UNIT 1
Basic Concepts of Object Oriented Programming

1. Class

Class is a framework that can be used to define the structure of an object – class is a logical construct – define the structure and behavior (data & code) that will be shared by a set of objects - used to create new data types (objects).

Example

1. Blue print of house plan
2. Circuit diagram of electronic device.

2. Object

Object is a real world entity - object is a run time instance of a class - Object is a user defined variable of type class.

Example: TV, RADIO, TABLE, etc...

3. Data Hiding

The insulation of data from direct accessing is called data hiding. That is the user cannot access the private members outside of the class directly.

Example: - Consider a real world object RADIO. It contains many data items (speaker, amplifier, etc...) and functions (on, off, play, stop, etc…). Here the user's doses not access the data items directly. Only by using the functions the users can access the data items.

4. Encapsulation

Binding of data and methods with in a single unit (object) is called as encapsulation. In the previous example, when you bind the data items and functions then you will get a real world object RADIO.

Binds together code and the data it manipulates, and keep both safe from outside interference and misuse – "Shifting gears does not turn on the headlights"

5. Abstraction

Abstraction is an ability, which shows the essential things and hides the unwanted things of an object from the user.

Example: - Manual of electronic devices**:** Here the manual contains the information about, "How to operate/use the device/system" - it does not contain the internal design information.

Humans mange complexity through abstraction

Ex: do not think car as a thousands of individual parts – think of it as a well-defined object – ignore the details of how the engine, transmission, breaking systems work – from the outside, the car is a single object

6. Function Overloading

Same function name with different number of arguments or/and different type of arguments available with in a class.

Example: - void sum (int, int);

void sum (float, float);

void sum (int ,float , int);

7. Inheritance

Inheritance is the process of creating new class or classes from an already existing class - Process by which one object acquires the properties of another object

8. Overriding

- **Overloading:** An adjustable screwdriver may work on different types of screws based on the attachment *(bit)* supplied.
- **Overriding**: You want to change a CV (Resume) template as per your needs. So, you copy it and make the necessary changes *(in the copied file)*.

You overload a method when you want a method to perform different functions based on the type of data supplied. On the other hand, you override a method when you think that the given specification for a method does not meet your requirements.

9. Polymorphism

Polymorphism is a technique to call multiple functions by using a same function call - One function call invokes multiple functions.

Allows one interface to be used for a general class of actions – specific action is determined by the exact nature of the situation.

Ex: A dog's sense of smell is polymorphic – if the dog smells a CAT, it will bark and run after it – if the dog smells its food, it will slaver and run to its bowl

Ex: **Car accelerator** – it is a polymorphic function - what happens after you press it is different for different cars (implementation defined).

Ex: Function Overloading is a kind of polymorphism.

Operator Overloading is a kind of polymorphism

Birth of Modern Programming

- ✓ **Innovation in language design was driven by the need to solve a fundamental problem that the preceding languages could not solve.**
- ✓ **When a computer language is designed, trade-offs are often made, by,**

- ✓ **Ease-of-use Versus Power**
- ✓ **Safety Vs Efficiency**
- ✓ **Rigidity Vs Extensibility**

✓ **Assembly language**

- ✓ **To write highly efficient programs**
- ✓ **Not easy to learn or use effectively**
- ✓ **Code debugging is quite difficult**

✓ **Prior to C language,**

- **FORTRAN - FORmula TRANslation - IBM-1950s**
- **To write efficient programs for scientific & engineering applications**
- **Not good for systems code**

✓ **COBOL – *1959***

- **Common Business Oriented *Language***

- Run on the mainframe & PC
- Issues: syntax, compatability with earlier versions
- No academic computer scientists participated in the design of COBOL
- Committee from commerce & government

✓ **BASIC - Beginner's All-purpose Symbolic Instruction Code - 1964**

- John G. Kemeny and Thomas E. Kurtz
- Easy to learn – not very powerful
- Lack of structure - uncertain for large programs

Note: Not designed around structured principles – rely on GOTO as a primary program control – mass of jumps and

conditional branches – difficult to understand the program

- ✓ **PASCAL – Designed: 1968 to 1969**
  - **Published: 1970 by Niklaus Wirth**
  - **Named in honor of the mathematician Blaise Pascal**
  - **Structured – less efficiency for system-level code**
  - **Object Pascal (1985) – derived from Pascal**

1970:

- ✓ **Begginging of computer revolution – demand for software – attempted to create better computer language – hardware becomes common – computers are not kept behind locked**

doors – unlimited access to machines for programmers

- ✓ **Leads to the Invention of C language**
- ✓ **Invented & implemented by Dennis Ritchie on UNIX operating system**

- ✓ **Prior to C**
  - ✓ **BCPL - Basic Combined Programming Language**
  - ✓ **Developed by Martin Richards - 1966 -**
  - ✓ **BCPL is influenced the B language, invented by Ken Thompson - 1969**
- ✓ **C is the beginning of the modern age of computer programming**

- ✓ **Powerful – efficient – structured – easy to learn**
- ✓ **C is a programmer's language: designed, implemented, and developed by real working programmers - reflecting the way that they approached the job of programming**

Need for C++

- ✓ **1970 to 1980 – C is the dominant computer programming language – still widely used**
- ✓ **Once a project reaches a certain size, its complexity exceeds what a programmer can manage**

- ✓ **C: to write moderately complex programs**
  - **between 25,000 and 1,00,000 lines of code**
- ✓ **Why C++? To manage larger & complex programs – broken the barrier to handle larger programs**

1980s:

- ✓ **Structured approach to Object Oriented Programming (OOP)**
- ✓ **Invented a new way of writing programs**
- ✓ **OOP: Helps to organize complex programs through the use of**

- Inheritance – Encapsulation - Polymorphism

✓ **C++: Invented by Bjarne Stroustrup in 1979 at Bell Lab in New Jersey**

- **"C with Classes"**
- **Extends C by adding object-oriented features**
- **1983 renamed as "C++"**

✓ **Not a completely new programming language - an enhancement to an already highly successful one**

Origin of Java Programming Language

✓ **1980s to 1990s – OOP using C++ took hold.**

- ✓ **Development of software for consumer electronic devices like TV's, Remote Controls and Cellular Phones etc.**
- ✓ **WWW & Internet – forces computer language evolution forward**
- ✓ **Another revolution in programming**
- ✓ ***Birth of Java***

Milestones in the development of Java

Year	Development
1990	Sun Microsystems decided to develop a special software to manipulate electronic devices. A team of Sun Microsystems programmer including James Gosling, Patrick Naughton, Chris warth, Edfrank and Mike Sheridan was formed to undertake this task.
1991	After exploring the possibility of using the most popular object oriented language C++, the team announced a new language "**Oak**". It took 18

	months to develop the first version.
1992	The team demonstrated the application of their new language to control a list of home devices.
1993	The World Wide Web appeared on the internet and transformed the text based internet into graphical rich environment. The team came up with the idea of developing web Applet, using the new language that could run on all types of computers connected to the Internet.
1994	Developed a web browser called HotJava to locate and run Applet programs on Internet.
1995	Public announcement - "Oak" was renamed as "Java" - Companies including Netscape and Microsoft announced their support to Java.
1996	Java is a leader for Internet programming

- ✓**Primary motivation:** **Need for a platform - independent (architecture neutral) language – to create software to be embedded in various consumer electronic devices (different types of CPUs are used as controllers)**
- ✓**C, C++ & other languages:** **designed to be compiled for a specific target – requires a full C/C++ compiler targeted for that CPU – compilers are expensive & time-consuming to create**
- ✓**Solution:** **platform-independent language - produce code that would run on a variety of CPUs under differing environments**

- ✓**Programmers view: Portable programs are Elusive / Indefinable - old discipline of programming**
- ✓***Competitors like Intel, Macintosh & UNIX, most programmers, stayed within their boundaries***
- ✓**Need for portable code was reduced - Java taken a back seat to other**
- ✓**Advent of Internet & Web: Internet consists of diverse, distributed universe, populated many types of computers, operating systems and CPUs – Need to run the same program on many types of platform - Problem of portability returned with a vengeance**

- ✓ **Switch from consumer electronics to Internet programming – Internet ultimately led to Java's large-scale success**
- ✓ **Java derives the familiar syntax of C and echoing the object oriented features of C++**
- ✓ **Java is a programmer's language – gives full control to the programmer - language for professional programmers**

Working principle of C/C++ compiler:

Compiler: convert high level language into corresponding machine level language

Object program: Machine dependent code - Set of instruction design to run on a particular machine such as IBM, Intel, Macintosh (MAC), etc...

