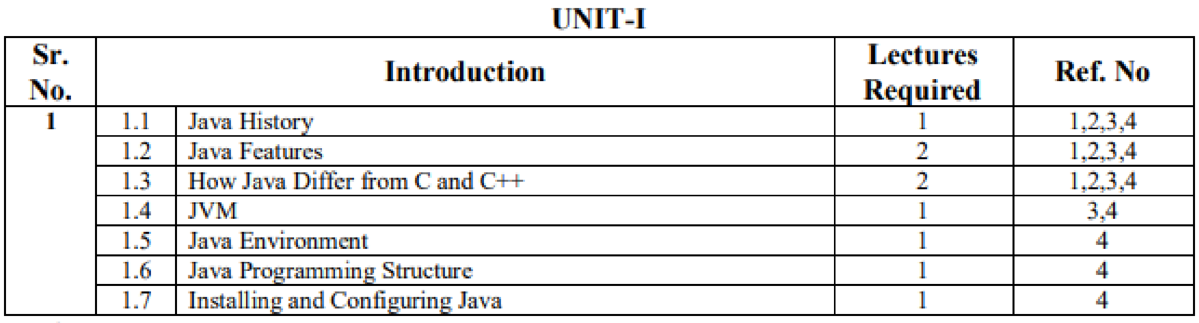


UNIT-I

Sr. No.	Introduction		Lectures Required	Ref. No
1	1.1	Java History	1	1,2,3,4
	1.2	Java Features	2	1,2,3,4
	1.3	How Java Differ from C and C++	2	1,2,3,4
	1.4	JVM	1	3,4
	1.5	Java Environment	1	4
	1.6	Java Programming Structure	1	4
	1.7	Installing and Configuring Java	1	4

Unit -1

- **Java History**
- Java is general purpose, object orented prog language developed by James Gosline and his team working in Sun Micro Sytem of USA in 1991
- Orignially this language was named as OAK and later on renamed to Java.
- Java was designed for the devlopment of software for consumer electrocnic devices like TV,VCR, Microwave, Toasters, etc
- This language was developed to solve the challenges of the languages existing at that time in terms of :
  - Simpliciy,
  - Reliability and
  - Portability
- Java developed from its pre-decessor languages C & C++, by using their many features.
- The historical milestones of java language development are:

1990	Team headed by James Gosling formed to decide software language for consumer electronic device.
1991	Announce of Ook
1992	“Green Project” team demonstrated application using this new language for home appliances, hand-held devides and touch sensitive screen.
1993	For www, “Green Proj Team” suggested the idea of “Web Applet”.
1994	To run java applet, concept of java enabled browser, “Hot java” introduced.
1995	Due to legal problems “Oak” renamed to java.
1996	Java established as Internet programming language and also general purpose OO language.
1997	Sun MS released JDK 1.1
1998	Sun MS released SDK 1.2
1999	Sun MS released J2EE
2000	J2SE with SDK1.3 was released
2002	J2SE with SDK1.4 was released
2004	J2SE with JDK5.0 was released

- By this time, java established in market in following editions:-
  - Standard Edition
  - Enterprise Edition
  - Micro Edition
- **Java Features**
- Modern programming languages (C++, VB, etc) were having many problems.
- To overcome these James Gosling & his team provided solution as Java language.
- While developing this language, C & C++ languages were used as predecessor.
- They added many new features and many useless features were removed.

- Due ot this, java can be used to develop application and also web applets.
- The different features supported are:-

1) Compiled and Interpreter based
2) Platform Independent
3) Object Oriented
4) Robust and secure
5) Distributed
6) Simple, small and familiar
7) Multithreaded & Interactve
8) High performance
9) Dynamic & Extensible

1] Compiled and Interpreter based

- As like any other high-level language, java lang. code also to be compiled.

-First it get converted to intermediate code having Bytecode instruction. It is done by “javac” compiler & saved in file with .class extenion.

- once the bytecode is ready, it can be interpreted using interpreter i.e. java interpreter.

- in interpretation, each bytecode instruction get converted into machine code line by line and executed.

2] Platform – independent and portable

- Platform of any Computer or device represented using Hardware (Micro-processor) & software (Operating System).

- Java code developed on one h/w –s/w platform can be easily transferred and executed on any other platform hence it is called platform independent. E.g. java code developed on Win-7 with P-3 processor can be executed on Linux –Celeron-II platform.

3] Object Orineted

- Java is true /pure object oriented language & hence supports all OO features like class, object, inheritance, polymorphism, encapsulation, binding, etc.

- Every function of java language belongs to class.

4] Robust & Secure –

- Java code is said to be robust (avoid possible runtime error) and also secure due to following reasons:

- strict compile and run-time checking or datatypes

- garbage collection for virtual memory management

- exception handling to handle all possible runtime errors

- no pointers hence no chances of damage.

- virus free & secure web applets.

5] Distributed –

- java helps to devlop network programs (socket programming) , hence java program developed on one system can be easily transferred on other.

6] Simple, small and familiar

- Java doest not uses redundant features of C & C++ (pointer, pre-processor directives, etc) hence it becomes simple to use.

- Due to OO features, reusability is posible hence lengthy code is not required and java code becomes small.

- most the java syntax is as like C & C++ hence, if any famililar with C, C++ can easily understand java code.

7) Multithreaded & Interactve

- By this , single java code can be divided into multiple smaller executable parts to handle multiple tasks simultaneously.

- Also highly interactive , user-friendly graphical application can also be built using java language.

8) High performance

- Due to bytecode creation, it is possbile that, java code executes at high speed and also due to multi-threading there is no waste and wating time.

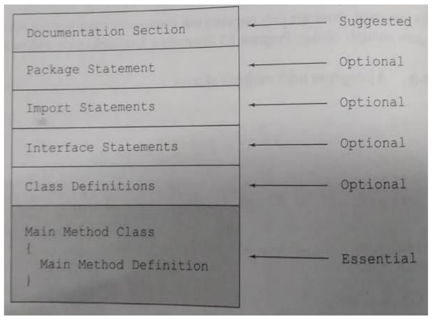
9) Dynamic & Extensible

- java langugage code and be easily linked with new and updated libraries hence it is said to be dynamic language.

- Also java allows to call or access functions of other programming language to be called in java code hence it is extensible.

❖ **Java programming structure**

- as java is programming language, it allows to write java program. The general structure to write a java program can be given as:



❖ From above structure it is clear that, java program uses many sections, some are optional and some are compulsory.

❖ These different sections are :

1. **Documentation section** – it is one sugeested section for use. It is used to give short defination about the purpose of java program

❖ It helps in easy understanding about the program. It can be given as comment line either using:

a) Single comment – using //

b) Multi-line – using /*….*/

2. **Package statement** – It is one optional section in java program.

❖ It is used to inform compiler about the package name to which the current java program belongs.

❖ For this, java language supports "package" keyword, that can be used as:

```
package <PackageName>;

package myproj;
```

3. **Import statement -** It is one optional section in java program.

❖ It is necessary to include the required java lib/package for current program as like header files of C & C++.

❖ For this "import" keyword can be used in following ways:

a) to include single class

```
import java.lang.System;
```

b) to include all classes

```
import java.lang.*;
```

4. **Interface statement –**

- It is one optional section in java program.

- Interface concept is newly added feature in java language to overcome the limitaiton of multiple inheritance.

- Interface is as like class with some similarities and many differences.

- Interface can be created using "interface" keyword as:

```
interface <IntfaceName>
{
  ----
  ----
}
```

5. class Defination – class is one the most fundamental component of OO programming.

- As java is pure OO language, it needs at least one class to execute the program i.e. Main Class.

- A class containing defination of main method is called main class.

- A class in java code can be created as:

```
class <ClassName>
{
  ----
  ----
}
```

6. main method defination – this is one compulsory part of executable java program.

- As like C, C++, a java program execution starts with main method.

- main method must be defined within main class as:

```
class <ClassName>
{
        public static void main(String args[ ])
        {
                ----
               ----
        }
}
```

- ❖ Once the java prog. is coded, it must be saved by “ClassName” with “.java” extension.

Complete .e.g

- ❖ **How java differs from C & C++**

- Java as a language, developed from C & C++ as pre-decessor languages. Hence it is having many common features.

- Also to make joava more powerful many redundant features are removed and new features are added, due to this java shows considerable differences between with C & C++.

Java Vs C

C	**Java**
It belongs to POP category	It belongs to OOP category
C lang supports sizeof, typedef keywords.	These are not supported by java.
C lang supports struct ,union datatypes	These are not supported by java.
C lang supports storage class keywords like auto, extern,register	These are not supported by java.
C supports signed & unsgined types.	These are not supported by java.
C supports pre-processor directives like #include, #define, #ifdef	These are not supported by java.
Functions with default argument can be defined.	These are not supported by java.
C lang allows to create and use pointer.	Pointer not supported by java.
Function without argument in C can use void in parenthesis	Function without argument in java must be with **<u>empty parenthesis</u>**
Not supported by C lang.	Java supports new operators like **instanceof,** >>>

C++ Vs Java

C++	**Java**
It is only extension of OOPS	It is pure OOPS language.
C++ supports pre-processor directives like #include, #define, #ifdef	These are not supported by java.
Function with argument in C++ can use void in parenthesis	Function without argument in java must be with **<u>empty parenthesis.</u>**
Functions with default argument can be defined.	These are not supported by java.
C++ lang allows to create and use pointer.	Pointer not supported by java.
C++ supports Operator Overloading.	Operator Overloading not supported by java.
C++ supports multiple inheritance	Java does not support multiple inheritance.
C++ supports concept of global variables	Java does not supports globar variable, instead it uses class variable
Destructor function is supported	Destructor function is replaced by finalize( ) function.
Main method in C++ class program, must be outside the class	Main method must be within class
Class defination must ends with ; (semicolon)	Class defination not ended by ; (semicolon)

- ❖ **JVM**

- As java is high level language, hence its code must be translated before execution.

- To implment java language, it requres its own runtime environment i.e called JRE.

- JRE creates a virtual machine with-in real machine hence it is called JVM i.e. machine within machine.

- When java code is compiled by compiler, first it gets converted into bytecode / intermediate code which is **not machine specific (dependent)**.

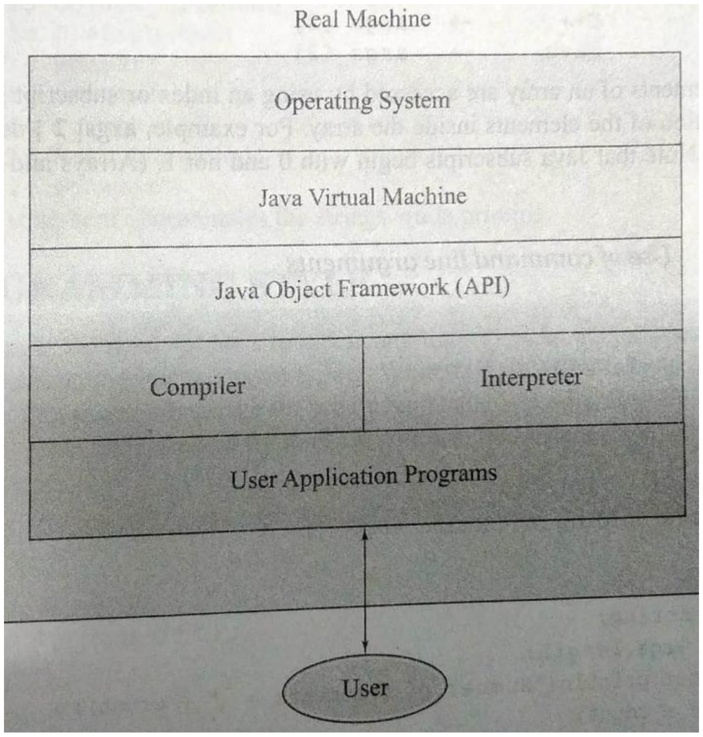
$$\begin{tikzcd}\boxed{\text{Java Program (Source Code)}} \arrow[r] & \boxed{\text{Java Compiler}} \arrow[r] & \boxed{\text{Virtual Machine (Bvte Code)}}\end{tikzcd}$$

- And this bytecode is used by virtual machine for further processing.

- When java is interpreted by java interpreter, each line of byte code is conveted into machine code and then executed as:

$$\begin{tikzcd}\boxed{\text{Bytecode}} \arrow[r] & \boxed{\text{Java Interpreter}} \arrow[r] & \boxed{\text{Machine Code}}\end{tikzcd}$$

- ❖ The internal structure of JVM can be represented as:



- From above it is clear that, OS is working between Real machine and JVM.

- User creates and submit user program to compiler and interpreter uses API (Application Programming Interface) which is collection of java library used to compile any program.

