Module 2
(Objects and arrays, Namespaces, Nested classes, Constructors, Destructors.) covered in module 1.

Introduction to Java: Java's magic: the Byte code; Java Development Kit (JDK); the Java
Buzzwords, Object-oriented programming; Simple Java programs. Data types, variables and
arrays, Operators, Control Statements.

Introduction to Java

Java is a general-purpose object oriented programming language developed by Sun Microsystems.

Two types of Java applications can be created.

1. Stand alone Java application.
2. Web applets.

Stand alone Java application

Stand alone Java applications are programs written in Java to carry out a specific tasks on a stand alone system.

Executing a stand-alone Java program contains two phases:

a. Compiling source code into bytecode using **javac** compiler.

b. Executing the bytecode program using **java** interpreter.

Java environment includes a large number of development tools and hundreds of classes and methods.

Java development tools are part of the systems known as Java development kit (JDK) and the classes and methods are part of the Java standard library known as **Java standard Library (JSL)** also known as **Application Program Interface (API)**.

Java Features/Java Buzzwords:

1. Compiler and Interpreted
2. Architecture Neutral/Platform independent and portable
3. Object oriented
4. Robust and secure.
5. Distributed.
6. Multithreaded and interactive.
8. Dynamic and extendible.

1. Compiled and Interpreted:

Usually a computer language is either compiled or interpreted. Java combines both these approaches;

First java compiler translates source code into bytecode instructions. Bytecodes are not machine instructions,

In second step, java interpreter generates machine code that can be directly executed by the machine that is running the java program.

2. Architecture Neutral/Platform independent and portable

The concept of **Write-once-run-anywhere** (known as the Platform independent) is one of the important key features of java language that makes java as the most powerful language.

The programs written on one platform can run on any platform provided the platform must have the JVM.

3. Object oriented

In java everything is an Object. Java classes can be easily extended since it is based on the Object model. Java is a pure object oriented language.

4. Robust and secure

Java is a robust language; (Robust programming is **a style of programming that focuses on handling unexpected termination and unexpected actions**.)

Java makes an effort to eliminate error situations by emphasizing mainly on compile time error checking and runtime checking.

Absence of pointers in java, accounts for full data security.

5. Distributed

Java is designed for the distributed environment of the internet.

7. Multithreaded and interactive

With Java's multi-threaded feature it is possible to write programs that can do many tasks simultaneously. This design feature allows developers to construct smoothly running interactive applications.

8. Dynamic and extendible

Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment.

Java programs can carry an extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

Java Development kits(jdk)

Java development kit has a number of Java development tools.

(1) Appletviewer: Enables to run Java applet.
(2) javac: Java compiler.
(3) java: Java interpreter.
(4)javah :Produces header files for use with native methods.
(5) javap :Java disassembler.
(6) javadoc :Creates HTML documents for Java source code file.
(7) jdb :Java debugger which helps us to find the error.

Introduction to Java : **Java Program**

File name : **First.java**

```
class First
{
  public static void main(String args[ ])
  {
     System.out.println(“Welcome to JAVA”);
  }
}
```

“class First” declares a class, which is an object-oriented construct. **First** is a Java identifier that specifies the name of the class.

A single java program file can have ‘n’ number of classes. **But the name of the class which contains main function must match the name of the file.** Hence, this program name is First.java

Introduction to Java : First Java Program

Since, Java is a pure object oriented programming language, even main function must be declared inside the class.

Since, main is public and static no need of creating an object of type class First, the executor in this JVM (Java Virtual Machine) will call the main function by using class name.
Ex:

```
First.main();
```

Execution of a java program starts from main function.

public key word is an access specifier that declares the main method accessible by executor.

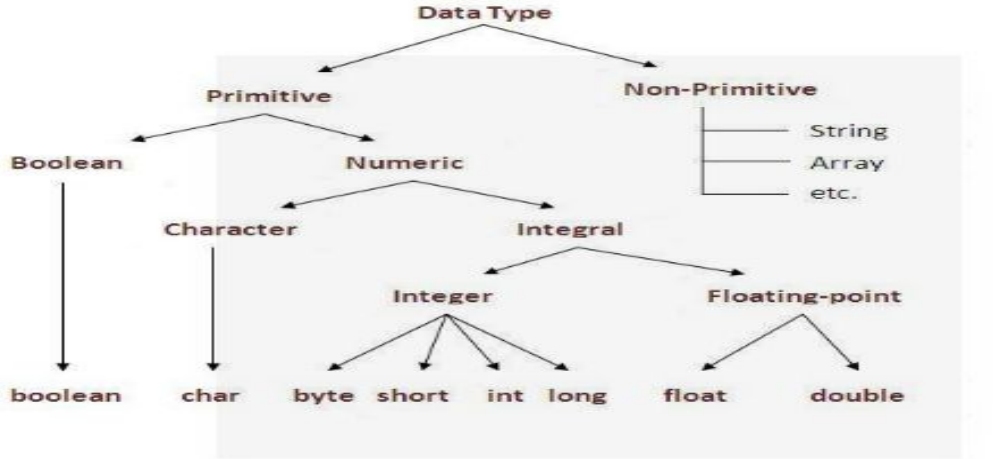
static keyword defines the method as one that belongs to the entire class and not for a particular object of the class. **main must always be declared as static.**

The type modifier void specifies that the method main is not returning any value.

Introduction to Java : **Data types**

In java, data types are classified into two categories

1.Primitive & 2.Non-Primitive Data type



Data Type	Default Value	Default size
boolean	False	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

Integer Type

Java provides four types of Integers. They are byte, short, int, long. All these are sign, positive or negative.

byte: smallest integer type is byte. This is a signed 8-bit type that has a range from –128 to 127. A byte variable is declared with the keyword "byte". Ex: byte b, c;

short: short is a signed 16-bit type. It has a range from –32768 to 32767. Specially used in 16 bit computers. Short variables are declared using the keyword short. Ex: short a, b;

int: Most commonly used Integer type is int. Signed 32 bit type has a range from –2147483648 to 2147483648. Ex: int a, b, c;

long: long is a 64 bit type and useful in all those occasions where int type of memory is not enough. Ex: long a, b;

Floating point types

Floating point numbers also known as real numbers are useful when evaluating an expression that requires fractional precision. The two floating-point data types are float and double.

float: float type specifies a single precision value that uses 32-bit storage. float keyword is used to declare a floating point variable. Ex: float a, b;

double: Double type of variable is declared with double keyword and uses 64-bit value.
Ex: double c;

characters: Java data type to store characters is char. char data type of Java uses Unicode to represent characters. Unicode defines a fully international character set that can have all the characters of human language. Java char is a 16-bit type. The range is 0 to 65536.
Ex: char a='p';

boolean: Java has a type called boolean for logical values. It can have only one of two possible values. They are true or false. Ex: boolean a; a=true;

Stack memory is used to **store primitive type values and the addresses of objects.**

Non-primitive types
A variable of type class is termed as non-primitive types in java.

