D.N.R. COLLEGE (AUTONOMOUS): BHIMAVARAM

DEPARTMENT OF COMPUTER SCIENCE

OBJECT ORIENTED PROGRAMMING WITH JAVA - IV

IV SEMESTER

OBJECT ORIENTED PROGRAMMING WITH JAVA

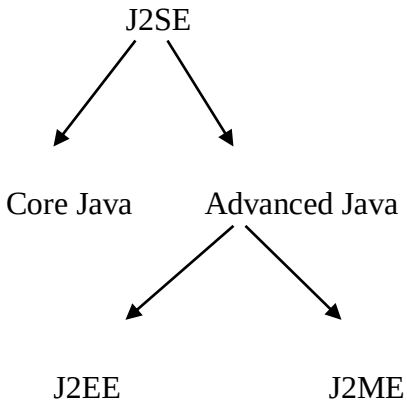
UNIT I

INTRODUCTION TO JAVA

HISTORY OF JAVA:

Java is a powerful versatile programming language for developing software running on mobile devices Java is an object-oriented programming language developed by James Gosling and Colleagues at Sun Microsystems in the early 1990s.

- In 1972 'C' language was developed by Dennis Ritche
- In 1979 C++ was developed by B Jarne Strustrup. It is a combination of both object oriented and procedure oriented language.
- In 1990 java was developed by James Goasling by team James Goasling,Edfrank,Patrick Naughton,Criss worth,Mike Sheridon at Sun Microsystems.
- Originally java called "OAK"
- In 1995 OAK is renamed by Java.
- In the same year JAVA was renamed as J2SE
- J2SE divided into two parts.



1. Core java : Core java is used for design small applications in computer but no use in real.
2. Advanced java : Advanced java is divided into two parts,

   I). J2EE- J2EE is used in internet and web applications

   II).J2ME- J2ME is used for mobile applications.

FEATURES OF JAVA: The following are the features of the JAVA language.

1. Compiled and Interpreted
2. Simple and Small
3. Platform Independent

4. Portable
5. Multithreaded & Interpreted
6. Distributed
7. Robust
8. Secure
9. High Performance
10. Dynamic
11. Object Oriented and these are explained below

1. Compiled and Interpreted

Usually a computer language is either compiled or interpreted. Java combines both these approaches thus Java is a two-stage system.

First, Java compiler translates source code into what is known as byte code instructions. Byte codes are not machine instructions.

In the second stage, the Java interpreter generates machine code that can be directly executed by the system.

Thus Java is both compiled and interpreted language.

2. Simple and Small

Java is very simple and small language. For example, Java does not use pointers, pre-processor header files, goto statement and many others. It also eliminates operator overloading and multiple inheritance.

3. Platform Independent

A program or technology is said to be platform independent if and only if which can run on all available operating systems.

Java code can be run on multiple platforms. Hence JAVA language is treated as platform independent language.

4. Portable

Java supports the feature of Portability. Java programs can be easily moved from one computer to another computer and anywhere.

5. Multithreaded

Multithreaded means handling multiple tasks simultaneously. Java supports multithreaded programs. 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 scrolling a page and at the same time we can download video from a remote computer.

6. Distributed

Java is designed as a distributed language for creating the applications on networks. It has the ability to share both data and programs. Java programs can open and access remote objects on Internet as easily as they can do in a local system.

7. Robust

Java is a most strong language, it provides many securities to make certain reliable code. The languages like C, C++ are treated as week programming languages (Not Robust), since they are unable to deal with runtime errors. Runtime errors are occurred when we enter invalid input.

8. Secure

Java is one of the most secured programming languages. To provide the security to our java real time applications, we need not to write our own security code. Since java library (API) contains readily available security programs for protecting confidential information from unauthorized users. Hence java is one of the powerful secured programming language.

9. High Performance

Java is one of the High Performance programming language because of the following reasons.

- Automatic memory management (Garbage Collector).
- Magic of byte code (Execution of the java application is very faster compared to other programming language applications).
- According to industry experts java programmer performance is high because java programming environment is free from pointers. Hence development of an application takes less time.

10. Dynamic

In any programming language memory can be allocated in two ways. They are

- Static Memory Allocation
- Dynamic Memory Allocation

Java programming does not follow the static memory allocation but it always follows dynamic memory allocation.

Dynamic memory allocation is one in which memory will be allocated at run time. In java programming to allocate the memory space dynamically we use an operator called “new”. “new” operator is known as dynamic memory allocation operator.

11. Object Oriented

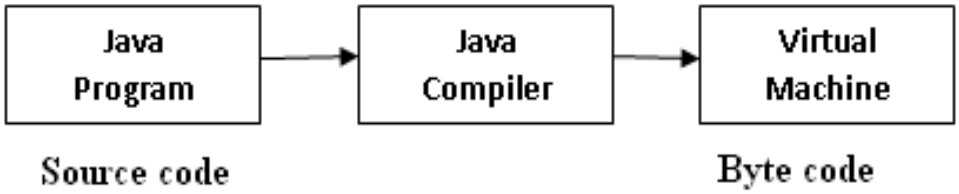
Any programming language that satisfies all the principles of OOP is called as an Object Oriented Programming Language. The Java language satisfies all the principles of OOP. Hence it is called an Object

Oriented Programming Language. Java is a true object oriented language. Almost everything in java is an object.

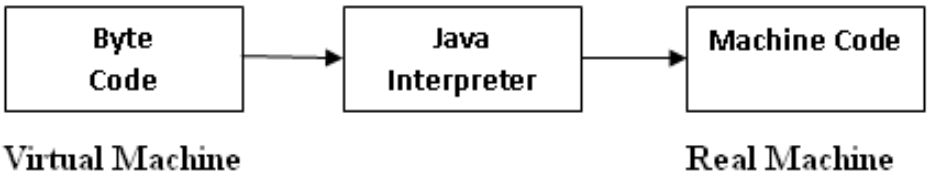
JAVA VIRTUAL MACHINE (JVM):

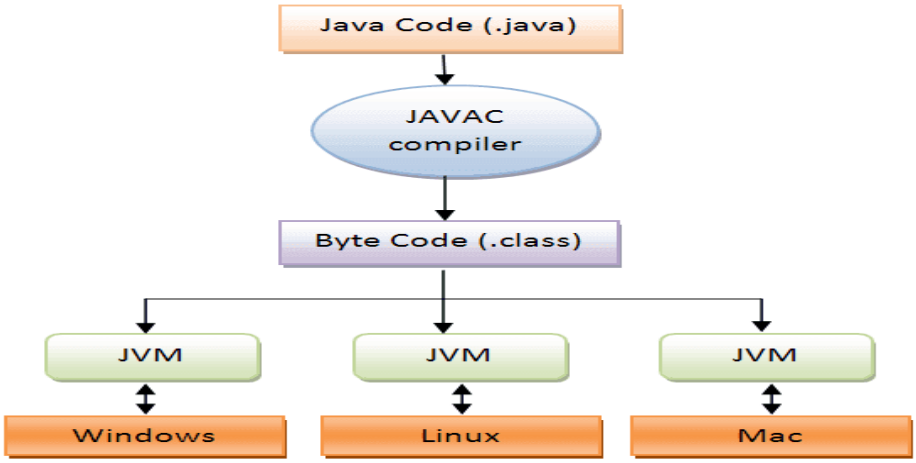
JVM is the heart of the entire Java program execution process. It is responsible for taking the **.class** and converting each byte code instruction into the machine language instruction that can be executed by the microprocessor.

First of all, the **.java** program is converted into a **.class** consisting of byte code instructions by the java compiler for a machine called Java Virtual Machine as shown in below. The JVM exists only inside the computer memory.



The byte code is not a machine specific code (machine code). The machine code is generated by the Java Interpreter by acting as an intermediary between the virtual machine and the real machine as shown in below.





PARTS OF JAVA: There are three main components of Java language: Java Virtual Machine(JVM), Java Runtime Environment(JRE) and Java Development Kit(JDK) respectively.

1. Java Virtual Machine

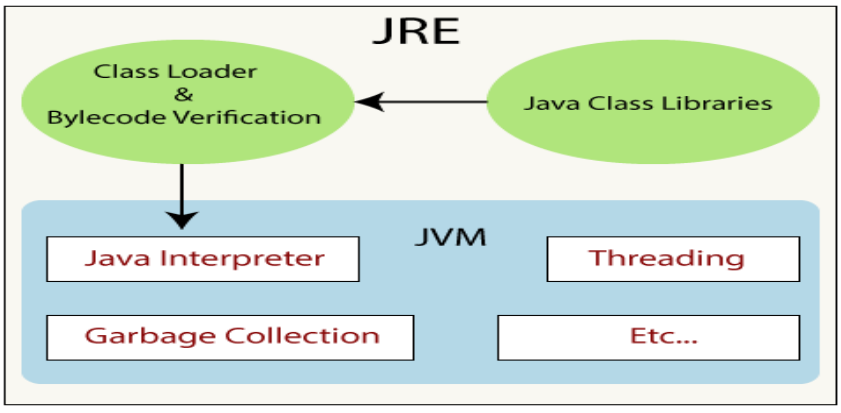
JVM is the heart of the entire Java program execution process. It is responsible for taking the **.class** and converting each byte code instruction into the machine language instruction that can be executed by the microprocessor.

First of all, the **.java** program is converted into a **.class** consisting of byte code instructions by the java compiler for a machine called Java Virtual Machine as shown in below. The JVM exists only inside the computer memory.

The byte code is not a machine specific code (machine code). The machine code is generated by the Java Interpreter by acting as an intermediary between the virtual machine and the real machine as shown in below.

2. Java Runtime Environment: The Java Runtime Environment (JRE) facilitates the execution of java programs. It primarily comprises of the following:

- **Java Virtual Machine (JVM):** It is a program that interprets the intermediate Java byte code and generates the desired output.
- **Runtime class libraries:** These are a set of core class libraries that are required for the execution of java programs.
- **User interface tool kit:** AWT and Swing are example of toolkits that support varied input methods for the users to interact with the application program.
- **Deployment technologies:** JRE comprises the following key deployment technologies.
    - **Java plug-in:** Enables the execution of a Java applet on the browser.
    - **Java Web Start:** Enables remote-deployment of an application.



3. Java Development Kit(JDK): The Java Development Kit comes with a collection of tools that are used for developing and running Java programs. They include

a. **Applet viewer:** It is used to run Java applets.

b. **javac (Java Compiler) :** The Java compiler is used to translates Java source code into byte code files that the interpreter can understand.

c. Java (Interpreter) : It is used to translate byte code into object code

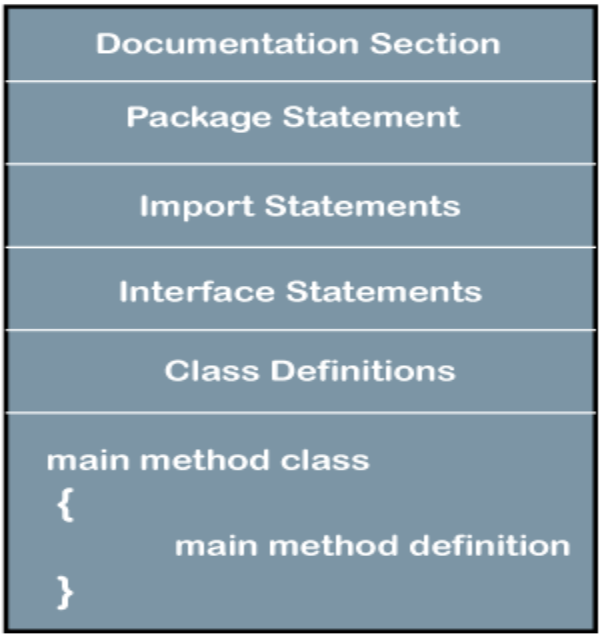
d. **Javadoc (for creating HTML documents):** It is used to create HTML format documentation from Java source code files.

e. **javah (for C header files):** It Produces header files by using with native methods.

f. **javap (Java disassemble):** Which enables us to convert byte code files into a program description.

g. **jdb (Java debugger) :** Which helps us to find errors in our programs.

JAVA PROGRAM STRUCTURE:



Structure of Java Program

Documentation Section :

The documentation section define a set of comment lines giving the name of the program, author and other details. There are three types of comment lines in java is,

1) Single line comments denoted as //.....//
2) Multiline comments denoted as /*.....

……..*/

3) Documented comments denoted as /**.....

.....**/

Package statement :

Package: Package nothing but a set of classes and interfaces

for example,

import java.lang.*;

In the above statement lang package is a sub package of java. It contains predefine classes and predefine methods.

Import Statements :

The import statement instructs the interpreter to load the classes contained in the packages.

Interface statement:

An interface is like a class but includes a group of method declarations.

Class definitions:

A java program may contain multiple class definitions. Classes are the primary and essential elements of java program

Ex : class sample

```
{
    public static void main (String args[])
      {
        System.out.println("Dnr College");
      }
}
```

In the above program class is a keyword and sample is a identification or class name

1) public is the access modifier

2) static is also a modifier

3) void is written type means it does not written any value.

4) System is predefine class

5) out is predefine object.

6) println is predefine method

Main method :

Since every java stand alone program requires a main method. Class is the essential part of a java program. The main method create objects of various classes and establishes communications between them.

DIFFERENCES BETWEEN C, C++ AND JAVA: The following table describes the major differences between C, C++, and Java.

S.N.	Basis	C	C++	Java
1	**Origin**	The C language is based on BCPL.	The C++ language is based on the C language.	The Java programming language is based on both C and C++.
2	**Programming Pattern**	It is a procedural language.	It is an object-oriented programming language.	It is a pure object-oriented programming language.
3	**Approach**	It uses the top-down approach.	It uses the bottom-up approach.	It also uses the bottom-up approach.
4	**Code Execution**	The code is executed directly.	The code is executed directly.	The code is executed by the JVM.
5	**Platform Dependency**	It is platform dependent.	It is platform dependent.	It is platform-independent because of byte code.
6	**Translator**	It uses a compiler only to translate the code into machine language.	It also uses a compiler only to translate the code into machine language.	Java uses both compiler and interpreter and it is also known as an interpreted language.
7	**Pointer Concept**	It supports pointer.	It also supports pointer.	Java does not support the pointer concept because of security.
8	**Constructor/ Destructor**	It does not support constructor and destructor.	It supports both constructor and destructor.	It supports constructors only.
9	**Exception Handling**	It does not support exception handling.	It supports exception handling.	It also supports exception handling.
10	**goto Statement**	It supports the goto statement.	It also supports the goto statement.	It does not support the goto statements.

NAMING CONVENTIONS AND DATA TYPES

NAMING CONVENTIONS : Java naming convention is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method, etc.

All the classes, interfaces, packages, methods and fields of Java programming language are given according to the Java naming convention. If you fail to follow these conventions, it may generate confusion or erroneous code.

Naming Conventions of the Different Identifiers

The following table shows the popular conventions used for the different identifiers.

Identifiers Type	Naming Rules	Examples

Class	It should start with the uppercase letter.
	It should be a noun such as Color, Button, System, Thread, etc.
	Use appropriate words, instead of acronyms.
		public class **Employee**
		{
		//code snippet
		}

Interface	It should start with the uppercase letter.
	It should be an adjective such as Runnable, Remote, ActionListener.
	Use appropriate words, instead of acronyms.
		interface **Printable**
		{
		//code snippet
		}

Method	It should start with lowercase letter.
	It should be a verb such as main(), print(), println().
	If the name contains multiple words, start it with a lowercase letter followed by an uppercase letter such as actionPerformed().
		class Employee
		{
		// method
		void **draw()**
		{
		//code snippet
		}
		}

Variable	It should start with a lowercase letter such as id, name.
	It should not start with the special characters like & (ampersand), $ (dollar), _ (underscore).
		class Employee
		{
		// variable

	If the name contains multiple words, start it with the lowercase letter followed by an uppercase letter such as firstName, lastName.
	Avoid using one-character variables such as x, y, z.
		int **id**;
		//code snippet
		}

Package	It should be a lowercase letter such as java, lang.
	If the name contains multiple words, it should be separated by dots (.) such as java.util, java.lang.
		//package
		package **com.javatpoint;**
		class Employee
		{
		//code snippet
		}

Constant	It should be in uppercase letters such as RED, YELLOW.
	If the name contains multiple words, it should be separated by an underscore(_) such as MAX_PRIORITY.
	It may contain digits but not as the first letter.
		class Employee
		{
		//constant
		static final int **MIN_AGE** = 18;
		//code snippet
		}

DATA TYPES IN JAVA

Definition:-"Data types define what kind of data that the variable will hold."

(OR)

"Data type defines which type of data it is."

Data type specifies the size and the type of value that can be stored.

In java language the data types are classified into two types,

1. Primitive data type
2. Non primitive data type

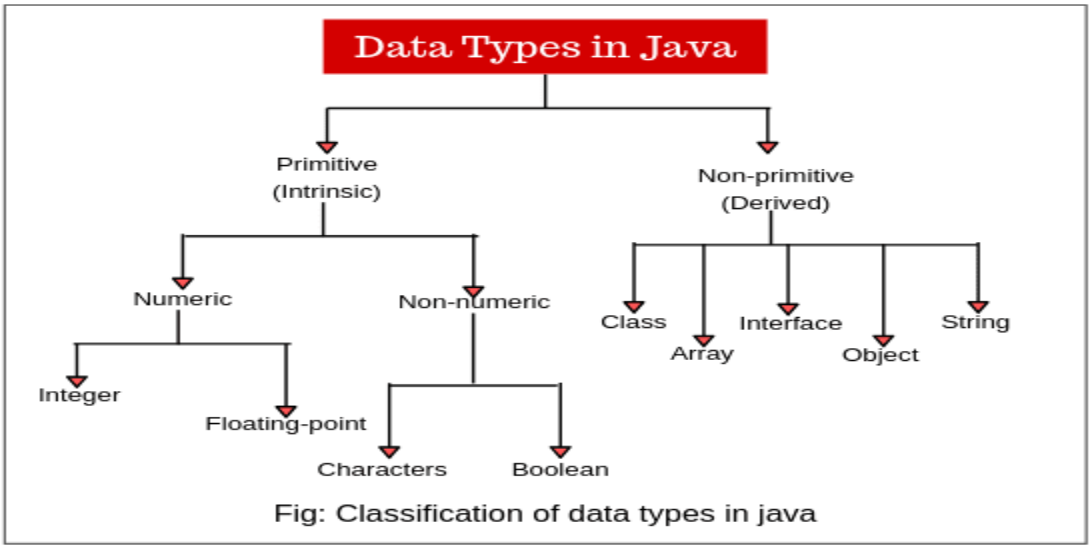


Fig: Classification of data types in java

1. Primitive data type

In Java language, primitive data types are the building blocks of data manipulation. Primitive data types again divided into four types as follows,

a) Integer type: Integer type can hold whole numbers. Java supports four types integers, they are

- byte
- short
- int
- long

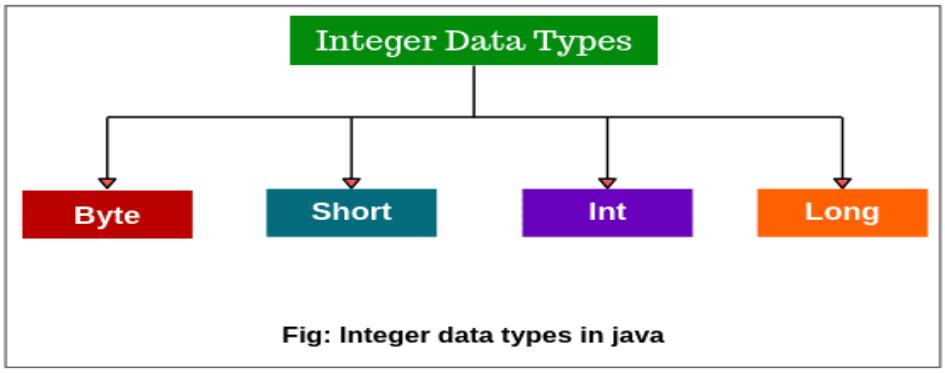


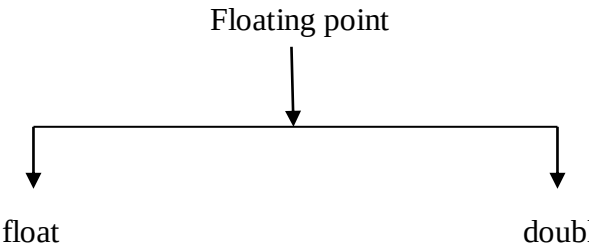
Fig: Integer data types in java

Java values are signed. It means they can be positive or negative. Java does not support unsigned integers.

sno	Data type	Size(in bytes)	range
1	byte	1	+127 to -128
2	short	2	+32767 to -32767
3	int	4	+2147483647 to -2147483647
4	long	8	+9223372036854775807 to -9223372036854775808

b) Floating point data types :

These can hold fractional numbers such as 27.30, -1.78. There are two types of floating point data types in java.



The floating type values are single precision numbers while the double type represents the double precision numbers.

Float ----- 4 bytes

Double ----- 8 bytes

c) Character type:

In order to store character constants in memory, java provides a character data type called "char"

The character data type can occupies 2 bytes in memory. It can hold only a single character.

d) Boolean type:

Boolean type is used when you want to test a particular condition during the execution of the program. There are only two values that a boolean type can take i.e True or False.