❖ Java Runtime Env

- To implment java programming on any system & also to execute java program on any system, a special environment is required i.e. Java Runtime Environment.

- JRE no only helps to build java program but also to compile and execute it.

- JRE consist of:

i) JVM – It is virtual machine within real machine. In this, first bytecode is created by compiler.

- This bytecode is further converted into machine code for execution.

- due to this, java lang becomes machine independent language.

ii) Runtime Class libraries – These are most basic collection of built-in classes.

- These classes are useful to execute java program & all java library classes are available in built-in packages.

e.g. String class in java.lang package.

iii) User Interface Toolkits – Toolkit is collection of built-in libraries using which java based GUI (Graphical User Interfaces) can be developed.

- Very commonly used java toolkits for building GUI's are AWT & Swing.

iv) Deployment Technologies – once a java based software or application is developed it can be installed / deployed on any machine using:

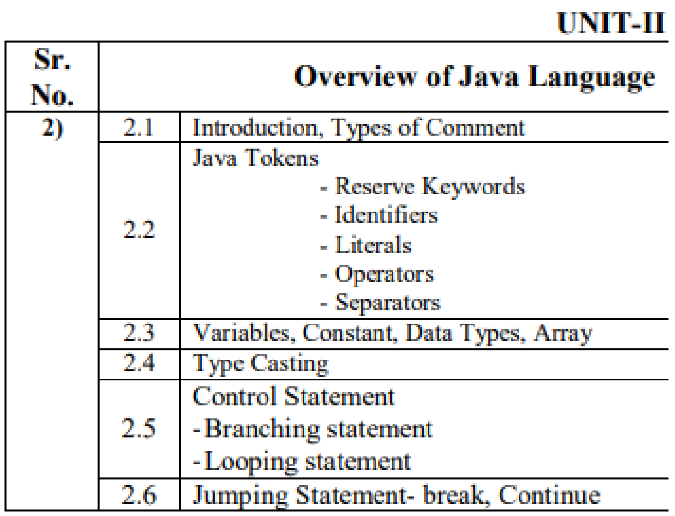
a) Java plug-in – it helps java applet to be exeucted in browser.

b) Java web start – it helps java program executed remotely.

❖ Installing & configuring java

- Installation steps to be noted in practical while performing java installation.

Unit – 2 Overview of Java Language



UNIT-II

Sr. No.		Overview of Java Language
2)	2.1	Introduction, Types of Comment
	2.2	Java Tokens - Reserve Keywords - Identifiers - Literals - Operators - Separators
	2.3	Variables, Constant, Data Types, Array
	2.4	Type Casting
	2.5	Control Statement -Branching statement -Looping statement
	2.6	Jumping Statement- break, Continue

❖ Type of comment

- It is necessary to make the program readable and easy to understand.

- for this, programming language allows to use comment.

- Comment is read-only statement in program code which neither compiled or executed.

- Compiler will directly ignore such comment line even if it is containg error.

- comment in prog code is used to give short defination about program code or purpose of program for easy understanding.

- comment in java program can be given in one of the following way:

1. Single line comment – any line in program code starting with // can be considered as single line comment.

2. Multi-line comment – in case if set of lines to be declared as comment, it is to be enclosed in /*….*/.

e.g.

❖ Java Tokens

- Every high-level language code first must be compiled before execution.

- For this compiler is used which compiles the high-level language code.

- A java prog code also gets compiled into intermediate code or byte code stored in .class file.

- During such compilation, each high-level language instruction get converted into smaller pieces.

- Each smallest unit or piece of instruction is called token.

- in java prog code following 5 types of tokens can be considered:

1. Keyword	2. Identifier
3. Literal	4. Operator
5. Separator	

1. Reserved Keywords – Keyword is one of the most fundamental entity of programming language.

- It is predefined word with predefined meaning.

- it is always used in small case and it can not be used as an identifier.

- java supports almost all the keywords of C, C++ and some new keywords, thus java having 50 keywords like int, float, switch, super, throw, this, etc.

2. Identifiers – It is one token created by programmer in prog code.

- it is a valid name given to class, variable, function, interface, object, package, etc.

- while creating an identifier , some rules must be followed:

i. It must start either with alphabet or underscore as:

int RollNo, int _rollno;

ii. It must not contain blank space, reserve character like -, period (.)

iii. It must not be a reserve keyword

iv. It may contain digits, Underscore as

int Num1, int roll_no.

- Variable name can be of any length but it is recommended it should be meaningful.

3. Literals – This is one type of token created by java compiler from java prog code.

- Literal is sequence of characters (digits, letters, or other character) that represent constant values.

- Such constant value can be stored in java program variable.

- Literal not only represents value but also represents:

- **How any such value is stored?**
- **How such value to be processed?**

- The different types of literals supported are:

i. Integer Literal – it is numeric value also complete value without decimal point such as:

rollno=21, Marks=436, etc

ii. Floating point Literals - it is numeric value with decimal point and also called as **<u>fractional or real value</u>** such as:

temp=32.4, tax=3.5, height=5.4, perMarks=67.21.

iii. Character literals – it is value made of alphabets or other characters with single char .e.g

grade='A', etc

iv. String literals – It is value made of group of characters e.g.

StudentName="John";

v. Boolean literals – this type of literal represents only "true" or "false" value.

4. Operators – This is one type of token created by java compiler from java prog code.

- Operator is one special symbol that performs a predefined operation.

- Operator uses value or variable to complete its operation which is called Operand.

- Operators basically classified as:

A) Unary Operator – Operator using single operand such as i++, ++i, etc

B) Binary Operator - Operator using two operands such as a+b, a*b; etc

C) Ternary Operator - Operator using three operands such as ?:

- As per the operations that an operator performs, they are further classified as:

I. Arithmetic operator
II. Relational Operator
III. Logical Operator
IV. Assignment operator
V. Increment & Decrement operator
VI. Special operator

5. Separators – It is also one type of token created by java compiler.

- This token is used to divide the code and group it properly.

- following are the different java separators:

Name	Description
Parentheses ( )	It is used while defining the function and calling to specify parameters. e.g. f1([parm])

Braces { }	It is used to define code block and to give set of array values.
	e.g.
	void f1() {…}
	n[3]={2.4.5}

Brackets [ ]	Used to specify array size int n[3]
Semicolon **;**	Used as sentence terminator i.e. generally at the end of each line.
Comma **,**	To separate values , variable declarations int a, b, c; int n[3]={10 , 30 , 40};
Period ●	Used to show package and sub package name, object & member java.awt.event ob.display()

❖ **Variable**

- A java program may uses different programming elements & can perform different operations.

- Collecting and storing the required data element to complete the execution is also one of the imp task of java program.

- To collect and store the required data element java program uses Variable.

- Variable is labeled temp memory location to store required data element.

- A programming entity whose value can be changed is called Variable.

- A variable must be declared before its first use anywhere in java program as:

<DataType> <VarName>;

e.g. int n;

- at a time multiple variable of same data type can also be declared as:

int a, b, c;

- Variable Name is a valid name or "Identifier" as per the following rules:

- It must starts with alphabet (small / capital) or underscore (_) e..g int Rno; char _sName;
- It must not contain blank space or any reserve character like (int roll-no, rollno. [invalid])
- It may contain underscore or digit like Num1, roll_no;

- Variable declaration is compulsory as it informs about following to compiler:

  - Name and datatype of variable.
  - Memory size of variable
  - Scope of variable

- Once a variable is declared it can be initialized in of the following ways:
  1. **Static Initialization** – if the value to variable is assigned such that it is fixed and does not get change is called static initialization as:

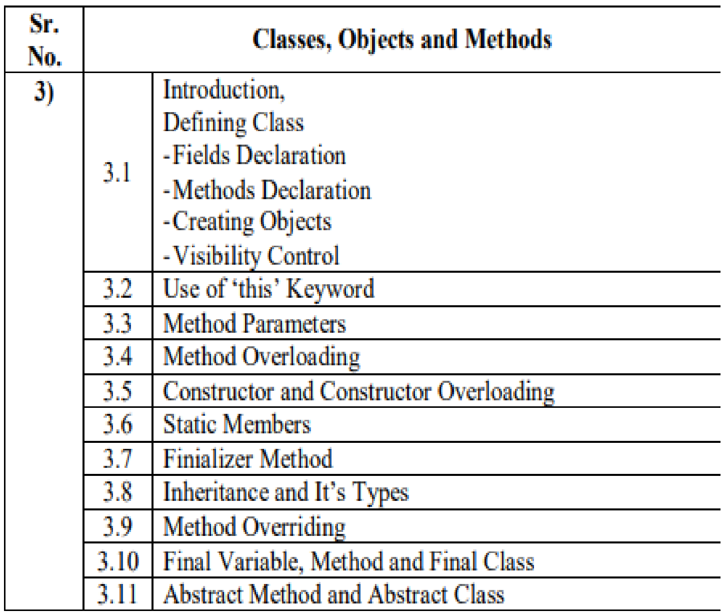
<DataType> <VarName>= Value;
e.g. int n=100;
or
int n;
n=100;

  2. **Dynamic Initialization** – if the value of variable assigned at runtime it is called dynamic initialization.

- For this, java language supports use of input statements.
- In java language "Scanner" lib class can be used to get different inputs like nextInt( ), nextFloat( ), nextDouble( ), nextChar( ), etc

e.g.
```
class VarDemo
{
public static void main( )
  {
     int  n=100; // static initialization.
     int m;
      Scanner sc=new Scanner(System.in);
    System.out.println("Enter a int value:");
    m=sc.nextInt( ) ; //Dy initialization
   System.out.println( " static value n="+n);
 System.out.println( " Dynamic value n="+m);
   }
}
```

- **Constant**
- **Data types**
- **Type Casting**
- **Control statement**
- **Branching**
- **Looping**
- **Jumping**
- **Break**
- **Continue**



Sr. No.	Classes, Objects and Methods	
3)	3.1	Introduction, Defining Class -Fields Declaration -Methods Declaration -Creating Objects -Visibility Control
	3.2	Use of 'this' Keyword
	3.3	Method Parameters
	3.4	Method Overloading
	3.5	Constructor and Constructor Overloading
	3.6	Static Members
	3.7	Finializer Method
	3.8	Inheritance and It's Types
	3.9	Method Overriding
	3.10	Final Variable, Method and Final Class
	3.11	Abstract Method and Abstract Class

- Introduction – OO programming uses many fundamental entities like class, methods, data members, objects, etc.

❖ For this, all these members to be properly created before use.

- **Defining Class** – Class is one of the most fundamental entity of OO programming.

❖ Class is one User Defined Data type which is created for by programmer for a specific user application.

❖ Class as user defined data type can enclose two types members:

  - Data members
  - Function member

❖ A java program can be designed with single or multiple classes.

❖ A class in java program can be created using "class" keyword as:

```
class <ClassName> [extends BaseClassName]

{

        [data members];

        [ function members];

}
```

❖ As oppose to , class defined in C++ program, a class defination of java program never ends with simicolon (;).

e.g.

```
class A
{
    int x; // Data member
void f1() // function member
{}
}
```

❖ In single class, single or multiple data members or same or different types can be enclosed.

❖ In single class multiple data members can also be enclosed.

❖ For execution of java program along with other classes, there must be at least one main class.

❖ The class in which main method is defined is called Main Class.

e.g.

```
class A
{
   int x, y;
          void display( )
          {
                    x=10;  y=20;
          System.out.println("Addition="+(x+y)
);
          }
}
class MainCls
{
public static void main(String ar[])
 {
   A ob=new A();
          ob.display();
 }
}
```

- **Fields Declaration** – As class is User Defined Data type, it can enclose data as well as function member.

❖ Data membert in class represents, data fields which are also called as variable.

❖ Data field in class used to store required data elements.

❖ The variables enclosed in class but outside of any method is called "Instance Variable".

❖ Variable or data field in class can be added as:

```
          <dataType> VarName;
```

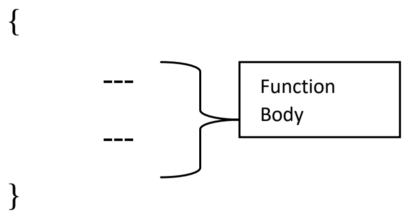
- Var Name must be an identifier i.e. a name given with all rules like:
    - It must starts with alphabet or Undersocre (roll_num).
    - It must not contain blank space, reserve character.
    - It may enclose digit (num1)

e.g.

- **Method Declaration-** As class is User Defined Data type, it can enclose data as well as function member.

