Unit-1 Introduction to Java

- Java is a general purpose object oriented programming language developed by Sunmicrosystem in 1991.
- Originally it is called as OAK, by James Gosling. One of the invertors of the language.
- Java was designed for development of software for consumer electronic devices like TV, toasters and such other electronic machines.

✱ Java Revolution:—

1990 :- Sun microsystem decided to develop ~~to~~ special software that could be used a maniculate consumer electronic devices a team of sun microsystem programmers controlled by James Gosling.

1991 :- After exploring the possibility of using the most popular object oriented language C++, the team announced a new language named as OAK.

1992 :- The team known as green Project team by Sun microsystem demonstrated the application of their new language to control a listop at home applines or home devices by using a hand held device by tiny touch sensitive screen.

1993 :- The world wide web appeared as on the internet and transformed the text based internet into a graphical bridge environment.

1994:- The team developed a web browser called as the hot Java. the locate and run applet programmer on internet..

1995:- OAK was renamed as Java due to some legal region.

1996:- Java established itself not only as a reader for internet programming but also a general purpose object oriented programming language Sun microsystem releases version as Java development Kit 1.0 (JDK).

1997:- JDK 1.1.

1998:- Sun microsystem releases 1.2 which run as a part of software development Kit SDK 1.2.

1999:- Java releases Java 2 platform Standard Edition (J_2SE) and enter Price edition (J_2EE)

2000:- J2SE with SDK 1.3 was released.

2002:- SDK 1.4.

2004:- JDK (5.0) also known as J2SE 5.0.

2006:- J2SE 6.0

2011:- Java Standard edition 7 was released.

Java Features

- Java features are explained in the following parts such as
  1. Compiled & interpreted
  2. Platform independent & Portable.
  3. Object Oriented
  4. Robust & secure.
  5. Distributed
  6. Familiar, simple & small.
  7. Multi threaded interact
  8. High Performance
  9. Dynamic and Extensible

1) Compiled & interpreted :-

- A computer language is either compiled & interpreted.
- Java combines both approaches by making Java a 2 stage system.
- First, Java compiled translate source code into the byte code instruction (.class file).
- Byte code are not machine instructions and therefore in the 2nd stage the Java interpreted generator machine code that can be directly executed by machine i.e, running the Java Programming.

2) Platform independent & Portable :- / Architecture Neutral

- Java programs could be easily moved from one computer system to another anywhere & anytime.
- Changes & upgrades in operating system process and system resources will not force any changes in Java program.

- This is the reason why Java has become a popular language for programming internet which interconnects different kind.
- Java ensure ~~protability~~ portability in 2 ways such as
  (i) Java compiler generates byte code instruction that can be implemented on any machine.
  (ii) The size of primitive data type are machine independent.

Note:-
Some times it is also known as WORA (write once run any where). ~~Thease~~ These features is possible due to presents of platform independent & portable.

3) Object Oriented:-
- Java is true object oriented programming language almost everything in Java is an object.
- All program code and data reside within object and classes. the object model in Java is simple and easy to extend.

4) Robust & Secure:-
- Java is Robust/powerful language. It provides many safeguards to ensure reliable code.
- It has strict compiled time and run time checking for data types.
- It is design as a garbage collected language, which is reliving the programmers vertually memory management progress.
- Security becomes an important issue for a language i.e, used for programming an internet.

- Threat of viruses and abuse of resource are everywhere.
- The absence of pointers in Java ensures that the programs can't gain the specific memory location without proper authorization

5) Distributed:-

- Java is design as a distributed language for creating application network
- It has the ability to share both data & program.
- Java applications can open access remote objects or internet as easily as they can access in a local system. due to that Java application enables multiple program & multiple remote location to Colaborate and work together on a single project.

6) Familiar, simple & small:-

- Java is a simple & small language
- Many features of C & C++, that are repeated are souzes of an releable code are ~~too~~ not part of Java.

  Ex:- Java doesn't use pointers, pre-processor header files, go to statement & many others.
- It also eliminate operator overloading & inheritance
- Familiarity is another important features of Java to make the language look familar due to the existing programming concept of C & C++ language.

7) Multithreaded Interact:-

- Multithreaded means handling multiple task simultaneously.

- Java supports multithreaded programs this means that we did not wait for the application to the finished one task before beginning another.

  Ex:- Audio clip, while scrolling a page and at the some time downloads some application form other computer or internet due to this features it improves the interaction performance of graphical.

- Java runtime terms with tools that allow multi-process synthesation & designing as well as running interacting system.

8) High Performance

- Java performance is impressive for an interpreted language which is mainly due to use of intermediate byte code.
- Java speed is compairable to the C/C++.
- Java ~~ara~~ architecture is also design to reduce over heads during runtime.

9) Dynamic & extensible:-

- Java is a dynamic language.
- Java is capable of dynamical linking in new class libraries method & objects.
- Java can also determine the type of class through a coding by making it possible to either dynamically on abroad the program by defining on the response.

- Java program support function written in such as c/c++ this function are known as <u>native methods</u>.

Java Environment.

- Java Environment includes a large no. of development tools and hundreds of classes and method.
- The development tools are the part of the system known as JDK (Java development kit) and classes and method are part of the Java standard library (JSL) also known as application programming interface.
- This environment is divided into 2 parts i.e, JDK and application programming interface or (JSL).

JDK (Java development kit):-

- It comes with a collection of tools that are used for developing and running Java programs.
- This tools are explain below.

(i) applet viewer.
(ii) Java (Java interpreter)
(iii) Java c (Java compiler)
(iv) Java p (Java disassembler)
(v) Java h (supporting for c or c++ headerfile)
(vi) Jdb (Java debugger)
(vii) Java doc (for creating html document)

(1) Applet viewer:-

- It enables us to run Java applet with a specific web browser is hot Java / other web browser.
- It specially used for executing graphical base programming application.

(2) Java (Java interpreter):-

- It stands for Java interpreter which translate byte code files that the ~~to the~~ machine understandable that the interpreter can understand.

(3) Java c (Java compiler):-

- It stands for Java compiler which translate ~~enables~~ Java source code to byte code file i-e, the interpreter can understand.

(4) Java p (Java dis assembler):-

- It stands for Java dis assembler which translate enables to convert byte code files into program description.

(5) Java h (for supporting c or c++ headerfile):-

- It produces headerfiles for used with native methods.

(6) Java Jdb (Java debugger):-

- It stands for Java debugger which helps of to find errors in our program.

(7) Java doc (creating html document):-

- It creates html file into a program description on Java format documentation is relatable html format.

(II) Application Programming interface (API):-

- Java standard library includes a hundred classes are method grouped into several functional packages.
- Most commonly ~~packa~~ used packages.

(1) Language supports package:-

- A collection of classes & methods required for implementing basic features of Java.
- Its stands for lang package. (Math function power)
- It can be written in the program like

Keyword - (import Java.lang.*;)

(2) Utilities Package:-

- A collection of classes to provide utilities function such as data & time function; (connecting one a poly screen)
- It can be denoted as util package.

Keyword:- import Java.util.*;

(3) input output Package:-

- A collection of classes required for input output manipulation.
- It can be denoted as io package

Keyword:- import Java.io.*;

(4) Networking Package:-

- A collection of classes for communicate with other computer via internet.
- It can be denoted net package.
- It can be written as

Keyword:- import Java.net.*;

(5) Abstract Window tool Kit Package (AWT Package)

- It contain classes i.e implements platform independent graphical user interface.
- It can be denoted as ~~AW~~ awt package.

Keyword:- import. Java. awt. *;

(6) Applet Package:-

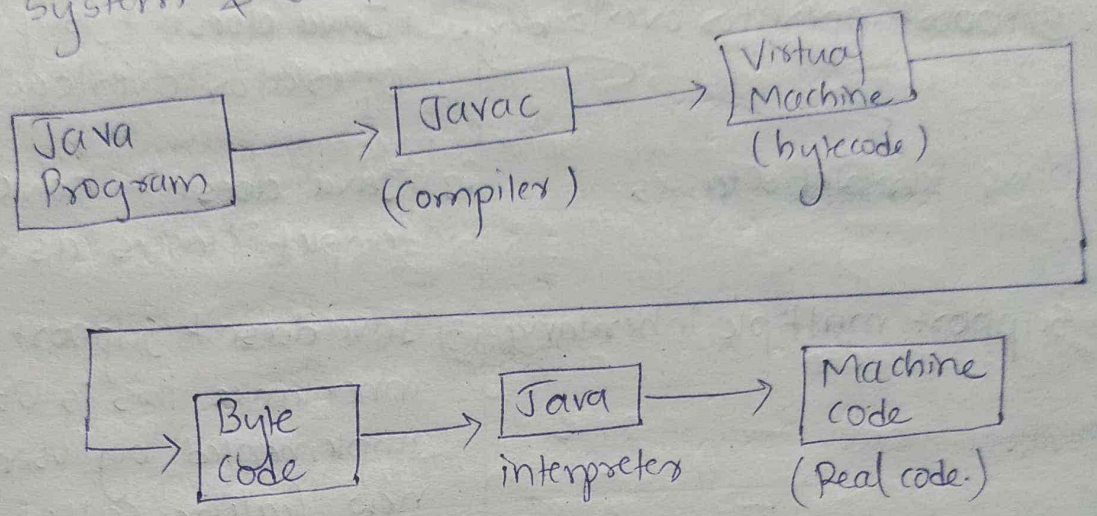
- This includes a set of classes that allows us to create Java applets.
- It can be denoted as applet.
- It can be written in the program like

Keyword:- import. Java. applet. *;

* Java Virtual Machine (JVM):-

- All language compilers translate source code to machine code for a specific computer.
- Java compiler also does the same thing here Java language achieve architecture nutrality or platform independent by the concept of Java compiler which produces an intermediate code known as bytecode for a machine that does not exist.
- This machine is called as Java virtual machine & it exists only inside the computer memory.
- It is a simulated computer within the computer and does all measure function of a real computer.
- The virtual machine code are not a machine specific. The machine specific generated by the Java interpreter by acting as an intermediate between virtual machine and real machine.

- The Java object frame one acts as an intermediate between the user program and virtual machine which it term act as the intermediate between Operating system & Java object frame one.



Difference between 'C' and 'Java' :-

C	Java
(1) 'C' includes keyword statement such as sizeof and typedef	(1) Java doesnot includes 'C' unique statement keyword such as sizeof and typedef.
(2) 'C' includes this 2 datatypes struct & union.	(2) Java doesnot include datatype struct & union.
(3) 'C' define this modifier keywords like auto, extern, register, signed & unsigned.	(3) Java doesn't define type modifier keywords like auto, extern, register, signed and unsigned.
(4) 'C' support an explicit pointer types.	(4) Java doesn't support an explicit pointer types.
(5) 'C' have pre-processor and it can use pre-defined headerfiles such as #include, #define, & #indef.	(5) Java doesnot have pre-processor and can't use some pre-defined headerfiles such as #include, #define, & #ifdef.
(6) 'C' also add the level structure like break and continue statement.	(6) Java also add the level structure like break & continue statement.

Difference between C++ & Java :-

C++	Java
(1) C++ support operator overloading.	(1) Java doesn't support operator overloading.
(2) C++ have templet classes.	(2) Java doesn't have templet classes as in C++.
(3) C++ support multiple inheritance.	(3) Java doesn't support multiple inheritance this is used or implemented by using a new feature called interface.
(4) C++ support global variable	(4) Java doesn't support global variable. Every variable and methods are declares within a classes.
(5) C++ support headerfile concept.	(5) Java doesn't support headerfile concept.
(6)	(6) Java has replaced the destructor function with a finalize().
(7) ~~Sup~~ C++ support pointer concept	(7) Java doesn't support pointer

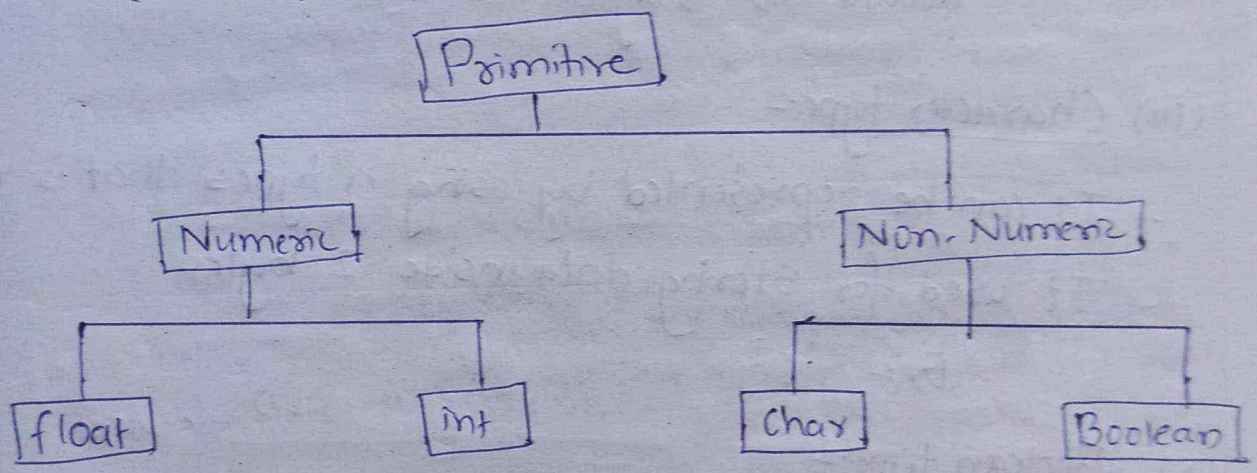
Datatypes :-

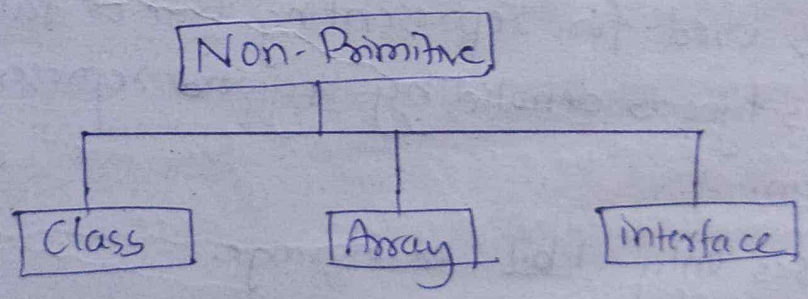
- Every variable in Java as a datatypes.
- Datatypes specific the size of types of value that can be store it is divided into 2 types.
- Datatypes specify the different sizes and values that can be stored in the variable

(1) Primitive (Primary datatype)
(2) Non-Primitive (Derive datatype)

* Primitive types divided into 2 types
  (1) Numeric (float, int)
  (2) Non-Numeric (char, Boolean).

* Non-Primitive divided into 3 types
  (1) Class
  (2) Array
  (3) Interface





(1) <u>Integer types</u>:-
- Integer types can hold whole numbers such as 123, -123, -10, 5 it is further divided into following ways such as

byte → 1 byte
short → 2 byte
int → 4 byte
long → 8 byte.

(ii) Floating types:-

- The floating point can be represented by using to fractional part with decimal parts. this type Variable are single precision no. on double precision no.

Ex:- 12.567 12.536

-5.74 12.536×10^2

- It is divided into 2 ways such as

float → 4 byte

double → 8 byte

(iii) Character type:-

- It can be represented by using a types that is char.
- It used for storing data inside 2 bytes.

Ex:- 'c'

(iv) Boolean type:-

- It is used for representing true or false value. So this type is denoted by keyword represented by boolean.
- It uses only 1 bit of storage

Declaration of Variable

- Variables are name of storage locations. which can be design suitable variable names we must declared them to the compiler.
- It is declared in 3 condition such as.

(i) It tells the compiler what the variable name is
(ii) Its specifies what type of data the variable will hold.
(iii) The case of declaration decides the scope of variable.

Imp

Eg:- int a → declaration
(i) a = 10; → assignment
(ii) int a = 10; → initialization.

Scope of Variables:-

- Java variables are actually classified into 3 types variable.
  1. Instance variable
  2. Class variable
  3. local variable
- Instance & class variable declared inside a class.
- Instance variable are created when the objects are instanced or object are created.
- class variables are global of a class & belong to the entire set of objects that class create. Only one memory location is created for class variable
- Variables declared are used inside methods are called as local variables.
- They are also called because of using with in a method. Local variable can also be declared inside program blocks that define between opening classes & closing classes.

A. Creating Simple Java Program:-

- It is generally created in the following ways such as

(I) class declaration:-

- The fast line should be declared a class and its name which is an object oriented concepts.

  Ex:- class demo.

(II) Opening braces:-

- Every class definition in Java beginning with an opening braces & end with a matching closing braces.

(III) Main line:-

- The third line which defines a method as main().
- It can be written with a program like

  public static void main(String args[])

- This lines contains a no. of keywords such as

  (I) Public:- The keyword public is an access specifier that declares the main method as unprotected and making it accessible to all other classes.

  (II) Static Keyword:-

  - This declared this method as one that belongs to the entire class and not a part of any object of the class.
  - The main must be declared as static. Since the intrepreter uses this method before any object are created.

Void:-

- Void is a type modifier.
- It states that the main method does not returns any value.
- The main method contains a parameter. i.e, (String args[]). Here the argument args is a string array type or and array of objects of the class type string.
- (String is a predefined class) → (fast char always capital)

Output line:-

- The output line in this line we represent a statement System.out.println(); or System.out.print();.

