Unit - III

Object Oriented Programming In Java

DATE 9-10-23

* Review of OOP :-
- classes describes properties and behaviours of objects.
- objects is run time entity (real world entity).
- objects is an instance of class. Memory allocated to object is in HEAP (in JVM).

```
class student
{
        int roll_no;          } Data member
        char name[20];
        void display()  ← member function
        {
          System.out.println("Name is:"+name);
          System.out.println("Roll No=" +roll_no);
        }
}

class Demo
{
  Public static void main (String args[])
  {
        student S1= new student();
        S1. display();
  }
}
```

Hash code - Unique ref. no is allocated to object variable.
(i.e memory is allocated to objects in heap by Java (JVM))

OOP features-
1. classes
2. objects
3. Encapsulation
4. Inheritance - reusability of code.

⑤ Polymorphism.
⑥ Abstraction - Through access specifiers.

* S1.Hashcode() use to display unique ref no (Hash code of S1 objects).
i.e System.Out println("Hashcode of object S1:"+ S1.hashcode());
* hashcode() is under lang package
- Hashcode is unique identification numbers alloted to objects by JVM.
- It is also called reference numbers which is created base on the location of the object in memory and is unique for all objects, except for String objects.

- In above example the filename is demo.java because the mainclass/method belonging to that class.
- If Variable declared but not assign value to it then by default it considers as for default value of instance variable -

int, byte, short, long } 0
char → blankspace
String → NULL

float, double } 0.0

Boolean → false
Class → Null

initialising instance variable-

```
int roll.no = 1;
string name = "abc";
char a = 'h';
```

* Access specifies :-
  a) Private - data member accessible only inside class
  b) Public - data member/classes can accessible inside or out
  c) Protected -
  d) default -

- We are not declaring the class as private because the JVM unable to execute the program because data / class is private.

- A class cannot be declared as private because it will not be available to java compiler and hence a compile time error occurs.

- But inner classes can be declared as private.

Protected :- Accessible outside class but with same directory.

default - Accessible outside class but within same directory.

* Constructor :-
- In java only two constructors are available
  1) default
  2) parameterized.

- Same name as classname.
- constructor is invoked at once But normal method can invoke multiple times.

```
class Student
{
    private string name;
    private int roll-no;
    Student()
    {
        name = "abc";
        roll-no = 5;
    }
    Student (string n, int r)
    {
        name=n;
        roll-no = r;
    }
    void display()
    {
     s.o.p(name);
     s.o.pc roll-no);
    }
}
class demo
{
Public static void main (String args[])
{
    Student s1 = new Student();
    s1. display();
    Student s2 = new Student ("ABC", 10);
    s2. display();
}
}
```

* A constructor is use to initialise the instance variable of class.
* A method

	② A method is use for any general purpose processing or calculation.
③ A constructor name and classname should be same.	③ A method name or classname can be same or different.
④ A constructor is called at time of creating object.	④ A method can be called after creating the object.
⑤ A constructor is called only once per object.	⑤ A method can called several times.
⑥ A constructor is called and executed automatically.	⑥ A method is executed when it is called.

* write a java program to perform addition of two number with parameterised method / methods with parameter and having return type.

```
class Addition
{
        int no1, no2, add;
    int addTwoNo(int a, int b)
    {
            no1 = a;
            no2 = b;
            add = no1 + no2;
        return add;
    }
}
class Demo
{
Public static void main(String args[])
{
        Addition A1 = new Addition();
intresult=A1. addTwoNo (10, 90);
        Sout
    System.out.println ("The sum of twonumber"
                        + result);
}
}
```

* Static method :-

* this :-

- It's used for variable, class method, constructor.

```
class demo
{
    demo()
    {
        this(10); // calling parameterized constructor
        this.display();
    }
    demo(int x)
    {
        this.x = x
    }
    void display()
    {
    System.out.print("X=" +x);
    }
}
class Thisdemo // Method main() called by class Thisdemo. main() by JVM.
{
    Public static void main(String args[])
    // To make code available to JVM.
    {
        demo d = new demo();
    }
}
```