- As data field represents data elements required for class, simillarly it also needs function member.
- Function member represents the operation code used by class to operate & manipulate data member.
- In sinle class either single or multiple methods can be defined.
- Melthod in class can be added as:

  <ReturnType> FuncName([parm])



- **Return type** – it shows whether function returns value or not. If function returning value which type of value it is returning.
- **Function Name** – it is valid name or identifier given as per the rules.
- **Function Body** – it contains actual operational code or set of instruction that function performs.
- A function can be created with or with-out parameter.
- If function is created with parameter, it may with sinlge or multiple parameters of same or different types:

  void f1( ) {} // function without param

  void f2(int n) {} // function with single parm

  void f3(int n, int m) {} // function with multiple parameters of same type

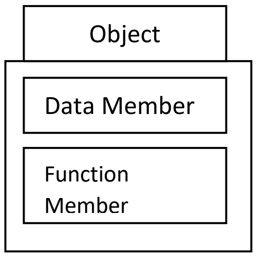
  void f4(int n, double d) {} // function with multiple parameters of different types.

  e.g.

```
class A
{
   int x, y;
          void display( )
          {
                    x=10;  y=20;
System.out.println("Addition="+(x+y));
          }
  void show()
  {
                    x=10;  y=20;
    System.out.println("Subt="+(x-y));
  }
}
class MainCls
{
public static void main(String ar[])
 {
   A ob=new A();
          ob.display();
          ob.show();
 }
}
```

- **Creating Object –** As like class, Object is also one of the most fundamental entity of Object Oriented Programming.

- Object is said to be one **Runtime Entity** of OOP or **instance** of class.
- Object is one allocated memory block which reserves space for data mamber and function member.



- Obect on any class in java program can be created using **Declaration and Initialization** as:

  ClassName ObjName=new ClassName( ) ;

  MathCls obj=new MathCls( );

- ❖ Alternatively it is also possible that, declaration and initialization can be separated as:
- ❖ ClassName ObjName; // Only declaration

ObjName=new ClassName( ) ; // initialization
e.g.

```
MathCls obj
obj=new MathCls( );
```

- ❖ Object is necessary to call / access the members of class with help of Member

  Accessing Operator (**.**) as:
  ObjName.MemberName;

e.g.
Obj.display( );

- ❖ On single class single or multiple objects can be created as:

```
MathCls obj1=new MathCls( );

MathCls obj2=new MathCls( );

MathCls obj3=new MathCls( );
```

- ❖ When multiple objects created, each object gets separate / individual copy of class members.

Complete E.g.

```
class A
{
   int x, y;
         void display( )
         {
                  x=10;  y=20;
System.out.println("Addition="+(x+y));
         }

}
class MainCls
{
public static void main(String ar[])
 {
   A ob=new A();
         ob.display();
 }
}
```

- **Method Overloading**

- ❖ A class may contain single method or multiple methods.
- ❖ To define a method, proper name and signature is specified.
- ❖ This method name and it's signature is necessary to call the function.
- ❖ If a software contains too many functions with different names and different signature it may causes following problems:
  - o Remembering name of each method and its signature at the time of calling.
  - o If calling statement mismatched, will cause error.
- ❖ To solve these problems, OOP supports the concept of Method Overloading.
- ❖ Method Overloading is a OOPs technique by which mutliple methods are defined by same name but different signature.
- ❖ The difference in signature may be:
  - o Parameter
  - o Number of parameters
  - o Types of parameters
  - o Sequence of parameters
- ❖ When a method is redifned by different signature is called Method Overloading.
- ❖ Such redefined method is called "Overloaded Method".
- ❖ For successful method overloading following rules are important:
  - o Multiple functions in same class must be defined by same name.
  - o With different signature
  - o Return data type of function may be same or different, is not important.
- ❖ Method overloading can be represented as:

```
void f1 (  ) {} // Defination 1

void f1( int ) {} // Def. 2

void f1( double ) { } // Def. 3

void f1 (int , double ) {} Def 4

void f1 ( double, int ) {} Def 5
```

- ❖ Thus above five defination of f1 ( ) method shows difference in signature in terms of

Parameter, type of parameter, number of parameter and sequence of parameter.

e.g.

```
class A
{
   int x, y;
  void display( )
  {
System.out.println(" Overloaded Function");
  }
  void display( int x)
  {
System.out.println("Square="+(x*x));
  }
 void display( int x, int y)
  {
System.out.println("Addition="+(x+x));
  }
 void display( String s)
  {
System.out.println("Length="+s.length( ));
  }
}
class MainCls
{
public static void main(String ar[])
  {
    A ob=new A();
        ob.display( );
        ob.display(10);
        ob.display(5,10);
        ob.display("ABC");
 }
}
```

❖ In Method Overloading, the calling staement, and the function defination is linked at run time hence it is called **Late Bining or Dynamic Binding**.

❖ **Constructor**

A class in java program can enclose data as well as function members.

On class when object is created, some memory to be allocated for its data and fucntion members.

Every class used to have a special member function called as construtor.

Constructor is special member function of class which completes the job of **Allocation** & **Initialization**.

Constructor is implicity created and called, in case if constructor to be defined, following rules are important:

i. No return type to be given as constructor never returns value.

ii. Function name and class name must be same

iii. Construtor can be with or without parameter.

iv. Constructor get called implicitly / automatically when the object on class is created.

Construcor can be created as:

```
class ClassName
{
  ClassName( ) // Construtor Defination
  {
  }
}
```

❖ Constructor can be defined with or without parameter.
❖ Constructor without parameter is called "Default Constructor".

e.g.

```
class A

{

 int n;

  A ( )

     {

       n=100;

      System.out.println("Constructor Initialized");

     }

  void display( )

     {

       S.o.p("Square="+( n*n ));

     }

 }

class MainCls
{
```

```
public static void main(String ar[ ])
 {
   A ob=new A( ); // call the constructor
  ob.display( );
}
}
```

- **Constructor Overloading**
- OOPS supports many features and one such important feature is Method Overloading.
- It means, multiple methods defined by same name but differfent signature.
- As constructor is also one function / method of class, hence it can be also overloaded.
- Construtor Overloading which means Constructor redefined in class with different signature.
- Difference of signature involves:
  - Parameter
  - Number of parameter
  - Type of parameters
  - Sequence of parameters
- When in same class mutliple constructors are defined by same name with different signature is called Construtor Overloading as:

```
class ClassName
{
   ContstName (  ) {}
   ConstName ( int ) {}
 ConstName(int , int ) {}
 ConstName(double, String) {}
ConstName(String , double ) {}
 }
```

- In all above definations, ConstNames are same only difference of signature is maintained.
- Such constructor defined with different signature is called "**<u>Overloaded Constructor</u>**".
- A constructor defined without parameter is called Default Constructor.
- Any constructor get called implcitly when object on that is created.
- If constructor is defined with parameter, then while creating object or calling constructor parameter must be passed.

e.g.

```
class A
{
      A ( )
        {
          System.out.println("It is default Constructor");
        }
A(int n)
{
S.o.p("Square="+(n*n));
}
A(int a, int b)
{
S.o.p("Addition="+(a+b));
}

 }
class MainCls
{
  public static void main(String ar[ ])
   {
       A ob1=new A( ); // call the default constructor
A ob2=new A(20 ); // call the constructor with one int parameter

A ob3=new A(20,100 ); // call the constructor with two int parameters.

      }
}
```

- Inheritance & Types – OOP offers many features and many benefits, such as encapsulation, data hiding, method overloading, overrridng, etc.

- One such important feature is Inheritance.

- In real-time software development, one common factor is , if a software to be created it undergoes in many stages.

- Like Requirment analysis, designing, coding, testing and then deployment.

- all these stages takes too much time. In case, if already a software is devloped with some same features, generally the process is repeated which again takes too much time.

- to overcome these problems, Inheritance technique of OOPs can be used.

- Inhertiance is a process by which from existing class new class can be created.

- The existing class is called "Base Class or Super Class or Parent Class".

- The newly created class is called "Derived Class or Sub class or Child Class".

- due to inheritance, following effects can be observed:

  - copy of each data meber is given from base class to derived class.

  - Derived class gets Permission to call base class function without object of Class Name.

- Using inheritance, when class or code is created, it gives advantage of Reusability.

- that means if the code is required again, no need to recode and test it, the existing class can be inherited to create new class.

- to implement inheritance in java language, a special keyword is used i.e. "extends" as

```
class ClassName extends BaseClassName

{

}
```

- in java language, following inheritnace types can be implmented.

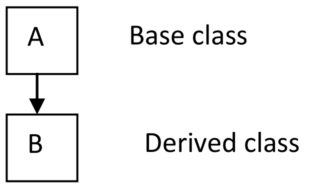
1. Single Inheritance   2. Multi-level Inheritance

3. Hierarchial   4. Hybrid Inheritance

1. Single Inheritance – In this type of Inheritance, only one base class and only derived class is involved.

- When in inheritance process, if only one base class and one derived class is used, it is called Single Inheritance.

- Single Inheritance can be represented as:



- single inheritance can be applied as:

```
Class ClassName extends BaseClasssName

{

}
```

e.g.

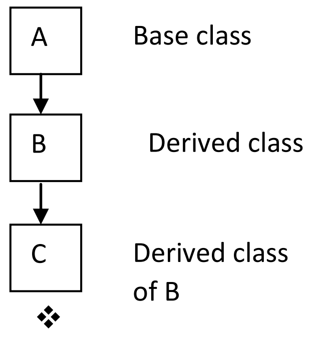
```
class A // Base class
{
   int n;
  void getInput( )
   {
    Scanner sc=new Scanner(System.in);
   S.o.p("Enter a num");
          n=sc.nextInt();
   }
   }
class B extends A // Creating derived class
{
   void display( )
{
    getInput( );
S.o.p("Square Root="+Math.sqrt(n));
}
}
class MainCls
{
  public static void main(String ar[] )
   {
      B obj=new B( );
      obj.display( );
}
}
```

2. **Multi-Level Inheritance** – In this type of inheritance, chain of base and derived classes gets created.

- When in Inheritance process, a derived class is further extended to create its child/dervied class it is called Multi-level inheritance.

- thus, in this type, a derived also becomes base class for its derived class.

- It can be shown as:



- In this, "C" becomes grand child and "A" becomes Grand parent class.

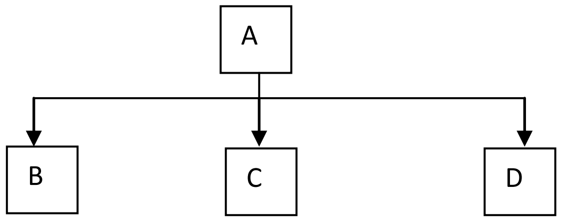
- C as grand child class inherists properties directly from its immediate parent class "B" and indirectly from "A".

e.g.

```
class A // Base class
{
   int n;
  void getInput( )
   {
    Scanner sc=new Scanner(System.in);
   S.o.p("Enter a num");
          n=sc.nextInt();
   }
   }
class B extends A // Creating derived class
{
   void display( )
   {
     getInput( );
    S.o.p("Square Root="+Math.sqrt(n));
    }
  }
class C extends B // Creating derived class
{
   void show( )
     {
       getInput( );
      S.o.p("Cube="+(x*x*x));
     }
}

class MainCls
{
  public static void main(String ar[] )
   {
     B obj1=new B( );
     obj1.display( );
    C obj2=new C( );
     obj2. show( );
      }
}
```

3. Hierarchical Inheritance

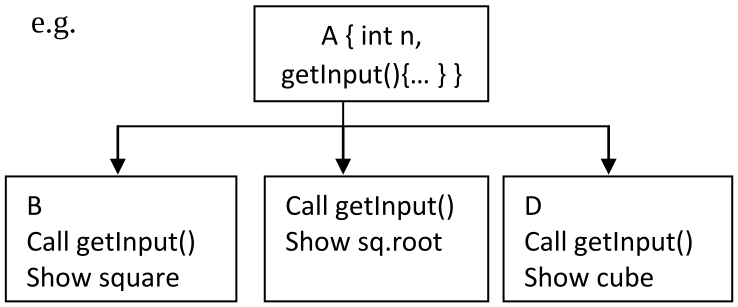
- In this type of inheritance, multiple bases classes are extended / derived from single parent class.
- Due to this, each derived class will get separate copy of base class members.
-



- Hierarchial Inheritance is mainly used to provide **<u>common reusable features</u>**.

- Such feature is created once in base class, and all the derived classes can use it.

e.g.