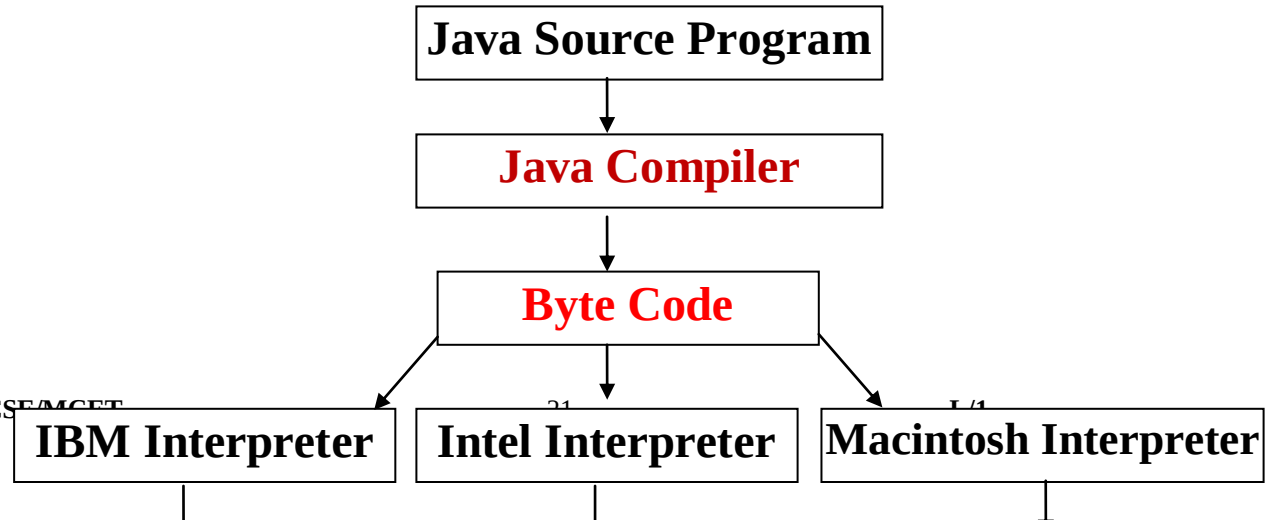
Working principle of Java compiler

Java's Magic: The Byte code:

- ✓ **It is a text code or machine independent code**
- ✓ **Output of Java Compiler is not Executable code (Key for security & portability)**
- ✓ **Byte code: highly optimized set of instructions designed to be executed by the Java Run-time System (JVM)**

Java Interpreter / Java Runtime System / Java Virtual Machine (JVM)

- ✓**It is a software processor – differ from platform to platform - an interpreter for byte code.**
- ✓**Includes the Byte Code Loader and the Java Interpreter.**
- ✓**The Byte Code Loader on the client machine will pick up the Byte Code file and validate it.**
- ✓**The validated byte code files are passed to the java Interpreter, which translate the files into executable instructions for the type of computer hardware and operating system.**



Variable: Variables are the basic unit of storage in a program. It is treated as an identifier to the memory location. A value in a variable can change during the program execution.

Declaration: type identifier; or type identifier = value;

Where type specifies the data type and identifier specify the name of the variable.

Example: int a, b; float f = 12.45f;

Data type:

Integer:

1. byte: The smallest integer type is byte. Memory size is 8 bit. Range of byte is -128 to +127.

Example: byte A, B; //where A and B are variables of type byte.

2. short: Short data type is mostly applicable to 16 bit computer. Range of short is -32,768 to + 32,767. Memory size is 2 bytes. (16 bit).

Example: short c, d; // where c and d are variables of type short.

3. int: The most commonly used integer type is int. Range of int is –2,147,483,648 to +2,147,483,647. Memory size is 4 bytes.

Example: int x, y; // where x and y are variables of type int.

4. long: Long data type is useful when an int type is not large enough to hold the desired value. Memory size of long is 8 bytes. (Range is: (-2 power 63) – to - (2 power 63-1)

Example: long r, s; // where r and s are variables of type long.

Floating-point types:

Floating point types are also known as real numbers. Floating point is used, when you are evaluating expressions that require fractional precision. There are two types.

1. float: Float can be useful when representing dollars and cents. In float the precision part is very small. Memory size is 4 bytes.

Example: float d; //where d is a variable of type float. d = 42.5f;

2. double : Double data types use 8 bytes to store a value. The mathematical functions sin (), cos(), sqrt () etc returns a value of type double.

Example: double pi; // where “pi” is a variable of type double. pi = 3.1416;

Note: Default scope of floating point data type is double.

Program

```
class Area
  {   public static void main(String arg[ ] )
        {     double pi, r, A;
              pi = 3.1416;
              r  = 3.2;
              A = pi * r * r;
              System.out.println("Area of circle is :" + A);
        }
  }
```

Character:

Single character enclosed in single quotes. Java support UNICODE character set. Unicode character set contains all human languages available in the world. Memory size is 2 bytes per character. Range of Unicode char set is 0 to 65,536.

Example:

```
char ch1, ch2;
ch1= 88; ch2 = 'Y';
System.out.println(" Ch1 is :" + ch1 + "Ch2 is :" + ch2 );
```

Output: Ch1 is : X Ch2 is : Y

Boolean:

This type is return by all relational operator such as (A < B). Memory size is 1 bit. (true or false). Conditional expressions in ‘ if ’ and ‘ for ‘ require boolean type. Default value is false.

Example: boolean b = true;

```
if(b)
    System.out.println("value of b is true");
else
    System.out.println("Value of b is false");
```

Array: Array is a group of same type values that are referred by a common name.

Declaration: type var_name [ ];

Where type specifies the data type of the array and var-name specify the name of the array. **"new"** is a special type operator that can be used to allocate memory for any array variable.

General format: `var_name = new type[size];`

Example: 1. int mark[ ]; mark = new int[20];

2 int mark[ ] = new mark[20];

3. int mark[ ] = { 78,29,34,87.....};

Average of N numbers using command line argument:

```
class ArrayDemo
{
   public static void main ( String Arg[ ] )
```

```
        {
            int Num[ ],Len;
            float Sum=0.0f,Avg=0.0f;
            Len  = Arg.length;
            Num =  new int[Len];
            for(int i=0 ; i<Len; i++)
               {    Num[i] =  Integer.parseInt (Arg [i]);
                    Sum = Sum+Num[i];
               }
           Avg = Sum / Len;
           System.out.println ("Average  is # : "+Avg );
        }
 } //Run : java ArrayDemo 1 2 3 4 5
```

Multi-dimensional Array:

Multidimensional arrays are actually array of arrays. To declare a multidimensional array variable we must specify each additional index using another set of square brackets.

Declaration: type var-name [ ] [ ] = new type[row] [col];

Example:

```
1. int matrix [ ] [ ] =  new int[3][3];
     // Allocate 3 rows and 3 columns for the variable matrix.
2. int matrix[ ] [ ];
     matrix = new int[2][3];
```

// Allocate 2 rows and 3 columns for the variable matrix;

Program

```
class ArrayDemo
{
    public static void main(String Arg[ ])
        {  if(Arg.length<2)
            {   System.out.println("Error : Run : java ArrayDemo1 <R><C><Value>");
                return ;
            }
          int R,C;
          R = Integer.parseInt (Arg[0]);
          C = Integer.parseInt (Arg[1]);
          if(Arg.length < (R*C)+2)
            {    System.out.println(" Error : Insufficient array elements.....");
                 return ;
            }
         int i,j,k=2;
         int matrix[ ] [ ] = new int[R][C];
         for(i=0;i<R;i++)
             for(j =0;j<C;j++)
                   matrix[i][j] = Integer.parseInt (Arg[k++]);
         for(i=0;i<R;i++)
            {
```

```
                for (j= 0;j<C;j++)
                    System.out.print(matrix[i][j]+"  ");
                System.out.println();
            }
        }
    } //Run:  java ArrayDemo 2 2 4 3 2 5
```

Another way of array declaration: type [ ] var-name;

Example: 1. int [ ] A = new int[10];

2. int [ ][ ]B = new int[2][4];

String:

- Sequence of characters enclosed with in double quotes.
- **Example:** String str; where str is a object of type String. Here String is a predefined class.
  str = "welcome";

Type Conversion or Type Casting:

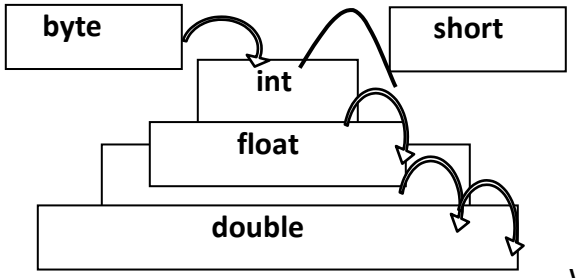
Convert one data type into another data type such as float to int or double to int. i.e., to create a conversion between two incompatible types we must use a casting operator.

