Table of Content

Sr	Topic	Page No
1	Introduction to Java	1
2	Object Oriented Programming	27
3	Java Control Statements	60
4	Array in Java	61
5	Wrapper Classes	65
6	String and String Buffer	67
7	Inheritance	80
8	Interface	103
9	Package	107
10	Vector Classes	113
11	Exception Handling	120
12	Multithreaded Programming	137
13	Applet	155
14	Collection framework	176
15	Previous Examination questions	197
16	University Questions and answers	211

1. INTRODUCTION TO JAVA

Java is an Object Oriented Programming language derived from C++. Java was first developed by James Gosling, Patrick Naughton, Chris Warth, Ed Frank and Mike Sheridan at Sun Microsystems, Inc. in 1992. The language was initially called „Oak" but was renamed to Java in 1995. In year 2010, Sun microsystem is acquired by Oracle.

Java Development Kit (JDK): Java Software is available downloaded (free) in different versions of JSDK (Java System Development Kit) also called as JDK. JDK are available in single Executable file on net. JDK stands for Java Development Kit, also called as Java System Development Kit (JSDK). It is a software made up of different utilities, using which user can compile, debug (check for errors) and execute Java programs. Java (JDK) consists of Java Compiler, Java Runtime Environment (JRE) and some demo Java programs.

JRE: This mainly consists of JVM and Java API. JVM stands for Java Virtual Machine and also called as Java interpreter. This is used to execute Java programs. Java consists of hundreds of in built Classes / interfaces and thousands of methods in them. This set of classes and interfaces is called as Application Programming Interface (API).

Demo programs: This consists of some Java programs to assist new Java programmer to develop small program.

The classes and interfaces (i.e. Java API) are categorized according their use/purpose and are stored under multiple packages.

For different operating systems different versions of JDK are available. For example, for Java programming in Windows, one should use JDK for Windows, and programming on UNIX system one should use JDK for UNIX.

The commonly used utilities of JDK are:

1. Java Compiler: This is a javac.exe. Used to compile Java application. It compiles the Java files into class files i.e. Bytecode files. Like any other compiler it checks syntax errors and displays them. If there is no error in a program it is compiled to one or more class files.

2. Java Virtual Machine (JVM): This is java.exe. Used to run a Java application. JVM loads a specified Class file (byte-code file) into computer and step by step interprets the instructions and executes them. This is an important part of Java that makes the language Platform independent.

3. Java disassembler: This is javap.exe. Enables to convert bytecode file into program description (actually class description).

4. Java Debugger: This is jdb.exe. Using this programmer can move through a programm step by step and check errors (called as debugging).

5. Appletviewer: This is appletviewer.exe. It is a small Java browser. Appletviewer enables us to run Java applets (without using Java enabled browser like IE).

6. JAR: It is a Java Archive utility. This is commonly used to compress multiple files (like Java source file, class files and other files like image, sound files) into one JAR file i.e. we can create one .jar file using this utility.

Comparison between Procedural Programming Language and Object Oriented programming

	Procedure Oriented Programming	Object Oriented Programming
Divided Into	In POP, program is divided into small parts called **functions**.	In OOP, program is divided into parts called **objects**.
Importance	In POP, Importance is not given to **data** but to functions as well as **sequence** of actions to be done.	In OOP, Importance is given to the data rather than procedures or functions because **it** works as a **real world**.
Approach	POP follows **Top Down approach**.	OOP follows **Bottom Up approach**.
Access Specifiers	POP does not have any access specifier.	OOP has access specifiers named Public, Private, Protected, etc.
Data Moving	In POP, Data can move freely from function to function in the system.	In OOP, objects can move and communicate with each other through member functions.
Expansion	To add new data and function in POP is not so easy.	OOP provides an easy way to add new data and function.
Data Access	In POP, most function uses Global data for sharing that can be accessed freely from function to function in the system.	In OOP, data can not move easily from function to function, It can be kept public or private so we can control the access of data.
Data Hiding	POP does not have any proper way for hiding data so it is **less secure**.	OOP provides Data Hiding so provides **more security**.
Overloading	In POP, Overloading is not possible.	In OOP, overloading is possible in the form of Function Overloading and Operator Overloading.
Examples	Example of POP are: C, VB, FORTRAN, Pascal.	Example of OOP are: C++, JAVA, VB.NET, C#.NET.

Drawbacks of Object Oriented Programming

1. Object oriented programming is **complex to implement** because every entity in it is object. We can access the methods and attributes of particular class using object of that class.
2. If some of the members are declared as **private,** then those **members** are **not accessible** by the object of another class. In such a case you have to make use of inheritance property.
3. In Object oriented programming, everything must be arranged in the forms of **classes and modules.** For the lower level applications, it is not desirable feature.

Applications of Object Oriented Programming: Various applications in which object oriented programming is preferred are-

- Business logic applications.
- Real time systems
- Knowledge based and expert systems
- Web based applications
- Simulation and modeling
- Object oriented databases
- Applications in distributed environment
- CAD/CAM Systems
- Office automation system
- Games programming

Importance of Java to the Internet

- The evolution of Internet helped make Java a leading programming language. Over the network there are two very broad categories of objects that are transmitted between the server and the client's computer.

- These are the passive and active objects in the form of information and active programs. When an email is viewed it is viewing passive data.

- A downloaded program until it is executed is also passive data. A self-executing program is dynamic data. This data is called as a active agent on the client computer.

- These dynamic networked programs create problems in the area of security and portability. Earlier to the evolution of Java, the cyberspace was closed to half the entities that are now present there.

- Java effectively handled these concerns. This was done by introducing a new form of program, an Applet.

a. Java Applets:

- An alphabet is a special program that is transmitted over the Internet. It gets automatically executed by a Java – compatible web browser. An applet is downloaded only on the demand of the user.

- The applet is said to be an "Intelligent Program". This is because the applet tends to react to user input and change its status dynamically.

- Working with applets very exciting. This would not have been possible without a programming language like Java. Java took care of two of the major issues of concern while dealing with applets. These were security and portability of any program code over the Internet.

b. Security:

- Whenever a program is downloaded from the Internet, there is a risk of virus infection to the computer. Before the evolution of Java not much programs were downloaded from the Internet due to the fear of virus.

- Also certain malicious programs, that gather private information like credit card numbers, bank account numbers/balances, could be downloaded. Java provides a solution to this by providing a "firewall" between the downloaded application and the computer.

- Whenever a Java program/applet a downloaded and executed on a Java – compatible web browser, there is no fear of virus infection.

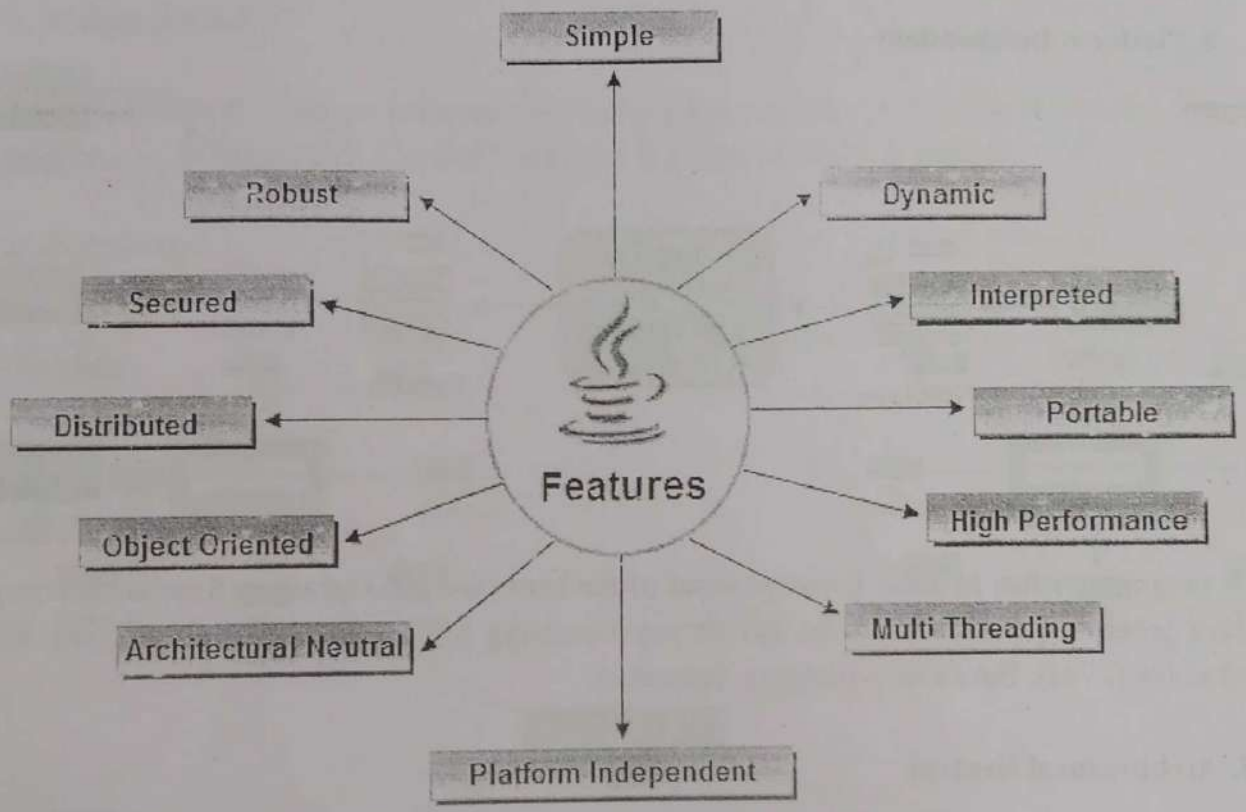
- This is because the Java program is confined or restricted to the Java execution environment. It is not allowed to access other parts of the computer. Thus assuring security.

c. Portability:

- These are 'n' different types of computers with varying configurations of the CPU and operating systems. These are connected to the Internet.

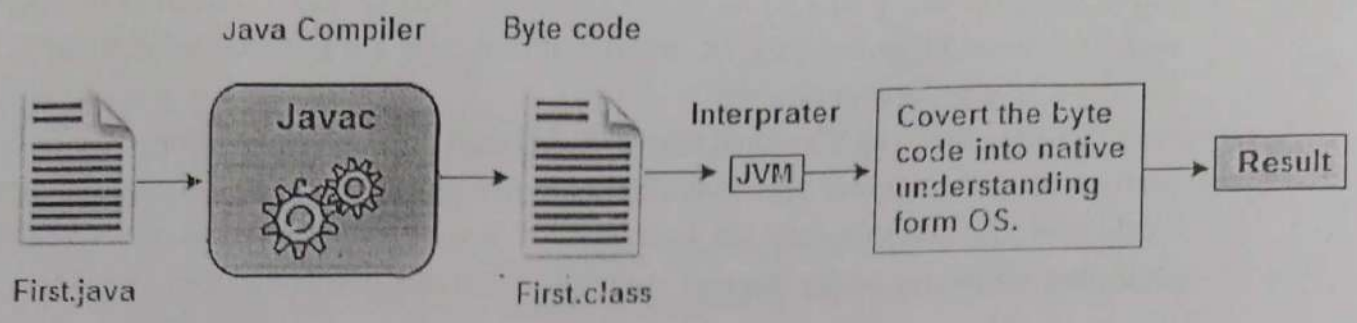
- For a program that needs to be downloaded and then executed, means it is required to make the same program work on varied platforms. i.e. a way needs to be found to make the same Java program portable.
- Java provides the Java Virtual Machine (JVM) that answers this question.
- Many types of computer are connected and operating systems are in use throughout the world and are connected to the internet. For downloading programs through different platforms connected to the internet, we require portable, executable code is needed.

Features of Java

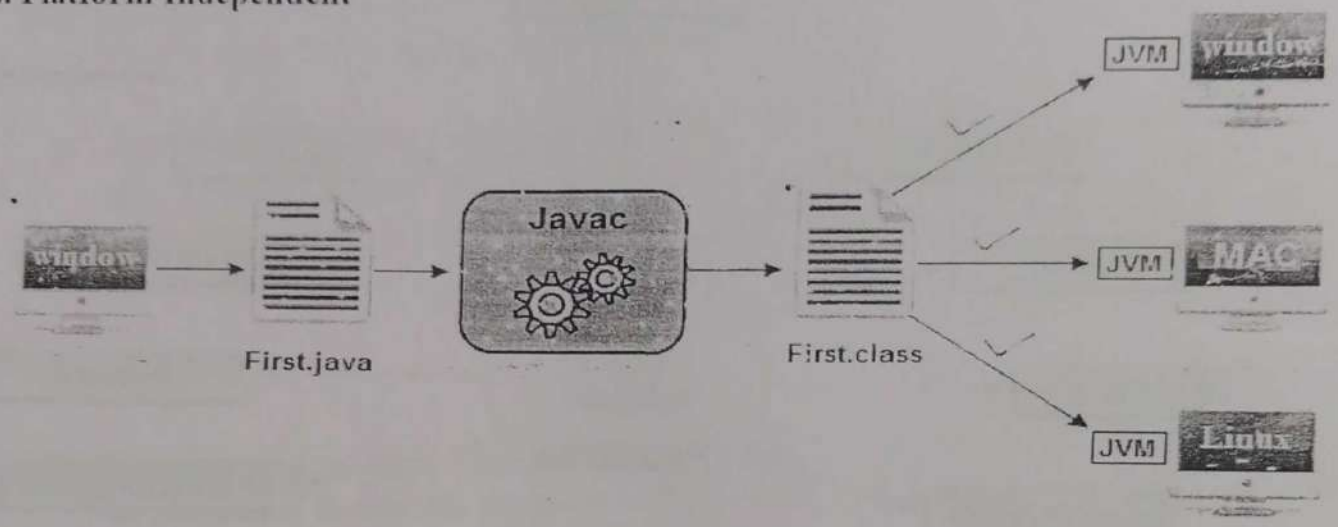


1. Simple Programming Environment & Powerful: It is simple because of the following factors:

a. It is free from pointer due to this execution time of application is improve. [whenever we write a Java program without pointers then internally it is converted into the equivalent pointer program].
b. It have Rich set of API (application protocol interface).
c. It have Garbage Collector which is always used to collect un-Referenced (unused) Memory location for improving performance of a Java program. It contains user friendly syntax for developing any applications.

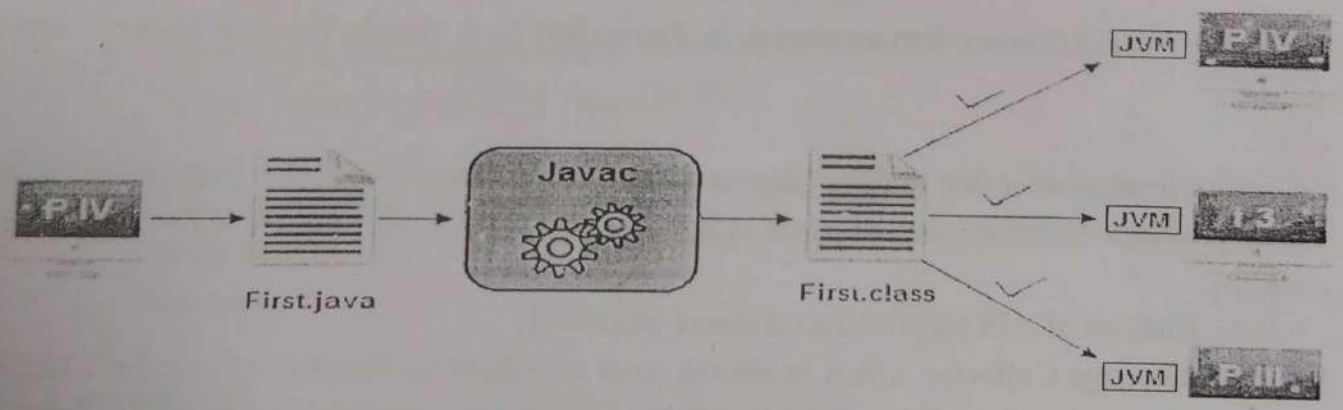


2. Platform Independent



A program written in Java, is independent of the hardware and Operating System. A compiled Java program (bytecode) can be run on any Operating System with the help of Java Virtual Machine (JVM). But JVM is platform dependent.

3. Architectural Neutral



Architecture represents processor. A Language or Technology is said to be Architectural neutral which can run on any available processors in the real world without considering their architecture and vendor (providers) irrespective to its development and compilation. The languages like C, CPP are treated as architectural dependent.

4. Portable

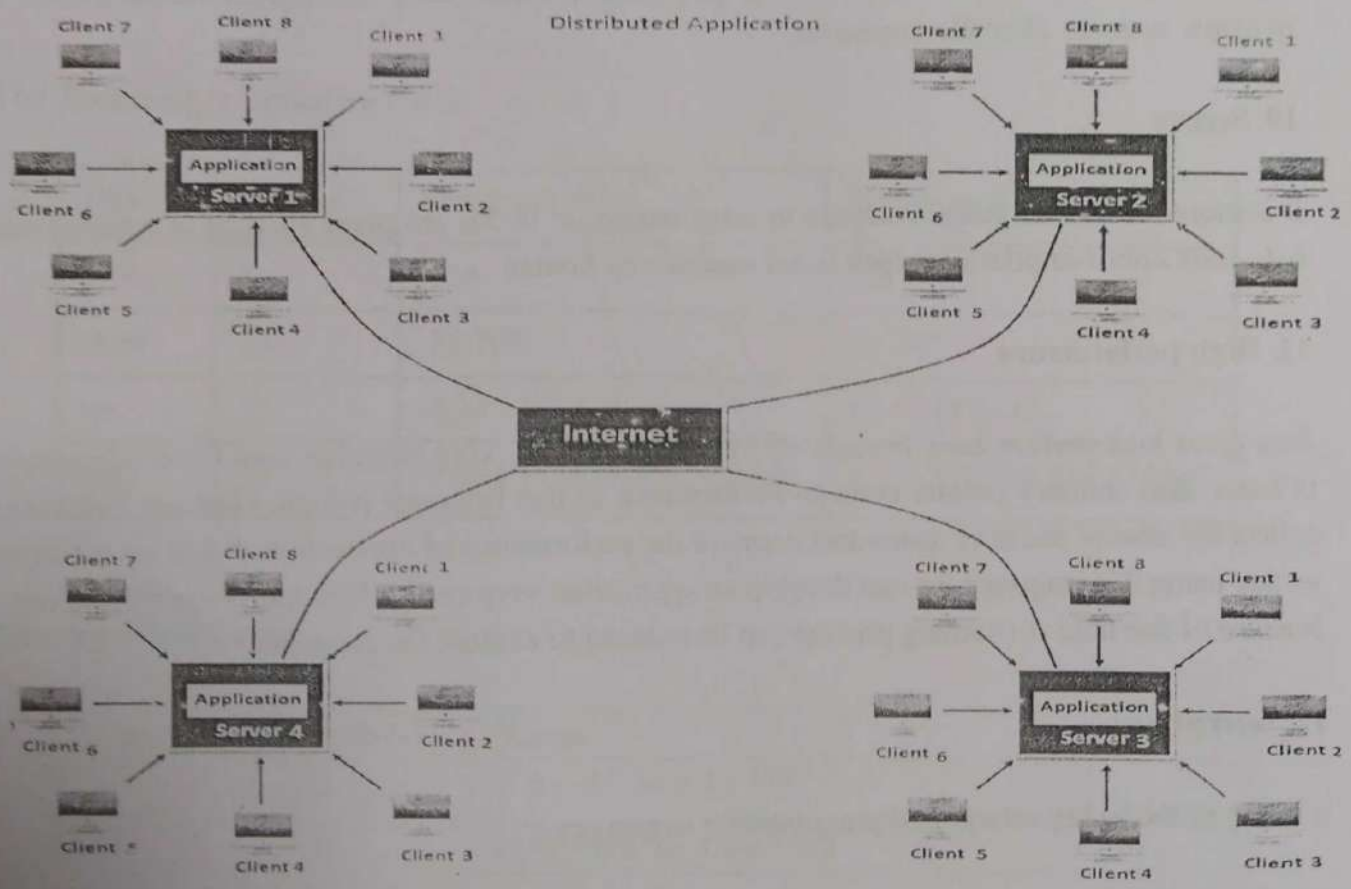
Portability = platform independent + architecture

If any language supports platform independent and architectural neutral feature known as portable. The languages like C, CPP, Pascal are treated as non-portable language. It is a portable language.

5. Multithreaded

A flow of control is known as thread. When any Language execute multiple thread at a time that language is known as multithreaded Language. It is multithreaded Language.

6. Distributed



Using this language, we can create distributed application. RMI and EJB are used for creating distributed applications. In distributed application multiple client system are depends on multiple server systems so that even problem occurred in one server will never be reflected on any client system.

Note: In this architecture same application is distributed in multiple server system.

7. Networked

It is mainly design for web based applications, J2EE is used for developing network based applications.

8. Robust

Simply means of Robust is strong. It is robust or strong Programming Language because of its capability to handle Run-time Error, automatic garbage collection, lack of pointer concept, Exception Handling. All these points make it robust Language.

9. Dynamic

It supports Dynamic memory allocation due to this memory wastage is reduce and improve performance of application. The process of allocating the memory space to the input of the program at a run-time is known as dynamic memory allocation. To programming to allocate memory space by dynamically we use an operator called 'new' 'new' operator is known as dynamic memory allocation operator.

10. Secure

It is more secured language compare to other language; In this language all code is covered into byte code after compilation which is not readable by human.

11. High performance

Java gives high performance because of following reasons: This language uses Bytecode which is faster than ordinary pointer code so Performance of this language is high. Garbage collector. collect the unused memory space and improve the performance of application. It has no pointers so that using this language we can develop an application very easily. It supports multithreading, because of this time consuming process can be reduced to execute the program.

12. Interpreted

It is one of the highly interpreted programming languages.

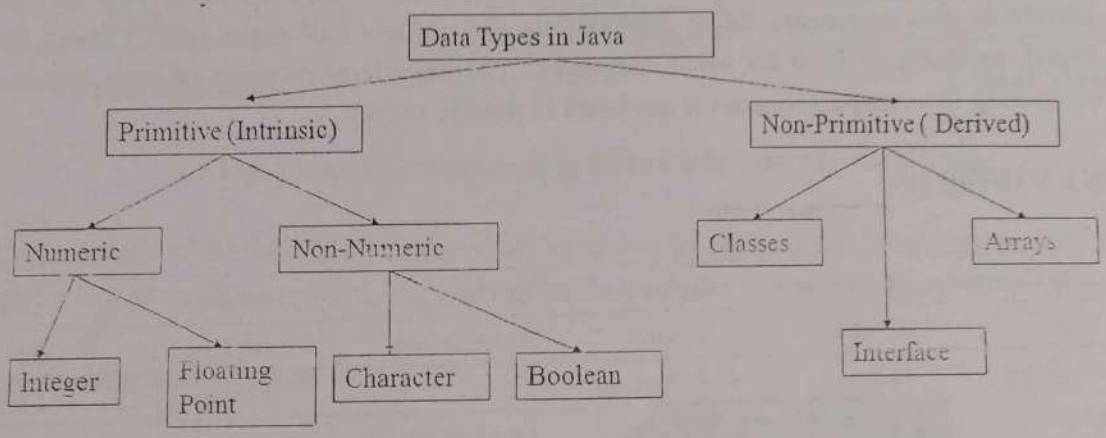
13. Object Oriented

It supports OOP's concepts because of this it is most secure language, for this topic you can read our oop's concepts in detail.

Data Types in Java:

Integer Types

Java consists of four integer types: byte, short, int, and long, which are defined as 8-, 16-, 32-, and 64-bit signed values as summarized in Table below:



The Java integer primitive types.

Type	Bit Size	Minimum Value	Maximum Value
byte	8	-256	255
short	16	-32,768	32,767
int	32	-2,147,483,648	2,147,483,647
long	64	-9,223,372,036,854,775,808	9,223,372,036,854,775,807

Floating-Point Types

Type name	size(bits)	Range
float	32	1.4 e -45 to 3.4 e +38
double	64	4.9 e -324 to 1.8 e +308

By default, Java treats floating-point type as double data-type.

Ex. value 2.5 is treated as double type data. To use the value as float type we attach a letter f or F to the value. i.e. 2.5f / 2.5F is treated as float type value.

Boolean: Java supports two Boolean datatypes -true and false.

Character Literals:

Type name	Size in bits
char	16

This datatype is used in Java to hold characters. Java uses Unicode to represent characters.

String Literals: Although there is no string primitive type in Java, you can include string literals in your programs. Most applications and applets will make use of some form of string literal, probably at least for error messages. A string literal consists of zero or more characters (including the escape sequences) enclosed in double quotes.

KEYWORDS

abstract	boolean	break	byte	case	catch
char	class	const	continue	default	do
double	else	extends	final	finally	float
for	goto	if	implements	import	instanceof
int	interface	long	native	new	package
private	protected	public	return	short	static
strictfp	super	switch	synchronized	this	throw
throws	transient	try	void	volatile	while
assert					

Java Virtual Machine (JVM) & Byte Code

All language compilers translate source code into machine code for a specific computer. Java compiler also does the same thing. Then, how does Java achieve architecture neutrally? The answer is that the Java compiler produces an intermediate code coed known as byte code for a machine that does not exist. This machine is called the Java Virtual Machine and it exists only inside the computer memory. It is a simulated computer within the computer and does all major functions of a real computer. Figure illustrates the process of compiling a Java program into byte code and is referred as virtual machine code.

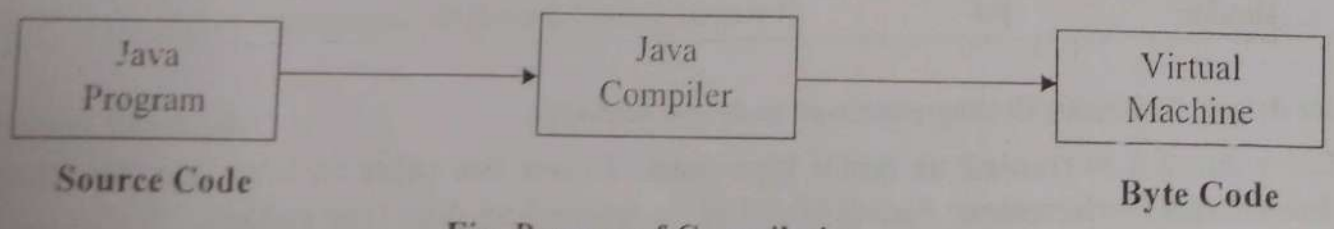


Fig. Process of Compilation

The virtual machine code is not machine specific. The machine specific code (known as machine code) is generated by the Java interpreter by acting as an intermediary between the virtual machine and the real machine as shown in Figure below. Remember that the interpreter is different for different machine.

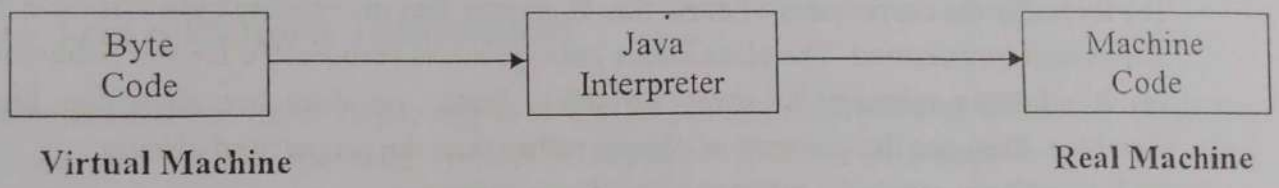
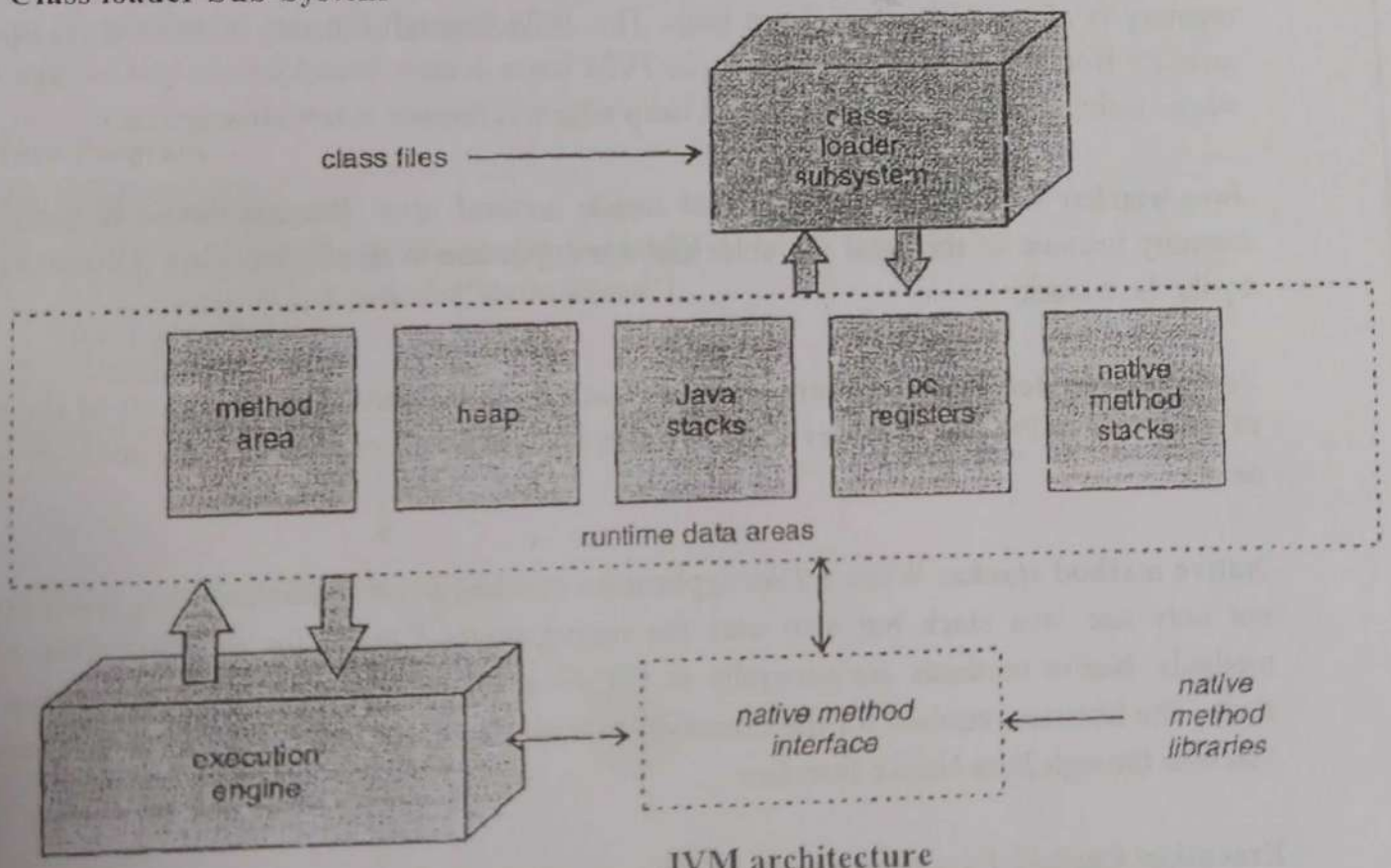


Fig. Process of Converting Byte Code into Machine Code

JVM consists of the following components as shown in the diagram. The components shown are abstract in nature. It completely depends upon the designer to develop these components.

Class loader Sub-System



JVM architecture

When a Java file is compiled, class files are generated. When we invoke a class file for execution, the class loader sub-system is called by Java Virtual Machine to load the class file into JVM memory space. The task of class loader sub-system is to load the classes and interfaces in the memory. Two types of loaders are available in a Java Virtual Machine: a boot strap loader, and a user-defined class loader. The job of the class loader sub-system is not only to load the

class file but to also check the correctness of the loaded class. It allocates the required memory to static variables (static variables are explained in class fundamental). The class loader sub-system performs its task in a sequential way:

(a) It loads a class file.
(b) It checks the correctness of class file. If anyone has manipulated the class file, then that class file cannot be executed. The class loader sub-system is responsible for this behavior.
(c) It allocates memory to static variables. Static variables are otherwise known as class variables. They are the property of classes rather than the property of objects.
(d) It transforms symbolic reference into direct reference.
(e) It sets default value of all static variables.

Method Area: It is a logical memory component of JVM. This logical section of memory holds information about classes and interfaces. Static variables are treated as class variables, because they take memory from method area. The size of the method area is not fixed. It shrinks and expands according to the size of the application. In Java when an object or an array is created, memory is allocated to them from heap. The JVM through the use of new operator allocates memory from the heap for an object. The JVM has a demon thread known as Garbage Collector whose task is to free those objects from heap whose reference is not alive in stack.

