www.gdcnarasannapeta.ac.in
www.narasannapeta.jkc@gmail.com

GOVERNMENT DEGREE COLLEGE

NARASANNAPETA-SRIKAKULAM DIST.-532421.
Accredited by NAAC 'B' Grade
(Affiliated to DR.B.R.Ambedkar University)

STUDY MATERIAL

Detartment of Computer Science

Procedure Oriented Programming (POP):-

Characteristics:-

1. Emphasis is on algorithms or functions.
2. Large programs are divided into smaller programs known as functions.
3. Most of the functions share global data.
4. Data move openly around the system from function to function.
5. Functions transform data from one location to another.
6. New data & functions are added difficulty whenever necessary.
7. Follows ***top-down*** approach in program design.

Drawbacks:-

1. In a large program it is very difficult to identify what data is used by which function.
2. It does not model real world problems very well.
3. New data and functions can not be added easily.

Object Oriented Programming (OOPs):- It treats data as a critical element in the program development and does not allow data to flow freely around the system. Oops tries data more closely to the function that operate on it and protects it from unintentional modifications by other function.

Features of OOP:-

1) Oops forces on **data** rather than functions.
2) Programs are divided into objects.
3) Data structures are designed characterize the objects.
4) Data is hidden cannot be accessed by external functions.
5) Objects many communicate with each other through methods.
6) New data and methods can be easily added.
7) It follows ***Bottom-up*** approach.

Object Oriented Programming Concepts:-

1) Class:-

- It is a user defined data type.
- It is a collection of variables and methods.
- These variables are called as "Instance Variable" and these methods are called as "Instance Methods".
- In Java, all information is stored with in the classes only.

2) Object:-

- It is also a user defined data type.
- Objects are basic runtime entities in an object-oriented system.
- It may represent a person, a bank account or thing.
- Object is an instance of a Class.
- It is a Class's variable.
- When a program is executed, the object interacts by sending messages to one another.

For example, "customer" and "account" are two objects in a banking program, and then the customer object may send a message to the account object requesting for the balance. Each object contains data and code to manipulate the data.

3) Data Encapsulation:-

- The covering up data and methods into a single unit is known as encapsulation. It is the most striking feature of a class.
- The data is not accessible to the outside world and only methods in the class access it.
- This insulation of the data from direct access by the program is called **data hiding**.

4) Data abstraction:-

- It is representing essential features without including background details or explanations.
- Classes use the concept of abstraction and are defined as a list of abstract attributes like size, type, weight and cost, Methods that operate on these attributes.

5) Inheritance:-

- It is the process of objects of one class gains the properties of object of another class.
- It supports the concept of hierarchical classification.
- It provides the idea of reusability.
- This means we can add additional features to an existing class without modifying it.
- This is deriving a new class from the existing one.
- The new class will have combined features of both these classes.

6) Polymorphism:-

- It is another important OOP concept.
- It means the ability to take more than one form.
- It is one form is used in many ways.
- For example, an operation may exhibit different behaviors in different instances.
- The behavior depends upon the types of data used in the operation.
- For example, the operator '+' is used for numbers and also used for strings concatenation.
- These are mainly divided into 2 types.
    1. Compile-Time Polymorphism (Eg:- Method Overloading)
    2. Run-Time Polymorphism (Eg:- Method Overriding)

7) Dynamic Binding:-

- Binding refers to the linking of a function call to the code to be executed.
- Bindings are mainly 2 types. **Static Binding** or **Dynamic Binding**
- Dynamic Binding means that the code is associated with a function call at runtime.
- Static Binding means that the code is associated with a function call at compile-time.
- It is associated with polymorphism and inheritance.

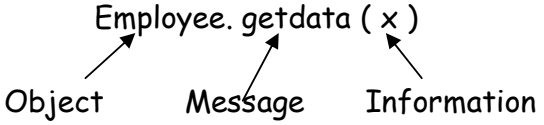
8) Message Passing:- An OOP consists a set of objects that communicate with each other. It involves

1. Creating classes that define objects and their behavior
2. Creating objects from class definitions
3. Establishing communication among objects.

Objects communicate with one another by sending and receiving information.

Message passing involves specifying the name of the object, the name of the method and the information to be sent.

For example **Employee. getdata (x),** Here **Employee** is the **object**, **getdata** is the **message** and **x** is the parameter that contains information.



Benefits of OOP:- It improves greater programmer productivity, better quality of software and lesser maintenance cost.

1) Through Inheritance, **Reusability Of Code.**
2) Data hiding helps the programmer to build secure programs.
3) It can be easily upgraded from small to large system.
4) Software complexity can be easily managed.
5) Message passage techniques for communication between objects make the interface description with external systems much easier.
6) It is possible to have multiple objects to coexist without interference.
7) It is used to partition the work in a project based on objects.

Applications of OOP:-

1. It is used to design Graphical User Interface Applications.
2. Real time systems can be developed
3. Object Oriented Databases
4. Simulation and Modeling
5. Computer Aided Design (CAD), Computer Aided Manufacturing (CAM)
6. Neural Networks
7. Artificial Intelligence

JAVA:-Java is a general-purpose and True Object Oriented Programming language developed by Sun Microsystems of USA in 1991. Originally called "Oak" by James Gosling, One of the inventors of the language. Java was designed for the development software for consumer electronic devices like TVs, VCRs, Cell phones, etc. Java is simple, portable and highly reliable. Java is purely Object Oriented.

Java's designers have borrowed the best features of many existing languages like C and C++ and added a few new features to form simple and easy to learn.

Java was mainly developed two types programming one as a **General programming** and other is **Internet Programming**. The general-purpose programs are known as **Applications** and programs written for Internet are known as **Applets**.

FEATURES OF JAVA LANGUAGE:- The inventors of Java wanted to design a language solutions to some of the problems are occurred in modern programming language. The main features are

1) Compiled and Interpreted Language:-

- Usually a computer language is either compiled or interpreted.
- Java combines both these systems.
- First, Java compiler translates source code into Bytecode.
- Bytecode are not machine-readable instructions.
- Second, Java Interpreter generates Bytecode instructions into machine-readable instructions that can be executed by the machine.
- So Java is both a compiled and an interpreted language.

2) Platform Independent:-

- Java is the first programming language that is not tied to any particular hardware or any operating system.
- Java programs can be easily moved from one computer system to another or one platform to another.
- Changes and Upgrades in Operating Systems, processors and system resources will not change in Java Programs.
- This is the reason Java has become a popular for programming on Internet.

3) Object Oriented:-

- Java is a true object oriented language.
- Almost everything in Java is an Object.
- All programs data and code within objects and classes.

4) Robust and Secure:-

- Java is a Robust (Strong) language.
- It provides many safeguards to reliable code.
- It has strict compile time and runtime checking for data.
- In java the concept of Exception handling removes the errors and also risk of crashing the system.
- Security becomes an important issue for a language used for programming on Internet.
- Java systems not only verify all memory access but also no viruses are communicated with an applet.

5) Distributed:-

- Java is designed as a distributed language for creating applications on networks.
- It has shared both data and programs on networks.
- Java applications can open and access remote objects on Internet as easily as in a local system.
- The data can be moved from one system to another by using Java programs.

6) Simple and Small:-

- Java is a small and simple language.
- Many features of C and C++ are added in Java and some of the features like pointers, header files, operator overloading, templates are not part of Java.

7) Multithreaded:-

- Java supports multithreading programming.
- It means handling multiple works simultaneously.
- This means that we need not wait for the application to finish one task before beginning another.
- For example, we can listen to an audio clip while typing a java program.

8) High performance:-

- Java performance is impressive for an interpreted language the use of byte code.
- Java architecture is designed to reduce errors during runtime.
- The multithreading enhances execution speed of Java programs.

9) Dynamic and Extensible:-

- Java is a dynamic language.
- It is dynamically linking in new classes, methods and objects.
- Java programs support functions written in C and C++.
- These functions are known Native methods. These are linked dynamically at run time.

Difference between C and JAVA:-

1. Java does not support header files.
2. Java does not support pointers.
3. Java does not global variables.
4. Java does not support struct, union and enumerated data types.
5. Java does not support GOTO.

Difference between C++ and Java:-

1. Java does not support operator overloading.
2. Java does not have template classes as in c++.
3. Java does not support multiple inheritances of classes.
4. Java does not support global variables. Every variable and method is declared within a class.
5. Java does not use pointers.
6. No header files are used in Java
7. Java has replaced the destructor function with finalize () method.

Minimum Hardware & Software Requirements:-

Java is currently supported on Windows 95 and other windows versions. The minimum hardware requirements of Java are

- IBM compatible 486 processor.
- Minimum 8 MB RAM.
- Minimum 2GB Hard disk.
- A windows compatible sound card
- CD ROM, mouse

Web Page:- A file is used in Internet is called as webpage. HTML (Hypertext Markup Language) is used to design or create web pages. A Web page is basically made up of text and HTML tags. Web pages are stored using file extension **.html**. A web page is also known as HTML page or HTML document.

World Wide Web:- It is an open-ended information retrieval system designed to be used in the internet. This system contains known as web pages that provide both information and controls. **Tim Berner's Lee** developed it. It is graphical part of the Internet. It is also known as collection of web sites.

Web site:- It is collection of hundreds of webpage. The starting webpage of the web site is home page.

Web browsers:-

- It is software/application used to access the internet.
- They are used to navigate/searches the web pages in the Internet.
- They allow us to retrieve the information spread across the Internet and display it using the HTML.

- Examples of web browsers are **hot java**, **internet explorer** and **Netscape navigator**.

Hot Java is the default web browser from sun micro systems that enables the display of interactive content on the web. Internet Explorer is another popular browser developed by Microsoft for windows.

Java Environment:-

- It includes a large number of Development Tools and hundreds of Classes and Methods.
- It contains mainly 2 parts.
    1. Java Development Kit (JDK)
    2. Java Standard Library (JSL)
- The Development tools are part of the system known as Java Development Kit (JDK) and the classes and methods are part of the Java standard Library (JSL) or Application Program Interface (API).

1) Java Development Kit (JDK):-

It comes with a collection of tools that are used for developing and running java programs. They include

a) Appletviewer:-
- It is used to run Java Applets.

b) Javac:-
- It is also known as Java Compiler.
- It translates the Java source code files to Byte code files that the interpreter can be understand.

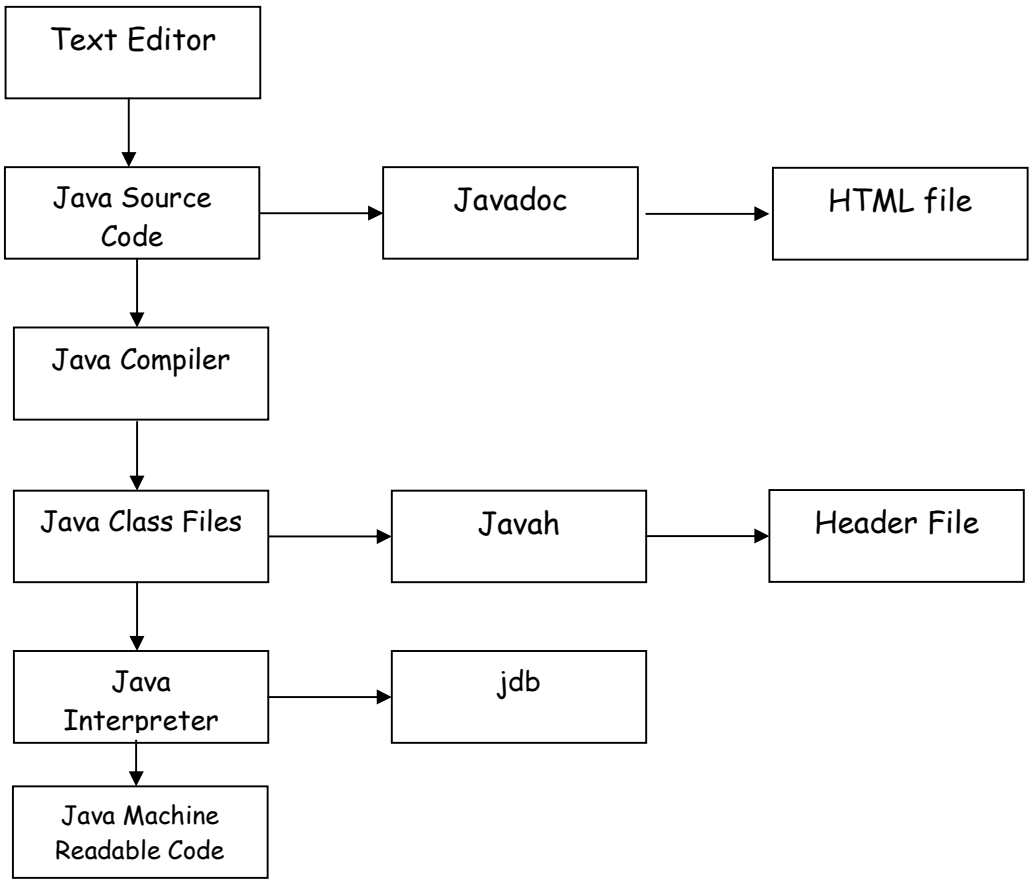
c) Java:-
- It is also known as Java Interpreter.
- It runs java applets and Java applications by reading and interpreting Byte-code files.

d) Javah:-
- It produces header files for use with native methods.

e) Javap:-
- It is also known as Java Disassembler.
- It enables to convert byte-code files into a program description (source code).

f) Javadoc:-
- It creates HTML format documentation from Java Source code files.

g) Jdb:-
- It is also known as Java debugger.
- It helps us to find errors in our programs at compile time/runtime.



