checkbox"/> a int \$index; | <input type="checkbox"/> b int index1; |
| <input type="checkbox"/> c int ind1x    | <input type="checkbox"/> d int index;  |

**Explanation :** The variable name should not begin with special symbols like @, #, \$, % and so on.

**Q.2 Which of the following is true for variable names in C?**

- a They can contain alphanumeric characters as well as special characters
- b Variable names cannot start with a digit
- c It is not an error to declare a variable to be one of the keywords (like goto, static)
- d None of these

**Q.3 Which of the following is true of l-values and r-values ?**

- a An l-value is a logical value, and an r-value is a real value.
- b l-values are always to the left of r-values
- c An l-value refers to a variable's location while an r-value to its current value
- d l-values are local and r-values are relative.

**Q.4 Which is the longest scope in the following code?**

```
#include<stdio.h>
int x;
int main()
{
    int y;
    fun();
    return 0;
}
void fun()
{
    int z;
}
```

- |                              |   |
|------------------------------|---|
| <input type="checkbox"/> a x | <input type="checkbox"/> b y            |
| <input type="checkbox"/> c z | <input type="checkbox"/> d both a and b |

**Explanation :** The variable x is a global variable and its scope is entire program including all the functions.

**Q.5** The variables which can be accessed by all modules in a program, are called \_\_\_\_\_.

- |  |   |
|--|---|
| <input type="checkbox"/> a local variables   | <input type="checkbox"/> b internal variables |
| <input type="checkbox"/> c external variable | <input type="checkbox"/> d global variables   |

**Q.6** The value of an automatic variable that is declared but not initialized will be \_\_\_\_\_.

- |  |  |
|--|--|
| <input type="checkbox"/> a 0             | <input type="checkbox"/> b -1            |
| <input type="checkbox"/> c Unpredictable | <input type="checkbox"/> d none of these |

**Explanation :** If the variable is not initialized then it takes the garbage value. Hence the unpredictable value is present in the uninitialized variable.

**Q.7** Which of the following is not data type in Pascal?

- |                                 |                                    |
|---------------------------------|------------------------------------|
| <input type="checkbox"/> a real | <input type="checkbox"/> b float   |
| <input type="checkbox"/> c char | <input type="checkbox"/> d integer |

**Q.8** What will happen if null pointer is converted to bool \_\_\_\_\_.

- a The bool value is evaluated to true
- b The bool value is evaluated to false
- c error is raised
- d None of these

**Explanation :** In pointer, the nonzero pointer is converted to true and zero pointer is converted to false.

**Q.9** What is the output of the following code?

```
int main(void)
{
    char name = 'P';
    P' = 10; /* assigning to character P number 10 */
    return 0;
}
```

- |   |  |
|---|--|
| <input type="checkbox"/> a name will contain value 10 | <input type="checkbox"/> b P will contain 10 |
| <input type="checkbox"/> c Syntax error               | <input type="checkbox"/> d None of these     |

**Explanation :** The assignment 'P'=10 will cause the error: l-value required as left operand of assignment.

**Q.10** Choose the correct statement

- |   |  |
|---|--|
| <input type="checkbox"/> a Reference is stored on stack | <input type="checkbox"/> b Reference is stored on heap |
| <input type="checkbox"/> c Reference is stored on queue | <input type="checkbox"/> d None of these               |

**Q.11 Discriminated unions are supported by \_\_\_\_\_.**

- |                               |   |
|-------------------------------|---|
| <input type="checkbox"/> a C  | <input type="checkbox"/> b C++          |
| <input type="checkbox"/> c ML | <input type="checkbox"/> d All of these |

**Q.12 In C, reference is declared using the symbol \_\_\_\_\_.**

- |                               |                              |
|-------------------------------|------------------------------|
| <input type="checkbox"/> a *  | <input type="checkbox"/> b & |
| <input type="checkbox"/> c && | <input type="checkbox"/> d ! |

**Q.13 Choose the correct statement.**

I. Array of References can be created.

II Change in reference changes the referent.

- |  |   |
|--|---|
| <input type="checkbox"/> a Only I is correct         | <input type="checkbox"/> b Only II is correct           |
| <input type="checkbox"/> c Both I and II are correct | <input type="checkbox"/> d Neither I and II are correct |

**Q.14 Choose the correct statement.**

- |  |
|--|
| <input type="checkbox"/> a Reference must be initialized within a function.    |
| <input type="checkbox"/> b Reference must be initialized outside the function. |
| <input type="checkbox"/> c Reference must be always initialized.               |
| <input type="checkbox"/> d None of these                                       |

**Q.15 Choose the correct statement.**

- |   |
|---|
| <input type="checkbox"/> a A reference is not a constant pointer    |
| <input type="checkbox"/> b Reference is automatically de-referenced |
| <input type="checkbox"/> c Both a and b                             |
| <input type="checkbox"/> d none of these                            |

**Q.16 Choose the correct option.**

- |  |
|--|
| <input type="checkbox"/> a A referenced variable need not be de-referenced to access value.              |
| <input type="checkbox"/> b A referenced variable need to be de-referenced to access value.               |
| <input type="checkbox"/> c It depends upon the type of reference whether to de-refer it to access value. |
| <input type="checkbox"/> d None of these   |

**Q.17 Choose the correct statement I. The variable and its reference are linked together.**

II. We can change the value of variable via its reference.

- |  |  |
|--|--|
| <input type="checkbox"/> a Only I        | <input type="checkbox"/> b Only II       |
| <input type="checkbox"/> c Both I and II | <input type="checkbox"/> d None of these |

**Q.18 If an expression contains double, float, and long then the data type of that expression becomes \_\_\_\_\_.**

- |                                 |                                   |
|---------------------------------|-----------------------------------|
| <input type="checkbox"/> a int  | <input type="checkbox"/> b float  |
| <input type="checkbox"/> c long | <input type="checkbox"/> d double |

**Q.19 Explicit type conversion is known as \_\_\_\_\_.**

- |                                       |  |
|---------------------------------------|--|
| <input type="checkbox"/> a conversion | <input type="checkbox"/> b casting       |
| <input type="checkbox"/> c separation | <input type="checkbox"/> d None of these |

**Q.20 What is strong type system?**

- a The type system in which only built-in data types are allowed.
- b The type system in which only user defined data types are allowed.
- c The type system that guarantees not to generate type errors.
- d None of these.

**Q.21 For representing logical values \_\_\_\_\_ data type is used in C++.**

- |                                 |   |
|---------------------------------|---|
| <input type="checkbox"/> a int  | <input type="checkbox"/> b bool             |
| <input type="checkbox"/> c char | <input type="checkbox"/> d all of the above |

**Explanation :** The C++ allows the boolean data type for denoting true or false values.

**Q.22 Looping in a program means \_\_\_\_\_.**

- a jumping to the specified branch of program
- b repeat the specified lines of code
- c both of above
- d none of these

**Q.23 Which of the following is the correct order of evaluation for the below expression?**

$$z = x + y * z / 4 \% 2 - 1$$

- |   |   |
|---|---|
| <input type="checkbox"/> a * / \% + - = | <input type="checkbox"/> b = * / \% + - |
| <input type="checkbox"/> c / * \% - + = | <input type="checkbox"/> d * \% / - + = |

**Q.24 Which of the following statement allows the programmer to make the control to the beginning of the loop, without executing the next statement inside the loop ?**

- |                                  |                                     |
|----------------------------------|-------------------------------------|
| <input type="checkbox"/> a While | <input type="checkbox"/> b Continue |
| <input type="checkbox"/> c Go to | <input type="checkbox"/> d If       |

**Q.25 Which selection process is an example of multiple branches from a single expression?**

- |                                       |  |
|---------------------------------------|--|
| <input type="checkbox"/> a If... Then | <input type="checkbox"/> b Select Case |
| <input type="checkbox"/> c Do Loop    | <input type="checkbox"/> d For....Next |

**Q.26 What is required to reference an element in an array ?**

- |  |   |
|--|---|
| <input type="checkbox"/> a Array name    | <input type="checkbox"/> b Index value of the element |
| <input type="checkbox"/> c Element value | <input type="checkbox"/> d Both a and b               |

**Q.27 The following the exit controlled loop \_\_\_\_\_.**

- |                                  |                                     |
|----------------------------------|-------------------------------------|
| <input type="checkbox"/> a while | <input type="checkbox"/> b do-while |
| <input type="checkbox"/> c for   | <input type="checkbox"/> d go to    |

**Q.28 The scope of a variable refers to : \_\_\_\_\_.**

- |  |   |
|--|---|
| <input type="checkbox"/> a The length of the variable        | <input type="checkbox"/> b The name of the variable     |
| <input type="checkbox"/> c The accessibility of the variable | <input type="checkbox"/> d The lifetime of the variable |

**Q.29 The attributes of three arithmetic operators in some programming language are given below.**

Operator	Precedence	Associativity	Arity
+	High	Left	Binary
-	Medium	Right	Binary
*	Low	left	Binary

**The value of the expression  $2-5+1-7*3$  in this language is \_\_\_\_\_?**

- |                              |                              |
|------------------------------|------------------------------|
| <input type="checkbox"/> a 1 | <input type="checkbox"/> b 2 |
| <input type="checkbox"/> c 3 | <input type="checkbox"/> d 9 |

**Explanation :**

$2-(5+1)-7*3$  as + is having high precedence.

$$= 2-6-7*3$$

$= 2-(6-7)*3$  as - has medium precedence

$$= 2-(-1)*3$$

$$= 3*3$$

$$= 9$$

**Q.30 The goal of structured programming is to : \_\_\_\_\_.**

- a Have well indented programs
- b Be able to infer the flow of control from the compiled code
- c Be able to infer the flow of control from the program text
- d Avoid the use of GOTO statements

**Answer Keys for Multiple Choice Questions :**

Q.1	A	Q.2	b	Q.3	c	Q.4	a
Q.5	d	Q.6	c	Q.7	b	Q.8	b
Q.9	c	Q.10	a	Q.11	c	Q.12	b
Q.13	b	Q.14	c	Q.15	b	Q.16	a
Q.17	c	Q.18	d	Q.19	b	Q.20	c
Q.21	b	Q.22	b	Q.23	a	Q.24	b
Q.25	b	Q.26	d	Q.27	b	Q.28	c
Q.29	d	Q.30	c				



## Notes

## UNIT- III

# 3

# Java as Object Oriented Programming Language - Overview

### Syllabus

**Fundamentals of JAVA, Arrays :** one dimensional array, multi-dimensional array, alternative array declaration statements,

**String Handling :** String class methods, **Classes and Methods :** class fundamentals, declaring objects, assigning object reference variables, adding methods to a class, returning a value, constructors, this keyword, garbage collection, finalize() method, overloading methods, argument passing, object as parameter, returning objects, access control, static, final, nested and inner classes, command line arguments, variable - length arguments.

### Contents

3.1	Fundamentals of Java .....	Dec.-17, May-18, 19, .....	Marks 6
3.2	Java Program Structure		
3.3	How to Write a Java Program ?		
3.4	Comments in Java		
3.5	Variables		
3.6	Keywords		
3.7	Data Types and Size		
3.8	Operators		
3.9	User Defined Data Types		
3.10	Control Statements		
3.11	Jump Statements		
3.12	Arrays .....	Dec.-17, Nov.-18,.....	Marks 6
3.13	String Handling .....	May-19,.....	Marks 6
3.14	Classes and Methods		

3.15 Objects	
3.16 Adding Methods to Class	
3.17 Constructor	..... <b>Dec.-17, May-18,</b>
	..... <b>Nov.-18, 19,</b> ..... Marks 8
3.18 this keyword	..... <b>Dec.-19,</b> ..... Marks 6
3.19 Garbage Collection	
3.20 finalize() Method	
3.21 Overload Methods	
3.22 Argument Passing	
3.23 Object as Parameter	
3.24 Returning Object	
3.25 Access Control	
3.26 Static Members	
3.27 Nested and Inner Class	
3.28 Command Line Argument	
3.29 Variable Length Arguments	

## 3.1 Fundamentals of Java

### 3.1.1 Features of Java

Java was invented by James Gosling, Patrick Naughton, Chris Warth, Ed Frank and Mike Sheridan at Sun Microsystems, Inc. in 1991.

Java is getting popular because of its features. Following is a list of buzzwords that makes Java a popular programming language.

- Simple
- Secure
- Platform independent
- Object oriented
- Distributed
- Portable
- High performance
- Robust
- Multithreaded
- Interpreted
- Dynamic

Let us discuss these features in detail.

#### 1. Java is simple and small programming language

Java is a simple programming language. Even though you have no programming background you can learn this language comfortably. The programmers who worked on C or C++ can learn this language more efficiently because the syntax of Java resembles with C and C++. In fact Java is made more clear and adaptable than C++.

#### 2. Java is robust and secure

Following are reasons that make Java more secured than other programming languages -

1. Java does not support the concept of pointers directly. This makes it impossible to accidentally reference memory that belongs to other programs or the kernel.
2. The output of Java compiler is not executable code but it is a bytecode. The instructions of bytecode is executed by the Java Virtual Machine (JVM). That means JVM converts the bytecode to machine readable form. JVM understand only the

bytecode and therefore any infectious code can not be executed by JVM. No virus can infect bytecode. Hence it is difficult to trap the internet and network based applications by the hackers.

3. In programming languages like C or C++ the memory management is done explicitly by the user. That means user allocates or deallocates the memory. Whereas in Java its automatically done using garbage collection. Thus user can not perform the memory management directly.
4. If an applet is executing in some browser then it is not allowed to access the file system of local machine.

#### **3. Java is a platform independent and portable programming language**

Platform independence is the most exciting feature of Java program. That means programs in Java can be executed on variety of platforms. This feature is based on the goal - *write once, run anywhere and at anytime forever*.

Java supports portability in 2 ways - Java compiler generates the byte code which can be further used to obtain the corresponding machine code. Secondly the primitive data types used in Java are machine independent.

#### **4. Java is known as object oriented programming language**

Java is popularly recognised as an object oriented programming language. It supports various object oriented features such as data encapsulation, inheritance, polymorphism and dynamic binding. Everything in Java is an object. The object oriented model of Java is simple and can be extended easily.

#### **5. Java is multithreaded and interactive language**

Java supports multithreaded programming which allows a programmer to write such a program that can perform many tasks simultaneously. This ultimately helps the developer to develop more interactive code.

#### **6. Java can be compiled and interpreted**

Normally programming languages can be either compiled or interpreted but Java is a language which can be compiled as well as interpreted. First, Java compiler translates the Java source program into a special code called bytecode. Then Java interpreter interprets this bytecode to obtain the equivalent machine code. This machine code is then directly executed to obtain the output.

## **7. Java is known for its high performance, scalability, monitoring and manageability**

Due to the use of bytecode the Java has high performance. The use of multi-threading also helps to improve the performance of the Java. The J2SE helps to increase the scalability in Java. For monitoring and management Java has large number of Application Programming Interfaces(API). There are tools available for monitoring and tracking the information at the application level.

## **8. Java is a dynamic and extensible language**

This language is capable of dynamically linking new class libraries, methods and objects. Java also supports the functions written in C and C++. These functions are called native methods.

## **9. Java is a designed for distributed systems**

This feature is very much useful in networking environment. In Java, two different objects on different computers can communicate with each other. This can be achieved by Remote Method Invocation(RMI). This feature is very much useful in client-server communication.

## **10. Java can be developed with ease**

There are various features of Java such as Generics, static import, annotations and so on which help the Java programmer to create a error free reusable code.

### **3.1.2 Difference between C, C++ and Java**

Sr. No.	C	C++	Java
1.	C is a language that needs to be <b>compiled</b> .	The C++ is a language that needs to be <b>compiled</b> .	The Java is a language that gets <b>interpreted</b> and <b>compiled</b> .
2.	C is platform <b>dependent</b> .	C++ is platform <b>dependent</b> .	Java is a <b>platform independent</b> .
3.	C does not support multi-threading programming.	C++ does not support <b>multi-threading</b> programming.	Java supports <b>multi-threading</b> .
4.	C does not have facility to create and implement the graphical user interface.	C++ does not have facility to create and implement the graphical user interface.	Using Java, one can design very interactive and <b>user friendly</b> graphical user interface.

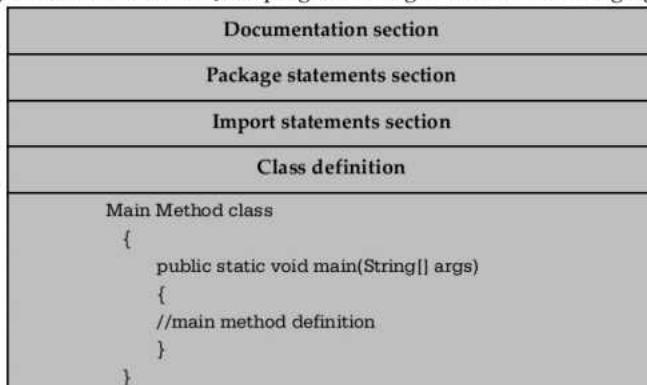
5.	Database handling using C is very complex. The C does not allow the database connectivity.	Database handling using C++ is very complex. The C++ does not allow the database connectivity.	<b>Java servlets</b> can be created which can interact with the <b>databases</b> like MS-ACCESS, Oracle, MySQL and so on.
6.	C code can not be embedded in any scripting language.	C++ code can not be embedded in any scripting language.	Java code can be <b>embedded within a scripting language</b> by means of applet programming.
7.	C does not have any exception handling mechanism.	C++ has exception handling mechanism using try-catch block.	Java has exception handling mechanism using <b>try-catch</b> and <b>throw</b> .
8.	C does not supports any object oriented programming concept. It is a procedure oriented programming language.	C++ supports <b>multiple inheritance</b> . It is an <b>object oriented programming language</b> .	Java does <b>not support multiple inheritance</b> however it makes use of interface. It is an <b>object oriented programming language</b> .
9.	In C we can use the <b>pointers</b> .	In C++ we can use the <b>pointers</b> .	In Java there is <b>no concept of pointers</b> .
10.	C does not support templates.	C++ supports <b>templates</b> .	Java does not support the concept of <b>templates</b> .
11.	In C there is no class. The structure is used instead of that.	In C++ we can write a program without a class.	In Java, there must be <b>at least one class</b> present.
12.	C is simple to use and implement.	C++ is simple to use and implement.	Java is <b>safe</b> and more <b>reliable</b> .
13.	C can be compiled with variety of compilers.	C++ can be compiled with variety of compilers.	Java can be compiled using an unique compiler.

**Review Questions**

1. Justify the meaning of each characteristic of Java in the statement "Java is simple, architecture neutral, portable, interpreted and robust and secured programming language".  
**SPPU : Dec.-17, Marks 6**
2. Explain why Java is secured, portable and dynamic ? Which of the concepts in Java ensures these ?  
**SPPU : May-18, Marks 6**
3. Explain Java's role in Internet. Justify the following features of Java  
1. Secure 2. Architecture neutral 3. Distributed  
**SPPU : May-19, Marks 6**
4. Distinguish between C and Java.
5. Distinguish between C++ and Java.

**3.2 Java Program Structure**

The program structure for the Java program is as given in the following figure -



**Documentation section** : The documentation section provides the information about the source program. This section contains the information which is not compiled by the Java. Everything written in this section is written as comment.

**Package section** : It consists of the name of the package by using the keyword **package**. When we use the classes from this package in our program then it is necessary to write the package statement in the beginning of the program.

**Import statement section** : All the required Java API can be imported by the import statement. There are some core packages present in the Java. These packages include the classes and methods required for Java programming. These packages can be imported in the program in order to use the classes and methods of the program.

**Class definition section :** The class definition section contains the definition of the class. This class normally contains the data and the methods manipulating the data.

**Main method class :** This is called the main method class because it contains the main() function. This class can access the methods defined in other classes.

### Review Question

1. *Describe the structure of Java program.*

### 3.3 How to Write a Java Program ?

- Any Java program can be written using simple **text editors** like notepad or wordpad.
- The **file name should be same as the name of the class** used in the corresponding Java program. Note that the name of the program is *Firstprog* and class name is *Firstprog*.
- The **extension** to the file name should be **.java**. Therefore we have saved our first program by the name *Firstprog.java*.

#### Java Program[Firstprog.java]

```
/*  
This is my First Java Program  
*/  
class Firstprog  
{  
public static void main(String args[])  
{  
System.out.println("This is my first java program");  
}  
}
```

- The output of this program can be generated on the command prompt using the commands

```
javac filename.java  
java filename
```

- The sample output is as shown below for the program *Firstprog.java*

```
D:\>javac Firstprog.java
D:\>java Firstprog
This is my first java program
D:\>
```

### Program explanation

In our First Java program, on the first line we have written a comment statement as

```
/*
This is my First Java Program
*/
```

This is a multi-line comment. Even a single line comment like C++ is also allowed in Java. Hence if the statement would be

```
//This is my First Java Program
```

Is perfectly allowed in Java. Then actual program starts with

```
class Firstprog
```

Here *class* is a keyword and *Firstprog* is a variable name of class. Note that the name of the class and name of your Java program **should be the same**. The class definition should be within the curly brackets. Then comes

```
public static void main(String args[])
```

This line is for a function **void main()**. The main is of type *public static void*.

The *public* is a access mode of the *main()* by which the class visibility can be defined. Typically *main* must be declared as *public*.

The parameter which is passed to main is *String args[]*. Here *String* is a class name and *args[]* is an array which receives the command line arguments when the program runs.

```
System.out.println("This is my first java program");
```

To print any message on the console *println* is a method in which the string "This is my first java program" is written. After *println* method, the newline is invoked. That means next output if any is on the new line. This *println* is invoked along with *System.out*. The

*System* is a predefined class and *out* is an output stream which is related to console. The output as shown in above figure itself is a self explanatory. Also remember one fact while writing the Java programs that it is a **case sensitive** programming language like C or C++.

### 3.4 Comments in Java

In Java there are two ways by which the comment can be specified.

**Method 1 :** Using the // the single line comment can be specified.

**Method 2 :** Using /\* and \*/ the multi-line comment can be specified.

### 3.5 Variables

- A variable is an identifier that denotes the storage location.
- Variable is a fundamental unit of storage in Java. The variables are used in combination with identifiers, data types, operators and some value for initialization. The variables must be declared before its use. The **syntax** of variable declaration will be -  

```
data_type name_of_variable [=initialization][,=initialization][,...];
```
- Following are some **rules** for variable declaration -
  - The variable name should not with digits.
  - No special character is allowed in identifier except underscore.
  - There should not be any blank space with the identifier name.
  - The identifier name should not be a keyword.
  - The identifier name should be meaningful.
- For example :

```
int a,b;
char m='a';
byte k=12,p,t=22;
```

### 3.6 Keywords

Keywords are the reserved words which are enlisted as below -

abstract	default	int	super
assert	double	interface	switch
boolean	else	long	this
byte	extends	native	throw
break	final	new	throws
case	for	package	transient

catch	float	private	try
char	goto	protected	void
class	if	public	volatile
const	implements	return	while
continue	import	short	true
do	instanceof	static	false
			null

### 3.7 Data Types and Size

Various data types used in Java are byte, short, int, long, char, float, double and boolean.

#### byte

This is in fact smallest integer type of data type. Its width is of 8-bits with the range – 128 to 127. The variable can be declared as byte type as

```
byte i,j;
```

#### short

This data type is also used for defining the signed numerical variables with a width of 16 -bits and having a range from – 32,768 to 32,767. The variable can be declared as short as

```
short a,b;
```

#### int

This is the most commonly used data type for defining the numerical data. The width of this data type is 32-bit having a range 2,147,483,648 to 2,147,483,647. The declaration can be

```
int p,q;
```

#### long

Sometimes when *int* is not sufficient for declaring some data then *long* is used. The range of *long* is really very long and it is – 9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. The declaration can be

```
long x,y;
```

#### float

To represent the real number(i.e. number that may have decimal point) float data type can be used. The width is 32-bit and range of this data type is 1.4e – 045 to 3.4e + 038.

**double**

To represent the real numbers of large range the double data type is used. Its width is 64-bit having the range 4.9e – 324 to 1.8e + 308.

**char**

This data type is used to represent the character type of data. The width of this data type is 16-bit and its range is 0 to 65,536.

Let us see one simple Java program which makes use of various data types.

**boolean**

Boolean is a simple data type which denotes a value to be either true or false.

**Java program**

Note that for displaying the value of variable, in the **println** statement the variable is appended to the message using + [In C we use comma in printf statement but in Java + is used for this purpose].

**Java Program [DatatypeDemo.java]**

```
/*
This program introduces use of various data type
in the program
*/
class DatatypeDemo
{
    public static void main(String args[])
    {
        byte a=10;
        short b=10*128;
        int c=10000* 128;
        long d=10000*1000*128;
        double e=99.9998;
        char f='a';
        boolean g=true;
        boolean h=false;
    }
}
dynamic initialization
System.out.println("The value of a=: "+a);
System.out.println("The value of b=: "+b);
System.out.println("The value of c=: "+c);
System.out.println("The value of d=: "+d);
System.out.println("The value of e=: "+e);
System.out.println("The value of f=: "+f);
f++;
System.out.println("The value of f after increment=: "+f);
```

```

System.out.println("The value of g=: "+g);
System.out.println("The value of h=: "+h);
}
}

```

**Output**

```

The value of a=: 10
The value of b=: 1280
The value of c=: 1280000
The value of d=: 1280000000
The value of e=: 99.9998
The value of f=: a
The value of f after increment=: b
The value of g=: true
The value of h=: false

```

**Dynamic initialization**

Java is a flexible programming language which allows the dynamic initialization of variables. In other words, at the time of declaration one can initialize the variables (*note that in the above program we have done dynamic initialization*). In Java we can declare the variable at any place before it is used.

**3.8 Operators**

Various operators that are used in Java are enlisted in following table -

Type	Operator	Meaning	Example
Arithmetic	+	Addition or unary plus	c=a+b
	-	Subtraction or unary minus	d= -a
	*	Multiplication	c=a*b
	/	Division	c=a/b
	%	Mod	c=a%b
Relational	<	Less than	a<4
	>	Greater than	b>10
	<=	Less than equal to	b<=10
	>=	Greater than equal to	a>=5

	<b>==</b>	Equal to	x==100
	<b>!=</b>	Not equal to	m!=8
<b>Logical</b>	<b>&amp;&amp;</b>	And operator	0&&1
	<b>  </b>	Or operator	0  1
<b>Assignment</b>	<b>=</b>	is assigned to	a=5
<b>Increment</b>	<b>++</b>	Increment by one	++i or i++
<b>Decrement</b>	<b>--</b>	Decrement by one	-- k or k--

### Arithmetic operators

The arithmetic operators are used to perform basic arithmetic operations. The operands used for these operators must be of numeric type. The Boolean type operands can not be used with arithmetic operators.

### Java Program

```
///////////////////////////////
//Program for demonstrating arithmetic operators
/////////////////////////////
class ArithOperDemo
{
    public static void main(String[] args)
    {
        System.out.println("\n Performing arithmetic operations ");
        int a=10,b=20,c;
        System.out.println("a= "+a);
        System.out.println("b= "+b);
        c=a+b;
        System.out.println("\n Addition of two numbers is "+c);
        c=a-b;
        System.out.println("\n Subtraction of two numbers is "+c);
        c=a*b;
        System.out.println("\n Multiplication of two numbers is "+c);
        c=a/b;
        System.out.println("\n Division of two numbers is "+c);
    }
}
```

**Output**

Performing arithmetic operations

a = 10

b = 20

Addition of two numbers is 30

Subtraction of two numbers is -10

Multiplication of two numbers is 200

Division of two numbers is 0

**Relational operators**

The relational operators typically used to denote some condition. These operators establish the relation among the operators. The <, > <=, >= are the relational operators. The result of these operators is a Boolean value.

**Java Program**

```
////////////////////////////////////////////////////////////////
//Program for demonstrating relational operators
////////////////////////////////////////////////////////////////
import java.io.*;
import java.lang.*;
import java.util.*;
public class RelOper
{
    public static void main(String args[])
    {
        int a,b,c;

        a=10;
        b=20;
        if(a>b)
        {
            System.out.println("a is largest");
        }
        else
        {
            System.out.println("b is largest");
        }
    }
}
```

## Logical Operators

The logical operators are used to combine two operators. These two operands are Boolean operators.

### Java Program

```
///////////////////////////////
//Program for demonstrating logical operators
///////////////////////////////

import java.io.*;
import java.lang.*;
public class LogicalOper
{
    public static void main(String args[])throws IOException
    {
        boolean oper1,oper2;
        oper1=true;
        oper2=false;
        boolean ans1,ans2;
        ans1=oper1&oper2;
        ans2=oper1|oper2;
        System.out.println("The oper1 is: "+oper1);
        System.out.println("The oper2 is: "+oper2);
        System.out.println("The oper1&oper2 is: "+ans1);
        System.out.println("The oper1|oper2 is: "+ans2);
    }
}
```

### Output

```
The oper1 is: true
The oper2 is: false
The oper1&oper2 is: false
The oper1|oper2 is: true
```

## Special operators

- There are two special operators used in Java-**instanceof** and **dot** operators.
- For determining whether the object belongs to particular class or not- an **instanceof** operator is used. For example -
- Ganga instance of river if the object `ganga` is an object of class `river` then it returns true otherwise false.
- The **dot** operator is used to access the instance variable and methods of class objects.

For example -

```
Customer.name //for accessing the name of the customer
Customer.ID   //for accessing the ID of the customer
```

### Conditional Operator

The conditional operator is “?” The syntax of conditional operator is

```
Condition?expression1:expression2
```

Where expression1 denotes the true condition and expression2 denotes false condition.

For example :

```
a>b?true:false
```

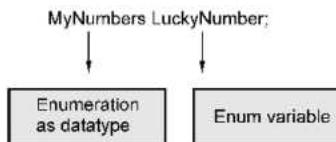
This means that if a is greater than b then the expression will return the value true otherwise it will return false.

### Review Questions

1. Explain various operators used in Java.
2. What special operators in Java ?

### 3.9 User Defined Data Types

- The Java provides the facility of user defined data type by using **enum**.
- An **enum** type is a special data type that enables for a variable to be a set of predefined constants. The variable must be equal to one of the values that have been predefined for it. In Java the enumeration is created using the **enum** keyword.
- The declaration of enum variable is much more similar to the primitive data type.



- Following program shows the use of enum keyword in Java

#### EnumDemo.Java

```
enum MyNumbers
{
    One,Two,Three,Four,Five,Six,Seven,Eight,Nine,Ten
}

public class EnumDemo
{
```

```
public static void main(String args[])
{
    MyNumbers LuckyNumber;

    LuckyNumber = MyNumbers.One;
    System.out.println("My Lucky Number is: " +LuckyNumber);
    System.out.println();
}
```

**Output**

```
D:\>javac EnumDemo.java
D:\>java EnumDemo
My Lucky Number is: One
```

**Benefits of enum**

1. Enum is said to be **typesafe**. That means you can not assign anything else other than predefined enum constant to an enum variable.
2. The enum has its **own namespace**.
3. **Adding new constant** to enum is very easy. One can add the constant without breaking existing code.

**3.10 Control Statements**

Programmers can take decisions in their program with the help of control statements. Various control statements that can be used in Java are -

- |                          |                         |
|--------------------------|-------------------------|
| 1. if statement          | 2. if else statement    |
| 3. while statement       | 4. do...while statement |
| 5. switch case statement | 6. for loop             |

Let us discuss each of the control statement in details -

**1. if statement**

The if statement is of two types

- i) **Simple if statement** : The if statement in which only one statement is followed by that is statement.

**Syntax**

```
if (apply some condition)
    statement
```

**For example**

```
if(a>b)
    System.out.println("a is Biig!");
```

**ii) Compound if statement :** If there are more than one statement that can be executed when if condition is true. Then it is called compound if statement. All these executable statements are placed in curly brackets.

**Syntax**

```
if(apply some condition)
{
    statement 1
    statement 2
    .
    .
    .
    statement n
}
```

**2. if...else statement**

The syntax for if...else statement will be -

```
if(condition)
    statement
else
    statement
```

**For example**

```
if(a>b)
    System.out.println("a is big")
else
    System.out.println("b :big brother")
```

The if...else statement can be of compound type even. For example

```
if(raining==true)
{
    System.out.println("I won't go out");
    System.out.println("I will watch T.V. Serial");
    System.out.println("Also will enjoy coffee");
}
else
{
    System.out.println("I will go out");
    System.out.println("And will meet my friend");
```

```
System.out.println("we will go for a movie");
System.out.println("Any how I will enjoy my life");
}
```

**if...else if statement**

The syntax of if...else if statement is

```
if(is condition true?)
    statement
else if(another condition)
    statement
else if(another condition)
    statement
else
    statement
```

**For example**

```
if(age==1)
    System.out.println("You are an infant");
else if(age==10)
    System.out.println("You are a kid");
else if(age==20)
    System.out.println("You are grown up now");
else
    System.out.println("You are an adult");
```

Let us now see Java programs using this type of control statement -

**Java Program [ifelsedemo.java]**

```
/*
This program illustrates the use of
if else statement
*/
class ifelsedemo
{
public static void main(String[] args)
{
int x=111,y=120,z=30;
if(x>y)
{
if(x>z)
    System.out.println("The x is greatest");
else
    System.out.println("The z is greatest");
}
```

```
else
{
    if(y>z)
        System.out.println("The y is greatest");
    else
        System.out.println("The z is greatest");
}
}
}
```

**Output**

```
The y is greatest
```

**Java Program [ifdemo.java]**

```
/*This program illustrates the use of
if...else if statement
*/
class ifdemo
{
    public static void main(String args[])
    {
        int basic=20000;
        double gross,bonus;
        if(basic<10000)
        {
            bonus=0.75*basic;
            gross=basic+bonus;
            System.out.println("Your Salary is = "+gross);
        }
        else if(basic> 10000&&basic<=20000)
        {
            bonus=0.50*basic;
            gross=basic+bonus;
            System.out.println("Your Salary is = "+gross);
        }
        else if(basic>20000&&basic<=50000)
        {
            bonus=0.25*basic;
            gross=basic+bonus;
            System.out.println("Your Salary is = "+gross);
        }
        else
        {
            bonus=0.0;
            gross=basic+bonus;
            System.out.println("Your Salary is = "+gross);
        }
    }
}
```

```
}
```

```
}
```

```
}
```

```
Your Salary is= 30000.0
```

**Output****3. while statement**

This is another form of while statement which is used to have iteration of the statement for the any number of times. The syntax is

```
while(condition)
{
    statement 1;
    statement 2;
    statement 3;
    ...
    statement n;
}
```

**For example**

```
int count=1;
while(count<=5)
{
    System.out.println("I am on line number "+count);
    count++;
}
```

Let us see a simple Java program which makes the use of while construct.

**Java Program [whiledemo.java]**

```
/*
This is java program which illustrates
while statement
*/
class whiledemo
{
    public static void main(String args[])
    {
        int count=1,i=0;
        while(count <= 5)
        {
            i=i+1;
            System.out.println("The value of i= "+i);
            count++;
        }
    }
}
```

**Output**

```
The value of i= 1
The value of i= 2
The value of i= 3
The value of i= 4
The value of i= 5
```

**4. do... while statement**

- This is similar to while statement but the only **difference** between the two is that in case of **do...while** statement the statements inside the **do...while** must be executed at least once.
- This means that the statement inside the do...while body gets **executed first** and then the **while** condition is checked for next execution of the statement, whereas in the **while** statement first of all the condition given in the while is checked first and then the statements inside the **while body** get executed when the condition is true.

- **Syntax**

```
do
{
    statement 1;
    statement 2;
    ...
    statement n;
}while(condition);
```

**For example**

```
int count=1;
do
{
    System.out.println("I am on the first line of do-while");
    System.out.println("I am on the second line of do-while");
    System.out.println("I am on the third line of do-while");
    System.out.println("I am on the forth line of do-while");
    System.out.println("I am on the fifth line of do-while");
    count++;
}while(count<=5);
```

**Java Program [dowhiledemo.java]**

```
/*
This is java program which illustrates
do...while statement
*/
```

```
class dowhiledemo
{
    public static void main(String args[])
    {
        int count=1,i=0;
        do
        {
            i=i+1;
            System.out.println("The value of i= "+i);
            count++;
        }while(count<=5);
    }
}
```

**Output**

```
The value of i= 1
The value of i= 2
The value of i= 3
The value of i= 4
The value of i= 5
```

**5. Switch statement**

Switch statement allows you to select any desired condition from multiple cases.

Here is a sample program which makes use of switch case statement -

**Java Program [switchcasedemo.java]**

```
/*
This is a sample java program for explaining
Use of switch case
*/
class switchcasedemo
{
    public static void main(String args[])
    throws java.io.IOException
    {
        char choice;
        System.out.println("\nProgram for switch case demo");
        System.out.println("Main Menu");
        System.out.println("1. A");
        System.out.println("2. B");
        System.out.println("3. C");
        System.out.println("4. None of the above");
        System.out.println("Enter your choice");
        choice=(char)System.in.read();
```

```

switch(choice)
{
    case '1':System.out.println("You have selected A");
        break;
    case '2':System.out.println("You have selected B");
        break;
    case '3':System.out.println("You have selected C");
        break;
    default:System.out.println("None of the above choices made");
}
}
}
}

```

**Output**

Program for switch case demo  
Main Menu  
1. A  
2. B  
3. C  
4. None of the above  
Enter your choice  
2  
You have selected B

Note that in above program we have written **main()** in somewhat different manner as -

```
public static void main(String args[])
throws java.io.IOException
```

This is **IOException** which must be thrown for the statement **System.in.read()**. Just be patient, we will discuss the concept of Exception shortly! The **System.in.read()** is required for reading the input from the console[**note that it is parallel to scanf statement in C**]. Thus using **System.in.read()** user can enter his choice. Also note that whenever **System.in.read()** is used it is necessary to write the main with **IOException** in order to handle the input/output error.

**6. for loop**

**for** is a keyword used to apply loops in the program. Like other control statements for loop can be categorized in simple for loop and compound for loop.

**Simple for loop :**

```
for (statement 1;statement 2;statement 3)
execute this statement;
```

**Compound for loop :**

```
for(statement 1;statement 2; statement 3)
{
    execute this statement;
    execute this statement;
    execute this statement;
    that's all;
}
```

Here

Statement 1 is always for **initialization** of conditional variables,

Statement 2 is always for **terminating** condition of the for loop,

Statement 3 is for representing the **stepping** for the next condition.

**For example :**

```
for(int i=1;i<=5;i++)
{
    System.out.println("Java is an interesting language");
    System.out.println("Java is a wonderful language");
    System.out.println("And simplicity is its beauty");
}
```

Let us see a simple Java program which makes use of for loop.

**Java Program [forloop.java]**

```
/*
This program shows the use of for loop
*/
class forloop
{
    public static void main(String args[])
    {
        for(int i=0;i<=5;i++)
            System.out.println("The value of i: "+i);
    }
}
```

**Output**

```
The value of i: 0
The value of i: 1
The value of i: 2
The value of i: 3
The value of i: 4
The value of i: 5
```

**Example 3.10.1** Write a Java program to find factorial of a given number.

**Solution :**

```
class FactorialProgram
{
    public static void main(String args[])
    {
        int i,fact=1;
        int number=5;//The factorial of 5 is calculated
        i=1;
        while(i<number)
        {
            fact=fact*i;
            i++;
        }
        System.out.println("Factorial of "+number+" is: "+fact);
    }
}
```

## Output

Factorial of 5 is: 120

**Example 3.10.2** Write a Java program to display following pattern.

10

**Solution :**

```
class TriangleDisplay
{
    public static void main(String[] args)
    {
        int i,j,k;
        for(i=1; i<=5; i++)
        {
            for(j=4; j>=i; j--)
            {
                System.out.print(" ");
            }
            for(k=1; k<=(2*i-1); k++)
            {
                System.out.print("*");
            }
            System.out.println();
        }
    }
}
```

```
    {
        System.out.print("*");
    }
    System.out.println("");
}
}
```

**Output**

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

**Example 3.10.3** Write a Java program to display following number pattern.

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

**Solution :**

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

**Output**

```
D:\>javac NumberPatternDisplay.java
D:\>java NumberPatternDisplay
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
D:\>
```

**3.11 Jump Statements**

There are two types of jump statements used in Java - break and continue.

**1. break statement**

When a break statement is encountered in a loop then the loop is terminated and the control resumes the next statement following the loop. For example

**Java Program**

```
public class breakDemo
{
    public static void main(String[] args)
    {
        for (int i = 1; i <= 10; i++)
        {
            if (i == 5)
            {
                break; // terminate loop if i is 5
            }
            System.out.print(i + " ");
        }
        System.out.println("Loop is terminated using break!!");
    }
}
```

**Output**

```
1 2 3 4 Loop is terminated using break!!
```

**Program explanation :** In above program, the for loop is executed for i = 1 to i = 10. But when i value reaches to 5, the break statement is encountered. Due to this, the control terminates the for loop and the **System.out.println** statement outside the for loop will be executed. Hence is the output.

## 2. Labeled break

Sometimes the control needs to be transferred to a specific instruction. This can be done using the statements like **goto**. Java provides the support for jump statement using labeled break.

The syntax for labeled break is

```
break label;
```

*label* is the name of the block which need to be skipped. Following program demonstrates the use of labeled break.

### Java Program

```
class breakLblDemo
{
    public static void main(String args[])
    {
        int i=5;
        first:
        {
            second:
            {
                third:
                {
                    System.out.println("Before the break.");
                    if(i==5)
                        break second; // break out of second block
                    System.out.println("Inside third block.");
                }
                System.out.println("Inside second block.");
            }
            System.out.println("Inside first block.");
        }
    }
}
```

### Output

```
Before the break.
Inside first block.
```

**Program explanation :** In above program, we have created three blocks, by labeling them as first, second and third. Inside the third block the labeled break statement is encountered to break the second block. Hence the control skips execution of second and third block and reaches to the first block. Hence is the output.

### 3. Continue

When a continue statement is encountered inside the body of a loop, remaining statements are skipped and loop proceeds with the next iteration. For example

#### Java Program

```
public class continueDemo
{
    public static void main(String[] args)
    {
        for (int i = 1; i <= 10; i++)
        {
            if (i == 5)
            {
                continue;
            }
            System.out.print(i + " ");
        }
    }
}
```

#### Output

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

**Program explanation :** In above program, when the value of  $i = 5$  then it is skipped and loop proceeds with further remaining values of  $i$ . Hence is the output.

**Note that :** You cannot use break to transfer control to a block of code that does not enclose the break statement.

### 4. Labeled continue

The labeled loop can be continued using

```
continue label;
```

Following java program, illustrates the labeled continue statement.

#### Java Program

```
class continueLblDemo
{
    public static void main(String args[])
}
```

```

{
    Label1: for(int i=0;i<10;i++)
    {
        if(i%2==0)
        {
            System.out.println("Even value: = "+i);
            continue Label1;
        }
        System.out.println("Odd value: = "+i);
    }
}

```

**Output**

```

Even value: = 0
Odd value: = 1
Even value: = 2
Odd value: = 3
Even value: = 4
Odd value: = 5
Even value: = 6
Odd value: = 7
Even value: = 8
Odd value: = 9

```

**Difference between break and continue**

Sr. No.	<b>break</b>	<b>continue</b>
1.	The break is used to terminate the execute the loop.	The continue statement is not used to terminate the execution
2.	It breaks iteration.	It skips the iteration.
3.	Break is used in loops and in switch.	Continue is used only in loop.
4.	When this break is executed, the control will come out of the loop.	When continue is executed the control will not come out of the loop, in fact it will jump to next iteration of loop.