All variables/objects of type class must acquire memory dynamically using **new** operator in java.

There is no delete operator to reclaim the memory that has already been allocated.

The job of reclaiming memory will be done by a system program, which is part of JVM termed as **garbage collector.**

Introduction to Java : **Non-primitive types**

Ex:

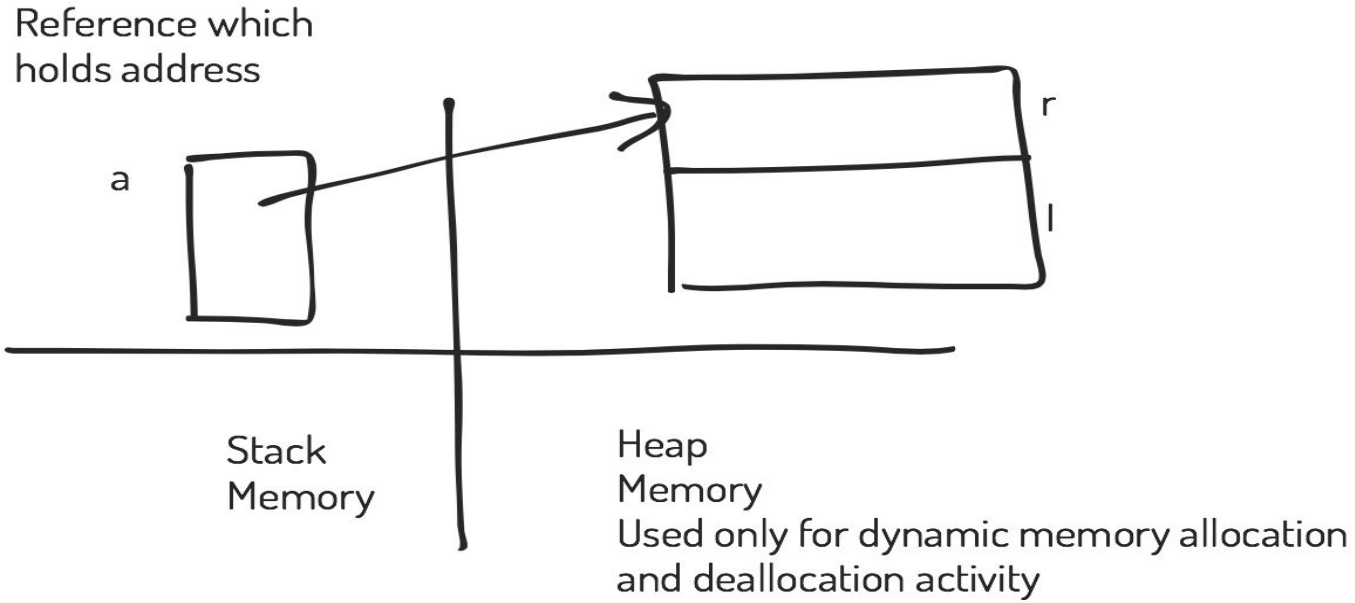
```
class complex {
        private  int r, i;     // no colon after private/public keyword.

        public void display( )
        {
                System.out.println("In display function ");
        }
}  // no ; required

class test {
        public static void main(String args[] ) {
                complex   a;
                a  = new complex( );
                a.display( );
        }
}
```

Introduction to Java : **Non-primitive types**

In the statement "complex a", 'a' is termed as reference.

Reference definition

A reference is **an address that indicates where an object's variables and methods are stored**.



Since, address cannot be accessed explicitly in Java, members of class are referred by only (.) dot operator.

Memory which is allocated to store information of type complex in heap will be reclaimed automatically, when the program is about to conclude its execution.

Glimpse of Inheritance

Inheritance is one of the main pillar in any OOP language, and hence java too.

Inheritance helps one class to inherit the properties of another.

Properties of a class are nothing but DM, MF, SDM, SMF, which can be inherited and used for other purpose.

"extends" is the keyword used to inherit the properties from one class to another.

Introduction to Java : Glimpse of Inheritance

Class, which shares its attributes are termed as base class.

Class, which receives the attributes are termed as derived class.

Ex:

```
class base   {
        public void print()  {
                System.out.println("In base print");
        }
}

class drd extends base {
        public void display() {
                System.out.println("In drd display( )");
        }
}
```

Ex:

```
class Test {
    public static void main(String[] args){
        drd a = new drd( );      a.print();
    }
}
```

Understanding "System.out.println" statement in java

Java **System.out.println()** is used to print an argument that is passed to it.

The statement can be broken into 3 parts which can be understood separately as:

System: It is a final class defined in the java.lang package.

If a class is declared as final it cannot be inherited further

out: This is an instance of **PrintStream** type, **out** is a public and static member field of the System class.

Ex:

```
package java.lang;
class PrintStream
{
  // task of PrintStream is to communicate with standard output device
 public void Println()
 {
   ……...
 }
}
```

```
final class System
   {
       public static PrintStream out;  //has a kind of relation -

      public static InputStream in;

       public System()
       {  out = new PrintStream(); }
   }
```

println(): PrintStream class has a public method **println()**, hence we can invoke the same on "out" as well. This is an upgraded version of print(). It prints any argument passed to it and adds a new line to the output. System.out represents the Standard Output Stream.

COMMAND LINE ARGUMENTS in C

Arguments that are passed to main program

Any type of value passed via CL will be stored as strings/char arrays in memory.