Application Program Interface (API):- The Java Standard Library or Application Program Interface includes hundreds of classes and methods grouped into a several Package. Package is stores collection of classes and methods. It contains some Predefined packages. These are

1) **Language Packages:-** It is a collection of classes and methods required for implementing basic features of Java. It contains Mathematical functions, String classes and functions, etc.
2) **Utility Packages:-** It is a collection of classes and methods to provide utility functions such as data & time functions and classes.
3) **Input/Output Packages:-** It is a collection of classes and methods are required for input/output manipulations.
4) **Networking Packages:-** It is a collection of classes for communication with other computers. These classes are used in networking.
5) **AWT Packages:-** AWT means Abstract Window Toolkit. It is collection of classes and methods that implements Graphical User Interface applications.
6) **Applet Packages:-** This include a set of classes and methods that allows us to create Java Applets.

Java Program Structure:- A Java program may contain many classes. Classes contain data members and methods that operate on the data members of the class. A Java program may contain one or more sections.

1) Documentation section:- It comprises a set of comment lines giving the name of the program, author of the program and other details. Java also uses a two styles of comments in C++
   i.e /*…… */ and // and also third style comment is //* ……. */.
2) Package Section:- The first statement allowed in a Java file is a Package Statement. This Statement declares a package name and informs the compiler that the classes defined.
3) Import Statement:- This is similar to the #include statement in C. This statement instructs the interpreter to load the class contained in the package.
4) Interface Statements:- An interface is like a class but includes a group of method declarations. This is also an optional section.
5) Class Definitions:- A Java program may contain multiple class definitions. Classes are the primary elements of a Java program. The number of classes used depends on the complexity of the problem.
6) Main Method Class:- Every Java program requires a main method, this class and establishes part of a Java program. The main method creates objects of various classes and establishes communication between them. On reaching the end of main, the program terminates and the control passes back to the operating system.

Documentation Section // Optional
Package Statements // Optional
Import Statements
Interface Statements // Optional
Class definitions

Main Method Class
{
Main Method Definition;
}

Java Tokens:-

- In Java, any information is stored with in the class.
- Class contains variables and methods.
- Smallest individual units in a program are known as Tokens.
- A Java program is a collection of tokens, comments and white spaces.
- Java includes five types of tokens. They are
  1) **Keywords:-**
     - These are predefined words or Reserved Words.
     - It has special meaning.
     - We cannot change this meaning. The compiler defines these.
     - Java has reserved 60 words as Keywords.
     - These are written in lower case letters.
  2) **Identifiers:-**
     - These are programmer designed tokens.
     - These are used for naming classes, methods, objects, packages, variables, etc..
     - Java Identifiers follow the following rules.
       1. They must not begin with a digit.
       2. They can have alphabets (A..Z), digits (0-9), underscore (_) and dollar sign ($).
       3. They can be any length.
       4. Java is Case sensitive. Upper case and lower case letters are different
  3) **Literals:-**
     - These are sequence of characters or digits that represent constant values to be stored in variables.
     - Java Language specifies 5 types of Literals. These are
       - Integer constants
       - Float Constants
       - Character Constants
       - String Constants
       - Back Slash Constants.
  4) **Operators:-**
     - An Operator is a symbol that tells the computer to perform mathematical and logical manipulation. Operators are used in Java program to manipulate data and variables.
  5) **Separators**:- They are symbols used to indicate groups of code are divided and arranged. They basically define the shape and function of our code.

Parentheses**()**:- It is used to enclose parameters in method definition and invocation, also used for defining precedence in expressions, containing expressions for flow control.

Braces**{}**:- It is used to contain the values of automatically initialized arrays and define a block of code for classes, methods and local scopes.

Brackets **[]**:- It is used to declare array types.

Semicolon ;:- It is used to separate statements.

Comma (,):- It is used to separate consecutive identifiers in a variables declaration, also used initialize two or more variables in a for statement.

Period (.):- It is used to separate package names from sub-packages and classes, also used to separate variable or method from a reference variable.

Sample Java Program:-

```
class tara
{
public  static  void  main (String  args[])
{
System.out.println ("Sample Java Program");
}
}
```

The first line "class tara" declares a class. Java is a true object oriented language. Every thing must be placed inside a class. Every class definition in java begins with an "{" and ends with "}" appearing in the last line.

public static void main(String args[]) defines main method in Java. It is similar to main() in C/C++. Every Java application program must include main(). This is the starting point for the interpreter to begin execution of the program. A Java application can have any number of classes but only one main method in initiate execution.

Public:- This keyword is an access specifier that declares the main() and access to all other classes.

Static:- This keyword declares the main() belongs to the entire class not part of any objects of the class.

Void:- It states that the main() does not return any value.

String args[ ] declares array of objects of the predefined **String** class.

System.out.println() is similar to the **printf()** statement of C or **cout<<** construct of C++. The **println()** method is a member of **out** object, which is the static data member of **System** class. Every Java statement must ends with semicolon.

Implementing of Java Application program involves compiling & Running the Java Program.

Compiling the program, we must run Java Compiler (javac) with the name of the source file on the command line.

C:\jdk1.4.2\bin> javac tara.java

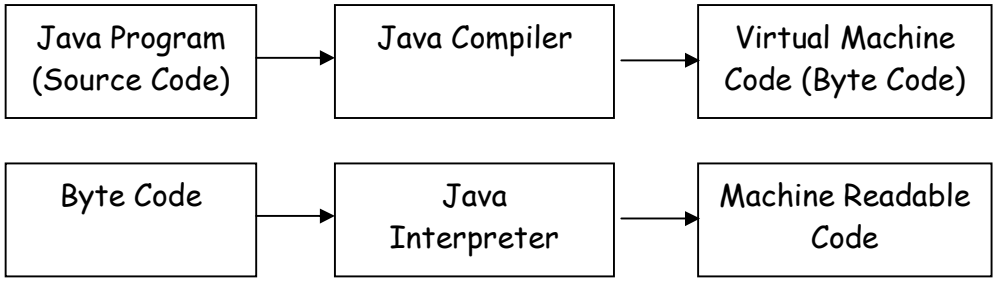
If no errors in a program, the Java Compiler creates a Bytecode file called **tara.class.**

Running the Program:- After compiling, we need to use the Java Interpreter (java) to run a program on the command line.

C:\JDK1.4.2\bin> java tara

The interpreter looks for the main method in a program and begins execution when executed output will come.

Java Virtual Machine(JVM):- All language compilers translate source code into machine code. Java compiler produces an intermediate code known as Bytecode for a machine that does not exist. The Java Virtual Machine exists only inside the computer memory and the process of compiling a java program into Bytecode i.e. virtual machine code. The virtual machine code is not machine understandable. The Java interpreter generates the Machine code by acting as an intermediary between the virtual machine and the real machine.



Data Types:- Every variable is Java has a data type. Data types specify the size and type of values that can be stored. Data types in Java are mainly two types. These are

1) Primitive Data Type (Predefined Data Type)
2) Non-Primitive Data Type (User Defined Data Type)

Primitive Data Types:-

- These are mainly combination of numeric and non-numeric data types.
- These are mainly two types.
  - **Integer type**
  - **Float type**.

Integer Type:-

- Integer type can hold whole numbers. Eg:- 123,-96 ,89.
- Java supports 4 types of Integer.
- These are **byte, short, int** and **long**.
- Java does not support unsigned types.

Byte:-

- It stores smallest integer values.
- It occupies **one Byte** memory.
- Its range is -128 to +127 or (-2^7 to 2^7-1)

Short:-

- It occupies **two Bytes** Memory.
- Its range is -32,768 to +32,767 or (-2^{15} to $2^{15}-1$)

Int:-

- It occupies **four Bytes** Memory.
- Its range is -2147483648 to 2147483647 or (-2^{31} to $2^{31}-1$)

Long:-

- It occupies **Eight Bytes** Memory.
- It stores large integer values. Its range is (-2^{63} to $2^{63}-1$).

Float Type:-

- This type to hold numbers containing fractional parts (Decimal values).
  Ex:- 12.67, 89.45.
- These are two kinds of floating point storage in Java. These are **float** and **double**.

Float types values are single precision numbers.

- It occupies **4 Bytes** in memory.
- The float range is **3.4 E-038 To 3.4 E+038.**

Double types represent double precision numbers.

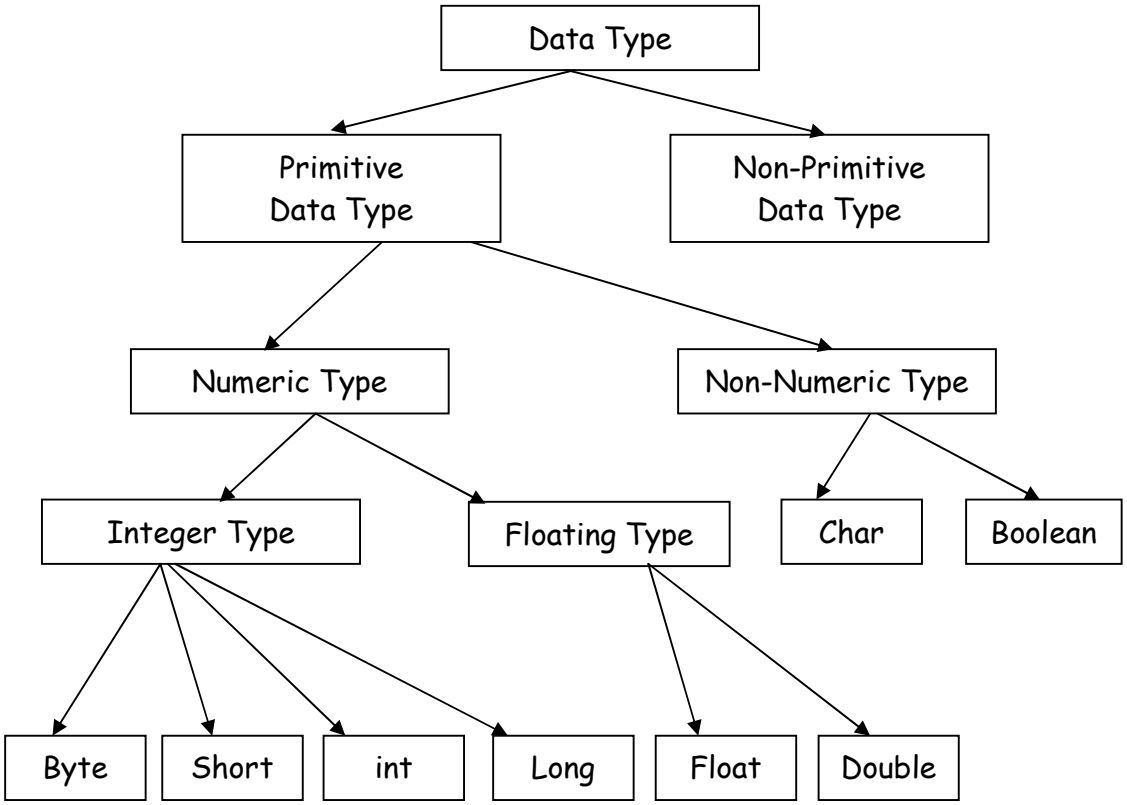
- It occupies **8 Bytes** memory.
- These are used greater precision in storage of floating point numbers. All mathematical function (Sin, Cos, sqrt) return double values.
- Its range is **1.7e-308 to 1.7e+308**.

Character Type:-

- It stores character type values in memory.
- Java provides a character data type **char**.
- It occupies **2 Bytes** memory, but it can store only one character.

Boolean Type:-

- It is used to test a particular condition during execution of the program.
- It can take **true** or **false** values.
- It uses only **one bit** of storage.



Operators:- An Operator is a symbol that tells the computer to perform mathematical and logical manipulation. Operators are used in programs to manipulate data and variables. Java supports 8 types of Operators. These are

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Assignment Operators
5. Increment/Decrement Operators
6. Conditional Operators
7. Bitwise Operators
8. Special Operators

1. Arithmetic Operators:- Java provides all basic arithmetic operators + (addition), - (subtraction), * (Multiplication), / (Division) and % (Modulus Operator). These operators can operate on any built-in numeric data type of Java. We cannot use these operators on **Boolean** data type. In C/C++ modulus operator (%) can be used in integers only but in Java % operator can be applied to the floating-point data type.

In Modulus division, the sign of the result is always sign of the first operand. **Ex:-** -14%3=-2 -14%-3=-2 14%-3=2

When all the operands are integers, the expression is called integer expression and the operation is called Integer arithmetic. Integer Arithmetic always gives integer value.

An arithmetic operation involving only real operands is called real arithmetic. A real operand values either in decimal or exponential notation.