Boolean type is declared by the keyword "boolean" and uses only 1 bit of storage.

2. Non-Primitive data type

These data types are not actually defined by the programming language but are created by the programmer. Non-Primitive data types again divided into four types as follows,

a) Class: Class is a collection of objects that share common name and similar behaviour. Class is treated as user defined data type and the object of that class is treated as variable. We can create any numbers of objects for that class.

b) Object: Object is an instance of a class. An object is nothing but a self-contained component which consists of methods and properties to make a particular type of data useful.

c) Array: Array is a group of similar data items, that can share same name of same data type.

Ex: int a[10];

d) Interface: An interface is a basically a kinf of class, interfaces contains methods and variables.

The difference between class and interface are, in interface we can use only abstract methods and final variables.

Example :

```
interface   interface_name
   {
      Variable declarations;
     Methods declarations;
   }
```

e) String: In Java, a string is a sequence of characters. For example, *"hello"* is a string containing a sequence of characters *'h'*, *'e'*, *'l'*, *'l'*, and *'o'*.

LITERALS: In java literals are a sequence of digits, alphabets, letters those are used for representing the value of the variable.

Java language consists of 5 types of literals

a. Integer literals.
b. Floating point literals
c. Character literals
d. String literals
e. Boolean literals.

CONSTANTS : "Constants in java refers the value that doesn't change during the execution of the program is called Constant"

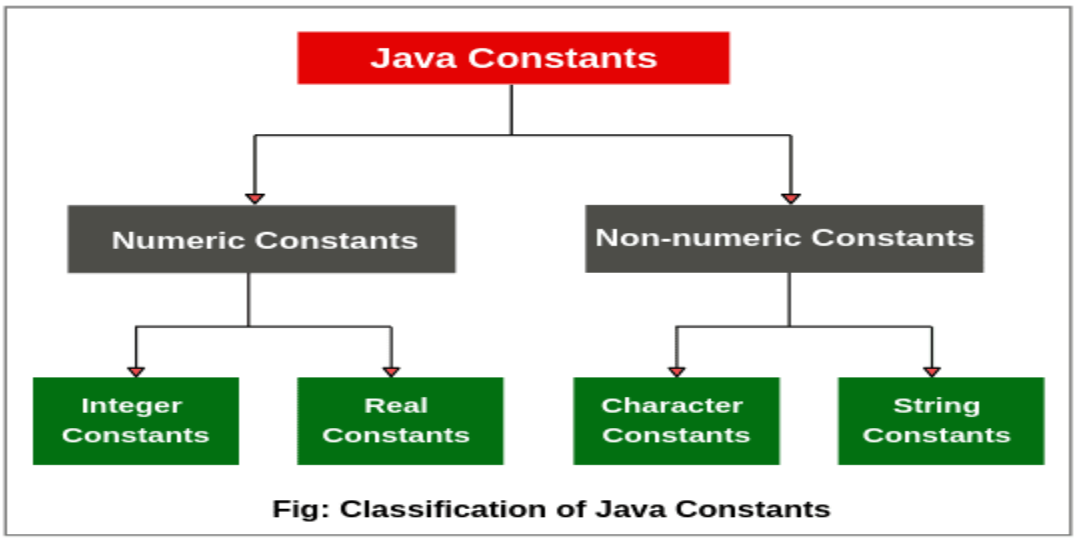


Fig: Classification of Java Constants

1. Numeric Constants:

Numeric constants are the values fixed throughout the program execution. Numeric constants are classified into two types:

• Integer constants

• Floating-point constants

Integer constants

To define an integer constant the following rules have to be followed:

Integer constants are broadly classified into three types:

Decimal integer constant

Octal integer constant

Hexadecimal integer constant

Real Constants:

Real constants consists of number with fractional part.

Example : 2.5, -1.7

Floating-point constants : It may contains four parts

1. Whole number
2. Decimal point
3. Fractional part
4. Exponential part

Example :

2.56e2 [2.56 – Mantissa

2 – Exponential part]

2. Non numeric Constants:

Non numeric constants have values that are start with a letter. Non numeric constants are classified into two types:

a. Single character constants: A single character constant contains a single character enclosed with pair of single quotation

Example : '5', 'C', '.'

b. String Constants: A string constant contains sequence of characters enclosed between double quotes

The characters may be alphabets, digits, special characters and blank spaces

Example:

"Hello java"

"a+b"

"1997"

c. Back slash character constants: Java supports some special back slash character constants that are used in output methods.

Example:

\n – stands for new line character

\t – stands for horizontal tab space

VARIABLE: A Variable is a name of the memory location used to store data value. A variable may take different values at different times during the execution of the program. Variable names consists of alphabets , digits, underscore(_), and dollar ($) characters.

Initialization of variable:

The process of giving initial values to a variable is called Initialization. A variable must be given value after it has been declared but before it is used in an expression.

This can be achieved in two ways,

1. By using an assignment statement
2. By using read statement

1. By using Assignment statement:

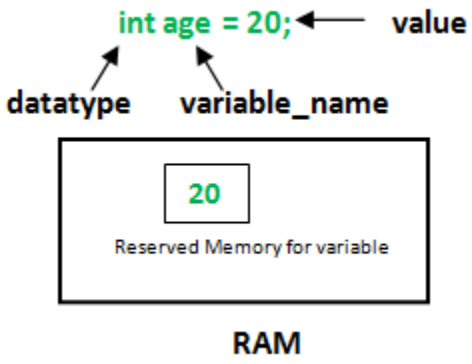
A simple method of giving value to a variable is

Syntax:

Data type variable name = value;

Example:

int age=20;



Example program for Assignment statement

```
//Addition of two numbers
class Addition
  {
     public static void main(String args[])
        {
          int a=10,b=20,c;
          c=a+b;
```

```
        System.out.println("c = "+c);
      }
  }
```

OUTPUT : **D:\> javac Addition.java**

D:\> java Addition

c = 30

2. By using read statement :

We can also give values to variables through the keyboard using the readLine() method.

Example :

```
int a=Integer.parseInt(in.readLine());
```

Example program for read statement

// Addition of two numbers

```
class Addition
  {
    public static void main(String args[])
      {
       try
           {
             DataInputStream dis=new DataInputStream(System.in);
             int a,b,c;
             System.out.println("Enter a value   :");
             a=Integer.parseInt(dis.readLine());
             System.out.println("Enter b value :");
             b=Integer.parseInt(dis.readLine());
             c=a+b;
            System.out.println("c ="+c);
           }
            catch(Exception e)
           {
           }
      }
  }
```

OUTPUT : **D:\> javac Addition.java**

D:\> java Addition

Enter a value : 10

Enter b value : 20

c = 30

TYPES OF VARIABLES: Java variables can be classified into three types

1. Instance Variables
2. Static Variables
3. Local Variables

1. Instance Variable

A variable declared inside the class but outside the body of the method, is called instance variable. We call the instance variables by using the object.

Without creating an object there is no use of instance variable why because whenever we create an object then only memory will be allocated for instance variables.

// Example program instance variables

```
class InstanceDemo
  {
    int a=10; // Instance variable
    public static void main(String args[])
      {
        InstanceDemo d=new InstanceDemo();
        System.out.println(d.a);
        System.out.println("a value is :"+d.a);
      }
  }
```

OUTPUT :

D:\>javac InstanceDemo.java

D:\>java InstanceDemo

10

10

2. Static Variable

A variable which is declared as static is called static variable. Static variables lso known as Class variables. Static variables are very flexible to access, we can access these static variables directly or by using object name, or by using class name.

// Example program static variables

```
class StaticDemo
  {
     static int a=10; // static variable
     public static void main(String args[])
       {
          System.out.println(a);
          StaticDemo d=new StaticDemo();
          System.out.println(d.a);
          System.out.println(StaticDemo.a);
       }
  }
```

OUTPUT :

```
D:\>javac StaticDemo.java
D:\>java StaticDemo
10
10
10
```

3. Local variables

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

this keyword :

'this' keyword is used to access the instance variables whenever the instance variable names and the local variables are same.

// Example program for Local variables

```
class LocalDemo
  {
     int a=10; // Instance variable
     void show()
```

```
    {
        int a=15; //Local variable
        System.out.println(a);
        System.out.println(this.a);
    }
  public static void main(String args[])
    {
        LocalDemo d=new LocalDemo();
        d.show();
        System.out.println(d.a);
    }
}
```

OUTPUT :

```
D:\>javac LocalDemo.java
D:\>java LocalDemo
15
10
10
```

OPERATORS IN JAVA

OPERATORS

Java operators can be classified into the following related categories.

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Assignment operators
5. Increment and Decrement operators]
6. Conditional operators
7. Bitwise operators
8. Special operators

1. Arithmetic operators :

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication, etc... Java provides the basic arithmetic operators.

Operator	Example	Meaning
+	a+b	Addition
-	a-b	Subtraction
*	a*b	Multiplication
/	a/b	Division
%	a%b	Modulus/Modular division

//Example program for Arithmetic operators

```
class Arithmetic
    {
       public static void main(String args[])
        {
            int a=10;
            int b=5;
            System.out.println(a+b);
            System.out.println(a-b);
            System.out.println(a*b);
            System.out.println(a/b);
            System.out.println(a%b);
        }
    }
```

Output : D:\> javac Arithmetic.java

D:\> java Arithmetic

15

5

50

2

0

2. Relational operators

An expression containing a relational operator is called Relational expression. The value of relational expression is either 'True' or 'False'.

Java supports six relational operators as follows,

Operatore	Meaning
<	Less than
<=	Less than (or) equal to
>	Greater than
>=	Greater than (or) equal to
= =	Equal to
!=	Not equal to

Example :

4<=10 →True

4>10 → False

4==10 → False

10==10 → True

//Example program for Relational operators

```
public class Relational
    {
       public static void main(String args[])
         {
            int a = 10;
            int b = 20;
           System.out.println("a < b = " + (a < b) );
           System.out.println("a <= b = " + (a <= b) );
           System.out.println("a > b = " + (a > b) );
           System.out.println("a >= b = " + (a >= b) );
           System.out.println("a == b = " + (a == b) );
           System.out.println("a != b = " + (a != b) );
         }
    }
```

Output : D:\> javac Relational.java

D:\> java Relational

a < b = true

a <= b = true

a > b = false

a >= b = false

a == b = false

a != b = true

3. Logical operators

In addition to relational operators, java has 3 logical operators,

Operator	Meaning	Example
&&	Logical AND	(a>b)&&(a<=b)
||	Logical OR	(a<b)||(a==b)
!	Logical NOT	(a<=b)!(a!=5)

The logical operators '&&' and '||' are used for combining two or more relations. Like realational expressions, the logical expressions also has a value of 'True' or 'False'.

//Example program for Logical operators

```
class Logical
    {
        public static void main(String args[])
          {
              boolean a = true;
              boolean b = false;
              System.out.println("a && b = " + (a&&b));
              System.out.println("a || b = " + (a||b) );
              System.out.println("!a = " + !(a));
         }
}
```

Output : D:\> javac Logical.java

D:\> java Logical

a && b = false

a || b = true

!a = false

4. Assignment operators

The assignment operators are used to assign the value of an expression to a variable. The common assignment operator is equal to(=).

In addition java has a set of short hand assignment operators, which are used in the form of **v op = exp;**

Where 'v' is a variable

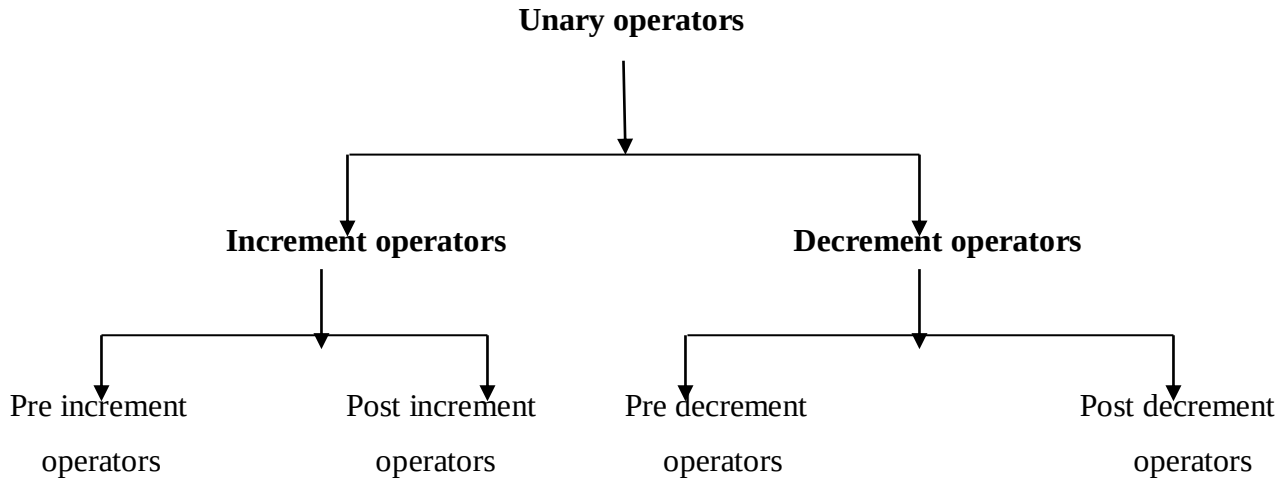
'exp' is an expression

'op' is a Binary operator

Statement with sample Assignment operator	Statement with shorthand operation
a=a+1	a+=1
a=a-1	a-=1
a=a*(n+1)	a*=n+1
a=a%b	a%=b
a=a/(n+1)	a/=n+1

5. Increment and Decrement operators

It is also called as Unary operators. Java provides two increment and decrement operators which are unary increment (++) and decrement (--) operators. Increment and decrement operators are used to increase or decrease the value of an operand by one.



Ex :- Initial value of x=10

Expression	Initial value of x	Final value of y	Final value of x
y=++x	10	11	11
y=x++	10	10	11
y= --x	10	9	9
y=x—	10	10	9

//Example program for Increment and decrement operators

```
class IncDec
  {
     public static void main(String args[])
       {
          int x=1;
          int y=3;
          int u;
          int z;
          u=x++;
          z=++y;
          System.out.println(x);
          System.out.println(y);
          System.out.println(u);
          System.out.println(z);
       }
  }
```

Output : D:\> javac IncDec.java

D:\> java IncDec

2

4

1

4

6. Conditional operators

The conditional operator is also known as the ternary operator. This operator consists of three operands and is used to evaluate Boolean expressions.

Syntax :

exp1 ? exp 2? exp 3

Exp 1 is evaluated first, if it is true then exp 2 is evaluated and becomes the value of the conditional expression. If it is false then the exp 3 is evaluated and becomes the value of the conditional expression.

Example 1:- int x=(10<20)?30:40;

Output :- 30

Example 2:- int x=(10>30)?30:40;

Output :- 40

Example 3:- int x=(10>20)?30:(40>50)?60:70;

Output :- 70

//Example program for Conditional operator

```
class Conditional
  {
    public static void main(String args[])
      {
        int a=10;
        int flag=(a<0)?0:1;
        if(flag==1)
          System.out.println("Positive number");
        else
          System.out.println("Negative number");
      }
  }
```

Output : D:\> javac Conditional.java

D:\> java Conditional

Positive number

7. Bitwise operators

Bitwise operators are used to test bits or shifting them to the right or left. Bitwise operators are divided into 3 types,

1. Bitwise logical operators (**&,|,^**)
2. Bitwise shift operators**(<<,>>)**
3. Bitwise complement**(~** tild operator)

1. Bitwise logical operators (&,|,^) :

OP1	OP2	OP1&OP2	OP1|OP2	OP1^OP2
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

a. Bitwise AND(&): The result of AND (&) operation is 1, if both bits are having a value of 1 otherwise it is Zero.

b. Bitwise OR(|) : The result of OR (|) operation is 1, if atleast one of the bit has a value of 1 and two bits have value of 1 otherwise it is zero.

c. Bitwise EXCLUSIVE OR(^)(XOR) : The result of EXCLUSIVE OR(^) is 1, if only one of the bits is one otherwise it is zero.

2. Bitwise shift operators :

The Shift operators are used to move bit pattern either to the left or right.

a. Bitwise Left shift :

Syntax: - op << n

Example:- **1 1 1 1 1 1 1 0 << 2**

1 1 1 1 1 0 0 0

b. Bitwise Right shift :

Syntax: - op >> n

Example :- **0 0 0 1 1 1 1 1 >> 3**

0 0 0 0 0 0 1 1

c. Bitwise Complement (~) :

Bitwise complement is an unary operator (works on only one operand). It is denoted by **~.** The ~ operator inverts the bit pattern. It makes every 0 to 1, and every 1 to 0.

8. Special operators

Java supports some special operators such as

a) instanceOf operator : instanceOf operator is used for object reference variable. This operator checks whether the object is to a particular class or not.

Example :- person instanceOf Student

It is true, if the object person belongs to the class Student otherwise it is false.

b) Dot(.) operator : The dot (.) operator is used to access instance variables and methods of a class.

Example 1:- s1.age → Variable

↓

Object

Example 2:- s1.show();

↓

method

c) new operator : new operator is used to create objects for a class

Example :- Student s = new Student();

↓ ↘

Object name Operator

CONTROL STATEMENTS IN JAVA

CONTROL STATEMENTS: Java provides three types of control flow statements.

1. Decision Making statements
   - if statements
   - switch statement
2. Loop statements
   - do while loop
   - while loop
   - for loop
   - for-each loop
3. Jump statements
   - break statement

- continue statement

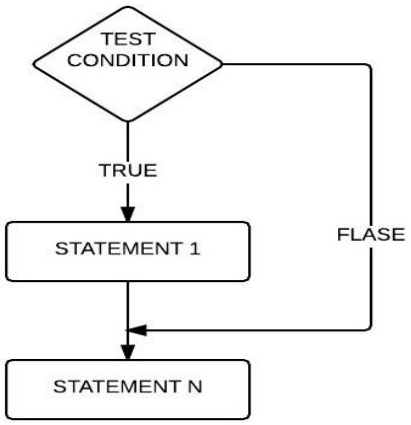
1. DECISION MAKING WITH IF STATEMENT

1.Simple if statement :

Syntax : if(condition)

```
{
  Statement block;
}
  Statement x;
```

Flowchart :



If the test condition is true, then statement 1 is executed, followed by statement N. If the condition is False, then the statement N will execute. Because it is out of the if condition block and it has nothing to do with the condition result

//Example program on simple if statement

```
class SimpleIfDemo
    {
        public static void main(String args[])
          {
            int n;
            n=Integer.parseInt(args[0]);
            if(n%2==0)
              {
                 System.out.println("Even number");
              }
            System.out.println("Simple if statement program");
          }
```

}

Output 1 : **D:\> javac SimpleIfDemo.java**

D:\> java SimpleIfDemo 10

Even number

Simple if statement program

Output 2 : **D:\> javac SimpleIfDemo.java**

D:\> java SimpleIfDemo 5

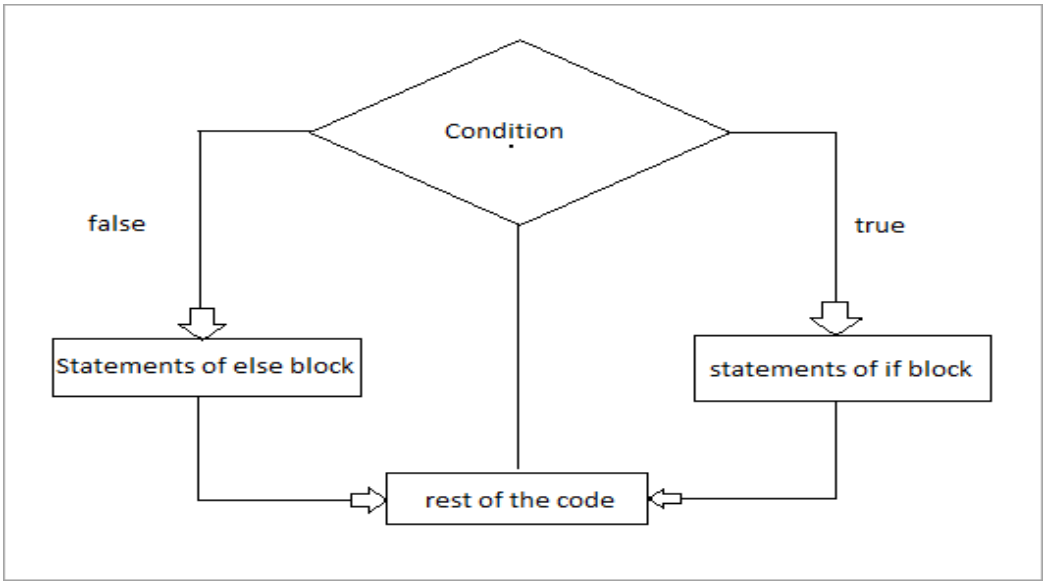
Simple if statement program

2. if else statement :

Syntax : if(condition)

```
    {
      True block statements;
    }
  else
    {
      False block statements;
    }
 Statement x;
```

Flowchart :



If the test condition is true, then the true block statements will be executed otherwise the false block statements will be executed and controller jumps to rest of the code.