## 5. Example :

```
public class breakdemo
{
    public static void main(String[] args)
    {
        for (int i = 1; i <= 20; i++)
        {
            if (i % 2 == 0)
            {
                break;
            }
            // skip the even numbers
            System.out.println(i + " ");
        }
        //will display only odd numbers
    }
}
```

**Output**

1

## Example :

```
public class continuedemo
{
    public static void main(String[] args)
    {
        for (int i = 1; i <= 20; i++)
        {
            if (i % 2 == 0)
            {
                continue; // skip the even numbers
            }
            //will display only odd numbers
            System.out.println(i + " ");
        }
    }
}
```

**Output**

```
1
3
5
7
9
11
13
15
17
```

**Review Question**

- What is the difference between break and continue statement ? Explain it with examples.

**3.12 Arrays****SPPU : Dec.-17, Nov.-18, Marks 6**

- Definition :** Array is a collection of similar type of elements.
- Using arrays the elements that are having the same data type can be grouped together.
- If we use bunch of variables instead of arrays, then we have to use large number of variables.
- Declaring and handling large number of variables is very inconvenient for the developers.

- There are two types of arrays -

1. One dimensional arrays
2. Two dimensional arrays

### 3.12.1 One Dimensional Array

- Array is a collection of similar type of elements. Thus grouping of similar kind of elements is possible using arrays.
- Typically arrays are written along with the size of them.
- The **syntax** of declaring array is -

```
data_type array_name[size];
```

and to allocate the memory -

```
array_name=new data_type[size];
```

where *array\_name* represent name of the array, **new** is a keyword used to allocate the memory for arrays, *data\_type* specifies the data type of array elements and *size* represent the size of an array. For example :

```
a=new int[10];
```

[ Note that in C/C++ this declaration is as good as int a[10] ]

- After this declaration the array **a** will be created as follows

**Array a[10]**

0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---

- That means after the above mentioned declaration of array, all the elements of array will get initialized to zero. Note that, always, while declaring the array in Java, we make use of the keyword **new** and thus we actually make use of dynamic memory allocation.
- Therefore arrays are **allocated dynamically** in Java. Let us discuss on simple program based on arrays.

#### Java Program [SampleArray.Java]

```
/*
This is a Java program which makes use of arrays
*/
class SampleArray
{
public static void main(String args[])
{
int a[];
a=new int[10];
```

```

System.out.println("Storing the numbers in array");
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;
System.out.println("The element at a[5] is: "+a[5]);
}
}

```

**Output**

Storing the numbers in array

The element at a[5] is : 6

**Program explanation :**

In above program,

- 1) We have created an array of 10 elements and stored some numbers in that array using the array index. The array that we have created in above program virtually should look like this -

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

- 2) The **System.out.println** statement is for printing the value present at a[5]. We can modify the above program a little bit and i.e instead of writing

```

int a[];
a=new int[10];

```

these two separate statements we can combine them together and write it as -

```
int a[]=new int[10];
```

That means the declaration and initialization is done simultaneously.

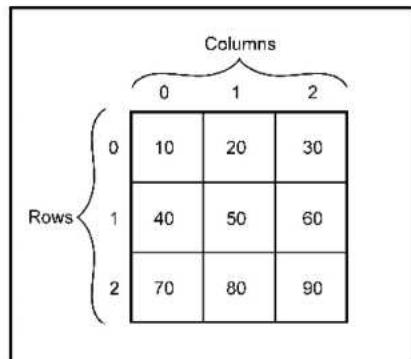
- 3) Another way of initialization of array is

```
int a[] = {1,2,3,4,5,6,7,8,9,10};
```

That means, as many number of elements that are present in the curly brackets, that will be the size of an array. In other words there are total 10 elements in the array and hence size of array will be 10.

### 3.12.2 Two Dimensional Array

- The two dimensional arrays are the arrays in which elements are stored in rows as well as in columns.
- For example



- The two dimensional array can be declared and initialized as follows

#### Syntax

```
Data_type array_name[][] = new data_type[size][size];
```

#### Example

```
int a[][] = new int[3][3];
```

Let us straight away write a Java program that is using two dimensional arrays -

#### Java Program [Sample2DArray.java]

```
/*
This is a Java program which makes use of 2D arrays
*/
class Sample2DArray
{
public static void main(String args[])
{
    int a[][]=new int[3][3];
    int k=0;
    System.out.println("\tStoring the numbers in array");
    for(int i=0;i<3;i++)
    {
```

```

for(int j=0;j<3;j++)
{
    a[i][j]=k+10;
    k=k+10;
}
}

System.out.println("You have stored...");
for(int i=0;i<3;i++)
{
    for(int j=0;j<3;j++)
    {
        System.out.print(" " + a[i][j]);
    }
    System.out.println();
}
}
}
}

```

**Output**

Storing the numbers in array  
 You have stored...  
 10 20 30  
 40 50 60  
 70 80 90

The typical application of two dimensional arrays is performing matrix operations.

**Example 3.12.1** Write a program in Java to perform the addition of two matrices (multidimensional arrays) and set the diagonal elements of resultant matrix to 0.

**SPPU : Dec.-17, Marks 6**

**Solution :**

```

class Matrix
{
    public static void main(String args[])
    throws java.io.IOException
    {
        int a[][]={{
            {10,20,30},
            {40,50,60},
            {70,80,90}
        }};
        int b[][]={{
            {1,2,3},
            {4,5,6},
            {7,8,9}
        }};
    }
}

```

```

int c[][]=new int [3][3];
for(int i=0;i<3;i++)
{
    for(int j=0;j<3;j++)
    {
        c[i][j]=a[i][j]+b[i][j];
    }
}
System.out.println("The Addition is...");
for(int i=0;i<3;i++)
{
    for(int j=0;j<3;j++)
    {
        System.out.print(" "+c[i][j]);
    }
    System.out.println();
}
for(int i=0;i<3;i++)//setting diagonal elements to 0
{
    for(int j=0;j<3;j++)
    {
        if(i==j)
            c[i][j]=0;
    }
}
}
}

```

**Example 3.12.2** Write a program in Java which reads a matrix of size 3 by 3 and performs the addition of elements in each row and each column. The program prints the each row and column additions.

SPPU : Nov.-18, Marks 6

**Solution :**

```

public class Test
{
    public static void main(String[] args)
    {
        int rows, cols, Row_Sum, Col_Sum;

        int a[][] = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
        };
    }
}

```

```
//Calculates number of rows and columns present in given matrix
rows = a.length;
cols = a[0].length;

//Calculates sum of each row of given matrix
for(int i = 0; i < rows; i++)
{
    Row_Sum = 0;
    for(int j = 0; j < cols; j++)
    {
        Row_Sum = Row_Sum + a[i][j];
    }
    System.out.println("Sum of " + (i+1) + " row: " + Row_Sum);
}
System.out.println("-----");

//Calculates sum of each column of given matrix
for(int i = 0; i < cols; i++)
{
    Col_Sum = 0;
    for(int j = 0; j < rows; j++)
    {
        Col_Sum = Col_Sum + a[j][i];
    }
    System.out.println("Sum of " + (i+1) + " column: " + Col_Sum);
}
```

**Output**

Sum of 1 row: 6

Sum of 2 row: 15

Sum of 3 row: 24

-----

Sum of 1 column: 12

Sum of 2 column: 15

Sum of 3 column: 18

**3.13 String Handling****SPPU : May-19, Marks 6**

String is a collection of characters. In Java **String** defines the object. We can make use of **String** to denote the collection of characters. Let us look at a sample program in which the string object is used in order to handle some text.

**Java Program [StringDemo.Java]**

```
/*This is a Java program which makes use of strings
*/
class StringDemo
{
public static void main(String args[])
{
String s="Hello, How are You?";
System.out.println(s);
}
}
```

**Output**

```
Hello, How are You?
```

**String Literal**

In Java String can be represented as a sequence of characters enclosed within the double quotes.

**For example**

```
String s="Hello,How are you?"
```

We can initialize the string using the empty string literal. For example

```
String mystr=""
```

Similarly the escape sequence can be used to denote a particular character in the string literal. For example we can use back slash followed by double quote to print the double quote character.

```
"\\" "
```

Aleternatively, String can be denoted as the array of characters. For example :

```
char str[]={'P','R','O','G','R','A','M'};
```

Now we can have a variable `s` which can then be initialized with the string "PROGRAM" as follows -

```
String s =new String(str);
```

**3.13.1 Finding Length of String**

We can find the length of a given string using the method `length()`. Following program shows the use of `length()` method for strings

**Java Program [Str\_lengthDemo.java]**

```
class Str_lengthDemo
{
    public static void main(String[] args)
    {
        char str[]={‘P’,‘R’,‘O’,‘G’,‘R’,‘A’,‘M’};
        String S=new String(str);
        System.out.println(“The string S is ”+S);
        System.out.println(“The length of string is ”+S.length());
        System.out.print(“The string in character array is ”);
        for(int i=0;i<S.length();i++)
            System.out.print(str[i]);
    }
}
```

**Output**

```
D:\>javac Str_lengthDemo.java
D:\>java Str_lengthDemo
The string S is PROGRAM
The length of string is 7
The string in character array is PROGRAM
```

**String in reverse direction**

There is no direct method for reversing the string but we can display it in reverse direction. Following is the Java program for the same

**Java Program[Str\_reverse.java]**

```
*****
Printing the String in reverse order
*****
class Str_reverse
{
    public static void main(String[] args)
    {
        char str[]={‘S’,‘T’,‘R’,‘I’,‘N’,‘G’};
        String S=new String(str);
        System.out.println(“The string S is ”+S);
        System.out.print(“The string written in Reverse order ”);
        for(int i=S.length()-1;i>=0;i--)
            System.out.print(str[i]);
    }
}
```

**Output**

```
The string S is STRING
The string written in Reverse order GNIRTS
```

### 3.13.2 Concatenation

We can concatenate two strings using + operator which is illustrated by following code -

#### Java Program [Test.java]

```
class Test
{
    public static void main(String[] args)
    {
        String fruit=new String("mango");
        System.out.println("I like "+fruit +" very much");
    }
}
```

#### Output

```
I like mango very much
```

We can make use of a method **concat()** for concatenating two strings. The above program is slightly modified for the use of concat() -

#### Java Program

```
class Test
{
    public static void main(String[] args)
    {
        String str=new String("I like ");
        String fruit=new String("mango very much");
        System.out.println(str.concat(fruit));
    }
}
```

#### Output

```
I like mango very much
```

### 3.13.3 Character Extraction

As we know that the string is a collection of characters. **String** class provides the facility to extract the character from the string object. There is a method called **charAt(index)** with the help of which we can extract the character denoted by some index in the array.

**For example :**

```
String fruit=new String("mango");
char ch;
```

```
ch=fruit.charAt(2);
System.out.println(ch);
```

The output of above code fragment will be n because at the index position 2 of string object fruit the character n is present.

### 3.13.4 String Comparison

Sometimes we need to know whether two strings are equal or not. We use method **equals()** for that purpose. This method is of Boolean type. That is, if two strings are equal then it returns true otherwise it returns false.

The syntax is -

```
boolean equals(String Str);
```

Let us see one illustrative program -

#### Java Program [StringCompareDemo.Java]

```
class StringCompareDemo
{
    public static void main(String[] args)
    {
        String str1=new String("INDIA");
        String str2=new String("india");
        if(str1.equals(str2)==true)
            System.out.println("\n The two strings are equal");
        else
            System.out.println("\n The two strings are not equal");
    }
}
```

#### Output

```
D:\>javac StringCompareDemo.java
```

```
D:\>java StringCompareDemo
```

```
The two strings are not equal
```

The above program will generate the message "The two strings are not equal" if str1 and str2 are not equal but if we write same str1 and str2 then naturally the output will be "The two strings are equal". This string comparison is case sensitive. But if we want to compare the two strings without caring for their case differences then we can use the method **equalsIgnoreCase()**.

### 3.13.5 Searching for the Substring

We can look for the desired substring from the given string using a method **substring()**. The syntax of method **substring()** is as follows -

```
String substring(int start_index,int end_index)
```

#### Java Program [SubstringDemo.java]

```
class SubstringDemo
{
    public static void main(String[] args)
    {
        String str1=new String("I love my country very much");
        System.out.println("\n One substring from the given sentence is:"+str1.substring(2,6));
        System.out.println("\n And another substring
is:"+str1.substring(18));
    }
}
```

#### Output

```
D:\>javac SubstringDemo.java
```

```
D:\>java SubstringDemo
```

```
One substring from the given sentence is:love
And another substring is:very much
```

### 3.13.6 Replacing the Character from String

We can replace the character by some desired character. For example

#### Java Program [replaceDemo.java]

```
class replaceDemo
{
    public static void main(String[] args)
    {
        String str=new String("Nisha is Indian ");
        String s=str.replace('i','a');
        System.out.println(s);
    }
}
```

#### Output

```
D:\>javac replaceDemo.java
```

```
D:\>java replaceDemo
```

```
Nasha as Indaan
```

### 3.13.7 Upper Case and Lower Case

We can convert the given string to either upper case or lower case using the methods `toUpperCase()` and `toLowerCase()`. Following program demonstrates the handling of the case of the string -

#### Java Program [CaseDemo.Java]

```
class caseDemo
{
    public static void main(String[] args)
    {
        String str=new String("Nisha is Indian ");
        System.out.println("\n The original string is: "+str);
        String str_upper= str.toUpperCase();
        System.out.println("\n The Upper case string is: "+str_upper);
        String str_lower= str.toLowerCase();
        System.out.println("\n The Lower case string is: "+str_lower);
    }
}
```

#### Output

```
D:\>javac caseDemo.java
```

```
D:\>java caseDemo
```

The original string is: Nisha is Indian

The Upper case string is: NISHA IS INDIAN

The Lower case string is: nisha is Indian

#### Review Questions

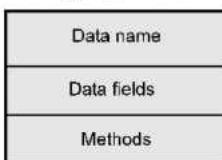
1. Write a program for concatenating two strings.
2. List and explain any five string functions with an example program.
3. What are strings in Java ? Explain following operations of class strings in Java with example -  
i) To find length of string ii) To compare two strings iii) Extraction of character from string.

SPPU : May-19, Marks 6

### 3.14 Classes and Methods

Each class is a collection of data and the functions that manipulate the data. The data components of the class are called data fields and the function components of the class are called member functions or methods.

Thus the state of an object is represented using **data fields** (data members) and behaviour of an object is represented by set of **methods** (member functions). The class template can be represented by following Fig. 3.14.1.



**Fig. 3.14.1 Class template**

The **Java Class** can written as follows

```
class Customer {
    int ID;
    int Age;
    String Name; } } } } }
```

Data Field

```
Customer() { } } } }
```

Constructor

```
Customer(int ID) { } } } }
```

Method

Encapsulation means binding of data and method together in a single entity called class. Thus a class is used to achieve an **encapsulation** property.

This class is not having the main function. The class that contains main function is called **main class**.

Following is a simple Java program that illustrates the use of class

```
///////////////////////////////
//Program for demonstrating the concept of class
/////////////////////////////
import java.io.*;
import java.lang.*;
import java.math.*;
public class MyClass
{
    String name;
    int roll;
    double marks; }
```

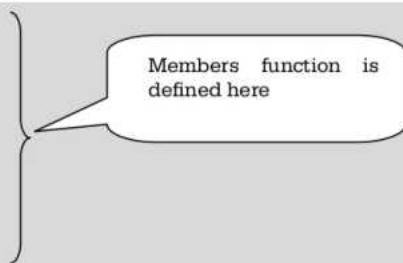
Data members

```

public void display(String n,int r,double m)
{
    System.out.println();
    System.out.println("Name: "+n);
    System.out.println("Roll number: "+r);
    System.out.println("Marks: "+m);
}

public static void main(String args[])
{
    int a,b;
    MyClass obj1=new MyClass();
    MyClass obj2=new MyClass();
    MyClass obj3=new MyClass();
    obj1.display("Amar",10,76.65);
    obj2.display("Akbar",20,87.33);
    obj3.display("Anthony",30,96.07);
}
}

```

**Output**

Name: Amar  
Roll number: 10  
Marks: 76.65

Name: Akbar  
Roll number: 20  
Marks: 87.33

Name: Anthony  
Roll number: 30  
Marks: 96.07

**Program explanation**

In above program,

1. We have declared one simple class. This class displays small database of the students.
2. The data members of this class are **name**, **roll** and **marks**. There is one member function named **display** which is for displaying the data members.
3. In the **main** function we have created three objects **obj1,obj2** and **obj3**. These objects represent the real world entities. These entities are actually the instances of the class.

4. The class representation of the above program can be as follows -

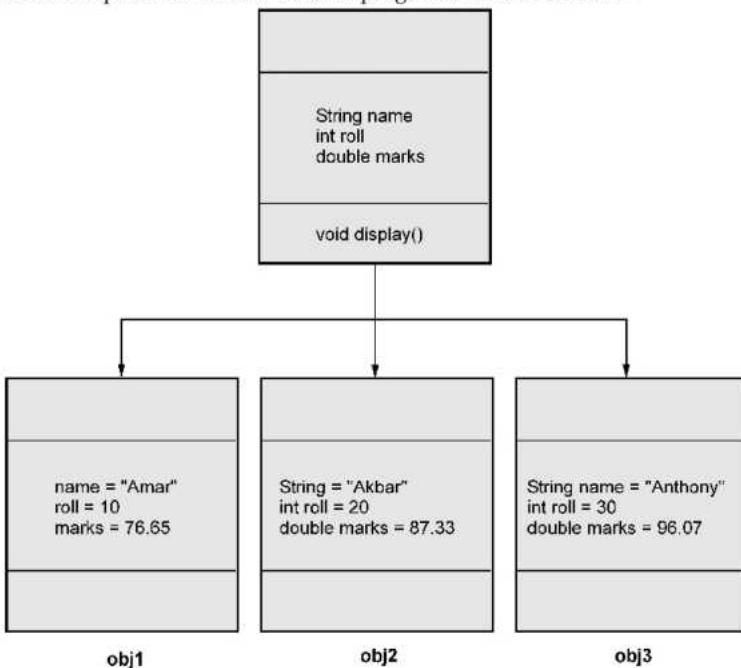


Fig. 3.14.2

### Review Question

1. How data and methods are organized in an object oriented program ?

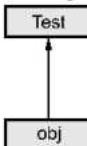
### 3.15 Objects

**Definition :** Objects are nothing but the instances of the class. Hence a block of memory gets allocated for these instance variables.

- For creating objects in Java the operator **new** is used.

```
Test obj; // Declaration of object obj.  
obj = new Test(); // obj gets instantiated for class Test
```

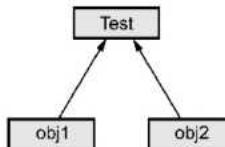
- We instantiate one object **obj**. It can be represented graphically as -



- Now if we create two objects then it can be

```
Test obj1, obj2;
obj1 = new Test();
obj2 = new Test();
```

- It can be graphically as shown below -



- The declaration of objects is similar to declaration of variable. When we declare class name followed by object name then corresponding objects get created.
- Let us see the Java program, which makes use of classes and objects -

### Java Program [Rectangle.java]

```
/*This is a Java program which shows the
use of class in the program
*/
class Rectangle
{
int height;
int width;
void area()
{
int result=height*width;
System.out.println("The area is "+result);
}
}
class classDemo
{
public static void main(String args[])
{
Rectangle obj=new Rectangle();
obj.height=10;//setting the attribute values
obj.width=20;//from outside of class
obj.area();//using object method of class is called
}
}
```

**Class with attributes and methods defined within it.**

**Another class in which main () function is written.**

### Output

The area is 200

### Program explanation :

In the above program,

1. We have used two classes one class is **classDemo** which is our usual one in which the **main** function is defined and the other class is **Rectangle**.
2. In **Rectangle** class we have used height and width as attributes and one method **area()** for calculating area of rectangle.
3. In the main function we have declared an object of a class as

```
Rectangle obj=new Rectangle();
```

And now using **obj** we have assigned the values to the attributes of a class.

1. The operator **new** is used to create an object. The objects access the data fields and methods of the class using the **dot operator**. This operator is also known as the **object member access operator**. Thus data field *height* and *width* are called as **instance variables**.
2. And the method, **area** is referred as **instance method**. The object on which the instance method is invoked is known as **calling object**.

### 3.16 Adding Methods to Class

Methods are nothing but the functions defined by the particular class.

```
class classname
{
    declaration of member variables;
    definition of method
    {
        ...
        ...
    }
}
```

#### How to add method to a class ?

Following is a simple Java program that illustrates the method inside the class

#### Java Program

```
import java.io.*;
import java.lang.*;
import java.math.*;
public class MyClass
{
    String name;
    int roll;
    double marks;
}
```

Data members

```

public void display(String n,int r,double m)
{
    System.out.println();
    System.out.println("Name: "+n);
    System.out.println("Roll number: "+r);
    System.out.println("Marks: "+m);
}

public static void main(String args[])
{
    int a,b;
    MyClass obj=new MyClass();
    obj.display("Amar",10.76.65);
}
}

```

Members function  
is defined here

### Output

```

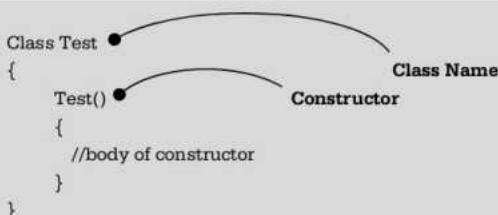
Name: Amar
Roll number: 10
Marks: 76.65

```

### 3.17 Constructor

**SPPU : Dec.-17, May-18, Nov.-18, 19, Marks 8**

- **Definition :** The constructor is a specialized method for **initializing objects**.
- Name of the constructor is same as that of its class name. In other words, the name of the constructor and class name is same.



- Whenever an object of its associated class is created, the constructor is invoked automatically.
- The constructor is called constructor because it creates values for data fields of class.

Following is Java program that makes use of a constructor.

### Java Program[Rectangle1.java]

```

public class Rectangle1 {
    int height;                                Data fields
    int width;
    Rectangle1() {                            Simple constructor
    {
        System.out.println(" Simple constructor: values are initialised... ");
        height=10;
        width=20;
    }
    Rectangle1(int h,int w) {                  Parameterised constructor
    {
        System.out.println(" Parameterised constructor: values are initialised... ");
        height=h;
        width=w;
    }
    void area() {                           method defined
    {
        System.out.println("Now, The function is called... ");
        int result=height*width;
        System.out.println("The area is "+result);
    }
}
class Constr_Demo {
    public static void main(String args[])
    {
        Rectangle1 obj=new Rectangle1();
        obj.area(); //call the to method
        Rectangle1 obj1=new Rectangle1(11,20);
        obj1.area(); //call the to method
    }
}

```

Object created and simple constructor is invoked

Object created and parameterised constructor is invoked

### Output

```

F:\test>javac Rectangle1.java
F:\test>java Constr_Demo
Simple constructor: values are initialised...
Now, The function is called...
The area is 200
Parameterised constructor: values are initialised...
Now, The function is called...
The area is 220

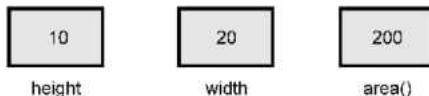
```

Note the method of running the program

### Program explanation

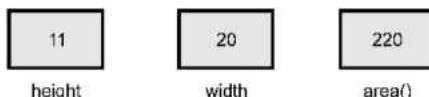
In above program,

1. There are two classes - **Rectangle1** and **Constr\_Demo**.
2. In the **Rectangle1** class, data fields, method and constructor is defined.
3. In the **Constr\_Demo** class the objects are created. When an object **obj** gets created, then the simple constructor **Rectangle1()** gets invoked and the data fields **height** and **width** gets initialised



Hence the **area** function will compute the area =  $10 * 20 = 200$ .

4. When an object **obj1** gets created, then the parameterised constructor **Rectangle1(11, 20)** gets invoked and the data fields **height** and **width** gets initialised.



Hence the **area** function will compute the area =  $11 * 20 = 220$ .

5. The class **Constr\_Demo** is called **main class** because it consists of **main** function.

#### 3.17.1 Special Properties of Constructor

1. Name of constructor must be the same as the name of the class for which it is being used.
2. The constructor must be declared in the **public** mode.
3. The constructor gets invoked automatically when an object gets created.
4. The constructor should not have any return type. Even a **void** data type should not be written for the constructor.
5. The constructor can not be used as a member of union or structure.
6. The constructors can have default arguments.
7. The constructor **can not be inherited**. But the derived class can invoke the constructor of base class.
8. Constructor can make use of **new** or **delete** operators for allocating or releasing memory respectively.

9. Constructor can not be **virtual**.
10. Multiple constructors can be used by the same class.
11. When we declare the constructor explicitly then we must declare the object of that class

### 3.17.2 Types of Constructors

Various types of constructors are -

1. No argument(Default) constructor
2. Parameterized constructor
3. Copy constructor

1. **No argument constructor** : The default constructor is a constructor for which there are no arguments or parameters passed. For example -

#### Java Program

```
public class Test
{
    int a;
    Test() //no argument constructor
    {
        a=0;
    }
}

class MainClass {
    public static void main(String args[])
    {
        Test obj=new Test();
    }
}
```

2. **Parameterized constructor** : The parameterized constructor is a constructor in which the parameters are passed to the constructor function. For example -

#### Java Program

```
public class Test
{
    int a;
    Test(int x)//parameterized constructor
    {
        a=x;
    }
}
```

```
    }  
  
}  
class MainClass {  
    public static void main(String args[])  
    {  
        Test obj=new Test(10);  
    }  
}
```

### 3. Copy constructor

Copy constructor is a kind of constructor which can be initialised using the object of the same class. This object is created already. The use of copy constructor is -

1. To initialize one object from another object of same type.
2. The object can be passed as a parameter to the function.
3. The copy of the object can be returned from the function.

#### Syntax of copy constructor

```
classname (const classname &obj)  
{  
    // body of constructor  
}
```

#### Java Program

Following example shows how copy constructor can be used to initialize another object of the same class.

```
*****  
Program to read complex numbers and to copy it into another using copy constructor  
*****  
class Complex  
{  
  
    private double real, img;  
    public Complex(double r, double i)//parameterized constructor  
    {  
        this.real = r;  
        this.img = i;  
    }  
  
    Complex(Complex c) // copy constructor  
    {  
        System.out.println("Invoking Copy constructor....");  
    }  
}
```

```

real = c.real;
img = c.img;
}
void display()
{
    System.out.println(real + " + " + img + "i");
}
}

public class TestClass
{
    public static void main(String[] args)
    {
        Complex obj1 = new Complex(10,20);
        Complex obj2 = new Complex(obj1);
        Complex obj3 = obj2;
        obj3.display();
    }
}

```

Invoking Copy Constructor

### Output

Invoking Copy constructor....  
10.0 + 20.0i

**Example 3.17.1** Write a program in Java which defines class CONVERSION which converts one unit of length into another using multiplying factor. This class has data members unit\_in, unit\_out and multiplier. When user creates object, constructor accepts value of multiplier and sets this for further conversion of units. The object uses methods to get value of unit\_in and output value of unit\_out and stores these in class variables.

SPPU : Dec.-17, Marks 8

### Solution :

```

//import Scanner as we require it.
import java.util.Scanner;
class Conversion
{
    float unit_in,unit_out;
    int multiplier;
    public Conversion(int m)//constructor defined
    {
        multiplier=m;
    }
    public void convert()
    {

```

```

        System.out.println("Enter Length in meter:");
        Scanner input = new Scanner(System.in);
        unit_in = input.nextFloat();
        unit_out = unit_in*multiplier;
        System.out.println("Length in centimeter = "+unit_out);
    }
}

public class Main
{
    public static void main (String[] args)
    {
        Conversion obj=new Conversion(100);//passing the value of multiplier
        //while creating an object
        obj.convert();
    }
}

```

**Review Questions**

- What is use of constructors ? What are types of constructors in Java ? Give example of each type. SPPU : May-18, Marks 7
- What is a constructor ? Show with example the use and overloading of default, parameterized and copy constructor. SPPU : Nov.-18, Marks 7
- What is a constructor ? What are its different types ? Demonstrate with suitable example the different types of constructors used in Java. SPPU : Nov.-19, Marks 6

**3.18 this keyword**SPPU : Dec.-19, Marks 6

- The **this** keyword is used by a method to refer the object which invoked that method.
- The **this** keyword is used inside the method definition to refer the current object. That means **this** is a reference to the object which is used to invoke it.

Following is a Java program in which the method uses the keyword **this**

**Java Program**

```

///////////////////////////////
//Program for demonstrating the keyword this
///////////////////////////////

import java.io.*;
import java.lang.*;

public class thisDemo
{
    int a,b;

```

```
public void Sum(int a,int b)
{
    this.a=a;
    this.b=b;
}
public void display()
{
    int c=a+b;
    System.out.println("a = "+a);
    System.out.println("b = "+b);
    System.out.println("Sum = "+c);
    System.out.println();
}
public static void main(String args[])
{
    thisDemo obj1=new thisDemo();
    thisDemo obj2=new thisDemo();
    obj1.Sum(10,20);
    obj1.display();
    obj2.Sum(40,50);
    obj2.display();
}
```

**Output**

```
a = 10
b = 20
Sum = 30

a = 40
b = 50
Sum = 90
```

**Program explanation**

In above program,

1. We have created two objects for the class - **obj1** and **obj2**.
2. The value of a is assigned to be 10 and value of b as 20 for the object **obj1** using the keyword **this**.
3. Similarly the **obj2** posses the values of a and b as 40 and 50 respectively. Note that the keyword **this** is useful to refer the current object.

int a=10; int b=20;

this.a=a;  
this.b=b;  
this statement makes the current value of a(which belongs to **obj1**) as 10.  
Similarly the value of b is initialised with 20

int a=40; int b=50;

this.a=a;  
this.b=b;  
this statement makes the current value of a(which belongs to **obj2**) as 40.  
Similarly the value of b is initialised with 50

**Example 3.18.1** Demonstrate through a program in Java how constructors can be used for Initializing the elements of matrix with '0' in the example on addition of 2 matrices.

SPPU : Dec.-19, Marks 6

**Solution :**

```
import java.io.*;
import java.util.*;

class Matrix
{
    private int A[][] = new int[3][3];
    private int B[][] = new int[3][3];
    private int C[][] = new int[3][3];

    public Matrix() // Constructor that initializes matrix to 0
    {
        for(int i=0;i<3;i++)
        {
            for(int j=0;j<3;j++)
            {
                A[i][j] = 0;
                B[i][j] = 0;
            }
        }
    }

    public void show()
    {
        System.out.println("Resultant Matrix is ...");
        for(int i=0;i<3; i++)
        {
            for(int j=0;j<3;j++)
            {
                System.out.print(C[i][j]+ " ");
                System.out.println("");
            }
        }
    }
}
```

```
public void get_matrix()
{
    System.out.println("Enter matrix A");
    Scanner input = new Scanner(System.in);
    for(int i=0;i<3; i++)
    {
        for(int j=0;j<3;j++)
            A[i][j] = input.nextInt();
    }
    System.out.println("Enter matrix B");
    for(int i=0;i<3; i++)
    {
        for(int j=0;j<3;j++)
            B[i][j] = input.nextInt();
    }
}
public void addition()
{
    for(int i=0;i<3; i++)
    {
        for(int j=0;j<3;j++)
            C[i][j]=A[i][j]+B[i][j];
    }
}
}
class Test
{
    public static void main(String args[])
    {
        Matrix M1 = new Matrix();
        M1.get_matrix();
        M1.addition();
        M1.show();
    }
}
```

### Review Question

1. Explain about 'this' keyword with an example.

### 3.19 Garbage Collection

- Garbage collection is a method of automatic memory management.

- It works as follows -

**Step 1 :** When an application needs some free space to allocate the nodes and if there is no free space available to allocate the memory for these objects then a system routine called **garbage collector** is called.

**Step 2 :** This routine then searches the system for the nodes that are no longer accessible from an external pointer. These nodes are then made available for reuse by adding them to available pool. The system can then make use of these free available space for allocating the nodes.

- Garbage collection is usually done in two phases - marking phase and collection phase.
- In **marking phase**, the garbage collector scans the entire system and marks all the nodes that can be accessible using external pointer.
- During **collection phase**, the memory is scanned sequentially and the unmarked nodes are made free.

**Marking phase :** For marking each node, there is one field called **mark field**. Each node that is accessible using external pointer has the value **TRUE** in marking field. For example -

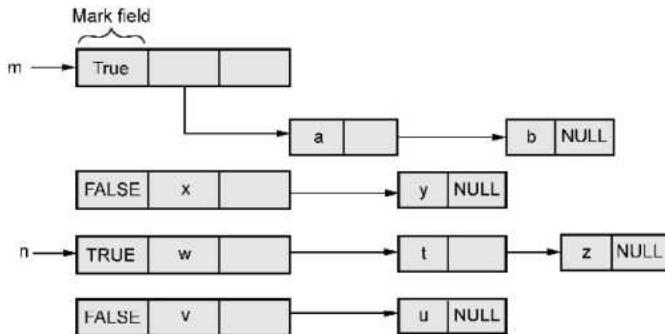


Fig. 3.19.1

#### Collection phase

During collection phase, all the nodes that are marked FALSE are collected and made free. This is called **sweeping**. There is another term used in regard to garbage collection called **Thrashing**.

Consider a scenario that, the garbage collector is called for getting some free space and almost all the nodes are accessible by external pointers. Now garbage collection routine executes and returns a small amount of space. Then again after some time system demands for some free space. Once again garbage collector gets invokes which returns very small amount of free space. This happens repeatedly and garbage collection routine is executing almost all the time. This process is called **thrashing**. Thrashing must be avoided for better system performance.

#### Advantages of garbage collection

1. The manual memory management done by the programmer (i.e. use of malloc and free) is time consuming and error prone. Hence automatic memory management is done.
2. Reusability of memory can be achieved with the help of garbage collection.

#### Disadvantages of garbage collection

1. The execution of the program is paused or stopped during the process of garbage collection.
2. Sometime situation like thrashing may occur due to garbage collection.

#### Review Question

1. Explain the garbage collections.

#### 3.20 finalize() Method

- Java has a facility of automatic garbage collection. Hence even though we allocate the memory and then forget to deallocate it then the objects that are no longer in use - get freed.
- Inside the **finalize()** method you will specify those actions that must be performed before an object is destroyed.
- The garbage collector runs periodically checking for objects that are no longer referenced by any running state or indirectly through other referenced objects. To add finalizer to a class simply define the finalize method.
- The **syntax** is

```
finalize() the code is -  
void finalize()  
{
```

```
finalization code
}
```

- Note that **finalize()** method is called just before the garbage collection. It is not called when an object goes out-of-scope.

### 3.21 Overload Methods

**Definition :** Overloading is a mechanism in which we can use many methods having the same function name but can pass different number of parameters or different types of parameter.

Overloading is a mechanism in which we can use many methods having the same function name but can pass different number of parameters or different types of parameter.

**For example :**

```
int sum(int a,int b);
double sum(double a,double b);
int sum(int a,int b,int c);
```

That means, by overloading mechanism, we can handle different number of parameters or different types of parameter by having the same method name. Following Java program explains the concept overloading -

#### Java Program [OverloadingDemo.java]

```
public class OverloadingDemo {
    public static void main(String args[]) {
        System.out.println("Sum of two integers   ");
        Sum(10,20); <----- line A
        System.out.println("Sum of two double numbers   ");
        Sum(10.5,20.4); <----- line B
        System.out.println("Sum of three integers   ");
        Sum(10,20,30); <----- line C
    }
    public static void Sum(int num1,int num2)
    {
        int ans;
        ans=num1+num2;
        System.out.println(ans);
    }
    public static void Sum(double num1,double num2)
```

```
{  
    double ans;  
    ans=num1+num2;  
    System.out.println(ans);  
}  
public static void Sum(int num1,int num2,int num3)  
{  
    int ans;  
    ans=num1+num2+num3;  
    System.out.println(ans);  
}  
}
```

**Output**

```
F:\test>javac OverloadingDemo.java  
F:\test>java OverloadingDemo  
Sum of two integers  
30  
Sum of two double numbers  
30.9  
Sum of three integers  
60
```

**Program explanation**

In above program, we have used three different methods possessing the same name. Note that,

**On the line A**

We have invoked a method to which two integer parameters are passed. Then compiler automatically selects the definition **public static void Sum(int num1, int num2)** to fulfil the call. Hence we get the output as 30 which is actually the addition of 10 and 20.

**On line B**

We have invoked a method to which two double parameters are passed. Then compiler automatically selects the definition **public static void Sum(double num1,double num2)** to fulfil the call. Hence we get the output as 30.9 which is actually the addition of 10.5 and 20.4.

**On line C**

We have invoked a method to which the three integer parameters are passed. Then compiler automatically selects the definition **public static void Sum(int num1,int num2,int num3)**.

**num2,int num3)** to fulfil the call. Hence we get the output as 60 which is actually the addition of 10, 20 and 30.

### 3.22 Argument Passing

Parameter can be passed to the function by two ways -

- 1. Call by value
- 2. Call by reference

#### 1. Call by value

In the **call by value** method the value of the actual argument is assigned to the formal parameter. If any change is made in the formal parameter in the subroutine definition then that change does not reflect the actual parameters. Following Java program shows the use of parameter passing by value -

#### Java Program

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Program implementing the parameter passing by value
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
public class ParameterByVal
{
    void Fun(int a,int b)
    {
        a=a+5;
        b=b+5;
    }
    public static void main(String args[])
    {
        ParameterByVal obj1=new ParameterByVal();
        int a,b;
        a=10;b=20;
        System.out.println("The values of a and b before function call");
        System.out.println("a = "+a);
        System.out.println("b = "+b);
        obj1.Fun(a,b);
        System.out.println("The values of a and b after function call");
        System.out.println("a = "+a);
        System.out.println("b = "+b);

    }
}
```

**Output**

The values of a and b before function call

a = 10

b = 20

The values of a and b after function call

a = 10

b = 20

**Program explanation**

The above Java program makes use of the parameter **passing by value**. By this approach of parameter passing we are passing the actual values of variables a and b.

In the function **Fun** we are changing the values of a and b by adding 5 to them. But these incremented values will remain in the function definition body only. After returning from this function these values will not get preserved. Hence we get the same values of a and b before and after the function call.

**2. Call by reference**

The parameter passing by reference allows to change the values after the function call. But use of variables as a parameter does not allow to pass the parameter by reference. For passing the **parameter by reference** we **pass the object** of the class as a parameter. Creating a variable of class type means creating reference to the object. If any changes are made in the object made inside the method then those changes get preserved and we can get the changed values after the function call.

**3.23 Object as Parameter**

- The object can be passed to a method as an argument.
- Using **dot operator** the object's value can be accessed. Such a method can be represented syntactically as -

```
Data_Type name_of_method(object_name)
{
    //body of method
}
```

Following is a sample Java program in which the method **area** is defined. The parameter passed to this method is an **object**.

**Java Program[ ObjDemo.java]**

```

public class ObjDemo {
    int height;
    int width;
    ObjDemo(int h,int w)
    {
        height=h;
        width=w;
    }
    void area(ObjDemo o)
    {
        int result=(height + o.height)*(width + o.width);
        System.out.println("The area is "+result);
    }
}
class Demo {
    public static void main(String args[])
    {
        ObjDemo obj1=new ObjDemo(2,3);
        ObjDemo obj2=new ObjDemo(10,20);
        obj1.area(obj2);
    }
}

```

F:\test>javac ObjDemo.java  
F:\test>java Demo

The area is 276

**Output**

Method with object as parameter

height=2 and o.height=10  
width=3 and o.width=20  
Hence,  
result=(2+10)\*(3+20)  
=12\*23  
=276

**Program explanation**

- 1) The method, **area** has object **obj2** as argument. And there is another object named **obj1** using which the method **area** is invoked.
- 2) Inside the method **area**, the height and width values of invoking object are added with the height and width values of the object to be passed.
- 3) When an object needs to be passed as a parameter to the function then constructor is built. Hence we have to define **constructor** in which the values of the object are used for initialization.

### 3.24 Returning Object

We can return an object from a method. The data type such method is a class type.

#### Java Program[ObjRetDemo.java]

```
public class ObjRetDemo {
    int a;
    ObjRetDemo(int val)
    {
        a=val;
    }
    ObjRetDemo fun()
    {
        ObjRetDemo temp=new ObjRetDemo(a+5); //created a new object
        return temp; //returning the object from this method
    }
}
class ObjRet {
    public static void main(String args[])
    {
        ObjRetDemo obj2=new ObjRetDemo(20);
        ObjRetDemo obj1;
        obj1=obj2.fun(); //obj1 gets the value from object temp
        System.out.println("The returned value is = "+obj1.a);
    }
}
```

Data type of this method is name of the class

#### Output

```
F:\test>javac ObjRetDemo.java
F:\test>java ObjRet
The returned value is = 25
```

### 3.25 Access Control

- In object oriented programming, a class can protect its member variables or methods from accessing by other entities. That means a class can allow to get the access for particular variable from outside or particular variable can not be accessed at all by the outside entity. This access control mechanism is set by means of **access specifiers**.
- Access specifiers control access to data fields, methods and classes.

- There are three modifiers used in Java -
  - public
  - private
  - default modifier

**public** allows classes, methods and data fields accessible from any class

**private** allows classes, methods and data fields accessible only from within the own class.

- If public or private is not used then by default the classes, methods, data fields are assessable by any class in the same package. This is called package-private or package-access. A package is nothing but the grouping of classes.

#### For example :

```
package Test;
public class class1
{
    public int a;
    int b;
    private int c;
    public void fun1() {
    }
    void fun2() {
    }
    private void fun3() {
    }
}
public class class2
{
    void My_method() {
        class1 obj=new class1();
        obj.a;//allowed
        obj.b;//allowed
        obj.c;//error:cannot access
        obj.fun1();//allowed
        obj.fun2();//allowed
        obj.fun3();//error:cannot access
    }
}
```

```
package another_Test
public class class3
{
    void My_method() {
        class1 obj=new class1();
        obj.a;//allowed
        obj.b;// error:cannot access
        obj.c;//error:cannot access
        obj.fun1();//allowed
        obj.fun2();//error:cannot access
        obj.fun3();//error:cannot access
    }
}
```

In above example,

- We have created two packages are created namely- **Test** and **another\_Test**.
- Inside the package **Test** there are two classes defined - **class1** and **class2**.
- Inside the package **another\_Test** there is only one class defined and i.e. **class3**.
- There are three data fields – **a**, **b** and **c**. The data field **a** is declared as public, **b** is defined as default and **c** is defined as private.
- The variable **a** and method **fun1()** both are accessible from the classes **class2** and **class3** (even if it is in another package). This is because they are declared as **public**.
- The variable **b** and method **fun2()** both are accessible from **class2** because **class2** lies in the same package. But they are not accessible from **class3** because **class3** is defined in another package.
- The variable **c** and method **fun3()** both are not accessible from any of the class, because they are declared as private.

**Protected mode** is another access specifier which is used in inheritance. The protected mode allows accessing the members to all the classes and subclasses in the same package as well as to the subclasses in other package. But the non subclasses in other package can not access the protected members.

The effect of access specifiers for class, subclass or package is enlisted below -

Specifier	Class	Subclass	Package
<b>private</b>	Yes	-	-
<b>protected</b>	Yes	Yes	Yes
<b>public</b>	Yes	Yes	Yes

For example, if some variable is declared as protected, then the class itself can access it, its subclass can access it and any class in the same package can also access it. Similarly if the variable is declared as private then that variable is accessible by that class only and its subclass can not access it.

### Review Question

1. Discuss the relative merits of using protected access vs. private access in super class.

### 3.26 Static Members

- The static members can be **static data member** or **static method**.
- The static members are those members which can be accessed **without using object**.

Following program illustrates the use of static members.

#### Java Program[StaticProg.java]

```
*****
Program to introduce the use of the static method and
static variables
*****
class StaticProg
{
    static int a=10;
    static void fun(int b)
    {
        System.out.println("b= "+b);
        System.out.println("a= "+a);

    }
}
class AnotherClass
{
    public static void main(String[] args)
    {
        System.out.println("a= "+StaticProg.a);
        StaticProg.fun(20);

    }
}
```

#### Output

```
a= 10
b= 20
a= 10
```

#### Program explanation

In above program,

1. We have declared one static variable **a** and a static member function **fun()**.
2. These static members are declared in one class **StaticProg**.
3. Then we have written one more class in which the **main** method is defined. This class is named as **AnotherClass**.
4. From this class the static members of class **StaticProg** are accessed *without using any object*. Note that we have simple used a syntax

**ClassName.staticMember**

Hence by using **StaticProg.a** and **StaticProg.fun** we can access static members.

### 3.27 Nested and Inner Class

Inner classes are the nested classes. That means these are the classes that are defined inside the other classes. The syntax of defining the inner class is -

```
Access_modifier class OuterClass
{
    //code
    Access_modifier class InnerClass
    {
        //code
    }
}
```

Following are some **properties** of inner class -

- The outer class can inherit as many number of inner class objects as it wants.
- If the outer class and the corresponding inner class both are **public** then any other class can create an instance of this inner class.
- The inner class objects do not get instantiated with an outer class object.
- The outer class can call the private methods of inner class.
- Inner class code has free access to all elements of the outer class object that contains it.
- If the inner class has a variable with same name then the outer class's variable can be accessed like this -

`outerclassname.this.variable_name`

When a one class is defined inside other class then it is called nested classes. The scope of the inner class is bounded by the scope of the outer class.

