Java

10/01/2023

What is Java ?

⇒ WORA - Write Once run anywhere
⇒ It is derived from C & C++
(syntax) (OOPS concept)
⇒ It is developed by James Borley — Java
⇒ firstly known as oak
⇒ JVM is platform dependent

Java edition

1. J2SE - Java 2 Standard edition
2. J2EE - Java 2 Enterprise edition (large scale app in industry)
3. J2ME - Java 2 Micro edition (program for memory constraint device)

Characteristics of Java

⇒ Java is Object oriented & simple
⇒ Java is distributed (RMI - Remote method invocation)
⇒ Java is compiled & interpreted language
⇒ Java is robust (tough, strong) deallocate memory by itself
⇒ Java is Architectural neutral
⇒ Java is platform independent
⇒ Java is portable
⇒ Java is multi-threaded
⇒ Java is dynamic (features can be added at runtime)

⇒ Java is secure
⇒ Java provides high performance
Java does not support pointer & structure & union
⇒ Java does not support global variable
⇒ Java does not support multiple inheritance
⇒ There is a concept of interface to achieve multi inheritence
⇒ There is no concept of library it has (API) application program interface
⇒ Boolean datatypes are in form of true & false not internally converted into 0's & 1's

11/01/23

Basic program

```
class Hello {
    Public static void main (string args[])
    {
        System.out.println ("Hello world");
    }
}
```

NOTE :- string is a class not data type in java)

Out = output stream

⇒ .java is the extension
⇒ To compile we have a command

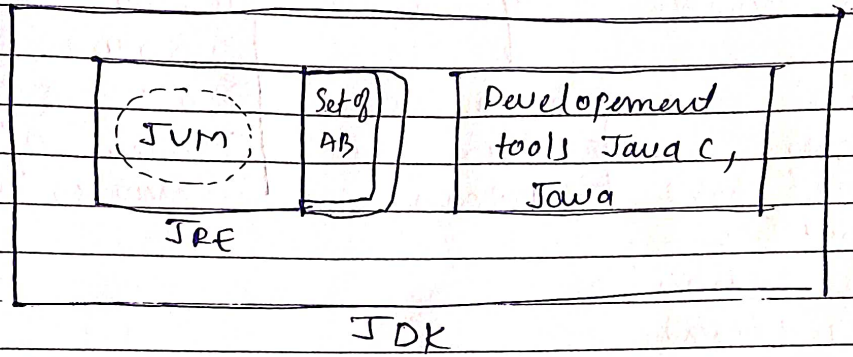
that is java C Hello.java

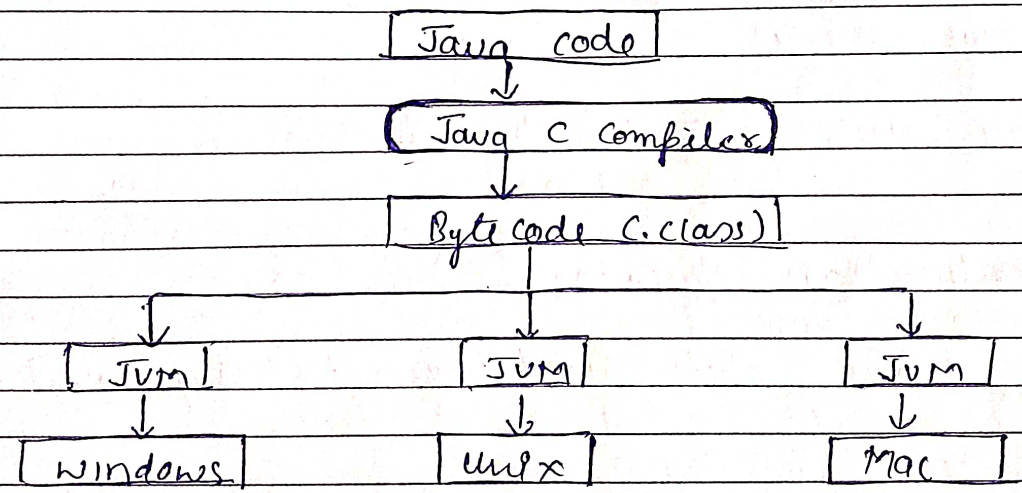
⇒ To run :- java file name

⇒ JDK :- It is a tool (JVM, JRE)

