What *is Java?*

Java is a high level prgg. Lang. It was introduced by "**SUN MicroSystems" in June 1995. It was officially released in November 1995.** It was developed by ***a team under James Gosling***. Its original name was "**OAK**" meant for consumer electronic devices and later renamed to Java. Java has become the standard for Internet applications. It provides interactive processing and easy use of graphics and animation on the Internet.

Since the Internet consists of different types of computers and operating systems. A common lang. needed to enable computers. To run prgs that run on multiple plot forms. This need was fulfilled by Java, and it so on. Because the lang.of choice for the Internet.

Java is Object-Oriented lang.built on C and C++. It derives its syntax from C and its Object-Oriented features are influenced by C++.

Java can be used to create two types of prgs. Those are **Applications** and **Applets**.

An application is a prg.that runs on the user's computers under the operating system. An Applet is a small window based prg.that runs on HTML page using a java enabled web browser like internet Explorer, Netscape Navigator or an Applet Viewer.

Features of Java

1. Simple
2. Object Oriented
3. Plot form independent
4. Robust
5. Secure
6. Distributed
7. Multithreaded
8. Dynamic

1. **Simple :** Java follows the syntax of C and Object Oriented principles of C++. It eliminates the complexities of C and C++. Therefore, Java has been made simple.

2. **Object-Oriented:** It is an Object-Oriented prgg.lang. The object model in Java is simple. Although influenced by its procedures. It was designed to be source code compatible with any other lang.

3. **Plot form Independent:** Plot form is the combination of operating system and microprocessor. Java prg.works in all plot forms. It is achieved by JVM(Java Virtual Machine). The philosophy of java is "***Write Once, Run any where***" (WORA).

4. **Robust:** Java is strictly a typed lang. It has both a compiler and an interpreter. Compiler checks code at run time and garbage collection is taking care of by java automatically. Thus it is a robust lang.

5. **Secure:** java developers have taken all care to make it a secure prgg.lang. For Ex. Java Applets are restricted from Disk I/O of local machine.

6. **Distributed:** Java is designed for the distributed environment of the Internet. It handles TCP/IP protocols. Java's remote method invocation (RMI) make distributed prgg. Possible.

7. **Multithreaded:** Java was designed to meet the real-world requirement. To achieve this, java supports multithreaded prgg. It is the ability to run any things simultaneously.

8. **Dynamic:** Java prgrs.carry with them a substantial amount of run-time type information. That is run time. This makes it possible to dynamically link code in a safe and secure manner.

The Java Prgg. Environment

Two types of prgs that can be developed in java are Applets and Applications. Applets are java prgs. That run as part of a web page and they depend on a web browser in order to run. Applications are prg.that are stored on the user's system and they do not need a web browser in order to run.

The Java prgg environment includes a number of development tools to develop applet and application. The development tools are part of the system known as "***Java Development Kit***" or "***JDK***". The JDK include the flg.

1. Packages that contain classes.
2. Compiler
3. Debugger

JDK provides tools in the **bin** directory of JDK and they are as follows.

Javac : Javac is the java compiler that translates the source code to byte codes. That is, it converts the source file. Namely the **.java** file to **.class** file.

Java : The java interpreter which runs applets and Applications by reading and interpreting the byte code files. That is, it executes the **.class** file.

Javadoc : Javadoc is the utility used to produce documentation for the classes from the source code files.

JDB: JDB is a debugging tool.

The way these tools are applied to build and run application programs is as follows.

The source code file is created using a text editor and saved with a **.java** (with Extension).

The source code is compiled using the java compiler **javac**. Which translates source code to byte codes.

The compiled code is executed using the java interpreter **java**. Which runs applications and applets by reading the byte codes. Byte codes are instructions that are generated for a virtual machine. A ***virtual machine is a prg.*** that processes these generalized instructions to machine specific code.

JVM

Java is both ***compiled*** and an ***interpreted*** lang. First the java compiler translates source code into the byte code instructions. In the next stage the java interpreter converts the byte code instructions to machine code. That can be directly executed by the machine running the java prg.

The intermediate code namely the bytecode produced by the java compiler is for a machine that exists only on the memory of a computer. This machine or the '***Simuloated computer within the computer***' is known as the "***Java Virtual Machine***" or **JVM**.

The JVM can be thought as a mini operating system that forms a layer of abstraction where the underlying hardware. The operating system and the compiled code is concerned the compiler converts the source code into a

code that is based on the imaginary computer instructions set. The compiled code is not processor specific. The interpreter converts these instructions for the underlying hardware to which the interpreter is targeted. The portability feature of the **.class** file helps in the execution of the prg on any computer with the java virtual machine. This helps in implementing the "***write once and run any where***" feature of java.

The lifetime of a JVM

The runtime instance of the java virtual machine has a clear mission in life. To run one java application. When a java application starts a run time instance is ban when the application completes the instance dies. If you start three java applications at the same time on the same computer using the same can create and implementation. You will get three JVM instances. Each java application runs inside its own java virtual machine.

A java virtual machine instance starts running its solitary application by invoking the main() method of some initial class. The main() method must be public, static, return void and accept one parameter: a string array. Any class with such a main() method can be used as the starting point for a java application.

Java Runtime Environment

JRE consists of the java virtual machine. The java plot form core classes and supporting files. It is the run time part of the java Development Kit. No compiler, no debugger, no tools. The JRE is the smallest set of executables and files that constitute the standard java plot form.

JRE consists of the JVM interacting with hardware on one side and the JVM and the prg on the other. The java runtime environment runs code compiled for the JVM by:

Coding the .class files

Performed by the 'class loader'.

The class loader performs security checks here if the ***.classes*** are required across the network.

Verifying bytecode.

Performed by the 'bytecode verifier'.

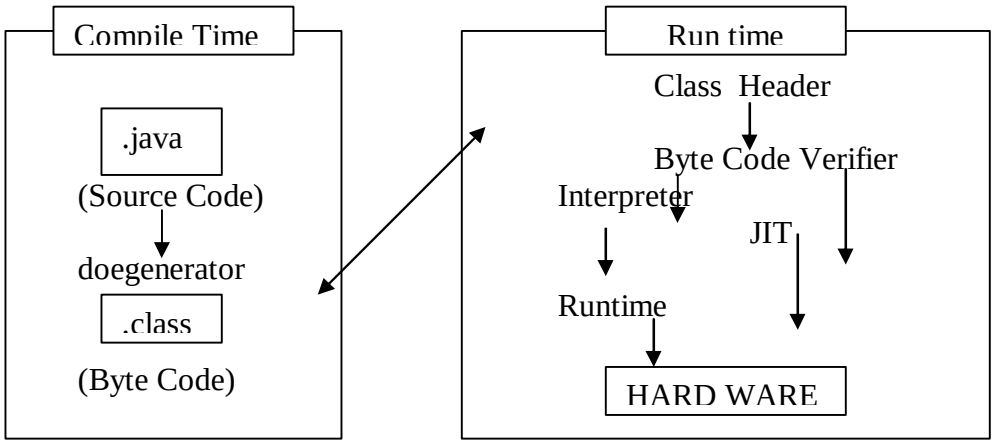
The bytecode verifier verifies the format of the code object type conversions and checks for the violation of access rights.

Executing the code

Performed by the runtime interpreter.

The interpreter executes the bytecode and makes calls to the underlying hard.

The flg. Fig.illustrates the above steps in sequence.



From source code to executable.

JIT

In the java prgg.lang. and environment a Just – In – Time (JIT) compiler is a prg that runs java bytecode (a prg that contains instructions that must be interpreted) into instructions that can be sent directly to the processor. (machine code or native code).

After you have written a java prg the source code is compiled to byte code by JVM. When a JIT is present the JVM does something different. After reading in the ***.class*** file for interpreter locations, it hands the ***.class*** file to the JIT. The JIT will take the byte code and compiled them into native code for the machine that you are running on. It is faster to grab the bytecodes compiled them and run the resulting executable with JIT than with just interpreter them. The JIT is an integral part of the JVM. So you nicer notice its there, except your, java runs faster. Some environments allow you to choose whether or not to JIT code.

Java is dynamic lang. So you are not allowed to "Statistically" compile all the ***.class*** files into machine code until they are actually called. Thus the JIT is really "***just in time***". Since it compiles methods on a methods called. If you call the same method more than once, the JIT code can really pay off as you do not have to re-JIT the method and can simply re-execute the native code.

Steps to execute a java prg:

- ? Open any editor such as notepad
- ? Type the source code
- ? Save it with **.java** extension (File Name and class name must be same)
- ? Compile it
- ? Run it

Steps to compile a java prg:

- ? Open Dos prompt
- ? Install ***DOSKEY***
- ? Set the path
  Ex: path=**c:\jdk1.2.2\bin** and press enter key
- ? Move to your working directory/folder
- ? Compile the prg
  Ex: **javac filename.java**
- ? Run the prg
- ? Ex: **java filename**

Note: jdk1.2.2 is the java's installation directory. Some times it could be jdk1.3 or other.