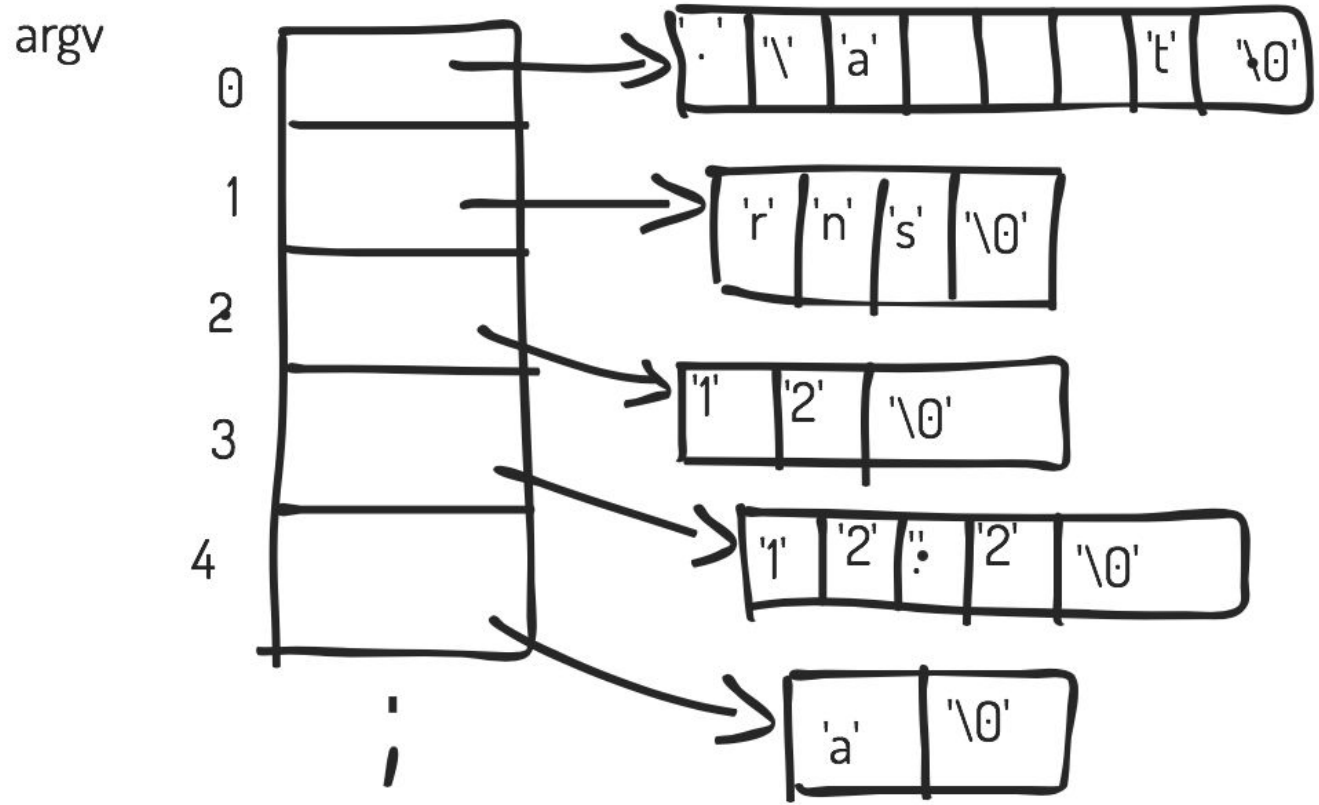
Two arguments are there in C/C++ which represents command line.

int main(int argc, char * argv[ ])

First argument maintains argument count
Second argument maintains information about arguments in string format.

By default argc value will be 1, representing the path of the executable file.
Ex: executable file in unix versions are ./a.out
In windows os executable file will be .exe extension, having the .c file name as the primary one. (.exe is secondary file name)