//Example program on if else statement

class IfelseDemo

```
{
    public static void main(String args[])
      {
        int n;
        n=Integer.parseInt(args[0]);
        if(n%2==0)
            System.out.println("Even number");
        else
            System.out.println("Odd number");
      }
}
```

Output 1 : **D:\> javac IfelseDemo.java**

D:\> java IfelseDemo 10

Even number

Output 2 : **D:\> javac IfelseDemo.java**

D:\> java IfelseDemo 5

Odd number

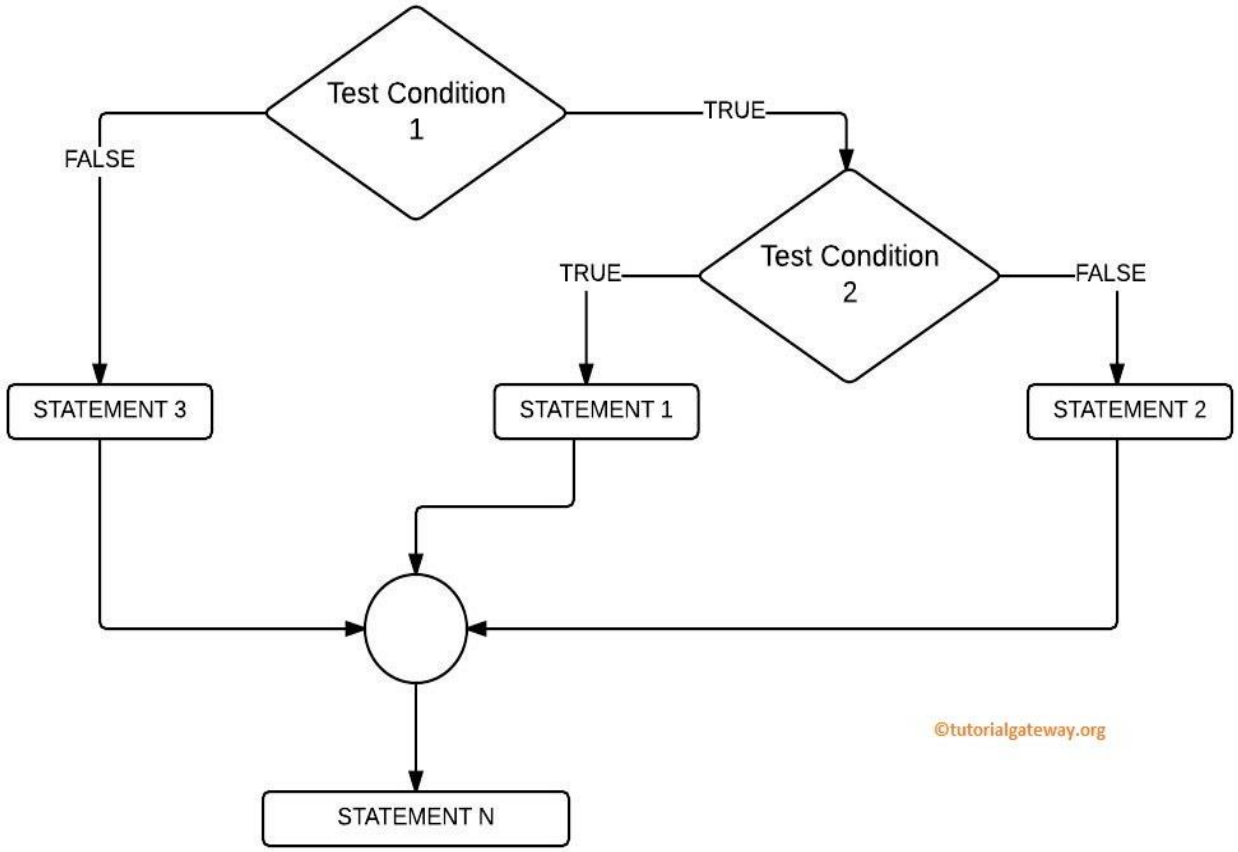
3. Nested if else statement :

When if statement contains another if statement is called Nested if else statement.

Syntax :

```
if(condition 1)
    {
      if(condition 2)
          Statement 1;
      else
          Statement 2;
    }
else
    {
      statement 3;
    }
Statement x;
```

Flowchart :



//Example program on nested if else statement

```
class NestedIfDemo
   {
      public static void main(String[] args)
         {
            int a,b,c;
            a=Integer.parseInt(args[0]);
            b=Integer.parseInt(args[1]);
            c=Integer.parseInt(args[2]);
            if(a>b)
               {
                 if(a>c)
                  {
                    System.out.println(a + " is largest number.");
                  }
                 else
                  {
```

```
                System.out.println(c + " is largest number.");
            }
        }
      else if(b>c)
        System.out.println(b + " is largest number.");
      else
        System.out.println(c + " is largest number.");
    }
}
```

Output 1 : **D:\> javac NestedIfDemo.java**

D:\> java NestedIfDemo 10 20 15

20 is largest number

Output 2 : **D:\> javac NestedIfDemo.java**

D:\> java NestedIfDemo 30 15 20

30 is largest number

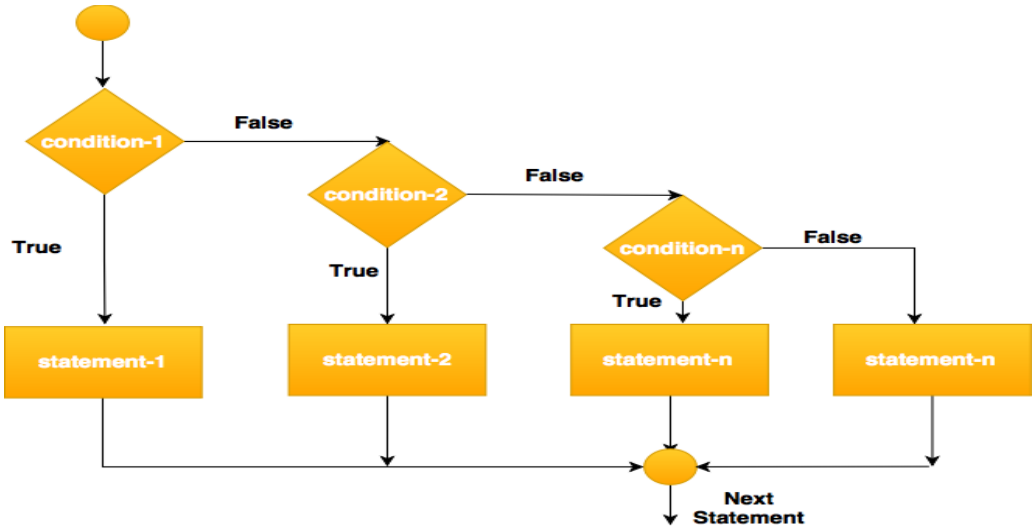
4. if else if ladder statement :

A number of conditions arises in a sequence then we can use if else if ladder statement to solve the problem in simple manner.

Syntax :

```
if (condition 1)
    {
      Statement 1;
    }
  else
    {
      if(condition 2)
      Statement 2;
      else
     default statements;
    }
```

Flowchart :



//Example program on if else if ladder statement

```
class IfelseIfLadder
  {
    public static void main(String args[])
      {
        int htno,sub1,sub2,sub3,total;
        double avg;
        htno=Integer.parseInt(args[0]);
        sub1=Integer.parseInt(args[1]);
        sub2=Integer.parseInt(args[2]);
        sub3=Integer.parseInt(args[3]);
        if((sub1>=35)&&(sub2>=35)&&(sub3>=35))
          {
            total=sub1+sub2+sub3;
            avg=total/3;
            if(avg>=85)
                System.out.println("O grade");
                else if((avg>=75)&&(avg<85))
                System.out.println("A grade");
                else if((avg>=65)&&(avg<75))
                System.out.println("B grade");
```

```
                else if((avg>=55)&&(avg<65))
                System.out.println("C grade");
                else
                  System.out.println("D grade");
            }
          else
              {
                System.out.println("FAIL");
              }
       }
   }
```

Output 1 : **D:\> javac IfelseIfLadder.java**

D:\> java IfelseIfLadder 100 75 85 95

O grade

Output 2 : **D:\> javac IfelseIfLadder.java**

D:\> java IfelseIfLadder 100 25 85 95

FAIL

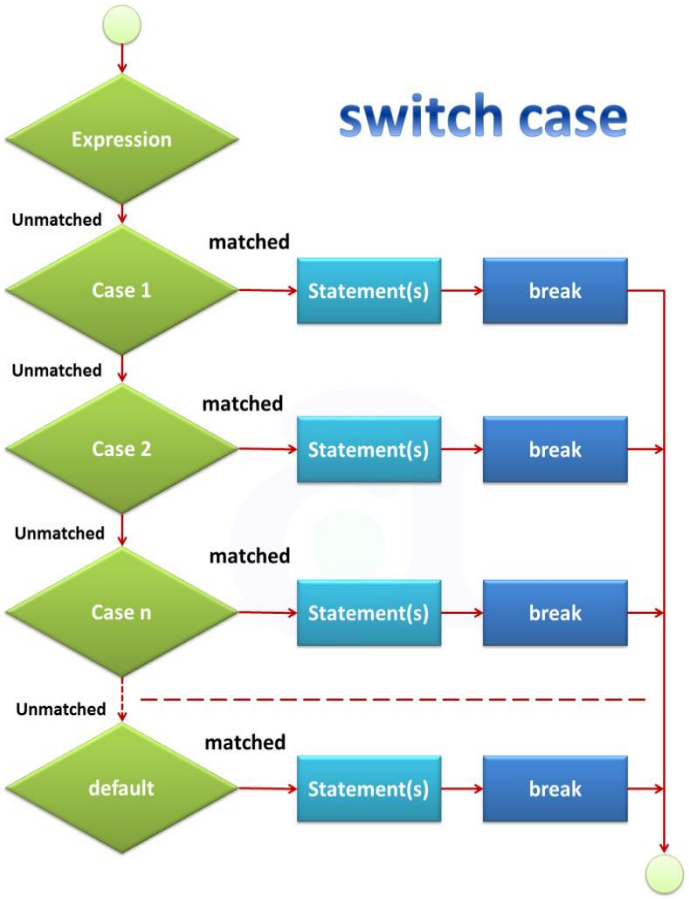
5. Switch case statement :

The switch case statement is a multi branching statement. It checks whether expression matches one of the constant value. The switch case statement requires only one argument which is checked with no. of case options.

Syntax : switch(expression)

```
            {
               case 1: statement 1;
                       break;
               case 2: statement 2;
                       break;
               case 3: statement 3;
                       break;
               default :default statements;
            }
```

Flowchart :



//Example program on switch case statement

```
class SwitchDemo
{
   public static void main(String s[])
   {
      int week= 2;
      String day;
      switch(week)
      {
         case 1: day="Sunday";
                 break;
         case 2: day="Monday";
                 break;
         case 3: day="Tuesday";
                 break;
```

```
        case 4: day="Wednesday";
                break;
        case 5: day="Thursday";
                break;
        case 6: day="Friday";
                break;
        case 7: day="Saturday";
                break;
        default: day="Invalid day";
      }
        System.out.println(day);
     }
  }
```

Output 1 : **D:\> javac** SwitchDemo.**java**

D:\> java SwitchDemo

Monday

2. LOOPING STATEMENTS

Looping: "The process of repeatedly executing a block of statements is known as Looping". Depending on the position of control statements in the loop, loop may be classified into entry control loop or exit control loop.

Java supports three types of Looping statements,

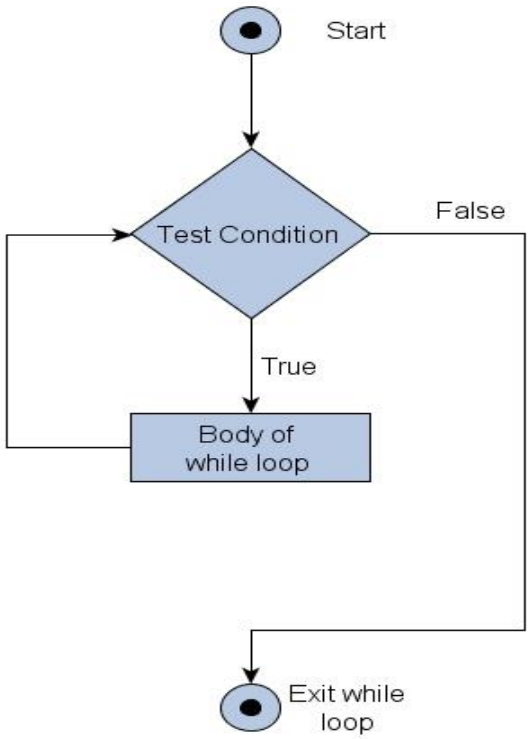
1. while loop (Entry control loop)
2. do-while loop (exit control loop)
3. for loop (entry control loop)

1. while loop : In while loop one or more statements are repeatedly executed until a particular condition is false.

Syntax :

```
while(condition)
{
  Body of the loop;
}
 Expression;
```

Flowchart :



In while loop, first the test condition is tested, if the condition is true then the body of the loop will be executed, after execution of the body, the test condition is once again evaluated and if it is true, the body is executed once again and this process is repeated until the test condition becomes false.

It is also called as entry control loop, in while loop conditions are tested before starting of the loop execution.

//Example program on while loop

```
import java.util.*;
class Palindrome
 {
    public static void main(String args[])
       {
          int n,n1,sum=0,r;
          Scanner sc=new Scanner(System.in);
          System.out.println("Enter one number    :");
          n=sc.nextInt();
          n1=n;
         while(n>0)
          {
            r=n%10;
```

```
            sum=(sum*10)+r;
            n=n/10;
           }
      if(n1==sum)
         System.out.println("Given number is palindrome");
      else
         System.out.println("Given number is not palindrome");
    }
}
```

Output 1 : **D:\> javac** Palindrome**.java**

D:\> java Palindrome

Enter one number :121

Given number is palindrome

Output 2 : **D:\> javac** Palindrome**.java**

D:\> java Palindrome

Enter one number :152

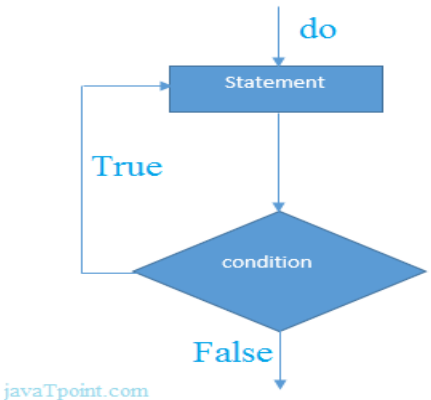
Given number is not palindrome

2. do-while loop : do-while loop is similar to while loop, however there is a difference between them: In while loop, condition is evaluated before the execution of body of the loop but in do-while loop condition is evaluated after the execution of body of the loop.

Syntax :

```
do
  {
    Body of the loop;
  } while(test condition);
```

Flowchart :



Once reaching the do statement, the program proceeds to evaluate the body of the loop first, at the end of the loop the test condition in the while statement is evaluated. If the condition is true, the program continues to evaluate body of the loop once again. This process is continuous up to condition is false. When the condition becomes false the loop will be terminated. The test condition is evaluated at the bottom of the loop.

//Example program on do-while loop

```
class DowhileDemo
   {
      public static void main(String[] args)
         {
           int i = 1, n = 5;
          // do...while loop from 1 to 5
          do
           {
             System.out.println(i);
             i++;
           } while(i <= n);
   }
}
```

Output : **D:\> javac** DowhileDemo.**java**

D:\> java DowhileDemo

1
2
3
4
5

3. for loop : The Java *for loop* is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop. for loop is another controlled loop in java.

Syntax : for(Initialization;test_condition;Increment/Decrement)

```
{
    Body of the loop;
}
```

The for loop contains 3 parts,

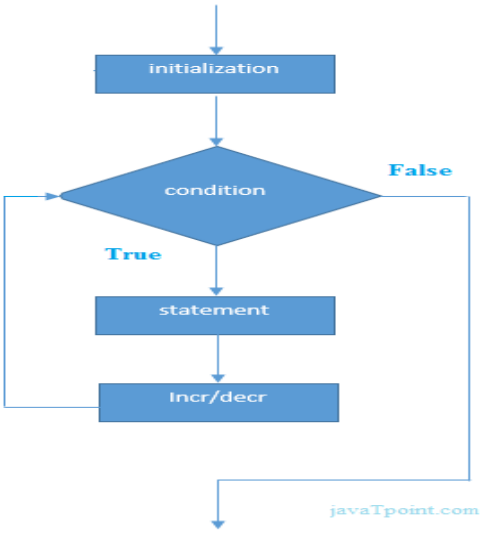
a. Initialization
b. Test condition
c. Increment/Decrement

In for loop the initialization of variable is done first then the value of the control variable is tested using the test condition.

If the condition is true then the body of the loop will be executed. If the condition is false then the body of the loop will be terminated.

When the body of the loop is executed the control transfers back to the for statement i.e. increment/decrement. After increment/decrement the test condition is again evaluated.

Flowchart :



//Example program on for loop

```
class forDemo
   {
     public static void main(String[] args)
        {
           int i,n = 5;
           for(i=1;i<=n;i++)
             {
               System.out.println("Java is fun");
             }
        }
   }
```

Output : **D:\> javac forDemo.java**

D:\> java forDemo

Java is fun

Java is fun

Java is fun

Java is fun

Java is fun

// Example program to check the given number is prime or not using for loop

import java.util.*;

```
class Prime
   {
      public static void main(String[] args)
        {
           int n;
           Scanner sc=new Scanner(System.in);
           System.out.println("Enter one number     :");
           n=sc.nextInt();
           boolean flag = false;
           for (int i = 2; i <= n / 2; ++i)
             {
                if (n % i == 0)
                  {
                     flag = true;
                     break;
                  }
             }
          if (!flag)
                System.out.println(n+ " is a prime number.");
          else
                System.out.println(n+ " is not a prime number.");
        }
   }
```

Output 1:

E:\>javac Prime.java

E:\>java Prime

Enter on value :

5

5 is a prime number

Output 2:

E:\>javac Prime.java

E:\>java Prime

Enter on value :

6

6 is not a prime number

3. JUMPING STATEMENTS: Jumping statements transfer the execution control to the other part of the program. There are two types of jump statements in Java, i.e., break and continue.

1. **break:** the break statement is used to break the current flow of the program and transfer the control to the next statement outside a loop or switch statement. However, it breaks only the inner loop in the case of the nested loop.

The break statement cannot be used independently in the Java program, i.e., it can only be written inside the loop or switch statement.

2. continue: Unlike break statement, the continue statement doesn't break the loop, whereas, it skips the specific part of the loop and jumps to the next iteration of the loop immediately.

INPUT AND OUTPUT: Accepting input from the keyboard, Reading input with java.util.Scanner class, Displaying output with System.out.println ()

Java **Scanner class** allows the user to take input from the console. It belongs to **java.util** package. It is used to read the input of primitive types like int, double, long, short, float, and byte. It is the easiest way to read input in Java program.

```
import java.util.*;
class Palindrome
 {
    public static void main(String args[])
       {
          int n,n1,sum=0,r;
          Scanner sc=new Scanner(System.in);
          System.out.println("Enter one number    :");
          n=sc.nextInt();
          n1=n;
```

```
        while(n>0)
         {
          r=n%10;
          sum=(sum*10)+r;
          n=n/10;
         }
      if(n1==sum)
        System.out.println("Given number is palindrome");
      else
        System.out.println("Given number is not palindrome");
    }
 }
```

Output 1 : **D:\> javac** Palindrome**.java**

D:\> java Palindrome

Enter one number :121

Given number is palindrome

Output 2 : **D:\> javac** Palindrome**.java**

D:\> java Palindrome

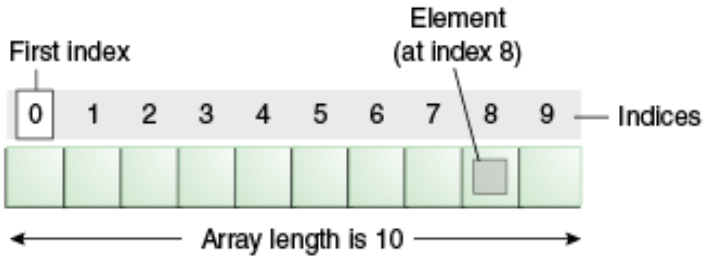
Enter one number :152

Given number is not palindrome

ARRAYS

ARRAYS: An array is a collection of similar elements of same data type. Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

1. A Java array variable can also be declared like other variables with [] after the data type.
2. The variables in the array are ordered and each have an index beginning from 0.



Advantages

1. **Code Optimization:** It makes the code optimized; we can retrieve or sort the data efficiently.

2. **Random access:** We can get any data located at an index position.

Disadvantages

1. **Size Limit:** We can store only the fixed size of elements in the array. There is no chance of increasing or decreasing the size based on our requirements. It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically.
2. **We can store only homogeneous (similar) data type values.**

TYPES OF ARRAYS

There are two types of array,

1. One Dimensional array
2. Two Dimensional array

1**. One Dimensional array :** The variable name using only one subscript is known as One Dimensional array.

Creating an array involves 3 steps :

a. Array declaration
b. Creating memory location
c. Initialization of arrays

a. Array declaration: Arrays in java may be declared as follows,

Syntax 1: Data type Array name[ ];

Example: int salary[ ];

Syntax 2: Data type [ ]array name;

Example: int [ ]salary;

Syntax 3: Data type[ ] array name;

Example : int[ ] salary;

Note: At the time of declaration, we cannot specify the size of the array otherwise we will get compile time error.

b. Creating memory location: After declaring an array, we need to create memory for the array. We can create memory for an array by using new operator.