An arithmetic operation involving different type of operands is called mixed arithmetic.

In C/C++ **%** operator does not support float type values. In Java % operator support float type values.

2. Relational Operators:- Java supports six relational operators. These are used in checks the relations or conditions between two operands. These are **<**, **>**, **>=**, **<=**, == and **!=**. A simple relation or condition contains only one relational operator.

Syntax:- **arithmetic expression2 Operator arithmetic expression2**

Example:- a>b, a>c.

When arithmetic expressions are used on either side of a relational operator, the expression is evaluated first and then the results compared.

3. Logical Operators:- Like in C/C++, Java also three logical operators. These are **&&** (Logical And), **||** (Logical OR) and **!** (Logical Not). The Logical operators && and || are used combining two or more relations. A & B are two conditions.

A	B	A&&B	A||B
False	False	False	False
False	True	False	True
True	False	False	True
True	True	True	True

If all the conditions are **true** then the Logical And (**&&**) will be **true** otherwise **false**.

If all the conditions are **false** then the Logical Or (**||**) will be **false** otherwise **true.**

4. Assignment Operators:- It is used to assign the some value to a variable and an expression. The Assignment operator is "=". Java has a set of shorthand assignment operators +=, -+, *=, /= and %=.

Syntax:- variable operator = expression

Example:- a +=b => a=a+b

The use of shorthand assignment operators has results in a more efficient and easier to read.

5. Increment/Decrement Operators:- These are called as Unary operators. Java has two unary operators. ++ Uses incremented by 1 and -- uses decremented by 1.

Syntax:- ++Variable (Pre-Increment) Variable++ (Post Increment)

Example:- ++a => a=a+1 --a => a=a-1

6. Conditional Operators:- Java has a two conditional Operators. **?** and **:** are called as ternary operators. This operator is used to construct conditional expression.

Syntax:- variable = (Condition) ? Expression1 : Expression2.

Example:- big = (a>b) ? a : b

If condition is true then the Expression1 is evaluated otherwise Expression2 is evaluated.

7. Bitwise Operators:- These operators are used to for testing the bits or shifting them to the right or left. Bitwise operators may not be applied to **float** or **double**. Java has 7 bitwise operators. These are **&** (Bitwise And), **|** (Bitwise Or), **^** (Bitwise Exclusive Or), **>>** (Right Shift), **<<** (Left Shift) and **~** (1's compliment).

A	B	A&B	A|B	A^B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Bitwise And (&):- It is like a Logical And (&&). Logical And (&&) Calculates Logical Conditions only, but Bitwise And (&) calculates bits (0 or 1) only.

The Bitwise And (&) is 1, if all the bits have a value 1,

The Bitwise AND (&) is 0. If any one bit is 0.

Bitwise Or:- It is represented by the symbol (|).

The result of Bitwise Or (|) is 1, if at least one of the bits have a value 1.

The Bitwise OR is 0, If all the bits are 0's.

Bitwise Exclusive Or:- It is represented by the symbol " ^ ".

The result of Exclusive Or is 0, if all the bits are 0's and 1's. Otherwise it is 1.

Example:- a=51 b=34

A	→	1	1	0	0	1	1	
B	→	1	0	0	0	1	0	
A & B	**→**	**1**	**0**	**0**	**0**	**1**	**0**	**=> 34**
A	→	1	1	0	0	1	1	
B	→	1	0	0	0	1	0	
A | B	**→**	**1**	**1**	**0**	**0**	**1**	**1**	**=> 51**

A	→	1	1	0	0	1	1	
B	→	1	0	0	0	1	0	
A ^ B	**→**	**0**	**1**	**0**	**0**	**0**	**1**	**=> 17**

Bitwise Left Shift (<<):- The shift Operators are used to move bit patterns either to the left or to the right.

The Left shift operator causes all the bits in the operand to be shifted to the left by number of positions.

The left most n bits in the original bit pattern will be shifted and rightmost '**n**' bit positions will be filled with 0 s.

Eg:-	A	→			1	1	0	0	1	1 = 51
	A<<1	→		1	1	0	0	1	1	0 = 102
	A<<2	→	1	1	0	0	1	1	0	0 = 204

Bitwise Right Shift (>>):- The right shift operations causes all the bits in the operand to be shifted to the right by '**n**' positions.

The rightmost '**n**' bits will be lost.

The leftmost n bit positions are vacated will be filled with 0s.

Eg:-	A	→		1	1	0	0	1	1 = 51
	A>>1	→		0	1	1	0	0	1 = 25
	A>>2	→		0	0	1	1	0	0 = 12

CONTROL STRUCTURES:-

- These are collection of tools and techniques to process the data efficiently.
- A Java program is a set of statements executed sequentially.
- When a program breaks the sequential flow and jumps to another part of the code is called branching.
- When the branching is based on a particular condition is called conditional branching.
- If branching takes place without any decision is called unconditional branching. Java supports mainly 4 control structures. These are

1. Single Conditional Control Structure
2. Multi Conditional Control Structure
3. Loop Conditional Control Structure
4. Un Conditional Control Structure

Single Conditional/Decision/Directional Control Structure:- Java Statements are executed based on a single condition. In this type Java has two decision making statements. They are **if** and **if-else**.

If Statement:- The **If** statement is a powerful single decision making statement and is used to control the flow of execution of statement.

It checks the TRUE condition only.

Syntax:- **if (Condition)**

Statement1;

The **if** statement checks the condition first, If the condition is **true** then the **statement1** will be executed, otherwise it skips another block.

If-Else Statement:- The If-Else statement is an extension of the simple if statement. It is basically two-way decision statement. It checks the condition is either TRUE or FALSE.

Syntax:- **if (Condition)**

Statement1;

Else

Statement2;

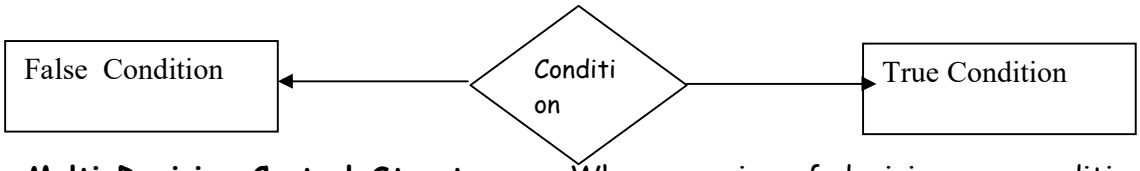
If the condition is **true** then the **statement1** will be executed, otherwise **statement2** will be executed.

Example:- if (a>b)

System.out.println (a + "is big");

else

System.out.println (b + "is big");



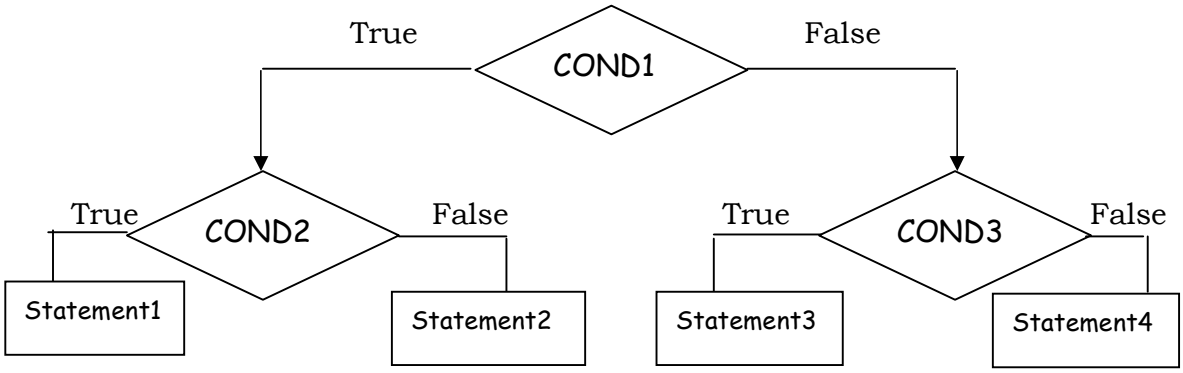
Multi Decision Control Structure:- When a series of decisions or conditions are involved, we use the multidecision control structures. These are **Nested If** and **Switch** statements.

Nested If:- When a series of decisions are involved and we check the two or more conditions at a time we used **Nested If**. **Nested If** is an **if** statement within another **if** statement.

Syntax1:- if (condition1)

```
        {   if (condition2)
                {   Statement1;
                }
            else
                {   Statement2;
                }
        }
    else {  if (condition2)
                {   Statement1;
                }
            else
                {   Statement2;
                }
        }
```

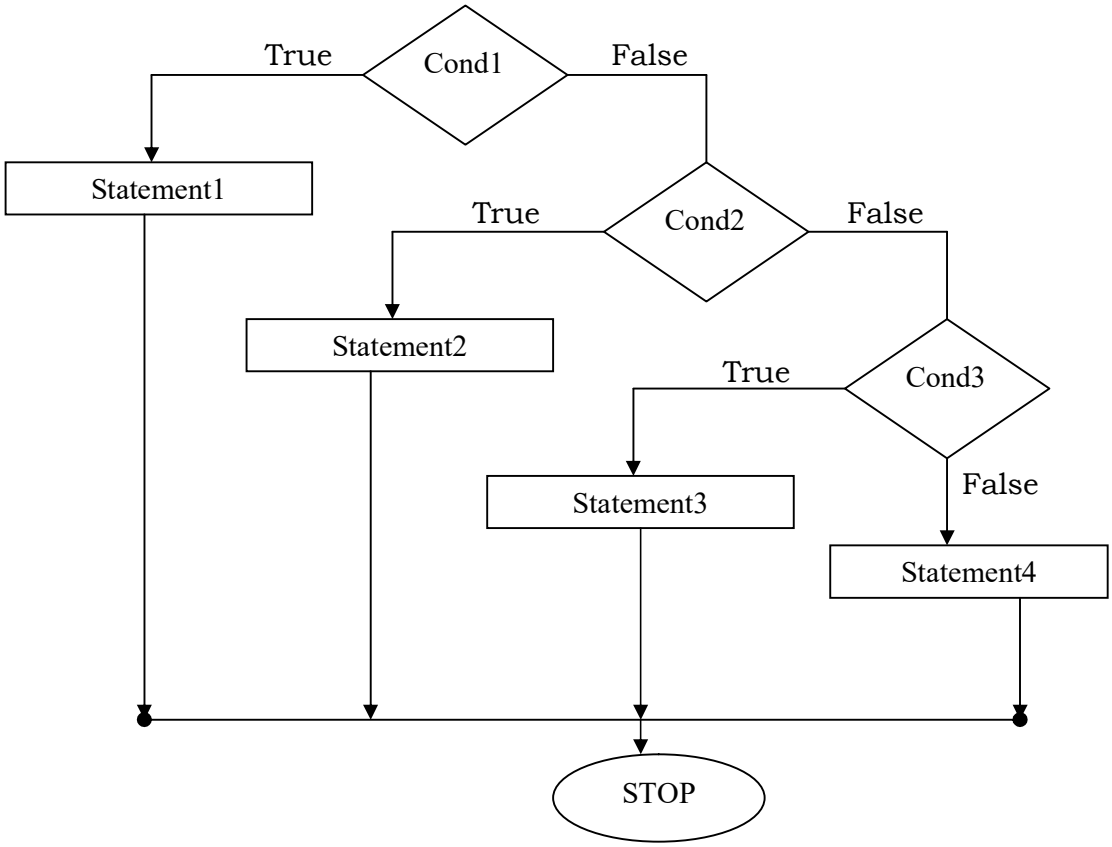
If the **condtion1** is TRUE then check the **Condition2**. If the **Condition2** is TRUE then **statement1** will be executed otherwise **statement2** will be executed. If the **condition1** is FALSE then check the **condition3**; If the **condition3** is TRUE then **statement3** will be executed otherwise **statement4** will be executed.



Syntax2:-

```
if (Condition1)                eg:-   if ((a>b)&&(a>c))
        Statement1;                          System.out.println("a is big");
        else if (Condition2)                 else if (b>c)
                Statement2;                  System.out.println("b is big");
                else if (condition3)         else
                        Statement3           System.out.println("c is big");
                        else
                        Statemnt4;
```

If the **condition1** is true then **statement1** will be executed otherwise it checks the **condition2**, if the **condition2** is true then **statement2** will be executed otherwise checks the **condition3**, if the **condition3** is true then **statement3** will be executed otherwise **statement4** will be executed.



Switch Statement:-

- Java has a built-in multiway decision statement known as a **switch**.
- The switch statement test the value of a given variable or expression against a list of **case** values.
- When a match is found, a block of statements with that **case** is executed.
- When the match is found, the remaining conditions are not executed, then we use **break** statement.
- Break is used to terminate the loop.
- The value of a given variable is not matched, and then executes the **default** case.