```
class A
{
    ...
}

class B
{
    ...
    ...
    Variable of class A is accessible here
}
...
Variable of class B is accessible here
}
class B is totally unknown here
```

### Java Program

```
////////////////////////////
//Demonstrating the concept of nested and inner class
////////////////////////////
import java.io.*;
import java.util.*;
class A
{
int a=10;
void show()
{
B obj_B=new B();
obj_B.display();
}
class B
{
void display()
{
System.out.println("The value of a is = "+a);
}
}
public static void main(String[] args)
{
A obj_A=new A();
obj_A.show();
}
}
```

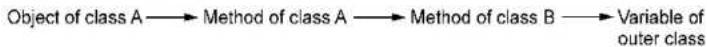
### Output

The value of a is = 10

### Program explanation

In above program, the class B is within the class A. The class A has one function **show()** which invokes the function of class B. The class B defines a function **display()** which invokes the variable **a** of class A. That means inner class can access the contents of the outer class.

The access chain will be



**Fig. 3.27.1 Access chain**

### Uses of inner class

1. Inner classes are used to logically group the classes that are used in one place.

2. The inner class is can be hidden from the other classes in the same package and still have the access to all the members (private also) of the enclosing class.
3. The inner class shares a special relationship with the outer class i.e. the inner class has access to all members of the outer class and still have its own members.
4. Inner classes are used in the situation in which the level of encapsulation is high.

### 3.28 Command Line Argument

- Using command line arguments we can provide the input to the program during the execution.
- For example while executing the program test.java we can provide the command line argument like this -

c:\>java test 10 20

10 and 20 are the command line arguments

- The command line parameters can be read in an array of strings. For keeping track of total number of command prompt arguments a integer count can be maintained.

#### Java Program[CommandDemo.java]

```
class CommandDemo
{
    public static void main(String args[])
    {
        int i=0;
        String str;

        System.out.println("\n Following are the strings entered at command prompt");
        while(i<args.length)
        {
            str=args[i];
            i=i+1;
            System.out.println(str);
        }
        System.out.println("\n The total number of arguments are..." +i);
    }
}
```

#### Output

D:\>java CommandDemo aaa bbb ccc ddd eee fff

Following are the strings entered at command prompt

```
aaa
bbb
ccc
ddd
eee
fff
```

The total number of arguments are...6

### 3.29 Variable Length Arguments

- Java allows the method to use variable length arguments i.e. it allows to create the methods that can accept the arguments of length zero or multiple. This is called as varargs in Java. The varargs uses three dots after the data type. The syntax is as follows -

Return type name\_of\_function(data\_type ... variable\_name)

Following Java program illustrates the use of such variable arguments.

#### Java Program

```
class Test
{
    public static void main(String[] args)
    {
        System.out.println("Sum: "+fun(10,20));
        System.out.println("Sum: "+fun(10,20,30));
        System.out.println("Sum: "+fun(10,20,30,40,50));
    }
    public static int fun(int ... val)
    {
        int total=0;
        for(int x:val)
            total+=x;
        return total;
    }
}
```

#### Output

```
Sum: 30
Sum: 60
Sum: 150
```

**Program explanation :** In above program,

- 1) We have used function **fun** to which variable number of arguments can be passed. This function is for computing the sum of all the numbers passed to it as function argument.
- 2) We have invoked the variable argument function for **three times**,
  - a. In the **first call** we have passed two arguments,
  - b. In the **second call** we have passed three arguments and
  - c. In the **third call** we have passed five arguments and the corresponding total of the passed arguments is obtained.

### 3.30 Multiple Choice Questions

**Q.1 Write once and run anywhere at anytime forever is a feature of \_\_\_ programming**

- |                                 |                                      |
|---------------------------------|--------------------------------------|
| <input type="checkbox"/> a C    | <input type="checkbox"/> b C++       |
| <input type="checkbox"/> c Java | <input type="checkbox"/> d Smalltalk |

**Q.2 Java is made useful for distributed systems. It is possible due to \_\_\_ feature of JAVA.**

- |  |  |
|--|--|
| <input type="checkbox"/> a inheritance | <input type="checkbox"/> b RMI           |
| <input type="checkbox"/> c API         | <input type="checkbox"/> d none of these |

**Explanation :** The RMI stands for Remote Method Invocation. Two different objects on two remote machine can communicate with each other due to Remote Method Invocation feature.

**Q.3 Java is said to be \_\_\_.**

- |  |  |
|--|--|
| <input type="checkbox"/> a compiled                      | <input type="checkbox"/> b interpreted   |
| <input type="checkbox"/> c both compiled and interpreted | <input type="checkbox"/> d none of these |

**Explanation :** First, Java compiler translates the Java source program into a special code called bytecode. Then Java interpreter interprets this bytecode to obtain the equivalent machine code. This machine code is then directly executed to obtain the output.

**Q.4 Java Virtual machine takes \_\_\_ as input and produces output.**

- |  |   |
|--|---|
| <input type="checkbox"/> a source code       | <input type="checkbox"/> b byte code    |
| <input type="checkbox"/> c intermediate code | <input type="checkbox"/> d machine code |

**Q.5 Compiler converts the Java program into an intermediate language representation which is called \_\_\_\_\_.**

- |                                     |                                      |
|-------------------------------------|--------------------------------------|
| <input type="checkbox"/> a byte     | <input type="checkbox"/> b bit       |
| <input type="checkbox"/> c bytecode | <input type="checkbox"/> d byteclass |

**Q.6 Java code is said to be \_\_\_ so that it can easily run on any system.**

- |                                     |  |
|-------------------------------------|--|
| <input type="checkbox"/> a portable | <input type="checkbox"/> b discrete      |
| <input type="checkbox"/> c simple   | <input type="checkbox"/> d none of these |

**Q.7 Four types of integers in Java are \_\_\_\_\_.**

- |  |  |
|--|--|
| <input type="checkbox"/> a bit, byte, short and long | <input type="checkbox"/> b nibble, byte,int and long |
| <input type="checkbox"/> c byte, short, int and long | <input type="checkbox"/> d none of these             |

**Q.8 Which of the following is not Java keyword ?**

- |  |  |
|--|--|
| <input type="checkbox"/> a this        | <input type="checkbox"/> b public        |
| <input type="checkbox"/> c synchronize | <input type="checkbox"/> d none of these |

**Q.9 What is the range of byte data type in Java ?**

- |  |  |
|--|--|
| <input type="checkbox"/> a -1024 to 1024   | <input type="checkbox"/> b -128 to 127   |
| <input type="checkbox"/> c -32768 to 32768 | <input type="checkbox"/> d None of these |

**Q.10 Which of the following can be contained in float data type ?**

- |                                     |                                     |
|-------------------------------------|-------------------------------------|
| <input type="checkbox"/> a 1.7e-308 | <input type="checkbox"/> b 3.4e-038 |
| <input type="checkbox"/> c 1.7e+388 | <input type="checkbox"/> d 3.4e-068 |

**Explanation :** The range of float data type.

**Q.11 What is the range of char data type in Java ?**

- |  |  |
|--|--|
| <input type="checkbox"/> a -128 to 127 | <input type="checkbox"/> b 0 to 256      |
| <input type="checkbox"/> c 0 to 65535  | <input type="checkbox"/> d None of these |

**Q.12 Choose the correct statement.**

- |  |   |
|--|---|
| <input type="checkbox"/> a boolean a='false' | <input type="checkbox"/> b boolean a='true' |
| <input type="checkbox"/> c boolean a="true"  | <input type="checkbox"/> d boolean a=true   |

**Q.13 What is the output of following code ?**

```
class Test
{
    public static void main(String args[])
    {
        char i = 'X';
        i++;
        System.out.print((int)i);
    }
}
```

- |                               |  |
|-------------------------------|--|
| <input type="checkbox"/> a 89 | <input type="checkbox"/> b 88            |
| <input type="checkbox"/> c 87 | <input type="checkbox"/> d None of these |

**Explanation :** The ASCII value of X is 88, it is incremented by one hence output is 89.

**Q.14 What is the output of the following code ?**

```
class Test
{
    public static void main(String args[])
    {
        boolean a = true;
        boolean b = false;
        if (!a)
            System.out.println(a);
        else
            System.out.println(b);
    }
}
```

- |                                 |                                  |
|---------------------------------|----------------------------------|
| <input type="checkbox"/> a true | <input type="checkbox"/> b false |
| <input type="checkbox"/> c 0    | <input type="checkbox"/> d 1     |

**Q.15 How many primitive data types are there in Java ?**

- |                              |                               |
|------------------------------|-------------------------------|
| <input type="checkbox"/> a 4 | <input type="checkbox"/> b 7  |
| <input type="checkbox"/> c 8 | <input type="checkbox"/> d 10 |

**Q.16 In Java byte, short, int and long all these are \_\_\_\_\_.**

- |  |  |
|--|--|
| <input type="checkbox"/> a signed        | <input type="checkbox"/> b unsigned      |
| <input type="checkbox"/> c both of these | <input type="checkbox"/> d none of these |

**Q.17 Size of int in Java is \_\_\_\_\_.**

- |                                   |  |
|-----------------------------------|--|
| <input type="checkbox"/> a 16 bit | <input type="checkbox"/> b 32 bit        |
| <input type="checkbox"/> c 64 bit | <input type="checkbox"/> d none of these |

**Q.18 The smallest integer type is \_\_\_ and its size is \_\_\_ bits.**

- |                                       |                                     |
|---------------------------------------|-------------------------------------|
| <input type="checkbox"/> a short , 16 | <input type="checkbox"/> b short, 8 |
| <input type="checkbox"/> c byte, 8    | <input type="checkbox"/> d byte, 32 |

**Q.19 Which of the following can be the operands of arithmetic operators ?**

- |                                    |   |
|------------------------------------|---|
| <input type="checkbox"/> a Integer | <input type="checkbox"/> b Char         |
| <input type="checkbox"/> c Boolean | <input type="checkbox"/> d Both a and b |

**Explanation :** The arithmetic operators are +,-,/,\* and %. Only boolean type of operand is not for arithmetic operators.

**Q.20 What is the output of the following code ?**

```
class Test
{
    public static void main(String args[])
    {
        int a = 10;
        int b = 20;
        int c;
        int d;
        c = ++b;
        d = a++;
        c++;
        b++;
        ++a;
        System.out.println(a + " " + b + " " + c);
    }
}
```

 a 11 21 21 b 12 21 22 c 12 22 22 d None of these**Q.21 What will be the output of the following code ?**

```
class Test
{
    public static void main(String args[])
    {
        int x = 11;
        double y = 11.11;
        boolean b = (x = y);
        System.out.println(b);
    }
}
```

 a true b false c Syntax error d exception

**Explanation :** The syntax error occurs because instead of `(x=y)` there should be `(x==y)`.

The single `=` is assignment operator and `==` is equality operator which returns true or false.

**Q.22 What is the output of the following code ?**

```
class Test
{
    public static void main(String args[])
    {
        double a = 55.66;
        int b = 55;
        a = a % 10;
        b = b % 10;
        System.out.println(a + " " + b);
    }
}
```

- a 5 5       b 5.659999999999997 5  
 c 5.659999999999997 5.0       d None of these

**Q.23 What will be the result of following code ?**

```
int ++i=10;
System.out.println(++i);
```

- a 100       b Displays error because ++i is not enclosed within the double quotes in print statement.  
 c Displays error because ++i is not valid identifier.  
 d none of these.

**Q.24 What will be the output of the following code ?**

```
if(1+1+1==3)
    System.out.print("TRUE");
Else
    System.out.print("FALSE");
```

- a TRUE       b FALSE  
 c Syntax error       d None of these

**Q.25 What will be the output of the following code ?**

```
public class Test
{
    public static void main(String args[])
    {
        int x=10;
        x*=8+2;
        System.out.println(x);
    }
}
```

- a 20  
 c Syntax error

- b 100  
 d None of these

**Explanation :** The expression is converted to  $x = x * (8 + 2) = 10 * 10 = 100$

**Q.26 What is the output of the following code ?**

```
public class Test
{
    public static void main(String args[])
    {
        int a=4;
        System.out.print(++a*10);
    }
}
```

- a 40  
 c 41

- b 50  
 d Syntax error

**Explanation :** Due to pre-increment operator the variable a becomes = 5. Then  $5 * 10 = 50$  will be printed as an output.

**Q.27 For determining whether the object belongs to particular class or not \_\_\_\_\_ is used.**

- a dot operator  
 c instanceof operator

- b -> operator  
 d None of these

**Q.28 Which of the following is a selection statement ?**

- a if  
 c do..while

- b for  
 d while

**Q.29 What is the output of the following code ?**

```
public class Test
{
    public static void main(String args[])
    {
        int a=1;
        if(a)
            System.out.println("Hi");
        else
            System.out.println("Hello");
    }
}
```

- a Hi  
 c Syntax error

- b Hello  
 d None of these

**Explanation :** The value of variable a is integer which can not be converted boolean type which results in syntax error.

**Q.30 The syntax error will be on line \_\_\_\_\_.**

```
public class Test
{
    public void loop()
    {
        int x = 0;
        while ( 1 ) /* Line A */
        {
            System.out.print("x plus one is " + (x + 1)); /* Line B */
        }
    }
}
```

- |  |                                     |
|--|-------------------------------------|
| <input type="checkbox"/> a Line A and Line B | <input type="checkbox"/> b Line A   |
| <input type="checkbox"/> c Line B            | <input type="checkbox"/> d No error |

**Explanation :** Using the integer 1 in the while statement, or any other looping or conditional construct for that matter, will result in a compiler error. This is old C Program syntax, not valid Java.

**Q.31 In Java \_\_\_ can only test for equality whereas \_\_\_ can evaluate any type of Boolean expression.**

- |                                       |  |
|---------------------------------------|--|
| <input type="checkbox"/> a if, switch | <input type="checkbox"/> b if, break     |
| <input type="checkbox"/> c switch, if | <input type="checkbox"/> d none of these |

**Q.32 For how many time "Hello India" will be printed ?**

```
public class Test
{
    public static void main(String args[])
    {
        int count = 0;
        do
        {
            System.out.println("Hello India");
            count++;
        } while (count < 10);
    }
}
```

- |                              |                               |
|------------------------------|-------------------------------|
| <input type="checkbox"/> a 0 | <input type="checkbox"/> b 8  |
| <input type="checkbox"/> c 9 | <input type="checkbox"/> d 10 |

**Q.33 What is the output of the following code ?**

```
class Test
{
    public static void main(String args[])
    {
        int i = 0;
        for ( ; i < 10; ++i)
        {
            if (i % 2 == 0)
                continue;
            else if (i == 8)
                break;
            else
                System.out.print(i + " ");
        }
    }
}
```

 a 1 3 5 7 9 b 8 c 1 3 5 7 d 1 2 3 4 5 6 7 8 9

**Explanation :** Whenever i is divisible by 2 remainder body of loop is skipped by continue statement, therefore if condition  $i == 8$  is never true as when i is 8, remainder body of loop is skipped by continue statements of first if. Control comes to print statement only in cases when i is odd.

**Q.34 Choose the incorrect declaration.**1. `int a[ ]=new int[10]`2. `int[ ] a=new int[10]`3. `int a[ ];``a=new int[10]`4. `int a[ ]=int[10] new` a 1 b 2 c 3 d 4**Q.35 Choose the incorrect statement.** a Array can be initialized with the comma separated expression enclosed within the curly bracket. b In Java, the only way to initialize the array using new operator. c Array can be initialized only when they are declared. d None of these.

**Explanation :** The array can be initialized

i) using new operator -

```
int a=new int[10];
```

ii) using the expression enclosed in curly bracket -

```
int a[ ]={10,20,30,40};
```

**Q.36 What is the output of the following code ?**

```
public class Test
{
    public static void main(String[] args){
        int[] a = new int[0];
        System.out.print(a.length);
    }
}
```

a 0

b Garbage value

c Syntax error

d 1

**Q.37 What will be the output of the following code ?**

```
class Test
{
    public static void main(String args[])
    {
        int arr[] = new int[] {0 , 1, 2, 3, 4, 5, 6, 7, 8, 9};
        int n = 6;
        n = arr[arr[n] / 2];
        System.out.println(arr[n] / 2);
    }
}
```

a Syntax error

b 0

c 1

d 3

**Explanation :** n=6. The n=arr[arr[n]/2]=arr[arr[6]/3]=arr[6/2]=arr[3]=3

In the System.out statement arr[3]/2=3/2=1. Hence the output is 1.

**Q.38 The necessary condition for automatic type conversion in Java is \_\_\_\_\_.**

a the destination type should be smaller than the source type

b the destination type should be larger than the source type

c there is no automatic conversion allowed in Java

d none of these

**Q.39 When integer value is converted to an double the fractional part is \_\_\_\_**

- a truncated
- b padded
- c conversion from integer to double is not possible in Java
- d none

**Q.40 What will be the output of the following code ?**

```
class Test
{
    public static void main(String args[])
    {
        double a = 299.81;
        int b = 330;
        byte c = (byte) a;
        byte d = (byte) b;
        System.out.println(c + " " + d);
    }
}
```

- |   |   |
|---|---|
| <input type="checkbox"/> a 42 72            | <input type="checkbox"/> b 43 73        |
| <input checked="" type="checkbox"/> c 43 74 | <input type="checkbox"/> d Syntax error |

**Explanation :** The range of byte is -128 to 127. The double and integer values are converted to byte. Hence the type conversion from larger data type to smaller data type will occur. The modulo of larger variable by range of smaller datatype will occur. Range of smaller data type i.e. byte = -128 to 127=256

Hence c = a modulo 256 = 299.81 modulo 256 = 43

And d = b modulo 256 = 330 modulo 256 = 74

**Q.41 String is a \_\_\_ in Java.**

- |  |  |
|--|--|
| <input type="checkbox"/> a object              | <input type="checkbox"/> b class         |
| <input checked="" type="checkbox"/> c variable | <input type="checkbox"/> d none of these |

**Q.42 For extracting a character from particular location \_\_\_ method of string class is used.**

- |   |                                     |
|---|-------------------------------------|
| <input type="checkbox"/> a char()                 | <input type="checkbox"/> b charat() |
| <input checked="" type="checkbox"/> c charIndex() | <input type="checkbox"/> d charAt() |

**Q.43 For checking the equality of two strings \_\_\_ method is used in Java.**

- |   |  |
|---|--|
| <input type="checkbox"/> a equals             | <input type="checkbox"/> b equal         |
| <input checked="" type="checkbox"/> c isEqual | <input type="checkbox"/> d None of these |

**Q.44 For finding the length of a string \_\_\_ method is used in Java.**

- |                                     |  |
|-------------------------------------|--|
| <input type="checkbox"/> a len()    | <input type="checkbox"/> b lengthCount() |
| <input type="checkbox"/> c length() | <input type="checkbox"/> d None of these |

**Q.45 The operator \_\_\_ can be used for concatenating two strings.**

- |                               |  |
|-------------------------------|--|
| <input type="checkbox"/> a +  | <input type="checkbox"/> b ->            |
| <input type="checkbox"/> c ++ | <input type="checkbox"/> d None of these |

**Q.46 The constructor \_\_\_ is used to create an empty string object.**

- |   |  |
|---|--|
| <input type="checkbox"/> a String()     | <input type="checkbox"/> b String(0)     |
| <input type="checkbox"/> c String(void) | <input type="checkbox"/> d None of these |

**Q.47 What is the output of the following code ?**

```
class Test
{
    public static void main(String args[])
    {
        int a[] = {96,97,98,99,100};
        String s = new String(a,1,4);
        System.out.println(s);
    }
}
```

- |                                 |  |
|---------------------------------|--|
| <input type="checkbox"/> a abcd | <input type="checkbox"/> b abc           |
| <input type="checkbox"/> c bcdd | <input type="checkbox"/> d None of these |

**Q.48 Character data type can not store following value.**

- |  |  |
|--|--|
| <input type="checkbox"/> a Special character | <input type="checkbox"/> b Digit         |
| <input type="checkbox"/> c String            | <input type="checkbox"/> d None of these |

**Q.49 For converting the string to upper case \_\_\_ method of String class is used.**

- |                                       |  |
|---------------------------------------|--|
| <input type="checkbox"/> a toUpper()  | <input type="checkbox"/> b Upper()       |
| <input type="checkbox"/> cUpperCase() | <input type="checkbox"/> d toUpperCase() |

**Answer Keys for Multiple Choice Questions :**

<b>Q.1</b>	c	<b>Q.2</b>	b	<b>Q.3</b>	c	<b>Q.4</b>	b
<b>Q.5</b>	c	<b>Q.6</b>	a	<b>Q.7</b>	c	<b>Q.8</b>	c
<b>Q.9</b>	b	<b>Q.10</b>	b	<b>Q.11</b>	c	<b>Q.12</b>	d
<b>Q.13</b>	a	<b>Q.14</b>	b	<b>Q.15</b>	b	<b>Q.16</b>	a
<b>Q.17</b>	b	<b>Q.18</b>	c	<b>Q.19</b>	d	<b>Q.20</b>	c
<b>Q.21</b>	c	<b>Q.22</b>	b	<b>Q.23</b>	c	<b>Q.24</b>	a
<b>Q.25</b>	b	<b>Q.26</b>	b	<b>Q.27</b>	c	<b>Q.28</b>	a
<b>Q.29</b>	c	<b>Q.30</b>	b	<b>Q.31</b>	c	<b>Q.32</b>	d
<b>Q.33</b>	a	<b>Q.34</b>	d	<b>Q.35</b>	b	<b>Q.36</b>	a
<b>Q.37</b>	c	<b>Q.38</b>	b	<b>Q.39</b>	b	<b>Q.40</b>	c
<b>Q.41</b>	b	<b>Q.42</b>	d	<b>Q.43</b>	a	<b>Q.44</b>	c
<b>Q.45</b>	a	<b>Q.46</b>	a	<b>Q.47</b>	a	<b>Q.48</b>	c
<b>Q.49</b>	d						



## Notes

## UNIT - IV

# 4

# Inheritance, Packages and Exception Handling using Java

### Syllabus

**Inheritances** : member access and inheritance, super class references, Using super, multilevel hierarchy, constructor call sequence, method overriding, dynamic method dispatch, abstract classes, Object class.

**Packages and Interfaces** : defining a package, finding packages and CLASSPATH, access protection, importing packages, interfaces (defining, implementation, nesting, applying), variables in interfaces, extending interfaces, instance of operator, fundamental, exception types, uncaught exceptions, try, catch, throw, throws, finally, multiple catch clauses, nested try statements, built-in exceptions, custom exceptions (creating your own exception sub classes).

**Managing I/O** : Streams, Byte Streams and Character Streams, Predefined Streams, Reading console Input, Writing Console Output, Print Writer class.

### Contents

4.1	Introduction to Inheritance	
4.2	Types of Inheritance	
4.3	Implementation of Single Inheritance .....	<b>May-18,</b> ..... Marks 7
4.4	Member access and Inheritance	
4.5	Super Class Reference	
4.6	Using Super .....	<b>Nov.-19,</b> ..... Marks 6
4.7	Use of Final Keyword .....	<b>Dec.-17, May-19,</b> ..... Marks 2
4.8	Multilevel Hierarchy .....	<b>May-19,</b> ..... Marks 7
4.9	Constructor Call Sequence	
4.10	Method Overriding .....	<b>Dec.-17, May-19,</b> ..... Marks 2
4.11	Polymorphism .....	<b>Dec.-17, May-18,</b> <b>Nov.-18, 19,</b> ..... Marks 7
4.12	Abstract Classes .....	<b>Dec.-17, May-19,</b> ..... Marks 2
4.13	Object Class	
4.14	Defining a Package	

4.15 Finding Packages	
4.16 Adding Class to a Package	
4.17 CLASSPATH	
4.18 Access Protection .....	<b>Nov.-18,</b> ..... Marks 6
4.19 Importing Packages	
4.20 Nested Packages	
4.21 JAVA API Package	
4.22 Interfaces .....	<b>Dec.-17, May-18,19,</b> ..... Marks 7
4.23 Variables in Interfaces .....	<b>Nov.-18,</b> ..... Marks 7
4.24 Extending Interfaces	
4.25 Creation and Implementation of Interface	
4.26 Multiple Inheritance .....	<b>Nov.-19,</b> ..... Marks 7
4.27 The Instance of Operator	
4.28 Fundamentals	
4.29 Exception Types .....	<b>Dec.-17, May-19,</b> ..... Marks 6
4.30 try, catch, throw, throws and finally.....	<b>Dec.-17, Nov.-18,</b> ..... Marks 6
4.31 Uncaught Exception .....	<b>May-19,</b> ..... Marks 6
4.32 Multiple Catch Clauses	
4.33 Nested try Statements	
4.34 Built-in Exceptions .....	<b>Dec.-17,</b> ..... Marks 6
4.35 Custom Exceptions	
4.36 Streams	
4.37 Byte Streams and Character Streams .....	<b>Dec.-17,</b> ..... Marks 3
4.38 Predefined Streams .....	<b>Nov.-19,</b> ..... Marks 6
4.39 Reading Console Input	
4.40 Writing Console Output.....	<b>Dec.-17, Nov.-18,</b> ..... Marks 7
4.41 Print Writer Class .....	<b>May-19, Dec.-19,</b> ..... Marks 7
4.42 FileInputStream/FileOutputStream .....	<b>May-18,</b> ..... Marks 6
4.43 Console I/O using Scanner Class.....	<b>May-18,</b> ..... Marks 7
4.44 Multiple Choice Questions with Answers	

**Part I : Inheritance****4.1 Introduction to Inheritance**

**Definition of Inheritance :** Inheritance is a mechanism in Java by which derived class can borrow the properties of base class and at the same time the derived class may have some additional properties.

The inheritance can be achieved by incorporating the definition of one class into another using the keyword **extends**.

**Advantages of Inheritance**

One of the key benefits of inheritance is to **minimize** the amount of **duplicate code** in an application by sharing common code amongst several subclasses.

- 1. Reusability :** The base class code can be used by derived class without any need to rewrite the code.
- 2. Extensibility :** The base class logic can be extended in the derived classes.
- 3. Data hiding :** Base class can decide to keep some data private so that it cannot be altered by the derived class.
- 4. Overriding :** With inheritance, we will be able to override the methods of the base class so that meaningful implementation of the base class method can be designed in the derived class.

**Concept of Super and Sub Class**

- The inheritance is a mechanism in which the **child class** is derived from a **parent class**.
- This derivation is using the keyword **extends**.
- The parent class is called base class and child class is called derived class.
- For example

```
Class A    ← This is Base class or super class
{
    ...
}

Class B extends A   ← This is Derived class or subclass
{
    ...
    // uses properties of A
}
```

- Inheritance is represented diagrammatically as follows

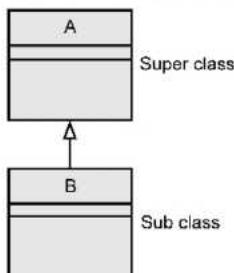


Fig. 4.1.1 Representing inheritance

## 4.2 Types of Inheritance

### 1. Single inheritance :

- In single inheritance there is one parent per derived class. This is the most common form of inheritance.
- The simple program for such inheritance is –

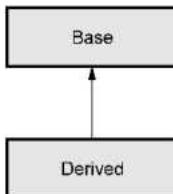


Fig. 4.2.1

### 2. Multiple inheritance :

In multiple inheritance the derived class is derived from more than one base class.

Java **does not implement multiple inheritance** directly but it makes use of the concept called interfaces to implement the multiple inheritance.

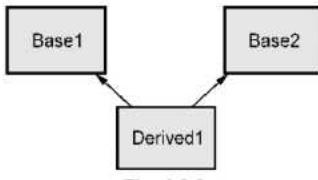


Fig. 4.2.2

### 3. Multilevel inheritance :

When a derived class is derived from a base class which itself is a derived class then that type of inheritance is called multilevel inheritance.

For example - If class A is a base class and class B is another class which is derived from A, similarly there is another class C being derived from class B then such a derivation leads to multilevel inheritance.

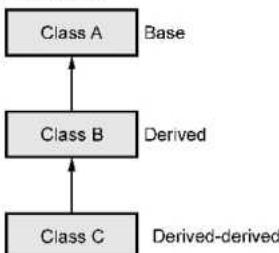


Fig. 4.2.3

#### 4. Hybrid inheritance :

When two or more types of inheritances are combined together then it forms the hybrid inheritance. The following Figure represents the typical scenario of hybrid inheritance.

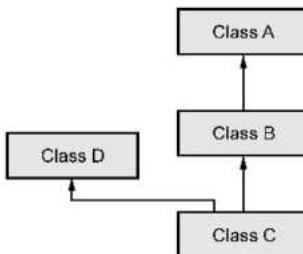


Fig. 4.2.4

#### 4.3 Implementation of Single Inheritance

SPPU : May-18, Marks 7

- The class which is inherited is called the **base class** or the **superclass** and the class that does the inheriting is called the **derived class** or the **subclass**.
- The method defined in base class can be used in derived class. There is no need to redefine the method in derived class. Thus inheritance promotes **software reuse**.
- The subclass can be defined as follows -

```

class nameofSubclass extends superclass
{
    variable declarations
    method declarations
}
    
```

- Note that the keyword **extends** represents that the properties of superclass are extended to the subclass. Thus the subclass will now have both the properties of its own class and the properties that are inherited from superclass.

- Following is a simple Java program that illustrates the concept of single inheritance -

```

class A
{
    int a;
    void set_a(int i)
    {
        a=i;
    }
    void show_a()
    {
        System.out.println("The value of a= "+a);
    }
}
class B extends A //extending the base class A
{
    int b;
    void set_b(int i)
    {
        b=i;
    }
    void show_b()
    {
        System.out.println("The value of b= "+b);
    }
    void mul()
    {
        int c;
        c=a*b;
        System.out.println(" The value of c= "+c);
    }
}
class InheritDemo1
{
    public static void main(String args[])
    {
        A obj_A=new A();
        B obj_B=new B();←
        obj_B.set_a(10);
        obj_B.set_b(20);
        obj_B.show_a();
        obj_B.show_b();
        obj_B.mul();
    }
}

```

Note that object of class B is  
accessing method of class A

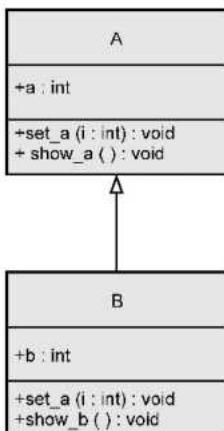
**Output**

```
The value of a= 10
The value of b= 20
The value of c= 200
```

**Program Explanation**

In above program, we have created two classes : class **A** and **B**. In **class A** we have declared one integer **a** and in class **B** we have declared an integer **b**. There are two methods defined in class A namely: **set\_a** and **show\_a**. Similarly, in class B there are two methods defined namely: **set\_b** and **show\_b**. As the name suggests these methods are for setting the values and for showing the contents.

In the class **InheriDemo1**, in the **main** function we have created two objects for class A and class B. The program allows us to access the variable a (belonging to class A) and the variable b (belonging to class B) using the object for class B. Thus it is said that class B has inherited value of variable a.



**Fig. 4.3.1 Single inheritance**

The class A is called **Superclass** and the class B is called **Subclass**. A **Superclass** is also called as **parent class** or **base class**. Similarly, the **Subclass** is also called as **child class** or **derived class**.

**Review Question**

- What is inheritance ? What are advantages of using inheritance ? Show by example the simple inheritance in Java.

**SPPU : May-18, Marks 7**

## 4.4 Member access and Inheritance

Access modifiers control access to data fields, methods, and classes. There are three modifiers used in Java -

- public
- private
- default modifier

**public** allows classes, methods and data fields accessible from any class.

**private** allows classes, methods and data fields accessible only from within the own class.

If public or private is not used then by default the classes, methods, data fields are assessable by any class in the same package. This is called **package-private** or **package-access**. A package is essentially grouping of classes.

**For example :**

```
package Test;
public class class1
{
    public int a;
    int b;
    private int c;
    public void fun1() {
    }
    void fun2() {
    }
    private void fun3() {
    }
}
public class class2
{
    void My_method() {
        class1 obj=new class1();
        obj.a;//allowed
        obj.b;//allowed
        obj.c;//error:cannot access
        obj.fun1();//allowed
        obj.fun2();//allowed
        obj.fun3();//error:cannot access
    }
}

package another_Test
public class class3
{
    void My_method() {
        class1 obj=new class1();
        obj.a;//allowed
        obj.b;// error:cannot access
        obj.c;//error:cannot access
        obj.fun1();//allowed
        obj.fun2();//error:cannot access
        obj.fun3();//error:cannot access
    }
}
```

In above example,

- we have created two packages are created namely - **Test** and **another\_Test**.
- Inside the package **Test** there are two classes defined – **class1** and **class2**
- Inside the package **another\_Test** there is only one class defined and i.e. **class3**.
- There are three data fields – **a**, **b** and **c**. The data field **a** is declared as public, **b** is defined as default and **c** is defined as private.
- The variable **a** and method **fun1()** both are accessible from the classes **class2** and **class3**(even if it is in another package). This is because they are declared as **public**.
- The variable **b** and method **fun2()** both are accessible from **class2** because **class2** lies in the same package. But they are not accessible from **class3** because **class3** is defined in another package.
- The variable **c** and method **fun3()** both are not accessible from any of the class, because they are declared as private.

**Protected** mode is another access specifier which is used in inheritance. The **protected** mode allows accessing the members to all the classes and subclasses in the same package as well as to the subclasses in other package. But the non subclasses in other package can not access the protected members.

The effect of access specifiers for class, subclass or package is enlisted below -

Specifier	Class	Subclass	Package
<b>private</b>	Yes	-	-
<b>protected</b>	Yes	Yes	Yes
<b>public</b>	Yes	Yes	Yes

For example, if some variable is declared as protected, then the class itself can access it, its subclass can access it, and any class in the same package can also access it. Similarly if the variable is declared as private then that variable is accessible by that class only and its subclass can not access it.

#### 4.5 Super Class Reference

A subclass can invoke the constructor method defined by the superclass. The syntax for the **super** will be

```
super();
```

or

```
super(parameter_list);
```

To invoke the constructor of superclass, **super()** or **super(parameter\_list)** statement should appear in the first line of the subclass constructor.

### Java Program

```

class A
{
    int a;
    A(int i)
    {
        this.a=i;
        System.out.println("a is Initialised...");
    }
    void show_a()
    {
        System.out.println("The value of a= "+a);
    }
}
class B extends A
{
    int b;
    B(int i)
    {
        super(i);
        this.b=i+5;
        System.out.println("b is Initialised...");
    }
    void show_b()
    {
        System.out.println("The value of b= "+b);
    }
    void mul()
    {
        int c;
        c=a*b;
        System.out.println("The value of c= "+c);
    }
}
class SuperConstrDemo
{
    public static void main(String args[])
    {
        B obj_B=new B(20); // Note that the object of subclass is created only.
        obj_B.show_a();
        obj_B.show_b();
    }
}

```

It must be on the first line of constructor of the subclass. It calls the constructor of the superclass

```

        obj_B.mul();
    }
}
}

```

**Output**

```

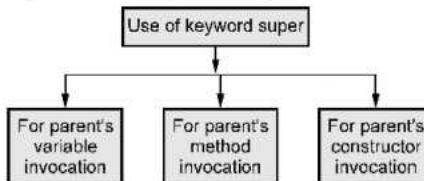
F:\test>javac SuperConstrDemo.java
F:\test>java SuperConstrDemo
a is Initialised...
b is Initialised...
The value of a= 20
The value of b= 25
The value of c= 500

```

**4.6 Using Super****SPPU : Nov.-19, Marks 6**

Super is a keyword used to access the immediate parent class from subclass.

There are three ways by which the keyword **super** is used.

**Fig. 4.6.1 Use of super**

Let us understand these uses of keyword super with illustrative Java programs.

**1. The super() is used to invoke the class method of immediate parent class.****Java Program[B.java]**

```

class A
{
    int x=10;
}

class B extends A
{
    int x=20;
    void display()
    {
        System.out.println(super.x);
    }
    public static void main(String args[])
    {
        B obj=new B();
        obj.display();
    }
}

```

**Output**

**Program Explanation :** In above program,

- (1) class A is an immediate parent class of class B. Both the class A and Class B have variables x.
- (2) In class A, the value of x variable is 10 and in class B the value of variable x is 20.
- (3) In display function if we would write

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

The output will be 20 but if we use super.x then the variable x of class A will be referred. Hence the output is 10.

## 2. The super() is used to access the class variable of immediate parent class.

```
class A
{
    void fun()
    {
        System.out.println("Method: Class A");
    }
}

class B extends A
{
    void fun()
    {
        System.out.println("Method: Class B");
    }
    void display()
    {
        super.fun();
    }
    public static void main(String args[])
    {
        B obj=new B();
        obj.display();
    }
}
```

### Output

Method: Class A

**Program Explanation :** In above program,

- (1) The derived class can access the immediate parent's class method using **super.fun()**. Hence is the output.
- (2) You can change **super.fun()** to **fun()**. Then note that in this case, the output will be invocation of subclass method **fun**.

**3. The super() is used to invoke the immediate parent class constructor.**

```
class A
{
    A()
    {
        System.out.println("Constructor of Class A");
    }
}
class B extends A
{
    B()
    {
        super();
        System.out.println("Constructor of Class B");
    }
    public static void main(String args[])
    {
        B obj=new B();
    }
}
```

### Output

```
Constructor of Class A
Constructor of Class B
```

**Program Explanation :** In above program, the constructor in class B makes a call to the constructor of immediate parent class by using the keyword **super**, hence the print statement in parent class constructor is executed and then the print statement for class B constructor is executed.

### Review Question

1. Elaborate the significance of keyword **super** in Java ? Demonstrate with example each of the case.

**SPPU : Nov.-19, Marks 6**

### 4.7 Use of Final Keyword

**SPPU : Dec.-17, May-19, Marks 2**

The final keyword can be applied at three places -

- For declaring variables
- For declaring the methods
- For declaring the class

#### 4.7.1 Final Variables and Methods

- A variable can be declared as final. If a particular variable is declared as final then it cannot be modified further.

- The final variable is always a constant.
- For example -

```
final int a=10;
```

- The final keyword can also be applied to the method. When final keyword is applied to the method, the method overriding is avoided. That means the methods those are declared with the keyword **final** cannot be overridden.

Consider the following Java program which makes use of the keyword **final** for declaring the method -

```
class Test
{
    final void fun()
    {
        System.out.println("\n Hello, this function declared using final");
    }
}
class Test1 extends Test
{
    final void fun()
    {
        System.out.println("\n Hello, this another function");
    }
}
```

#### Output

```
D:\>javac Test.java
Test.java:10: fun() in Test1 cannot override fun() in Test; overridden method is final
final void fun()
^
1 error
```

#### Program Explanation

The above program, on execution shows the error. Because the method **fun** is declared with the keyword **final** and it cannot be overridden in derived class.

#### **4.7.2 Final Classes to Stop Inheritance**

If we declare particular class as final, no class can be derived from it. Following Java program is an example of final classes.

