

Java

Java is a programming language developed by the member of green project under Sun Microsystem for new days handheld electronic consumer devices. The Java team which includes James Gosling, Mike Sheridan and Patrick Naughton discovered that the existing language like C and C++ had limitation in terms of reliability and portability. They modeled their new language OAK on C and C++ but removed many features of C and C++ that were considered as sources of problem. Finally, Java was developed as a highly reliable, portable, simple and very powerful programming language.

Java Milestone:

Year	Development
1990	Sun Microsystem decided to develop special software that could be used to manipulate consumer electronic devices. A team formed under the guidance of James Gosling.
1991	After finding the possibility of using C++ in new language the team announced a new language named "OAK"
1992	The team demonstrated the application of their new language in a tiny touch-sensitive screen.
1993	The WWW can transform the text based data into graphical rich environment using applet which can run on any type of computer connected to the internet.
1994	Team developed a browser named "Hot Java" to run applet program on the internet.
1995	OAK renamed with Java with some feature.
1996	Sun release Java Development Kit 1.0
1997	Sun release Java Development Kit 1.1 with some new function.
1998	Sun release Java2 with some extended features.
1999	Java release two platform: J2SE and J2EE
2000	J2SE with SDK 1.3(modified to support optional compatibility with CORBA)
2002	J2SE with SDK 1.4(Internet Protocol version 6 (IPv6) support)
30 Sep, 04	J2SE with JDK 5.0 instead of 1.5(Enhanced for each loop)
11 Dec, 06	Java SE 6 (Sun replaced the name "J2SE" with Java SE, Hot Spot VM 10);
28 Jul, 11	Java 7(JVM support for dynamic languages) HotSpot VM 21
18 Mar, 14	Java 8(Launch JavaFX applications, lambda expressions
14 Oct, 14	Java 8 with 20th update(Java 8.2), Java Control Panel option to disable sponsors; JAR file attribute
2016	Java SE 9(including better support for multi-gigabyte heaps, better native code integration, and a self-tuning JVM.)
Around 2018	Java SE 10(There is speculation of introducing objects without identity (value types), as well as moving towards 64-bit addressable arrays to support large data sets.

According to Sun Microsystems; Java is a simple, object oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded and dynamic language.

Features of Java:

Simple: It was designed to be easy for the professional programmer to learn and use effectively. If you are an experienced C++ programmer, moving to java requires a little effort. That's why java is simple.

Object Oriented: Basic unit of problem solving in java is an object that's why java is object oriented programming language. According to some programmer it is pure object oriented language because even a main function of java can not execute without object oriented concept.

Distributed: Java handles TCP/IP protocol so accessing a resource using URL is not much different from accessing a file. It allowed intra-address space messaging through which objects on two different computers to execute procedure remotely.

Interpreted: Java enables the creation of cross platform program by compiling the code into the intermediate language called java bytecode. This code can be interpreted on any system that has Java Virtual Machine.

Robust: De-allocation of memory is completely automatic in Java because it provides garbage collection for unused objects. In a well written Java program all run time errors can be managed by program.

Secure: Java Program does not support pointer means it does not provide memory manipulation that make java program safe against virus affection and other memory problems.

Architecture Neutral: Since java generates byte code after compilation of source code which can run on any hardware platform and are not specific to any processor. This makes java programmer to attaining their goal “Writ Once; run anywhere, any time, forever”

Platform Independent: Since java code can run on variety of platform which is possible through virtual machine technique which makes java “write once run anywhere any time”.

Multithreaded: This allows program to write programs that do many things simultaneously means you can think specific behavior of your program not multitasking.

Dynamic: It is possible to dynamically link code like a small fragment of byte code may be dynamically updated on running system. Maintaining different version of an application is easy also in Java

JVM(Java Virtual Machine)

It is a program that interprets java bytecode and generates the desired output. Program written in Java is highly portable only because of bytecode and JVM concept.

Bytecode

Bytecode is a highly optimized set of instruction designed to be executed by the Java runtime system called the JVM resulting after compilation of source code.

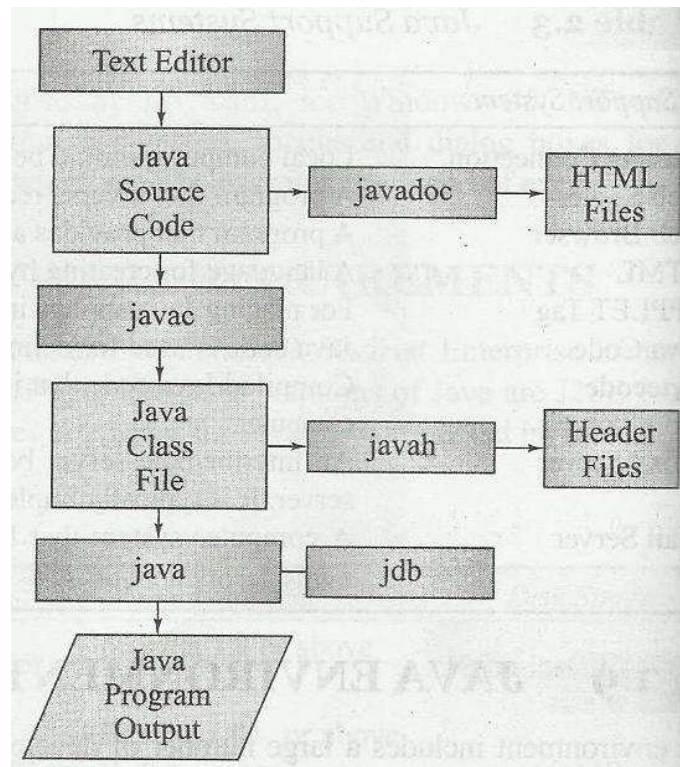
Difference between C and Java

1. C supports keywords sizeof and typedef is not supported by Java.
2. Java doesn't support data type struct and union.
3. Java doesn't have modifier keywords like auto, extern, register, signed, unsigned.
4. Java doesn't support pointer.
5. Java doesn't have preprocessor like #define, # include etc.
6. Function with no argument must be declared with an empty parenthesis not with void keyword.

Difference between C++ and Java

1. Java does not support operator overloading.
2. Java does not have template class as in C++.
3. Java does not support multiple inheritance like C++ but it can be implemented using interface technique.
4. Java haven't global variable.
5. It does not have header file.
6. There is no need to de-allocate memory in java. It done this itself.

Working Mechanism of Java



Types of Java Program

1. Stand Alone Applications: Application programs are stand-alone programs that are written to carry out certain tasks on local computer such as solving equations, reading and writing files etc.
2. Applet: Applets are small Java programs developed for Internet applications. An applet located in distant computer can be downloaded via Internet and executed on a local computer using Java capable browser.
3. Servlets: A servlet is a Java program that is used to extend the capabilities of servers that host applications accessed by means of a request-response programming model.
4. Package: These are collection of classes that can be shared by other java program.

First Java Program

To develop a java program:

-> Open Notepad -> Type code like below.

```

class Test
{
    public static void main(String arg[])
    {
        System.out.print("Welcome to the world of java");
    }
}
  
```

-> Save it with Classname.java in a folder. Suppose e:\foldername\Test.java

-> Open DOS and go to saved program drive and folder by using commands like:

-> To go to particular drive in dos type drivename then colon like "e:" and hit enter.

-> To go to saved folder by typing CD command with folder name. As: CD foldername and press enter.

-> Type <Javac> <filename.java> and press enter to compile source code. Once java code is compiled a file with same name in same folder with .class extension will be created which is nothing but the bytecode.

-> Type <java> <filename> and press enter to run program.

Note in the program:

Class: This is a keyword used to declare a class. In the program Test is a class.

Public: is an access specifier allows the method to be executed from another class.

Static: Since, java is a pure object oriented programming language, encountering a problem of calling main function without object. To solve this, static function in object oriented programming is a function which can be called without creating an object.

Void: is a return type of the function main.

Main(): is a function from where java starts program execution.

String arg[]: is argument type.

System: To communicate with the computer a program needs to use certain resources like display device and keyboard. Java makes use of all these resources with the help of this class System which includes all the methods that are required to work with these resources.

Out: Out is an object encapsulated inside the System class, and represents the output device.

print() or println(): This are methods written within System class used to display output on the screen.

Java Character Set

The smallest unit of a java language characters used to write java tokens. These are defined by Unicode character set taken from the large number of language worldwide.

As: a.....z, A.....Z, 0.....9, _, \$ etc.

Java Tokens

In simple, a Java program is a collection tokens, comments and white spaces. Smallest individual unit in a program are known as tokens. Java language includes five types of tokens. They are:

- ➡ Keywords
- ➡ Identifiers
- ➡ Literals
- ➡ Operators
- ➡ Seprators

Keywords

Keywords are some special words whose meaning is already defined in Java compiler. It can not be used as an identifier (class name, function name, object name, program name) because if we are going to do so means we are trying to change the meaning of that keyword which is not allowed by java compiler.

Abstract	Byte	Class	Do	Extends	For	Import
Long	Private	Short	Switch	Throw	Volatile	Assert
Case	Const	Double	Final	Goto	Instanceof	Native
Protected	Static	Synchronized	Transient	While	Boolean	Catch
Continue	Else	Finally	If	Int	New	Public
Strictfp	This	Try	Break	Char	Default	Enum
Float	Implements	Interface	Package	Return	Super	Throw, void

Identifier

Identifiers are nothing but the name of a class, methods, variable, objects, labels, package and interface.

Identifier naming convention:

1. Its name can include alphabets, numbers, special symbols, as well as some alphabets from French and German language.
2. Its name can be of any length.
3. It must be start from an alphabets or an underscores symbol.
4. White spaces are not allowed.
5. Keywords can not be used as an identifier.
6. Uppercase and lowercase letters are distinct.

Literals

Literals in java are a sequence of character that represents constant value to be stored in a variable. Java has these following literals:

- Integer literals
- Floating point literals
- Character literals
- String literals

- Boolean literals

Operators

Operators are symbols that represent some operation on given data (operand).

Some Basic operators in precedence order are:

- Increment/Decrement Operator
- Arithmetic Operator
- Comparison Operator
- Logical Operator
- Assignment Operator

Increment/Decrement Operator

This operator increase or decrease the value of variable by one. It is of two types:

Pre Increment/ Decrement:

It first increase or decrease its value then assign its final value to others.

As: if X=5; y=++x;

Then final values of X will 6 and y will 6.

Post Increment /Decrement: It first assigns its value to others then increases itself.

As: X=5; Y=X++;

Then the final value of X will 6 and Y will 5.

Arithmetic Operator

Operator	Use	Meaning
+	Op1+Op2	Adds op1 and op2
-	Op1-Op2	Subtract op2 from op1
*	Op1*Op2	Multiply op1 and op2
/	Op1/Op2	Divide op1 by op2
%	Op1%Op2	Computes the remainder of dividing op1 by op2

// A program to add two numbers and print result

```
public class Test
{
    public static void main(String arg[])
    {
        int a=10;
        int b=20;
        int c;
        c=a+b;
        System.out.println("The sum of two numbers is: "+c);
    }
}
```

Comparison Operator

Operator	Use	Meaning
==	Op1==Op2	Equals
!=	Op1!=Op2	Not Equal
<	Op1<Op2	Less Than
>	Op1>Op2	Greater Than
<=	Op1<=Op2	Less Than or Equal To
>=	Op1>=Op2	Greater Than or Equal To

Logical Operator

Operator	Use	Meaning(Returns true if)
&&	Op1&&Op2	If both are true
	Op1 Op2	If one of them is true
!	!Op	If Op is false

&	Op1&Op2	If both are true, always evaluate Op1 and Op2
	Op1 Op2	If one is true, always evaluate Op1 and Op2

Assignment Operator

= (used to assign value to a variable), +=, -=, *=, /=

Bitwise Operator

Operator	Use	Meaning
&		Bitwise AND
		Bitwise OR
^		Bitwise XOR
<<		Left Shift
>>		Right Shift
>>>>		Zero fill right shift
~		Bitwise Complement
<<=		Left Shift Assignment
>>=		Right Shift Assignment
>>>=		Zero Fill Right Shift Assignment
X&=y		AND Assignment
X =y		OR Assignment
X^=y		XOR Assignment

According to operand it takes operator can be divided into three categories:

- Unary
- Binary
- Ternary

Unary: The operator which takes a single operand to operate.

As: -, --, ++

Binary: The operator which takes two operands to operate.

As: +, - etc.

Ternary Operator: The operator which takes three operands to operate.

As: ?:(Conditional Operator)

Separators

Separators are some special symbols used to indicate where group of code are divided and arranged. They basically define the shape and function of our code.