The general form of the **switch** is

Syntax:-
```
switch(expression | value)
{
case value1: statement1; break;
case value2: statement2; break;
case value3: statement3; break;
default:     statementn;
}
```

Example:-
```
switch(ch)
{
case 1:    System.out.println("sum=" + a+b); break;
case 2:    System.out.println("Mul=" + a*b); break;
default:   System.out.println("Invalid Choice:");
}
```

Loop Control Structure:-

- The process of repeatedly executing a block of statements is known as looping.
- The statements in the block may be executed any number of times from 0 to infinite number.
- If a loop is not ended, it is called as an infinite loop.
- In looping a sequence of statements are executed until condition for the termination of the loop.
- A loop consists of two segments, one is **Body of loop** and the second one is **control statement.** Control statement tests condition and then directs the repeated execution of the statements in the body of the loop.

  A looping process contains 3 steps. These are
  1. Setting and Initialization of a variable (**Initialization)**
  2. Test for a condition for execution of the loop (**Condition**)
  3. Execution body of the loop (**Body of the Loop**)

The Looping control structure may be two types.

Entry level loop:- In the Entry level loop, the conditions are tested before the start of the loop. **while** and **for** are entry leveled loop.

Exit level loop:- In the Exit level loop, the condition is tested at the end of the loop. **Do-while** loop is exit level loop.

Java language provides 3 loop control structures. They are

1) while loop 2) do while loop 3) for loop.

The While Statement :-

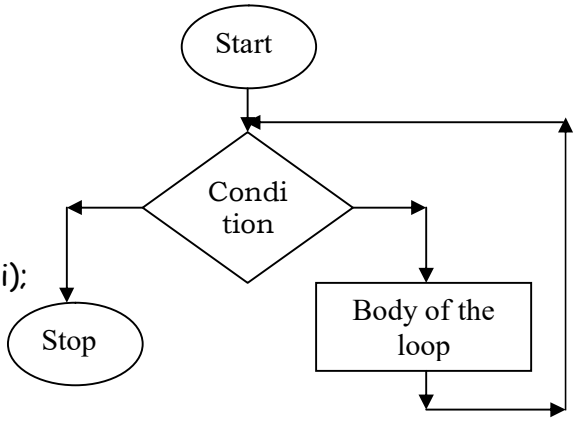
- The while statement is an entry level loop statement.
- It initialize the variable first then check the condition.
- If the condition is true, then the body of the loop is executed one time. After execution of the body, it checks the condition again.
- This process was repeated until the condition is failed.

Syntax:-

```
initialization;
While (Condition)
{
        Body of the loop;
}
```

Example:-

```
i=1;
While(i<=10)
{  System.out.println("i=" + i);
        i++;
}
```



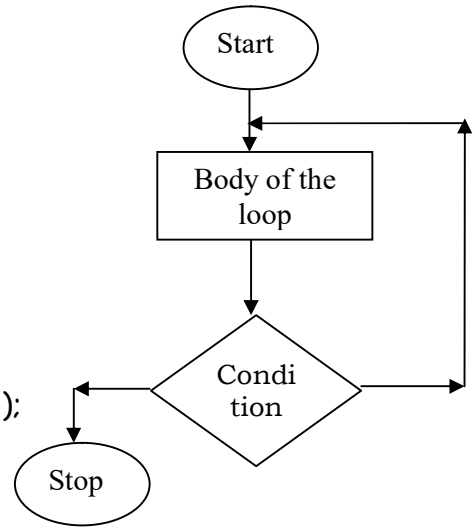
1. Start
2. Initialization of a variable
3. Check the condition. If the condition is true Goto step4; Otherwise Goto step5.
4. Execute body of the loop then goto step2.
5. Stop.

The Do While Statement:-

- It is an Exit-level Loop.
- In this loop was executed first and after checks the condition.
- The **Do** statement to evaluate the body of the loop first.
- At the end of the loop, checks the condition in the **while** statement.
- If the condition is TRUE then the loop was executed again once.
- This process was repeated until the condition is failed.

Syntax:- Initialization;
Do
{
body of the loop;
} while (condition);

Example:- i=1;
Do
{
System.out.println("i=" + i);
i++;
} while (i<=10);



Do while loop executes the body of the loop at least one time even though the condition is failed.

The For Statement:-

- The **for** loop is another *entry-controlled* loop. The general form of for loop is

Syntax:- for (initialization; test condition; increment/decrement)
{
body of the loop;
}

The three sections enclosed within parentheses must be separated by semicolons. One of the important point is all three sections are placed in the for statement with in the one line.

Example:- for(i =1; i<=10; i++)
{
System.out.println(" i= " + i);
}

The execution of the **for** statement is as follows:

1. *Initialization* of the variable first.
2. Check the test *Condition*. If the condition is true, then the body of the loop is executed; otherwise the loop is terminated.
3. After ending the **for** loop it goto the increment/decrement section.
4. After the increment/decrement it again checks the condition. If the condition is true this process will be repeated otherwise goto end.

In for loop more than one variable can be initialized and incremented/decremented at a time in the for statement by using comma (,) operator.

Example:- for(i=1, s=0; i<=10; i++, s=s+i)
{ body of the loop

```
        }
```

Nesting of for loops:- One for statement within another for statement is called nested for.

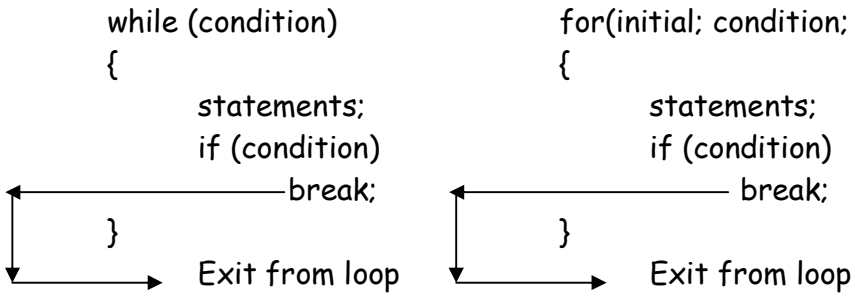
Syntax:-

```
for(initilization1; condition1; inc/dec1)
{
    for (initilization2; condition2; inc/dec2)
        {
            body of the loop;
        }
}
```

Jumps in Loops:- Loops perform a set of operations repeatedly until the l variables fails to satisfy the test condition. When executing a loop it becomes desirable to skip a part of the loop or leave the loop as soon as certain condition occurs. Java permits a jump from one statement to the *ending* or *beginning* of loops. We use two statements jump from loop. These are ***break*** and ***continue***.

When ***break*** statement is encountered inside a loop, the loop is immediately existed and program continuous with the statement immediately following the loop. When the loops are nested, the break would only exit from the loop containing it. The break will exit only a single loop.

Example:-



Example Program:-

```
class breakexam
{
	public static void main(String arg[])
	{
		for(int i=1;i<=10;i++)
		{
			if (i==5)
			{
				break;
			}
			System.out.println("i=" + i);
		}
	}
}
```

Java supports another similar statement called the ***continue*** statement. It causes the loop to be continued with the next iteration after skipping any statements in between.

Example:-

```
while (condition)
        {
                statements;
                if (condition)
                        continue;
        }
                Exit from loop
```

```
for(initial; condition; inc/dec)
        {
                statements;
                if (condition)
                        continue;
        }
                Exit from loop
```

Example Program:-

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

CLASSES AND OBJECTS

Class: -

- Java is a true object oriented language.
- In java any information is stored within the class.
- A class is a user-defined data type.
- It is a collection of variables and methods.
- These variables are called as **instance variables** and these methods are called as **instance methods**.

```
Syntax:-    class  classname
            {
                   instance variables;
                   instance methods;
            }
```

Adding variables:-

- The variables are declared within the class.
- They are created whenever an object of the class is created.
- We can declare the instance variables same as the local variables.

Adding methods:-

- A class with only data fields has no life.
- Methods are declared and defined inside the class but immediately after the declaration of instance variables.
- The general form of a method declaration is

```
Syntax:-    returntype   method_ name ( parameter list)
            {
                   Body of the method;
            }
```

The **return datatype** specifies the type of value the method would return. It can be int, float, void etc. If the method does not return any value, then it would be declared as **void**. The method name is valid identifier. The parameter list is always enclosed in parentheses.

```
Example:-   class  sample
            {      int a, b;
                   void getdata(int x, int y)
                   {       a=x;
                           b=y;
                   }
                   void  display()
                   {       System.out.println("a=" + a);
                           System.out.println("b="+ b);
                   }
            }
```

Creating Objects:-

- It is also user defined data type.
- Object is an instance of the class.
- Class's variables are called as objects.
- An object in java is a block of memory that contains space to store all the instance variables.
- Objects in java are created using the **new** operator.
- The **new** operator creates an object of the specified class and returns a reference to that object.

General form object creation is

Syntax1:- **classname objectname;** // declare the object

Syntax2:- **objectname = new classname( );** // Memory allocation

Syntax3:- **classname objectname = new classname ( )**

The first statement declaring the variable holds the object reference and the second one assigns the memory allocation or object reference to the variable.

Example1:- **sample s1;**

Example2:- s1 = new sample ( );

General Example for creating the object

Example3:- sample s1 = new sample ( );

Accessing Class Members:- All variables must be assigned before they are used. Outside the class, we can not access the instance variables and the methods directly.

Syntax:- **objectname . variablename or**
Objectname . methodname (parameter)

Example:- s1. a=10 or s1. get (10,20)

Program:-

```
class rect
{       int l, b;
        void getdata(int x, int y)
        {       l=x;
                b=y;
        }
        void display()
        {
        System.out.println("Length=" + l);
        System.out.println("Breadth=" + b);
        }
        int area()
        {       return (l * b);
        }
}
```

```
class rectarea
{       public static void main(String args[])
        {       rect r1 = new rect ();
                r2 = new rect ();
                r1.getdata(10,20);
                r1.display();
                System.out.println("Area=" + r1.area());

        }
}
```

Method Overloading:-

- Methods names are same but the parameters are different and also definitions are different. This concept is called as Method Overloading.
- It is the compile-time polymorphism.
- In this, methods are executed depending on the parameters.
- We can design a family of methods with same method name but with different argument lists.
- The method would perform different operations depending on the argument list in the method call.
- The correct method to be called is checking the number of arguments and data type of arguments.
- A method call first matches the prototype having same number and data type of arguments and then calls the appropriate method for execution.

Example:-

```
class  room
{       int     l, b;
        void area()
        {       l=10; b=20;
                System.out.println("Rectangle area=" + (l*b));
        }
        void    area(int x, int y)
        {       l=x;
                b=y;
                System.out.println("Rectangle area=" + (l*b));
        }
        void    area(int x)
        {       l=b=x;
                System.out.println("Square area=" + (l*b));
        }
}
```

```
class overload
    {   public static void main(String args[])
        {   room r1 = new room ();
            r1.area();
            r1.area(30);
            r1.area(40,50);
        }
    }
```

Constructors:-

- It is special Method. It is same name as the class name.
- It does not specify any return type, not even void.
- It is automatically called when the object is initialized.
- It initializes the variables with in the class.
- Constructors have parameters is called parameter constructors.
- A class can contain two or more constructors is called as constructor overloading.

Example:-

```
class sample
{   int a,b;
    sample()
    {   a=10;  b=20;
    }
    sample(int x)
    {   a=b=x;
    }
    sample(int x, int y)
    {   a=x;   b=y;
    }
    void display()
    {   System.out.println("a=" + a);
        System.out.println("b=" + b);
    }
};
class conover
{   public static void main(String[] args)
    {   sample s1=new sample();
        sample s2=new sample(30);
        sample s3=new sample(40,50);
        s1.display();   s2.display();   s3.display();
    }
}
```

Syntax:-

```
class sample
    {
        int a,b;
        sample( int x)
        {
        body of method;
        }
    }
```

Static Members:-

- It is a collection of static variables and static member functions.
- These are declared as static are called static members.
- These members are associated with the class itself.
  - ⊙ Static variables are initialized to zero.
  - ⊙ Static variables are used to have a common to all instances of a class.
  - ⊙ These are accessed by the class name.

Syntax:- classname. variablename

Static methods are also called using the classname. Static methods have several restrictions:

- ⊙ They can only call other static methods. are
- ⊙ They can only access static data.
- ⊙ They can not refer to **this** or **super.**

```
class sample
{
        static int a=20;              // static variable
        static int b=30;              // static variable
        static void display()         // static method
        {
                System.out.println("a=" + a);
                System.out.println("b=" + b);
        }
};
class staticexample
{
        public static void main(String[] args)
        {

                sample.display();     //      call static method
        }
}
```

Nesting Methods:-

- A method of a class can be called another method of same class is called as nesting method.
- A method can be called by using only its name by another method of the same class. This is known as nesting of methods.

Program:-

```
class sample
{
        int a,b;
        sample (int x, int y)                  // parameter constructor
        {
                a=x;
                b=y;
        }
        void display()
        {
                System.out.println("a=" + a);
                System.out.println("b=" + b);
                System.out.println("big=" + big());       // call another method
        }
        int big()
        {
                if (a>b)
                        return (a);
                else
                        return(b);
        }
};
class  nesting
{
        public static void main(String[] args)
        {
                sample s1=new sample(10,20);
                s1.display();
        }
}
```

Inheritance:-

- We can create new classes, reusing the properties of the existing ones.
- The mechanism of deriving a new class from an old one is called inheritance.
- The old class is referred to as the **super class** or **parent class** or **base class** and the new one is called the **sub class** or **child class** or **derived class**.
- The sub class inherits the some or all the properties from the super class.
- A sub class can also inherit properties from more than one super class or more than one level.