Syntax: Data type Array name[ ] = new Data type[Size];

Example: int number[ ] = new int[5];

c. Initialization of Arrays: The final step is to put values into the array is known as Iniatialization.

Syntax 1: Data type Array name[ ] = new Data type[Size];

Example: int number[ ] = new int[5];

number[0]=10;

number[1]=20;

number[2]=30;

number[3]=40;

number[4]=50;

Syntax 2: Data type Array name[ ]={List of values};

Example: int number[ ]={10,20,30,40,50};

Example program 1 for One Dimensional array

```
class Testarray
  {
     public static void main(String args[])
         {
            int a[ ]=new int[5];//declaration and instantiation
            a[0]=10;//initialization
            a[1]=20;
            a[2]=70;
            a[3]=40;
            a[4]=50;
            //Traversing array
             for(int i=0;i<a.length;i++) //length is the property of array
             System.out.println(a[i]);
         }
  }
```

Output : D:\> javac Testarray.java

D:\> java Testarray

10

20

30

40

50

Example program 2 for One Dimensional array

```
class Testarray1
    {
        public static void main(String args[])
            {
               int a[]={10,20,30,40};//declaration, instantiation and initialization
               //printing array
               for(int i=0;i<a.length;i++)//length is the property of array
               System.out.println(a[i]);
             }
    }
```

Output : **D:\> javac Testarray1.java**

D:\> java Testarray1

10

20

30

40

2**. Two Dimensional array :** The list of items can be given in one variable name using two subscripts and such variable is called Two dimensional array.

Creating an array involves 3 steps :

a. Array declaration
b. Creating memory location
c. Initialization of arrays

a. Array declaration:

Syntax 1: Data type Array name[ ][ ];

Example: int salary[ ][ ];

Syntax 2: Data type [][]array name;

int [ ][ ]salary;

Note: At the time of declaration, we cannot specify the size of the array otherwise we will get compile time error.

b. Creating memory location: After declaring an array we need to create memory for that array by using new operator.

Syntax: Data type array name[ ][ ] = new Data type[Size][Size];

Example: int salary[ ][ ] = new int[2][2];

c. Initialization of arrays: The final step is to put values into the array, this process is known as Initialization.

Syntax: Data type Array name = new Data type[Size][Size];

Example: int salary[ ][ ]=new int[2][2];

salary [0][0]=10;

salary [0][1]=20;

salary[1][0]=30;

salary[1][1]=40;

Example program for Two Dimensional array

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

          Scanner sc=new Scanner(System.in);
          System.out.println("Enter Row length of an array : ");
          int row=sc.nextInt();
          System.out.println("Enter column length of an array : ");
          int column=sc.nextInt();
          int a[][]=new int[row][column];//declaration
          System.out.print("Enter " + row*column + " Elements to Store in Array :\n");
          for (int i = 0; i < row; i++)
         {
            for(int j = 0; j < column; j++)
             {
                 a[i][j] = sc.nextInt();
             }
        }
         System.out.print("Elements in Array are :\n");
         for (int i = 0; i < row; i++)
        {
```

```
            for(int j = 0; j < column; j++)
            {
              System.out.println("Row ["+i+"]:  Column ["+j+"] :"+a[i][j]);
            }
         }
       }
   }
```

Output :

```
D:\> javac TwoDimensional.java
D:\> java TwoDimensional
Enter Row length of an array :
2
Enter column length of an array :
2
Enter 4 Elements to Store in Array :
10 20 30 40
Elements in Array are :
Row [0]:  Column [0] :10
Row [0]:  Column [1] :20
Row [1]:  Column [0] :30
Row [1]:  Column [1] :40
```

COMMAND LINE ARGUMENTS IN JAVA:

"Passing arguments from the command prompt to the variables is known as Command line arguments."

The main purpose of command line arguments is customization of main method. We must pass atleast one argument from the command prompt. We can pass 'n' arguments from the command prompt.

Program :

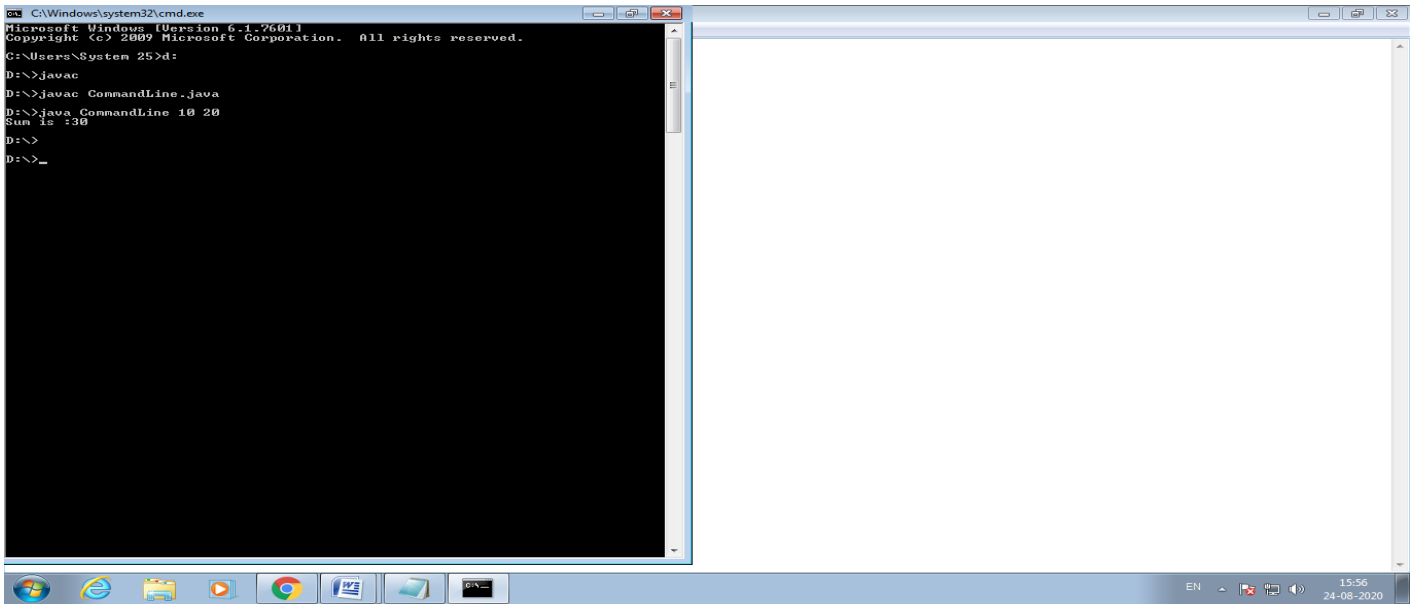
The following program illustrates the use of command line arguments.

//CommandLine.java

```
class  CommandLine
{
   public static void main(String args[])
        {
          int a=Integer.parseInt(args[0]);
```

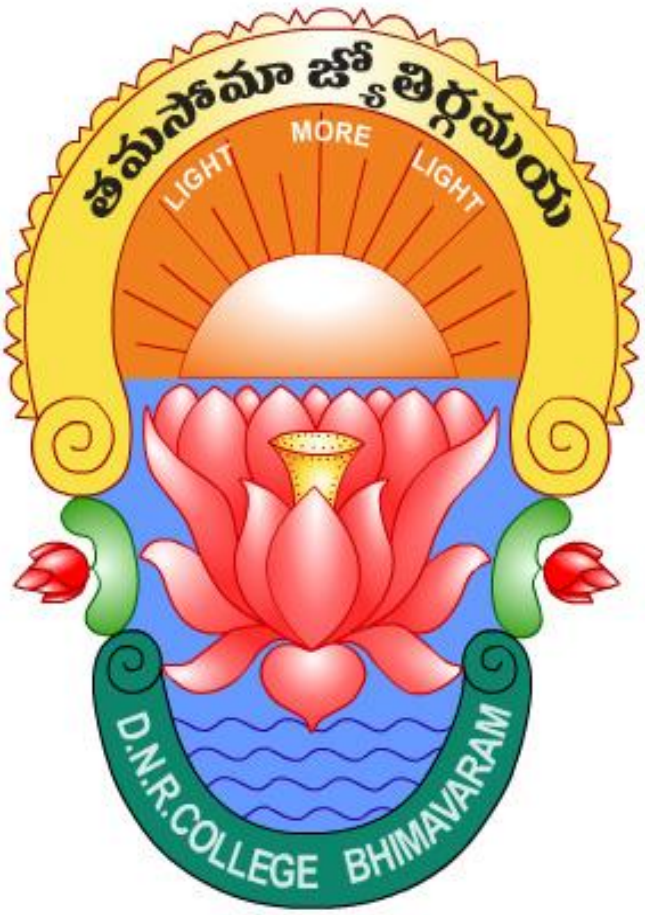
```
        int b=Integer.parseInt(args[1]);
        int c= a+b;
      System.out.println("Sum is :"+c);
       }
}
```

OUTPUT :



D.N.R. COLLEGE (AUTONOMOUS): BHIMAVARAM

DEPARTMENT OF COMPUTER SCIENCE



OBJECT ORIENTED PROGRAMMING WITH JAVA – IV

UNIT II

IV SEMESTER

UNIT II

STRINGS

String: A group of characters is called String. For example, “hello” is a string containing a sequence of characters ‘h’,’e’,’l’,’l’,’o’. Java string provides a lot of concepts that can be performed on a string such as compare, concatenation, equal, split, lemgth etc.

Creating Strings

The most direct way to create a string is to write

String greeting = "Hello world!”;

Whenever it encounters a string literal in your code, the compiler creates a String object with its value in this case, "Hello world!'.

1) Example program for java strings

```
class StringDemo
  {
     public static void main(String args[])
        {
          // create strings
          String first = "Java";
          String second = "Python";
          String third = "JavaScript";
          // print strings
          System.out.println(first);   // print Java
          System.out.println(second);  // print Python
          System.out.println(third);   // print JavaScript
        }
  }
```

Output : **D:\> javac StringDemo.java**

D:\> java StringDemo

Java

Python

JavaScript

2) Example program for java strings

```
public class StringExample
  {
```

```
    public static void main(String args[])
     {
        String s1="java";//creating string by java string literal
        char ch[]={'s','t','r','i','n','g','s'};
        String s2=new String(ch);//converting char array to string
        String s3=new String("example");//creating java string by new keyword
        System.out.println(s1);
        System.out.println(s2);
        System.out.println(s3);
    }
}
```

Output : **D:\> javac StringExample.java**

D:\> java StringExample

java

strings

example

Immutability of Strings

In java, string objects are immutable. Immutable simply means un-modifiable or un-changeable. Once string object is created its data or state can't be changed but a new string object is created. If we are trying to perform any changes, with those changes will create new object. This non-changeable nature is nothing but Immutability of the string object

STRING CLASS METHODS: The String class has a set of built-in methods that you can use on strings.

Method	Description
1. s2=s1.toLOwerCase();	Converts the String to all lower case letters
2. s2=s1.toUpperCase();	Converts the String to all upper case letters
3. s2=s1.replace("x","y");	Replace x with y
4.s2=s1.trim();	Remove white spaces at the beginning and end of the string s1.
5. s1.equals(s2);	Compares two strings. Returns true if the strings are equal, and false if not
6.s1.length();	Returns the length of a specified string
7.s1.charAt(n);	Returns the character at the specified index (position)
8.s1.compareTo(s2);	Compares two strings lexicographically

9.s1.concat(s2);	Appends a string to the end of another string
10.s1.split();	Split s1 string into an array of objects.
11.s1.isEmpty();	Checks s1 string is empty or not.
12.s1.lastIndexOf();	Returns the position of the last found occurrence of specified characters in a string
13. s1.contains();	Checks whether a string contains a sequence of characters
14. s1.endsWith();	Checks whether a string ends with the specified character.

Example program for String handling functions

1. Example program Converting Lower case to Upper case

```
class StringDemo
  {
    public static void main(String args[])
      {
        String s1=new String("hello");
        String s2=s1.toUpperCase();
        System.out.println(s2);
      }
  }
```

Output : **D:\> javac StringDemo.java**

D:\> java StringDEmo

HELLO

2. Example program Converting Lower case to Upper case

```
class StringDemo
  {
    public static void main(String args[])
      {
        String s1=new String("HELLO");
        String s2=s1.toLowerCase();
        System.out.println(s2);
      }
  }
```

Output : **D:\> javac StringDemo.java**

D:\> java StringDemo

hello

3. Example program Replacing string

```
class StringDemo1
  {
    public static void main(String args[])
      {
        String s1=new String("hello");
        String s2=s1.replace("hello","dnr");
        System.out.println(s2);
      }
  }
```

Output : **D:\> javac StringDemo.java**

D:\> java StringDemo

Dnr

4. Example program Trim a string

```
class StringDemo1
  {
    public static void main(String args[])
      {
        String s1=new String("      Hello      ");
        String s2=s1.trim();
        System.out.println(s2);
      }
  }
```

Output : **D:\> javac StringDemo.java**

D:\> java StringDemo

Hello

Wrapper Classes: Primitive data types may be converted into object type by using Wrapper classes.

1. Taking of a variable and putting in a object is known as Wrapping.
2. Taking out that variable from an object is known as Unwrapping

Converting primitive data type to object type by using Constructor method

Examples: Integer obj=new Integer(i);

Float obj=new Float(f);

Long obj=new Long(l);

Converting object data type to primitive data using type value method

Example: int i=obj.intValue();

float f=obj.floatValue();

long l=obj.longValue();

Vector: Vector is like the *dynamic array* which can grow or shrink its size. Unlike array, we can store n-number of elements in it as there is no size limit. That can hold objects of any type and any number.

1. The vector is resizable (or) Growable array
2. Duplicate values are allowed
3. Insertion order is preserved
4. Null insertion is possible
5. Heterogeneous objects are allowed
6. Random access interface

INTRODUCTION TO OOPS

OOP(Object oriented programming): Object-Oriented Programming is a methodology or paradigm to design a programs using classes and objects. It simplifies software development and maintenance by providing some concepts

FEATURES OF OOP: There are 8 main features of OOPS as follows.

1) Class

2) Object

3) Inheritance

4) Polymorphism

5) Abstraction

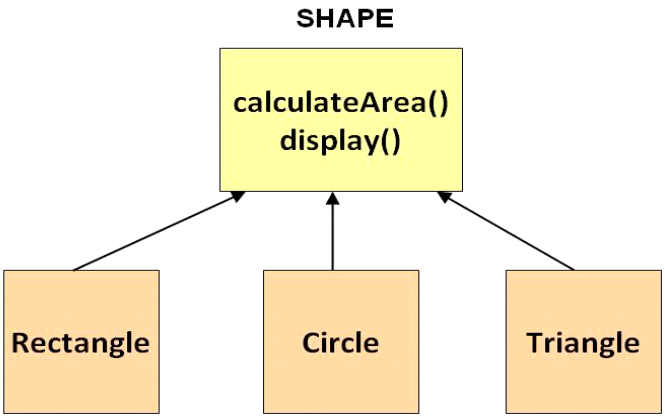
6) Encapsulation

7) Dynamic Binding

8) Message Communication

1**. Class**: Class is a set of objects with similar properties and common behaviour. A class is a collection of objects of similar type. Once class is defined, we can create any number objects belongs to that class

Ex: Class is SHAPE.



2) Objects:

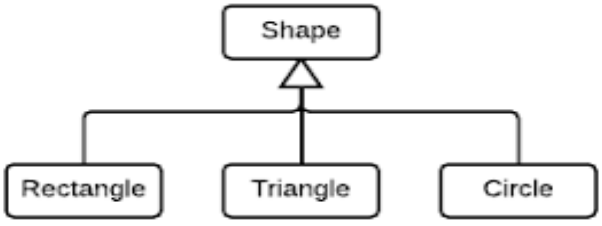
Objects are basic runtime entities, they may be characterized a location, a bank account and a table of information .Each object holds data and code to operate on data. Object is an instance of a class. An object contains properties and methods.

Ex: Object is Car.

Object : Car
Properties : Colour : Cost :
Methods : Start() Stop() Drive()

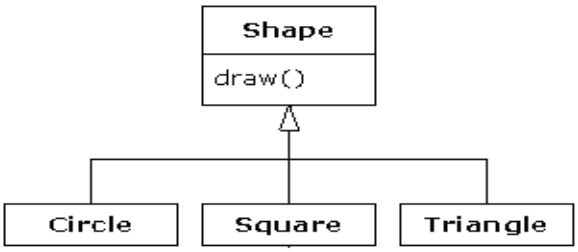
3) Inheritance:

Inheritance is the process of acquiring properties from one class to another class. It supports the concept of hierarchical classification. A super class exists in a hierarchical relationship with its subclass. One super class can have many subclasses. In OOP the concept inheritance provides the idea of code reusability.



4) Polymorphism:

Ability to take more than one form is known as Polymorphism. Polymorphism means many forms. An operation may takes different behaviour in different instances.



5) Data Abstraction :

Data abstraction refers to the act of representing essential features without including background details that is hiding the data.

Ex: Making a phone call

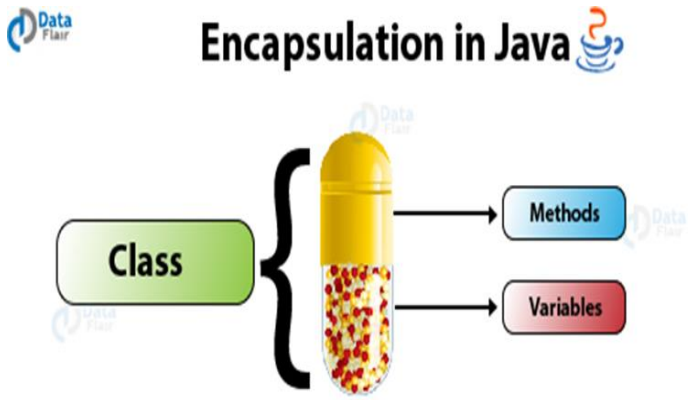
Information in | Data and methods | Information out

We know only how to make a phone call and how to receive a phone call but we don't know internally what goes on.

6) Data Encapsulation :

Wrapping up of data and member functions into single unit(binding the data) is known as Data encapsulation. This is most useful feature for class.

Ex : java class.



7) Dynamic Binding:

The code related with the given procedure call is not known until the time of call run time.

Ex: class A

```
{
void add();
}
class B
{
void add();
}

class C
{
public static void main(String args[])
{
A  x=new  A();
x.add();
}
}
```

In the above example the function add is not known their code until the time of call.

8) Message Communication :

An object oriented program consists set of objects that communicate with each other . It involves the following basic steps.

1. Creating class the define objects and their behaviour.
2. Creating objects from class definitions.
3. Establishing communication among objects.
4.Objects communicate with one another by sending and receiving information.

Sending Object Receiving Object

BENFITS OF OOP:

- Through inheritance we can eliminate redundant code
- The principle of data hiding helps the programmer to build secure programs
- Software complexity can be easily managed
- Object oriented system can be easily upgraded from small to large system.
- It is easy to partition the work in a project based on objects
- Message passing techniques for communication between objets.

APPLICATION OF OOP:

- The most popular application of OOP is windows.
- There is hundreds of windows system developed by using OOP techniques.

Application of OOP includes:

a. Real-time Systems
b. Simulation and medalling
c. Object oriented data base
d. Hyper text, hyper media
e. AI and expert system

CLASSES AND OBJECTS

Class: Class is a set of objects with similar properties and Common behaviour.

Object : Objects are basic run time entities. Once Class is defined we can create any number of objects.

Syntax for Class:-

```
class classname
 {
    field declaration;
    method declaration;
 }
```

1.Field declaration:

We Can create any number of field in a class. These variables are called instance variables.

Syntax:-

```
class Rectangle
{
int length;
int width;
}
```

2. Method Declaration:

A method which contains the actual implementation code. Methods are necessary for manipulating the data contained in the class. Methods are declared inside of the class. But immediately after declaration of variable.

The general form of method is,

Syntax:-

```
return type methodname (parameter list)
    {
        Body of the method;
    }
```

Example:-

```
class Rectangle
    {
        int length;
        int width;
        void get data (int x, int y)
          {
            length=x;
            width=y;
          }
    }
```

Creating Objects:-

Objects in java are created by using new operator. The new operator creates an object of the specified class and returns a reference to the object.

Syntax:-

```
classname  objectname = new  classname();
```

Example:-

```
Rectangle  rect1= new  Rectangle();
Rectangle  rect2= new  Rectangle();
```

ACCESSING CLASS MEMBERS:

In outside of the class, we cannot access the instance variables and methods directly. We must use object with dot(.) operator to access the class members.

Syntax:-

```
object  name.variable name;
object  name.method name;
```

Example:-

```
class Student
    {
        void display()
         {
            int  a=10, b=20, c;
            c=a+b;
            System.out.println("Sum of  c is  ="+c);
         }
    public static void main(String args[])
        {
            Student  s=new Student();
            s.display();
         }
    }
```

Output : **D:\> javac** Student**.java**

D:\> java Student

Sum of c is = 30

TYPES OF VARIABLES: Java variables can be classified into three types

1. Instance Variables
2. Static Variables
3. Local Variables

1. Instance Variable

A variable declared inside the class but outside the body of the method, is called instance variable. We call the instance variables by using the object.

Without creating an object there is no use of instance variable why because whenever we create an object then only memory will be allocated for instance variables.

// Example program instance variables