General format:

- **(target-type) value;** // where target-type specify the desired type to convert the specified value.
- Conversion is necessary in java. The destination type is larger than the source type.

Example:

1. int A; float F = 12.95f; A = (int)F;
2. byte B; int A = 123; B = (byte)A;
3. byte a = 10 , b = 20; byte c = (byte) (a+b);



Water- falls model

Operators:

Arithmetic Operators **Bitwise Operators (Bitwise logical** operators)

Operators	Meaning	Operators	Meaning
+	Addition	~	unary NOT (Bitwise Complement)
-	Subtraction	&	AND (Bitwise and)
/	Division	|	OR (Bitwise or)
%	Modulation	^	X-OR (Bitwise x-or)
*	Multiplication	>>	Right Shift
++	Increment	<<	Left Shift
--	Decrement	&=	Bitwise AND Assignment
+=	addition assignment	|=	Bitwise OR Assignment
- =	Subtraction assignment	^=	Bitwise X-OR Assignment

/=	division assignment	>>=	Bitwise Right shift Assignment
%=	Modulation assignment	<<=	Bitwise Left shift Assignment
*=	Multiplication assignment		

Table for Bitwise logical operator:

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

Example: int a = 3 , b = 6;

int c = a | b; int d = a & b; int e = a ^ b; int f = ~a;

Left shift operator (<<):

- The left shift operator shift all bits in a value to the left a specified number of times.
- **General Format: value << num;**
- Where 'num' specify the number of position to left shift the value. The operator '<<' move all the bits in the specified value to the number of bit positions specified by 'num'.
- **Example:** int a = 7; a = a << 2;

Right shift operator (>>)

- The right shift operator shift all bits in a value to the right a specified number of times.
- **General Format: value >> num;**
- Where 'num' specify the number of position to right shift the value. The operator '>>' move all the bits in the specified value to the number of bit positions specified by 'num'.
- **Example:** int a = 7; a = a >> 2;

Relational Operators:

Relational operators determine the relationship between the two operands. The relational expressions return boolean type result. The relational operators are mostly used in "if" statements and various loop statements.

Relational Operators **Boolean Logical Operators**

Operator	Meaning	Operator	Meaning
= =	equal to	&	logical AND
! =	Not equal to	|	logical OR
<	Less than	^	logical X-OR
>	grater than	&&	short-circuit AND
<=	Less than or equal to	||	short-circuit OR
> =	gather than or equal to	!	logical unary NOT
		&=	AND assignment
		| =	OR assignment
		^ =	X-OR assignment
		= =	equal to
		!=	not equal to

Table for Boolean Logical Operator:

A	B	A|B	A&B	A^B	~A
False	False	False	False	False	True
False	True	True	False	True	True
True	False	True	False	True	False

True	True	True	True	False	False

Example: int A=30, B=20;

boolean C = (A > B);

System.out.prinln ("Value of C is: " + C); **output:** Value of c is: true

Example: boolean A = true, B = false;

boolean C = A & B; boolean D = A ^ B;

System.out.println (" value of c is "+C); **output:** value of c is: false

System.out.println (" value of d is:"+D); **output:** value of d is: true

Assignment operator:

- The assignment operator is the single equal sign.
- **General format: var = expression;**
- It is used to assign the value of the expression to the 'var'. '= ' is the assignment operator. Here, the type of variable must be suitable with the type of expression.
- **Example :** int a , b, c; a = b + c; a = b = c = 50; (chain assignment)

Conditional operator (or) Ternary operator

- **expression1? expression 2 : expression 3**
- When expression1 is true, then expression2 is evaluated .Otherwise expression3 is evaluated.
- **Example:** int A , B, C;
  int D = (A>B)? (A-B): (A+B);

Expressions: An expression is a combination of operators, constants and variables arranged as per the rules of the language.

Types Expressions:

- Constant Expressions (Ex: char ch = 'S';)
- Integral Expressions (1. integer expression 2. char expression)
- Floating Expressions (Ex: float f = 23.5 + 9.25;)
- Relational Expressions (Ex: if(X<Y))
- Logical Expressions (Ex: if ((X>Y) && (X>Z)))
- Bitwise Expressions (Ex: A<<2;)
- Compound Expressions (combination of the above expressions)

Control Flows

Selection Statements:

1. Simple if statement:

Syntax:

```
if (condition)
    Statement 1;
```

Value return by the condition is either true or false. If the condition is true, then statement1 is executed. Otherwise statment2 is executed. If the statement is single, then parenthesis ({ }) is not necessary. Compound statements must be enclosed within the parenthesis.

2. Nested if statement:

- One if statement contain another if statement.

Syntax:

```
if (conditon1)
    {  if (condition 2)
            statement1;
       else
```

If condition1 is true, then condition2 is executed. If condition2 is true then statement1 is executed. Otherwise statement2 is executed. If conditio1 is false, then statement3 is executed.

3. if - else - if ladder:

```
if(condition1)
        statement1;
if(condition2)
        statement2;
.....          ......
```

4. Switch statement

```
switch ( expression )
  {
    case label1:   //statements
                   break;
    case label2:   //statements
                   break;
```

The result of expression may be char or short or byte or int. The value of the expression is compared with case labels. If any case label match with the expression, then the set of statements followed by the case label will be executed. If any one of the case labels are doesn't match with the expression, then the default statements will be executed.

Iteration Statements:

1. While Loop:

```
while (condition )
  {
    ..... ..... .....
    ..... ..... ..... // body of the
```

The set of statements will be executed as long as the value of condition is true. The condition can be any Boolean expression. When the condition becomes false, then the control goes to the next statement followed by the loop. If the condition is initially false, then the body of the loop will not be executed at all.

2. Do - While Loop:

```
do
  {
     ....  .......
```

The do - while statement always executes the body of the loop, at least one's. Because it's conditional expression is evaluated at the bottom of the loop. Every time do -while loop first execute the body of the loop and then evaluate the conditional expression. If the condition is true, then the loop will be repeated. Otherwise the loop will be terminated.

3. for Loop:

```
for (initialization; condition; iteration)
  {
     ......  ........  ........ // body of the loop
```

- The initialization part executed first and only one time (beginning of the loop) and set the initial value of the control variable.
- The condition part will be executed second. This must be a Boolean expression. If the condition is true, then the body of the loop will be executed. Otherwise the loop will be terminated.
- The iteration part is an expression, that increment or decrement the loop control variable.

Jump Statements:

1. Break: The break statement has three uses.

1. Terminates a statement sequence in a statement.

2. It can be used to exist from a loop.

3. It can be used as a civilized form of **goto** statement.

Example:

```
1.  for ( int i=0 ;i<10 0;i++)
        {   if(i= =10)  break;
            System.out.prinln ("value of i is:"+i);
    }
2.      int n=0;
    while (n<20)
    {   if( n = = 10)   break;
            System.out.println ("value of n is:" + n);
            n++;
    }
```

- The general format of the break label statement is: **break label;**
- where 'label' is a name that identify a block of statements.' label' is any valid java identifier followed by a colon(**:**)

Break Label:

```
class BreakDemo
  {  public static void main (String Arg[ ])
     {      boolean b = true;
            first:  {
              second:     {
                    third:       {
                                        System.out.println(" Block 3 is executed ");
```

```
                    if( b ) break second;
                }
                System.out.println("Block 2 not executed ");
            }
            System.out.println("Block 1  executed ");
        }
    }
}
```

Output: Block 3 is executed

Block 1 is executed

2. Continue:

In 'while' and 'do- while' loops a 'continue' statement transfer the control directly to the conditional expression. But in a 'for' loop control first go to the iteration part and then go to the conditional expression.

A continue statement may specify a label to determine which enclosing loop to continue.