Java stacks: Method codes are stored inside method area. For execution, a method needs memory because of the local variables and the arguments it has taken. This purpose is fulfilled by the Java stack.

Program counter (PC) registers: It keeps track of the sequence of execution of the program. PC registers or Program counter register holds the addresses of the instructions to be executed next.

Native method stacks: When a Java application invokes a native method, that application does not only use Java stack but also uses the native method stack for the execution of native methods. Native methods are generally in C/C++ and are executed inside the native method stack. The libraries required for the execution of native methods are available to the Java Virtual Machine through Java Native Interface.

Execution Engine: Execution engine generates and executes the Java byte code. It contains an interpreter and Just-in Time compiler.

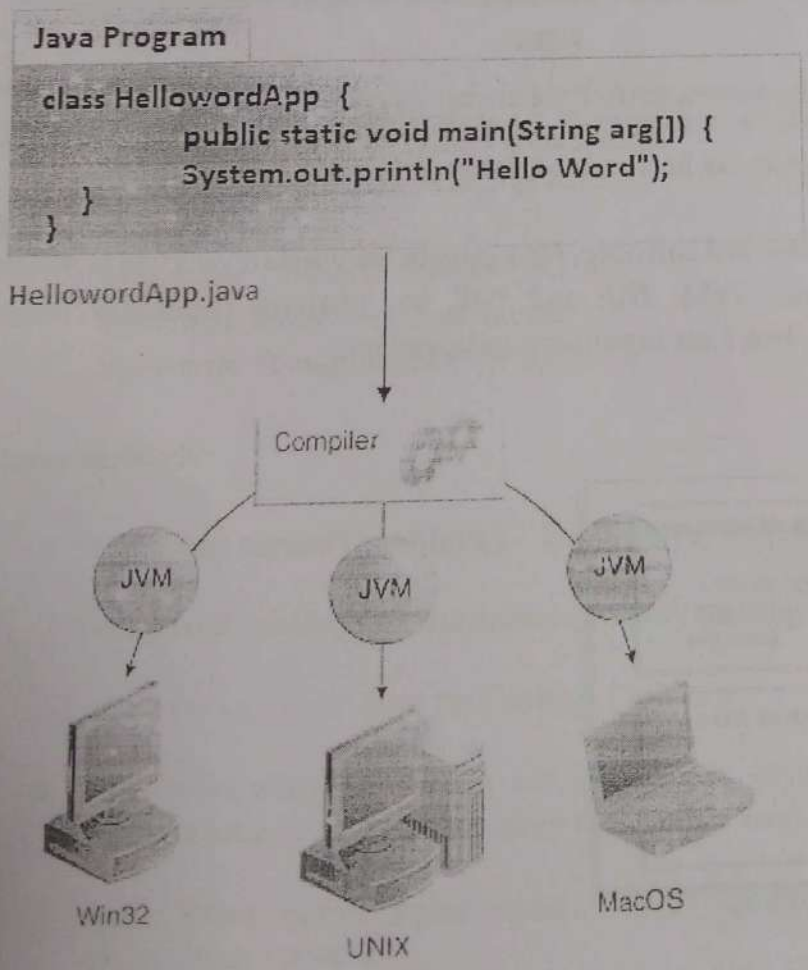
BYTE CODE: It is the unique characteristic property of the Java programming language. It is something like a normal text file. Therefore, it cannot be affected by virus. It is an intermediate between a human readable source and a machine readable source. Byte codes are platform-

independent. Therefore, JVM is platform dependent to make Java programming platform independent. Just like C source codes, the byte codes
are also portable. The cost of platform independent behavior of byte codes is paid by its slower execution speed when it is to the execution of native codes.

Q. How Java is Platform Independent

A programming language or technology is said to be platform independent if and only if which can run on all available operating systems with respect to its development and compilation. (Platform represents Operating System).

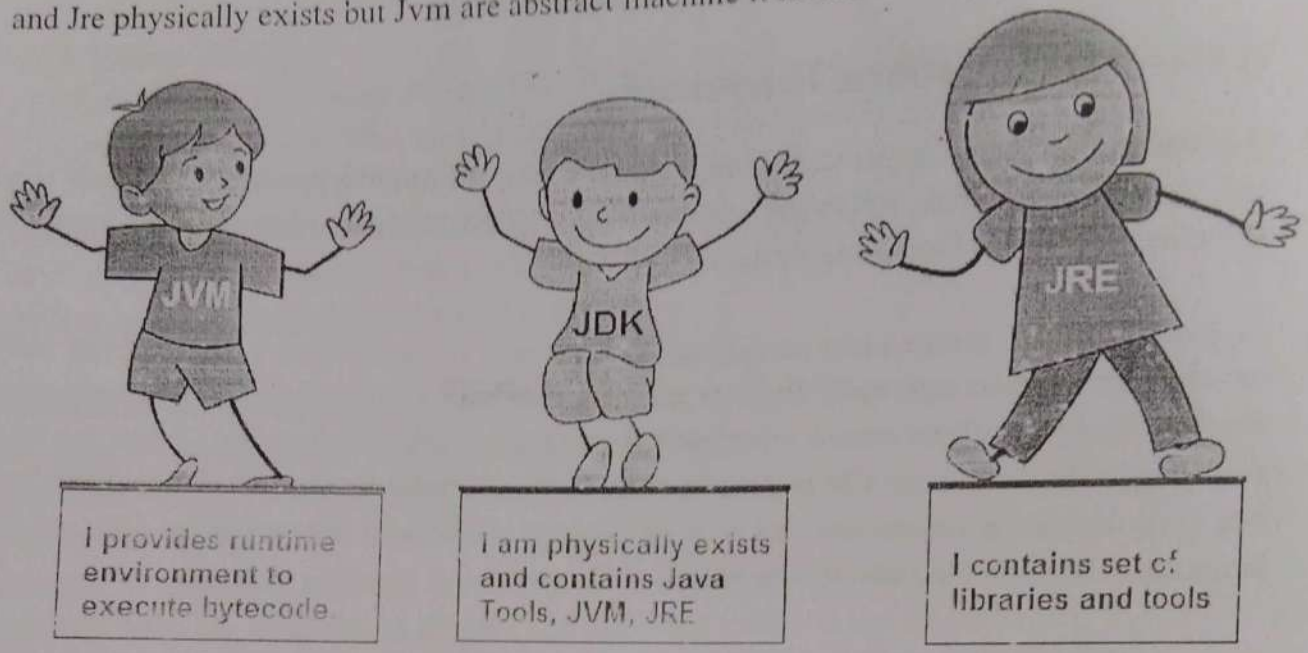
Java is a platform independent programming language, Because when you install jdk software on your system then automatically jvm are install on your system. For every operating system separate jvm is available which is capable to read .class file or byte code. When we compile your java code then .class file is generated by javac compiler these code are readable by jvm and every operating system have its own jvm so jvm is platform dependent but due to jvm java language is become platform independent.



Imp. Note: JAVA is platform independent but JVM is platform dependent.

Difference between JDK, JVM and JRE

Jvm, Jre, Jdk these all the backbone of java language. Each component has separate works. Jdk and Jre physically exists but Jvm are abstract machine it means it not physically exists.

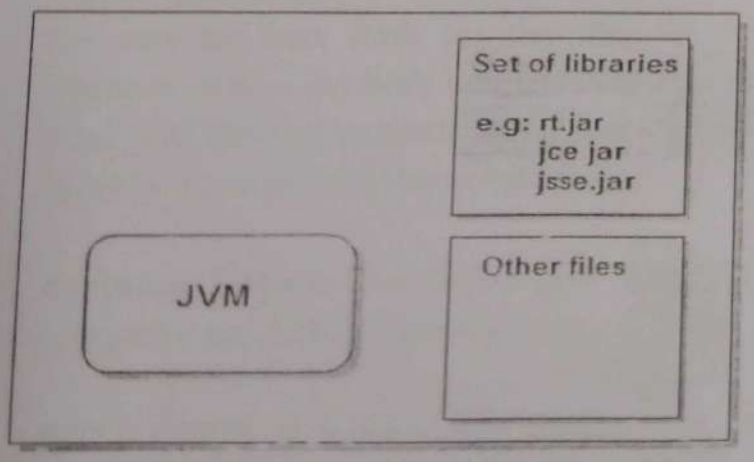


JVM

JVM (Java Virtual Machine) is a software. It is a specification that provides runtime environment in which java bytecode can be executed. It not physically exists.

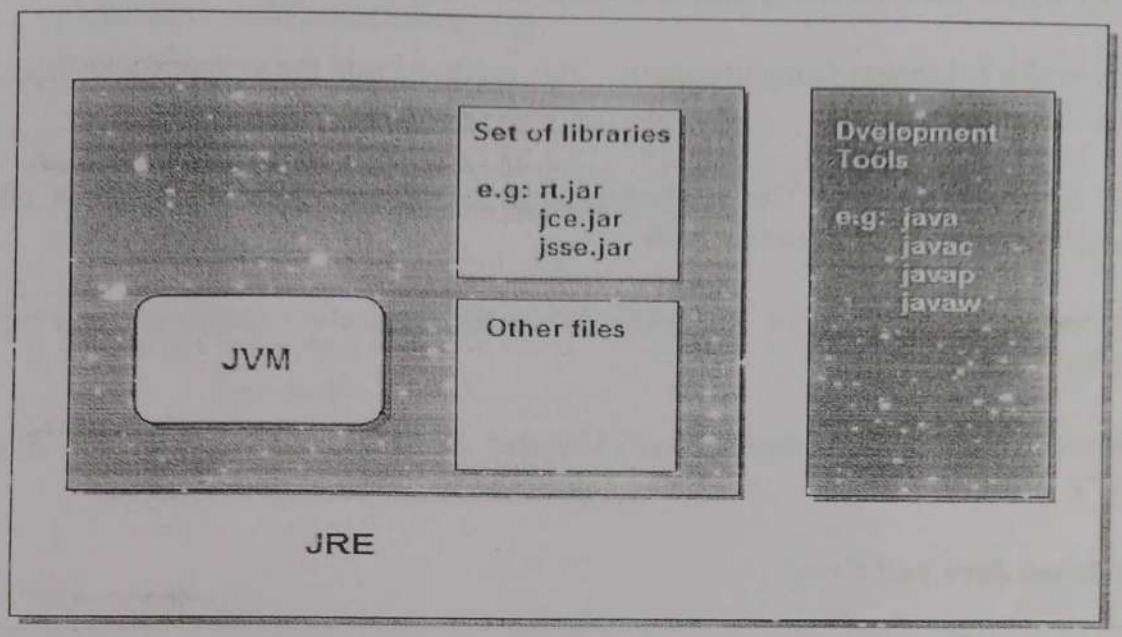
JVMs are not same for all hardware and software. for example for window os JVM is different and for Linux VJM is different. JVM, JRE and JDK are platform dependent because configuration of each OS differs. But, Java is platform independent.

JRE



JRE

The Java Runtime Environment (JRE) is part of the Java Development Kit (JDK). It contains set of libraries and tools for developing java application. The Java Runtime Environment provides the minimum requirements for executing a Java application. It physically exists. It contains set of libraries + other files that JVM uses at runtime.



JDK

The System class - The class provides System related information and also gives handle to some of the system functions like IO with console.

Static Fields: in, out, err.

System.out: It refers to output device.
System.in: It refers to input device.
System.err: It outputs error messages.

Some methods:

1. static long currentTimeMills() - It returns the current time in milliseconds
2. static void exit(int) - It terminates currently running JVM.
3. static void gc() – It runs the Garbage Collector.
4. static void arraycopy (object src, int srcStart, object dest, int destStart, int length) - It copies entire or part of one array to the other array.
5. static String getenv(String name) - This method gets the value of the specified environment variable.
6. static Properties getProperties() - This method determines the current system properties.

7. static SecurityManager getSecurityManager() - This method gets the system security interface.

8. static void load(String filename) - This method loads a code file with the specified filename from the local file system as a dynamic library.

9. static void loadLibrary(String libname) - This method loads the system library specified by the libname argument.

10. static long nanoTime() - This method returns the current value of the most precise available system timer, in nanoseconds.

11. static void runFinalization() - This method runs the finalization methods of any objects pending finalization.

12. static void setSecurityManager(SecurityManager s) - This method sets the System security.

Difference between Java and C++:

	Java		C++
1	It does not support operator overloading.	1	It supports operator overloading.
2	It does not provide the concept of template classes.	2	It provides the support for template classes.
3	It does not provide support for multiple inheritance, rather it is achieved by interfaces.	3	It provides the support for multiple inheritance.
4	It does not support the global variables.	4	It provides the support for global variables.
5	It does not provide support for explicit pointers.	5	It uses both implicit and explicit pointers.
6	Similar to C++, java does not use the destructor function, Instead executes the finalize( ) method to destroy the objects.	6	It uses the destructor function to destroy the objects.
7	It does not have any header file.	7	You need to include header files in C++.
8	It provides the support for exceptional handling.	8	It does not support exceptional handling.
9	It supports multithreading.	9	It do not support multithreading.
10	It supports automatic garbage collection.	10	It does not have automatic garbage collection.
11	Java does not support friend function & friend class concept.	11	C++ support friend function & friend class concept.
12	Its purely object oriented.	12	Its partially object oriented.

Programs:

Q.1 WAP to find factorial of a number.

```
import java.util.*;
class Factorial
{
    public static void main(String x[ ])
    {
        Scanner s=new Scanner(System.in);
        int n, fact=1;
        System.out.println("Enter a number");
        n=s.nextInt();

        for (int i=1;i<=n ;i++ )
        {       fact=fact*i;
        }
        System.out.println("Factorial="+fact);
    }
}
```

Q.2 WAP to print first n Fibonacci numbers.

```
import java.util.*;
class Fibonacci
{
    public static void main(String args[])
    {
        int a=0,b=1,c=1;
        Scanner s=new Scanner(System.in);

        System.out.print("How many Fibonacci numbers");
        int n=s.nextInt();
        System.out.println("Fibonacci Series");
        for (int i=1; i<=n; i++ )
        {
            System.out.println(c);
            c=a + b;
            a=b;
            b=c;
        }
    }
}
```

Output:

```
How many Fibonacci numbers  4
Fibonacci Series
1
1
2
3
```

Q.3 WAP to print Reverse of Number

```
import java.util.*;
class Reverse
{
    public static void main(String[] args)
    {
        int num, rev=0,temp;
        Scanner s=new Scanner(System.in);

        System.out.println("Enter a number");
        num=s.nextInt();
        while(num!=0)
        {
            temp=num%10 ;
            rev=rev*10+temp;
            num=num/10 ;
        }
        System.out.print("Reverse : "+rev);
    }
}
```

Output:

```
Enter a number
354
Reverse: 453
```

Q.4 1+2+3+……+n

Hint:

```
int sum=0;
for( int i=1; i<n; i++  )
{        sum=sum+I;
}
```

Q.5 WAP to print the sum of digits.

```
class SumOfDigits
{
        public static void main(String [ ]args)
        {
                int num, temp, sum=0;
                Scanner s=new Scanner(System.in);
                System.out.println("Enter a number");
                num=s.nextInt();

                while (num != 0)
                {               temp = num % 10 ;
                                sum = sum + temp ;
                                num = num / 10 ;
                }
                System.out.println("sum of digits is= "+sum);
        }
}
```

Output:

```
Enter a number
123
sum of digits is= 6
```

Q.6 WAP to find Permutation and Combination

```
import java.util.*;
class NP
{       public static void main(String args[])
        {
                int P , C ;
                int fact1=1,fact2=1,fact3=1;
                Scanner s=new Scanner(System.in);

                System.out.println("Enter the n and r");
                int n=s.nextInt();
                int r=s.nextInt();
```

```
        for (int i=1;i<=n; i++)//n!
        {
                fact1=fact1*i ;
        }

        for(int i=1; i<=r; i++)  // r!
        {
                fact2=fact2*i ;
        }

        for(int i=1; i<=(n-r); i++)  // (n-r)!
        {
                fact3=fact3*i ;
        }

        P = fact1 / fact2 ;
        C = fact1 / (fact2*fact3);

        System.out.print("nPr= "+P);
        System.out.print("\t nCr= "+C);
    }
}
```

Output:

```
Enter the n and r
6
2
nPr= 360  nCr= 15
```

Q.7 WAP to test whether the no is Armstrong.

```
import java.util.*;
class Palindrome
{
  public static void main(String[] args)
  {
    int num,sum=0,temp;
    Scanner s=new Scanner(System.in);

    System.out.println("Enter a number");
    num=s.nextInt();

    int n=num;
    while(num!=0)
    {
      temp =num%10 ;
      sum =sum+ temp*temp*temp;
      num = num/10 ;
    }

    if (sum==n)
    {
        System.out.println("Number is Palindrome");
    }
    else
    {
        System.out.println("Number is Not a Palindrome");
    }
  }
}
```

Output:

```
Enter a number
153
Number is a Palindrome
```

Q.8 WAP to whether number is prime or not.

```
import java.util.*;
class TestPrime
{
        public static void main(String[] args)
        {
                Scanner s=new  scanner(System.in);
                System.out.println("Enter a no");
                int num=s.nextInt();

                int b=2;

                while( n % b !=0)
                {
                        b++;
                }
                if (n==b)
                {
                  System.out.println("It is prime no");
                }
                else
                {
                  System.out.println("Not prime no");
                }
        }
}
```

Q. 9 $1/2^2 - 1/4^2 + 1/6^2 - 1/8^2 + \ldots\ldots\ldots 1/n^2$

Hint:

```
int sum=0,s=1;
for ( i=2; i<=n ; i+=2 )
{
    sum=sum + 1.0/(i*i*s);
    s= -1x s;
}
```

Q.10 WAP to calculate GCD and LCM

```
class GCDLCM
{
        public static void main(String x[ ])
        {
                int n1,n2,num1,num2,lcm;
                Scanner s=new Scanner(System.in);

                System.out.print("Enter any two numbers");
                num1=s.nextInt();
                num2=s.nextInt();
                n1=num1;
                n2=num2;
                while (n1 != n2 )
                {
                        if(n1>n2)
                        {               n1=n1-n2;
                        }
                        else
                        {               n2=n2-n1;
                        }
                }
                lcm=(num1*num2)/n1;
                System.out.println("GCD="+n1);
                System.out.println("LCM="+lcm);
        }
}
```

Output:

```
Enter any two numbers
150 130
GCD=10
LCM=1950
```

Q.11 Convert Decimal number to binary number.

```
import java.util.*;
class Binary
{
    public static void main(String[] args)
    {
        int a[ ]=new int[10];
        int i;

        Scanner s=new Scanner (System.in);
        System.out.print ("Enter a decimal no");
        int num=s.nextInt();

        for (i=0; num!=0; i++)
        {
            a[ i ]= num%2;
            num = num/2;
        }

        System.out.print ("Required number=:");
        for (i=i-1 ; i>=0 ; i--)
        {
            System.out.print (a[i]+" ");
        }
    }
}
```

Output:

```
Enter a decimal no 10
Required number=:1 0 1 0
```

Q.12 $1^2+2^2+3^2+\ldots\ldots\ldots n^2$

Hint:

```
int sum=0;
for( int i= 1 ; i<=n ;i++)
{
    sum=sum+i *i ;
}
```

Q.13 WAP to print alphabets and their Ascii values

```
class TestAscii
{
   public static void main(String[] args)
   {
      System.out.println ("Character\tAscii");
      for (char i='A';i<='Z';i++)
      {
            System.out.println (i+"\t\t"+(int)i);
      }
       for (char i='a';i<='z';i++)
       {
            System.out.println (i+"\t\t"+(int)i);
       }
   }
}
```

Q.14 WAP to print Ascii values and corresponding character equivalent

```
class TestAscii1
{
  public static void main(String[] args)
  {
      System.out.println ("Ascii\t\tCharacter");

      for (int i=65;i<=90;i++)
      {
              System.out.println (i+ "\t\t" + (char)i)
      }

      for (int i=97;i<=122;i++)
      {
              System.out.println (i + "\t\t"+ (char)i ):
      }
   }
}
```

Q.15 1+2+3+..... +n

Hint:

```
double sum=0;
for( int i=1; i<n; i++ )
{
        sum = sum+i;
}
```

Q.16 1/1+1/2+1/3+.....1/n

Hint:

```
double sum =0;
for (int i=1 ; i<=n ; i++ )
{
   sum =sum + 1.0/i ;
}
```

Q.17. 1/2 +3/4+5/6+--------n/(n+1)

Hint:

```
double sum=0;
for (int i=1 ; i<=n ; i=i+2 )
{
   sum =sum + i /( i+1);
}
```

Q.5 WAP to calculate total time taken by program for execution.

```
class cpu
{
  public static void main(String args[])
  {
        long Initial=System.currentTimeMillis();
        double s=0;
        for(int i=1;i<=100000000;i++)
        {
                s=s+1/ i ;
        }

        long Final=System.currentTimeMillis();
        System.out.println("Time required= "+(Final - Initial)+" ms");
        }
}
```

Output: Time required=1576ms

2. OBJECT ORIENTED PROGRAMMING

Object is the physical as well as logical entity whereas class is the only logical entity.

class: Class is a blue print which is containing only list of variables and method and no memory is allocated for them. A class is a group of objects that has common properties.

A class in java contains:

Data Member
Method
Constructor
Block
Class and Interface

Object: Object is a instance of class. object has state and behaviors.

An Object in java has three characteristics:

State
Behavior
Identity

State: Represents data (value) of an object.

Behavior: Represents the behavior (functionality) of an object such as deposit, withdraw etc.

Identity: Object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. But, it is used internally by the JVM to identify each object uniquely.

Class is also can be used to achieve user defined data types.

	Class	Object
1	Class is a container which collection of variables and methods.	object is a instance of class
2	No memory is allocated at the time of declaration	Sufficient memory space will be allocated for all the variables of class at the time of declaration.
3	One class definition should exist only once in the program.	For one class multiple objects can be created.

Inheritance: When one object acquires all the properties and behaviours of parent object i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Abstraction: Data abstraction refers to, providing only essential information to the outside world and hiding their background details, i.e., to represent the needed information in program without presenting the details.

Hiding internal details and showing functionality is known as abstraction. For example: phone call, we don't know the internal processing. In java, we use abstract class and interface to achieve abstraction.

Encapsulation: Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Polymorphism: Polymorphism in java is a concept by which we can perform a single action by different ways. Polymorphism is derived from 2 greek words: poly and morphs. The word "poly" means many and "morphs' means forms. So polymorphism means many forms.

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

If you overload static method in java, it is the example of compile time polymorphism. Here, we will focus on runtime polymorphism in java.

Runtime Polymorphism in Java

Runtime polymorphism or Dynamic Method Dispatch is a process in which a call to an overridden method is resolved at runtime rather than compile-time.

In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

Message Passing: Objects can communicate with each other's by passing message same as people passing message with each other. Objects can send or receive message or information. Message passing involves name of object, name of function (message) and information to be send.

For example, student.mark(name). Here student is object, mark is message, name is information.

Variable:

Variable is a memory location where actual values are stored. Variable is one whose value always changes throughout the program. In another words Variable is nothing but its name given to a data.

The variable gives following information i) The data type it can hold ii) Its name iii) Its value

We distinguish between the lifetimes of variables in 3 contexts with respect to the class

1) **Instance Variables (Non-static variables):** Instance Variables are members of a class and created for each object of the class and will have its own copies of these variables which are local to the objects. The values of these variables at any given time constitute the state of the object. Instance variables exist as long as the object they belong exists. Instance variables with same names cannot be in a single class.

2) **Static Variables (class variables):** Static Variables are also the members of a class, but not created for any object of a class and therefore belong only to the class. They are created when the class is loaded at runtime and exist as long as the class exists. Only one copy of static variable is created and gets shared amongst all the object.

3) **Local Variables:** Variables which are declared inside methods or the block are called as local variable. Local variable is created for each execution of the method or block. After the execution of the method or blocks, Local variables are no longer accessible.

Access Specifiers - There are four types of access specifiers present in Java. These are:

(1) public

(2) no-access or default access

(3) protected

(4) private

Classes encapsulate the data-members and methods defined inside them. According general conventions of OOP, the implementations of the methods and the data-members should remain hidden inside the class body, by default. The class programmer can control the access to the members of the class. The users of the class can access only those members which the class-programmers allows.

There are four levels of access control in Java, one of them is default (also called as friendly access) Java defines three keywords to specify the access control for the members of the class as well as for the class it-self. These are: private, protected and public. One of these keywords can be written before any member declaration or definition.

In above example the data-member sum, of the class Some Class is declared as private member of the class and the method as public member. Following points discuss the use and working of these keywords.

private:

- Private members of a class are accessible only inside the class. i.e. these members can be accessed only by the other members of the same class.
- Classes cannot be private, but inner classes can be declared private.
- Private members are not accessible out of the class anywhere as well as not accessible to the sub-classes i e. private members of class are not inherited into sub-classes.
- Usually the data-members of a class are kept private. Sometimes, methods are declared private, if these methods are to be used inside the class only, and not to be accessed by outsiders.
- Obviously, if private, they will not be accessible outside the current package.

protected:

- Protected members of a class are accessible inside the body of a class, or to any of its sub-class.
- Also, protected members of a class are accessible in the classes in the same package, through object of that class. For other classes they behave like private members of the class.
- A protected member of a class is inherited into the sub-class with the same access control hence, for a sub-class of a sub-class the member is accessible.
- Normal class cannot be declared protected, but inner class can be protected.

public:

- public members of a class are accessible inside the class, as well as anywhere outside the class also.
- These members of a class accessible to any part of a program, but through object of class.
- If member is public and static it can be accesses by any part of a program by using class name directly.
- Public members are accessible to the sub-classes lying anywhere in the program i.e. inside the same package or outside the package.
- If class or a class member is declared public it can be used outside the package also.
- A class is declared public to make it accessible outside the current package.

default access:

- When members of a class are defined without any access specifier, they are given a default access. Such members are partly public i.e. they are accessible to the other parts of the program (through objects) which are in the same package.
- Also, they are inherited into sub-classes with same default access and are accessible in the subclasses.
- If a class is defined without any access specifier i.e. with default access, the class is accessible to the classes in the current package.

Ex.

```
class Data
{
        private int m; // private variable
        private int n; // private variable
        int p; //default variable
        public void setdata() // public method
        {
                //code
        }
        // code for other members.
}
```

Static Variables and Static Methods

Static variables are declared outside the method and constructor. Variables and methods can be declared as static by preceding declaration by a keyword static and these are known as static variables. **There is only one copy of the static variable that exists for a particular class. These members are common to all objects and belong to the class.**

They are defined as follows:

```
static int count;                    // static variable
static void SomeFunction( );         // static method
```

static members are accessed using the class name followed by the dot (.) operator. Since these members belong to a class rather than individual objects they are known as **class variables** and **class methods.** static variable is useful is to keep count of how many objects of a class are created.

When same data need to be shared by all the objects, we declare that variable as static.

Rules for declaring Static Members:

1. Object of a class cannot refer static members.
2. static method can access only static variables.
3. static variables can't refer to 'this' keyword or super keyword in any way.
4. To initialize the static variables, they should be declared inside a block which is executed once when class is loaded.
5. The block should be preceded by a keyword called as static.
6. static members can be accessed directly without creating object of a class.
7. If static method to be called is present in another class then it is called by classname.methodname( )
8. If static method is to be called is present in same class then it can be called by its name only.

Program1:

```
class Demo
{
    static void fun ()
    {
        System.out.println("Hello!");
    }

    public static void main (String x[ ])
    {
      fun (); //fun is a static method and main is static therefore main can call fun directly.
      Demo.fun () ;    // fun can be called through class name since fun is static.
      Demo x=new Demo();
      x.fun() ;    //fun can also be called through reference name
    }
}
```

Output:

```
Hello!
Hello!
Hello!
```

Program2

```
class Demo
{
        public static void main (String f[ ])
        {
            Demo x=new Demo ();
            x. funl () ;
        }

        static void fun ()
        {
            System.out.println("Hello!"};
        }

        void fun1()
        {
            System.out.println("Hi");

            fun() ;  // fun is called directly from a non-static method.
            Demo.fun() ;   // fun is called from non-static method through class name.

            Demo x=new Demo();
            x. fun();   //fun is called from a non-static method through object name.
        }
}
```

Output:

```
Hi
Hello!
Hello!
Hello!
```

Program3

```
class MathOperation {
   static int mul(int x,int y)
   {
        return(x*y);
   }

   static int divide(int x,int y)
   {
        return(x/y);
   }
}
```

```
class MathApplication
{
        public static void main(String args[])
        {
                int a = MathOperation.mul(3,5);
                System.out.println("a ="+a);   // a=15

                int b = MathOperation.divide(a,2);
                System.out.println("b ="+b): //b=7
        }
}
```

Difference between non-static and static variable

	Non-static variable	Static variable

1	These variable should not be preceded by any static keyword Example:
	class A
	{
	int a;
	}
		These variables are preceded by static keyword.
		Example
		class A
		{
		static int b;
		}

2	Memory is allocated for these variable whenever an object is created	Memory is allocated for these variable at the time of loading of the class.
3	Memory is allocated multiple time whenever a new object is created.	Memory is allocated for these variable only once in the program.
4	Non-static variable also known as instance variable while because memory is allocated whenever instance is created.	Memory is allocated at the time of loading of class so that these are also known as class variable.
5	Non-static variable are specific to an object	Static variable is common for every object that means there memory location can be sharable by every object reference or same class.
6	Non-static variable can access with referencename. Syntax referencename. variablename	Static variable can access with class name Syntax class_name.variable_name

Note: static variable not only can be access with class reference but also some time it can be accessed with object reference.

Difference between Static and non-static method in Java

In case of non-static method memory is allocated multiple time whenever method is calling. But memory for static method is allocated only once at the time of class loading. Method of a class can be declared in two different ways

- Non-static methods
- Static methods

Difference between non-static and static Method

	Non-Static method	Static method

1	These methods never be preceded by static keyword
	Example:
	
	void fun1()
	{
	
	.....
	
	.....
	
	}
		These methods always preceded by static keyword
		Example:
		
		static void fun2()
		{
		
		.....
		
		.....
		
		}

2	Memory is allocated multiple time whenever method is calling.	Memory is allocated only once at the time of class loading.
3	It is specific to an object so that these are also known as instance method.	These are common to every object so that it is also known as member method or class method.
4	These methods always access with object reference Syntax: Objref.methodname();	These property always access with class reference Syntax: className.methodname();
5	If any method wants to be execute multiple time that can be declare as non static.	If any method wants to be execute only once in the program that can be declare as static .

Note: In some cases static methods not only can access with class reference but also can access with object reference.

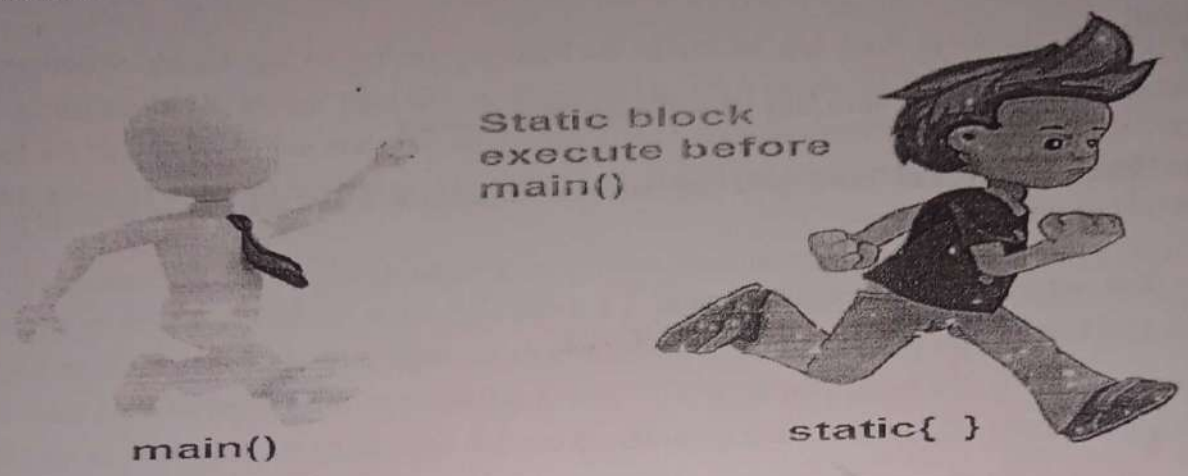
Difference between Method and Constructor?