Doskey: it is a dos command, which maintain a history of commands given by user. User can simply scroll through and use those commands by pressing up arrow and down arrow.

In the Java prg:

public: It indicates that main() can be called outside the class.

static: It is an access specifier, which indicates that main() can be called directly without creating an object to the class.

void: It indicates that the method main() doesn't return a value.

main(): It is a method which is an entry point into the java prg. When you run a java prg.main() is called first.

String args* [ ]*: String is a class, which belongs to **java.lang** package. It can be used as a string data type in java.

args*[]*: It is an array of string type. It is used to store command line args.

System: It is a class, which belongs to **java.lang** package.

out: It is an output stream object, which is a member of System class.

println(): It is a method supported by the output stream object "**out**". It is used to display any kind of output on the screen. It gives a new line after printing the output.

print(): It is similar to println(). But doesn't give a new line after printing the output.

Control Statements:

Def: To control the flow of execution of sequential instructions in a prg. Those are called Control statements. Those are,

1. Conditional statements
2. Looping statements
3. Jumping statements

1. ***Conditional statements:***
    a. If statement
    b. Switch statement
2. ***Looping statements:*** The block of statements are executed specified no. of times or until the condition is unsatisfied. Those are called Looping statements.
    a. For
    b. While
    c. Do-while
3. ***Jumping statements:*** The control transfers from one location to another in a prg. Those are called jumping statements.
    a. Break
    b. Continue
    c. Return
    d. Goto

Comments :

There are 3 types of comments defined by java. Those are
Single line comment, multilane comment and the third type is called *documentation comment*. This type of comment is used to produce an HTML file that documents your prg.

// this is single line comment.

/* this multiple
line comment*/

/** this documentation comment */

Key words:

Keywords are the words. Those have specifics meaning in the compiler. Those are called ***keywords***. There are 49 reserved keywords currently defined in the java language. These keywords cannot be used as names for a variable, class or method. Those are,

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

Data types :

Def : The data, which gives to the computer in different types are called Data Types.

Java defines 8 types of data: ***byte, short, int, long, float, double, char*** *and* ***boolean***. These can be put in 4 types.

Integer: this group includes ***byte, short, int*** and ***long,*** which are whole valued signed numbers.
Floating-point numbers: This group includes ***float*** and ***double***, which represents numbers with fractional precision.

Character: this group includes ***char***, which represents symbols in a character set, like letters and numbers.
Boolean: this group includes ***boolean***, which is a special type for representing *true / false* values.

Name	***width(Bytes)***	***Range***
long	8	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807
int	4	-2147483648 to +2147483647
short	2	-32,768 to +32,767
byte	1	-128 to 127
double	8	$-3.4*e^{308}$ to $3.4*e^{308}$
float	4	$-1.7*e^{38}$ to $1.7*e^{38}$
char	2	0 to 65,536
boolean	bit	0 or 1.

Operators:

Def: Operator means a special symbol. That can perform a specific operation. There are 4 types of operators.
Those are ***arithmetic, relational, bitwise*** **and** ***Boolean logical*** operators. And ***assignment*** operator, ***?*** operators are used.

1. Arithmetic operators:
   +, -, *, / , %, ++, --, +=, -=, *=, /=, %=.
2. Relational operators:
   >, >=, <, <= , = = , !=
3. Bitwise Operators:
   ~, ^, &, |, >>, >>>, <<, &=, |=, ^=, <<=,>>=, >>>= .
4. Boolean logical operators
   &, |, ^, ||, &&, !, &= ,|=, ^=, == ,!=, ? : .

Reading input from reader:

It is in two types. Those are,
1. Command line arguments
2. By using readLine() method

Through Command line arguments : (used rarely)

Arguments can be passed to a java prg at the time of running it. Such arguments are known as command line arguments. Those arguments are stored in the string array which is a parameter of main().

By using readLine() method:

Integer : It is a class belongs to ***java.lang*** package.
parseInt(): It is a method. It is supported by the class ***Integer***. It is used to convert a number from string format to integer format.

readLine():

The method **readLine()** is used to take input from user at runtime. This method is defined in the class ***DataInputStream***. This class is available in **java.io** package. By default this method reads any kind of input in String format. That should be converted to the required types.
readLine() methods throws an *IOException* should be handled or throws class has to be used.

java.io :

It is a package. It contains various classes required for input and output opertions.

DataInputStream,
DataOutputStream,
FileInputStream

IOException:

It is an Exception class. It is a part of java.io packages. It is required to be caught in every method when readLine() is used. Else throws class is used to mention IOException.

System.in:

It is parameter of the *constructor(*new*)* of *DataInputStream*. It indicates **standard input**.i.e. Key board.
'**in**' is an object of DataInputStream class. It is required in order to call readLine() method.

Naming conversions in java:

Class :

If class name is a single word, the first letter should be in uppercase. If it is a multiple word class, the first letter of every word should be in uppercase.

Ex: Student
EmployeeDetails etc.

Method/Function:

If method is a single word, it should be in lowercase. If it is a multiple word, the first word is in lowercase and from second word onwards first letter of every word should be in uppercase.

Ex: Read()
readLine()
parseInt()
itemValueChanged() etc.

Data Members(variables)

Naming conversions to variables is similar to that of methods.

Eg: marks
studentMarks
studentAverageMarks
rateOfInterest etc.

Note: If a word have bracket that is called Method or Data Member.

Arrays:

Collection of similar data elements and having a same name. Its elements can be accessed by using an index or subscribed number which always start from zero / zero based index.

Types of Arrays :

1) Single dimensional array
2) Multi dimensional array

Single dimensional array : it is an array with only one dimension. It can be visualized as a single row or a column.

Declaration :

```
Datatype variable[]=new datatype[size];

Or

Datatype variable[];
Variable=new datatype[size];
```

Two dimensional array : It is an array with two dimensions. It can be visualized as a matrix of rows and columns.

Declaration:

```
Datatype variable[][]=new datatype[rows][col];
```

Or
Datatype variable[][]
Variable=new datatype[row][col];

Sorting:

Sorting is a process of rearranging array elements (ascending or descending). The following are the sorting techniques.

1. ***Selection sort***
2. ***Bubble sort***

1. ***Selection sort:*** In this technique, the first element is selected and compared with all other elements. If any other element is less than the first element swapping should take place.

   By the end of this comparison, the least element most top position in the array. This is known as pass1.

   In pass II, the second element is selected and compared with all other elements. Swapping takes place if any other element is less than selected element. This process continuous until array is sorted.

   The no. of passes in array compare to size of array –1.

2. ***Bubble sort:*** This technique compares last element with the preceding element. If the last element is less than that of preceding element swapping takes place.

   Then the preceding element is compared with that previous element. This process continuous until the II and I elements are compared with each other. This is known as pass 1.

   This way the number of passes would be equal to size of array –1.

Class :

Syntax :

```
import statements ;
class classname
{
        data members / variables;
```

```
        methods / functions ;
}
```

syntax to creating an object :

```
classname objname = new classname ( ) ;
or
classname objname ;
objname = new classname ( ) ;
```

syntax to creating an array of objects :

```
classname arrayobj[] = new classname [ size];
```

---

constructors

1. It is a member function, whose name is same name of class.
2. It can be with or without parameters.
3. It has no return type.
4. It is used to construct an object and it is used to initialize data members.
5. It is also used to allocate memory to data members
6. It is invoked implicitly. Whenever an object is created.

   Implicitly ✍ Automatically
   Explicitly ✍ done by user

Syntax :

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

Destructor:

1. It is a member function whose name is same as the class. But preceded by a ~ (tilde) symbol.
2. It has no return type.
3. It cannot have parameters.

4. It is implicitly called when object is no longer required.
5. It is used to release the memory and close files and database conversions.

(No Destructor concept in java)

Note :- ***java does not support destructor. But it is supported by C,C++.***

Garbage collector:

It is a prg which will destroy object and release memory when they are no longer required.

The process of releasing memory occupied by objects, files and database connections are known as Garbage collector.

In java prg has to do nothing to deallocate memory.
If prgr wants to do garbage collector manually before an object is destroyed. He has to ***override the finalize*** method. The general form is as follows.

```
protected void finalize()
{
--------------
--------------
}
```

This method may be called before an object is Garbage collected. The things to be done before that can be coded in this method.

By writing the following lines of code, garbage collected can be subjected to do its job. But, objects are destroyed by garbage collector only when they are not required in memory.

Nameless object:

We can create a nameless object, when no methods are to be called explicitly. But it invoke constructor.

Syn :

```
new  classname() ; or new classname(parameters);
```

Method overloading:

Polymorphism in java can be implemented by using method overloading & method over riding.
Method overloading is a process of defining many methods with the same name. But parameter must be different. A method acts upon different sets of parameters. But, basic operation is same.

Advantages :

1. when many methods are performing same operations. It would be easy to call same method with different types of arguments.
2. we can overriding many methods and remember them for the same operation.

Call by Value:

When values of built in types are passed as arguments to a function. It is known as Call by value. The changes made to formal parameters in the called function are not reflected in the actual arguments class.