* Static method :-

```
class demo
{
    Static int X=10;
    demo()
    {
```

```
        this(10);
    }
    static void display()
    {
        S.o.p("x="+x);
    }
}
class ThisDemo
{
    public static void main(String args[])
    {
        Demo d=new Demo();
        Demo.display();
    }
}
```

Instance Variable	Static Variable
① An instance variable is a variable whose seperate copy is available to each object.	① A static variable has a single copy in memory and is shared by all object.
② Instance variables are created in the objects on heap memory.	② Where as static variables are stored on method area.
③	③

* Static block -

```
class demo4
    static {
        S.o.p("This is static block");
    }
public static void main(String args[])
 {
    S.o.p("This is main method");
 }
}
```

-Static block has more priority than static method.
so output: This is static block
This is main method.

* Inheritance :- Types of inheritance in java.
1) Single
2) multilevel
3) hierarchical
// multiple - leads to ambuiguity.

① Single inheritance -

```
class A
{
    display A()
    {
     S.o.p("This is base class");
    }
}
    class
class B extends A
{
    display B()
```

17-10-23

```
        }
            S.o.p ("This is child class");
        }
}
class Demo
{
    public static void main ( String args[])
    {
        class b = new B();
        b.displayB();
        b.displayA();
    }
}
```

* class one
{
    int i;
    void display()
    {
    S.o.p ("This is superclass variable",+i);
    }
}
class two extends one
{
    int i = 20;
    super.display();
    void display()
    {
        S.o.p ("This is subclass variable",+i);
    }
    S.o.p ("This is superclass var",+super.i);
}

```
class demo
{
    public static void main( String' args[])
    {
        class two t =new two();
          t.display();
    }
}
```

→ OUTPUT:

This is superclass var 10.
This is subclass var 20.
This is subclass var 10.

```
class One
{
      one()
      {
      s.o.p ("one");
      }
}
class two extends one
{
        two()
        {
          s.o.p ("two");
        }
}
class demo
{
    public static void main()
      {
```

```
        two t= new two();
      }
}
```

→ OUTPUT: one
two

//default constructor of base class get call implicitly by default constructor of subclass.

* parameterised constructor in classes:-

```
class one
{
    int i;
    one (int a)
    {
        this.i=a;
    }
    void display (a);
    {
      S.o.p(" This is superclass var ",+i);
    }
}
class Two extends one
{
    Two ( int a, int b)
    {
        super (a);
        i=b;
    }
    void display ( )
    {
      super.display();
```

```
        S.o.p ("This is subclass", +b);
        S.o.p ("sub class var i =", +super.i);
    }
}
Class demo
{
    public static void main (String args[])
    {
        Two t = new Two (10,20);
        t.display();
    }
}
```

→ OUTPUT:

This is super class var 10
This is subclass class var 20
This is super class var 10

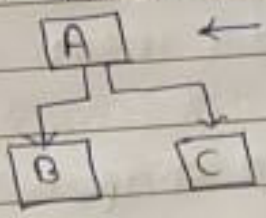
(*) ~~Hierarchical~~ inheritance :-
Multilevel

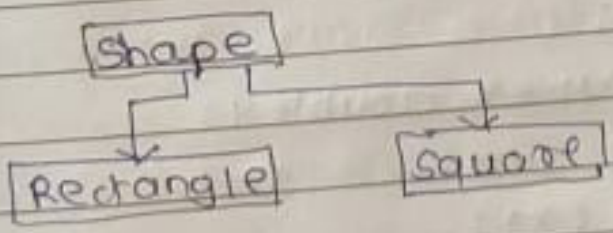
```
class one
{
    one()
    {
        S.o.p ("one");
    }
}
class Two extends one
{
    Two()
    {
        S.o.p ("Two");
    }
}
```

```
class Three extends Two
{
        Three()
        {
          S.o.p(" Three");
        }
}
class Demo
{
    public static void main(String args[])
    {
      Three t = new Three();
    }   t
}
```