Steps for Run the Program

1. First search the Notepad and write the program code
2. After writing the program code, then we have to save the files as classname.java.
3. Then search 'cmd' and enter it
4) Then we have to choose the drive, where the folder has been stored. The code was to open the drive [d: ↵]
5) Then we have to type ~~cd cda~~ cd foldersname. ↵
6) we have to type javac classname.java.
7) Then we have to type java classnam.
8) Then as continue....

Ex:- W.A.P to display your name by using Java.

```
-> import Java.io.*;
   class Demo
     {
     public static void main(String args[])
       {
         System.out.println("Computerscience");
       }
     }
```

O/P

Computer Science

Operator

- It is a special symbols, when it ~~be~~ to be embedded with any data value or operand, so it is compulsory perform a special tasks and provides a desired objective.

- There are many types of operators in Java.
  (1) Unary Operator
  (2) Arithmatic Operator
  (3) Shift operator
  (4) Relational Operator.
  (5) Assignment Operator.
  (6) Terenary Operator.
  (7) Logical Operator
  (8) Bitwise Operator

(1) Unary Operator:-

- The java unary operator require only one operand. Unary operators are used to perform various operations i.e :

- incrementing/ decrementing a value by one.
- negating an expression.
- inverting the value of a boolean.

Eg:-

```
import java.io.*;
class operator
{
    public static void main(String args[])
    {
        int x=10;
        System.out.println(x++);
        System.out.println(++x);
        System.out.println(x--);
        System.out.println(--x);
    }
}
```

O/P

```
10
12
12
10
```

(2) Arithmetic Operator:-

- Java arithmatic operators are used to perform addition, subtraction, multiplication and division. They act as basic mathematical operations.

Eg:-

```
import java.io.*;
class Operator
{
 public static void main(String args[])
 {
   int a=10;
   int b=5;
 System.out.println(a+b);
 System.out.println(a-b);
 System.out.println(a*b);
 System.out.println(a/b);
 System.out.println(a%b);
```

3

3

O/p

15

5

50

2

0

(3) Shift Operator:-

- There are two types of shift operators.
  - (i) Left shift operator
  - (ii) Right shift Operator.

(i) Left shift Operator:-

- The java left shift operator << is used to shift all of the bits in a value to the leftside of the specified number of times.

Eg:-

```
import java.io.*;
class Operator
{
  public static void main(String args[])
  {
    System.out.println(10<<2);
    System.out.println(10<<3);
    System.out.println(20<<2);
    System.out.println(15<<4);
  }
}
```

O/P:-

```
40
80
80
240
```

(ii) Right Shift Operator:-

- The java right shift operator >> is used to move left operands value to right by the numbers of bits specified by the right operand.

Eg:-

```
import java.io.*;
class Operator
{
  public static void main( String args[]);
  {
    System.out.println(10>>2);
    System.out.println(20>>2);
    System.out.println(20>>3);
```

3

3

```
o/p
    2
    5
    2
```

u) Relational Operator:-

- This operator to be used for creating an association between more than one operand in logical format.
- It provides different types of comparison operator through which the mapping cardinality to be possible. So that, it also called as comparison Operator.

Symbol code	Meaning	Eg:- O/P a=6, b=5
>	greater than	a>1 => 1
>=	greater or equal to	a>=b => 1.
<	less than	a<b => 0
<=	less than equal to	a<=b => 0
==	equal to	a==b => 0.
!=	not equal to	a!=b => 1

Eg:-

```
import java.io.*;
class Operator
{
  public static void main(String args[])
  {
    int a=6;
    int b=5;
   System.out.println(a>b);
   System.out.println(a>=b);
   System.out.println(a<b);
   System.out.println(a<=b);
   System.out.println(a==b);
   System.out.println(a!=b);
  }
}
```

O/p

True

True

False

False

True

(5) Assignment Operator:-

- Assignment operator is one of the most common operator.
- It is used to assign the value on its right to the operand on its left.

Eg:-

```
import java.io.*;
class assignment
{
    public static void main(String args[])
    {
        int a=10;
        int b=20;
        a+=4;
        b-=4;
        System.out.println(a);
        System.out.println(b);
    }
}
```

O/P
14
16

(6) Terenary Operator :-

- Java terenary operator is used as one linear replacement for if-then ~~else~~ ~~if~~ else statement and used a lot in java programming.
- It is the only conditional operator which takes three operands.

Eg:-

```
import java.io.*;
class operator
{
public static void main (String args[])
{
    int a=2;
    int b=5;
int min=(a<b)?a:b;
System.out.println(min);
}
}
```

O/P
2

(7) Logical Operator:-

- The logical operator to be used for creating and association between one or more than one operand.
- Here, the association result to be always comes on the format of boolean.
- This operator to be used especially on the design of logical circuit manipulation internally above hardware.
- It provides by default four types of logical operator which is to be executed by using one operand or two operand.
- So that it is to be called unary and binary operator.

a) Logical AND operator (&&) (•)

- When both the operand contains non-zero datavalue, then the desired outcomes to be 1 (True), otherwise 0 (False).
- It is normally dot product of the two data value

Eg:- a=7, b=4;

o/p :- a&&b;

1.

(b) Logical OR (||)

- When any ~~any~~ one of the operand to be contain non-zero data value, then the desired outcomes to be 1 otherwise 0.
- It is normally sum of the two data value

Eg:- a=7, b=4.

o/p:- a||b=1.

(c) Logical NOT (!)

- It is a Unary operator
- When the operand holds non-zero value, then it provides 0 outcomes and viceversa.

Eg:- a = 7;
!a = 0;

d) Logical exclusive OR (^).

- When both the operand contains similar types of data value (means both the operand contains non-zero value or both the operand contains zeroes value, then the desired outcomes. to be zero, otherwise 1.

Eg:- ① a=5, b=6.
a^b = 0

Eg:-

```
import java.io.*;
class Logical
{
    public static void main (String args[])
{
        int a=4, b=5, c=6;
        boolean result;
result = (a>b) || (c>a);
System.out.println(result);
result = (a>b) && (c>a);
System.out.println(result);
}
}
```

3
3

O/P
True
False

(8) Bitwise Operator:-

- It is a binary or unary operator through which all the assembly code data or decimal data to be executed in 16-bit binary code format and provides their resultant outcomes also in assembly code format.

Operator Symbol	Meaning	eg:- a=6, b=7
&	Bitwise AND operator function as logical AND operator	c=a&b =6
|	Bitwise OR operator function as logical OR operator	c=a|b =7
^	Bitwise XOR same as logical XOR	c=a^b; =1
~	It is similar as logical NOT	c=~a =-7
>>	It is shifted bit by bit data value from right to left	a=4 c=a>>2=1
<<	It shifted bit by bit data one by one from left to right	a=4 c=a<<2

Eg:-

```
import java.io.*;
class bitwise
{
  public static void main(String args[])
  {
    int a=6, b=7;
    int c;
```

```
System.out.println(c=a&b);
System.out.println(c=a|b);
System.out.println(c=a^b);
System.out.println(c=~a);
System.out.println(c=a>>2);
System.out.println(c=a<<2);
}
}
```

O/P

```
6
7
1
-7
1
24
```

Decision Making Statement

- Decision making statement is divided into three types such as
  (1) if-statement
  (2) Switch case
  (3) Looping statement.

1) Decision making statement with if statement:-

- The if-statement is a powerful decision statement which is use to control the flow of execution of statement. So, it is further divided into four types.

(i) if - statement

(ii) if - else

(iii) Nested if

(iv) else if

(1) <u>Simple if statement</u> :-

- The simple if statement is represented by

```
if(expression/condn)
  {
     Statement;
  }
```

Eg:- W.A.P to input a number and find out greater number.

→

```
import java.io.*;
import java.util.*;
class Greater
 {
   public static void main(String args[])
    {
      int a,b;
      Scanner sc = new Scanner(System.in);
      a = sc.nextInt();
      b = sc.nextInt();
    if (a>b)
    System.out.println(" a is greater");
    if (b>a)
    System.out.println(" b is greater");
    }
 }
```

O/P
3
5
b is greater

(ii) if-else Statement:-

- This statement to be avoid all the negative features of 'if' statement i.e, it executed if block data, when the condition to be true, otherwise it executed else block when the condition to be false.

```
if (condition)
 {
   statement;
 }
else
   statement;
```

Eg:- W.A.P to input a number and findout the greater number.

```
import java.io.*;
import java.util.*;
class Greater
 {
   public static void main(String args[]),
   {
     int a,b;
   Scanner sc = new Scanner(System.in);
   a = sc.nextInt();
   b = sc.nextInt();
```

```
if (a>b)
System.out.println("a is greater");
else
System.out.println("b is greater");
}
}
```

```
O/P
3
6
b is greater
```

(iii) Nested - if Statement:-

- When the if-block or else block to be contain multiple no. of alloting if-else block within itself is called nested if-else.

Syntax:-

```
if<condition>
  {
    if<condition>
      {
        Statement;
      }
    else
      Statement;
  }
else
  {
    if <condition>
      Statement;
    else
      Statement;
  }
```

Eg: W.A.P to input the three numbers find out the greatest number.

```
import java.io.*;
import java.util.*;
class greatest
{
    public static void main (String args[])
    {
        int a,b,c;
        Scanner sc=new Scanner (System.in);
        a= sc.nextInt();
        b= sc.nextInt();
        c= sc.nextInt();
    if(a>b)
    {
        if(a>c)
        System.out.println(" a is greatest number.");
        else
        System.out.println(" c is greatest number.");
    }
else
    {
        if(b>c)
        System.out.println(" b is greatest number.");
        else
        System.out.println(" c is greatest number.");
    }
}
}
```

O/P

```
3
5
7
c is greatest number
```

(iv) else-if Statement:-

Syntax:-

```
if<condition>
  {
     statement;
  }
else if (condition)
  statement;
else
  {
     statement;
  }
```

Eg:- W.A.P input the three number and findout the greatest.

```
import java.io.*;
import java.util.*;
class greatest
 {
   public static void main (String args[])
    {
      int a,b,c;
      Scanner sc = new Scanner (System.in);
      a = sc.nextInt();
      b = sc.nextInt();
      c = sc.nextInt();
```

```
if ((a>b) && (a>c))
  {
    System.out.println("a is greatest");
  }
  else if ((b>a) && (b>c))
  System.out.println("c is greatest");
else
  {
    System.out.println("c is greatest");
  }
}
}
```

```
o/p
    3
    5
    4
    c is greatest
```

2) Switch case:- (menu driven)

- Java has been in multiway decision statement known as switch. The switch statement test the value of a given variable of express against at least of class value and when a match found, then a block of statement associated with that case is executed.
- The general form of switch statement the explain is syntax.

```
Switch (expression)
{
    case value 1:
            Statement;
            break;

    case value 2:
            Statement;
            break;

    &  :
    :
    case value n:
            Statement;
            break;
    default:
            Statement;
    }
```

Eg:- W.A.P input two numbers and display it addition, substraction, multiplication, division by using switch case

→

```
import java.io.*;
import java.util.*;
class demo
{
    public static void main(String args[])
    {
        int a, b, ch, add;
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter two number");
        a = sc.nextInt();
        b = sc.nextInt();
```

```
System.out.println("Press 1 for addition");
System.out.println("Press 2 for substraction");
System.out.println("Press 3 for multiplication");
System.out.println("Press 4 for division");
System.out.println("Enter your choice");
        ch = sc.nextInt();

 Switch(ch)
  {
     case 1:
            add = a+b;
        System.out.println("Add= "+add);
        break;
     case 2:
            sub = a-b;
      System.out.println("Substraction = "+sub);
      break;
     case 3:
            mul = a*b;
      System.out.println("multiplication= "+mul);
      break;
     case 4:
            float division = (float) a/b;
        System.out.println("division = "+div);
        break;
   default:
        System.out.println("None of choice");
    }

  }
}
```

(3) Looping Statement:- / Iterative Statements:

- Java suppose iterative statements which can perform repetative operations.
- The process of repetative executing a block of statement known as looping.
- It is divided into 3 types.
  - (1) While
  - (2) doWhile
  - (3) for

(1) While:-

- The simplest looping structure is Java is the while statement so it can be represented as while condition.

Syntax:-

```
Counter;
while (Condition)
  {
    logical statement;
    increament/decreament;
  }
```

Eg:- W.A.P to display 1 to 10 with while loop.

```
- import java.io.*;
import java.util.*;
class while
  {
    public static void main( String args[])
      {
        int a=1;
        // Scanner sc = new Scanner (System.in);
```

```
while (a<=10)
  {
    System.out.println(" value of a = " +a);
    a++;
  }
}
}
```

o/p

```
value of a = 1
value of a = 2
value of a = 3
value of a = 4
value of a = 5
value of a = 6
value of a = 7
value of a = 8
value of a = 9
value of a = 10
```

(2) do while :-

The do-while structure was initialized then execute atlast to check the condition.

Syntax:

```
counter;
do
  {
    Statement;
    increament / decrement;
  }
while(condition);
```

Eg:- W.A.P to display 1 to 10 with do while.

```
import java.io.*;
class dowhile
{
    public static void main(String args[])
    {
        int a=1;
        do
        {
            System.out.println("value of a = "+a);
            a++;
        }
        while(a<=10);
    }
}
```

o/p

```
value of a = 1
value of a = 2
value of a = 3
value of a = 4
value of a = 5
value of a = 6
value of a = 7
value of a = 8
value of a = 9
value of a = 10
```

(3) for loop:-

The for loop is another looping structure that provides the more concise loop control structure so it can be presented as

Syntax:-

```
for (counter; condition; incre/decre)
  {
    statement;
  }
```

Eg:- W.A.P to display 1 to 10 with for loop.

```
import java.io.*;
class looping
{
  public static void main (String args[])
  {
    for(int a=1; a<=10; a++)
    {
      System.out.println("value of a= "+a);
    }
  }
}
```

o/p:-

```
value of a = 1
value of a = 2
value of a = 3
value of a = 4
value of a = 5
value of a = 6
value of a = 7
value of a = 8
value of a = 9
value of a = 10
```

1) W.A.P to input a ~~prime~~ number and check whether it is prime or not

→

```
import java.io.*;
import java.util.*;
class prime
{
    public static void main(String args[])
    {
        Scanner sc = new Scanner(System.in);
        int p, cnt = 0;
        System.out.println("Enter a number=");
        p = sc.nextInt();
        for (int i = 1; i <= p; i++)
        {
            if (p % i == 0)
            {
                cnt++;
            }
        }
        if (cnt == 2)
        {
        System.out.println("p is a prime number");
    }
    else
    {
        System.out.println("p is not a prime number");
    }
}
}
```

O/P

```
Enter a number=
5
p is a prime number
```

Command line Argument

- The java command - line argument is an argument ie passed at the time of running the java program.
- The arguments passed from the console can be received in the java program and it can be used as an input.

(Received a main method parameter).

① Program

```
import java.io.*;
class commandline
{
  public static void main (String args[])
  {
   // String obj = new String();
     String obj1, obj2;
     obj1 = args[0];
     obj2 = args[1];
  System.out.println(" Command line values= " +obj1+" " +obj2);
  }
}
```

O/P

```
Java commandline Computerscience ITM
Commandline values = Computerscience ITM
```

② Program

W.A.P input two numbers and add them with the help of commandline arguments.

```
->  import java.io.*;
    class commandline
    {
      public static void main(String args[])
      {
        int a, b, add;
        a = Integer.parseInt(args[0]);
        b = Integer.parseInt(args[1]);
        add = a+b;
        System.out.println(" Command line addition values = " +add);
      }
    }
```

O/P

```
java commandline 4 5
Command line addition values = 9
```

Wrapper Class

Shortnote

- A wrapper class is specially designed to handling primitive datatypes like int, float, char, double.
- Primitive datatypes may be converted into object types by using the wrapper classes which are contain in Java.lang package.

- Wrapper classes can be ~~classified into~~ described as

Primitive datatype	Wrapper class
int	Integer
float	Float
long	Long
double	Double
boolean	Boolean

- The wrapper classes have a no. of unique methods for handling primitive datatypes & object which are explain below.

29/08/2020

Conversion

1) Converting primitive type numbers to object number using constructor method.

Eg:-

(i) int a = 10;
Integer intval = new Integer(a);

(ii) float b = 6.7f;
Float floatobj = new Float(b);

2) Converting object numbers to primitive numbers using typevalue() methods

Eg:- (i) int i = intval.intvalue();

(ii) float j = floatobj.floatvalue();

③ Converting numbers to strings using ~~to str~~ toString() method

Eg:- String str;
str = Integer.toString(a);
→ str = Float.toString(b);

④ Convert String objects to numeric objects using valueOf():-

Eg:- String str = "100";
Integer obj = new Integer();
obj = Integer.valueOf(str);

⑤ Convert String objects to primitive numbers using parse():-

Eg: (i) int i;
i = Integer.parseInt(str);

(ii) String str = "10.56";
float j;
j = Float.parseFloat(str);

① Program

```
import java.io.*;
class wrapperex
{
    public static void main(String args[])
    {
      int a=10;
      Integer intval = new Integer(a);
      System.out.println("object type = "+intval);
      System.out.println("Primitive type = "+a);
```

```
int i = intval.intvalue();
System.out.println(" convert Object type to primitive type ="+i);
String str = "100";
Integer obj;
obj = Integer.valueOf (str);
System.out.println("String convert to object type ="+obj);
}
}
```

```
o/p
    java wrapperex
    Object type =10
    Primitive type =10
    convert object type to primitive type =10
    String convert to object type =100
```

(2) W.A.P input two number and findout power.

```
import java.io.*;
import java.lang.*;
class powerex
{
 public static void main(String args[])
 {
  double a = 5;
  double p=2;
  double powval = Math.pow(a,p);
     System.out.println("Power value= "+powval);
 }
}
```

```
o/p
    25
```

③ W.A.P input and find the squareroot

```
import java.io.*;
import java.lang.*;
class sqrt
{
    public static void main(String args[])
    {
        double num=36;
        double sroot = Math.sqrt(num);
        System.root.println("Squareroot value = "+sroot);
    }
}
```

o/p

6

Programs on Wrapper class

①

```
import java.io.*;
import java.util.*;
class WrapperEx
{
    public static void main (String args[])
    {
        Scanner sc = new Scanner(System.in);
        int a;
        a=90;
    Integer obj = new Integer(a);
    System.out.println("Primitive type value= "+a);
    System.out.println("Wrapper class value= "+obj);
    String mark = "450";
    int m = Integer.parseInt(mark);
    System.out.println("Converted value = "+m);
```

```
String per = "34.56";
float fltvalue = Float.parseFloat(per);
System.out.println("Converted value = "+fltvalue);
```

3
3

O/P

```
Primitive type value = 90
wrapper class value = 90
Converted value = 450
Converted value = 34.56
```

03/09/2020

Type casting:-

- There is a need to store a ~~variable~~ value of one type into a variable of another type in ~~search~~ such situation. We must cast the ~~variable~~ value to be stored by proceeding it with the type name in paranthesis.