A { int n, getInput(){… } }
B Call getInput() Show square
Call getInput() Show sq.root
D Call getInput() Show cube

- ❖ Thus in above e.g., declaring one variable and getting input in it, becomes common feature.
- ❖ It is created once in base class "A" and getting reused in all derived classes.

e.g.

```
class A // Base class
{
   int n;
  void getInput( )
   {
    Scanner sc=new Scanner(System.in);
   S.o.p("Enter a num");
          n=sc.nextInt();
   }
   }
class B extends A // Creating derived class
{
   void showSquare( )
   {
     getInput( );
    S.o.p("Square ="+ (n*n));
    }
  }
```

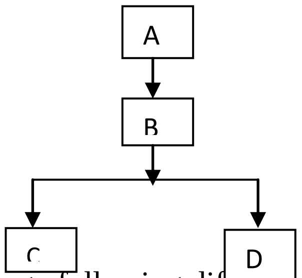
```
class C extends A // Creating derived class
{
   void showRoot( )
     {
        getInput( );
       S.o.p("Sq Root="+Math.sqrt(x));
     }
}
class D extends A // Creating derived class
{
   void showCube( )
     {
        getInput( );
       S.o.p("Cube="+(x*x*x));
     }
}
class MainCls
{
  public static void main(String ar[] )
    {
      B obj1=new B( );
      obj1. showSquare ( );
     C obj2=new C( );
      obj2. showRoot ( );
     D obj3=new D( );
      obj2. showCube ( );
     }
}
```

4. Hybrid Inheritance

- When the other inheritance types are combinedly used in single java program, it is said to be using Hybrid Inheritance.

- Hybrid Inheritance is applied by combining other inheritance types such as Single, Multi level and Hierarchical.

- Due to this, in single java program advantages of other inheritance types can be utilized.



- In above e.g., following different inheritances are combined like

Single Inheritance :   A→B

Multilevel Inheritance : A→ B → C B→C , B→D
Hierarchical Inheritance : hierarchial inheritance

❖ Thus it is clear that, above e.g. using Hybrid inheritance.

e.g

```
class A
{
  int n;
}
class B extends A
{
   void getInput( )
   {
    Scanner sc=new Scanner(System.in);
   S.o.p("Enter a num");
        n=sc.nextInt();
   }
}

class C extends B
{
  void showOddEven( )
   {
     getInput( );
if( n % 2==0)
        System.out.print("Even Number");
else
        System.out.print("Even Number");
    }
 }
class D extends B
{
  void showPosNeg( )
   {
     getInput( );
if( n >0)
        System.out.print("Positive Number");
else
        System.out.print("Negative Number");
    }
  }
class MainCls
{
  public static void main(String ar[])
  {
     C ob1=new C( );
        ob1.showOddEven( );
        ob2.showPosNeg( );
   }
}
```

- **Method Overriding**

- OOPs provides many advantages and features such as encapsulation, overloading (method and constructor), data hiding, inheritance, etc

- One such imporant feature of OOP with inheritance is method overriding.

- When, Inheritance is applied, it is possible that, from existing class (Base class) , new class (Derived class) can be created.

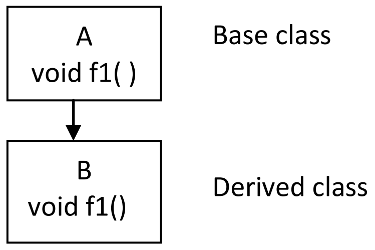
- Due to this, the derived class inherits properties of base class.

- Along with inheritance, it is also possible to apply method overriding.

- **<u>Method Overriding is OOPs technique by which, bases class method can be redefined in derived class</u>**.

- For Method overriding, following is must:

  - ➢ There must be atleast one base and one derived class.
  - ➢ Base class method must be redefined in derived class by same name.



- ❖ The base class method which is redefined in derived class is called Overridden Method.
- ❖ But when such base class overridden method, if called from dervied class method, there may chances of recursion.
- ❖ As the base class and derived class method name is same, the calling statement may causes recursion
- ❖ The derived class method may calls itself infintely.

e.g.

```
class A // Base Class
{
void display ( ) // Overridden method
   {
     S.o.p("I am Base Class");
   }
}
class B entends A // Derived Class
{
void display ( )
   {
          display ( ); // chances of recursion.
     S.o.p("I am Derived Class");
   }
}
```

- Thus when display ( ) method in above e.g. called from drived class, it will not call base class method, but recursively calls itself.
To solve this problem, java language supports "super" keyword.

- When a method, called with "super" keyword, then it is compulsory for JVM to call base class method as:

super.MethodName( );

super.dispaly( );

e.g

```
class A // Base Class
{
void display ( ) // Overridden method
   {
     S.o.p("I am Base Class");
   }
}
class B entends A // Derived Class
{
void display ( )
   {
          super.display ( ); // calls base class method.
     S.o.p("I am Derived Class");
   }
}
class MainCls
{
  public static void main( String args[ ])
  {
    B ob=new B( );
          ob.display( );
}
}
```

Output –

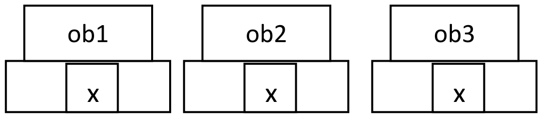
I am Base Class.
I am Derived Class

❖ **Static Members**

- As java is OOP language, it uses class as its fundamental entity.
- For every executable java program, there must be at least one class defined with main ( ).
- Class as a user defined data type can enclose data & function members.
- By default the members of class, data or function declared without static keyword and they are non-static.
- Static members in java class can be of following types:
- 1. Static Data Member 2. Static Function Member.
- **1. Static Data Member –** the variables enclosed with-in class are called data members which represents data elements or data field of class.
- As per general OO programming rule, on a class either single or multiple objects can be created.
- For every created object, JVM gives separate copy of data member.
- In such case, there may be chances of memory wastage if particular object is not created like:

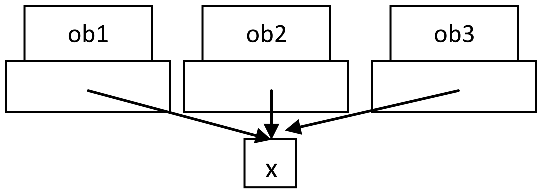
```
class A
{
    int x;
    ---
    ---
    ---
}
```

- On above class "A" if following objects are created:
- A ob1=new A( );
- A ob2=new A( );
- A ob3=new A( );
- Thus each object gets separate copy of "x" data member as



- Thus in above case, once the execution of ob1 is over, its memory is purely get wasted.
- To avoid this, java langauge supports concept of static members.

- When a variable or data member is declared with "static" keyword, it becomes static data member as:
- static <DataType> VariableName;
- static int x;
- it is also possible that, while declaring static member, value can also be assigned as:
- static <DataType> VariableName=value;
- static int x=100;
- when a data member becomes static, it shows following important factors:
- only one copy of data member is created for all objects.
- Same copy of data member is reused by all members as:
-



- **Advantages :-**
  i. Resuse of memory of data member.
  ii. No wastage of data member copy.
- **DisAdvantages –**
  i. Value created by one object will be replaced by other object.
  ii.Value is not safe.

Imp Rule – A static data member can only be accessed by static function member.

2. Static Data Member

- A java class, as like data member can also enclose function member.
- generally function in class is added without "static" keyword hence it is non-static functions.
- if a function is non-static, then such function can be called outside using its object.

```
class A
{
    void f1()
    {--- ---- --- }
}
```

- thus, to call f1( ) method of class "A", object of class to be created & then it can be accessed as:

```
A ob=new A( ); // declaring object
ob.f1( );// calling function
```

- but in many cases, it is not possible to create object on class, but still its function member to be called.

- in such case, such function member to be declared as static.
- A static function member is function in class defined using "static" keyword as:

```
static <ReturnType> FuncName( [ arg ] )
{
---
---
}
static void display( )
{
---
---
}
```

- if the function member is static, it does not require to object to call, but can be called directly with the help of ClassName as:

```
ClassName.FuncName([arg]);
A.display( );
```

- ***Imp Rule – A static function member can only access only staticdata member.***

e.g.

```
class A
{
   static int x;
   static void display( )
     {
x=10; // accessing static data member
       S.o.p("Cube="+(x*x*x));
     }
}
class MainCls
{
  public static void main(String ar[] )
   {
     A.display(); // calling static function member.
     }
}
```

- There are many built-in classes having static functions which can be called directly by class name such as:

Math.sqrt(int);   Integer.parseInt( )

Double.parseDouble( )

❖ Final Class, Variable, Method

- As java is programming language, it uses all basic programming elements like variable, methods, etc.
- Also as java is pure OO programming language, uses "class" uses its one of the basic entity to execute program.
- generally these elements, variable, class & method are non-final but in some cases, these programming entities to be declared final.

❖Final Variable❖

- Variable is one programming entity, which is used to store data elements.
- But one variable at a time can store only one data value, if some other value is assigned, its previuos value get replaced.
- In may cases it is necessary to maintain the value of variable.
- In many cases, change in value of variable should not be allowed.
- For this reason, such variable should be declared as final variable.
- When a variable in java prorgram is declared with "final" keyword it becomes final variable.
- Such variable shows following important characteristcs:
    - o Value of such variable can not get changed.
    - o Value of such variable can be used and reused many times.
    - o Such variable becomes "**<u>Symbolic Constant</u>**" in java program.
  - In real life, there are many such constants which are used such as PI=3.14.
  - **Following are the important rules** to create and use final variable / symbolic constant :
  1. At the time of declaration itself, value must be assigned, as later on its value can not be changed.
  2. It must be declared with class but outside method i.e. class variable.
  - It is recommended that, variable name to be in capital letters for easy understanding.

  - A final variable in java program can be created as:
    final <DataType> VarName=Value;
    final double PI=3.14;
    final int MaxMarks=100;
  - If in program, we try to change the value of final variable, it will show compiler error such as **" can not assign value to final variable"**.
    e.g.
    class A

```
{
  final double PI=3.14;
 void display( int r)
 {
   System.out.println("Area of
Circle="+(PI*r*r));
  }
}
class MainCls
{
 public static void main(String ar[])
  {
    A obj=new A( );
    obj.display(5);
  }
}
```

❖Final class❖

- class is one the most basic entity of OOP and for every executable java program there must be atleast one class.
- class as user defined datatype combines data and function members.
- OOP supports many features and one such is Inheritance by which a class can be sub classed or extended to create sub class.
- in many cases, it is necessary to restrict a class to be inherited , for this it is created as final class.
- when a class in java program is declared with "final" keyword it becomes final class as:

```
final class <ClassName>
{
}
```

- when a class is declared as "final" , it can be used in inheritance process or it's derived/child class can not be created.
- if a java program attempts to inherit such final class, java compiler will show **compiler error** that "**can not inherit from final class**".

```
final class A
{
  void display()
  {
    System.out.println("I am final class");
  }
}
class MainCls
{
 public static void main(String ar[])
  {
    A obj=new A( );
    obj.display();
  }
}
```

❖Final method❖

- class is one the most basic entity of OOP and for every executable java program there must be atleast one class.
- class as user defined datatype combines data and function members.
- OOP supports many features and one such is Inheritance by which a class can be sub classed or extended to create sub class.
- When inheritance is applied, there can be another realted feature i.e. Method Overriding.
- It means, a base class method, is redefined in derived class and such method becomed overridden method.
- but in some cases, this method overriding should not be allowed and for this , it is created as final method.
- A final method is method in class which is creted with "final" keyword as:

```
final <RetType> FuncName( [arg])
{
---
---
}
```

- When such method is attempted to be redefined / override in derived class, compiler shows error "Can Not Override".

```
class A
{
  final void display()
  {
    System.out.println("I am final class");
  }
}
class MainCls
{
 public static void main(String ar[])
  {
    A obj=new A( );
    obj.display();
  }
}
```

❖ **Finalizer method**

- As Java is OOP language, uses class, object , constructor , etc.