Example:

```
for ( int i = 0; i<20 ; i++ )
    {   System.out.println (" i : " + i );
        if ( ( i % 2 ) = = 0 )  continue;
            System.out.println ( );
    }
```

Continue label:

```
class ContinueDemo
  { public static void main (String Arg[ ])
        {
         first:
              for( int i=0;i<10;i++)
               {
                  for(int j=0;j<10;j++)
                     {  if( j>i )
                           {   System.out.println( );
                                continue first;
                           }
                       System.out.print((i*j)+"   ");
                    }
               }
           System.out.println( );
        }
  }
```

3. Return:

A return statement is used to transfer the control back to the caller of the method. Used to return a value to the calling method. A return statement immediately terminates the method is which it is executed.

Syntax:

```
return ( value );
```

Example:

```
boolean b = true;
System.out.println(" Before return statement");
if ( b ) return;
System.out.println(" After return statement");
```

The Java Buzzwords – key features of the Java language

1. **Simple:** Basic knowledge of C and C++ is necessary to write java programs.
2. **Secure & Portable:** Can't create **virus** programs using java language - java compiler doesn't give machine code directly - gives an intermediate code called "Byte Code" or "Text Code" - Can move a java program from one system to another system easily.

3. Object Oriented

- Uses the object oriented programming paradigm

4. Robust

- Execute reliably in a variety of systems
- Strictly typed language: checks the code at compile time & run time
- Main reasons for program failure: (a) memory management mistakes, (b) mishandled exceptional conditions (run-time errors)
- Eliminates these problems by, automatic memory management (deallocation is completely automatic) - provides garbage collection for unused objects - Java handles runtime errors (Exception Handling)

5. Multithreaded

- Java was designed to create interactive and networking programs
- Supports multithreaded programming – programs do many things simultaneously
- Simultaneous execution of more than one process
- **Synchronization**: enables to write smoothly running interactive programs

6. Architecture-neutral

- Programmer's problem: program written today may not run tomorrow – even on the same machine
- Upgrades in: Operating System, Processor, changes in core system resource does not affect JVM.

Goal: Write Once; Run Anywhere, any Time, Forever

7. Interrupted and High Performance:

- Typically a computer language is either compiled or interrupted
- But, Java combines both of them
- creation of cross-platform programs by compiling into an intermediate representation – Byte Code: Interpreted by any system that provides a JVM

8. Distributed

- Java is designed for the distributed environment of the Internet
- **Remote Method Invocation**: Objects on two different computers can execute procedures remotely

Note:

✓ **Java's influence is the C# language**

- ✓ **Developed by Microsoft to support .NET framework**

Basic concepts of Java

- ✓ Java support **UNICODE** character set - contains all human languages.
- ✓ Range: set: 0 - 65,535 / 2 bytes per character.
- ✓ Java is case sensitive language.
- ✓ **Class Name:** All weight letters (first char in a word) should be in upper case.
  String, **D**ata**I**nput**S**tream and **S**ystem
- ✓ **Method Name:** All weight letters except the first one should be in upper case.
  draw**L**ine( ), read**L**ine( ) and parse**I**nt( )

- Java can be used to create two types of programs.
  1. **Application:** program that runs on your computer under the operating system.
  2. **Web Program:** program design to be transmitted over the internet and executed by a java compatible web server

(1) JDK (JAVA DEVELOPMENT KIT)

It includes a number of development tools that are used for developing and running java programs.

The tools available in the JDK are follows:

1. javac (java compiler)

This translates the java source program to byte code files.

2. java (java interpreter)

This runs applications by loading and interpreting the byte code file.

2. API (Application Program Interface)

- This includes hundreds of classes and methods grouped into several functional packages.
- Most commonly used packages are,

1) Language support package: (java.lang)

Require to implementing basic feature of java. This package is automatically imported by the java compiler.

2) Utilities package (java.util)

Provide utility functions such as date and time functions.

3) Input/ Output package (java.io)

Require for implementing Input/Output manipulation.

4) Networking package (java.net)

Used for communicating with other computers via internet.

5) AWT package (java.awt)

Abstract Widow Toolkit - Require to implement platform independent GUI applications.

6) Event package (java.event)

Require to implement event handling programs.

7) SQL package (java.sql)

Support sql query.

8) Math package (java.math)

Provide mathematical representations.

The JDK Directory tree structure:

When JDK is installed onto a computer it creates its own subdirectories and copies files into these sub directories appropriately.

The directory tree structure is generally as follows:

Directory	Contents of the Directories
\java\bin	Programs that make up the Java Development Kit. The compiler, Debugger and so on.
\java\demo	Demonstration programs
\java\include	Header files for use in combining Java and C/C++ programs.
\java\jre	It stands for Java Runtime Environment
\java\lib	The standard Java class supplied with the

	JDK
\java\src	It contains all the source files for the Java packages. It is initially stored as a src.jar file.

(2) JDK (JAVA DEVELOPMENT KIT)

It includes a number of development tools that are used for developing and running java programs. The Java Development Kit is character based and not graphical like Visual Basic.

The tools available in the JDK are follows:

1. javac (java compiler)

This translates the java source program to byte code files.

2. java (java interpreter)

This runs applications by loading and interpreting the byte code file.

3. appletviewer

To run java applets

4. javadoc

Create HTML format document from java source code file.

5. javah

That is C header and Stub file creator.

6. javap (java disassembler)

This enables us to covert byte code files into a program description.

7. jdb (java debugger)

This helps us to find errors in our programs.

2. API (Application Program Interface)

- This includes hundreds of classes and methods grouped into several functional packages.
- Most commonly used packages are,

(2) API (Application Program Interface)

- This includes hundreds of classes and methods grouped into several functional packages.

Most commonly used packages are

1) Language support package: (java.lang)

Require to implementing basic feature of java. This package is automatically imported by the java compiler.

2) Utilities package (java.util)

Provide utility functions such as data and time functions.

3) Input/ Output package (java.io)

Require for implementing Input/Output manipulation.

4) Networking package (java.net)

Used for communicating with other computers via internet.

5) AWT package (java.awt)

Abstract Widow Toolkit

Require to implement platform independent GUI applications.

6) Applet package (java.applet)

Require to crate java applets.

7) Event package (java.event)

Require to implement event handling programs.

8) Image package (java.image)

Support operations in image.

9) SQL package (java.sql)

Support sql query.

10) Math package (java.math)

Provide mathematical representations.

The JDK Directory tree structure:

When JDK is installed onto a computer it creates its own subdirectories and copies files into these sub directories appropriately. The directory tree structure is generally as follows:

Directory	Contents of the Directories
\java\bin	Programs that make up the Java Development Kit. The compiler, Debugger and so on.
\java\demo	Demonstration programs
\java\include	Header files for use in combining Java and C/C++ programs.
\java\include-old	Contains the Header files use by JDK1.1
\java\jre	It stands for Java Runtime Environment
\java\lib	The standard Java class supplied with the JDK
\java\src	It contains all the source files for the Java packages. It is initially stored as a src.jar file.

- ✓ A just-in-time (JIT) compiler is a program that turns Java bytecode into instructions that can be sent directly to the processor.
- ✓ In the past, most programs written in any language have had to be recompiled, and sometimes, rewritten for each computer platform.
- ✓ Biggest advantages of Java are that you only have to write and compile a program once.
- ✓ The JVM on any platform will interpret the compiled bytecode into instructions understandable by the particular processor.

- ✓ However, the virtual machine handles one bytecode instruction at a time. Using the Java just-in-time compiler (really a second compiler) at the particular system platform compiles the bytecode into the particular system code.
- ✓ Once the code has been compiled by the JIT compiler, it will usually run more quickly in the computer.
- ✓ The just-in-time compiler comes with the virtual machine - used optionally - compiles the bytecode into platform-specific executable code that is immediately executed.

- ✓ Sun Microsystems suggests that it's usually faster to select the JIT compiler option, especially if the method executable is repeatedly reused.
- ✓ used to improve the performance - compiles parts of the byte code that have similar functionality at the same time and reduces the amount of time needed for compilation.
- ✓ the term compiler refers to a translator from the instruction set of a JVM to the instruction set of a specific CPU.

JVM (Java Virtual Machine)

An abstract machine - specification that provides runtime environment in which java bytecode can be executed

JVMs are available for many hardware and software platforms - JVM is platform dependent

What is JVM?

It is **a specification** - working of JVM is specified - Implementation has been provided by Sun and other companies.

An implementation - known as JRE (Java Runtime Environment)

A Runtime Instance - an instance of JVM is created when the java command is prompted to run the java class

What it does?