Syntax:-

```
type variable 1
= (type) variable 2;
```

Ex:- int variable a = 20;
then byte b = (byte) a; (Explicit type casting).

-> Casting input a smaller time may result in a ~~class~~ loss of some data. Similarly the float & double can be cast to any other type except boolean.

-> Casting a floating point value to an integer will result in a last of the factunal part.

-> To asked that result in no loss of information.:-

From	To
① Byte	① Short, char, int, long, float, double
② Short	② int, long, float, double
③ int	③ long, float, double.
④ long	④ float, double
⑤ float	⑤ double

- There are two types of typecasting.
  (1) Automatic Typecasting
  (2) External Typecasting.

Eg:- j = a / b;
j = (float) a/b;

~~Java Methods~~

- The automatic conversion is possible to assigning a variable of one type to a variable of a different type without a cast. This is known as automatic type conversion or implicit conversion.
  Eg:- byte b = 75;
  int a = b;

Java Methods

- A Java method is a collection of statements that are grouped together to perform an operation.
- System.out.println() method,
  eg:- The system actually executes several statements in order to display a message on the console.

methods / function → as a paranthessis ()

Syntax

```
type methodname(arguments)
{
    // statements.
}
```

→ public static int methodName (int a, int b)

```
{
    // body
}
```

Eg:-

```
int display()
{
    //Statement
}
```

Program

① W.P add the two numbers given in the program.

```
import java.io.*;
class Add
{
    Void add()
    {
        int a, b, c;
        a = 10;
        b = 20;
        c = a + b;
    System.out.println(" Add = " + c);
    }
public static void main (String args[])
{
    add();
}
}
```

O/P

Add = 30

Buffer Reader class

- The Bufferedreader class of Java is used to read the stream of characters from the specified source (character input stream).
- The constructor of this class accepts an inputstream object as a parameter.
- This class provides a method named read() and readline() which reads and returns the character and nextline from the source (respectively) and returns them.

☆. Instantiate an InputStreamReader class bypassing your Inputstream object as a parameter.

☆. Then, create a BufferReader, by passing the above obtained InputstreamReader object as a parameter.

☆. Read the data from the current reader as string using the readLine() or read() method.

Data can be input
Scanner class
Command line
Buffer reader

③ Program

```
① import java.io.BufferedReader;
  import java.io.IOException;
  import java.io.InputStreamReader;

  class Employee
  {
      String name;
      int id;
      int age;
```

```
Employee(String name, int age, int id)
 {
     this.name = name;
     this.age = age;
     this.id = id;
 }
public void displayDetails()
 {
    System.out.println("Name:"+this.name);
    System.out.println("Age:"+this.age);
    System.out.println("Id:"+this.id);
 }
}
public class ReadData
 {
    public static void main(String args[])
    throws IOException
    {
       BufferedReader reader = new BufferedReader(
                    (new InputStreamReader(System.in));
       System.out.println("Enter your name:");
       String name = reader.readLine();
       System.out.println("Enter your age:");
       int age = Integer.parseInt(reader.readLine());
       System.out.println("Enter your Id:");
       int id = Integer.parseInt(reader.readLine());
```

```
Employee std = new Employee(name, age, id);
std.displayDetails();
}
}
```

O/P

```
Enter your name:
Krishna
Enter your age:
25
Enter your Id:
1233
Name: Krishna
Age: 25
Id: 1233
```

(2) W.A.P add two numbers by Buffered reader class.

```
import java.io.*;
public class BufferedReaderExample
{
    public static void main(String args[]) throws Exception
    {
        InputStreamReader r = new InputStreamReader(System
        BufferedReader br = new BufferedReader(r);
        System.out.println("Enter your data");
        String num1 = br.readLine();
        String num2 = br.readLine();
        int add = Integer.parseInt(num1)+Integer.parseInt(num2)
        System.out.println("Add = "+add);
    }
}
```

Dynamic input or runtime input :- (Scanner Class)

- Scanner is a class in java.util package used for obtaining the input of the primitive types like int, double, etc and strings.
- It is the easiest way to read input in a java program, though not very efficient if you want an input method for scenarios where time is a constraint like in competitive programming.
- To create an object of Scanner class, we usually pass the predefined object System.in, which represents the standard input stream. We may pass an object of class file if we want to read input from a file.
- To read strings, we use nextline().

next Int() → It is used for inputing for integer value

nextfloat → It is used for inputing float type value

next line :- It is also used for string type value with space

next :- It is used for string type value.

Ex:- Scanner sc = New Scanner (System.in);

Program

W.A.P input two nos and add them by using Scanner class.

→
```
import java.io.*;
import java.util.*;
class add
{
    public static void main (String args[])
    {
        int a,b,c;
    Scanner sc = new Scanner(System.in);
    System.out.println ("Enter 2 no");
    a = sc.nextInt();
```

```
    b = sc.nextInt();
    c = a+b;
System.out.println("Add = "+c);
  }
}
```

O/p

```
Enter 2 no: 5
6
Add = 11
```

Instance Variables	Static (class) variables
- Instance variables are declared in a class, but outside a method, constructor or any block.	- Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.
- Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.	- Static variables are created when the program starts and destroyed when the program stops.
- Instance variables can be accessed directly by calling the variable name inside the class. However, within static methods (when instance variables are given accessibility), they should be called using the fully qualified name. "Object Reference Variable Name".	- Static variables can be accessed by calling with the class name. ClassName.VariableName.
- Instance variables hold values that must be referenced by more than one method, constructor or block or essential parts of an object's state that must be present throughout the class.	- There would only be one copy of each class variable per class, regardless of how many objects are created from it.

END OF THE CHAPTER ... Chiku Pradhan

Object - Oriented Programming Overview

Object:-

(i) In java it is essentially a block of memory of an object that contains space to store all the instance variables.

(ii) Creating an object is also refers to as instance.

(iii) Objects in java & created by using the new operator.

(iv) The new operator creates an object of the specified class & returns a reference to the object.

(v) So that the object is threated as

Definition: The state and behaviours of the base program component is known objects.

Syntax

<classname> <obj name>;

<obj name> = new <class name()>;

Ex:- We have a class rectangle which is define in the above class.

So that we can create an object of rectangle object.

Rectangle ret = new Rectangle();

- The method Rectangle is the default constructor of the class we can create any number of objects of rectangle.

Rectangle().

Rectangle ret = new Rectangle()

declare allocate memory

Class:-

(i) A class is essentially description of how to make an object that contains fields and methods in java data items are called as fields and functions are known as method.

(ii) Calling a specific method of a class by using an object

(iii) In Java the class type has been defined with a specific variable is known as instances of classes or an object.

- It is the collection of heterogeneous types of datamember of methods for providing some identification or objective.

Syntax

```
class (class name)
{
    fields;
    methods;
}
```

Program

① Ex:-

```
class Rectangle
{
    float length, breadth, area;
    void area()
    {
        length = 2.5f;
        breadth = 3.5f;
    area = length * breadth,
System.out.println("area" + "float");
}
}
```

① W.A.P to input length & breadth to calculate area of rectangle by using obj & class function.

→

```
import java.io.*;
import java.util.*;
class Rectangle
{
    float length, breadth, area;
    void input()
    {
        Scanner sc = new Scanner(System.in);
    length = sc.nextFloat();
    breadth = sc.nextFloat();
    }
    void area()
    {
        area = length * breadth;
    System.out.println(" area = " + area);
    }
}
class Rectangleem
{
    public static void main(String args[])
    {
        Rectangle rect = new Rectangle();
        rect.input();
        rect.area();
    }
}
```

With a Single class

```
import java.io.*;
import java.util.*;
class Rectangle
 {
   float length, breadth, area;

   void input()
    {
      Scanner sc = new Scanner (System.in);
        length = sc.nextFloat();
        breadth = sc.nextFloat();
    }
   void area()
    {
        ar = length * breadth;
   System.out.println("Area" +ar);
    }
 public static void main (String args[])
  {
    Rectangle Rect = new rectangle();
      rect.input();
      rect.area();
  }
}
```

(2) W.A.P to input two number and add them with the help of classes & object.

-> with single class

```
import java.io.*;
class Add
{
    int a,b,c;
    void input()
    {
        a=10;
        b=20;
    }
    void add()
    {
        c=a+b;
        System.out.println("Addition="+c);
    }
    public static void main(String args[])
    {
        Add obj=new Add();
        obj.input();
        obj.add();
    }
}
```

With double class

```
import java.io.*;
class Add
{
    int a,b,c;
    void input();
    {
        a=10;
        b=20;
    }
    void add()
    {
        c=a+b;
        System.out.println("Addition =" +c);
    }
}
Class Addresult.
{
    public static void main(String args[])
    {
        Add obj=new Add();
        obj.input();
        obj.add();
    }
}
```

Constructor

- A constructor is a special type of member function of method.
- This enables a object that initialized all the variables all the object.
- Similarly Java supports a special type of method called as Constructor that enables an object to initialized itself when it is created.
- Constructors are the same name as the class itself.
- Don't specific a return type. this is because it returns the instance of the class itself.

- It is divided into two types:
  1) Default Constructor
  2) Parameterised Constructor

1) Default Constructor:-

- Whenever a constructor is declared without any parameter or argument this is known as Default constructor.
- The default constructor is used to provide the default values to the objects like 0, null, etc... depending on the type

Syntax of Constructor (Inside)

```
class<classname>
{
    int x;
    Constructor classname();
    {
        object initialization;
    }
}
```

Syntax

```
<class-name>(){}
```

Note:- If there is no constructor in a class, compiler automatically creates a default constructor.

For Constructor:-

- This types of function cannot be return any types of datavalue even void.

For Default Constructor

① Program

W.A.P to input radius and find the area of a circle

```
import java.io.*;
import java.util.*;
class Circle
{
    float rad, pi, ar;
  Circle()
  {
     pi = 3.14f;
  }
```

```
void input()
  {
      Scanner sc = new Scanner (System.in);
      System.out.println(" Enter radius of circle = ");
      rad = sc.nextFloat();
  }

void area()
  {
     ar = pi * rad * rad;
     System.out.println( "Circle Area= " +ar);
  }
}

class CircleArea
  {
     public static void main (String args[])
       {
          Circle ci = new Circle();
          ci.input();
          ci.area();
       }
     }
```

② W.A.P to input two number or directly add the number

```
import java.io.*;
class Add
  {
    int a,b,c;
```

```
Add ()
 {

 }
void input()
 {
    a=10;
    b=20;
 }
void add()
 {
   c=a+b;
   System.out.println("Addition =" +c);
 }
}
class Addresult.
 {
   public static void main (String args[])
    {
      Add obj = new Add();
      obj.input();
      obj.add();
    }
 }
```

2) Parametrized Constructor

- It is a special type of method which contains some types of parameter or arguments.
- It passes different types of data value simultaneously.

Syntax

```
sum( int a, float b);
```

Program

1) W.A.P input the length and breadth and find its area of rectangle.

```
import java.io.*;
class ParameterizedEx
{
    float len, bt, area;
    {
        len = x;
        bt = y;
    }
  void area()
  {
    area = len * bt;
    System.out.println(" Rectangle area = "+area);
  }
}
class ParameterisedMain
{
    public static void main(String args[])
    {
```

```
ParameterisedEx obj = new ParameterisedEx (2.3f, 4.5f);
obj.area();
}
}
```

Object as a Parameter

- Whenever a function which containing a parameter which is object type that is known as ~~as~~ a object as a parameter.

Program

```
import java.io.*;
import java.util.*;
class ObjectConstructor
{
  float len, br, area;
  ObjectConstructor()
  {

  }
  ObjectConstructor(ObjectConstructor ob1, ObjectConstructor ob2)
  {
    len = ob1.len + ob2.len;
    br = ob1.br + ob2.br;
  }
```

```
void input()
{
    Scanner sc = new Scanner(System.in);
    len = sc.nextFloat();
    br = sc.nextFloat();
  }
void displayArea()
 {
    area = len * br;
    System.out.println(" Rectangle Area=" + area);
 }
}

class ObjectParameter
{
  public static void main(String args[])
  {
      ObjectConstructor rect1 = new ObjectConstructor();
      ObjectConstructor rect2 = new ObjectConstructor();
      rect1.input();
      rect2.input();
      ObjectConstructor rect3 = new ObjectConstructor(rect1, rect2);
      rect1.displayArea();
      rect2.displayArea();
      rect3.displayArea();
  }
}
```

O/P

~~Java~~ java ObjectParameter

2.3

2.6

4.2

4.5

Rectangle Area = 5.9799995

Rectangle Area = 18.9

Rectangle Area = 46.149998

(1) Constructor Overloading in Java

- A constructor is just like a method but without return type. It can also be overloaded like Java methods.
- Constructor overloading is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

Program

```
import java.io.*;
import java.util.*;
class ObjectConstructor
{
```

```
float len, br, area;
ObjectConstructor()
{
    len = 0.0f;
    br = 0.0f;
}
ObjectConstructor(ObjectConstructor ob1, ObjectConstructor ob2)
{
    len = ob1.len + ob2.len;
    br = ob1.br + ob2.br;
}

void input()
{
    Scanner sc = new Scanner(System.in);
    len = sc.nextFloat();
    br = sc.nextFloat();
}
void displayArea()
{
    area = len * br;
    System.out.println("Rectangle Area = " + area);
}
}
class ObjectParameter
{
    public static void main(String args[])
    {
```

```
ObjectConstructor rect1 = new ObjectConstructor();
ObjectConstructor rect2 = new ObjectConstructor();
rect1.input();
rect2.input();
ObjectConstructor rect3 = new ObjectConstructor(rect1, rect2);
rect1.displayArea();
rect2.displayArea();
rect3 displayArea();
}
}
```

Java Constructor	Java Method
1) A constructor is used to initialize the state of an object.	1) A method is used to expose the behaviour of an object
2) A constructor must not have a return type.	2) A method must have a return type.
3) The constructor is invoked implicity.	3) The method is invoked explicity.
4) The Java compiler provides a default constructor if you don't have any constructor in a class.	4) The method is not provided by the compiler many case.
5) The constructor name must be same as the class name.	5) The method name may or may not be same as the class name.

(11) Method Overloading

- If a class has multiple methods having same name but different in parameters, it is known as Method Overloading.
- If we have to perform only one operation, having same name of the methods increases the readability of the program.
- Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int, int) for two parameters, and b(int, int, int) for three parameters then it may be difficult for you as well as other programmers to understand the behaviour of the method because its name differs.
- So, we perform method overloading to figure out the program quickly.

Advantages

- Method overloading increases the readability of the program.

Different ways to Overload the method.

- There are two ways to overload the method in Java.

  (1) By changing number of arguments

  (2) By changing the data type

Program

W.A.P input numbers and add them with the help of method overloading.

```
import java.io.*;
import java.util.*;
class add
{
    int sum;

  void add()
   {
      int a = 10, b = 20;
      sum = a+b;
      System.out.println("Sum value1 = "+sum);
   }

  void add(int x,int y)
   {
      sum = x+y;
      System.out.println("Sum value2 = "+sum);
   }

  void add(float x, float y)
   {
      float s = x+y;
      System.out.println("sum value3 = "+s);
   }
}
```

```
class demooverload
{
    public static void main( String args[])
    {
        add obj = new add();
        obj.add(5,4);
        ~~obj.add(~~
        obj.add(5.2f,6.1f);
    }
}
```

O/p

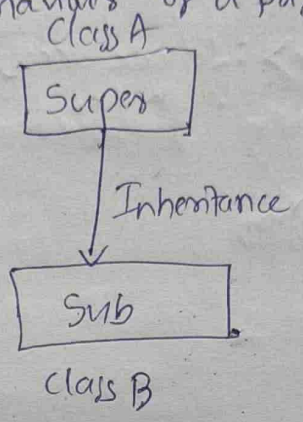
```
java demooverload
Sum value1 = 30
Sum value2 = 9
Sum value3 = 11.299999
```

Inheritance

- Java classes can be reused in several ways. This is basically done by using new or creating new classes by reusing the properties of existing ones.
- This mechanism is deriving a new class from an old class is known as an inheritance.
- It provides one capability to fetch some datamembers and methods for creating new program from an existing or program class. That means, to creating a new

program or class from an existing old program or class features.