How to Include Comments in Java

Type 1 comment: /*

Multi line comments

*/

Type 2 comment // This line is ignored by the compiler

Type 3 comment /* Documentation comment. This type of comment is used by the tool javadoc for automatic generation of documentation */

Data types

Data Type	Size	Default Value	Range
Byte	8	0	-128 to 127 or 0 to 255
Short	16	0	-32768 to +32767
Int	32	0	-2147483648 to -2147483647
Long	64	0L	-9223372036854775808 to 9223372036854775808
Float	32	0.0f	$3.4e^{-038}$ to $1.7e^{+38}$
Double	64	0.0d	$3.4e^{-038}$ to $1.7e^{+38}$ with double precision

Char	16	Null	Single character.
Boolean	1	False	True or False

Escape Sequence Characters

Escape	Meaning
\n	New Line
\t	Tab
\b	Back Space
\r	Carriage Return
\f	Form Feed
\\	Backslash
\'	Single quote
\"	Double quote
\ddd	Octal
\xdd	Hexadecimal
\udddd	Unicode Character

//A program to demonstrate various arithmetic operators

```

class arithmeticOperator
{
    public static void main(String args[])
    {
        int num=5, num1=12, num2=20, result;
        result=num+num1;
        System.out.println("Sum of num and num is :(num+num1=)" + result);
        result=num%num1;
        System.out.println("Modulous of num and num1 is :(num%num1= )" + result);
        result*=num2;
        System.out.println("Product of result and num2 is :(result*num2= )" + result);
        System.out.println("Value of num before the operation is :"+num);
        num++;
        System.out.println("Value of num after operation of ++ operator is :"+num);

        double num3=25.75;
        double num4=14.25;
        double res;
        res=num3-num4;
        System.out.println("num3-num4 is : "+res);
        res-=2.50;
        System.out.println("res-=2.50 is :"+res);
        System.out.println("Value before -- operator is :"+res);
        res--;
        System.out.println("Value of res after -- operator is :"+res);
    }
}

```

Type Casting

The process of converting one data type to another is called type casting.

Syntax:

Type variable1=(type) variable2;

As: int m=50;

```
byte n= (byte)m;  
long count= (long)n;
```

Note: Casting into smaller type may result in loss of data.

Casting that results in loss of information

```
Byte:  short, char, int, long, float, double  
Short: int, long, float, double  
Char:  int, long, float, double  
Int:   long, float, double  
Long:  float, double  
Float: double
```

// Demonstration of how the automatic casting takes place in java

```
class autoTest  
{  
    public static void main(String args[])  
    {  
        long a=10,result;  
        int b=7;  
        result=a+b;  
        System.out.println("a+b= "+result);  
    }  
}
```

Control Statements in Java

Control statements are the statements whose execution can be controlled by a given condition. It is of three types:

1. Decision Making Statements
 - a. If-----else statement
 - i. Nested if
 - ii. If else ladder
 - b. Switch statements
2. Looping Statements
 - a. While loop
 - b. Do while loop
 - c. For loop
3. Others
 - a. Break
 - b. Continue
 - c. Label

If else statement

‘If statement’ in java is conditional statement that executes their body depending on given conditions. If given condition is true then it executes its true body otherwise else part will be executed.

Syntax:

```
if(condition)  
{  
    true statement/s block;  
}  
else  
{  
    false statement/s block;  
}
```

// A program to checks whether student has passed or not.

```
public class test  
{  
    public static void main(String arg[])  
    {  
        int mark=45;  
        if(mark<35)  
        {  
            System.out.println("The student has failed");  
        }  
    }  
}
```



```

    }
    else
    {
        System.out.println("The student has passed");
    }
}

```

If else ladder

Syntax:

```

if(condition1)
{
    true statement/s block1;
}
else if(condition2)
{
    true statement/s block2;
}
.....
.....
else{
    default statement/s block;
}

```

//Demonstrate if-else-if statement

```

class IfElse
{
    public static void main(String args[])
    {
        int month=4;
        String season;
        if(month==12||month==1||month==2)
            season="Winter";
        else if(month==3||month==4||month==5)
            season="Spring";
        else if(month==6||month==7||month==8)
            season="Summer";
        else if(month==9||month==10||month==11)
            season="Autumn";
        else
            season="Bogus Month";
        System.out.println("April is in the "+season+ ".");
    }
}

```

Switch Statements

Switch statement in java is also a conditional statement which executes their body on the given condition. It is also an alternative of if else ladder statement. **Switch** differs from **if** in that switch can only test for equality whereas if can evaluate any type of Boolean expression.

Syntax:

```

switch(expression)    // here expression must be of type byte, short, int or char
{
    case value 1: { statement body of case 1; break;}
    case value 2: { statement body of case 2; break;}
    .....
    .....
    case value n: { statement body of case n; break;}
    default: { statement of default body}
}

```

// Program to demonstrate switch statement

```
public class switchuse
{
    public static void main(String args[])
    {
        int weekday=3;
        switch(weekday)
        {
            case 1: System.out.println("Sunday"); break;
            case 2: System.out.println("Monday"); break;
            case 3: System.out.println("Tuesday"); break;
            case 4: System.out.println("Wednesday");break;
            case 5: System.out.println("Thursday"); break;
            case 6: System.out.println("Friday");  break;
            case 7: System.out.println("Saturday"); break;
            default: System.out.println("Not a valid day");
        }
    }
}
```

Conditional Operator

The conditional operator is otherwise known as the ternary operator and is considered to an alternative to the if else construct.

Syntax:

<test>?<true part>:<false part>

// Program to demonstrate ternary operator

class Ternary

```
{    public static void main(String arg[])
    {
        int i=10;
        int j=78;
        int z=0;
        z=i<j?i:j;
        System.out.println("The value assigned to z is: "+z);    //Result will be: 10
    }
}
```

Iteration Control Statement

Iteration Control Statements are the statements which executes their body repeatedly till given condition is true. It has three main parts to control the execution of program namely; initialization counter, termination condition and loop increment/decrement counter. It is of three types: while loop, do while loop and for loop.

While Loop

The while loop is a looping statements which executes their body repeatedly till given condition is true. It is also called as **entry/pre control** looping statement because it first checks the given condition then executes their body.

Syntax:

```
Initialization counter;
while(condition)
{
    Body of the loop;
    Increment/decrement counter;
}
```

// A program to find factorial of a given number

```
class Fact
{
    public static void main(String arg[])
    {
        int n=5;
        int fact=1;
        int i=1;
        while(i<=n)
        {
            fact*=i;
            i++;
        }
        System.out.println("Factorial of "+n+"is"+fact);
    }
}
```

Do...While Loop

The do...while loop is a looping statement which executes their body repeatedly till given condition is true. It is also called as **post control** looping statement because it first executes their body then condition is checked. It is different from while in the sense that it executes their body at least once even if given condition is false.

Syntax:

```
Initialization counter;
do
{
    Body of the loop;
    Increment/decrement counter;
```

```

        }while(condition)
// A program to find factorial of a given number
class Fact
{
    public static void main(String arg[])
    {
        int n=5;
        int fact=1;
        int i=1;
        do{
            fact*=i;
            i++;
        } while(i<=n)
        System.out.println("Factorial of "+n+"is"+fact);
    }
}

```

For Loop

The **for loop** is a looping statement which executes their body repeatedly till given condition is true. It is also called as **concise looping** statement because its initialization counter, termination condition and increment/decrement counter all are in a single line.

Syntax:

```

for(init counter; termination counter; incre/decrece counter)
{
    body of the for loop;
}

```

As:

```

class Fact
{
    public static void main(String arg[])
    {
        int num;
        int fact=1;
        for(num=5;num>1;num--)
        {
            fact*=num;
        }
        System.out.println("Factorial of 5 is :"+fact);
    }
}

```

Break Loop

The break statement is a conditional statement that halts the execution of the current loop and forces control out of the loop.

Syntax:

```

If(condition)
{
    break;
}

```

//A Program to demonstrate the break statement

```

class breakDemo
{
    public static void main(String args[])
    {
        for(int count=1;count<=100;count++)
        {
            if(count==10)
            {
                break;
            }
            System.out.println("The value of num is :"+count);
        }
        System.out.println("The loop is over");
    }
}

```

Continue Statement

The continue statement in java used to skip the execution of a loop body after continue keyword when a certain condition is true.

Syntax:

```
if(condition)
{
    continue;
}
```

// A program to demonstrate continue statement

```
class continueDemo
{
    public static void main(String args[])
    {
        for(int count=1;count<10;count++)
        {
            if(count%2==0)
            {
                continue;
            }
            System.out.println(" ");
            System.out.println(count+" ");
        }
    }
}
```

Nested Loop

//Table from 1 to 10

```
class Table
{
    public static void main(String args[])
    {
        int table;
        for(int i=1;i<=10;i++)
        {
            for(int j=1;j<=10;j++)
            {
                table=i*j;
                System.out.print(table);
                System.out.print("\t");
            }
            System.out.println(" ");
        }
    }
}
```

Home Work

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

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

WAP to find whether number is prime or not.

```
1
2 4
3 6 9
4 8 12 16
5 10 15 20 25
ABCDEFGHIGFEDCBA
ABCDEF FEDCBA
ABCDE EDCBA
ABCD DCBA
ABC CBA
AB BA
A A
```

User Input in Java

First Method(Through Command Line)

```
class CommadArgTest
{
    public static void main(String arg[])
    {
        int count=0;
        count=arg.length;
        System.out.println("Number of argument in program is :"+count);
// Suppose two argument is passed in the program
        System.out.println("First argument is: "+arg[0]);
        System.out.println("Second argument is: "+arg[1]);
        System.out.println("It will concatenate the passed argument"+(arg[0]+arg[1]));
    }
}
```

Note: By default, passed argument in java through command line is an string even if you passed a numeric value. To convert string into integer we use:

```
int <variable name>=Integer.parseInt(arg[arrayindex]);
```

As:

```
class CommadArgTest
{
    public static void main(String arg[])
    {
        int count=0;
        count=arg.length;
        System.out.println("Number of argument in program is :"+count);
// Suppose two argument is passed in the program
        System.out.println("First argument is: "+arg[0]);
        System.out.println("Second argument is: "+arg[1]);
// Conversion of string argument into integer
        int n1=Integer.parseInt(arg[0]);
        int n2=Integer.parseInt(arg[1]);
        System.out.println("It will add the passed argument: "+(n1+n2));
        System.out.println("It will concatenate the passed argument: "+n1+n2);
    }
}
```

Second Method(Using Scanner Class):

```
import java.util.Scanner;
class AddNumber
{
    public static void main(String arg[])
    {
        int x, y, z;
        System.out.println("Enter two integer value");
        Scanner in= new Scanner(System.in);
        x= in.nextInt();
        y=in.nextInt();
        z=x+y;
        System.out.println("Sum of entered value is :"+z)
    }
}
```

Third Method (Through DataInputStream Class object)

```
import java.io.*;
class JavaIOTest
{
    public static void main(String arg[])throws java.io.IOException
    {
        float PA;
        float IR;
        float Interest;
        int time=0;
        DataInputStream in= new DataInputStream(System.in);
        System.out.print("Enter Principal Amount");
        System.out.flush();
        String Pstring=in.readLine();
        PA=Float.valueOf(Pstring);
        System.out.print("Enter Interes Rate");
        System.out.flush();
        String Istring=in.readLine();
        IR=Float.valueOf(Istring);
        System.out.print("Enter Time");
        System.out.flush();
        String Tstring=in.readLine();
        time=Integer.parseInt(Tstring);
        Interest=(float)(PA*IR*time)/100;
        System.out.println("Interest of given date is :"+Interest);
    }
}
```

Fourth Method(Through InputStreamReader Class)

```
import java.io.*;
class JavaIOTest
{
    public static void main(String arg[])throws java.io.IOException
    {
        String str;
        InputStreamReader ir=new InputStreamReader(System.in);
        BufferedReader br=new BufferedReader(ir);
        System.out.println("Enter Something");
        System.out.flush();
        str=br.readLine();
        System.out.println("This is what you have typed : "+str);
    }
}
```

Fifth Method

```
import java.io.*;
class JavaIOTest
{
    public static void main(String arg[])throws java.io.IOException
    {
        int i;
        byte[] c=new byte[10];
        System.out.println("Enter Something");
        for(i=0;i<10;i++)
        {
            c[i]=(byte)System.in.read();
        }
        System.out.println("This is what you have typed : ");
        System.out.write(c);
    }
}
```

Array in Java

An array is nothing but the collection of element of same data type. It allocates contiguous allocation of memory. An array index starts from zero and ends with size-1.

Declaration Syntax:

<datatype>[]<array_var_name>; or,
<datatype><array_var_name>[];

As:

int []num; or,
int num[];

Java automatically can not allocate memory for a declared array. To allocate memory we use new operator like this:

Syntax: <array_var_name>= new <var_data_type>[size];

As: number= new int[10];

// A program to accept 10 numbers from user and print them with their sum value.

```
import java.util.Scanner;
```

```
class arrTest
```

```
{  
    public static void main(String arg[])  
    {  
        int i;  
        int sum=0;  
        int []num;  
        num=new int[10];  
        Scanner in= new Scanner(System.in);  
        for(i=0;i<10;i++)  
        {  
            num[i]= in.nextInt();  
            sum+=num[i];  
        }  
        for(i=0;i<10;i++)  
        {  
            System.out.println(num[i]);  
        }  
        System.out.println("Sum of given number is: "+sum);  
    }  
}
```

Home Work

Ques1: Write a program to accept a matrix and print their row wise sum and column wise sum.