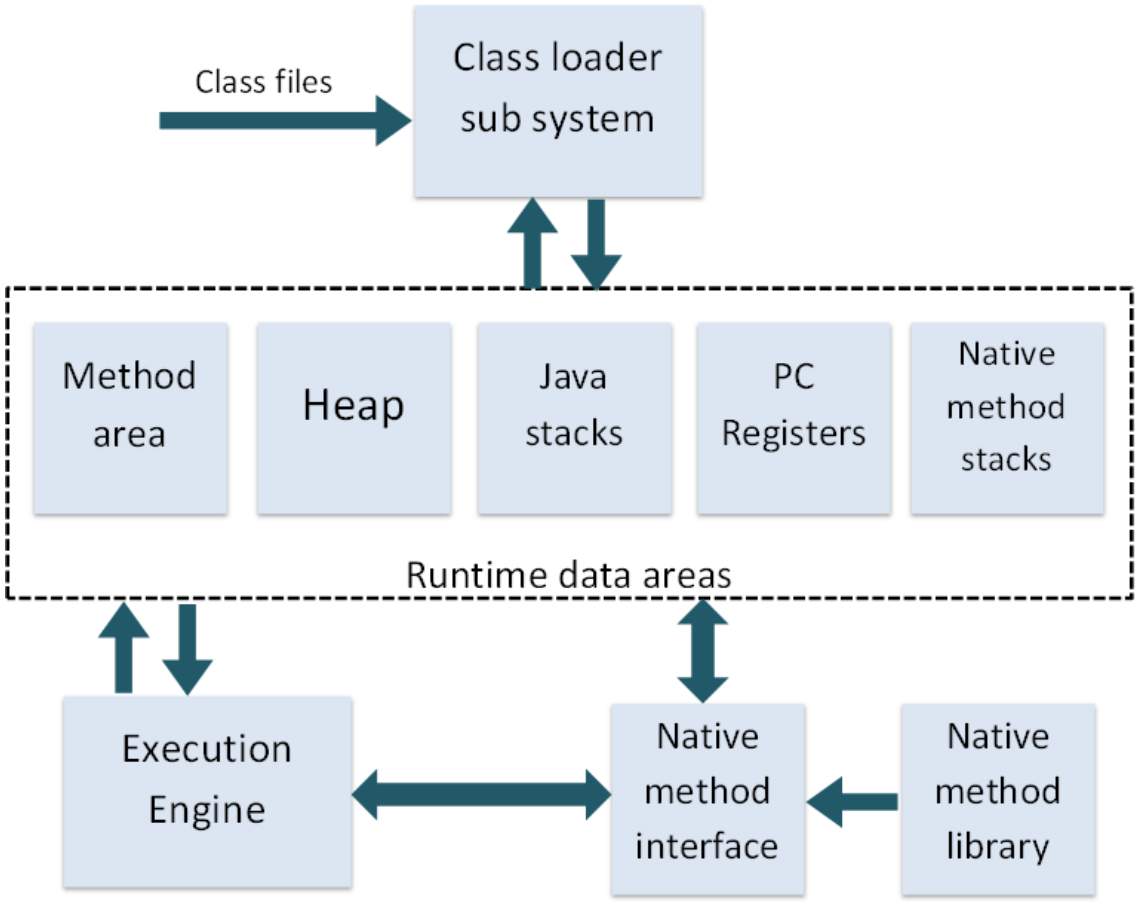
The JVM performs following operation:

- Loads code - Verifies code
- Executes code - Provides runtime environment

JVM provides definitions for the:

- Memory area - Class file format - Register set -Garbage-collected heap - Fatal error reporting

Internal Architecture of JVM



1) Class loader: is a subsystem of JVM that is used to load class files.

2) Class (Method) Area: stores per-class structures such as the runtime constant pool, field and method data, the code for methods.

3) Heap: is the runtime data area in which objects are allocated.

4) Stack: stores frames - holds local variables and partial results, and plays a part in method invocation and return - each thread has a private JVM stack, created at the same time as thread - a new frame is created each time a method is invoked - a frame is destroyed when its method invocation completes.

5) Program Counter Register: contains the address of the Java virtual machine instruction currently being executed.

6) Native Method Stack: contains all the native methods used in the application.

7) Execution Engine: includes (a) a virtual processor, (b) Interpreter: Read bytecode stream then execute the instructions, (c) Just-In-Time (JIT) compiler.

Output Function:

General Format:

System.out.println (value); or System.out.print (value);

System.....> Predefined class

out .....> Output Stream Class object connected to the CONSOLE

println ()...> Output Stream Class method

Example:

```
int A =10, B =20, C;
C = A + B;
System.out.println ("Value of A:" + A);
System.out.println ("Value of B:" + B);
System.out.println ("Value of C:" + C);
```

Or

```
System.out.println (" A: " + A + " B: " + B +" C: "+ C);
System.out.println (A + B + C);
```

Output is: 60

IDE: (Integrated Development Environment)

For C and C++ IDE are TC and TCC.

IDE is not available for java - So we type java program in a Textpad or Notepad or any text editor

Recent IDEs: Eclipse / NetBeanns / JBuilder / JDeveloper / JCreator etc..

Java does not support,

- Signed and unsigned key words.
- Struct, union, enum key words.
- Preprocessor commands. (#include, #define and typedef).

- sizeof( ) and delete operator.
- goto statement
- Explicit pointers
- Automatic type conversion
- Global variables and functions
- Function with default arguments.

Structure of java program:

- Documentation section
- Package declaration section
- Package import section.
- Interface definition section.
- Class definition section.

Optional

- Main class definition section. (Necessary)

NOTE:

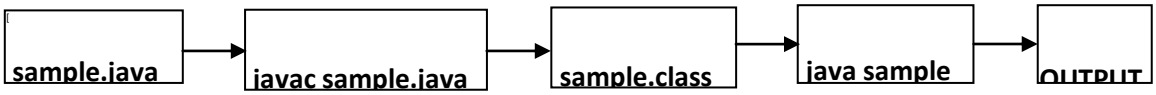
C and C++ main function.....> main (int argc, char *argv [ ]) → Optional

Java main function .....> main (String arg [ ]) → Necessary

Program

```
class sample
 {
    public static void main (String arg[ ] )
     {
            System.out.println ("WELCOME");
      }
 }
```

File Name: sample.java (file name and main method class name must be same)



public: Allows the programmer to control the visibility of class members. When a class member is preceded by public then that member may be access by outside of the class. In this case main must be public.

static: Allows main() to be called with out having the instance of the class. This is necessary, since main method is called by the JVM before any objects are made.

void: Tells the compiler that the main method does not return any value.

String arg []: The main method contains a parameter arg[] of type String. arg[] receives the command line arguments during the program execution.

Command Line Argument:

- Passing argument to the main method during the runtime is called as command line argument.

Program for command line argument

```
class CmdLine
 {
   public static void main(String arg[ ] )
     {
         int count;
         count=arg.length;
         System.out.println("Total Number of Argument is : " + count);
         for(int i= 0;i<count;i++)
              System.out.println(" Argument # " +( i+1)+ "is :" + arg[i] );
    }
}
```

File Name: CmdLine.java

Compile: javac CmdLine.java

Run: 1. java CmdLine this is my first command line argument program

2. java CmdLine * //display all files in the local directory.(just like 'dir')

3. java CmdLine C:\windows* // display all files in the c:\windows directory.

Program

```
class AddDemo
```

```
 {
   public static void main(String arg[ ] )
     {
            int a,b,c;
               a =Integer.parseInt(arg[0]);
            b =Integer.parseInt( arg[1] );
            c =a+b;
            System.out.println(" Sum is  # : " +c );
    }
}
// run: java AddDemo 30 45
```

CLASS

```
class <class name>
 {
    scope type var1;
    scope type var2;
    . . ....
    scope returntype methodname(Argument list)
     {
```

Object:

```
classname <object name> = new
classname();
```

or

Scope (Access Control)

Accessible From (With in a package)	public	private	protected	Default (friendly)
Own class	Yes	Yes	Yes	Yes
Sub class	Yes	No	Yes	Yes
Non subclass	Yes	No	Yes	Yes

Program

```
calss Sample
  {
        public int A;
        private int B;
        int C;
        void setdata(int x)
          {  B = x;   }
         void show ( )
          {
              System.out.println(" Value of A is :"+ A);
              System.out.println(" Value of B is :"+ B);
              System.out.println(" Value of C is :"+ C);
          }
  }
class TestAccess
    {
        public static void main(String arg[ ])
           {
              Sample S = new Sample();
              S.A = 100;
              S.B =20;
              S.setdata(20);
              S.C=95;
```

```
            S.show();

        }

    }
```

Constructor

- Constructor is special type member function which executes automatically whenever an object is created.