```
class InstanceDemo
  {
    int a=10; // Instance variable
    public static void main(String args[])
      {
```

```
        InstanceDemo d=new InstanceDemo();
        System.out.println(d.a);
        System.out.println("a value is :"+d.a);
    }
}
```

OUTPUT :

D:\>javac InstanceDemo.java

D:\>java InstanceDemo

10

10

2. Static Variable

A variable which is declared as static is called static variable. Static variables lso known as Class variables. Static variables are very flexible to access, we can access these static variables directly or by using object name, or by using class name.

// Example program static variables

```
class StaticDemo
  {
    static int a=10; // static variable
    public static void main(String args[])
      {
        System.out.println(a);
        StaticDemo d=new StaticDemo();
        System.out.println(d.a);
        System.out.println(StaticDemo.a);
      }
  }
```

OUTPUT :

D:\>javac StaticDemo.java

D:\>java StaticDemo

10

10

10

3. Local variables

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

this keyword :

'this' keyword is used to access the instance variables whenever the instance variable names and the local variables are same.

// Example program for Local variables

```
class LocalDemo
  {
    int a=10; // Instance variable
    void show()
      {
        int a=15; //Local variable
        System.out.println(a);
        System.out.println(this.a);
      }
    public static void main(String args[])
      {
        LocalDemo d=new LocalDemo();
        d.show();
        System.out.println(d.a);
      }
  }
```

OUTPUT :

```
D:\>javac LocalDemo.java
D:\>java LocalDemo
15
10
10
```

CONSTRUCTORS: **"**A Constructor is a special type of method that is used to initialize the object. Java constructor is invoked (activated) at the time of object creation. It constructs the values i.e., provides the data for object that's why it is called as constructor."

Rules for creating Constructor:-

There are basically two rules defined for the constructor.

- Constructor name must be same as the class name.
- Constructor must have no explicit return type.

Types of java Constructors:- There are two types of java constructors,

1. Default Constructor

2. Parameterized Constructor

1. Default Constructor:- A constructor that have no parameters is known as default constructor.

Purpose of Default Constructor:- It provides the default values to the object (like 0,null etc) depending on the type.

Example:-

```
class  Student
 {
      int  id;
      String  name;
      void display()
         {
            System.out.println(id+"  "+name);
         }
      public static void main(String args[])
       {
         Student s1=new Student();
         s1.display();
         Student  s2=new  Student();
         s2.display();
       }
}
```

Output :

```
D:\> javac Student.java
D:\> java Student
0     null
0     null
```

Above class we are not creating any constructor so, compiler provides a default constructor. 0 and null values are provided by default constructor.

2. Parameterized Constructor:- A Constructor that have parameters is known as parameterized constructors.

Purpose of Default Constructor:-

Parameterized constructor is used to provide different values to the distinct objects.

Example :

```
class Student
    {
        int  id;
        String  name;
        Student(int a,String b)
          {
             id=a;
             name=b;
          }
         void display()
          {
             System. out. println(id+"  "+name);
          }
         public static void  main(String args[])
           {
              Student  s1=new Student(10, "Ramu");
              s1.display();
              Student  s2=new  Student(11,"Syam");
              s2.display();
           }
    }
```

Output :

```
D:\> javac Student.java
D:\> java Student
10    Ramu
11    Syam
```

METHODS IN JAVA

METHODS IN JAVA

"A method in Java or Java Method is a collection of statements that perform some specific task and return the result to the caller". A Java method can perform some specific task without returning anything. Methods in Java allow us to reuse the code without retyping the code

TYPES OF METHODS IN JAVA:

"A method in Java or Java Method is a collection of statements that perform some specific task and return the result to the caller". A Java method can perform some specific task without returning anything. Methods in Java allow us to reuse the code without retyping the code.

There are two types of methods in Java

1. Predefined Method: In Java, predefined methods are the method that is already defined in the Java class libraries is known as predefined methods. It is also known as the standard library method or built-in method. We can directly use these methods just by calling them in the program at any point.

2. User-defined Method: The method written by the user or programmer is known as a user-defined method. These methods are modified according to the requirement.

METHOD DECLARATION: The method declaration provides information about method attributes, such as visibility, return-type, name, and arguments.

The general form of method is,

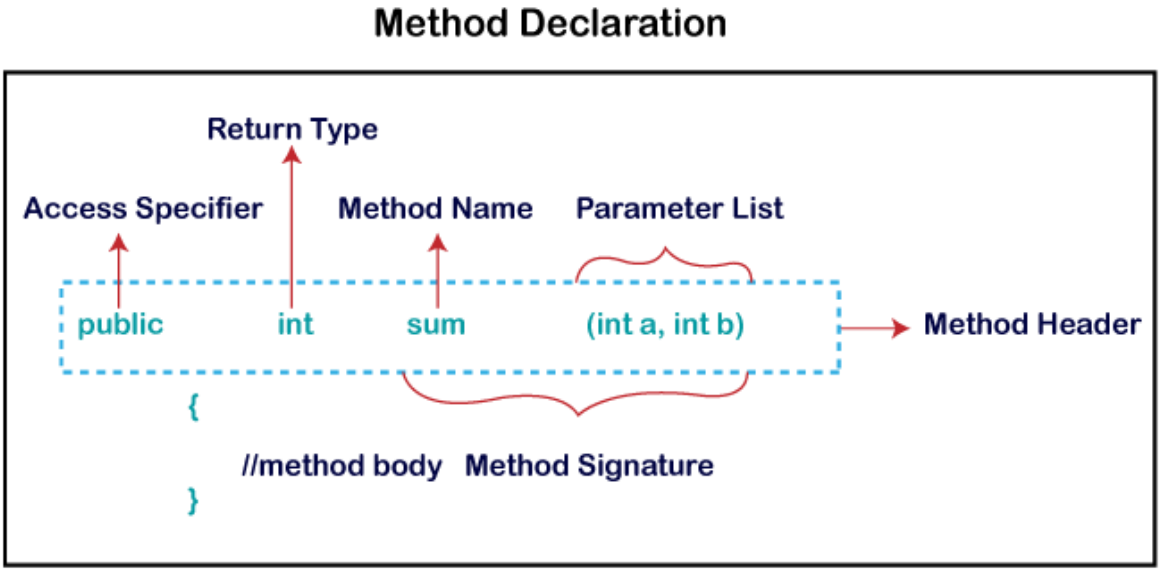
Syntax:-

```
return type methodname (parameter list)
  {
     Body of the method;
  }
```

Example:-

```
class Rectangle
    {
        int length;
        int width;
        void get data (int x, int y)
          {
             length=x;
             width=y;
           }
```

}



Method Signature: Every method has a method signature. It is a part of the method declaration. It includes the **method name** and **parameter list**.

Access Specifier: Access specifier or modifier is the access type of the method. It specifies the visibility of the method. Java provides **four** types of access specifier:

- **Public:** The method is accessible by all classes when we use public specifier in our application.
- **Private:** When we use a private access specifier, the method is accessible only in the classes in which it is defined.
- **Protected:** When we use protected access specifier, the method is accessible within the same package or subclasses in a different package.
- **Default:** When we do not use any access specifier in the method declaration, Java uses default access specifier by default. It is visible only from the same package only.

Return Type: Return type is a data type that the method returns. It may have a primitive data type, object, collection, void, etc. If the method does not return anything, we use void keyword.

Method Name: It is a unique name that is used to define the name of a method. It must be corresponding to the functionality of the method. Suppose, if we are creating a method for subtraction of two numbers, the method name must be **subtraction().** A method is invoked by its name.

Parameter List: It is the list of parameters separated by a comma and enclosed in the pair of parentheses. It contains the data type and variable name. If the method has no parameter, left the parentheses blank.

Method Body: It is a part of the method declaration. It contains all the actions to be performed. It is enclosed within the pair of curly braces.

STATIC MEMBERS IN JAVA

The **static keyword** in Java is used for memory management mainly. We can apply static keyword with variables, methods, blocks and nested classes.

1.Static variable :

If we declare any variable as static is known as static variable. The static variable can be used to refer to the common property of all objects (which is not unique for each object)

Example :

Company name of employees

College name of Students

Example program :

```
class Student
  {
     int rollno;//instance variable
     String name;
     static String college ="DNR COLLEGE";//static variable
     Student(int r, String n)
       {
          rollno = r;
          name = n;
       }
    void display()
       {
         System.out.println(rollno+" "+name+" "+college);
       }
    public static void main(String args[])
          {
             Student s1 = new Student(111,"Ramu");
             Student s2 = new Student(222,"Raju");
             s1.display();
             s2.display();
          }
  }
```

Output : **D:\> javac Student.java**

D:\> java Student

111 Ramu DNR COLLEGE

222 Raju DNR COLLEGE

2.Static method :

- If we apply static keyword to any method is known as static method.
- A static method can be invoked without the need for creating an instance of a class.
- A static method can access static data member and can change the value of it.

Restrictions for the static method :

There are two main restrictions for the static method. They are:

1. The static method cannot use non static data member or call non-static method directly.
2. "this" and "super" keywords are not used in this static context.

Example program

```
class Student
 {
   int rollno;
   String name;
   static String college = "DNR";
   //static method to change the value of static variable
   static void change()
       {
          college = "BV RAJU";
       }
   Student(int r, String n)
      {
          rollno = r;
          name = n;
      }
   //method to display values
   void display()
     {
       System.out.println(rollno+" "+name+" "+college);
     }
   public static void main(String args[])
```

```
    {
      change(); // Student.change();
      Student s1 = new Student(111,"Ramu");
      s1.display();
      Student s2 = new Student(222,"Raju");
      s2.display();
      Student s3 = new Student(333,"Ravi");
      s3.display();
    }
}
```

Output : **D:\> javac Student.java**

D:\> java Student

111 Ramu BV RAJU

222 Raju BV RAJU

333 Ravi BV RAJU

3.Static block :

- Is used to initialize the static data member.
- It is executed before the main method at the time of class loading.

Example program

```
class StaticBlock
    {
     static
        {
          System.out.println("static block is invoked");
        }
     public static void main(String args[])
         {
          System.out.println("Hello main");
         }
    }
```

Output : **D:\> javac StaticBlock.java**

D:\> java StaticBlock

static block is invoked

Hello main

4.Nested class :

A class can be made **static** only if it is a nested class.

1. Nested static class doesn't need reference of Outer class
2. A static class cannot access non-static members of the Outer class

Example program :

```
class StaticClassDemo
   {
      private static String str = "DNR COLLEGE";
     //Static class
     static class NestedDemo
         {
        //non-static method
        public void display()
             {
                 System.out.println(str);
          }
          }
      public static void main(String args[])
        {
            NestedDemo obj = new NestedDemo();
            obj.display();
  }
}
```

Output : **D:\> javac StaticClassDemo.java**

D:\> java StaticClassDemo

DNR COLLEGE

this keyword :

'this' keyword is used to access the instance variables whenever the instance variable names and the local variables are same. The main purpose of using this keyword in Java is to remove the confusion between class attributes and parameters that have same names.

// Example program for Local variables

```
class LocalDemo
```

```
{
    int a=10; // Instance variable
    void show()
      {
        int a=15; //Local variable
        System.out.println(a);
        System.out.println(this.a);
      }
    public static void main(String args[])
      {
        LocalDemo d=new LocalDemo();
        d.show();
        System.out.println(d.a);
      }
}
```

OUTPUT :

D:\>javac LocalDemo.java

D:\>java LocalDemo

15

10

10

SUPER KEYWORD: By using super keyword, we can access the class members of the super class from sub class. It is used to call super class methods from sub class. super keyword is used to call the super class constructor in the sub class constructor. The most common use of the super keyword is to eliminate the confusion between super classes and subclasses that have methods with the same name.

Example program:

Department of Computer Science Object Oriented Programming with JAVA Page 7

```
class A
{
int a,b;
void show()
{
System.out.println(&quot;b value in super class :&quot;+b);
```

```
}
}
class B extends A
{
int a,b;
B(int p,int q)
{
a=p;
super.b=q;
}
void show()
{
super.show();
System.out.println(&amp;quot;b value in super class :&amp;quot;+super.b);
System.out.println(&amp;quot;a value in sub class :&amp;quot;+a);
}
public static void main(String args[])
{
B obj=new B(10,20);
obj.show();
}
}
```

Output : D:\> javac B.java

D:\> java B

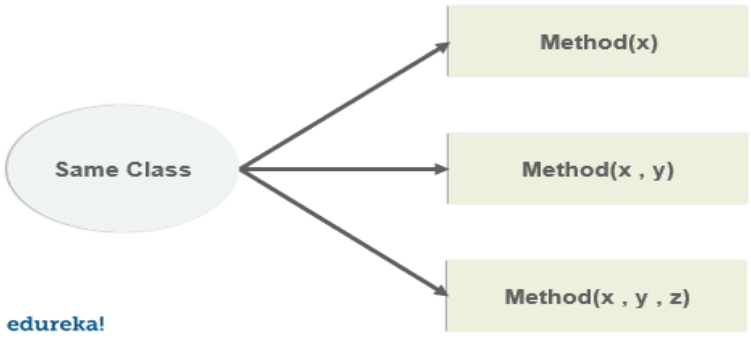
b value in super class : 20

b value in super class : 20

a value in sub class : 10

METHOD OVERLOADING

Definition : "If a class contains multiple methods with same name but different parameters is known as Method overloading."



Advantages:

Method overloading *increases the readability of the program.*

Different ways to overload the method

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

1. By changing number of arguments :

In this example, we have created two methods, first sum() method performs addition of two numbers and second sum method performs addition of three numbers.

Example program:

```
class Addition
  {
    void sum(int a,int b)
       {
          System.out.println(a+b);
       }
    void sum(int a,int b,int c)
       {
          System.out.println(a+b+c);
       }
    public static void main(String args[])
       {
          Addition a=new Addition();
           a.sum(10,20);
           a.sum(10,20,30);
       }
```

```
}
```

Output : D:\> javac Addition.java

D:\> java Addition

30

60

2. By changing the data type :

In this example, we have created two methods that differs in data type. The first sum method receives two integer arguments and second sum method receives two double arguments.

Example program:

```
class Addition
 {
   void sum(int a,int b)
      {
        System.out.println(a+b);
      }
   void sum(double a,double b)
      {
        System.out.println(a+b);
      }
   public static void main(String args[])
      {
        Addition a=new Addition();
         a.sum(10,20);
         a.sum(10.5,10.5);
      }
 }
```

Output : D:\> javac Addition.java

D:\> java Addition

30

21.0

METHOD OVERRIDING

Definition : "If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding.**"

Advantages:

1. Method overriding is used to provide the specific implementation of a method which is already provided by its super class.
2. Method overriding is used for runtime polymorphism.

Rules for Java Method Overriding

1. The method must have the same name as in the parent class
2. The method must have the same parameter as in the parent class.

Example program:

```
class Vehicle
  {
     void run()
       {
          System.out.println("Vehicle is running");
       }
  }
class Bike extends Vehicle
    {
       void run()
         {
            System.out.println("Bike is running");
         }
      public static void main(String args[])
        {
           Bike b= new Bike();
           b.run();
           Vehicle v=new Vehicle();
           v.run();
        }
  }
```

Output : D:\> javac Bike.java

D:\> java Bike

Bike is running

Vehicle is running

INHERITANCE

Definition: Acquiring properties from one class to another class is called Inheritance.

Inheritance is one of the important feature of object oriented programming. Inheritance is a mechanism, in which one object acquires all the properties and behaviours of parent objects. The idea behind inheritance is that we can create new classes built upon existing classes.

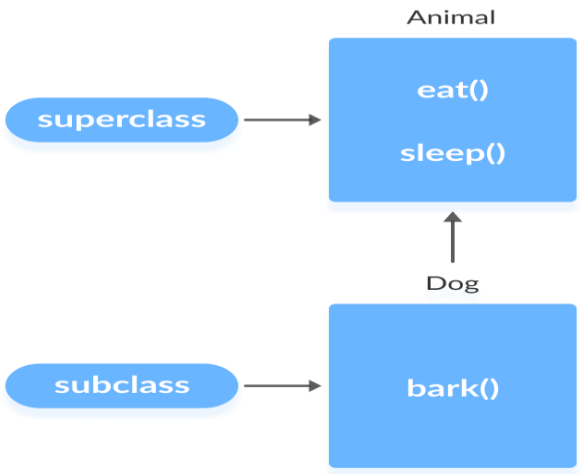
When we inherit from an existing class, we can reuse methods and fields of parent class and we can add new methods and fields also.

Syntax :

```
class subclass-name extends superclass-name
    {
        //methods and fields
    }
```

Example :

```
class Dog extends Animal
    {
      eat()
      sleep()
      bark()
    }
```



Advantages:

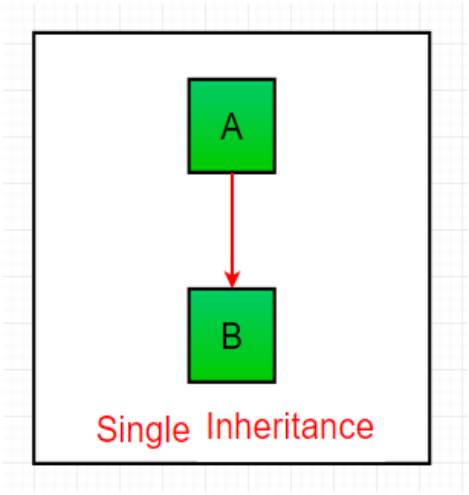
- For code re usability(To reuse the existing code without writing the new code)
- For method overriding(Run time polymorphism can be achieved)

TYPES OF INHERITANCE

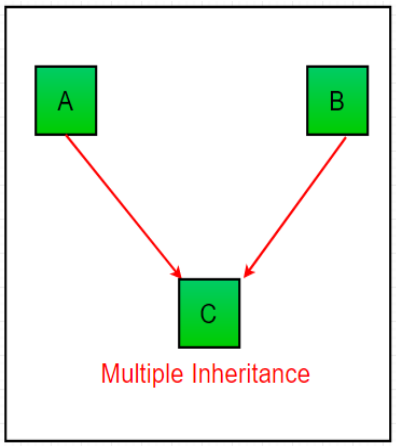
Java supports five types of inheritances as follows.

1. Single inheritance
2. Multiple inheritance
3. Hierarchical inheritance
4. Multi level inheritance
5. Hybrid inheritance

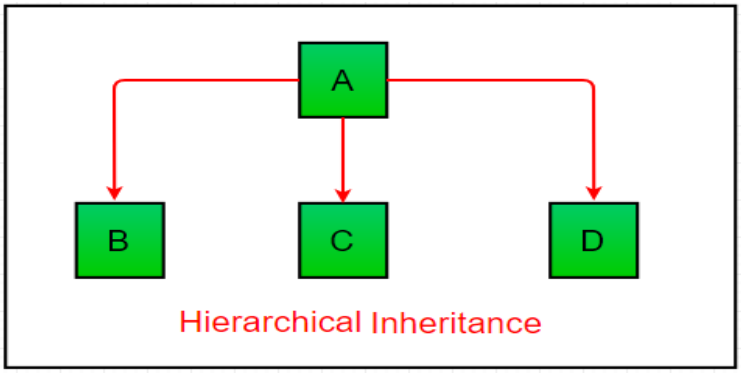
1. Single inheritance: Deriving a class from only one base class(super class) is known as Single inheritance. In below image, the class A serves as a base class for the derived class B.



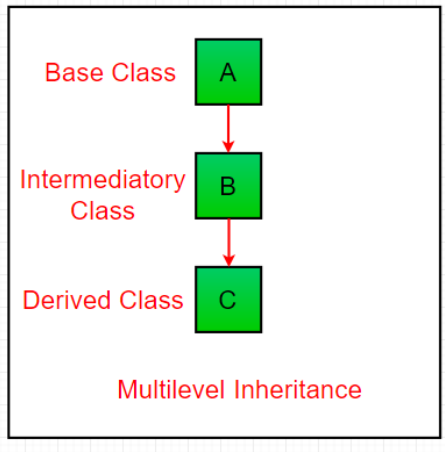
2. Multiple inheritance: Deriving a class from one or more base classes is known as Multiple inheritance. Java does **not** support multiple inheritance with classes. In java, we can achieve multiple inheritance only through Interfaces. In image below, Class C is derived from interface A and B.



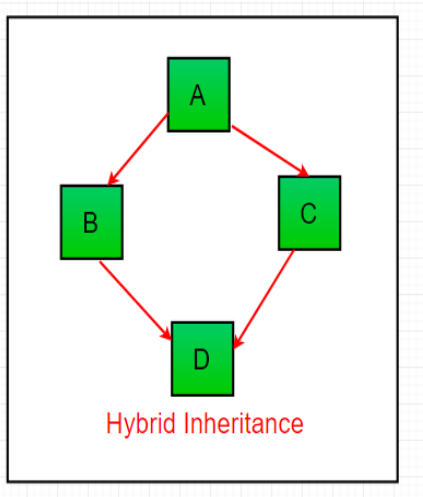
3. Hierarchical inheritance: Deriving several classes from one base class is known as Hierarchical inheritance. In below image, the class A serves as a base class for the derived class B,C and D.



4. Multi level inheritance: Deriving a class from another derived class is known as Multi level inheritance. In below image, the class A serves as a base class for the derived class B, which in turn serves as a base class for the derived class C. In Java, a class cannot directly access the grandparent's members.



5. Hybrid inheritance: Deriving a class from one or more derived classes that is derived from only one base class is known as Hybrid inheritance. In below image, the class A serves as a base class for the derived class B, C, which in turn B,C serves as a base classes for the derived class D. Since java doesn't support multiple inheritance with classes, the hybrid inheritance is also not possible with classes. In java, we can achieve hybrid inheritance only through Interfaces.



// Example program for Single inheritance