COMMAND LINE ARGUMENTS in C



COMMAND LINE ARGUMENTS in C

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    int i;
    printf("%d\n",argc);
    for(i=0;i<argc;i++)
      printf("%s\n",argv[i] );
    return 0;
}
```

Introduction to Java : **Built-in class String**

String is a build in class in java to store a string or set of character informations within it.

Since, String is a class a reference must be created and then object must be created using new operator.

The following constructor is associated with the String class in java.

String()

Initializes a newly created String object so that it represents an empty character sequence.

Ex: String a = new String( ); System.out.println(a);

Reference ‘a’ is basically pointing to empty string set.

a=”rnsit” will allocate sufficient dynamic memory to store rnsit and that memory will be referenced by a.

Ex:

String a = new String(“rnsit”);

String class contains a parameterized constructor which receives “rnsit” as a value, memory is allocated to store the passed value, and the passed value is copied to the acquired memory.

Values stored in String instances such as ‘a’ are **immutable.**

a[0] =’J’; cannot be done.

A new set of values can be assigned to ‘a’, but older values cannot be modified in terms of characters.

a[0] = ‘R’; not allowed

a= a + “ engineering”; allowed; here old value is combined with new string

a = “rnsit engineering college” allowed; here old value is replaced by new string.

(Remaining constructors of String class will be considered afterwards).

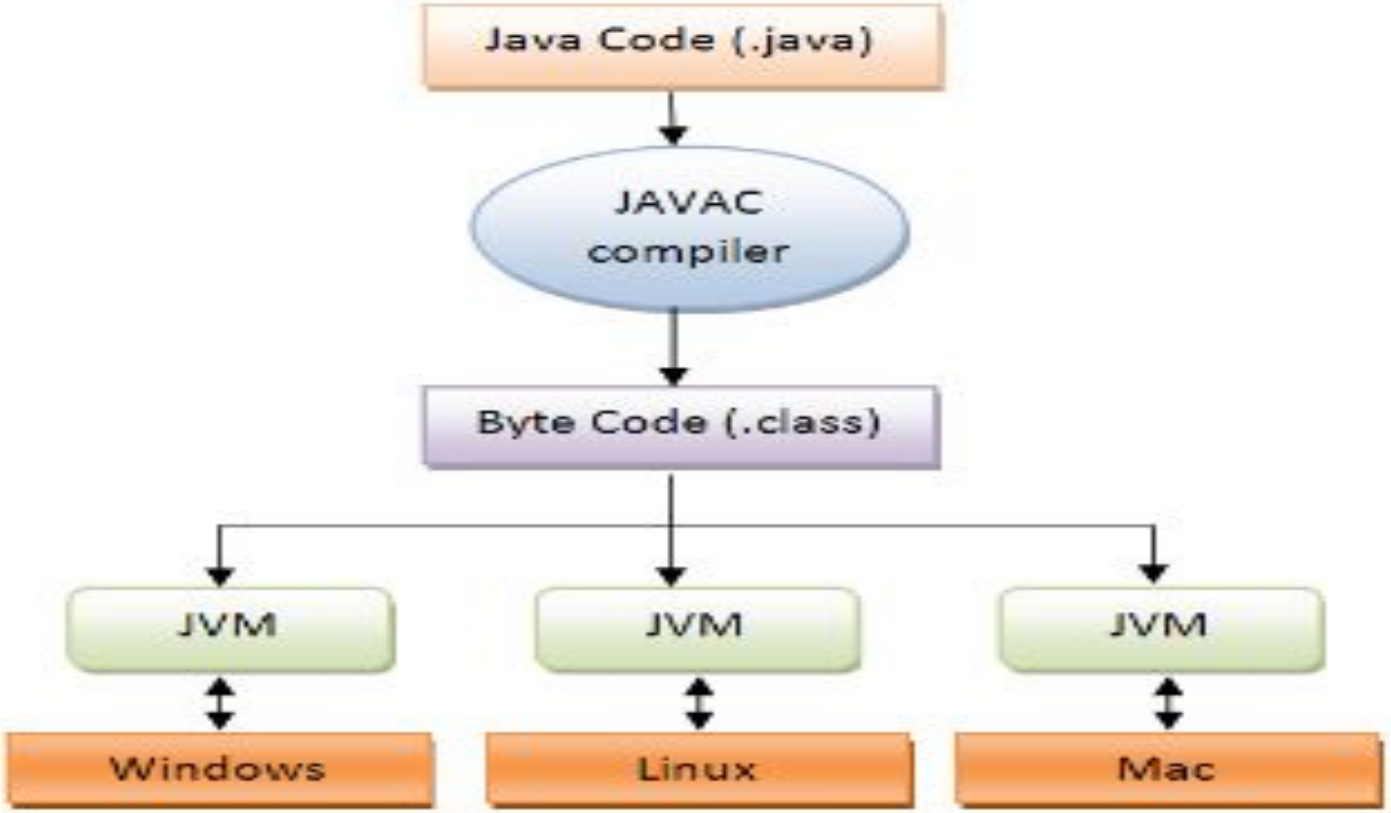
Introduction to Java : **JVM (Java Virtual Machine)**

The concept of Write-once-run-anywhere (known as the Platform independent) is one of the important key features of java language that makes java as the most powerful language.

The programs written on one platform can run on any platform provided the platform must have the JVM(Java Virtual Machine).

A Java virtual machine(JVM) is a virtual machine that can execute Java bytecode. It is the code execution component of the Java software platform.

Introduction to Java : **JVM (Java Virtual Machine)**



Introduction to Java : **Java Program structure**

Java program structure contains six stages.

(1) Documentation section:

The documentation section contains a set of comment lines describing the program.

(2) Package statement:

The first statement allowed in a Java file is a package statement. This statement declares a package name and informs the compiler that the classes defined here belong to a package.

package student; // student is a directory in java

A package is a namespace that organizes a set of related classes.

Package are nothing but different directories in computer.

(3) Import statements:

Import statements instruct the compiler to load the specific class, which belongs to the mentioned package.

import student.test; student is main directory test is subdirector

Package or directory word are synonym in java

All the classes in test subdirectory will be imported or included to the source file, before compilation.

(4) Class definition:

A Java program may contain multiple class definitions.

(5) Main method class:

The main method creates objects of various classes and establishes communication between them. On reaching the end of the main the program terminates and the control goes back to the operating system.

Introduction to Java : **Accepting values from console**

In order to accept values from console built-in class Scanner must be used.

```
Scanner  sc = new Scanner(System.in);
```

After instantiating the object of type Scanner, the constructor of scanner will be called by passing **System.in** as the parameter. **in** is a public static member of type InputStream.

It is as binding the Scanner object with the standard input device, such as keyboard.

```
int n = sc.nextInt();
```

nextInt is a member function which is invoked using sc object.

The above statement accept's a value of type integer from the standard input device i.e keyboard.

Introduction to Java : **Accepting values from console**

Scanner class is widely used to parse/read text for strings and primitive types.

It is the simplest way to get input in Java.

Scanner class is declared in java.util package.

The facilities provided by the System class are the code to communicate with standard input, standard output, and error output streams;

All classes in java whether built-in or user defined, they inherit the properties of Supreme base class **Object**.

Fields of System class are:

1. **public static final InputStream in:**

   This stream is already open and ready to supply input data from standard input devices. Typically this stream corresponds to keyboard input or another input source specified by the host environment or user.

2. **public static final PrintStream out:**

3. **public static final PrintStream err:**

“final“ keyword is used to create constants in java. No constant keyword in java.

Ex:

```
class test   {
    public static void main(String[] args) {
            final int i=90;
            i = 900; // generates CTE
    }   }
```

Introduction to Java : **Arrays in Java**

Syntax for declaring an array variable or reference to array:

dataType[ ] arrayRef; OR dataType arrayRef[ ];

int[ ] myList; OR int myList [ ]; // 1-d array

Further memory will be allocated to store array element type using new operator.

arrayRef = new dataType[arraySize];

The above statement does two things:

It creates an array using new dataType[arraySize];

It assigns the reference of the newly created array to the variable arrayRefVar.

```
int [ ]  myList;
…….
….
  myList = new int[4];
```

All these can be combined into one statement as shown below.

dataType[ ] arrayRefVar = new dataType[arraySize];