- **Rules for creating constructor:**

  - Name : Constructor name is same as class name.
  - Scope : public / default
  - Return Type : Nil
  - Argument : Possible
  - Overloading : Possible

Uses:

1. Initialization
2. Resource (Memory) Allocation

The Finalize () method (just like destructor in C++)

Syntax:

```
protected void finalized ( )
   {
        // statements
```

Uses:

1. Resource De-allocation
2. Cleanup process

Program for Constructor and Constructor Overloading

```
class Time
  {   int Hr, Min;
      Time( )                 //  No Argument Constructor (default)
        {   Hr = 0;
            Min = 0;              }

      Time(int H, int M)   // Two Argument(int type) Constructor
        {   Hr   = H;
            Min = M;              }

      Time (double T)      // One Argument (float type) Constructor
        {  Hr = (int)T;
           Min = (int) ((T-Hr)*100);           }

      void display ( )
       {   System.out.println (" Hr: " + Hr +": "+"Min:"+ Min);   }
  }
class TestTime
   {   public static void main(String arg[ ])
         {     Time T = new Time( );       // Invoke the Default Constructor
               T.display();
               T = new Time(4,20);         // Invoke the 2 argument constructor
```

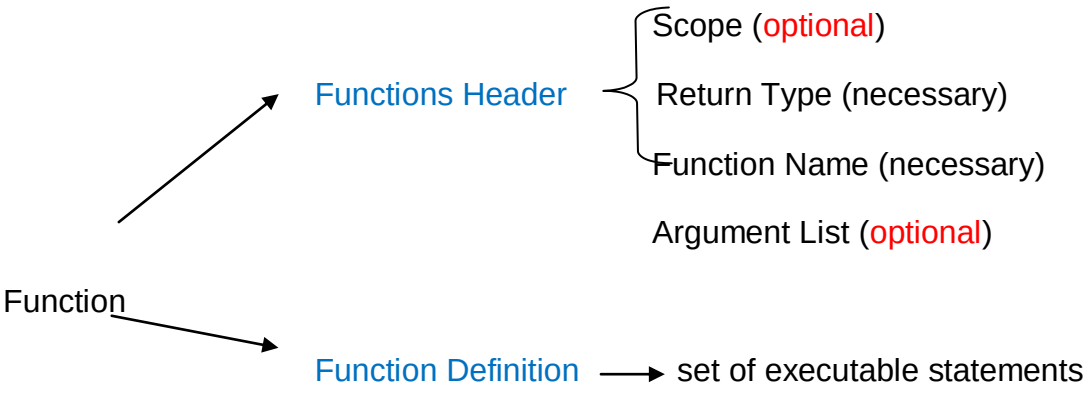
```
                T.display();
                Time T1 = new Time(4.12);  // Invoke the 1 argument constructor
                T1.display();
            }
    }
```

Program for finalize () method

```
class Test
    {   Test(  )
            {  System.out.println( "Inside Constructor ");          }
        protected void finalize ( )
            {  System.out.println( "Inside Finalize Method ");  }
    }
class TestFinalize
    {   public static void main (String arg[ ])
        {
            Test T1 = new Test( );
            Test T2 = new Test( );
            System.gc();            // call the garbage collector
            // Or T1.finalize();     // call the garbage collector
            Test T3 = new Test( );
        }
 }
```

Function:

```
scope returnType  MethodName( Arg List )
  {
        // body of the method
```



Program

```
class   FnDemo
   {    static int MAX ( int A, int B)
            {
                if( A > B)    return A;
                return B;                          }
```

```
        int MIN (int A, int B)
          {   if( A< B)    return A;
              return B;                          }


      public static void main(String arg[ ])
         {
            int A =30 , B = 20;
            FnDemo F = new FnDemo( );
            int Big = MAX( A , B);
            int Small = F.MIN( A , B);
            System.out.println( " Biggest Number is :" + Big );
            System.out.println( " Smallest Number is :" + Small );
         }
  }
```

Function Overloading

Function Overloading means, Same Function Name with different number of arguments and different type of arguments.

```
class   FnOverDemo
 {
    static void MAX ( int A, int B)
       {   if( A > B)
             System.out.println( " Biggest Number is :" + A );
          else
             System.out.println( " Biggest Number is :" + B );
       }

     static void MAX( float A, float B)
     {    if( A< B)
             System.out.println( " Biggest Number is :" + A );
          else
             System.out.println( " Biggest Number is :" + B );
     }

    static void MAX( int A, float B)
     {   if( A< B)
           System.out.println( " Biggest Number is :" + A);
```

```
        else
            System.out.println( " Biggest Number is :" + B);
    }

  public static void main(String arg[ ])
    {
        int A =30 , B = 20 , C = 40;
        float D = 25.26f , E = 10.5f , F = 45.2f;
        MAX( A , B);
        MAX( D , E);
        MAX( C , F);
    } // if the methods are non-static?
}
```

"this" pointer: "this" is a special type pointer variable, which contain the address of the current working instance.

Example:

Time T1, T2, T3;

T1.Sum (T2, T3); Here address of "this" is same as the address of the object T1. So T1.Hr is same as this.Hr.

Function Argument as an Object:

```
class Time
  {
      int Hr ,Min , Sec;
      Time( )
         {  Hr = 0;
            Min = 0;
            Sec = 0;
         }
      Time( int Hr , int Min , int Sec )
        {
              this.Hr = Hr ;
            this.Min = Min ;
            this.Sec = Sec ;
         }
      void SUM(Time  S, Time T )
      {
       Hr = S.Hr + T.Hr;
       // this.Hr is same as Hr, both of them refer the object T3.
        Min = S.Min + T. Min;
        Sec = S.Sec + T.Sec;
        if( Sec >60 )  {  Sec -= 60;  Min++ ;   }
        if( Min > 60 ) {  Min -= 60;  Hr++ ;    }
        if( Hr > 12 )    Hr -= 12;
```

```
        }
    void show( )
      {  System.out.println( Hr + " : " + Min + "  :  " + Sec );          }
}

class TestTime
{
    public static void main ( String arg[ ])
       {
            Time T1 = new Time (3, 20, 40);
                Time T2= new Time (3, 40, 30);
                Time T3 = new Time ( );
                T3.SUM( T1, T2 );
             // try for  T1. SUM ( T2 );
                System.out.print("Object T1 is     ");
                T1.show();
                System.out.print("Object T2 is     ");
                T2.show();
                System.out.print("T1 + T2   is     ");
                T3.show ();
       }
}
```

Functions return type as an Object:

```
class Complex
  {
     int Rp , Ip;
       Complex( )    {        }   // default constructor
       Complex(  int r ,int i)
          {       Rp = r;
                   Ip = i ;
           }
     void show ( )
          {
             if( Ip>0)
                        System.out.println( Rp + "+" + Ip + 'i');
               else
                        System.out.println( Rp +" " +Ip + 'i');
           }
     Complex SUM ( Complex C )
       {
           Complex T =new Complex( );
           T.Rp = this.Rp + C.Rp ;
           // Rp is same as this.Rp, both of refer the object C1
           T.Ip = this.Ip + C.Ip ;
           return  T;
```

```
        }
    }
class TestComplex
  {
    public static void main ( String arg[ ])
        {
            Complex C1 = new Complex( 4, 5);
              Complex C2 = new Complex( 4, -7);
              C1=C1.SUM(C2);    // C1 = C1 + C2;
              C1.show();
        }
   }
```

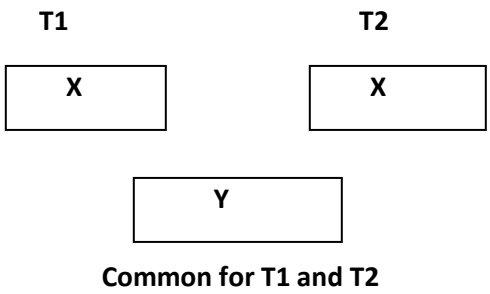
STATIC

1. Static data:

Memory is allocated for static data only one time. That memory location is share by the entire objects created by the class.

static data_type var_name;

```
class Test
  {    int X;
       static int Y;
  }
 Test T1 = new Test ( );
 Test T2 = new Test ( );
```



2. Static method

```
static return type methodName (Arg's )
      {
            // functiion definition
```

- Static method are use to operate other static member (both data and methods) of a class.

- We cannot operate non-static members inside the static method. (Using object reference it is possible)

- We can operate both static and non-static members inside a non-static method.