- the constuctor is used for initialization simillarly, java also uses the concept of **finalization**.
- java as oppose to C++, does not uses the destructor but uses following :

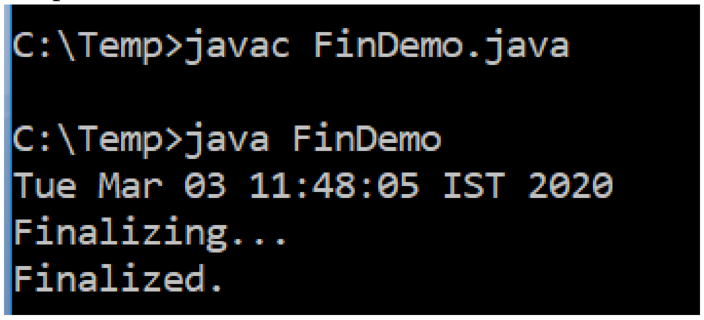
    Garbage Collection
    finalize( )

- Garbage collection – it is one automated process of JVM which is used to find the allocated but unused memory and make it free.
- Garbage collection is automatically called in following two cases:
    - When the system is in idle conditon.
    - When a program is facing shortage of memory problem.
- but in some cases, java program also uses, **non-object resource memory** such as File Desciptors, window system fonts, etc
- The garbage collection can only free object resource memory but can not frees non-object resource memory.
- hence there will be chances of more memory wastage, to avoid this problem, java langauge supports **finalizer method.**
- the finalizer method is used by defining finalize( ) method.
- this method will get called to release object based memory as well as non-object resource memory.
- there are many library classes which are defined with finalize( ) method such as GrogorianCalendar.
- while using this class, finalize( ) method can be used.

e.g.

```
import java.util.*;
public class FinDemo extends GregorianCalendar {

  public static void main(String[] args) {
    try {
      // create a new FinDemo object
      FinDemo ob = new FinDemo();

      // print current time
      System.out.println("" + ob.getTime());

      // finalize cal
      System.out.println("Finalizing Started");
      ob.finalize();
      System.out.println("Finalized.");

    } catch (Throwable ex) {
      ex.printStackTrace();
    }
  }
}
```

Output



```
C:\Temp>javac FinDemo.java

C:\Temp>java FinDemo
Tue Mar 03 11:48:05 IST 2020
Finalizing...
Finalized.
```

❖ Abstract Method & Abstract Class

Abstract Class

- A class which contains the abstract keyword in its declaration is known as abstract class.
- Abstract classes may or may not contain abstract methods, i.e., methods without body (public void get();)
- But, if a class has at least one abstract method, then the class must be declared abstract.
- If a class is declared abstract, it cannot be instantiated.
- To use an abstract class, you have to inherit it from another class, provide implementations to the abstract methods in it.
- If you inherit an abstract class, you have to provide implementations to all the abstract methods in it.

Syntax:-

```
abstract class ClassName
{
}
```

Abstract Methods

- If you want a class to contain a particular method but you want the actual implementation of that method to be determined by child classes, you can declare the method in the parent class as an abstract.
- "**abstract"** keyword is used to declare the method as abstract.
- You have to place the abstract keyword before the method name in the method declaration as

abstract <ReturnType> FuncName([arg]);

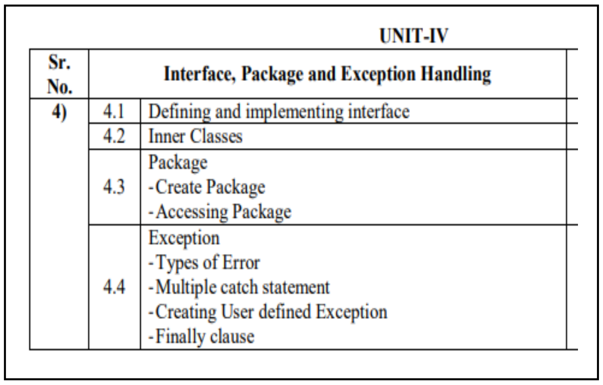
- An abstract method should only be declared, but not without method body.
- Instead of curly braces, an abstract method will have a semoi colon (;) at the end.

```
abstract class A
{
  abstract void display();  // declaring abst method
}
class AbDemo extends A
{
  void display() // Defining abst method
    {
        System.out.println("I am from Abstract Class");
    }
public static void main(String ar[])
{
   AbDemo ob=new AbDemo();
       ob.display();
}
}
```

❖ **this keyword**

- java as programming lanugage supports different tokens and one such token is keyword.
- there are many keywords supported java language, and one such important keyword is “this”.
- “this” as a keyword is useful in many cases such as:
this can be used to get the current object.
this can be used to invoke current object's method.
this() can be used to invoke current class constructor
- in other words, it can be said that, “this” keyword can be used to show:
    - current class
    - current method scope, etc
- e.g.

```
class A
    {
       int rno;String sname;
    A(int rno, String sname)
    {
      this.rno=rno; // assign rno value to rno of class.
    this.sname=sname; // assign sname value to sname of class.
    }
    void display()
    {
    System.out.println(“Roll
    No=”+rno+”Name=”+sname);
    }
    }
    class MainCls
    {
    public static void main(String ar[])
    {
      A ob=new A(101,”Abc”);
    ob.display();
    }
    }
```



UNIT-IV

Sr. No.	Interface, Package and Exception Handling	
4)	4.1	Defining and implementing interface
	4.2	Inner Classes
	4.3	Package -Create Package -Accessing Package
	4.4	Exception -Types of Error -Multiple catch statement -Creating User defined Exception -Finally clause

- Defining & Implementing Interface –

- Java as programming language derived from it predecessor languages C & C++.
- Due to this, it shows many similarites and also many differences.
- One important difference is in terms of inheritance i.e. java doesn’t support Multiple inheritance.
- This limitation can be solved using the concept of interface.
- Interface in java program is like a class which can include data member and function member.
- Interface in java program can be of following types:
    - **Built-in interface** – these are pre-defined interfaces avaialable in every jdk version such as:
      ActionListener, MouseListener, Runnable, etc
    - **User Defined Interface** – it is an interface created by programmer for any user application.
- Interface can be created using “interface” keword as:

```
interface <InterfaceName>
{
    [data member]
    [function member]
}
```

E.g.

```
interface Inft1
{
public int n=100;
public void display();
}
```

- Interface can enclose single or multiple data and function members.
- But as oppose to class, interface shows following important featuees or differences.

i. data member in interface always by default is final.

ii. Function member or method in interface must be abstract i.e only declared & not defined.

iii. As like class interface to be saved with ".java" extension, & can be compiled. **<u>But interface never get executed.</u>**

- Interface can be created and saved either in same java program file or can be separately in same location.
- **<u>Implementing interface</u>**
- Once an interface is ready, it can be used by any other java class.
- Using an existing interface is called implementing interface & for this, it uses special keyword i.e. "implements" as:

```
class <ClassName> implements <IntfName>
{
---
---
---
}
```

- The practical meaning of implmenting interface says:
  - The class which is implementing interface can use data member of interface but can not chance its value.
  - The class which is implementing interface must define the abstract method of interface.

e.g.

```
interface Ift1
{
 public double PI=3.14;
 public void display(int r);
}
```

→ Save it as: Ift1.java

→ Compile it as : C:\> javac Ift1.java

```
class A implements Ift1 // interface implemented by A class
{
  public void display(int r)
    {
      System.out.println("Area of Circle="+(PI*r*r));
    }
}
class IftDemo
{
 public static void main(String args[])
   {
        A ob=new A();
        ob.display(5);
   }
}
```

- Some important factors regarding interface are:
  - As like class, interface can also be extended using "extends" keyword.

```
interface A
{
---
}
interface B extends A
{
---
}
```

  - In case of interface, multiple inheritance is allowed as:

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

- Class can implment interface and use it, but vice-versa i.e. interface can use class not allowed.
- Each and every method of interface must be defined in class which is implementing interface.
- If interface containing multiple methods and any of the method from interface not required, then atleast, its empty defination to be given as:

```
interface Itf1
{
public void f1();
public void f2();
}
Class A implements Itf1
{
public void f1()
{
---  method Defination
----
}
public void f1(){} //empty defination
}
```

❖Inner Class❖

- class is one the most basic entity of OOP and for every executable java program there must be atleast one class.
- class as user defined datatype combines data and function members.
- as like control statements if, for, java language also allows to use class in nested form.
- when one class defined within other class it is called nested class.
- the class, in which other class is defined becomes outer class.
The class defined with-in outer class becomes inner class.
- Nested class can be given as:

```
class A // outer class
{
  ---
  ---
      class B // inner class
            {
               ---
               ---
            }
}
```

- the relation between, outer & inner class shows **important characteristics** such as: ❡
i. Inner class can access each and every member [data as well as function] of outer class.
ii. But outer class doesn't have any permission to access any member of inner class.
- **Rules** to access outer and inner class members:
i. outer class member can be directly access using its object.
ii. to access inner class member, object of inner class to be created with preceded by outer class as:
- OuterClassName.InnerClassName <ob>=<outClassObj>. New OuterClassName( )

e.g.

```
class A //outer class
{
   void display() // outer class method
    {
      System.out.print("I am Outer Class");
    }
  class B //inner class
     {
       void show() // inter class method
       {
        System.out.print("I am Inner Class");
        }
     }
}
class mcls
{
public static void main(String ar[])
 {
    A outObj=new A();
      outObj.display();
     A.B inObj=outObj.new B();
     inObj.show();
 }
}
```

◈Package◈

◈Creating & Accessing Package◈

- Java as programming language derived from it predecessor languages C & C++.
- Due to this, it shows many similarites and also many differences.
- One important difference is in terms of header files.
- C or C++ program requires inclusion of header files, but in case of java language, header files are replaced by package.
- **Package is a container which contains group of all related classes, interfaces or both for a single software.**
- In simple words, a package can be folder or directory in which all classes & interfaces required for a complete java application are stored.
- Packages gives many advantages:
    - Reusability – class created once & if stored in package, can be resued.
    - Unique Name – in once package class name will unique, but by same name class can be created in other package.
    - Class hiding – if class enclosed in package can be hidden for its internal details, but can be accessed from anywhere.
    - Less coding – due to reusability, length of code will be less.
- In java language, packages will be of following types:
- 1. System Package / Built-in package – it is also called API package which are already created and distributed with different JDK versions.

- There are many built-in packages supported java language such as:
- <diagram>

2. User Defined Package -

- This is the package created by programmer for a specific user application or sotfware.

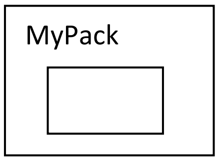
- these are not previously available in any JDK library.

<u>Creating Package</u>

- a user defined package can be created in following steps:

Step 1 : Create one folder in current working folder to be used as package. E.g.

C:\MyJava



- in above e.g., MyJava is current working folder, under which, MyPack folder is created as Package folder.

Step 2 : adding class to package

- once package is created, in this classes can be added.

- For this, while defining class, "package" keyword must be used to inform about package name as:

```
package <PackageName>;
package MyPack;
class <ClassName>
{
---
---
}
```

- a package may contain single class or any number of classes.

Step 3 : accessing package class

- once package & its class is ready, than it can be used by any other java program which is saved in its parnet folder.

- thus, if a java class stored in package folder, it can be accessed by other java program which stored in parent folder of package.

- for this, package class can be accessed in one of the following way:

1. Using Fully Qualified Name (FQN) – the packag class name written with complete package name as:

PackageName.className;

MyPack.Numbers;

2. using import statement – package class can be accessed using import statement.

- This statement can be used in following ways:

a. importing all classes – if from a package all or many classes to be accessed, following statement can be useful:

import packageName.*;

import MyPack.*;

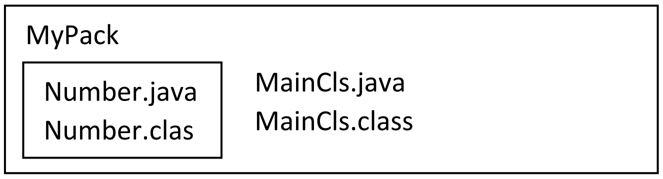
b. importing single class – if from a package single class to be accessed, following statement can be useful:

import PackageName.ClassName;

import MyPack.StudInfo;

e.g.

C:\MyJava



```
package MyPack;
public class Number
{
   public void display( int n)
     {
         if (n%2==0)
          System.out.println("Even Number");
         else
          System.out.println("Odd Number");

      }
}
```

- ➔ Save above program in MyPack folder as Number.java
- ➔ Also compile in same folder.

```
import MyPack.Number;
class MainCls
{
  public static void main(String ar[])
   {
       Number obj=new Number( );
        obj.display(10);
    }
}
```

- ➔ Save above program in MyJava folder as MainCls.java
- ➔ Also compile in same folder.
- ➔ Execute in same folder

- ➢ **Types of Error**

- a java program undergoes in different stages such as designing, coding, compiling and then execution phase.
- during these different stages, there are chances of different types of error may get generated.
- an Error is any invalid situation or statement in program which occurs when program breaks the rule of programming language environment.
- as program undergoes into different stages, following different types of error can be generated:
1. Compile time error 2. Logical Error
3. Runtime Error

1. Compile Time Error – When an error occurs during compilation phase of java program it is called compile time error.
- If a program having compile error , such program will not be compiled successfully and also does not get executed.
- this error is detected by compiler of respective language.
Such as:
- o Undefined symbol
- o Unclosed string, etc

- Such type of errors are easy to debug due to following reasons:
  - o It is detected by compiler itself.
  - o It shows line number of error
  - o Some advance compilers also shows reason & solution of error like NetBeans, Eclipse, etc

3. **Logical Error** – the error which is generated due to wrong or poor logic of program is called logical error.