Inheritance may take different forms

- ⊙ Single Inheritance
- ⊙ Multiple Inheritance
- ⊙ Multilevel Inheritance
- ⊙ Hierarchical Inheritance.

Java supports the concept of *reusability of code*. Java classes can be reused in many ways by using inheritance.

Single Inheritance:- The new sub class is developed from only one super class is called Single Inheritance.

Multiple Inheritance:- The new sub class is developed/derived from two or more super classes is called multiple Inheritance.

Multilevel Inheritance:- The new sub class is derived from another sub class is called multilevel inheritance.

Hierarchical Inheritance:- The two or more sub classes is derived from only one super class is called hierarchical Inheritance.

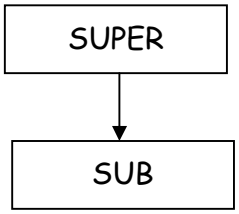
Defining a Subclass:- A sub class can be defined by specifying its relationship with the super class. The general form of defining a sub class is

Syntax:-

```
class subclassname extends superclassname
{   instance variables declaration;
    instance methods declaration;
};
```

- The keyword **extends** signifies that the properties of the **superclassname** are extended to the **subclassname**.
- The sub class will contain its own variables and methods and also super class variables and methods.

Single Inheritance:- The new sub class is derived from only one super class is called Single Inheritance. The sub class inherits some or all the properties of the super class.



Syntax:-
```
class  A
{      instance variables;
       instance methods;
}
class  B      extends      A
{      instance variables;
       instance methods;
}
```

Example:-
```
class rect
{      int l, b;
       void getrect(int x, int y)
       {      l=x;    b=y;
       }
       void area()
       {      System.out.println("Length=" + l);
              System.out.println("Breadth=" + b);
              System.out.println("Rectanle Area=" + (l*b));
       }
};
class  cube   extends rect
{      int h;
       void getcube(int x)
       {      h=x;
       }
       void volume()
       {      System.out.println("Height=" + h);
              System.out.println("Cube Volume=" + (l*b*h));
       }
};
class single
{      public static void main(String[] args)
       {      cube   c1 = new cube();
              c1.getrect (10, 20);
              c1.getcube (30);
              c1.area ();
              c1.volume ();
       }
}
```

Sub class Constructor:- A sub class constructor is used to construct the instance variable of both the subclass and the super class. The sub class constructor uses the keyword **super()** to call the constructor of the super class.

- **super** may only be used within a subclass constructor.
- **super** method can call to super class constructor must appear as the first statement within the subclass constructor.
- The parameters in the **super** call must match the order and type of the instance variables declared in the super class.

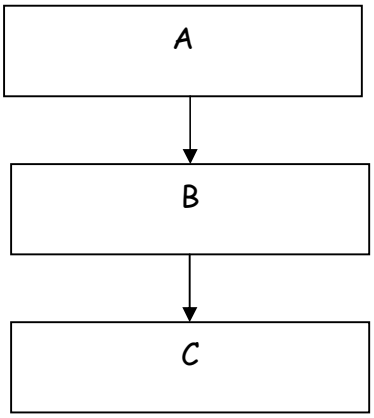
Program:-

```
class rect
{  int l, b;
   rect (int x, int y)
   {       l=x;
           b=y;
   }
   void area()
   {       System.out.println("Length=" + l);
           System.out.println("Breadth=" + b);
           System.out.println("Rectangle Area=" + (l*b));
   }
};
class      cube   extends rect
{  int h;
   cube   (int x, int y, int z)
   {
           super (x, y);
           h=z;
   }
   void volume()
   {
           System.out.println("Height=" + h);
           System.out.println("Cube Volume=" + (l*b*h));
   }
};
class single1
{
   public static void main(String args[])
   {       cube   c1 = new cube(10,20,30);
           c1.area();
           c1.volume();
   }
}
```

Multilevel Inheritance:- A new sub class is derived from another sub class is called as Multilevel Inheritance. The class **A** serves as a super class for the sub class **B**, It serves as a super class for the sub class **C**. The class **B** is known as intermediate super class. It provides a link for the inheritance between **A** and **C**.

Syntax: -

```
class  A
{      instance variables;
       instance methods;
};
class  B      extends      A
{      instance variables;
       instance methods;
}
class  C      extends      B
{      instance variables;
       instance methods;
}
```

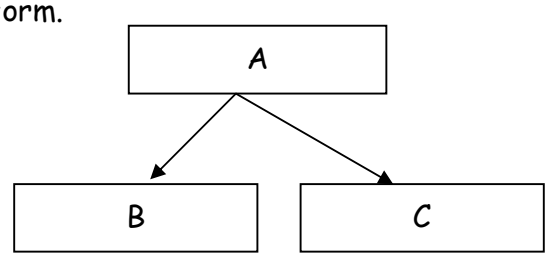


Example:-

```
class A
{
        int a;
        A(int x)              // super class constructor
        {
                a=x;
        }
        void dispA()
        {
                System.out.println("A=" + a);
        }
}
class B extends A
{       int b;
        B (int x, int y)
        {
                super(x);
                b=y;
        }
        void dispB()
        {
                System.out.println("B=" + b);
        }
}
```

```
        class C extends B
        {
        int c;
        C(int x,int y,int z)
        {
                super(x,y);
                c=z;
        }
        void dispC()
        {
                System.out.println("C=" + c);
        }
}
class multilevel
{
        public static void main(String arg[])
        {
                C c1=new C(10,20,30);
                c1.dispA();
                c1.dispB();
                c1.dispC();
        }
};
```

Hierarchical Inheritance:- Two or more sub classes are derived from only one super class is called as Hierarchical Inheritance. Inheritance is to use as a support to the hierarchical design of a program. Many programming problems can be displayed into a hierarchical form.



Example:-

```
class  A
{       instance variables;
        instance methods;
}
class  B extends    A
{       instance variables;
        instance methods;
}
class C extends     A
{       instance variables;
        instance methods;
}
```

Program:-

```
class A
{       int a;
        A(int x)
        {       a=x;
        }
}
class B extends A
{       int b;
        B(int x, int y)
        {       super(x);
                b=y;
        }
        void sum()
        {
                System.out.println("a=" + a);
                System.out.println("b=" + b);
                System.out.println("sum="+(a+b));
        }
}
class C extends A
{
        int c;
        C(int x,int y)
        {       super(x);
                c=y;
        }
        void mul()
        {       System.out.println("c=" + c);
                System.out.println("Mul="+(a*c));
        }
}
class hiearchical
{
        public static void main(String arg[])
        {
                B b1=new B(30,40);
                C c1=new C(10,20);
                b1.sum();
                c1.mul();
        }
};
```

Overriding Methods:-

- A method in the subclass that has the same name, same arguments and same return types as a method in the super class.
- Then the method defined in the sub class is executed instead of the one in the super class. This is known as overriding.
- It is the run-time polymorphism.

Program:-

```
class A
{
        int a;
        A(int x)
        {
                a=x;
        }
        void display()
        {
                System.out.println("Super Class");
                System.out.println("a=" + a);
        }
};
class B extends A
{
        int b;
        B(int x,int y)
        {
                super(x);
                b=y;
        }
        void display()
        {       System.out.println("Sub class");
                System.out.println("a=" + a);
                System.out.println("b=" + b);
        }
};
class  overriding
{
        public static void main(String[] args)
        {
                B  b1 = new B(10,20);
                b1.display();
        }
}
```

Final Variables:-

- A variable can declare as **final** then the variable is called final variable.
- The value of a final variable can never be changed.
- It is looks like a constant.
- Final variables behave like class variables and they do not take any space on individual objects of the class.

Syntax:- final datatype variablename = value;

Example:- final int SIZE=100;

Final Method:-

- A method can be declared as final then this method is called final method.
- All methods and variables can be overridden by default in subclasses.
- Final methods are used to prevent the subclasses from overriding the members of the super class.

```
Syntax:-    class A
            {
                int a=10;
                final void display()        // final method
                {
                    System.out.println("a=" + a);
                }
            }
            class B extends A
            {
            int b=20;
                void display()      // overriding concept
                {
                    System.out.println("b=" + b);
                }
            }
```

Then this program may give compile time error. Final method can not be overridden.

Final Class:- A class that cannot be subclassed is called a final class. Final class prevents any unwanted extensions to the class. Final class prevents the inheritance.

```
Example:-   final class A           //    final class
            {           body of the class;
            }
            class B     extends     A
            {           body of the class;
            }
```

Abstract Method:-

- A method can be declared as abstract then this method is called abstract method.
- It is declared with in the **super class** and defined only in **sub class**.
- These methods are executed in subclass only.
- A class can contain two or more abstract methods are called as abstract class.
- We can not create objects to the abstract class.

Arrays:- An array is a group of same data type elements are stored in line by line in one variable. Arrays may be two types. They are

- ⊙ Single dimensional arrays
- ⊙ Double dimensional arrays

One Dimensional Array:- A list of items can be given one variable name using only one subscript is called a one dimensional array.

Like any other variables, arrays must be declared and created in the computer memory before they are used. Creation of an array involves three steps.

1. Declaring the array
2. Create memory allocations
3. Initialize the array

1. Declaring the array:- An array may be declared in two forms. We do not enter the size of the arrays in the declaration.

Syntax:- **datatype arrayname [ ];** ⟶ Form 1

Syntax:- **datatype [ ] arrayname;**

Example:- int a [ ] or int [ ] a;

2. Creation of Array:- After declaring an array, we need to create memory space for array. Java allows us to create memory allocation arrays using **new** operator only.

Syntax:- **arrayname = new datatype [size];** ⟶ Form 2

Example:- **a = new int [10];**

It is also possible to combine these two steps declaration and creation into one.

Syntax:- datatype arrayname [ ] = new datatype [10];

Example:- int a [ ] = new int [10];

3. Initialization of Arrays:- we can put values into an array is called initialization. This is done using the array subscripts.

Syntax:- arrayname[subscript] = value

Example:- a[0] = 10; a[1] = 20.

Java creates an array starting with a subscript 0 and ends with a value one less than the size specified. Unlike C, Java protects arrays from overruns and underruns. We can access an array beyond its boundaries will generate an error messages. We can also initialize arrays automatically same as the normal variables.

Syntax:- datatype arrayname [ ] = { list of values };

Syntax:- int a [ ] = { 10, 20,30, 40};

In Java we can access the length of the array by a using the **arrayname.length. Example:- a. length**

Two dimensional arrays:- An array declaration can have multiple sets of square brackets. Two-dimensional array are stored in memory. These are like a matrix consisting rows and columns. Each row & column represents the values from 0 to size-1. Two-dimensional arrays are created like one-dimensional array.

Syntax:- datatype arrayname[ ] [ ] = new datatype [size1] [size2]

Example:- int a [ ] [ ] = new int [10][10]

We can initialize two dimensional arrays same as the like one dimensional arrays.

Example:- int a[2][3] = {0, 0, 0, 1, 1, 1}

It is initializes the elements of the first row to zero and the second row to one. The initialization is done by row by row.

STRINGS:-

- Strings represent a sequence of characters.
- In Java string is not a character array and is not NULL terminated.
- In Java, strings are class objects and implemented using two classes, **String** and **StringBuffer**.
- A Java string is an instantiated object of the **String** class.
- Java strings are more efficient than C strings.

a) String:-

- It creates strings of fixed length.
- We can not modify the length and content.
- We can not insert characters and substrings in the middle of a string.

Java strings may be declared as

Syntax:- String stringname; ⟶ (1)

Stringname = new String ("String1"); ⟶ (2)

Example:- String S1;

S1 = new String (" Tarakesh");

These two statements may be combined as

Syntax:- String stringname = new String(" string1");

Example:- String s1 = new String("Tarakesh");

Java strings can be concatenated using the " + " operator. We can also create and use arrays that contain strings. The statement is

Syntax:- String stringname [ ] = new String [size]

Example:- String s1[ ] = new String [5]

It will create an string array **s1** of size 5 to hold five string constants.