- We can call a static method without any object reference.

```
classname . methedname ( );
```

(Or)

```
classname . variablename = value ;
```

Program

```
class Test
  {   int   X ;
       void show()
        {
           System.out.println("Inside Show Method");
           System.out.println(" Value of X is :" + X);
        }
      public static void main (String  S [ ] )
        {       Test T = new Test ();
                T.X = 305;
                T. show();
        }
  }
```

Program

```
class Test1
 {    static int X;
      static void show()
        {      System.out.println ("Inside show method ");
               System.out.println ("Value of X is: " +X);
        }
```

```
    public static void main ( String s [ ] )
      {
            X = 100;
            show ();
      }
}
```

Program

```
class Test2
{   static int X;
    static void show()
      {  System.out.println("Inside show method ");
         System.out.println("Value of X is :"+ X);
      }
}
class TestStaticMethod
  {   public static void main (String s[ ] )
        {       Test2.X = 100;
                Test2.show();
        }
  }
```

// **for non-static?**

3. Static block

- Static block is executed first
- This block is executed only once, whenever the class is loaded
- This is used to initialized other static members of the class
- Static block can available anywhere in a program.

```
static
   {
        …….  …
                    // body of static block
```

Program

```
class TestStaticBlock
 {
     static int A =5;
     static int B ;
     TestStaticBlock ()
       {
           System.out.println("Inside Constructor");
       }
     static void show()
      {   System.out.println("Inside show method ");
```

```
        System.out.println("Value of B is:"+B);
    }
  static
    {
        System.out.println("Inside static block ");
        B = A*3;
    }
  public static void main ( String s [ ] )
    {
        TestStaticBlock T = new TestStaticBlock ();
        show();
    }
}
```

OUTPUT:

```
S:\>java Test
Inside static block
Inside Constructor
Inside show method
Value of B is: 15
```

Program for System.out.println()

```
class MyOutput
  {
     void myprintln()
       {       System.out.println("Inside println method");        }
 }

class MySystem
 {
      static MyOutput myout = new MyOutput();
 }

class TestOutput
  {
     public static void main(String s[])
       {
            MySystem.myout.myprintln();
      }
 }
```

Inheritance

Inheritance is the process of creating new class or classes from an already existing class. The existing class is known as **super class** (base class). The new class is known as **sub class** (extended class or derived class).

class <sub class name > extends <super class name>

{

…. ……..

…… ……….. // body of sub class

}

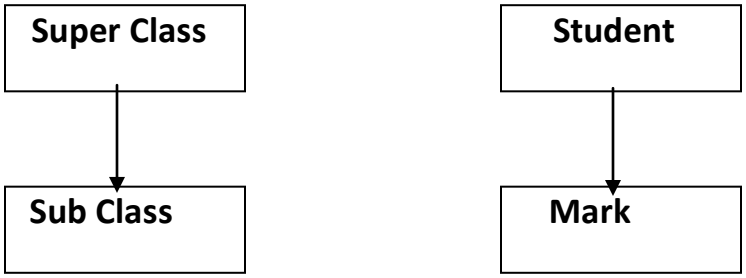
Advantage

- Code Reusability
  - Reduce the debugging time
  - Reduce the testing time
  - Reduce the development cost and manpower
  - Reduce the maintenance charge

Types of Inheritance

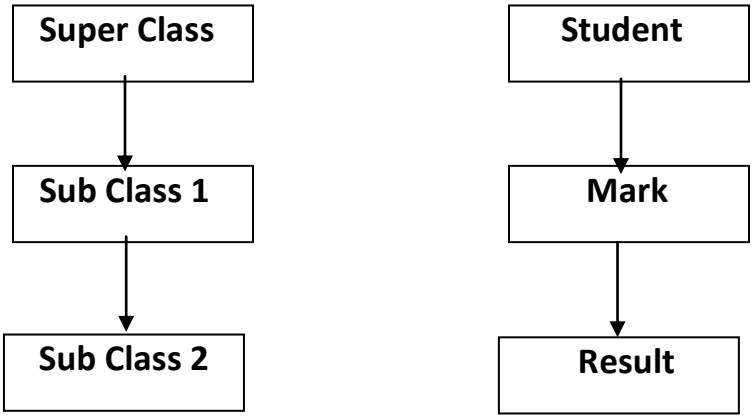
Single Inheritance

A class derived from a single base class



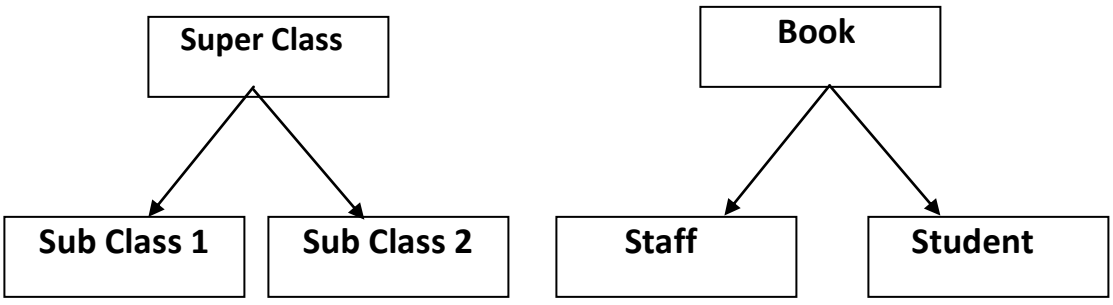
Multilevel Inheritance

A class derived from an already existing derived class.



Hierarchical Inheritance

More than one Class is derived from a single base class.



Note: Java does not support multiple inheritance and hybrid inheritance using class. But it is possible with interface.

Method overriding

A function in a sub class having the same name, argument list, scope, and return type as like its super class is called method overriding.

Passing Argument to Super Class Constructor through sub Class Constructor

class Base

```
{   private int  x, y;
```

```
    Base()                  {     x =0;       y = 0;       }
    Base( int a , int b)    {     x  = a;     y  = b;      }
    void show ( )           {     // display the value of x and y  }
}
```

class Derived extends Base

```
  {   private int z;

      Derived ()
       {  super(); // call the super class default constructor.
                   // it must be the 1st statement.
          z = 0;
       }
      Derived (int a, int b, int c)
       {  super (a, b); // call the super class 2-arg constructor
          z = c;
       }
     void show()
       {  super. show (); /call the super class show() method
          System.out.println ("Value of z is:" + z);
       }
```

```
  }
class TestSingle
 {  public static void main (String s [ ] )
      {  Derived d = new Derived ( );
         // invoke the Derived class default constructor
         d.show ();
         d= new Derived (10, 20, 30);
         // invoke the Derived class 3 argument constructor
         d.show ( );
      }
 }
```

Program for Single Inheritance

```
import java.io.*;
class Student
 {  private  int  RNo;
    private String SName;
    DataInputStream Din=new DataInputStream(System.in);
    void read()
      {
          try
```

```
            {    System.out.println ("enter the roll number");
                 RNo = Integer.parseInt (Din.readLine());
                 System.out.println ("enter the name");
                 SName = Din.readLine();
            }
          catch (IOException  e )
           { System.out.println("Error in Input Statement..");  }
      }
    void show ( )
     {    / Display the value of Rno and Name          }
}
class Mark extends Student
{   private  int  Mark1, Mark2, Mark3;
    void read ( )
    {
     super.read ();
      try
       {    System.out.println ("enter the mark1");
            Mark1 = Integer.parseInt (Din.readLine());
            System.out.println ("enter the mark2");
```

```
            Mark2 = Integer.parseInt (Din.readLine());
            System.out.println ("enter the mark3");
            Mark3 = Integer.parseInt (Din.readLine());
        }
       catch(IOException  e )
        { System.out.println("Error in Input Statement..");  }
    }
  void show ( )
   {  super.show();
      // Display the value of M1, M2 and M3
   }
}
class TestSingle
  {  public static void main (String arg[ ])
      {    Mark M = new Mark ( );
           M.read ();
           M.show ();
      }
  }
```