```
int [ ]  myList = new int[4];
```

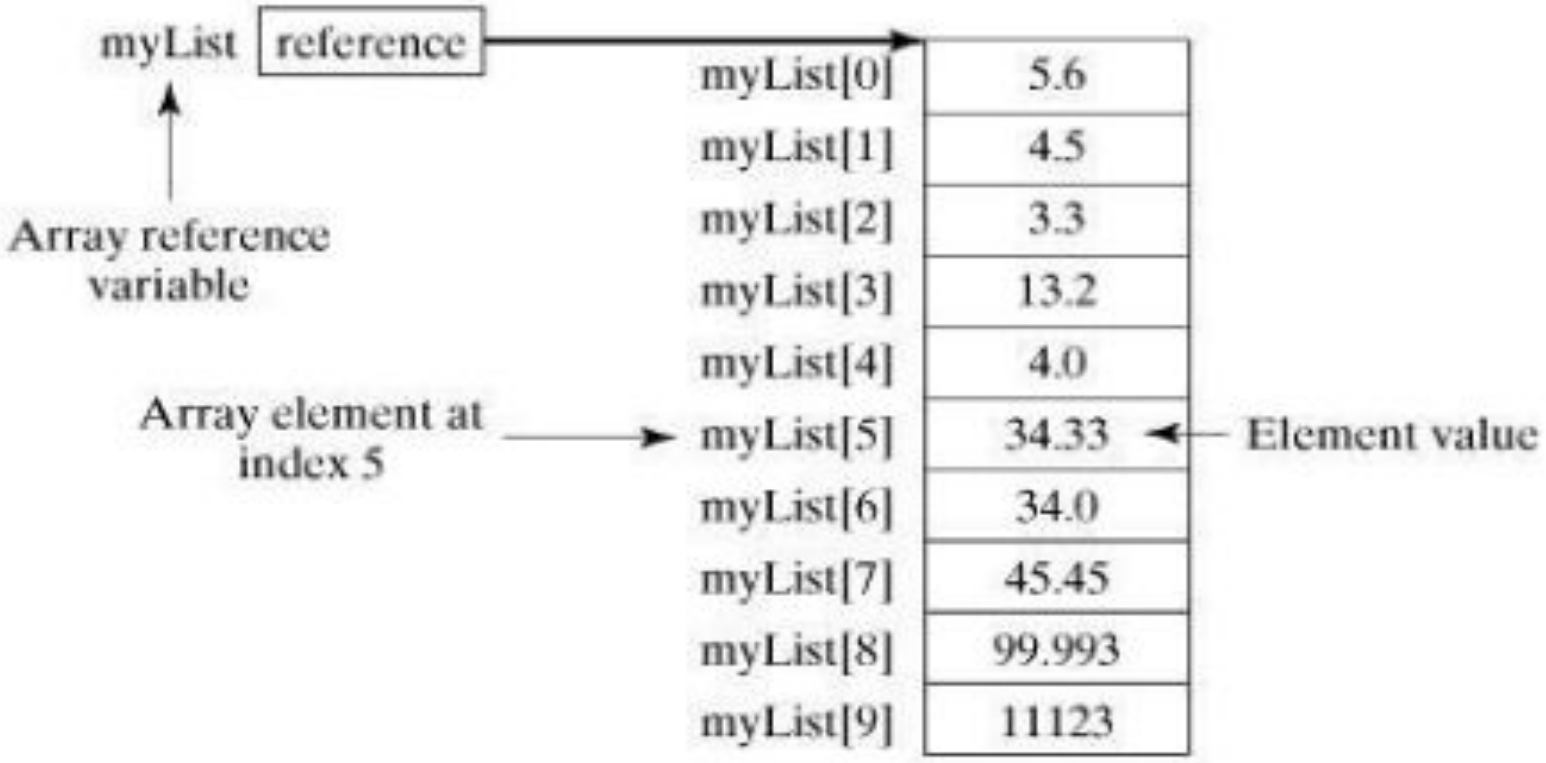
Alternatively you can create arrays and initialize as follows:

dataType[] arrayRefVar = {value0, value1, ..., valuek};

Ex:

```
int [ ] myList = {1, 2, 3, 4};

double[ ] myList = {5.6, 4.5, 3.3, 13.2, 4.0, 34.33, 34.0, 45.45, 99.993, 11123};
```

Initial values are counted and that will be the size of the array.


myList
reference
Array reference variable
Array element at index 5
myList[0]
myList[1]
myList[2]
myList[3]
myList[4]
myList[5]
myList[6]
myList[7]
myList[8]
myList[9]
5.6
4.5
3.3
13.2
4.0
34.33
34.0
45.45
99.993
11123
Element value

```
Ex:   class test  {
            public static void main( String[] argos) {
                 int [ ] a = {1, 2, 3, 4};

                for(int i=0;  i<4; i++)
                      System.out.println(a[i]);

                for(int i=0;   i<a.length;   i++)
                      System.out.println( a[i] );

                a[5]=50; //generates RTE ArrayIndexOutOfBoundsException
           }   }
```

length is the property associated with any type of arrays created in java.

length conveys the number of elements that can be kept in an array.

Array size must be considered in java programs, if not RTE will be generated.

1-D Arrays

Java program to accept n integer values and store it in an array and print it.

Java program to accept n char values and store it in an array and print it.

```
String b;
System.out.println("Enter "+n+" different values");
for(int i=0;i<n;i++)
{
        b= sc.next();
        a[i]=b.charAt(0);
}
```

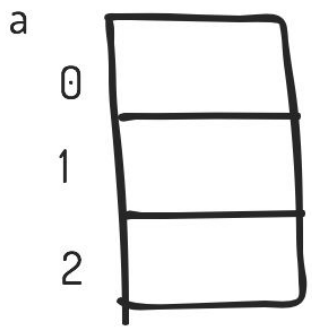
Java program to accept n student values store it in array and print it.

Java program to accept n student values store it in array and print it.

```
import java.util.*
class student  {
        private String name,usn;
        private static Scanner sc;
        public static void create_sc()
        {  sc = new Scanner(System.in); }

        public void accept()
        {
                System.out.println("Enter name ");
                name=sc.nextLine();
                System.out.println("Enter usn ");
                usn=sc.next();
        }
        public void display()
        {  System.out.println(name+" "+usn);  }        }
```

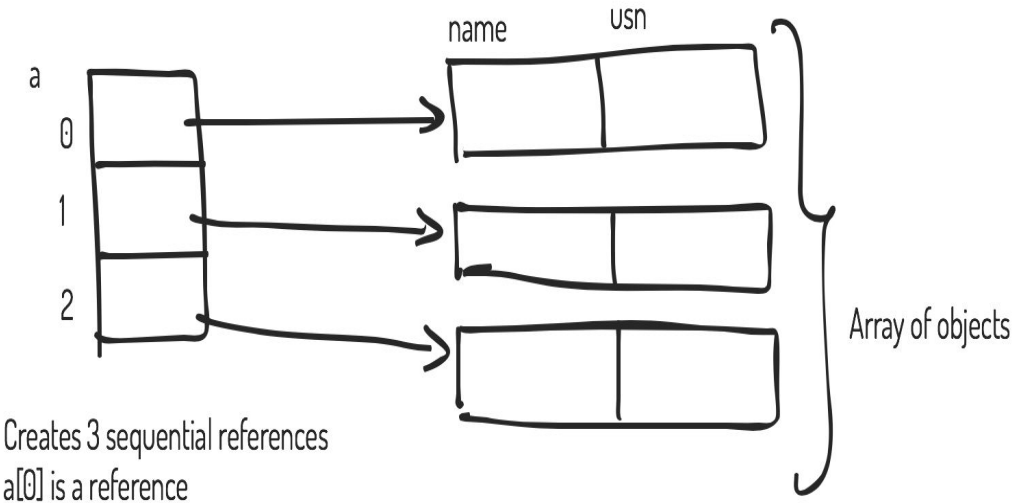
```
public class First {
    public static void main(String[] args) {
        student.create_sc();
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        student [] a = new student[n];   //  1

        for(int i=0;i<a.length;i++)
            a[i] = new student();        //  2

        System.out.println("Enter values for array ");
        for(int i=0;i<a.length;i++)
            a[i].accept();