	Method	Constructor
1	Method can be any user defined name	Constructor must be class name
2	Method should have return type	It should not have any return type (even void)
3	Method should be called explicitly either with object reference or class reference	It will be called automatically whenever object is created
1	Method is not provided by compiler in any case.	The java compiler provides a default constructor if we do not have any constructor

static block in java

static block is a set of statements, which will be executed by the JVM before execution of main method. At the time of class loading if we want to perform any activity we have to define that activity inside static block because this block executes at the time of class loading.

In a class we can take any number of static block but all these blocks will be executed from top to bottom.



Syntax

```
static
{
        ..............................
        //Set of Statements
        ..............................
}
```

Note: In real time application static block can be used whenever we want to execute any instructions or statements before execution of main method.

Example of static Block

```
class StaticDemo
{
        static
        {
                System.out.println("Hello how are u ?");
        }
        public static void main(String args[])
        {
                System.out.println("This is main()");
        }
}
```

Output

Hello how are u ?
This is main()

Run java program without main method

```
class StaticDemo
{
        static
        {
                System.out.println("Hello how are u ?");
        }
}
```

Output:
Hello how are u ?
Exception is thread "main" java.lang.no-suchmethodError: Main

Note: "Exception is thread "main" java.lang.no-suchmethodError: Main" warning is given in java 1.7 and its above versions

More than one static block in a program

```
class StaticDemo
{
        static
        {
                System.out.println("First static block");
        }
        static
        {
                System.out.println("Second Static block");
        }
        public static void main(String args[])
        {
                System.out.println("This is main()");
        }
}
```

Output
First static block
Second static block
This is main()

Note: "Here static block run according to their order (sequence by) from top to bottom.

Why a static block executes before the main method ?

A class has to be loaded in main memory before we start using it. Static block is executed during class loading. This is the reason why a static block executes before the main method.

Constructors

Constructor will be called automatically when the object is created. Constructor name must be similar to name of the class. Constructor should not return any value even void also. Because basic aim is to place the value in the object. (if we write the return type for the constructor then that constructor will be treated as ordinary method). Constructor definitions should not be static. Because constructors will be called each and every time, whenever an object is creating.

Constructor should not be private provided an object of one class is created in another class (Constructor can be private provided an object of one class created in the same class). Constructors will not be inherited from one class to another class (Because every class constructor is created for initializing its own data members). The access specifier of the constructor may or may not be private.

If the access specifier of the constructor is private, then an object of corresponding class can be created in the context of the same class but not in the context of some other classes. If the access specifier of the constructor is not private, then an object of corresponding class can be created both in the same class context and in another class context. The main purpose of a Constructor is to initialize the Object. If there is no constructor defined by the programmer, then JVM automatically executes the default constructor.

Default constructor: A constructor having no parameters is called the default constructor.

- If you define no constructors at all for a class, a trivial default constructor (with the empty body) is automatically generated.
- If you define any constructor, no default constructor is generated automatically and you will have to write your own.
- It is a good idea to have a default constructor, because client programmers usually assume its existence.

Program:

```
class Demo
{
        Demo( )  //default constructor
        {
            System.out.println("\nHello, World!!");
        }

        public static void main (String a[ ])
        {
            Demo x=new Demo();
        }
}
```

parameterized Constructor

A constructor which contains variables as parameters is called as parameterized constructor. It can be used to set different values to different objects of same class.

Program:

```
class Employee
{
        String name, address;
        long ph;
        double sal;

        Employee (String n, String a, long p, double s) //parameterized constructor.
        {
                name = n;
                address = a;
                ph = p;
                sal = s;
        }

        void display ()
        {
                System.out.println("Name = " +n "address = " +a "Phone No. = " +p "Salary = " +s);
        }
}

class EmployeeDemo
{
        public static void main(String args[ ])
        {
                Employee e = new Employee("Abhi", "Andheri", 26353993, 60000);
                e.display( );
        }
}
```

Constructor Overloading

1) When more than one constructor contains different number of parameters and type constructors are said to be overloaded.
2) The proper overloaded constructor is executed based upon the parameters specified when new is executed.
3) When constructor is overloaded then class must have a default constructor.
4) Constructors having same number of parameters then they must be different in the datatype.
5) Overriding is not possible at constructor level because the scope of constructor is within the class so that it is not possible to achieved overriding at constructor level.

Program:

```
class Volume
{
        double v;
        int len, bre, ht, rad, side;

        Volume(int l, int b, int h)
        {
                len = l;
                bre = b;
                ht = h;
        }

        Volume(int r, int h)
        {
                rad = r;
                ht = h;
                v = 3.14 * rad * rad * ht;
        }

        Volume(int s)
        {
                side = s;
                v = side * side * side;
        }

        Volume()
        {
                System.out.println ("Constructor Overloading");
        }

        void display( )
        {
                System.out.println ("Volume = " +v);
        }
}
class OverloadDemo {
        public static void main (String x [ ]) {
                Volume obj=new Volume ();

                Volume obj1 = new Volume (10, 20, 30);
                obj1.display ( );

                Volume obj2 = new Volume (5, 10);
                obj2.display ( );

                Volume obj3 = new Volume (13);
                obj3.display ( );
        }
}
```

The toString( ) Method

The toString() method is defined in 'Object' class. The toString () method returns a string that contains a description of the object on which it is called. This method is automatically called when an object reference is passed to print() or println() Method. It is also called when reference is used in concatenation expression. Every class implements toString() because it is defined by object.

General Form: **public String toString ();** // toString () return the description of an object.

To implement toString( ), simply return a string object that contains the human riddle string that approximately describes an object of your class

```
class Student
{
        String name;
        int roll;

        Student(String s, int r)
        {
                name=s;
                roll=r;
        }

        public string toString( )
        {
                return "Name="+name+ "\nRoll no="+roll;
        }
}
class StudentDemo
{
    public static void main(String x[ ])
    {
        Student s1=new Student("Amit",22 );
        String s2= "Student Details:\n:" +s1;
        System.out.println(s1);
        System.out.println(s2);
    }
}
```

Output:

```
Name= Amit
Roll no=22

Student Details:
Name= Amit
Roll no=22
```

continue: Continue is used to manually go down to next iteration without reaching the end of the current block.

Program:

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

Output: 1 3 5 7 9

Label break: In case of a labeled break the break should be followed by the labeled name.

```
class LabelBreak {
        public static void main(String args[]) {
                One:
                {
                        Two:
                        {
                                Three:
                                {
                                        System.out.println("Before");

                                        if(true)
                                        {
                                                break Two;
                                        }
                                        System.out.println("After");
                                }
                                System.out.println("After3");
                        }
                        System.out.println("After2");
                }
                System.out.println("After1");
        }
}
```

Output:
Before
After2
After1

Loop break: To throw the control out of loop.

```
class BreakFor
{
        public static void main(String[] args)
        {
                for (int i = 1; i < 10 ; i++ )
                {
                        System.out.println(i);
                        if(i = = 4)
                        {       break;
                        }
                }
        }
}
```

Output: 1 2 3

'this' keyword: It returns the starting memory address of calling object.

this: It refers to not static variables. If local variables and non-static variables are having the same name then local variables hides the identity of non-static variables. To access the non-static variables in presence of local variable, we add the 'this' keyword explicitly before every non-static variable.

```
class This1
{
        int a;

        This1(int a)
        {
                this.a = a;
        }
        public static void main(String args[])
        {
                This1 m = new This1(10);
                System.out.println(m.a);
        }
}
```

Output: 10

```
class This2
{
        int x=10;

        void met()
        {
                int x = 40;
                System.out.println(x+" ");
                System.out.println(this.x + " ");
        }
```

```
        public static void main(String args[])
        {
                This2 m = new This2();
                m.met();
                System.out.print(m.x + " ");
        }
}
```

Output: 40 10 10

this: when non-static method or constructor is executed, java interpreter adds 'this' keyword implicitly before **every non**-static variable.

```
class This3 {
        String name;
        int sal;

        This3(String s, int i)
        {
                name=s;
                sal=i;
                System.out.println("Name   : "+name);
                System.out.println("Salary : " + sal);
        }
        This3(String s)
        {
                this(s,0);
        }
        This3(int i)
        {
                this("Unknown",i);
        }
        This3()
        {
                this(0);
        }
        public static void main(String args[]) {
                This3 w = new This3("Maya", 1000);
                This3 x = new This3("Sangeeta");
                This3 y = new This3(5000);
                This3 z = new This3();
        }
}
```

Output:

```
Name   : Maya
Salary : 1000
Name   : Sangeeta
Salary : 0
Name   : Unknown
Salary : 5000
Name   : Unknown
Salary : 0
```

Java is Garbage Collector

C++ allows de-allocating memory space for the object dynamically by using delete operator. In Java de -allocation takes place automatically. The technique is called as Garbage Collection. When no reference to an object exists, that object is assumed to be no longer needed and the memory occupied by the Object can be reclaimed. There is no explicit need to destroy the object in C++. Garbage Collection can also be known as **Litter Recycling.** The Garbage Collection runs wherever the system is idle, or when a requested allocation to fails to find enough memory. Garbage Collection can occur sporadically during the execution of your program.

The garbage collector is under the control of the JVM. The JVM decides when to run the garbage collector but there are no guarantees, under any circumstances, that the JVM will comply. JVM will typically run the garbage collector when it senses that memory is running low. When your Java program makes a request for garbage collection by calling System.gc() method, JVM will usually runs garbage Collector but there is no guarantee. When object is without reference or no live thread can access it, such objects are ready for garbage collector.

The garbage collection routines that Java provides are members of the Runtime class. The Runtime class is a special class that has a single object (a Singleton) for each main program. The Runtime object provides a mechanism for communicating directly with the virtual machine. In order to get the Runtime object, you can use the method Runtime.getRuntime(), which returns the Singleton. The simplest way to for garbage collection (remember—just a request) is System.gc();

Theoretically, after calling System.gc(), you will have as much free memory as possible. We say theoretically because this routine does not always work that way. First, the JVM you are using may not have implemented this routine; the language specification allows this routine to do nothing at all. This is not to say that System.gc() is a useless method—it's much better than nothing. You just can't rely on System.gc() to free up enough memory so that you don't have to worry about the garbage collector being run.

The following program lets us know how much total memory the JVM has available to it and how much free memory it has. It then creates 10,000 Date objects. After this, it tells us how much memory is left and then calls the garbage collector (which, if it decides to run, should halt the program until all unused objects are removed). The final free memory result should indicate whether it has run. Let's look at the program:

```
import java.util.*;
class CheckGC
{
        public static void main(String [] args)
        {
            Runtime rt = Runtime.getRuntime();

            System.out.println("Total JVM memory: " + rt.totalMemory());
```

```
        System.out.println("Before Memory = " + rt.freeMemory());

        Date d = null;
        for(int i = 0;i<10000;i++)
        {
            d = new Date();
            d = null;
        }

        System.out.println("After Memory = " + rt.freeMemory());
        rt.gc(); // an alternate to System.gc()
        System.out.println("After GC Memory = " + rt.freeMemory());
    }
}
```

Output:

```
Total JVM memory: 1048568
Before Memory = 703008
After Memory = 458048
After GC Memory = 818272
```

Finalization

Object sometimes hold the non-java resources such as file handle or window characters front before destroying object of class, these resources must be freed. To handle this situation, java provides a mechanism called Finalization. For this purpose, finalize( ) method which is called at run time just before destroying the object of a class. Inside the finalize ( ) you will specify those actions that must be performed before an object is destroyed.

Finalize is a method of the class object which is called automatically by garbage Collector (gc) when an object becomes unreferenced. The garbage Collector can called explicitly by calling static function gc() of the System class.

```
System.gc();
```

The finalize method can be used to release resources like files, databases associated with the object.

General form of finalize( ) method.

```
protected void finalize( )
{
        // finalization code
}
```

Here, the keyword protected is specifies that prevents access to finalize( ) by code defined outside its class.

Thus, we can say that finalize performs works similar to destructors.

Program:

```
class FinalizeEx
{
        static int c;

        FinalizeEx()
        {
                c++;
                System.out.println("Object created="+c);
        }

        protected void finalize()
        {
                System.out.println("Object destroyed="+c);
                c--;
        }
}

class FinalizeExDemo
{
        public static void main(String args[])
        {
                FinalizeEx x1=new FinalizeEx();
                FinalizeEx x2=new FinalizeEx();
                FinalizeEx x3=new FinalizeEx();

                x1=null;
                x2=null;
                x3=null;
                System.gc(); // It calls finalize( ) method
                System.out.println("hello");
        }
}
```

Output:

```
Object created=1
Object created=2
Object created=3
hello
Object destroyed=3
Object destroyed=2
Object destroyed=1
```

Constants in Java

Constants can be declared in Java. Constant in java can be declared by using the keyword **final**. The value of cannot change throughout the program. The keyword **final** indicates that you can assign o the variable some value once then its value doesn't change throughput the program. In Java, by convention, constants are variables generally declared in **uppercase** only Ex. final int MAX = 100; where MAX is an integer constant and its fixed value is 100.

Hash code

When an object is created through a new operator by calling the constructor of the corresponding class, a unique identifier is assigned to the reference variable, known as hash-code. Hash-code is allotted by JVM. This can be seen in the following example:

Program:

```
class p
{       public static void main (String args[ ])
        {
                p x=new p ();
                System.out.println(x.hashCode());
        }
}
```

The method hashCode() returns the hash-code of the corresponding reference when an object is created. Hash-code is assigned to a reference variable only when memory is allocated from the heap area.

The local variables of primitive data types are created inside the stack area and the memory is also allocated from there. However, all the above complexities are not involved here. The local variables of primitive data types do not have any hash-code because they do not acquire memory from the heap area. To get a hash-code, the variable must be of reference type. This is shown by the following example:

Program:

```
class p
{       public static void main (String f[ ])
        {
                int mak=5;
                System.out.println (mak.hashCode());
        }
}
```

The above program will result in a compilation error because mak is not a reference variable and memory is not allocated to it from heap. The error is 'int' cannot be dereferenced'. The next question that arises is whether it is possible to allocate memory to variables of primitive data type from heap or not. The answer to this is yes.

System.arraycopy()

If we want copy of an array into another array, we must make a new array of the same length as the original and copy overall values as follows:

```
double [ ] prices = new double[data.length];

for(int i=0; i < data.length; i++)
{
        prices[i] = data[i];
}
```

Instead of the for loop you can also use the static System.arraycopy method. The method can be used to copy any portion of an array into another array.

System.arraycopy (from, fromStart , to, toStart , count);

from : name of the source array.
fromstart: start position in the source array
to: name of destination array
tostart: position in the second array from where array will get copied.
count: no of the elements to be copied from the source array.

To copy the entire data array into the prices array we say:

System.arrayCopy(data.0,prices,0,data.length);

Example:

```
class ArrayCopy
{
        public static void main(String[] args)
        {
                int x[] = {1,2,3,4,5};
                int y[] = new int[5];

                System.arraycopy(x,0,y,0,5);

                System.out.println("Array y after copying is :");

                for(int i=0;i<5;i++)
                {
                        System.out.print(" " + y[i]);
                }
        }
}
```

Output:

Array y after copying is : 1 2 3 4 5

Command Line Arguments

A Java application can accept any number of arguments from the command line. This allows the user to specify configuration information when the application is launched. The user enters command-line arguments when invoking the application and specifies them after the name of the class to be run. In other way, when the input values are passed along with interpreter's command then it initialise the String array which is nothing but compulsory parameter of main method.

Passing *command arguments* to a Java program

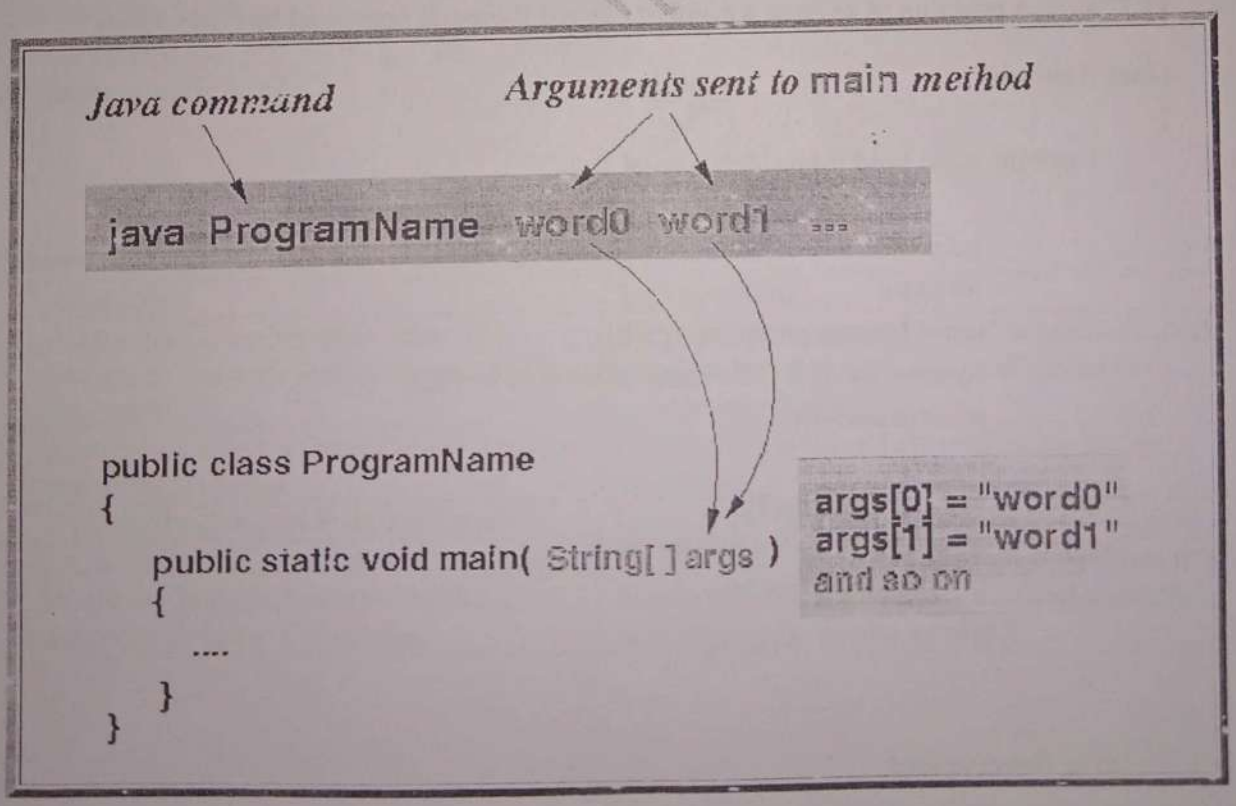
- **Fact:**

  - When we execute a Java program `ProgramName` using the following syntax:

    ```
    java  ProgramName  word0  word1  ...
    ```

    the Java system will *pass* the words:

    - `word0` to the *first* `String` parameter variable
    - `word1` to the *second* `String` parameter variable
    - And so on.



For example, consider the 'Good Morning' program we have written in previous practical , modify the program to say Hello to the user executing the program using command line argu We have to do following changes ,

Example

```
class Hello
{
        public static void main(String args[])
        {
                String str;
                str = args[0];
                System.out.println("Good " + str);
        }
}
```

To execute this program follow these steps

```
C:\> javac Hello.java
C:\> java Hello Morning
Good Morning
```

Programs on Command Line Arguments

Q.1 Write a program to reverse a number entered thorough command line arguments.

```
class Reverse
{
        public static void main(String args[ ])
        {
                int rem;
                int num;
                num=Integer.parseInt(args[0]);
                System.out.print("Reverse of number is=");
                while(num!=0)
                {
                        rem=num%10;
                        System.out.print(+rem);
                        num=num/10;
                }
        }
}
```

```
C:\> javac Reverse.java

C:\> java 12

Reverse of number is= 21
```

Q.2 Write a program to find the largest of any numbers passed to it through command line arguments

```
class largest
{
        public static void main(String args[])
        {
                int i, count, max, temp;
                count= args.length;
                max = Integer.parseInt(args[0]);
                System.out.println(max);
                for(i=0; i<count ; i++)
                {
                        temp = Integer.parseInt(args[i]);
                        if(temp > max)
                        {
                                max=temp;
                        }
                }
                System.out.println("The largest number is " + max);
        }
}
/*Output
C:\> javac largest.java
C:\> java largest  2  10  6  20  5
The Largest number is 20
*/
```

Difference between Abstraction and Encapsulation

Sr.	Abstraction	Encapsulation
1	Abstraction is a concept is formed by extracting common features from specific examples	Encapsulation is mechanism that binds the code and data together and keep it safe from the outside interference
2	Abstraction is implemented using interface and abstract classes	Encapsulation is implemented using access specifiers
3	It solves the problem in design Level	It solves the problem in implementation Level
4	Ex. A Calculator is powered by a battery source but how the battery modules is not necessary to know for the user	Ex. A Calculator solves a complex problem and displays the result but hides the implementation (calculating the result) involved

Inner Classes in Java Programming

If one class is existing within another class is known as inner class or nested class

Syntax

```
class Outerclass_name
{
        ....................
        ....................

        class Innerclass_name1
        {
            ....................
            ....................
        }
        class Innerclass_name1
        {
            ....................
            ....................
        }
        ....................
}
```

The main purpose of using inner class

To provide more security by making those inner class properties specific to only outer class but not for external classes. To make more than one property of classes private properties.

private is a keyword in java language. it is preceded by any variable that property can be access only within the class but not outside of it (provides more security).

If more than one property of class wants to make as private properties than all can capped under private inner class.

Syntax

```
class Outerclass_name
{
        private class Innerclass_name
        {
                ........................ //private properties
        }
}
```

Note: No outer class made as private class otherwise this is not available for JVM at the time of execution.

Rules to access properties of inner classes

Inner class properties can be accessed in the outer class with the object reference but not directly. Outer class properties can be access directly within the inner class. Inner class properties can't be accessed directly or by creating directly object.

Note: In special situation inner class property can be accessed in the external class by creating special objects with the reference of its outer class.

Example

```
class  A //outer class
{
        void  fun1()
        {
                System.out.println("Hello fun1()"); /* inner class properties should be access using
                object reference in outer class. */

                B  ob=new  B();

                ob.x=10

                System.out.println("x= "+ob.x);
                ob.fun2();
        }

        void fun3()  // outer class fun3()
        {
                System.out.println("Hello fun3()");
        }

        class  B  // inner class
        {
                int  x;  // inner class variable

                void fun2()     //inner class fun2()
                {
                        System.out.println("Hello fun2()");

                        fun3(); //outer class properties can be access directly
                }
        }
}
```

```
class  C // external class
{
        void fun3()
        {
                System.out.println("Hello fun3()");
        }
}


class IncDemo
{
        public static void main(String args[])
        {
                A  oa=new  A();
                oa.fun1();
                C   oc=new   C();
                oc.fun3();
        }
}
```

Output

Hello fun1()
X=10
Hello fun2()
Hello fun3()

Accessing inner class properties in the external class

1. If inner class in non static the object can be created with the following syntax

Syntax

```
class  Outer_class
{
        class  Inner_class
        {
                .. ........................
                ..........................
        }
                ..........................
                ..........................
}
```

```
class External_class
{
        Outer_class.Inner_Class objectrefernce=new Outer_Class.External_Class();
}
```

2. If inner class is static then object reference can be created with the following syntax

Syntax

```
class Outer_class
{
        static class Inner_Class
        {
                ..........................
                ..........................
        }
}
class External_Class
{
                Outer_class.Inner_Class objectrefernce=new Outer_Class.External_Class();
}
```

Example

```
class A//Outer class
{
        class B // non-static inner class
        {
                int x;   //inner class variable

                void fun1()     //inner class fun1()
                {
                System.out.println("Hello fun1()");
                }
        }
        static class C   //static inner class
        {
                int  y=20;  // inner class variable
                void fun2()
                {
                        System.out.println("Hello fun2()");
                }
        }
}
```

```
class IncDemo
{
        public static void main(String args[])
        {
                A.B  ob=new  A().new.B();
                System.out.println(ob.x);
                ob.fun1();
                A.C  oc=new  A.C();
                System.out.println(oc.y);
                oc.fun2();
        }
}
```

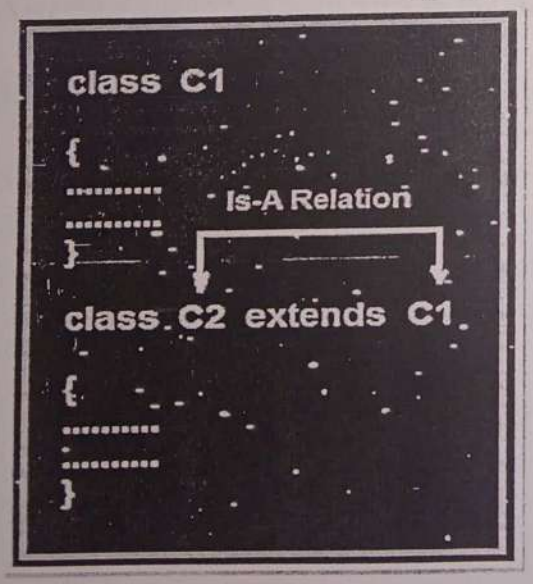
Relationship in Java

Type of relationship always makes to understand how to reuse the feature from one class to another class. In java programming we have three types of relationship they are.

- **Is-A Relationship**
- **Has-A Relationship**
- **Uses-A Relationship**

Is-A relationship

In Is-A relationship one class is obtaining the features of another class by using inheritance concept with extends keywords. In a IS-A relationship there exists logical memory space.



Example of Is-A Relation

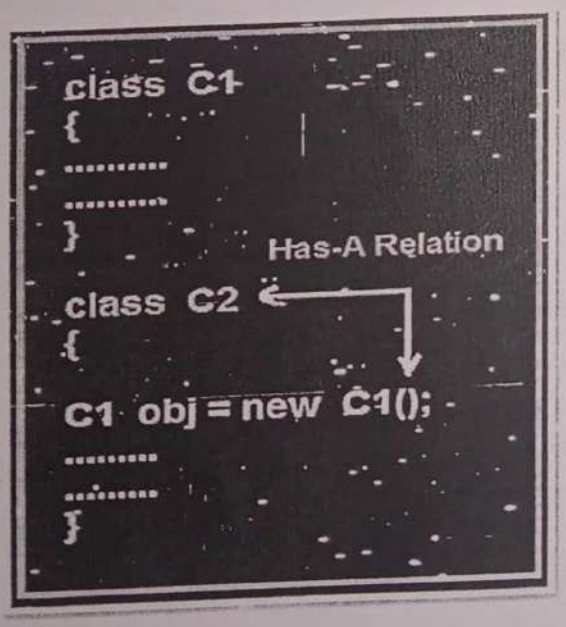
```
class Faculty
{
        float salary=30000;
}
class Science extends Faculty
{
        float bonous=2000;
        public static void main(String args[])
        {
                Science obj=new Science();
                System.out.println("Salary is:"+obj.salary);
                System.out.println("Bonous is:"+obj.bonous);
        }
}
```

Output

Salary is: 30000.0
Bonous is: 2000.0

Has-A relationship

In Has-A relationship an object of one class is created as data member in another class the relationship between these two classes is Has-A. In Has-A relationship there existed physical memory space and it is also known as part of or kind of relationship.



Example of Has-A Relation

```
class Employee
{
        float salary=30000;
}
```

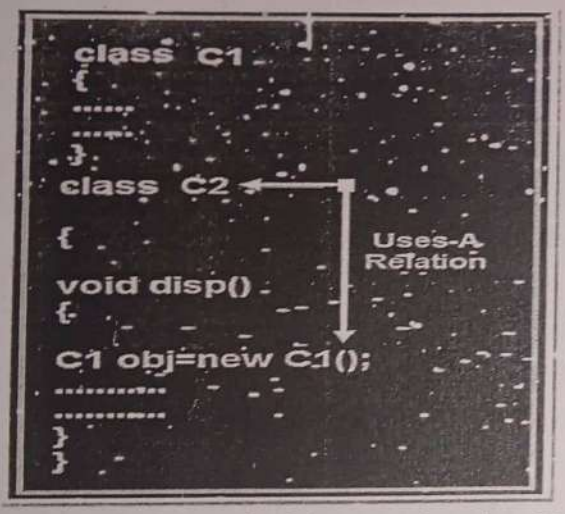
```
class Developer extends Employee
{
        float bonous=2000;
        public static void main(String args[])
        {
                Employee obj=new Employee();
                System.out.println("Salary is:"+obj.salary);
        }
}
```

Output

Salary is: 30000.0

Uses-A relationship

A method of one class is using an object of another class the relationship between these two classes is known as Uses-A relationship.



As long as the method is execution the object space (o1) exists and once the method execution is completed automatically object memory space will be destroyed.

Example of Uses-A Relation

```
class Employee
{
        float salary=30000;
}
class Salary extends Employee
{
        void disp()
        {
                float bonous=1000;
                Employee obj=new Employee();
                float Total=obj.salary+bonous;
                System.out.println("Total Salary is:"+Total);
        }
}
```

```
class Developer
{
        public static void main(String args[])
        {
                Salary s=new Salary();
                s.disp();
        }
}
```

Output: Total Salary is: 31000.0

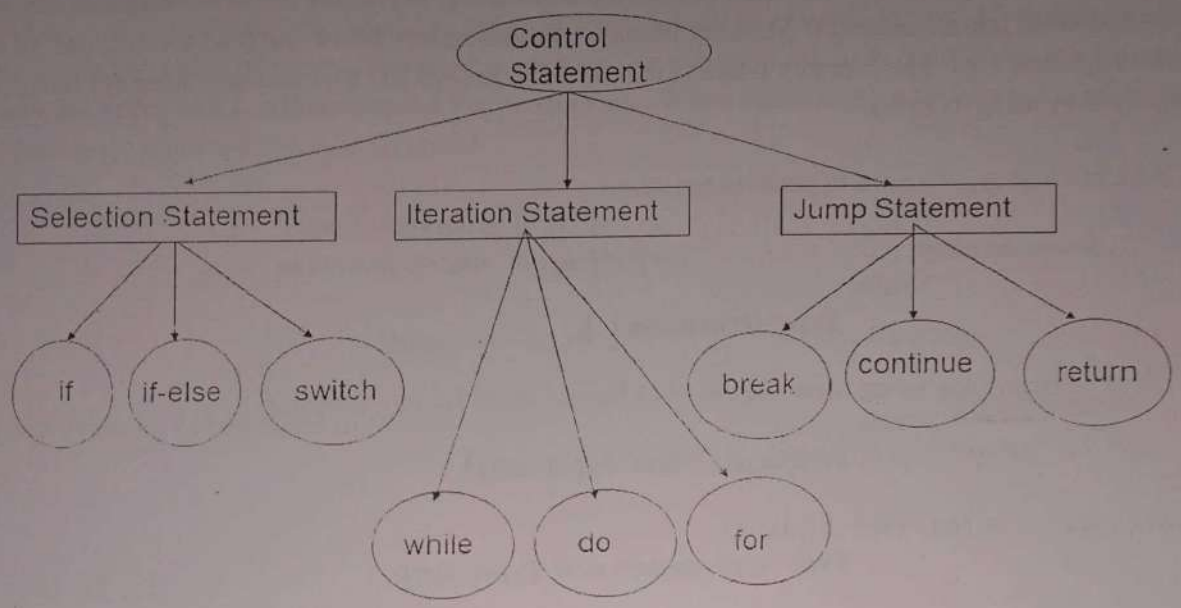
Note 1: The default relationship in java is Is-A because for each and every class in java there exist an implicit predefined super class is java.lang.Object.

Note 2: The universal example for Has-A relationship is System.out (in System.out statement, out is an object of printStream class created as static data member in another system class and printStream class is known as Has-A relationship).

Note 3: Every execution logic method (main()) of execution logic is making use of an object of business logic class and business logic class is known as Uses-A relationship.

Bhush

3. JAVA CONTROL STATEMENTS



For Details notes refer class Notes.

4. 1-D ARRAY IN JAVA

An **array** is an object that can be used to store a list of values. It is made out of a contiguous block of memory that is divided into a number of "Memory Blocks." Each Memory Block can hold a value, and all of the values are of the same type. The Memory Blocks are indexed 0 through SIZE -1 and each Memory Block can be accessed by using its *index*.

To create an array, you need to perform two steps :

1. declare an array:

   Syntax:

   Type arrayname [ ];

2. Create an array using '**new**' keyword (Allocate space for its elements):

   Syntax:

   arrayname= new Type[Size];

Above steps can be combined as follows

Type arrayname= new Type[Size];

Where Type may be any data type or reference Type (class name).