→ OUTPUT: one Two Three

③ Hierarchical inheritance -






```
class Shape
{       area=0;
        Shape()
        {
        S.o.p("area=", area);
        }
}
```

```
Class Shape
{
        double l;
        Shape (double l)
        {
            this.l = l;
        }
}
Class Square extends Shape
{
        Square (double l)
        {
            super(l);
        }
      Void area()
      {
        S.o.p("Area of Square is"+(l*l));
      }
}
Class Rectangle extends Shape
{
        double l, b;
      Rectangle( double x, double y)
        {
              super(x);
               b=y;
        }
      Void area()
        {
          S.o.p("area of rectangle=",l*b));
        }
}
```

```
class demo
{
    Public Static void main( String args[])
    {
        Square S = new square ();
          S. area ();
        Rectangle R = new rectangle ();
          R. area ();
    }
}
```

* Polymorphism

- Coercion is automatic conversion between different data types done by the compiler.

ex:- float a = 1.5f;
int x = int (a)

conversion of a float to int.

* Runtime polymorphism -

① Polymorphism with methods / function overloading:-

- function with same name but differ in signature (no: of arguments, data type of arguments).

ex:- add (int a, int b);
add (float a, float b, float c);
add (int a, int b, int c);

* write a java program to add two numbers using function overloading

```
Import java.util.scanner;
class Demo
{
    public static void main(String args[])
    {
        Add A1 = new Add();
        A1.add(10,20);
        A1.add(10,30,40);
        A1.add(1.2, 4.5);
    }
}
class Add
{
    int a, int b;
    Void add(int a, int b)
    {
      s.o.p("add="+(a+b));
    }
    void add(int a, int b, int c)
    {
      s.o.p("add=", +(a+b+c));
    }
    void add(float a, float b)
    {
      s.o.p("add=", +(a+b));
    }
}
```

function overloading	function overriding
① Writing two or more methods with same name but different signature is called function overloading.	① Writing two or more methods with same name and signature is called overriding.
② Method overloading is done in same class.	② Method overriding done in different class (subclass and base class).
③ In method overloading method returntype can be same or different.	③ In method overriding method returntype should also same.
④ Method overloading is used for code refinement.	④ Method overriding is code replacement.

② Static polymorphism -
- object binds to function is known at compile time.

```
Class One
{
Static void calculate (double x)
  {
      S.o.p ("Square =" + x.x);
  }
}
class Two extends One
{
```

```
    static void calculate(double x)
    {
      s.o.p("square root" + Math.sqrt(x));
    }
}
class Poly
{
    public static void main(String args[])
    {
        one o = new one(); // o is reference variable
        o.calculate(2.5);
    }
}
```

* - final methods (methods declared as final) cannot be overriden because they are not available to subclasses.

- method overloading is possible with final methods.

* final class-
- A final class is a class which is declared as final and it prevents the class from getting inherited.

final variable its value cannot be changed.

final int a = 3;

↑ similar to const int a = 3;

	Dynamic polymorphism	static polymorphism
(1)	Dynamic polymorphism is achieved at runtime.	
(2)	method overloading and method overriding are examples of dynamic	
(3)	polymorphism. Here jvm decides which method to be called at runtime.	

* Abstract class and method:

```
class Test
{
    void calc(int a, int b)
    {
        S.o.p ("Result is:", + (a+b));
    }
}
class Demo
{
    Public static void

abstract class mainclass
{
    abstract void calculate (double x)
}
class sub1 extends Mainclass
{
    Void calculate (doublex)
    {
```

```
        s.o.p (" Square is:"+(x*x));
    }
}
class sub2 extends Mainclass
{
        Void calculate (double x)
        {
         s.o.p ("Square root is:"+ main. (Sqrt(x)));
        }
}
class sub3 extends Mainclass
{
      Void calculate (double x)
      {
        s.o.p ("cube is:"+ (x*x*x));
      }
}
class Demo
{
  public static void main( String args[])
  {
     sub1 o1 = new sub1();
     sub2 o2 = new sub2();
     sub3 o3 = new sub3();
      o1. Calculate (5);
      o2. Calculate (0.5);
      o3. Calculate (2);
  }
}
```

- Abstract class contains zero or more abstract methods.
- An abstract class can contain instance variable and concrete methods in addition to abstract methods.
- All the abstract methods of the abstract class should be implemented in its subclasses.