⇒ JVM :- Java virtual machine

⇒ JRE :- Java runtime environment



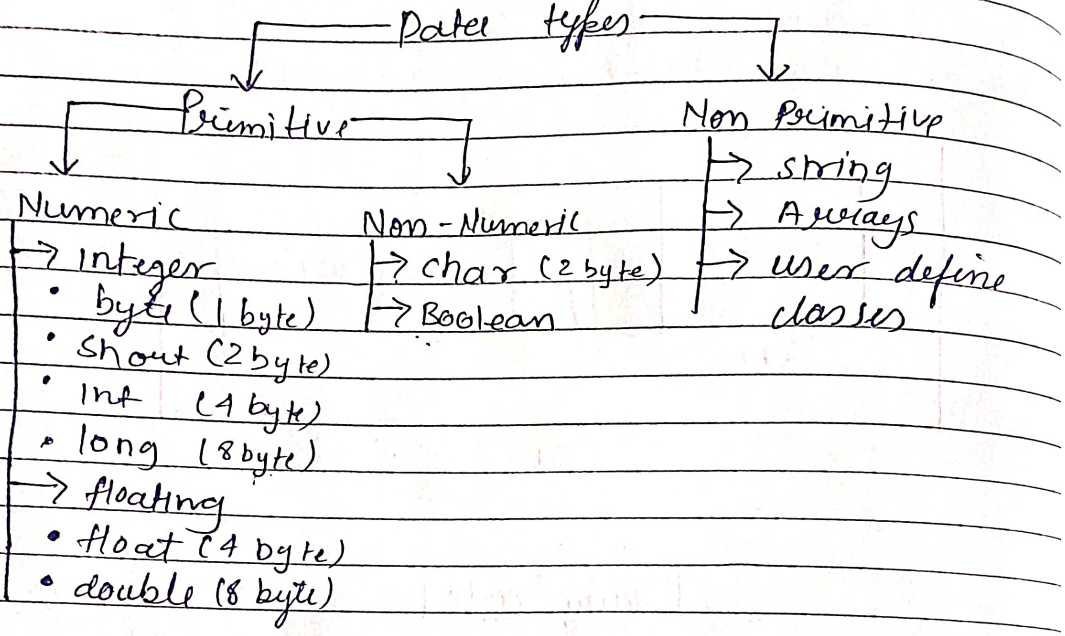


Java Program Execution

Data types

⇒ Java is strongly typed language

⇒ Java data types are of two type

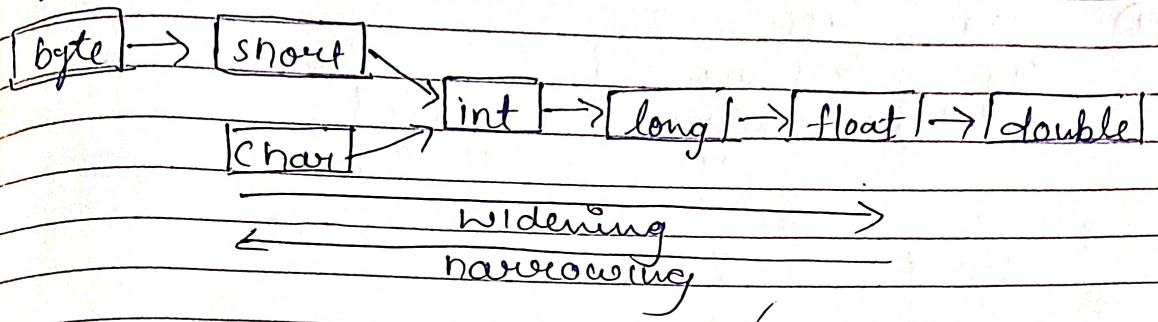


☆ Char uses unicode character set
($2^{16} = 65536$)

Concatination operator (+)

```
class dum {
    Public static void main (string arg[])
    { int i = 1 , j = 2 ,
      int s = i+j ;
      system . out . println ( "sum of " +i+ " & " +j+ "
                                +s) ;
    }
}
```

Type Conversion




```
class Conversion {
Public static void main(string args[])
{ byte b;
  int i= 257;
  double d= 323.142;
  s.o.p ("Conversion of int to byte");
    b= (byte) i;
    s.o.p (b); // output = 1
    i= (int) d;
    b= (byte) d;
    s.o.p (b) // output = 67
  }
}
```

Classes and objects

```
Class Box {
   int height, depth, width;
}
```

lang package is a default package
system class belong to lang package

General Syntax :-

① for class

```
Class class_name {
  datatype instance variable 1 ;
      "          "          " 2 ;
      :
  void method 1 () {
      ------ }
  void method 2 () }
      ------ ;
}
```

② for object

```
obj_name . var = 1 ;
obj-name . method () ;
```

⇒ 3 characterstics of objects are:-

1. State
2. Behaviour
3. Identity (values of objects are unique)

☆ Syntax of creating an obj.

```
Classname obj_name = new class_name ();

class Box {
  int height, width, breadth ; }
class Box_demo {
  Public static void main (String args[])
  { Box b1 = new Box ();
```

```
b1. height=10;   b1. width=y;   b1. breadth=1p;
S.O.P ("Valueme =" +(height x width x breadth);
}}
```

14/01/2023

Constructor

```
class Box {
int height, width, depth;
Box() {        // No argument constructor
  height = width = depth = 10;
}
void volume ()}
  return
}
```

⇒ Characteristics of java

① Same name as the class name
② A single class can have more than one constructo
③ It doesn't have any return type
④ When you do not define a constructor then java will use its default constructor and once you define the constructor then the default constructor will not be used

☆ Parameterised Constructor

```
Box( int h, int d, int w) // argument Constructor
{ height = h;
  width = w;
  depth = d;
}
```

20/01/2023

Arrays :- int arr[] = new int[10];

or

int[] arr = new int[10];

NOTE :- To take the input :-

import java.util.Scanner;

ex:- ① class Demo
{ Public static void main (String arr[])
{ scanner sc = new scanner (System.in);
System.out.Println ("Enter the two num
int n1 = sc.nextInt();
int n2 = sc.nextInt();
int sum = n1+n2;
System.out.Println (" sum is " +sum);
}
}

② class Demo
{ Public static void main (string arr[])
{ scanner sc = new scanner (system.in);
int arr[] = new int [10];
System out. println (" Enter the element");
for (int i=0; i > arr.length ; i++)
arr[i] = sc.nextInt();
int sum = 0;
for (int i=0 ; i<arr.length ; i++)
sum = sum + arr[i];

```
int max = arr[0]
for(int i=0; i<arr.length; i++)
{ if(arr[i]>max)
    max = a[i];
}}}
```

⇒ for each loop:- for(int i : arr)
sum = sum+1;

→ It is an enhanced for loop/for each loop.
→ We can't modify the array by this loop

23/01/2023

Variable:- ① Local (within method) ② Instance (within class/obj)
③ static:- Class _ Test {
int x = 10; // instance variable
Public static void main (string arr[]){// static method
s.o.p(x); // error (access through object)
Test t = new Test();
s.o.p(t.x); // 10
}
public void m,(){ // non static method
s.o.p(x); //10
}

use of static var is for memory management

- When a member is declared static, it can be accessed before any objects of its class are

created and without reference to any object

- Method declared as static have several restriction. They can only directly call other static methods. They can only directly access static data
- They can not refer to this or super keywords (super → inheritance)

```
class Counter {
static int count; // by default initiated to 0
Counter () {
count ++;
S.O.P. (count);
}
P.S.V.M (String ar[]) {
Counter c1 = new Counter (); // 1
Counter c2 = new Counter (); // 2
Counter c3 = new Counter (); // 3
```

- static block is used to initialise static data members. It is executed before main method at the time of class loading

```
Class use static {
static int a = 3, b;
static void math (int x) {
S.O.P ("x = " + x); // 40
S.O.P ("a = " + a); // 3
```

```
s.o.p ("b = " + b) ; //12        first static block
                                 will be called
① static {
    s.o.p ("static block initialized")
    b= a*4;
}②class Demo {
 P.s.v.m (string ar[]) {
 Use static . math (40);
 d. o.P ( use static .a); //3
                }       }
```

Method overloading

```
class Demo {
    void test () {
        s.o.p ("No parameters");
    }
    void Test (int i) {
        s.o.p ("i=" + i);
    }
    void test (int i, int j) {
        s.o.p ("i&j=" + i + " " + j);
    }
    void test (double d) {
        s.o.p ("d=" + d);
                }       }
class overload2Demo {
    P.s.v.m (string ar[]) {
    Demo d1 = new Demo ();
```

```
ob1.test();
ob1.test(10);
ob1.test(10,20);
ob1.test(20.50)
```

Constructor Overloading

```
class Box {
int width, height, depth; //instance variable
Box(){                    //No argument constructor
height = depth = width = -1;
}
Box(int l) { //No argument constructor
height = l;
width = l;
depth = l; }
Box (int h, int w, int d){
height = h; depth = d; width = w;
}
void volume(){
s.o.p("Volume =" + (height * width * depth);
}
}

class Box_Demo {
P.S.V.M (String arg[]){
Box b1 = new Box();
Box b2 = new Box(10);
Box b3 = new Box (5, 10, 15);
```

```
b1.volume();
b2.volume();
b3.volume();
}}
```

Command line arguments

⇒
```
class Demo {
    P.S.V.M (String args[]){
        if (args[0].equals("b"))
            s.o.p ("Hello");
        else
            s.o.p ("Hi");
    }
}
```