String Methods:- The String class defines a number of methods that allow us to manipulate the strings. String s1 = new String ("Tarakesh");

1. **toLowerCase () :-** It converts all the characters into lowercase letters.
   Syntax:- System.out.println(s1. toLowerCase ()) **O/P:- tarakesh**
   It converts the string s1 to all lowercase letters.
2. **toUpperCase ( ):-** It converts all the characters into Uppercase letters.
   Syntax:- System.out.println(s1. toUpperCase()) **O/P:- TARAKESH**
   It converts the string s1 to all uppercase letters.
3. **replace(char1,char2):-** It replaces all characters char1 to char2. Syntax:-
   s1. replace('a', 'A'); It converts all 'a' s to 'A' s.
   output:- t**A**r**A**kesh
4. **trim ():-** It removes white spaces at the beginning and end of the string.
   Syntax:- s1.trim()
5. **equals (string):-** It checks the two strings are equal or not. It returns **true** if the two strings are equal otherwise **false**. (Lower & upper case are different) Syntax:- s1. equals (s2)
6. **equalsIgnoreCase (string):-** It checks the two strings are equal or not and ignoring the case of characters. (Lower & Upper case letters are same)
   Syntax:- s1.equalsIgnoreCase(s2)
7. **length( ):-** It counts/gives the length of the string.
   Syntax:- s1.length()
8. **charAt (n):-** It displays a character depending on position (displays the **n**th character). Syntax:- s1.charAt(3);
   It displays 'a'. 4th character in the string "tarakesh"
9. **concat(string):-** It concatenates one to another string at end.
   Syntax:- s1.concat(s2) It concatenates s1 and s2.
10. **compareTo(string):-** It compares the two strings.
    It returns positive (>0) value if string1 > string2,
    It returns negative (<0) value if string1 < string2,
    and it returns zero (=0) if string1 equals string2.
    Syntax:- s1.compareTo(s2); It returns positive value if s1 > s2.
11. **substring (n):-** It extracts/gives some portion of a string starting from nth character to ending position.
    Syntax:- s1.substring(3) It displays 4th character to ending character.

12. **substring (n, m):-** It gives a substring starting from n^{th} character up to m^{th} character (not including m^{th}). S1 = new String("Tarakesh")

Syntax:- s1.substring(2,5). It displays "**rak**".

13. **indexOf(char):-** It gives the position of character of first occurrence in a string. Syntax:- s1.indexOf('a') O/P:- 1

14. **lastIndexOf(char):-** It gives the position of character of last occurrence in a string. s1="Tarakesh"

Example:- s1.lastIndexOf('a'); o/p:- 3

14. **toCharArray():-** This method is used to convert string object values into a character array.

Example:- char s2[] = s1.toCharArray();

```
// STRING SORTING
import java.io.*;
class strsort
{       public static void main(String[] args) throws IOException
        {       int i, n, j;
                String s1[] = new String[10];
                String t=null;
                DataInputStream di = new DataInputStream (System. in);
                System.out.println("Enter the no of strings:");
                n=Integer.parseInt(di.readLine());
                System.out.println("Enter the Strings line by line");
                for(i=0;i<n;i++)
                        s1[i]= di.readLine();
                System.out.println("Printing the Strings ");
                for(i=0;i<n;i++) System.out.print(s1[i] + "    ");
                for(i=0;i<n;i++) System.out.print(s1[i] + "    ");
                for(i=0;i<n;i++)
                {       for(j=i+1;j<n;j++)
                        {       if (s1[i].compareTo(s1[j]) >0)
                                {       t = s1[i];
                                        s1[i] = s1[j];
                                        s1[j] = t;
                                }
                        }
                }
        System.out.println("\n The Sorted Strings are:");
        for(i=0;i<n;i++) System.out.print(s1[i] + "    ");
        }
}
```

StringBuffer:-

- It is a peer class of **String**.
- It creates strings of flexible length that can be modified in length and content.
- We can insert characters and substrings in the middle of a string, or append another string to the end.

We can create a string by this method is

Syntax:- StringBuffer stringname = new StringBuffer("string1");

Example:- StringBuffer s1 = new StringBuffer("Ramesh");

StringBuffer Methods:-

1. setCharAt(n,'x'):- It modifies the n^{th} character to 'x'.

Example:- s1.setCharAt(2,'k'); s1="Ramesh"

It modifies 3^{rd} character to 'k' in string s1. O/P:- Rakesh

2. append(string):- It adds the string at the end of the first string.

Example:- s1.append("Babu"); s1="Ramesh"

"Babu" is added at the end of the s1. O/P:- RameshBabu

3. insert(n,string):- It inserts the new string at the particular position of the string. s1="Rama Rao"

Example:- s1.insert(4,"krishna") o/p:-Ramakrishna Rao

It inserts the string "krishna" at the 4^{th} position of the s1.

4. reverse():- It prints the string in reverse order. S1="RAMA"

Example:- s1.reverse() o/p:-AMAR

5. setLength(n):- It set the length of the string s1 to n. If n < s1.length () s1 is truncated. If n > s1.length () spaces are added to s1.

Example:- s1.setLength(15)

6. delete(start, end):- It deletes the character from **start** to **end-1** position. S1="RameshBabu"

Example:- s1.delete(3,6) o/p:-RamBabu

7. deleteCharAt (pos):- It deletes the specified character in a specified position within a string. s1="Ramamohan"

Example:- s1.deleteCharAt(3); o/p:-Rammohan

V E C T O R S:-

- Vector is also a predefined class.
- It is used to create a generic dynamic array.
- It can hold objects of any type and any number.
- The objects do not have to be homogeneous (i.e same type & same size).
- Vector methods are contained in the **java.util** package.

A vector can be declared without specifying any size explicitly. A vector can have an unknown number of items. When a size is specified, we can inserts any number of items may be put into the vector. We can create a vector

Syntax:- Vector vector_name = new Vector( )

Example:- Vector list = new Vector( );

Vector has many advantages then arrays. They are

- ⊙ It is convenient to use vectors to store objects.
- ⊙ A vector can be used to store a list of objects that may vary in size.
- ⊙ We can add and delete objects from the vector.

Drawback:- We can not directly store simple data types in a vector. We need to convert simple data types to objects. This can be done using the wrapper classes.

The vector class supports a number of methods can be used to manipulate the vectors created. They are

1. **list. addElement (item):-** It adds the item specified to the list at the end.
   Example:- list.addElement("Tarak");
2. **list.elementAt(pos):-**It gives the name of the object depending on position.
   Example:- list.elementAt(4)
3. **list. size():-** It gives the total number of objects in a vector.
   Example:- list. size()
4. **list. removeElement(item):-** It removes the specified element from the vector. Example:- list. removeElement("tara");
5. **list. removeElementAt(n):-** It removes the item stored in the n^{th} position of the list. Example:- list. removeElementAt(3)
6. **list. removeAllElements():-** It removes all the elements in the vector.
   Example:- list.removeAllElements()
7. **list. copyInto(array):-** It copies all items from vector to array.
   Example:- list.copyInto(a1)
8. **list. insertElementAt(item, n):-** It inserts the specified item at the n^{th} position of the vector.

Wrapper Classes:-

- Vectors can not handle primitive data type like **int, float, long, char** and **double**.
- Primitive data type may be converted into object types by using wrapper classes.
- Wrapper classes are contained in the **java.lang** package.

Wrapper classes for converting simple data types.

Simple Datatype	Wrapper Class
⊙ boolean	Boolean
⊙ char	Character
⊙ double	Double
⊙ float	Float
⊙ int	Integer
⊙ long	Long

1) Converting primitive numbers to Object numbers using constructor method.

- Integer intval = new Integer (i); Primitive integer to integer object
- Float floatval = new Float(f); Primitive float to float object
- Double doubleval = new Double (d) Primitive double to double object
- Long longval = new Long(l); Primitive long to long object

2) Converting object number to primitive number by using **typeValue ()**

int i = intval. intValue() converts integer object to primitive integer
float f = floatval. floatValue() converts float object to primitive float
long l = longval. longValue() converts long object to primitive long
double d = doubleval. doubleValue() converts double object to primitive double

3) Converting number to String using **toString ()** method

str = Integer. toString(i) Primitive integer to string
str = Float. toString(f) Primitive float to string
str = Long. toString(l) Primitive long to String
str = Double. toString(d) Primitive double to string

4) Converting string object to numeric object using the static method **valueOf()**

Dv = Double. valueOf(str) Converts string to Double object
Fv = Float. valueOf(str) Converts string to Float object
Iv = Integer. valueOf(str) converts string to Integer object
Lv = Long. valueOf(str) converts string to Long object

5) Converting Number String to Primitive numbers using **parseType()** Methods

int i = Integer. parseInt(str) converts string to primitive integer
float f = Float. parseFloat(str) converts string to primitive float
long l = Long. parseLong(str) converts string to primitive long
double d = Double. parseDouble(str) converts string to primitive double

I N T E R F A C E

- An interface is looks like a class.
- It also contains methods and variables but with a major difference.
- The difference is that interface defines only abstract methods and final fields.
- Abstract method is a method declared in a super class and always redefined in a subclass.
- Final variables are acts like constants.
- It supports multiple inheritances in java.

The syntax for defining an interface is very similar to a class.
The general form of an interface definition is

```
syntax:-    interface    interfacename
            {
                  variables declaration;
                  methods declaration;
            }
Example:-   interface    item
            {
                  static final int a=20;
                  void display();
            }
```

The method definition is not included in the interface. The method declaration ends with a semicolon. The class that implements this interface must define the code for the method.

Like classes, interfaces can also be extended. An interface can be sub interfaced from other interfaces. The new sub interface will inherit all the members of the super interface.

```
Syntax:-    interface    name1        extends      name2
            {
                  body of the interface
            }
```

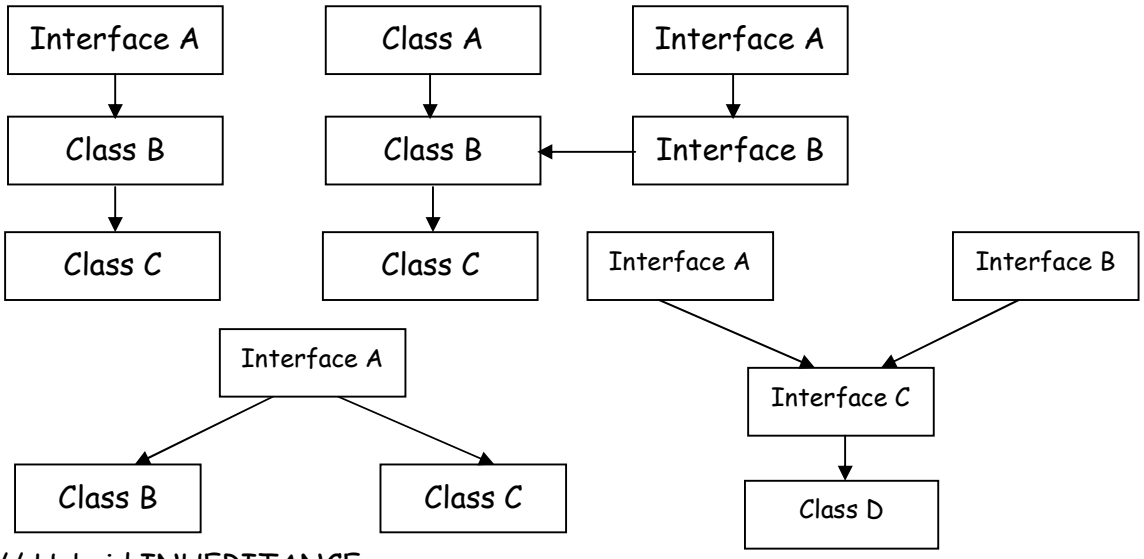
We can also combine several interfaces together into a single interface.

```
Syntax:-    interface    name1
            {     body of interface1        }
            interface    name2
            {     body of interface2        }
            interface    name3        extends      name1, name2
            {
                  body of interface3;
            }
```

Interfaces are used as **super class,** these properties are inherited by classes. A class inherits the properties of interface by using **implements** keyword. When a class implements more than one interface, they are separated by a comma.

Syntax:- **class** classname **extends** superclass **implements** interface
{
body of class name;
}

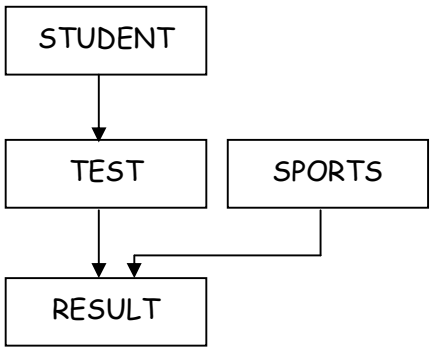
Various forms of interface implementation




```
// Hybrid INHERITANCE
class stud
{       int sno;
        void getstud(int x)
        {       sno=x;
        }
        void dispstud()
        {
        System.out.println("sno="+sno);
         }
}
class marks extends stud
 {       int mm,pm,cm;
         void getmarks(int x, int y, int z)
         {       mm=x;
                 pm=y;
                 cm=z;
         }
```

```
        void dispmarks()
        {
        System.out.println("mm="+mm);
        System.out.println("pm="+pm);
       System.out.println("cm="+cm);
        }
}
interface pmarks
{       int p1=40,  p2=50;
        void disppmarks ();
}

class result extends marks implements pmarks
{       int total;
public void disppmarks()
{
System.out.println("p1="+p1);
System.out.println("p2="+p2);
}
void display()
{       total=mm+pm+cm+p1+p2;
        dispstud();
       disppmarks();
        disppmarks();
       System.out.println("total="+total);
}
}
class multipul
{
public static void main(String arg[])
{       result r1=new result();
        r1.getstud(12345);
        r1.getmarks(70,80,50);
        r1.display();
}
}
```



Package:-

- It is a looks like a Directory/Folder.
- It can store a variety of classes and/or interfaces or methods together.
- Packages are acts as containers for classes.
- It is a concept similar to "class libraries" in other languages.