Call by Reference:

In this approach, an object is passed as an argument, which is assigned to an object reference are directly reflected to actual argument.

Access Specifiers:

Access Specifiers are the keywords that alter the meaning and accessibility of members of a class. They are,

private	***public***	***protected***	***static***
final	***abstract***	***native***	***synchronized***
transient	***volatile***		

private:

It can be applied to *variables* and *methods*. Private members of a class are accessible from within the class only. They cannot be accessed from outside the class and its subclass.

public:

It can be applied to *variables* (data members), *methods* and *classes*. Public member of class can be accessed globally any other code. They can be accessed from outside the class and sub class.

A public class can be accessed from any package.

protected:

It can be applied to *data members* and *methods*. Protected member of a class can be accessed from within the class, sub class in the same package and sub class in different package. It cannot be accessed from a non-subclass in different package.

Note: When no access specifier is mentioned for a data member, method or class. It assumes default / friend. There is no keywords such as default or friend. The default / friend members of a class can be accessed from anywhere except from sub class in different package and non-sub class from different package.

static : **(constant / fixed)**

It can be applied to data members, methods and some inner classes.

static data members :

1. static data members can be accessed by only static methods as well as non static.
2. static data members are not instance – specific. They are common to all objects of a class. Therefore, they are known as class variables.
3. They are not destroyed between function calls.
4. They can be accessed directly with the flg. Syn.
   classname.variblename ;
5. for all objects of a class, only one copy of static members exists in memory and it is shared by all objects. It is contrary to instance variables.

6. static variables are declared as .flg.
   (access specifier) **static int x ;**

static methods :

1. static method is a method whose definition is preceded by keyword **static.**
2. static methods can act upon only static variables.
3. static method can be called directly using class name without creating object.
4. The keyword "this" can't be used inside a static method, as static members don't belongs to any particular object.
5. A static method can call only other static methods

static block:

1. It is a piece of code enclosed in braces preceded by the keyword static.
2. static block is executed even before main() .
3. it is used to initialize some variables or any such things.
4. it can use only static variables & call only static methods.

Syn:

```
static
{
    statements;
    ------------
 }
```

final : (last / concluding)

This modifier can be applied to variables, methods and classes.

final data members:

Data member with final modifier becomes a constant.

final Method:

1. It is a method whose definition is preceded by keyword "final".

final returntype funname()
{
}

2. It can't be redefined in the sub class. This means that, a sub class can't override it.

final class:

1. It is a class whose definition is preceded by "final".
2. The final class can't be sub classed or derived. This means that we can't create a sub class from a final class.

```
// no inheritance
final class A
{
}

class  B extends A
{
}
```

The above code gives compile error. Because class A is a final class, which can't be derived.

abstract: **(conceptual / theoretical)**

It can be applied to methods and classes. It cannot be used with data members.

abstract Method:

1. It is a method, which has no body (definition) in the base class. The body should be implemented in the sub class only.
2. Any class with at least one abstract method should be declared as abstract.
3. If the sub class is not implementing an abstract method, its sub class has to implement it.

abstract Class:

1. Abstract class is a class that cannot be instantiated. (no creation of object).
2. It can be / must be inherited.

3. It can act as only base class in inheritance hierarchy. (levels)

concrete class: (real / actual)

Any class that can be instantiated is known as concrete class. All non-abstract class is nothing but concrete classes.

Note:

With respect to a class, abstract and final are exclusive. They both cannot be applied to a class of a time.

```
abstract final class A
{
}
```

(The above code is wrong)

native : (resident / local)

- ? It can be applied to only methods are used to implement some platforms dependent code in java.
- ? It does mean that functions defined in languages other than java such as C, C++ can be used in java using native method.
- ? Internet explorer has inbuilt JVM

synchronized: (matched / corresponding / harmonized)

It is used in multi-threaded programming. It allows only one thread to execute at a time.

serialization: It is a process of sending objects to a file to make them persistent (constant or permanent).

transient: (passing / temporary)

It can only be applied to data members. The transient variables will not be part of objects persistent state.

Ex : transient int netsal;

volatile: (unstable)

It can be applied to the variables only.

Ex : volatile int x;

The var. x is used in multiprocessing environment. When a processor requires it. It has to make copy of x and used it.
(Volatile is temporary)

strictfp:

It is used to only methods. It can't be applied to variables and classes. It is added by JDK1.5. When it is applied to a method that method follows IEEE. Floating point standard to process floats. IEEE stands for *Institute of Electrical and Electronic Engrs.*

Applicable of Access specifiers

	Variable	Method	Class
private	Y	Y	N
public	Y	Y	Y
protected	Y	Y	N
default	Y	Y	Y
static	Y	Y	Y
final	Y	Y	Y
abstract *	N	Y	Y
native	N	Y	N
synchronized	N	Y	N
transient	Y	N	N
volatile	Y	N	N
strictfp	N	Y	N

Inheritance:

Inheritance is a process of deriving a new class from existing class. The new class gets properties and methods inherited from

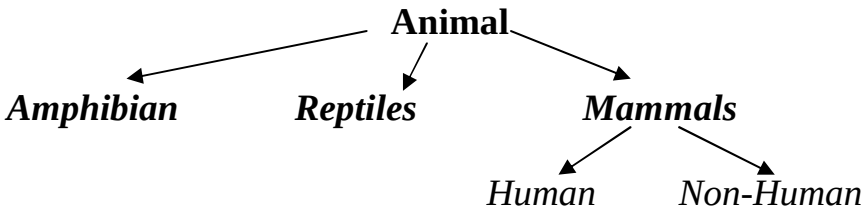
existing class. The greatest advantage of inheritance is "*reusability of code*".

Base class:

It is a class from which a new class inherits properties and methods. It is also known as *base class* or *parent class.*

Sub class:

It is a class, which is derived from an existing class. It is class known as child class or derived class.



In the above inheritance hierarchy the base class animal contains properties and methods that are common for all animals.

The sub class like Amphibians, Reptiles etc have same additional qualities in addition to the one's derived from Animal class.

Types of Inheritance :

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

1. Single Inheritance:

It is a process of deriving a new class from a single base class.

```
Eg.   Class A
      {
      statements ;
      }
      class B extends A
```

```
{
statements ;
}
```

2.Multiple inheritance:

It is a process of deriving a new class from more than one base class.

Eg.

```
Class  A
    {
        statements ;
    }
 class B
 {
  statements ;
 }
 class  C extends  A, B
{
 statements ;
}
```

Note:

Java does not support ***multiple inheritance*** with classes. But, it supports multiple inheritance using ***interface***s.

3. Multi level inheritance :

It is a process of deriving a new class from another deriving class.

Eg.

```
class A
{       }
class B extends A
{       }
class C extends B
{       }
```

4. Hierarchical Inheritance :

It is a process of deriving more than one class from a single base class.

Eg.

```
Class A
{
}

class B extends A
{       }
Class C extends A
{       }
```

Constructors and Inheritance:

When both super and sub classes are having a constructor, on creating an object for the subclass, base class constructor is called first then subclasses constructors.

Eg.

```
Class base
{
        public base()
        {
        statements ;
        }
}

class derived extends base
{
        public derived()
        {
        statements ;
        }
}

class constructDemo
{
 public static void main(String args[])
{
base b=new base();
}       }
```

Method overriding :

It is a process of redefining a method of super class in sub class. Thus one method can have many implementations.

Eg.

```
Class sample1
{
        public void display()
        {       }
}

class sample2 extends sample1
{
        public void display()
        {       }
}

class overriding
{
        public static void main(String aragsp[])
        {
        sample2  s2 = new sample2();
        s2.display();
        super.display(); // calling the display() in sample1 class.
        }
}
```

out put :

The display() in sample2 class is calling only.
The display() in sample1 class is calling.

Accessing methods of sub class through super class object.

It is possible to access methods of sub class by assigning sub class's object to base class reference.

Eg.

```
        Class emp
        {       }
        class manager extends emp
        {       }
```

```
class officer
{
p. s. v. m. (String args[])
{
emp e= new manager();
e.read();
e.display();
}
}
```

Dynamic method dispatch: (Dynamic Binding)

In java, base class reference can be assigned objects of sub class. When methods of sub class object are called through base class's reference. The mapping / binding or function calls to respective function takes place after running the program, at least possible moment. This kind of binding is known as "***late binding***" or "***dynamic binding***" or "***runtime polymorphism***".

Eg.

```
class A
{
        public void display()
        {       }
}

class B extends A
{
        public void display()
        {       }
}

class C extends A
{
        public void display()
        {       }
}

class DispatchDemo
{
        psvm (String  args[])
```

```
        {
        A ob1= new A();
        B ob2 = new B();
        C ob3 = new C();
        A r;

        r=ob1;
        r.display();

        r=ob2;
        r.display();

        r=ob3;
        r.display();
        }
}
```

In the above programs, the statement ***r.display( ),*** calls the display() method and based on the object. That has been assigned to base class reference. i.e. **r**. But the mapping / binding as to which method is to be invoked is done only at run time.