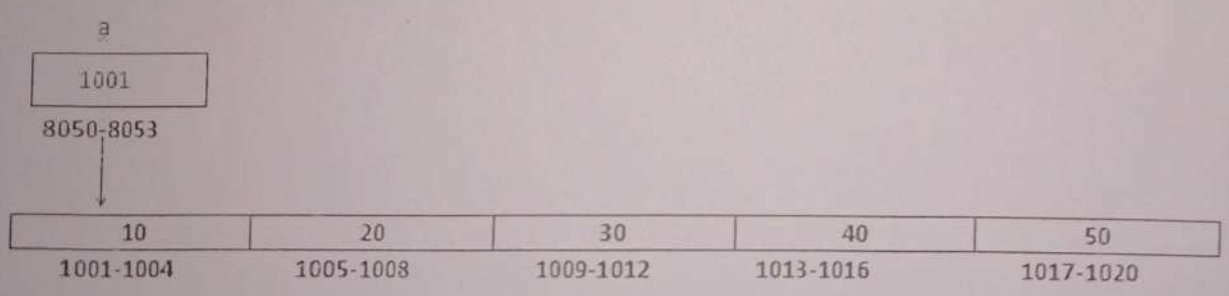
Ex: Assume n=5

```
int a[ ];
a=new int[n];
```

Or

```
int a[ ]=new int[n];
```

Memory Map:



Initialization of 1-D array:

Type arrayname[ ]={ value1,value2,value3,.....value n };

Ex: int a[ ]={10,20,30,40,50};

4. 2-D ARRAY

In addition to one dimensional we can create arrays of two or more dimensions. In Java, Multidimensional array are implemented as arrays of arrays. You need to perform two steps to work with multidimensional arrays: **1. Declare the array** and **2. Allocate space for its elements.** When constructing a two-dimensional array, you specify how many rows and columns you need:

Type arrayname [ ][ ];
arrayname= new Type [M][N];

Where M = no of rows N= no of columns.

Above two steps can be combined as follows

Type arrayname [ ][ ]=new Type[M][N];

Ex:

```
int a[ ][ ];
a=new int [M ][N ];
```

OR `int a[ ][ ]=new int[ M ][N];`

```
float a[][]=new float[2][2];
```

Here a is two dimensional array having 2 rows and 2 columns. i.e. size is 4, we can store 4 elements in that array.

Q.1 WAP for addition and multiplication of n numbers

```
import java.util.*;
class ArrayEx1
{
  public static void main(String[] args)
  {
    Scanner s=new Scanner(System.in);
    int a[],n,sum=0,mul=1,avg;
    System.out.println("Enter the size of array");
    n=s.nextInt();
    a=new int[n];
    System.out.println("Enter the Numbers");
    for(int i=0;i<n;i++)
    {
        a[i]=s.nextInt();
    }

    for(int i=0;i<n;i++)
    {   sum=sum+a[i];
        mul=mul*a[i];
    }
    avg=sum/n;
    System.out.println("Addition= "+sum);
    System.out.println("Multiplication= "+mul);
    System.out.println("Average= "+avg);
  }
}
```

Q.2 WAP to sort n numbers in ascending order. Also find minimum and maximum number.

```
class ArrayEx2
{
 public static void main(String[] args)
 {
        Scanner s=new Scanner(System.in);
        int a[ ],n,sum=0,mul=1;
        System.out.println("Enter the size of array");
        n=s.nextInt();
        a=new int[n];
        System.out.println("Enter the Numbers");
        for(int i=0;i<n;i++)
        {
                a[i]=s.nextInt();
        }
        for(int i=0;i<n-1;i++)
        {
                for(int j=0;j<n-1;j++)
                {
                        if(a[j]>a[j+1])
                        {       int temp=a[j];
                                a[j]=a[j+1];
                                a[j+1]=temp;
                        }
                }
        }

        System.out.println("Sorted numbers are");
        for (int i=0;i<n;i++)
        {       System.out.println(a[i]);
        }
        System.out.println("Largest Number="+a[n-1]);
        System.out.println("Smalletst Number="+a[0]);
 }
}
```

Q.3 WAP to calculate mean , standard deviation and Variance .

```
import java.util.*;
class ArrayEx3
{
 int x[],n;
 double sum1,sum2,mean,sd,var;

 void getdata()
 {
        Scanner s=new Scanner(System.in);
        System.out.println("Enter the size for array");
        n=s.nextInt();
        x=new int[n];
        System.out.println("Enter the elements");
        for(int i=0;i<n;i++)
        {       x[i]=s.nextInt();
        }
 }

 void cal()
 {
        for (int i=0;i<n;i++)
        {  sum1=sum1+x[i];
        }
        mean=sum1/n;
        for (int i=0;i<n;i++)
        {       sum2=sum2+((x[i]-mean)*(x[i]-mean));
        }
        var=sum2/n;
        sd=Math.sqrt(var);
 }
```

```
void putdata()
{
        System.out.println("Mean="+mean);
        System.out.println("Standard Devation="+sd);
        System.out.println("Variance="+var);
}
```

```
public static void main(String[] args)
{
        ArrayEx3 obj=new ArrayEx3();
        obj.getdata();
        obj.cal();
        obj.putdata();
}
}
```

Q.4) WAP in java for fitting a straight line through a set of points (X_i, Y_i), i== 1 to n. the straight line equation is y=mx + c and the values of m & c are given by

$$m = \frac{n\sum (X_i Y_i) - (\sum X_i)(\sum Y_i)}{n(\sum X_i^2) - (\sum Y_i)^2}$$

$$C = 1/n\,(\sum Y_i - m \sum X_i)$$

```
import java.util.*;
class ArrayEx4
{
  int x[],y[].n,sum1,sum2,sum3,sum4,mul=1;
  double m,c;
  void getdata()
  {
   Scanner s=new Scanner(System.in);
   System.out.println("Enter the size for array x and y");
   n=s.nextInt();
   x=new int[n];
   y=new int[n];
   System.out.println("Enter the set of x coordinates");
   for(int i=0;i<n;i++)
   {
        x[i]=s.nextInt();
   }

   System.out.println("Enter the set of y coordinates");
   for(int i=0;i<n;i++)
   {
        y[i]=s.nextInt();
   }
  }

  void cal()
  {
        for (int i=0;i<n;i++)
        {       sum1=sum1+x[i]*y[i];
                sum2=sum2+x[i];
                sum3=sum3+y[i];
                sum4=sum4+x[i]*x[i];
        }
   m= (1.0*n*sum1-sum2*sum3)/(n*sum4- sum3*sum3);
   c=(sum3-m*sum2)/n;
  }
```

```
  void putdata()
  {
        System.out.println("M="+m);
        System.out.println("N="+n);

  }
  public static void main(String[] args)
  {
        ArrayEx4 obj=new ArrayEx4();
        obj.getdata();
        obj.cal();
        obj.putdata();
  }
}
```

5. WRAPPER CLASSES

When we give the gifts, we wrap them nicely in the box. The glossy paper around a box makes in attract we have worked with simple data types. Now java wants to wrap them and make them more attract (useful). For this java uses interesting concept of wrapper class.

Vector **cannot** handle primitive (basic) data types like int, float, long, char and double. Vector can accept the data in the form of objects. Therefore there is need to convert basic data types into equivalent objects. So this conversion is done by using the wrapper classes which are present in java.lang package.

Basic data type	Wrapper Class
1) int	Integer
2) boolean	Boolean
3) char	Character
4) float	Float
5) long	Long
6) double	Double

As all these numbers are stored as objects, they take all the advantage of Object oriented technology. They can be passed to methods as parameters.

METHODS UNDER WRAPPER CLASSES

1) Converting basic data type to Numeric Object using Constructor Method)

```
int i=10;
float f=8.9f;
double d=897678.9878;
long l=989456767;

Integer intobj = new Integer(i); // convert integer to Integer Object
Double  doubleobj= new Double(d); // convert double  to Double Object
Float floatobj= new Float(f); // convert float to Float Object
Long longobj= new Long(l); //converts long to Long object
```

2) Converting Numeric Object to basic data type using typeValue( ) method.

```
int i = intobj . intValue( ); // converts Integer object to basic  integer
float f  =floatobj . floatValue( ); // converts Float object to basic float
long l = longobj . longValue( ); // converts Long object to basic long
double d = doubleobj . doubleValue( ); // converts Double object to basic double
```

3) **Converting basic data type to the string using static method toString( ).**

```
String str1 = Integer.toString(i);        // primitive integer to string
String str2 = Float.toString(f);          // primitive float to string
String str3 = Double.toString(d);// primitive double  to string
String str4= Long.toString(l); // primitive long to string
```

4) **Converting String Objects to Numeric Objects using the static method ValueOf( )**

```
String s1="100";
String s2="10.54 ";
String s3=" 12345";
String s4="312223.44222 "

Integer obj1  = Integer . ValueOf(s1); // converts string to Integer Object
Float obj2 = Float.valueOf(s2); // converts String to Float Object
Long obj3 = Long.valueOf(s3); // converts String to Long Object
Double obj4= Double.valueof(s4)// converts String to Double Object.
```

5) **Conversion of Numeric Strings to basic data type using passing methods**

```
int i = Integer.parseInt(str1);
float f = Float.parseFloat(str2);
double d= Double.parseDouble(str3);
long l=Long.parseLong(str4);
```

Example:

```
class WrapperDemo {
    public static void main(String[] args) {
            int i=10;
            float f=10.85f;
            double d=0.25632;
            long ph=98756;
            Integer IntVal=new Integer(i);
            Float FloatVal = new Float(f);
            Double DoubleVal=new Double(d);
            Long LongVal=new Long(ph);
            System.out.println(IntVal);
            System.out.println(FloatVal);
            System.out.println(DoubleVal);
            System.out.println(LongVal);
    }
}
```

6. STRINGS

Strings in java are considered as String Objects. Java has two types string. i.e. Mutable and Immutable String.

Mutable String: It can be modified in terms of length and content. Ex. StringBuffer class

Immutable String: It cannot be modified in terms of length and content. Ex. String class

String class is called as Immutable because any operation on string results into new String object by implicit execution of new **String( )** .

JVM manages separate memory (specialized heap memory) called as string constant pool for efficient memory utilization. In this part of memory, only string objects are stored. JVM monitors string constant pool to check whether new string object to be created is already present on the string constant pool. If it is present, then new String object will not be created only references are manipulated.

Strings are very important part of the programming language as they are used to process long textual/symbolic information. For example, when we visit a bank to open an account then we are asked for information such as name, address. The information we provide is in the form of ordered sequence of characters and symbols. The ordered sequence of characters and symbols is known as String.

Declaring and Constructing a String:

Strings represent a sequence of characters a jâva string is an instantiated object of the string class. Strings may be declared and created as follows:

```
String stringName;
stringName = new String("Java");
Or
String stringName = new String("Java");

Or
String stringName="java";
```

Note: stringName is actually a reference to the String object.

Methods of Strings:

The String class defines a number of methods that allows us to accomplish a variety of string manipulation tasks. The below table lists some of the most commonly used string methods and their tasks.

Method Call	Task performed
s2 = s1.toLowerCase()	Converts the string s1 to all lower case
s2 = s1.toUpperCase()	Converts the string s1 to all upper case
s2 = s1.replace('x','y');	Replaces all appearances of 'x' with 'y'
s2 = s1.trim()	Removes white spaces at the beginning & end of the s1.
s1.equals(s2)	Returns true if s1 is equal to s2
s1.equalsIgnoreCase(s2)	Returns true if s1 is equal to s2 ignoring the case of characters
s.length()	Gives the length of s.
s1.charAt(n)	Gives the nth character of s1.
s1.compareTo(s2)	Returns negative if s1<s2, positive if s1>s2 & zero if s1 = s2.
s1.concat(s2)	Concatenates s1 and s2.
s1.substring(n)	Gives substring starting from the nth character
s1.substring(n,m)	Gives substring starting from the nth character to m^{th} character
s1.indexOf('x')	Give the position of the first occurrence of 'x' in the string s1.
S1.lastindexOf('x')	Give the position of the last occurrence of 'x' in the string s1

Example.

```
class StringEx
{
        public static void main(String args[]) {
                String s1 = "India";
                System.out.println("s1 = " +s1);
                int len=s1.length();
                System.out.println("Length = "+len);
                String s2=s1.toLowerCase();
                System.out.println("Lower Case of s1 = " +s2);
                s2=s1.toUpperCase();
                System.out.println("Upper Case of s1 = " +s2);
                String s3=s1.replace('i','b');

                System.out.println("Upper Case of s1 = " +s3);
                System.out.println("s1 equals s2  = " +s1.equals(s2));
                System.out.println("s1 equals s2  = " +s1.equalsIgnoreCase(s2));

                System.out.println("s1 compares s2  = " +s1.compareTo(s2));
                System.out.println("s2 compares s1  = " +s2.compareTo(s1));
                System.out.println("Concatenate s1 with string = " +s1.concat(" is Great"));
                System.out.println("Substring of s1  = " +s1.substring(3));
                System.out.println("Substring of s1  = " +s1.substring(2,5));
                System.out.println("Index of i in s1  = " +s1.indexOf('i'));
        }
}
```

Output:

```
s1 = India
Length = 5
Lower Case of s1 = india
Upper Case of s1 = INDIA
Upper Case of s1 = Indba
s1 equals s2  = false
s1 equals s2  = true
s1 compares s2  = 32
s2 compares s1  = -32
Concatenate s1 with string = India is Great
Substring of s1  = ia
Substring of s1  = dia
Index of i in s1  = 3
```

Q.1 WAP to convert string into Uppercase characters

```
import java.util.*;
class  Convert
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter a string");
        String s=sc.nextLine();
        System.out.println("string in Uppercase= "+s.toUpperCase());

    }
}
```

Output:

```
Enter a string
hello
string in Uppercase= HELLO
```

Q.2 WAP to test whether string is palindrome or not.

```
import java.util.*;
class  TestPalindrome
{
        Scanner sc=new Scanner(System.in);
        String s,str="";
        char ch;
        void getdata()
        {
                System.out.println("Enter a string");
                s=sc.nextLine();
        }

        void test()
        {
                for (int i=s.length()-1;i>=0;i--)
                {
                        ch=s.charAt(i);
                        str=str+ch;
                }

                if (s.equals(str))
                {       System.out.println("String is Palindrome");
                }
                else
                {       System.out.println("String is not a Palindrome");
                }
        }

        public static void main(String[] args)
        {
                        TestPalindrome obj=new TestPalindrome();
                        obj.getdata();
                        obj.test();
        }
}
```

Output:

```
Enter a string
nitin
String is Palindrome
```

Q.3 WAP to count alphabets, digits, special symbols, blank spaces and words from a given st

```
import java.util.*;
class StringCount
{
        int alpha,digits,sp,bs,words;
        Scanner sc=new Scanner(System.in);
        String s;
        char ch;

        void getdata()
        {
                System.out.println("Enter a string");  s=sc.nextLine();
        }

        void count()
        {
                for (int i=0;i<s.length();i++)
                {
                        ch=s.charAt(i);

                        if ((ch>='A'&& ch<='Z')||(ch>='a' && ch<='z'))
                        {
                                alpha++;
                        }

                        if (ch>='0'&&ch<='9')
                        {
                                digits++;
                        }

                        if (ch==' ')
                        {
                                bs++;
                        }
                }
                words=bs+1;
                sp=(s.length())-alpha-digits;
        }
        void putdata()
        {
                System.out.println("No. of alphabets="+alpha);
                System.out.println("No. of digits="+digits);
                System.out.println("No. of blank spaces ="+bs);
                System.out.println("No. of words="+words);
                System.out.println("No. of  special symbol="+sp);
        }
```

```
    public static void main(String[] args)
    {
                StringCount obj=new StringCount();
                obj.getdata();
                obj.count();
                obj.putdata();
    }
}
```

Output:

```
Enter a string
life is beautiful
No. of alphabets=15
No. of digits=0
No. of blank spaces =2
No. of words=3
No. of special symbol=2
```

4. WAP to sort the names of student in alphabetical order.

```
import java.io.*;

class Sort
{
  public static void main(String args[]) throws IOException
  {
        String name[ ],temp;

        int n;

        DataInputStream d=new DataInputStream(System.in);

        System.out.println("How many students names to be sorted");
        n=Integer.parseInt(d.readLine());

        name=new String[n];
        System.out.println("Enter the names");

        for (int i=0;i<n ;i++)
        {
              name[i]=d.readLine();
        }
```

```
        for(int i=0;i<n-1;i++)
        {
                for(int j=0;j<n-1;j++)
                {
                        int diff=name[j].compareTo(name[j+1]);
                        if(diff>0)
                        {       temp=name[j];
                                name[j]=name[j+1];
                                name[j+1]=temp;

                        }
                }
        }

        System.out.println("Names after sorting :");
        for(int i=0;i<n;i++)
        System.out.print(name[i]+" ");
    }
}
```

Output:

```
How many students names to be sorted
5
Enter the names
dhiraj
amit
bharat
chetan
amey

Names after sorting :
amey   amit   bharat   chetan   dhiraj
```

String Buffer class :

The **StringBuffer** is a class which is alternative to the **String** class. But **StringBuffer** class is more flexible to use than the **String** class. StringBuffer creates strings of flexible length that can be modified in terms of both length and content. but in case of **String** class once the string is defined then it remains fixed. i.e. String creates strings of fixed length. we can insert characters and substrings in the middle of a string. or append another string to the end. The StringBuffer and the StringBuilder are almost one and the same. The StringBuilder or StringBuffer have three constructors and 30 methods.

Following are some simple methods used for StringBuffer

Name of method	Description
append(String str)	Appends the string to the buffer.
capacity ( )	It returns the capacity of the string buffer.
charAt(int index)	It returns a specific character from the sequence which is specified by the index.
delete(int start,int end)	It deletes the characters from the string specified by the starting and ending index.
insert(int offset,char ch)	It inserts the character at the position specifiedby the offset.
length()	It returns the length of the string buffer.
setCharAt(int index,char ch)	The character specified by the index from the stringbufferis set to ch.
setLength(int new_len)	It sets the length of the string buffer.
toString( )	It converts the string representing data in this string buffer.
replace(int start,int end , String str)	It replaces the characters specified by the new string
reverse()	The character sequence is reversed.

These methods can be illustrated with the help of following Java programs-

Example- Write a Java program that uses the length( ) and capacity ( ) methods of StringBuffer class.

Solution:

```
public class StringBuffDemo1{
 public static void main(String args[])
 {
 StringBuffer str=new StringBuffer("Java");
 System.out.println("The String Buffer is "+str);
 System.out.println("The length is "+str.length());
 System.out.println("The capacity is "+str.capacity());
 }
```

Output
The String Buffer is Java
The length is 4
The capacity is 20

Program Explanation: Using the StringBuffer/StringBuilder class, we can create a buffer of characters. The **length()** function returns the number of characters in the string and the **capacity** function returns the number of characters in the string +16 additional characters.

Java **StringBuilder** class - Java StringBuilder class is used to create mutable (modifiable) string. The StringBuilder class is same as StringBuffer class except that it is non-synchronized. It is available since 1.5.

Constructors of StringBuilder class

1.StringBuilder(): creates an empty string Builder with the initial capacity of 16.

2.StringBuilder(String str): creates a string Builder with the specified string.

3.StringBuilder(int length): creates an empty string Builder with the specified capacity as length.

Important methods of StringBuilder class

1. public StringBuilder append(String s) - It is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.

2. public StringBuilder insert(int offset, String s) – It is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.

3. public StringBuilder replace(int startIndex, int endIndex, String str) – It is used to replace the string from specified startIndex and endIndex.

4. public StringBuilder delete(int startIndex, int endIndex) - It is used to delete the string from specified startIndex and endIndex.

5. public StringBuilder reverse() – It is used to reverse the string.

6. public int capacity() – It is used to return the current capacity.

7. public void ensureCapacity(int minimumCapacity) – It is used to ensure the capacity at least equal to the given minimum.

8. public char charAt(int index) – It is used to return the character at the specified position.

9. public int length() – It is used to return the length of the string i.e. total number of characters.

10. public String substring(int beginIndex) – It is used to return the substring from the specified beginIndex.

11. public String substring(int beginIndex, int endIndex) – It is used to return the substring from the specified beginIndex and endIndex.

Java StringBuilder Examples

1) StringBuilder append() method- It concatenates the given argument with this string.

```
class A{
        public static void main(String args[]){
        StringBuilder sb=new StringBuilder("Hello ");
        sb.append("Java");//now original string is changed
        System.out.println(sb);//prints Hello Java
    }
}
```

2) StringBuilder insert() method - it inserts the given string with this string at the given position.

```
class A{
        public static void main(String args[]){
        StringBuilder sb=new StringBuilder("Hello ");
        sb.insert(1,"Java");//now original string is changed
        System.out.println(sb);//prints HJavaello
    }
}
```

3) StringBuilder replace() method – It replaces given string from the specified beginIndex and endIndex.

```
class A{
        public static void main(String args[]){
        StringBuilder sb=new StringBuilder("Hello");
        sb.replace(1,3,"Java");
        System.out.println(sb);//prints HJavalo
    }
}
```

4) StringBuilder delete() method - It deletes the string from the specified beginIndex to endIndex.

```
class A{
        public static void main(String args[]){
        StringBuilder sb=new StringBuilder("Hello");
        sb.delete(1,3);
        System.out.println(sb);//prints Hlo
        }
}
```

5) StringBuilder reverse() method – It reverses the current string.

```
class A{
        public static void main(String args[]){
        StringBuilder sb=new StringBuilder("Hello");
        sb.reverse();
        System.out.println(sb);//prints olleH
        }
}
```

6) StringBuilder capacity() method -The capacity() method of StringBuilder class returns the current capacity of the Builder. Default capacity of the Builder is 16. If the number of character increases from its current capacity, it increases the capacity by (oldcapacity*2)+2. Ex if your current capacity is 16, it will be (16*2)+2=34.

```
class A{
    public static void main(String args[]){
    StringBuilder sb=new StringBuilder();
    System.out.println(sb.capacity());//default 16
    sb.append("Hello");
    System.out.println(sb.capacity());//now 16
    sb.append("java is my favourite language");
    System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2
  }
}
```

7) StringBuilder ensureCapacity() method - It ensures that the given capacity is the minimum to the current capacity. If it is greater than the current capacity, it increases the capacity by (oldcapacity*2)+2. For example if your current capacity is 16, it will be (16*2)+2=34.

```
class A{
public static void main(String args[]){
    StringBuilder sb=new StringBuilder();
    System.out.println(sb.capacity());//default 16
    sb.append("Hello");
    System.out.println(sb.capacity());//now 16
```

```
        sb.append("java is my favourite language");
        System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2
        sb.ensureCapacity(10);//now no change
        System.out.println(sb.capacity());//now 34
        sb.ensureCapacity(50);//now (34*2)+2
        System.out.println(sb.capacity());//now 70
    }
}
```

Difference between String and StringBuffer?

	String	StringBuffer
1	The data which enclosed within double quote (" ") is by default treated as String class.	The data which enclosed within double quote (" ") is not by default treated as StringBuffer class
2	String class object is immutable	StringBuffer class object is mutable
3	When we create an object of String class by default no additional character memory space is created.	When we create an object of StringBuffer class by default we get 16 additional character memory space.

Difference between StringBuffer and StringBuilder?

All the things between StringBuffer and StringBuilder are same only difference is StringBuffer is synchronized and StringBuilder is not synchronized. synchronized means one thread is allow at a time so it thread safe. Not synchronized means multiple threads are allow at a time so it not thread safe.

	StringBuffer	StringBuilder
1	It is thread safe.	It is not thread safe.
2	Its methods are synchronized and provide thread safety.	Its methods are not synchronized and unable to provide thread safety.
3	Relatively performance is low because thread need to wait until previous process is complete.	Relatively performance is high because no need to wait any thread it allows multiple thread at a time.
4	Introduced in 1.0 version.	Introduced in 1.5 version.

Q.15 Develop a Java program that determines the number of days in a given semester. Input of program consists of the year, month and day information of the first and last days of a semester. program should accept the date information as a single string instead of accepting the year, month, day information separately. The input string must be in MM/DD/YYYY.

```
import  java.util.*;

class CalculateDays {
   public static void main(String args[]) {
        Scanner  sc=new Scanner(System.in);

        System.out.println("Enter the starting date in MM/DD/YYYY:");          String start=sc.next();

        System.out.println("Enter the last date in MM/DD/YYYY:");              String last=sc.next();

        int MM1=Integer.parseInt(start.substring(0,2));

        int DD1 =Integer.parseInt( start.substring(3,5));

        int YY1 =Integer.parseInt( start.substring(6,10));

        int MM2=Integer.parseInt(last.substring(0,2));

        int DD2 =Integer.parseInt(last.substring(3,5));

        int YY2 =Integer.parseInt(last.substring(6,10));

        Calendar C1 =Calendar.getInstance();

        Calendar C2 =Calendar.getInstance();

        C1.set(YY1,MM1,DD1);

        C2.set(YY2,MM2,DD2);

        long n=(C2.getTimeInMillis ( ) - C1.getTimeInMillis ( ) ) / (24*60*60*1000);

        System.out.println ("Number of days in Semester=" +n);
   }
}
```

Output

```
Enter the starting date in MM/DD/YYYY:
01/01/2016
Enter the last date in MM/DD/YYYY:
06/01/2016
Number of days in Semester=151
```

Type of package

Package are classified into two type which are given below.

1. **Predefined or built-in package**
2. **User defined package**

Predefined or built-in package

These are the package which are already designed by the Sun Microsystem and supply as a part of java API, every predefined package is collection of predefined classes, interfaces and sub-package.

User defined package

If any package is design by the user is known as user defined package. User defined package are those which are developed by java programmer and supply as a part of their project to deal with common requirement

Access Protection in Packages

	Private	Default	Protected	Public
Same Class	Yes	Yes	Yes	Yes
Same package subclass	No	Yes	Yes	Yes
Same package Non-subclass	No	Yes	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package Non-subclass	No	No	No	Yes

How to Create Package

1) Create a directory which has same name as the package.
2) Include package command along with the package name as the first statement in the program.

   package packagename;

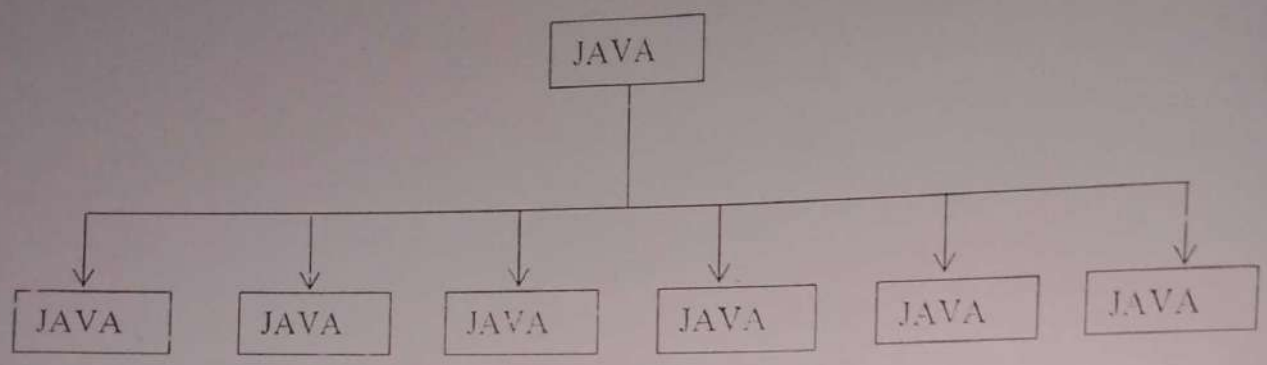
   Where package is a keyword and packagename is userdefined.
3) Write class definition.
4) Save the file in this directory as NameOfClass.java where where .NameOfClass is the name of the class.
5) Compile this file using javac.

The CLASSPATH variable points to the directories in which the classes that are available to import. CLASSPATH variable allows you to put your own class files in various directories and lets the JDK tools know where they are.

Example:

Built in Packages

- **Java API** is a collection of various packages in which several classes and methods grouped together according to their functionality.
- Following fig represents the most commonly used Java API packages-



The purpose of java system packages is as given below-

Package	Description
java.applet	This package is for creating and implementing applet.
java.awt	This package is for graphical user interface .Using the classes and methods of this package various components like button, text box, radio buttons and so on can be used.
java.lang	This package supports the use of strings, math functions, threads and exception.
java.util	Various utility classes such as vectors, date, and random numbers are used by using this package.
java.io	For supporting input and output operations this package is used in Java program.

java.lang Package

Purpose: This package provides core classes and interfaces used in design of Java language.

Interfaces:

Name	Description	Some methods
CharSequence	This interface is for getting sequence of characters. The read-only access to character sequence is provided by this method	int charAt(int i) – It returns the position of the character at specified index. *String* toString( ) – This method returns the sequence of characters. Int length() – it returns length of string.
Runnable	A class whose instance can be executed by a thread implements a *Runnable* interface.	

Classes:

Name	Description
Boolean	It's a wrapper class for Boolean values
Byte	It's a wrapper class for byte data.
Character	It's a wrapper class character values.
Class	For representing classes and interfaces in the java application this class is used.
ClassLoader	For handling loading of the classes this object is used.
Complier	This class is for providing support complier related activities.
Double	It's a wrapper class for Double values.
Float	It's a wrapper class for Float values.
Integer	It's a wrapper class for Integer values
Long	It's a wrapper class for Long values
Math	This class provides the support for the operations such as square root, exponentiation, and logarithm.
Object	The class Object is at the top of class hierarchy.
Runtime	For linking the Java application program with its runtime environment the instance of class *Runtime* is used.
StringBuffer	For handling the dynamic sequence of characters StringBuffer class is used.
String	It's a wrapper class for String values.
System	This is the most commonly used class in which many important fields and methods are defined. For example err, in and out are the most commonly used fields in this class.
Thread	A thread class creates a thread for a separate execution of the program.

10. VECTOR CLASS

1. Vector class is present under the package 'util' (java.util.vector).
2. Vector class is same as ArrayList but vector methods are synchronized for safety purpose.
3. Vector is the only class other than ArrayList which provides random access to the data elements.
4. Vector class is used to create the vector array which is dynamic in nature & stores homogeneous as well as heterogeneous type of data.
5. There is no upper limit to store number of data values into vector array i.e. vector array is growable.
6. Vector cannot store basic data type directly. It only requires the data in the form of object numbers.
7. Vector array can be declared without specifying the size. A vector array without size can store unknown no. of data elements that may vary in size.

Difference between Array and Vector

Arrays	Vectors
Array contains all the elements of similar data type.	Vector contains the elements that may be of various data type.
The size of array is fixed.	The size of array is varying. We can change the size of the vector whenever is required.
We can store the simple elements directly in the arrays.	We can not store the simple object in the database.
We can directly add the simple data type element in the array.	We can not directly add the simple data type element in the vector we need to add an object.

Advantages of Vector

1. Vector provides a way to store heterogeneous and homogeneous type of data in the form of object.
2. Vector can be used to store the data elements that may vary in size.
3. Vector class provides specialized methods for insertion, deletion, copy & other Manipulations.

Drawbacks of Vector

1. Vector cannot store basic data type directly.
2. Vector can only store the object type of data.
3. To store the basic data type in a vector, we require the conversion from basic data types to object numbers using wrapper classes.

Constructor of Vector class

1. **Default Constructor** - To create a vector array width default size = 10, we must execute the default Constructor.

   Ex. Vector V = new Vector ();

2. **Parameterized Constructor width size as n**: In the following constructor, vector array is created with initial size n and once the size of vector array is full, the size will be incremented in multiples of n.

   Ex. Vector V = new Vector (n);

3. **Parameterized Constructor (two parameter)** :

   Ex. Vector V = new Vector (n, m);

   Where n = initial size of vector array, m = increment of array as the capacity becomes full

Non-static Methods of vector class: Create a vector array width default size.

Ex. Vector V = new Vector ();

1. v.addElement (data) : It adds the data element at the end of vector.
2. v.insertElementAt (data, n) :The above method inserts the data element at n^{th} position in a vector.
3. v.removeElementAt (n) :It remove data element from n^{th} position of vector array
4. v1.copyInto (v2) : It copies vector array v1 into vector array v2
5. v.size () :It returns the size of vector array.
6. v.removeElementAll () : The method makes the vector array empty.
7. v.removeElement (data) : It removes 1st occurrence of data element in the vector array.
8. v.firstElement () : It gives first element of vector array.
9. v.lastElement () : It gives last element of vector array.
10. v.elementAt (n) : The above method returns data element from n^{th} position.
11. v.contains(data) : It returns true if vector contains the data otherwise it returns false.
12. v1.equals (v2): It returns true if v1 is equal to v2 otherwise false.