- It occurred during execution phase of program but not giving correct output or expected output, suc
h type of error is called logical error.
e.g. use “<” instead or where “>” is required,
loop without termination, initializing fact value to zero (fact=0) in calculation of factorial, etc.
- such error is caused due to :
  - o Poor logic
  - o Poor programming language knowledge
  - o Poor understanding of software requirements, etc
- Such types of errors are little bit difficult to debug as compare to compile error.
- It needs good logic and repeated exeuction of program code to solve the error

3. Runtime Error – when error occurred during execution phase of program is called runtime error.
- Such runtime error also called as Exception & runtime error are said to be most difficult types of error to debug.
- When runtime error occurs it causes following:
  - o It abnormally terminates the program
  - o Important house-keeping programming operatons like closing of DB, file, etc can be skipped.
- there are many programming situations where there are chances to generate runtime error such as:
  - o Division by zero
  - o Reading from non existing file
  - o Accessing array element with out of index range, etc
- Such runtime error or Exception is must be handled due to following reasons:
  - o Avoid abnormal termination of program.
  - o To make std software.
- to handle runtime error, java language supports special keywords as exception handler like try, catch, throws, etc.

➢ **Handling exception**

when error occurred during execution phase of program is called runtime error.
- Such runtime error also called as Exception & runtime error are said to be most difficult types of error to debug.
- When runtime error occurs it causes following:
  - o It abnormally terminates the program
  - o Important house-keeping programming operatons like closing of DB, file, etc can be skipped.
- there are many programming situations where there are chances to generate runtime error such as:
  - o Division by zero
  - o Reading from non existing file
  - o Accessing array element with out of index range, etc
- Such runtime error or Exception is must be handled due to following reasons:
  - o Avoid abnormal termination of program.
  - o To make std software.
- To handle any exception, java supports many built-in exception handling classes such as:
ArithmaticException,
ArrayIndexOutOfBoundsException, SQLException, etc.

- **<u>All these exception classes are derived from “Exception” base class</u>.**

- to handle runtime error, java language supports special keywords as exception handlers like:

1. throws, 2. try….catch 3.finally.

<u>1. throws</u> – this keyword as exception handler works in following way

It will call system defined exception handler.
It will show predefined error message.
It will stop further execution of program.

- “throws” keyword should be used with the function in which the code is enclosed which may generate exception.

- it can be written as:

```
<RetType> FuncName([arg]) throws <Exception>
{
---
---
}
```

e.g.

```
class Ex1
{
  public static void main(String ar[]) throws Exception
   {
      int n=10,m=0;
System.out.println("Addition="+(n+m));
System.out.println("Division="+(n/m));
System.out.println("Subtraction="+(n-m));
System.out.println("Multiplication="+(n*m));
    }
}
```

- This above program will only execute addition, from division statement exception generated and further execution will be stopped.

<u>1. try…catch</u> – these keywords are used to defined exception handling block.

- all those statement by which possibly exception may be generated, are enclosed in try block.

-In catch block, the code to handle exception can be enclosed.

- If exception occurs, it will execute catch block and still further code exeuction will be continued and thus abnormal termianation can be easily avoided.

- It can be used as:

```
try
{
---
---
---
}catch(<ExceptionClass> obj>{-- code ---}
```

e.g.

```
class Ex2
{
  public static void main(String ar[])
   {
      Scanner sc=new Scanner(System.in);
System.out.print("Enter Ist No:");
int n=sc.nextInt();
System.out.print("Enter IInd No:");
int m=sc.nextInt();

System.out.println("Addition="+(n+m));
try
{
  System.out.println("Division="+(n/m));
}catch(Exception ex)
  {
System.out.println("Division by Zero not allowed");
  }
System.out.println("Subtraction="+(n-m));
System.out.println("Multiplication="+(n*m));
    }
}
```

- Thus, in above code, if user enter 2nd value as non-zero, program succesffully performs all arithmatic operations.

- But it, 2nd number is zero, it will go to catch block and then also exeuctes remaining part of program.

➢ **Multiple catch statement**

In catch block, the code to handle exception can be enclosed.

- If exception occurs, it will execute catch block and still further code exeuction will be continued and thus abnormal termianation can be easily avoided.

- It can be used as:

```
try
{
---
---
---
}catch(<ExceptionClass> obj>{-- code ---}
```

- in many cases it is also found that, same code may generate multiple exceptions.

- And for any std. software it is necessary to handle each & every exception.

- For this, java language allows to use multiple catch blocks with single try block as:

try

```
{
---
---
---
}catch(<ExceptionClass> obj>{-- code1 ---}
catch(<ExceptionClass> obj>{-- code2---}
catch(<ExceptionClass> obj>{-- code3 ---}
```

- Thus, any number of catch blocks can be used with single try block.
- This will ensure, all possible exceptions can be handled successfully.

e.g.

```
class Ex3
{
  public static void main(String ar[])
   {
      Scanner sc=new Scanner(System.in);
try
{

System.out.print("Enter Ist No:");
int n=sc.nextInt();
System.out.print("Enter IInd No:");
int m=sc.nextInt();
System.out.println("Addition="+(n+m));
  System.out.println("Division="+(n/m));
System.out.println("Division by Zero not allowed");
System.out.println("Subtraction="+(n-m));
System.out.println("Multiplication="+(n*m));
}
 catch(NumberFormatException ex)
 {
System.out.println("Enter only Int value");
 }
 catch(ArtihmeticExcpeiton ex )
 {
 System.out.println("Division by Zero not allowed");
 }
}
}
```

➢ Creating user Defined Exception

- when error occurred during execution phase of program is called runtime error.
- Such runtime error also called as Exception & runtime error are said to be most difficult types of error to debug.
- java supports exceptoins of following types:

1. Built-in Exceptoin 2. User Defined Exception

-Built-in exceptions are predefined exception classes such as:
ArithmaticException,
ArrayIndexOutOfBoundsException, SQLException, etc.

- User Defined Exception – it is an exception class defined by programmer for a specific user application.
- e.g.

Entering –ve value rate or qty in billing application.
Entering out of range value in MarkSheet application.

- To handle such types of exception, User Defined exception can be create and used in following steps:

1. Create one class inherited from Exception lib class.
2. Define its constructor for handling error.

```
class <ClassName> extends Exception
{
---
---
}
```

3. User Defined exception must be explicitly called using "throw" keyword

```
throw new <ClassName>( );
```

e.g.

```
class NegValue extends Exception
{
   NegValue( )
   {
     System.out.println("Don't use –ve value for Rate or Qty");
    }
}

class Ex4
{
  public static void main(String ar[])
   {
      Scanner sc=new Scanner(System.in);
try
{

System.out.print("Enter Rate:");
int rate=sc.nextInt();
System.out.print("Enter Qty:");
int qty=sc.nextInt();
if(rate<=0 || qty<0)
throw new NegValue( ); // calling User Def Expt
else
System.out.println("Amount="+(rate*qty));
}
```

}
}

Finally clause

- java language supports many exception handlers & one such is finally clause.
- A java program during exeuction need to perform many different operations.
- Among these, some might be optional and some may be very compulsory or critical.
- **The critcal operatons that every java program must perform before closing or termination called as House Keeping Operations.**
- These house keeping operations may be :

  Closing of DB
  Closing of DB Connection
  Closing of file, etc

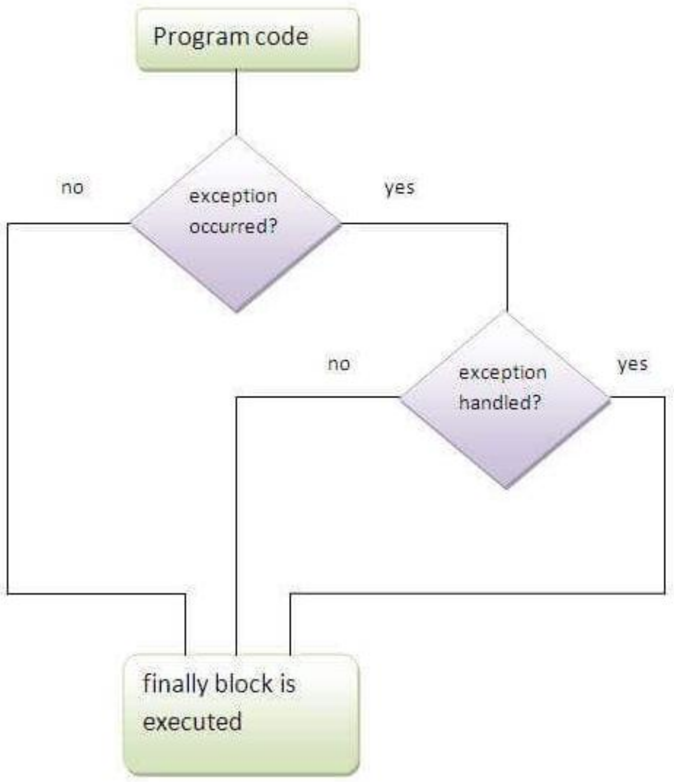
- But, in some cases, if exceptions generated, these important operations may get skipped and JVM will not perform these house keeping operations.
- To solve this problem Java Lang supports one important and very useful keyword or clause i.e. “finally”.
- using this keyword, a block of executeable code can be defined. But for this, there must by try block as:

```
try
{
---
---
---
}
finally
{
---
---
---
}
```

- To use, finally block, try block is compulsory, catch block is optional i.e. finally can be used with or with catch block.

```
try
{
---
---
---
}
catch(<ExcepClassName> obj) {------}
finally
{
---
---
---
}
```

-The flow chart representation of finally block can be given as:



- When finally block is added to code, for JVM it is compulsory to execute this code before closing the program.

e.g. Reading one character from given file. In this, in any case, the file must be closed before closing the program.

```
import java.io.*;
class Fio1
{
  public static void main(String args[])
 {
     FileReader Fobj = null;
     try {
       Fobj = new FileReader("input.txt");
       int c;
          c = Fobj.read() ;
         System.out.print((char)c);
     }catch(IOException ex){}
     finally
      {
          System.out.print(“This is Finally block”);

       }
   }
}
```

Unit - V

Sr. No.	String, Stream and Files		Lectures Required	Ref. No
5)	5.1	Introduction	1	1,2,3,4
	5.2	String Classes	1	1,2,4
	5.3	StringBuffer Class	1	1,2,4
	5.4	Stream Classes -Types of Streams - Byte Stream Classes - Character Stream Classes	2	1,2,4
	5.5	File Classes	1	1,2,4

◈String Class◈

- Java a programming language allows to use different types of data like int, float, double, boolean, char etc.
- A char variable at a time can store and process only 1 char, but in many real-time applications group of characters is needed.
- *<u>Group of character or sequence of characters in programming is called String.</u>*
- There are many examples where, a program need to store and process string such as StudName, EmpName, SubjName, etc.
- In java program, string can be created in one of the following ways:
    - Using char Array
    - Using String object
- As like, C & C++, in java langauge also char array can be used to create a string as:
- char x[ ]={"SRTMUN"};
- But use of char array is not recommened as it provides very less options for string manipulations.
- For this, String object can be used.
- String is one built-in class in java lang available in **<u>java.lang</u>** package.
- It's object can be created as:

    String <obj> = new String ("Value");
    String s=new String("SRTMUN");
    or
    String s="SRTMUN";

- This string class provides following built-in methods:

1. length() – this method counts and returns number of characters in string including spaces.
StringObj.length();

2. toLowerCase() – this method converts each character of string into lower / small case.
StringObj.toLowerCase();

3**. toUpperCase()** – this method converts each character of string into upper / capital case.
StringObj.toUpperCase();

4. **equals( )** – this method used to compare one string with other in terms of length, characters, and case. If all are same, it returns zero that means both strings are equal/same as:
StringObj.equals("StringValue");

5. equalsIgnoreCase( ) – this method used to compare one string with other only in terms of length and characters, but it ignores the case.
StringObj.equalsIgnoreCase("StringValue");

6. concat( ) – This string is used to append new string at the end of existing string.
StringObj.concat("String");

7. trim( ) - this string used to remove all leading (left side) and trailing (right side) spaces of given string.

8. substring(int start, int end) – it extract number of characters from start index to end index -1 as substring.
StringObj.substring(int start,int end)

9. charAt(index) – this method is used to extract only one character of given index from string.
StringObj.charAt(index);

e.g.

```
class str
{
 public static void main(String ar[])
  {
    String s="SRTMU";
   System.out.println("Length="+s.length());
System.out.println("Lower Case="+s.toLowerCase());
System.out.println("Compare="+s.equals("srtmu"));
System.out.println("Compare="+s.equalsIgnoreCase(
"srtmu"));
s=s.concat(" Nanded");
System.out.println("New String="+s);
System.out.println("one          char          from
string="+s.charAt(3));
System.out.println("Substring="+s.substring(3,7));
}
}
```

Output –

```
Length=5
Lower Case=srtmu
Compare=false
Compare=true
New String=SRTMU Nanded
one char from string=M
Substring=MU N
```

◈StringBuffer Class◈

-Java a programming language allows to use different types of data like int, float, double, boolean, char etc.
- A char variable at a time can store and process only 1 char, but in many real-time applications group of characters is needed.
- *Group of character or sequence of characters in programming is called String.*
- There are many examples where, a program need to store and process string such as StudName, EmpName, SubjName, etc.
- In java program, string can be created using **String object**.