⇒
```
class Demo {
P.S.V.m (String args[]){
int i = Integer.parseInt (args[0]);
int j = Integer.parseInt (args[1]);
int sum = i+j;
    s.o.p ("sum =" + sum);
}
}
```

Inheritance :-

We use extend Keyword for Inheritance

⇒ Types of inheritance :-

① single ② Multilevel ③ Multiple ④ Heirarchical

{NOTE:- at top in java there is Object}

⑤ Hybrid

⇒ Multiple inheritance is not supported in java but it can be achieved using interface

ex:- ① class Emp {
float salary = 40000;
}
class programmer extends Emp {
int bonus = 10,000;
P.S.V.M (string args[])
{ Programmer P= new Programmer ();
s.o.p ("Salary of Programmer =" +P.salary);
s.o.p ("Salary of Bonus =" +P.bonus);
}
}

② class A {
int i, j;
void show ();
{ s.o.p ("Values of i &j=" +i+" " +j);}
}
class B extends A {
int k;
void show1() {
s.o.p("Valu of k=" +k); }
}
void sum () {
int s= i+j+k;

```
    s.o.p ("sum =" +s); }
}
class Demo {
    P.S.V.M (String[] args) {
    A a = new A();
    a.i = 10; a.j = 20; a.show();
    B b = new B();
    b.i = 5; b.j = 10; b.k = 15;
    b.show();
    b.show1();
    b.sum();
                }                    }
```

A Super class variable can reference a Subclass object

→ A reference variable of a Super class can be assigned a reference to any Sub class derived from that Super class. When a reference to a Sub class object is assigned to a Super class reference variable you will have access only to those parts of the objects defined by the Super class. This is because the Super class has no knowledge of what a Sub class adds to it

30/01/2023

refer the ex that is written in Constructor overloading

```
class BoxWt extends Box{
  int weight;
  BoxWt (int w, int h, int d, int wt)
  { width = w;
    height = h;
    depth = d;
    weight = wt;
  }
}

class Demo{
  P.S.V.M (String ar[])
  { BoxWt W1 = new BoxWt(5,10,15,20);
    Box b1 = new Box();
    int vol = w1.volume();
    S.O.P ("Volume = "+vol);
    S.O.P ("Weight = "+w1.weight); //error
    b1 = w1;
    vol = b1.volume();
    S.O.P ("Volume = "+vol);
    S.O.P(" wt = "+b1.weight);
  }
```

Super Keyword :- It can be used with constructor, variable and methods. It is used to call them

▶ Constructors :-

```
Class BoxWt extends Box{
  int weight;
```

```
BoxWt(int w  int h  int d, int wt)
{ super(w, h, d); // it'll call argument construct
  weight = wt;    in super class
}
BoxWt() {
  super(); // it'll call no argument constructor
  wt = -1;        in super class
}
}
Class Demo {
  P.S.V.M (String arg[]) {
  Box Wt W1 = new BoxWt(5, 10, 15, 20);
  BoxWt W2 = new BoxWt();

  int vol = W1.volume();
  S.O.P("Volume = " + vol);
  SO.P("Weight = " + weight);
  B1 = W1;
}}
```

NOTE :- Super should be the first statement in any block in which ever you are using. By the help of super we can set / initialize the value of private data member

→ Variable :-

```
Class A {
int i;}
```

```
class B extends A {
  int i;
  B (int a, int b) {
    super.i = a;
      i = b;
                    }
  void show () {
    S.O.P ("Value of i in super class =" + super.
    S.O.P("Value of i in sub class = " + i);
} }
  class Demo {
    P.S.V.M (String ar[]) {
    B  b = new B (10, 20);
      b.show ();
  } }
```

NOTE :- for variable :- super.variable_name

→ Methods :- for method :- super.method();

```
class A {
  int i, j;
  A(int a, int b) {
    i = a;
    j = b;   }
  void show () {
    SO.P (" values of i & j = " + i + " " + j);
}
```

```
class B extends A {
int K;
B(int a, int b, int c) {
super (a, b);
  K = c;
}
void show () {
S.O.p ("Value of k = " +k);
}
class Demo {
P.S.V.M (String ar[]) {
B b = new B (10,20,30);
b.show ();
}
```

NOTE :- When you have same method name with same signature then the child class will overrride that method

Order of execution of Construction in Multilevel heirarchy

Constructors always executed from top to down

```
ex: class A {
      A() {
        S.O.p ("Inside A's constructor");
    }}
      class B extends A {
```

```
    B() {
    s.o.p ("Inside B's constructor");
  }}
    class C extends B {
      C() {
      s.o.p ("Inside C's constructor");
  }}
    class Example {
    P.S.V.M (String ar[])
      { C c1 = new C(); }
  }
```

31/03/2023

Dynamic Method Dispatch / Dynamic Binding

```
    Class A {
    void show() {
      s.o.p("A's show method");
  }}
  Class B extends A {
      void show() {
      s.o.p ("B's show method");
  Class C extends A { void show() {
    s.o.p("C's show method");
  }}
    class Demo {
    P.S.V.M (String ar[]) {
    A a = new A();
    B b = new B();
```

```
C c = new C ();
A r; r = a;
r.show ();
r = b;
r.show ();
r = c;
r.show ();
}}
```

⇒ It is the mechanism by which a call to an overridden method is resolved at run time rather than compile time. Using this mechanism java implements run time polymorphism

⇒ method overriding

```
class figure {
int dim1, dim2;
figure (int d1, int d2)
  dim1 = d1;
  dim2 = d2;
}
void Area () {
S.O.P ("Area is undefined");
}}
class Rectangle extends figure {
Rectangle (int d1, int d2) {
super (d1, d2);
}
```

// If we have any method that is abstract then use abstract before writing the name of that class like, abstract class figure

// abstract void Area(). By this we can declare it as an abstract

```
void area() {
  S.O.P("Area of Rectangle = " + (dim1 * dim2));
}}
class Triangle extends figure {
  Triangle (int d1, int d2) {
    super (d1, d2);
}
  void area() {
    s.o.p ("Area of triangle =" + (dim1 * dim2 / 2));
}}
class Demo {
  P.S.V.M (String ar[]) {
  figure f = new figure (1, 2);
   Rectangle r = new Rectangle (10, 15);
   Triangle t = new Triangle (10, 20);
   figure fg;
     fg = f;
   fg. area();
   fg = r;
   fg.area();
   fg = t;
   fg. area();
}}
```

NOTE :- We can't create object of Abstract class but we can create reference variable. Abstract class is not fully defined so we can't instantiate with 'new' Keyword.

you can have a abstract class without having any abstract method.

final Keyword:-
It can be used with
① Var ② method ③ class

① with variable:- It will act as constant
ex:- final int file.operator = 1;
that means the value can not be changed

② with method:- If we use final with method then we can not override that method
ex:- final void method();

③ with class:- If we use final with class then we can not inherit that class
ex:- final class A {
- - - - - - }

NOTE:- By default the methods of final class are final
⇒ A class cannot be final and Abstract together.