Q.1. WAP to add 6 elements into a vector and perform the following

1. Display the vector after adding 6 elements
2. delete from position 0,2 and 3.
3. Insert elements in position 1,2 and 4

```
import java.util.*;
class Vector1
{
        public static void main(String args[])
        {
                Vector v = new Vector();
                v.addElement(100);
                v.addElement(200);
                v.addElement(300);
                v.addElement("c++");
                v.addElement("java");
                v.addElement(220.55);
                System.out.println("Vector element are"+v);
                v.removeElementAt(0);
                v.removeElementAt(2);
                v.removeElementAt(3);
                System.out.println("Vector element are"+v);
                v.insertElementAt(50,1);
                v.insertElementAt("FE",2);
                v.insertElementAt(50,4);
                System.out.println("Vector element are"+v);
                System.out.println("Vector  array size= "+v.size());
        }
}
```

Output

```
Vector element are [100, 200, 300, c++, java, 220.55]
Vector element are [200, 300, java]
Vector element are [200, 50, FE, 300, 50, java]
Vector array size = 6
```

Q.2 WAP to add n strings in a vector array. Input new string and check whether it is present in the vector. If it is present delete it otherwise add it to the vector.

```
import java.util.*;
class Vector2
{       public static void main(String[] args)
        {
                Scanner sc=new Scanner(System.in);
                Vector v=new Vector();
                boolean  b=true;

                System.out.println("How many data values");
                int n=sc.nextInt();
                s=new String[n];

                System.out.println("Enter the data elements");
                for(int i=0;i<n ;i++ )
                {
                        s[i]=sc.next();
                }

                for (int i=0;i<n;i++ )// //add n elements to the vector array.
                {
                        v.addElement(s[i]);
                }
                System.out.println("Vector data elements="+v);
                System.out.println("Vector Size="+v.size());
                System.out.println("Enter the new data element");
                String data=sc.next();

                for (int i=0;i<v.size();i++ )
                {
                        if(data.equals(v.elementAt(i)))
                        {
                                v.removeElementAt(i);
                                i--;                    b=false;
                        }
                }
                if(b)
                {       v.addElement(data);
                }
                System.out.println("Vector array=" +v);
        }
}
```

Output:

```
How many data values
5
Enter the data elements
11 22 33 44 55
Vector data elements= [11, 22, 33, 44, 55]
Vector Size=5
Enter the new data element
33
Remove elements=33
After Deletion, Vector array=[11, 22, 44, 55]
```

Q.3 WAP to test whether the given element is present in the vector array.

```
import java.util.*;

class Vector3
{
    public static void main(String args[])
    {
        Vector v = new Vector(50);

        Scanner s=new Scanner(System.in);

        System.out.println("Enter the size of the array");
        int n=s.nextInt();

        System.out.println("Enter the elements");
        for(int i=0;i<n;i++)
        {
            v.addElement(s.next());
        }

        System.out.println("Enter the element to be searched");
        String data=s.next();

        if(v.contains(data))
        {
            System.out.println("Element is present");
        }
        else
        {   System.out.println("Element is not present");
        }
    }
}
```

Output

```
Enter the size of the array  5
Enter the elements
abc
xyz
333
44
-22
Enter the element to be searched
-22
Element is present
```

Q.4 WAP to find frequency of a element in the given Vector array.

```
class Vector4
{
        public static void main(String[] args)
        {
                Scanner sc=new Scanner(System.in);
                Vector v=new Vector();

                System.out.println("How many elements");
                int n=sc.nextInt();

                int count=0;

                System.out.println("Enter the elements");

                for (int i=0;i<n;i++ )
                {
                        v.addElement(sc.next());//               //add n elements to the vector array.
                }

                System.out.println("Vector elements are ="+v);

                System.out.println("Enter the new element");
                String data=sc.next();

                for (int i=0;i<v.size();i++ )
                {
                        if(data.equals(v.elementAt(i)))
                        {
                                count++;
                        }
                }
                System.out.println("Frequency ="+count);
        }
}
```

Output:

```
How many elements  5
Enter the elements
55  44  33  55  55
Vector elements are =[55, 44, 33, 55, 55]
Enter the new element
55
Frequency =3
```

Short note:

1. java.util.Random: Random class generates a stream of pseudo-random numbers. To create a new random number generator, use one of the following methods: new Random() , new Random(long seed). The form new Random () initializes the generator to a value based on the current time. The form new Random(long seed) seeds the random number generator with a specific initial value; use this if an application requires a repeatable stream of pseudo-random numbers. To create a pseudo-random number, use one of the following functions:

Random() : Creates a new random number generator.
Random(long) : Creates a new random number generator using a single long seed.
nextDouble() : Generates a pseudorandom uniformly distributed double value between 0.0 and 1.0.
nextFloat() : Generates a pseudorandom uniformly distributed float value between 0.0 and 1.0.
nextInt(): Generates a pseudorandom uniformally distributed int value.
nextLong(): Generate a pseudorandom uniformally distributed long value.

2. The java.util.Date: A wrapper for a date. This class lets you manipulate dates in a system independent way. To print today's date use:

```
Date d = new Date();     System.out.println("today = " + d);
```

To find out what day corresponds to a particular date:

```
Date d = new Date(63, 0, 16); // January 16, 1963
System.out.println("Day of the week: " + d.getDay());
```

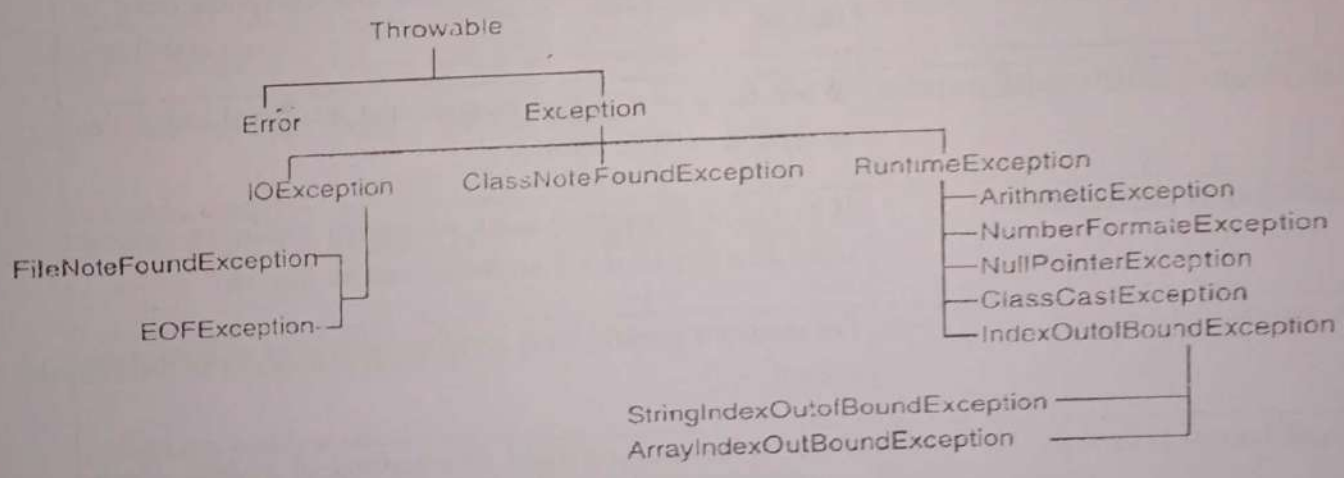
getMinutes() :Returns the minute.
getMonth() :Returns the month.
getSeconds() :Returns the second.
getTime() :Returns milliseconds.
getYear() :Returns the year after 1900.
setDate(int) :Sets the date
setDay(int) :Sets the day of the week.
setHours(int) :Sets the hours.
setMinutes(int) :Sets the minutes.
setMonth(int) :Sets the month.
setSeconds(int) :Sets the seconds

11. EXCEPTION HANDLING

1. **An Exception is an abnormal condition that is caused during execution (run-time environment) of a program.**
2. When the java interpreter encounters an error such as dividing an integer by zero, it creates an exception object and throws it. (i.e. displays message that error has occurred)
3. When there is run time error, the JVM generates Exception Object. The generated object needs to be caught. If not caught, then it stops flow of execution of a program.
4. Java handles exception using five keywords try, catch, finally, throws and throw.
5. If the exception object is not caught and handled properly, interprets will display an error message and will terminate the program causing to fail or crash.
6. If we want to continue with execution of remaining code, then we should try to catch the exception object thrown by the error condition and then display an appropriate message for corrective action. This task is known as Exception Handling.
7. The main subclass of Exception class is the Runtime Exception Class. For example, try to access of invalid array index, division by zero, etc cannot be known at compilation time, generated at runtime.

Object class & Its Subclasses for Exception

One of the subclass of Object class is throwable.



Two types of exceptions:

Checked Exceptions: Exceptions that are checked during the compilation process are called as checked Exceptions. A checked exception is an exception that is typically a user error or a problem that cannot be foreseen by the programmer.

Unchecked Exceptions: Exceptions which are thrown during Runtime by java interpreter are known as Unchecked Exceptions. Exceptions generated from runtime are called ***unchecked***

exceptions, since it is not possible for the compiler to determine that your code will handle the exception. Exception classes that descend from *RuntimeException* and *Error* classes are unchecked exceptions. Examples for *RuntimeException* are illegal cast operation, inappropriate use of a null pointer, referencing an out of bounds array element. *Error* exception classes signal critical problems that typically cannot be handled by your application. Examples are out of memory error, stack overflow, failure of the Java VM.

Thrown exceptions are referred to as ***checked exceptions***. The compiler will confirm at compile time that the method includes code that might **throw** an exception. Moreover the compiler requires the code that calls such a method to include this call within a **try** block, and provide an appropriate **catch** block to catch the exception.

When an (either checked or unchecked) exception is thrown, execution will attempt to immediately branch to the first **catch** block whose associated exception class matches the class or a superclass of the thrown exception. If the exception does not occur within a **try** block or the thrown exception is not caught in a matching **catch** block, execution of the method immediately terminates and control returns to the invoker of the method, where this process is repeated. The result is that the exception chain is escalated until a matching **catch** block is found. If not, the thread containing the thrown exception is terminated.

Exceptions:

Name	Description
ArithmeticException	For Handling an arithmetic error this exception must be thrown.
ArrayIndexOutOfBoundsException	When there is a chance of accessing an illegal index this exception must be thrown
ClassNotFoundException	If by specified name of class, we cannot obtain the desired class then this kind of Exception must be thrown
Exception	For raising a general error condition this kind of Exception is used.
IllegalAccessException	If we try to access a field or a method of a class which itself is not accessible then this kind of exception occurs
NullPointerException	Caused when an attempt to access an object with a Null reference is made.
IOException	When an illegal input/output operation is performed then this exception is raised.
IndexOutOfBoundsException	An index when gets out of bound . this exception will be caused.

ArrayStoreException	When a wrong object is stored in an array this exception must occur
EmptyStackException	An attempt to pop the element from empty stack is made then this exception occurs
RuntimeException	To show general run time error this exception must be raised.
ClassCastException	This type of exception indicates that one tries to cast ab object to a type to which the object to a type to which the object cannot be casted.
IllegalStateException	This exception shows that a method has been invoked at an illegal or inappropriate time. That means when Java environment or Java application is not in an appropriate state then the method is invoked.

Throwable class has two useful methods:

void **printStackTrace()**

Prints the Throwable and the Throwable's stack trace. This is what you've seen printed on the console when an exception is thrown but not caught, causing non-GUI programs to stop.

String **getMessage()**

Gets the detail message string of the Throwable.

```
public void f()
{       new Throwable("In X.f").printStackTrace();
}
```

Mechanism of Exception Handling

1) Find the problem (Hit exception)
2) Inform that error has occurred (Throw the exception)
3) Receive the error (catch the exception)
4) Take corrective actions (handle the exception)

try - catch Block

Program statements which need to be monitor / checked for exception are included in try block. If the exception occurs within the try block, then java runtime system creates an exception objects and throws it and catch block tries to match exception object thrown by try block with ExceptionType mentioned in the catch block.

Syntax

```
try
{
        /* block of code to checking errors (Generates an Exception) */
}
catch(ExceptionType ExceptionObject)
{
        /*Exception handler for ExceptionType (process the exception)*/
}
```

If it matches then exception is caught by catch block. Otherwise, default handler will terminate the program by displaying message. We should take corrective mechanism in catch block. The catch block cannot be separated from the try block. Once the errors object is generated within the try block, the remaining statement will not be executed .Instead control is transferred to catch block. Every try statement should have at least one catch statement, otherwise compiles reports errors.

Program : try and catch to handle arithmetic exception

```
class ExceptionDemo
{
        public static void main(String args[ ])
        {
                int a = 10,b = 5,c = 5;
                int x, y;

                try
                {   x = a / (b - c);
                }
                catch(ArithmeticException e)
                {
                        System.out.println("Division by zero");
                }
                y = a / (b + c);
                System.out.println("y = " +y);
        }
}
/*************************************OUTPUT*************************************/
        Division by zero
        y = 1
```

Multiple Catch Statements

When an exception in a try block is generated, java treats the multiple catch statements like cases in switch statement. The first statement whose parameter matches with the exception object will be executed, and the remaining statement will be skipped.

Syntax:

```
try
{
    /* block of code to checking errors (Generates an Exception) */
}
catch(Exceptiontype1 exceptionobject)
{
    // processes the exception of type Exceptiontype1
}
catch(Exceptiontype2 exceptionobject)
{
    // processes the exception of type Exceptiontype2
}
|
|
catch(ExceptiontypeN exceptionobject)
{
    // processes the exception of type ExceptiontypeN
}
```

Program

```
class MultipleHandleDemo
{
        public static void main(String args[ ])
        {
                int a[ ] = {5, 10};
                int b = 5;

                try
                {
                        b=b/(a[0]-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);
        }
}
```

finally block

The finally block is used to display the information or to execute the statements, which needs to executed irrespective of the catch block. This means that the finally block must be executed in spite of the fact that a catch block is executed or not. Some rules for using the finally block are as follows.

A finally block must be declared at the end of the last catch block. if finally block is declared before a catch block, then program will not compile successfully. Multiple finally blocks cannot be used with the try block. Doing so generates a compile time error starting that a finally block cannot be used without a try block, even though we have used the try block in the program.
Java Application may use the resources of database system, files, operating system. After the use of these resources, java must free those resources. Code to free the resources is written inside the finally Block. Garbage collector can be also called through finally Block.

Throwing our own Exception (Managing user defined Exception Handling)

There may be situation where we would like to throw our own Exception (user defined Exception).We can do that by using 'throw' clause. When you discover an error you can't handle locally, it's time to use the exception mechanism.
When throw statement is executed, it stops execution of current method and generates the exception object.
The syntax for the throw statement is:

throw new NameOfExceptionClass();

Ex. throw new NumberFormatException( );

'throws' clause

This keyword is used in Java to throw the exception objects from the called method to the calling method. It works similar to return statement. If a method generates an exception and does not handle the exception by catch block, then the programmer must specify the keyword throws so that the caller of the method can guard its code against that exception. If a method generates a checked exception and the programmer does not want to handle this exception, then he should throw it out by using the keyword throws, otherwise an error would be flagged by Java compiler. However, in the case of unchecked exception, if the programmer does not want to handle the exception, then he should not throw it out by using the keyword throws. -\ Exception handling provides two benefits. These are :(a) allows fixing the bug. (b) prevents automatic termination of the program.

Syntax to use 'throw' and 'throws' clause

```
returntype methodname(parameter list) throws NameOfExceptionclass
{
        if(condition)
        {               throw new NameOfExceptionclass();
        }
}
```

Program:

```
class MyException extends Exception
{
    MyExtention(String message)
    {
        Super(message);
    }
}

class TestMyException
{
    public static void main(String args[ ])
    {
        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");
        }
    }
}
```

OUTPUT

```
Caught my exception
Number is too small
I am always here
```

Difference between throw and throws?

	throw	throws
1	throw is a keyword used for hitting and generating the exception which are occurring as a part of method body	throws is a keyword which gives an indication to the specific method to place the common exceptions methods as a part of try and catch block for generating user friendly error messages
2	The place of using throw keyword is always as a part of method body.	The place of using throws is a keyword is always as a part of method heading
3	When we use throw keyword as a part of method body, it is mandatory to the java programmer to write throws keyword as a part of method heading	When we write throws keyword as a part of method heading, it is optional to the java programmer to write throw keyword as a part of method body.

Difference between checked Exception and un-checked Exception ?

	Checked Exception	Un-Checked Exception
1	checked Exception are checked at compile time.	un-checked Exception are checked at run time
2	Ex. IOException, SQLException, FileNotFoundException etc.	Ex: ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, NumberNotFoundException etc.

Q 1. WAP to illustrate ArithmeticException

```
class ArithmeticExceptionDemo
{
   public static void main(String  args[ ])
   {
      int a=10,b=5,c=5,d;
      try      // Program to demonstrates try and catch block
      {
            d=a/(b-c);// Exception here
            System.out.println (" This will not be Executed");
      }
      catch (ArithmeticException x)
      {
            System.out.println("Error Occurred"+ x); // catches divide-by-zero
            c=2;
            d=a/(b-c);
      }
```

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

Output

```
Error Occured  java.lang.ArithmeticException: / by
zero
a= 10
b= 5
c= 2
d= 3
```

Q.2) WAP to demonstrate try with multiple catches.

```
class  MultipleCatchDemo
{
        public static void main(String args[ ])
        {
                int arr[] = {5,20};
                int a = 5;
                try
                {       int b = arr[4]/(a-arr[1]);
                }
                catch(ArithmeticException e)
                {
                        System.out.println("\n Division by zero....."+e);
                }
                catch(ArrayIndexOutOfBoundsException e)
                {       System.out.println("\n Array index Error....."+e);
                }
                catch(ArrayStoreException e)
                {
                        System.out.println("\n Wrong Data Type....."+e);
                }
                System.out.println("\n After try/catch block....");
        }
}
```

Q.3) WAP to input the data through commandLine and Find out total valid and in-valid integer

```
import java.io.*;
class NumberDemo
{
    public static void main(String args[])
    {
        int a[ ]=new int[10];
        int j=0,count=0;

        for (int i=0;i<args.length ;i++ )
        {
            try
            {
                a[j]=Integer.parseInt(args[i]);
                j=j+1;
            }
            catch ( NumberFormatException e)
            {
                count++;   System.out.println(" Error Occured " + e );
            }
        }
        System.out.println("Total  valid integer are = "+j);
        for (int i=0;i<j ;i++ )
        {
            System.out.println( a[i]);
        }
        System.out.println("Total  In-valid integer are = "+count);
    }
}
```

Output:

```
C:\>javac NumberDemo.java
C:\>java NumberDemo    5.4 55 323 544 22
Error Occured java.lang.NumberFormatException: For input
string: "5.4"Total valid integer are = 4
323
544
22
Total  In-valid integer are= 1
```

Q.4) WAP to test whether the inputted data is byte/short/int/long/float/double/String/char .

```
import java.io. *;
class CheckData
{
        public static void main(String arg[]) throws IOException  {
                DataInputStream d=new DataInputStream(System.in);
                System.out.println("Enter a data of any type");
                String data=d.readLine() ;

                try
                {       byte b=Byte.parseByte(data);
                        System.out.println("The number is byte  "+b);       System.exit(0);
                }
                catch (NumberFormatException e)  {  }

                try
                {       short b=Short.parseShort(data);
                        System.out.println("The number is short integer="+b);
                                System.exit(0);
                }
                catch (NumberFormatException e){  }

                try
                {       int  i =Integer.parseInt(data);
                        System.out.println(" Number is integer  " +i);       System.exit(0);
                }
                catch(NumberFormatException e1)  {  }

                try
                {       long l=Long.parseLong(data);
                        System.out.println("Number is long " +l);    System.exit(0);
                }
                catch (NumberFormatException e4)  {  }
                try
                {       float f=Float.parseFloat(data);
                        System.out.println("Number is float " +f);    System.exit(0);
                }catch (NumberFormatException e3) {       }

                try
                {       double  d1 =Double.parseDouble(data);
                        System.out.println(" Number is double " +d1);  System.exit(0);
                }catch(NumberFormatException e2)  {       }
```

```
                if(data.length()==1)
                {       System.out.println("It is character");
                }
                else
                {       System.out.println("It is String");
                }
        }
}
```

Output:

```
Enter a data of any type
333.444
Number is float 333.444
```

Q.5) WAP to input the Student name, seatno, and marks of six Subjects through commandLine. If user enters invalid marks program should take corrective action

```
import java.io.*;

class StudentResultDemo
{
        public static void main(String args[]) throws IOException
        {
                int total, seatno;
                BufferedReader  b= new BufferedReader(new InputStreamReader (System.in));

                String name=args[0];
                seatno=Integer.parseInt(args[1]);

                for (int i=2;i<7 ;i++ )
                {
                        try
                        {       total=total+Integer.parseInt(args[i]);
                        }
                        catch ( NumberFormatException e)
                        {       System.out.println(" Error Occured invalid marks " + e );
                                System.out.println("Enter a marks ");
                                int m=Integer.parseInt(b.readLine());
                                total=total+m;
                        }
                }
                System.out.println("Result ");
                System.out.println("Name="+name+"Seat no="+seatno+"Total="+total );
        }
```

Q.6) WAP to demonstrate the nested try-catch and perform the following task.

1) If No commandLine argument are present, then program should generate divide by zero
2) If one commandLine argument is supplied then program should generate divide by zero
3) If two commandLine argument is supplied then program should generate IndexOutBoundsException

```
class ExceptionDemo5
{
        public static void main(String args[ ])
        {
                try
                {
                        int x = args.length;
                        System.out.println("Number of Command line arguments: "+x);
                        int y = 500/x;
                        try// nested try block
                        {
                                if (x==1)
                                {       x = x/(x-x);    // division by zero
                                }
                                if(x==2)
                                {       int arr[]={5,20,25};
                                        arr[20]= 100;   // generate an out-of-bounds exception
                                }
                        }
                        catch(ArrayIndexOutOfBoundsException e)
                        {
                                System.out.println("\n Array index out of Bounds...."+e);
                        }
                }
                catch(ArithmeticException e)
                {               System.out.println("\n Division by zero....."+e);
                }
        }
}
```

Q.7) WAP to calculate the Result. Result should consist of name, seatno and marks of semester I Exam. Create a User Defined Exception class MarksOutOfBoundsException. If Entered marks of any subject is greater than 100 or less than 0, and then program should create a user defined Exception of type MarksOutOfBoundsException and must have a provision to handle it.

```
import java.io.*;
class Student
{
        String name;
        int seatno;
        int M[ ];
        BufferedReader b= new BufferedReader (new InputStreamReader(System.in));
        int total;
        void getdata( )
        {
                try
                {       System.out.println("Enter a name");
                        name=b.readLine();

                        System.out.println("Enter the seatno");
                        seatno=Integer.parseInt(b.readLine());
                        M=new int[5];

                        for (int i=0;i<5 ;i++ )
                        {       System.out.println("Enter the marks of  sub " +(i+1));
                                M[i]=Integer.parseInt(b.readLine());
                        }
                }
                catch (IOException e) { }
        }

        void calresult() throws MarksOutOfBoundsException
        {
                        for (int i=0;i<5 ;i++ )
                        {
                                if(M[i] >100 || M[i] <0 )
                                {       throw new MarksOutOfBoundsException(M[i]);
                                }
                                else
                                {       total=total+M[i];
                                }
                        }
                        System.out.println("Name=" +name+ "Total marks= "+total);
        }
```

```
    public static void main(String[] args)
    {
        Student s=new Student( );
        boolean b=true;
        while(b)
        {
            try
            {       s.getdata();
                    s.calresult();
                    break;
            }
            catch (MarksOutOfBoundsException x)
            {       System.out.println(" Pleass validate the marks  " + x);
            }
        }
    }
}
```

Output:

```
Enter a name
Nitin Pai
Enter the seatno
2343
Enter the marks of  sub 1
55
Enter the marks of  sub 2
44
Enter the marks of  sub 3
66
Enter the marks of  sub 4
44
Enter the marks of  sub 5
888
Please validate the marks : 888
Pleass validate the marks  MarksOutOfBoundsException
Enter a name
Umesh Patil
Enter the seatno
3434
Enter the marks of  sub 1
55
Enter the marks of  sub 2
43
Enter the marks of  sub 3
67
Enter the marks of  sub 4
76
Enter the marks of  sub 5
88
Name=Umesh Patil  Total marks= 538
```

Q.8) WAP to create an User defined Exception class known as PayOutOfBoundsExcep
Organization does not offer basic salary less than 8000. If entered salary is less than 8000 th
program should create an Exception of Type PayOutOfBoundsException. Program should calcula
gross salary by considering DA, HRA,TA.

```
import java.io.*;
class PayOutOfBoundsException extends Exception
{
  PayOutOfBoundsException(double d )
  {
      System.out.println("Invalid Basic salary entered= "+d);
  }
}
class SalaryCal
{
      String name;
      int id, basic;
      double sal;
      DataInputStream d=new DataInputStream(System.in);

      void getdata( )
      {
        try
        {     System.out.println("Enter  name ");
              name=d.readLine();
              System.out.println("Enter  id ");
              id=Integer.parseInt(d.readLine( ));
              System.out.println("Enter basic salary ");
              basic=Integer.parseInt(d.readLine( ));
        }
        catch (IOException e)   {     }
      }
      void calculate( ) throws PayOutOfBoundsException
      {
              if(basic<8000)
              {
                throw new PayOutOfBoundsException( basic);
              }
              else
              {
                sal=basic+basic*0.72+basic*0.30+basic*25+1000;
              }
    }
```

```
    void putdata()
    {
            System.out.println("Employee details are:");
            System.out.println("Name="+name);
            System.out.println("id="+id+"salary="+sal);
    }
}

class SalaryCalDemo{
    public static void main(String x[ ])    {
            SalaryCal s=new SalaryCal();
            boolean b=true;

            while(b)
            {
                try
                {                s.getdata();
                                 s.calculate();
                                 s.putdata();
                                 b=false;

                }
                catch(PayOutOfBoundsException e) {        }
            }
        }
}
```

Output:

```
Enter name
Manish Patil
Enter id
3333
Enter basic salary
1000
Invalid Basic salary entered= 1000.0
Enter name
Sachin dodhia
Enter id
4444
Enter basic salary
9000
Employee details are:
Name=Sachin dodhia
id=4444 salary=244180.0
```

12. MULTITHREADED PROGRAMMING

What is Multithreading?

It is conceptual programming in which a program (process) is divided into two or more subprograms which can be implemented at the same time in parallel.

- Multithreaded means two or more parts of program can run concurrently i.e. at the same time you can run two or more parts (or method) of the program.
- Each part of that program called Thread.
- Each thread defined a separate path of execution
- Multithreading enables you to write very efficient program. Using multithreading we can maximize the use of CPU. Multithreading is part of Multitasking in Operating system.
- Multitasking comes in two flavor 1) Process based and 2) Thread based.
- Process based means separate program and Thread means part of the program
- For example If you are running MSWORD and MSEXCEL at the same time means you are running the two process concurrently.
- If you are running only MSWORD and you are typing as well as Spell CHECK running in background i.e. different thread in one process.
- Thread lightweight version of Process. Process is heavyweight

Example: One program can read input from the user while other can print the characters as read. It is similar to divide the work and assigning it to separate persons who can work on it simultaneously.

Difference between Multithreading and Multitasking

Multithreading	Multitasking
Thread is a fundamental unit of multithreading.	Program or process is a fundamental unit of multitasking.
Multiple parts of a single program gets executed in multithreading environment.	Multiple programs get executed in multitasking environment.
During multithreading the processor switches between multiple threads of the program.	During multitasking the processor switches between multiple programs or processes.
It is cost effective because CPU can be shared among multiple threads at a time.	It is expensive because when a particular process uses CPU other processes has to wait.
It is highly efficient.	It is less efficient in comparison with multithreading.
It helps in developing efficient application programs.	It helps in developing efficient operating system programs.

ADVANTAGES OF MULTITHREADING

1) **Faster Execution Time:**
Since separate threads can perform independent tasks, Multithreading can speed up execution time.

Ex. If there is a program where you need to read input from the user and operate on data as it comes. If you operate on parts of data it may happen that the user may not input data for a long time. In a normal program you would have to wait for it until user enters the data. However, in multithreaded programs while the user is idle, you can take the part of data entered by user already and work on it.

PROBLEMS IN MULTITHREADING:

1) **Data can be corrupted in a multithreaded environment:** If more than one thread manipulates shared variables or objects at the same time, corruption may result. Instance variables are shared between threads. If one is modified by a thread, the change affects the other threads.

2) **Possibility Deadlock:** This can occur due to circular dependency of one thread waiting for resource occupied by another and another waiting for the resource occupied by the first one.

HOW TO CREATE THREADS

There are two ways to create thread:

1) We can **create thread by extends the Thread class**
2) We can **create thread by implement the Runnable interface.**

FIRST APPROACH

Steps to create Thread by extending Thread class

1) Create any class and extend Thread in it. 2). Create instance(Object) of that class
3) Extending class should override run() method. 4)Call start() to begin execution of new Thread.

Syntax:

```
class NameOfChildThreadClass extends Thread
{
        public void run( )
        {
                //code associated with child thread class
        }
}
```

SECOND APPROACH

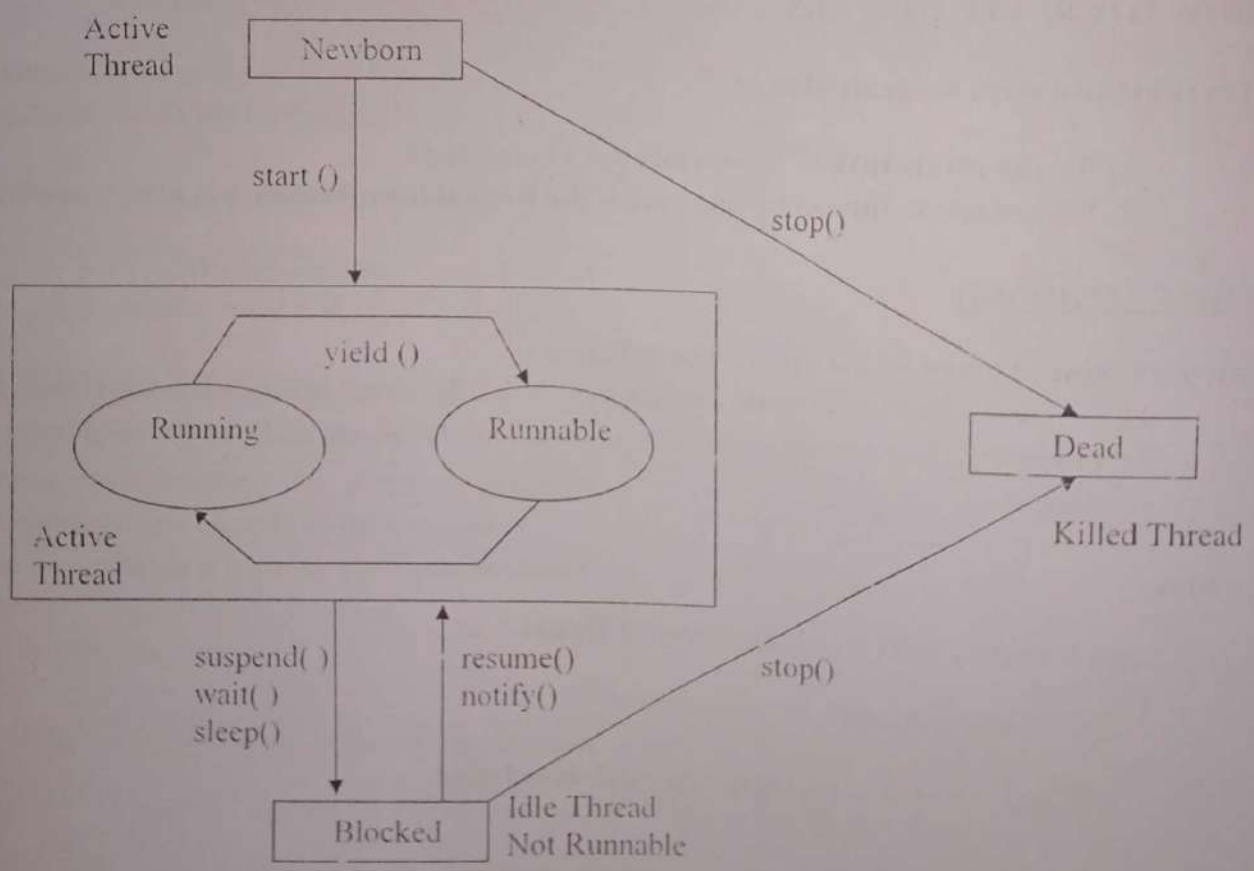
Steps to create Thread by implements runnable

1) A class implements the Runnable interface, providing the run() method that will be exec by the thread. An object of this class is a Runnable object.
2) An object of Thread class created by passing a Runnable object as argument to the Thre constructor. The Thread object now has Runnable object that implements the run() method.
3) The start() method is invoked on the Thread object created in the previous step. The start() method returns immediately after a thread has been spawned.