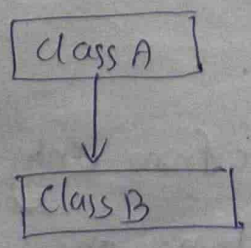
- Here, the old existing class to be called as superclass / base class / parent class.
- Inheritance is a mechanism in which one object acquires all the properties and behaviours of a parent object.



- The class that properties from another class is called subclass.
- The class whose properties are inherited by subclass is called super class / base class.

Advantages / Benefits of Inheritance

- It provides code reusability in nature.
- Here, the memory consumption is very less.
- Datavalue and method to be sharing from one object to other object.
- Here, the base class datavalue always to be execute with the help of derived class.



(derived class)

without permission it cannot be execute.

Types of Inheritance

- It is divided into following types such as

1. Single inheritance
2. Multilevel inheritance
3. Hierachall inheritance
4. Multiple inheritance (Not ~~a~~ in inheritance) otherwise is interface)

4) Multiple inheritance:-

- Java doesn't support directly multiple inheritance so this ~~do~~ concept is implemented by using the concept of interface.

1) Single inheritance

- A new class is derive from only one class. That means if a program to be contains one super class and only one derived class.

- Here, one subclass is inherited by one base class only.
- A subclass is created by using a keyword, "extends."

Syntax

```
Class A
{
    properties (field/method)
}
Class B extends A
{
    ————
    ————
    ————
}
```

Eg. Write a program add first two numbers and then substract with the help of single Inheritance.

```
import java.io.*;
class A
{
  int a,b,c;
  void input()
  {
    a=20;
    b=30;
  }
  void add()
  {
    c=a+b;
    System.out.println("Sum="+c);
  }
}
```

```
class B extends A
{
    void sub()
    {
        int x;
        x = a-b;
        System.out.println("substraction =" +x);
    }
}

class SingleInhentance
{
    public static void main(String args[])
    {
        B obj = new B();
        obj.input();
        obj.add();
        obj.sub();
    }
}
```

O/P

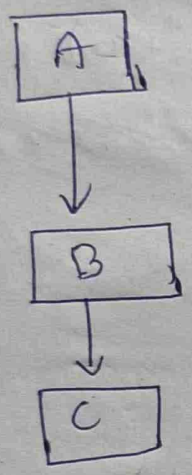
```
java SingleInhentance
Sum = 50
Substraction = -10
```

29/09/2020

2) Multilevel Inheritance

- This type of inheritance provides for creating multiple no. of derived class from another intermediate derived class.
- Here, the intermediate derived class to be otherwise called as intermediate base class.

E.g:- BabyDog class inherits the Dog class which again inherits the Animal class.



Program

W.A.P input values and find the area of a room and volume of a room with the help of ~~multilevel~~ multilevel inheritance

```
import java.io.*;
import java.util.*;

class Room
{
    float Len, brt;

    Room(float x, float y)
    {
        len = x;
        brt = y;
    }
}
```

```
class RoomArea extends Room
 {
    RoomArea(float l, float b)
     {
        super(l,b);
     }
  void area()
   {
      float ar = len * brt;
      System.out.println(" Room area = "+ar);
    }
class RoomVolume extends RoomArea
  {
     float ht;
   RoomVolume ( float ln, float bt, float h)
    {
       super ( ln, bt);
       ht = h;
     }
    void volume()
    {
       float vl = len * brt * ht;
       System.out.println (" RoomVolume = "+vl);
    }
  }
class Roommulti
 {
   public static void main (String args[])
```

```
{
    Scanner sc = new Scanner(System.in);
    float rl, rb, rh;
    System.out.println(" Enter len, brt and height=");
    rl = sc.nextFloat();
    rb = sc.nextFloat();
    rh = sc.nextFloat();
    RoomVolume obj = new RoomVolume (rl, rb, rh);
    obj.Volume();
    obj.Roomarea();
  }
}
```

O/p

```
Enter len, brt and height=
4.5
3.5
9.2
```

∵ Super class

calling a super class constructor from a child class

Eg:-
Room = A
RoomArea = B
RoomVolume = C

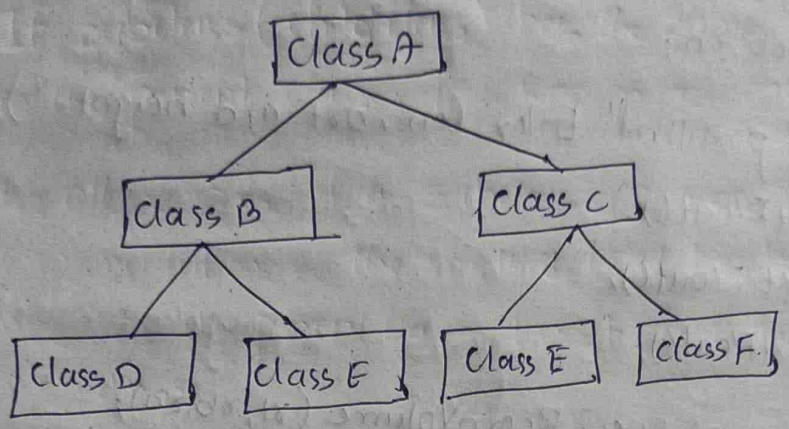
3) Hierarchical Inheritance ~~Exampl~~

- When two or more classes inherits a single class, it is known as hierarchial inheritance.

Eg:- Dog and cat classes inherits the Animal class.

~~Syntax~~:

Syntax



Program

Write a program add and substract and Multiply

```
import java.io.*;
import java.util.*;

class Add
{
    int a,b,c;
    Void input()
    {
        a=10;
        b=20;
    }
    void add()
    {
        c= a+b;
        System.out.println(" Addvalue= "+c);
    }

}
```

```
class Substract extends Add
 {
    void sub()
     {
        int d;
        d= a-b;
      System.out.println(" Substract value ="+d);
     }
 }

class Multiply extends Add
  {
     void mul()
      {
        int e;
        e= a*b;
        System.out.println(" Multiply value= "+e);
      }
  }
class ArithmaticIn
 {
   public static void main(String args[])
   {
     Substract obj = new substract();
     Multiply obj1=new Multiply();
     obj.input();
     obj.add();
     obj.sub();
     obj1.input();
     obj1.mul();
   }
 }
```

Method Overriding

- If subclass (child class) has the same method as declared in the parent class, it is known as method overriding.
- In otherwords, if a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

Usage of Method overriding

- Method overriding is used to provide the specific ~~implementation~~ implementation of a method which is already provided by its superclass.
- Method overriding is used for runtime polymorphism.

Rules for Method Overriding

- The method must have the same name as in the parent class.
- The method must have the same parameter as in the parent class.
- There must be an inheritance relationship

Program

```
import java.io.*;
class Overridemain
  {
      void maindisplay()
        {
          System.out.println("Main Override Display");
```

```
    }
}
class SubOverride extends Overridemain
{
    void maindisplay()
    {
        System.out.println("Suboverride Display");
    }
}
class SuperOverride
{
    public static void main(String args[])
    {
        SubOverride obj = new SubOverride();
        obj.maindisplay();
    }
}
```

Output

Suboverride Display.

Final Variables and Methods

- All methods and variables can be overriden by default in subclasses.
- If we wish to prevent the subclasses from overriding the members of the superclass.
- We can declare them as final using the keyword final as a modifier.

- Making a method final ensures that the functionality defined in this method will never be altered. in any way.
- Similarly, the value of a final variable can never be changed.
- Final variables, behave like class variables and they donot take any space on individual objects of the class.

Syntax

```
final <datatype> <variable name> = value
final <datatype> <methodname> ()
  {
    Statements.
  }
```

Eg:-

```
final float pi = 3.14f;
final void display()
  {
    System.out.println("Hello");
  }
```

Final classes:-

- A class that cannot be subclassed is called a final class. It can't be derived by another class
- The user or programmer like to prevent a class being further subclass for any security reasons/purpose.

Eg:-

```
final class SuperA
{
    It can be Subderived
}
```

or

```
final class subclass B extends parent A
{
    It can be derived by another class.
}
```

→ final is one sided only execute parent class. but not child class.

→ Method as final it can't be override

Abstract classes & methods:-

- A method final we ensured that the method is not predefined in a subclass. The method can never be subclass.
- Java allows us to do something that is exactly opposite to this. This means that we can indicate that a method must always be redefined subclass which is making overriding compulsory.
- This is done by using the modify keyword i.e, abstract. in the method defination.

Eg:-

```
abstract - Classdemo
{
    // final declaration,
    ---
    ---
    abstract void display()
    ---
    ---
}
```

- When a class contains one or more abstract methods then it should also be declared as abstract. ~~classes such as~~

~~1. We can't~~

→ There are some criteria for meeting abstract classes such as

1) We can't use abstract classes to instant a object directly.

```
Demo obj = new demo();
```

It is illegal

2) The abstract method of an abstract class must be define in its subclass.

3) We can't declared abstract constructors of abstract static method.

Program 1

```
import java.io.*;
abstract class College
{
    abstract void display();
}

class collegedetails extends college
{
    public void display()
    {
        System.out.println("It is an abstract class");
    }
}
```

```
class CollegeMain
{
    public static void main(String args[])
    {
        CollegeMain obj = new collegedetails();
        obj.display();
    }
}
```

Program 2:

```
import java.io.*;
import java.util.*;
abstract class demoab
{
    abstract void display();
}
class subdemo extends demoab
{
    int a,b;
  Subdemoab(int x,int y)
    {
        a=x;
        b=y;
    }
public void display()
  {
    System.out.println("value = "+a + " "+b);
  }
}
```

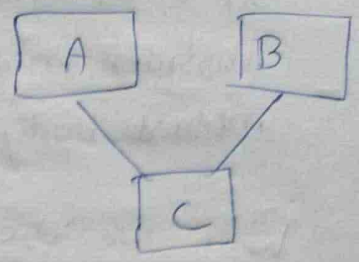
```
class mainab
{
    public static void main (String args[])
    {
        subdemoab obj = new subdemoab (10,20)
        obj.display();
    }
}
```

~~Disadvantages of Multiple Inheritance~~

Interface

- An interface is also a kind of class interfaces contain methods and variables but with a major different is that interfaces define with abstract method and final fields.
- This means that interfaces don't specific any code that implement data field contain only constant
- The interface as a responsibility that implement an interface to define the code for implimentation of this method.
- Java supports ~~and~~ the concept of interface which is used for creating multiple inheritance
- A Java class can't be subclass of more than one superclass, so that it can implement more than one interface which can value to create a multiple inheritance without any problem.

Syntax




```
interface <interface name>
{
    field declaration;
    method declaration;
}
```

-> Inside an interface is declared as static final type
<variable name> = value;

Ex:- static final float pi = 3.14f;

- The method declaration inside an interface which is consider an abstract type. which ends with a semicolon (;)

Ex:- interface Circle
```
{
    static final float pi = 3.14f;
    void area();
}
```

In Other Words

Interface

- An interface is a blueprint of a class. It has static constants and abstract methods.
- The interface is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

- In other words, we can say that interfaces can have abstract methods and variables. It cannot have a method body.

Why we use Java Interface?

- There are mainly three reasons to use interface
  - It is used to achieve abstraction.
  - By interface, we can support the functionality of multiple inheritance.
  - It can be used to achieve loose coupling.

Difference between class and interface

Class	Interface
(i) The members of a class can be constant or variable.	(i) The member of an interface are always declared as constant that final.
(ii) The class defination can contain the code for each of its method. This method can be abstract or non-abstract.	(ii) But the methods in interface are abstract in neutral.
(iii) It can be instanciated by declaring an objects.	(iii) It can't be used to declare objects it only be inheritance by class.
(iv) It can use various access specifier like public, private or protected.	(iv) It can only use the public access specifier.

1) Write a interface program to display "Hello dear Student".

```
import java.io.*;
interface display
{
    void display();
}
class subdisplay implements display
{
    void display()
    {
        System.out.println("Hello dear Students");
    }
}

class InterfaceEx
{
    public static void main(String args[])
    {
        subdisplay obj = new subdisplay();
        obj.display();
    }
}
```

O/P

```
java InterfaceEx
Hello dear students
```

2) Write a interface program to display pi value and "Hello dear students"

-)
```
import java.io.*;
interface display
{
    float pi=3.14f;
    void display();
}
class subdisplay implements display
{
    public void display()
    {
        System.out.println("Hello dear Students");
        System.out.println("pi value="+pi);
    }
}
class InterfaceEx
{
    public static void main(String args[])
    {
        subdisplay obj=new subdisplay();
        obj.display();
    }
}
```

O/P

```
java InterfaceEx
Hello dear Students
pi value= 3.14
```

3) Write a program to find area of a circle with the help of interface

→
```
import java.io.*;
interface display1
{
  float pi = 3.14f;
  void display();
}
interface display2 extends display1
{
    void area();
}
class subcircle implements display2
{
    float ar;
    public void area()
    {
      float r = 3.4f;
      ar = pi * r * r;
    }
  public void display()
{
  System.out.println("Area = " + ar);
}
}

class Interfacecircle
{
  public static void main (String args[])
  {
    subcircle obj = new subcircle();
    Obj.area();
  }
}
```

Packages

long answer

- Java package is a mechanism of grouping similar type of classes, interfaces, and sub-classes collectively based on functionality. When software is written in the Java programming language, it can be composed of hundreds or even thousands of individual classes. It makes sense to keep things organized by placing related classes and interfaces into packages.

- A Package can be defined as a grouping of related types (classes, interfaces, enumerations and annotations) providing access protection and namespace management.

- Programmers can define their own packages to bundle group of classes/interfaces, etc. It is a good practise to group related classes implemented by you so that a programmer can easily determine that the classes, interfaces, enumerations, and annotations are related.

- Since, the package creates a new namespace there won't be any name conflicts with names in other packages. Using packages, it is easier to provide access control and it is also easier to locate the related classes

Types of Packages:

There are two types of packages:-

(i) Built in Packages

(ii) User Defined Packages / Custom package.

i) Built in Package:-

- Built-in packages or predefined packages are those that come along as a part of JDK (Java Development Kit) to simplify the task of Java programmer.
- They consist of a huge number of predefined classes and interfaces that are a part of Java API's.
- Some of the commonly used built-in packages are java.lang, java.io, java.util, java.applet, etc. Here's a simple program using a built-in package.

Program

```
import java.util.ArrayList;
class BuiltinPackage
{
    public static void main(String args[])
    {
        ArrayList<integer> mylist=new ArrayList<>(3);
        mylist.add(3);
        mylist.add(2);
        mylist.add(1);
        System.out.println("The elements of List are:"+mylist);
    }
}
```

(ii) User Defined Packages:-

- User-defined packages are those which are developed by users in order to group related classes, interfaces and subpackages.

- Creating a package in Java is a very easy task. Choose a name for the package and include a package command as the first statement in the Java source file.

- The Java source file can contain the classes, interfaces, enumerations, and annotation types that you want to include in the package. For example, the following statement creates a package named MyPackage.

Example: package MyPackage;

- The package statement simply specifies to which package the classes defined belongs to.

Program

```
package Arithmatic;
public class Calculation
{
    public void add(int x, int y)
    {
        int z = x+y;
        System.out.println("Add = "+z);
    }
    public void sub( int x, int y)
    {
        int z = x-y;
        System.out.println("Sub= "+z);
    }
    public void mul (int x, int y)
    {
        int z = x*y;
        System.out.println("Multiply= "+z);
    }
```

```
public void div(int x, int y)
  {
    int z = x/y;
    System.out.println(" Division = " + z);
  }

}
```

Program

```
import java.io.*;
import Arithmatic.Calculation;
class ArithEx.
  {
    public static void main(String args[])
      {
        Calculation obj = new Calculation();
        obj.add(12,20);
        obj.sub(30,20);
        obj.mul(12,5);
        obj.div(10,2);
      }
  }
```

O/P

```
d:
cd Arithmatic
javac Arithmatic.java.
cd..
javac ArithEx.java.
java ArithEx
Add = 32
Sub = 10
Multiply = 60
Division = 5.
```

Advantages of Packages:-

i) **Re-usability:-** The classes contained in the packages of another program can be easily reused.

ii) **Name Conflicts:-** Packages help us to uniquely identify a class, for example, we have company sales. Employee and company, marketing, Employee classes

iii) **Controlled Access:-** Offers access protection such as protected classes, default classes and private class.

iv) **Data Encapsulation:-** They provide a way to ~~hide~~ hide classes, preventing other programs from accessing classes that are meant for internal use only.

v) **Maintainence:-** With packages, you can organize your project better and easily locate related areas.

~~20/10/2020~~

Inheritance	Aggregation
- Inheritance extends the functionality of a class by creating a subclass	- Aggregation create new functionality by taking other classes and combining them into a new class.

Polymorphism

- Poly means many and morphism means forms
- ~~Ability to~~ ~~wer~~ Two types of Polymorphism
  1. Runtime
  2. Compile time

we can perform in method overloading and method overriding

Chiku Pradhan

Upcasting and Downcasting in Java

A process of converting one data type to another is known as **Typecasting** and **Upcasting** and **Downcasting** is the type of object typecasting. In Java, the object can also be typecasted like the datatypes. **Parent** and **Child** objects are two types of objects. So, there are two types of typecasting possible for an object, i.e., **Parent to Child** and **Child to Parent** or can say **Upcasting** and **Downcasting**.

In Java, the object can also be typecasted like the datatypes. **Parent** and **Child objects** are two types of objects. So, there are two types of typecasting possible for an object, i.e., **Parent to Child** and **Child to Parent** or can say **Upcasting** and **Downcasting**.