Difference b/w Abstract class & final class

SN	Abstract class	final class
1	It helps to achieve abstraction	It helps to restrict other classes from accessing its property and method

2	All the abstract method should be overriden	Overriding concept doesn't arise as final class can't be inherited
3	can be inherited	can't be inherited
4	It can not be instantiated	It can be instantiated
5	Abstract class may have final method.	final class does not have abstract method.

[NOTE:- All the methods of string class are final.]

Interfaces:- It helps us to achieve full abstraction or complete abstraction. It consist only abstract method.

ex:-
```
interface A
{ int i = 1;
  void method ();
}
```

{ here the variables are by default 'public', 'static' & 'final' & all the methods are abstract method. }

⇒ The interface is implement in class by implements keyword

ex:-
```
class demo implements A {
    void method ();
}
```

Suppose if interface has two method and then we implement and if that class only implements One method then it will show an error To remove that error use abstract Keyword.

```
ex:- interface Figure {
Public void area (int l, int b);
}
    class Rectangle implements Figure {
Public void area (int len, int breadth)
    { int ar = len * breadth;
      S.O.p ("area of Rectangle =" + ar);
{Public void display (){ s.o.p ("Hello"); }}
class RX{
P.S.V.m (String args[])
{ Rectangle r = new Rectangle ();
    r. area ();
}}
```

NOTE:- We cannot create a object of interface but we can create reference variable

```
fig f = new Rectangle ();
  f. area (10, 20);
  f. display () // error
  r. display () // correct
```

variable f is declared as interface type variable and it is assigned an instance of rectangle class. It can only access area method (interface method) it can not access any other members of the rectangle class (Non interface methods)

⇒ Extending / Inheriting interfaces

```
interface fig {
    void area (int l, int b);
}
interface fig1 extends fig {
    void area1();
}
interface fig2 extends fig1 {
    void area2();
}
class ex implement fig2
{
}
class example implement fig, fig1
{ void area();
}
```

⇒ A class can implement more than one interface but it can extends only one class

An Interface can extend multiple interface at a time

```
interface IntStack {
    void push(int i);
    int pop(int i);
}
class StackExample implements IntStack
{ Private int stck[];
  Private int tos;
  stackExample(int size){
    stck = new int[size];
    tos = -1;
}
  Public void push(int item){
    if (tos == stck.length - 1)
      s.o.p("stack is full");
    else
      stck[++tos] = item;
}
  Public int POP(){
    if (tos < 0){
      s.o.p("stack underflow");
      return 0;
}
    else
      return (stack[tos--];
}
class MainExample {
P.s.v.m (String ar[]).
{StackExample se = new StackExample(5);
```

```
for (int i=0; i<5; i++)
        se.push(i)
for (int i=0; i<5; i++)
        s.o.p(pop());
}}
```

⇒ We use interfaces when there are no default implementation

Interface v/s Abstract class

S.NO	Interface	Abstract
1.	When we don't know anything about implementation then use interface	Use abstract class when there is partial implementation
2.	All the methods are public and abstract by default	Every method in abstract class need not be public and abstract
3.	Interface variable is always public, static & final	Every variable need not be public, static & final
4.	Interface variables are to be initialised compulsarily	There are no compulsions
5	Interface doesn't have constructor	It can have

Interface v/s class

S.no	Interface	class
1	Interface can only have abstract methods. Java 8 onwards, it can have default as well as static method	A class can have both an abstract as well as concrete methods
2	Interface supports multiple inheritance	Multiple inheritance is not supported
3	Only static and final variables are permitted	final, non static, static and non final variables supported
4	Interface can not implement an interface it can extends an interface	A class can implement an interface
5	Interface is declared using interface keyword	A class is declared using class keyword
6	Interface can not have a constructor	A class can have constructor methods
7	An Interface can only have public members	A class can have any type of member like private, public

Packages :- Package is a container of class. class consist of data whatever you class create or use it is stored on predefined in the ~~Table~~ Package

The default package of java is java.lang & java.util.*

⇒ Uses of Package :-

- Reusability
- When you write a same program & saved as same name then the compiler gives you this program is also exist
  It provides the convention for unique class name, prevent class name conflicts (Naming collision)
- It helps to manage the complexity of program
- Java uses file system directories to store the package

```
Package P;
import java.util.*;
    class A {
        }
    class B extend A {
        }
```

you need to creat a folder & name as the package name & then save in the file & set the path variable

Access protection

	Private	No modifier (Default)	Protected	Public
Same class	Y	Y	Y	Y
Same Packag subclass	N	Y	Y	Y
Same Packag non-subclass	N	Y	Y	Y
Different Packag sub class	N	N	Y	Y
Diff. Packag non subclass	N	N	N	Y

Class member acess

Inner class & Nested class

```
class A {
    data ;
    methods ();
        class B {
            }
    }
```

```
Class Outer {
  int outer_x = 100;
    void test(){
  Inner inner = new Inner ();
     inner. display () {
}}
```

```
    class Inner {
        void display () {
        S.O.P ("display : outer_x :" + outer_x);
        }
    } // end of Inner class
} // end of outer class

class InnerDemo {
    P.S.V.M (String ar[])
    { Outer O1 = new outer ();
        O1. test ();
    }
}
```

★ we always have non-static inner class & for accessing inner object we need to acess like Outer. inner

★ Outer class can acess inner members through object but the inner class can acess outer members directly.

10/02/202

Exception handling

try, catch, throw, throws, finally

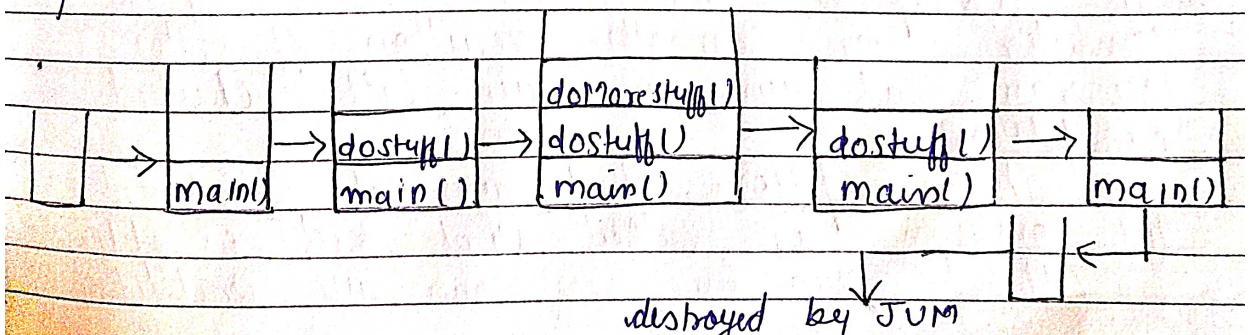
An exception is an abnormal condition that arises in a code at run time. It is highly

recommended to handle the exception. The main objective of exception handling is graceful termination of the program

```
class Demo {
public static void main ( String ar[] )
{ int a= 10;
  in  c= a/ 0;
  s.o.p ("This will not be printed");
} }
```