Ques2: Write a program to accept two matrices and print their element wise sum.

As:	2	4	6		1	2	3		3	6	9
	1	5	7		4	2	5		5	7	12
	4	3	7		6	7	9		10	10	16

Strings in Java

String is a sequence of character. But, unlike other programming that implement string as character array, java implements string as object of type String. When you create a string object, you are creating a string object whose value can not be changed after it has been created.

There is a companion class to **String** called **StringBuffer** whose object contains string that can be modified after they are created. Both String and StringBuffer classes are defined in java.lang. Thus, they are available to all programs automatically. Both these classes are declared as final so neither of class may be subclassed.

Syntax of declaring string:

```
String s= new String();           // will create a string object have no character in it.
```

```
char ch[]={ 'a', 'b', 'c' };
```

```
String s= new String(ch);         // will create a string object that initialize string object with abc.
```

```
String s="abc";                   //will create a string object have abc in it(string literals)
```

```
Char ch[]={ 'a', 'b', 'c', 'd', 'e', 'f' };
```

```
String s= new String(ch, 2, 3); // will create a string object have cde in it.
```

```
String s2= new String(s);         // will create a string object have cde in it.
```

```
byte ascii[]={ 65,66,67,68,69,70 };
```

```
String s= new String(ascii);      // this will create a string object s have ABCDEF in it.
```

```
String s2=new String(ascii, 2, 3); // this will create a string object s2 have CDE in it.
```

```
System.out.println(s.length()); // will display the length of a string object
```

String concatenation

If

```
char ch[]={ 'R', 'A', 'M' };
```

```
String s1= new String(ch);
```

```
String s2="is";
```

```
String s3="9";
```

```
int age=9;
```

Concatenation of two strings;

```
String s4= s1+s2+s3+"years old." //s4 will display RAM is 9 years old.
```

```
String s5="Mohan "+s2+ "7 years old."; //s5 will display Mohan is 7 years old.
```

```
String longstr="Ram and Syam were best friends."+ "Once they were going to forest "+ "for hunting"
```

Concatenation of a string and an integer:

```
String resStr=s1+s2+age+"years old." //resStr will display RAM is 9 years old.
```

```
String s="Four: "+2+2; //s will display Four: 22
```

```
String s="Four: "+(2+2); // s will display Four: 4
```

Concatenation of two strings using function;

```
String s1="Ram";
```

```
String s2=s1.concat("Kumar"); //s2 will display Ram Kumar
```

```
String s3="Mohan";
```

```
String s4=s1.concat(s3); // s4 will display Ram Mohan
```

```
System.out.println("abc".concat("cde")); // will display abccde
```

String Operation Related Function

charAt(int index): This function can be used to extract a character from given string index position.

As: `String s="Annex Institute of Technical";`
`char ch;`
`ch=s.charAt(3); // will assign e to ch.`

Or, `ch="Annex".charAt(3); // will assign e to ch.`

getChars(int srcStrtIndx, int srcEndIndx, char trgt[], int trgtStrt): This will extract characters from source start index to source end index and copy extracted text to the target variable at given target index position. Care must be taken to assure that target array is large enough to hold the number of characters in the specified substring.

```
String s="Annex Institute of Technical";
char buff[]=new char[5];
s.getChars(10,14,buff,0);
System.out.println(buff); // will display itute.
```

getBytes(int srcStrtIndx, int srcEndIndx, char trgt[], int trgtStrt): This is alternative of getChars() function that stores the characters in an array of bytes.

toCharArray(): If you want to convert a string object into character array the easiest way is to use toCharArray() function.

```
Syntax: String s="This is a"; s.toCharArray();
System.out.println(s[5]); // will display value of s object but up to 4th index only.
```

equals() and equalsIgnoreCase(): equals() function can be used to compare two string with case-sensitive. It will return true if given strings are equals in order, false otherwise. If you want to compare two strings ignoring case then use equalsIgnoreCase() function.

```
class equalsDemo
{
    public static void main(String args[])
    {
        String s1="Hello";
        String s2="Hello";
        String s3="Good-bye";
        String s4="HELLO";
        System.out.println(s1+ "equals"+s2+ "->" +s1.equals(s2));
        System.out.println(s1+ "equals"+s3+ "->" +s1.equals(s3));
        System.out.println(s1+ "equals"+s4+ "->" +s1.equals(s4));
        System.out.println(s1+ "equalsIgnoreCase"+s4+ "->" +s1.equalsIgnoreCase(s4));
    }
}
```

equals() versus ==

equals() is a method that compares the characters inside the string object whereas == is a operator that compares two object reference to see whether they are refer to same instances. == operator can also be used to compare two numeric variable for their equality.

As:

```
class eqlsVersEqLOptr
{
    public static void main(String args[])
    {
        String s1="Hello";
        String s2=new String(s1);
        int num1=20;
        int num2=20;
        System.out.println(s1+" equals"+s2+"->" +s1.equals(s2)); // will return true;
        System.out.println(s1+" =="+s2+"->" +(s1==s2)); // will return false
        System.out.println(num1+" =="+num2+"->" +(num1==num2)); //will return true.
    }
}
```

compareTo(): This function can be used to compare two strings and returns a numeric value according to comparison result. If string1 comes before string 2 it will result a negative value, if string 1 comes after string 2 it will result a positive value and if both are same in dictionary order then will result zero value.

As:

```
class compareTo
{
    public static void main(String args[])
    {
        String s1="Hari";
        String s2="Kunal";
        System.out.println("Invoking compare to method "+s1.compareTo(s2));
        System.out.println("Invoking compare to method "+s2.compareTo(s1));
    }
}
```

startsWith() and endsWith(): The startsWith() method determines whether a given String begins with a specified string. Conversely, endsWith() determines whether the String is ends with specified string. It will return true or false.

class StartAndEndWith

```
{
    public static void main(String args[])
    {
        String s1="Harishchandra kumar";
        System.out.println("EndsWith Method Invoking ->" +s1.endsWith("mar"));
        System.out.println("StartsWith Method Invoking ->" +s1.startsWith("Har"));
        System.out.println("StartsWith Method Invoking from Particular index ->" +s1.startsWith("ish",3));
    }
}
```

indexOf() and lastIndexOf(): indexOf(): searches for the first occurrence of a character of substring. Whereas lastIndexOf() searches for the last occurrence of a character or substring.

As:

class indexOfDemo

```
{
    public static void main(String args[])
    {
        String s="Now is the time for all good man to come to the aid of their country";
        System.out.println(s);
        System.out.println("indexOf(t)= "+s.indexOf('t'));
        System.out.println("lastIndexOf(t)= "+s.lastIndexOf('t'));
        System.out.println("indexOf(the)= "+s.indexOf("the"));
        System.out.println("lastIndexOf(the)= "+s.lastIndexOf("the"));
        System.out.println("indexOf(t,10)= "+s.indexOf('t',10));
        System.out.println("lastIndexOf(t,60)= "+s.lastIndexOf('t',60));
        System.out.println("indexOf(the,10)= "+s.indexOf("the",10));
        System.out.println("lastIndexOf(the,60)= "+s.lastIndexOf("the",60));
    }
}
```

replace(): The replace function can be used to replace a character or string with another character or substring.

As:

```
String s="Annex Institute of Technical Education";
System.out.println(s.replace('n', "bddf"));
```

substring(): This function can be used to extract a substring.

Syntax: substring(int startindex): //this will extract substring from given start index position.

substring(int startindex, int numChars); // this will extract given second argument number of character from start index position.

```
class StringReplace
{
    public static void main(String args[])
    {
        String org="This is a test. This is too";
        String search="is";
        String sub="was";
        String result=" ";
        int i;

        do
        {
            System.out.println(org);
            i=org.indexOf(search);
            if(i!=-1)
            {
                result=org.substring(0,i);
                result=result+sub;
                result=result+org.substring(i+search.length());
                org=result;
            }
        }
        while(i!=-1);
    }
}
```

StringBuffer

StringBuffer is a peer class of String that provides a growable character sequence. StringBuffer may have characters and substrings inserted in the middle or appended to the end. StringBuffer will automatically grow to make room for such additions and often has more characters preallocated than are actually needed, to allow room for growth. Once you create an object of StringBuffer class it automatically reserves room for 16 characters.

Syntax for creating a StringBuffer object;

StringBuffer sb=new StringBuffer(); //it will create an empty string buffer object which reserves 16 rooms for allocation

StringBuffer sb=new StringBuffer(20); //it will create an empty string buffer whose capacity to store characters is 20 only.

StringBuffer sb=new StringBuffer("Hello"); //it will create a string buffer object in which "Hello" is content plus it will reserves 16 more rooms for character allocation

Length() and capacity(): length function can be used to find the length of characters inside a StringBuffer object whereas capacity function can be used to find the capacity to store characters of a StringBuffer object.

```
class StringBufferDemo
{
    public static void main(String args[])
    {
        StringBuffer sb=new StringBuffer("Hello");
        System.out.println("buffer="+sb);
        System.out.println("length="+sb.length());
        System.out.println("capacity="+ sb.capacity());
    }
}
```

ensureCapacity(): If you want to preallocate room for a certain number of characters after a StringBuffer has been constructed. You can use ensureCapacity() to set the size of the buffer.

Syntax: StringBuffer sb=new StringBuffer(50);

Sb.ensureCapacity(20);

setLength(): This function can be used to set the length of a string buffer. The difference between ensureCapacity and setLength() function is that setLength function consumes 16 more rooms for future extension.

charAt(int index): This will return character of given index position.

setCharAt(): This function can be used to set character or string to the given index position.

```
class setCharAtDemo
{
    public static void main(String args[])
    {
        StringBuffer sb =new StringBuffer("Hello");
        System.out.println("Buffer before :"+sb);
        System.out.println("charAt(1) before :"+sb.charAt(1));
        sb.setCharAt(1,'i');
        sb.setLength(2);
        System.out.println("Buffer after :"+sb);
        System.out.println("charAt(1) after :"+sb.charAt(1));
    }
}
```

append(): This function can be used to append given character or string to the specified string.

// A program to demonstrate the use of append() method

```
class appendDemo
{
    public static void main(String args[])
    {
        String s;
        int a=42;
        StringBuffer sb=new StringBuffer(40);
        s=sb.append("a= ").append(a).append("! ").toString();
        System.out.println(s);
    }
}
```

insert(): This function can be used to insert a string to the given index position.

```
class insertDemo
{
    public static void main(String args[])
    {
        StringBuffer sb=new StringBuffer("I java!");
        sb.insert(2,"like ");
        System.out.println(sb);
    }
}
```

reverse(): This function can be used to reverse the given string.

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

```

        StringBuffer sb=new StringBuffer("abcdef");
        System.out.println(sb);
        sb.reverse();
        System.out.println(sb);
    }
}

```

delete() and deleteCharAt(): Delete function can be used to delete characters from given start position to end position.

```

class deleteDemo
{
    public static void main(String args[])
    {
        StringBuffer sb=new StringBuffer("This is a test");
        sb.delete(4,7);
        System.out.println("After deleted : "+sb);
        sb.deleteCharAt(0);
        System.out.println("After deleteCharAt : "+sb);
    }
}

```

replace(): This function can be used to replace a character or string with another character or string.

```

class replaceFun
{
    public static void main(String args[])
    {
        String s="Hello".replace('l','w');
        System.out.println(s);
    }
}

```

Object Oriented Programming

As in C language basic unit of problem solving was a procedure or function so C was a function oriented programming language like this, Object Oriented Programming is a programming languages in which basic unit of problem solving is an object.

Properties of OOP

Encapsulation: Encapsulation is a property of an OOP in which both data and functions are bundled together in a single unit. This is only the way through which data hiding principle of OOP can be accomplished.

Inheritance: Inheritance is a process of creating subclass from an existing class. This provides programmer a reusability of code means we can add more features to an existing class without modifying original code. As a result, redundancy can be maintained in program code.

Polymorphism: Identifiers having same name can behave differently depending on situation and its coverage. This feature of OOP is called polymorphism. As same function or constructor in java can behave differently depending on given number of argument and type of argument.

Advantages of Object Oriented Programming:

- OOP provides a clear modular structure for programs which makes it good for defining abstract data types where implementation details are hidden.
- OOP makes it easy to maintain and modify existing code as new objects can be created with small differences to existing ones.
- Since the design is modular, part of the system can be updated in case of issues without a need to make large-scale changes.
- The reuse of software also lowers the cost of development. Typically, more effort is put into the object-oriented analysis and design, which lowers the overall cost of development.