Typecasting is used to ensure whether variables are correctly processed by a function or not. In **Upcasting** and **Downcasting**, we typecast **a child object to a parent object** and **a parent object to a child object** simultaneously. We can perform Upcasting implicitly or explicitly, but downcasting cannot be implicitly possible.

1) Upcasting

Upcasting is a type of object typecasting in which a **child object** is typecasted to a **parent class object**. By using the Upcasting, we can easily access the variables and methods of the parent class to the child class. Here, we don't access all the variables and the method. We access only some specified variables and methods of the child class. **Upcasting** is also known as **Generalization** and **Widening**.

UpcastingExample.java

```
class Parent{
  void PrintData() {
    System.out.println("method of parent class");
  }
}

class Child extends Parent {
  void PrintData() {
    System.out.println("method of child class");
  }
}
class UpcastingExample{
  public static void main(String args[]) {

    Parent obj1 = (Parent) new Child();
```

2

```
    Parent obj2 = (Parent) new Child();
    obj1.PrintData();
    obj2.PrintData();
  }
}
```

Output:

method of childClass
method of child class

2) Downcasting

Upcasting is another type of object typecasting. In Upcasting, we assign a parent class reference object to the child class. In Java, we cannot assign a parent class reference object to the child class, but if we perform downcasting, we will not get any compile-time error. However, when we run it, it throws the **"ClassCastException"**. Now the point is if downcasting is not possible in Java, then why is it allowed by the compiler? In Java, some scenarios allow us to perform downcasting. Here, the subclass object is referred by the parent class.

Below is an example of downcasting in which both the valid and the invalid scenarios are explained:

DowncastingExample.java

```
//Parent class
class Parent {
    String name;

    // A method which prints the data of the parent class
    void showMessage()
    {
        System.out.println("Parent method is called");
    }
}

// Child class
class Child extends Parent {
    int age;

    // Performing overriding
    @Override
```

```
18.   void showMessage()
19.   {
20.       System.out.println("Child method is called");
21.   }
22.}
23.
24.public class Downcasting{
25.
26.   public static void main(String[] args)
27.   {
28.       Parent p = new Child();
29.       p.name = "Shubham";
30.
31.       // Performing Downcasting Implicitly
32.       //Child c = new Parent(); // it gives compile-time error
33.
34.       // Performing Downcasting Explicitly
35.       Child c = (Child)p;
36.
37.       c.age = 18;
38.       System.out.println(c.name);
39.       System.out.println(c.age);
40.       c.showMessage();
41.   }
42.}
```

Output:

Why we need Upcasting and Downcasting?

In Java, we rarely use **Upcasting**. We use it when we need to develop a code that deals with only the parent class. **Downcasting** is used when we need to develop a code that accesses behaviors of the child class.

Difference between Upcasting and Downcasting

These are the following differences between Upcasting and Downcasting:

S.No	Upcasting	Downcasting
1.	A child object is typecasted to a parent object.	The reference of the parent class object is passed to the child class.
2.	We can perform Upcasting implicitly or explicitly.	Implicitly Downcasting is not possible.
3.	In the child class, we can access the methods and variables of the parent class.	The methods and variables of both the classes(parent and child) can be accessed.
4.	We can access some specified methods of the child class.	All the methods and variables of both classes can be accessed by performing downcasting.
5.	Parent p = new Parent()	Parent p = new Child() Child c = (Child)p;

Java Garbage Collection

In java, garbage means unreferenced objects.

Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.

Advantage of Garbage Collection

- It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.
- It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

How can an object be unreferenced?

There are many ways:

- By nulling the reference
- By assigning a reference to another
- By anonymous object etc.

---

1) By nulling a reference:

```
1. Employee e=new Employee();
2. e=null;
```

2) By assigning a reference to another:

```
1. Employee e1=new Employee();
2. Employee e2=new Employee();
3. e1=e2;//now the first object referred by e1 is available for garbage collection
```

3) By anonymous object:

```
1. new Employee();
```

finalize() method

The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in Object class as:

```
1. protected void finalize(){}
```

Note: The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects).

gc() method

The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.

```
1. public static void gc(){}
```

Note: Garbage collection is performed by a daemon thread called Garbage Collector(GC). This thread calls the finalize() method before object is garbage collected.

Simple Example of garbage collection in java

```
1. public class TestGarbage1{
2.  public void finalize(){System.out.println("object is garbage collected");}
3.  public static void main(String args[]){
4.   TestGarbage1 s1=new TestGarbage1();
5.   TestGarbage1 s2=new TestGarbage1();
6.   s1=null;
7.   s2=null;
8.   System.gc();
9.  }
10.}
```

```
object is garbage collected
      object is garbage collected
```

Java Enums

The **Enum in Java** is a data type which contains a fixed set of constants.

It can be used for days of the week (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, and SATURDAY) , directions (NORTH, SOUTH, EAST, and WEST), season (SPRING, SUMMER, WINTER, and AUTUMN or FALL), colors (RED, YELLOW, BLUE, GREEN, WHITE, and BLACK) etc. According to the Java naming conventions, we should have all constants in capital letters. So, we have enum constants in capital letters.

Java Enums can be thought of as classes which have a fixed set of constants (a variable that does not change). The Java enum constants are static and final implicitly. It is available since JDK 1.5.

Enums are used to create our own data type like classes. The **enum** data type (also known as Enumerated Data Type) is used to define an enum in Java. Unlike C/C++, enum in Java is more *powerful*. Here, we can define an enum either inside the class or outside the class.

Java Enum internally inherits the *Enum class*, so it cannot inherit any other class, but it can implement many interfaces. We can have fields, constructors, methods, and main methods in Java enum.

Points to remember for Java Enum

- Enum improves type safety
- Enum can be easily used in switch
- Enum can be traversed
- Enum can have fields, constructors and methods
- Enum may implement many interfaces but cannot extend any class because it internally extends Enum class

Simple Example of Java Enum

```
1. class EnumExample1{
2. //defining the enum inside the class
3. public enum Season { WINTER, SPRING, SUMMER, FALL }
4. //main method
5. public static void main(String[] args) {
6. //traversing the enum
7. for (Season s : Season.values())
8. System.out.println(s);
9. }}
```

Autoboxing and Unboxing

① An autoboxing and unboxing are specially used for stack memory management. In that stack memory management, there are two types of operation. (push and pop)

- The autoboxing and unboxing the process of handling primitive datatypes in collection.
- We can use this features to convert primitive datatypes to wrapper class types automatically.

Program

```
import java.io.*;
import java.util.*;
Class Autounex
  {
    public static void main (String args[])
      {
        Stack<Integer> stackobj = new Stack<Integer>();
        Stack obj.push(10);
        Stack obj.push(20);        } Autoboxing
    int stacksum = stackobj.pop() + stackobj.pop();   ] Unboxing
    System.out.println ("Stacksum=" + stacksum);
      }
  }
```

O/P

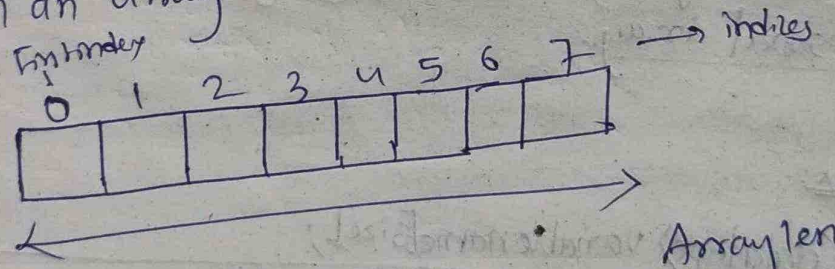
Stacksum=30

END OF CHAPTER ..... Chitu Pradhan.

Unit - III 102 20/10/2020

Array

- Array is an object which contains elements of similar datatype.
- The elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.
- Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.
- Array is an object of a dynamically generated class. Java array inherits the Object class, and implements the Serializable as well as Cloneable interfaces. We can store primitive values or objects in an array in Java.



Array length is 8.

→ Array is a collection of similar types of data elements stored in a contiguous memory location.

Advantages of Array

- **Code Optimization:-** It makes the code optimized, we can retrieve or sort the data efficiently.
- **Random access:-** We can get any data located at an index position.

Disadvantages

- Size limit:- We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically.

Types of Array

- There are two types of array.
  (I) Single Dimensional Array.
  (II) Multi Dimensional Array.

(I) Single Dimensional Array:-

- A list of items can be given one variable name using only one subscript and such a variable is called a single-subscripted variable or a one-dimensional array.

Syntax:-

<datatype> variable name[size];

datatype a[ ] = new datatype[ ];

Program

```
import java.io.*;
import java.util.*;

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

```
int a[] = new int[4];
for(int i=0; i<4; i++)
    {
        a[i]= sc.nextInt();
    }
System.out.println(" Display Element = ");
  for(int i=0; i<4; i++)
    {
        System.out.println("Value = "+a[i]);
    }
}
}
```

O/p

```
java demo
1
2
3
53
Display elements =
value = 1
value = 2
value = 3
value = 53
```

(ii) Multidimensional Array:-

- A multidimensional Array is an array with more than two dimensional i.e, rows and coloumns.

Syntax

```
int variablename[][] = new int[][];
```

Program

```
import java.io.*;
import java.util.*;
class array2D
{
    public static void main(String args[])
    {
        Scanner sc=new Scanner(System.in);
        int a[][] = new int[3][3];
        System.out.println("Enter array elements=");
        for(int r=0; r<3; r++)
        {
            for(int c=0; c<3; c++)
            {
                a[r][c]= sc.nextInt();
            }
        }
System.out.println("Display an array element in matrix
                    format=");

for(int r=0; r<3; r++)
  {
```

```
for (int c=0; c<3; c++)
  {
    System.out.println(a[r][c]+"  ");
  }
  System.out.println();
 }
}
}
```

O/P

Enter array element=

12
13
14
15
16
17
18
19
20

Display an array element in matrix format=

12	13	14
15	16	17
18	19	20

For displaying the elements

12	15	18
13	16	19
14	17	20

Just change in the last line

```
{ System.out.println(a[c][r]+" ");
}
```

(iii) Jagged Array

- Jagged array is array of arrays such that members arrays can be of different sizes, i.e, we can create a 2-D arrays but with variable number of colourms in each row. These types of arrays are also known as Jagged arrays.

Program

```
import java.io.*;
class Mainprog
{
    public static void main(String args[])
    {
        // Declaring 2-D array with 2 rows
            int arr[][] = new int[2][];

        // Making the above array jagged

        // First row has 3 coloumns
            arr[0] = new int[3];

        // Second row has 2 coloumns
            arr[1] = new int[2];

        // Intializing array
            int count=0;

    for(int i=0; i<arr.length; i++)
        for (int j=0; j<arr[i].length ; j++)
```

```
        arr[i][j] = count++;
System.out.println("Contents of 2D Jagged Array");
for (int i=0; i<arr.length; i++)
  {
    for (int j=0; j<arr[i].length; j++)
      System.out.println(arr[i][j] + " ");
      System.out.println();
  }
}.
```

O/P

```
Contents of 2D Jagged Array
0 1 2
3 4
```

String

String is basically an object that represents sequence of characters.

Eg:- `char arr[] = {1,2,4};`

```
char ch = {'j','a','v','a'};
String s = new String(ch);
int b = new int[10];
```

① Method:

```
String obj = " Hello";
System.out.println(" value = " + obj);
```

O/p :- Hello

② Method

```
String obj 1 = new String(" Computer science");
system.out.println(" Value = " + obj1);
```

O/p :- Computer Science

① Program

```
import java.io.*;
import java.util.*;
public class Javaapplication
{
    public static void main(String args[])
    {
        String obj;
        Scanner sc = new Scanner(System.in);
        System.out.println(" Enter String value = ");
        obj = sc.nextline();
        system.out.println("Value = " + obj);
    }
}
```

O/p

Enter String value = Hello Word
Value = HelloWorld

22/11/2020

String Methods:-

- The string class defines a number of methods that allow us to accomplish a variety of string manipulation tasks.
- There are some string methods commonly used i.e,

Method call	Task performed
s2 = s1.toLowerCase;	Small letter
s2 = s1.toUpperCase;	Capital letter
s2 = s1.replace('x', 'y');	Replace all appearances of x with y.
s2 = s1.trim();	Remove whitespaces at the beginning and end of string.
s1.equals(s2).	Returns 'true' if s1 is equal to s2.
s1.equalsIgnoreCase(s2)	Returns 'true' if s1=s2, ignoring the case of characters.
s1.length()	Gives the length of s1.
s1.charAt(n)	Gives n^{th} character of s1.
s1.compareTo(s2)	Returns negative if s1 < s2, positive if s1 > s2, zero, if s1 = s2.
s1.concat(s2)	Concatenates s1 and s2
s1.substring(n)	Gives substring starting from n^{th} character
s1.substring(n, m)	Gives substring starting from n^{th} character up to m^{th} (not including m^{th})
String.valueof(p)	Creates a string object of the parameter p (simple type or object)
p.toString()	Creates a string representation of the object p.
s1.indexof('x')	Gives the position of the first occurence of 'x' in the string s1.

s1.indexof('x', n)	Gives the position of 'x' that occurs after nth position in the string s1.
String.Valueof (Variable)	Converts the parameter value to string representation.

Programs

① W.A.P to display a result enter the data of a string.

→
```
import java.io.*;
import java.util.*;
class StringDemo
{
  public static void main(String args[])
  {
    String obj = new String();
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter String Value=");
    obj = sc.next();
    System.out.println("String Result=" +obj);
  }
}
```

O/P
```
Enter String value=
Hello
String Result = Hello
```

② W.A.P to display the uppercase of a data with String method.

→
```
import java.io.*;
import java.util.*;
```

```
class StringDemo
{
    public static void main(String args[])
    {
        String obj = new String();
        String obj1 = new String();
        Scanner sc = new Scanner(System.in);
        System.out.println(" Enter String value = ");
        obj = sc.next();
        obj1 = obj.toUpperCase();
        System.out.println(" String Result = "+obj);
        System.out.println(" String Result uppercase = "+obj1);
    }
}
```

O/P

```
Enter String value =
hellooo
String Result = hellooo
String Result Uppercase = HELLOOO
```

3) W.A.P to display the Lowercase of a data with string method.

```
import java.io.*;
import java.util.*;
class StringLower
{
    public static void main(String args[]);
```

```
{
    String obj= new String();
    String obj1 = new String();
  Scanner sc= new Scanner (System.in);
  System.out.println(" Enter String value = ");
  obj= sc.next();
  obj1 = obj.toLowercase();
  System.out.println(" String Result = "+obj);
  System.out.println(" String Result Lowercase = "+obj1);
  }
}
```

O/P

```
Enter String value =
HELLLOOO
String Result= HELLLOOO
String Result Lowercase = hellooo.
```

11) W.A.P to display both lowercase and concat string method data in a program.

→

```
import java.io.*;
import java.util.*;
class StringDemo
{
  public static void main(String args[])
  {
    String obj= new String();
    String obj1 = new String();
    String obj2 = new String();
```

```
Scanner sc = new Scanner (System.in);
System.out.println(" Enter String value=");
obj = sc.next();
obj1 = obj.toLowercase();
obj2 = obj.concat(obj1);
System.out.println(" String Result = " + obj);
System.out.println(" String Result Lowercase = " + obj1);
System.out.println(" String Result Concat Value = " + obj2);
}
}
```

O/P

```
Enter String value=
COMPUTER
String Result = COMPUTER
String Result Lowercase = computer
String Result Concat Value = COMPUTERcomputer
```

5) W.A.P to display both Lowercase, concat with lengthsize of the String method data.

```
import java.io.*;
import java.Util.*;
class StringDemo
{
  public static void main(String args[])
  {
    String obj = new String();
    String obj1 = new String();
    String obj2 = new String();
```

```
Scanner sc= new Scanner( System.in);
System.out.println(" Enter String value=");
obj= sc.next();
obj1= obj.toLowerCase();
obj2 = obj.concat (obj1);
int leng = obj.length();
System.out.println(" String Result= "+obj);
System.out.println(" String Result LowerCase = "+obj1);
System.out.println(" String Result Concat Value="+ obj2);
System.out.println(" string Length of obj= "+len);
}
}
```

o/p

```
Enter String value=
COMPUTER
String Result= COMPUTER
String Result LowerCase = computer.
String Result Concat value= COMPUTERcomputer.
String Length of obj = 8.
```

6) W.A.P to enter the data whether the two are equal or not by using String method.

->

```
import java.io.*;
import java.util.*;
class Stringequal
{
    public static void main (String args[])
    {
```

```
String obj = new String();
String obj1 = new String();
Scanner sc = new Scanner(System.in);
System.out.println(" Enter two String value = ");
obj = sc.next();
obj1 = sc.next();
if (obj.equals(obj1))
 {
   System.out.println(" Strings are equal. ");
 }
 else
  {
    System.out.println(" Strings are not equal. ");
  }
}
}
```

O/P

```
Enter two String value =
Computer
Monitor
Strings are not equal.
```

7) W.A.P to display the districts and array length with the help of string methods

```
import java.io.*;
import java.util.*;
class stringdistricts
 {
```

```
public static void main(String args[])
{
  String arr[]= {"Khorda","Cuttack","Anugul","Ganjam"};
  /*Scanner sc= new Scanner(System.in);
    System.out.println("Enter two String value=");
    obj=sc.next();
    obj1= sc.next(); */

  int size= arr.length;

 for(int i=0; i<size; i++)
   {
     System.out.println("String array values = "+
                 arr[i]+" Array length= "+size);
   }
 }
}
```

O/P

```
String array values = Khorda     Array length = 4
String array values = Cuttack    Array length = 4
String array values = Anugul     Array length = 4
String array values = Ganjam     Array length = 4
```

Q. W.A.P to enter the district and display the district with ascending order with String method.