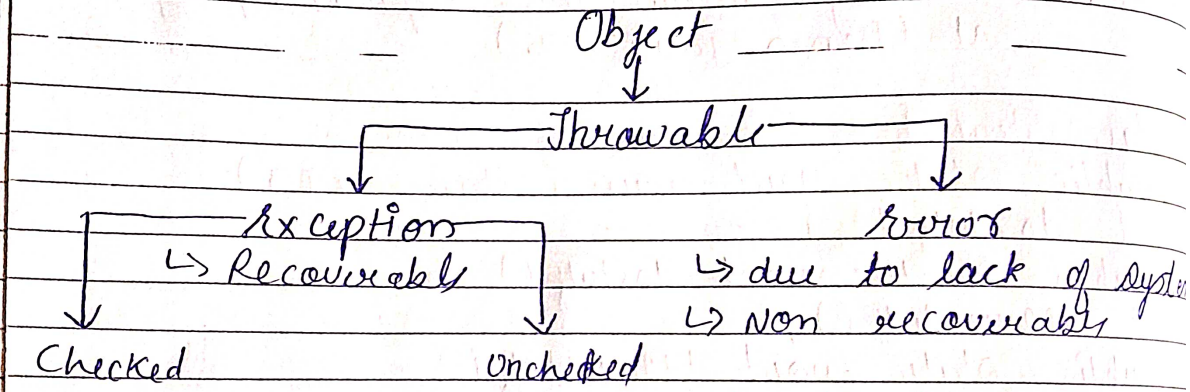
error :- Java.lang.Arithmatic Exception : / by zero
at (Demo.java :5)

```
class Test {
public static void main ( String ar[] )
 { dostuff ();}
public static void dostuff ()
{ doMorestuff ();}
public static void doMorestuff ()
 { s.o.p ("Hello");}
}
```



Run time stack Mechanism

⇒
```
class Test {
  P.S.V.m (String ar[])
  { int b = 10;
    try { s.o.p("Hello");
          int c = b/0;
          s.o.p ("This statement will not be pri
        }
    catch (ArithmeticException e)
    { s.o.p ("Division by zero error"); }
    s.o.p ("This will get printed");
}}
```



⇒ The exceptions which are checked by the compiler for smooth execution of the program at run time are called checked exception

The exceptions which are not checked by the compiler are called unchecked exception

All the run time exceptions are Unchecked

⇒ Use of throws Keyword :- for declaring the checked exception

⇒ Run time exception and its child classes, error & its child classes are unchecked and all the remaining are considered as checked exception

27/02/2023

* Multiple catch clause :- A try block can have multiple catch clause

Ex:-

```
class Example {
p-s.vm (String args[])){
try {
    int n= args.length;
    int c = 10/n
    int arr[] = {1,2};
      arr[5] = 10;
} catch (Arithmetic Exception e){
    S.O.p ("Divide by zero" +e); }
    Catch (Array Index Out Of Bounds Exception e)
{ S.O.p ("Array Index out of range"); }
    S.O.p ("After try catch statement");
}}
```

* The sequence of catch block must be sub class to superclass

When you have multiple catch statement the exception subclass must come before any of

its superclass

★ Throw :- throw ThrowableInstance;
ex:- throw new ArithmeticException("demo
It is used to throw an exception explic
from a method or a static block in
java.
The flow of execution stops immidiatel
after the throw statement

⇒
```
Class TestThrow {
p s v m ( String args[] )
static void validate (int age)
{ if (age <18){
    throw new ArithmeticException ("Not valid
} else
    s.o.p ("welcome to vote");
p.s.v.m(String a[]){
    validate (15);
}}
```

⇒
```
Class TestThrow {
static void demproc(){
  try {
  throw new NullPointerException ("demo");
} catch (NullPointerException e) {
  s.o.p ("Caught inside demproc");
  throw e;}
```

```
}
P. s vm (String args[]) {
    demo().doc();
}
catch (NullPointerException e) {
    S.o.p("Recaught :" + e);
}}}
```

```
P.s vm (String args[]) {
    deref00 c();
}
catch (NullPointerException e) {
    S.o.p("Recaught :" + e);
}}}
```

28/02/2023

⇒ throws :-

⇒ void validate() throws IOException

⇒ P.s vm (String args[]) throws IOException

In our program if there is any chance of raising checked exception compulsorily we should handle either by 'try, catch' or by throws keyword otherwise the code won't compile

The main objective of the throws keyword is to delegate the responsibility of exception handling to the caller method (the one who has called the method) then caller method is responsible to handle that exception. It is used only with the checked exceptions

⇒ finally :- It is nothing but a block of code basically it is used for clean up operation whatever task you want to run

compulsorily you write it in finally block

⇒ Custom Exception / user defined exception.

→ You can create your own exception by creating a class (exception class).

```
import java.util.Scanner
Class MinAccBalance extends Exception{
  String msg;
  Public MinAccBalance (String message)
  { this.msg = message; }
  Public String toString ()
  { return msg; }
}
Public class Ex{
Static double curr_bal = 1000;
ps vm (String ar[]) throws MinAccBalance{
Scanner s = new Scanner (System.in);
int n = s.nextInt ();
try {
    if (curr_bal < n)
  throw new MinAccBalance ("Insufficient funds");
  else
   s.o.p ("please Take the money =" + n);
} catch (MinAccBalance (Ab)
{ s.o.p (mb); }
}}
```

Throw
Throws
Throw new Throwable Instance
throws with method signature
validate () throws IOException, NullPointerException

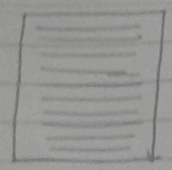
01/03/2023

- Throw ↳ ~~Exception~~ Explicitly throws an exceptions
  ↳ checked exception can't be throw
- Throws ↳ declare an exception
  using chained exception It forms a chain

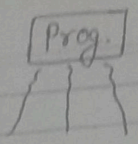
Multithreading :- It is a concept where you can have multiple threads in your program
It can perform multiple task in parallel or simultaneously eg :- Gaming
Multithread program contain 2 or more concurrent task. Each thread has a separate path of execution. A program in execution is process

Multitasking → Process based (feature of OS)
→ thread based (feature of program

Thread based :- Multiple process in one proce
all process run can currently

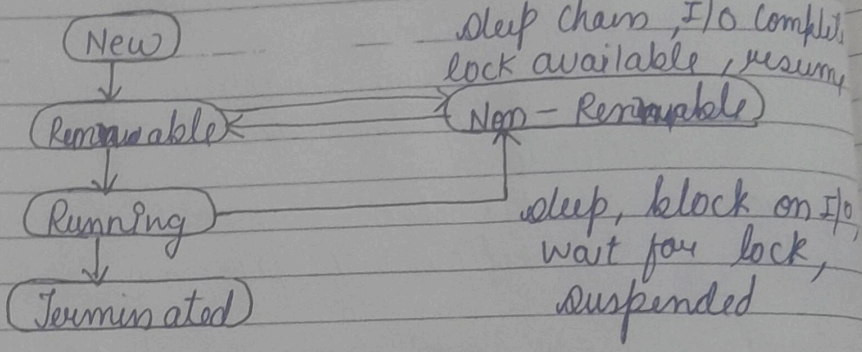


Execution is in a sequence either I/O operation / Logical operations



Processors are able to process multiple process at a single time is called multi processor
process → heavy weight
thread → light weight