- But String class having some limitations i.e. it creates static string or fixed length string.
- In this, in between the string new string can not be inserted or removed.
- to overcome these problems, java lib support StringBuffer class.
- StringBuffer is one peer class of String from java.lang package which is used to create dynmaic string.
- StringBuffer class can be used as:
StringBuffer <Obj>=new StringBuffer(“string”);
StringBuffer sb=new StringBuffer(“SRTMUN”);
- This class provides all the functionalities of String class and also supports some additional functions like:

1. append(String) – this method is used to append new string at the end of existing string as:
<StringBufferObj>.append(“String”);

2. insert(int index, string) – this method used to insert new string at given index position as:
<StringBufferObj>.insert(int index, “Newstring”);

3. replace(int start,int end, “string”) – this method used to replace charcaters from start index till end index -1 by new string.
<StringBufferObj>.replace(int index,int end “Newstring”);

4. delete(int start,int end) – this method used to delete charcaters from start index till end index -1 .
<StringBufferObj>.delete(int index,int end);

5. reverse( ) – this method is used to reverse the given string characters.
<StringBufferObj>.reverse();

e.g

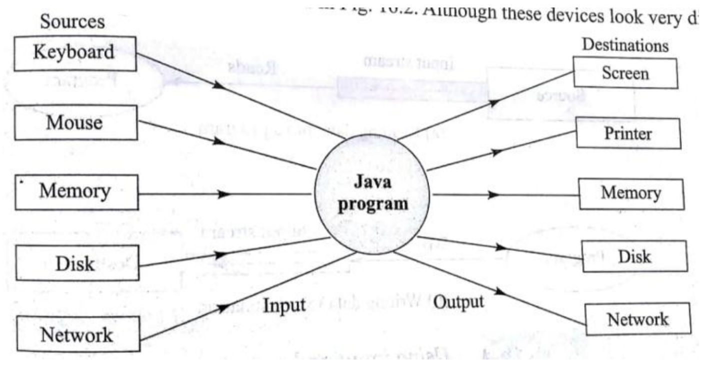
```
class strb
{
  public static void main(String ar[])
  {
     StringBuffer sb=new StringBuffer("Jack");

System.out.println("String="+sb+"\tLength="+sb.length());
sb.append(" Back");
System.out.println("New String="+sb);
sb.insert(5,"is ");
System.out.println("New String="+sb);
sb.replace(9,12,"lue");
System.out.println("New String="+sb);
sb.delete(5,7);
System.out.println("New String="+sb);
sb.reverse();
System.out.println("Rev String="+sb);
}
}
```

Output –

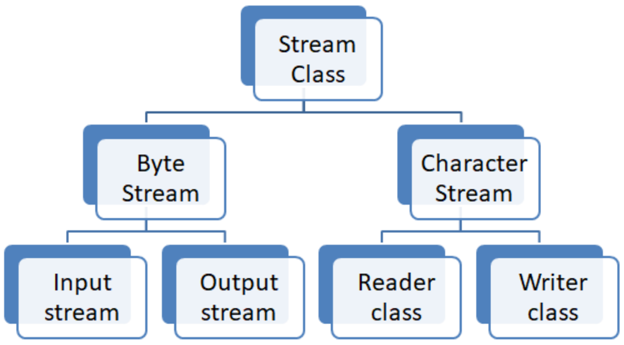
```
String=Jack     Length=4
New String=Jack Back
New String=Jack is Back
New String=Jack is Blue
New String=Jack  Blue
Rev String=eulB  kcaJ
```

◈Stream Class◈

- As like any other programming language, a java program can also different operations.
- one of the important feature of any program is, it must be interactive .
- A java program needs to get input with-in program and also should be able to give output from program.
- For this java program uses **Data Flow** which can be of following types:
    - flow of data into program
    - flow of daout out of program
- To complete this job of input and output data flow, java language uses concept of Stream.
- Stream is ordered sequence of data that a java program uses for input or to give output.
- Stream is path or channel using which data can be taken as input or given as output.
- A stream uses two ends, **Source** to get input and **destination** to give output.
- Stream in other words is – uniform, easy to use objected oriented interface between java program and input/output devices.
- A java program can get input from different sources and can give output to different sources as:



- The library classes which provides such stream and functionalities to complete input –output operations called Stream Classes.
- These classes are available in java.io package of java library.
- All the stream classes are braodly classified in two main categories based on type of data they use.
    - **Byte stream classes** – performs Input-Output operations using bytes.
    - **Character stream classes** – performs Input-Output operations using characters.
    - These stream classes are further sub-classified as:



◈ Character Stream Classes◈

- These classes for File I/O are used to perform read & write opepration on file using sequence of characters.
- for this, two most commonly used library classes are:

FileReader
FileWriter

i. FileReader – this lib class belongs to java.io. package.
- This class can be used to perform read operation on given file, for this it's object can be created as:
FileReader <Obj>=new FileReader("path & fileName");
e.g.
FileReader fr=new FileReader("input.txt");
- If file is available in current location of java program, only file name is sufficient, but if it is in different location than, complete path must be given as:
FileReader fr=new FileReader("C:/abc/input.txt");
- **<u>to get input from any file, that file must be existing.</u>**
- this class provides built-in method read( ) which can read one character at time from file.

<u>e.g.1 reading one character</u>

```
import java.io.*;
class Fio1
{
  public static void main(String args[])
{
    FileReader Fobj = null;
    try {
      Fobj = new FileReader("input.txt");
      int c;
 c = Fobj.read() ;
        System.out.print((char)c);
    }catch(IOException ex){}
    finally
     {
          try
        {
        Fobj.close();
        }catch(IOException e){}

      }
   }
}
```

<u>e.g.2 reading all characters from file.</u>

```
import java.io.*
class Fio2
{
  public static void main(String args[])
{
    FileReader Fobj = null;
    try {
      Fobj = new FileReader("input.txt");
      int c;
      while ((c = Fobj.read()) != -1)
        {
        System.out.print((char)c);
        }
    }catch(IOException ex){}
    finally
     {
          try
        {
        Fobj.close();
        }catch(IOException e){}
      }
  }
}
```
ii. FileWriter - this lib class belongs to java.io. package.

- This class can be used to perform write operation on given file, for this it's object can be created as:

FileWriter <Obj>=new FileWriter("path & fileName");

e.g.

FileWriter fr=new FileWriter("input.txt");

- If file is available in current location of java program, only file name is sufficient, but if it is in complete path must be given as:

FileWriter fr=new FileWriter("C:/abc/input.txt");

- While using FileWriter class, following factors to be rememberd:

1. if file is not existing, it will be created by JVM by given name in given location.
2. if file is existing, by default it will be overwritten.
3. to avoid overwritting of file, folllwing constructor to be used:

FileWriter fr=new FileWriter("input.txt",true);

- this class provides built-in method write( ) which can write one character at time in file.

e.g.
```
import java.io.*;
class Fio3
{
  public static void main(String args[])
{
    FileWriter Fobj = null;
    try {
      Fobj = new FileWriter("input.txt",true);
      Fobj.write('B');
    }catch(IOException ex){}
    finally
     {
         try
        {
        Fobj.close();
        }catch(IOException e){}
      }
  }
}
```
◈ File Class◈

Java File class represents the files and directory pathnames in an abstract manner.

This class is used for

creation of files and directories,

file searching,

File or Directory rename

To check file / directory properties like read-only, hidden, etc

File/directory deletion, etc.

The File object represents the actual file/directory on the disk. Following is the list of constructors to create a File object.

The object of this class can be created as:

File <obj>=new File(String pathname);

e.g.

File ob=new File("c:\\abc\\f1.txt");

File ob=new File("C:/abc/f1.txt");

- This class supports many file and directory related built-in methods:

getParent() – this method returns the name of parent directory in which file is present.

getPath()- this method returns complete path and name of file including directory and sub-directory.

exists() – this method returns true if the file is available by the given name else returns false.

canWrite() - this method returns true if the file is writable by else returns false.

canRead() - this method returns true if the file is readable by else returns false.

length() – this method returns the length of file as integer i.e. number of characters present in file.

```
import java.io.*;
class Fio3
{
  public static void main(String args[])
{
    File Fobj= new File("c:\\temp\\input.txt");
        System.out.println("Get Parent=" +
Fobj.getParent());
        System.out.println("Get Path=" +
Fobj.getPath());
        System.out.println("File is Existing=" +
Fobj.exists());
System.out.println("File is Writable=" +
Fobj.canWrite());
System.out.println("File is Readable="
+Fobj.canRead());
System.out.println("File is Length=" +
Fobj.length());
   }
}
```

Output –

```
Get Parent=c:\temp

Get Path=c:\temp\input.txt

File is Existing=true

File is Writable=true

File is Readable=true

File is Length=2
```

Unit -VI

Sr. No.	Applet Programming	
6)	6.1	Introduction
	6.2	Creating Applets
	6.3	Applet Life Cycle
	6.4	Applet Tag
	6.5	Passing Parameters to Applets
	6.6	Working with Graphics

◈ Introduction & Creating Applet◈

- Java programming language is used to create both types of program like application as well as applet.
- Application is java program which get executed using local system environment.
- Applet is small java program created for internet computing which can be transported over the internet and can be loaded in browser or applet viewer.
- due to applet programing, java contributed to convert static web pages into dynamic web pages.
- As like applicatoin , an applet can also perform different operations like:
    - ➢ Arithmetic operatoins
    - ➢ Dispay graphics,
    - ➢ Play sounds, get user input
    - ➢ Gives output
    - ➢ Create animation, etc
- Applet in java program can be created in following types:
- **Local applet** – if the applet code is saved on local hard disk and loaded in browser, it is local applet.
- **Remote applet** – when the applet code is saved on server, transported on client machine and loaded in browser, it is Remote Applet.
- To create an applet, services of two library classes are required such as:
    i. Applet class from java.applet package
    ii. Grpahics class from java.awt package.
- To create an applet, a java class to be create which must be extended from Applet class as:

```
class <ClassName> extends Applet
{
---
---
}
```

- Applet as a library class provides many built-in methods like
- init( ) – from where applet initialized
- start( ) – applet execution starts.
- paint( ) – it is used to perform different drawing operations on applet window.
- stop( ) – by this, applet exeuction is stopped temporarily.
- destroy( ) – applet exeution is stopped permenantly.

- As like application, an applet code also to be saved as className.java extension.
- As like applicaton, applet code can alo be compiled using java compiler.
- But to execute applet, it should be loaded in applet viewer or in any java enabled browser (e.g hot java) using HTML code.
- For this, HTML supports a special tag i.e. <Applet> - a paired tag - with different attributes.
- This HTML code can either be in same java code file enclose in comment entry /*….*/ or it can be stored in separate file.
- The HTML code can be given as:

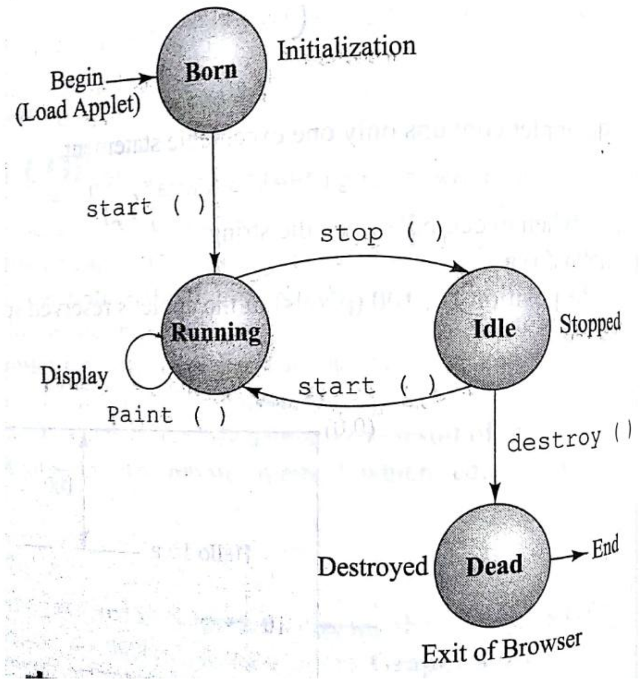
```
<Applet code="ClassName" width=int height=int>
</applet>
e.g.
import java.awt.*;
import java.applet.*
public class Ap1 extends Applet
{
  public void paint(Graphics g)
   {
       g.drawString("Hello & Welcome",10,20);
    }
}
/* <Applet code="Ap1" width=500 height=500>
</applet>
*/
```

- Save the above code as "Ap1.java"
- compile the above code as:
javac Ap1.java
  - Load the applet in applet viewer as:
  appletviewer Ap1.java
- in above e.g., drawString( ) is one lib function of Graphics class which can draw some output on given col (x) & row (y) location.