```
class A
 {
    void methodA()
       {
         System.out.println("Base class is Running");
       }
 }
class B extends A
  {
     void methodB()
        {
          System.out.println("Child class is Running");
        }
    public static void main(String args[])
       {
          B obj=new B();
          obj.methodA();
          obj.methodB();
      }
 }
```

Output : D:\> javac B.java

D:\> java B

Base class is Running

Child class is Running

D.N.R. COLLEGE (AUTONOMOUS): BHIMAVARAM

DEPARTMENT OF COMPUTER SCIENCE

OBJECT ORIENTED PROGRAMMING WITH JAVA - IV

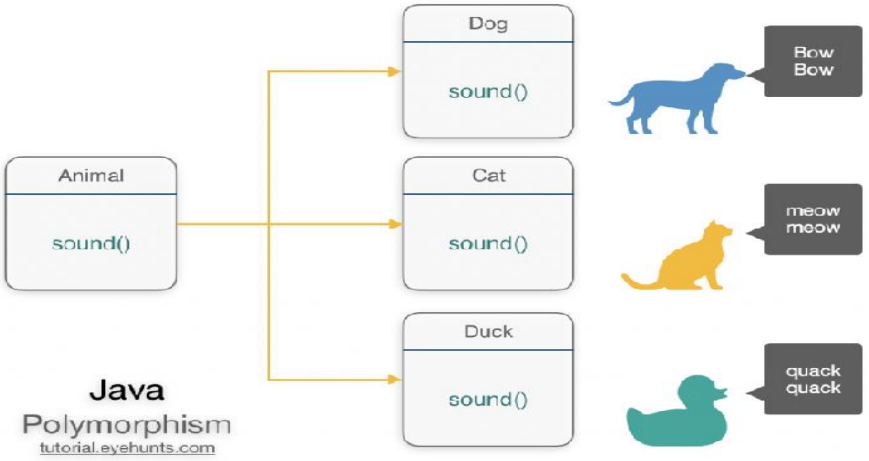
IV SEMESTER

UNIT III
POLYMORPHISM

POLYMORPHISM

Polymorphism means ability to take more than one form that is, the same entity (method or operator or object) can perform different operations in different scenarios. Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

Example: The sound of animals. We can have a common method sound but with this method, we are using sounds of different animals. The method sound will behave differently with respect to different animals.



There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

TYPES OF POLYMORPHISM

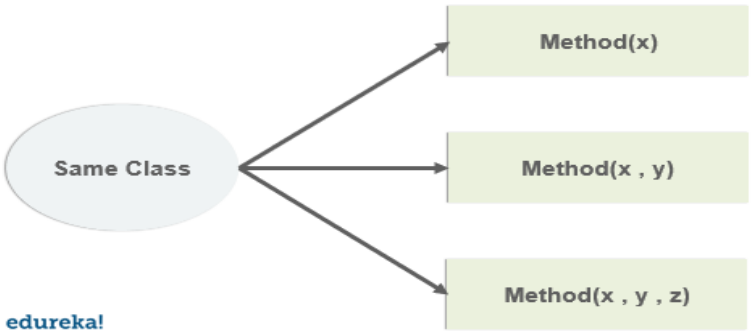
In Java polymorphism is mainly divided into two types:

- Compile-time Polymorphism
- Runtime Polymorphism

1: Compile-time polymorphism

It is also known as static polymorphism. This type of polymorphism is achieved by Method overloading or operator overloading.

Method Overloading: **Definition :** "If a class contains multiple methods with same name but different parameters is known as Method overloading."



Advantages:

Method overloading increases the readability of the program.

Different ways to overload the method

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

1. By changing number of arguments :

In this example, we have created two methods, first sum() method performs addition of two numbers and second sum method performs addition of three numbers.

Example program:

```
class Addition
 {
    void sum(int a,int b)
       {
          System.out.println(a+b);
       }
    void sum(int a,int b,int c)
       {
          System.out.println(a+b+c);
       }
    public static void main(String args[])
       {
          Addition a=new Addition();
           a.sum(10,20);
           a.sum(10,20,30);
```

```
    }
  }
```

Output : D:\> javac Addition.java

D:\> java Addition

30

60

2. By changing the data type :

In this example, we have created two methods that differs in data type. The first sum method receives two integer arguments and second sum method receives two double arguments.

Example program:

```
class Addition
 {
   void sum(int a,int b)
      {
         System.out.println(a+b);
      }
   void sum(double a,double b)
      {
         System.out.println(a+b);
      }
   public static void main(String args[])
      {
         Addition a=new Addition();
          a.sum(10,20);
          a.sum(10.5,10.5);
      }
 }
```

Output : D:\> javac Addition.java

D:\> java Addition

30

21.0

2: Runtime polymorphism: It is a process in which a function call to the overridden method is resolved at Runtime. This type of polymorphism is achieved by Method Overriding.

Method overriding

Definition : "If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding.**"

Advantages:

1. Method overriding is used to provide the specific implementation of a method which is already provided by its super class.
2. Method overriding is used for runtime polymorphism.

Rules for Java Method Overriding

1. The method must have the same name as in the parent class
2. The method must have the same parameter as in the parent class.

Example program:

```
class Vehicle
  {
     void run()
       {
          System.out.println("Vehicle is running");
       }
  }
class Bike extends Vehicle
    {
      void run()
        {
           System.out.println("Bike is running");
        }
      public static void main(String args[])
        {
           Bike b= new Bike();
           b.run();
           Vehicle v=new Vehicle();
           v.run();
        }
  }
```

Output : D:\> javac Bike.java

D:\> java Bike

Bike is running

Vehicle is running

TYPE CASTING

TYPE CASTING: The process of converting one data type to another data type is called Type casting. In java there are two types of type casting.

1. Implicit type casting
2. Explicit type casting

1. Implicit type casting : Converting smaller data type to bigger data type.

a. Program is responsible

b. There is no loss of information

c. It is also called as Up casting or Widening Casting

2. Explicit type casting: Converting bigger data type to smaller data type.

a. Program is responsible

b. There is loss of information

c. It is also called as Down casting or Narrowing

Syntax of Type casting :

Data type variable_name 1 = (Type)variable name 2;

Ex :

```
int m=10;
long count=(long)m;  // Implicit type casting
byte b=(byte)m;   // Explicit type casting
```

// Example program for Type casting

```
class TypeCasting
```

```
{
   public static void main(String args[])
    {
     int a=10;
     double b=20.14;
     float c=(float)a; // implicit type casting
     System.out.println("c=   "+c);
     int d=(int)b; // explicit type casting
     System.out.println("d  "+d);
    }
}
```

OUTPUT

```
F:\> javac TypeCasting.java
F:\> java TypeCasting
c=10.0
d=20
```

ABSTRACT CLASSES

ABSTARCTION IN JAVA

Abstraction: "Abstraction is a process of hiding implementation details and showing the functionality to the user."

Ways to achieve Abstraction:

There are two ways to achieve the abstraction in java

1. Abstract class
2. Interface

1. Abstract class : A class which is declared as abstract is known as an **abstract class**. It can have abstract and non-abstract methods. Abstract class needs to be extended and its method can be implemented.

Syntax: abstract class_name

```
{
  abstract method name;
}
```

Example: abstract Test

```
{
```

```
    abstract void show();
}
```

2. Abstract method: A method that can be declared with the keyword abstract is known as abstract method. It doesn't have any implementation.

Syntax:
```
abstract class_name
{
  abstract method name;
}
```

Example:
```
abstract Test
{
  abstract void show();
}
```

Example program

```
abstract class Shape
 {
  abstract void draw();//Implementation is provided by others i.e. unknown by end user
 }
class Rectangle extends Shape
 {
  void draw()
   {
    System.out.println("Drawing rectangle");
   }
 }
class Circle extends Shape
 {
  void draw()
   {
    System.out.println("Drawing circle");
   }
}
//In real scenario, method is called by programmer or user
class AbstractionDemo
```

```
{
  public static void main(String args[])
    {
      Shape s1=new Rectangle();
      s1.draw();
      Shape s2=new Circle();
      s2.draw();
    }
}
```

Output : D:\> javac AbstractDemo.java

D:\> java AbstractDemo

Drawing rectangle

Drawing circle

INTERFACES

Interface: An **interface in Java** is a blueprint of a class. Like classes, interface contains methods and variables. But the major difference between class and interface is that interfaces contain only abstract methods, static and final variables. It contains only abstract methods so, in java interface does not contains method body.

Advantages:

1. It is used to achieve abstraction by interface.
2. By using interface we can support the functionality of multiple inheritance.

Note:

1. Interface fields are public, static and final by default.
2. Interface methods are public and abstract.

Syntax of interface:

```
Interface interface name
    {
        Variable declaration;
        Method declarations;
    }
```

Here **interface** is the keyword and the **interface name** is just like a class name.

Declaring variables in interface:

Syntax:

static final data type variable name = value;

Example:

```
interface item
   {
      static final int code = 101;
      static final String name = “fan”;
      void display();
   }
```

ACCESSING INTERFACE VARIABLES:

The variables of an interface are always declared as “final”. Final variables are variables whose values are constants that cannot be changed.

Example program

```
interface SelectColour
 {
  int blue=4;
  int yellow=5;
  int red=6;
  public void choose(int colour);
 }
class Select implements SelectColour
 {
  public void choose(int colour)
    {
     switch(colour)
       {
         case blue: System.out.println("Blue");
                    break;
         case yellow: System.out.println("Yellow");
                    break;
         case red: System.out.println("Red");
                    break;
       }
    }
```

```
  public static void main(String args[])
   {
    int a,b,c;
    a=Integer.parseInt(args[0]);
    b=Integer.parseInt(args[1]);
    c=Integer.parseInt(args[2]);
    Select s=new Select();
    s.choose(a);
    s.choose(b);
    s.choose(c);
   }
}
```

Output : D:\> javac Select.java

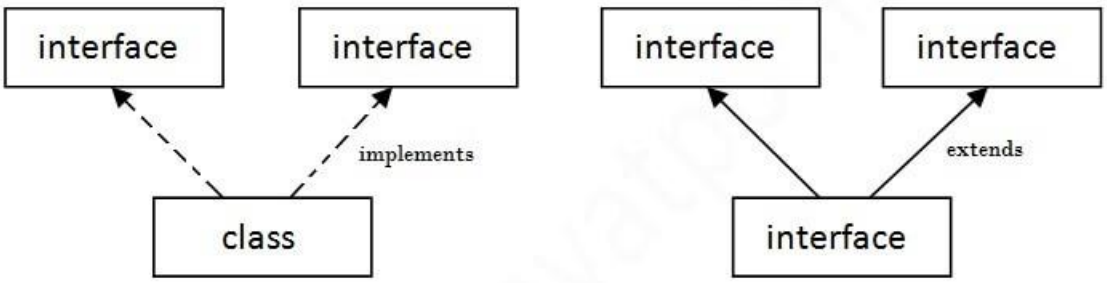
D:\> java Select 4 5 6

Blue

Yellow

Red

MULTIPLE INHERITANCE IN JAVA BY INTERFACE: If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.



Multiple Inheritance in Java

Example program:

```
interface Printable
   {
       void print();
```

```
    }
interface Showable
    {
        void show();
    }
class Test5 implements Printable,Showable
    {
        public void print()
            {
                System.out.println("Hello");
             }
        public void show()
            {
                System.out.println("Welcome");
            }
        public static void main(String args[])
            {
                Test5 t = new Test5();
                t.print();
                t.show();
            }
     }
```

Output : D:\> javac Test5.java

D:\> java Test5

Hello

Welcome

Note: Multiple inheritance is not supported through class in java, but it is possible by an interface, why?

Multiple inheritance is not supported in the case of class because of ambiguity. However, it is supported in case of an interface because there is no ambiguity. It is because its implementation is provided by the implementation class.

Example program:

```
interface Printable
```

```
    {
        void print();
    }
interface Showable
    {
       void print();
    }
class Test6 implements Printable, Showable
   {
       public void print()
         {
            System.out.println("Hello");
         }
       public static void main(String args[])
         {
            Test6 t = new Test6();
            t.print();
        }
   }
```

Output : D:\> javac Test6.java

D:\> java Test6

Hello

As we can see in the above example, Printable and Showable interface have same methods but its implementation is provided by class TestTnterface1, so there is no ambiguity.

DIFFERENCES BETWEEN ABSTRACT CLASS AND INTERFACE

Abstract class	Interface
1.If we are talking about the implementation but not completely then we should go for abstract class	1.If we don't know anything about implementation just we have requirement specification then we should go for interface.
2.Abstact class can have abstract and non abstract methods	2. Interface can have only abstract methods.

3. Every method in abstract class need not to be public and abstract.	3.Inside interface, every method is always public, abstract whether we are specified or not
4. Abstract class can have final, non final, static, non static variables.	4. Interface has only static and final variables.
5. Abstact class does not support multiple inheritance.	5. Interface supports multiple inheritance.
6. Abstract class can provide implementation of the interface.	6. Interface cannot provide implementation of the abstract class.

PACKAGES

Package: A **java package** is a group of similar types of classes, interfaces and sub-packages. Package can be divided into two types,

1. Built-in packages
2. User defined packages

Advantage of Java Package

- Java package is used to categorize the classes and interfaces so that they can be easily maintained.
- Java package provides access protection.
- Java package removes naming collision.

JAVA API PACKAGES (PREDEFINED PACKAGES)

java.lang	Language support classes. They include classes for primitive types, string, math functions, thread and exceptions.
java.util	Language utility classes such as vectors, hash tables, random numbers, data, etc.
java.io	Input/output support classes. They provide facilities for the input and output of data.
java.applet	Classes for creating and implementing applets.
java.net	Classes for networking. They include classes for communicating with local computers as well as with internet servers.
java.awt	Set of classes for implementing graphical user interface. They include classes for windows, buttons, lists, menus and so on.

PROCEDURE TO CREATE PACKAGE

1. Declare “package packagename” at the beginning of the program.

Syntax:- package mypack;

2. Define a class that is to be put in a package and it must be declared as “public” class.
3. Create a sub directory under directory where the main source file is stored.
4. Compile the source file then .class file will be created in the sub directory.
5. The sub directory name must match the package name exactly.

HOW TO ACCESS PACKAGES

There are three ways to access the package from outside the package.

1. import package.*;
2. import package.classname;
3. Fully qualified name.

1. import package.*;:- If we use package.* then all the classes and interfaces of this package will be accessible but not sub packages. The import keyword is used to make the classes and interface of another package accessible to the current package.

2. import package.classname;:- If we import package.classname then only declared class of this package will be accessible.

3. Fully qualified name:- If we use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But you need to use fully qualified name every time when you are accessing the class or interface.

Example program:

//Save this file as emp.java

```
package mypack;
public class emp
  {
     int eid;
     String ename;
     double sal;
     public emp()
        {
           eid=111;
           ename="Ravi";
           sal=50000;
        }
     public void display()
```

```
        {
            System.out.println("eid = "+eid);
            System.out.println("ename = "+ename);
            System.out.println("sal = "+sal);
        }
    }
```

//Save this file as emp2.java

```
import mypack.*;
class emp2 extends emp
    {
        String city;
        emp2()
          {
             city="BVRM";
          }
        public void display()
           {
              super.display();
              System.out.println("city = "+city);
           }
        public static void main(String args[])
           {
              emp2 e2=new emp2();
              e2.display();
           }
    }
```

Output:

```
D:\>md mypack
D:\>cd mypack
D:\mypack\>javac emp.java
D:\mypack\>cd..
D:\>javac emp2.java
D:\>java emp2
eid = 111
```

ename = Ravi

sal = 50000.0

city =BVRM

ADVANTAGE OF JAVA PACKAGE

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision.

4) Packages provide reusability of code .

5) To bundle classes and interfaces.

6) We can create our own Package or extend already available Package.

JAR FILES: A Java ARchive (JAR) file is a file format that combines many files into one. The Java environment differs from other programming environments in that the Java compiler does not generate machine code for a hardware-specific instruction set. Instead, the Java compiler converts Java source code into Java virtual machine instructions, which Java class files store. You can use JAR files to store class files. The class file does not target a specific hardware platform, but instead targets the Java virtual machine architecture.

You can use JAR as a general archiving tool and also to distribute Java programs of all types, including applets. Java applets download into a browser in a single Hypertext Transfer Protocol (HTTP) transaction rather than by opening a new connection for each piece. This method of downloading improves the speed at which an applet loads on a Web page and begins functioning.

The JAR format supports compression, which reduces the size of the file and decreases download time. Additionally, an applet author may digitally sign individual entries in a JAR file to authenticate their origin.

EXCEPTION HANDLING

Exception: An exception (or exceptional event) is an error that arises during the execution of a program. When an Exception occurs the normal flow of the program is disrupted and the program/Application terminates abnormally. It is an object which is thrown at runtime.

An exception handling in java is one of the powerful mechanisms to handle runtime errors. So that the normal flow of application can be maintained.

Advantages of Exception handling:

- The main advantage of exception handling is to maintain the normal flow of the application.

TYPES OF JAVA EXCEPTIONS

There are mainly two types of exceptions

1. Checked Exception
2. Unchecked Exception

1.Checked exceptions: Checked exceptions are checked at compile-time. It means if a method is throwing a checked exception then it should handle the exception using try-catch block or it should declare the exception using throws keyword, otherwise the program will give a compilation error. In checked exception try, catch blocks are compulsory.

Different types of checked exceptions are

- Illegal exceptions
- Class not found exception
- Error not found exception
- Method not found exception

2.Unchecked exception: Unchecked exceptions are not checked at compile time. It means if your program is throwing an unchecked exception and even if you didn't handle/declare that exception, the program won't give a compilation error. Most of the times these exception occurs due to the bad data provided by user during the user-program interaction. In Unchecked exception try, catch blocks are not compulsory.

Different types of Unchecked exceptions are

- Array index out of bounds exception
- Arithmetic format exception
- Number format exception
- Null point exception

TYPES OF ERRORS

Error is an illegal operation performed by the user which results in the abnormal working of the program. Programming errors often remain undetected until the program is compiled or executed. Errors are classified into two types.

1. Compiled time errors
2. Runtime errors

1. Compiled time errors: All syntax errors will be detected and displayed by the java compiler. These types of errors are known as Compile time errors. These errors will be detected by java compiler and displays the error onto the screen while compiling.

The most common compile time errors are,

a. Missing semicolon (;)
b. Missing braces in classes and method
c. Misspelling of identifiers and keywords
d. Missing double code in strings
e. Use of undeclared variable
f. Bad references to objects
g. Use of = in place of == operator

2. Runtime errors: Run Time errors are occurs during execution of program. Java compiler will not detect Run Time errors. Only Java Virtual Machine (JVM) will detect it while executing the program. Run time errors occurs due to wrong logic or may terminate due to errors.

The most common Run time errors are,

a. Dividing an integer by zero
b. Expecting an element i.e., out of bounds an array
c. Accessing character i.e., Out of bounds of the string
d. Converting invalid string to number
e. Attempting to use a negative size for an array
f. Trying to illegally change the state of a thread
g. Passing parameter i.e., not in a valid range

EXCEPTION HANDLING IN JAVA

If an exception occurs, which has not been handled by programmer then program execution gets terminated and a system generated error message is shown to the user. The purpose of exception handling mechanism is to detect and report an exceptional circumstance. So that appropriate action can be taken. The exception handling code that performs the following tasks,

a. Find the Exception
b. Through the exception
c. Catch the exception
d. Handling the exception

The error handling code basically consists of two segments. One is to detect errors and to throw exceptions and another one is to catch exceptions and to take appropriate actions.

Example program Exception handling

class Error