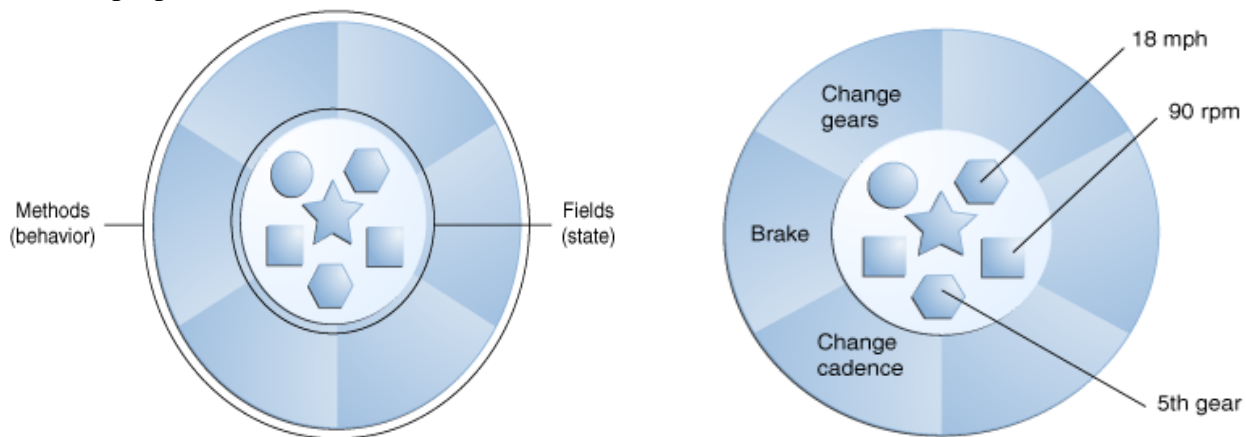
Application of OOP

- 🔗 The most popular applications up to now has been in the area of user interfaces design such as windows.
- 🔗 In real time application: is nothing but a dynamic system a dynamic system is a system that changes every moment band
- 🔗 Object Oriented Database
- 🔗 Hypertext and hypermedia
- 🔗 Artificial Intelligence and Expert System
- 🔗 Office Automation System
- 🔗 CAD and CAM System.

Class

As Java is an Object Oriented Programming language any concept written for a program is inside a class. When an application is written we need a basic building block called object whose all states and behavior is written inside a class.

Class provides a user to create their own data type or a template that can create objects (instance) of that class. It is a description of how to pack together a group of logically related data items (also called field or states or properties) and function (method or behavior or member function) that work on them.



Syntax of class declaration:

```
<class> <class name>
{
    field/data member /properties/state declaration;
    method/memberfunction/function declaration;
}
```

```
As:    class Rectangle
        {
        }
}
```

Field Declaration

Once you have decided to create a class now is the time to think for its field declaration also called instance variable because whenever an object of the class is created it is automatically instantiated.

```
As:    class Rectangle          or          class Rectangle
        {      int len;              {
            int wid;                  int len, wid;
        }                             }
```

A class having data but no member function has no life. So we need to declare member functions to manipulate with this data.

Method or Member Function Declaration

Function declaration Syntax:

```
<return type> <function name> (list of arguments)
{
    Member Function body
}
```

```
As:    void getData(int x, int y)
        {
            len=x;
            wid=y;
        }
```

Bundling both data and function together to accomplish encapsulation (Class Formation)

```
class Rectangle
{
    int len;
    int wid;

    void getData(int x, int y)
    {
        len=x;
        wid=y;
    }
}
```

Note: We can declare as many data member and member function as much we like.

```
class Rectangle
{
    int len;
    int wid;
    void getData(int x, int y)
    {
        len=x;
        wid=y;
    }
    int areaOfRect()
    {
        return (len*wid);
    }
}
```

or,

```
int areaOfRect()
{
    int area=len*wid;
    return (area);
}
```

Object in Java

An object is nothing but a instance of a class. An object has the same relationship with the class as a variable with its data type.

```
class Rectangle
{
    int len;
    int wid;
    void getData(int x, int y)
    {
        len=x;
        wid=y;
    }
    int areaOfRect()
    {
        return (len*wid);
    }
}

class RectTest
{
    public static void main(String arg[])
    {
        Rectangle r1=new Rectangle();
        Rectangle r2=new Rectangle();
        r1.getData(10,15); // Initializing object's properties through function
        r2.len=20; // Initializing object's properties through direct assignment
        r2.wid=10;
        int area=r2.len*r2.wid; // direct calculation in main program and assignment
        System.out.println(r1.areaOfRect()); // calculation through object with function
        System.out.println("Area of Second Rectangle: "+area);
        int area2=r1.areaOfRect(); // Assignment of a function calculation in main program variable
        System.out.println("Area of First Rectangle: "+area2);
    }
}
```


// Examples of multiple class in a program

```
class Rectangle
{
    int len;
    int wid;
    void getData(int x, int y)
    {
        len=x;
        wid=y;
    }
    int areaOfRect()
    {
        return (len*wid);
    }
    void dispPerim()
    {
        System.out.println("Perimeter of Rectangle: "+((2*(len+wid))));
    }
}

class Triangle
{
    int height;
    int base;
    void getData(int x, int y)
    {
        height=x;
        base=y;
    }
    int areaOfTri()
    {
        return (0.5*height*base);
    }
}

class Square
{
    int len;
    void getData(int x)
    {
        len=x;
    }
    void dispAreaSqr()
    {
        System.out.println("Area of square is: "+(len*len));
    }
    void sqrPerm()
    {
        System.out.println("Perimeter of Square is: "+(4*len));
    }
}

class Area
{
    public static void main(String arg[])
    {
        Rectangle rectObj=new Rectangle();
        Triangle triObj=new Triangle();
        Square sqrObj=new Square();
        rectObj.getData(10,15);
        triObj.getData(10,20);
        sqrObj.getData(10);
        rectObj.dispPerim();
        triObj.areaOfTri();
        rectObj.areaOfRect();
        sqrObj.sqrPerm();
    }
}
```

Constructors

Constructor is also a member function of a class which is automatically called whenever an object of a class is created. Its name should be same as class name having no return type in it. It is commonly used to initialize instance variable.

```
class Rectangle
{
    int len;
    int wid;
    Rectangle(int x, int y)
    {
        len=x ;
        wid=y ;
    }
    int areaOfRect()
    {
        return (len*wid);
    }
    void dispPerim()
    {
        System.out.println("Perimeter of Rectangle: "+((2*(len+wid))));
    }
}

class Area
{
    public static void main(String arg[])
    {
        Rectangle rectObj=new Rectangle(5,7);
        rectObj.dispPerim();
    }
}
```

Function and Constructor Overloading.

Constructor or function having same name can behave differently depending on number of argument and type of argument. This feature of OOP is called function or constructor overloading.

```
class Area
{
    int len;
    int wid;
    Area()                //Default Constructor
    {
        len=0;
        wid=0; }
    Area(int x, int y)    //Parameterized Constructor
    {
        len=x;
        wid=y;
    }
    Area(int x)
    {
        len=x;
        wid=x;
    }
    void dispRect()
    {
        System.out.println(len*wid);
    }
}

class ConsOver
{
    public static void main(String arg[])
    {
        Area obj1=new Area(10);
        Area obj2=new Area();
        Area obj3=new Area(10,15);
        obj1.dispRect();
        obj2.dispRect();
        obj3.dispRect();
    }
}
```

// A program to demonstrate function overloading

```
class FuncOver
{
    String name;
    char c;
    int num1;
    float f1;
    int num2;
    void getData(String n, int age)
    {
        name=n;
        num1=age;
        System.out.println("Name="+name);
        System.out.println("Age="+num1);
    }
    void getData(int n1, int n2)
    {
        num1=n1;
        num2=n2;
        System.out.println("Area of rectangle is: "+(num1*num2));
    }
    void getData(String strname, char ch)
    {
        name=strname;
        c=ch;
        System.out.println("Name:="+name);
        System.out.println("Marital Status="+c);
    }
    void getData(int n1, float f1)
    {
        f1=f1;
        num1=n1;
        System.out.println((float)(f1+num1));
    }
    void getData(String nm, char ms, int ag, float sal, int exp)
    {
        name=nm;
        c=ms;
        num1=ag;
        f1=sal;
        num2=exp;
        System.out.println("Name="+name+"Age="+num1);
    }
}

class FuncTest
{
    public static void main(String arg[])
    {
        FuncOver obj1=new FuncOver();
        obj1.getData("Pawan", 'y');
    }
}
```

Inheritance

Inheritance is nothing but the process of creating a subclass from an existing class. In the process of inheritance the class that is inherited is called the super class or main class or base class or parent class and the class that does the inheritance is called derived or sub or child class. A child class can have all the functionality of its base class and can contain more attributes of its own. The **extends** keyword is used to derive a class from super class, or in other words, we are extending the functionality of super class in sub class using extends keyword. Inheritance helps us to:

- Reduce the redundancy in code. Code redundancy means writing the same code in different places, leading to unnecessary replication of code. Inheritance helps us to reuse code.
- Maintain code easily, as the code resides at one place (super class). Any changes made to super class can automatically change the behavior in sub class.
- Extends the functionality of the existing class by adding more methods to the subclass

Syntax:

```
public class <subclass name> extends <super class name>
```

//A program to demonstrate the use of superclass

```
class A
{
    int i,j;
    void showij()
    {
        System.out.println("i and j :"+i+" "+j);
    }
}
class B extends A    //create subclass by extending class A
{
    int k;
    void showk()
    {System.out.println("k : "+k);
    }
    void sum()
    {
        System.out.println("i+j+k :"+(i+j+k));
    }
}
class simpleInheritance
{
    public static void main(String args[])
    {
        A superOb=new A();
        B subOb=new B();
        superOb.i=10;
        superOb.j=20;
        System.out.println("Contents of superOb :");
        superOb.showij();
        System.out.println();
        // The subclass has access to all public members of its superclass
        subOb.i=7;
        subOb.j=8;
        subOb.k=9;
        System.out.println("Content of subOb : ");
        subOb.showij();
        subOb.showk();
        System.out.println();
        System.out.println("Sum of i,j and k in subOb :");
        subOb.sum();
    }
}
```

Use of Super Keyword

Super keyword in java can be used to call data member, member function and constructor of super class in subclass of that class.

Syntax:

super.<datamember of superclass>=value;

super(list of variable);

super.functionSuper();

// Use of super() keyword to name hiding

```
class A
```

```
{    int i; }
```

```
class B extends A
```

```
{    int i;
```

```
    B(int a,int b)
```

```
    {    super.i=a;
```

```
        i=b;    }
```

```
    void show(){
```

```
        System.out.println("i in superclass : "+super.i);
```

```
        System.out.println("i in subclass : "+i);    }
```

```
}
```

```
class UseSuper
```

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

```
    {
```

```
        B subOb =new B(1,2);
```

```
        subOb.show();    }
```

```
}
```

// A program to show use of super keyword

```
class Square {
```

```
    int length;
```

```
    Square(int len)
```

```
    {    length=len; }
```

```
    void getPerimeter()
```

```
    {    System.out.println("Perimeter of square is "+(4*length));
```

```
    }}
```

```
class Rectangle extends Square{
```

```
    int width;
```

```
    Rectangle(int len, int wid)
```

```
    {    super(len);    //superclass constructor call
```

```
        width=wid;
```

```
    }
```

```
    void getPerimeter()
```

```
    {    System.out.println("Perimeter of rectangle is "+(2*(length+width)));
```

```
    }}
```

```
class Calculate{
```

```
    Calculate()
```

```
    { }
```

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

```
    {    Square sqObj=new Square(15);
```

```
        sqObj.getPerimeter();
```

```
        Rectangle recObj=new Rectangle(10,15);
```

```
        recObj.getPerimeter();
```

```
    }}
```

// A complete implementation of BoxWeight and super keyword

```
class Box {
```

```
    private double width;
```

```
    private double height;
```

```

private double depth;
Box(Box ob)
{
    width=ob.width;
    height=ob.height;
    depth=ob.depth;    }
Box(double w, double h, double d)
{
    width=w;
    height=h;
    depth=d;    }
Box() {
    width=-1;
    height=-1;
    depth=-1;    }
Box(double len) {
    width=height=depth=len;    }
double volume() {
    return width*height*depth;
}
}

```

```

class BoxWeight extends Box
{
    double weight;
    BoxWeight(BoxWeight ob) {
        super(ob);
        weight=ob.weight;    }
    BoxWeight(double w, double h, double d, double m) {
        super(w,h,d);
        weight=m;    }
    BoxWeight() {
        super();
        weight=-1;    }
    BoxWeight(double len, double m)    {
        super(len);
        weight=m;    }
}
class DemoSuper
{
    public static void main(String args[])
    {
        BoxWeight mybox1=new BoxWeight(10,20,15,34.3);
        BoxWeight mybox2=new BoxWeight(2,3,4,0.076);
        BoxWeight mybox3=new BoxWeight();
        BoxWeight mycube=new BoxWeight(3,2);
        BoxWeight myclone=new BoxWeight(mybox1);
        double vol;
        vol=mybox1.volume();
        System.out.println("Volumn of mybox1 is : "+vol);
        System.out.println("Weight of mybox1 is : "+mybox1.weight);
        System.out.println();
        vol=mybox2.volume();
        System.out.println("Volume of mybox2 is : "+vol);
        System.out.println("Weight of mybox2 is : "+mybox2.weight);
        System.out.println();
        vol=mybox3.volume();
        System.out.println("Volume of mybox3 is : "+vol);
        System.out.println("Weight of mybox3 is : "+mybox3.weight);
        System.out.println();
        vol=myclone.volume();
    }
}

```

```

        System.out.println("Volume of myclone is : "+vol);
        System.out.println("Weight of myclone is : "+vol);
        System.out.println();
        vol=mycube.volume();
        System.out.println("Volume of mycube is : "+vol);
        System.out.println("Weight of mycube is : "+mycube.weight);
        System.out.println();
    }
}

```

/*Create a class Doctor which contains basic information about doctors.