◈ Applet Life Cycle◈

- Java programming language is used to create both types of program like application as well as applet.
- Applet is small java program created for internet computing which can be transported over the internet and can be loaded in browser or applet viewer.
- Every java applet from it creation stage to end stage undergoes in different stages which shows its life cycle.

- The different stages of applet from its creation point to end point represents applet life cycle.
- Each stage of applet also uses it's built-in method supported by Applet class.
- Applet life cycle can be represented as:



- The diagram show following different stages in applet life cycle:

1. Initialization state [init ( ) method]
2. Running state [start( ) method]
3. Display state [paint( )]
4. Idle or stopped state [stop( )]
5. Dead state [destroy( )]

1. Initialization State – Every applet execution starts with initialization state.
- This state is controlled by init( ) method of Applet class.
- Thus, it is clear that every applet execution starts with init( ) method, which is used to perform following:
  - ➢ Creating object needed for applet
  - ➢ Setting up initial values
  - ➢ Loading images, fonts, etc
  - ➢ Setting color, etc
- In one life cycle, this stage comes only once and hence init( ) executed only once.
- This method can be defined as:

```
public void init( )
{
---
---
}
```

- After finishing its job, init( ) automatically calls to start() method.

2. Running State – After initialization state, applet enteres in this, Running state.
- This state is controlled by start( ) method of Applet class.
- this method can be defined as:

```
public void start( )
{
---
---
}
```

- This state can be repeated in one life of applet, as applet if it is in idle state can come back to running state.
- once the exeuction of start( ) method over, it automatically calls to paint( ) method.

3. Display State – When the job of running state is over, applet enters in display state.
- When applet is perforimg different drawing operation of applet window, it is said to be in display state.
- This state is controlled by paint( ) method of Applet class.
- this method is defined with abstract class "Graphics" of java.awt package as:

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

- this state can be repeated in one life of applet.

4. Idle State – When applet execution is temporarily stopped, it is said to be in idle state.
- If applet is in idle state, it is still in memory and also it does not releases the allocated objects.
- any applet enters in idle state when its stop( ) method is called.
- this stage can be controlled by stop( ) which can be defined as:

```
public void stop( )
        {
                ---
                ---
        }
```

- this stage can be repeated in one life cycle of applet.
- from this stage applet can either enter into to dead stage or running stage.

5. Dead State - When applet execution is permenantly stopped, it is said to be in Dead state.
- Every applet can not directly enter into dead state, first it enteres in idle state by calling stop( ) method

and then enters in dead state by called destroy( ) method.
- Before entereing in dead state following operations are performed on applet:
  - All the resources are taken back from applet.
  - All the objects created are deleted
  - Applet is unloaded from memory
- in one life cyle, dead state comes only once for any applet.
e.g.
Applet program showing its different stages:
```
  import java.awt.*;
  import java.applet.*
  public class Ap2 extends Applet
  {
        public void init( )
        {
         setBackground(Color.red);
        }
        public void start( )
        {
         setForeground(Color.blue);
        }
        public void paint(Graphics g)
        {
        g.drawString("Hello & Welcome",10,20);
        }
        public void stop( )
        {
        System.out.println("Applet stopped");
        }
        public void destroy( )
        {
        System.out.println("Applet Destroyed");
        }
  }
  /* <Applet code="Ap2" width=500 height=500>
  </applet>
  */
```

◈ Applet Tag◈

- Java programming language is used to create both types of program like application as well as applet.
- Applet is small java program created for internet computing which can be transported over the internet and can be loaded in browser or applet viewer.
- As oppose to application, applet can not be directly executed.
- for execution, applet to be loaded in browser using HTML code.
- For this HTML supports, a paired tag <Applet>…</Applet>
- this tag supports many attributes, among these, some are optional and some are compulsory.
- The different attributes are :

1. code – this is one **compulsory** attribute to be used with <Applet> tag.
this attribute is used to specify class name of local applet to be loaded in browser as:
<applet code="ClassName">…</Applet>

2. codebase – this attribute is used to specify the URL of remote applet to be loaded from internet.
<applet codebase="URL">…</Applet>

3. width - this is one **compulsory** attribute to be used with <Applet> tag.
- This attribute uses one int value which decides the horizontal size of applet as:
<applet code="ClassName" width=int>…</Applet>

4. height- this is one **compulsory** attribute to be used with <Applet> tag.
- This attribute uses one int value which decides the vertical size of applet as:
<applet code="ClassName" height=int>…</Applet>

5. hspace - This attribute uses one int value which decides the horizontal space before and after applet and also between two applets as:
<applet code="ClassName" hspace=int>…</Applet>

6. vspace - This attribute uses one int value which decides the vertical space before and after applet and also between two applets as:
<applet code="ClassName" vspace=int>…</Applet>

7. align – This attribute is used to decide applet alignment of applet in browser window or applet viewer.
- This attribute use values like LEFT,TOP,BOTTOM, RIGHT, MIDDLE.
- by default applet uses LEFT property as:
<applet code="ClassName" align=Right>…</applet>

8. alt – this attribute of <applet> tag used to assign alternate text which will be displayed in case if applet is not getting displayed.
- Generally it is used to show error message if applet is not getting loaded.

<applet code=”ClassName” alt=”text”>…</applet>
Syntax:

```
<Applet
code=”ClassName”
width=int       height=int
vspace=int      hspace=int
align=TOP|Left|Right|Bottom|Middle
alt=”text”>
</applet>
```

e.g.

```
  import java.awt.*;
  import java.applet.*
  public class Ap3 extends Applet
  {
    public void paint(Graphics g)
     {
        g.drawString(“Hello & Welcome”,10,20);
      }
  }
  /* <Applet code=”Ap3” width=500 height=500
  vspace=10 hspace=10 align=right alt=”Sorry Applet
  can not be Loaded”>
  </applet>
  */
```

◈ Passing Parameter Applet ◈

- Java programming language is used to create both types of program like application as well as applet.
- Applet is small java program created for internet computing which can be transported over the internet and can be loaded in browser or applet viewer.
- As oppose to application, applet can not be directly executed.
- for execution, applet to be loaded in browser using HTML code.
- as like java application, a java applet can also receive external input which can be provided by HTML code.

- For this 2 parts are required:
1] <Param> tag in HTML code as sender
2] getParameter( ) method in java code as receiver.

- HTML supports a special tag i.e. <Param> tag with its attributes as:

<Param name=”ValidName” Value=”any value”>

- the importance of these attributes are:

1. name – ths attribute specifies the valid name of parameter to be passed from HTML code to java applet.
- this name is case sensitive and must be valid name (without space)

2. value – this attribute specifes the actual value to be passed to java applet.
- one <param> tag can be used to pass only 1 value / paramet at a time.
- if mulitple parameters to be passed, multiple <Param> tags to be used.

getParameter()

- It is one lib function in Applet class which is used to receive parameter in java code.
- getParameter() method can be used as:

String <obj>=getParameter(“ParmName”);

- the parameter name must be same as given in HTML code in same case.
- it always receives parameter only in string type.
- It can receive only one parameter at a time

- if numeric value passed as parameter then it must be typecasted using:

Integer.parseInt( ) – converts string to int
Double.parseDouble( ) – converts string to double

e.g.

```
  import java.awt.*;
  import java.applet.*
  public class Ap4 extends Applet
  {
    public void paint(Graphics g)
     {
  String studname=getParameter(“sname”);
  String temp=getParameter(“marks1”);
  int m1=Integer.parseInt(temp);
  temp=getParameter(“marks2”);
  int m2=Integer.parseInt(temp);
  int tot=m1+m2;
        g.drawString(“Stud Name=”+studname,10,20);
        g.drawString(“Marks1=”+m1,10,40);
        g.drawString(“Marks2=”+m2,10,60);
        g.drawString(“Tot Marks=”+tot,10,80);

      }
  }
  /* <Applet code=”Ap3” width=500 height=500 >
  <param name=”sname” value=”David”>
  <param name=”marks1” value=”35”>
  <param name=”marks2” value=”45”>
  </applet>
  */
```

◈ Working with Graphics ◈

- A java program can perform different operations and provide many features.
- One such important feature of java lang. is use of graphics.
- A GUI based java program (application or applet) both provides drawing area where different drawing operations can be performed.
- The graphics drawing of java uses coordinate system made of 2D pixels using x(col) & y(row) order.
- It uses the origin 0,0 i.e. upper-left corner of the drawing area.
- Value of x (column) increases from left to right and value of y (row) increases from top to bottom.
- To perform different graphics related operations, java language supports special class "Graphics".
- "Graphics" is one abstract class on which object can not be created, it belongs to **java.awt package**.
- It supports following built-in methods:

1. drawString("string",int x,int y) – this method uses 3 parameters:

- string – to be drawn on graphics window
- x – value of column from where output to be started.
- y – value of row from where output to be started.

e.g.

drawString("Welcome", 10,30);

2. **drawLine(int x1,int y1,int x2,int y2)** – this method uses four int parameters, x1,y1 as origin and x2,y2 as end point for drawing line.

e.g.1

drawLine(10,20,100,20) – this will draw horizontal straight line.

e.g.2

drawLine(10,20,10,100) – this will draw vertical straight line.

3. drawRect(int x,int y, int width, int height) – this method uses four int parameters, x, y as origin and width and height decides the size.

- If the width and height is same it forms sqaure else it will draw rectangle.

e.g. 1

drawRect(10,20,100,100) – it will draw square.

drawRect(10,20,100,30) – it will draw rectangle.

4. fillRect(int x, int y, int width, int height) – the previous method draws empty rectangle (without colour filled).

- this method works in same way but it draws square or rectangle with filled colour.

fillRect(10,20,100,100)

5. drawRoundRect(int x,int y, int width, int height, int WidthArc, int HeightArc) – this method works as like drawRect() but it also uses two extra parameters for arc angles.

e.g.

drawRoundRect(10,20,30,30,10,12)

6. drawOval(int x,int y, int width, int height) – This method is used to draw circle or ellipse.

- when the width and height values are same, it draws circle else it will be ellipse.

- This method uses four parameters, x , y as origin and width and height to decide the size.

e.g.

drawOval(10,20,50,50);

7. fillOval(int x, int y, int widht, int height) – This method uses four parameters, x , y as origin and width and height to decide the size.

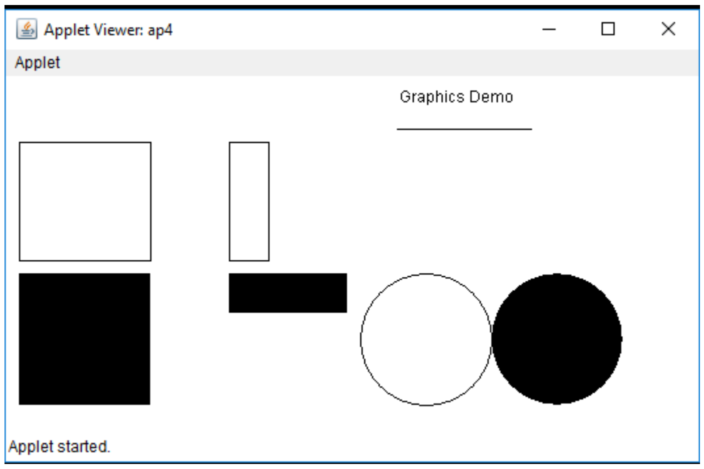
- It draws colur filled circle or ellipse.

fillOval(10,20,50,50);

Complete E.g

```
import java.awt.*;
import java.applet.*;
public class ap4 extends Applet
{
public void paint(Graphics g)
{
g.drawString("Graphics Demo",300,20);
g.drawLine(298,40,400,40);
g.drawRect(10,50,100,90);
g.fillRect(10,150,100,100);
g.drawRect(170,50,30,90);
g.fillRect(170,150,90,30);
g.drawOval(270,150,100,100);
g.fillOval(370,150,100,100);
}
}
/*
<applet code=ap4 width=700 height=700>
</applet>
*/
```

[use can either use above complete e.g. or any simple e.g.]


Applet Viewer: ap4
Applet
Graphics Demo
Applet started.