Interface:

Def: It is a collection of ***final data members*** and ***abstract methods.***

It is nothing but a pure abstract class. It has only abstract methods.

Syn:

```
        interface      Interfacename
        {
        data members ; // final data members
        methods() ; // abstract methods
        }

        class Classname implements Interfacename
        {
        data members ;
        methods()
```

```
{
statements ;
}
……
}
```

- **A class can't inherit an interface:**

```
interface A
{       }
class  B extends A
{       }
```

- **A class can only implement an interface:**

```
interface A
{       }
class  B implements A
{       }
```

- **A class can inherit another class and implement an interface:**

```
interface A
{       }
class  B
{       }
class C  extends B implements A
{       }
```

- **An interface can extend one or more interfaces:**

```
interface A
{       }
interface  B
{       }
interface  C  extends  A, B
```

```
{       }
class D implements C
{       }
```

? **An interface can't implement another interface:**

```
interface A
{       }
interface B implements A
{       }
```

? **A class can implement more than one interface:**

```
interface A
{       }
interface  B
{       }
class C  extends  A, B
{       }
```

Packages:

Package is a collection of related classes, interfaces and sub packages. Package in another package is called sub package.

Ex:

Java.lang
Java.io
Java.sql
Java.util
Java.security
Java.net
Java.awt
Java.applet etc.

Sub package: It is a package within another package.
Ex: java.awt.event

Javax.swing.text
Javax.servlet.http
Java.lang.reflection etc.

Advantages :

1. Better organization of classes or interfaces
2. Naming conflict can be avoided.

Importing a package:

Syn.

import packagename.classname;

Or

import packagename . * ;

Eg.

import java.io.DataInputStream ;

(import only specified class)

import java.io.* ;

(import all classes and interfaces)

Creating a user-defined package:

Syn.

package PACKAGENAME ;

public class CLASSNAME

{

statements ;

}

Compilation:

syn.

javac –d *path* filename. java

Or

Javac -d . filename.java

Here, **.** represent current location.

Class Member Access :

	Private	*default*	*protected*	*public*

Same class	Y	Y	Y	Y
Same package Sub class	N	Y	Y	Y
Same package Non-sub class	N	Y	Y	Y
Different package Sub class	N	N	Y	Y
Different package Non-sub class	N	N	N	Y

Exception Handling

No matter how good a program we write it may cause the following types of errors.

1. *Syntax Errors*
2. *Logical Errors*
3. *Runtime Errors*

Syntax Errors **:** It is an error which occurs when the rules of language are violated . If we are not following the rules, those cause the error.

Eg.

Statement missing semicolon ; .
Missing / expected '}'
Un-defined symbol varname etc.

Logical Error: It is an error caused due to manual mistake in the logical expressions. These errors can not be detected by the compiler.

Eg.

Netsal = (basic+da+hra) (+) pf

Runtime Error: It is an Error, which occurs after existing the program at runtime.

Eg.

Divide by zero.
File not found
No disc in the drive etc.

Exception:

It is nothing but runtime error. It can be handled to provide a suitable message to user.

Exception Handler:

It is a piece of code, it can be handle an exception.

Exception Handling Mechanism in Java:

Java uses a mechanism or model to handle exceptions. This model is known as "***catch and throw model***". This means that "***it identifies an exception and throws it and it will be caught by exception handler***". This model used the following keywords.

try ***catch*** ***throw*** ***throws*** ***finally***

Try block:

All statements that are likely to raise an exception at run time are kept in try block. This block will detect an exception and throws it. It will be handled by catch block.

Catch:

This block takes corrective steps when an exception is thrown. It provides a suitable message to the user. Then user will be able to proceed with the application.

A try block can follow multiple catch blocks.

Throw:

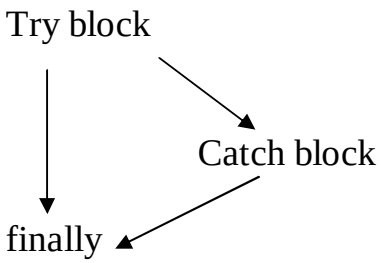
It is used to throw an exception.

Throws:

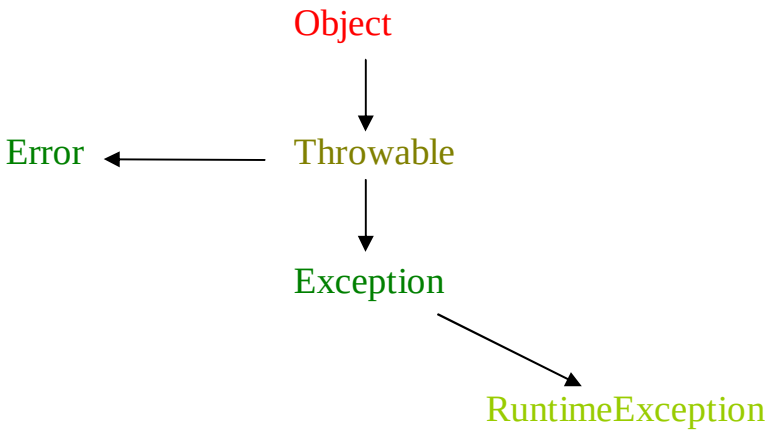
It is used to specify the possible exceptions that are caused by a method. But not handled by the method.

Finally:

This block is followed by catch block. This used to close ***files, data base connections*** etc. This block is executed irrespective of whether exception occurs or not.



All exception classes are sub classes of the super class Exception. The inheritance hierarchy is has given below.



Object class:

It is a class which is a super most classes in the java. Every class defined in java will automatically. Because, every class is a sub class of object class in java.

This class belong to **java.lang.object**. It provides the following functions.

1. toString()
2. hashCode()
3. equals()
4. notify()
5. wait()
6. notifyAll()
7. finalize()

Exception class:

This is the generic catch because exception class can handle any kind of exception. Hence, it should be kept at the end of all catches to ensure that no exception will go un-defined on running the prg.

Built-in Exception classes:

Exception	*Description*
ArithmeticException	Arithmetic error
ArrayIndexOutOfBounds Exception	Array index out of bounds
NullPointerException	Accessing an Object through null pointer
StringIndexOutOfBounds Exception	String index out of bounds
NumberFormatException	Given no. is not number
IllegalMethodException	Method being called is illegal.
ClassNotFoundException	class not found

IllegalArgumentException	Argument is illegal.
MalFormedURLException	URL is not well formed
SQLException	SQL statement is not well.
Exception	Generic Exception
IOException	Input / output exception
IllegalAccessException	class is not accessible
FileNotFoundException	unable to locate a file.

* * *

user-defined exception:

When built-in exceptions do not meet our specific requirements. We can define our own exception classes such exceptions are known, as user-defined exception must extend the base class exception.

Syn.

```
class classname  extends  Exception
{

}
```

Call Stack:

It is a stack, which stores fun calls in LIFO(Last In First Out) manner.

Eg.

```
Class stackdemo
{
psvm(String args[])        {
doIt();
```

```
        }
        psv doIt()
        {
                doMore();
        }

        p s v doMore()
        {
                doMuchMore();
        }

        p s v doMuchMore()
        {
                sopln("Hello");
        }
}
}
```

MULTI THREADS

Multi tasking:

It is the ability of o.s. to execute multiple prg or tasks simultaneously. It uses only one processor is able to execute many applications. For user it appears to be all task are running simultaneously (at a time). However, o.s. follows time sharing model to schedule the tasks. It allocates time to each task and switch between them. Thus at any given time only one task is under execution.
Multi tasking is in two ways. 1. Pre-emptive 2. Non-preemptive.

Pre-emptive:
In this approach operating system gives control to a task and gets it back. Once the given time is over.

Non-pre emptive:

In this approach, o.s. gives control to a task and cannot get it back unless it is returned to the task after specified time.

MULTI PROCESSING:

It is a concept where multiple processors will process the given job or input. Therefore, the computer with multiprocessing can process large volumes of data in a short span of time. Such computers can be called as **super computer**.

These computers are used for machine critical applications such as weather forecasting, prediction of Earth Quakes, Cyclones, Tsunami and Launching and controlling the Rockets.

/* Types of Multiprocessing.
1.Symmetric
2.Asymmetric

Symmetric:

In this approach a processor will always try to share the job of other process. Once its part is over. This is to ensure that all processors are utilized to optimum performance and no processor is kept idle.

Asymmetric:

In this approach, a processor is kept idle once its given portion of job is completed. */

Multi Threading:

A prg. Under execution is called "***process***".

Thread: It is a smallest unit of executable prg. It is a portion of prg under execution.

Multi threading is a process of executing multiple threads simultaneously. By running multiple threads, we can ensure that many things are done at a time.

For user it appears to be all threads are running concurrently. However, at any point of time only one thread is under execution. It is managed by o.s. It follows time sharing model.

"*We can do many jobs at a time*".

Note: ***by default*** every java program, we write its thread name is "***main***". It belongs to a default thread group by name "***main***".