- We cannot create object to abstract class. But, we should create another subclass an implement abstract methods.
- We can create a reference of abstract class type and this reference of abstract class type can we used to refer to objects of its subclass.

The reference of abstract class can not refer to individual method of its subclass

Interfaces :

```
interface Simple
{
    void a();
    void b();
    void c();
}
class Sub1 implements Simple
{
    void a()
    {
        System.out.println("This is a method");
    }
}
class Demo
{
    public static void main(String args[])
    {
        Sub1 s1 = new Sub1();
        s1.a();
    }
}
```

- An interface is specification of a method prototype.
- An interface contains only abstract methods which are incomplete methods.
- So, user needs to create seperate classes where we can implement all the methods of the intermediate face.

This classes are called implementing classes.

Note : Interface methods are public by default and abstract also.

All methods should implementing in implementing class

- If any method is not implemented then that implementation class should be declared as abstract

-

* Multiple inheritance-
- Multiple Inheritance in java acheived through interfaces.

∴ Thus C1 class implements methods of I1 and I2

class ↓ extends class

interfaces ↓ implements class

interface ↓ extends interface

interface ✕ implements interface

Ex:

```
interface Printable {
    void print(); // abstract method
}

interface showable {
    Void Show(); // abstract method.
}
Class demo implements Printable, showable
{
    public Void print()
    {
        S.o.p ("Hello");
```

```
interface Drawable
{
    void draw();
}
class Square implements Drawable
{
    void draw()
    {
        S.o.p("This is square");
    }
}
class circle implements Drawable;
{
    void draw()
    {
        S.o.p ("This is circle");
    }
}
class Demo
{   public static void main (String args[])
    {
        Drawable Square S = new Square();
        s.draw();
        circle c = new circle();
        c.draw();
    }
}
```

* We cannot create object to an interface but we can create a reference of interface type.

- All the methods of interface should be implemented in its implementation classes.

```
public void show()
{
    s.o.p("Good morning");
}
}
class sample {

public static void main(String args[])
{
    demo d = new demo();
    d.show();
    d.print();
}
}
```

or

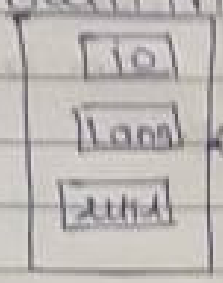
```
interface printable {
    void print();
}
interface showable extends printable {
    void show();
}

class demo implements printable,showable
{
    public void print()
    {
        s.o.p("Hello");
    }
    public void show()
    {
        s.o.p("Hello world");
    }
}
```

```
class Sample
{
    public static void main(String args[])
    {
        Demo d = new Demo();
        d.show();
        d.print();
    }
}
```

- ~~An~~ abstract class is written	- Interfaces
① An abstract class is written when there are some	① An interface is written when all features are Implemented differently in different objects.
② whenever an abstract class is written it is duty of programmer to provide subclasses to it.	② An interface is written when the programmer wants to leave the implementation to the third party vendors.
An abstract class contains some abstract methods and some concret methods.	③ An interface contains only abstract methods.
An abstract class contain instance variables.	④ An interface can not contain instance variabl (it contains only cons

* Java packages
built in packages

1) io
2) lang
3) util
- A package represents a directory that contains related group of classes.



java. import java.lang.*;

Advantages of packages
- packages are useful to arrange related classes and interfaces into a group.
- Packages hide the classes and interfaces in seperate subdirectory so that accidental deletion of classes and interfaces will not take place.

① java.lang lang → language
- This package consist of primary classes and interfaces essential for developing basic java program.