Java thread Model
Thread exists in several steps



Threads have priority, Java assigns each thread a priority, a thread can voluntarily preempted by higher priority thread. The default priority of thread is file

Minimum priority :- 1
higher priority :- 10
default (Normal) :- 5

There is a thread class done in Java
There are two ways to create threads
↳ by extending thread class
↳ by runnable interface

```
class ThreadDemo {
psvm (String arg[]){
Thread t = Thread.CurrentThread()
S.O.P ("Current Thread =" +t);
try {
   for (int i=0; i<10; i++)
      S.O.P (i);
   Thread.sleep (1000);
} catch (InterruptedException e){
   S.O.P ("MainThread Interrupted");}
}}
```

02|03|2023

⇒ Application of Multithreading
- to implement multimedia graphics
- develop animation
- develop video games

↳ Thread scheduler is a part of JVM which decides which thread to execute when multiple threads are waiting

★ Creating a thread by extending the thread class

```
class MyThread extends Thread {
  public void run()
    for(int i=0; i<5; i++)
    { s.o.p("child thread running"); }
}
class ThreadDemo {
p s v m (String args[]) {
MyThread t = new MyThread();
  t.start();
for(int i=0; i<5; i++)
s.o.p("Main Thread running");
}}
```

★ What are the function of start()

↳ It'll register the thread with the Thread scheduler

↳ All mandatory and low level activities are performed by start()

↳ It'll invoke the run method ~~that~~ without executing the start method there is no chance of starting a new thread

```
t.start();
try{
  for(int i=0; i<5; i++)
  {s.o.p("Main Thread running");
    Thread.sleep(500);
}} catch(InterruptedException e)
```

S.O.P ("Main Thread Interrupted");
}}}

Same program as previous just change the program from t.start();

Output of this program:-

Main Thread running
Child Thread running
Child Thread running
Main Thread running
Child Thread running
Child Thread running
Main Thread running
Main Thread running
Main Thread running
Child Thread running

Output of previous program

Main Thread running
Main Thread running
Main Thread running
Main Thread running
Main Thread running
Child Thread running
Child Thread running
Child Thread running
Child Thread running
Child Thread running

↳ By implementing the Runnable interface

```
class MyRunnable implements Runnable{
  Public void run(){
  try{
      for(int i=0; i<5; i++)
      { S.O.P ("child thread running");
        Thread.sleep(200);
  }}catch(      )
      {          }
}}
```

```
class ThreadDemo {
psvm (String args[]) {
MyRunnable r = new MyRunnable (0);
t.start ();
try {
  for (int i=0, i<5; i++)
  { s.o.p ("Main Thread running");
    Thread.sleep (500);
  } catch (InterruptedException e) {
    s.o.p ("Main Thread interrupted")
}}}
```

This is the best approach among the two approaches.

Synchronization

↳ It is a built in mechanism in java which lets only one thread use a shared resource at a time

↳ It is used to prevent thread interference and prevent inconsistancy problem

↳ Every object in java has a unique lock which is achieved using the 'synchronized' keyword and this keyword is available only with methods & block and it cannot be apply with class and variable

```
class Table {
synchronized void printTable(int n){
  for(int i=1; i<=10; i++)
  { s.o.p(n*i);
    try {
        Thread.sleep(500);
  } catch (InterruptedException e)
    { s.o.p(e);}
}}}
class MyThread extends Thread{
  Table t;
  MyThread(Table t){
    this.t = t;}
  public void run(){
    t.printTable(5);
}}
class MyThread2 extends Thread {
  Table t;
  MyThread2(Table t){
    this.t = t;
  Public void run(){
    t.print Table(10);}}
class Test {
psvm(String ar[]){
Table obj = new Table();
MyThread t1 = new MyThread(obj);
MyThread t2 = new MyThread2(obj);
t1.start(), t2.start();}}
```

Before use of Synchronized Keyword :-

output :- 5
10
10
15
⋮

After use of Synchronized Keyword :-

output :- 5 10
10 20
15 30
⋮
50 100

⇒ Disadvantage of Synchronized Keyword

→ It increases the waiting time of thread
→ Affects the performance of the system
→ Constructor can not be synchronized

NOTE- Hence a synchronized block can be used to increase the performance whenever it is required

before use of Synchronized Keyword :-

output :- 5
10
10
15
:

After use of Synchronized Keyword :-

output :-

5	10
10	20
15	30
:	:
50	100

⇒ Disadvantage of Synchronized Keyword

→ It increases the waiting time of thread
→ Affects the performance of the system
→ Constructor can not be synchronized

NOTE Hence a synchronized block can be used to increase the performance whenever it is required

★ Methods that thread class have

→ getName → get → getPriority → isAlive
→ join → sleep → run → start

★ two ways to determine whether the thread is finished or not

↳ final
↳ final

(Join) This
meth

Jo
↳ If
othe
↳ If
If
can

In
It
→ w

★ F

1.

2.

3.

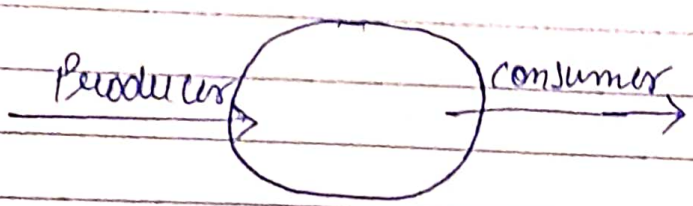
↳ final boolean isAlive()
↳ final void join() throw InterruptedException

(Join) This method is used to wait for some other method to terminate.

Join() v/s sleep
↳ If a thread wants to wait until completing some other thread then we should use join()
↳ If we don't want to wait then we use sleep
If a thread don't want wait for a particular amount of time then we should use sleep

InterThread Communication
It is achieved using three methods:-
→ wait() → notify() → notifyAll()

★ Producer - Consumer Problem



Queue

1. When the queue is empty then the consumer will wait for the queue to get felled again
2. When the queue get filled then the producer will notify it
3. When there are multiple thread then by

producer will notifyAll.

- wait() :- When you call a wait() then it calls the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor and calls notify() & notifyAll()
- notify() :- It wakes up the thread that call wait on the same object
- notifyAll() :- wakes up all thread that called wait method.

producer will notifyAll.

wait() :- When you call a wait() then it calls the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor and calls notify() & notifyAll()

notify() :- It wakes up the single thread that call wait() on the same object

notifyAll() :- wakes up all thread that called wait method.