```
final class Test
{
    void fun()
    {
        System.out.println("\n Hello, this function in base class");
    }
}
```

```

}
class Test1 extends Test
{
final void fun()
{
System.out.println("\n Hello, this another function");
}
}

```

**Output**

```

Test.java:8: cannot inherit from final Test
class Test1 extends Test
^
1 error

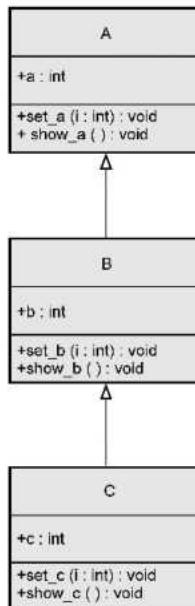
```

**Review Question**

1. State the use of the following constructs in Java with example : Final method declaration in super class while inheritance.

**SPPU : Dec.-17, May-19, Marks 2****4.8 Multilevel Hierarchy****SPPU : May-19, Marks 7**

The multilevel inheritance is a kind of inheritance in which the derived class itself derives the subclasses further.

**Fig. 4.8.1 Multilevel inheritance**

In the following program, we have created a base class A from which the subclass B is derived. There is a class C which is derived from class B. In the function main we can access any of the field in the class hierarchy by created the object of class C.

**Java Program[MultiLvlInherit.java]**

```
class A
{
    int a;
    void set_a(int i)
    {
        a=i;
    }
    void show_a()
    {
        System.out.println("The value of a= "+a);
    }
}

class B extends A
{
    int b;
    void set_b(int i)
    {
        b=i;
    }
    void show_b()
    {
        System.out.println("The value of b= "+b);
    }
}

class C extends B
{
    int c;
    void set_c(int i)
    {
        c=i;
    }
    void show_c()
    {
        System.out.println("The value of c= "+c);
    }
    void mul()
    {
        int ans;
        ans=a*b*c;
```

```

        System.out.println("The value of ans= "+ans);
    }

}

class MultiLvlInherit
{
    public static void main(String args[])
    {
        A obj_A=new A();
        B obj_B=new B();
        C obj_C=new C();
        obj_C.set_a(10);
        obj_C.set_b(20);
        obj_C.set_c(30);
        obj_C.show_a();
        obj_C.show_b();
        obj_C.show_c();
        obj_C.mul();
    }
}

```

**Output**

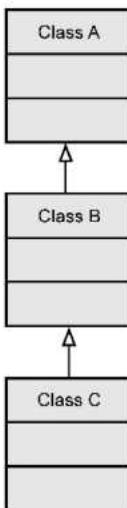
The value of a= 10  
 The value of b= 20  
 The value of c= 30  
 The value of ans= 6000

**Review Question**

- What is inheritance? How multilevel inheritance is achieved in Java? Explain with suitable example.

**SPPU : May-19, Marks 7****4.9 Constructor Call Sequence**

- A constructor invokes its superclass's constructor explicitly and if such explicit call to superclass's constructor is not given then compiler makes the call using **super()** as a **first** statement in constructor.
- Normally a superclass's constructor is called before the subclass's constructor. This is called **constructor chaining**.
- Thus the calling of constructor occurs from **top to down**.
- In the following Java program, there are three classes namely: A, B and C.

**Fig. 4.9.1 Constructor chaining**

The constructor chaining can be illustrated by following Java program.

#### **Java Program[C.java]**

```

class A
{
public A()
{
    System.out.println("1. Statement in class A");
}
public B(int i)
{
    System.out.println(i+". Statement in class B");
}
}

public class C extends B
{
public static void main(String arg[])
{
    new C();
}
public C()
{
    System.out.println("4. Statement in class C");
}
}
  
```

**Output**

```
F:\test>javac C.java
F:\test>java C
1. Statement in class A
2. Statement in class B
3. Statement in class B
4. Statement in class C
```

**Program Explanation**

- When the constructor for derived class **C** is invoked by **new C()** statement then the constructor for its superclass **B** is invoked.
- In turn, class **B** invokes the constructor for class **A**. This is called constructor chaining.

**4.10 Method Overriding****SPPU : Dec.-17, May-19, Marks 2**

**Definition :** Method overriding is a mechanism in which a subclass inherits the methods of superclass and sometimes the subclass modifies the implementation of a method defined in superclass.

- The method of superclass which gets modified in subclass has the **same name** and **type signature**.
- The overridden method must be called from the subclass.
- Consider following Java Program, in which the method(named as **fun**) in which **a** is assigned with some value is modified in the derived class. When an overridden method is called from within a subclass, it will always refer to the version of that method re-defined by the subclass. The version of the method defined by the superclass will be hidden.

**Java Program[OverrideDemo.java]**

```
class A
{
    int a=0;
    void fun(int i)
    {
        this.a=i;
    }
}
class B extends A
{
    int b;
    void fun(int i)
```

```

    {
        int c;
        b=20;
        super.fun(i+5);
        System.out.println("value of a:"+a);
        System.out.println("value of b:"+b);
        c=a*b;
        System.out.println("The value of c= "+c);
    }
}

class OverrideDemo
{
    public static void main(String args[])
    {
        B obj_B=new B();
        obj_B.fun(10);//function re-defined in derived class
    }
}

```

**Output**

```

F:\test>javac OverrideDemo.java
F:\test>java OverrideDemo
value of a:15
value of b:20
The value of c= 300

```

**Program Explanation**

In above program, there are two class - **class A** and **class B**. The class **A** acts as a superclass and the class **B** acts as a subclass. In class **A**, a method **fun** is defined in which the variable **a** is assigned with some value. In the derived class **B**, we use the same function name **fun** in which, we make use of **super** keyword to access the variable **a** and then it is multiplied by **b** and the result of multiplication will be printed.

**Rules to be followed for method overriding**

1. The private data fields in superclass are not accessible to the outside class. Hence the method of superclass using the private data field cannot be overridden by the subclass.
2. An instance method can be overridden only if it is accessible. Hence private method cannot be overridden.
3. The static method can be inherited but cannot be overridden.

4. Method overriding occurs only when the name of the two methods and their type signatures is same.

#### Difference between Method Overloading and Method Overriding

Method Overloading	Method Overriding
The method overloading occurs at <b>compile time</b> .	The method overriding occurs at the run time or <b>execution time</b> .
In case of method overloading <b>different number of parameters</b> can be passed to the function.	In function overriding the <b>number of parameters</b> that are passed to the function are <b>the same</b> .
The overloaded functions may have <b>different return types</b> .	In method overriding all the methods will have <b>the same return type</b> .
Method overloading is performed <b>within a class</b> .	Method overriding is normally performed between two classes that have <b>inheritance relationship</b> .

#### Review Question

1. State the use of the following constructs in Java with example : method overriding.

SPPU : Dec.-17, May-19, Marks 2

#### 4.11 Polymorphism

SPPU : Dec.-17, May-18, Nov.-18, 19, Marks 7

**Definition :** Polymorphism is a mechanism which allows to have many forms of the method having the same name.

That means, a method A0 may take an instance of an object of one class and may execute its method or the same method A0 may take another instance of an object and may execute the method belonging to another class.

In the following Java program we have taken the superclass A and build a multilevel inheritance.

Using object of corresponding class corresponding method is invoked by the same function fun.

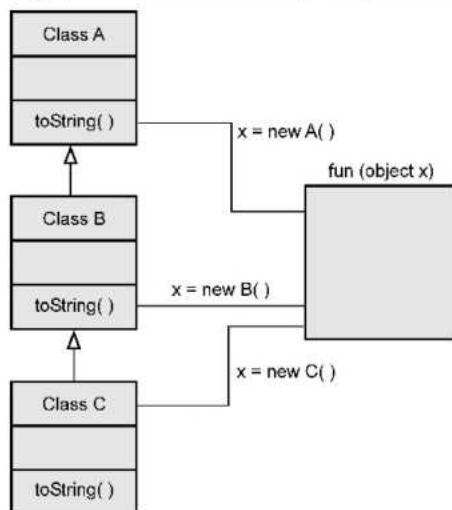


Fig. 4.11.1 Polymorphism

The implementation is given by following Java program -

#### Java Program[PolymorphismDemo.java]

```

class A extends Object
{
    public String toString()
    {
        return "A";
    }
}
class B extends A
{
    public String toString()
    {
        return "B";
    }
}
class C extends B
{
    public String toString()
    {
        return "C";
    }
}
  
```

```

}

public class PolymorphismDemo
{
    public static void main(String[] args) {
        fun(new C());//invokes the method toString() of class C
        fun(new B());//invokes the method toString() of class B

        fun(new A());//invokes the method toString() of class A
    }

    public static void fun(Object x) {
        System.out.println(x.toString());
    }
}

```

**Output**

F:\test>javac PolymorphismDemo.java

F:\test>java PolymorphismDemo  
C  
B  
A

**Program Explanation**

- 1) The same function **fun** is used to execute the methods of various class.
- 2) The selection of the method of the class (i.e. `toString` method) depends upon the instance of the object which is passed to the function **fun**. Which implementation is to be used is determined dynamically by the Java Virtual Machine. This mechanism is called **dynamic binding**.

**4.11.1 Dynamic Method Dispatch**

- The dynamic method dispatch is also called as **runtime polymorphism**.
- During the run time polymorphism, a call to overridden method is resolved at run time.
- The overridden method is called using the reference variable of a super class(or base class). The determination of which method to invoke is done using the object being referred by the reference variable.

Following is a simple Java program that illustrates the Run time polymorphism.

```

class Base
{
void display()
{
    System.out.println("\n Base Method Called");
}
}
class Derived extends Base
{
void display() //overridden method
{
    System.out.println("\n Derived Method Called");
}
}
public class RunPolyDemo
{
    public static void main(String args[])
    {
        Base obj=new Derived(); //obj is reference to base class
                               // which is referred by the derived class
        obj.display(); //method call which is determined at run time
    }
}

```

### Output

D:\test>javac RunPolyDemo.java

D:\test>java RunPolyDemo

Derived Method Called

### Difference between Static and Dynamic Dispatch

Or

### Difference between Compile Time Polymorphism and Run Time Polymorphism

Sr.No.	Static Dispatch	Dynamic Dispatch
1.	The function call resolution occurs at compile time.	The function call resolution occurs at run time.
2.	It uses type information using which the function call resolution takes place.	It uses actual object of a class using which the function call resolution takes place.
3.	The private, static and final methods are resolved during static dispatch.	Only virtual methods are resolved during dynamic dispatch.

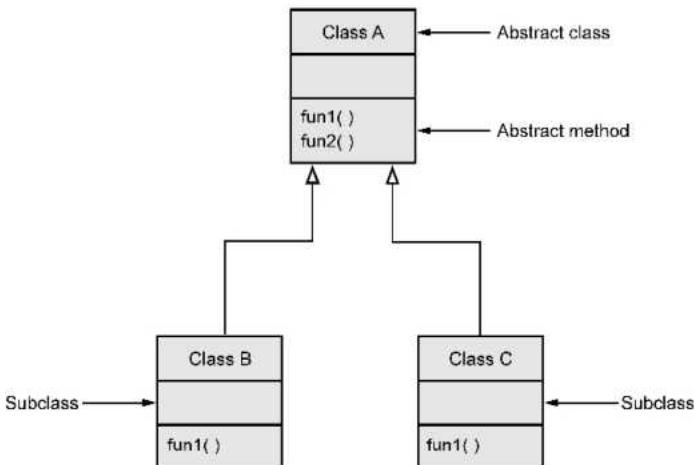
4.	It is not flexible.	It is flexible.
5.	Execution is slower	Faster execution.

**Review Question**

1. Explain the concept of dynamic dispatch while overriding method in inheritance. Give example and advantages of doing so. **SPPU : Dec.-17, Marks 5**
2. Difference Static and Dynamic dispatch **SPPU : May-18, Marks 3**
3. State the difference in compile time and runtime polymorphism. Show how this is implemented in Java for overriding of methods **SPPU : Nov.-18, Marks 6**
4. What is Polymorphism ? Which type of polymorphism is method overriding ? Demonstrate method overriding in Java. **SPPU : Nov.-19, Marks 7**

**4.12 Abstract Classes****SPPU : Dec.-17, May-19, Marks 2**

- In the inheritance hierarchy, the classes become more specific as we move down towards the subclasses and if we move up in this hierarchy classes become more general.
- The superclass is supposed to be the most general or less specific. Sometimes superclass is so general and less specific that it does nothing but lists out only common features of other classes. Then such a superclass is referred as **abstract class**.

**Fig. 4.12.1 Abstract class**

- For example - In the following Java program we have created three classes - class A is a superclass containing two methods, the class B and class C are inherited from class A. The class A is an abstract class because it contains one abstract method **fun1()**. We have defined this method as abstract because, its definition is overridden in the subclasses B and C, another function of class A that is **fun2()** is a normal function. Refer Fig. 4.12.1.

**Java Program[AbstractClsDemo.java]**

```
abstract class A
{
    abstract void fun1();
    void fun2()
    {
        System.out.println("A:In fun2");
    }
}

Class B extends A
{
    void fun1()
    {
        System.out.println("B:In fun1");
    }
}

class C extends A
{
    void fun1()
    {
        System.out.println("C:In fun1");
    }
}

public class AbstractClsDemo
{
    public static void main(String[] args)
    {
        B b=new B();
        C c=new C();
        b.fun1(); //invoking the overridden method of class B
        b.fun2();
        c.fun1(); //invoking the overridden method of class C
        c.fun2();
    }
}
```

This method is so abstract that it has no definition body.  
This is an abstract method.

**Output**

```
F:\test>javac AbstractClsDemo.java
F:\test>java AbstractClsDemo
B:In fun1
A:In fun2
C:In fun1
A:In fun2
```

**Program Explanation**

In above program,

- 1) The class **A** is a superclass. It is an abstract class as well. The name of this class is preceded by the keyword **abstract**. This class is abstract because it contains an abstract method **fun1**. The method is called abstract because it does not have any definition body. Note that the abstract method should be declared with the keyword **abstract**.
- 2) There are two classes **B** and **C** which are subclasses of superclass **A**. The function definition **fun1** is overridden in these classes.
- 3) In the **main function** we can access the methods of the subclasses by instantiating their objects. That is why we have created **b as an object of class B** and **c as an object of class C**. Using these objects appropriate **fun1** can be invoked. Note that the **fun2** will always be from class **A** even if we call it using the object of class **B** or **C**.
- 4) If we write a statement **A a=new A0** i.e. if we instantiate the abstract class then it will generate compiler error. That means the **abstract classes can not be instantiated**.

**Difference between Abstract Class and Concrete Class**

<b>Abstract class</b>	<b>Concrete class</b>
The <b>abstract</b> keyword is used to define the abstract class.	The keyword <b>class</b> is used to define the concrete class.
The abstract class have <b>partial or no implementation</b> at all.	The concrete class contains the data members and member functions defined within it.
Abstract class <b>can not be instantiated</b> .	Concrete class are usually instantiated in order to access the belonging data members and member functions.

Abstract classes may contain <b>abstract methods</b> .	Concrete classes contain concrete method i.e. with code and functionality. Concrete class can not contain abstract method.
Abstract classes <b>need to be extended</b> in order to make them useful.	Concrete classes may or may not be extended.
Abstract classes always act as a <b>parent class</b> .	Concrete class can be parent or child or neither of the two.

**Review Question**

1. State the use of the following constructs in Java with example : Abstract class declaration

SPPU : Dec.-17, May-19, Marks 2

**4.13 Object Class**

- In Java there is a special class named **Object**. If no inheritance is specified for the classes then all those classes are subclass of the **Object** class.
- In other words, **Object** is a superclass of all other classes by default. Hence it is specified as -  

```
public class A { ... } is equal to public class A extends Object { ... }
```
- The most commonly method of object class being used is **toString()** method. Let us discuss these it with the help of example -

**The **toString** Method of Object Class**

- The **toString** method returns the String type value. Hence the signature (syntax) of this method is -

```
public String toString()
```

- If we invoke the **toString** method, by default then it returns a string which describes the object. This returned string consists of the character "@" and object's memory address in hexadecimal form. But we can override the **toString** method.

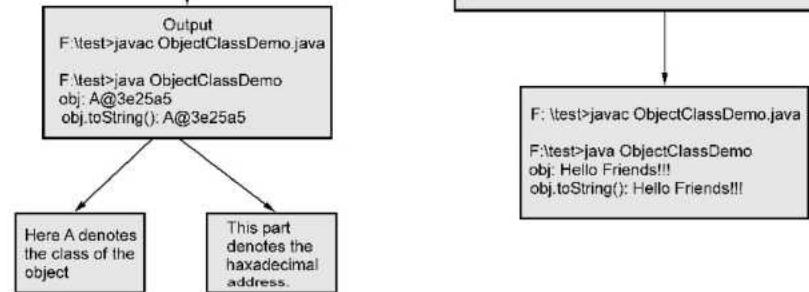
- Following are the two illustrations that support the above discussion -

**Illustration 1 :**

```
class A extends Objects
{
}
class B extends A
{
}
class ObjectClassDemo
{
    public static void main(String args[])
    {
        A obj=new A();
        System.out.println("obj: "+obj);
        System.out.println("obj.toString(): "+obj.toString());
    }
}
```

**Illustration 2 :**

```
Class A extends Object
{
    public String toString()//Method is overridden
    {
        String str="Hello Friends!!!";
        return str;
    }
}
Class B extends A
{
}
Class ObjectClassDemo
{
    public static void main(String args[])
    {
        A obj=new A();
        System.out.println("obj: "+obj);
        System.out.println("obj.toString(): "+obj.toString());
    }
}
```

**Fig. 4.13.1**

The above method `toString` is overridden in second illustration, in order to print the desired string.

## Part II : Packages and Interfaces

### 4.14 Defining a Package

- Package is defined using a keyword `package`.

- The syntax for declaring the package is

```
package name_of_package
```

- This package is defined at the beginning of the program.
- The package statement defines the name space in which the classes are stored. If we omit the package then the default classes are put in the package that has no name.

- Basically Java creates a directory and the name of this directory becomes the package name.
- For example - In your program, if you declare the package as -

```
package My_Package;
```

here we must create the directory name **My\_Package** in the current working directory and the required classes must be stored in that directory. Note that the name of the package and the name of the directory must be the same. This name is **case sensitive**.

- We can create hierarchy of packages. For instance if you save the required class files in the subfolder **MyPkg3** and the path for this subfolder is C:\MyPkg1\MyPkg2\MyPkg3 then the declaration for the package in your java program will be -

```
package MyPkg1.MyPkg2.MyPkg3;
```

## 4.15 Finding Packages

In this section we discuss **how to develop a program** which makes use the classes from other package.

**Step 1 :** Create a folder named **My\_Package**.

**Step 2 :** Create one class which contains two methods. We will store this class in a file named **A.java**. This file will be stored in a folder **My\_Package**. The code for this class will be as follows-

### Java Program[A.java]

```
package My_Package; //include this package at the beginning
public class A
{
    int a;
    public void set_val(int n)
    {
        a=n;
    }
    public void display()
    {
        System.out.println("The value of a is: "+a);
    }
}
```

Note that the class and the methods defined must be public.

Note that this class contains two methods namely- **set\_val** and **display**. By the **set\_val** method we can assign some value to a variable. The **display** function displays this stored value.

**Step 3 :** Now we will write another java program named **PackageDemo.java**. This program will use the methods defined in **class A**. This source file is also stored in the subdirectory **My\_Package**. The java code for this file is-

#### Java Program[**PackageDemo.java**]

```
import My_Package.A; //The java class A is referenced here by import statement
class PackageDemo
{
    public static void main(String args[]) throws NoClassDefFoundError
    {
        A obj=new A(); //creating an object of class A
        obj.set_val(10); //Using the object of class A, the methods present
        obj.display(); //in class A are accessed
    }
}
```

**Step 4 :** Now, open the command prompt and issue the following commands in order to run the package programs

```
D:\>set CLASSPATH=.;D\;
D:\>cd My_Package
D:\My_Package>javac A.java
D:\My_Package>javac PackageDemo.java
D:\My_Package>java PackageDemo
The value of a is: 10
D:\My_Package>
```

Setting the class path

Creating the classes A.class and package Demo.Class files

Running the test program which uses the class **A.Class** stored in another file

#### 4.16 Adding Class to a Package

We can add some class in the existing package. For that purpose following steps are followed-

**Step 1 :** Write a class (in a separate java file )which is to be added in the existing package.  
Suppose I want to add the class B in the existing package **My\_Package**. I will create a **B.java** file and store the code in it as follows-

#### Java Program[B.java]

```
package My_Package; //include this package at the beginning
public class B
{
```

```
public void show()
{
    System.out.println("class B is now added in the package");
}
```

**Step 2 :** The test program **PackageDemo.java** can be edited as follows -

[Note that edited code is in bold]

#### Java Program[**PackageDemo.java**]

```
import My_Package.A;
import My_Package.B; //reference to a B class by import statement
class PackageDemo
{
    public static void main(String args[]) throws NoClassDefFoundError
    {
        A Aobj=new A();
        B Bobj=new B(); //creating an object for class B
        Aobj.set_val(10);
        Aobj.display();
        Bobj.show(); //using the object of B class the corresponding function can be accessed.
    }
}
```

**Step 3 :** Issue the following commands to execute the package with newly added class.

D:\My\_Package>javac A.java

D:\My\_Package>javac B.java

D:\My\_Package>javac PackageDemo.java

D:\My\_Package>java PackageDemo

The value of a is: 10

class B is now added in the package

### 4.17 CLASSPATH

- The packages are nothing but the directories. For locating the specified package the java run time system makes use of current working directory as its starting point. Thus if the required packages is in the current working directory then it will be found. Otherwise you can specify the directory path setting the **CLASSPATH** statement.
- For instance- if the package name **My\_Package** is present at prompt D:\> then we can specify

```
set CLASSPATH=.;D:\;
```

D:\&gt;cd My\_Package

D:\My\_Package\&gt; Now you can execute the required class files from this location

**4.18 Access Protection****SPPU : Nov.-18, Marks 6**

The effect of access specifiers for class, subclass or package is enlisted below -

Specifier	Class	Subclass	Package
private	Yes	-	-
protected	Yes	Yes	Yes
public	Yes	Yes	Yes

For example, if some variable is declared as protected, then the class itself can access it, its subclass can access it, and any class in the same package can also access it. Similarly if the variable is declared as private then that variable is accessible by that class only and its subclass can not access it.

**Review Question**

- What is use of a package in Java ? How CLASSPATH helps to store and retrieve the classes ? How is access protection provided for packages ?

**SPPU : Nov.-18, Marks 6****4.19 Importing Packages**

- All the standard classes in Java are stored in named packages.
- There is no standard class present in Java which is unnamed. But it is always complicated to write the class name using a long sequence of packages containing dot operator. Hence the import statement is needed.
- The **import** statement can be written at the beginning of the Java program, using the keyword **import**.
- For example -

import java.io.File

or

import java.io.\*

Either a class name can be specified explicitly or a \* can be used which indicated that the Java compiler should import the entire package. The \* increases the compiler time, hence it is a good practice to use specific class wherever possible.

**4.20 Nested Packages**

**Definition:** Nested Package is one package inside another package.

**Creating Nested Package****Step 1 :** Create a folder named **My\_Pkg1****Step 2 :** Create one class which contains two methods. We will store this class in a file named **A.java**. This file will be stored in a folder **Pkg1**. The code for this class will be as follows -**Java Program[A.java]**

```
package Pkg1; //include this package at the beginning
public class A
{
    int a;
    public void set_val(int n)
    {
        a=n;
    }
    public void display()
    {
        System.out.println("The value of a is: "+a);
    }
}
```

Note that the class, and the methods defined must be **public**

**Step 3 :** Write a class (in a separate java file) which is to be added in the nested package. Create another directory named **Pkg2** inside the directory **Pkg1**. Then add the class **B** in the existing package **Pkg2**. I will create a **B.java** file and store the code in it as follows -**Java Program[B.java]**

```
package Pkg1.Pkg2; //include nested package at the beginning
public class B
{
    public void show()
    {
        System.out.println("class B is now added in the package");
    }
}
```

**Step 3 :** Now we will write another java program named **PackageDemo.java**. This program will use the methods defined in class **A**. This source file is also stored in the subdirectory **Pkg1**. The java code for this file is -**Java Program[PackageDemo.java]**

```
import Pkg1.A;
import Pkg1.Pkg2.B; //reference to a B class by import statement
```

```

class PackageDemo
{
    public static void main(String args[]) throws NoClassDefFoundError
    {
        A Aobj=new A();
        B Bobj=new B(); //creating an object for class B
        Aobj.set_val(10);
        Aobj.display();
        Bobj.show(); //using the object of B class the corresponding function can be accessed.
    }
}

```

**Step 4 :** Issue the following commands to execute the package with newly added class.

```

C:\Users\admin>d:
D:\> set CLASSPATH=.;D\;
D:\> cd Pkg1
D:\Pkg1>javac A.java
D:\Pkg1>javac PackageDemo.java
D:\Pkg1>cd Pkg2
D:\Pkg1\Pkg2>javac B.java
D:\Pkg1\Pkg2>cd..
D:\Pkg1>java PackageDemo
The value of a is: 10
class B is now added in the package

```

### Review Question

1. Write a Java program to implement nested packages.

### 4.21 JAVA API Package

- We have used various classes and interfaces from java library packages using the import statement.
- If we know the specific class that needs to be used in our program from a particular package then it can be mentioned in **import** statement.
- For example -

```
import java.net.InetAddress
```

- We can also specify it as

```
import java.net.*
```

Here \* means all the classes from **java.net** package.

Some of the useful packages are enlisted as below -

### 1. Java.io

**Purpose :** This package provides useful functionalities for performing input and output operations through data streams or through file system.

#### Interfaces :

Name	Description	Some methods
DataInput	This interface provides methods for reading the bytes from binary stream and converting it into any of the primitive data types of Java.	<p><i>char readChar()</i> - It reads input character and returns the character value.</p> <p><i>boolean readBoolean()</i>- It reads input byte if the input is non zero then it returns true and if input is zero then it returns false</p> <p><i>String readLine()</i>- It reads the next line of the input text</p> <p><i>readInt(),readDouble(),readFloat()</i></p> <p>- These functions are for reading the input and returning int,double and float values respectively.</p>
DataOutput	This interface provides the methods for converting data from Java primitive types to series of bytes.	<p><i>Void write(byte bt[])</i>- This method is for writing the bytes in the array 'bt' to output stream.</p> <p><i>void writeBoolean(boolean val)</i></p> <p>-This method is for writing the boolean value 'val' to output stream</p> <p>Similarly methods <i>writeChar(),writeInt, writeDouble()</i> methods are for writing the corresponding type of data to the output stream.</p>
ObjectInput	This interface is for reading the objects.	<p><i>int read()</i>-It reads the byte of data</p> <p><i>int read(byte bt[])</i>-It reads into array of bytes 'bt'</p> <p><i>object readObject()</i>- It reads and returns an object</p> <p><i>skip(long k)</i>-skips k bytes of input.</p> <p><i>close()</i> method is closing the stream</p>
ObjectOutput	This interface is for writing the object.	<p><i>flush()</i>- This method is for flushing the stream.</p> <p><i>write(byte bt[])</i>- This method is for writing an array of bytes.</p>

**Classes :**

Name	Description
BufferedInputStream	For creating buffered input stream
BufferedOutputStream	For creating buffered output stream
BufferedReader	Reads the text from character input stream by buffering the characters.
BufferedWriter	Writes the text to character output stream by buffering the characters.
ByteArrayInputStream	For storing the bytes read from the stream into the internal buffer this class is used.
ByteArrayOutputStream	For writing the bytes into the buffer as an output stream this class is useful.
DataInputStream	This class helps to read data of primitive data types from underlying input stream.
DataOutputStream	This class helps to write data of primitive data types to an output stream.
FileReader	This class is for reading the character files.
FileWriter	This class is for writing the character files.
InputStream	This class is for providing the input stream of bytes.
InputStreamReader	This class is responsible for reading the bytes and to convert to character type.
OutputStream	This class is for providing the input stream of bytes.

**Exceptions :**

Name	Description
IOException	This is the most commonly used exception for raising the exceptions for input or output.
FileNotFoundException	If the file specified by some pathname is not obtained then this exception must be thrown.

**2. Java.net**

**Purpose :** This package is for providing the useful classes and interfaces for networking applications which are used in sockets and URL.

**Interfaces :**

Name	Description	Some methods
ContentHandlerFactory	This interface is for defining the factory for content handlers.	<i>createContentHandler (String MIME_type)</i> - creates ContentHandler.
SocketImplFactory	This interface is for defining the factory for implementing the sockets.	<i>createSocketImpl()</i> - creates a new instance for implementing socket.
URLStreamHandlerFactory	For implementing URL stream protocol handlers the factory can be defined by this interface.	<i>createURLStreamHandler (String p)</i> - For specific protocol 'p' a new URLStreamHandler instance can be created by this method.

**Classes :**

Name	Description
ContentHandler	This class is a superclass of all the classes that read the data from a class URLConnection. It also builds the appropriate local object based on MIME types.
DatagramSocket	This class represents the Datagram Socket(UDP socket)
DatagramPacket	This class represents the Datagram Packet (UDP packets for containing data.)
InetAddress	This class represents the IP address.
InetSocketAddress	The IP socket Address is a combination of IP address and port number. To implement such IP socket address this class is used.
ServerSocket	For implementing server-side sockets this class is useful.
Socket	For implementing client side sockets this class is useful.
SocketAddress	This class helps to represent the socket address without specification of protocol.
URL	This class is for creating a reference of Uniform Resource Locator which points to WWW.
URI	This class provides the object of Uniform Resource Identifier.
URLConnection	For establishing a communication between application program and URL this class is used.

**Exceptions :**

Name	Description
UnknownHostException	If the IP address of corresponding Host could not be located then this kind of error is indicated.
MalformedURLException	This exception is for indicating the bad URL.
BindException	If a failure occurs while binding socket to local address or port then this exception must be thrown.
ConnectException	If connection between socket and remote address and port could not get established then this exception must be thrown.
ProtocolException	This exception is for raising error in underlying protocol.
SocketException	This exception is for raising error in socket communication.

**3. java.lang**

**Purpose :** This package provides core classes and interfaces used in design of Java language.

**Interfaces :**

Name	Description	Some methods
charSequence	This interface is for getting sequence of characters. The read-only access to character sequence is provided by this method.	int charAt(int i)- It returns the position of the character at specified index.String toString()- This method returns the sequence of characters.int length() -It returns length of string
Runnable	A class whose instance can be executed by a thread implements a Runnable interface.	

**Classes :**

Name	Description
Boolean	It's a wrapper class for Boolean values.
Byte	It's a wrapper class for byte data.
Character	It's a wrapper class character values.
Class	For representing classes and interfaces in the Java application this Class is used.

ClassLoader	For handling loading of the classes this object is used.
Compiler	This class is for providing support compiler related activities.
Double	It's a wrapper class for Double values.
Float	It's a wrapper class for Float values.
Integer	It's a wrapper class for integer values.
Long	It's a wrapper class for Long values.
Math	This class provides the support for the operations such as square root, exponentiation, and logarithm.
Object	The class Object is at the top of class hierarchy.
Runtime	For linking the Java application program with its runtime environment the instance of class Runtime is used.
StringBuffer	For handling the dynamic sequence of characters StringBuffer class is used.
String	It's a wrapper class for String values.
System	This is the most commonly used class in which many important fields and methods are defined. For example err, in and out are the most commonly used fields in this class.
Thread	A Thread class creates a thread for a separate execution of the program.

**Exceptions :**

Name	Description
ArithmaticException	For handling an arithmetic error this exception must be thrown.
ArrayIndexOutOfBoundsException	When there is a chance of accessing an illegal index this exception must be thrown.
ClassNotFoundException	If by specified name of the class, we can not obtain the desired class then this kind of exception must be thrown.
Exception	For raising a general error condition this kind of Exception is used.
IllegalAccessException	If we try to access a field or a method of a class which itself is not accessible then this kind of exception occurs.
IndexOutOfBoundsException	If an index of set or vector or a string is accessed and if it is out of scope then this exception occurs.

**4. java.util**

**Purpose :** Some commonly used utilities are random number generation, event model, date and time facilities, system properties are stored in util package.

**Interfaces :**

Name	Description	Some methods
Collection	The collection represents the group of objects called elements. These elements are related to the interfaces such as Set and List.	<i>boolean add(Object obj)</i> - for adding the specific element in the collection this method is used. <i>void clear()</i> - This method is for clearing all the elements from the collection. <i>boolean equals(object obj)</i> - For comparing the specific object with some element of the collection this method is used. If two objects are equal then it returns true. <i>int size()</i> - To get the total number of elements in the collection this method is used. <i>boolean isEmpty()</i> - If the collection has no element then it returns true otherwise false.
EventListener	This interface is used to handle the events.	
Map	In order to map the keys to the values this interface is used.	The <i>add()</i> , <i>clear()</i> , <i>size()</i> , <i>equals()</i> , <i>isEmpty()</i> are some methods that can be used by map. Similarly, <i>Object get(Object key)</i> -For retrieving the value associated with some key, this method is used. <i>Set keySet()</i> - To display all the keys in the map this method is used. <i>Object get (Object key, Object val)</i> -To associate the value with the corresponding key this method is used.
Set	The set is a collection of elements in which there is no duplicate element.	Similar to the methods in Map.

**Classes :**

Name	Description
AbstractMap	This class provides the skeleton for the Map interface.
AbstractSet	This class provides the skeleton for the Set interface.
Date	To represent the current system date this class is useful.
Dictionary	A class that associates the keys with the corresponding values.
Calendar	To display the current system date this class is used.
HashTable	This class is for hash table representation.
Stack	For a stack data structure this class is used.
Vector	For representing the array of objects which can be further expanded is this is the class.
Random	For generating random numbers this class is used.

**Exceptions :**

Name	Description
EmptyStackException	For handling the exception of empty stack this exception must be raised.
NoSuchElementException	To indicate that there exists no such element which is expected, this error must be thrown.

**5. java.awt**

**Purpose :** This package contains all the interfaces and classes that are required for creating the graphical user interface(GUI).

**Interfaces :**

Name	Description	Some methods
ActiveEvent	An interface for events which can dispatch themselves in another thread. For example, system code should use an ActiveEvent to invoke user code securely.	<i>void dispatch()</i> - This method is used to dispatch the events.
Adjustable	This interface is for adjustable numerical value which is contained within some range.	<i>void addAdjustmentListener(AdjustmentListenerL)</i> -For listening adjustment events the listener is added by this method. <i>int getMinimum()</i> - To obtain the minimum value of adjustable object this method is used. <i>int getValue()</i> -To obtain the value of adjustable object this method is used. <i>void setMaximum(int Max)</i> - This method is for setting maximum value to adjustable object. <i>void setMinimum(int Min)</i> - This method is for setting minimum value to adjustable object.
LayoutManager	For the container class which is for layout this interface is used.	<i>void layoutContainer(Container Root)</i> - For laying out the specific container this method is used.Similarly <i>addLayoutComponent()</i> and <i>removeLayoutComponent()</i> methods are used for adding or removing specific components from the layout.
Paint	This interface is for deciding the color pattern for Graphics2D operations.	

PrintGraphics	Its an abstract class for Print Graphics context.
Shape	This interface defines some graphical shape of an object.
Stroke	This interface is used to define the outline of the shape object.

**Classes :**

Name	Description
AWTEvent	This is the topmost event of all the awt events.
BorderLayout	This class defines the layout in which there are five components such as north, south, east, west and center.
Button	This class creates a labelled button.
Component	Its an object for graphical representation. This graphical representation can be displayed on the screen so that the user can interact with it.
Canvas	This class is for creating the blank rectangular area.
Container	It's a generic AWT container object that contains components java.awt.Component └─ java.awt.Container
Graphics	Its an abstract class responsible for creating a GUI.
Graphics2D	Its an extended class of Graphics for providing fine control over the co-ordinate systems and geometry.
Image	Its an abstract class for creating graphical images.
Window	It's an object which has no borders and no menubars.

**Exceptions :**

Name	Description
AWTException	If any error related to Abstract Window Toolkit has occurred then it can be handled by raising this exception.

**6. java.applet**

**Purpose :** The java.applet package contains all the necessary classes and interfaces required for handling applets.

**Interfaces :**

Name	Description	Some methods
AppletContext	This interface is used for the applet's environment	<i>Applet getApplet(String Str)-</i> finds and returns the applet given by the name Str. <i>Image getImage(URL url)</i> - Returns an image specified by the URL object. <i>void showDocument(URL url)</i> - It displays the document which can be viewed using the given URL..
AppletStub	This interface is for implementing the applet viewer.	<i>URL getCodeBase()</i> -This method is for obtaining the base URL. <i>URL getDocumentBase()</i> - This method returns the URL of the document which contains an applet.
AudioClip	For playing the sound clip this interface is used.	<i>void play()</i> -For playing the sound clip this method is used. <i>void stop()</i> -For stopping the sound clip this method is used. <i>void loop()</i> - This method loops the playing of audio clip.

**Classes :**

Name	Description
Applet	It's a base class for creating an applet. The applet is a small program which cannot run on its own but it runs if embedded in some application.

**4.22 Interfaces****SPPU : Dec.-17, May-18, 19, Marks 7**

- The interface can be defined using following syntax

```
access_modifier interface name_of_interface
{
    return_type method_name1(parameter1,parameter2,...parametern);
    ...
    return_type method_name1(parameter1,parameter2,...parametern);
    type static final variable_name=value;
    ...
}
```

- The access\_modifier specifies the whether the interface is public or not. If the access specifier is not specified for an interface then that interface will be accessible to all the classes present in that package only. But if the interface is declared as public then it will be accessible to any of the class.
- The methods declared within the interface have no body. It is expected that these methods must be defined within the class definition.

**Difference between Class and Interface**

Class	Interface
The class is denoted by a keyword <b>class</b>	The interface is denoted by a keyword <b>interface</b>
The class contains data members and methods. But the methods are defined in class implementation. Thus class contains an executable code.	The interfaces may contain data members and methods but the methods are not defined. The interface serves as an outline for the class.
By creating an instance of a class the class members can be accessed.	You can not create an instance of an interface.
The class can use various access specifiers like public, private or protected.	The interface makes use of only public access specifier.
The members of a class can be constant or final.	The members of interfaces are always declared as final.

**Review Questions**

1. State two major differences in class and an interface

**SPPU : Dec.-17, May-18, Marks 4**

2. What is meant by Packages and interfaces in Java? Explain it with suitable example

**SPPU : May-19, Marks 7**

**4.23 Variables in Interfaces**

**SPPU : Nov.-18, Marks 7**

- The variables can be assigned with some values within the interface. They are implicitly final and static. Even if you do not specify the variables as final and static they are assumed to be **final** and **static**.
- The members of interface are static and final because –
  - 1) The reason for being static** – The members of interface belong to interface only and not object.
  - 2) The reason for being final** – Any implementation can change value of fields if they are not defined as final. Then these members would become part of the implementation. An interface is pure specification without any implementation.

**Review Question**

1. What is an Interface ? What is the difference in class and an Interface ? What is use of declaring variables in an interface in Java ?

**SPPU : Nov.-18, Marks 7**

## 4.24 Extending Interfaces

- Interfaces can be extended similar to the classes. That means we can derive subclasses from the main class using the keyword **extend**, similarly we can derive the subinterfaces from main interfaces by using the keyword **extends**.
- The syntax is

```
interface Interface_name2 extends interface_name1
{
    ...
    ...
    ...
}      Body of interface
```

- For example

```
interface A
{
    int val=10;
}
interface B extends A
{
    void print_val();
}
```

That means in **interface B** the **display** method can access the value of variable **val**.

Similarly more than one interfaces can be extended.