② java.util util - utility
- It contains classes and interfaces like stack, linked list, Hash table, vectors, arrays, etc.

system date and time can be display by

③ java.io → Stands for input & output.
- This package contains streams are useful to store data in form of files and also to perform input, output related task.

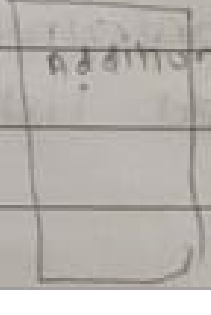
④ java.awt → abstrat window tool kit
This package helps to develope GUI.

⑤ java.net - (networking.net)
Client sever programing can be done by using this ~~class~~ package.

* User defined packages-
Syntax;
Package nameofpackage
class classname
{

}

```
Package pack
class Addition
{
    double d1, d2;
    Addition( double a, double (b) )
    {
        d1 = a;
        d2 = b;
    }
public Void add ()
    {
```



```
        S.o.p ("Addition is " + (a+b));
    }
}
import pack.Addition;
class Use
{
    Public static void main( String args[])
    {
        Addition a = new Addition (10, 15);
            a.add();
    }
```

CLASSPATH-

It is an enviroment variable that tells java compiler where to look for class files to import.

CLASSPATH is generally set to a directory or a JAR file.
(Java Archive file)

A JAR file is the file can contains compress version of .class files, Audiofile, imagefiles, or directory.

Exception handling & multithreading

* Exception handling

Exception - The error does not caught by compiler

checked	Unchecked
- Detect during compilation	- detect during execution

ex:-

```
BufferReader br = new BufferReader (InputStreamReader
                                    (System.in)
        br.readline();
```

- Exception is class in java.

① checked exception-
- The exception that are checked at compilation time by the java compiler are called checked exceptions.

② - The exception that are checked by JVM are called unchecked exception.

checked exception:

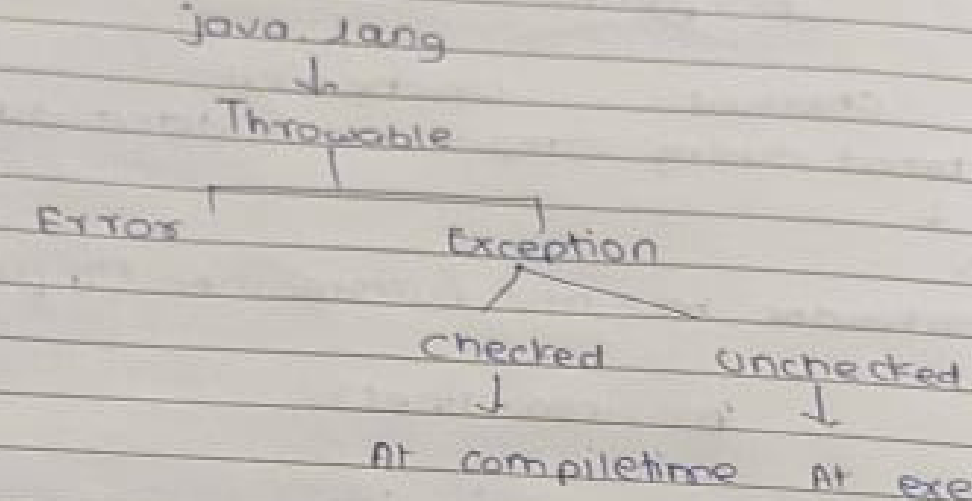
ex:-

```
Public static void main( String args[]) throws
                                        IoException
{
    BufferReader br = new BufferReader (InputStream
                        Reader (System.in))
    br.readline();
```

All exceptions are belong to java.lang package

Unchecked exception ex:
divide by zero:
Int a=100, b=0, c;
c = a/b;