13/03/2023

Strings

Strings are immutable

It is a class not a basic data type so we create the object of string class

StringBuffer, String and many more, all these classes are found in java.lang package and these classes are final also

strings are immutable means the contents of the string instance cannot be changed after it has been created but a variable declared as string reference can be changed to point at some other string object at any time

★ Ways to create String :-

→ String s = new String();
→ String s1 = new String("Hello");
→ char c[] = {'a', 'b', 'c', 'd'};
→ String s2 = new String(c)

↳ To find the length :- s2.length()
↳ To extract the character from string :- s2 = s.charAt(0)
↳ let s = HelloWorld ; char ch[10];
⇒ s.getchars(int Source Start, int Source end, char target [], int targetstart);
⇒ s.getchars(0, 4, ch, 0);
output = {'H', 'e', 'l', 'l', 'o'}
↳ To compare Strings :- equals(), equalIgnoreCase()
String s1 = new String("Hello");
String s2 = new String("Hello");
String s3 = new String("HELLO");
s1.equals(s2) //output : True
s1.equals(s3) //output : False
s1.equalsIgnoreCase(s3) //output : True
↳ int compareTo(s)
s1.compareTo(s2)
↳ String s2 = s1.substring(5); // World
String s3 = s1.substring(0,5); // Hello
here let s1 = new String("HelloWorld");
↳ toLowerCase(), toUpperCase();

★ diff. b/w '==' and equals()
== this will compare the reference variable

equals() compare the character inside the string and == operator compares two object references.

StringBuffer sb = new StringBuffer("Hello");

⇒ To append any new string to already existing string :- sb.append("World");
max size will be 16.

☆ StringBuffer & StringBuilder is same but a difference is that StringBuffer is not Synchronized

14/03/2023

Applets :-

Java programs
- Application
- Applet

we will import java.applet.*
java.awt.*
to use applet
awt = abstract window toolkit
applet package has Applet class in it

there are several methods that we'll use

- init() – 1st method to be called by brawser & it is executed only once. It does all the initation task
- start() :- It is called after init Method & each time the applet are revisited by the user. It can be executed multiple times
- stop() :- To stop the appled, when applet are get minimised or closed this methood get called
- destroy() :- before destroy stop method is called destroy() is called to allocate the memory
- paint() :- It is the method of awt package which is used to draw the applets.
The paint method is called when the applet begins execution. It is used to execute the applet

```
import java.awt.*;
import java.applet;
/* < applet code = "SimpleApplet" height-200
   width) = 200>
  </applet> */
public class SimpleApplet extends Applet{
  public void init() {
    setBackground (color.yellow); }
  public void paint(Graphics g) {
  g.drawString ("Helloworld", 50, 100);
}}
```

★ Applets doesn't have main method

To view the applet files there are two ways
① Web browser
② AppletViewer

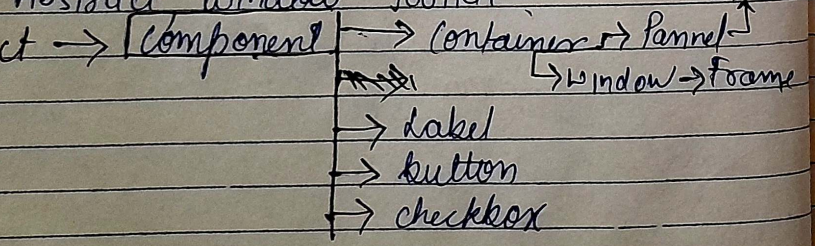
⇒ Uses of Applet
1. Applets are used on internet to create dynamic webpage
2. It also help us to make graphics and game

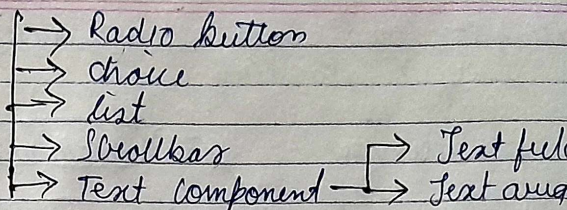
Diff. b/w Applet and Application

→ Applets donot need main method whereas an Application requires main method for execution
→ Applets require java compatable web browsers to run whereas application can directly run on your machine because they are standalone program
→ Applets uses GUI framework to interact with user whereas Application have I/O classes to interact with user

24/03/23

AWT :- Abstract window Toolkit





Classes of AWT

A window is an imaginary ~~rectangular~~ box on the screen without any border or title bar. & a frame represent a window with some title and the border

ex:-

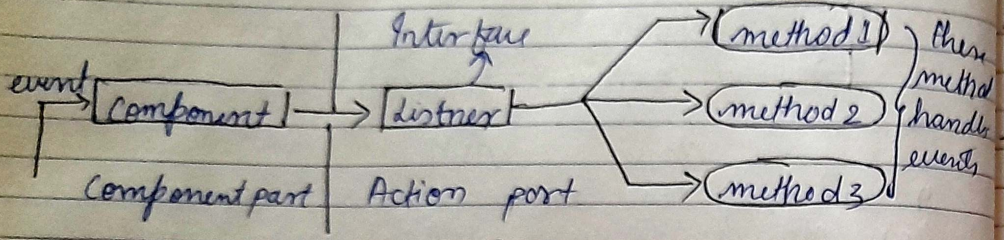
① Frame f = new Frame();
f.setSize(400, 400);
f.setVisible(true);

② Button b = new Button(); //without label
Button b = new Button("Submit"); //with label
String l = b.getLabel();
b.setLabel("submit");

③ Checkbox cb = new Checkbox() //without label
Checkbox cb = new Checkbox("label") //with label

# Graphics:- It is a class
drawRect(2,2,20,10);
drawLine(int x, int y, int x1, int y1);
drawRect(int x, int y, int w, int h);

Event handling :- Java uses Event Delegation model for event handling



Event - Delegation Model

Event handling is the mechanism that control the event & decide that what should happen if an event occur

★ Classes & interface for event handling

ActionEvent	ActionListner
MouseEvent	MouseListner
KeyEvent	KeyListner
ItemEvent	ItemListner
TextEvent	TextListner

Event classes → ~~Event~~ Listner Interface

Event Delegation model represents that when an event is generated by the user on a component, it is delegated to a listner interface and the listner calls a method in response to the event finally the

★ listner needs to be registered with the source object so that listner can receive the event notification

event is handled by the method

What is the benifit of Event Delegation Model?