Benefits of packages

- The classes contained in the package of other programs can be easily reused.
- Packages provide a way to **hide** classes thus preventing other programs or packages from accessing classes.

Packages are mainly divided into two parts.

They are **Application Packages** and the **user-defined packages.**

Java API provides a large number of classes grouped into different packages. They are

- **Language Packages:-** It is a collection of classes and methods required for implementing basic features of Java. It contains **Mathematical functions, String classes and functions, wrapper classes** etc.
  Ex:- java.lang
- **Utility Packages:-** It is a collection of classes and methods to provide utility functions such as **vectors, hash tables, data & time** functions and classes.
  Ex:- java.util
- **Input/Output Packages:-** It is a collection of classes and methods are required for input/output manipulations. They provide facilities for the input and output of data.
  Ex:- java.io
- **Networking Packages:-** It is a collection of classes for communication with other computers. These classes are used in networking. They include classes for communicating with local computers as well as with internet servers.
  Ex:- java.net
- **AWT Packages:-** AWT means Abstract Window Toolkit. It is collection of classes and methods that implements Graphical User Interface applications. They include classes for windows, buttons, lists, menus and so on.
  Ex:- java.awt
- **Applet Packages:-** This include a set of classes and methods that allows us to creating and implementing Java Applets.
  Ex:- java.applet

We can access the classes stored in a package in two ways. The first approach is to use the fully qualified class name of the class that wants to use.

The package name containing the class and then appending the class name to it by using the dot operator. For example, the **Color** in the **awt** package.

Syntax:- import **packagename. Classname;**

Example:- import java. awt. Color;

In many situations, we want to use a class in a number of places in the program or we use many classes contained in a package.

Syntax:- **import** **packagename. *;**

Example:- import java. Awt. *;

Creating User-Defined packages:- We can create our own user-defined packages involving the steps.

- Declare the package at the beginning of a file using the form.

  Syntax:- **Package** package name;
- Define the class is to be put in the package and declare it **public.**
- Create a subdirectory under the directory where the main source files are stored.
- Store the listing as the **classname.java** file in the subdirectory created.
- Compile the file. This creates **.class** file in the subdirectory.
- The sub-directory name must match the package name exactly.

```
Syntax:-  package     packagename;
          public class classname
          {     body of the class;
          }
Example:- package     p1;
          public class A
          {     public void dispA()
                {
                      System.out.println("Class A");
                }
          }
```

This source file should be named **A. java** and stored in the sub-directory **p1**. Now compile this java file. The resultant **A.class** will be stored in the same sub-directory **p1**.

```
Example:- import p1. *;
          Class pkgtest
          {     public static void main(String arg[])
                {
                      A  s1 = new  A();
                      s1.dispA();
                }
          }
```

This program imports the **A.class** from the package **p1.** The source file should be saved as **pkgtest. java** and then compiled. The source file and compiled file would be saved in the directory **p1.**

Importing classes from other packages:-

```
        Package p2;
        Public class B
        {       public void dispB()
                {
                        System.out.println("Display A");
                }
        }
import p1.A.
import p2.B;
class pkg
{       public static void main(String arg[])
        {       A a1 = new A();
                B b1 = new B();
                a1.dispA();
                b1.dispB();
        }
}
```

Sub classing an import class:-

```
import p1.A;
class          C       extends       A
{
void dispC()
{
System.out.println("Display Class C");
}
}
class pkgtest
{
public static void main(String arg[])
{
C       c1      =       new    C();
a1.dispA();
c11.dispC();
}
}
```

Visibility Control:- Java provides 5 types of visibility control. These are **public, protected, friendly access (default), privateprotected** and **private.**
Public Access:- It is visible only to the entire class, subclasses and other classes in the same package and other packages in same class and subclasses. It is accessible to every where in the java.
Syntax:- **public int a;** **public void display()**

Protected:- It is visible only to all classes and subclasses in the same package and also to subclasses in other package. Non-subclasses in other package can not access the **protected** variable.
Syntax:- protected int c; protected void display()

Friendly access:- It is the default access specifier in Java. It is accessed in the same package only. It is accessed in the same classes, subclasses and other classes in the same package.
Syntax:- int a; void big()

Private protected:- It is declared with two keywords **private** and **protected.** It gives the visibility level in between the **protected** and **private**. It is be visible only in subclasses in same packages and other packages.
Syntax:- private protected int a;

Private:- It is highest degree of protection. They are accessible only within their own class. They cannot be inherited.
Syntax:- private int a; private void display()

	Public	Protected	Friendly	Private Protected	Private
Same class	Yes	Yes	Yes	Yes	Yes
Sub class In same package	Yes	Yes	Yes	Yes	No
Other classes in same package	Yes	Yes	Yes	No	No
Subclass in other package	Yes	Yes	No	Yes	No
Other classes in other package	Yes	No	No	No	No

MULTITHREADED PROGRAMMING

- **Thread** is similar to a java program that has a single flow of control.
- It has beginning, a body, and an end, and executes commands sequentially.
- Multithreading is a program is divided into two or more subprograms, which can be implemented or executed at the same time in parallel.
- For example, one subprogram can display an animation on the screen while another may build the next animation to be displayed.

Java allows us to use multiple flows of control in developing programs. Each flow of control a separate small program known as thread. A program that contains multiple flows of control is known as multithreading.

Threads are extensively used in Java enabled browsers such as Hotjava.

Creating Threads:-

- Threads are implemented in the form of objects that contain a method called **run( )**.
- The **run ( )** method is the heart and soul of any thread.
- The **run()** method should be invoked by an object of the thread.
- The **run()** method invoked with the help of another thread method called **start()**.

A new thread can be created in two ways.

1. **By creating a thread class:-**

   It define a class that extends **Thread** class and executes **run()** method.

   - Creating a new class extending the **Thread** class.
   - Creating a **run()** method in a class preceded by **public** keyword.
   - Creating a object to the thread class.
   - Accessing the **run()** method using **start()** method.

Syntax:-

```
class classname extends Thread
{
        public void run()
        {       }
}
```

Example:-

```
class A extends Thread
{       public void run()
        {
                body of the run
        }
}
```

Example:-

```
class A extends Thread
{
        public void run()
        {
                for(int i=1;i<=5;i++)
                {
                        System.out.print("thread A="+i);
                }
        }
}
class Threadtest
{
        public static void main(String mdv[])
        {
                A a1=new A();
                a1.start();

        }
}
```

2. **By converting a class to a thread**:- Define a class that implements **Runnable** interface. The **Runnable** interface has only one **run()** method.

Syntax:-

```
class classname implements Runnable
{   public void run()
    {       }
}
```

- Declare the class as implementing the **Runnable** interface.
- Implement the run() method.
- Create a thread by defining an object, instantiated from this Runnable
- Call the threads **start ()** method to run the thread.

We can create and run an instance or object of our thread class.

Syntax:- classname objectname = new classname()

Example:- A a1 = new A();

We can invoke or execute the run method by using start()

Syntax:- Thread_object.start();

Example:- a1.start()

Example:-

```
class A implements Runnable
{
        public void run()
        {
                for(int i=1;i<=5;i++)
                {
                        System.out.println("i="+i);
                }
        }
};
class  ttest1
{
        public static void main(String[] args)
        {
                A a1=new A();
                Thread a2=new Thread(a1);
                a2.start();
        }
}
```

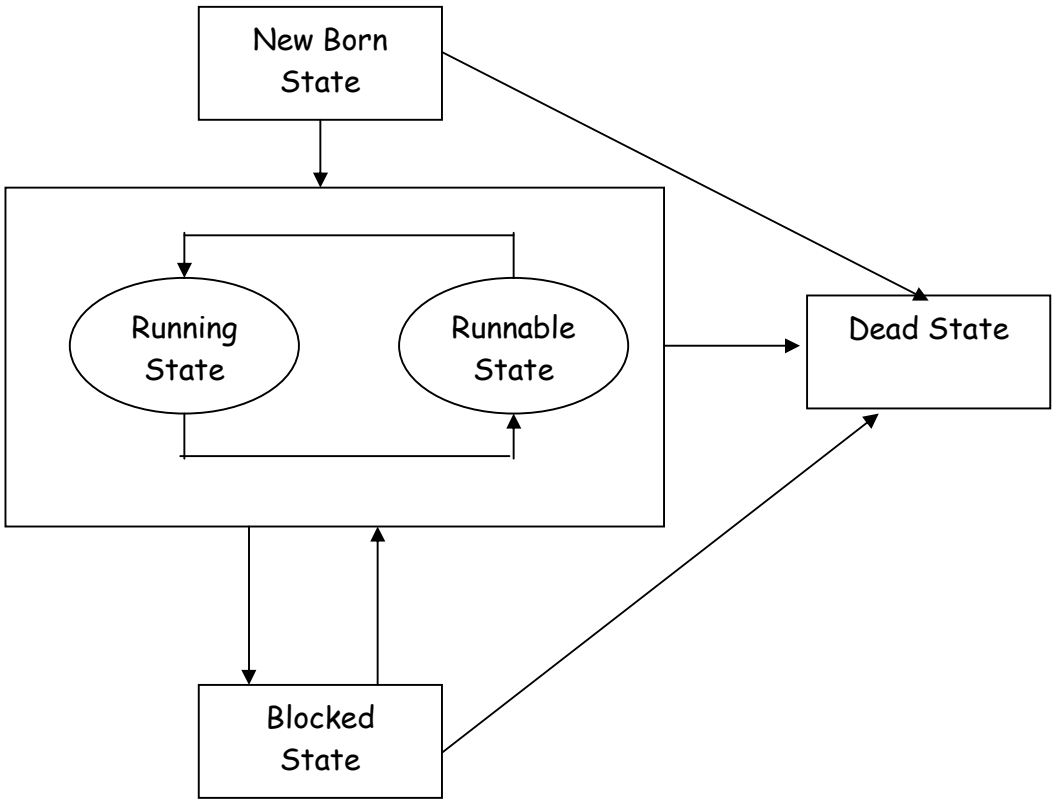
Program:-

```
class A       extends Thread
{       public void run()
        {
                for(int i=1;i<=5;i++)         System.out.println("Thread A: i="+ i);
        }
};
class B       extends Thread
{
        public void run()
        {
                for(int j=1;j<=5;j++)         System.out.println("Thread B: j="+ j);
        }
};
class C       extends Thread
{
        public void run()
        {       for(int k=1;k<=5;k++)
                        System.out.println("Thread C: k="+ k);
        }
};
class multithread
{       public static void main(String[] args)
        {       A       a1      =       new     A();
                B       b1      =       new     B();
                C       c1      =       new     C();
                a1.start();
                b1.start();
                c1.start();
        }
}
```

Life Cycle of a Thread:- The life time of a thread can have 5 states. They are

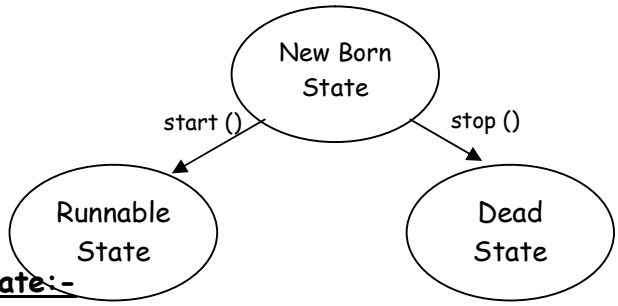
1. New born state
2. Runnable state
3. Running state
4. Blocked state
5. Dead state

A thread is always in one of these 5 steps. Thread can move from one state to another.



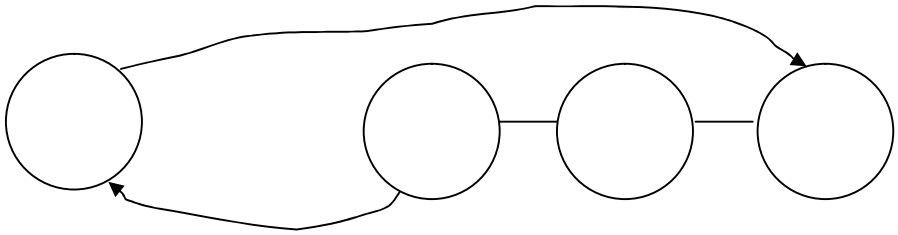
Newborn state:-

- We create a new thread; the thread is born and is in newborn state.
- The thread is not ready for running.
- It can be schedule for running using **start ()** method.
- It can be Kill by using **stop ()** method.
- The thread was scheduled it will move to the Runnable state.
- The thread was killed it will move to Dead state.



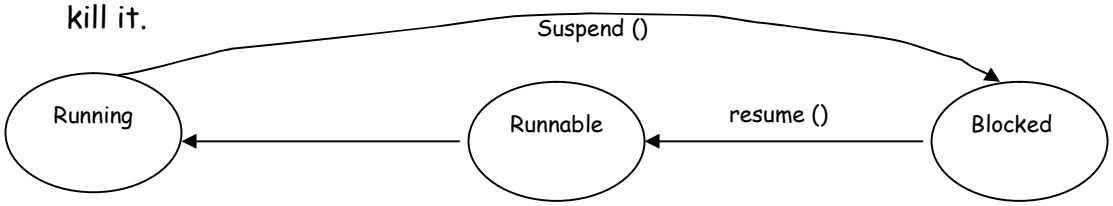
Runnable State:-

- It means the thread is ready for execution and is waiting for the availability of the processor.
- The threads are joined in the **queue**.
- Threads are executed/scheduled depends their priorities.
- High priority threads are executed first than lower priorities threads.
- If all the threads have equal priority then they are executed in round robin fashion i.e. first-come, first serve.