Syntax:

```
class NameOfChildThreadClass implements Runnable
{
        public void run( )
        {
                //code associated with child thread class
        }
}
```

THREAD LIFE CYCLE



Thread Life Cycle or Thread Transitions:

- Life cycle of thread can be expressed as different thread states.
- Different thread states can be as follows
  1) Newborn state 2) Runnable state 3) Running state 4) Blocked State 5) Dead state.
- A thread is always in one of these five states. It can switch from one state to other state
- As shown in figure during the life cycle of thread, thread is always in one of the state from five states.
- **New Born State:** The moment you create the thread we can say thread is born and we can say thread state is newborn state.
- After thread is born we can block the thread or stop thread by using standard thread method from Thread class.
- **Runnable State** The runnable thread means that the thread is ready for execution and its waiting for availability of the processor. That means thread is in the queue.
- Thread follows the priority higher priority thread will run first then lower.
- In some situation the same or equal priority thread run in round robin fashion. However if we want to relinquish control to other thread to equal priority before its turn, we can use yield () method.
- **Running state** means simply thread is running and utilizing the processor.
- Running thread can be suspended by suspend() methods and can be resume by resume() method.
- Running thread we can put to sleep means for the specified time period we can put thread to sleep means out of queue for specified the time period inside sleep(time in milliseconds) method, Once the time period is elapsed the thread re enter in runnable state.
- We can say to thread to wait till some event occur by wait() method and same thread can resume its operation.by notify() method.
- **Blocked state** means thread prevented from entering into runnable state. It can achive by using suspend() method.
- **Dead State:** Just like human being every born thread has to die thread life get over when run() method get over It is the natural death of thread. However we can kill it by using stop() method thus causing the premature death to it. Thread can be killed at any state for example as soon as thread born, or when thread is running, or even when thread is not runnable state.

CONSTRUCTOR OF THREAD CLASS

default constructor: When the objects of thread class are created then default constructor are implicitly executed and names of threads are automatically generated starting with Thread-0,Thread-1,Thread-2,....Thread-n. Consider following example

```
Thread t1=new Thread(); // name of t1 is Thread-0
Thread t2=new Thread(); // name of t2 is Thread-1
Thread t3=new Thread(); //// name of t3 is Thread-2
```

Parameterized Constructor (String as parameter): When object of Thread class takes string parameter, Parameterized constructor is implicitly executed and string name becomes the new name of the thread. Consider following example

Thread t1=new Thread (**"java thread"**); **// name of t1 is java thread**

THREAD METHODS

Java's Entire multithreading system builds upon Thread class. Thread class defines several method to manage Thread operations. Few of them like as follows

Method name	Meaning
1) getName()	Use to Obtain thread's name
2) setName(String)	It renames the thread.
3) getPriority ( )	Obtain a thread's Priority
4) setPriority(int)	It sets priority to the thread.
5) isAlive ( )	Determine if thread is still running.
6) join( )	When join() is executed on a thread, then currently running thread is terminated after execution of all the remaining threads.
5) sleep()	It is static method and suspend the thread for a period of time.
7) start( )	When start() method is called on thread object, thread becomes eligible for the execution. If CPU is free, then thread execution start automatically by calling run method.
8) yield ( )	It is static method. When it is called on the thread, it allows higher priority to be executed first.
9) wait()	It changes thread state from active to inactive state.
10) resume()	If thread is sent to a blocked state using suspend method then it can be brought back to runnable state using resume method.
11) notify()	If thread is sent to a blocked state using wait method then it can be brought back to runnable state using notify().
12) notifyAll()	It bring all the waited threads from blocked state to runnable state.
13) currentThread()	It is static method and returns reference of currently running thread.
14) activeCount()	It returns the number of running thread.

THREAD EXCEPTIONS

- While doing multithreaded program in java java provided effective way to handle the errors and exception.
- Java uses its standard mechanism to deals with exception and error is try-catch block
- Java throw IllegalThreadStateException whenever we try to invoke a method that thread cannot handle in a given state for example slleping thread cannot deal with resume() method because sleeping thread cannot receive any instructions.

- Whenever we like to thow an exception in thread we have to supply appropriate exception handler to catch it.
- Following is few forms of catch

```
try
{
        // Statements which need to monitored for Exception
}
catch(ThreadDeath e)
{
        -------------;   //Killer Thread
}
catch  (InterruptedException e)
{
        -----------;        //Cannot handle it in the current state
}
catch (IllegalArgumentException e)
{
        -----------;     //Illegal method argument
}
catch (Exception e)
{
        -------------; //any other exception deal with thread
}
```

THREAD PRIORITY

- In multithread programming we have to decide thread scheduling i.e which thread run first then for how much time and which thread should wait and which thread should be suspend and for how much time.
- All thread scheduling can be done with thread priority in java
- If priority is same then java runtime environment will decide which thread run.
- Thread priority ranges from 1 to 10 and default priority is NORM_PRIORITY.
- 1 MIN_PRIORITY
- 5 NORM_PRIORITY
- 10 MAX_PRIORITY

SYNCHRONIZATION IN THREAD

When two or more threads need to access to a common resource, then these threads need some way to ensure that the resource will be used by only one thread at a time. The process of ensuring one access at a time by one thread is called synchronization. The synchronization is the concept which is based on **monitor**. Monitor is used as mutually exclusive lock. A thread that needs regular and frequent access to the common resource must acquire the lock first and then it is eligible to use the common

resource. When one thread is using the common resource other threads remains in the waiting state until the thread finishes its execution.

In java every object has implicit monitor associated with it. For entering in object's monitor, the method is associated with a keyword **synchronized.** When a particular method is in synchronized state then all other threads have to be there in waiting state.

There are two ways to achieve the synchronization-

1. **Using Synchronized Blocks (Statements)** 2. **Using Synchronized Methods**

Note: synchronization is a keyword (access modifier in java)

Synchronized block

Whenever we want to execute one or more than one statement by a single thread at a time(not allowing other thread until thread one execution is completed) than those statement should be placed in side synchronized block.

```
class Class_Name implement Runnable or extends Thread
{
        public void run()
        {
                synchronized(this)
                {
                        ......................
                        ......................
                }
        }
}
```

Synchronized method

Whenever we want to allow only one thread at a time among multiple thread for execution of a method than that should be declared as synchronized method.

```
class Class_Name implement Runnable or extends Thread
{
        public void run()
        {
                synchronized void fun()
                {
                        ..................
                        ..........................
                }
```

```
            public void run()
            {
                fun();
                ..........................
            }
        }
}
```

Program:

```
class MyThread  extends Thread
{
        String message [] = {"I ","Love ","Java ","Very ","Much"};
         public void run()
         {
                    displayList( );
         }
         public static synchronized void displayList( )
         {
                for (int i=0;i< message. Length() ;i++ )
                {
                        System.out.println("\n" +Thread.currentThread().getName() + message[i]);
                        try
                        {
                                Thread.sleep(2000);
                        }
                        catch (InterruptedException e )        { }
                }
         }
    }
    public class ThreadSync
    {
            public static void  main(String args[])
            {
                    MyThread thread1=new MyThread();
                    MyThread thread2=new MyThread( );

                    thread1.setName("Thread 1:  ");
                    thread2.setName("Thread 2:  ");

                    thread1.start();
                    thread2.start();
            }
    }
```

Output:

```
Thread 1: I
Thread 1: Love
Thread 1: Java
Thread 1: Very
Thread 1: Much

Thread 2: I
Thread 2: Love`
Thread 2: Java
Thread 2: Very
```

Q.1 WAP to print the multiplication table of five, seven and thirteen by creating three threads

```
class FiveTable extends Thread
{
    public void run()
    {
      for (int i=1;i<=10;i++)
      {
       System.out.println(i+" Fives are "+(i*5));
      }
    }
}
class SevenTable extends Thread
{
    public void run()
    {
     for (int i=1;i<=10;i++)
     {
            System.out.println(i+" Sevens are "+(i*7));
     }
    }
}

class ThirteenTable extends Thread
{
    public void run()
    {
      for (int i=1;i<=10;i++)
      {
            System.out.print(i+"Thirteens are "+(i*13));
      }
    }
}
```

```
class MultiThreadingDemo
{
    public static void main(String[] args)
    {
        FiveTable five=new FiveTable();
        SevenTable seven=new SevenTable();
        ThirteenTable t=new ThirteenTable();
        five.start();
        seven.start();
        t.start();
    }
}
```

First Run

```
1 Fives are 5
1 Thirteens are 13
1 Sevens are 7
2 Fives are 10
2 Thirteens are 26
2 Sevens are 14
3 Fives are 15
3 Thirteens are 39
3 Sevens are 21
4 Fives are 20
4 Thirteens are 52
4 Sevens are 28
5 Fives are 25
5 Thirteens are 65
5 Sevens are 35
6 Fives are 30
6 Thirteens are 78
6 Sevens are 42
7 Fives are 35
7 Thirteens are 91
7 Sevens are 49
8 Fives are 40
8 Thirteens are 104
8 Sevens are 56
9 Fives are 45
9 Thirteens are 117
9 Sevens are 63
10 Fives are 50
10 Thirteens are 130
10 Sevens are 70
Press any key to continue . . .
```

Second Run

Again random output

Third Run

Again random output

On each run, we will get different output.

Q.2 WAP to explain the thread sleep

```
class FiveTable extends Thread
{
   public void run()
   {
     try
     {
       Thread.sleep(1000);
     }
     catch (InterruptedException e) { }

     for (int i=1;i<=10;i++)
     {
        System.out.println(i+" Fives are "+(i*5));
     }
   }
}

class SevenTable extends Thread
{
   public void run()
   {
      try
      {
            Thread.sleep(2000);
      }
      catch (InterruptedException e)   {   }

     for (int i=1;i<=10;i++)
     {
            System.out.println(i+"Sevens are "+(i*7));
     }
   }
}
```

```
class NineTable extends Thread
{
  public void run()
  {
     try
     {
          Thread.sleep(36);
     }catch (InterruptedException e)
     {
     }

      for (int i=1;i<=10;i++)
      {
                System.out.println(i+ "Nines are  "+(i*9));
      }
     }
  }

  class ThreadSleepDemo
  {
     public static void main(String[] args)
     {
        FiveTable five=new FiveTable();
        SevenTable seven=new SevenTable();
        NineTable nine=new  NineTable();
        five.start();
        seven.start();
        Nine.start();
        System.out.println("main is died ");
     }
  }
```

Output

Q.3) WAP to create a child thread. Make the execution of child thread infinite. As soon as user enters a single character and child thread should be killed. (i.e. child thread execution must be stopped.

```
import java.io.*;
class ChildThread extends Thread
{
  public void run()
  {
   int x=0;

    while(true)
    {
      System.out.println("Running , ix=" +x);
       x++;

      try
      {       Thread.sleep(1000);
      }catch(InterruptedException e){ }
    }
  }
}

class MyMainThread {
  public static void main(String args[] ) throws IOException  {
   MyThread1 t=new MyThread1();        //Creating instance of Thread
   t.start();

   DataInputStream d=new DataInputStream(System.in);
   System.out.println("Enter a character");

   char ch=(char)d.read( );//wait for keyboard input

   t.stop();      //stop the thread
  }
}
```

Output:

```
Enter a character
Running , ix=0
Running , ix=1
Running , ix=2
Running , ix=3
Running , ix=4
Running , ix=5
```

Q.4 WAP to create the child thread. Comment on the execution of main and Child Thread.

```
class ChildThread implements Runnable
{
   public void run()
   {
      try
      {
         for(int i=5;i>0;i--)
         {
               System.out.print("Child Thread: " +i);
               Thread.sleep(500);
         }
      }
      catch(InterruptedException e)
      {
      }

      System.out.println("\nExiting Child Thread");

   }
}
class ChildThreadDemo
{
   public static void main(String[] args)
   {
      ChildThread i= new ChildThread();
      Thread t=new Thread(i);
      t.start();

      try
      {
         for (int j=5;j>0 ;j-- )
         {
               System.out.println("Main Thread: " +j);
               Thread.sleep(1000);
         }
      }
      catch(InterruptedException e)
      {
      }
      System.out.println("\nMain Thread Exiting");
   }
}
```

Output:

First Run

```
Main Thread: 5
Child Thread: 5
Child Thread: 4
Main Thread: 4
Child Thread: 3
Child Thread: 2
Main Thread: 3
Child Thread: 1

Exiting the Child Thread
Main Thread: 2
Main Thread: 1
Main Thread Exiting ...
Press any key to continue . . .
```

Second Run

```
Main Thread: 5
Child Thread: 5
Child Thread: 4
Main Thread: 4
Child Thread: 3
Child Thread: 2
Main Thread: 3
Child Thread: 1
Exiting the Child Thread

Main Thread: 2
Main Thread: 1

Main Thread Exiting ...
```

Comment:

1. Executions of threads are dependent on Operating system.
2. Main thread may die before child thread and vice-versa.

Q. 5 WAP to implement the Thread class Methods.

Program 1:

```
class Employee extends Thread
{
        public void run()
        {
            System.out.println("Employee Thread");
        }
}

class  ThreadNamingDemo
{
    public static void main(String[]  args)
    {
        Employee myname=new Employee();
        System.out.println("\nOriginal Employee thread is "+myname);
        myname.setName("Akhila's Thread ");
        System.out.println("New name of Employee thread  is ="+myname.getName());
        System.out.println("Name  of Employee thread name is ="+myname);
    }
}
```

Output:

```
Original Employee thread is Thread[Thread-0,5,main]
New name of Employee thread  is =Akhila's Thread
Name  of Employee thread name is =Thread[Akhila's Thread ,5,main]
```

Program 2:

```
class ThreadMethod1
{       public static void main(String[] args)
        {
                Thread t= Thread.currentThread();
                System.out.println("Current thread " + t);
                System.out.println(" Name of the current Thread is " +t.getName() );
                System.out.println(" Priority : " +t.getPriority());
                t.setName("mythread");
                System.out.println("After the name change " +t);
                t.setPriority(2);
                System.out.println(" after the priority change : " +t);
                System.out.println(" Total no of the active thread are " +t.activeCount());
        }
}
```

Output:

```
Current thread Thread[main,5,main]
Name of the current Thread is main
Priority : 5
After the name change Thread[mythread,5,main]
After the priority change : Thread[mythread,2,main]
Total no of the active thread are 1
```

Q.6) WAP to implement the join() Method.

```
class MyChildThread extends Thread
{
        public void run()
        {
                        for (int i=5; i>0 ;i-- )
                        {       System.out.println(" Child thread " +i);
                        }
                        System.out.println(" Exiting from the child thread .....");
        }
}
class MainThread {
        public static void main(String[] args) throws InterruptedException {
                MyChildThread a=new MyChildThread();
                a.start();
                a.join();

                for(int k=5;k>0;k--)
                {       System.out.println("Running the Threads ..... " + k);
                }
                System.out.println (" Exiting from the main thread..... .....");
        }
}
```

Output:

```
Child thread 5
Child thread 4
Child thread 3
Child thread 2
Child thread 1
Exiting from the child thread.....

Running the Threads ..... 5
Running the Threads ..... 4
Running the Threads ..... 3
Running the Threads ..... 2
Running the Threads ..... 1
Exiting from the main thread.....
```

Q.7) WAP to create four threads. During lifetime of each thread, each thread should increment the counter continuously. Print the four counters.

```
class Counter extends Thread
{
  int counter = 0;
  static boolean b=true;
  public void run()
  {
    while (b)
    {
      counter++;
    }
  }
}

class TotalCounts
{
  public static void main(String args[])
  {
    Counter thread1 = new Counter( );
    Counter thread2 = new Counter( );
    Counter thread3 = new Counter( );
    Counter thread4 = new Counter( );

    thread1.setPriority(10);
    thread2.setPriority(1);
    thread3.setPriority(1);
    thread4.setPriority(1);

    thread1.start();
    thread2.start();
    thread3.start();
    thread4.start();
    thread1.b=false;

    System.out.println("Thread 1 counted: " + thread1.counter);
    System.out.println("Thread 2 counted: " + thread2.counter);
    System.out.println("Thread 3 counted: " + thread3.counter);
    System.out.println("Thread 4 counted: " + thread4.counter);
  }
}
```

Output

```
First Run
Thread 1 counted: 13974
Thread 2 counted: 12663
Thread 3 counted: 0
Thread 4 counted: 0

Second Run
Thread 1 counted: 2199
Thread 2 counted: 0
Thread 3 counted: 0
Thread 4 counted: 0
```

13. JAVA APPLETS

Q.1 What is Applet? Give its advantages over applications.

Ans.:

1) Java programs are classified into two groups namely applications and applets.
2) An applet is a Java program that executes inside a web page. An applet is loaded and executed in Java enabled browser.
3) It can access the resources of the host machine.
4) In another way, applets are small applications that are accessed on an internet Server, transported over the Internet, automatically installed, and run as a part of a web document.
5) After an Applet arrives on the client, it has limited access to resources, so that it can produce an arbitrary multimedia user interface and run complete computations without introducing the risk of viruses or breaching data integrity.

ADVANTAGES OF APPLETS

1) Applets are portable and supported by most web browsers.
2) Applets are cross platform and can run on Windows, Mac OS and Linux platform.
3) Applets can work all the version of Java Plug-in.
4) An applet runs on sandbox, so the user does not need to trust the code, so it can work without security approval.

DISADVANTAGES OF APPLETS

1) Java plug-in is required to run applet.
2) Java applet requires JVM, so first time it takes significant startup time.

Q. Explain the difference between Java Application and Java Applet.

Java Application	Java Applet
1) must be installed on local machine.	1) needs no explicit installation on local machine.
2) must be run explicitly within a java-compatible virtual machine.	2) loads and runs itself automatically in java enabled browser.
3) can run with or without a GUI.	3) must run within a GUI (using Abstract Window Toolkit).
4) start execution with its main method.	4) start execution with its init method.
5) once started, manages its own flow of execution.	5) has its flow of execution det6ermined partly by its browser context.
6) has not inherent security restriction.	6) has significantly security control to prevent malicious or poorly written applets from harms users system.

Q.2 Give description on Applet class.

Ans.:

1) The Applet class is contained in the java.applet package. It provides all necessary support for applet execution, such as starting and stopping.
2) It also provides method that load and display images and methods that loads audio clips.
3) Applet class extends the AWT class panel and it extends container, which extends component. This class provides support for Java's windows based, graphical interface. Thus Applet provides all of the necessary support for window based activities.

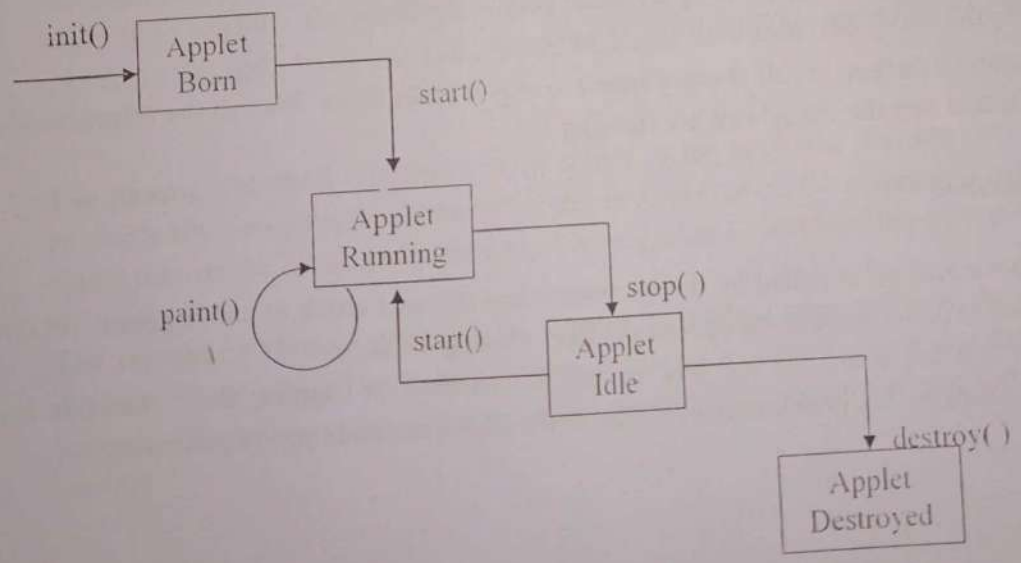
Q. 3 Explain Applet Architecture.

Ans.:

1) Since an applet is windows – based program, that's why applet are event driven. An Applet waits until an event occurs. The AWT notifies the applet about an event by calling an event handler that has been provided by the applet.
2) Once this happens, Applet must take appropriate action and then quickly return control to the AWT.
3) For example, when the user clicks a mouse inside the applet's window, a mouse click event is generated. If the user presses a key while the applet's window has input focus, a key pres event is generated.
4) Applet can contain various controls, such as push buttons and check boxes.
5) When the user interact with these controls, an event is generated.

Q. 4 Describe life cycle of an applet.

Ans.:



LIFE CYCLE OF APPLET

An applet can react to major event such as

1) It can initialize itself.
2) It can start running.
3) It can stop running.
4) It can perform a final cleanup, in preparation for being unloaded.

Consider these four major events, we say that applet goes though 4 states :

1) Born
2) Running
3) Idle
4) Destroyed

BORN STATE (APPLET IS LOADED)

init ( ) Method

The init ( ) method is called the first time an applet is loaded into the memory of a computer, we can initialize variables and add components like buttons and check boxes in init ( ) method

start ( ) Method

The start method is called immediately after the init ( ) method and every time the applet receives focus as a result scrolling in the active winds. we can use this method when we want to restart a process, such as thread animation, every time the applet receives the focus.

stop ( ) Method

The stop ( ) method is called every time the applet loses focus. We can use this method to reset variable and stop the thread that are running.

destroy ( ) Method

The above method is called by the browser when the user move to another page. we can use this method to perform clean-up operations likes closing a life.

If it is not necessary to use any or all the method of Applet. these methods are automatically called by Java environment. None of the above methods accept parameters.

Q. 5 How does initialization and termination of applet occurs ?

Ans.:

When an applet begins, the AWT calls the following methods. in the sequence.

1. init ( ) 2. start ( ) 3. paint ( )

When an applet is terminated. the following sequence of method call take place.

1. stop ( ) 2.destroy ( )

NOTE : Students must describe above two methods in detail. (Refer Q. 4)

Following are the steps required for preparing the applets-

1. Write an applet code in a file and save it using the filename extension **.java.**
2. Compile and create an executable file. This file also have the extension **.class.**
3. Design a web page using the <applet> code inside the web page.
4. Embed the <applet> code inside the web page.
5. Develop the HTML file.
6. Execute and test the applet program.

Q. 6 What is use of update ( ), paint ( ) and repaint ( ) methods ?

Ans.:

When we scroll to an applet, the screen has to be refreshed to show the new content. Windows handles this by making the area (rectangle) that has to be refreshed. The area is then painted to display the result of scrolling. This is handled by update ( ) and paint ( ) methods.

The update ( ) Method : It takes a Graphic class object as a parameter. When a applet area needs to be redrawn, the windows system starts the painting process. The update ( ) method is called to clear the screen and calls the paint ( ) method. The screen is then refreshed by the system.

The paint ( ) Method : It draws the graphics of the applet in drawing area. The method is automatically called the first time the applet is displayed on the screen and and every time the applet receives the focus. The paint ( ) can be triggered by invoking the repaint( ) method.

The repaint ( ) Method : We call the repaint ( ) method when we want the applet area to be redrawn. The repaint ( ) methods calls the update ( ) method to signal that the applet has to be updated. The default action of update ( ) method is clear the applet area and call the paint ( ) method.

Q. 7 WAP to draw an Applet with the message "Life without LOVE is not a life".

Ans.:

```
import java.applet.*;
import java.awt.*;
/*<applet code = "Message Applet" width = 500 height = 500> </applet> */
public class MyApplet extends Applet
{
        public void paint (Graphics g)
        {
                g.drwaString("Life without LOVE is not a life", 10, 20);
        }
}
```

Q. 8 State some simple Applet display Methods.

Ans.:

1) **void drawstring (message, int x, int y);**
   To output a string to an applet. Message is the string to be output beginning at x, y.
2) **void setBackground (Color newcolor)**
   To set the background color of an applet's window.
3) **void setForeground (color newcolor)**
   To set the foreground color. (i.e. color in which text is shown)

Q. 8 Write a program to show the applet skeleton.

Ans:

```
import java.applet.*;
import java.awt.*;
/*<applet code = AppletSkel.class" width = 400 height = 400> </applet>*/

public class AppletSkel extends Applet
{
        public void init ( ) // called first
        {       // initialization
        }

        public void start ( )  // called second
        {
                // start or resume execution
        }

        public void stop ( )
        {
                // suspends execution
        }
```

```
        public void destroy ( )
        {
                // perform shutdown activities
        }

        public void paint (Graphics g)
        {
                // redisplay content of windows
        }
}
```

Q. 9 What is the use of HTML APPLET tag ? Give the element (attributes) of it.

Ans.: The APPLET tag is used to embed (insert) an applet in HTML document. The APPLET tag takes zero or more parameters

Syntax :

<APPLET CODE = "name of the .class file that extends java.applet.Applet" CODE BASE = "path of the class file"
HEIGHT = maximum height of applet in pixel
WIDTH = maximum width of applet in pixel
HSPACE = horizontal space between the applet and the rest of HTML in pixel
SPACE = Vertical space between the applet and the rest of HTML in pixel
ALIGN = alignment of applet with respect to rest of page
ALT = "alternate text to be displayed if browser do not support applets">
<PARAM NAME = "Parameter-name" VALUE = "Value-of-parameter">
<PARAM NAME = "Parameter name" VALUE = "Value-of-parameter">
</APPLET>
Ex. <APPLET CODE = "AppletDemo.class" HEIGHT = 450 WIDTH = 500> </APPLET>

Q. 10 How will you pass parameters to an applet ? Give a suitable example.
Ans.:

The Applet tag in HTML allows us to pass parameters to our applet. To retrieve parameter, we use getParameter ( ) method. It returns the value of specified parameters in the form of string object.
Thus for numeric and Boolean values, we need to convert their string representations into their internal formats.

Example:

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

public class ParamTestApplet extends Applet
{
    Font f= new Font("TimesRoman", Font.BOLD, 40);
    String name;

    public void init( )
    {
        name = getParameters ("name");

        if(name = = null)
        {
            name = "friend";
        }
        name = "Have  nice day" + name;
    }//end of init( )

    public void paint (Graphics g)
    {
        g.setFont (f);
        g.setColor (Color.darkGray);
        g.drawString (name, 50, 50);
    }
}  // end  of the class.
```

In above program. getParameter ( ) method, which accepts the name (a string) as its parameters. If no value is given to the variable "name" then default is given by stat.name = "friend"

```
<applet code = "ParameterTestAppiet.class" WIDTH = 400 HEIGHT  = 400>

<ALIGN = TOP>
<PARAM NAME = name VALUE = "PRIYA">
</applet>
```

The ParamTestApplet.html file must be included along with ParamTestApplet.html file must be
included along with ParamTestApplet.java in the same directory. If HTML file is present in other directory then codebase attribute is used to specify the path.

Q. 11 Give details of getDocumentBase( ), showDocument ( ) and getCodeBase ( ) methods of applet.

Ans.:

In many cases of applet, we need to explicitly load media and text.java will allow the applet to load data from the directory holding HTML file that started the applet (the document base) and the directory from which the applet's .class file was loaded (the base code).

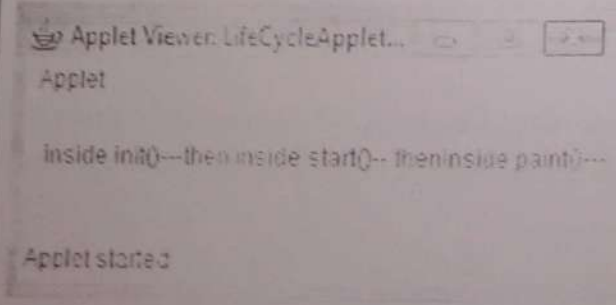
These directories are returned as URL object by getDocumentBase ( ) and getCodeBase ( ). They can be concatenated with a string that names the file you want to load.

To actually load another file, we will use showDocument ( ) method defined in Applet Context interface. Once we obtained document into view by calling showDocument ( ). This method do not return any value or throws exception.

Q. 1 .WAP to demonstrate Life Cycle of Applet.

```
import java.awt.*;
import java.applet.*;
/*<applet code="LifeCycleApplet.class"
height=600 width=1000></applet>*/

public class LifeCycleApplet  extends Applet
{
  String msg="";

  public void init()
  {
      msg = "inside init()---then ";
  }
  public void start()
  {
      msg=msg+ "inside start()-- then ";
  }
  public void paint( Graphics g)
  {
    msg= msg+"inside paint()---then";
    g.drawString(msg,10,30);
  }
  public void stop()
  {
    msg= msg+"inside stop()-- then";
  }
  public void destroy()
  {
    msg=msg+ "inside destroy()--  then";
  }
}
```

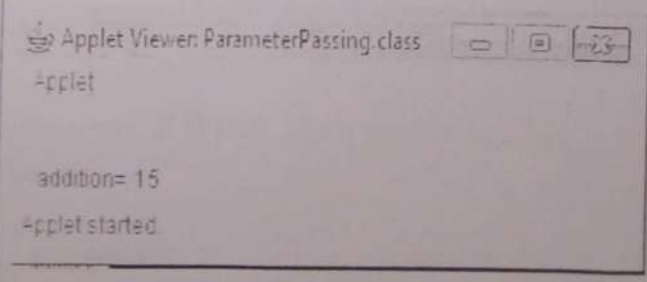
Output:



Q.2 WAP to pass two nos through html file to an applet and calculate addition of two nos.

```
import java.applet.*;
import java.awt.*;
/*
<APPLET
        CODE=ParameterPassing.class
        WIDTH=500
        HEIGHT=500 >
        <PARAM NAME =no1 VALUE = "5">
        <PARAM NAME=no2 VALUE="10">
</APPLET>

*/
public class ParameterPassing extends Applet
{
  String n1,n2;
  int add;

  public void start()
  {
    n1= getParameter("no1");
    n2=getParameter("no2");

add=Integer.parseInt(n1)+Integer.parseInt(n2);
  }

  public void paint(Graphics g)
  {
     g.drawString("addition= "+add, 10, 140);
  }

}
```

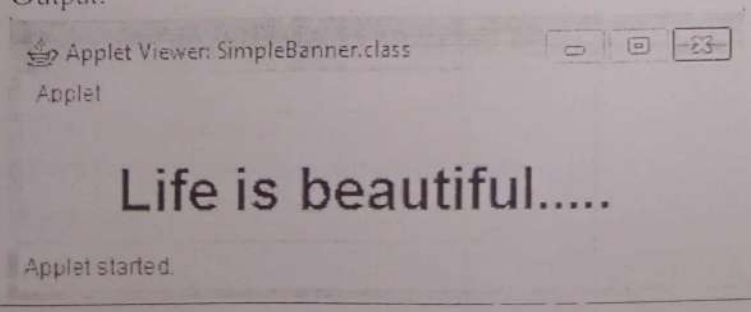
Output:



3. WAP to create Advertisement Banner in an Applet.