        System.out.println("Array values are ");
        for(int i=0;i<a.length;i++)
            a[i].display();                                  }          }
```

student [ ]. a. =. new student [3];



Creates 3 sequential references
a[0] is a reference
a[1] is a reference
and a[2] is also a reference

student [ ]. a. =. new student [3];



Creates 3 sequential references
a[0] is a reference
a[1] is a reference
and a[2] is also a reference

2-D Arrays

Java program to accept n integer values store it in an 2d-array and print it.

Java program to accept n char values store it in an 2d-array and print it.

Java program to accept n student values store it in 2d-array and print it.

Introduction to Java : **Arrays in Java - 2-D Arrays**

2-D arrays are a collection of 1-d arrays in java.

GF: <data-type> [][] <2d-array-name> = new <data-type> [row-size][column-size];

Ex: int [ ][ ] a = new int [3][2];

char [ ][ ] b = new char[3][2];

Numerical 2-d array

```
Scanner sc = new Scanner(System.in);
int m,n,i,j;
m=sc.nextInt();
n=sc.nextInt();

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

for(i=0;i<m;i++)
        for(j=0;j<n;j++)
                a[i][j] = sc.nextInt();

for(i=0;i<m;i++)
{
        for(j=0;j<n;j++)
                System.out.print(a[i][j]+"\t");
        System.out.println();
}
```

Character 2-d array

```
Scanner sc = new Scanner(System.in);
int m,n,i,j;
m=sc.nextInt();
n=sc.nextInt();

char [][] a = new char[m][n];

for(i=0;i<m;i++)
        for(j=0;j<n;j++)
                a[i][j] = sc.next().charAt(0);

for(i=0;i<m;i++)
{
        for(j=0;j<n;j++)
                System.out.print(a[i][j]+"\t");
        System.out.println();
}
```

Student 2-d array

```
package first;
import java.util.*;
class student      {
        private String name,usn;
        private static Scanner sc;
        public static void create_sc()
        {       sc = new Scanner(System.in);      }
        public void accept()   {
                System.out.println("Enter name ");
                name=sc.next();
                System.out.println("Enter usn ");
                usn=sc.next();
        }
        public void display()  {
                System.out.print("[ "+name+","+usn+" ]");   }
}
```

Student 2-d array

```
public class First {

    public static void main(String[] args) {
        student.create_sc();
        Scanner sc = new Scanner(System.in);
        int m,n,i,j;
        m = sc.nextInt();          n = sc.nextInt();

        student [][] a = new student [m][n];

        for(i=0;i<a.length;i++)
          for(j=0;j<a[i].length;j++)
              a[i][j] = new student();
```

Student 2-d array

```
            System.out.println("Enter values for array ");
            for(i=0;i<a.length;i++)
                  for(j=0;j<a[i].length;j++)
                        a[i][j].accept();

            System.out.println("Array values are ");
            for(i=0;i<a.length;i++)   {
                  for(j=0;j<a[i].length;j++)
                        {   a[i][j].display();    System.out.print(" ");   }
                  System.out.println();
            }
      }
}
```

Introduction to Java : **for-each loop in Java**

for-each enables to traverse the complete array sequentially without using an index variable.

GF:

```
for( type  var : array)
{
      Statements using var;
}
```

var just receives the value of array from 0th index to length-1 index in each iteration of the foreach loop.

value in var can be used by the statements inside foreach loop.

If ‘var’ is modified array contents will not be modified.

Introduction to Java : **for-each loop in Java**

Ex:

```
int [] a = {11,22,33,44};

for(int   i  :  a)
        System.out.println(i);
```

Limitations of foreach loop

1. For-each loops are **not appropriate when array contents are to be modified.**

2. For-each loops do not keep track of index. Hence, array index cannot be obtained using For-Each loop.

3. For-each only iterates forward over the array in single steps.

4. For-each **cannot process two decision making statements** at once

Introduction to Java : Printing array of strings using foreach

Ex:

```
String [] a = {"rnsit", "msrit", "rvce"};

for(String i : a)
        System.out.println(i);
```

Ex:

```
int [][]  a = new int[3][2];
int i,j,k=1;

for(i=0;i<a.length;i++)
        for(j=0;j<a[i].length;j++)
                a[i][j] = k++;


for(int [] p : a)
        for (int q : p )
                System.out.println(q);
```

Introduction to Java : Type casting in Java

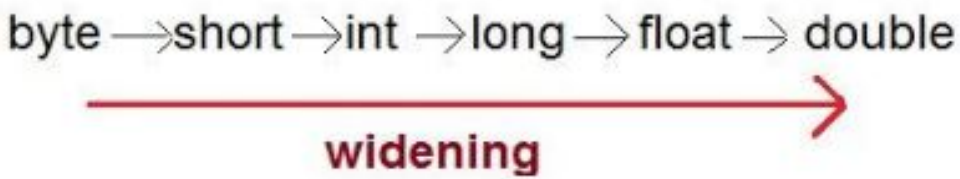
It is often necessary to store a value of one type into the variable of another type.

Type casting can be done in two ways.

Ex:

```
int x = 10;
byte y = (byte)x;
```

In Java, type casting is classified into two types,Widening Casting(Implicit)



Introduction to Java : Type casting in Java

Narrowing Casting(Explicitly done)

Widening or Automatic type conversion

Automatic Type casting take place when, the two types are compatible or the target type is larger than the source type

Ex:

```
class Test  {
  public static void main(String[] args)   {
     int i = 100;
     long l = i; //no explicit type casting required
     float f=l; //no explicit type casting required
     System.out.println("Int value "+i);
    System.out.println("Long value "+l);
    System.out.println("Float value "+f);            }  }
```

Narrowing or Explicit type conversion

When assigning a larger type value to a variable of smaller type, then there is a need to perform explicit type casting.

Ex:

```
class Test {
   public static void main(String[] args) {
     double d = 100.04;
     long l = (long)d; //explicit type casting required
     int i = (int)l;//explicit type casting required
     System.out.println("Double value "+d);
     System.out.println("Long value "+l);
     System.out.println("Int value "+i);
   }
}
```

Java operators:

Java operators can be categorized into following ways:

(1) Arithmetic operator
(2) Relational operator
(3) Logical operator
(4) Assignment operator
(5) Increment and decrement operator
(6) Conditional operator
(7) Bitwise operator
(8) Special operator.

Arithmetic operator: The Java arithmetic operators are:

+	: Addition
-	: Subtraction
•	: Multiplication
/	: Division
%	: Remainder

Relational Operator:

<	: Is less then
<=	: Is less then or equals to
➢	: Is greater then
>=	: Is grater then or equals to
==	: Is equals to
!=	: Is not equal to

Logical Operators:

&&	: Logical AND
||	: Logical OR

Introduction to Java : Operators in Java

!	: Logical NOT

Assignment Operator:

+=	: Plus and assign to
-=	: Minus and assign to
*=	: Multiply and assign to.
/=	: Divide and assign to.
%=	: Mod and assign to.
=	: Simple assign to.

Increment and decrement operator:

++	: Increment by One {Pre/Post)
--	: Decrement by one (pre/post)

Conditional Operator: Conditional operator is also known as ternary operator. The conditional operator is :

Exp1 ? exp2 : exp3

Bitwise Operator: Bit wise operator manipulates the data at Bit level. These operators are used for tasting the bits. The bit wise operators are:

&	: Bitwise AND
!	: Bitwise OR
^	: Bitwise exclusive OR
~	: One's Complement.
<<	: Shift left.
>>	: Shift Right.
>>>	: Shift right with zero fill

Example:

Introduction to Java : Operators in Java

Bitwise operator works on bits and performs bit-by-bit operation. **Assume if a = 60; and b = 13;** now in binary format they will be as follows:

```
a = 0011 1100
b = 0000 1101
-----------------
a&b = 0000 1100
a|b = 0011 1101
a^b = 0011 0001
~a  = 1100 0011
```

<<	**Binary Left Shift Operator.** The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>>	**Binary Right Shift Operator**. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111
>>>	**Shift right zero fill Operator**. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

Special Operator - Instanceof operator

The **instanceof** is an operator that returns true if the object on the left hand side is an instance of the class given in the right hand side.

This operator allows us to determine whether the object belongs to the particular class or not.

Ex: class comp { }

```
comp a = new comp( );
if  ( a  instanceof  comp)
      System.out.println("a is an instanceof comp");

if  (a  instanceof  Object)
      System.out.println("a is an instanceof Object too");
```

Dot operator in Java

The dot(.) operator is used to access the instance members (using objects) or class members of a class.

length Vs length()

int [] a = new int[3];

array.length :

length is a final variable applicable for arrays. With the help of a length variable, size of the array can be obtained.

String a=”rnsit”;

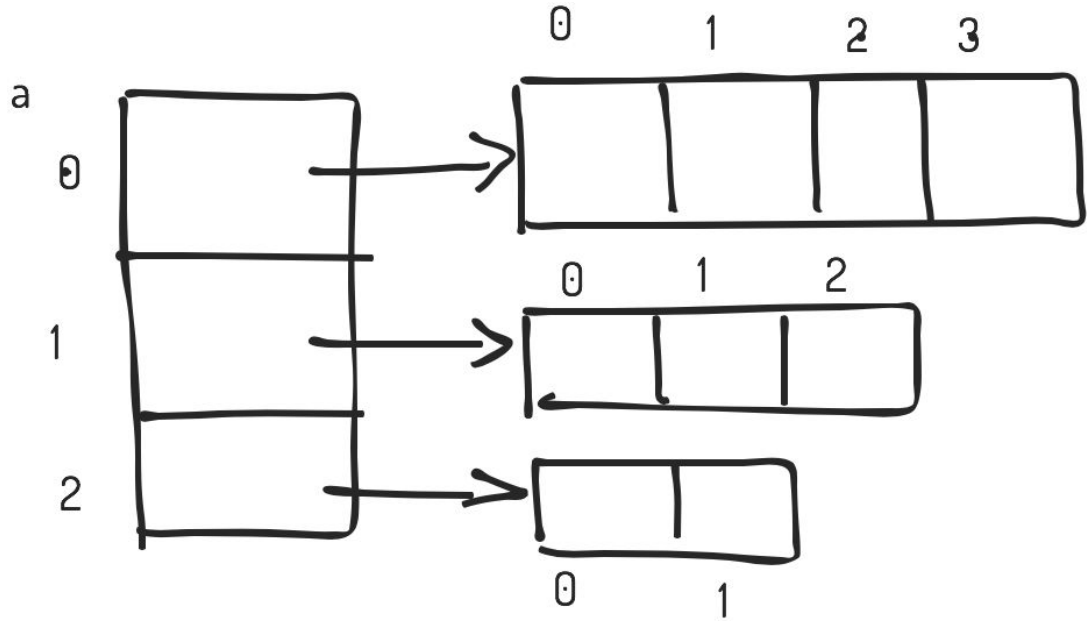
a.length() :

length() method is a method which is applicable for string objects. length() method returns the number of characters present in the string.

Introduction to Java : Skewed Array

A standard 2-d array stores equal amount of information in each row.

A skewed array is an array of arrays, where in each row will store different number of informations within them.



Introduction to Java : Skewed Array

Ex:

```
int [][] a  =  new int[3][];

a[0] = new int[4];
a[1] = new int[3];
a[2] = new int[2];

for(int i=0;i<a.length;i++)
        System.out.println(a[i].length);