Threads in java:

Java has built-in support for threads. But other languages depends upon o.s. Support for multi threaded prgg.

Creating threads in java:

There are two ways to create threads.

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

Thread:

It is a class, which belongs to **java.lang** package. It is used to create threads.

Creating threads by extending thread class:

```
class classname extends Thread
{
        public void run( )  //built-in methods
        {
                //code
        }
}
```

Runnable:

It is an interface, belongs to **java.lang** package used to crate threads.

Creating threads by implementing thread class:

```
class classname implements Thread {
        public void run( )  //built-in methods
        {
                //code
```

```
        }
}
```

We have created an object of Thread class and start() method is called. Start() method automatically calls the run() method.

Choosing an approach:

It is a possible to create threads in 2 ways.

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

when ur class is not sub class of another class. We can follow either of approaches.

If ur class is already a sub class of another sub class, to take it a Thread class, you con't follow the first approach, but the second one, implementing Runnablein suitable.

Thread states:

Born : a newly created thread is in a born state.
Ready : after a method is created, it is in its ready state waiting for start() method to be called.

Running : Thread enters the running state when it starts executing.
Sleeping : Execution of a thread can be halted temporarily by using sleep() method. The thread becomes ready after sleep time expires.
Waiting: Thread is in waiting state. If wait() method has been invoked.
Blocked: When the thread waits for an input / output operation.
Dead : after the run() method has finished or the threads stop() method is called.

***

Javap:

It is a java's class dis-assembler. It is used to new method of any class. This is a part of JDK.

Eg.
javap java.lang.String
(It displays all methods of String class on to the screens.)

Redirecting output to a text file:
javap java.lang.String > string.txt
(Redirects output to the file *string.txt*. Which can be operand in notepad)

Methods supported by Thread class:

isAlive( ) ? return true if the thread is alive.

getName( ) ? returns the name of the thread

start( ) ? used to start a thread by calling the method run().

resume( ) ? to resume the execution of a suspended thread.

yield( ) ? causes the currently executing thread to temporarily pause and allow other threads to executed.

setName(String obj) ? Sets the name of the thread. The name is passed as argument.

join( ) ? waits for the thread to die.

isDaemon( ) ? checks if the thread is a Daemon thread.

activeCount( ) ? returns the number of active threads.

sleep( ) ? used to suspend a thread for a certain period of time.

* * *

Thread Priority:

Every thread that is created is given. Some priority level, which signifies the amount of time to be allotted for that thread. In other words, priority determines how much importance to beginners to a thread. By default every thread priority is 5. This priority can be increased or decreased by using the method **setPriority**() .

Eg.

t1. setPriority(6);

The parameter in bracket indicates priority level of a thread. It can be from 1 to 10 Or it can be either of the flg. Constants.

Thread.MAX_PRIORITY (10)
Thread.MIN_PRIORITY (1)
Thread.NORM_PRIORITY (5)

getPriority( ) : It returns the priority of the thread.

Eg.

System.out.println(t1.getPriority());

Thread Synchronization: (matched or corresponding)

It is a process of allowing only one thread to make use of a sharable resource. While other threads are waiting. This is required only when multiple threads are trying to access the same resources concurrently. Consider the scenario (state or situation). Where threads such as t1,t2, t3, t4 are to use a file for read, write, insert and update operations respectively and simultaneously. Then the file will be left in an inconsistent state. Because a resource can not used by more than one thread uses the file, the threads are to be synchronized.

The flg. is the general synchronization method:

```
synchronized returntype  funname(parameters) {
        // code goes here
 }
```

wait – notify mechanism:

It is the mechanism used in thread synchronization. It uses the following methods.
1. wait( ) 2. notify( ) 3. notifyAll( )

wait(): Causes the thread to wait until it is notified.
notify(): Casuses a thread to be ready to make use of the resource.

notifyAll(): Notifies all threads to be ready.

Conditions for thread execution:

Thread is:

- ? Not of highest priority.
- ? Put to sleep using sleep() method.
- ? Suspend using suspend() method.
- ? Thread is waiting because wait() method was called.
- ? Explicitly yielded using yield() method.
- ? Blocked for file I /O.

Types of Threads:

Threads are in two types. Those are,
1. Daemon threads 2. User Threads

Daemon Thread:
It is a thread, which runs in the background and helps run the user defined thread.

Eg.
Garbage collector is a low priority daemon thread. I.e. collects garbage.

User Thread:
It is a thread defined by user. Which will be run by a daemon thread.

Dead Lock :
It is a situation where two threads will have circular dependency. Java serves this problem by releasing the locks.

STRING HANDLING

String is a group of characters. To manipulate strings java provides the flg. classes.
1. String 2. StingBuffer

String:

It is used to create String objects. String objects are "immutable". This means that they can't be modified. Though String is a class. It can be used as a data type.
Eg.

```
String s = "java";
```

StringBuffer:

It is also used to create string objects. But these objects are mutable (changeable). This means that they can be modified. It should be instantiated as follows.

Eg.
StringBuffer s = new StringBuffer("java");

Constructors of String class:

String():
Creates a String object. Which is empty.
Eg.

```
String s = new String();
```

String (String str):
Creates a string object with given string.
Eg.

```
String s1= new String("java");
String s2=new String(s1);
```

String(char[] ch):
Creates a string object with given character array.
Eg.

```
Char ch={'v','i','s','i','o','n'};
String s= new String (ch);
```

String (char ch[], int startindex, int numchar):
Creates a string object with from start index to num of characters of given array.

Eg.
Char ch[]={'v','i','s','i','o','n' };
String s= new String(ch,2,5);

Methods of String class:

charAt(int):
returns the character specified by the index..
Eg.
String s="hello java";
Char ch=s.charAt(4);

concat(String str):
It concatenates the given string with invoking string object and the result is assigned to a new string.
Eg.
String s1= "Hello";
String s2="java";
String s3=s1.concat(s2);

copyValueOf(char ch[]):
copies the given character array into a string object.

Eg.
char ch[]= {'j','a','v','a'};
String s=String.copyValueOf(ch);

copyValueOf(char ch[], int startindex, int numchar):
Copies the given character array from start index to number of characters into a string object.

Eg.
char ch[]= {'j','a','v','a'};
String s=String.copyValueOf(ch,2,2);

endsWith(String str):
It returns true. If the invoking string object ends with given string.

Eg.
String s="Welcome to java";
boolean b=s.endsWith("java");

startsWith(String str):
It returns true. If the invoking string object starts with given string.

Eg.
String s=”Welcome to java”;
boolean b=s.startsWith(“java”);

equals(obj):
Returns true if the invoking string object and given object are equal.

Eg.
String s1= “java”;
String s2=”j2se”;
boolean b=s2.equals(s1);

equalsIgnoreCase(obj):
Returns true if the invoking string object and given object are equal. But it ignores case.

getBytes():
Returns the invoking string object as a byte array.

Eg.
String s=”java”;
byte b[]= s.getBytes();

getChars(int startindex, int endindex, char ch[], int startindex):
it returns the invoking string object from start index to endindex to array ch at specified index.

Eg.
String s=”welcome to java”;
Char ch[]=new char[20];
s.getChars(2,5,ch,0);

toLowerCase():
converts String object to lower case.
Eg.
String s=new String (“hello java”);
s.toLowerCase();

toUpperCase():
converts String object to upper case.
Eg.
String s=new String (“hello java”);

s.toUpperCase();

trim():
It removes spaces in the invoking string object.
Eg.
String s = “ hai “;
s.o.p(trim(s));

valueOf():
It is used to convert given parameter to a string.
Eg.
String s= “ “;
Int x=250;
S=String.valueOf(x);

replace():
It is used to replace a character of invoking object with another character and result in assigned to other string object.
Eg.
String s=”J2SE”;
String s2=s.replace(“s”,”e”);

substring():
It returns a string within another string from specified index.
Eg.
String s=”java is good”;
String s1=s.substring(5);

substring(int, int):
It returns a string from specified starting index to ending index of string object.
Eg.
String s=”java is good”;
String s1=s.substring(5,10);

indexOf():
It returns the index of specified string within the invoking string object.
Eg.
String s=”java is good”;
Int x =s.indexOf(“is”);

lastIndexOf():
It returns the position / index of last occurrence of specified string in the invoking string object.
Eg.
String s=”java is good and is very nice”;

Int x =s.lastIndexOf(“is”);

split():

It splits the invoking string object based on the criteria given as parameter and the result is assigned to other string object.

Eg.
String s="hello, good, java";
String x[]=new String[5];
x=s.split(",");
for(int i=0;i<x.length;i++)
System.out.println(x[i]);

Output:
hello
good
java

replaceFirst():

Replaces the first occurance of the string (parameter1) other string (parameter2) and result is assigned to another string object.

Eg.
String s=new String (“java is as good as any other language”);
String s1=s.replaceFirst(“as”, “very”);

StringBuffer:

This class is also used to manipulate strings. It objects are mutable. This means that they can be modified. This class belongs to java.lang package.