```
import java.awt.*;
import java.applet.*;
//<applet code = "SimpleBanner.class" Height=650 Width=1000> </applet>
public class SimpleBanner extends Applet implements Runnable
{
  String msg = "Life is beautiful.....";
  Thread t;
  Font f=new Font("Time new roman",Font.BOLD,30);
  public void init()
  {                setBackground(Color.cyan);   setFont(f);
  }
  public void start()
  {      t = new Thread(this);
         t.start();
  }
  public void run()
  {      try
         {
                for( ; ; )
                {
                   repaint();        Thread.sleep(100);
                   char ch = msg.charAt(0);
                   msg = msg.substring(1,msg.length());
                   msg += ch;
                }
         }
         catch (InterruptedException e){ }
  }
  public void paint(Graphics g)
  {              g.drawString(msg,50,50);
  }
}
```

Output:



Q.4 WAP in an applet where the Ball will move inside the applet window. Whenever it will touch the corner of window, it will move inside the window. At any point Ball will never move out of the Applet Window.

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

/*<applet code="Bouncingball.class"  width=700
height=500></applet>*/

public class Bouncingball extends Applet implements Runnable
{
        int x=30,y=30;
        Thread t;
        boolean  flag1=true,flag2=true,flag3=true;
        boolean  flag4=true, flag=true;

        public void init()
        {
            t=new Thread(this);
            t.start();
        }
        public void run()
        {
            while(flag)
            {
                try
                {
                    x++;y++;
                    if(y>500)
                    {  y--;x++;
                    }
                    if(x>700)
                    {  y--;x--;
                    }
                    if(y<0)
                    {  y++;x--;
                    }
                    if(x<0)
                    {  y++;x++;
                    }
                Thread.sleep(10);
                }
                catch(Exception e) { }
                repaint();
            }
}

public  void destroy()
{
        flag1=false;
        flag2=false;
        flag3=false;
        flag=false;
        flag4=false;
}

public  void paint(Graphics g)
{
        g.setColor(Color.red);
        g.drawOval(x,y,140,140);
        g.setColor(Color.green);
        g.fillOval(x,y,140,140);
}
}
```

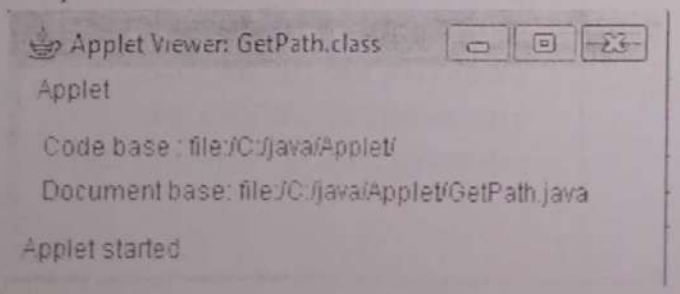
5. WAP to display images in applet.

```
import java.applet.*;
import java.awt.*;
import java.net.*;
/*<applet code="Imagedem.class" height=650 width=1000>
</applet>*/

public class Imagedem extends Applet
{
   Image img;
   public void init()
   {
         img = getImage(getCodeBase(),"noname.jpg");
   }
   public void paint(Graphics g)
   {      g.drawImage(img,5,50,this);
   }
}
```

6. WAP to implement getCodeBase( ) and getDocumentBase( ).

```
import java.awt.*;
import java.applet.*;
/*<applet code="GetPath.class" width=1100 height=650> </applet>*/
public class GetPath extends Applet
{
  public void paint(Graphics g)
  {
   String msg="Code base . " +getCodeBase();
   g.drawString(msg,10,20);
   msg="Document base: " +getDocumentBase();
   g.drawString(msg,10,60);
  }
}
```

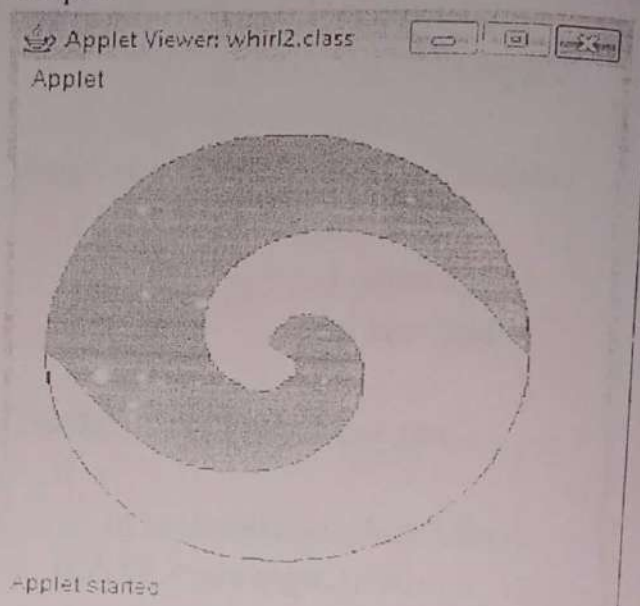
Output:



Q.7 WAP to display same image saved with four different angles one after another repeatedly so that image will appear to be rotating on the applet.

```
import java.awt.*;
import java.applet.*;
/*<APPLET   CODE = whirl2.class
   WIDTH = 300   HEIGHT = 300 >  </APPLET>
*/
public class whiri2 extends Applet implements
Runnable
{
   Image img[] = new Image[4];
   Image currentImage;
   int i = 0;
   Thread t;
   boolean b = true;
   public void init()
   {
    img[0]= getImage(getCodeBase(), "Img1.gif");
    img[1] = getImage(getCodeBase(), "Img2.gif");
    img[2] = getImage(getCodeBase(), "Img3.gif");
    img[3] = getImage(getCodeBase(), "img4.gif");
   }

   public void start()
   {
     t = new Thread(this);
     t.start();
   }
  public void run()
  {
    while(b)
    {
       currentImage = img[i];
       i++;
      if(i > 3)
      {   i= 0;
      }
      repaint( );
      try
      {       Thread.sleep(100);
      }
      catch(InterruptedException e) {   }
    }
}
   public void paint (Graphics g)
   {
   g.drawImage(currentImage, 10, 10, this);
   }
}
```

Output:



7. WAP to display Smiley face in the applet.

```
import java.awt.*;
import java.applet.*;
/*
<applet code="SmileyFace.class"
width=1000      height=650>
</applet>
*/
public class SmileyFace extends Applet
{
public void paint(Graphics g)
{
   setBackground(Color.red);
   setForeground(Color.yellow);

   g.drawOval(200, 150,200,200);
   g.drawOval(240, 200,20,20);
   g.drawOval(335, 200,20,20);
   g.drawLine(300,210,280,270);
   g.drawLine(280,270,310,270);
   g.drawArc(250,250,100,60,180,180);
}
}
```

AWT-(Abstract Window Toolkit) :

Three categories of AWT controls are :

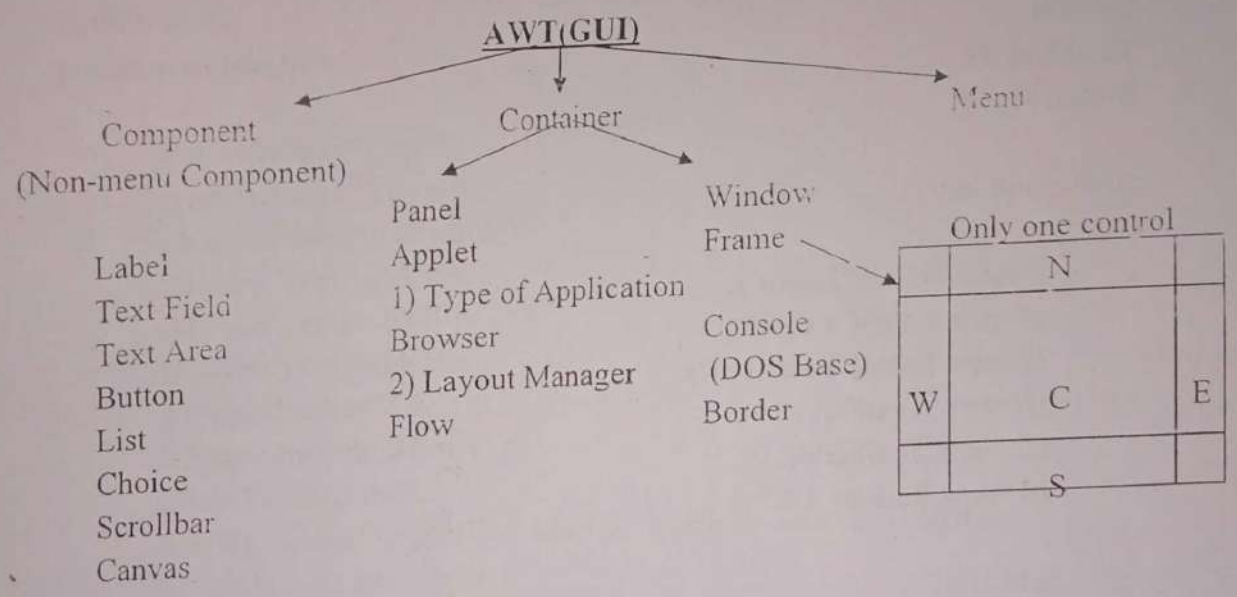
1. Non-Menu Components
2. Container Components
3. Menu Components

Two categories of Container Components are :

1. Panel and Applet 2.Window and Frame

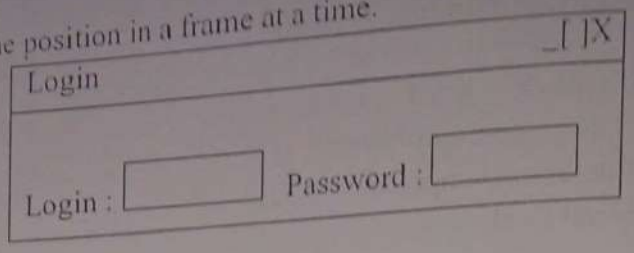
Two reasons for the bifurcation :

1. Based on the type of Application
2. Based on the type of Layout Manager



Panel and Applet has Flow Layout as its default Layout Manager and Frame and Window has Border as its Layout Manager

In a Border Layout one control in one position in a frame at a time.



Applet and Application

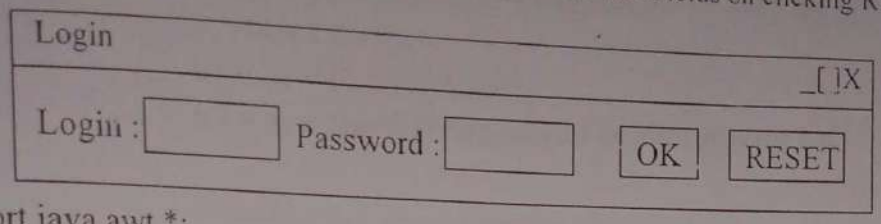
8) Applet

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

public class LoginApplet extends Applet
{
        Frame f;
        Label l1, l2;
        TextField tf1, tf2;
        Button b1;

        public void init()
        {
                f=new Frame("Login");
                l1=new Label("Login");
                l2=new Label("Password");
                tf1=new TextField(10);
                tf2=new TextField(10);
                b1=new Button("OK");

                add(l1);
                add(tf1);
                add(l2);
                add(tf2);
                add(b1);

        }
}
```

Click new → File → html and save this file as CodeDocTextApplet.html

```
<BODY>
<APPLET code= LoginApplet.class HEIGHT=200 WIDTH=400 >
</APPLET>
</BODY>
```

Applet 9 : Take a Login and Password from the user and display it on the third TextField which appears only on clicking OK button and clear both the TextFields on clicking RESET button



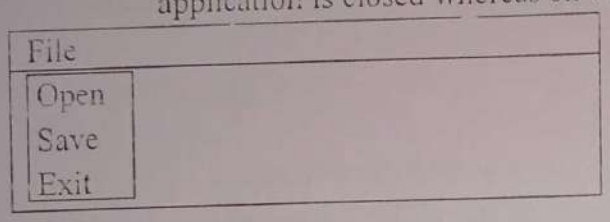

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class LoginApplet1 extends Applet implements ActionListener {
        Frame f;
        Label l1, l2;
        TextField tf1, tf2, tf3;
        Button b1,b2;
        public void init()
        {
                f=new Frame("Login");
                l1=new Label("Login");
                l2=new Label("Password");
                tf1=new TextField(10);
                tf2=new TextField(10);
                tf3=new TextField(35);
                b1=new Button("OK");
                b2=new Button("RESET");
                tf3.setVisible(false);
                add(l1); add(tf1); add(l2); add(tf2); add(b1); add(b2); add(tf3);
                b1.addActionListener(this);
                b2.addActionListener(this);
                tf2.setEchoChar('*');
        }
        public void actionPerformed(ActionEvent e)
        {
                if(e.getSource() == b1)
                {       String s1 = tf1.getText();
                        String s2 = tf2.getText();
                        tf3.setText("Login : "+s1+" "+"Password : "+s2);
                        tf3.setVisible(true);
                }
                else
                {       tf3.setVisible(false);
                        tf1.setText(" ");
                        tf2.setText("");
                }
```

```
            validate();  // to rearrange controls
            repaint();   // to get back to original screen

        }
}
```

Click new → File → html and save this file as LoginApplet1.html

```
<BODY>
<Applet code=LoginApplet1.class height=100 width=400 >
</Applet>
</BODY>
```

Application10: Menu "File" is displayed on the Frame. On clicking File the Popup containing "Open", "Save" and "Exit" options are displayed. On clicking "Open" option FileDialog appears which let user to open file from hard disk. On clicking "Save" option FileDialog appears which let user to save file to hard disk. On clicking "Exit" option Panel containing "YES" and "NO" Buttons appear. On clicking "YES" Button application is closed whereas on clicking "NO" Button, Panel disappear.



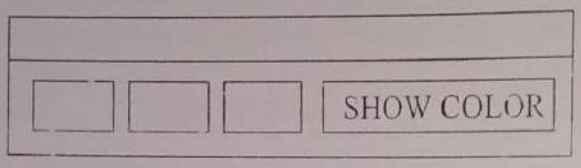

```
Menu m;
MenuBar mb;
MenuItem m1, m2, m3;
```

```
import java.awt.*;
import java.awt.event.*;
class FrameMenu implements ActionListener
{
        Frame f;
        Menu m;
        MenuBar mb;
        MenuItem m1, m2, m3;
        Panel p;
        Button b1, b2;
        Dialog d;
        FileDialog fd1, fd2;

        FrameMenu()
        {
                f=new Frame("Frame with Menu");
                m = new Menu("File");
                mb = new MenuBar();
                m1 = new MenuItem("Open");
                m2 = new MenuItem("Save");
                m3 = new MenuItem("Exit");
                fd1 = new FileDialog(f, "Open File",FileDialog.LOAD);
```

```
        fd2= new FileDialog(f, "Save File",FileDialog.SAVE);
        d = new Dialog(f, "Are You Sure");
        p = new Panel();
        b1 = new Button("YES");
        b2 = new Button("NO");

        //Change in Layout Manager if necessary
        //f.setLayout(new FlowLayout());
        m.add(m1);
        m.add(m2);
        m.add(m3);
        mb.add(m);
        f.setMenuBar(mb);
        m1.addActionListener(this);
        m2.addActionListener(this);
        m3.addActionListener(this);
        f.setSize(200,200);
        f.setVisible(true);
    }
    public void actionPerformed(ActionEvent e)
    {
        if (e.getSource() == m1)
        {
            fd1.show();
        }
        else    if(e.getSource() == m2)
        {
            fd2.show();
        }
        else    if(e.getSource() == m3)
        {
            d.setSize(100,100);
            d.setVisible(true);
            d.add(p);
            p.add(b1);
            p.add(b2);
            b1.addActionListener(this);
            b2.addActionListener(this);
        }
        else if(e.getSource() == b1)
        {
            System.exit(0);
        }
```

```
            else if(e.getSource() == b2)
            {
                    d.setVisible(false);
            }
            f.validate();
            f.repaint();
        }
        public static void main(String [] args)
        {       FrameMenu x = new FrameMenu();
        }
}
```

Applet11: Take color value in three TextField for r, g, and b respectively. On clicking "SHOW COLOR" Button, change the Background color to that color.




```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class FrameColors extends Applet implements ActionListener
{
        Frame f;
        TextField tf1, tf2, tf3;
        Button b1;

        public void init()
        {
                f = new Frame("Frame With Scroll");
                tf1 = new TextField(5);
                tf2 = new TextField(5);
                tf3 = new TextField(5);
                b1 = new Button("Show Color");

                add(tf1);       add(tf2);       add(tf3);       add(b1);

//              add.setLayout(new FlowLayout());
                tf1.addActionListener(this);            tf2.addActionListener(this);
                tf3.addActionListener(this);            b1.addActionListener(this);
        }
        public void actionPerformed(ActionEvent e)
        {       repaint();
        }
```

```
    public void paint(Graphics g)
    {
            String x = tf1.getText();
            String y = tf2.getText();
            String z = tf3.getText();

            int a = Integer.parseInt(x);
            int b = Integer.parseInt(y);
            int c = Integer.parseInt(z);

            Color d = new Color(a,b,c);
            setBackground(d);
    }
}
```

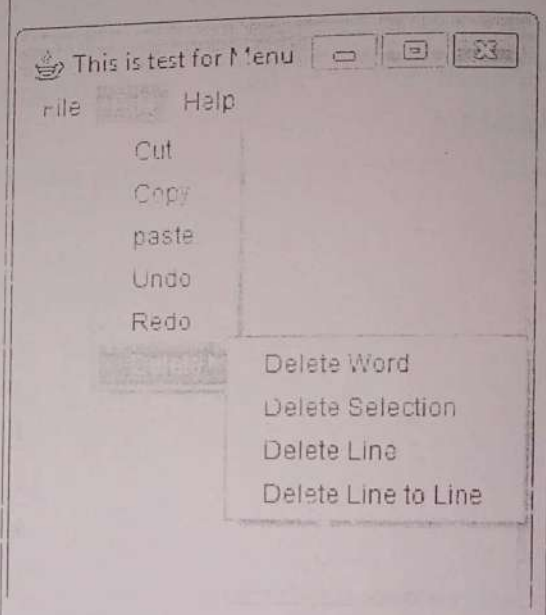
Click new → File → html and save this file as FrameColors.html

```
<BODY>
<Applet code=FrameColors class height=100 width=400 ></Applet></BODY>
```

12. WAP to create different Menus.

```
import java.awt.*;
class MyMenu
{
  public static void main(String x[]) {
      Frame f=new Frame("This is test for Menu");
      MenuBar main= new MenuBar();
      f.setMenuBar(main);

      Menu filemenu=new Menu("File");
      Menu editmenu=new Menu("Edit");
      Menu helpmenu=new Menu("Help");

      main.add(filemenu);
      main.add(editmenu);
      main.add(helpmenu);

      MenuItem new1=new MenuItem("New");
      MenuItem open=new MenuItem("Open");
      MenuItem close=new MenuItem("Close");
      MenuItem line=new MenuItem("-");

      filemenu.add(new1);
      filemenu.add(open);
      filemenu.add(close);
      filemenu.add(line);

      MenuItem cut=new MenuItem("Cut");
      MenuItem copy=new MenuItem("Copy");
      MenuItem paste=new MenuItem("paste");
      MenuItem undo=new MenuItem("Undo");
      MenuItem redo=new MenuItem("Redo");
      Menu delete=new Menu("Delete");

      editmenu.add(cut);
      editmenu.add(copy);
      editmenu.add(paste);
      editmenu.add(undo);
      editmenu.add(redo);
      editmenu.add(delete);
      Menu more=new Menu("More");
      helpmenu.add(more);
      MenuItem dword=new MenuItem("Delete Word");
      MenuItem dsel=new MenuItem("Delete Selection");
      MenuItem dline=new MenuItem("Delete Line");
      MenuItem dl=new MenuItem("Delete Line to Line");
      editmenu.add(delete);

      delete.add(dword);
      delete.add(dsel);
      delete.add(dline);
      delete.add(dl);

  f.setSize(1000,1000);
  f.setVisible(true);
  }
}
```

Output:



12. COLLECTION FRAMEWORK

Collections in java is a framework that provides an architecture to store and manipulate the group of objects. All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Java Collections.

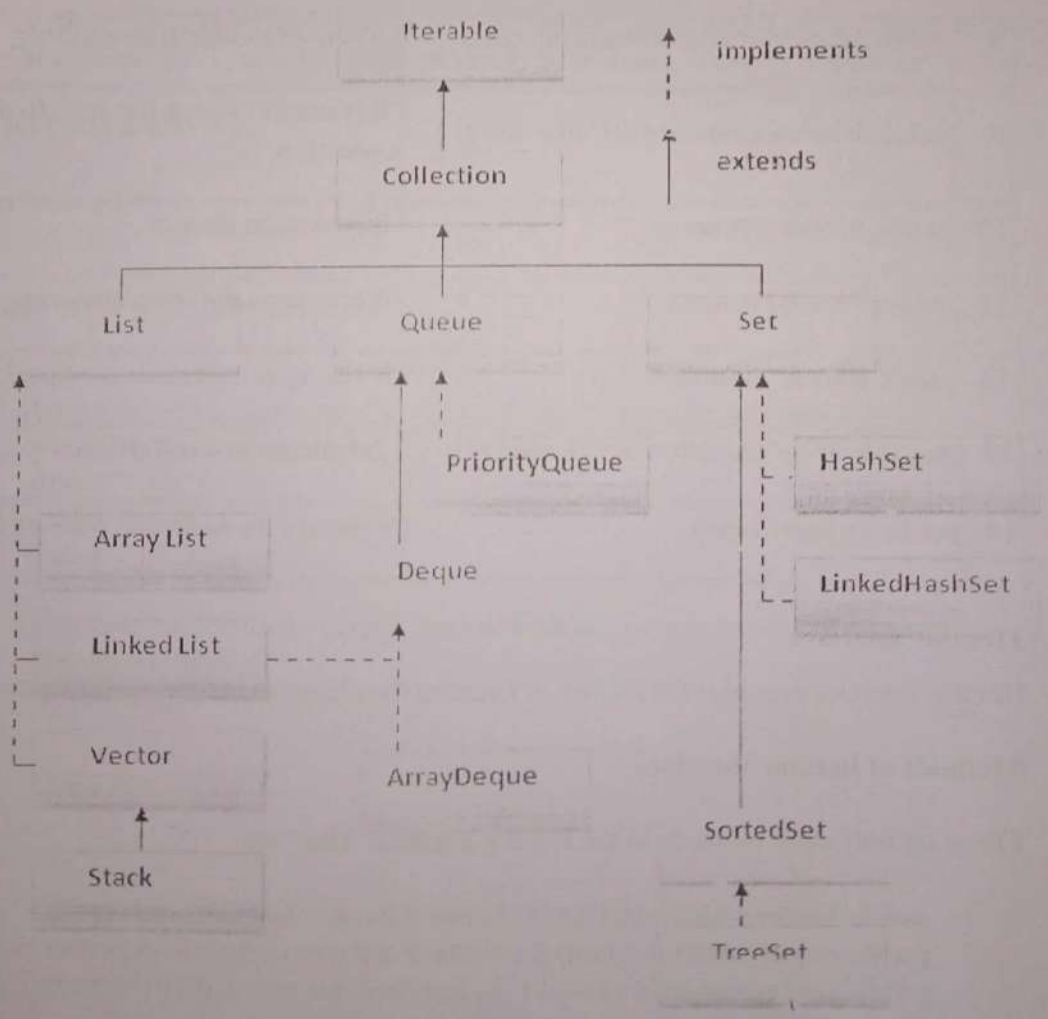
Java Collection simply means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque etc.) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet etc). **Collection in java** represents a single unit of objects i.e. a group.

What is framework in java

- provides readymade architecture.
- represents set of classes and interface.
- is optional.

What is Collection framework: Collection framework represents a unified architecture for storing and manipulating group of objects. It has: 1. Interfaces and its implementations i.e. classes 2. Algorithm

Hierarchy of Collection Framework: Let us see the hierarchy of collection framework. The **java.util** package contains all the classes and interfaces for Collection framework.



Methods of Collection interface

There are many methods declared in the Collection interface. They are as follows:

No.	Method	Description
1	public boolean add(Object element)	It is used to insert an element in this collection.
2	public boolean addAll(collection c)	It is used to insert the specified collection elements in the invoking collection.
3	public boolean remove(Object element)	It is used to delete an element from this collection.
4	public boolean removeAll(Collection c)	It is used to delete all the elements of specified collection from the invoking collection.
5	public boolean retainAll(Collection c)	It is used to delete all the elements of invoking collection except the specified collection.
6	public int size()	It return the total number of elements in the collection.
7	public void clear()	It removes the total no of element from the collection.
8	public boolean contains(object element)	It is used to search an element.
9	public boolean containsAll(Collection c)	It is used to search the specified collection in this collection.
10	public Iterator iterator()	It returns an iterator.
11	public Object[] toArray()	It converts collection into array.
12	public boolean isEmpty()	It checks if collection is empty.
13	public boolean equals(Object element)	It matches two collection.
14	public int hashCode()	It returns the hashcode number for collection.

Iterator interface

Iterator interface provides the facility of iterating the elements in forward direction only.

Methods of Iterator interface

There are only three methods in the Iterator interface. They are.

1. **public boolean hasNext()** it returns true if iterator has more elements.
2. **public object next()** it returns the element and moves the cursor pointer to the next element.
3. **public void remove()** it removes the last elements returned by the iterator. It is rarely used.

Java ArrayList class

- Java ArrayList class uses a dynamic array for storing the elements. It extends AbstractList class and implements List interface.
- Java ArrayList class can contain duplicate elements.
- Java ArrayList class maintains insertion order.
- Java ArrayList class is non synchronized.
- Java ArrayList allows random access because array works at the index basis.
- In Java ArrayList class, manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list.

Java Non-Generic Vs Generic Collection

Java collection framework was non-generic before JDK 1.5. Since 1.5, it is generic. Java new generic collection allows you to have only one type of object in collection. Now it is type safe so typecasting is not required at run time.

Let's see the old non-generic example of creating java collection.

```
1. ArrayList al=new ArrayList ();//creating old non-generic arraylist
```

Let's see the new generic example of creating java collection.

```
1. ArrayList<String> al=new ArrayList<String> ();//creating new generic arraylist
```

In generic collection, we specify the type in angular braces. Now ArrayList is forced to have only specified type of objects in it. If you try to add another type of object, it gives *compile time error*.

Example of Java ArrayList class

```
1. import java.util.*;
2. class TestCollection1
3. {
4.     public static void main(String args[])
5.     {
6.
7.         ArrayList<String> al=new ArrayList<String> ();//creating arraylist
8.         al.add("Ravi");//adding object in ArrayList
9.         al.add("Vijay");
10.        al.add("Ravi");
11.        al.add("Ajay");
12.
13.        Iterator itr=al.iterator();//getting Iterator from arraylist to traverse elements
14.        while(itr.hasNext()){
15.         System.out.println(itr.next());
16.    }
17.
18. }
```

Output:

```
Ravi
Vijay
Ravi
Ajay
```

User-defined class objects in Java ArrayList

```
1. class Student
2. {
3.     int rollno;
4.     String name;
5.     int age;
6.
7.     Student(int rollno,String name,int age)
8.     {
9.         this.rollno=rollno;
10.        this.name=name;
11.        this.age=age;
12.    }
13. }
```

```
1. import java.util.*;
2. public class TestCollection3
3. {
4.     public static void main(String args[])
5.     {
6.         //Creating user-defined class objects
7.         Student s1=new Student(101,"Sonoo",23);
8.         Student s2=new Student(102,"Ravi",21);
9.         Student s2=new Student(103,"Hanumat",25);
10.
11         ArrayList<Student> al=new ArrayList<Student>();//creating arraylist
12.        al.add(s1);//adding Student class object
13.        al.add(s2);
14.        al.add(s3);
15.
16.        Iterator itr=al.iterator();
17.        //traversing elements of ArrayList object
18.        while(itr.hasNext())
19.        {
20.            Student st=(Student)itr.next();
21.            System.out.println(st.rollno+" "+st.name+" "+st.age);
22.        }
23. }
24. }
```

Output:

```
101  Sonoo  23
102  Ravi  21
103  Hanumat  25
```

Example of addAll(Collection c) method

```
1. import java.util.*;
2. class TestCollection4
3. {
4.         public static void main(String args[])
5.         {
6.             ArrayList<String> al=new ArrayList<String>();
7.             al.add("Ravi");
8.             al.add("Vijay");
9.             al.add("Ajay");
10.
11.            ArrayList<String> al2=new ArrayList<String>();
12.            al2.add("Sonoo");
13.            al2.add("Hanumat");
14.            al.addAll(al2);
15.            Iterator itr=al.iterator();
16.            while(itr.hasNext())
17.            {
18.                System.out.println(itr.next());
19.            }
20.        }
21. }
```

Output:

```
Ravi
Vijay
Ajay
Sonoo
Hanumat
```

Example of removeAll() method

```
1. import java.util.*;
2. class TestCollection5
3. {
4.         public static void main(String args[])
5.         {
6.             ArrayList<String> al=new ArrayList<String>();
7.             al.add("Ravi");
8.             al.add("Vijay");
9.             al.add("Ajay");
10.
11.            ArrayList<String> al2=new ArrayList<String>();
12.            al2.add("Ravi");
13.            al2.add("Hanumat");
14.
15.            al.removeAll(al2);
16.
17.            System.out.println("iterating the elements after removing the elements of al2...");
18.            Iterator itr=al.iterator();
19.            while(itr.hasNext()){
20.                System.out.println(itr.next());
21.            }
22.        }
23. }
```

Output:

```
iterating the elements after removing the elements of al2...
Vijay
Ajay
```

Example of retainAll() method

```
1. import java.util.*;
2. class TestCollection6
3. {
4.         public static void main(String args[])
5.         {
6.             ArrayList<String> al=new ArrayList<String>();
7.             al.add("Ravi");
8.             al.add("Vijay");
9.             al.add("Ajay");
10.            ArrayList<String> al2=new ArrayList<String>();
11.            al2.add("Ravi");
12.            al2.add("Hanumat");
13.
14.            al.retainAll(al2);
15.
16.            System.out.println("iterating the elements after retaining the elements of al2...");
17.            Iterator itr=al.iterator();
18.
19.            while(itr.hasNext());
20.            {
21.                System.out.println(itr.next());
22.            }
23. }
24. }
```

Output:

```
iterating the elements after retaining the elements of al2...
Ravi
```

Java LinkedList class

- Java LinkedList class uses doubly linked list to store the elements. It extends the AbstractList class and implements List and Deque interfaces.
- Java LinkedList class can contain duplicate elements.
- Java LinkedList class maintains insertion order.
- Java LinkedList class is non synchronized.
- In Java LinkedList class, manipulation is fast because no shifting needs to be occurred.
- Java LinkedList class can be used as list, stack or queue.

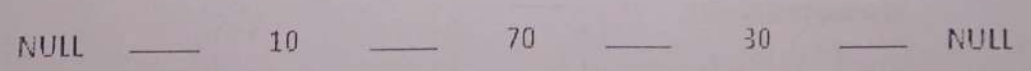


fig- doubly linked list

Java LinkedList Example

```
import java.util.*;
public class TestCollection7
{
        public static void main(String args[])
        {
            LinkedList<String> al=new LinkedList<String>();
            al.add("Ravi");
            al.add("Vijay");
            al.add("Ravi");
            al.add("Ajay");

            Iterator<String> itr=al.iterator();
            while(itr.hasNext()){
             System.out.println(itr.next());
            }
        }
}
```

Output: Ravi
Vijay
Ravi
Ajay

Example of ArrayList and LinkedList in Java

Let's see a simple example where we are using ArrayList and LinkedList both

```
import java.util.*;
class TestArrayLinked
{
        public static void main(String args[])
        {
            List<String> al=new ArrayList<String>();//creating arraylist
            al.add("Ravi");//adding object in arraylist
            al.add("Vijay");
            al.add("Ravi");
            al.add("Ajay");

            List<String> al2=new LinkedI ist<String>();//creating linkedlist
            al2.add("James");//adding object in linkedlist
            al2.add("Serena");
            al2.add("Swati");
            al2.add("Junaid");

            System.out.println("arraylist: "+al);
            System.out.println("linkedlist: "+al2);
        }
}
```

Output:
arraylist: [Ravi,Vijay,Ravi,Ajay]
linkedlist: [James,Serena,Swati,Junaid]

List Interface is the subinterface of Collection.It contains methods to insert and delete elements in index basis.It is a factory of ListIterator interface.

Commonly used methods of List Interface:

1. public void add(int index,Object element);
2. public boolean addAll(int index,Collection c);
3. public object get(int Index position);
4. public object set(int index,Object element);
5. public object remove(int index);
6. public ListIterator listIterator();
7. public ListIterator listIterator(int i);

Java ListIterator Interface

ListIterator Interface is used to traverse the element in backward and forward direction.