→ The two parts can be developed in two different environment

→ We can modify the one part without affecting the other part, so that debugging and maintainance becomes easy.

```
import java.awt.*;
import java.awt.event.*; // Sub Package of event. Listner
class MyButtons extends Frame implements ActionListner
{
    Button b1, b2, b3;
    MyButton() {
        b1 = new Button("Yellow");
        b2 = new Button("Blue");
        b3 = new Button("Red");
        add(b1); add(b2); add(b3);
        b1.addActionListner(this);
        b2.addActionListner(this);
        b3.addActionListner(this); }
    public void actionPerformed(ActionEvent ae)
    { String str = ae.getActionCommand();
        if(str.equals("Yellow"))
            setBackground(Color.Yellow);
        if(str.equals("Blue"))
            setBackground(Color.Blue);
        if(str.equals("Red"))
```

```
class ThreadDemo{
psvm (String args[]){
MyRunnable r = new MyRunnable (8);
t.start ();
try {
  for (int i=0 ; i<5 ; i++)
  { s.o.p ("Main Thread running");
    Thread. sleep (500);
  }} catch (InterruptedException e) {
    s.o.p ("Main Thread interrupted")
}}}
```

This is the best approach among the two approaches.

Synchronization

↳ It is a built in mechanism in java which lets only one thread use a shared resource at a time

↳ It is used to prevent thread interference and prevent inconsistancy problem

↳ Every object in java has a unique lock which is achieved using the 'synchronized' keyword and this keyword is available only with methods & block and it cannot be apply with class and variable

```
class Table {
Synchronized void PrintTable (int n){
  for (int i=1; i<=10; i++)
  { s.o.p (n*i);
    try {
      Thread. sleep (500);
  } catch (InterruptedException e)
    { s.o.p (e);}
}}}
class MyThread extends Thread{
  Table t;
  MyThread (Table t){
    this.t = t; }
  public void run(){
    t. printTable (5);
}}
class MyThread2 extends Thread {
  Table t;
  MyThread2 (Table t){
    this.t = t;
  Public void run(){
    t. print Table (10);}}
class Test {
psvm (String ar[]){
Table obj = new Table ();
MyThread t1 = new MyThread (obj);
MyThread t2 = new MyThread2 (obj);
t1. start (), t2 : start (); }}}
```

before use of Synchronized Keyword :-

output :- 5
10
10
15
⋮

After use of Synchronized Keyword :-

output :-

5	10
10	20
15	30
⋮	⋮
50	100

⇒ Disadvantage of Synchronized Keyword

→ It increases the waiting time of thread
→ Affects the performance of the system
→ Constructor can not be synchronized

NOTE- Hence a synchronized block can be used to increase the performance whenever it is required

★ Methods that thread class have,

→ getName → getPriority → isAlive
→ join → sleep → run → start

★ two ways to determine whether the thread is finished or not

↳ final boolean isAlive()
↳ final void join() throw InterruptedException

(Join) This method is used to wait for some other method to terminate.

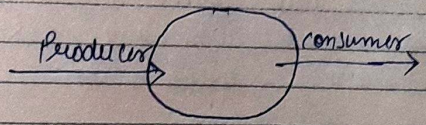
Join() v/s sleep

↳ If a thread wants to wait until completing some other thread then we should use join()
↳ If we don't want to wait then we use sleep If a thread dont want wait for a particular amount of time then we should use sleep

InterThread Communication

It is achieved using three methods :-
→ wait() → notify() → notifyAll()

★ Producer - Consumer Problem



Queue

1. When the queue is empty then the consumer will wait for the queue to get felled again
2. When the queue get filled then the producer will notify it
3. When there are multiple thread then the

producer will notifyAll.

- wait() :- When you call a wait() then it calls the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor and calls notify() & notifyAll()
- notify() :- It wakes up the single thread that call wait() on the same object
- notifyAll() :- wakes up all thread that called wait method.

13/03/2023

Strings

↳ Strings are immutable
↳ It is a class not a basic data type so we create the object of string class
↳ StringBuffer, String and many more, all these classes are found in java.lang package and these classes are final also

strings are immutable means the contents of the String instance cannot be changed after it has been created but a variable declared as string reference can be changed to point at some other string object at any time

★ Ways to create String :-

→ String s = new String();
→ String s1 = new String("Hello");
→ char c[] = {'a', 'b', 'c', 'd'};
→ String s2 = new String(c)

↳ To find the length :- s2.length()
↳ To extract the character from string :- s = s.charAt(i)
↳ let s = HelloWorld, char ch[10];
⇒ s.getchars(int sourceStart, int sourceEnd, char target[], int targetStart);
⇒ s.getchars(0, 4, ch, 0);
output = {'H', 'e', 'l', 'l', 'o'}
↳ To compare Strings :- equals(), equalIgnoreCase()

String s1 = new String("Hello");
String s2 = new String("Hello");
String s3 = new String("HELLO");
s1.equals(s2) //output : True
s1.equals(s3) //output : False
s1.equalsIgnoreCase(s3) //output : True

↳ int compareTo(s)
s1.compareTo(s2)
↳ String s2 = s1.substring(5); // World
String s3 = s1.substring(0,5); // Hello
here let s1 = new String("HelloWorld");
↳ toLowerCase(), toUpperCase();

★ diff. b/w '==' and equals()
== this will compare the reference variable

→ String S[] = f.list();

Try with Resources (ARM) :-
⇒ Automatic resource management (ARM)
⇒ Two ways two close a file :-
- fileInputStream fin = new fileInputStream("a.txt");
  fin.close();
- By Autoclosable interface
  try (fileInputStream f = new fileInputStream("a.txt")
  { ... }

Java Collection framework :- It is a collection of classes and interfaces. It comes under java.util packages. All the interfaces are inherited from Collection class.

Interface	classes
Set<Integer>	Hashset, LinkedHashset
List<String>	Stack, LinkedList, ArrayList, Vector
Queue	LinkedList
Map	HashMap, HashTable

⇒ Set didn't allow duplicate element but list allows

01/04/2023

Generic Type :- It represents a class or an interface that is type safe it can act on any data type

```
class A{
    Integer x; }
class B{
    String x; }
```

It acts only upon objects. It works on any datatype so we cannot specify a particular datatype when designing the class.

```
class Myclass <T>
{//class code
    T obj; }
Myclass <String> s = new Myclass <>();
```

It is also called as 'parameterised type' because they use a parameter that determines which data type they should work upon.

```
public class GenericEx{
public static <T> void PrintArray(T[] inputArray)
{ for (T element : inputArray)
    S.O.P(element); }
P.S.V.m(String args[]){
Integer [] intArray = {1,2,3,4,5};
Double [] doubleArray = {1.1, 2.2, 3.3, 4.4, 5.5};
Character [] charArray = {'H','E','L','L','O'};
S.O.P("Integer Array contains");
    PrintArray(intArray);
S.O.P("Double array contains");
    PrintArray(doubleArray);
S.O.P("Character array contains");
    PrintArray(charArray);
}}
```

Garbage Collection It is run by JVM

⇒ How can an object be unreferenced?
↳ By nullifying the reference
↳ By assigning a reference to another
↳ By anonymous objects (Nameless objects are called anonymous object)
ex :- new student (). display ();
We use anonymous objects in inner class

⇒ System . gc ()
⇒ Runtime . getRuntime (). gc ()
⇒ ~~finally~~ finalize () :- this method is called before objects memory deallocation,