Constructors of StringBuffer

StringBuffer():

It creates an empty string buffer object.
Eg.
StringBuffer s= new StringBuffer();

StringBuffer(int):

It creates a string buffer object with initial capacity is 16.
Eg.
StringBuffer s= new StringBuffer(16);

StringBuffer(String_str):
It creates String Buffer object to given string.
Eg.
StringBuffer s=new StringBuffer("java");

Methods of StringBuffer class:

length():
It returns the length of string buffer object.
Eg.
StringBuffer s=new StringBuffer("Welcome to java");
int x = s.length();

capacity():
It sreturns the capacity of string buffer object.

Eg.
StringBuffer("Welcome");
Int x=s.capacity();

ensureCapacity(int):
It is used to set capacity of StringBuffer object.

Eg.
StringBuffer s=new StringBuffer("helloworld");
s.ensureCapacity(20);

setLength(int):
to set length of StringBuffer object.

Eg.
StringBuffer s=new StringBuffer("hello world");
s.setLength(20);

charAt(int):
returns a character at specified index.

Eg.
StringBuffer("Vision computer education");
Char ch=s.charAt(3);

setCharAt(int, char):
Set given char. of specified index.
Eg.
StringBuffer s=new StringBuffer("hello world");
s.setCharAt(3,k);

getChars(int startindex, int endindex, char target[], int target-startindex):
It copies a substring of StringBuffer object from start index to end index into target array.

Eg.
Char ch[]=new char[20];
SB= new SB("hello, very good java");
s.getChars(1,4,ch,0);

append(String str):
To append given string to the string buffer object.

Eg.
StringBuffer s=new StringBuffer("Hello");
s.append("java");

delete(int startindex, int endindex):
Deletes character of string buffer object from start index to end index.
Eg.
StringBuffer s=new StringBuffer("hello java");
s.delete(3,6);

deleteCharAt(int):
Deletes character at specified index.
Eg.
SB s=new SB("j2me");
s.deleteCharAt(2);

replace():
It is used to replace some text of string buffer object with given string.

Eg.
SB s=new SB("Hello King java");
s.repace(2,7,"vision");

* * *

I/O Packages

The standard streams in java such as **System.in, System.out** and **System.err** cannot deal all kinds of I/O. In order to create all kinds of I/O operations, develops of java defined many classes meant for I/O. Those classes are kept in a package by name **java.io**.

FILE:
It is a collection of data or information stored persistently in a storage mediums.
Eg. HD, FD, CD etc.

Stream:
It is a sequence of bytes or characters.

Character stream class:
These are the streams where data is flown in the form of bytes.

Input stream:
A stream, which is used to read data from a resource.
Out stream:
A stream, which is used to write data to a source.

Data Input Interface:
Used to read bytes from a binary string and reconstruct data in any of the java primitive types.
Allows us to convert data that is in java modified **Unicode Transmission Format (UTF 8)** to string format.
Data input interface defines a num. Of methods including methods for reading java primitive data types.

Data Output Interface:
Used to reconstruct data that is in any of the java primitive types into a series of bytes and writes them into a binary system.
Allows us to converts string into java modified **UTF-8** format & writing it into a stream.
All methods under Data Output Interface throw an IOExceptions in case of an error.

InputStream class:
An abstract class that defines how data is received.
The basic purpose of this class is to read data from an input stream.

Methods	**Purpose**
int read()	The most important method of input stream. It reads bytes of data from a stream.
int available()	This method returns the num. Of bytes that can be read without blockage. It returns the number of available bytes.
void close()	This method closer the input streams. It releases the resources associated with the stream.

void mark() This method is used to marks the correct position in the stream.

boolean markSupported() This method indicates whether the streams supports mark and reset capabilities.

long skip(long n) This is method skips n bytes of input.

OutputStream class:

An abstract class that defines the way that which outputs are written to streams.

This class is used to write data to stream.

Methods	**Purpose**
Void write()	This is the most fundamentals operations in outputstream class the methods writes a byte. It can also write an array of bytes.
Void flush ()	This method flushes the stream. The buffered data is written to the output stream.
Void close()	This methods closes the stream. It is used to release any resource associated with the stream.

File Input Stream

Used to read input from a file in the form of stream.

Commonly used constructors of this class.

FileInputStream (String filename): creates an input stream that we can used to read bytes from a file. Filename is full pathname of a file.

FileInputStream(File name): creates an input stream that we can use to read bytes from a file. Name is a file object.

File Output Stream

This class is used to write output to a file stream. Its constructors are,

FileOutputStream (String filename): creates an output streams that we can use to write bytes to a file.

FileOutputStream (File name): creates an output stream that we can use to write bytes to a file.

FileOutputStream (String filename, boolean flag): creates an output stream that we can use to write bytes to a file. If flag is true, file is opened.

ByteArrayInputStream (byte b[],int start, int num): creates a byte array input stream that begins with the character at start position and is num bytes long.

ByteArrayOutputStream :

Used to create an output stream using a byte array as the destination.
This class defines two constructors.
One takes an integer argument

Second one does not take any argument.
Additional methods like **toByteArray()** and toString() convert the streams to a byte away or string object respectively.

* * *

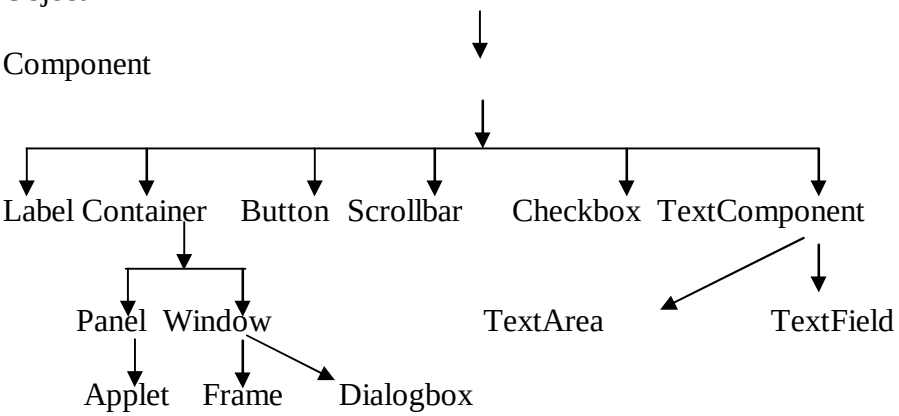
AWT

AWT is the package, which provides various visual interface controls, containers, Fonts, Graphics and Eventhandling classes and interfaces.

Control:
Visual control: it is a component that forms graphical user interface.
Eg.
Toolbox, Listbox, Combobox, Checkbox, Label, TextArea, Scrollbar, Button, Checkbox, RadioButton etc.

Containers:
It is a visual component, which holds other GUI contents.
With the help of the classes and interface of a AWT packages. GUI applications or windows applications can be developed.

Eg.
Window, Frame, Applet, Panel, Dialog etc.



Event Handling in Java:

Java uses a model to handle events in which it delegates events handling of various kinds of events into classes and interfaces. This model is known as Event delegation model.

Event:
It is action triggered by user for another event or system.

Event Handler: It is a piece of code that handles an event.

Eg. Click , double click, drag mouse etc.

The following table shows various event classes. The corresponding listener interfaces and their purpose.

Event Class	Purpose	Listener Interface
ActionEvent	click on button, Menu items, Double click on A list items.	ActionListener
ItemEvent	chechbox, Radiobuttons, List items class	ItemListener
TextEvent	Text fields & Text areas	TextListener
AdjustmentEvent	Scrollbar	AdjustmentListener
FocusEvent	When components Gain or lose focus	FocusListener
ComponentEvent	When a component Is moved, raised or hidden	ComponentListener
WindowEvent	when window is opend, Closed, min., max., acti-Vated or deactivated.	WindowListener
MouseEvent	for mouse events	MouseListener
KeyEvent	for key board events	KeyListener

HTML

An HTML document is a text file that contains the elements that a browser uses to format a document, to display multimedia objects, and to create hyperlinks.

An HTML document file contains many elements. An element consists of a start tag, an end tag and the data characters enclosed by the two tags. A tag defines a format to apply or an action to take. An element can have one or more attributes and values. A tag starts with a less than(<) sign and ends with a greater than(>) sign. Tag and attribute names are not case-sensitive, but are typically written in uppercase to distinguish them from the data characters.

An end-tag consists of the tag name immediately preceded by a slash(/). Some tags require that you always provide the matching end-tag; others allow you to omit the end tag if the result is clear and unambiguous.

This is the simple HTML document.

```
<HTML>
<HEAD>
<TITLE> simple HTML Document </TITLE>
</HEAD>
<BODY>
<P> A very simple HTML Document.
</BODY>
</HTML>
```

An HTML document starts with an HTML tag. This marks the file as an HTML document. The matching end tag (</HTML>) is the last tag in the file. The HEAD tag marks beginning of the document header. Typically, the TITLE element appears here. Internet Explorer displays this element in its title bar.