```
interface A
{
    int val=10;
}
interface B
{
    void print_val();
}
interface C extends A,B
{
    ...
    ...
}
```

Even-though methods are declared inside the interfaces and sub-interfaces, these methods are not allowed to be defined in them. Note that methods are defined only in the classes and not in the interfaces.

## 4.25 Creation and Implementation of Interface

- It is necessary to create a class for every interface.
- The class must be defined in the following form while using the interface

```
class Class_name extends superclass_name
implements interface_name1,interface_name2, ...
{
    //body of class
}
```

- Let us learn how to use interface for a class

**Step 1 :** Write following code and save it as **my\_interface.java**

```
public interface my_interface
{
    void my_method(int val);
}
```

Do not compile this program. Simply save it.

**Step 2 :** Write following code in another file and save it using **InterfaceDemo.java**

**Java Program[InterfaceDemo.java]**

```
class A implements my_interface
{
    public void my_method(int i)
    {
        System.out.println("\n The value in the class A: "+i);
    }
    public void another_method() //Defining another method not declared in interface
    {
        System.out.println("\nThis is another method in class A");
    }
}
class InterfaceDemo
{
    public static void main(String args[])
    {
        my_interface obj=new A();
        //or A obj=new A() is also allowed
        A obj1=new A();
        obj.my_method(100);
        obj1.another_method();
    }
}
```

Defining the method declared  
the interface

Object of class A is of type  
my\_interface. Using this  
object method can be  
accessed.

**Step 3 :** Compile the program created in step 2 and get the following output

```
F:\test>javac InterfaceDemo.java
F:\test>java InterfaceDemo
```

The value in the class A: 100

This is another method in class A

**Program Explanation :** In above program,

- 1) The interface **my\_interface** declares only one method i.e. **my\_method**.
- 2) This method can be defined by class **A**.
- 3) There is another method which is defined in class **A** and that is **another\_method**.

Note that this method is not declared in the interface **my\_interface**. That means, a class can define any additional method which is not declared in the interface.

## 4.26 Multiple Inheritance

SPPU : Nov.-19, Marks 7

**Definition :** Multiple inheritance is a mechanism in which the child class inherits the properties from more than one parent classes.

- For example

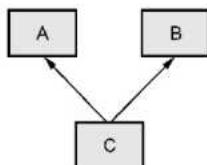


Fig. 4.26.1 Multiple Inheritance

- **Java does not support multiple inheritance** because it creates ambiguity when the properties from both the parent classes are inherited in child class or derived class. But it is supported in case of **interface** because there is no ambiguity as implementation is provided by the implementation class.

**Implementation :**

```
interface A
{
    void display1();
}

interface B
{
    void display2();
}
```

```

class C implements A,B
{
    public void display1()
    {
        System.out.println("From Interface A");
    }
    public void display2()
    {
        System.out.println("From Interface B");
    }
    public static void main(String args[])
    {
        C obj = new C();
        obj.display1();
        obj.display2();
    }
}

```

**Output**

From Interface A  
From Interface B

**Program Explanation : In above program,**

- 1) we have created both the parent interfaces and the derived class is using method declaration from the parent interface.
- 2) Note that the definition of these methods is present in the child class. Thus multiple inheritance is achieved with the help of interface.

**Review Question**

1. *Describe Inheritance. List and explain the different types of Inheritance. Demonstrate how Java supports Multiple Inheritance.*

**SPPU : Nov.-19, Marks 7**

**4.27 The Instance of Operator**

- While type casting the objects of the class, the object to be type casted must be an instance of the subclass.
- If some superclass is not an instance of subclass then the run-time exception **ClassCastException** occurs. Hence before doing type casting operation, we must know whether particular object is an instance of some class. This can be done using **instanceof** operator.
- The "Instanceof" is an operator used to test if an object is instance of some class.

- The syntax is

```
name_of_object instanceof class_name
```

Consider following example in which, instanceof operator is used before performing the downcasting.

**Java Program[instanceof.java]**

```
class LivingBodies extends Object
{
}
class Human extends LivingBodies
{
}
class Boy extends Human
{
}

public class instanceofDemo
{
    public static void main(String[] args)
    {
        Boy B1=new Boy();
        System.out.println("The value of object B1: "+B1);
        Human H=B1;
        if(H instanceof Boy)
        {
            System.out.println("It is now safe to downcast the object");
            Boy B2=(Boy)H;
            System.out.println("The value of object B2: "+B2);
        }
    }
}
```

**Output**

```
F:\test>javac instanceofDemo.java
```

```
F:\test>java instanceofDemo
```

```
The value of object B1: Boy@42e816
```

```
It is now safe to downcast the object
```

```
The value of object B2: Boy@42e816
```

Note that **instanceof** is a keyword in which every letter is in lowercase.

### Part III : Exception Handling

#### 4.28 Fundamentals

- Exception in Java is an indication of some **unusual event**. Usually it indicates the error.
- In Java, exception is handled using five keywords **try, catch, throw, throws** and **finally**.
- The Java code that you may think may produce exception is placed within the **try block**.

Let us see one simple program in which the use of try and catch is done in order to handle the exception *divide by zero*.

#### Java Program [ExceptionDemo.Java]

```
class ExceptionDemo
{
    public static void main(String args[])
    {
        try
        {
            int a,b;
            a=5;
            b=a/0;
        } } A try block which includes the
lines for which exception must
be raised
        catch(ArithmaticException e)
        {
            System.out.println("Divide by Zero\n");
        }
        System.out.println("...Executed catch statement");
    }
}
```

Exception is caught by catch block. A try-catch pair must exist

#### Output

Divide by Zero

...Executed catch statement

Inside a **try** block as soon as the statement: `b=a/0` gets executed then an arithmetic exception must be raised, this exception is caught by a **catch** block. Thus there must be a **try-catch pair** and catch block should be immediate follower of try statement. After execution of catch block the control must come on the next line. These are basically the exceptions thrown by java **runtime systems**.

**4.28.1 Benefits of Exception Handling**

Following are the benefits of exception handling -

1. Using exception the main application logic can be separated out from the code which may cause some unusual conditions.
2. When calling method encounters some error then the exception can be thrown. This avoids crashing of the entire application abruptly.
3. The working code and the error handling code can be separated out due to exception handling mechanism. Using exception handling, various types of errors in the source code can be grouped together.
4. Due to exception handling mechanism, the errors can be propagated up the method call stack i.e. problems occurring at the lower level can be handled by the higher up methods.

**4.28.2 Difference between Error and Exception**

Error	Exception
Errors can not be handled.	Exceptions can be handled using exception handler.
Program crashes or stops working when an error occurs.	Program reports user friendly message about the abnormal situation and exits gracefully.

**Review Question**

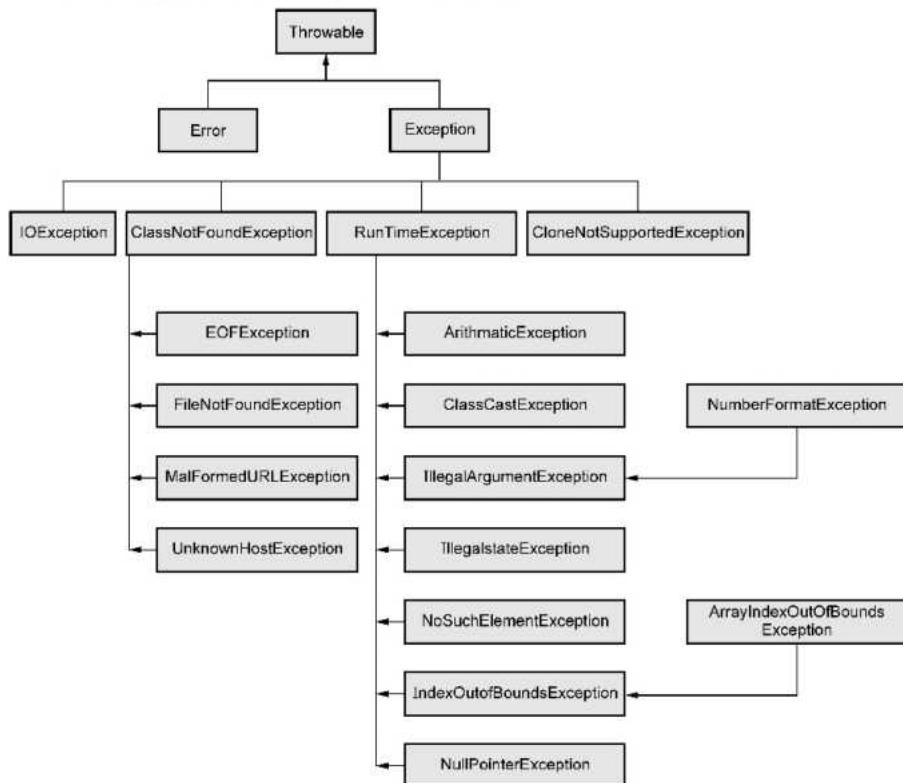
1. What is exception handling. Give the benefits of it.

**4.29 Exception Types**

SPPU : Dec.-17, May-19, Marks 6

- There are two type of exceptions in Java
- 1) **Checked Exception :** These types of exceptions need to be handled explicitly by the code itself either by using the **try-catch** block or by using **throws**. These exceptions are extended from the **java.lang.Exception** class.  
**For example :** IOException which should be handled using the try-catch block.
  - 2) **UnChecked Exception :** These type of exceptions need not be handled explicitly. The Java Virtual Machine handles these type of exceptions. These exceptions are extended from **java.lang.RuntimeException** class.  
**For example :** ArrayIndexOutOfBoundsException, NullPointerException, Run Time Exception.

The hierarchy of built in exceptions is as shown below –



**Fig. 4.29.1 Exception hierarchy**

### Review Questions

- Define the term exception. State the advantage of exception handling. What are types of exceptions?
- What are the types of Exceptions? Explain any three built-in Exceptions with suitable example

SPPU : Dec.-17, Marks 6

SPPU : May-19, Marks 6

### 4.30 try, catch, throw, throws and finally

SPPU : Dec.-17, Nov.-18, Marks 6

Various keywords used in handling the exception are -

**try** - A block of source code that is to be monitored for the exception.

**catch** - The catch block handles the specific type of exception along with the try block.  
Note that for each corresponding try block there exists the catch block.

**finally** - It specifies the code that must be executed even though exception may or may not occur.

**throw** - This keyword is used to throw specific exception from the program code.

**throws** - It specifies the exceptions that can be thrown by a particular method.

#### 4.30.1 Handling try-catch Block

- The statements that are likely to cause an exception are enclosed within a **try** block. For these statements the exception is thrown.
- There is another block defined by the keyword **catch** which is responsible for handling the exception thrown by the try block.
- As soon as exception occurs it is handled by the **catch** block. The catch block is added immediately after the try block. Refer Fig. 4.30.1.

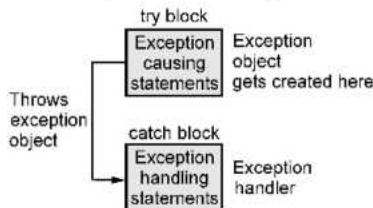


Fig. 4.30.1

Following is an example of **try-catch** block

```

try
{
//exception gets generated here

}

catch(Type_of_Exception e)
{

//exception is handled here
}
  
```

- If any one statement in the try block generates exception then remaining statements are skipped and the control is then transferred to the catch statement.

**Java Program[RunErrDemo.java]**

```

class RunErrDemo
{
    public static void main(String[] args)
    {
        int a,b,c;
        a=10;
        b=0;
        try
        {
            c=a/b;
        }
        catch(ArithmaticException e)
        {
            System.out.println("\n Divide by zero");
        }
        System.out.println("\n The value of a: "+a);
        System.out.println("\n The value of b: "+b);
    }
}

```

Exception occurs because the element is divided by 0.

Exception is handled using **catch** block

**Output**

Divide by zero

The value of a: 10

The value of b: 0

Note that even if the exception occurs at some point, the program does not stop at that point.

**Example 4.30.1** Write a program in Java to calculate the value of  $((x + y)/(x - y))$ . Program should prevent the condition  $x - y = 0$ .

SPPU : Dec.-17, Marks 6

**Solution :**

```

import java.util.Scanner;
public class ExpressionEval
{
    public static void main(String args[])
    {
        int x,y,N,D;
        float result;
        Scanner input = new Scanner(System.in);
        System.out.println("Enter value of x: ");
        x = input.nextInt();
        System.out.println("Enter value of y: ");

```

```

y = input.nextInt();
try
{
    N=x+y;
    D=x-y;
    result=N/D;
    System.out.println("The Result = "+result);
}
catch(ArithmeticException e)
{
    System.out.println(e);
}
}
}
}

```

#### 4.30.2 Using finally

- Sometimes because of execution of **try** block the execution gets break off. And due to this some important code (which comes after throwing off an exception) may not get executed. That means, sometimes try block may bring some unwanted things to happen.
- The **finally** block provides the assurance of execution of some important code that must be executed after the try block.
- Even though there is any exception in the try block the statements assured by **finally** block are sure to execute. These statements are sometimes called as **clean up code**. The syntax of **finally** block is

```

finally
{
    //clean up code that has to be executed finally
}

```

- The finally block always executes. The finally block is to free the resources.

#### Java Program [finallyDemo.java]

```

/*
This is a java program which shows the use of finally block for handling exception
*/
class finallyDemo
{
    public static void main(String args[])
    {
        int a=10,b=-1;
        try

```

```

    {
        b=a/0;
    }
    catch(ArithmeticException e)
    {
        System.out.println("In catch block: "+e);
    }
    finally
    {
        if(b!= -1)
            System.out.println("Finally block executes without occurrence of exception");
        else
            System.out.println("Finally block executes on occurrence of exception");
    }
}
}

```

**Output**

In catch block: java.lang.ArithmetiException: / by zero  
 Finally block executes on occurrence of exception

**Program Explanation: In above program,**

- 1) on occurrence of exception in try block the control goes to catch block, the exception of instance *ArithmetiException* gets caught.
- 2) This is divide by zero exception and therefore */ by zero* will be printed as output.

**4.30.3 Using throws**

- When a method wants to throw an exception then keyword **throws** is used.
- The syntax is -

```

method_name(parameter_list) throws exception_list
{
}
}

```

Let us understand this exception handling mechanism with the help of simple Java program.

**Java Program**

```

/* This programs shows the exception handling mechanism using throws
 */
class ExceptionThrows
{
    static void fun(int a,int b) throws ArithmeticException
}

```

```

{
    int c;
    try
    {
        c=a/b;
    }
    catch(ArithmeticException e)
    {
        System.out.println("Caught exception: "+e);
    }
}
public static void main(String args[])
{
    int a=5;
    fun(a,0);
}
}

```

**Output**

Caught exception: java.lang.ArithmaticException: / by zero

**Program Explanation:** In above program,

- 1) the method *fun* is for handling the exception *divide by zero*. This is an arithmetic exception hence we write

```
static void fun(int a,int b) throws ArithmaticException
```

This method should be of *static* type.

- 2) Also note as this method is responsible for handling the exception the **try-catch** block should be within *fun*.

**Review Question**

1. What is an exception in Java ? What do you mean by handling an exception ? Give example to show the use of try( ), Catch( ) methods.

**SPPU : Nov.-18, Marks 6**

**4.31 Uncaught Exception**

**SPPU : May-19, Marks 6**

- If there exists some code in the source program which may cause an exception and if the programmer does not handle this exception then java runtime system raises the exception.
- Following example illustrates how Java runtime system deals with an **uncaught exception**.

- When we use an index which is beyond the range of index then **ArrayIndexOutOfBoundsException** occurs.

Following Java program illustrates it

#### Java Program [ExceptionProg.java]

```
class ExceptionProg
{
    static void fun(int a[])
    {
        int c;
        c=a[0]/a[2];
    }
    public static void main(String args[])
    {
        int a[]={10,5};
        fun(a);
    }
}
```

#### Output

```
F:\>javac ExceptionProg.java
F:\>java ExceptionProg
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 2
        at ExceptionProg.fun(ExceptionProg.java:6)
        at ExceptionProg.main(ExceptionProg.java:11)
F:\>
```

#### Program Explanation :

When Java runtime system detects an attempt to use an array index which is out of bound then it constructs the new exception object and throws the corresponding exception. The default handler displays the description of exception, prints the stack trace from the point at which the exception occurred and then terminates the program.

Note that in the above output, the name of the method as **ExceptionProg.fun** and the line number 6 is displayed to represent the position problematic statements. It also displays the name of the exception as **ArrayIndexOutOfBoundsException** to represent the type of exception that is caused by the above code.

#### Review Question

- What are uncaught exceptions? State the use of try(), catch(), throw() methods

SPPU : May-19, Marks 6

### 4.32 Multiple Catch Clauses

- It is not possible for the try block to throw a single exception always.
- There may be the situations in which different exceptions may get raised by a single try block statements and depending upon the type of exception thrown it must be caught.
- To handle such situation multiple catch blocks may exist for the single try block statements.
- The syntax for single try and multiple catch is -

```
try
{
    ...
    ...//exception occurs
}
catch(Exception_type e)
{
    ...
    ...//exception is handled here
}
catch(Exception_type e)
{
    ...
    ...//exception is handled here
}
catch(Exception_type e)
{
    ...
    ...//exception is handled here
}
```

### Example

#### Java Program[MultipleCatchDemo.java]

```
class MultipleCatchDemo
{
    public static void main (String args [ ])

```

```

{
    int a[] = new int [3];
    try
    {
        for (int i = 1; i <=3; i++)
        {
            a[i] = i *i;
        }

        for (int i = 0; i <3; i++)
        {
            a[i] = i/i;
        }
    }
    catch (ArrayIndexOutOfBoundsException e)
    {
        System.out.println ("Array index is out of bounds");
    }
    catch (ArithmaticException e)
    {
        System.out.println ("Divide by zero error");
    }
}
}

```

**Output**

Array index is out of bounds

**Note** If we comment the first for loop in the try block and then execute the above code we will get following output -

Divide by zero error

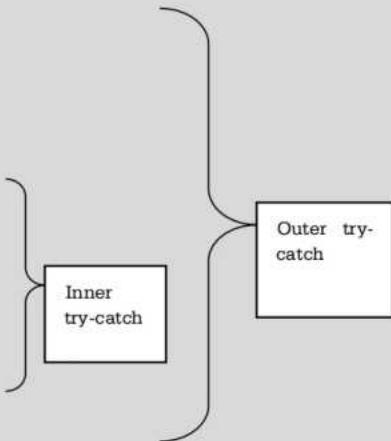
### 4.33 Nested try Statements

- When there are chances of occurring multiple exceptions of different types by the same set of statements then such situation can be handled using the nested try statements.
- Following is an example of nested try-catch statements -

#### Java Program[NestedtryDemo.java]

```
class NestedtryDemo
{
    public static void main (String[] args)
    {
        try
        {
            int a = Integer.parseInt (args [0]);
            int b = Integer.parseInt (args [1]);
            int ans = 0;

            try
            {
                ans = a / b;
                System.out.println("The result is "+ans);
            }
            catch (ArithmaticException e)
            {
                System.out.println("Divide by zero");
            }
        }
        catch (NumberFormatException e)
        {
            System.out.println ("Incorrect type of data");
        }
    }
}
```



#### Output

```
D:\>javac NestedtryDemo.java
```

```
D:\>java NestedtryDemo 20 10
```

The result is 2

```
D:\>java NestedtryDemo 20 a
```

Incorrect type of data

Outer catch handles the error

```
D:\>java NestedtryDemo 20 0
Divide by zero
```

Inner catch handles the error

### 4.34 Built-in Exceptions

SPPU : Dec.-17, Marks 6

Various common exception types and causes are enlisted in the following table -

Exception	Description
ArithmaticException	This is caused by error in Math operations. For e.g. Divide by zero.
NullPointerException	Caused when an attempt to access an object with a Null reference is made.
IOException	When an illegal input/output operation is performed then this exception is raised.
IndexOutOfBoundsException	An index when gets out of bound ,this exception will be caused.
ArrayIndexOutOfBoundsException	Array index when gets out of bound, this exception will be caused.
ArrayStoreException	When a wrong object is stored in an array this exception must occur.
EmptyStackException	An attempt to pop the element from empty stack is made then this exception occurs
NumberFormatException	When we try to convert an invalid string to number this exception gets caused.
RuntimeException	To show general run time error this exception must be raised.
Exception	This is the most general type of exception

ClassCastException	This type of exception indicates that one tries to cast an object to a type to which the object can not be casted.
IllegalStateException	This exception shows that a method has been invoked at an illegal or inappropriate time. That means when Java environment or Java application is not in an appropriate state then the method is invoked.

### i) Demonstrative **ArrayIndexOutOfBoundsException**

Now to handle this exception we can write the following modified code -

```
class ExceptionProg
{
    static void fun(int a[])
    {
        int c;
        try
        {
            c=a[0]/a[2];//Exception occurs due to this line
        }
        catch(ArrayIndexOutOfBoundsException e)
        {
            System.out.println("Caught exception: "+e);
        }
    }
    public static void main(String args[])
    {
        int a[]={10,5};
        fun(a);
    }
}
```

Then we will get following output

### Output

```
F:\>javac ExceptionProg.java
F:\>java ExceptionProg
Caught exception: java.lang.ArrayIndexOutOfBoundsException: 2
F:\>
```

**ii) Demonstrating ArithmeticException**

This type of exception occurs due to error in Math operation. Following is a simple Java program that shows illustration of this type of exception.

```
public class ArithmeticExceptionDemo
{
    public static void main(String args[])
    {
        try
        {
            int a=10;
            int b=0;
            int c;
            c=a/b;//Exception gets raised due to this line
        }
        catch(ArithmaticException e)
        {
            System.out.println("Caught Exception "+e);
        }
    }
}
```

**Output**

```
D:\test>javac ArithmeticExceptionDemo.java
```

```
D:\test>java ArithmeticExceptionDemo
Caught Exception java.lang.ArithmaticException: / by zero
```

**iii) Demonstrating NumberFormatException**

When we try to convert invalid string to number then **NumberFormatException** occurs. Following is a simple Java program that shows illustration of this type of exception.

```
public class NumberFormatDemo
{
    public static void main(String args[])
    {
        try
        {
            String str="Hello";
            int num=Integer.parseInt(str);//Exception gets raised due to this line
            System.out.println(num);
        }
        catch(NumberFormatException e)
        {
            System.out.println("Caught Exception "+e);
        }
    }
}
```

```

    }
}
}

```

**Output**

```

D:\test>javac NumberFormatDemo.java
D:\test>java NumberFormatDemo
Caught Exception java.lang.NumberFormatException: For input string: "Hello"

```

**(iv) Demonstrating NullPointerException :**

The NullPointerException occurs when program attempts to use an object reference that has NULL value. Following Java program illustrates this type of exception.

**Java Program**

```

import java.io.*;
class NullPtrExceptionDemo
{
    public static void main(String [] args)
    {
        String str=null;
        try
        {
            if(str.equals("India"))
                System.out.print(str);
        }
        catch(NullPointerException e)
        {
            System.out.println("Error!!! NullPointerException caught");
        }
    }
}

```

**Output**

```

Error!!! NullPointerException caught

```

**(v) IndexOutOfBoundsException :**

This exception is thrown when a program attempts to access a value in an indexable collection in such a way that the value of index is outside the valid range of indices. Following example program shows that the string is of length 5 and we try to access a character at index 10. Due to this the IndexOutOfBoundsException is thrown.

**Java Program**

```

import java.io.*;
class IndexExceptionDemo
{

```

```

public static void main(String [] args)
{
    String str="India";
    try
    {
        char ch=str.charAt(10);
        System.out.print(ch);
    }
    catch(IndexOutOfBoundsException e)
    {
        System.out.println("Error!!! IndexOutOfBoundsException caught");
    }
}
}

```

**Output**

Error!!! IndexOutOfBoundsException caught

**Review Question**

- State with example the use of the following built in exceptions in Java :

  - IndexOutOfBoundsException()*
  - NullPointerException()*
  - ArrayIndexOutOfBoundsException()*

SPPU : Dec.-17, Marks 6

**4.35 Custom Exceptions**

- We can throw our own exceptions using the keyword **throw**.
- The syntax for throwing out own exception is -

**throw new Throwable's subclass**

Here the **Throwable's subclass** is actually a subclass derived from the **Exception** class.

For example -

**throw new ArithmeticException();**

Throwable's subclass

Let us see a simple Java program which illustrates this concept.

**Java Program[MyExceptDemo.java]**

```

import java.lang.Exception;
class MyOwnException extends Exception
{
}

```

```

MyOwnException(String msg)
{
    super(msg);
}
}

class MyExceptDemo
{
    public static void main (String args [])
    {
        int age;
        age=15;
        try
        {
            if(age<21)
                throw new MyOwnException("Your age is very less than the condition");

        }
        catch (MyOwnException e)
        {
            System.out.println ("This is My Exception block");
            System.out.println (e.getMessage());
        }
        finally
        {
            System.out.println ("Finally block:End of the program");
        }
    }
}

```

**Output**

This is My Exception block  
 Your age is very less than the condition  
 Finally block:End of the program

**Program Explanation: In above code,**

- 1) the age value is 15 and in the try block - the if condition throws the exception if the value is less than 21.
- 2) As soon as the exception is thrown the catch block gets executed. Hence as an output we get the first message "This is My Exception block".
- 3) Then the control is transferred to the **class MyOwnException**(defined at the top of the program). The message is set and it is "Your age is very less than the condition".
- 4) This message can then printed by the catch block using the **System.out.println** statement by means of **e.message**.
- 5) At the end the finally block gets executed.

**Part IV: Managing I/O****4.36 Streams**

- **Stream** is basically a channel on which the data flow from sender to receiver.
- An input object that reads the stream of data from a file is called **input stream** and the output object that writes the stream of data to a file is called **output stream**.

**4.37 Byte Streams and Character Streams**

SPPU : Dec.-17, Marks 3

**4.37.1 Byte Stream**

- The byte stream is used for inputting or outputting the bytes.

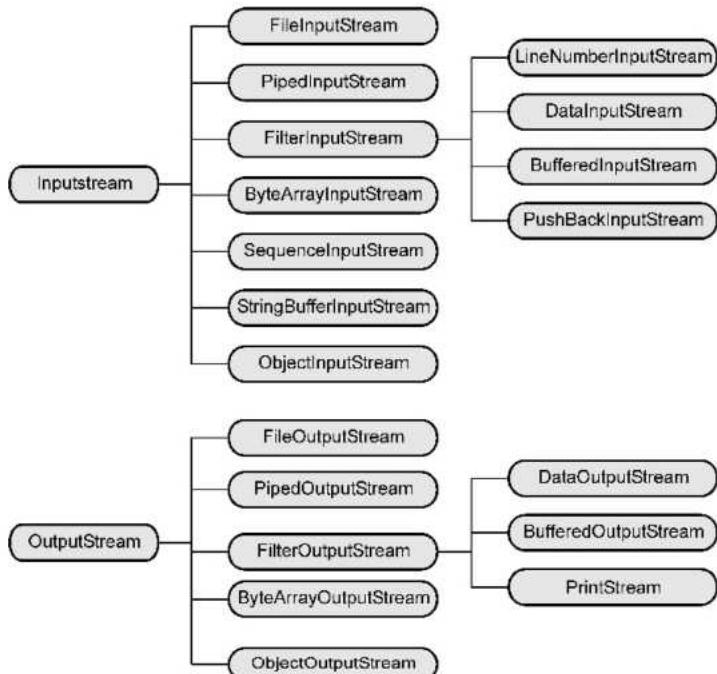


Fig. 4.37.1 Stream classes

- There are two super classes in byte stream and those are **InputStream** and **OutputStream** from which most of the other classes are derived.
- These classes define several important methods such as **read()** and **write()**. The input and output stream classes are as shown in Fig. 4..37.1

### 4.37.2 Character Stream

- The character stream is used for inputting or outputting the characters.
- There are two super classes in character stream and those are **Reader** and **Writer**.
- These classes handle the Unicode character streams. The two important methods defined by the character stream classes are **read()** and **Write()** for reading and writing the characters of data.

### 4.37.3 Difference between Byte and Character Stream

Sr.No.	Byte Stream	Character Stream
1.	The byte stream is used for inputting and outputting the bytes.	The character stream is used for inputting and outputting the characters.
2.	There are two super classes used in byte stream and those are – InputStream and OutputStream	There are two super classes used in byte stream and those are – Reader and Writer
3.	A byte is a 8-bit number type that can represent values from – 127 to 127. Ascii values can be represented with exactly 7 bits.	A character is a 16 bit number type that represents Unicode.
4.	It never supports Unicode characters	It supports Unicode characters.

#### Review Question

1. What is difference between byte streams and character streams ?

SPPU : Dec.-17, Marks 3

### 4.38 Predefined Streams

SPPU : Nov.-19, Marks 6

- For displaying the messages on console we make use of **System.out** statement. There is a class named **System** which is defined in **java.lang** package.
- All Java programs automatically import the **java.lang** package.
- System contains three predefined stream variables - **in**, **out** and **err**.

- These are declared public and static within the system.
- The **System.out** defines the standard OutputStream by default. This is **console**.
- The **System.in** refers to standard InputStream which is the **keyboard** by default.
- The **System.err** are the objects for standard input stream and error.

### Review Question

1. What is meant by stream in Java? How does Java manage Input/Output using streams?  
Differentiate ByteStream, characterStream and Predefined Stream SPPU : Nov.-19, Marks 6

### 4.39 Reading Console Input

- In Java input is taken using **System.in**. For that we wrap **System.in** in a **BufferedReader** object to create character stream
- Along with it we should also mention the abstract class of **BufferedReader** class which is **InputStreamReader**. Hence the syntax is,

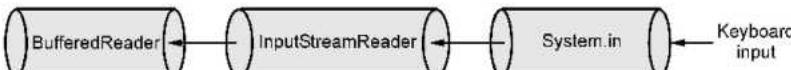
```
BufferedReader object_name = new  
BufferedReader(new InputStreamReader(System.in));
```

- For example the object named **obj** can be created as

```
BufferedReader obj = new BufferedReader(new  
InputStreamReader(System.in));
```

- Then, using this object we invoke **read()** method. For e.g. **obj.read()**. But this is not the only thing needed for reading the input from console; we have to mention the exception **IOException** for this purpose. Hence if we are going to use **read()** method in our program then function main can be written like this -

```
public static void main(String args[]) throws IOException
```



- We can read characters, strings or numeric values from the console.

Let us understand "how to read console input ?" with the help of same Java programs

## 1. Reading Character

### Java program [ReadChar.java]

```
/*
This is a java program which is for reading the input from console
*/
import java.io.*;
class ReadChar
{
public static void main(String args[])
throws IOException
{
//declaring obj for read() method
BufferedReader obj=new BufferedReader(new InputStreamReader(System.in));
int count=0;
char ch;
System.out.println("\n Enter five characters");
while(count<5)
{
ch=(char)obj.read();//reading single character
System.out.println(ch);//outputting it
count++;//count to keep track of 5 characters
}
}
}
```

### Output

```
Enter five characters
hello
h
e
l
l
o
```

This program allows you take the input from console. As given in above program, **read()** method is used for reading the input from the console. The method **read()** returns the integer value, hence it is typecast to char because we are reading characters from the console. And last but not least we should write,

```
import java.io.*;
```

in our java program, because these I/O operations are supported by the java package **java.io**.

## 2. Reading Strings

### Java Program [ReadString.java]

```
/*
This is a java program which is for reading the input from console
*/
import java.io.*;
class ReadString
{
    public static void main(String args[])
    throws IOException
    {
        BufferedReader obj=new BufferedReader(new InputStreamReader(System.in));
        String s;
        s=obj.readLine(); //for reading the string
        while(!s.equals("end"))
        {
            System.out.println(s);
            s=obj.readLine();
        }
    }
}
```

### Output

```
hello how are you
hello how are you
end
```

Note that to read the string input we have used `readLine()` function. Otherwise this program is almost same as previous one.

## 4.40 Writing Console Output

SPPU : Dec.-17, Nov.-18, Marks 7

The simple method used for writing the output on the console is `write()`. Here is the syntax of using `write()` method –

```
System.out.write(int b)
```

The bytes specified by `b` has to be written on the console. But typically `print()` or `println()` is used to write the output on the console. And these methods belong to `PrintWriter` class.

**Java Program**

```
class WriteData
{
    public static void main(String args[])
    {
        int val;
        val=10;
        char c='A';
        System.out.write(val);
        System.out.write('\n');
        System.out.write(c);
    }
}
```

The **write()** method is not always preferred to write the console output rather **print** and **println** methods are easier to use.

**Review Question**

1. Demonstrate the use of console class to get inputs and show results

**SPPU : Dec.-17, Marks 3**

2. What is Character Streams and Byte Streams ? State any two examples of each predefined Character and Byte Stream classes for I/O in Java.

**SPPU : Nov.-18, Marks 7**

**4.41 Print Writer Class**

**SPPU : May-19, Dec.-19, Marks 7**

- The **System.out** is used to write the contents of the console. Similarly the **PrintWriter** class is also useful in writing the contents to the console.
- It is mostly recommended for debugging purpose.
- The **PrintWriter** class is one of the character based classes.
- It has several constructors. One of the constructor is

**PrintWriter(OutputStream outputStream, boolean flushOnNewline)**

- Here, **outputStream** is an object of type **OutputStream**, and **flushOnNewline** controls whether Java flushes the output stream every time a **println()** method is called. If **flushOnNewline** is true, flushing automatically takes place. If false, flushing is not automatic.
- This class supports **print** and **println** methods.
- The writer object for **PrintWriter** class is created as follows -

**Printer pw = new PrintWriter(System.out);**

Following Java program shows how to write the contents on the console window

### Java Program

```
import java.io.*;
public class PrintWriterDemo
{
    public static void main(String[] args)
    {
        char c='A';
        int val=100;
        double d = 1234.567;
        // create a new writer
        PrintWriter pw = new PrintWriter(System.out);
        pw.print(c);
        // change the line
        pw.println();
        pw.print(val);
        pw.println();
        pw.print(d);
        // flush the writer
        pw.flush();
    }
}
```

### Output

```
A
100
1234.567
```

### Review Question

1. Define the following:
 

i) Streams	ii) Bytes Streams
iii) Character Streams	iv) Predefined Streams

Enlist the methods of PrintWriter Class with example

**SPPU : May-19, Marks 7**

2. Illustrate use of methods in PrintStream Class to implement Student Information system.

**SPPU : Dec.-19, Marks 5**

### 4.42 FileInputStream/FileOutputStream

**SPPU : May-18, Marks 6**

The FileInputStream class creates an InputStream using which we can read bytes from a file. The two common constructors of FileInputStream are -

```
FileInputStream(String filename);
FileInputStream(File fileobject);
```

In the following Java program, various methods of **FileInputStream** are illustrated -

#### Java Program[FileStreamProg.java]

```
import java.io.*;
class FileStreamProg
{
    public static void main(String[] args) throws Exception
    {
        int n;
        InputStream fobj = new FileInputStream("f:/I_O_programs/FileStreamProg.java");
        System.out.println("Total available bytes: " + (n = fobj.available()));
        int m = n - 400;
        System.out.println("\n Reading first " + m + " bytes at a time");
        for (int i = 0; i < m; i++)
            System.out.print((char) fobj.read());
        System.out.println("\n Skipping some text");
        fobj.skip(n / 2);
        System.out.println("\n Still Available: " + fobj.available());
        fobj.close();
    }
}
```

#### Output

```
F:\I_O_programs>javac FileStreamProg.java
F:\I_O_programs>java FileStreamProg
Total available bytes: 569
Reading first 169 bytes at a time
import java.io.*;
class FileStreamProg
{
    public static void main(String[] args) throws Exception
    {
        int n;
        InputStream fobj = new FileInputStream("f:/I_O_program
Skipping some text
Still Available: 116
F:\I_O_programs>
```

Reading this much text from the file

The **FileOutputStream** can be used to write the data to the file using the **OutputStream**. The constructors are-

```
FileOutputStream(String filename)
FileOutputStream(Object fileobject)
FileInputStream(String filename,boolean app);
FileInputStream(String fileobject,boolean app);
```

The `app` denotes that the append mode can be true or false. If the append mode is true then you can append the data in the file otherwise the data can not be appended in the file.

In the following Java program, we are writing some data to the file.

#### Java Program[FileOutputStreamProg.java]

```
import java.io.*;
class FileOutputStreamProg
{
    public static void main(String[] args) throws Exception
    {
        String my_text = "India is my Country\n" + "and I love my country very much.";
        byte b[] = my_text.getBytes();
        OutputStream fobj = new FileOutputStream("F:\\" + "I_O_programs\\output.txt");
        for (int i = 0; i < b.length; i++)
            fobj.write(b[i]);
        System.out.println("\n The data is written to the file");
        fobj.close();
    }
}
```

#### Output

```
Command Prompt
F:\\" + "I_O_programs">javac FileOutputStreamProg.java
F:\\" + "I_O_programs">java FileOutputStreamProg
The data is written to the file
F:\\" + "I_O_programs">type output.txt
India is my Country
and I love my country very much.
F:\\" + "I_O_programs">>
```

**Example 4.42.1** Write a program that copies the content of one file to another by removing unnecessary spaces between words.

**Solution :**

```
import java.io.*;
public class CopyFile
{
    private static void CopyDemo(String src, String dst)
    {
        try
```

```

{
    File f1 = new File(src);
    File f2 = new File(dst);
    InputStream in = new FileInputStream(f1);
    OutputStream out = new FileOutputStream(f2);
    byte[] buff = new byte[1024];
    int len;
    len=in.read(buff);
    while (len > 0)
    {
        String text=new String(buff);//converting bytes to text
        text = text.replaceAll("\\s+", " ");//removing unnecessary spaces
        buff=text.getBytes();//converting that text back to bytes
        out.write(buff,0,len);//writing bytes to destination file
        len=in.read(buff);//reading the remaining content of the file
    }
    in.close();
    out.close();
    System.out.println("File copied.");
}
catch(FileNotFoundException ex)
{
    System.out.println(ex.getMessage() + " in the specified directory.");
    System.exit(0);
}
catch(IOException e)
{
    System.out.println(e.getMessage());
}
}

public static void main(String[] args)
{
    CopyDemo(args[0],args[1]);
}
}
}

```

**Output**

D:\test>javac CopyFile.java

D:\test>javac CopyFile.java

D:\test>java CopyFile in.txt out.txt

File copied.

D:\test>type in.txt

This is my India.

I love my country.

```
D:\test>type out.txt
```

This is my India. I love my country.

**Example 4.42.2** What is difference in character and byte streams in Java ? Give any two input and any two output classes for character streams.

SPPU : May-18, Marks 6

**Solution :** Difference – Refer section 4.37.3.

Programs using character input classes

#### A. Output Classes

(1) Using BufferedWriter Class

```
import java.io.*;
public class BufferedWriterProgram {
    public static void main(String[] args) throws Exception {
        FileWriter writer = new FileWriter("output.txt");
        BufferedWriter buffer = new BufferedWriter(writer);
        String str="Hello friends, how are you?";
        buffer.write(str);
        buffer.close();
        System.out.println("Written Data to the File...");
    }
}
```

(2) Using StringWriter Class

```
import java.io.*;
public class StringWriterProgram {
    public static void main(String[] args) {
        try {
            FileWriter fw = new FileWriter("output.txt");
            StringWriter sw = new StringWriter();
            sw.write("Hello friends, how are you?");
            fw.write(sw.toString());
            sw.close();
            fw.close();
            System.out.println("Written Data to the file...");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

#### B. Input Classes

(1) Using BufferedReader Class

```
import java.io.*;
public class BufferedReaderProgram {
```

```

public static void main(String[] args) throws Exception {
    FileReader reader = new FileReader("output.txt");
    BufferedReader buffer = new BufferedReader(reader);
    int i;
    System.out.println("\tThe contents of the file are...");
    while((i=buffer.read())!=-1) {
        System.out.print((char)i);
    }
    buffer.close();
    reader.close();
}
}

```

## (2) Using StringReader Class

```

import java.io.*;
public class StringReaderProgram {
    public static void main(String[] args) {
        String str = "Hello friends, how are you?";
        StringReader reader = new StringReader(str);
        // create a char array to read chars into
        char buff[] = new char[50];
        try {
            // read characters into an array.
            reader.read(buff, 0,str.length());
            System.out.println(buff);
            reader.close();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}

```

## 4.43 Console I/O using Scanner Class

SPPU : May-18, Marks 7

The **Console** is a window of the operating system through which users can interact with the application program. The interaction consists of **text input** from the **standard input** (usually keyboard) or text display on the **standard output** (usually on the computer screen). These actions are also known as **input-output operations**. **Input** is any information that is needed by your program to complete its execution. **Output** is any information that the program must convey to the user.

### 4.43.1 Console Input

The **Scanner** class has a method **NextLine** which returns a line of text typed by the user on the console. For reading some line of text typed by the user through keyboard, following code can be used -

#### For Reading the Text

**Step 1 :** The scanner class is present in the `java.util` package. Hence include following line in your Java program

```
import java.util.Scanner;
```

**Step 2 :** Create a Scanner using following code

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

**Step 3 :** Prompt the user to type something

```
System.out.println("Enter Something...");
```

**Step 4 :** Read the line of text entered by the user.

```
String input=scanner.nextLine();
```

**Step 5 :** Now you can display whatever is typed by the user

```
System.out.println("You have entered..." +input);
```

#### For Reading the integer

Normally, when user types 123 on the console using the keyboard, it is interpreted as "123" that is string 123 and not the integer value 123. Therefore we need to convert the string to int value. Following are the steps to do so.

**Step 1 :** Create a Scanner using following code

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

**Step 2 :** Prompt the user to type something

```
System.out.println("Enter Some integer value...");
```

**Step 3 :** Read the line of text entered by the user.

```
String input=scanner.nextLine();
```

**Step 4 :** Convert the string to int value. Use the `Integer` class to parse the string of characters into an integer.

```
int number = Integer.parseInt( input );
```

### 4.43.2 Console Output

For the Console output the `System.out.print(...)` and `System.out.println(...)` are used.

#### For example -

```
System.out.print(" This is a line");
```

**Example 4.43.1** What is the use of CharacterArrayReader( ) and CharacterArrayWriter( )

methods in Java ? Write a program which reads string of 10 characters from the user. Program extracts and prints the substring from the given string using above methods.

SPPU : May-18, Marks 7

**Solution :**

**CharArrayReader :** The Java CharArrayReader class allows to read the contents of a char array as a character stream.

**CharArrayWriter :** It makes it possible to write characters via the Writer methods and convert the written characters into a char array.

**Java Program :**

```
import java.io.*;
import java.util.*;
class Test
{
    public static void main(String arg[]) throws IOException
    {
        System.out.println("Enter a string of 10 characters: ");
        Scanner input = new Scanner(System.in);
        String str = input.nextLine();
        if(str.length() > 10)
            str = str.substring(0, 9);
        char c[] = str.toCharArray();
        CharArrayReader car = new CharArrayReader(c);
        int i;
        System.out.println("Displaying the string using CharArrayReader");
        while((i = car.read()) != -1)
            System.out.print((char)i);
        System.out.println("\nDisplaying the substring");
        CharArrayWriter caw = new CharArrayWriter();
        caw.write(str, 2, 5);
        System.out.println(caw.toString());
    }
}
```

**4.44 Multiple Choice Questions**

Q.1 The term \_\_\_\_\_ refers to a way of organizing classes that share properties.

- |   |   |
|---|---|
| <input type="checkbox"/> a) object-oriented | <input type="checkbox"/> b) encapsulation |
| <input type="checkbox"/> c) polymorphism    | <input type="checkbox"/> d) inheritance   |

**Q.2 Which of the keyword is used in subclass to call the constructor of superclass ?**

- |                                       |                                    |
|---------------------------------------|------------------------------------|
| <input type="checkbox"/> a superclass | <input type="checkbox"/> b main    |
| <input type="checkbox"/> c super      | <input type="checkbox"/> d extends |

**Q.3 Which of the following variables can not be inherited ?**

- |                                    |                                      |
|------------------------------------|--------------------------------------|
| <input type="checkbox"/> a public  | <input type="checkbox"/> b static    |
| <input type="checkbox"/> c private | <input type="checkbox"/> d protected |

**Q.4 Which of these keywords is used to inherit a class ?**

- |                                   |                                    |
|-----------------------------------|------------------------------------|
| <input type="checkbox"/> a super  | <input type="checkbox"/> b this    |
| <input type="checkbox"/> c extent | <input type="checkbox"/> d extends |

**Q.5 Which of these keywords is used to refer to member of base class from subclass ?**

- |                                  |                                  |
|----------------------------------|----------------------------------|
| <input type="checkbox"/> a Upper | <input type="checkbox"/> b Super |
| <input type="checkbox"/> c this  | <input type="checkbox"/> d that  |

**Q.6 The reference of an object is created using \_\_\_\_\_ keyword.**

- |                                 |                                   |
|---------------------------------|-----------------------------------|
| <input type="checkbox"/> a this | <input type="checkbox"/> b static |
| <input type="checkbox"/> c new  | <input type="checkbox"/> d class  |

**Q.7 Which of these is a correct way of calling a constructor having no parameters of super class A by subclass B ?**

- |   |   |
|---|---|
| <input type="checkbox"/> a Super(void); | <input type="checkbox"/> b Superclass.(); |
| <input type="checkbox"/> c Super.A();   | <input type="checkbox"/> d Supper()       |

**Q.8 An abstract class \_\_\_\_\_.**

- a is a class which cannot be instantiated
- b is a class which has no methods
- c is a class which has only abstract methods
- d is a class which has only overridden methods

**Q.9 Following keyword is used to define the abstract class**

- |  |                                     |
|--|-------------------------------------|
| <input type="checkbox"/> a abstractclass | <input type="checkbox"/> b abstract |
| <input type="checkbox"/> c ABS           | <input type="checkbox"/> d empty    |

**Q.10 \_\_\_\_\_ is a package that contains abstract keyword**

- |                                      |  |
|--------------------------------------|--|
| <input type="checkbox"/> a java.lang | <input type="checkbox"/> b java.util   |
| <input type="checkbox"/> c java.io   | <input type="checkbox"/> d java.system |

**Q.11 What is polymorphism in Java ?**

- a It is when a single variable is used with several different types of related objects at different places in a program.
- b It is when a program uses several different types of objects, each with its own variable.
- c It is when a single parent class has many child classes.
- d It is when a class has several methods with the same name but different parameter types.

**Q.12 What is an advantage of polymorphism ?**

- a Variables can be re-used in order to save memory.
- b The same program logic can be used with objects of several related types.
- c Constructing new objects from old objects of a similar type saves time.
- d None of these

**Q.13 Polymorphism can be applied to :**

- |   |   |
|---|---|
| <input type="checkbox"/> a math operators | <input type="checkbox"/> b method names |
| <input type="checkbox"/> c object names   | <input type="checkbox"/> d both a and b |

**Q.14 Which statement best describes overriding a method ?**

- a Methods in a base class and derived class have the same name but different visibility modifiers.
- b Methods in a base class and derived class have the same name and have the same number and types of parameters.
- c Methods in a base class and derived class have the same name but have different return types.
- d Methods in a base class and derived class have the same name but different number or types of parameters.