Program for Multilevel Inheritance

```
import java.io.*;
class Student
 {    private  int  RNo;
      private String SName;
      DataInputStream Din=new DataInputStream(System.in);

      void read( )
        {    try
              {
                 / get the input for Rno and Name
              }
             catch (IOException e)
             { System.out.println("Error in Input Statement.."); }
        }
      void show( )
         {   // display the value of Rno and Name       }
 }
```

```
class Mark extends Student
 {
      int  Mark1, Mark2, Mark3;
      void read ( )
       {
            super.read ();
            try
             {
                get the value for M1, M2 and M3
             }
            catch (IOException e)
            { System.out.println("Error in Input Statement..");  }
       }
     void show ( )
      {
        super.show();
         // display the value of M1, M2 and M3
      }
```

```
 }
class Result extends Mark
  {
     int Total;        double Avg;
     void   calculate()
      {
            super.read();
            Total = Mark1 + Mark2 + Mark3;
            Avg = Total/3;
      }
     void show()
     {
       super.show();
       System.out.println ("Total is : " + Total);
       System.out.println ("Average is :"+ Avg);
     }
  }

class TestMultilevel
   {
```

```
    public static void main (String arg[ ])
      {
         Result R = new Result ();
         R.calculate();
         R.show ();
      }
}
```

Program for Hierarchical Inheritance

```
import java.io.*;
class Book
{   int BNo; String BName, Author;
    DataInputStream Din = new DataInputStream(System.in);
    void getBook()
      {
        try
          {
            System.out.println("Enter Book Number");
            BNo = Integer.parseInt(Din.readLine());
```

```
                System.out.println("Enter Book Name ");
                BName= Din.readLine();
                System.out.println ("Enter Author Name ");
                Author = Din.readLine();
              }
            catch (IOException  e )
            {System.out.println("Error in Input Statement..");  }
      }
   void show ( )
    {       System.out.println ("Book Name    : " + BName);
            System.out.println ("Book Number  : " + BNo);
            System.out.println ("Author Name  : " + Author);
    }
}
```

```
class Student extends Book
 {  private  int  RNo;    String Major;
    private String SName;
    void read ( )
     {  super.getBook();
        try
         {       RNo = Integer.parseInt (Din.readLine());
                 SName = Din.readLine();
                 Major = Din.readLine();
         }
       catch (IOException  e )
          {System.out.println("Error in Input Statement..");  }
    }
   void show ( )
    {     super.show();
          / display the value of name, rno and major.
    }
 }
class Staff extends Book
 {  String Name, Dept;
```

```
    void read ( )
      {  super.getBook();
         try
          {  System.out.println("Enter staff Name ");
             Name = Din.readLine();
             System.out.println("Enter the Dept ");
             Dept = Din.readLine();
          }
         catch (IOException  e )
         {  System.out.println("Error in Input Statement..");  }
       }
    void show ( )
      {    super.show();
           / display the value of name and dept
      }
}
class TestHie
 {
   public static void main (String arg[ ])
    {
```

```
            Student S = new Student ();
            System.out.println("Enter student information");
            S.read();
            Staff K = new Staff();
            System.out.println("Enter staff information");
            K.read();
            S.show();
            K.show();
        }
    }
```

Note: Modify the main program-using array of object and getting the choices like staff/student, read/show……

Abstract

1. Abstract Method

```
abstract returnType methodName (Arg List);
```

- ❖ Abstract method does not have definition.
- ❖ We should declare Abstract methods inside Abstract class
- ❖ We must override the Abstract methods in the Sub class. Here abstract method forces inheritance.
- ❖ Abstract method is a key to Polymorphism.
- ❖ In general, Method declaration inside the super class and definition must be in the sub class.

2. Abstract Class

```
abstract class <class Name>
 {
      ……
```

- ❖ We cannot initialize an abstract class object.
- ❖ But we can create object reference for an abstract class.

- **Example:** **Time T; // is possible.**
  Time T = new Time (); // is not possible

- We can initialize an abstract class object by using its sub class.
- **Example:** Shape S; // where Shape is an abstract class
  S = new Circle ( ); // where Circle is a subclass for Shape
- Here abstract class forces inheritance.

Steps to Implement Polymorphism:

- Design a Hierarchical Inheritance.
- Create one or more abstract methods in the super class.
- Override the super class abstract methods in the sub class.
- Create a super class object reference in main function.
- Initialize the super class object by its sub class.
- Invoke the overridden methods.

```
abstract class Shape
 {
   int X;
   DataInputStream Din = new DataInputStream (System. in);
   abstract void read() throws IOException;
   abstract void FindArea();
   abstract void show();
 }
class Circle extends Shape
{   double Area;
   final double Pi = 3.1416;
   void read() throws IOException
     {   System.out.println(" Enter the radios value");
```

```
            X = Integer.parseInt(Din.readLine());
      }
    void FindArea()
    {  Area = Pi * X * X;  }
    void show()
    {  System.out.println("SHAPE IS CIRCLE");
       System.out.println("***************");
       System.out.println("\tRadius :"+X);
       FindArea();
       System.out.println("\tArea of Circle is:"+Area);
   }
}
```

class Square *extends* Shape

```
{
      double Area;
      void read() throws IOException
        {
          System.out.println(" Enter the side value");
          X = Integer.parseInt(Din.readLine());
```

```
        }
    void FindArea()
     {  Area = 4 * X;  }

    void show()
     {     System.out.println("SHAPE IS SQUARE");
           System.out.println("***************");
           System.out.println("\tSide Value is :"+X);
           FindArea();
          System.out.println("\tArea of Square is:"+Area);
     }
}
class TestPoly
  {   public static void main (String arg[ ]) throws IOException
       {     Shape S;
             S = new Circle ();
             S.read();
             S.show();
             S = new Square ();
             S.read();
```

```
            S.show();
        }
    }
```

// Modify the main class definition by getting the choice

Try: Create a base class SHAPE and derive two classes Rectangle and Triangle from SHAPE. Find the area of triangle and rectangle.

Final

1. Final Data

- Final data is same as constant data. We can't change the value of a final data.

```
final scope data_type var_name = value;
```

final int A = 10;

final static double pi = 3.1413;

2. Final Method (Opposite to abstract method)

- We can't override the final method. Here final keyword prevents method overriding.

```
final scope return_type methodName( Arg  List)
  {
     :::::::::::::  // body of the method
```

```
class Base
    {
        final int A=10;
        final void show()
        {   A = 30;
            System.out.println ("Inside the Final Method");
         }
    }
class Derived extends Base
    {
         void show()
           {  System.out.println( "Inside Sub Class "); }
    }
class TestFinalMethod
    {
    public static void main (String arg[ ])
            {
                Derived D = new Derived ();
                D.show();
```

```
        }
    }
```

3. Final Class (Opposite to Abstract Class)

- We can't extend a final class. Here, final key word prevents inheritance.

```
final class  <className>
  {
      ::::::::::::  // body of the class
```

```
final class Base
  {   void show()
        {  System.out.println("Inside  Final Class");  }
  }
class Derived extends Base
  {
     void show()
        {  System.out.println( "Inside Sub Class"); }
```

```
}
```

class TestFinalClass

```
{   public static void main (String arg[ ])
    {       Derived D = new Derived ();
            D.show();
    }
}
```

```
import java.io.*;
class Input
{  int I;   float F;          double D;     String str;    char ch;
   void ReadData()
   {  DataInputStream Din = new DataInputStream(System.in);
      try
{       System.out.println("Enter integer value");
                I = Integer.parseInt(Din.readLine());
                System.out.println("Enter float value");
                F = loat.parseFloat(Din.readLine());
                System.out.println("Enter double value");
                D = Double.parseDouble(Din.readLine());
```

```
                    System.out.println("Enter a string ");

                    str = Din.readLine();

                    System.out.println("Enter a character");

                    ch =(char)Din.read();

            } catch(IOException e) {  System.out.println("Error in try block");   }

        }

    void ShowData()

    {     System.out.println("Integer is  :" + I);

            System.out.println("Float is    :" + F);

            System.out.println("Double is   :" + D);

            System.out.println("String is   :" + str);

            System.out.println("character is:" + ch);

    }

}

class TestInput

 { public static void main(String arg[])

   {      Input m = new Input();

          m.ReadData();                    m.ShowData();

   }
```

}

```
import java.io.*;
class Money
{       int Rs, Ps;
        void ReadData()
        {
                DataInputStream D = new DataInputStream(System.in);
                 try
                {       System.out.println("Enter Rs value");
                        Rs = Integer.parseInt(D.readLine());
                        System.out.println("Enter Ps value");
                        Ps = Integer.parseInt(D.readLine());
                } catch(IOException e) { System.out.println("Error in try block");  }
     }
    void ShowData()
        {
                System.out.println("Rs: "+Rs + "." + Ps);
```

```
            }

    }

    class TestInput

     {

        public static void main(String arg[])

        {       Money m = new Money();

                m.ReadData();

                m.ShowData();

        }

    }
```