The BODY tag appears at the start of the content of the document. The BODY element encloses the body text, images and multimedia objects. The P element adds a new paragraph with a carriage return and line feed. The ending P tag is typically omitted.

```
<p> click <A HREF = “www.koti.ac.in/” > here</A> to visit us.
```

<p align=center> this is center aligned.

Character Formatting:

We can use a variety of tags to set the size and style of the text characters. The B or STRONG tag to make text bold. The I or EM tag to make text italic. The S or STRIKE tag to strike out text and the U tag to underline text.

<p> This is bold; this is <I>italic</I>.
<p> <Strong> this phrase is bold;
<EM> This is italic. </EM>

if you use the FONT tag to change text size , you can specify either a fixed or relative size. A fixed size is a number in the range 1 to 7. A relative size is a +ve or –ve number preceded by the plus(+) or minus(-) sign. That indicates a size that is relative to the base font size, as set using the BASEFONT tag.

<BASEFONT SIZE=3> this sets the base font size to 3.
<Font size =+4 > now the font size is 7.
 now the font size is 2.

The FACE attribute with the FONT tag to set the facename of the font used for text. E.g. "Arial", "Times new Roman", "Courier New" etc.

<H1><Font face = "Times New Roman"> Koti Software Solutions Pvt. Ltd. </H1>

we can set colors in to HTML document by using the color attributes of the BODY, FONT,HR,MARQUEE and TABLE tags.
Bgcolor attribute used for backgrounds

HTML supports the following predefined color names:

AQUA BLACK BLUE FUCHSIA GRAY GREEN LIME
MAROON NAVY OLIVE PURPLE RED
SILVER TEAL WHITE YELLOW.

A red-green-blue color value consists of three byte values, with each value specifying the intensity of the corresponding color.
E.g.

#00FF0000 is red because the red value is set to full intensity and green and blue are set to zero.

#0000FF00 is green and
#000000FF is blue.

IMAGES AND MULTIMEDIA

You can embed images, and even video clips in your HTML document by using the IMG and BGSOUND tags.

The img tag to insert images into our document. You specify the image source, typically a GIF or JPEG file, and specify the image attributes, such as the width and height, alignment and so on.

<IMG SRC="the earth.gif" WIDTH=46 HEIGHT=46 ALT="Picture of the earth">

Applets

1. Applet is a java program with GUI that can be embedded over web pages.
2. Applets are used to make web pages dynamic and interactive.
3. Applets execute in the client machine.
4. Applets are executed by browser.

Differences b/w applications and applets

Application	Applet
Is a Java program, which is stand alone. It executes in the local machine.	It is a Java program with GUI, that is embedded in HTML pages.
It is executed by java interpreter.	It is executed by browser.
Main() is the entry point	init() is the entry point into applet.
Can use disc I/O of local machine	Can't use disc I/O of local machine.

Creating applet

1. import java.applet.*;
2. public class ClassName extends Applet
3. In place of constructor, init() method is used
4. No main() method
5. No setSize(), setVisible(), setTitle()
6.

Develop LoginApplet

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/* <applet code="LoginApplet" height=200 width=200>
    </applet> */
public class LoginApplet extends Applet implements

ActionListener {
  Label l1, l2;
  TextField t1, t2;
  Button b1, b2;
  public void init() {
          l1=new Label("User ID:");
          l2=new Label("Password:");
          t1=new TextField(10);
          t2=new TextField(10);
          t2.setEchoChar('*');
          b1=new Button("Submit");
          b2=new Button("Reset");
          b1.addActionListener(this);
          b2.addActionListener(this);
          add(l1);
          add(t1);
          add(l2);
          add(t2);
          add(b1);
          add(b2);
 }
 public void actionPerformed(ActionEvent e) {
          if(e.getSource()==b1) {
           if(t1.getText().equals("vision") &&

t2.getText().equals("nrt"))
             showStatus("Valid User");
   else
          showStatus("Invalid User");
           }
          else {
           t1.setText("");
           t2.setText("");
          }
 }
}
```

appletviewer

It is a tool provided by JDK. It is used to test applets before embedding them over web pages.

Usage:
appletviewer FileName.java

Eg:
appletviewer LoginApplet.java

It takes .java file as input. It looks for a comment in .java file. From comment it takes .class file name, width and height and executes the applet.

Embedding Applet on web page

```
<html>
<body bgcolor="cyan" text=orange>
<h1 align=center>Tesing LoginApplet</h1>
<hr size=2>
<center>
<applet code=LoginApplet height=200 width=200>
</applet>

</body>
</html>
```

Save: Filename.html
Open: in browser

Homework:
Develop Calculator Applet.

Applet Life cycle methods

1. init()
2. start()
3. paint()
4. stop()
5. destroy()

When applet is started:

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

When applet is closed:

1. stop()
2. destroy()

init() is called only once in the life cycle of applet.

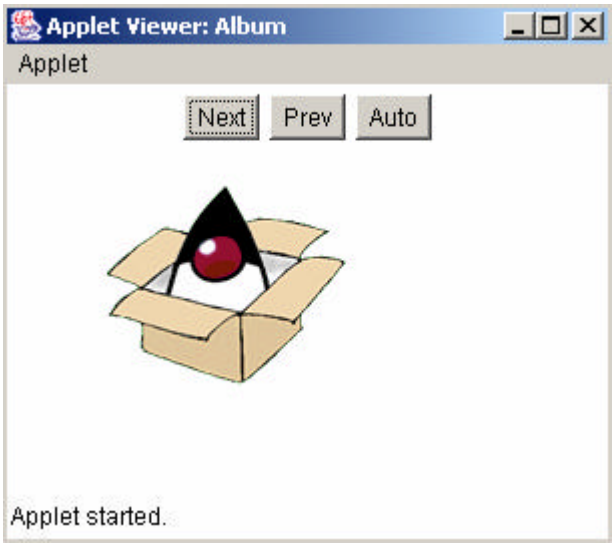
start() is called no. of times.

stop() is called no. of times.

Drawing images on Applet

```
import java.awt.*;
import java.applet.*;
/*<applet code=ImageDemo height=300 width=300></applet> */
public class ImageDemo extends Applet{
Image i,j,k;
public void init() {              i=getImage(getCodeBase(),"javalogo.gif");
                  j=getImage(getCodeBase(),"globe.gif");
        k=getImage(getCodeBase(),"thumbsup.gif");
  }
public void paint(Graphics g) {                  g.drawImage(i,20,20,this);
                  g.drawImage(j,60,60,this);
        g.drawImage(k,150,100,this);
} }
```

Homework:
Develop an Album applet




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

```
/*<applet code=Album width=300 height=300> </applet> */

public class Album extends Applet implements ActionListener
{

	Image img;
	String
imgs[]={"boat.gif","box.gif","dukeplug.gif","globe.gif","javalogo.gif","m
agnify.gif"};
	Button next,prev,auto;
	int i,j;

	public void init()
	{


		i=0;
		img=getImage(getCodeBase(),imgs[0]);
		next=new Button("Next");
		prev=new Button("Prev");
		auto=new Button("Auto");
		add(next);
		next.addActionListener(this);
		add(prev);
		prev.addActionListener(this);
		add(auto);
		auto.addActionListener(this);

		setSize(300,300);
		setVisible(true);

	}

	public void paint(Graphics g)
	{
		g.drawImage(img,50,50,this);

	}

	public void actionPerformed(ActionEvent e)
	{

	if(e.getSource()==next)
	{
```

```
                if(i<5)
                        i++;
                else
                        i=0;

                img=getImage(getCodeBase(),imgs[i]);

        }
        else if(e.getSource()==prev)
        {

                if(i>0)
                {
                        i--;
                }

                else
                {
                        i=5;
                }

                img=getImage(getCodeBase(),imgs[i]);

        }

        else if(e.getSource()==auto)
        {

        for(j=0;j<6;j++)
        {
                img=getImage(getCodeBase(),imgs[j]);
                repaint();
        }
        }

        repaint();
        }
}
```

Passing parameters to an Applet

Runtime parameter can be passed to an applet. It is dine using <PARAM> element within <APPLET> element.

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

```
public class AppletPara extends Applet {
Image i;
public void init() {
String imagename=getParameter("image");
        i=getImage(getCodeBase(),imagename);
}
public void paint(Graphics g) {
g.drawImage(i,50,50,this);
} }
```

```
<HTML>
<HEAD>
        <TITLE>Applet Parameters</TITLE>
</HEAD>
<BODY BGCOLOR=orange text=blue>
<FONT COLOR=RED SIZE=10>
<MARQUEE><U><H1 align=center></H1>Applet

Parameters</U></MARQUEE>
</FONT>
<center>
<APPLET CODE=AppletPara height=200 width=200>
<param name="image" value="boat.gif">
 </applet>
</center>
</BODY></HTML>
```

Drawing on Applets

paint() method is used to draw shapes on applet. The Graphics object g supports methods to draw various shapes.