```
import java.io.*;
import java.util.*;
class DistrictSort
{
    public static void main(String args[])
    {
        Scanner sc = new Scanner(System.in);
        String dist[] = new String[5];
        System.out.println(" Enter 5 districts:");
        for(int i=0; i<5; i++)
        {
            dist[i] = sc.next();
        }
    String temp=null;
    for(int i=0;i<5;i++)
    {
        for(int j=i+1; j<5;j++)
        {
            if(dist[j].compareTo(dist[i]<0)
            {
                temp = dist[i];
                dist[i]= dist[j];
                dist[j] = temp;
            }
        }
    }
```

```
System.out.println("District Sorting:");
for(int i=0; i<5; i++)
  {
    System.out.println(dist[i]);
  }
 }
}
```

O/P

```
Enter 5 districts:
BHUBANESWAR
CUTTACK
ANUGUL
JAGATSINGHPUR
GANJAM

District Sorting:
ANUGUL
BHUBANESWAR
CUTTACK
GANJAM
JAGATSINGHPUR
```

String Buffer Class

- StringBuffer is a peer class of String.
- String creates strings of fixed length, StringBuffer creates strings of flexible length that can be modified in terms of both length and content.

- Some commonly used String Buffer methods.

Method	Task
s1.setCharAt(n, 'x')	Modifies the nth character to x
s1.append(s2)	Appendes the string s2 to s1 at the end.
s1.insert(n, s2)	Inserts the string s2 at the position n of the string s1.
s1.setLength(n)	Sets the length of the string s1 to n. If n<s1.length() s1 is truncated. If n>s1.length() zeros are added to s1

Programs

① W.A.P to display the StringBuffer value.

→
```
import java.io.*;
import java.util.*;
class StringBuf
{
  public static void main(String args[])
  {
    StringBuffer obj = new StringBuffer("Object Language");
    System.out.println("Display StringBuffer value=" +obj);
  }
}
```

O/P

Display StringBuffer value = Object language

Stream classes

- Stream class contains a large no. of predefined classes which is referring to java.io.package for processing all types of data. It is divided into two types such as

  ① ByteStream class
  ② CharacterStream class

① ByteStream class:—

- ByteStream classes have being designed to provide functional features for creating and manipulating streams and files for reading and writing bytes.
- Since, the streams are unidirectional, they can transmit bytes in only one direction, thatswhy it is divided into two types such as

  (i) InputStream class
  (ii) OutputStream class

(i) InputStream class:—

- InputStream classes that are used to read 8-bit bytes that include a super class known as InputStream and a no. of subclasses for supporting various input related functions. This subclasses are

  (i) FileInputStream
  (ii) ObjectInputStream
  (iii) StringBufferInputStream etc.

- The superclass InputStream is an abstract class and we cannot create instances of this class, thatswhy this class defines some methods which are related to input function such as.

(i) reading Bytes:- read(), read(byte b[]), read(byte b[], int n, int m)

read():- It reads a byte from the InputStream.

read(byte b[]):- It reads an array of bytes into b.

read(byte b[], int n, int m):- It reads m-bytes into b starting from n^{th} byte.

(ii) Closing Streams:- In this method is used to close the InputStream by taking a method close().

(iii) Marking Position in Stream:- This technique is used by a method as named as available(). So, this method gives no. of bytes available in the InputStream.

(iv) Skipping ahead in a stream:- This technique is used a method named as skip(n). It skips over n-bytes from the InputStream.

(ii) OutputStream class:-

- OutputStream classes are derived from the base class i.e, OutputStream.

- The several subclasses of the OutputStream can be used for performing the output operation. Similarly the OutputStream classes contains some subclasses such as

  (i) FileOutputStream
  (ii) ObjectOutputStream
  (iii) StringBufferOutputStream

- This Stream class includes some methods such as

(i) WriteByte():- It is used some methods such as write(), write(byte b[]), write(byte b[], int n, int m)

write():- It writes a byte through the OutputStream.

write(byte b[]):- It writes all bytes in the array b through the OutputStream.

write(byte b[], int n, int m):- It writes m-bytes from array b starting from the nth byte.

(ii) ClosingStream:- It is used a method like ~~close()~~ close(), which is closed the OutputStream.

(iii) FlushingStream:- It is used a method which flushes the OutputStream.

① Write a program to read and write bytes.

```
import java.io.*;
class Byteex
{
    public static void main(String args[])
    {
```

```
byte b[]= {'H', 'e', 'L', 'L', 'O'};

  try
  {
      FileOutputStream fo = new FileOutputStream
                                ("file1.txt");

      fo.write(b);
      fo.close();
  }
catch(Exception e)
  {
      System.out.println(e);
  }

}

}
```

O/P

Hello

② Character Stream classes:-

- CharacterStreams can be used to read and write 16-bit unicode characters.
- There are two kinds of CharacterStream classes such as
  - (i) ReaderStream Classes
  - (ii) WriterStream Classes

(i) ReaderStream classes:-

- ReaderStream Classes are designed to read character from the files.
- Reader class is the base class for all other Classes like
  - (i) FileReader, StringReader, BufferReader, etc

This classes are functionally very similar to the InputStream classes except InputStream use bytes as their fundamental unit of information.

- The Reader class contains methods that are identical to those available in the InputStream class.

(ii) WriterStream classes:-

- The WriterStream classes is an abstract class which acts as a base class for all the other WriterStream classes, which is a collection of some subclasses such as
  - (i) FileWriter, BufferWriter, StringWriter, etc.
- This base class provides support for all output operations by defining methods that are identical to those in OutputStream classe.

Program

```
import java.io.*;
class CopyCharacters
{
    public static void main(String args[])
    {
        File inFile = new File("file1.text");
        File outFile = new File("output.txt");
        try
        {
            FileReader ins = new FileReader(inFile);
            FileWriter outs = new FileWriter(outFile);
            int ch;
            while((ch=ins.read()) != -1)
            {
                outs.write(ch);
            }
            ins.close();
            outs.close();
        }
    catch (IOException e)
        {
            System.out.println(e);
            System.exit(-1);
        }
    }
}
```

Compressing and Decompressing File

- The DeflaterOutputStream and InflaterInputStream classes provide mechanism to compress and decompress the data in the deflate compression format.

① DeflaterOutputStream Class :-

- The DeflaterOutputStream class is used to compress the data in the deflate compression format.
- It provides facility to the other compression filters, such as GZIPOutputStream.

Program

```
import java.io.*;
import java.util.zip.*;
class Compress
{
    public static void main(String args[]) throws IOException
    {
        FileInputStream fis = new FileInputStream
                                ("hello.txt");
        FileOutputStream fos = new FileOutputStream
                                ("outhello.txt");
        DeflaterOutputStream dos = new DeflaterOutputStream
                                (fos);
        int data;
        while( (data = fis.read() ! =-1 )
            {
              dos.write(data);
            }
```

```
    fis.close();
    dos.close();
  }

}
```

② InflaterInputStream class:-

- The InflaterInputStream class is used to decompress the file in the deflate compression format.
- It provides facility to the other decompression filters, such as GZIPInputStream class.

Program

```
import java.io.*;
import java.util.zip.*;
class Decompress
  {
    public static void main(String args[]) throws IOException
    {
      FileInputStream fis= new FileInputStream("outhello.txt");
      FileOutputStream fos = new FileOutputStream("Decompress
                                                  file.txt");
      InflaterInputStream iis = new InflaterInputStream(fis);
      int data;
      while ((data = iis.read())! = -1)
        {
          fos.write(data);
        }
      ~~close~~ fos.close();
                iis.close();
    }
}
```

1) PrintStream Class in Java

- A PrintStream adds functionality to another output stream, namely the ability to print representations of various data values conveniently.
- Unlike other output streams, a PrintStream never throws an IOException; instead, exceptional situations merely set an internal flag that can be tested via the checkError method.
- Optionally, a PrintStream can be created so as to flush automatically.
- All characters printed by a PrintStream are converted into bytes using the platform's default character encoding.
- The PrintWriter class should be used in situations that require writing characters rather than bytes.

Constructors and Description

① PrintStream(File file):-

Creates a new print stream, without automatic line flushing, with the specified file.

② PrintStream(File file, String csn):-

Creates a new print stream, without automatic line flushing, with the specified file and charset.

Program

```
import java.io.*;
class PrintStreamTest
  {
    public static void main (String args[]) throws Exception
    {
      FileOutputStream fout = new FileOutputStream ("BSC.txt");
      PrintStream pout = new PrintStream (fout);
      pout.println( 1900);
      pout.println("Hello Java");
      pout.println("Welcome to Java");
      pout.close();
      fout.close();
    }
  }
```

2) PrintWriter class.

- Java PrintWriter class is the implementation of Writer class.
- It is used to print the formatted representation of objects to the text output stream.

Class declaration

Declaration:- java.io.PrintWriter class.

1. public class PrintWriter extends Writer.

Methods of PrintWriter class

Method	Description
void println(boolean x)	It is used to print the boolean value.
void println(char[]x)	It is used to print an array of characters.
void println(int x)	It is used to print an integer.
void flush()	It is used to flushes the stream
void close()	It is used to close the stream

Program

```
import java.io.File;
import java.io.PrintWriter;
class PrintWriterEx
  {
    public static void main(String args[]) throws Exception
    {
      // Data to write on Console using PrintWriter.
  PrintWriter writer = new PrintWriter(System.out);
  writer.write("Hello Students");
  writer.flush();
  writer.close();
// Data to write in file using PrintWriter
  PrintWriter writer1 = new PrintWriter(new File("testout.txt"));
  Writer1.write("Like Java, Spring ...etc");
  writer1.flush();
  writer1.close();
```

End of CHAPTER

... Chitku Pradhan

Managing Errors and Exceptions

Error:-

- Errors are the wrongs that can make a program go wrong.
- An error may produce an incorrect output or may terminate the execution of the program abruptly or even may cause the system to crash.

Types of Errors

- Errors are broadly classified into two categories.
  (1) Compile-time errors.
  (2) Run-time errors.

(1) Compile-Time Errors:-

- All syntax errors will be detected and displayed by the Java compiler and therefore these errors are known as compile-time errors.
- Whenever the compiler displays an error, it will not create the .class file. It is therefore necessary that we fix all the errors before we can successfully compile and run the programs.

Program

```
Class Error1
{
    public static void main (String args[])
    {
        System.out.println(" Hello Java!")   //missing ;
    }
}
```

(2) Run-time errors:-

- A program may compile successfully creating the .class file but may not run properly. such programs may produce wrong results due to wrong logic or may terminate due to errors such as stack overflow.

- Most common run-time errors are:

1) Dividing an integer by zero.
2) Accessing an element that is out of the bounds of an array.
3) Trying to store a value into an array of an incompatible class or type.
4) Trying to cast an instance of a class of one of its subclasses.
5) Passing a parameter that is not in a valid range or value for a method.
6) Trying to illegally change the state of a thread.
7) Attempting to use a negative size for an array.
8) Using a null object reference as a legitimate object reference to access a method or a variable.
9) Converting invalid string to a number.
10) Accessing a character that is out of bounds of a string.

Program

```
import java.io.*;
import java.util.*;
class ExceptEx
{
```

```
public static void main (String args[])
  {
      int a = 4, b;
      b = a/0;
      System.out.println(" Result= "+b);
    }
  }
```

Exceptions

- An exception is a condition that is caused by a run time error in the program.
- When the java interpreter encounters an error such as dividing an integer by zero, it creates an exception object and throws it
- If the exception object is not caught and handled properly, the interpreter will display an error message and terminate the program.
- The purpose of exception handling mechanism is to provide a means to detect and report an "exceptional circumstance" so that appropriate action can be taken. The mechanism suggests incorporation of a separate error handling code that performs the following tasks:

  1) Find the problem (Hit the exception).
  2) Inform that an error has occurred (Throw the exception.
  3) Receive the error information (Catch the exception).
  4) Take corrective actions (Handle the exception).

- The error handling code basically consists of two segments, one to detect errors and to throw exceptions and the other to catch exceptions and to take appropriate actions.

Common Java exceptions

Exception Type	Cause of Exception
ArithmeticException	Caused by math errors such as division by zero.
ArrayIndexOutofBoundsException	Caused by bad array indexes
ArrayStoreException	Caused when a program tries to store the wrong type of data in an array.
FileNotFoundException	Caused by an attempt to access a nonexistent file
IOException	Caused by general I/O failures, such as inability to read from a files.
NullPointerException	Caused by referencing a null object.
NumberFormatException	Caused when a conversion between Strings and number fails.
OutOfMemoryException	Caused when there's not enough memory to allocate a new object.
SecurityException	Caused when an applet tries to perform an action not allowed by the browser's security setting.
StackOverFlowException	Caused when the system runs out of stack space.
StringIndexOutOfBoundsException	Caused when a program attempts to access a nonexistent character position in a string.

Syntax of Exception Handling Code

- The basic concepts of exception handling are throwing an exception and catching it.
- Java uses a keyword try to preface a block of code that is likely to cause an error condition and "throw" an exception.
- A catch block defined by the keyword catch "catches" the exception "thrown" by the try block and handles it appropriately. The catch block is added immediately after the try block.

Syntax

Exception Handling

```
try
{
    ========
    ========
}
catch (TypeOf Exception Object)
{
    ========
    ========
}
```

① **Program**

```
import java.io.*;
import java.util.*;
import java.lang.*;
```

```
class ExceptEx
 {
    public static void main(String args[])
   {
     try
     {
       int a=4,b;
       b= a/0;
      System.out.println(" Result = "+b);
     }
    catch( ArithmeticException e)
     {
      System.out.println(" Number Divided By Zero ", e);
     }

   }
 }
```

② Program

```
import java.io.*;
import java.Util.*;
import java.lang.*;
class ExceptEx
 {
   public static void main( String args[])
    {
      try
       {
         int a=4, b;
         b=a/0;
        System.out.println(" Result = "+b);
```

```
    }
    catch (ArithmeticException e)
    {
      System.out.println("NumberDividedBy zero" +e);
    }
    System.out.println("Hello World");
  }
}
```

- There are two types of Exception
  1. Predefined Exception (Above example)
  2. Userdefined Exception

Multiple catch Statements

- In a program it is possible to have more than one catch statements.

  (In a program one try block will keep but catch block will store more than one.)

Syntax

```
try
{
  ____
}
catch (Exception e)
{
  ____
}
```

```
catch (Exception e)
  {
    ==
    ==
  }
  :
catch (Exception-N e)
  {
    ==
    ==
  }
```

Program

```
import java.io.*;
class Excepex
 {
    public static void main (String args[])
     {
       try
       {
         int a,b,c;
         a = 10;
         b = 10;
         c = a/(a-b);
       System.out.println("Div="+c);
       }
     catch (ArithmeticException e)
       {
         System.out.println(e);
       }
```

```
catch( ArrayStoreException e)
  {
    System.out.println(e);
  }
catch( Exception e)
  {
    System.out.println(e);
  }
}

}
```

Finally Block

- The finally block follows a try block or a catch block. A finally block of code always executes irrespective of occurence of an Exception
- Using a finally block allows you to run any cleanup-type statements that you want to execute no matter what happens in the protected code.
- A finally block appears at the end of the catch blocks and has the following syntax.

Syntax

```
try
  {
    ====
  }
catch(    )
  {
    ====
  }
finally
  {
    ====
  }
}
```

Program

```
import java.io.*;
class ExcepEx
{
   public static void main(String args[])
   {
      try
      {
         int a,b,c;
         a=10;
         b=10;
         c= a/(a-b);
      System.out.println("Div = "+c);
      }

      catch(Exception obj)
      {
         System.out.println(obj);
      }

      finally
      {
         System.out.println(" Hello Students");
      }
   }
}
```

O/P

java.lang.ArithmeticException: / by zero Hello Students

Throwing Exception / Throwing our own Exception

- Exception is a subclass of Throwable and therefore MyException is a subclass of Throwable class.

An object of a class that extends Throwable can be thrown and caught.

Syntax

```
throw new Throwable subclass;
```

Eg:-

```
throw new ArithmeticException();
```

Program

```
import java.io.*;
import java.lang.*;
class Myexception extends Exception
 {
   Myexception(String msg)
    {
      super(msg);
    }
 }
class TestEx
 {
   public static void main(String args[])
    {
```

// Userdefined Exception