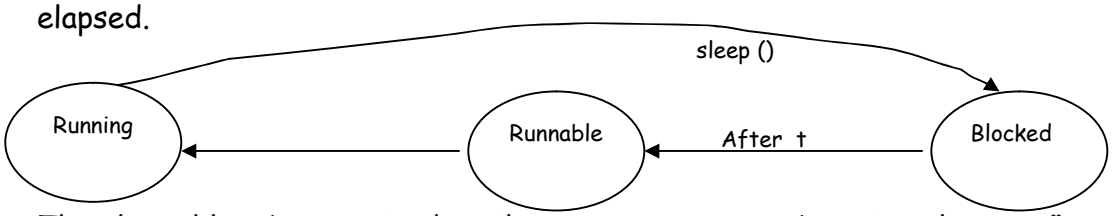


Running State:-

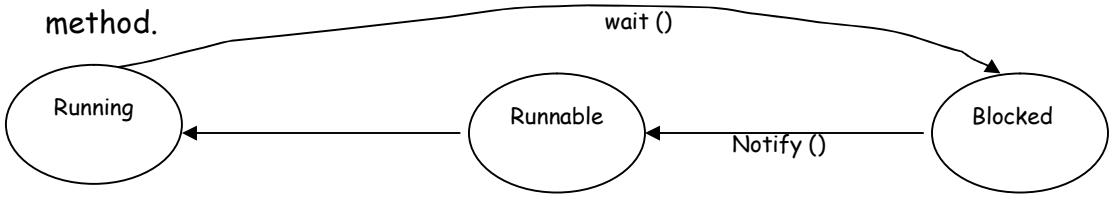
- It is the state that the processor has given its time to the thread for its execution.
- A higher priority thread runs the thread.
- A running thread may give up its control in one of the situations.
  1. It has been suspended using **suspend()** method. A suspended thread can be revived or recalled by using the **resume()** method. We want to suspend a thread for some time due to some reason, but do not want to kill it.



  2. It has been made to sleep. We can put a thread to sleep for a specified time period using the method **sleep(time)** where time is milliseconds. The thread re-enters the Runnable state as soon as this time period is elapsed.



  3. The thread has been waited until some event occurs by using the **wait()** method. The thread can be scheduled to run again using the **notify()** method.



Blocked State:-

- A thread is said to be blocked when it is not permitted from entering into the Runnable state and the running state.
- This happens when the thread is suspended, sleeping or waiting some time.
- A blocked thread is considered "not Runnable" but not dead and to run again.

Dead State:-

- Every thread has a life cycle.

- A running thread ends its life when it has completed executing its **run()** method.
- It is a natural death.
- A thread can be killed as soon as it is born, or while it is running or even it is in "not Runnable" condition.

Thread Priority:- In Java, each thread is assigned a priority, it affects the order in which it is scheduled for running. The threads of the same priority are given equal preference by the java scheduler. Java permits us to set the priority of a thread using **setPriority ()** method.

Syntax:- threadname.setPriority (number)

The number is an integer value to set the thread priority. The thread class defines several priority constants.

MIN_PRIORITY = 1
NORM_PRIORITY = 5
MAX_PRIORITY = 10

The default setting is NORM_PRIORITY.

EXCEPTION HANDLING:-

An Error may produce an incorrect output or may terminate the execution of the program suddenly or even may cause the system to crash.

Errors may broadly be classified into two categories

- compile time errors
- runtime errors

Compile time errors:- All syntax errors will be detected and displayed by the Java compiler. These errors are known as compile-time errors. Whenever the compiler displays an error, it will not create the .class file. The most common compile-time errors

- Missing semicolons
- Missing brackets in classes and methods
- Misspelling of identifiers
- Missing double quotes in strings
- Use of undeclared variables

Run-time errors:- These errors was occurred at the run time. A program may compile successfully create the **.class** file but may not run properly. Most common run time errors are

- Dividing an integer by zero
- Accessing an element that is out of the bounds of an array
- Trying to store a value into an array of an class type
- Converting invalid string to a number
- Accessing a character that is out of bounds of a string

Exception:-

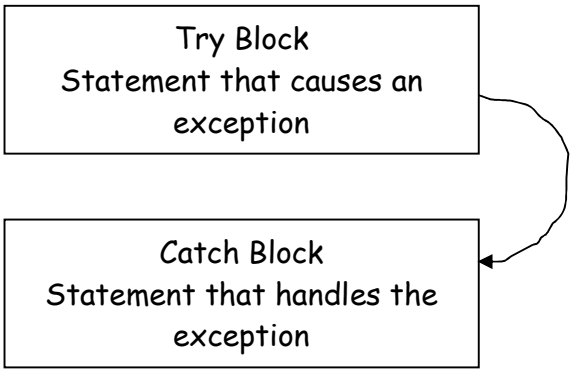
- An exception is a condition that is caused by a run-time error in the program.
- Exception handling mechanism is to provide to detect and report runtime errors, so that appropriate action can be taken.
- This mechanism suggests error-handling code. That performs the some task.
    1. Find the problem
    2. Inform that an error has occurred
    3. Receive the error information
    4. Catch &Take corrective actions.

The error handing code basically consists of two segments, one to detect errors and to throw exceptions and the other to catch exception and to take corrective actions.

PREDEFINED EXCEPTIONS:-

ArithmeticException	It is caused by Mathematical errors such as division by zero
ArrayIndexOutOfBoundsException	Caused by bad array Indexes
ArrayStoreException	Caused a program tries to store the wrong type of data in an array
FileNotFoundException	Caused by an attempt to access a non-existent file
IOException	Caused by General I/O failures
NumberFormatException	Caused when a conversion between string and number fails
OutOfMemoryException	Caused when there is not enough memory to allocate a new object
NullPointerException	It caused by referencing a null object
StringIndexOutOfBoundsException	Caused when a program attempts to access a nonexistent character position in a string
StackOverflowException	Caused when the system runs out of the stack

Syntax of Exception Handling:-



Java uses a keyword **try** to a block of code that is cause an error condition and **throws** an exception. A catch block defined by the keyword **catch** "catches" the exception "thrown by the try block and handles it properly.

```
Syntax:-    try
            {
                body of the try block;
            }
            catch(Exceptiontype      e)
            {
                statement;  }
```

The try block can have one or more statements that could generate an exception. If any one statement generates an exception, the remaining statements in the block are skipped and execution jumps to the catch block is placed next to the try block.

The catch block too can have one or more statements are necessary to process the exception.

Example:-

```
import java.io.*;
class  error
{       public static void main(String    arg[]) throws IOException
        {
        DataInputStream  di=new DataInputStream(System.in);
        int a,b,c,d;
        try
        {
        System.out.println("enter the values a,b & c");
        a = Integer.parseInt(di.readLine();
        b=Integer.parseInt(di.readLine());
        c=Integer.parseInt(di.readLine());
        d= (a / (b-c));
        System.out.println("d=" + d);
        }
        catch(ArithmeticException e)
        {       System.out.println("Divisiable by 0");
        }
}
```

} The program displays an error message; if the (b-c) is equal to zero otherwise it displays the "d" value.

Multiple catch statements:- The try block can have more than one catch block. When an exception in a **try** block is generated, the java treats the multiple **catch** statements like **cases** in a **switch** statement. The first statement matches the exception will be executed and the remaining statements will skip.

Java supports another statement known as **finally** statement that can be used to handle an exception that is not caught by an of the catch statements. This block can be used to handle any exception generated within a try block. It is like a **default** case in a **switch** statement.

Syntax:-

```
try    {
                Body of the try block;
        }
catch(Exceptiontype1      e)
```

```
                {       body of catch1;             }
        catch(Exceptiontype2    e)
                {       body of catch2;             }
        finally
                {       body of the finally block; }
```

Example:-

```
import java.io.*;
class  error1
{
        public static void main(String    arg[]) throws IOException
        {       int n,i;
                int     a[] =new int[10];
                DataInputStream  di=new DataInputStream(System.in);
                try
                {
                System.out.println("enter n value");
                n=Integer.parseInt(di.readLine());
                for(i=0;i<=n;i++)
                        a[i]=Integer.parseInt(di.readLine());
                System.out.println("the printing an array");
                for(i=0;i<=n;i++)
                        System.out.println(a[i]+ "   ");
                }
        catch(ArrayIndexOutofBoundsException   e)
        {
        System.out.println("Array Index Error");
        }
        catch(ArrayStoreException  e)
        {
                System.out.println("Invalid Type stored");
        }
        catch(ArithmeticException   e)
        {
                System.out.println("Invalid Type stored");
        }
        finally
        {
        System.out.println("Unexcept Error");
        }
        }
        }
```

APPLET PROGRAMMING

Applets:-

- Applets are small java programs that are used in Internet computing.
- They can be transported over the Internet from one computer to another.
- It can perform arithmetic operations, display graphics, play sounds, accept user input, create animation and play interactive games.

Local Applet:-

- An applet was developed locally and stored in a local system is known as a local applet.
- When a web page is trying to find a local applet, it does not need to use the Internet.
- It simply searches the directories in the local system and locates and loads the specified applet.

Remote Applet:-

- An applet which was developed by someone and stored on a remote computer connected to the internet. We can download the remote applet onto our system via the Internet and run it.

Applet differ from Applications:- Applets are not full featured application programs.

- Applets do not use the **main ()** method for initiating the execution of the code.
- Applets cannot be run independently. They are run from inside a web page using the HTML tag.
- Applets cannot read from or write to the files in the local computer.
- Applets cannot communicate with other servers on the network.
- Applets cannot run any program from the local computer.
- It does not use native methods.

APPLET LIFE CYCLE:- Every Java applet inherits a set of default behavior from the **Applet** class. The applet states include

- Born or Initialization state
- Running State
- Idle State
- Dead or destroyed state
- Display state

Initialization or Born State:-

- It is the first state in the applet life cycle.
- Applet enters the initialization state when it is first loaded.
- It uses the **init ()** method of **Applet** class, then applet is born.
- At this stage, we can create objects needed by applet, set up initial values, load images or fonts and set of colors.
- This initialization occurs only once in the applet's life cycle.

```
public void init()
{
        body of the action
}
```

Running State: -

- Applet enters the running state when the system calls the **start ()** method of Applet Class.
- This occurs automatically after the applet is initialized.
- Starting can also occur if the applet is already in "stopped" or "idle" state.
- The **start ()** method may be called more than once.

```
public void start ()
{
        body of the action
}
```

Idle or Stopped State:-

- An applet becomes idle when it is stopped from running.
- Stopping occurs automatically when we leave the page containing the currently running applet.
- We can stop the state by using **stop ()** method.
- The **stop ()** method may be called more than once.

```
public void stop ()
{
        body of the action
}
```

Dead State:-

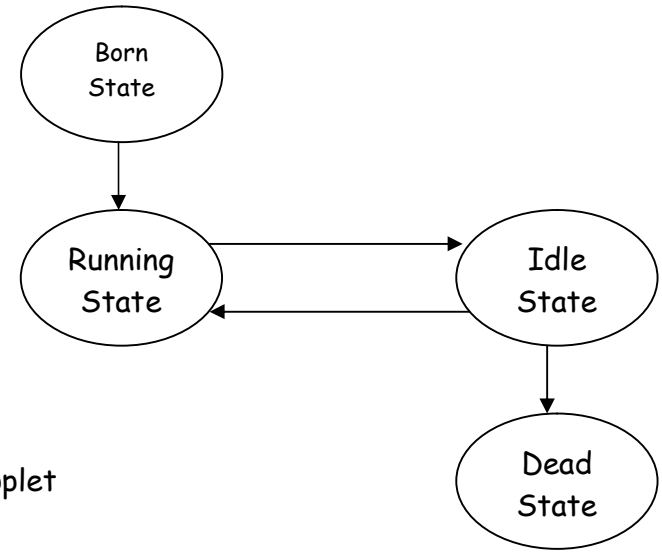
- An applet is said to be dead when it is removed from memory.
- This occurs automatically by invoking the **destroy ()** method when we quit the browser.
- Like initialization, destroying stage occurs only once in the applet's life cycle.

```
public void destroy ()
{
        body of the action
}
```

Display State:-

- Applet moves to the display state whenever it has to perform some output operation on the screen.
- This happens immediately after the applet enters into the running state.
- The **paint ()** method is called to accomplish this task.
- Almost every applet will have a **paint ()** method.

```
public void paint (Graphics g)
{
        body of the action
}
```




```
import java.applet.*;
import java.awt.*;
public class app extends Applet
{
        public void init()
    {   body of initialization;
    }
    public      void start()
    {   body of start
    }
    public      void stop()
    {   body of stop
    }
    public      void destroy()
    {   body of destroy
    }
    public      void paint(Graphics g)
    {   body of display
    }
}
```