There are two categories of doctors. Extends the Doctor class to represent 'Specialist' contains the basic information like name, idNumber and address. It also contains specialist specific information like 'speciality'. Similarly 'NonSpecilist' also contains basic information like name, idNumber and address. Declare objects of these classes and display their respective informaton.*/

/*Write a program to create a base class called Employee and two subclass named Manager and Director . The Employee class contains three attributes: name, basic and address and a method called show() , which displays the values of the attributes. The manager class has an additional attribute known as transportAllowance. Create objects of Manaager and Director class and display their details.*/

Use of Final Keyword

Final is a keyword used to declare a final(constant type) instance variable or class or function, object. A variable declared with the final keyword can not change its own value during runtime. A class declared with final keyword can not be inherited. Similarly, a function declared using final keyword can not be overridden.

Syntax:

```

<final><access specifier><datatype><variable name>=<value>; //instance variable declaration
<final><access specifier(not needed)>< class><class name> // final class declaration
<final><access specifier><return type><function name><list of args> //final function

```

As:

// A program to demonstrate the final modifier

```

final class finalDemo
{
    int height;
    finalDemo(int h)    {
        height=h;    }
    public static void main(String args[])
    {
        final finalDemo finalObj=new finalDemo(25);
        //finalObj=new finalDemo(32);    //Error
        //finalObj.height=32;    //Error
        System.out.println(finalObj);
    }
}

```

```

class finalTest extends finalDemo    //Error

```

Use of Abstract Keyword

Abstract is a keyword which can be used with a class or a member function. A class which has been declared as an abstract class can not be directly instantiated. An abstract class contains all common properties and member function of its subclass but body of the function is not defined in the superclass. A abstract function does not have its own body in super class but can be defined in subclass. A class which has an abstract function is abstract by default.

Syntax:

```

<abstract><access specifier> <class><class name>
{
    <abstract> function()
    { It can not have its own body in superclass}
}

```

// A program to demonstrate abstract Modifier

```
abstract class Shape
{
    protected double length;
    protected double width;
    Shape(final double num1, final double num2)
    {
        length=num1;
        width=num2; }
    abstract double area();
}
class Square extends Shape
{
    Square(final double num1)
    {
        super(num1,num1); }
    double area()
    {
        System.out.println("Area of square is : ");
        return length*width; }
}
class Triangle extends Shape
{
    Triangle(final double num1,final double num2)
    {
        super(num1,num2); }
    double area()
    {
        System.out.println("Area of triangle is : ");
        return(0.5*length*width); }
}
class calculateAreaAbstractDemo
{
    protected calculateAreaAbstractDemo()
    {
    }
    public static void main(final String args[])
    {
        Shape finObj;
        Square recObj=new Square(10);
        Triangle triObj=new Triangle(12,8);
        finObj=recObj;
        System.out.println(finObj.area());
        finObj=triObj;
        System.out.println(finObj.area());
    }
}
```

/* Write a program to override an abstract method named add() of the base class Addition. The add() method will add two numbers in the NumberAddition class and concatenate the strings in the TextConcatenation class. Declare and initialize the attributes in the constructor of the base class Addition*/

```
abstract class Addition
{
    int num1;
    int num2;
    String textObj1;
    String textObj2;
    Addition()
    {
    }
    Addition(int numb1,int numb2)
    {
        num1=numb1;
        num2=numb2; }
    Addition(String Obj1, String Obj2)
    {
        textObj1=Obj1;
        textObj2=Obj2;
    }
    abstract void add();
}
```



```

class NumberAddition extends Addition
{
    NumberAddition()
    {
    }
    NumberAddition(int n1, int n2)
    {
        super(n1,n2);
    }
    void add()
    {
        System.out.println("\n=====");
        System.out.println("Adding values : ");
        System.out.println("=====");
        System.out.println("\nTwo numbers are "+num1+" and "+num2);
        System.out.println("After adding number the output is "+(num1+num2));
    }
}

class TextConcatenation extends Addition
{
    TextConcatenation()
    {
    }
    TextConcatenation(String obj1,String obj2)
    {
        super(obj1,obj2);
    }
    void add()
    {
        System.out.println("\n=====");
        System.out.println("Concatenating values :");
        System.out.println("=====");
        System.out.println("\nTwo strings are "+textObj1+" and "+textObj2);
        System.out.println("After concatenating the output is "+textObj1+" "+textObj2);
    }
}

class TestAddition
{
    TestAddition()
    {
    }
    public static void main(String args[])
    {
        Addition objAdd;
        NumberAddition numAdd=new NumberAddition(10,15);
        numAdd.add();
        objAdd=numAdd;
        objAdd.add();
        TextConcatenation textAdd=new TextConcatenation("Harish","Kunal");
        objAdd=textAdd;
        objAdd.add();
        textAdd.add();
    }
}

```

Note: Rules for overriding methods.

- The method name and the order of arguments should be identical to that of the super class method.
- The return type of the methods must be the same.
- The overriding method can not be less accessible than the method it overrides. Means, if method is declared as public in super class then it can not be override with private access specifier in subclass.
- An overriding method can not more raise more exceptions than those raised by the superclass.

Access Specifier

An access specifier determines which feature of class (data member or member function or class itself) may be used by other class. Java has three main access specifiers:

- Public
- Private
- Protected

Public Access Specifier: All class except inner class(a class within class) can have public access specifier. We can declare class, data member and member function as public. The identifier which is declared as public have scope everywhere in the program.

As:

```
<public> <class> <classname>      {  
    public int publicvariable;  
    public void publicmethod();  
    {      }                          }
```

Private Access Specifier: Only object of the same class can access the private variable or method. You can declare only variables, methods, and inner classes as private.

As: private int variablename;

Protected Access Specifier: The variables, methods and inner classes that are declared as protected are accessible to the subclasses of the class in which they are declared.

As: protected int variablename;

Note: If you don't specify any access specifier the scope is friendly. A class, variable, or method that has friendly access is accessible to the all class of a package.

	Private	No modifier	Protected	Public
Same Class	Y	Y	Y	Y
Same Package Sub class		Y	Y	Y
Same package non sub class		Y	Y	Y
Different package subclass			Y	Y
Different Package non subclass				Y

/*Illustration of superclass and derived class. A superclass called Vehicles and a derived class called car are created.*/

```
class Vehicle
```

```
{    protected String name ="Honda Civic";  
    protected String color ="Red";  
    protected int seats=5;  
    Vehicle()  
    {      }  
    void showDetail() {  
        System.out.println("Lines are from the base class Vehicles");  
        System.out.println("Name -"+name);  
        System.out.println("Color -"+color);  
    }      }
```

```
class Car extends Vehicle
```

```
{    Car()  
    {      }  
    void show()  
    {    System.out.println("Lines are from derived class Car");  
        System.out.println("Name of the vehicle -"+name);  
        System.out.println("Color of the Vehicle -"+color);  
        System.out.println("Number of seats -"+seats);  
        System.out.println("=====");    }      }
```

```
class CarTest
```

```

{
    CarTest()
    {
    }
    public static void main(String args[])
    {
        Car vehicle =new Car();
        vehicle.show();
        vehicle.showDetail();
    }
}

```

Use of Static Keyword

We can use static keyword with an instance variable or method. As we know, whenever an object of a class is created then a copy of variable is automatically created for every object of a class. But, **static variable** also called class variable is a type of variable which is shared among all object of that class.

Syntax:

```

static <datatype><variablename>;
static int count;

class test
{
    static int count;
    test() {
        count++;
    }
    void show(){
        System.out.println("Value of count is:"+count);
    }
}

class Demo
{
    public static void main(String arg[])
    {
        test obj=new test();
        test obj2=new test();
        test obj3=new test();
        obj3.show();
    }
}

```

Static Function: As we know a method can not be called without an object. But, static function also called class method is a type of function which can be called without an object of that class.

Declaration Syntax:

```

<static> <returntype><function name>(list of args)
{ body of static function }

```

Calling Syntax:

```

Classname.functionname(list of args)

```

```

As: class test
{
    static int count;
    test(){
        count++;
    }
    static int showcount() {
        return count;
    }
}

class Demo
{ public static void main(String arg[]) {
    System.out.println(test.showcount());
    test obj=new test();
    test obj2=new test();
    test obj3=new test();
    System.out.println(test.showcount());
}
}

```

Interface

The type of inheritance in which a subclass has been derived from multiple parents is called Multiple Inheritance. Multiple inheritance in java can not be directly implemented. To do so, java has a technique called interface. Using the keyword interface you can fully abstract a class from its implementation. Using interface, you can specify what a class must do, but not what does it. Interfaces are similar to class but they lack instance variables, and their methods are declared without any body. Once it is defined, any number of classes can implement an interface.

Also, one class can implement any number of interfaces. By providing the interface keyword, java allows you to fully utilize the “One interface, multiple methods” aspect of polymorphism. In order for a method to be called from one class to another, both classes need to be present at compile time so the java compiler can check to ensure that the method signatures are compatible.

Variable can be declared inside of interface declarations. They are implicitly final and static, meaning they can not be changed by the implementing class. If you do not specify any access specifier in an interface method then by default it is public. They must also be initialized with a constant value.

The general form of interface declaration is:

```
<interface> <interfacename>
{
    variable declaration; //only constant type(final type)
    method declaration;   // abstract method type(method does not have body)
}
```

Variables are declared as follows:

```
<static> <final> <datatype> <variablename>=value;
```

Methods are declared with no body in it:

```
<returntype> methodname(list or args);
```

As:

```
interface Area
{
    static final int len=20;
    final static String name="Harish";
    void show();
}
```

// Program to implement interface

```
interface Area
{
    final static float pi=3.14F;
    float compute(float x, float y);
}
class Rectangle implements Area
{
    public float compute(float x, float y)
    {
        return (x*y);
    }
}
class Circle implements Area
{
    public float compute(float x, float y)
    {
        return (pi*x*x);
    }
}
class InterfaceTest
{
    public static void main(String args[])
    {
        Rectangle rect=new Rectangle();
        Circle cir=new Circle();
        Area area;
        area=rect;
        System.out.println("Area of Rectangle: "+area.compute(10,20));
        area= cir;
        System.out.println("Area of Circle is: "+area.compute(10,15));
    }
}
```

// Implementation of multiple interface type

```
class Student
{
    int rollNum;
    void getNumber(int n)
    {
        rollNum=n;
    }
    void putNumber()
    {
        System.out.println("Roll Number: "+rollNum);
    }
}

class Test extends Student
{
    float sub1, sub2;
    void getMarks(float m1, float m2)
    {
        sub1=m1;
        sub2=m2;
    }
    void putMarks()
    {
        System.out.println("Marks Obtained is :");
        System.out.println("Part 1=" +sub1);
        System.out.println("Part 2= "+sub2);
    }
}

interface Sports
{
    float sportWt=6.0F;
    void putWt();
}

class Results extends Test implements Sports
{
    float total;
    public void putWt()
    {
        System.out.println("Sport Wt="+sportWt);
    }
    void display()
    {
        total=sub1+sub2+sportWt;
        putNumber();
        putMarks();
        putWt();
        System.out.println("Total Score is : "+total);
    }
}

class Hybrid
{
    public static void main(String arg[])
    {
        Results student1=new Results();
        student1.getNumber(1234);
        student1.getMarks(27.5F, 33.0F);
        student1.display();
    }
}
```

Package

Package is nothing but the collection of classes and sub package. As we know a unique name had to be used for each class to avoid name collision. Java provides a mechanism for portioning the class name space into more manageable chunks. This mechanism is the package. The package is both a naming and a visibility control mechanism. You can define classes inside a package that are not accessible by code outside the package. You can also define class members that are only exposed to other members of the same package.

You can create a package using **package** keyword followed by package name. Any class declared within that file will belong to the specified package. If you omit the package statement, the class names are put into default package, which has no name.