**Q.15 \_\_\_\_\_ is the process of using same name for two or more functions.**

- |   |  |
|---|--|
| <input type="checkbox"/> a Method overriding      | <input type="checkbox"/> b Constructor overloading |
| <input type="checkbox"/> c Constructor overriding | <input type="checkbox"/> d Method overloading      |

**Q.16 With Polymorphism :**

- a One method can have multiple names
- b One object can have multiple names
- c Many methods can share the same name
- d Many objects can share the same name

**Q.17 When does exceptions in Java arises in code sequences ?**

- |                                      |   |
|--------------------------------------|---|
| <input type="checkbox"/> a Runtime   | <input type="checkbox"/> b Compile time |
| <input type="checkbox"/> c Free time | <input type="checkbox"/> d Hard time    |

**Q.18 An error occurred at runtime is called as \_\_\_\_**

- |   |   |
|---|---|
| <input type="checkbox"/> a exception          | <input type="checkbox"/> b runtime error      |
| <input type="checkbox"/> c compile time error | <input type="checkbox"/> d run time exception |

**Q.19 Exception generated in try block is caught in \_\_\_\_ block.**

- |                                   |                                    |
|-----------------------------------|------------------------------------|
| <input type="checkbox"/> a catch  | <input type="checkbox"/> b throw   |
| <input type="checkbox"/> c throws | <input type="checkbox"/> d finally |

**Q.20 Select the keyword which is not a part of exception handling.**

- |                                     |                                    |
|-------------------------------------|------------------------------------|
| <input type="checkbox"/> a try      | <input type="checkbox"/> b catch   |
| <input type="checkbox"/> c throwing | <input type="checkbox"/> d finally |

**Explanation :** To throw an exception the keyword throwing is used.

**Q.21 For monitoring the code that might have exception we use \_\_\_\_ keyword.**

- |                                   |                                    |
|-----------------------------------|------------------------------------|
| <input type="checkbox"/> a try    | <input type="checkbox"/> b catch   |
| <input type="checkbox"/> c throws | <input type="checkbox"/> d finally |

**Explanation :** The code which is to be examined for exception is kept inside a try block.

**Q.22 For throwing the exception manually following keyword is used :**

- |                                  |                                    |
|----------------------------------|------------------------------------|
| <input type="checkbox"/> a try   | <input type="checkbox"/> b catch   |
| <input type="checkbox"/> c throw | <input type="checkbox"/> d finally |

**Q.23 The class Exception is present in \_\_\_\_ package.**

- |   |                                      |
|---|--------------------------------------|
| <input type="checkbox"/> a java.io        | <input type="checkbox"/> b java.util |
| <input type="checkbox"/> c java.exception | <input type="checkbox"/> d java.lang |

**Q.24 The class Exception is a subclass of class \_\_\_\_\_.**

- |  |                                      |
|--|--------------------------------------|
| <input type="checkbox"/> a Object      | <input type="checkbox"/> b Throwable |
| <input type="checkbox"/> c IOException | <input type="checkbox"/> d Error     |

**Explanation :** Actually Throwable is a superclass of all exception classes.

**Q.25 Exception and Error are immediate subclasses of the class \_\_\_\_\_.**

- |                                      |  |
|--------------------------------------|--|
| <input type="checkbox"/> a Object    | <input type="checkbox"/> b IOException |
| <input type="checkbox"/> c Throwable | <input type="checkbox"/> d Error       |

**Q.26** The divide by zero exception can be thrown using \_\_\_\_\_.

- |   |  |
|---|--|
| <input type="checkbox"/> a) ArithmeticException   | <input type="checkbox"/> b) NullPointerException |
| <input type="checkbox"/> c) NumberFormatException | <input type="checkbox"/> d) IOException          |

**Explanation :** The division is an arithmetic operation and when divide by zero exception occurs it is through an ArithmeticException.

**Q.27** \_\_\_\_\_ block gets executed compulsory whether exception is caught or not

- |                                   |                                     |
|-----------------------------------|-------------------------------------|
| <input type="checkbox"/> a) try   | <input type="checkbox"/> b) catch   |
| <input type="checkbox"/> c) throw | <input type="checkbox"/> d) finally |

**Q.28** In Java, a library of classes is called \_\_\_\_\_

- |                                       |  |
|---------------------------------------|--|
| <input type="checkbox"/> a) folder    | <input type="checkbox"/> b) package        |
| <input type="checkbox"/> c) directory | <input type="checkbox"/> d) an application |

**Q.29** Which of these access specifiers can be used for a class so that its members can be accessed by a different class in the different package?

- |                                     |   |
|-------------------------------------|---|
| <input type="checkbox"/> a) Public  | <input type="checkbox"/> b) Protected     |
| <input type="checkbox"/> c) Private | <input type="checkbox"/> d) None of these |

**Q.30** Which is the correct way to import the package name mypkg ?

- |  |  |
|--|--|
| <input type="checkbox"/> a) Import mypkg   | <input type="checkbox"/> b) import mypkg.        |
| <input type="checkbox"/> c) import mypkg.* | <input type="checkbox"/> d) import package mypkg |

**Q.31** The syntax to import package is \_\_\_\_\_

- |  |  |
|--|--|
| <input type="checkbox"/> a) import package packagename | <input type="checkbox"/> b) import * package |
| <input type="checkbox"/> c) import packagename         | <input type="checkbox"/> d) import package   |

**Q.32** Which of these is a mechanism for naming and visibility control of a class and its content?

- |  |                                      |
|--|--------------------------------------|
| <input type="checkbox"/> a) object     | <input type="checkbox"/> b) packages |
| <input type="checkbox"/> c) interfaces | <input type="checkbox"/> d) none     |

**Q.33** In a java program, package declaration \_\_\_\_\_ import statements.

- |  |   |
|--|---|
| <input type="checkbox"/> a) must precede           | <input type="checkbox"/> b) must succeed  |
| <input type="checkbox"/> c) may precede or succeed | <input type="checkbox"/> d) none of these |

**Q.34** How can a class be imported to a program?

- |  |  |
|--|--|
| <input type="checkbox"/> a) import classname       | <input type="checkbox"/> b) import * classname     |
| <input type="checkbox"/> c) import class classname | <input type="checkbox"/> d) class import classname |

**Q.35** The class string belongs to \_\_\_ package.

- |  |  |
|--|--|
| <input type="checkbox"/> a java.awt    | <input type="checkbox"/> b java.lang   |
| <input type="checkbox"/> c java.applet | <input type="checkbox"/> d java.string |

**Q.36** The first statement in Java source file is \_\_\_\_\_

- |  |   |
|--|---|
| <input type="checkbox"/> a package statement | <input type="checkbox"/> b import statement |
| <input type="checkbox"/> c main statement    | <input type="checkbox"/> d extend statement |

**Q.37** \_\_\_\_\_ package is used by compiler itself. So it does not need to be imported for use.

- |  |                                      |
|--|--------------------------------------|
| <input type="checkbox"/> a java.math   | <input type="checkbox"/> b java.awt  |
| <input type="checkbox"/> c java.applet | <input type="checkbox"/> d java.lang |

**Q.38** Which of these package contain classes and interfaces used for performing Input/output operations ?

- |                                      |   |
|--------------------------------------|---|
| <input type="checkbox"/> a java.util | <input type="checkbox"/> b java.lang        |
| <input type="checkbox"/> c java.io   | <input type="checkbox"/> d all of the above |

**Q.39** Which of these keywords is used by a class to use an interface defined previously?

- |                                       |                                       |
|---------------------------------------|---------------------------------------|
| <input type="checkbox"/> a Import     | <input type="checkbox"/> b Imports    |
| <input type="checkbox"/> c implements | <input type="checkbox"/> d Implements |

**Q.40** In Java, multiple inheritance can be achieved using \_\_\_\_\_

- |  |   |
|--|---|
| <input type="checkbox"/> a interface   | <input type="checkbox"/> b abstraction  |
| <input type="checkbox"/> c inheritance | <input type="checkbox"/> d polymorphism |

**Q.41** The number of interfaces extended by an interface is \_\_\_\_\_

- |                              |                                       |
|------------------------------|---------------------------------------|
| <input type="checkbox"/> a 1 | <input type="checkbox"/> b Any number |
| <input type="checkbox"/> c 2 | <input type="checkbox"/> d zero       |

**Q.42** The stream that is used for general Input and output not usually character data is called \_\_\_\_\_.

- |  |   |
|--|---|
| <input type="checkbox"/> a Byte stream | <input type="checkbox"/> b character stream |
| <input type="checkbox"/> c Reader      | <input type="checkbox"/> d Writer           |

**Q.43** \_\_\_ is the name of the abstract base class for streams dealing with general purpose (non-character) input?

- |  |   |
|--|---|
| <input type="checkbox"/> a InputStream | <input type="checkbox"/> b OutputStream |
| <input type="checkbox"/> c Reader      | <input type="checkbox"/> d Writer       |

**Q.44** \_\_\_\_ is the name of the abstract base class for streams dealing with character input?

- |   |  |
|---|--|
| <input type="checkbox"/> a) InputStream | <input type="checkbox"/> b) OutputStream |
| <input type="checkbox"/> c) Reader      | <input type="checkbox"/> d) Writer       |

**Q.45** The \_\_\_\_ package contains the File class in Java

- |                                       |                                       |
|---------------------------------------|---------------------------------------|
| <input type="checkbox"/> a) java.io   | <input type="checkbox"/> b) java.file |
| <input type="checkbox"/> c) java.util | <input type="checkbox"/> d) java.lang |

**Answer Keys for Multiple Choice Questions :**

Q.1	d	Q.2	c	Q.3	c	Q.4	d
Q.5	b	Q.6	a	Q.7	d	Q.8	a
Q.9	b	Q.10	a	Q.11	a	Q.12	b
Q.13	d	Q.14	b	Q.15	d	Q.16	c
Q.17	a	Q.18	a	Q.19	a	Q.20	c
Q.21	a	Q.22	c	Q.23	d	Q.24	b
Q.25	c	Q.26	a	Q.27	d	Q.28	b
Q.29	a	Q.30	c	Q.31	c	Q.32	b
Q.33	a	Q.34	a	Q.35	b	Q.36	a
Q.37	d	Q.38	c	Q.39	c	Q.40	a
Q.41	b	Q.42	a	Q.43	a	Q.44	c
Q.45	a						



# **5**

## **Multithreading in Java**

### **Syllabus**

**Concurrency and Synchronization, Java Thread Model :** Thread priorities, Synchronization, Messaging, Main Thread, Creating thread: Implementing Thread using thread class and Runnable interface. Creating multiple threads using is Alive() and join().

**Web Based Application in Java :** Use of JavaScript for creating web based applications in Java, Introduction to Java script frameworks- ReactJS, VueJS, AngularJS (open source).

### **Contents**

- 5.1 *Introduction to Thread*
- 5.2 *Creating Thread*
- 5.3 *Thread Model*
- 5.4 *Thread Priorities*
- 5.5 *Synchronization Messaging*
- 5.6 *Creating Multiple Threads using Alive() and join()*
- 5.7 *Introduction to JavaScript*
- 5.8 *Using JavaScript in HTML*
- 5.9 *Use of JavaScript for Creating Web Applications*
- 5.10 *AngularJS*
- 5.11 *ReactJS*
- 5.12 *VueJS*
- 5.13 *Multiple Choice Questions*

**Part I : Concurrency and Synchronization, Java Thread Model****5.1 Introduction to Thread**

**Definition of thread :** Thread is a tiny program running continuously. It is sometimes called as light-weight process.

**Review Question**

1. Define - thread.

**5.2 Creating Thread**

- In Java we can implement the thread programs using two approaches -
  - Using **Thread class**
  - using **Runnable interface**.

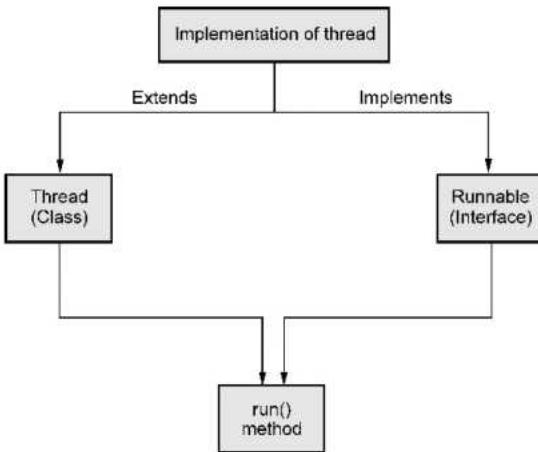


Fig. 5.2.1 : Creation of Java thread

As given in Fig. 5.2.1, there are two methods by which we can write the Java thread programs one is by extending **thread class** and the other is by implementing the **Runnable interface**.

- The **run()** method is the most important method in any threading program. By using this method the thread's behaviour can be implemented. The run method can be written as follows -

```
public void run()
{
    //statements for implementing thread
}
```

For invoking the thread's **run** method the object of a thread is required. This object can be obtained by creating and initiating a thread using the **start()** method.

### 5.2.1 Extending Thread Class

- The **Thread** class can be used to create a thread.
  - Using the **extend** keyword your class extends the **Thread** class for creation of thread. For example if I have a class named **A** then it can be written as
- ```
class A extends Thread
```
- Constructor of Thread Class :** Following are various syntaxes used for writing the constructor of Thread Class.

```
Thread()
Thread(String s)
Thread(Runnable obj)
Thread(Runnable obj, String s);
```

- Various commonly used methods during thread programming are as given below -

| Method    | Purpose                                                 |
|-----------|---------------------------------------------------------|
| start()   | The thread can be started and invokes the run method.   |
| run()     | Once thread is started it executes in run method.       |
| setName() | We can give the name to a thread using this method.     |
| getName() | The name of the thread can be obtained using this name. |
| join()    | This method waits for thread to end                     |

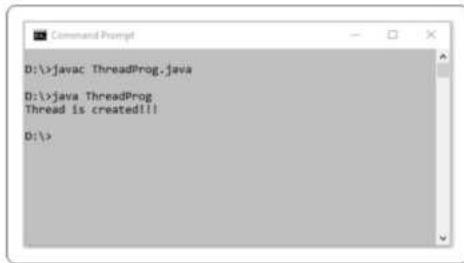
- Following program shows how to create a single thread by extending Thread Class.

#### Java Program

```
class MyThread extends Thread
{
    public void run()
    {
        System.out.println("Thread is created!!!");
    }
}
```

```
}

class ThreadProg
{
    public static void main(String args[])
    {
        MyThread t=new MyThread();
        t.start();
    }
}
```

**Output****Program Explanation : In above program,**

- 1) We have created two classes.
- 2) One class named **MyThread** extends the **Thread** class. In this class the **run** method is defined.
- 2) This **run** method is called by **t.start()** in **main()** method of another class **ThreadProg**
- 3) The thread gets created and executes by displaying the message “Thread is created!!!”

**5.2.2 Using Runnable Interface**

- The thread can also be created using **Runnable** interface.
- Implementing thread program using **Runnable interface** is **preferable** than implementing it by extending the **thread** class because of the following two reasons -
  1. If a class extends a **thread class** then it can not extends any other class which may be required to extend.
  2. If a class **thread** is extended then all its functionalities get inherited. This is an expensive operation.

- Following Java program shows how to implement **Runnable interface** for creating a single thread.

### Java Program

```
class MyThread implements Runnable
{
    public void run()
    {
        System.out.println("Thread is created!");
    }
}
class ThreadProgRunn
{
    public static void main(String args[])
    {
        MyThread obj=new MyThread();
        Thread t=new Thread(obj);
        t.start();
    }
}
```

### Output



### Program Explanation :

- In above program, we have used interface Runnable.
- While using the interface, it is necessary to use **implements** keyword.
- Inside the main method
  1. Create the instance of class **MyClass**.
  2. This instance is passed as a parameter to **Thread** class.

3. Using the instance of class **Thread** invoke the **start** method.
4. The start method in-turn calls the **run** method written in **MyClass**.

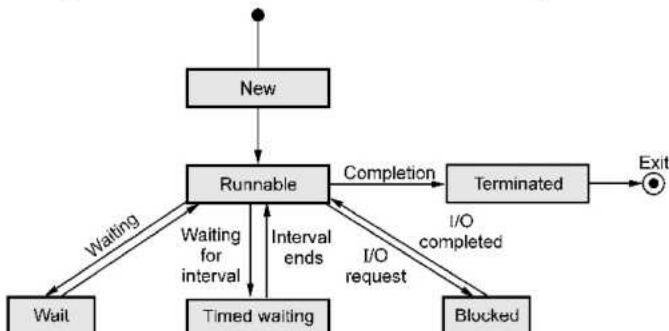
The **run** method executes and display the message for thread creation.

### Review Question

1. Explain different methods of creating threads in Java

### 5.3 Thread Model

The Thread model represents the life cycle which specifies how a thread gets processed in the Java program. By executing various methods. Following figure represents how a particular thread can be in one of the state at any time.



**Fig. 5.3.1 Thread model**

Let us get introduced with each of these methods one by one -

#### 1) New state

- When a thread starts its life cycle it enters in the new state or a create state.

#### 2) Runnable state

- This is a state in which a thread starts executing.
- Waiting state
- Sometimes one thread has to undergo in waiting state because another thread starts executing.

#### 3) Timed waiting state

- There is a provision to keep a particular threading waiting for some time interval. This allows to execute high prioritized threads first. After the timing gets over, the

thread in waiting state enters in runnable state.

#### 4) Blocked state

- When a particular thread issues an Input/Output request then operating system sends it in blocked state until the I/O operation gets completed. After the I/O completion the thread is sent back to the runnable state.

#### 5) Terminated state

- After successful completion of the execution the thread in runnable state enters the terminated state.

#### Review Question

- Explain life cycle methods of thread model in Java.

### 5.4 Thread Priorities

- In Java threads scheduler selects the threads using their priorities.
- The thread priority is a simple integer value that can be assigned to the particular thread.
- These priorities can range from 1 (lowest priority) to 10 (highest priority).
- There are two commonly used functionalities in thread scheduling -

1. setPriority
2. getPriority

- The function **setPriority** is used to set the priority to each thread

```
Thread_Name.setPriority(priority_val);
```

The `priority_val` is a constant value denoting the priority for the thread. It is defined as follows -

- MAX\_PRIORITY = 10

- MIN\_PRIORITY = 1

- NORM\_PRIORITY = 5

- The function **getPriority** is used to get the priority of the thread.

```
Thread_Name.getPriority();
```

- Preemption**

- Preemption is a situation in which when the currently executed thread is suspended temporarily by the highest priority thread.

- The highest priority thread always preempts the lowest priority thread.

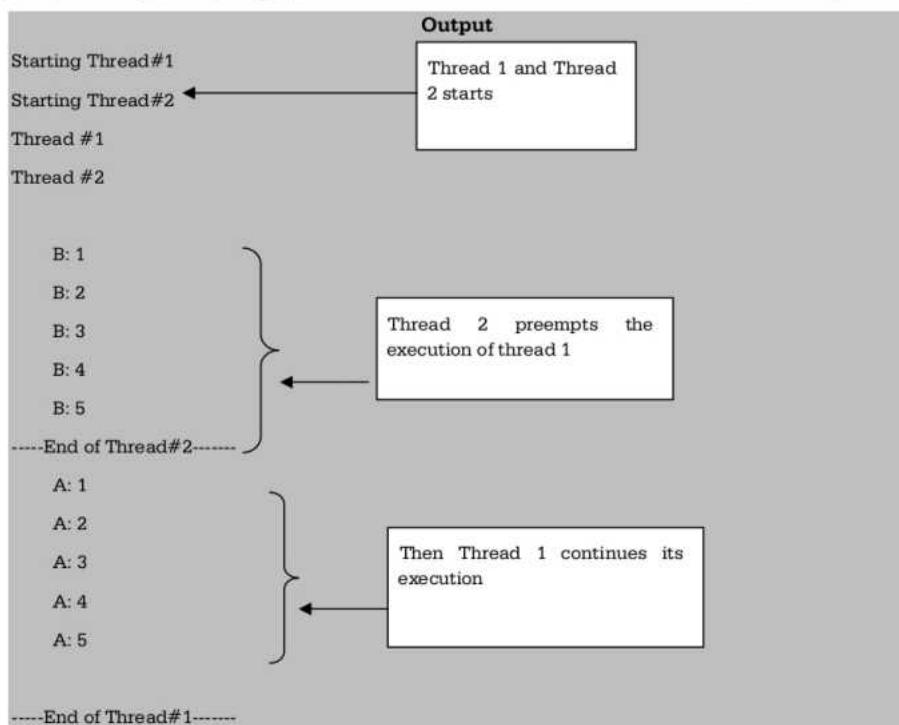
Let us see the illustration of thread prioritizing with the help of following example -

**Java program [Thread\_PR\_Prog.java]**

```
class A extends Thread
{
    public void run()
    {
        System.out.println("Thread #1");
        for(int i=1;i<=5;i++)
        {
            System.out.println("\tA: "+i);
        }
        System.out.println("\n-----End of Thread#1-----");
    }
}

class B extends Thread
{
    public void run()
    {
        System.out.println("Thread #2");
        for(int k=1;k<=5;k++)
        {
            System.out.println("\tB: "+k);
        }
        System.out.println("\n-----End of Thread#2-----");
    }
}

class Thread_PR_Prog
{
    public static void main(String[] args)
    {
        A obj1=new A();
        B obj2=new B();
        obj1.setPriority(1);
        obj2.setPriority(10);//highest priority
        System.out.println("Starting Thread#1");
        obj1.start();
        System.out.println("Starting Thread#2");
        obj2.start();
    }
}
```



### Program explanation

In above program

- 1) We have created two threads - the Thread#1 and Thread#2.
- 2) We assign highest priority to Thread#2.
- 3) In the main function both the threads are started but as the Thread#2 has higher priority than the thread#1, the Thread#2 preempts the execution of Thread#1. Thus Thread#2 completes its execution and then Thread#1 completes.

### Review Questions

1. What are different methods used to set priorities of thread in Java ?
2. Write a Java program illustrating the concept of thread priority.

## 5.5 Synchronization Messaging

- **Definition :** When two or more threads need to access shared memory, then there is some way to ensure that the access to the resource will be by only one thread at a time. The process of ensuring one access at a time by one thread is called **synchronization**.
- The synchronization is the concept which is based on monitor. Monitor is used as mutually exclusive lock or mutex. When a thread owns this monitor at a time then the other threads can not access the resources. Other threads have to be there in waiting state.
- In Java every object has implicit monitor associated with it. For entering in object's monitor, the method is associated with a keyword synchronized. When a particular method is in synchronized state then all other threads have to be there in waiting state.
- There are two ways to achieve the synchronization -

1. Using Synchronized Methods
2. Using Synchronized Blocks (Statements).

Let us make the method synchronized to achieve the synchronization by using following Java program -

### 1. Using Synchronized Method

```
class Test
{
    synchronized void display(int num)
    {
        System.out.println("\nTable for "+num);
        for(int i=1;i<=10;i++)
        {
            System.out.print(" "+num*i);
        }
        System.out.print("\nEnd of Table");
    try
    {
        Thread.sleep(1000);
    }
}
```

This is a synchronized method

```
        }catch(Exception e){}
    }
}

class A extends Thread
{
    Test th1;
    A(Test t)
    {
        th1=t;
    }
    public void run()
    {
        th1.display(2);
    }
}

class B extends Thread
{
    Test th2;
    B(Test t)
    {
        th2=t;
    }
    public void run()
    {
        th2.display(100);
    }
}

class MySynThread
{
    public static void main(String args[])
    {
        Test obj=new Test();
```

```
A t1=new A(obj);
B t2=new B(obj);
t1.start();
t2.start();
}
}
```

**Output**

```
D:\>javac MySynThread.java
D:\>java MySynThread
Table for 2
2 4 6 8 10 12 14 16 18 20
End of Table
Table for 100
100 200 300 400 500 600 700 800 900 1000
End of Table
D:\>
```

**Program Explanation :**

- In above program we have written one class named **Test**. Inside this class the synchronized method named **display** is written. This method displays the table of numbers.
- We have written two more classes named **A** and **B** for executing the thread. The constructors for class **A** and Class **B** are written and to initialize the instance variables of class **Test** as **th1** (as a variable for class **A**)and **th2**(as a variable for class **B**)
- Inside the **run** methods of these classes we have passed number 2 and 10 respectively and **display** method is invoked.
- The **display** method executes firstly for thread **t1** and then for **t2** in synchronized manner.

**2. Using Synchronized Block**

- When we want to achieve synchronization using the synchronized block then create a block of code and mark it as synchronized.

- Synchronized statements must specify the object that provides the intrinsic lock.

## Syntax

The syntax for using the synchronized block is -

```
synchronized(object reference )  
{  
statement;  
statement; //block of code to be synchronized  
. . .  
}
```

## Java Program

```
class Test  
{  
    void display(int num)  
    {  
        synchronized(this)  
        {  
            System.out.println("\nTable for "+num);  
            for(int i=1;i<=10;i++)  
            {  
                System.out.print(" "+num*i);  
            }  
            System.out.print("\nEnd of Table");  
            try  
            {  
                Thread.sleep(1000);  
            }catch(Exception e){}  
        }  
    }  
}  
class A extends Thread  
{  
    Test th1;  
    A(Test t)  
    {  
        th1=t;  
    }  
    public void run()  
    {  
        th1.display(2);  
    }  
}
```

This is a synchronized Block

```
}

class B extends Thread
{
    Test th2;
    B(Test t)
    {
        th2=t;
    }
    public void run()
    {
        th2.display(100);
    }
}
class MySynThreadBlock
{
    public static void main(String args[])
    {
        Test obj=new Test();
        A t1=new A(obj);
        B t2=new B(obj);
        t1.start();
        t2.start();
    }
}
```

**Output**

```
D:\>javac MySynThreadBlock.java
D:\>java MySynThreadBlock
Table for 2
 2 4 6 8 10 12 14 16 18 20
End of Table
Table for 100
 100 200 300 400 500 600 700 800 900 1000
End of Table
D:\>
```

**Review Questions**

1. What is thread synchronization ?
2. Explain how to achieve thread synchronization in Java ?

## 5.6 Creating Multiple Threads using Alive() and join()

There are two methods that are commonly used while handling more than one thread in the Java program and these methods are **isAlive()** and **join()**. The purpose of these two methods is to check whether a thread has finished its execution or not.

### 5.6.1 The isAlive() Method

This method is used to check if the thread is alive or not. This method returns **true** if thread is still running and not finished otherwise it returns false.

#### Syntax

```
public final boolean isAlive()
```

#### Java Program

```
public class ThreadDemo extends Thread  
{  
    public void run()  
    {  
        System.out.println("Begin");  
        try {  
            Thread.sleep(500);  
        }  
        catch(InterruptedException ie)  
        {  
        }  
        System.out.println("End");  
    }  
    public static void main(String[] args)  
    {  
        ThreadDemo t1=new ThreadDemo();  
        ThreadDemo t2=new ThreadDemo();  
        t1.start();  
        System.out.println("Is Thread1 alive? "+t1.isAlive());  
        System.out.println("Is Thread2 alive? "+t2.isAlive());  
        t2.start();  
        System.out.println("Now Is Thread2 alive? "+t2.isAlive());  
    }  
}
```

**Output**

```
C:\Windows\System32\cmd.exe
D:\JavaPrograms>javac ThreadDemo.java
D:\JavaPrograms>java ThreadDemo
Begin
Is Thread1 alive? true
Is Thread2 alive? false
Now Is Thread2 alive? true
Begin
End
End

D:\JavaPrograms>
```

**Program Explanation:** In above Java program

- 1) We have created two threads **t1** and **t2**.
- 2) Using **start** method, **t1** starts running and we display the status of thread **t1** using **isAlive()** method in **System.println** statement. On the Output screen we get its status as **true**.
- 3) But as thread **t2** is not started, we get the return value of method **isAlive()** for **t2** as **false**.
- 4) Once we start running the thread **t2** using **t2.start()** we get the output of **isAlive()** method **true**.

### 5.6.2 The join() Method

This method waits until the thread on which it is called terminates.

#### Syntax

```
final void join() throws InterruptedException
```

Using **join** method, we tell our thread to wait until the specified thread completes its execution.

#### Java Program

```
public class ThreadDemo extends Thread
{
    public void run()
    {
        System.out.println("Begin");
        try {
            Thread.sleep(500);
        }
        catch(InterruptedException ie)
        {
            System.out.println("Exception caught : " + ie);
        }
    }
}
```

```
    }
    catch(InterruptedException ie)
    {
    }
    System.out.println("End");
}
public static void main(String[] args)
{
    ThreadDemo t1=new ThreadDemo();
    ThreadDemo t2=new ThreadDemo();
    System.out.println("Starting first thread");
    t1.start();
    try
    {
        t1.join(); //waiting for t1 to finish
    }
    catch(InterruptedException e)
    {
    }
    System.out.println("Starting another thread");
    t2.start();
}
}
```

**Output**

```
C:\Windows\System32\cmd.exe
D:\JavaPrograms>javac ThreadDemo.java
D:\JavaPrograms>java ThreadDemo
Starting first thread
Begin
End
Starting another thread
Begin
End

D:\JavaPrograms>
```

**Review Question**

1. Illustrate the use of `IsAlive()` in Java.

## Part II : Web Based Application in Java

### 5.7 Introduction to JavaScript

JavaScript was developed by Netscape in 1995. At that time its name was LiveScript. Later on Sun Microsystems joined the Netscape and then they developed **LiveScript**. And later on its name is changed to **JavaScript**.

#### 5.7.1 Difference between Java and JavaScript

| Sr. No. | Java                                                                                                    | JavaScript                                                                                                                                                           |
|---------|---------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1.      | Java is a programming language.                                                                         | JavaScript is a scripting language.                                                                                                                                  |
| 2.      | Java is an object oriented programming language.                                                        | Javascript is not a object oriented programming language. Its object model is far different than a typical object oriented programming language such as C++ or Java. |
| 3.      | Java is strongly typed language and the type checking is done at compile time.                          | In JavaScript variables need not be declared before their use. Checking the compatibility of type can be done dynamically.                                           |
| 4.      | Objects in Java are static. That means the number of data members and method are fixed at compile time. | In JavaScript the objects are dynamic. That means we can change the total data members and method of an object during execution.                                     |

#### 5.7.2 Features of JavaScript

Following are some features of JavaScript

- Browser Support :** For running the JavaScript in the browser there is no need to use some plug-in. Almost all the popular browsers support JavaScripting.
- Structure Programming Syntax:** The Javascript supports much commonly the structured language type syntax. Similar to C-style the programming blocks can be written.
- It automatically inserts the semicolon at the end of the statement, hence there is no need to write semicolon at the end of the statement in JavaScript.

4. **Dynamic Typing** : It supports dynamic typing, that means the data type is bound to the value and not to the variable. For example one can assign integer value to a variable say 'a' and later on can assign some string value to the same variable in JavaScript.
5. **Run Time Evaluation** : Using the **eval** function the expression can be evaluated at run time.
6. **Support for Object** : JavaScript is object oriented scripting language. However handling of objects in JavaScript is somewhat different than the conventional object oriented programming languages. JavaScript has a small number of in-built objects.
7. **Regular Expression** : JavaScript supports use of regular expressions using which the text-pattern matching can be done. This feature can be used to validate the data on the web page before submitting it to the server.
8. **Function Programming** : In JavaScript functions are used. One function can accept another function as a parameter. Even, one function can be assigned to a variable just like some data type. The function can be run without giving the name.

#### Review Question

1. *Enlist the features of JavaScript.*

### 5.8 Using JavaScript in HTML

The JavaScript can be directly embedded within the HTML document or it can be stored as external file.

#### Directly embedded JavaScript

The syntax of directly embedding the JavaScript in the HTML is

```
<script type="text/javascript">  
    ...  
    ...  
    ...  
</script>
```

**Example 5.8.1** Write a JavaScript to display Welcome message in JavaScript.

**Solution :**

```
<!DOCTYPE html >
<html >
  <head>
    <title> My First Javascript </title>
  </head>
  <body>
    <center>
      <script type="text/javascript">
        /*This is the First JavaScript*/
        document.write(" Welcome to First Page of Javascript");
      </script>
    </center>
  </body>
</html>
```

### Output



### Script Explanation :

In above script

- (1) we have embedded the javaScript within

```
<script type="text/javascript">
...
</script>
```

- (2) There is comment statement using /\* and \*/. Note that this type of comment will be recognized only within the <script> tag. Because, JavaScript supports this kind of comment statement and not the XHTML document.

- (3) Then we have written **document.write** statement, using which we can display the desired message on the web browser.

### Indirectly Embedding JavaScript

If we want to embed the JavaScript indirectly, that means if the script is written in some another file and we want to embed that script in the HTML document then we must write the script tag as follows -

```
<script type="text/javascript" src="MyPage.js">
...
...
</script>
```

Javascript is which is to be embedded is in the file MyPage.js

We will follow the following steps to use the external JavaScript file.

**Step 1 : Create an XHTML document as follows -**

**XHTML Document[FirstPg.html]**

```
<!DOCTYPE html >
<html>
    <head>
        <title> My First Javascript </title>
    </head>
    <body>
        <center>
            <script type="text/javascript" src="my_script.js">
            </script>
        </center>
    </body>
</html>
```

This is an external javascript file,it can be specified with the attribute **src**

**Step 2 :**

**JavaScript[my\_script.js]**

```
/*This is the First JavaScript*/
document.write(" Welcome to First Page of Javascript");
```

**Step 3 : Open the HTML document in Internet Explorer and same above mentioned output can be obtained.**

**Advantages of indirectly embedding of JavaScript**

1. Script can be hidden from the browser user.
2. The layout and presentation of web document can be separated out from the user interaction through the JavaScript.

**Disadvantages of indirectly embedding of JavaScript**

1. If small amount of JavaScript code has to be embedded in XHTML document then making a separate JavaScript file is meaningless.
2. If the JavaScript code has to be embedded at several places in XHTML document then it is complicated to make separate JavaScript file and each time invoking the code for it from the XHTML document.

**Review Question**

1. Explain the ways by which the JavaScript can be embedded in web document.

**5.9 Use of JavaScript for Creating Web Applications**

The JavaScript has the operators and control structures just similar to Java / C or C++. In this section we will discuss only few essential features that are used in JavaScript for building the web application.

**5.9.1 Input and Output**

- For displaying the message on the web browser the basic method being used is **document.write**. To print the desired message on the web browser we write the message in double quotes inside the document.write method.
- **For example**  
`document.write("Great India");`
- We can pass the variable instead of a string as a parameter to the document.write.

**For example**

```
var my_msg="Great India";
document.write(my_msg);
```

Note that the variable my\_msg should not be passed in double quotes to document.write otherwise the result the string "my\_msg" will be printed on the browser instead of the string "Great India".

- We can combine some HTML tags with the variable names in document.write so that the formatted display can be possible on the web browser.

**For example** - if we want to print the message "Great India" on the web browser in bold font then we can write following statements -

```
var my_msg="Great_India";
document.write("<strong>" +my_msg+ "</strong>");
```

### 5.9.2 Pop-Up Box

- One of the important features of JavaScript is its **interactivity** with the user.
- There are **three types of popup boxes** used in JavaScript by which user can interact with the browser.

**alert box :** In this type of popup box some message will be displayed.



**confirm box :** In this type of popup box in which the message about confirmation will be displayed. Hence it should have two buttons **Ok** and **Cancel**.



**Prompt box** is a type of popup box which displays a text window in which the user can enter something. Hence it has two buttons **Ok** and **Cancel**.



#### JavaScript[PopupBox.html]

```
<!DOCTYPE html>
<html >
<head>
<title>Introduction to pop up box</title>
</head>
<body>
<p>Experiment with the popup boxes by clicking the buttons(OK and Cancel) on them</p>

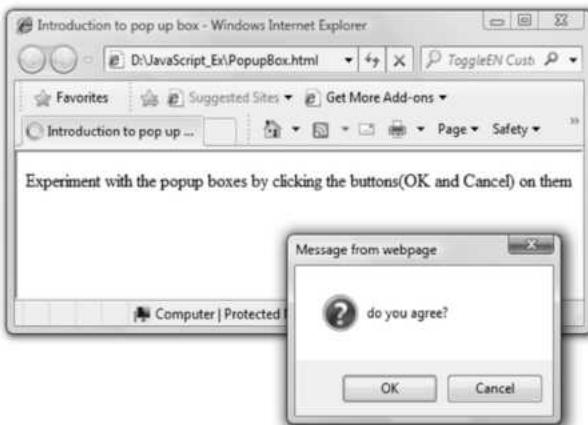
<script type="text/javascript">
if(confirm("do you agree?"))
    alert("You have agreed");

```

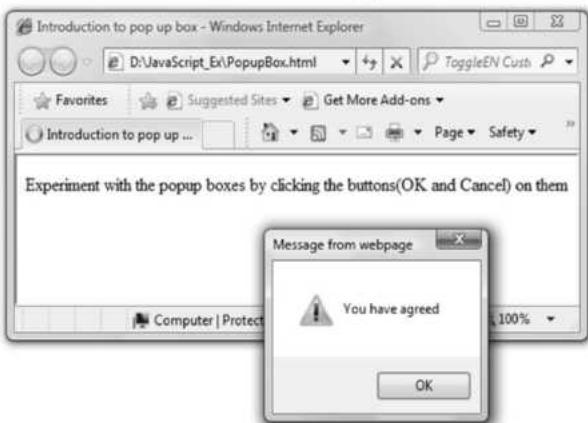
```

else
    input_text=prompt("Enter some string here...","");
/*the value entered in prompt box is returned
and stored in the variable text */
    alert("Hi "+input_text);
</script>
</body>
</html>

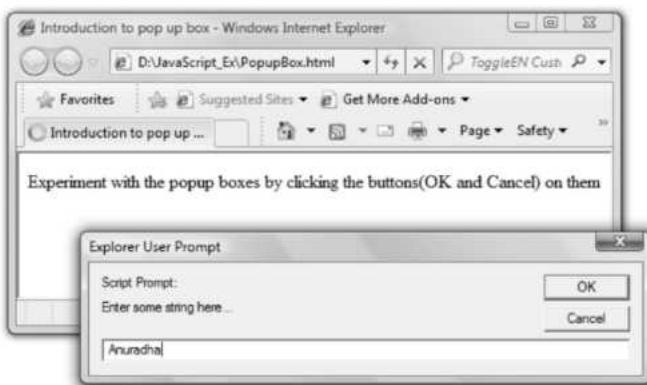
```

**Output**

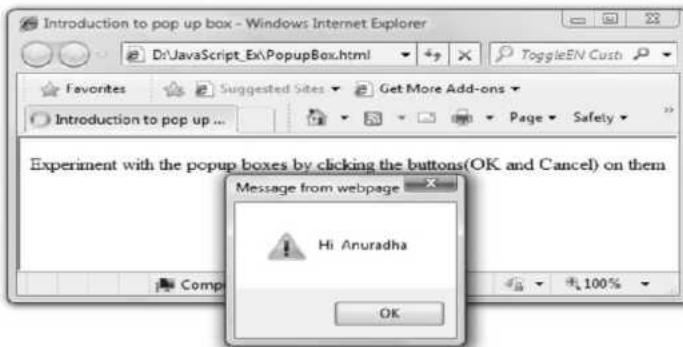
If we click on OK button then an alert box will appear.



On load, the confirm box appears and if we click on Cancel button then the prompt box will appear. We can type some string within it.



Click OK button and we will get the alert box as follows -



Thus alert, confirm and prompt boxes cause the browser to wait for user response. The user responds by clicking the button on these popup boxes.

### **Script Explanation**

On loading this script using web browser first of all a confirm box will be displayed. If we click on OK button an alert box will appear. Otherwise a prompt box will be displayed. Again if we click on the OK button of prompt box again an alert box will appear which will display the string which you have entered on prompt box. If you run the above script you will get all these effects.

#### **5.9.3 Function**

- We can write the functions in the JavaScript for bringing the modularity in the script.

- Separate functions can be created for each separate task. This ultimately helps in finding the bug from the program efficiently.
- We can define the function anywhere in the script either in head or body section or in both. But it is a standard practice to define the function in the head section and call that function from the body section.
- The keyword **function** is used while defining the function.
- The syntax for defining the function is

```
function name_of_function (arg1,arg2,...argn)  
{  
...  
Statements  
}
```

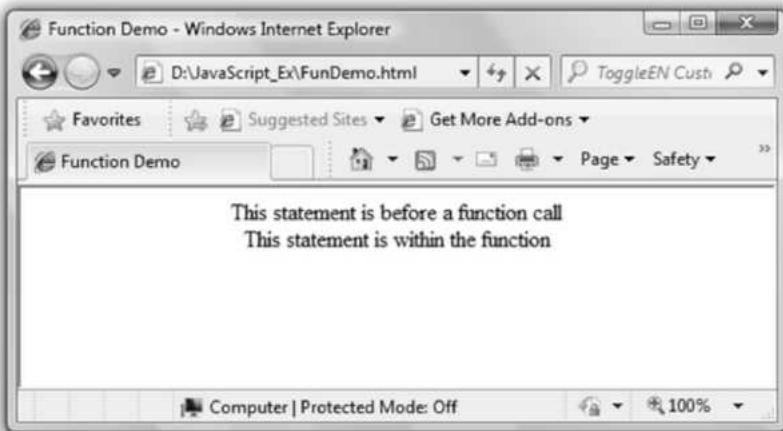
Here is a simple illustration in which we have written a function **my\_fun()**

#### JavaScript[FunDemo.html]

```
<!DOCTYPE html>  
<html>  
<head>  
<title>Function Demo</title>  
<script type="text/javascript">  
function my_fun()  
{  
    document.write("This statement is within the function");  
}  
</script>  
</head>  
<body>  
<center>  
<script type="text/javascript">  
    document.write("This statement is before a function call");  
    document.write("<br>");  
    my_fun();  
    call to the function  
</script>  
</center>  
</body>  
</html>
```



Definition of function

**Output****Script Explanation**

The above code is pretty simple. We have defined one function named `my_fun` in the **head** section of the HTML document and called it from the **body** section. The corresponding `write` statements are used to display the messages on the browser window.

**5.9.4 Form Design**

- Form is a typical layout on the web page by which a user can interact with the web page.
- Typical component of forms are text, **text area**, **checkboxes**, **radio buttons** and **push buttons**. These components of form are also called as **form controls or controls**.
- HTML allows us to place these form components on the web page and send the desired information to the destination server.
- All these form contents appear in the `<form>` tag. The form has an **attribute action** which gets executed when user clicks a button on the form.

**5.9.4.1 Text**

- Text is typically required to place one line text. For example if you want to enter some name then it is always preferred to have Text field on the form.
- The text field can be set using  
`<input type="text" size="30" name = "username" value=" ">`

The input type is **text** and the **value** of this text field is “ ” That means the blank text field is displayed initially and we can enter the text of our choice into it. There is **size** parameter which allows us to enter some size of the text field.

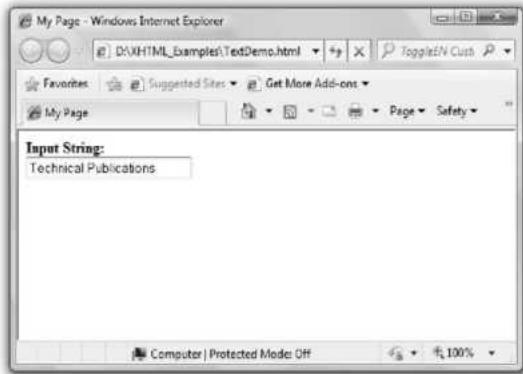
- Some other parameters or attributes can be
  - **maxlength** that allows us to enter the text of some maximum length.
  - **name** indicates name of the text field.
  - **align** denotes the alignment of the text in the text field. The alignment can be left, right, bottom and top.