```
float x = 5, y=1000;

try
  {
    float a = x/y;
   if (a < 0.01)
    {
      throw new Myexception("Number is too small");
    }
  }
catch( Myexception obj)
  {
    System.out.println(obj.getMessage());
  }
}
}
```

O/P

Number is too small

Suppose we will not write .getMessage() just write S.O.p(obj);

O/P

Myexception: Number is too small

Multithreading Programming

Thread:-

- Thread is a facility to allow multiple activities within a single process
- Thread is referred as lightweight process
- A thread is a series of executed statements.
- Each thread has its own program counter, stack and local variables.
- A thread is a nested sequence of method calls.
- Thread shares memory space, files and per-process state.

~~Multithreaded programs:-~~

Single-threaded programs:

- A thread is similar to a program that has a single flow of control.
- It has a beginning, body, and an end, and executes commands, sequentially.
- All main programs in our earlier examples are called single threaded program.

  Eg:- Addition, substraction, exception, autoboxing and unboxing each and every types of programs are called single threaded programs.

Multithreaded programs:

- Multithreading is a conceptual programming paradigm where a program is divided into two or more subprograms, which can be implemented at the same time in parallel.

  Eg: One subprogram can display an animation on the screen while another may build the next animation to be displayed

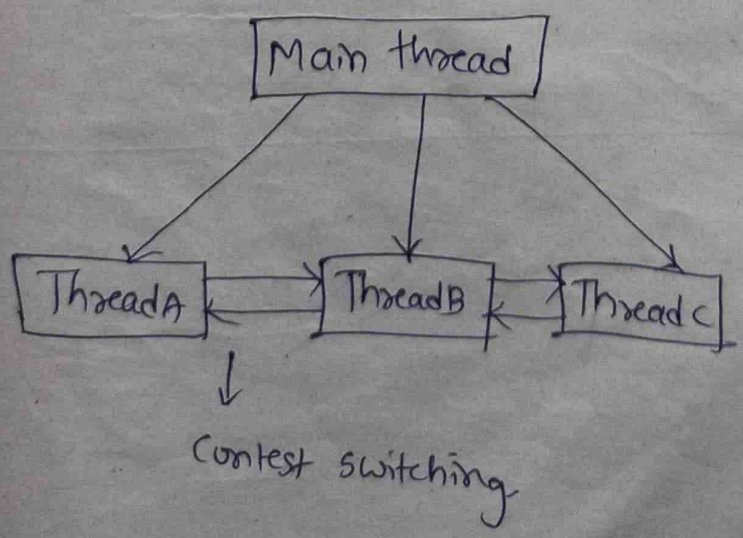
  - This is something similar to dividing a task into subtasks and assigning them to different people for execution independently and simultaneously.

Single thread program syntax

```
Class ABC
{
    ==
    =

}
```

Multithread program syntax



Difference between Multithreading and Multitasking

Multithreading	Multitasking
① It is a programming concept in which a program or a process is divided into two or more subprograms or threads that are executed at the same ~~time~~ time in parallel.	① It is an operating system concept in which multiple tasks are performed simultaneously.
② It supports execution of multiple parts of a single program simultaneously.	② It supports execution of multiple programs simultaneously.
③ The processor has to switch between different parts or threads of a program.	③ The processor has to switch between different programs or processes.
④ It is highly efficient.	④ It is less efficient in comparison to multithreading.
⑤ A thread is the smallest unit in multithreading.	⑤ A program or process is the smallest unit in a multitasking environment.
⑥ It helps in developing efficient programs.	⑥ It helps in developing efficient operating systems.
⑦ It is cost-effective in case of context switching.	⑦ It is expensive in case of context switching.

Creating Threads

- Creating threads in Java is simple.
- Threads are implemented in the form of objects that contain a method called run().

run() :- The heart and soul of any thread.

- It makes up the entire body of a thread and is the only method in which the thread's behaviour can be implemented.

Syntax

```
public void run()
{
    ========
}
```

- ~~Creating~~ The run() method should be invoked by an object of the concerned thread.
- This can be achieved by creating the thread and initiating it with the help of another thread method called start().

- A new thread can be created in two ways
  1. By creating a thread class.
  2. By converting a class to a thread.

① By creating a thread class:

Defines a class that extends Thread class

② By converting a class to a thread:-

A class that implements Runnable interface.
Thread class object predefined.

- The Runnable interface has only one method, run(), that is to be defined in the method with the code to be executed by the thread.

Thread class program

```
import java.io.*;
import java.lang.*;
class ThreadA extends Thread
  {
    public void run()
      {
        for (int i=1; i<=5; i++)
          {
            System.out.println("Thread A Executes = "+i);
          }
        System.out.println("Thread A Exit");
      }
  }
```

```
class ThreadB extends Thread
  {
    public void run()
      {
        for (int j=1; j<=5 ;j++)
          {
            System.out.println("ThreadB Execute = "+j);
          }
      System.out.println("ThreadB Exit");
    }
}
class ThreadC extends Thread
  {
    public void run()
      {
        for(int k=1; k<=5; k++)
          {
            System.out.println("ThreadC Execute ="+k);
          }
      System.out.println("ThreadC Exit");
    }
}
```

```
class Mainthread
  {
    public static void main (String args[])
    {
      ThreadA obj1 = new ThreadA();
      ThreadB obj2 = new ThreadB();
      ThreadC obj3 = new ThreadC();
      obj1.start();
      obj2.start();
      obj3.start();
    }
}
```

Output

```
Javac Mainthread.java
Java Mainthread
ThreadA Execute = 1
ThreadA Execute = 2
ThreadB Execute = 1
ThreadB Execute = 2
ThreadB Execute = 3
ThreadC Execute = 1
ThreadB Execute = 4
ThreadA Execute = 3
ThreadB Execute = 5
ThreadB Exit
ThreadC Execute = 2
ThreadA Execute = 4
ThreadC Execute = 3
ThreadC Execute = 4
ThreadA Execute = 5
ThreadC Execute = 5
ThreadC Exit
ThreadA Exit
```

Output will come different.
By compiling the program when n times

Extending the Thread class

- We can make our class runnable as thread by extending the class java.lang.Thread.
- This gives us access to all the thread methods directly.
- It includes the following steps:

① Declare the class as extending the Thread class.

② Implement the run() method that is responsible for executing the sequence of code that the thread will execute.

③ Create a thread object and call the start() method to initiate the thread execution.

Declaring the class

- Thread class can be extended as follows

```
class MyThread extends Thread
{
    ____
    ____
    ____
}
```

Implementing the run() method

```
public void run()
{
    ____
    ____
    ____
}
```

Starting New Thread

- To actually create and run an instance of our thread class we must write.

```
MyThread aThread = new MyThread();
aThread.Start();
```

Example of using the Thread class

- The use of Thread class for creating and running threads in an application.
The statements like

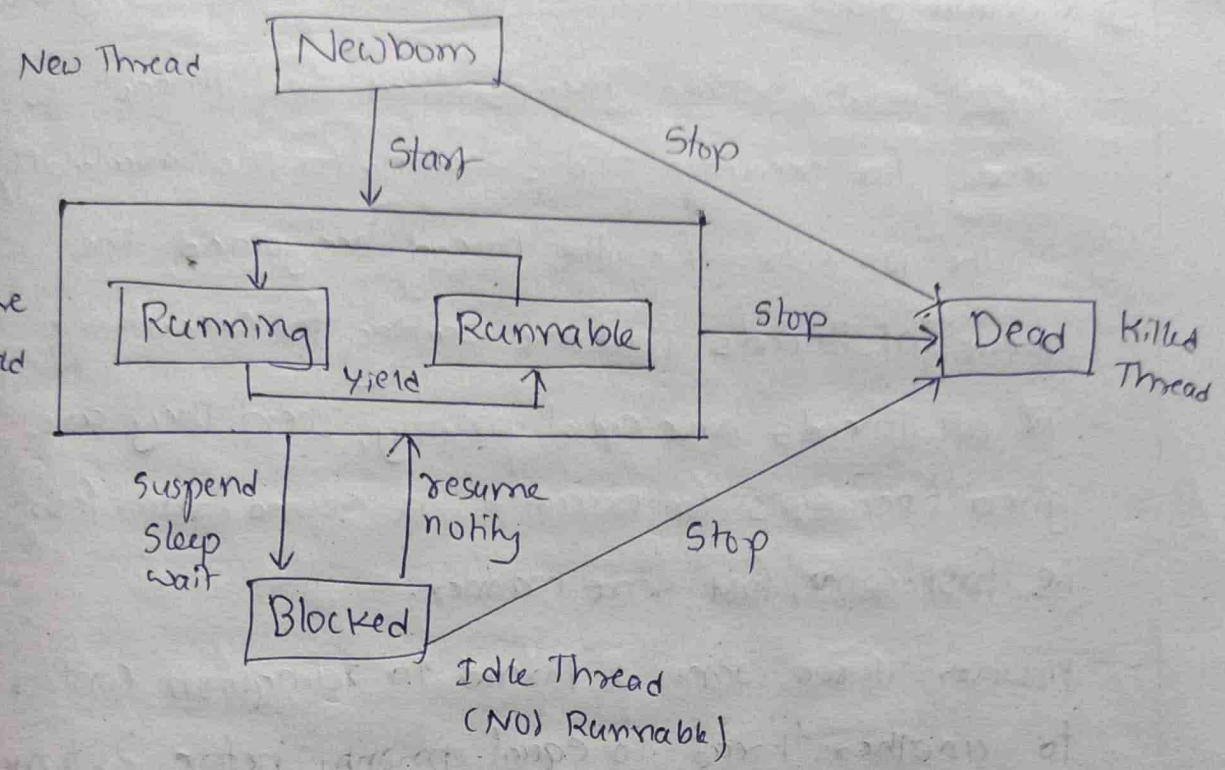
```
new A().Start();
```

Life Cycle of a Thread

- During the lifetime of a thread. There are many states it can enter. They include

1. Newborn State
2. Runnable state
3. Running State
4. Blocked state
5. Dead State.

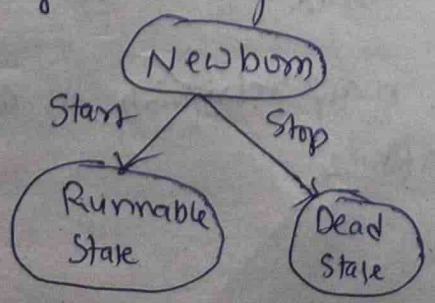
- A thread is always in one of these five states. It can move from one state to another via a variety of way.



State Transition diagram of a thread

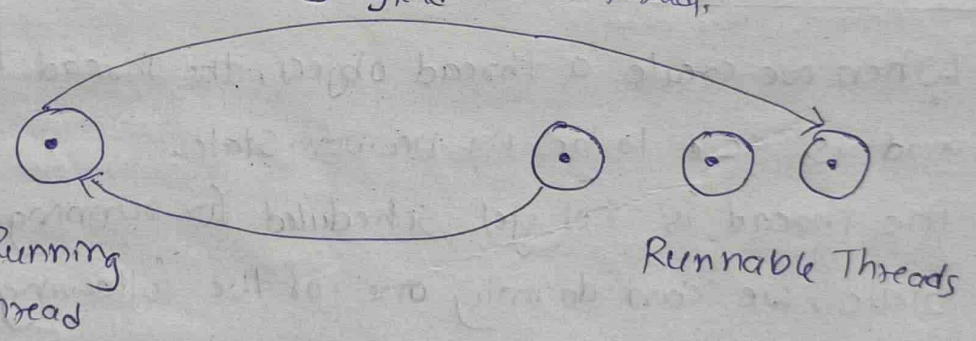
① Newborn State

- When we create a thread object, the thread is born and is said to be in newborn state.
- The thread is not yet scheduled for running. At this state, we can do only one of the following things with it.
  - Schedule it for running using start() method.
  - Kill it using stop() method.
- If scheduled it ~~for running~~ moves to the runnable state. If we attempt to use any other method at this stage, an exception will be thrown.



② Runnable State:-

- The runnable state means that the thread is ready for execution and is waiting for the availability of the processor. i.e, the thread has joined the queue of threads that are waiting for execution.
- If all threads have equal priority, then they are given time slots for execution in round robin fashion, i.e, first-come, first-serve manner.
- However, if we want a thread to relinquish control to another thread to equal priority before its turn comes, we can do so by using the yield() method.
- A yield() method is a static method of Thread class and it can stop the currently executing thread and will give a chance to other waiting threads.

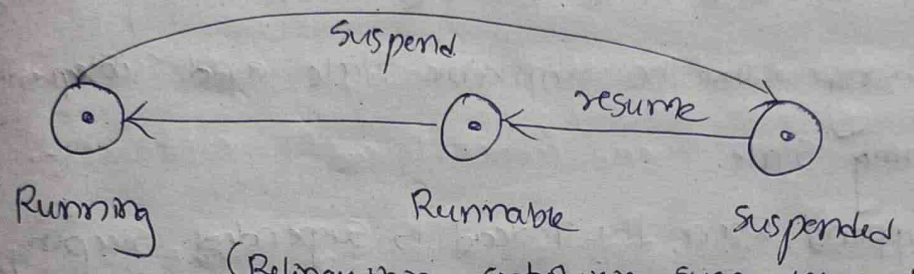


Relinquishing control using yield() method

③ Running state:

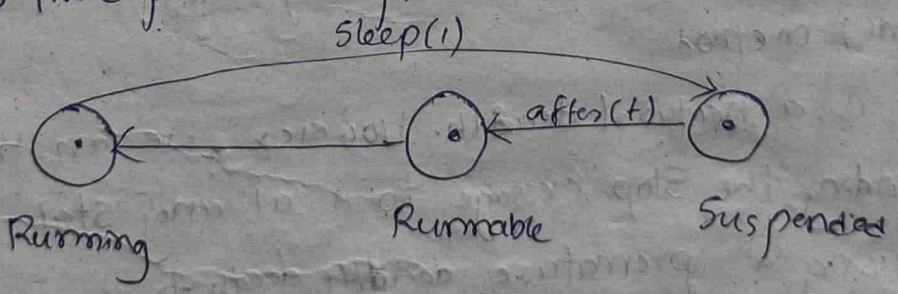
- Running means that the processor has given its time to the thread for its execution. The thread runs until it relinquishes control on its own or it is preempted by a higher priority thread.
- A running thread may relinquish its control in one of the following situations:

1. It has been suspended using suspend() method.
   - A suspended thread can be revived by using the resume() method.
   - This approach is useful when we want to suspend a thread for some time due to certain reason, but do not want to kill it.



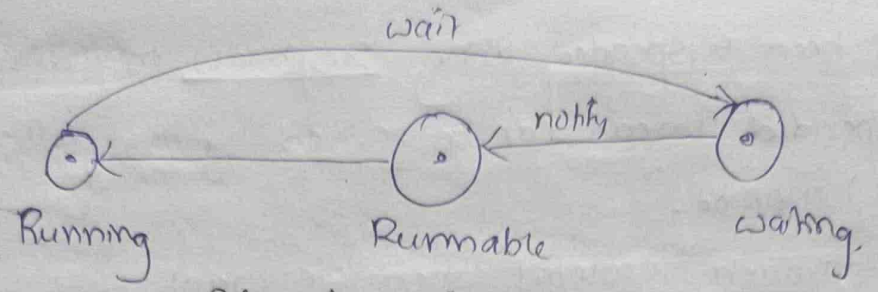
(Relinquishing control using suspend() method)

2. It has been made to sleep. We can put a thread to sleep for a specified time period using the method sleep(time) where time is in milliseconds.
   - This means that the thread is out of the queue during this time period.
   - The thread re-enters the runnable state as soon as this time period is elapsed.



(Relinquishing control using sleep() method)

3. It has been told to wait until some even occurs. This is done using the wait() method. The thread can be scheduled to run again using the notify() method.



Relinquishing control using wait() method

(4) Blocked state:-

- A thread is said to be blocked when it is prevented from entering into the runnable state and subsequently the running state.
- This happens when the thread is suspended, sleeping, or waiting in order to satisfy certain requirements.
- A blocked thread is considered "not runnable" but not dead and therefore fully qualified to run again.

(5) Dead state:-

- Every thread has a life cycle. A running thread ends its life when it has completed executing its run() method.
- It is a natural death. However, we can kill it by sending the stop message to it at any state thus causing a premature ~~death~~ death to it.
- A thread can be killed as soon it is born, or while it is running, or even when it is "not runnable" (blocked) condition.

Thread Priority

- Thread Priority affects the order in which it is schedule for running the threads that we have the same priority are given equal treatment the java schedular and they share the processor on a first come first served.
- Java allows us to set the priority of a thread by using the setPriority().
- It is denoted by a syntax:

  Threadname.setPriority(int num);

- Here, the int num is an integer type value to which the threads Priority is set.
- The Thread class defines several Priority constants such as

  1. MIN_PRIORITY = 1
  2. NORM_PRIORITY = 5
  3. MAX_PRIORITY = 10

- The integer value may assume one of these constants or any value between 1 and 10, but the default setting is NORM_PRIORITY.

Program

```
import java.io.*;
class TestMulti extends Thread
 {
    public void run()
     {
        System.out.println("running thread name is:"+
                Thread.currentThread().getName());
        System.out.println("running thread priority is:"
            + Thread.currentThread().getPriority());
     }
 }

class TestMultiPriority1
 {
    public static void main(String args[])
     {
        TestMulti m1 = new TestMulti();
        TestMulti m2 = new TestMulti();
        TestMulti m3 = new TestMulti();
    m1.setPriority(Thread.MIN_PRIORITY);
    m2.setPriority(Thread.MAX_PRIORITY);
    m3.setPriority(Thread.NORM_PRIORITY);
    m2.start();
    m1.start();
    m3.start();
     }
}
```

O/P

```
java TestMultiPriority1
running thread name is : Thread-1
running thread name is : Thread-2
running thread priority is : 5
running thread name is : Thread-0
running thread priority is : 10.
running thread priority is : 1.
```

Implementing Runnable Interface (2nd way of Thread class)

- The Runnable interface declares the run(), i.e, required for implementing Threads in our programs.
- This program can be programed in following steps. Such as

① Declare the class as implementing the Runnable interface.

② Implement the run().

③ Create a Thread by defining an object i.e, instanticated from this Runnable class as the target of the Thread.

④ Call the Threads start() to run the Thread.

Program