Syntax for defining package is:

```
<package><package name>;  
class classname{  
    body of class; }
```

As:

```
package pkg;
```

Any class you declare to be part of **pkg**(package) must be stored in a directory(folder) pkg. More than one file can include the same package statement. You can also create a hierarchy of package. To do so, simply separate each package name from the one above it by use of a period. The general form of a multileveled package is:

```
package pkg1[.pkg2.[pkg3]];
```

A package declared as package java.awt.image; needs to be stored in java/awt/image, java\awt\image, java:awt:image on your UNIX, Windows, or Macintosh files system.

//Writing Package Program

Suppose, first create a folder in E drive like e:\java\pkg

```
package pkg;
```

```
public class pkgTest
```

```
{  
    public void show()  
    {  
        System.out.println("This is displayed using package");  
    }  
}
```

Save it with .java extension in e:\java\pkg\pkgTest.java and only compile it.

Write another program

```
import pkg.*;
```

```
class pkgDemo
```

```
{  
    public static void main(String arg[])  
    {  
        pkgTest obj=new pkgTest();  
        Obj.show();  
    }  
}
```

Exception Handling in Java

Exception can be defined as an abnormal event that occur during program execution and disrupts the normal flow of execution. The abnormal event or unexpected situations that might occur during program execution are:

- Running out of memory
- Resource allocation errors.
- Inability to find a file.
- Problem in network connectivity.

Java handles exceptions in object oriented way. You can use a hierarchy of exception classes to manage runtime error. The class at the top of the exception class hierarchy is called **Throwable**. Two classes are defined from Throwable class- **Error** and **Exception**. Error class can be used for catastrophic failure such as VirtualMachineError. The Exception class is used for the exceptional conditions that have to be trapped in a program. Java has several predefined exceptions as well as user can create their own. Some commonly used built in exceptions are:

- Arithmetic Exception: This exception is thrown when an exceptional arithmetic condition has occurred. Such as division by zero.
- NullPointerException: This exception is thrown when an application attempts to use null where an object is required. An object that has not been allocated memory holds a null value. This may happen due to following reason.
 - Using an object without allocating memory for it.
 - Calling the methods of a null object.
 - Accessing or modifying the attribute of a null object.
- ArrayIndexOutOfBoundsException: This error occurs when an attempt is made to access an array element beyond the index of the array. As if you like to access sixth element of an array which have only five elements in it.

```

class ExceptionCode
{
    public void calculate()
    {
        try
        {
            int num=0;
            int num1=42/num;
        } catch (Exception e)
        {
            System.out.println("Super class Exception catch clause");
        }
    }
}

class Extest
{
    public static void main(String args[])
    {
        ExceptionCode obj=new ExceptionCode();
        obj.calculate();
    }
}

```

To handle exception java uses try, catch, finally and throw block.

try: All code in which there is a chance for a runtime error should be written within try block.

catch: catch block contains code that is executed if and when the exception handler is invoked. A try block can have multiple catch block associate with it.

```

try {

    } catch (IndexOutOfBoundsException e) {
        System.err.println("IndexOutOfBoundsException: " + e.getMessage());
    } catch (IOException e) {
        System.err.println("Caught IOException: " + e.getMessage());
    }
}

```

finally: When an exception is raised, the rest of the statements in the try block are ignored. Sometimes, it is necessary to process certain statement irrespective of whether an exception is raised or not. The finally block is used for this purpose.

// A program to demonstrate try with multiple catch

```

class Erro
{
    public static void main(String arg[])
    {
        int a[]={5, 10};
        int b=5;
        try{

```

```

        int x=a[2]/b-a[1];
    }
    catch(ArithmeticException e)
    {
        System.out.println("Division by zero");
    }
    catch(ArrayIndexOutOfBoundsException e)
    {
        System.out.println("Array Index Error");
    }
    catch(ArrayStoreException e)
    {
        System.out.println("Wrong data type");
    }
    int y=a[1]/a[0];
    System.out.println("Y="+y) ;
}
}
// program to demonstrate finally block
import java.lang.Exception ;
class MyException extends Exception
{
    MyException(String message)
    {
        super(message) ;
    }
}
class TestMyException
{
    public static void main(String arg[])
    {
        int x=5, y=1000;
        try
        {
            float z=(float) x/(float)y;
            if(z<0.01)
            {
                throw new MyException("Number is too small");
            }
        }
        catch(MyException e)
        {
            System.out.println("Caught my Exception");
            System.out.println("e.getMessage()");
        }
        finally
        {
            System.out.println("I am always here");
        }
    }
}
// Demonstration of Nested try
class NestTry
{
    public static void main(String args[])
    {
        try
        {

```



```

    int a=args.length;
    int b=42/a;
    System.out.println("a= "+a);
    try
    {
        if(a==1) a=a/(a-a);
        if(a==2)
        {
            int c[]={1};
            c[42]=99;
        }
    } catch(ArrayIndexOutOfBoundsException e)
    {
    }
    } catch(ArithmeticException e)
    {
        System.out.println("Divide by Zero: "+e);
    }
    }
}

// Another program to demonstrate nest try
class MethNestTry
{
    static void nesttry(int a)
    {
        try
        {
            if(a==1) a=a/(a-a); // if two command are used then generate anOutOfBoundException
            if(a==2)
            {
                int c[]={1};
                c[42]=99; // Generate an OutOfBoundException
            }
        }
        catch(ArrayIndexOutOfBoundsException e)
        {
            System.out.println("ArrayIndexOutOfBound Error : "+e);
        }
    }
    public static void main(String args[])
    {
        try
        {
            int a=args.length;
            int b=42/a;
            System.out.println("a= "+a);
            nesttry(a);
        }
        catch(ArithmeticException e)
        {
            System.out.println("Divided by Zero : "+e);
        }
    }
}

```

Applet

Applet is a java program which can run only on web browser or through appletviewer tool. An applet developed locally and stored in a local system is known as a local applet. A remote applet is that which is developed by someone else and stored on a remote computer connected to the internet.

Applet differ from normal application in the sense that:

- a) Applet do not use main() method for initiating the execution of the code.
- b) Unlike stand-alone application, applets cannot be run independently. They are run from inside a Web page using special feature known as HTML tag.
- c) Applet can not read from or write to the files in the local computer.
- d) Applet cannot communicate with other servers on the network.
- e) Applet cannot run any program from the local computer.
- f) Applet are restricted from using libraries from other language such as C or C++. (Java language supports this feature through native method.

All applets are subclass of the Applet class in the java.applet package. All applets must be declared public. An applet displays information on the screen by using the paint method. This method is available in java.awt.Component class. Paint() method takes an object of Graphics class as a parameter. Graphics class has a method drawString to display text. It requires position to be specified as argument.

The java.applet package is the smallest package in the java API. The Applet class is the only class in the applet package. The Applet class has over 20 methods that are used to display images, play audio files, and respond when you interact with it. The life cycle of the applet is implemented using init(), start(), stop(), and destroy().

Use javac to compile the applet code and appletviewer to execute them. You can view applets in any browser that is Java-enabled.

// A program to demonstrate the use of drawString() method

```
import java.applet.*;
import java.awt.*;
/* <applet code= "Test.class", Height= 200 Width=300></applet>
public class Test extends Applet
{
    public void paint(Graphics g)
    {
        g.drawString("This is displayed by paint method",20,30);
    }
}
```

// A program to draw an oval

```
import java.awt.*;
import java.applet.*;
public class DrawOval extends Applet
{
    public void paint(Graphics g)
    {
        g.drawOval(20,20,20,30);
        //g.drawOval(int x1,int y1, int width, int height);
    }
}
```

// A program to draw a rectangle

```
import java.awt.*;
import java.applet.*;
public class DrawRectangle extends Applet
{
    public void paint(Graphics g)
    {
        g.drawRect(10,10,20,20);
        //g.drawRect(int X1,int X2,int width,int height);
    }
}
```

Some useful function used in applet

```
public abstract void drawString(String text, int x, int y)
public abstract void drawLine(int x1, int y1, int x2, int y2)
public abstract void drawRect(int x1, int y1, int width, int height)
public abstract void fillRect(int x1, int y1, int width, int height)
public abstract void draw3DRect( int x1, int y1, int width, int height, boolean raised)
public abstract void drawRoundRect(int x1, int y1, int width, int height, int arcWidth, int arcHeight)
public abstract void fillRoundRect(int x1, int y1, int width, int height, int arcWidth, int arcHeight)
public abstract void drawOval(int x1, int y1, int width, int height)
public abstract void fillOval(int x1, int y1, int width, int height)
```

// A program to demonstrate applet

```
import java.awt.*;
import java.applet.*;
public class DemoApplet extends Applet
{
    String s;
    public void init()
    {
        s="My Applet";
    }
    public void paint(Graphics g)
    {
        g.drawString(s,70,80);
    }
}
```

//A program to add an image in your program

```
import java.awt.*;
import java.applet.*;
public class DisplayImage extends Applet
{
    Image img;
    public void init()
    {
        img=getImage(getCodeBase(), "duke.gif");
    }
    public void paint(Graphics g)
    {
        g.drawImage(img,20, 20, this);
    }
}
```

Design an applet to display a filled circle with the text "Circle" positioned approximately in the center of it. The fill color of circle should be blue.The font style for the text "Circle" should be bold Times New Roman with font size of 14 and color set to white.

```
import java.awt.*;
import java.applet.*;
public class DrawCircleWithText extends Applet
{
    public void paint(Graphics g)
    {
        g.setColor(Color.blue);
        g.fillOval(200,200,100,100);
        String str="Circle";
        Font objFont=new Font("Times New Roman",Font.BOLD,14);
        g.setColor(Color.white);
        g.drawString(str,240,250);
    }
}
```

// A program to test a counter variable with init(), start(), stop() and destroy() method.

```
import java.awt.*;
import java.applet.*;
/*<applet code=CounterTest.class height="500" width="300"></applet>*/
public class CounterTest extends Applet
{
    int initcounter=0;
    int startcounter=0;
    int stopcounter=0;
    int destroycounter=0;
    public void init()
    {
        initcounter++;
        repaint();
    }
    public void start()
    {
        startcounter++;
        repaint();
    }
    public void stop()
    {
        stopcounter++;
        repaint();
    }
    public void destroy()
    {
        destroycounter++;
        repaint();
    }
    public void paint(Graphics g)
    {
        g.drawString("init has been invoked "+String.valueOf(initcounter) + " times. ",20, 20);
        g.drawString("start has been invoked"+String.valueOf(startcounter)+" times.",20, 35);
        g.drawString("stop has been invoked "+String.valueOf(stopcounter)+" times.",20, 50);
        g.drawString("destroy has been invoked"+String.valueOf(destroycounter)+" times.",20, 65);
    }
}
```

// A program to create an applet with a blue background and display the text "java is cool" in white color

```
import java.awt.*;
import java.applet.*;
public class BackgroundColor extends Applet
{
    public void paint(Graphics g)
    {
        setBackground(Color.blue);
        g.setColor(Color.white);
        g.drawString("Java is cool",100,100);
    }
}
```

// A program to demonstrate different buttons

```
import java.awt.*;
import java.applet.*;
/*<APPLET CODE="ButtonDemo" Height="500" width="300"></applet>*/
public class ButtonDemo extends Applet
{
    Button b1=new Button(); //Button without Label
    //To set a label on above button call setLabel() method
    Button b2=new Button("Button with Label"); //A button with Label
    public void init()
    {
        add(b1);
        add(b2);
    }
}
```

```

    }
}
// This program explains the use of Font class to change the style of text being displayed.
import java.awt.*;
import java.applet.*;
/*<applet code=ChangeFont height="500" width="300"></applet>*/
public class ChangeFont extends Applet
{
    public void paint(Graphics g)
    {
        Font myfont=new Font("Monotype Corosiva",Font.BOLD,18);
        g.setFont(myfont);
        setBackground(Color.blue);
        g.setColor(Color.white);
        g.drawString("Java is cool",100,100);
    }
}

```

// A program to demonstrate the different CheckBoxes

```

import java.applet.*;
import java.awt.*;
public class CheckboxDemo extends Applet
{
    Checkbox chb1=new Checkbox(); //Checkbox without Label
    Checkbox chb2=new Checkbox("Checkbox with Label"); //Checkbox with label

    public void init()
    {
        add(chb1);
        add(chb2);
    }
}

```

// A program to demonstrate Checkboxgroup

```

import java.awt.*;
import java.applet.*;
public class CheckboxGroupDemo extends Applet
{
    CheckboxGroup mygroup=new CheckboxGroup();
    Checkbox c1=new Checkbox("Manipal",mygroup,true);
    Checkbox c2=new Checkbox("Udupi",mygroup,true);
    Checkbox c3=new Checkbox("Aptech",mygroup,true);
    public void init()
    {
        add(c1);
        add(c2);
        add(c3);
    }
}