### Example Code

#### HTML Document [TextDemo.html]

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Page</title>
  </head>
  <body>
    <form>
      <b>Input String:</b><br/><input type="text" size="25" value="">
    </form>
  </body>
</html>
```

#### Output



### Script Explanation :

In above document

- 1) We have the label “Input String” just before the **<input>** tag. We can also specify the

label by using the **<label>** tag as follows -

```
<label>Input String: <br/><input type="text" size="25" value=""></label>
```

- 2) Thus the label gets bound to the text box. This aspect is always beneficial for a web programmer because using label control we can focus on the corresponding text box contents.
- 3) Initially the text box field is blank. We can type some text inside this text box.

#### 5.9.4.2 Checkbox

- It is the simplest component which is used particularly when we want to make some selection from several options.
- For having the checkbox we have to specify the input type as **checkbox**. For example

```
<input type="checkbox" name="option1" value="mango" checked="checked">Mango<br/>
```

- If we want to get the checkbox displayed as checked then set **checked="checked"**

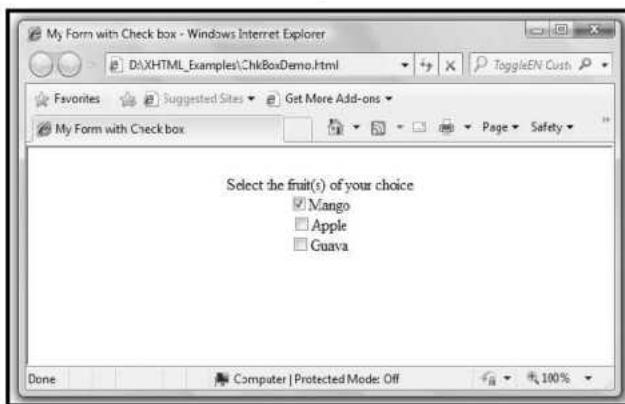
#### Example Code

##### HTML Document[ChkBoxDemo.html]

```
<!DOCTYPE html>

<html>

<head>
    <title>My Form with Check box</title>
</head>
<body>
    <form name ="checkboxForm">
        <div align="center"><br>
            Select the fruit(s) of your choice<br />
            <input type="checkbox" name="option1" value="mango"
checked="checked">Mango<br/>
            <input type="checkbox" name="option2" value="apple">Apple<br/>
            <input type="checkbox" name="option3" value="guava">Guava<br/>
        </div>
    </form>
</body>
</html>
```

**Output****Script Explanation**

- 1) In the above program to set some checkbox in checked state we can mention the attribute **checked** as **checked**.
- 2) We can set the **value** attribute as “ ” but this then the checkbox will not get associated with any value. The **Mango**, **Apple** and **Guava** are the labels of the checkboxes.

**5.9.4.3 Radio Button**

- This form component is also used to indicate the selection from several choices.
- Using **input type="radio"** we can place radio button on the web page.
- This component allows us to make only one selection at a time.
- We can create a group of some radio button component.
- Following HTML document displays the radio buttons for two different groups.

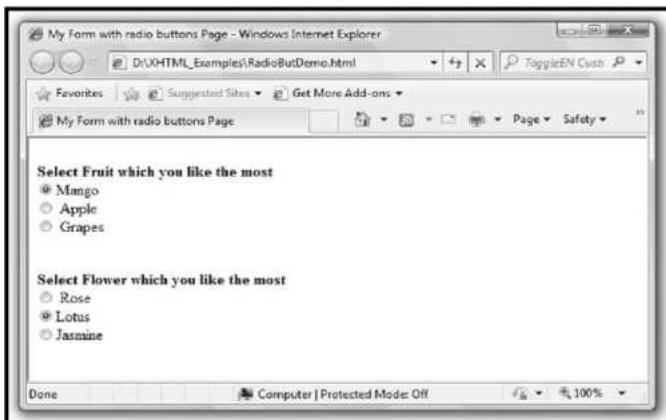
**HTML Document[RadioButDemo.html]**

```
<!DOCTYPE html>
<html >
  <head>
    <title>My Form with radio buttons Page</title>
  </head>
  <body>
    <form name="myform">
      <div align="left"><br>
        <b>Select Fruit which you like the most</b><br>
```

```

<input type="radio" name="group1" value="Mango">Mango<br/>
<input type="radio" name="group1" value="Apple" checked> Apple<br/>
<input type="radio" name="group1" value="Grapes"> Grapes
<br/><br/><br/>
<b>Select Flower which you like the most</b><br/>
<input type="radio" name="group2" value="Rose"> Rose<br/>
<input type="radio" name="group2" value="Lotus"> Lotus<br/>
<input type="radio" name="group2" value="Jasmine" checked> Jasmine<br/>
</div>
</form>
</body>
</html>

```

**Output****5.9.4.4 Button**

- We can create the button using `<input type = "button">` tag.
- There are two types of buttons that can be created in HTML. One is called **submit** button and the another one is **reset** button.
- Various parameters of submit button are
  - 1) **name** denotes the name of the submit button.
  - 2) **value** is for writing some text on the text on the button.
  - 3) **align** specifies alignment of the button.

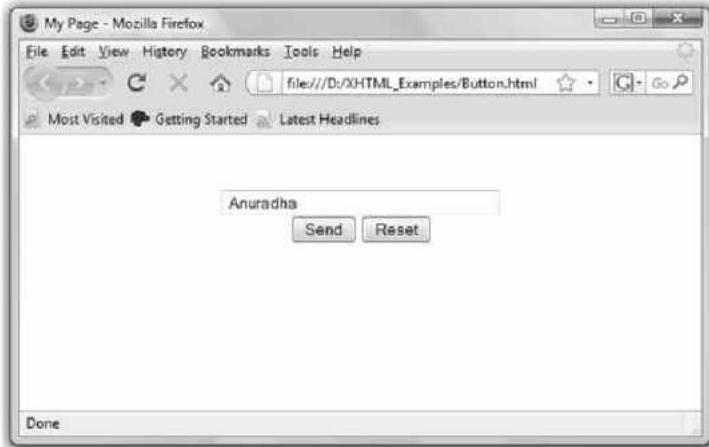
## Example Code

### HTML Document[Button.html]

```
<!DOCTYPE html>
<html >
    <head >
        <title > My Page </title >
    </head >
    <body >

<form name = "myform" action = "http://www.localhost.com/cgi-bin/hello.cgi" method = "POST" >
    <div align = "center" >
        <br /> <br />
        <input type = "text" size = "35" value = " " >
        <br /> <input type = "submit" value = "Send" >
        <input type = "reset" value = "Reset" > <br />
    </div >
</form >
</body >
</html >
```

### Output



## Script Explanation

- 1) In above HTML document, we have used the form whose name is "**myform**".
- 2) There are two attributes associated with the form tag and those are **action** and **method**. The **action** parameter indicates the address and the cgi script where the contents should go and **method** parameter is for the methods for submitting the

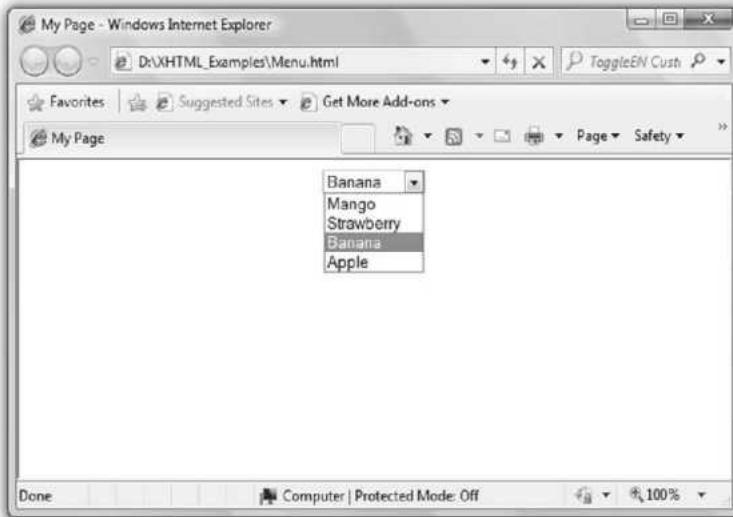
data. The **method** can be **get** or **post**. Thus by specifying the action and method for a form we can send the desired data at desired location.

#### 5.9.4.5 Select Element

- HTML allows us to have pop down menu on the web page so that the desired selection can be made.
- The parameter **select** is for the menu component and **option** parameter is for setting the values to the options of drop down menu.
- We can make some specific option selected by **selected value = .**
- In the following HTML document we have created one drop down menu in which various fruits are enlisted. By default “Banana” is set as selected.

#### HTML Document [Menu.html]

```
<!DOCTYPE html>
<html >
  <head>
    <title> My Page </title >
  </head>
  <body>
    <form name="myform">
      <div align="center">
        <select name="My_Menu">
          <option value="Mango">Mango</option>
          <option value="Strawberry">Strawberry</option>
          <option selected value="Banana">Banana </option>
          <option value="Apple">Apple</option>
        </select>
      </div>
    </form>
  </body>
</html>
```

**Output**

**Example 5.9.1** Write a form to collect details of a user such as name, address, radio button to choose subject of book he wants to buy, dropdown to choose favorite author, comments for the last book he read.

**Solution :**

```
<!DOCTYPE html >
<html >
<head>
<title>My Page </title>
</head>
<body>
<form>
<b>Name:</b><input type="text" size="20" value=""><br/>
<b>Address:</b><input type="text" size="35" value=""><br/>
<b>Subjects:</b><br/>
<input type="radio" name="authors" value="Web Programming">Web Programming<br/>
<input type="radio" name="authors" value="Computer Network">Computer Network<br/>
<input type="radio" name="authors" value="Software Engineering">Software Engineering<br/>
<input type="radio" name="authors" value="Data Structures">Data Structures<br/>
<b>Select favorite Author:</b>
<select name="MyMenu">
<option value="AAA">AAA</option>
```

```
<option value="BBB">BBB</option>
<option value="CCC">CCC</option>
<option value="DDD">DDD</option>
</select>
<br>
<b>Comments:</b><br>
<textarea cols="30" rows="10" name="comments">
</textarea>
<br/><br/>
<input type="submit" value="Submit"/>
<input type="reset" value="Clear"/>
</form>
</body>
</html>
```

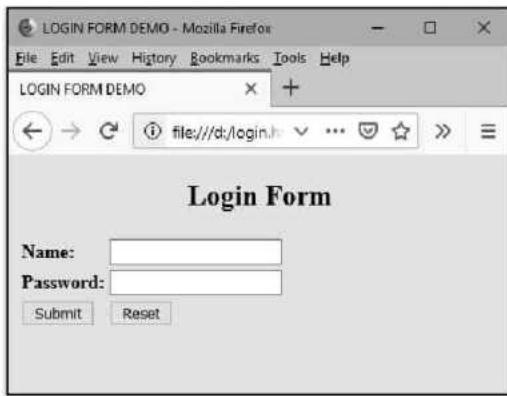
**Output**

The screenshot shows a web browser window titled "My Page". The address bar displays "file:///D:/test.". The page content is a form with the following fields:

- Name:**
- Address:**
- Subjects:**
  - Web Programming
  - Computer Network
  - Software Engineering
  - Data Structures
- Select favorite Author:**
- Comments:**
- Buttons:**

**Example 5.9.2** Write a form to make login and password.**Solution :**

```
<html>
<head>
    <title>LOGIN FORM DEMO</title>
</head>
<body bgcolor="khaki">
    <center>
        <h2>Login Form</h2>
    </center>
    <form name="form1">
        <table>
            <tr>
                <td><b>Name:</b></td>
                <td><input type="text" name="userName">
            </tr>
            <tr>
                <td><b>Password:</b></td>
                <td><input type="password" name="pwd"></td>
            </tr>
            <tr>
                <td><input type="submit" name="submit" value="Submit"></td>
                <td><input type="reset" name="reset" value="Reset"></td>
            </tr>
        </table>
    </form>
</body>
</html>
```

**Output**

## Part III : Introduction to JavaScript Framework

### 5.10 AngularJS

- Angular JS is a powerful JavaScript framework.
- It is an open source front-end enabled web application framework.
- It is licensed under Apache license version 2.0.
- It extends HTML DOM with additional attributes. It is more responsive to user actions.

#### Advantages

Following are the advantages of Angular JS

1. **Built by Google Engineers** : The Angular JS is maintained by dedicated Google Engineers. That means - there is huge community for development and enhancement.
2. **MVC Support** - It is based on model view controller (MVC) architecture which provides separation of model, view and controller components.
3. **Intuitive** - It is more intuitive as it makes use of HTML as a declarative language. Moreover, it is less brittle for reorganizing.
4. **Comprehensive** : AngularJS is a comprehensive solution for rapid front-end development. It does not need any other plugins or frameworks.
5. **Rich Features** : There is a wide range of features supported by Angular JS such as Data binding, dependency injection, enterprise level testing and so on.
6. **Unit Testing** : The Angular JS code is unit testable.
7. **Reusable Code** : Angular JS support for using the reusable components.
8. **Less Code** : It supports for less code and more functionality.

#### Disadvantages

1. As Angular JS is based on JavaScript framework, it is not secure.
2. There are multiple ways to do the same thing with AngularJS. Sometimes, it can be hard for novices to say which way is better for a task.

#### Features of Angular JS

There are core features of Angular JS that are widely used by web developers. These are -

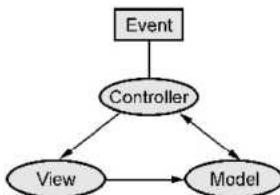
1. **Data Binding** : It allows the automatic synchronization between model and view components.

2. **Scope** : There are some objects from model that can be correlated to controller and view.
3. **Controller** : These are basically JavaScript functions that are bound to particular scope.
4. **Directives** : The directives is designed to give HTML new functionality.
5. **Filters** : This feature allows to select subset from array of items.
6. **Routing** : This feature allows to switch between multiple views.
7. **Services** : There is a support for many built in services for Angular JS.
8. **Dependency Injection** : It is a software design pattern that helps the developer to develop and understand the application easily.

### 5.10.1 MVC Architecture

The MVC stands for Model, view and controller. It is a pattern for the architectural framework. It consists of three parts -

1. **Model** : This part of the architecture is responsible for managing the application data. This module responds to the request made from **view**. The **model** gives instructions to **controller** to update when the response is made.
2. **View** : This part takes care of the presentation of data. The data is presented in desired format with the help of **view**. This is a script based system using JSP, ASP, PHP and so on.
3. **Controller** : The controller receives input, validates it and then performs business operations that modify the state of the data model. The **controller** basically responds to user request and performs interaction with **model**.



### 5.10.2 Directives

- In Angular JS, the capacity of HTML is extended using the **ng-directives**.
- The directives are specified with **ng** prefix.
- The three commonly used ng-directives are
  1. **ng-app** : This directive defines an AngularJS application.
  2. **ng-model** : This directive binds the value of HTML controls to application data. These controls can be input, select, text area and so on.
  3. **ng-bind** : This directive binds application data to HTML view.

Let us now learn and understand how to develop Angular JS application.

**Step 1 :** Create an Angular JS code as follows in a Notepad. Save it using the extension .htm or .html

#### AngularJS Document[FirstApp.html]

```
<html>
  <head>
    <title>My First AngularJS Application</title>
    <script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
  </head>
  <body>
    <h1>AngularJS Application</h1>
    <div ng-app = "">
      <p>Enter your Name: <input type = "text" ng-model = "name"></p>
      <p>Welcome <span ng-bind = "name"></span></p>
    </div>
  </body>
</html>
```

#### Output



#### Script Explanation :

1. The AngularJS script is written within `<html>` tags. The AngularJS is based on JavaScript framework. The framework for AngularJS is loaded using the `<script>` tag. It is as follows -

```
<script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
```

**2. Defining np-app directive :** The np-app directive is defined in above script as follows -

```
<div ng-app = "">  
...  
...  
</div>
```

**3. Defining np-model directive :** The **textbox** control is modeled using np-model directive

```
<p>Enter your Name: <input type = "text" ng-model = "name"></p>
```

**4. Defining np-bind :** We can bind the value entered in the textbox with **name** using np-bind directive.

```
<p>Welcome <span ng-bind = "name"></span></p>
```

Hence, we can see on the output screenshot, that whatever name entered by the user gets bound with the **Welcome** message.

The above output can be viewed by opening any popular web browser and by entering the URL of corresponding AngularJS script.

Various Directives supported by AngularJS are -

**1. ng-app      2. ng-init      3.ng-model      4.ng-repeat**

Let us discuss these directives with suitable scripting examples

#### **1. ng-app Directive :**

This directive starts the AngularJS application. It basically defines the **root element**. It initializes the application automatically when the web page containing the AngularJS code is loaded. For example

```
<div ng-app="">  
...  
</div>
```

#### **2. ng-init Directive :**

The ng-init directive is used for initialization of AngularJS data.

For example

#### **DirectiveDemo1.html**

```
<html>  
<script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>  
<body>  
    <div ng-app="" ng-init="msg='I love my country!!!'">  
        <h3>{{msg}}</h3>  
    </div>  
</body>  
</html>
```

**Output**

3. **ng-model** : The ng-model directive binds the value of HTML control to application data. These controls can be input, select, text area and so on. For example

**AngularJS Document[FirstApp.html]**

```
<html>
  <head>
    <title>My First AngularJS Application</title>
    <script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
  </head>
  <body>
    <h1>AngularJS Application</h1>
    <div ng-app = "">
      <p>Enter your Name: <input type = "text" ng-model = "name"></p>
      <p>Welcome <span ng-bind = "name"></span></p>
    </div>
  </body>
</html>
```

## Output

# AngularJS Application

Enter your Name:

Welcome Mr.XYZ

Name in text box are represented using ng-model directive and is linked with application data using ng-bind

**4. ng-repeat :** The ng-repeat element directive repeats the HTML element.

### AngularJS Document [DirectiveRepeat.html]

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
<body>
<div ng-app="" ng-init="nums=[10,20,30,40,50]">
<p>The elements are -</p>
<ul>
<li ng-repeat="i in nums">
{{ i }}
</li>
</ul>
</div>
</body>
</html>
```

## Output



**Review Question**

1. What is angular JS ? Enlist its advantages and disadvantages.

**5.11 ReactJS**

Let us start the introduction of ReactJS with a real-world example. Suppose that on a Facebook page, you click the “like” on some photograph and then by the time you scroll down the same page, the like count gets incremented by some number, even without reloading your Facebook page, This magic change happens because of out ReactJS!!!

ReactJS is an open source, component-based front end JavaScript library maintained by Facebook.

This library is responsible only for the view layer of the application. That means this JavaScript for building user interfaces.

**Features of ReactJS**

- **Virtual DOM :** DOM stands for Document Object Model. It also provides a way to update the content, structure and style of the document. Virtual DOM is a representation of original DOM. When user updates something on the web application, DOM gets updated. Updating the DOM is very slow, most of the javascript frameworks update the whole DOM which makes it slower. But actually there is no need to update the whole DOM, instead, these frameworks should update only the part of DOM that is required to update. This is what the virtual DOM does. This is why ReactJS's DOM manipulation is much faster than other frameworks. Due to this property, whenever any change is made in the web application, then those changes are reflected on the web page within no time.
- **Components :** This feature allows the web developer to create custom elements which can be reused in HTML.
- **JSX:** JSX is an extension of JavaScript syntax. JSX can be seen as a combination of javascript and XML. The syntax of JSX is very simple that makes writing components very easy.
- **One way Data Binding :** The ReactJS is designed in such a way that it follows, unidirectional or one way data binding. That means, data is allowed to flow in one direction at a time. This helps in achieving greater control over the application. This makes helps in increasing the efficiency of an application.

### 5.11.1 Installation of ReactJS

#### Step 1 : Install NodeJS

Open the site <https://nodejs.org/en/download/> and click on the link with respect to the operating system you have.

The screenshot shows the Node.js download page. It has two main sections: 'LTS' (Recommended For Most Users) and 'Current' (Latest Features). Under 'LTS', there are links for Windows Installer (.msi), Windows Binary (.zip), macOS Installer (.pkg), macOS Binary (.tar.gz), Linux Binaries (x64), Linux Binaries (ARM), and Source Code. Under 'Current', there are links for 32-bit and 64-bit versions of the Windows, macOS, and Linux installers, along with ARMv7 and ARMv8 binary files. The 'Source Code' section also links to node-v14.15.4.tar.gz.

LTS	Current
Windows Installer (.msi)	32-bit
Windows Binary (.zip)	32-bit
macOS Installer (.pkg)	64-bit
macOS Binary (.tar.gz)	64 bit
Linux Binaries (x64)	64-bit
Linux Binaries (ARM)	ARMv7
Source Code	node-v14.15.4.tar.gz
	ARMv8

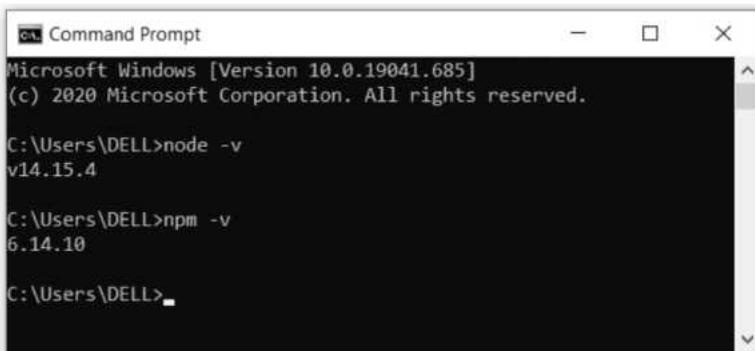
Download the installation file. Click on the Run the software for installation. Accept the defaults and keep clicking the **Next** button. Finally, the NodeJS gets installed on your system.

We can now verify the installation. For that matter open command prompt and enter the commands

```
node -v
```

```
npm -v
```

It is as shown below -



```
Command Prompt
Microsoft Windows [Version 10.0.19041.685]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\DELL>node -v
v14.15.4

C:\Users\DELL>npm -v
6.14.10

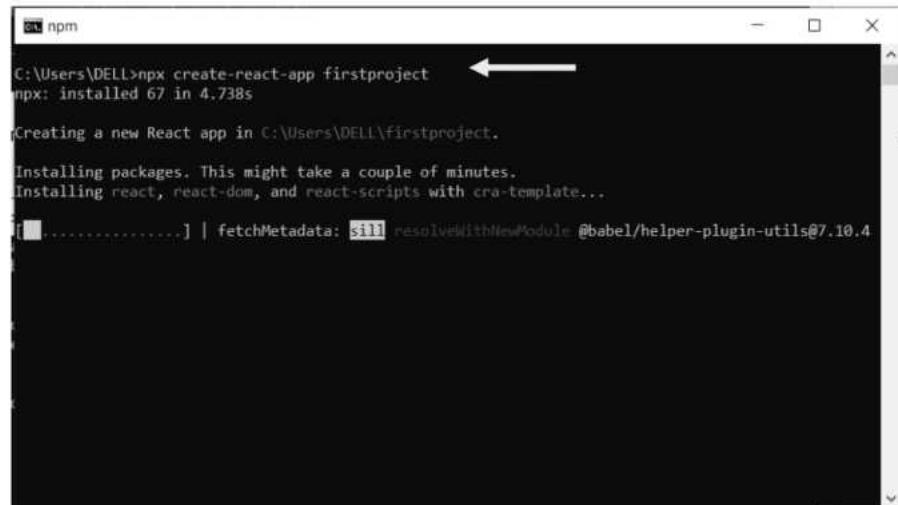
C:\Users\DELL>
```

### Step 2 : Install Create-React-App Tool

The create-react-app needs to be installed will be installed using following prompt window.

```
C:\Users\DELL>npm install -g create-react-app
```

On successful installation, we can create our application by creating a new project by the following command -



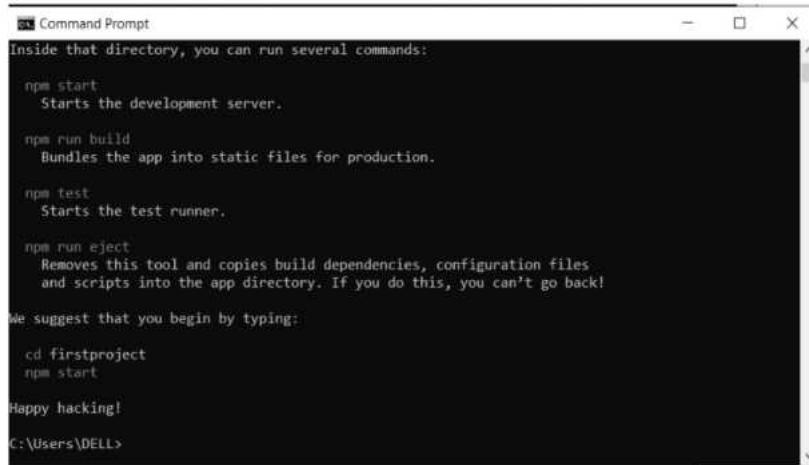
```
npm
C:\Users\DELL>npx create-react-app firstproject ←
npx: installed 67 in 4.738s

Creating a new React app in C:\Users\DELL\firstproject.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...

[■ .....] | fetchMetadata: sill resolveWithNewModule @babel/helper-plugin-utils@7.10.4
```

Now we can see the terminal window as follows -



```
Command Prompt
Inside that directory, you can run several commands:
npm start
  Starts the development server.

npm run build
  Bundles the app into static files for production.

npm test
  Starts the test runner.

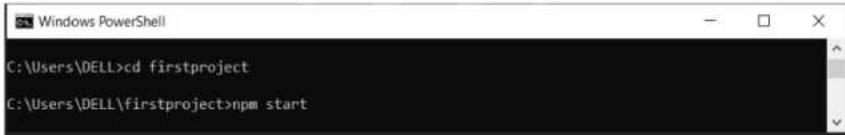
npm run eject
  Removes this tool and copies build dependencies, configuration files
  and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

  cd firstproject
  npm start

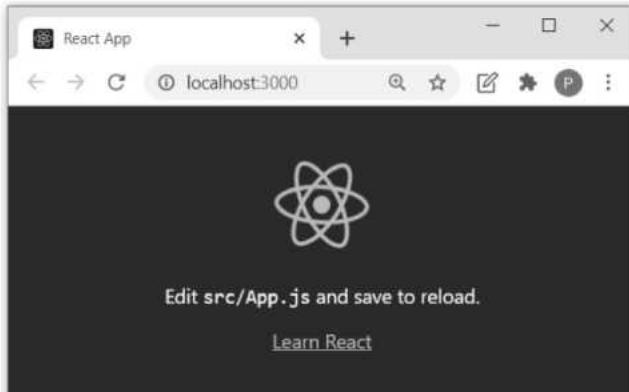
Happy hacking!
C:\Users\DELL>
```

Now we can access the server in order to access the web application. Type the following command in the command prompt window.



```
Windows PowerShell
C:\Users\DELL>cd firstproject
C:\Users\DELL\firstproject>npm start
```

The web browser will show following ReactJS page.



### 5.11.2 Writing Simple Web Applications

In this section, we will discuss how to create a simple web application using ReactJS script.

**Example 5.11.1** Write a ReactJS application to display the "welcome user" message.

**Solution :**

**Step 1 :** Create your project folder(Refer steps mentioned in the installation process).

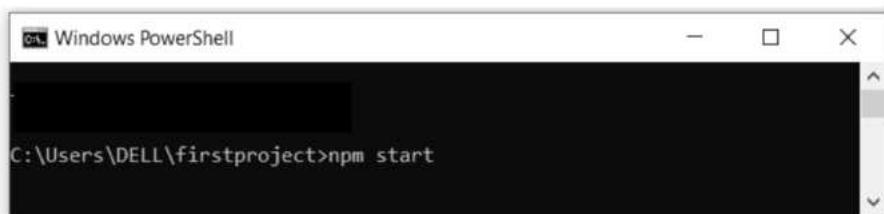
Open the file **index.js** This file is in your project under the **src** folder. Edit it as follows -

```
import React from 'react';
import ReactDOM from 'react-dom';
const mystr = <h1>Welcome User!!!</h1>
ReactDOM.render(mystr, document.getElementById('root'));
```

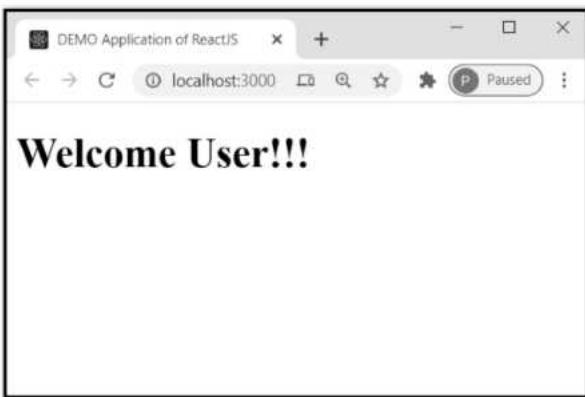
**Step 2 :** Now open index.html file. This file is present in your folder's public folder. Edit it as follows -

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1" />
<title>DEMO Application of ReactJS</title>
</head>
<body>
<div id="root"></div>
</body>
</html>
```

**Step 3 :** Now open the command prompt window and type following command



The browser window will open and will display the output as follows -



### Review Question

1. *What is ReactJS ? Enlist its features.*

### 5.12 VueJS

- Vue.js is an open source progressive framework for JavaScript used to build web interfaces and one-page applications.
- Vue.js is also used for both Desktop and mobile app as a front end development tool.
- The name Vue is for resembling the word “view”.

### Features of VueJS

- 1) Components : This feature allows the web developer to create custom elements which can be reused in HTML.
- 2) Event Handling : Any event can be handled using VueJS script with the help of `v-on` attribute.
- 3) Data Binding : This feature helps to manipulate or assign values to HTML attributes or change the style.
- 4) Animation : Vue JS helps to show animation of the elements on the web page.
- 5) Computed Properties : This is a feature that can immediately sense the changes made in the UI and make necessary actions. There is no need to perform additional coding for that.

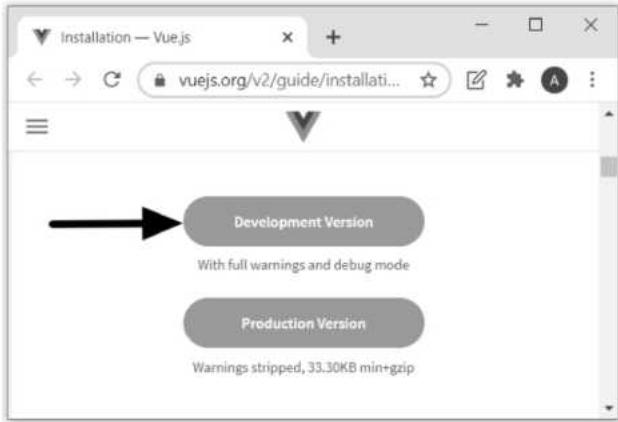
- 6) Ligheweight and Efficient : The Vue JS is very light weight and efficient JavaScript framework.

### 5.12.1 Installation of VueJS

There are various methods by which you can install Vue JS. Following are the two ways by which you can install the Vue JS for developing the web application based on it.

**Method 1 :** Simply download the `vue.js` file from the official web site <https://vuejs.org/v2/guide/installation.html>

Select the **Development version** for downloading the `vue.js`



Then in your web application you can simply refer to this file using script tag as follows -

```
<script type = "text/javascript" src = "vue.js"></script>
```

#### Method 2 :

For prototyping or learning purposes, you can use insert directly the following code in your javascript

```
<script src="https://cdn.jsdelivr.net/npm/vue@2.6.12/dist/vue.js"></script>
```

Similarly, you can also use -

```
<script src="https://unpkg.com/vue"></script>
```

### 5.12.2 Writing Web Application

In this section, we will discuss how to create a simple web application using VueJS script.

- Step 1 :** Download Vue.js file from the official web site and get it downloaded  
(Refer Method 1 of section 5.12.1) in the current working directly.

- Step 2 :** Write the following Vue JS script

```
<html>
<head>
</script>
<title>
    Learn Vue
</title>
<script src="https://unpkg.com/vue"></script>
</head>
<body>
    <div id = "myfirstapp">
        <h1>{{title}}</h1>

    </div>
    <script>
        const myvuecomponent = new Vue({
            el : "#myfirstapp",
            data : {
                title: "Demo Script for Vue JS"
            }
        })
    </script>
</body>
```

### Output



### Explanation

In above script

- 1) We have assigned an ID for div tag and that is "myfirstapp"
- 2) Then inside the <script> tag, created a vue component using Vue constructor.
- 3) The fields of Vue constructor are el, data and methods.

- 4) The **el** represents the element and to which the **div id** is assigned with preceding # symbol.
- 5) The **data** is a function which returns an object literal. Here we have defined a title inside the **data** function. It is then called using an interpolation i.e. `{}()` as shown below -

```
<h1>{{title}}</h1>
```

- 6) You can make any changes in the title string of present inside the **data** and those changes will be immediately reflected in the output displayed on the browser.

### 5.12.3 Basic Components of VueJS Script

- While writing the any Vue JS application, we need to create the **instance of Vue**, which is called as **root Vue Instance**.
- It is created as follows -

```
Var myVue = new Vue({  
  //options  
})
```

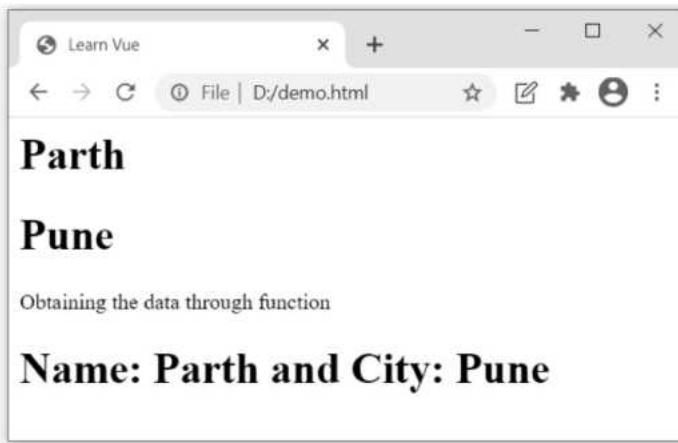
- For this Vue component, there is a parameter called **el**. It takes only the id of DOM element. This id must be preceded by # symbol.
- Then Vue component may contain parameters such as **data** and **methods**. The **data** defines the data values and **methods** define some function. Following are example scripts that illustrates these parameters -

**Example 5.12.1** Write VueJS script to display the name and city of the user

**Solution :**

```
<html>  
<head>  
</script>  
<title>  
  Learn Vue  
</title>  
<script src="https://unpkg.com/vue"></script>  
</head>  
<body>  
  <div id = "myfirstapp">  
    <h1>{{name}}</h1>  
    <h1>{{city}}</h1>  
    <p> Obtaining the data through function </p>  
    <h1>{{myfunction()}}</h1>  
  </div>
```

```
<script>
  const myvuecomponent = new Vue({
    el: "#myfirstapp",
    data: {
      name: "Parth",
      city: "Pune"
    },
    methods: {
      myfunction: function() {
        return "Name: "+this.name+" and City: "+this.city;
      }
    }
  })
</script>
</body>
</html>
```

**Output**

**Example 5.12.2** Write a VueJS script to welcome the user and display the current date and time.

**Solution :**

```
<html>
<head>
</script>
<title>
  Learn Vue
</title>
```

```
<script src="https://unpkg.com/vue"></script>
</head>
<body>
  <div id = "myfirstapp">
    <h1>{{title}}</h1>
    <p>{{paragraph}}</p>
  </div>
<script>
  console.log("Script Started!!!")

  const myvuecomponent = new Vue({
    el : "#myfirstapp",
    data : {
      title: "Demo Script for Vue JS",
      paragraph :"This page is created on: " + new Date().toLocaleString()
    }
  })
</script>
</body>
```

**Output****Review Question**

1. What is VueJS ? Enlist its features.

### 5.13 Multiple Choice Questions

Q.1 \_\_\_\_ is a program divided into subprograms and gets executed simultaneously.

- |                                            |                                              |
|--------------------------------------------|----------------------------------------------|
| <input type="checkbox"/> a Multiprocessing | <input type="checkbox"/> b Multithreading    |
| <input type="checkbox"/> c Multitasking    | <input type="checkbox"/> d None of the above |

Q.2 Which of the following method is used to start the execution of the thread ?

- |                                  |                                  |
|----------------------------------|----------------------------------|
| <input type="checkbox"/> a run   | <input type="checkbox"/> b start |
| <input type="checkbox"/> c begin | <input type="checkbox"/> d init  |

Q.3 In Java a thread can be created using :

- |                                                             |  |
|-------------------------------------------------------------|--|
| <input type="checkbox"/> a Only by extending thread class   |  |
| <input type="checkbox"/> b Only by using runnable interface |  |
| <input type="checkbox"/> c Both (a) and (b)                 |  |
| <input type="checkbox"/> d None of these                    |  |

Q.4 When a thread class is extended, \_\_\_\_ method is overridden to start the thread.

- |                                  |                                  |
|----------------------------------|----------------------------------|
| <input type="checkbox"/> a run   | <input type="checkbox"/> b start |
| <input type="checkbox"/> c begin | <input type="checkbox"/> d init  |

Q.5 Which of the keyword is used for the method to indicate that only one thread should execute the method at a time ?

- |                                         |                                  |
|-----------------------------------------|----------------------------------|
| <input type="checkbox"/> a static       | <input type="checkbox"/> b final |
| <input type="checkbox"/> c synchronized | <input type="checkbox"/> d const |

Q.6 Thread priority is \_\_\_\_\_.

- |                                  |                                 |
|----------------------------------|---------------------------------|
| <input type="checkbox"/> a long  | <input type="checkbox"/> b int  |
| <input type="checkbox"/> c float | <input type="checkbox"/> d char |

**Explanation :** The thread priorities are denoted by integer numbers.

Q.7 Choose the correct statement about thread.

- a Thread are subdivision of process.
- b One or more threads runs in the context of process.
- c Threads can execute any part of process. And same part of process can be executed by multiple threads.
- d All of the above

**Q.8 What is difference between starting thread with run() and start() method ?**

- a There is no difference
- b When you call start() method, main thread internally calls run() method to start newly created thread
- c run() calls start() method internally
- d None of these

**Q.9 How can thread go from waiting state to runnable state ?**

- a notify() is used
- b When sleep time is up
- c resume() method is called
- d All of the above

**Q.10 Javascript is a \_\_\_\_ language.**

- a application
- b scripting
- c programming
- d database

**Q.11 JavaScript is designed for following purpose -**

- a To Perform Server Side Scripting Opcion
- b To Execute Query Related to DB on Server
- c to Add Interactivity to HTML Pages.
- d To Style HTML Pages

**Q.12 The Javascript tag must be placed between \_\_\_\_.**

- a head
- b title and head
- c head and body
- d only body

**Q.13 Javascript is an \_\_\_\_ language.**

- a interpreted
- b compiled

**Q.14 Which was the first browser to support JavaScript ?**

- a IE
- b Mozilla Firefox
- c Netscape
- d Google Chrome

**Q.15 What is a controller in MVC ?**

- a It is a software code that stores the data.
- b It is a software code that renders the user interface.
- c It is a software code that controls the interactions between the model and view.
- d None of the above.

**Q.16 AngularJS expressions are written using \_\_\_\_\_.**

- a double braces like {{ expression }}
- b single braces like {expression}
- c small bracket like (expression)
- d capital bracket like [expression]

**Answer Keys for Multiple Choice Questions :**

Q.1	b	Q.2	b	Q.3	c	Q.4	a
Q.5	c	Q.6	b	Q.7	d	Q.8	c
Q.9	d	Q.10	b	Q.11	c	Q.12	c
Q.13	a	Q.14	c	Q.15	c	Q.16	a



## **UNIT - VI**

# **6**

# **Logical and Functional Programming**

### **Syllabus**

**Functional Programming Paradigm :** Understanding symbol manipulation, Basic LISP functions, definitions, predicates, conditionals and scoping, Recursion and iteration, Properties List array and access functions, Using lambda definitions, printing, reading and atom manipulation.

**Logic Programming Paradigm :** An Overview of Prolog, Syntax and Meaning of Prolog Programs, Lists, Operators, Arithmetic, Using Structures.

### **Contents**

- 6.1 Introduction to Functional Programming
- 6.2 Understanding Symbol Manipulation
- 6.3 Basic LISP Functions
- 6.4 Definitions, Predicates, Conditional and Scoping
- 6.5 Recursion and Iteration
- 6.6 Properties, A- Lists, Arrays and Access Function
- 6.7 Using Lambda Definitions
- 6.8 Printing, Reading and Atom Manipulation
- 6.9 An overview of Prolog
- 6.10 Syntax and Meaning of Prolog Programs
- 6.11 Lists, Operators and Arithmetic
- 6.12 List Structures
- 6.13 Comparison between Functional Programming and Logical Programming and Object Oriented Programming Languages Functional
- 6.14 Multiple Choice Questions

## Part I : Functional Programming Paradigm

### 6.1 Introduction to Functional Programming

- Functional languages are those languages in which the basic **building block is functions**.
- In functional programming the programmer is concerned only with functionality and not memory related variable storage and assignment sequence.

#### Features of Functional Programming Language

1. The functional programming languages are modelled on the concept of **mathematical functions** and make use of only conditional expressions and recursion for the computational purpose.
2. The programs are constructed by **building functional applications**. That means, the values produced by one or more functions become the parameters to another functions.
3. The **purest form** of functional programming languages use neither variables nor assignment statements, although this is relaxed somewhat in most applied functional languages.
4. The functional languages can be categorized in **two types** - **Pure functional languages** and **Impure functional languages**.
  - a. The **pure functional language** support only functional paradigm. Example of pure functional language is ML, Haskell.
  - b. The **impure functional language** can be used to write imperative style programs(For example C is a imperative style programming language). Example of impure functional programming language is LISP.

#### Commonly used Functional Programming Languages

LISP, Sisal, Haskell, ML, APL, Scheme are some commonly used functional languages.

#### Applications of Functional Programming Languages

1. The main domain of functional programming language is **Artificial Intelligence**. This language is used for building **expert systems, knowledge representation, machine learning, natural language processing, modeling speech and vision**.
2. Functional language is also used for building **mathematical software**.

The LISP statements can be executed on CLISP. CLISP is a portable ANSI Common Lisp implementation and development environment by Bruno Haible.

After installation of CLISP following window and > prompt will appear.