*



Syntax:
try
{

}
Catch(IoException e)
{ object

exception can be display with
e.printStackTrace();
}

ex: Class Exp
{
public static void main(String args[])
{
try {
int a[]= {10,12};

```
        S.o.p(" open files");
        int n= args.length();
        S.o.p ("n=" + n);
        int a = 10/n;
        S.o.p ("a=" + a);
        int c= a+e(5);
    }
    Catch ( ArithmeticException ae)
    {
        S.o.p( ae); // ae.printstacktrace();
    }
    finally  // keyword use to end the main() if theire is any error caught/or not. exception
    {
        S.o.p(" class files");
    }
}
Catch( ArrayIndexOutOfBoundsException ab)
{
    S.o.p( ab);
}
```

throw
throws - use to throw explicit exception explicitly out of the program

① Throws-clause -
- If the programmer is not handling runtime exception java compiler may not give any error related to runtime exception.
- But the rule is that the programmer should handled checked exception.

- In this case the programmer can use throws clause to throw the exception explicitly.

ex:-

Throw clause:-

-Throw clause is used in the try block for transferring the control to catch block.

ex:-

```
class Sample
{
    public static void demo()
    {
        try {
            s.o.p("try block");
        Throw new NullPointerException(
                                "Exception data");
        }
        Catch( NullPointerException ne)
        {
            s.o.p(ne);
        }
    }
class Throw demo
{
    P.S.V.M( )
    {
        Sample demo(); // function call by
                       class name bec
                       demo function is
    }
}
```

- Throw clause is use in software testing to test whether program is handling all the exceptions according to the programmer.

* Built in exceptions :-

checked unchecked

① Unchecked exception-
- detected during runtime

① ArithmaticException
② ArrayIndexOutOfBound
③ ArrayStoreException
④ ClassCastException
⑤ EnumConstantNotPresentException
⑥ IllegalArgumentException.
⑦ IllegalMonitorStateException.
⑧ IllegalStateException
⑨ NegativeArraySizeException
⑩ Nullpointer exception.
⑩ NumberFormatException
⑪ SecurityException
⑫ StringIndexOutOfBoundException.
⑬ TypeNotPresentException.
⑭ UnsupportedOperationException.

ex:

NULL pointers

① class NullPointer
{
public static void main( )
{
try {

```
        String a= null;
        s.o.p ( a.charAt(0));
      }
      Catch( NullPointerException ne)
      {
        s.o.p(" string IS NULL");
      }
  }
}
```

② class StringIndex
```
{
  p.s.v m( )
  {
    try{
        String a = "Hello";
        char c = a.charAt(10));
        s.o.p(c);
      }
    Catch( StringIndexOutOfBound e)
      {
        s.o.p(" string out of bound");
      }
  }
}
```

③ class Hoformal
```
{
    public static void main()
    {
        try{
            int no= Integer.parseInt("No");
```

```
            S.o.p(no);
        }
        catch (NumberFormatException e)
        {
            S.ap ("No Format Exception");
        }
    }
}
```

* User defined exception :-

```
class MyException extends Exception
{
    Private Static int accno[] = { 1001, 1002,
                                   1003, 1004, 1005};
    Private Static String name[] = {"abc", "xyz",
                "pqr", "lmp", "stq"};
    Private Static double bal[] = {1000.02,
            2000.05, 3000.06, 4000.01, 5000.0
    MyException()
    {
    }
    MyException (str)
    {
        super (str);
    }
}
class
  public static void main( String args[])
{
    try {
```

```
S.o.p( "Accno : Name : bal" );
for( i=0; i<5 ; i++)
{
   S.o.p ( accno[i] + name[i] + bal[i]);
*
if( bal [i] < 1000)
MyException me= new MyException ( "less") ;
   throw me;
}
}
}
Catch( MyException me)
{
   me. printStractionTrace ( );
}
}
```

* Nested try :-

```
class NestTry {

   public static void main( String args[])
   try {
      try
      {
         int a = args. length;
         int b = 42/a;
         S.o.p(" a=" + a );
      try {
         if( a==1)
            a = a/a-a;
```

```
        if(a==2)
        {
            int c[] = {1};
            int c[42] = 99;
        }
    }
    catch(ArrayIndexOutOfBoundException e)
    {
        S.o.p (e);
    }
}
catch( ArithmeticException e)
{
    S.o.p(" divide by 0 "+e);
}
}
}
```