```

// A program to demonstrate textField and TextArea

```

import java.awt.*;
import java.applet.*;
public class TextFieldAndArea extends Applet
{
    Label l1,l2;
    TextField t1;
    TextArea t2;
    Button b1,b2;
    public void init()
    {
        l1=new Label("Enter Name :",Label.CENTER);
        l2=new Label("Enter Message :",Label.CENTER);
        t1=new TextField(10);
        t2=new TextArea(" ",5,10);
        b1=new Button("Submit");
    }
}

```

```

        b2=new Button("Cancel");
add(l1);
add(t1);
add(l2);
add(t2);
add(b1);
add(b2);
    }
}

```

// A program to test controls

```

import java.awt.*;
import java.applet.*;
public class ControlTest extends Applet
{
    Label l1=new Label("Choose One",Label.CENTER);
    Checkbox c1=new Checkbox("Manipal");
    Checkbox c2=new Checkbox("Udupi");
    Button b1=new Button("Submit");
    Button b2=new Button("Cancel");
    public void init()
    {
        add(l1);
        add(c1);
        add(c2);
        add(b1);
        add(b2);
    }
}

```

// A program to demonstrate the Font and FontMetrics Class

//In this program, the printed line always be in the center of the applet with red background

```

import java.awt.*;
import java.applet.*;
/*<applet code=DisplayBackground.class height="500" width="500"></applet>*/
public class DisplayBackground extends Applet
{
    public void paint(Graphics g)
    {
        String str="This will show text with red background color";
        g.setColor(Color.red);
        setBackground(Color.cyan);
        //setForeground(Color.yellow);
        Font objFont=new Font("Monotype corosiva",Font.BOLD|Font.ITALIC,24);
        FontMetrics fm=getFontMetrics(objFont);
        g.setFont(objFont);
        int numx=(getSize().width-fm.stringWidth(str))/2;
        int numy=getSize().height/2;
        g.drawString(str,numx,numy);
    }
}

```

// A program to display tripleOval

```

import java.awt.*;
import java.applet.*;
/*<applet code=TripleOval.class height="500" width="300"></applet>*/
public class TripleOval extends Applet
{
    public void paint(Graphics g)
    {
        g.drawOval(200,200,50,50);
        g.drawOval(200,75,50,300);
        g.drawOval(75,200,300,50);
    }
}

```

```
}  
//A program to demonstrate drawPolygon() method
```

```
import java.awt.*;  
import java.applet.*;  
public class DrawPolygon extends Applet  
{  
    public void paint(Graphics g)  
    {  
        int numx[]={ 30,200,30,200,30};  
        int numy[]={ 30,30,200,200,30};  
        int num=5;  
        g.drawPolygon(numx,numy,num);  
    }  
}
```

repaint(): This method is used in case an applet is to be repainted. The repaint() calls the update() method, to clear the screen of any existing content. The update method in turn calls the paint() method that then draws the content of the current frame. Repaint() can be done mentioning time in milliseconds to be performed in future. Repaint() method takes four argument to update only a part of the screen. The first two arguments are the X and Y coordinate, next two arguments are the width and height of the image. This help in faster updation of the screen.

```
import java.awt.*;  
import java.applet.* ;  
public class UserIn extends Applet  
{  
    TextField text1, text2 ;  
    public void init()  
    {  
        text1= new TextField(8) ;  
        text2= new TextField(8) ;  
        add(text1) ;  
        add(text2) ;  
        text1.setText('0') ;  
        text2.setText('0') ;  
    }  
    public void paint(Graphics g)  
    {  
        int x=0, y=0, z=0 ;  
        String s1, s2, s3 ;  
        g.drawString("Enter number in each box", 10, 50);  
        try{  
            s1=text1.getText();  
            x=Integer.parseInt(s1);  
            s2=text2.getText();  
            y=Integer.parseInt(s2);  
        }catch(Exception e) { }  
        z=x+y;  
        s3=String.valueOf(z);  
        g.drawString("The Sum is : ", 10, 75 );  
        g.drawString(s3, 100, 75);  
    }  
    public Boolean action(Event event, Object object)  
    {  
        repaint();  
        return true;  
    }  
}
```

Multithreading

Any application we are working on that requires two or more things to be done at the same time. Multithreading is a property of java through which multiple thread of same program can run simultaneously. Through this property java enables us to use multiple flows of control in developing program. Each flow of control may be thought of as a separate tiny program (or module) known as thread that runs in parallel. The ability of a language to support multithreads is referred to as concurrency. Since thread in java are subprograms of a main application program and share the same memory space, they are known as lightweight threads or lightweight process.

Multithreading is useful in a number of ways. It enables programmer to do multiple things simultaneously. They can divide a long program into threads and execute them in parallel. For example, we can send tasks such as printing into the background and continue to perform some other task in the foreground. This approach would considerably improve the speed of our program.

Threads are extensively used in java enabled browsers such as HotJava. These browsers can download a file to the local computer, display a Web page in the window, output another web page to a printer or so on.

Difference between multithreading and multitasking

Multithreading	Multitasking
It is a programming concept in which a program or a process is divided into two or more subprograms or threads that are executed at the same time in parallel.	It is operating system concept in which multiple tasks are performed simultaneously.
It supports execution of multiple parts of a single program simultaneously	It supports execution of multiple programs simultaneously.
The processor has to switch between different parts or thread of a program.	The processor has to switch between different programs of process.
It is highly efficient.	It is less efficient in comparison to multithreading.
A thread is the smallest unit in multithreading.	A program or process is the smallest unit in multitasking environment.
It helps in developing efficient program.	It helps in developing efficient operating system.
It is cost effective in terms of context switching.	It is expensive in case of context switching.

Creating thread in java is simple. Thread are implemented in the form of objects that contain a method called run(). The run() method is the heart and soul of any thread. It makes up the entire body of a thread and is the only method in which the threads behavior can be implemented.

The run() method should be invoked by an object of the concerned thread. This can be achieved by creating the thread and initiating it with the help of another thread method called start().

A new thread can be created in two ways:

1. **By creating a thread class:** Define a class that extends Thread class and override its run() method with the code required by the thread.
2. **By converting a class to a thread:** Define a class that implements Runnable interface. The runnable interface has only one method, run(), that is to be define in the method with the code to be executed by the thread.

// A program to demonstrate multithreading.

```
class A extends Thread
{
    public void run()
    {
        for(int i=1;i<=5;i++)
        {
            System.out.println("\t From threadA : i= "+i);
        }
        System.out.println("Exit from A ");
    }
}

class B extends Thread
{
    public void run()
    {
        for(int j=1;j<=5;j++)
        {
            System.out.println("\t From threadB : j= "+j);
        }
    }
}
```



```

    }
    System.out.println("Exit from B ");
}
}
class C extends Thread
{
    public void run()
    {
        for(int k=1;k<=5;k++)
        {
            System.out.println("\t From threadA : k= "+k);
        }
        System.out.println("Exit from C ");
    }
}
}
class ThreadTest
{
    public static void main(String arg[])
    {
        A objA=new A();
        objA.start();
        B objB=new B();
        objB.start();
        C objC=new C();
        objC.start();
    }
}

```

Lifetime of a thread

During the lifetime of a thread, there are many states it can enter. They are:

- a) Newborn State
- b) Runnable State
- c) Running State
- d) Blocked stated
- e) Dead state

Newborn state: When a thread is created, it is in the new state. Newborn implies that the thread object has been created but it has not started running. It requires the start() method to start it.

Runnable: A thread is in runnable states if it has allocates all its resources to take CPU time.

Running: A thread is to in running state, when it is executing a set of instruction. The run() method contains the set of instruction. This method is called automatically after start() method.

Blocked: The thread could be run but there is something that prevents it while a thread is in the blocked state. The scheduler will simply skip over it and not given it any CPU time. Untill a thread reenters the running state it will not perform any operation.

Dead: The normal way for a thread to die is by returning from it's run() method. We can also call stop function, but this throws an exception that's a subclass of error. Remember that throwing an exception should be special event and not part of normal program execution; thus the use of stop() is discourraged. We can also use a destroy() method that should never be called if we can avoid it, since it is drastic and does not release object lock.

```

class A extends Thread
{
    public void run()
    {
        for(int i=1;i<=5;i++)
        {
            if(i==1) yield();
            System.out.println("\t From threadA : i= "+i);
        }
        System.out.println("Exit from A ");
    }
}
}
class B extends Thread
{
    public void run()
    {
        for(int j=1;j<=5;j++)
        {
            System.out.println("\t From threadB : j= "+j);
        }
    }
}

```

```

        if(j==3) stop();
    }
    System.out.println("Exit from B ");
}
}
class C extends Thread
{
    public void run()
    {
        for(int k=1;k<=5;k++)
        {
            System.out.println("\t From threadA : k= "+k);
            if(k==1)
                try{sleep(1000);}catch(Exception e){ }
        }
        System.out.println("Exit from C ");
    }
}
}
class ThreadTest
{
    public static void main(String arg[])
    {
        A objA=new A();
        B objB=new B();
        C objC=new C();
        System.out.println("Start thread A");
        objA.start();
        System.out.println("Start thread B");
        objC.start();
        System.out.println("Start thread C");
        objB.start();
        System.out.println("End of the main thread");
    }
}

```

to set the priority of thread

```

objThread.setPriority(Thread.MAX/MIN_PRIORITY );
or
objThread.setPriority(intNumber);

```

As:

```

class A extends Thread
{
    public void run()
    {
        for(int i=1;i<=5;i++)
        {
            if(i==1) yield();
            System.out.println("\t From threadA : i= "+i);
        }
        System.out.println("Exit from A ");
    }
}
}
class B extends Thread
{
    public void run()
    {
        for(int j=1;j<=5;j++)
        {
            System.out.println("\t From threadB : j= "+j);
            if(j==3) stop();
        }
        System.out.println("Exit from B ");
    }
}
}
class C extends Thread
{
    public void run()
    {
        for(int k=1;k<=5;k++)
        {
            System.out.println("\t From threadA : k= "+k);
        }
    }
}
}

```

```

        if(k==1)
            try{ sleep(1000);}catch(Exception e){ }
    }
    System.out.println("Exit from C ");
}
}
class ThreadTest
{
    public static void main(String arg[])
    {
        A objA=new A();
        B objB=new B();
        C objC=new C();
        objC.setPriority(Thread.MAX_PRIORITY);
        objB.setPriority(objA.getPriority()+1);
        objA.setPriority(Thread.MIN_PRIORITY);
        System.out.println("Start thread A");
        objA.start();
        System.out.println("Start thread B");
        objB.start();
        System.out.println("Start thread C");
        objC.start();
        System.out.println("End of the main thread");
    }
}

```

Using runnable Interface

```

class Test implements Runnable
{
    public void run()
    {
        for(int i=1; i<=10; i++)
        {
            System.out.println("\t Thread Test "+i);
        } System.out.println("End of threadX");
    }
}
class RunnableTest
{
    public static void main(String arg[])
    {
        Test objRun= new Test();
        Thread objThread=new Thread(objRun);
        objThread.start();
        System.out.println("End of main Thread");
    }
}

```

RMI (Remote Method Invocation)

The **RMI** (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM.

The RMI provides remote communication between the applications using two objects *stub* and *skeleton*.

Understanding stub and skeleton

RMI uses stub and skeleton object for communication with the remote object.

A **remote object** is an object whose method can be invoked from another JVM. Let's understand the stub and skeleton objects:

stub

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

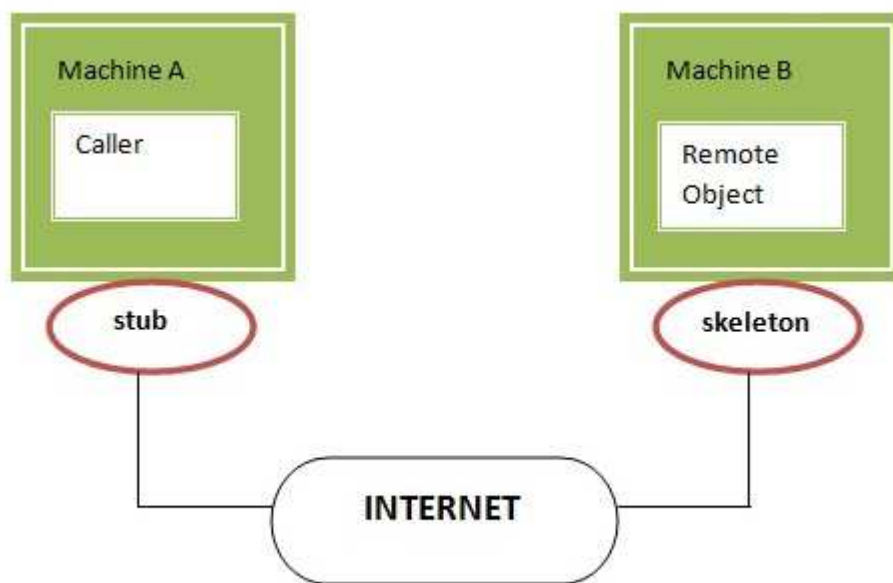
1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
3. It waits for the result
4. It reads (unmarshals) the return value or exception, and
5. It finally, returns the value to the caller.

skeleton

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits (marshals) the result to the caller.