```
puntambekar@puntambekar-Inspiron-N4010: ~
puntambekar@puntambekar-Inspiron-N4010: $ clisp
i i i i i i    00000   0     00000000   00000   00000
I I I I I I I    8     8     8     8     0 8 8
I \ '+' / I     8     8     8     8     8 8 8
\ '-' / -' /    8     8     8     00000 80000
- - - - - - -    8     8     8     8     8 8 8
-----+----- 00000 8000000 0008000 00000 8

Welcome to GNU CLISP 2.49 (2010-07-07) <http://clisp.cons.org/>

Copyright (c) Bruno Haible, Michael Stoll 1992, 1993
Copyright (c) Bruno Haible, Marcus Daniels 1994-1997
Copyright (c) Bruno Haible, Pierpaolo Bernardi, Sam Steingold 1998
Copyright (c) Bruno Haible, Sam Steingold 1999-2000
Copyright (c) Sam Steingold, Bruno Haible 2001-2010

Type :h and hit Enter for context help.
```

In LISP, nearly **everything** is function. For example

(+ (\* 10 20) 30)

will result in

230

## Review Questions

1. What is functional programming? Enlist the features of it.
  2. Name the commonly used functional programming languages.
  3. State the applications of functional programming languages.

## 6.2 Understanding Symbol Manipulation

LISP was invented by John McCarthy in 1958. It is high level programming language. It was first implemented by Steve Russell on an IBM 704 computer.

The concept of symbol manipulation can be understood with the help of following features -

- **Symbol manipulation is like working with words and sentences :**
    - The group of characters represent the words, group of words represent the sentence. The group of sentences form paragraphs. Thus, by adding more and more grouping eventually, larger and larger structures are possible.

- In the similar manner, In Lisp, the fundamental unit used is atom. The group of atoms form a **list**. The group of many lists form **higher order lists**. Atoms and lists collectively called as **symbolic expressions**. Hence symbol manipulation is nothing but **list processing**.
- **Symbol Manipulation is needed to make computers Intelligent :**
  - Nearly all the intelligent programs are written in LISP.
  - As Lisp programs have an ability to perform calculus problems, geometric analogy problems, programs diagnosing the blood infections and many more.
  - Lisp is a language in which most research is done on knowledge base and expert systems.
  - Lisp also provides support for natural language processing, speech and vision recognition systems.
- **LISP is the right symbol-manipulation Language to Learn.**
  - LISP is oriented towards programming at terminal with quick response.
  - LISP can be better used with editing and debugging tools.
  - LISP is basically list processing language, hence it can be very easily used for symbol manipulation.
  - The function and data form of LISP is very much similar. Hence one LISP function can be used to analyze another.
- **LISP programs are easy to learn.**

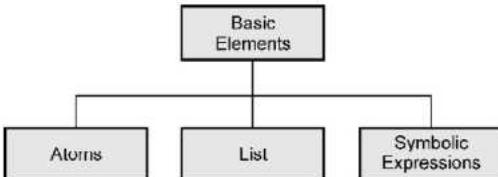
### Review Question

1. Explain the symbol manipulation aspect of LISP programming.

## 6.3 Basic LISP Functions

### 6.3.1 Building Blocks

There are three main building blocks of LISP -



**(1) Atoms**

All expressions in LISP are made up of lists. The list is a linear arrangement of objects separated by blanks and surrounded by parentheses. The objects are made up of either atoms. Space separated elements of a list are known as ATOMS.

Some examples of atoms are -

```
name
123
Abc
a
c d
```

**(2) Lists**

Lists are parenthesized collection of sublists or atoms. For example -

```
(a b (c d) e)
(10 20 30)
```

**(3) Symbolic Expressions**

Expressions can be created with the help of operators. There are various arithmetic operators such as +, -, \* and / used for manipulating the LISP. Following illustrates the use of operators

```
> (+ 2 3)
5
> (* 2 3)
6
> (- 5 3)
2
> (* (+ 2 3)(- 5 3))
10
>
```

Note that the evaluation of expression is based on prefix notation. Prefix notation is a notation in which Operator comes first and then two operands appear. For instance + 2 3 will be converted to 2+3 and result of operation will be 5

Similarly, we can use various comparison operators such as =, /=, <, > and so on.

Operator	Purpose	Example
<	Less than. It returns true if first operand is less than second argument.	> (< 10 20) T

>	Greater than. It returns true if first operand is greater than second argument.	> (> 20 10) T
=	If first operand is equal to second operand then it returns true.	> (= 10 10) T > (= 10 20) NIL
/=	If first operand is not equal to second operand then it returns T otherwise it returns NIL	> (/= 10 20) T > (/= 10 10) NIL
<=	Less than equal to.	> (<= 10 10) T
>=	Greater than equal to.	> (>= 20 10) T

### 6.3.2 Lisp Manipulation Function

The list can be represented as follows

```
>(10 20 30)
```

The list is enclosed within the brackets. The elements in the list are separated by space. The list containing no item is called the empty list. On entering the empty list it will return the value NIL.

```
> ()  
NIL
```

Using the word list we can also create a list.

```
> (list 10 20 30)  
(10 20 30)
```

There are various functions that can be used for manipulation of lists.

#### 1. Length :

The length of empty list is 0. We can find out the length of the list as follows -

```
> (length '(10 20 30 40 50))  
5
```

## 2. Append :

The append is used to add the element in the list at the end. For example

```
> (append '(10 20 30) '(40 50))  
(10 20 30 40 50)
```

We can also create a definition of append function as follows -

```
> (define (append x z)  
  (cond ((null? x) z)  
        (else (cons (car x) (append (cdr x) z)))) )
```

## 3. Map :

The map is basically applied to a function. Using map we can extend the function from elements to list. Map returns a list of the results of applying the function to elements taken from each list. For example

```
>(map abs '(1 -2 3 -4 -5))  
(1 2 3 4 5)  
> (map + '(1 2 3) (4 5 6))  
(5 7 9)
```

Thus map can handle more than one list.

### 6.3.3 Mathematical Functions

There are various mathematical functions that are illustrated as follows -

```
>(abs -10)  
10  
>(sqrt 25)  
5  
>(min 10 20)  
10  
>(max 10 20)  
20
```

### 6.3.4 Eval Function

The eval function helps in evaluating an expression. The evaluation function returns the value of the evaluation

```
>(eval (+ 10 20))  
30
```

### 6.3.5 Basic Lisp Primitives

#### (1) car,cdr and cons functions

- The car returns first element of a non empty list.

For example -

```
> (car '(1 2 3 4))  
1
```

- The **cdr** returns rest of the list after the first element is removed. It is pronounced as could-er.

```
> (cdr '(1 2 3 4))  
(2 3 4)
```

- We can combine two lists using **cons**. The **cons** function takes two arguments and returns a new cons cell containing the two values.

```
> (cons 10 '(20 30 40))  
(10 20 30 40)
```

## (2) Selectors

There are two primitives of list selectors in Lisp – **First** and **Rest**

- First takes a list as an argument and returns the first element of that list. It works like this :

```
>(first '(a e i o u))  
a  
>(first '((a e) f g))  
(a e)
```

- Rest takes a list as an argument and returns the list, minus its first element.

```
>(rest '(a e i o u))  
(e i o u).  
>(rest '((a e) i o u))  
(i o u)  
>(rest '((a b) (c d)))  
((c d))
```

## (3) The **setf** function

**setf** changes the value of place to be new value. The syntax is

```
(setf place1 newvalue1  
      place2 newvalue2)  
...  
      placen newvaluen)
```

For example –

```
(setf foo '(1 2 3)) ;foo => (1 2 3)  
(1 2 3)  
  
foo          ;foo => (1 2 3) as defined above  
(1 2 3)
```

```
(car foo)      ;the first item in foo is 1
1

(setf (car foo) 4) ;set or setq will fail since (car foo) is not a symbol
4

foo          ;the fist item in foo was set to 4 by setf
(4 2 3)
```

**Review Question**

1. Explain the mathematical functions and eval function used in LISP
2. What are different list manipulation functions used in LISP?
3. Explain various operators used in LISP with the help of example.
4. Illustrate the use of car, cdr and cons functions used LISP

**6.4 Definitions, Predicates, Conditional and Scoping****6.4.1 Definitions**

The operator **defun** is used to define the procedure. The syntax of procedure writing in LISP is

```
defun (proc_name(argument1 argument2)
           body of procedure)
```

**For example** - procedure for addition of two numbers is as follows -

```
> (defun mysum(a b)
(+ a b))
Now we can use this procedure the way we want.
> (mysum 2 3)
5
```

**Example 6.4.1** Write a procedure in LISP to calculate cube of number.

**Solution :**

<pre>&gt;(defun cube(a)   (* (* a a) a)) &gt; (cube 5) 125</pre>	<b>Output</b>
------------------------------------------------------------------	---------------

**Example 6.4.2** Write a procedure to display sum of squares.

**Solution :**

```
> (defun square(x)
  (* x x))
> (defun sum-of-square(start end)
  (if(> start end)
  0
  (+ (square start) (sum-of-square(+ 1 start) end)))
)
)
> (sum-of-square 1 5)
55
```

**Output**

### 6.4.2 Predicates

Predicate is a procedure that returns a value that indicates true or false. The false value is indicated by **NIL** and the true value is indicated by **T**. Thus NIL and T are the special symbols.

Name of the predicates usually ends with p. For example – evenp, oddp, zerop and so on.

Following are different types of predicates

#### (1) Equality Predicates

**Equal** : It takes two arguments and if two arguments are equal then it returns T otherwise it returns NIL.

```
> (eq 'g 'h)
NIL
> (eq 5 5)
T
```

**Eq** : It specifies that the items involved are the same symbol

For example –

```
(setf x 'happy)
(setf y 'happy)
(eql x y) = t
```

#### (2) Data Type Predicates

These predicates test for individual data type.

For example -

(LISTP S) tests if S is a list

### (3) Empty List Predicates

**null** : It takes one argument and returns T if the argument evaluates to nil, otherwise it returns NIL

```
>(null (NIL))
T
```

### (4) List-Membership Predicates

**>member( item list)** The value is nil if the item is not in the list, else it's the sublist starting from the first occurrence of the item;

For example - The value of (member 'c '(a b c d e)) is (c d e)

### (5) Number Predicates

**Evenp** : It takes one numeric argument. If the number is even then it returns T otherwise it returns NIL

```
>(evenp 10)
T
>(evenp 11)
NIL
```

**Oddp** : It takes one numeric argument. If the number is odd then it returns T otherwise it returns NIL

```
>(oddp 10)
NIL
>(oddp 11)
T
```

**numberp** : It takes one argument and returns true if the argument is a number otherwise returns NIL.

```
>(numberp 10)
T
>(numberp 'a)
NIL
```

### 6.4.3 Conditional Statements

- The conditions can be specified using if. The syntax of writing if is  
(if(**conditional expression**) (**true-expression**) (**false expression**))

The **true-expression** part will be executed only if the condition specified in **condition expression** is true, otherwise the **false expression** part will be executed. For example

```
>(if(> 10 20) 10
NIL
```

As 10>20 is false and there is no false expression, NIL will be returned.

We will change the above expression as follows -

```
> (if (> 10 20) 10 20)
20
```

- We can have **nested if statement**. The example is as shown below -

```
> (if(= 10 10) (if(> 5 1) 5 1))
5
```

- If we want to display a series of expression on the if condition then these expressions are represented by a block. For this block **progn** is used. **progn** can take any number of expressions, and evaluates each of its expressions in order. **progn** then returns the value of the last expression.

```
> (if (> 20 10)
```

```
(progn
  (print "Hello" )
  (print "Welcome" ))
  (print "Bye" )
```

**True expression multiple statements are executed for true expression.**

**False expression.**

```
)
```

#### Output

```
"Hello"
"Welcome"
"Welcome"
```

- The **dotimes** is used for repeatedly executing the statements. The **dotimes** needs a variable. The variable value gets incremented to reach to a value specified by **high-value**. The **dotimes** also requires **optional-return-value** which will be displayed only when the variable reaches to **high-value**.

```
> (dotimes (a 5 "Bye") (print "Hello"))
>Hello"
>Hello"
>Hello"
>Hello"
>Hello"
Bye"
```

- Similar to **if** the **cond** is used to specify the conditional statements. Following example is an illustrative one.

```
> (cond
  ((> 20 10) (print "Hi"))
  ((< 20 10) (print "Hello"))
  )
```

**Output**

```
"Hi"
"Hi"
```

**Example 6.4.3** Write a procedure to display sum of squares.

**Solution :**

```
> (defun square(x)
  (* x x))
> (defun sum-of-square(start end)
  (if(> start end)
    0
    (+ (square start) (sum-of-square(+ 1 start) end)))
  )
  )
```

**Output**

```
> (sum-of-square 1 5)
55
```

#### 6.4.4 Scoping

##### Concept of Bound and Free variables

A variable x is **bound** in an expression E if the meaning of E **remains unchanged** by uniform replacement of a variable x by y for every occurrence of x in E.

For example –

```
(lambda (x) (*x 2))
(lambda (y) (*y 2))
```

In above expression variable x is bound

A variable x is **free** in an expression E if the meaning of E **gets changed** by uniform replacement of a variable x by y for every occurrence of x in E.

For example –

```
(lambda (x) (*x y))
(lambda (y) (*x z))
```

Here x is still bound but y is a free variable.

#### Review Questions

1. How will you define a function in LISP ? Explain with suitable example.
2. Explain any three commonly used predicates in LISP.
3. Explain various conditional statements used in LISP.
4. Explain the concept of Bound and free variables.

## 6.5 Recursion and Iteration

- **Recursive function** is a function which is called by itself for more than once.
- A function is recursive if it calls itself –
  - Boundary condition: not recursive and
  - Recursive condition: must be a smaller problem to converge
- For example –

```
(defun power (x y)
  (if (= y 0) 1
    (* x (power x (- y 1)))))
```

```
>power (3 4)
81
```

- In LISP, the **iterative functions** can be written using do, dotimes, dolists. The syntax of dotimes is

```
(dotimes (<counter> <limit> <result>) <body>)
```

For example

```
(defun power_It (x y)
  (let ((result 1))
    (dotimes (count y result)
      (setf result (* x result)))))
```

```
>power_It (3 4)
81
```

## 6.6 Properties, A- Lists, Arrays and Access Function

### Properties

- Lisp allows to assign properties to symbols. For example – For a Student object we can assign properties like name, class, marks and so on.
- Properties are created by using **get** within a **setf** form.
- For example –

```
> (setf (get 'student 'name) '(Ankita))
(ANKITA)
> (setf (get 'student 'rollNo) '(101))
(101)
> (setf (get 'student 'marks) '(98))
(98)
> (write (get 'student 'name))
```

```
(ANKITA)(ANKITA)
> (write (get 'student 'rollNo))
(101)(101)
```

- We can use **symbol-plist** function that allows to see all the properties of symbol. For example -

```
> (write (symbol-plist 'student))
(MARKS (98) ROLLNO (101) NAME (ANKITA))
```

### Association Lists(A-Lists)

An **association list** or **a-list** for short, records a mapping from keys to values. Using **assoc** we can obtain the most recent binding of key to value. For example - following assoc returns the appropriate values for the specified keys.

```
>(assoc 'one '((one 1) (two 2) (one 3)))
(ONE 1)
Another example
(assoc 'age '((name aa) (age 30) (salary 10000)))
(AGE 30)
```

### Arrays

- An array is a sequence of contiguous memory locations in which we can store objects of any kind.
- In LISP using **ARRAY** function we can write

```
(ARRAY MyTable 20 20)
MyTable
```

This states that MyTable is an array that has two indices from 0 to 19 each.

- We can store a value at specific location in the array using STORE function. For example -

```
(STORE (MyTable 12 15) 199)
199
```

### Access Function

- Access functions are the functions designed to retrieve and change the data.
- It is built using the LISP primitives like GET, ASSOC, CAR and CDR
- Access functions make the programs more transparent. Access functions are normally easier to find and understand.

**Review Questions**

1. Illustrate the use of `setf` and `get` in LISP.
2. What is the use of association Lists in LISP ? Explain.

**6.7 Using Lambda Definitions**

The lambda function is used to define a function which returns some value. The syntax of this function is

`(lambda (parameters) Body)`

For example -

```
>((lambda (x y) (* x y)) 10 20)
200
```

Similarly we can assign a value to a function by the return value of some function.

```
>(define (add n) (lambda (x) (+ x y)))
add
>((add 10) 20)
30
```

Here we have defined the function `add` which returns an object of type function. In order to use the special operator function and apply it to a *lambda expression*. It may be abbreviated as #form. In our example above we used a shorthand notation provided by the macro LAMBDA. We will write it as

```
> (defun add (y)
  #'(lambda (x) (+ x y)))
ADD
```

Now if we try the command

```
>(add 10)
```

It will return the Lambda definition

**Review Question**

1. Write a short note on - *Lambda definitions in LISP*.

**6.8 Printing, Reading and Atom Manipulation**

- **Printing and Reading:** The `print`, `write` are the output functions which are used to display the values or text on the console. Whereas `read` function is for reading the values from the input streams. Following program shows the use of **printing and reading** functions.

- **Example**

```
defun MyFun()
```

```
(print "Enter some number:")
(setq a(read))
(print "You have entered...")
(write a))
```

When above code is executed then we get following output

```
Enter some number: 111
```

```
You have entered...111
```

- **Seq Construct :** We can directly set the values to the symbol using **setq** function.

For example – following statement assigns value 123 to variable x

```
>(setq x 123)
```

- **Let Construct :** The syntax of Let construct is as follows -

```
(let ((var1 val1) (var2 val2).. (varn valn)), last_val )
```

Where var1, var2, ..varn are variable names and val1, val2, .. valn are the initial values assigned to the respective variables.

When let is executed, each variable is assigned the respective value and lastly the s-expression is evaluated. The value of the last expression evaluated is returned.

```
(let (x = 2, y = 3, z = x + y) print(x + y + z))
```

This will return 10

### Review Question

1. Explain the use of printing and reading in LISP.

## Part II : Logic Programming Paradigm

### 6.9 An overview of Prolog

- Logic programming is a programming paradigm in which the set of sentences are written in **logical form**. The logic programs consists of **facts and rules** about some problem domain.
- Logic programming can be viewed as **controlled deduction**.
- An important concept in logic programming is the separation of programs into their logic component and their control component.
- Kowalski illustrates the logic programming by following equation

$$\text{Algorithm} = \text{Logic} + \text{Control}$$

- where "Logic" represents a logic program in the form of facts and rules. The "Control" represents how the algorithm can be implemented by applying the rules in particular order.
- The concept of logic programming is linked up with a language called prolog. Prolog is acronym of PROgramming in LOGic.

## How to Program ?

In this section we will get introduced with some basic features of logic programming using prolog. The programs in prolog can be implemented and tested using SWI-Prolog systems. We can get the interpreter for the prolog from [www.swi-prolog.org/download/stable](http://www.swi-prolog.org/download/stable). Choose the appropriate binaries depending upon your operating system version.

The interpreter window appears on which **? is a prompt**. We can type and test various simple prolog statements on this prompt window. Here are some sample examples of execution and testing of SWI-Prolog.

```

SWI-Prolog (Multi-threaded, version 7.2.2)
File Edit Settings Run Debug Help
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic), or ?- apropos(Word).

1 ?- 10>5.
true.

2 ?- 2+3>11.
false.

3 ?- write('Enjoy Prolog!!').
Enjoy Prolog!!
true.

4 ?-

```

Note that every statement in prolog must be terminated by dot.

## Features of Prolog

- 1) Prolog is used in **artificial intelligence and expert system** to solve complex problems.
- 2) It is used on **pattern matching algorithm**.
- 3) It is used to develop systems based on **object oriented concepts**.

- 4) Prolog is used to build **decision support systems** that aid organizations in decision-making. For example- decision systems for medical diagnoses.

## 6.10 Syntax and Meaning of Prolog Programs

### Relations

- Prolog makes use of relations instead of using **functions**. Relations are more flexible and treats the arguments and results uniformly.
- For example - Consider following list

[a,b,c]

- As we know that the list can be considered as [H|T]. That means in the list H i.e. head occurs first and then comes T i.e. tail. Hence prolog will treat [a,b,c] as

[a,b,c] = [a | [b,c]].

### Queries

- In prolog, the simple queries can be created to check whether particular tuple belongs to a relation.
- In below given execution, the first two queries represent the simple H and T relation between the members of the list. Hence the answer turn out to be true or yes.

The screenshot shows the SWI-Prolog interface with the title bar "SWI-Prolog (Multi-threaded version 7.2.2)". The menu bar includes File, Edit, Setting, Run, Debug, and Help. The main window displays the following interactions:

```

3 ? - [a,b,c] = [a | [b,c]]
True

4 ? - [a,b,c] = [a,b | [c]]
True

5 ? - [a,b,c] = [a,b,c | [d]]
False

6 ?

```

- But in the third query since d is not the member of the list [a,b,c] the answer turn out to be false.

### Terms, Goals, Facts and Rules

- A Prolog program consists of a number of clauses. Each clause is either a fact or a rule. After a Prolog program is loaded (or consulted) in a Prolog interpreter, users

can submit goals or queries, and the Prolog interpreter will give results (answers) according to the facts and rules.

### (1) Terms :

- Prolog term is a constant, a variable, or a structure. A constant is either an atom or an integer. Atoms are the symbolic values of Prolog.
- A variable is any string of letters, digits, and underscores that begins with an uppercase letter or an underscore ( \_ ). Variables are not bound to types by declarations. The binding of a value, and thus a type, to a variable is called an instantiation.

### (2) Goals :

- A goal is a statement starting with a predicate and probably followed by its arguments. In a valid goal, the predicate must have appeared in at least one fact or rule in the consulted program, and the number of arguments in the goal must be the same as that appears in the consulted program. Also, all the arguments (if any) are constants.
- The purpose of submitting a goal is to find out whether the statement represented by the goal is true according to the knowledge database (i.e. the facts and rules in the consulted program). This is similar to proving a hypothesis - the goal being the hypothesis, the facts being the axioms and the rules being the theorems.

### (3) Fact

- A fact must start with a predicate (which is an atom) and end with a fullstop. The predicate may be followed by one or more arguments which are enclosed by parentheses. The arguments can be constants, numbers, variables or lists. Arguments are separated by commas.
- The syntax of fact is

```
pred(arg1, arg2, ... argN).
```

Where

pred is the name of the predicate and arg1, ...argN are the arguments

### For example -

```
man(anand).
```

```
man(arun).
```

```
woman(anuradha).
```

```
woman(jayashree).
```

```
parent(anand,parth). % means anand is parent of parth
Parent(anuradha,parth).
parent(arun,anuradha).
parent(jayashree,anuradha).
```

The result as either true or false depending upon the facts. It is as follows -

The screenshot shows the SWI-Prolog interface with the title bar "SWI-Prolog (Multi-threaded version 7.2.2)". The menu bar includes File, Edit, Setting, Run, Debug, and Help. The main window displays the following query results:

```
1 ? - parent(anand,parth),
True
2 ? - parent(arun,anuradha)
True
3 ? - parent(arun,anand),
False
4 ?
```

#### (4) Rule :

- A rule can be viewed as an extension of a fact with added conditions that also have to be satisfied for it to be true. It consists of two parts. A rule is no more than a stored query. Its syntax is

head :- body.

where

head is a predicate definition (just like a fact),

:- is the neck symbol, sometimes read as "if"

body is one or more goals (a query)

#### For example -

```
father(F,C):-man(F),parent(F,C).
```

#### Example using Fact and Rule

```
/* Facts */
```

```
man(anand).
man(arun).
woman(anuradha).
```

```
woman(jayashree).

parent(anand,parth). % means anand is parent of parth
parent(anuradha,parth).
parent(arun,anuradha).
parent(jayashree,anuradha).
```

```
/* A general rule */
father(F,C):-man(F),parent(F,C).
mother(M,C):-woman(M),parent(M,C).
If we query
?- father(X,parth).
```

The above query can be read who is father of parth? The answer is X=anand is a father of parth.

### 6.10.1 Response to Query

Control represents how a language **computes a response to a query**. The control execution is based on two rules -

- Goal Order** : That means choose the **leftmost subgoal**
- Rule Order** : That means select the **first applicable rule**.

A **goal** is a statement starting with a **predicate** and probably followed by its **arguments**. In a valid goal, the predicate must have appeared in at least one fact or rule in the consulted program, and the number of arguments in the goal must be the same as that appears in the consulted program. Also, all the arguments (if any) are constants.

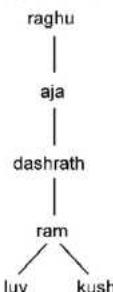
The purpose of submitting a goal is to find out whether the statement represented by the goal is true according to the knowledge database (i.e. the facts and rules in the consulted program). Let us understand how control in prolog works with the help of simple example. Following is a **family.pl** file in which the rules and facts are written.

```
/*Rules*/
gfather(X,Y):- father(X,Z),parent(Z,Y).
parent(X,Y):- father(X,Y).

/*Facts*/

father(dashrath,ram). %means dashrath is father of ram
father(raghu,aja).
father(aja,dashrath).
father(ram,luv).
father(ram,kush).
```

The tree for the above facts can be as given below -



**Fig. 6.10.1**

The goals can be executed on the prompt window as follows -

```

SWI-Prolog (Multi-threaded, version 7.2.2)
File Edit Settings Run Debug Help

1 ?- gfather(aja,ram).
true.

2 ?- parent(X,Y).
X = dashrath,
Y = ram .

3 ?- father(X,Y), father(Y,ram).
X = aja,
Y = dashrath
  
```

Now consider the goal **gfather(aja,ram)**.

As we know there are two rules for execution of control -

- 1. Goal Order :** That means choose the leftmost subgoal
- 2. Rule Order :** That means select the first applicable rule.

Here the rules are

```

gfather(X,Y):- father(X,Z),parent(Z,Y).
parent(X,Y):- father(X,Y).
  
```

**Step 1 :** For **gfather(aja,ram)** we apply the first applicable rule(**Rule ordering**)

```
gfather(X,Y):- father(X,Z),parent(Z,Y)
```

Here the leftmost subgoal i.e. **father (X,Z)** is chosen first. (**Goal ordering**)

**Step 2 :** The match for **father(X,Z)** is searched in the fact list with **X=aja**. We will find the match with **father(aja,dashrath)**. Hence value of **Z=dashrath**.

Hence the first rule now becomes

```
gfather(aja,ram) = father(aja,Z),parent(Z,Y). With Z=dashrath and Y=ram
```

**Step 3 :** For the subgoal  $\text{parent}(Z,Y)$  select the rule

```
parent(X,Y):- father(X,Y).
```

with  $X=Z=\text{dashrath}$  and  $Y=\text{ram}$ . The parent will return  $\text{father}(X,Y)=\text{father}(\text{dashrath},\text{ram})$  which comes out to be true. Hence we get the answer as **true**.

While executing the control the prolog choose a rule by

- searching sequentially in the program from top to bottom whose head matches with the goal with possible unifier.
- Remember the position of the matched rule.
- Join the rule body in front of the sequence of sub goals to be solved.
- Repeat until either goal is satisfied or is failed.

### Another example

Consider another example for goal ordering and rule ordering. Consider following prolog database

#### Rule order affects the Search for Solutions

##### Step 1 :

family1.pl

```
/*Rules*/
descend(X,Y):- child(X,Y).
descend(X,Y):- child(X,Z),descend(Z,Y).

/*Facts*/
child(luv,ram).
child(ram,dashrath).
child(dashrath,aja).
child(aja,raghu).
```

Now try out following queries



**Step 2 :** Now just change the ordering of the rules as follows -

### family2.pl

```
/*Rules*/
descend(X,Y):- child(X,Z),descend(Z,Y).
descend(X,Y):- child(X,Y).

/*Facts*/

child(luv,ram).
child(ram,dashrath).
child(dashrath,aja).
child(aja,raghu).
```

We will get following result on execution of descend query

The screenshot shows the SWI-Prolog interface. The title bar says "SWI-Prolog (Multi-threaded, version 7.2.2)". The menu bar includes File, Edit, Settings, Run, Debug, Help. Below the menu is a help message: "For help, use ?- help(Topic), or ?- apropos(Word)". The main window contains the query "1 ?- descend(luv,X)." and the response "X = raghu".

Thus due to change in ordering of the rules we get different results.

### Goal Order Changes the Solutions

**Step 3 :** Now just change the ordering of the goals by keeping the ordering of the rules as it is. We will create another version of the above program by changing the order of goal.

### family3.pl

```
/*Rules*/
descend(X,Y):- descend(Z,Y),child(X,Z).
descend(X,Y):- child(X,Y).

/*Facts*/

child(luv,ram).
child(ram,dashrath).
child(dashrath,aja).
child(aja,raghu).
```

We will get following result on execution of descend query

The screenshot shows a window titled "SWI-Prolog (Multi-threaded, version 7.2.2)". The menu bar includes File, Edit, Settings, Run, Debug, Help. Below the menu is a message: "For help, use ?- help(Topic). or ?- apropos(Word)." A command line window shows the input "1 ?- descend(luv,X)." followed by the error message "ERROR: toplevel: Undefined procedure: descend2 (DWIM could not correct goal)." The prompt "?- ." is visible at the bottom.

you will get an error message ("out of local stack", or something similar). Prolog is looping. Why? Well, in order to satisfy the query `descend(luv,X)` Prolog uses the first rule. This means that its next goal will be to satisfy the query

`descend(luv,X1)`

for some new variable `X1`. But to satisfy this new goal, Prolog again has to use the first rule, and this means that its next goal is going to be

`descend(luv,X2)`

for some new variable `X2`. And of course, this in turn means that its next goal is going to be `descend(luv,X3)` and then `descend(luv,X4)`, and so on. Thus change in the goal order has resulted in procedural disaster.

## Standard Terminology

The standard terminology is that in a rule the leftmost item of the body must be identical with the rule's head.

**Step 4 :** To overcome the above error causing situation just change the rule ordering as follows

```
/*Rules*/
descend(X,Y):- child(X,Y).
descend(X,Y):- descend(Z,Y),child(X,Z).

/*Facts*/
child(luv,ram).
child(ram,dashrath).
child(dashrath,aja).
child(aja,raghu).
```

Now the output will be

The screenshot shows a window titled "SWI-Prolog (Multi-threaded, version 7.2.2)". The menu bar includes File, Edit, Settings, Run, Debug, and Help. The main area contains the query "2 ?- descend(luv,X)." followed by the response "X = ram".

### Review Questions

1. Explain the goal ordering and rule ordering.
2. Give the example explaining how goal ordering changes the solution in prolog.
3. Explain the basic elements of Prolog

## 6.11 Lists, Operators and Arithmetic

### Lists

- A list is an ordered sequence of objects and lists. In Prolog, a list is written as its elements separated by commas and enclosed in brackets. For example:  
[] ← represents empty list  
[a,b,c,[d]].
- In Prolog list elements are enclosed by brackets and separated by commas.  
[1,2,3,4]

### Operator

Predicate calculus is a fundamental notation for representing and reasoning with logical statements. It extends propositional calculus by introducing the quantifiers, and by allowing predicates and functions of any number of variables.

- 1) **Propositional symbols** : P, Q, R, S, T, ... denote propositions, or statements about the world that maybe either true or false, such as "Sun is bright" or "India is a country".
- 2) **Truth symbols** : true, false.
- 3) **Connectives** :  $\wedge$  (and),  $\vee$  (or),  $\neg$  (not),  $\Rightarrow$ ,  $=$
- 4) **Propositional calculus sentences** : Every propositional symbol and truth symbol is a sentence.

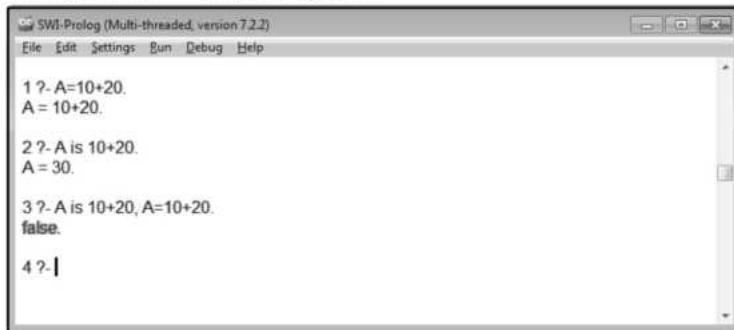
For example The negation of a sentence is a sentence. Denoted as  $\neg P$

Any two sentences connected by one of  $\{\wedge, \vee, \Rightarrow, =\}$  are a sentence. For instance  $P \vee \neg Q \Rightarrow R$ .

- 5) **Conjuncts and disjuncts** : In expressions of the form  $P \wedge Q$ , P and Q are called conjuncts.  
 In  $P \vee Q$ , P and Q are called disjuncts.
- 6) **Symbols** : The symbols ( ) and [ ] are used to group symbols into sub-expressions and so control their order of evaluation and meaning.  $(P \vee \neg Q) = R$  and  $P \vee (\neg Q = R)$  are different.

### Arithmetic

The operator = is used for unification in Prolog. Following example illustrates the use of = for binding with the corresponding value.



```

SWI-Prolog (Multi-threaded, version 7.2.2)
File Edit Settings Run Debug Help

1 ?- A=10+20.
A = 10+20.

2 ?- A is 10+20.
A = 30.

3 ?- A is 10+20, A=10+20.
false.

4 ?- |

```

In above code, variable A is bound to 10+20.

The infix operator is evaluates the expression hence we get A=30 for the second query statement.

In the third query statement, 10+20 does not unify with 30 Hence we get **false** as a result of third query.

### 6.12 List Structures

A list is an ordered sequence of objects and lists. In Prolog, a list is written as its elements separated by commas and enclosed in brackets. For example :

[ ] ← represents empty list  
 [a,b,c,[d]].

The list contains Head and Tail part. For example - if we type following query

?- [H|T] = [10,20,30,40].

We will get

**H = 10,**

**T = [20, 30, 40].**

Now consider following execution of the list

The screenshot shows the SWI-Prolog IDE interface. The title bar reads "SWI-Prolog (Multi-threaded, version 7.2.2)". The menu bar includes File, Edit, Settings, Run, Debug, and Help. The main window displays the following interaction:

```

3 ?- [a,b,c]=[X,Y,Z | T].
X = a,
Y = b,
Z = c,
T = [].

4 ?-

```

Now if we execute

```
?- [a,b,c] = [a|[b|[c|[]]]].
```

Then we will get -

true

**Example 6.12.1** Consider following PROLOG facts

*head([H | T], H).*

*tail([H | T], T).*

*list([H | T], H, T).*

What will be

i) *head([], X).*

ii) *list([], X, Y).*

iii) *head([a, b, c, d], X).*

iv) *tail([a], X).*

v) *list([a, b], X, Y).*

**Solution :**

i) no

The fact for head returns the head of the list and we have passed an empty list. The empty list has neither head, nor tail hence we get the answer as no or false.

ii) no

The fact for list returns head in X and tail in Y. With the empty list we will get no head and no tail. Hence is the answer.

iii) X = a

The fact for head returns head of the list. As a is the head for the list [a, b, c, d], we get the answer a.

- iv)  $X = []$   
 v)  $X = a$   
 $Y = [b]$

**Review Question**

1. Explain the list manipulation in prolog.

### 6.13 Comparison between Functional Programming and Logical Programming and Object Oriented Programming Languages Functional

#### 1. Programming Vs. Logical Programming

Functional Programming	Logical Programming
Functional programming uses functions.	Logical programming uses predicates.
The functions can return a value.	Predicate cannot return a value. The predicate can be true or false.
In functional programming, the variable denotes the concrete value.	In logic programming, a variable does not need to refer to concrete value.
Functional programming is normally used for modelling reasoning.	Logical programming is normally used for modelling knowledge.
Examples – ML, HASSELL are functional programming languages.	Example – PROLOG is a logical programming language.

#### 2. OOP Vs. Logical Programming Vs. Functional Programming

Object Oriented Programming	Logical Programming	Functional Programming
It is based on objects and classes.	It is based on Predicates	It is based on evaluation of functions.
It is mainly used for enterprise level applications.	It is used for modelling knowledge.	It is mainly used for mathematical computations.
Program execution is by means of exchange of messages between objects.	Program execution is by means of proving the logic.	Program execution is by means of evaluation of functions.
Final result is denoted by state of objects.	Final result is denoted by failure or success of logic.	Final result is denoted by value of the main function.
Examples - Java, C++	Examples - Prolog	Example - LISP, ML, Haskell

**6.14 Multiple Choice Questions**

**Q.1 Which of the following is a famous functional programming language ?**

- |                                |                                    |
|--------------------------------|------------------------------------|
| <input type="checkbox"/> a C++ | <input type="checkbox"/> b Java    |
| <input type="checkbox"/> c C#  | <input type="checkbox"/> d Haskell |

**Q.2 Which of the following is a famous functional programming language ?**

- |                                |                                 |
|--------------------------------|---------------------------------|
| <input type="checkbox"/> a C++ | <input type="checkbox"/> b Java |
| <input type="checkbox"/> c C#  | <input type="checkbox"/> d LISP |

**Q.3 Arguments are \_\_\_\_ a part of LISP Syntax.**

- |                                      |                                              |
|--------------------------------------|----------------------------------------------|
| <input type="checkbox"/> a sometimes | <input type="checkbox"/> b always            |
| <input type="checkbox"/> c never     | <input type="checkbox"/> d not considered as |

**Q.4 What is the output of the following LISP statement ? (+ 2 10)**

- |                               |                                  |
|-------------------------------|----------------------------------|
| <input type="checkbox"/> a 3  | <input type="checkbox"/> b 12    |
| <input type="checkbox"/> c 20 | <input type="checkbox"/> d Error |

**Q.5 \_\_\_\_ is the notation used for writing LISP Syntax.**

- |                                    |                                          |
|------------------------------------|------------------------------------------|
| <input type="checkbox"/> a Infix   | <input type="checkbox"/> b Prefix        |
| <input type="checkbox"/> c Postfix | <input type="checkbox"/> d None of above |

**Q.6 What does Lisp stands for ?**

- |                                            |                                             |
|--------------------------------------------|---------------------------------------------|
| <input type="checkbox"/> a Like Processing | <input type="checkbox"/> b Light Processing |
| <input type="checkbox"/> c List Processing | <input type="checkbox"/> d None of above    |

**Q.7 What is called the symbol manipulation in LISP ?**

- |                                            |                                          |
|--------------------------------------------|------------------------------------------|
| <input type="checkbox"/> a Lists           | <input type="checkbox"/> b Atoms         |
| <input type="checkbox"/> c List processing | <input type="checkbox"/> d None of above |

**Q.8 A prolog query can be made up of only two subgoals.**

- |                                 |                                  |
|---------------------------------|----------------------------------|
| <input type="checkbox"/> a TRUE | <input type="checkbox"/> b FALSE |
|---------------------------------|----------------------------------|

**Q.9 Which of the following language is a declarative language ?**

- |                                   |                                  |
|-----------------------------------|----------------------------------|
| <input type="checkbox"/> a C#     | <input type="checkbox"/> b Algol |
| <input type="checkbox"/> c Prolog | <input type="checkbox"/> d Java  |

**Q.10 What Are The Features Of Prolog Language ?**

- |                                                     |                                             |
|-----------------------------------------------------|---------------------------------------------|
| <input type="checkbox"/> a Intelligent Systems      | <input type="checkbox"/> b Expert Systems   |
| <input type="checkbox"/> c Natural Language Systems | <input type="checkbox"/> d all of the above |

**Answer Keys for Multiple Choice Questions :**

Q.1	d	Q.2	d	Q.3	b	Q.4	b
Q.5	b	Q.6	c	Q.7	c	Q.8	b
Q.9	c	Q.10	d				



# SOLVED MODEL QUESTION PAPER (In Sem)

## Principles of Programming Languages

S.E. (Computer) Semester - IV (As Per 2019 Pattern)

Time : 1 Hour]

[Maximum Marks : 30

N. B. :

- i) Attempt Q.1 or Q.2, Q.3 or Q.4.
- ii) Neat diagrams must be drawn wherever necessary.
- iii) Figures to the right side indicate full marks.
- iv) Assume suitable data, if necessary.

- Q.1** a) Explain the importance of studying programming language. (Refer section 1.1) [4]  
b) Explain the concept of environment framework in programming environment.  
(Refer section 1.4) [3]  
c) What is programming paradigms ? Explain the imperative programming paradigms along with its merits and demerits. (Refer section 1.6) [8]

OR

- Q.2** a) Explain various classes of binding times. (Refer section 1.5) [6]  
b) Explain the concept of virtual computers. (Refer section 1.5.4) [4]  
c) Explain the effect of programming environment on language design in brief.  
(Refer section 1.4) [5]

- Q.3** a) Explain various types of string operations. (Refer section 2.1.2.2) [3]  
b) Explain the concept of regular and jagged array with examples.  
(Refer section 2.1.4.7) [4]  
c) Explain in detail different parameter passing methods. (Refer section 2.4.4) [8]

OR

- Q.4** a) What is unconditional branching ? Also discuss the problems associated with unconditional branching. (Refer section 2.3.3) [5]  
b) Explain the term short circuit evaluation. (Refer section 2.2.5) [3]

- c) How to implement parameterized ADT in C++ ? Explain it with any suitable example. (Refer section 2.5.2) [7]

## SOLVED MODEL QUESTION PAPER (End Sem)

# Principles of Programming Languages

S.E. (Computer) Semester - IV (As Per 2019 Pattern)

Time :  $2\frac{1}{2}$  Hours

[Maximum Marks : 70]

N. B. :

- Attempt Q.1 or Q.2, Q.3 or Q.4, Q.5 or Q.6, Q.7 or Q.8.
- Neat diagrams must be drawn wherever necessary.
- Figures to the right side indicate full marks.
- Assume suitable data, if necessary.

- Q.1** a) Explain Java's role in Internet. Justify the following features of Java  
1. Secure    2. Architecture neutral    3. Distributed (Refer section 3.1) [8]
- b) Explain jump statements used in Java. (Refer section 3.11) [10]

OR

- Q.2** a) What is use of constructors ? What are types of constructors in Java ? Give example of each type. (Refer section 3.17) [10]
- b) Write a program in Java to perform the addition of two matrices (multidimensional arrays) and set the diagonal elements of resultant matrix to 0.  
(Refer example 3.12.1) [8]

- Q.3** a) What is inheritance ? What are advantages of using inheritance ? Show by example the simple inheritance in Java. (Refer section 4.3) [5]
- b) Elaborate the significance of keyword super in Java ? Demonstrate with example each of the case. (Refer section 4.6) [12]

OR

- Q.4** a) Differentiate between method overloading and method overriding.  
(Refer section 4.10) [3]

b) What is inheritance ? How multilevel inheritance is achieved in Java ? Explain with suitable example. (Refer section 4.8) [8]

c) Write a program that copies the content of one file to another by removing unnecessary spaces between words. (Refer example 4.42.1) [6]

**Q.5** a) Explain different methods of creating threads in Java. (Refer section 5.2) [4]

b) Write a Java program illustrating the concept of thread priority. (Refer section 5.4) [8]

c) What is angular JS ? Enlist and explain the features of it. (Refer section 5.10) [6]

**OR**

**Q.6** a) Explain how to achieve thread synchronization in Java ? (Refer section 5.5) [8]

b) Explain life cycle methods of thread model in Java. (Refer section 5.3) [6]

c) Write a form to make login and password. (Refer example 5.9.2) [4]

**Q.7** a) What is functional programming ? Enlist features. Also enlist commonly used functional programming languages. (Refer section 6.1) [6]

b) Write a short note on - Lambda definitions in LISP. (Refer section 6.7) [6]

c) Explain the symbol manipulation aspect of LISP programming. (Refer section 6.2) [5]

**OR**

**Q.8** a) What is logic programming ? Explain the features of Prolog. (Refer section 6.9) [6]

b) Explain - Terms, Goals, Facts and Rules used in Prolog with example. (Refer section 6.10) [8]

c) Compare functional programming with logic programming paradigm. (Refer section 6.13) [3]