Commonly used methods of ListIterator Interface:

1. public boolean hasNext();
2. public Object next();
3. public boolean hasPrevious();
4. public Object previous();

Example of ListIterator Interface:

```
import java.util.*;
public class TestCollection8
{
    public static void main(String args[])
    {
        ArrayList<String> al=new ArrayList<String>();
        al.add("Amit");
        al.add("Vijay");
        al.add("Kumar");
        al.add(1,"Sachin");

        System.out.println("element at 2nd position: "+al.get(2));

        ListIterator<String> itr=al.listIterator();

        System.out.println("traversing elements in forward direction...");
        while(itr.hasNext()){
        System.out.println(itr.next());
        }
        System.out.println("traversing elements in backward direction...");
        while(itr.hasPrevious()){
        System.out.println(itr.previous());
    }
  }
}
```

Output:

```
element at 2nd position: Vijay
traversing elements in forward direction...
Amit
Sachin
Vijay
Kumar
traversing elements in backward direction...
Kumar
Vijay
Sachin
Amit
```

Difference between ArrayList and LinkedList

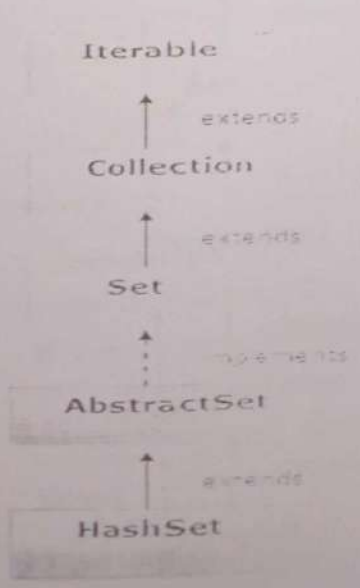
ArrayList and LinkedList both implements List interface and maintains insertion order. Both are non synchronized classes. But there are many differences between ArrayList and LinkedList classes that are given below.

ArrayList	LinkedList
1) ArrayList internally uses **dynamic array** to store the elements.	LinkedList internally uses **doubly linked list** to store the elements.
2) Manipulation with ArrayList is **slow** because it internally uses array. If any element is removed from the array, all the bits are shifted in memory.	Manipulation with LinkedList is **faster** than ArrayList because it uses doubly linked list so no bit shifting is required in memory.
3) ArrayList class can **act as a list** only because it implements List only.	LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces.
4) ArrayList is **better for storing and accessing** data.	LinkedList is **better for manipulating** data.

Java HashSet class: It uses hashtable to store the elements. It extends AbstractSet class and implements Set interface. It contains unique elements only.

Difference between List and Set:

List can contain duplicate elements whereas Set contains unique elements only.



Example of HashSet class:

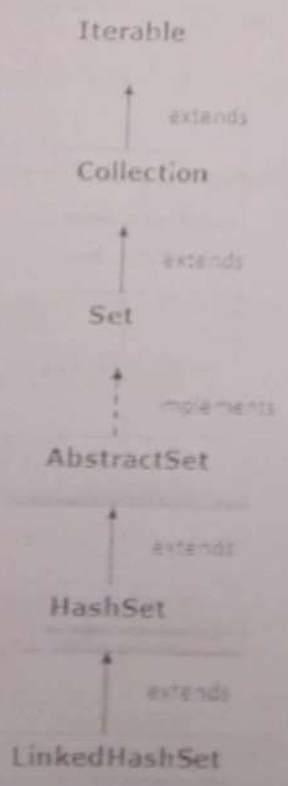
```
1. import java.util.*;
2. class TestCollection9
3. {
4.        public static void main(String args[])
5.        {
6.            HashSet<String> al=new HashSet<String>();
7.            al.add("Ravi");
8.            al.add("Vijay");
9.            al.add("Ravi");
10.           al.add("Ajay");
11.
12.           Iterator<String> itr=al.iterator();
13.           while(itr.hasNext()){
14.           System.out.println(itr.next());
15.           }
16.       }
17. }
```

Output:

```
Ajay
Vijay
Ravi
```

Java LinkedHashSet class:

- contains unique elements only like HashSet. It extends HashSet class and implements Set interface.
- maintains insertion order.



Example of LinkedHashSet class:

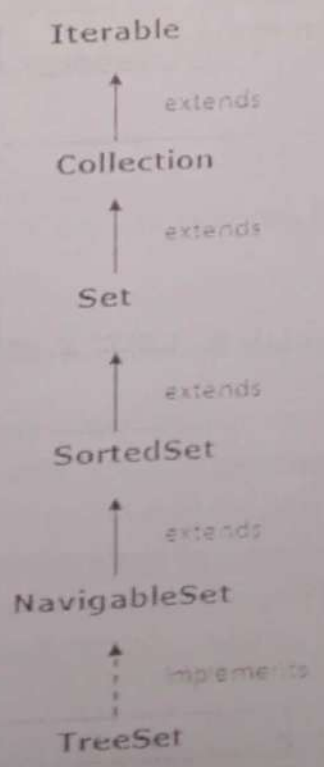
```
1. import java.util.*;
2. class TestCollection10
3. {
4.      public static void main(String args[])
5.      {
6.          LinkedHashSet<String> al=new LinkedHashSet<String>();
7.          al.add("Ravi");
8.          al.add("Vijay");
9.          al.add("Ravi");
10.         al.add("Ajay");
11.
12.         Iterator<String> itr=al.iterator();
13.         while(itr.hasNext()){
14.          System.out.println(itr.next());
15.         }
16.     }
17. }
```

```
Output:>Ravi
        Vijay
        Ajay
```

Java TreeSet class

- contains unique elements only like HashSet. The TreeSet class implements NavigableSet interface that extends the SortedSet interface.
- maintains ascending order.

Hierarchy of TreeSet class:



Example of TreeSet class:

```
1. import java.util.*;
2. class TestCollection11
3. {
4.     public static void main(String args[])
5.     {
6.         TreeSet<String> al=new TreeSet<String>();
7.         al.add("Ravi");
8.         al.add("Vijay");
9.         al.add("Ravi");
10.        al.add("Ajay");
11.
12.        Iterator<String> itr=al.iterator();
13.        while(itr.hasNext()){
14.         System.out.println(itr.next());
15.        }
16.    }
17. }
```

```
Output: Ajay
        Ravi
        Vijay
```

Java Queue Interface

The Queue interface basically orders the element in FIFO(First In First Out)manner.

Methods of Queue Interface :

```
1. public boolean add(object);
2. public boolean offer(object);
3. public remove();
4. public poll();
5. public element();
6. public peek();
```

PriorityQueue class:

The PriorityQueue class provides the facility of using queue. But it does not orders the elements in FIFO manner.

Example of PriorityQueue:

```
1. import java.util.*;
2. class TestCollection12
3. {
4.     public static void main(String args[])
5.     {
6.         PriorityQueue<String> queue=new PriorityQueue<String>();
7.         queue.add("Amit");
8.         queue.add("Vijay");
9.         queue.add("Karan");
10.        queue.add("Jai");
```

```
        queue.add("Rahul");
        System.out.println("head:"+queue.element());
        System.out.println("head:"+queue.peek());

        System.out.println("iterating the queue elements:");
        Iterator itr=queue.iterator();
        while(itr.hasNext()){
        System.out.println(itr.next());
        }

        queue.remove();
        queue.poll();

        System.out.println("after removing two elements:");
        Iterator<String> itr2=queue.iterator();
        while(itr2.hasNext()){
        System.out.println(itr2.next());
        }

    }
}
```

```
Output:head:Amit
       head:Amit
       iterating the queue elements:
       Amit
       Jai
       Karan
       Vijay
       Rahul
       after removing two elements:
       Karan
       Rahul
       Vijay
```

Java Map Interface

A map contains values based on the key i.e. key and value pair.Each pair is known as an entry.Map contains only unique elements.

Commonly used methods of Map interface:

1. **public Object put(object key,Object value):** is used to insert an entry in this map.
2. **public void putAll(Map map):**is used to insert the specified map in this map.
3. **public Object remove(object key):**is used to delete an entry for the specified key.
4. **public Object get(Object key):**is used to return the value for the specified key.
5. **public boolean containsKey(Object key):**is used to search the specified key from this map.
6. **public boolean containsValue(Object value):**is used to search the specified value from this map.
7. **public Set keySet():**returns the Set view containing all the keys.
8. **public Set entrySet():**returns the Set view containing all the keys and values.

Entry

Entry is the subinterface of Map.So we will access it by Map.Entry name.It provides methods to get key and value.

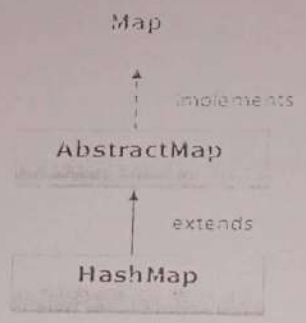
Methods of Entry interface:

1. **public Object getKey():** is used to obtain key.
2. **public Object getValue():**is used to obtain value.

Java HashMap class

- A HashMap contains values based on the key. It implements the Map interface and extends AbstractMap class.
- It contains only unique elements.
- It may have one null key and multiple null values.
- It maintains no order.

Hierarchy of HashMap class:



Example of HashMap class:

```
1. import java.util.*;
2. class TestCollection13
3. {
4.     public static void main(String args[])
5.     {
6.         HashMap<Integer,String> hm=new HashMap<Integer,String>();
7.
8.         hm.put(100,"Amit");
9.         hm.put(101,"Vijay");
10.        hm.put(102,"Rahul");
11.
12.        for(Map.Entry m:hm.entrySet()){
13.        System.out.println(m.getKey()+" "+m.getValue());
14.    }
15. }
16. }
```

Output:102 Rahul
100 Amit
101 Vijay

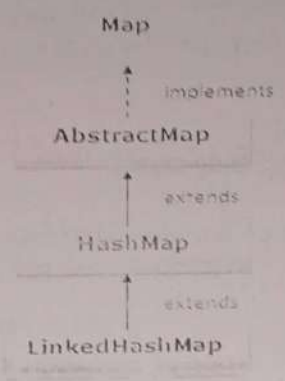
What is difference between HashSet and HashMap?

HashSet contains only values whereas HashMap contains entry(key and value).

Java LinkedHashMap class

- A LinkedHashMap contains values based on the key. It implements the Map interface and extends HashMap class.
- It contains only unique elements.
- It may have one null key and multiple null values.
- It is same as HashMap instead maintains insertion order.

Hierarchy of LinkedHashMap class:



Example of LinkedHashMap class:

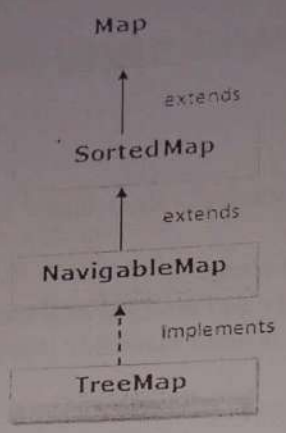
```
1. import java.util.*;
2. class TestCollection14
3. {
4.      public static void main(String args[])
5.      {
6.       LinkedHashMap<Integer,String> hm=new LinkedHashMap<Integer,String>();
7.       hm.put(100,"Amit");
8.       hm.put(101,"Vijay");
9.       hm.put(102,"Rahul");
10.
11.       for(Map.Entry m:hm.entrySet()){
12.          System.out.println(m.getKey()+" "+m.getValue());
13.         }
14.    }
15. }
```

Output:100 Amit
101 Vijay
103 Rahul

Java TreeMap class

- A TreeMap contains values based on the key. It implements the NavigableMap interface and extends AbstractMap class.
- It contains only unique elements.
- It cannot have null key but can have multiple null values.
- It is same as HashMap instead maintains ascending order.

Hierarchy of TreeMap class:



Example of TreeMap class:

```
1.  import java.util.*;
2.  class TestCollection15{
3.   public static void main(String args[])
4.  {
5.  TreeMap<Integer,String> hm= new TreeMap<Integer,String>();
6.
7.   hm.put(100,"Amit");
8.   hm.put(102,"Ravi");
9.   hm.put(101,"Vijay");
10.  hm.put(103,"Rahul");
11.
12.  for(Map.Entry m:hm.entrySet(){
13.   System.out.println(m.getKey()+" "+m.getValue());
14.  }
15. }
16. }
```

Output:100 Amit
101 Vijay
102 Ravi
103 Rahul

What is difference between HashMap and TreeMap?

1) HashMap is can contain one null key. TreeMap can not contain any null key.

2) HashMap maintains no order. TreeMap maintains ascending order.

Java Hashtable class

- A Hashtable is an array of list.Each list is known as a bucket.The position of bucket is identified by calling the hashcode() method.A Hashtable contains values based on the key. It implements the Map interface and extends Dictionary class.
- It contains only unique elements.
- It may have not have any null key or value.
- It is synchronized.

Example of Hashtable:

```
1. import java.util.*;
2. class TestCollection16
3. {
4.      public static void main(String args[])
5.      {
6.              Hashtable<Integer,String> hm=new Hashtable<Integer,String>();
7.
8.              hm.put(100,"Amit");
9.              hm.put(102,"Ravi");
10.             hm.put(101,"Vijay");
11.             hm.put(103,"Rahul");
12.
13.             for(Map.Entry m:hm.entrySet()){
14.              System.out.println(m.getKey()+" "+m.getValue());
15.     }
16. }
```

```
Output:103 Rahul
       102 Ravi
       101 Vijay
       100 Amit
```

Difference between HashMap and Hashtable

HashMap and Hashtable both are used to store data in key and value form. Both are using hashing technique to store unique keys.But there are many differences between HashMap and Hashtable classes that are given below.

HashMap	Hashtable
1) HashMap is **non synchronized**. It is not-thread safe and can't be shared between many threads without proper synchronization code.	Hashtable is **synchronized**. It is thread-safe and can be shared with many threads.
2) HashMap **allows one null key and multiple null values**.	Hashtable **doesn't allow any null key or value**.
3) HashMap is a **new class introduced in JDK 1.2**.	Hashtable is a **legacy class**.
4) HashMap is **fast**.	Hashtable is **slow**.
5) We can make the HashMap as synchronized by calling this code Map m = Collections.synchronizedMap(hashMap);	Hashtable is internally synchronized and can't be unsynchronized.
6) HashMap is **traversed by Iterator**.	Hashtable is **traversed by Enumerator and Iterator**.
7) Iterator in HashMap is **fail-fast**.	Enumerator in Hashtable is **not fail-fast**.
8) HashMap inherits **AbstractMap** class.	Hashtable inherits **Dictionary** class.

Sorting

We can sort the elements of:

1. String objects
2. Wrapper class objects
3. User-defined class objects

Collections class provides static methods for sorting the elements of collection.If collection elements are of Set type, we can use TreeSet.But We cannot sort the elements of List.Collections class provides methods for sorting the elements of List type elements.

Method of Collections class for sorting List elements

public void sort(List list): is used to sort the elements of List.List elements must be of Comparable type.

Note: String class and Wrapper classes implements the Comparable interface.So if you store the objects of string or wrapper classes, it will be Comparable.

Example of Sorting the elements of List that contains string objects

```
import java.util.*;
class TestSort1{
public static void main(String args[]){

ArrayList<String> al=new ArrayList<String>();
al.add("Viru");
al.add("Saurav");
al.add("Mukesh");
al.add("Tahir");

Collections.sort(al);
Iterator itr=al.iterator();
while(itr.hasNext()){
System.out.println(itr.next());
}
}
}
```

```
Output: Mukesh
        Saurav
        Tahir
        Viru
```

Example of Sorting the elements of List that contains Wrapper class objects

```
import java.util.*;
class TestSort2
{
    public static void main(String args[])
    {
      ArrayList al=new ArrayList();
      al.add(Integer.valueOf(201));
      al.add(Integer.valueOf(101));
      al.add(230);//internally will be converted into objects as Integer.valueOf(230)

      Collections.sort(al);

      Iterator itr=al.iterator();
      while(itr.hasNext()){
      System.out.println(itr.next());
      }
    }
}
```

Output:101
201
230

Comparable interface

Comparable interface is used to order the objects of user-defined class. This interface is found in java.lang package and contains only one method named compareTo(Object).It provide only single sorting sequence i.e. you can sort the elements on based on single datamember only. For instance it may be either rollno, name, age or anything else.

Syntax:

public int compareTo(Object obj): is used to compare the current object with the specified object.

We can sort the elements of:

1. String objects
2. Wrapper class objects
3. User-defined class objects

Collections class provides static methods for sorting the elements of collection.If collection elements are of Set type, we can use TreeSet.But We cannot sort the elements of List.Collections class provides methods for sorting the elements of List type elements

Method of Collections class for sorting List elements

public void sort(List list): is used to sort the elements of List. List elements must be of Comparable type.

Note: String class and Wrapper classes implements the Comparable interface. So if you store the objects of string or wrapper classes, it will be Comparable.

Example of Sorting the elements of List that contains user-defined class objects on age basis

Student.java

```
class Student implements Comparable
{
    int rollno;
    String name;
    int age;
    Student(int rollno,String name,int age){
    this.rollno=rollno;
    this.name=name;
    this.age=age;
}

public int compareTo(Object obj)
{
    Student st=(Student)obj;
    if(age==st.age)
    return 0;
    else if(age>st.age)
    return 1;
    else
    return -1;
}

}
```

```
1. import java.util.*;
2. import java.io.*;
3.
4. class TestSort3
5. {
6.     public static void main(String args[])
7.     {
8.
9.         ArrayList al=new ArrayList();
10.        al.add(new Student(101,"Vijay",23));
11.        al.add(new Student(106,"Ajay",27));
12         al.add(new Student(105,"Jai",21));
13.
14         Collections.sort(al);
15.        Iterator itr=al.iterator();
16.        while(itr.hasNext()){
17.        Student st=(Student)itr.next();
18.        System.out.println(st.rollno+""+st.name+""+st.age);
19.        }
20.    }
21. }
```

Output: 105 Jai 21
101 Vijay 23
106 Ajay 27

Difference between ArrayList and Vector

ArrayList and Vector both implements List interface and maintains insertion order.
But there are many differences between ArrayList and Vector classes that are given below.

ArrayList	Vector
1) ArrayList is **not synchronized**.	Vector is **synchronized**.
2) ArrayList **increments 50%** of current array size if number of element exceeds from its capacity.	Vector **increments 100%** means doubles the array size if total number of element exceeds than its capacity.
3) ArrayList is **not a legacy** class, it is introduced in JDK 1.2.	Vector is a **legacy** class.
4) ArrayList is **fast** because it is non-synchronized.	Vector is **slow** because it is synchronized i.e. in multithreading environment, it will hold the other threads in runnable or non-runnable state until current thread releases the lock of object.
5) ArrayList uses **Iterator** interface to traverse the elements.	Vector uses **Enumeration** interface to traverse the elements. But it can use Iterator also.

Example of Java ArrayList

Let's see a simple example where we are using ArrayList to store and traverse the elements.

```
1. import java.util.*;
2. class TestArrayList21
3. {
4.        public static void main(String args[])
5.        {
6.
7.         List<String> al=new ArrayList<String>();//creating arraylist
8.         al.add("Sonoo");//adding object in arraylist
9.         al.add("Michael");
10.        al.add("James");
11.        al.add("Andy");
12.        //traversing elements using Iterator
13.        Iterator itr=al.iterator();
14.        while(itr.hasNext()){
15.        System.out.println(itr.next());
16.        }
17.       }
18. }
```

Output:

```
Sonoo
Michael
James
Andy
```

Example of Java Vector

Let's see a simple example of java Vector class that uses Enumeration interface.

```
1. import java.util.*;
2. class TestVector1
3. {
4.        public static void main(String args[])
5.        {
6.         Vector<String> v=new Vector<String>();//creating vector
7.         v.add("umesh");//method of Collection
8.         v.addElement("irfan");//method of Vector
9.         v.addElement("kumar");
10.        //traversing elements using Enumeration
11.        Enumeration e=v.elements();
12.        while(e.hasMoreElements()){
13.         System.out.println(e.nextElement());
14.         }
15.       }
16. }
```

Output:

```
umesh
irfan
kumar
```

Programs

DEC 2007

1 Write an object oriented program (in terms of classes and objects) to evaluate the expression for standard deviation as given below: [10 M]

$$\left[\frac{\sum_{i=1}^{N}(\bar{X}-X_i)^2}{N}\right]^{1/2}$$

2 Write a program to arrange the names of students in descending order of their mark, input data consists of student details such as name, ID no, subject marks of Mathematics, Physics, Chemistry and total of these three. [20 M]

JUN 2008 [Total = 60 Marks]

1 Write an Object Oriented program in JAVA that uses Euclid's Algorithm to display the greatest common divisor of two integers. The greatest common divisor of two numbers is the largest number that divides into both numbers. Here is how the algorithm works?

(i) Find the reminder after dividing the larger number by the smaller number, using the modulus operator.

(ii) Change the larger number to the smaller number and the smaller number to the reminder from step (i).

(iii) Keep doing this until the reminder is zero.

Incorporate member functions for data input, displaying the result, default constructor and constructor with two parameters. Also create objects to reference the member functions. [10 M]

2 Develop a JAVA program that determines the number of days in a given semester. Input to the program is the year, month and day information of the first and last days of a semester. The program should accept the date information as a single string instead of accepting the year, month and day information separately. The input string must be MM/DD/YYYY format. [10 M]

3 Write a JAVA program to compute the distances S fallen by an object in freefall. The formula is:

$$S = S_0 + V_0 * t + \frac{1}{2} * a * t^2$$

Make table of S for t = 1, 5, 10, 15, 20, ... 100. [10 M]

4 Write a object oriented program to arrange the names of students in descending order of their mark, input data consists of student details such as name, ID no, marks of Mathematics, Physics, Chemistry. Use array of objects. [10 M]

5
```
Interface Matrix
{
        final static int M=5, N=5;//Matrix indices
        void readMatrix();//Read a Matrix
        void displayMatrix();//Display a Matrix
        void addMatrix();//Add two Matrix
        void multiMatrix();//Multiply two Matrix
        void transposeMatrix();//Transpose a Matrix
}
```

Implement the above interface using a suitable JAVA class program and also develop the main program. [20 M]

DEC 2008 [Total = 57 Marks]

1 Write a program in JAVA to find ${}^{n}C_{r}$ and ${}^{n}P_{r}$. [5 M]

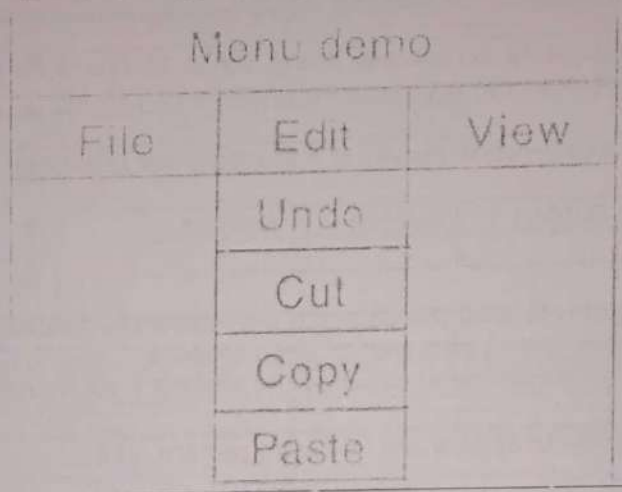
2 Write a program to determine the sum of the series for a given value of n. [5 M]
1 + 1/2 + 1/3 + ... + 1/n

3 Write a program in JAVA for fitting a straight line through set of points (x_i, y_i) = 1, ... , n. the straight line equation is y = mx + cand the values of m and c are given by [10 M]

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

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

4 Write a program to display the following menus and submenus. [12 M]



5 Assume that a bank maintains two kinds of account for its customers. The saving account provides compound interest and withdrawal facilities but no check book facility. The current account provides check book facility but no interest. Current account holders should also maintain the minimum balance and if the balance falls below this level, a service charge is imposed.
Cerate a class account that stores customer name, account number and type of account. From this derive classes current-acct and saving-acct to make them more specific to their requirements. Include the necessary methods to in order to achieve the following tasks:

(i) Accept deposit from a customer and update the balance. [10 M]
(ii) Display the balance.
(iii) Compute and deposit interest.
(iv) Permit withdrawal and update the balance.
(v) Check for the minimum balance, impose penalty, if necessary and update the balance.

6 Write a program to create multiple threads. [5 M]

7 Write a program to check whether given string is palindrome or not. [5 M]

8 Will the code compile? If not, why? If yes, write output. [5 M]

i)

```
import circle;

class Import Class
{
        public static void main(String args[])
        {
                circle.NewCircle nc = new  circle.NewCircle();
                System.out.println("Hello World ! "):

        }
}
```

ii)

```
int  m = 10;
int n = 7;
while (m % n >= 0)
{
        m =  m + 1;
        n = n + 2;
}
```

JUN 2009 [Total = 72 Marks]

1 Write a program to accept any string up to 15 characters. Display the elements of string with their element nos. Use toCharArray() functions. [10 M]

2 The annual examination results of 5 students are tabulated as follows:- [10]

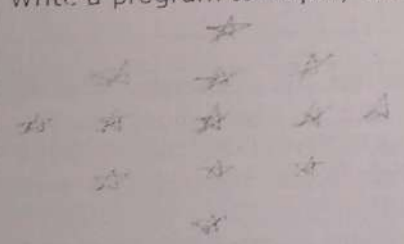
Roll No.	Subject 1	Subject 2	Subject 3

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

(i) Total marks obtained by each student

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

3 Write a program to display the following pattern. [10]



4 Write a program to display volume of sphere and hemisphere. Make use of abstract class. [10]

5 Write a program to illustrate use of isAlive() and join() and sleep() method. Create three classes A, B and C. Create another class "IaliveJoin" to invoke isAlive() and join() methods for three child thread. [12]

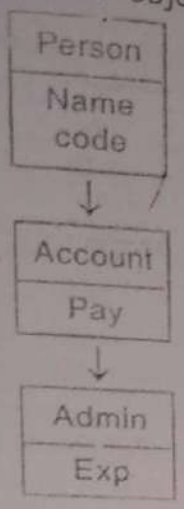
6 Write a program that accepts a shopping list of items and perform the following operations. [10]

(i) Add an item at specified location

(ii) Delete an item in the list

(iii) Print the contents of the vector.

7 Consider a class network given. The class 'Admin' derives the information from the class 'Account' which in turn derives information from the 'Person' class. Write a program to display 'Admin' object. [10]



DEC 2009 [Total = 70 Marks]

1 Write a program to print the following pattern. [10]

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

2 Write a program to find largest of three integers accepted from command line. [10]

3 Write a program to capitalize the first character of each word from a given sentence (example : all the best All The Best) [10]

4 Write a program to search an integer in an Array of integers. [10]

5 Write a program to accept student names form the command line and store them in vector. [10]

6 Write a program to print * / * / * / * / * / * / * / using two child thread. (hint: One thread responsible for printing * and another for /) [10]

7 Write an applet to display the following: [10]

JUN 2010 [Total = 55 Marks]

1 Write a java program to find factorial of given number. The number will be provided by the user at run time. [10]

2 Create a class employee with data members empid, empname, designation and salary. Write methods getEmployee() – To take user input, showGrade() – to display grade of employee based on salary. showEmployee() to display employee details. [10]

3 Write a program to find out number of upper case, lower case, blank spaces, number of digits from the string. [10]

4 Write a program to accept five integers from user and arrange in ascending order. [10]

5 Write a abstract class program to calculate area of circle, rectangle and triangle. [10]

6 Write a java applet to display "Hello java". [5]

DEC 2010 [Total = 80 Marks]

1 Create an applet which displays "All the best". [10]

2 Write a program that reads a 4 digit integer and breaks it in a sequence of individual digits e.g. 1691 should be display as 1 6 9 1. [10]

3 Write a program to print the following pattern. [10]

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

4 Write a program to find out number of upper case, lower case, blank spaces from the string. [10]

5 Write a program to determine the sum of the series: [10]

$$1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \ldots \frac{1}{n}$$

Take value n from the user.

6 Write a program that uses Vector class to add 5 names of student initially. Display a menu :-

(i) Add new name
(ii) Delete name
(iii) Display name

Based on the choice entered by the user take appropriate action. [10]

7 Write a program to display area of circle and triangle using the concept of abstract class. [10]

8 Write a program to create multiple threads. (One thread may display alphabets and another may display numbers. [10]

JUN 2011 [Total = 40 Marks]

1 Create 3 Vectors Bank, Deposit, Withdraw. Perform following operations: [20]

a) New customer – data should be added in bank vector.
b) Withdrawal of requested amount if the customer exists in Bank vector. The Account no., withdrawal amount and date should be added into withdrawal vector and amount should be deducted from balance in Bank vector.
c) Deposition of requested amount if the customer exists in Bank vector. The Account no.,

withdrawal amount and date should be added into Deposit vector and amount should be added to balance in Bank vector.

d) View the contents of an account.

2 Write a program in Java to accept a no and compute summation of digits recursively. [10]

3 Write a Java program to accept no from command line and print the sum of cube of individual digits. [10]

DEC 2011 [Total = 60 Marks]

1 Write a program to create multiple threads. Demostrate the use of isAlive(), setName(), getName(), setPriority(), and getPrriority() methods of the Thread class. [10]

2 Write a program to evaluate the sum of the following series: [10]

$$1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \dots \frac{1}{n^2}$$

Take value n from the user.

3 Write a program to read 5 names of students from the command line and store them in vector. Sort the list in alphabetical order and display. [10]

4 Write a program to count the frequency of occurrence of a given character in a given line of text. For example the line of text is: We are students and eat Eggs. [10]
The given character is: e
Then the frequency of occurrence is = 5.

5 Write a program to display the area of square and rectangle using the concept of overloaded constructor. [10]

6 Write a java Applet to display the house as shown below: [10]

JUN 2012 [Total = 70 Marks]

1 Write a program to evaluate sum of following series: [10]

$$1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \dots \frac{1}{n^2}$$

Take value of x and n as non zero-positive integers from user.

2 Write a program to replace all occurrences of a particular character in a line of text, with a new character. Take the input line and the two characters as a user input. For example if the text line is : keep our city clean [10]
The character 'e' to be replaced by '#' then the output is : k##p our city cl#an

3 Write an interactive program to create a Vector object and populate with names of cities. Program should display and implement following menu choices: [10]

(i) Add city – if city name already exists in the Vector then program should display appropriate message.
(ii) Remove city – if city does not exists in Vector then program should display appropriate message.
(iii) Display all – should display entire Vector content

4 Assume that a Bank maintains two kinds of accounts for its customers. One called savings account and the other current account. The savings account interest rate is 5% per year and for current account there is no interest. Savings account should hold minimum balance but there is no restriction on current account. Create an abstract class called Account that stores customer name, account number. Derive savings and current account classes from the Account class and add specific attribute to each of the derived classes. Write an interactive program to create specific type of account and it should allow following functionality: [10]
(i) Accept deposit and update the balance.
(ii) Compute and deposit interest.
(iii) Permit withdrawal and update the balance.
Each of these functionality should be coded in appropriate classes.

5 Write a java applet to display a circle, a line, and an arc inside the applet's visible area. [10]

6 Each post-paid telephone connection object stores its telephone number, customer name and address. The telephone company wants to keep tab of how many post paid connections have been sold. Write a program to implement the above requirement and display total post-paid connections sold by the company.
What additions will have to be made in post-paid telephone connection object to make the above possible? [10]

7 The mains voltage supplied by a sub-station is measured at hourly interval for 72 hours.
Write a program to display following:
(i) The recorded voltage for 72 hours
(ii) The hours (numbered 1 to 72) at which the recorded voltage varied from the mean by more than 10%.

DEC 2012 [Total = 75 Marks]

1 Write a program to demonstrate the use of command line Arguments. [5]

2 Write a program to add the list of items using Vector and delete an item in the list according to user's choice. [10]

3 Write a program to display following tree: [5]

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

4 Write a program to demonstrate access protection in different packages. [10]

5 Write a program to display whether the inputted number is prime or not. [10]

6 Write a program to create Thread by extending 'Thread class'. [5]

7 WAP to check the font name and font size of string by passing parameters to applet. [5]

[5]

8 Write a program to open "notepad" using Java application program. [5]

9 Write a program for matrix addition. [5]

10 Write a program to demonstrate try and multiple catches. [5]

11 Write a program to demonstrate overloaded constructor. [5]

12 Write a program to demonstrate single inheritance. [5]

MAY 2013 [Total = 55 Marks]

1 Write a program in Java to accept two integer numbers using command line arguments and print result of addition of both numbers. [5]

2 Write a program to determine sum of the series: [10]

$$1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \ldots \frac{1}{n}$$

3 Write a program to find sum of digits of a user entered number and print the sum. [10]

4 Write a program to find transpose of a matrix, and print the transposed matrix. [5]

If matrix $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$, then the transpose of $A = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$

5 Write a program to create and implement interfaces. [5]

6 Write a program to count the occurrence of a given character in a given line of text e.g. if the line of text is "All the best", and given character is 't', the count of 't' is 2. [10]

7 Write a program to create a class Account with data members customer-ID, customer-name, account-no, account-type (saving and current) and balance. Write methods get-Account() to take input from users and display-Account() to show account details. [10]