```
import java.applet.*;
import java.awt.*;
/*<applet code=PaintApplet width=300 height=300> </applet> */
public class PaintApplet extends Applet {
public void init() {      setBackground(Color.yellow);
        setForeground(Color.blue);
 }
public void paint(Graphics g) {         g.fillRect(50,50,50,50);
        g.setColor(Color.red);g.fillOval(110,110,60,60);
Font f=new Font("Times New Roman",Font.BOLD,20);
g.setFont(f);
g.setColor(Color.green);
g.drawString("Happiness is an attitude",150,200);
g.drawLine(100,100,300,300);
 } }
```

Connecting to Websites

Games

Playing Audio

Collections

Drabacks of Array:

1. Fixed size (static)
2. Can store only one kind of data

Collection

It is a group of objects. It stores data in the form of objects. It supports the following operations:

1. Adding objects
2. Removing objects
3. Searching for objects
4. Sorting objects
5. Navigating thru collection.

java.util package

This package provides the following.

1. Legacy (old) collection classes
2. New collections framework (DS)
3. Utilities and
4. Other useful classes.

Auto boxing: converting the primitive data types to objects.

✍ All collections can store objects only.

Primitive data types are not supported. However, the JDK1.5 onwards primitives are automatically converted to the corresponding wrapper types. This feature is known as Auto boxing. Auto boxing is vice versa.

✍ collections frame work is a part of util package.

Legacy classes:

1. Dictionary
2. Hashtable
3. Properties
4. Vector
5. Stack

Legacy Interface

Enumeration

(It is used to navigate thru legacy collection classes)

Enumeration: It is a legacy collection interface. Which is used to iterate through legacy collections. It has the following methods.

nextElement() returns value of an elements.

hasMoreElements() returns true if elements are present.

Dictionary
It is an abstract class. Which has methods that allow us to add and manipulate key value pairs. Key must be unique(duplicates are not allowed). Values can be duplicated.

Hashtable
It is sub class of Dictionary. It allows data to be stored in the form of key/value pairs.
Eg: HashtableDemo.java

Properties
It is sub class of Hashtable. It has all qualities of Hashtable. In addition to them, it does the following:

1. Can read key/value pairs from a file.
2. Can write key/value pairs to a file.
3. Can obtain System properties.
4. Can read all system properties in the form of key value pairs.

Eg. PropertiesDemo.java (read system properties)
Eg. PropertiesToFile.java
Eg. PropertiesFromFile.java

Vector: it is a dynamic array. Its size can be increased / decreased at runtime. It can store any kind of data in the form of objects.
Eg. VectorDemo.java

Stack: It is a subclass of vector. In addition to vector's qualities, it can be used as a LIFO data structure. It provides methods like.

push() : adds an item to stack
pop() : removes an item from stack
peek() : retrieves an item without removing it from stack.

Eg. StackDemo.java
New Collections Frame work
It is a set of classes and interfaces that are used to create and manipulate collections.

Interfaces	Classes
Collection	Lists
List	Sets
Set	Maps
Map	Queues
SortedSet	Utilities

SortedMap
Queue

The classes of collections framework are classified into the flg. categories.

1. SETS: these are the classes that implement set interface.
   Eg. HashSet, LinkedHashSet, TreeSet

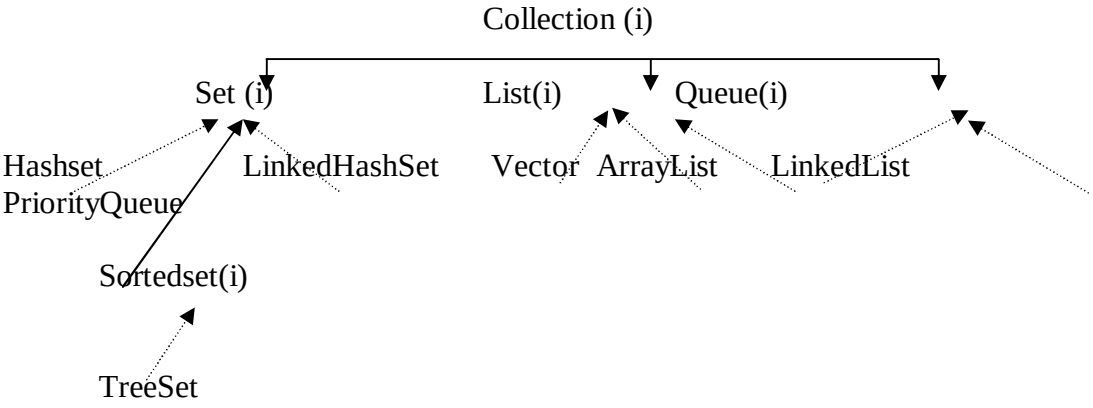
2. SETS: these are the classes that implements list interfaces.
   Eg. ArrayList, LinkedList, Vector

3. MAPS: These are the classes that implements map interface.
   Eg. HashMap, LinkedHashMap, TreeMap, HashTable.

4. QUEUES: These are the classes that implements queues interface.
   Eg. PriorityQueue

5. UTILITIES: These are the classes that provide various static methods that perform sorting, searching and other operation on collection.
   Eg. Collections, Arrays.

Hierarchy of Collection Classes:



The sets class allows a group of objects to be added without duplicates.

Hash set does not retain the order.

TreeSet sorts the elements automatically(ascending order).
Linked hashset retains the order.

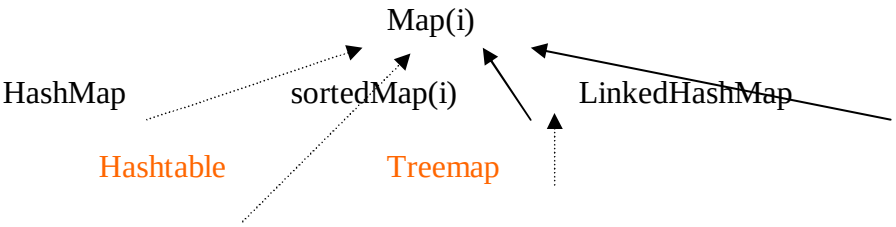
Lists allow a group of objects to be added with duplicates.

Vector is a dynamic order, which is ordered, but it contains synchronized methods.

Arraylist similar vector without synchronized method.

Linked list is an ordered list of objects. It can also be used as first in first out (queue) data structure.

Priority queue, the items will be sorted based on the priority given.



All maps stores a group or key value pairs.

Hashtable does not retain the order. It has synchronized methods.

Hashmap similar to Hashtable without synchronized methods.

Treemap sorts elements automatically based on the keys (key is unique)

Linked Hashmap retains the order.
Eg. HashSetDemo.java

Eg. TreeSet.java : same prg., in & output elements are in ascending order.

Eg. LinkedHashSet.java : same prg., output retains the order.

Eg. ArrayListDemo.java
Eg. MailingList.java
Eg. HashMapDemo.java

LinkedHashMap: It retains the order.

TreeMap : It sorted in the ascending order by key.

Collections: It is a class, which has static methods that implement sorting & searching algorithms, these methods are used to operate on any collection object.
Eg. CollectionAlgorithmsDemo.java

Arrays: this class has various static methods that implement sorting, searching algorithms. These methods can operate on all kinds of arrays.
Eg. ArrayDemo.java

StringTokenizer: It is a class of util package. It is used to break a string into pieces known as tokens.

Eg. Test.java

Networking

Terminology

1. Client
   It is a computer, which makes request.
2. Server
   Computer, which gives response.
3. Request
   what client makes is request.
4. Response
   What server gives is response.
5. Web page
   A file/page over Internet with .html/.htm extension.
6. Web site
   A collection related/inter-linked web pages.
7. WWW
   World Wide Web. It is a collection of all web servers in the world.
8. Network
   Connection b/w two or more computers in order to share information and resources.
9. Types of networks
   a. LAN
   b. MAN
   c. WAN
10. Internet
    It is a network of networks. It is a global communication medium. It is a universal database of knowledge. It is a public network.
11. Intranet
    It is a private network of a company.
12. Extranet
    Extended private network.
13. Netizen
    A person who uses Internet.

14. ITZen
    A person who knows IT.
15. Protocol
    It is a set of rules and regulations to be followed by client and server.
16. TCP/IP
    It is the backbone of Internet. It contains sub protocols such as
    a. HTTP
    b. FTP
    c. SMTP
17. HTTP
    Used for web pages.
18. FTP
    Used to get files.
19. SMTP
    Used in mailing.
20. UDP
    User datagram protocol. Not reliable.
21. URL
    Universal/Uniform Resource Locator. It is nothing but Internet address
    Eg:
    http://www.yahoo.com
22. Datagram/Packet
    When a message is sent over network, it is broken into pieces known as datagrams/packets.
23. Types of communications
    Asynchronous and Synchronous.
24. Types of net connections
    a. Direct cable
    b. Dial up
25. Computer Identification
    a. by name
    b. by IP address
26. IP Addressing
    a. It is a 32-bit addressing scheme. It is used to identify network.
27. Port
    It is a logical number, which makes communication possible.
28. Socket
    a. It is a Java class, which enables communication between client and server programs.

<u>Asynchronous :</u>
Need not be online.

Eg. Mailing

Synchronous:
Need be online.
Eg. Telephone
Chatting