Scanner sc = new Scanner(System.in);
for(int i=0;i<a.length;i++)
        for(int j=0;j<a[i].length;j++)
                a[i][j] = sc.nextInt();
```

Introduction to Java : Overriding in Java

In an object-oriented programming language,

Overriding is a feature that allows a derived class to provide a specific implementation of a method that is inherited by its superclasses.

When a method in a subclass has the same name, same formal parameter type, and same return type(or sub-type) as a method in its super-class,

then the method in the subclass is said to *override* the method in the super-class.

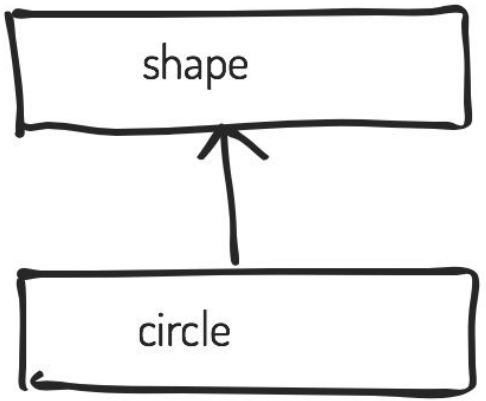
Introduction to Java : Overriding in Java

Ex:

```
class shape  {
        public void area()
        {  System.out.println("Nothing to calculate"); }
}
class circle extends shape  {
        public void area( )   //overridden function
        {  System.out.println("Area of a circle");  }
}
```

Signature of method area( ) in both shape and circle classes are same.

Method area( ) in circle is considered as **overridden** function.



Introduction to Java : Overriding in Java

In the first step of inheritance, member functions of base class can be inherited and in-turn used in the derived class.

In the second step, member functions of base class that are inherited can be restructured to solve a different task, pertaining to the needs of the derived class.

In order to do this, the member functions which is inherited from base class, will be redefined with the same signature in the scope of derived class.

This task is termed as **function overriding**.

The task of member function in base class is termed as **generalized** task
and the member function overridden in the derived class is termed as **specialized** task.

Depending on the logical necessity in derived class, the functions that are inherited might be or might not be overridden.

Introduction to Java : Overriding in Java

Two types of polymorphism exists in OOP

Compile time polymorphism (function overloading)

Runtime polymorphism (function overriding)

Compile time polymorphism means, compiler can decide which member function to call during compilation time itself.

Ex: comp a = new comp( ); a.accept(); // an example of CTP

Class comp is not inheriting any base class properties and a is not **generic reference/base class** reference, hence compiler knows which member function to call.

Runtime polymorphism means, compiler cannot decide, which member function to call until runtime, in other words.

The decision of a member function to call will be staggered until execution time.

Introduction to Java : Overriding in Java

Ex:
```
class base{
        public void disp( ) { System.out.println("in base"); }
}
class drd  extends base {
        public void disp( ) { System.out.println("in drd"); }
}

drd a = new drd( );       a.disp( );
```

drd is inheriting properties from class base, and 'a' is not a base class reference.
Hence, a.disp( ) accounts for CTP.

```
base a = new base( );
```

'a' is a base class reference, and 'a' can point to an object of type base or 'a' can point to an object of type drd.

a.disp( ); accounts for RTP.

a = new drd( ); //approved by compiler

'a' is a reference of type base and can point to an object of type drd.

a.disp( ); accounts for RTP.

Steps required to experience RTP

1) Inheritance 2) Function overriding 3) base class reference

4) overridden function and base class function must be called, using base class reference.

In other words Generalization and Specialization must be invoked

.

Introduction to Java : Overriding in Java

Use of Method overriding : is to achieve **specialized** functions and to also can achieve **Runtime polymorphism.**

If function overriding is done in class hierarchy, then 2 types of functions can be used, one is generalized property and other one is specialized property.

OOP language provides the facility of using one base class reference to access these multiple functionalities in the inheritance hierarchy, termed as **base-class/generic reference**.

Generic reference means it can point to an object of itself or the object of a class derived from it.

Only in inheritance hierarchy these generic references exist.

Main usage of generic references is **“one interface multiple invocation”.**

Introduction to Java : Overriding in Java

```
Ex:     class circle extends shape  {
                    public void area()   //overridden function
                    {  System.out.println("Area of a circle");  }

                    public  void display( )
                    {  System.out.println("in display function");  }
                }

          shape  p = new circle( );
          p.display( );  // generates CTE
```

Using base class reference only the properties passed on to derived class can be accessed, since display( ) is a member function of class circle, it cannot be invoked using ‘p’.

Introduction to Java : Overriding in Java

Rules for overriding a method

1. final methods of base cannot be overridden in derived class.

2. Static methods of base cannot be overridden, if a static method of the same signature is defined in derived class, the super class static method is hidden, termed as **method hiding**.

	Superclass Instance Method	Superclass Static Method
Subclass Instance Method	Overrides	Generates a compile-time error
Subclass Static Method	Generates a compile-time error	Hides

3. Private methods of a base class cannot be overridden

4. Signature of overridden method in derived class must match the signature of the method in base class.

5. The method present in base class can be called from the overridden method using the 'super.method-name( )' keyword.

Ex:
```
class shape  {
        public void area()
        {  System.out.println("Nothing to calculate"); }        }

    class circle extends shape  {
        public  void area()   //overridden function
        {  super.area();   System.out.println("Area of a circle");  }
```

Introduction to Java : Reading values from a file in Java

Ex:

```
import java.io.File;
Scanner sc=null;
try {
  sc = new Scanner(new File("/Users/hemanth/eclipse-workspace/first/src/first/No"));
   }
   catch(Exception e)
   {}

   int n=sc.nextInt();
   for(int i=0;i<n;i++)
         System.out.println(sc.nextInt());
```

File: No

4 1 2 3 4

Introduction to Java : Wrapper classes in java

A **Wrapper class** is a **class** which contains the primitive data types (int, char, short, byte, etc). In other words, **wrapper classes** provide a way to use primitive data types (int, char, short, byte, etc) as objects. These **wrapper classes** come under **java.util** package.

Integer is a wrapper class to store int type of value in java
Float is a wrapper class to store float type of value in java

Ex: **Integer a = 10;** // a is an object of built-in wrapper class Integer and its value is 10.
Float b = 20; // object b is of type built-in wrapper class Float and its value is 20.

There are several members functions associated with each wrapper classes which can be used for logical purpose in the program.

All primitive types as a wrapper class in java to store primitive values in class type.

Introduction to Java : Command Line Arguments in Java

Ex: //Run menu - Run Configurations

```
for(String i : args)
        System.out.println(i);
```

Values passed from CLA will be stored from 0th index itself.

Ex: consider CLA passed 12 12.2 R RNSIT

```
int i = Integer.parseInt(args[0]);
float j = Float.parseFloat(args[1]);
char a = args[2].charAt(0);
String b = args[3];

System.out.println(i);     System.out.println(j);
System.out.println(a);  System.out.println(b);
```