```
{
    public static void main(String args[])
        {
            int a=10;
        int b=5;
        int c=5;
        int x,y;
        try
          {
            x=a/(b-c);
          }
            catch(ArithmeticException e)
              {
            System.out.println("Divison by zero");
              }
                y=a/(b-c);
            System.out.println("y="+y);
         }
  }
```

Output : D:\> javac Error.java

D:\> java Error

Divison by zero

Exception in thread "main" java.lang.ArithmeticException: / by zero

at error.main(error.java:17)

D.N.R. COLLEGE (AUTONOMOUS): BHIMAVARAM

DEPARTMENT OF COMPUTER SCIENCE

OBJECT ORIENTED PROGRAMMING WITH JAVA - IV

IV SEMESTER

UNIT IV
STREAMS

Stream: Streams are the sequence of data that are read from the source and written to the destination. An input stream is used to read data from the source. And, an output stream is used to write data to the destination.

STREAM CLASSES IN JAVA

All streams in Java are represented by classes in java.io package. This package contains a lot of stream classes that provide abilities for processing all types of data.

We can classify these stream classes into two basic groups based on the date type. They are as follows:

- Byte Stream Classes (support for handling I/O operations based on bytes)
- Character Stream Classes (support for managing I/O operations on characters)

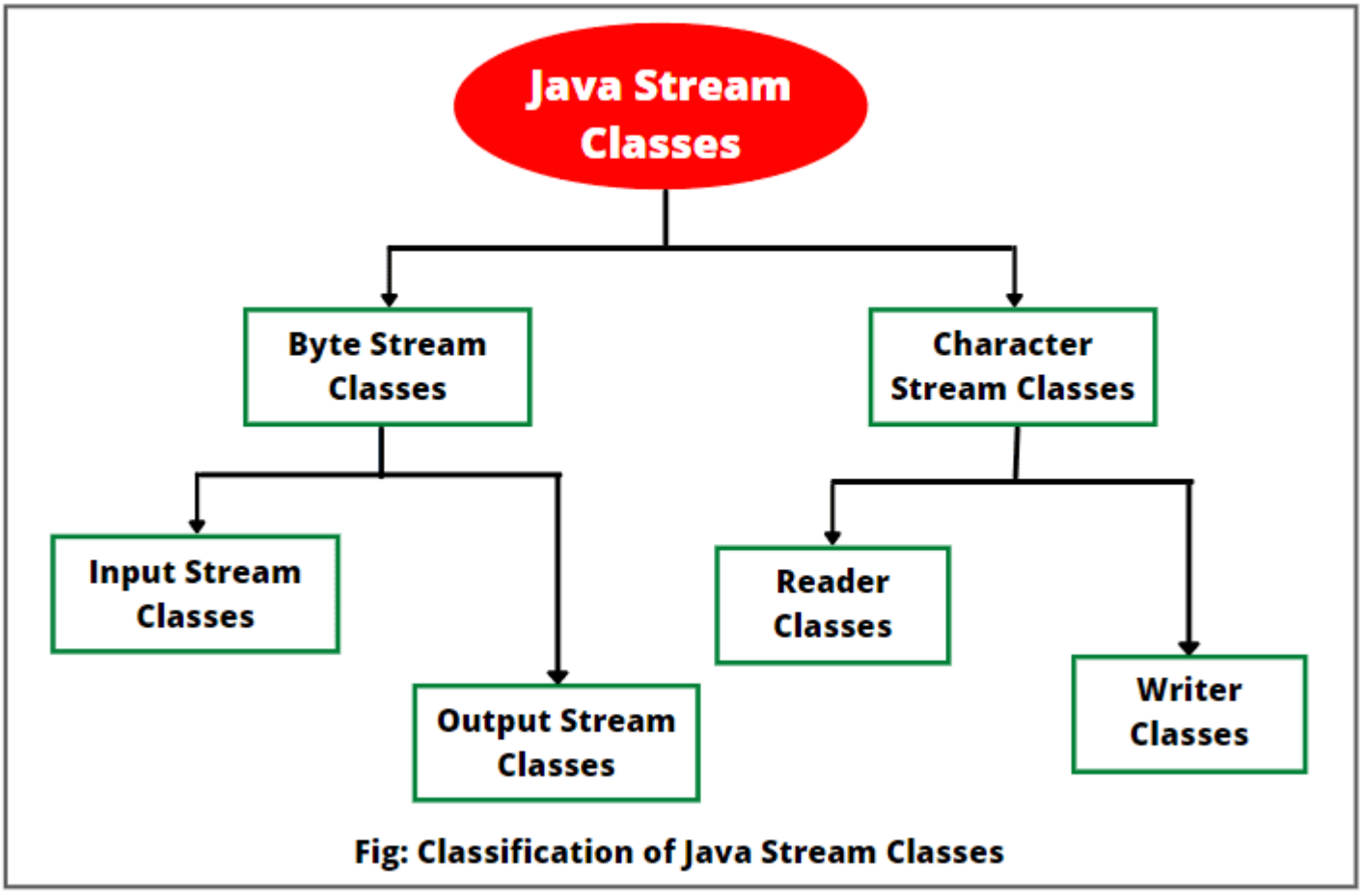


Fig: Classification of Java Stream Classes

1. Byte Stream Classes

There are two kinds of byte stream classes in Java. They are as follows:

a. InputStream classes

b. OutputStream classes

a. InputStream Classes in Java: InputStream class is an abstract class. It is the root class for reading binary I/O data. It is the superclass of all classes representing an input stream of bytes. Since InputStream class is an abstract class, we cannot create an object of this class. We must use subclasses of this class to create an object.

The several subclasses of Java InputStream class can be used for performing several input functions. They are listed with a brief description in the below table:

InputStream Subclass	Description
BufferedInputStream	It adds buffering abilities to an input stream. In simple words, it buffered input stream.
ByteArrayInputStream	Input stream that reads data from a byte array.
DataInputStream	It reads bytes from the input stream and converts them into appropriate primitive-type values or strings.
FileInputStream	This input stream reads bytes from a file.
ObjectInputStream	Input stream for objects

b. OutputStream Classes in Java: OutputStream class is an abstract class. It is the root class for writing binary data. It is a superclass of all classes that represents an output stream of bytes.

Since like InputStream, OutputStream is an abstract class, therefore, we cannot create object of it. The hierarchy of classification of OutputStream classes has shown in the above diagram.

The several subclasses of OutputStream class in Java can be used for performing several output functions. They are listed with a brief description in the below table:

OutputStream Subclass	Description
BufferedOutputStream	It adds buffering abilities to an output stream. In simple words, it buffered output stream.
ByteArrayOutputStream	Output stream that writes data to a byte array.
DataOutputStream	It converts primitive-type values or strings into bytes and outputs bytes to the stream.
FileOutputStream	It writes byte stream into a file.
ObjectOutputStream	Output stream for objects

2. CharacterStream classes: CharacterStream classes in Java are used to read and write 16-bit Unicode characters. In other words, character stream classes are mainly used to read characters from the source and write them to the destination. They can perform operations based on characters, char arrays, and Strings.

Types of CharacterStream classes in Java

Like byte stream classes, character stream classes also contain two kinds of classes. They are named as:

a. Reader Stream classes
b. Writer Stream classes

a. Reader Stream Classes: Reader stream classes are used for reading characters from files. Reader is an abstract superclass for all other subclasses such as BufferedReader, StringReader, CharArrayReader, etc.

b. Writer Stream Classes: Writer stream classes are used to write characters to a file. In other words, They are used to perform all output operations on files. Writer stream classes are similar to output stream classes with only one difference that output stream classes use bytes to write while writer stream classes use characters to write

CREATE A FILE USING FILEOUTPUTSTREAM

Java FileOutputStream Class: Java FileOutputStream is an output stream used for writing data to a file. If we have to write primitive values into a file, use FileOutputStream class. We can write byte-oriented as well as character-oriented data through FileOutputStream class. But, for character-oriented data, it is preferred to use FileWriter than FileOutputStream.

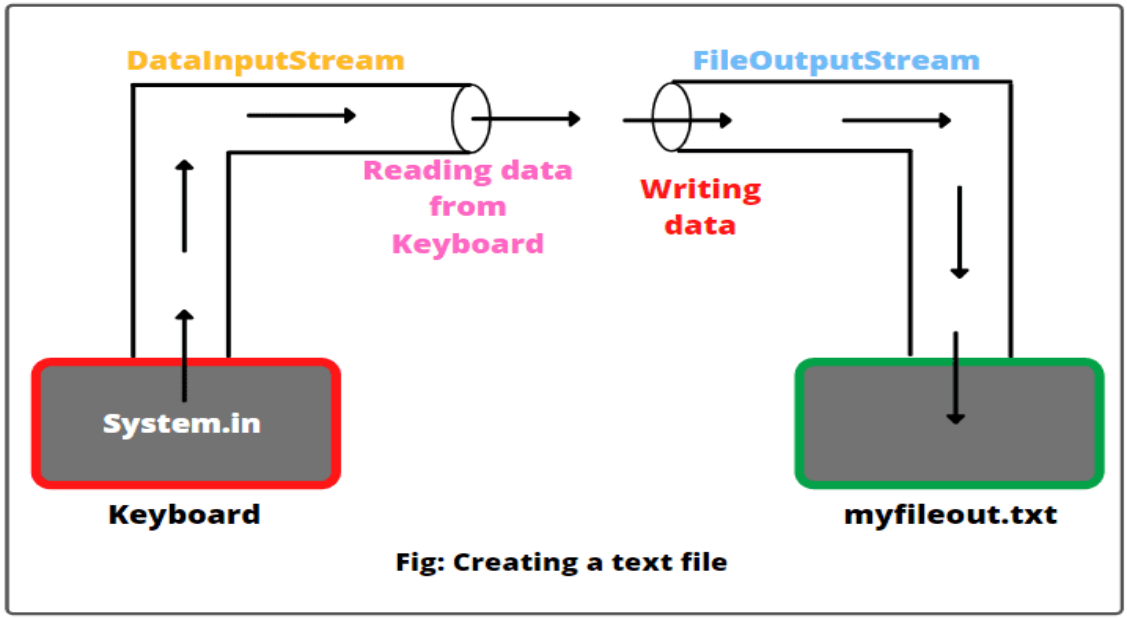


Fig: Creating a text file

FileOutputStream class declaration

public class FileOutputStream **extends** OutputStream

Example program for creating a file using Java FileOutputStream for write String

```
import java.io.FileOutputStream;
public class FileOutputStreamExample
  {
      public static void main(String args[])
          {
            try
              {
                FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
                String s="Welcome to javaTpoint.";
                byte b[]=s.getBytes();//converting string into byte array
                fout.write(b);
                fout.close();
                System.out.println("success...");
```

```
            }
            catch(Exception e)
              {
                 System.out.println(e);
              }
        }
}
```

Output: D:>\ javac FileOutputStreamExample .java

D:>\ java FileOutputStreamExample

success

Welcome to javaTpoint

READING DATA FROM A FILE USING FILEINPUTSTREAM

Java FileInputStream class obtains input bytes from a file. It is used for reading byte-oriented data (streams of raw bytes) such as image data, audio, video etc. You can also read character-stream data. But, for reading streams of characters, it is recommended to use FileReader class.

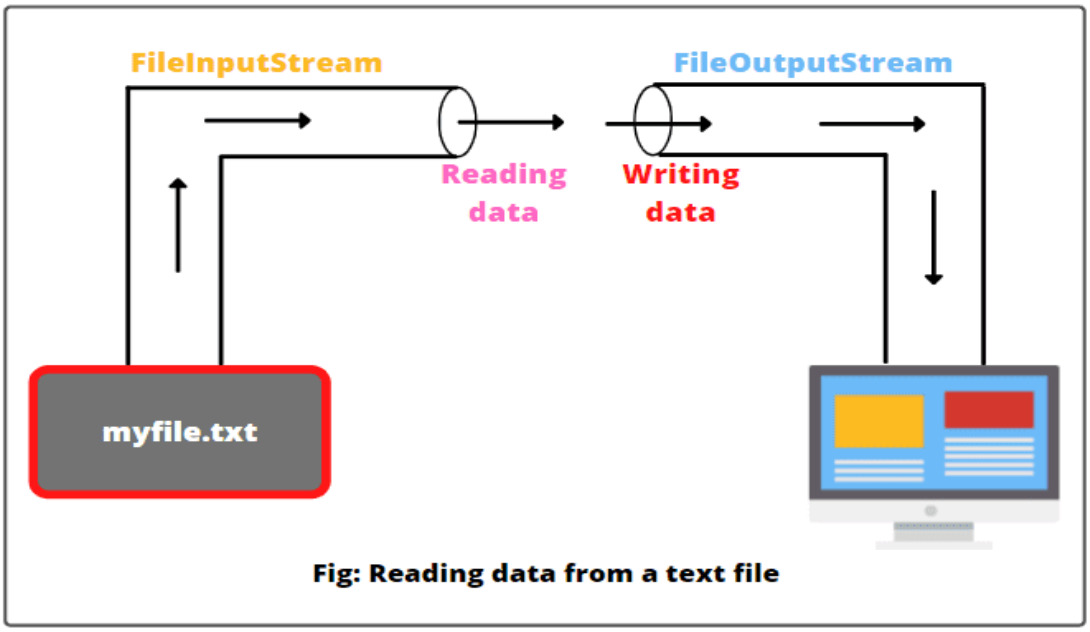


Fig: Reading data from a text file

Java FileInputStream class declaration

public class FileInputStream extends InputStream

Example program for read all characters from a file using FileInputStream

```
import java.io.FileInputStream;
public class DataStreamExample
    {
```

```
    public static void main(String args[])
      {
        try
          {
            FileInputStream fin=new FileInputStream("D:\\testout.txt");
            int i=0;
           while((i=fin.read())!=-1)
               {
                    System.out.print((char)i);
               }
          fin.close();
         }
         catch(Exception e)
           {
           System.out.println(e);
           }
      }
}
```

Output: D:>\ javac DataStreamExample.java

D:>\ javac DataStreamExample.java

Welcome to javaTpoint

ZIPPING AND UNZIPPING FILES IN JAVA

Zipping : ZIP is a common file format that compresses one or more files into a single location. It reduces the file size and makes it easier to transport or store. A recipient can unzip (or extract) a ZIP file after transport and use the file in the original format.

Unzipping: To unzip a zip file, we need to read the zip file with ZipInputStream and then read all the ZipEntry one by one. Then use FileOutputStream to write them to file system. We also need to create the output directory if it doesn't exists and any nested directories present in the zip file.

THREADS

Thread: Thread means separate flow of execution.

Multithreading: Multithreading is java feature where a program is divided into two or more sub programs. This can be implemented at the same time in parallel.

DIFFERENT WAYS OF CREATING THREADS

There are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface

1. By extending Thread class:

This is the first way to create a thread by a new class that extends Thread class and create an instance of that class. The extending class must override run() method. It includes the following steps,

i. Declare the class as extending the Thread class.

ii. Implement the run() method that is responsible for executing the sequence of code that the thread will execute.

iii. Create a thread object and call the start() method to initiate the thread execution.

Syntax:

```
class MyThread extends Thread
    {
        ----------
        ----------
    }
```

Example program:-

```
class MyThread extends Thread
  {
    public void run()
    {
        for(int i=0;i<=5;i++)
          {
            System.out.println("Child thread");
          }
    }
  }
class ThreadDemo
```

```
{
    public static void main(String args[])
      {
        MyThread t = new  MyThread();
        t.start();
        for(int i=0;i<=5;i++)
          {
            System.out.println("Main method");
          }
    }
}
```

Output : D:\> javac ThreadDemo.java

D:\> java ThreadDemo

Main method

Main method

Main method

Main method

Main method

Child thread

Child thread

Child thread

Child thread

Child thread

2. By implementing Runnable interface:

The easiest way to create a thread is to create a class that implements the Runnable interface. After implementing Runnable interface, the class needs to implement the run() method. It includes the following steps,

i. Declare the class as implementing the Runnable interface

ii. Implementing the run method

iii. Create a thread by defining an object that is instantiated from this runnable class as the target of the thread.

iv. Call the start() method to initiate the thread execution.

Syntax: class class name implements Runnable

```
{
    ----------
    ----------
}
```

Example program:

```
class MyRunnable implements Runnable
    {
      public void run()
      {
          for(int i=0;i<=5;i++)
            {
              System.out.println("Child thread");
            }
      }
  }
class ThreadDemo1
    {
       public static void main(String args[])
         {
           MyRunnable r= new  MyRunnable();
           Thread t=new Thread(r);
           t.start();
           for(int i=0;i<=5;i++)
                {
                System.out.println("Main method");
                }
        }
    }
```

Output : D:\> javac ThreadDemo1.java

D:\> java ThreadDemo1

Main method

Main method

Main method

Main method

Main method

Child thread

Child thread

Child thread

Child thread

Child thread

LIFE CYCLE OF A THREAD : During the life time of a thread there are many states it can enter, they can be include,

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

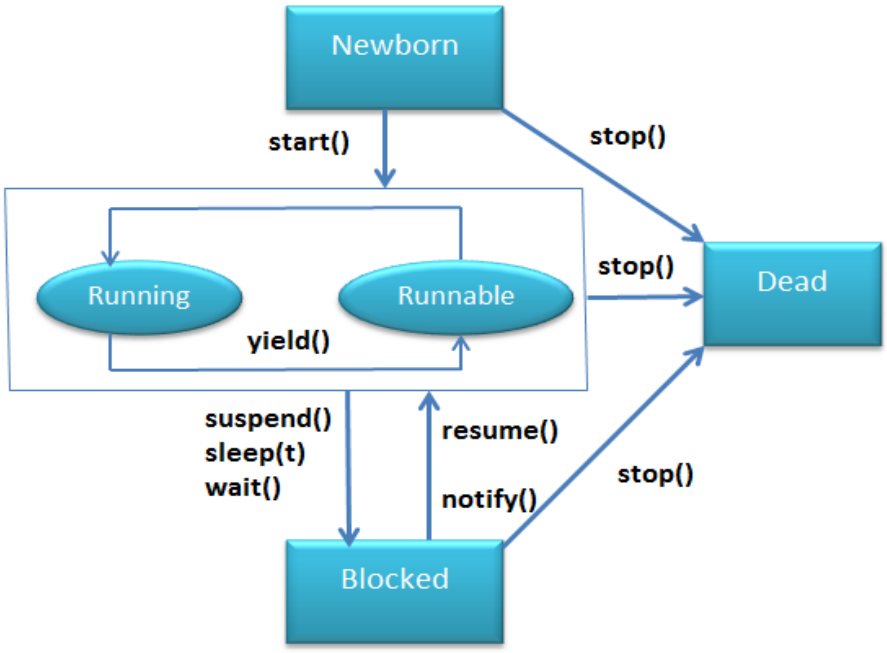


Fig: Life Cycle of Thread

1.New born state: When an instance of thread class is created then the thread enters into new state.

Thread t=new Thread();

We have to invoke the start() method to start the thread

t.start();

2.Runnable state: When start() method is invoked, then the thread enters into runnable state. The Runnable state means that the thread is ready for execution and is waiting for the availability of the processor.

yield() method: This method pauses the currently executing thread temporarily for giving a chance to the remaining waiting threads of priority to execute.

If there are no waiting threads or all waiting threads have a lower priority then the same thread will continue its execution.

3.Running state: Running means that the processor has given its time to that thread for its execution.

4.Block state: A thread is in the *blocked* state when it is currently not eligible to run.

suspend() method: A thread is suspended with a call suspend(). The suspended thread can be resumed using the resume() method.

wait() method: A thread can be made wait for some specified condition is to be satisfied by calling wait() method.

5.Dead state: Once the running thread finished executing, it's state is changed to Dead and it is considered to be not alive. We can also kill the thread by sending stop message at any state. i.e. it causes premature death. To stop a thread we use the following syntax:

Syntax: t1.stop();

THREAD PRIORITIES

- Each thread have a priority. Priorities are represented by a number between 1 and 10. Where 1 is the minimum priority, 10 is the maximum priority.
- Default priority for main thread is always 5.
- In most cases, thread schedular schedules the threads according to their priority (known as preemptive scheduling). But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.
- The Thread class defines the following constants to represent some standard priorities.

  Thread.MIN_PRIORITY=1

  Thread.MAX_PRIORITY=10

  Thread.NORM_PRIORITY=5
- The scheduler will use priorities while allocating processor.
- The thread which is having highest priority will get chance first for execution.

- If two threads having same priority then we can't expect execution order, it depends on the thread scheduler
- Thread class defines the following methods to get and set priorities if a thread.
  i. **public final int getPriority();**
  ii. **public final void setPriority(int newPriority);**
- The default priority only for the main method is 5. But for all remaining thread default priority will be inherited from parent to child.

Example program for setPriority() methods:

```
class MyThread extends Thread
    {
      public void run()
      {
           for(int i=0;i<=5;i++)
              {
                   System.out.println("Child thread");
              }
      }
  }

class ThreadPriorityDemo
     {
        public static void main(String args[])
          {
            MyThread t = new  MyThread();
            t.setPriority(10);
            t.start();
                 for(int i=0;i<=5;i++)
                      {
                   System.out.println("Main method");
                      }
          }
      }
```

Output : D:\> javac ThreadPriorityDemo.java

D:\> java ThreadPriorityDemo

Main method

Main method

Main method

Main method

Main method

Child thread

Child thread

Child thread

Child thread

Child thread

- If both main thread and child thread have the same priority, we can't expect execution of the order and output.
- In the above program, main thread has the priority 5 and the child thread has the priority 10. Hence child thread will get the chance first followed by the main thread.
- Some platforms won't provide proper support for the thread priority.

ADVANTAGES OF THREADS

1. We can execute multiple tasks of an application at a time
2. Reduces the complexity of a big applications
3. Helps to improve the performance of an application drastically
4. Utilizes the max resources of multiprocessor systems
5. Better user interface in case of GUI based applications
6. Reduces the development time of an application
7. All the threads are independent , any unexpected exception happens in any of the thread will not lead to an application exit.

MULTITASKING : Executing several tasks simultaneously is called multi-tasking. There are 2 types of multi-tasking,

1. Process based multitasking
2. Thread-based multi-tasking

1.Process based multitasking: Executing several tasks simultaneously. When each task is a separate independent program or process is called Process based multitasking.

Example: While typing a java program in editor we can able to listen to audio songs by an mp3 player in our system at the same time we can download a file from the net. All these jobs are executing together and independent of each other hence, this is Process-based multi-tasking. Process-based multitasking is best suited at OS Level.

2.Thread based multitasking: Executing several tasks simultaneously. When each task is a separate independent part of the same program is called Thread based multitasking. It is best suitable for the programmatic level.

MULTITHREADING IN JAVA: Multithreading in Java is a process of executing multiple threads simultaneously.

A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Java Multithreading is mostly used in games, animation, etc.

Advantages of Java Multithreading

1) It **doesn't block the user** because threads are independent and you can perform multiple operations at the same time.

2) You **can perform many operations together, so it saves time**.

3) Threads are **independent**, so it doesn't affect other threads if an exception occurs in a single thread.

D.N.R. COLLEGE (AUTONOMOUS): BHIMAVARAM

DEPARTMENT OF COMPUTER SCIENCE

OBJECT ORIENTED PROGRAMMING WITH JAVA - IV

IV SEMESTER

UNIT V
APPLETS

Applet : Applet is a java program that runs under a java compatible browsers such as Netscape, Hot java, Internet explorer and Google chrome. It runs inside the browser and works at client side. Applet program is embedded in the webpage to generate the dynamic content.

An applet allows web documents to be both animated and interactive. It can perform arithmetic operations, display graphics, play sounds, accept user input, create animations and play interactive games.

TYPES OF APPLET: Applets are divided into two types

1. Local applet
2. Remote applet

1. Local applet: An applet developed locally and stored in a local system that is known as a **Local Applet**. Local applet doesn't require the Internet Connection.

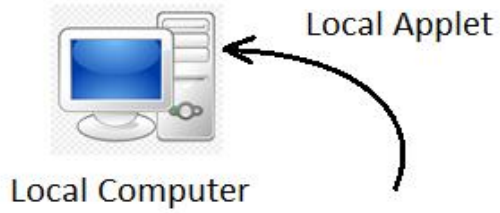


Fig : Local Applet

2.Remote applet: When an **applet** is developed by someone else and stored on a remote computer connected to the Internet. If our system is connected to the Internet, we can download the remote applet onto our system via the Internet. This type of applet is called as Remote applet.

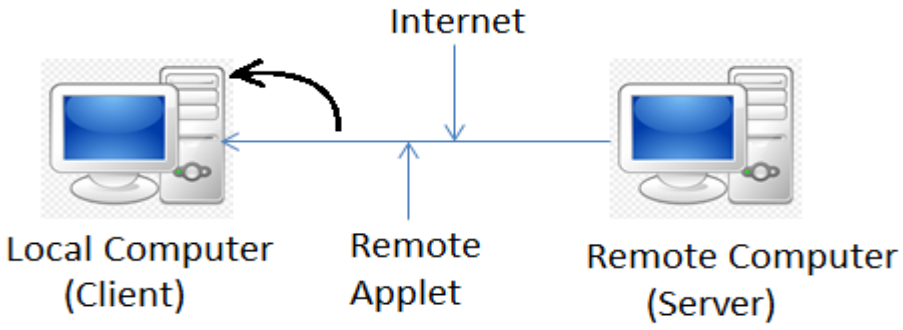


Fig : Remote Applet

STEPS TO CREATE AND EXECUTE APPLET PROGRAM

An applet is a Java program that runs in a Java-compatible browser such as Internet explorer. This feature allows users to display graphics and to run programs over the Internet. An applet allows web documents to be both animated and interactive.

Step 1: Import applet package and awt package: To create an applet, our program must import the Applet class. This class is found in the java.applet package. The Applet class contains code that works with a browser to create a display area. We also need to import the java.awt package. "awt" stands for "Abstract Window Toolkit". The java.awt package includes classes like Graphics.

Step 2: Extend the Applet class: Then, a class must be defined that inherits from the class 'Applet'. It contains the methods to paint the screen. The inherited class must be declared public.

Step 3: Override the paint method to draw text or graphics: The paint method needs the Graphics object as its parameter.

```
public void paint(Graphics g)
    {
        ……
    }
```

The Graphics class object holds information about painting. We can invoke methods like drawLine(), drawCircle() and etc using this object.

Syntax: The program looks something like this.

```
import java.applet.*;
import java.awt.*;
public class MyApplet extends Applet
 {
   public void paint(Graphics g)
   {
       g.drawString("Welcome to Apoorva",50,50);
   }
}
```

Step 4: Compiling the Program: After writing the program, we compile it using the command "javac MyApplet.java". This command will compile our code so that we now have MyApplet.class file.

Step 5: Adding applet to HTML document: To run the applet we need to create the HTML document. The BODY section of the HTML document allows APPLET tag. The HTML file looks something like this:

```
<HTML>
<BODY>
 <APPLET  code="myapplet.class" width=200 height=200> </APPLET>
</BODY>
</HTML>
```

Step 6: Running an applet: The applet can be run in two ways

- By using appletViewer:

  Syntax: appletViewer filename.html

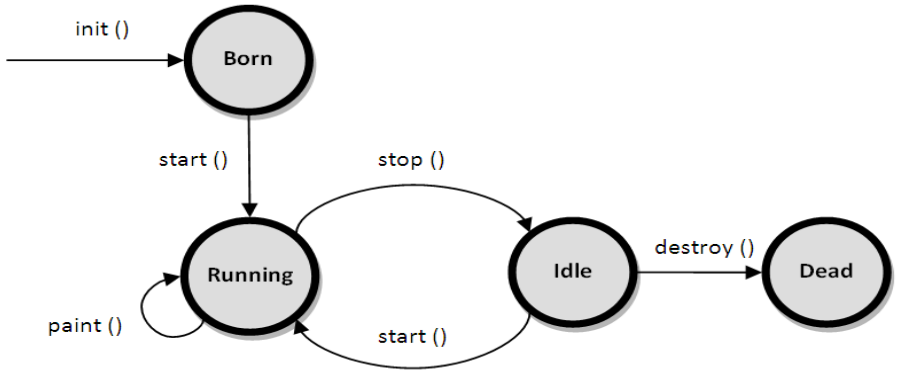
- By using web browser:

  Open the web browser, type in the full address of html file.

LIFE CYCLE OF AN APPLET OR VARIOUS STATES INVOLVED IN AN APPLETS

Every applet can be said to be anyone of the following states,

1. New born state
2. Running state
3. Idle state
4. Dead state



1. New born state:

- init() method begins the applet life cycle.
- The init() method is called only one time in the life cycle of an applet.
- All the initialization such as initialization of variables, methods, objects, images, sounds are loaded in the init() method.
- After initialization of an init() method user can interact with the applet.

Syntax: public void init()

```
{
```

```
    ----------  // statements
    ----------
}
```

2. Running state:

- After initialization, this state will automatically occur by invoking the start () method of an applet class.
- start() method may be called multiple times.
- In the start() method user can interact with the applet.

Syntax: public void start()

```
{
    ----------  // statements
    ----------
}
```

3. Idle state:

- The idle state will make the execution of the applet to be halted temporarily.
- Applet moves to the state when the currently executed applet is minimized (or) when the user switches over to another page.
- At this point the stop() method is invoked from the idle state, the applet can move to the running state.
- The stop () method can be called multiple times in life cycle of applet or should be called at least one time.

Syntax: public void stop()

```
{
    ----------  // statements
    ----------
}
```

4. Dead state:

- When the applet program terminates, the destroy function invoked. Which make an applet to be in dead state.
- The destroy() method is called only one time in the life cycle of an applet like init() method.
- It performs memory cleanup process.

Syntax: public void destroy()

```
{
    ----------  // statements
    ----------
}
```

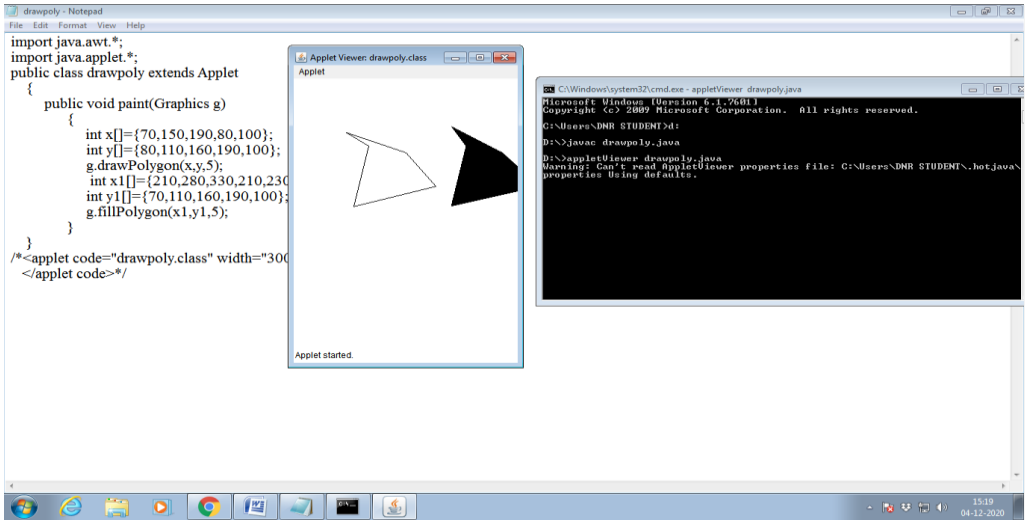
Example program for draw various polygons using applet

```
import java.awt.*;
import java.applet.*;
```

```
public class drawpoly extends Applet
  {
     public void paint(Graphics g)
          {
              int x[]={70,150,190,80,100};
              int y[]={80,110,160,190,100};
              g.drawPolygon(x,y,5);
              int x1[]={210,280,330,210,230};
              int y1[]={70,110,160,190,100};
              g.fillPolygon(x1,y1,5);
          }
  }
/*<applet code="drawpoly.class" width="300" height="400">
  </applet code>*/
```

Output: **D:\>javac drawpoly.java**

D:\>appletViewer drawpoly.java



ADVANTAGES OF APPLET

There are many advantages of applet. They are as follows:

- It works at client side so less response time.
- Secured
- It can be executed by browsers running under many plateforms, including Linux, Windows, Mac Os etc.

- They increase interactivity for users.
- Java has constituted security, by this feature of java; an applet doesn't produce any harmful activity to your computer.
- Database integration is another important advantage of applets.
- They can pay out different type of action on client-side machine like-
    1. Playing sound
    2. Viewing images
    3. Get user input
    4. Get mouse clicks
    5. Get user keystrokes

JAVA DATABASE CONNECTIVITY

JDBC: JDBC stands for Java Database Connectivity; it is an application programming interface (API) for the programming language Java, which defines how a client may access a database. It is a Java-based data access technology used for Java database connectivity.

(OR)

JDBC API is a Java API that can access any kind of tabular data, especially data stored in a Relational Database. It is developed in order to move data from frontend to the backend. JDBC works with Java on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX.

DATABASE SERVERS

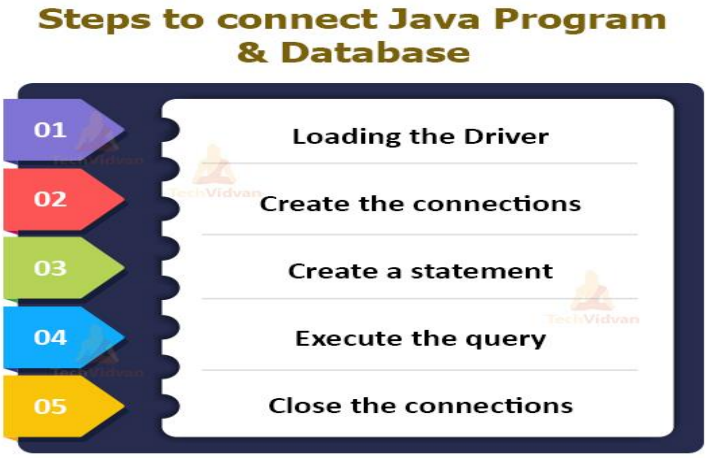
A database is a repository of data. We can store data permanently in a database and retrieve it later whenever needed by using some query commands. Data is stored generally in the form of tables in these databases. To retrieve data from the tables and give it to the users, we need some program. This program is called 'database client' program.

DATABASE CLIENTS

Client programs connect to database servers and retrieve the data from them. The same thing can be done through a Java program which connects to a database and retrieves data from it. This technology is called Java Database Connectivity (JDBC).

JDBC CONNECTIVITY STEPS OR STEPS INVOLVED IN JAVA DATABASE CONNECTIVITY

1. Loading the JDBC Driver
2. Connection to the DBMS
3. Creating and executing statements
4. Processing data returned by the DBMS
5. Terminating the connection with DBMS



1. LOADING THE JDBC DRIVER

The JDBC Driver must be loaded before connecting the DBMS. The driver is loaded by calling the Class.forName() method and passing the name of the driver. The following example shows to load the type 4 driver.

Class.forName("oracle.jdbc.driver.OracleDriver");

The Class.forName() method throws a ClassNotFoundException if an error occurs when loading the JDBC driver.

```
try
{
        Class.forName("oracle.jdbc.driver.OracleDriver");
}
catch(ClassNotFoundException e)
{
        System.out.println(e.getMessage());
}
```

2. CONNECT TO THE DBMS

Once the driver is loaded, We must connect to the DBMS using the DriverManager.getConnection() method.

The DriverManager.getConnection() method required three arguments.

- The first argument is **URL** of the database.
- Second argument is **user name.**
- Third argument is **password**.

The **URL** consists of three parts. These are

Jdbc: which indicates that JDBC protocol

<subprotocol> which is the JDBC driver name

<subname> which is the name of the database. It represents the DSN (data source name), a name given to the database for the reference in the Java program.

The DriverManager.getConnection() method returns Connection object if access is granted, otherwise the getConnection() method throws SQLException.

Connection con;

```
try
```

```
{
        Class.forName("oracle.jdbc.driver.OracleDriver");
        con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","dnr");
}
catch(ClassNotFoundException e)
{
        System.out.println(e.getMessage());
}
catch(SQLException e)
{
        System.out.println("cannot not connect to the database");
}
```

3. CREATE AND EXECUTE A SQL STATEMENT

Once JDBC Driver is loaded and Connection is established to DBMS, to send a SQL query to the DBMS for processing.

The con.createStatement( ) method is used to create a Statement object . The Statement object is used to execute a query and return a ResultSet object that contains one or more rows.

boolean execute(String SQL) : This method is used to execute the DDL statements. This method returns a boolean value of true if DDL statement is successfully executed otherwise return false.

int executeUpdate(String SQL) : This method is used to execute the INSERT, UPDATE, or DELETE SQL statements. This method returns the numbers of rows affected by the execution of the SQL statement.

ResultSet executeQuery(String SQL) : This method is used to execute the SELECT statements. This method returns a ResultSet object.

ResultSet is an object that contains the results (rows) of executing a SQL statement on a database.

4. PROCESSING DATA RETURNED BY THE DBMS

The ResultSet object is assigned the results received from the DBMS after the query is processed. The ResultSet objet consists of methods used to interact with data that is returned by the DBMS.

5. TERMINATE THE CONNECTION TO THE DBMS

The connection to the DBMS is terminated by using the close() method of the Connection object. Closing the database connection automatically closes the ResultSet, it is better to close the ResultSet before closing the connection.

```
con.close();
```

JDBC DRIVER TYPES

JDBC Drivers Classified into four groups.

Type 1 : JDBC to ODBC Driver

Type-1 driver or JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. Type-1 driver is also called Universal driver because it can be used to connect to any of the databases.

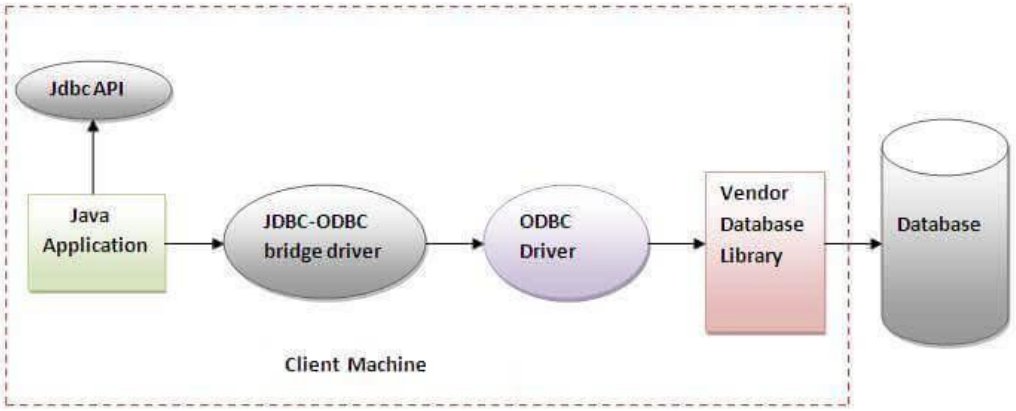


Figure- JDBC-ODBC Bridge Driver

Type 2: Native-API driver

The Native API driver uses the client -side libraries of the database. This driver converts JDBC method calls into native calls of the database API. In order to interact with different database, this driver needs their local API, that's why data transfer is much more secure as compared to type-1 driver.

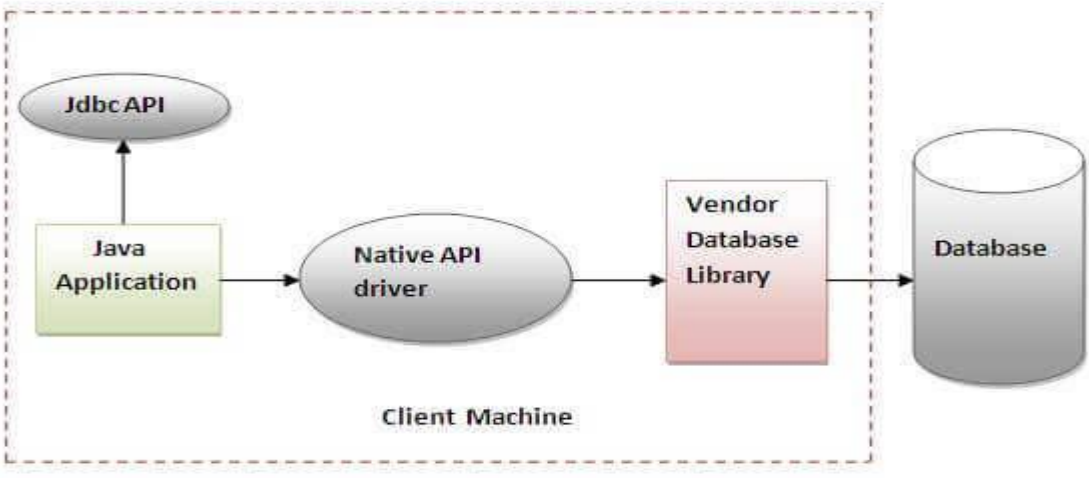


Figure- Native API Driver

Type 3: Network protocol driver

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. Here all the database connectivity drivers are present in a single server, hence no need of individual client-side installation.

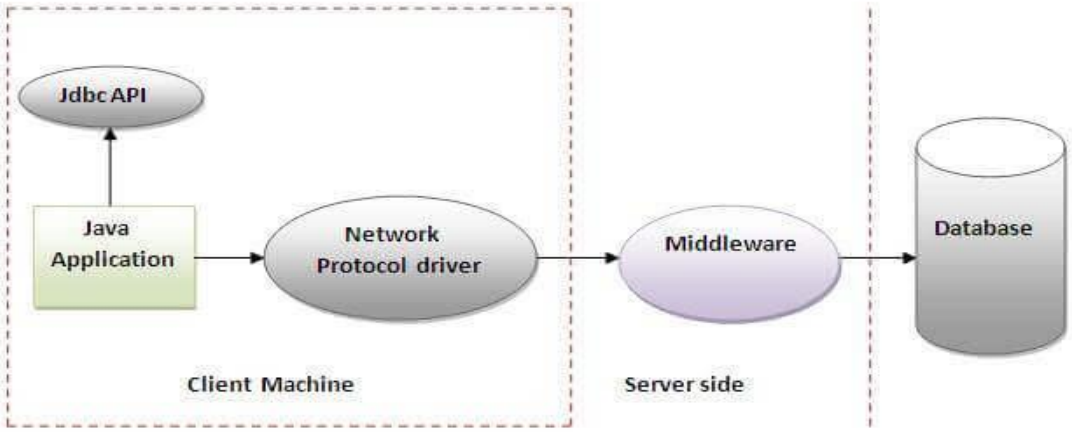


Figure- Network Protocol Driver

Type 4: Native-protocol/Thin driver

Type-4 driver is also called native protocol driver. This driver interact directly with database. It does not require any native database library, that is why it is also known as Thin Driver.

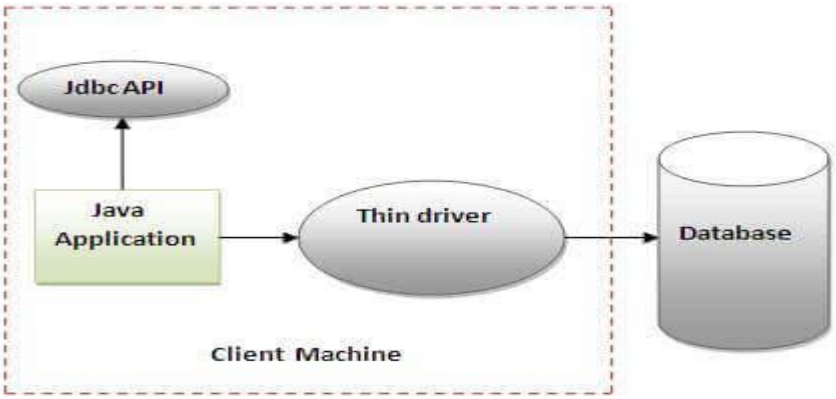


Figure- Thin Driver