```
import java.io.*;
import java.lang.*;
class Runex implements Runnable
  {
    public void run()
      {
        for(int i=1; i<=5; i++)
          {
            System.out.println("Runnable Thread="+i);
          }
      System.out.println("Exit Runnable");
      }
  }

class Runmain
  {
    public static void main(String args[])
      {
        Runex objr = new Runex();
        Thread objt = new Thread(objr);
        objt.start();
      }
  }
```

O/P

Runnable Thread=1
Runnable Thread=2
Runnable Thread=3
Runnable Thread=4
Runnable Thread=5
Exit Runnable

Program

```
import java.io.*;
class Examplerun implements Runnable
{
    public void run()
    {
        System.out.println("Thread has ended");
    }
}
class Examplecclass
{
    public static void main(String args[])
    {
        Examplerun ex = new Examplerun();
        Thread t1 = new Thread(ex);
        t1.start();
        System.out.println("Hi");
    }
}
```

O/p

```
Java Examplecclass
Hi
Thread has ended.
```

Synchronization in Java

- Synchronization in java is the capability to control the access of multithreads to any shared resource.

Eg:- The method that will read information from a file and the method that will update the same file may be declared as schronized.

Syntax:

```
Synchronized void update()
{
  ==
}
```

Why we use Synchronization

- The synchronization is mainly used to
  1. To prevent thread interference.
  2. To prevent consisting problem.

Types of Synchronization

There are two types of Synchronization
1. Process Synchronization
2. Thread Synchronization

~~1. Thread Synchronization~~

~~There~~

Program

```
import java.io.*;
class Table
{
    synchronized void printTable(int n)
    {
        for( int i=1; i<=5; i++)
        {
            System.out.println(n*i);
            try
            {
                Thread.sleep(400);
            }
            catch( Exception e)
            {
                System.out.println(e);
            }
        }
    }
}
class MyThread1 extends Thread
{
    Table t;
    MyThread1 (Table t1)
    {
```

```
        this.t = t1;
    }
    public void run()
    {
        t.printTable(5);
    }

}
class MyThread2 extends Thread
{
    Table t;
    MyThread2(Table t2)
    {
        this.t = t2;
    }
    public void run()
    {
        t.printTable(100);
    }

}
public class TestSynchronization2
{
    public static void main(String args[])
    {
```

```
Table obj = new Table();
MyThread1 t1 = new MyThread1(obj);
MyThread2 t2 = new MyThread2(obj);
t1.start();
t2.start();
}

}
```

Output

```
java TestSynchronization2
5
10
15
20
25
100
200
300
400
500
```

Java Thread suspend() method:-

- The suspend() method of thread class puts the thread from running to waiting state.
- This method is used if you want to stop the thread execution and start it again when a certain event occurs.
- This method allows a thread to temporarily cease execution.
- The suspended thread can be resumed using the resume() method.

Syntax

```
public final void suspend().
```

Program

```
import java.io.*;
class JavaSuspendExp extends Thread
  {
    public void run()
      {
        for(int i=1; i<5; i++)
          {
            try
              {
                Sleep(500);
                System.out.println(Thread.
                  currentThread().getName()+" = "+i);
              }
```

```
catch (InterruptedException e)
  {
     System.out.println(e);
  }
System.out.println();
}
}
}

class Suspendmain
  {
     public static void main (String args[])
     {
        // creating three threads
     JavaSuspendExp t1 = new JavaSuspendExp ();
     JavaSuspendExp t2 = new JavaSuspendExp();
     JavaSuspendExp t3 = new JavaSuspendExp();
        // call run() method
     t1.start();      // Thread-0
     t2.start();      // Thread-1

        // suspend t2 thread
     t2.suspend();
        // call run() method
     t3.start();   // Thread-2
     t2.resume();
     }
  }
```

O/P

Thread-2 = 1
Thread-1 = 1
Thread-0 = 1
Thread-2 = 2
Thread-1 = 2
Thread-0 = 2
Thread-2 = 3
Thread-1 = 3
Thread-0 = 3
Thread-2 = 4
Thread-0 = 4
Thread-1 = 4

Java Thread resume() method :-

- The resume() method of thread class is only used with suspend() method. This method is used to resume a thread which was suspended using suspend() method.
- This method allows the suspended thread to start again.

Syntax

```
public final void resume().
```

Program

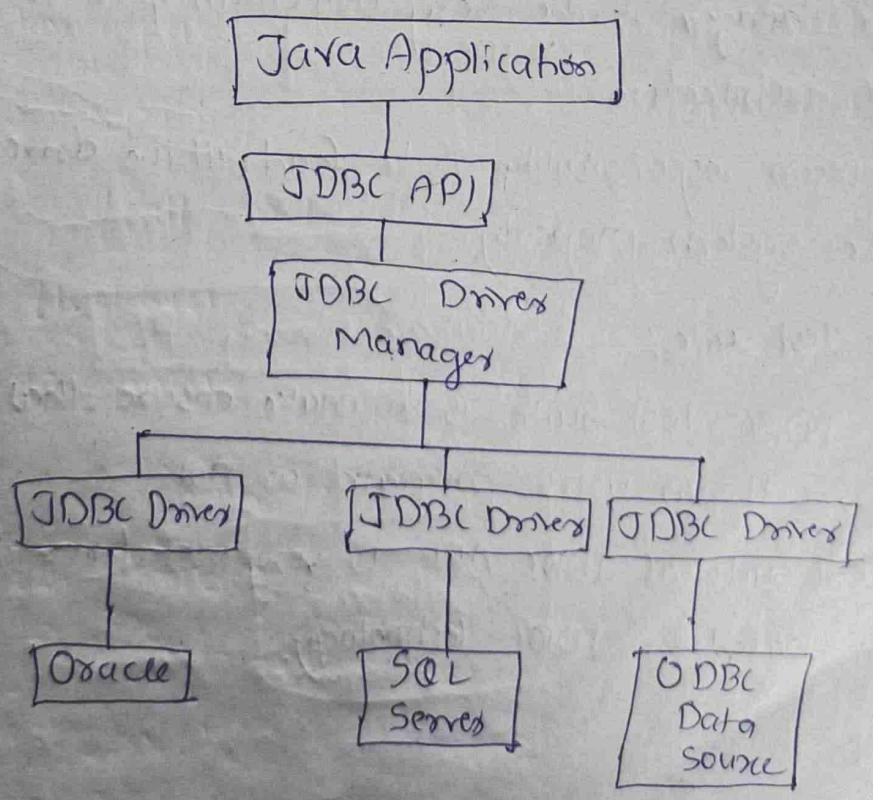
Above program----

JDBC (Java Database Connectivity)

- JDBC stands for Java Database Connectivity.
- JDBC is a Java API to connect and execute the query with the database.
- JDBC is a part of JavaSE (Java Standard Edition).
- JDBC API uses JDBC drivers to connect with the database.

JDBC Architecture :-

- JDBC API supports both two-tier and three-tier processing models for database access.
- JDBC consists of two layers.
  - (i) JDBC API :- This provides the application to JDBC Manager connection.
  - (ii) JDBC Driver API :- This supports the JDBC Manager-to-Driver Connection.



JDBC Components

- There are four type of components are in JDBC
  1. JDBC API
  2. JDBC Driver Manager
  3. JDBC Test Suite
  4. JDBC - ODBC Bridge.

(I) JDBC API:-

- JDBC API provides the facility for accessing the relational database from the Java programming language
- The latest version of JDBC API is divided into two packages.
  (i) java.sql
  (ii) javax.sql

(II) JDBC Driver Manager:-

- JDBC DriverManager is a class in JDBC API.
- Driver Manager is the very important part of the JDBC architecture
- The main responsibility is to load all the drivers found in the system property.

(III) JDBC Test Suite:-

- The JDBC Test Suite is to make ensure that the JDBC drivers will run user's program or not
- The test suite of JDBC API is very useful for testing a driver based on JDBC technology.

④ JDBC - ODBC Bridge:-

- The JDBC-ODBC Bridge is also known as JDBC type1 driver.
- The JDBC type1 driver is a database driver that utilizes the ODBC driver to connect the database..
- This driver translates JDBC method calls into ODBC function calls.

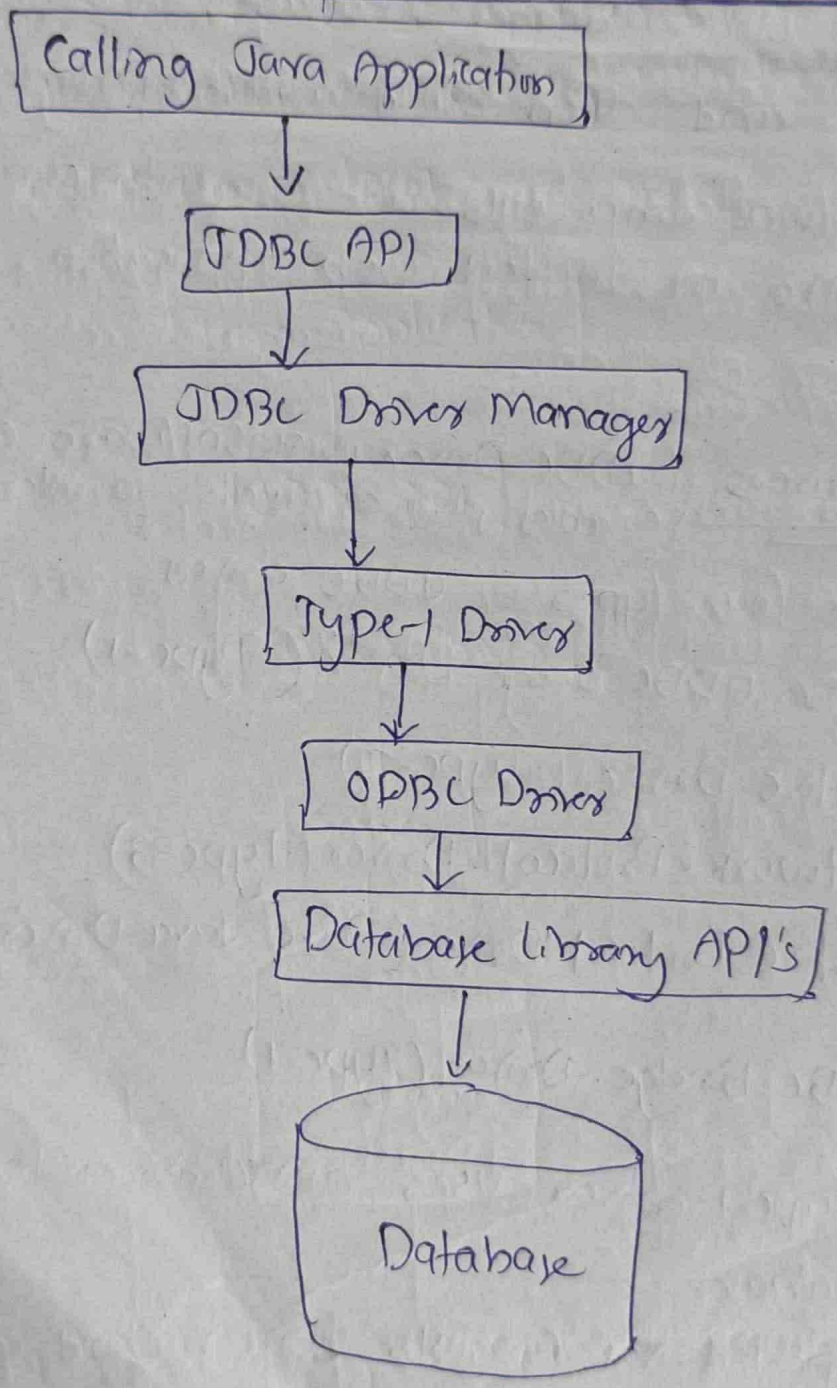
We can conclude from this topics are
- The first two components of JDBC, ie, JDBC API and the JDBC Driver Manager manages to connect to the database and then build a java program.
- The last two components are used to communicate with ODBC or to test web application.

JDBC Drivers (JDBC Driver is a software component that enables java application to interact with the database

- There are four types of JDBC drivers:
  1. JDBC-ODBC Bridge Driver (Type-1)
  2. Native Driver (Type-2)
  3. Network-Protocol Driver (Type-3)
  4. Native Protocol Driver (Pure Java Driver) (Type-4)

① JDBC-ODBC Bridge Driver (Type-1)

- The Type-1 driver uses ODBC driver to connect to the database.
- The Type-1 driver converts JDBC method calls into the ODBC function calls.
- This is discouraged because of thin Driver/Type-4 Driver


Calling Java Application
JDBC API
JDBC Driver Manager
Type-1 Driver
ODBC Driver
Database Library API's
Database

Advantages

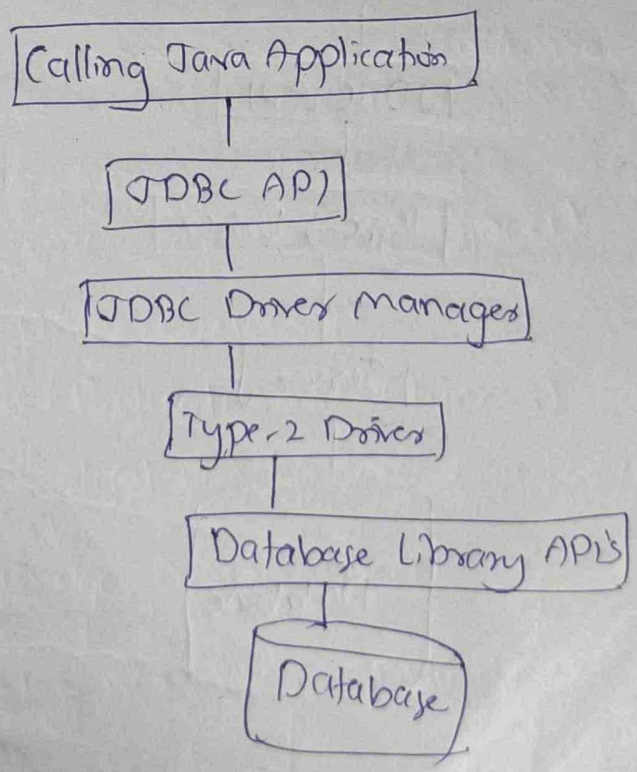
- Easy to use.
- It can be easily connected to any database

Disadvantage

- Type-1 performance is degraded because JDBC method call is converted into the ODBC function calls
- The ODBC driver needs to be installed on the client machine.

2 Native-API Driver (Type-2)

- The Native API Driver uses the client-side libraries of the database
- The driver converts JDBC method calls into native calls of the database API.



Advantages :-

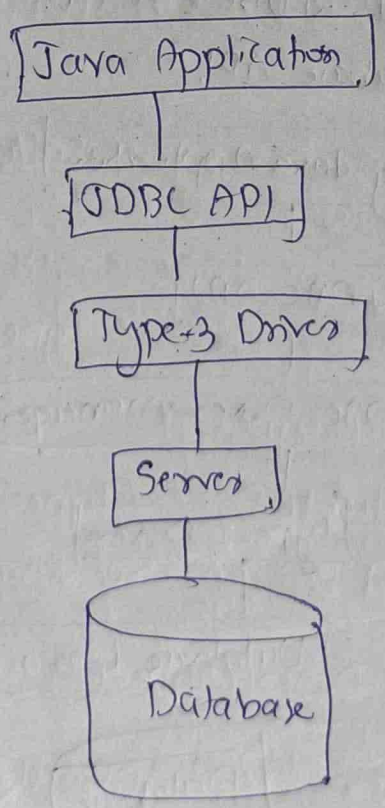
- Type-2 performance is upgraded than Type-1 Driver

Disadvantages

- The Native driver needs to be installed on the each client machine.
- The vendor client library needs to be installed on client machine

③ Network Protocol Driver (Type-3)

- The Type-3 driver uses application server that converts JDBC calls directly or indirectly into the vendor-specific database protocol.



Advantage :-

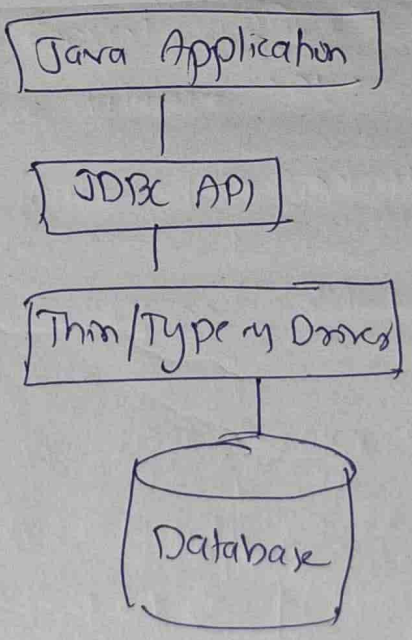
- No client side library is required because of application server that can perform many tasks like auditing, load balancing, etc

Disadvantages:-

- Network support is required on client machine.
- It requires database-specific coding to be done in the middle tier.
- ~~Network support is required on client m/c~~
- Maintainence of network protocol becomes costly

(iv) Native Protocol Driver (Type-4) / thin driver:-

- The thin driver converts JDBC calls directly into the vendor-specific database protocol
- That's why it is known as thin driver



Advantage

- Better performance than all other drivers.
- No software is required at client side or server side.

Disadvantage:-

- Drivers depend on the Database.

Using JDBC

There are 7 steps for using JDBC

1. Load the driver
2. Define the connection URL
3. ~~Establish~~ the connection
   Establish
4. Create a statement object
5. Execute a query
6. Process the results
7. Close the connection.

END OF CHAPTER ....Chiku Pradhan