In the Java 2 SDK, an stub protocol was introduced that eliminates the need for skeletons.



Understanding requirements for the distributed applications

If any application performs these tasks, it can be distributed application.

1. The application need to locate the remote method
2. It need to provide the communication with the remote objects, and
3. The application need to load the class definitions for the objects.

Wrapper Class

Each of Java's eight primitive data types has a class dedicated to it. These are known as wrapper classes, because they "wrap" the primitive data type into an object of that class. The wrapper classes are part of the java.lang package, which is imported by default into all Java programs.

The following two statements illustrate the difference between a primitive data type and an object of a wrapper class:

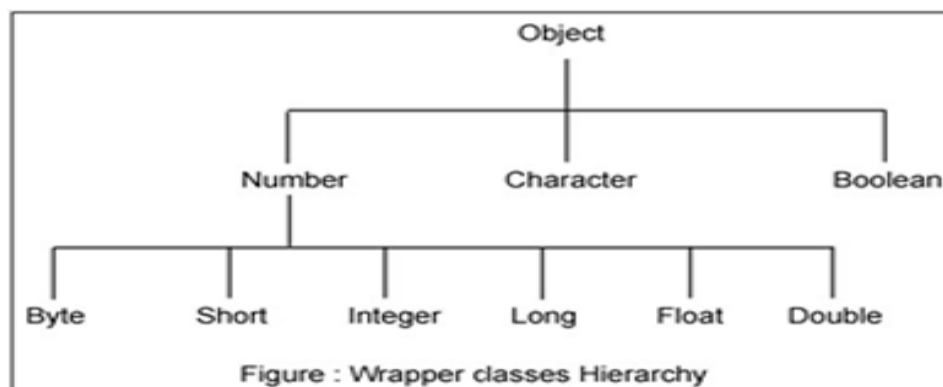
```
int x = 25;
Integer y = new Integer(33);
```

The first statement declares an int variable named x and initializes it with the value 25. The second statement instantiates an Integer object. The object is initialized with the value 33 and a reference to the object is assigned to the object variable y.

Below table lists wrapper classes in Java API with constructor details.

Primitive	Wrapper Class	Constructor Argument
boolean	Boolean	boolean or String
Byte	Byte	byte or String
Char	Character	char
Int	Integer	int or String
Float	Float	float, double or String
Double	Double	double or String
Long	Long	long or String
Short	Short	short or String

Below is wrapper class hierarchy as per Java API



As explain in above table all wrapper classes (except Character) take String as argument constructor. Please note we might get NumberFormatException if we try to assign invalid argument in constructor. For example to create Integer object we can have following syntax.

```
Integer intObj = new Integer (25);

Integer intObj2 = new Integer ("25");
```

Here in we can provide any number as string argument but not the words etc. Below statement will throw run time exception (NumberFormatException)

```
Integer intObj3 = new Integer ("Two");
```

The following discussion focuses on the Integer wrapperclass, but applies in a general sense to all eight wrapper classes.

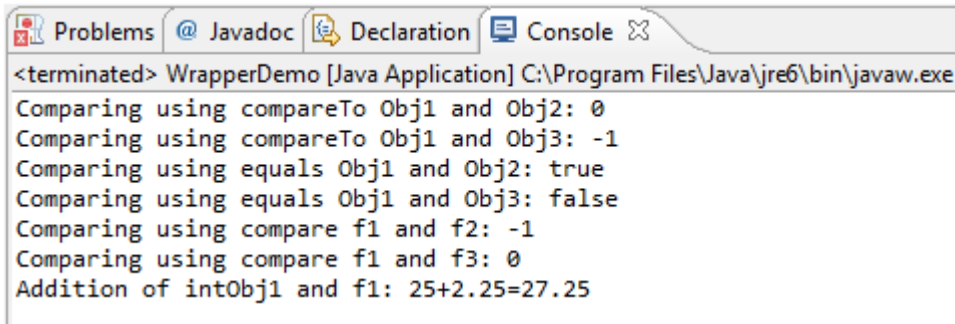
The most common methods of the Integer wrapper class are summarized in below table. Similar methods for the other wrapper classes are found in the Java API documentation.

Method	Purpose
parseInt(s)	returns a signed decimal integer value equivalent to string s
toString(i)	returns a new String object representing the integer i
byteValue()	returns the value of this Integer as a byte
doubleValue()	returns the value of this Integer as an double
floatValue()	returns the value of this Integer as a float
intValue()	returns the value of this Integer as an int
shortValue()	returns the value of this Integer as a short
longValue()	returns the value of this Integer as a long
int compareTo(int i)	Compares the numerical value of the invoking object with that of i. Returns 0 if the values are equal. Returns a negative value if the invoking object has a lower value. Returns a positive value if the invoking object has a greater value.
static int compare(int num1, int num2)	Compares the values of num1 and num2. Returns 0 if the values are equal. Returns a negative value if num1 is less than num2. Returns a positive value if num1 is greater than num2.
Boolean equals(Object intObj)	Returns true if the invoking Integer object is equivalent to intObj. Otherwise, it returns false.

Let's see java program which explain few wrapper classes methods.

```
1. public class WrapperDemo {
2.     public static void main (String args[]){
3.         Integer intObj1 = new Integer (25);
4.         Integer intObj2 = new Integer ("25");
5.         Integer intObj3= new Integer (35);
6.         //compareTo demo
7.         System.out.println("Comparing using compareTo Obj1 and Obj2: " + intObj1.compareTo(intObj2));
8.         System.out.println("Comparing using compareTo Obj1 and Obj3: " + intObj1.compareTo(intObj3));
9.         //Equals demo
10.        System.out.println("Comparing using equals Obj1 and Obj2: " + intObj1.equals(intObj2));
11.        System.out.println("Comparing using equals Obj1 and Obj3: " + intObj1.equals(intObj3));
12.        Float f1 = new Float("2.25f");
13.        Float f2 = new Float("20.43f");
14.        Float f3 = new Float(2.25f);
15.        System.out.println("Comparing using compare f1 and f2: " +Float.compare(f1,f2));
16.        System.out.println("Comparing using compare f1 and f3: " +Float.compare(f1,f3));
17.        //Addition of Integer with Float
18.        Float f = intObj1.floatValue() + f1;
19.        System.out.println("Addition of intObj1 and f1: "+ intObj1 +"+" +f1+"=" +f );
20.    }
21. }
```

Output



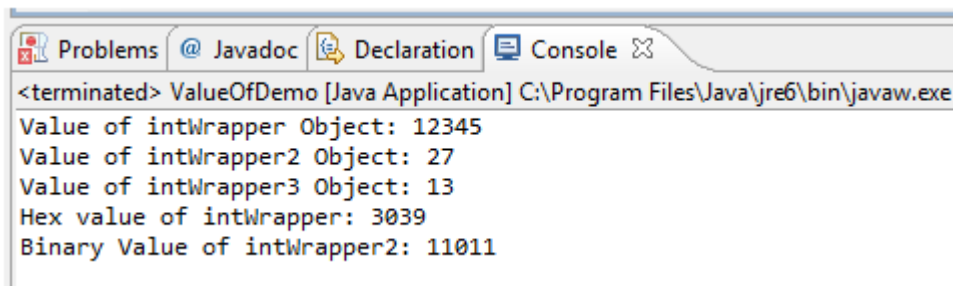
```
<terminated> WrapperDemo [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe
Comparing using compareTo Obj1 and Obj2: 0
Comparing using compareTo Obj1 and Obj3: -1
Comparing using equals Obj1 and Obj2: true
Comparing using equals Obj1 and Obj3: false
Comparing using compare f1 and f2: -1
Comparing using compare f1 and f3: 0
Addition of intObj1 and f1: 25+2.25=27.25
```

valueOf (), toHexString(), toOctalString() and toBinaryString() Methods:

This is another approach to create wrapper objects. We can convert from binary or octal or hexadecimal before assigning value to wrapper object using two argument constructor. Below program explains the method in details.

```
1. public class ValueOfDemo {
2.     public static void main(String[] args) {
3.         Integer intWrapper = Integer.valueOf("12345");
4.         //Converting from binary to decimal
5.         Integer intWrapper2 = Integer.valueOf("11011", 2);
6.         //Converting from hexadecimal to decimal
7.         Integer intWrapper3 = Integer.valueOf("D", 16);
8.         System.out.println("Value of intWrapper Object: "+ intWrapper);
9.         System.out.println("Value of intWrapper2 Object: "+ intWrapper2);
10.        System.out.println("Value of intWrapper3 Object: "+ intWrapper3);
11.        System.out.println("Hex value of intWrapper: " + Integer.toHexString(intWrapper));
12.        System.out.println("Binary Value of intWrapper2: "+ Integer.toBinaryString(intWrapper2));
13.    }
14. }
```

Output



```
<terminated> ValueOfDemo [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe
Value of intWrapper Object: 12345
Value of intWrapper2 Object: 27
Value of intWrapper3 Object: 13
Hex value of intWrapper: 3039
Binary Value of intWrapper2: 11011
```

Vector

The **Vector class** implements a growable array of objects. Similar to array, elements of Vector can be accessed using an integer index. However, the size of a Vector can grow or shrink as needed to accommodate adding and removing items after the Vector has been created.

Vector is synchronized which means it is suitable for thread-safe operations but it gives poor performance when used in multi-thread environment. It is recommended to use ArrayList (it is non-synchronized, gives

good performance) in place of Vector when there is no need of thread-safe operations. Here is the list of tutorials published on Vector class.

Vector implements List Interface. Like ArrayList it also maintains insertion order but it is rarely used in non-thread environment as it is synchronized and due to which it gives poor performance in searching, adding, delete and update of its elements.

Three ways to create vector class object:

Method 1:

```
Vector vec = new Vector();
```

It creates an empty Vector with the default initial capacity of 10. It means the Vector will be re-sized when the 11th elements needs to be inserted into the Vector. Note: By default vector doubles its size. i.e. In this case the Vector size would remain 10 till 10 insertions and once we try to insert the 11th element It would become 20 (double of default capacity 10).

Method 2:

Syntax: `Vector object= new Vector(int initialCapacity)`

```
Vector vec = new Vector(3);
```

It will create a Vector of initial capacity of 3.

Method 3:

Syntax:

```
Vector object= new vector(int initialcapacity, capacityIncrement)
```

```
Vector vec= new Vector(4, 6)
```

Here we have provided two arguments. The initial capacity is 4 and capacityIncrement is 6. It means upon insertion of 5th element the size would be 10 (4+6) and on 11th insertion it would be 16(10+6).

Complete Example of Vector in Java:

```
import java.util.*;

public class VectorExample {

    public static void main(String args[]) {
        /* Vector of initial capacity(size) of 2 */
        Vector<String> vec = new Vector<String>(2);

        /* Adding elements to a vector*/
        vec.addElement("Apple");
        vec.addElement("Orange");
        vec.addElement("Mango");
        vec.addElement("Fig");

        /* check size and capacityIncrement*/
        System.out.println("Size is: "+vec.size());
        System.out.println("Default capacity increment is: "+vec.capacity());

        vec.addElement("fruit1");
        vec.addElement("fruit2");
        vec.addElement("fruit3");

        /*size and capacityIncrement after two insertions*/
        System.out.println("Size after addition: "+vec.size());
        System.out.println("Capacity after increment is: "+vec.capacity());

        /*Display Vector elements*/
        Enumeration en = vec.elements();
        System.out.println("\nElements are:");
        while(en.hasMoreElements())
            System.out.print(en.nextElement() + " ");
    }
}
```



```
}  
}
```

Output:

```
Size is: 4  
Default capacity increment is: 4  
Size after addition: 7  
Capacity after increment is: 8
```

```
Elements are:  
Apple Orange Mango Fig fruit1 fruit2 fruit3
```

Some Important Vector methods are:

1. **void addElement(Object element):** It inserts the element at the end of the Vector.
2. **int capacity():** This method returns the current capacity of the vector.
3. **int size():** It returns the current size of the vector.
4. **void setSize(int size):** It changes the existing size with the specified size.
5. **boolean contains(Object element):** This method checks whether the specified element is present in the Vector. If the element is been found it returns true else false.
6. **boolean containsAll(Collection c):** It returns true if all the elements of collection c are present in the Vector.
7. **Object elementAt(int index):** It returns the element present at the specified location in Vector.
8. **Object firstElement():** It is used for getting the first element of the vector.
9. **Object lastElement():** Returns the last element of the array.
10. **Object get(int index):** Returns the element at the specified index.
11. **boolean isEmpty():** This method returns true if Vector doesn't have any element.
12. **boolean removeElement(Object element):** Removes the specified element from vector.
13. **boolean removeAll(Collection c):** It Removes all those elements from vector which are present in the Collection c.
14. **void setElementAt(Object element, int index):** It updates the element of specified index with the given element.