* Multithreading :-

```
class current
{
    p.s.v.m (String args[])
    {
        S.o.p("current thread execution");
        Thread t = Thread.currentThread();
        S.o.p(t);
        S.o.p("Name is =" + getName());
    }
}
```

OUTPUT: Thread[main,5,main]
Name is = main

- Thread is light weight resource.
- Creating thread

extend Implement Runnable

```
Class Myclass implement Runnable
class Myclass extends Thread
{
    public void run()
    {
    }
}
Myclass obj = new Myclass();
Thread t = new Thread(obj);
t.start();
```

→

```
class MyThread extends Thread
{
    public void run()
    {
        for( int i=0 ; i<1000;i++)
        {
            S.o.p(i);
            boolean stop = false;
            if (stop == true)
              return;
        }
    }
}
class Demo
{
    p.s.v.m( )
    {
        MyThread obj = new MyThread();
        Thread t = new Thread ( obj );
            t.start();
    }
}
```

Assignment:

Write a java program that currently implements producer, consumer problem using concept of interthread communication.

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

```
Producer obj1=new Producer();
Consumer obj2=new Consumer();
Thread t1= new Thread(obj1);
Thread t2= new Thread(obj2);
   t2.start();
   t1.start();
  }
}

Class Producer extends Thread
{
   Class belongs to lang package
   String Buffer sb;
   boolean dataprodoveq =false;
   Producer()
   {
      Sb= new StringBuffer();
   }
   public void run()
   {  Synchronised(Sb){
   for( int i=1; i<=10; i++)
   {
      try{
         sb.append(i+":");
         Thread.Sleep(100);  Suspend execution of thread till the time reach.
         S.o.p("Appending");
      }Catch(Exception e)
      {
      }
   dataprodoves =true;
      sb.notify()
   }}
}
```

```
class Consumer extends Thread
{
    Producer prod;
    Consumer (Producer prod){
        {
            this.prod = prod;
        }
    Public void run()
    { synchronized (prod.sb){
        try{
            while(prod.wait();
                !=prod.dataproclover)
                Thread.sleep(100);
        } catch (Exception e)
            {

            }
        S.o.p (prod.sb);
    }
}
```

- obj.nohify() methods realeses and object and sends a nohification to a waiting thread. that object is available.

- obj.nohifyall() is uesful to send nohification to all waiting threads at once that object is available.

- obj.wait() method is makes a thread wait for the object till it receives a nohification from nohify method or nohify all method.

* Thread priorities :-
- range : 0 to 10
min max
5 - default priority
- Priority assigned to thread is done by JVM/Programmer
- Variable use to find priority :-
public static int min-priority;
max-priority;
norm-priority;
public final void setPriority(int priorityNo) // set Priority to thread.
public final int getPriority()
t1. getPriority() // by default we get 5.

Unit V

* functional programming :-
- function is basic building block.
- Ex. LISP
- This language requise Interpreter to compile and execute code.
- Machine dependent - it require only compile code to execution

features-
- machine independent language.
- It uses iterative design methodology.
- It supports high level debugging.
- functions are defined as first class functions.
- LISP is expression based language.
- It supports all types of data type object, structures, List, vectors, arrays, set, trees, hash tables and symbols.
- Variables are immutable.

LISP - developed by John Mccarthy in 1958 at MIT.

High level programming language.
Expression base language.
Ex:-

(write-line string) ← Syntax

(write-line "Hello world")

Extension to save source file .lisp
(terpri) - use to print output on newline.
LISP - List processing programming language.

* Syntax in LISP:-
- It provides input/output library, microsystem and control structures for manipulation of data.
- Basic building block
  - Atoms
  - List (a b (e f g))
  - strings "Hello to LISP"
- comments given using ; (semicolon)

ex: ; Addition of two numbers
(write-line "addition")
(write (+ 2 3))

* function in LISP:
- functions are recursive.
- Syntax:-

```
(defun functionname (parameters)
 " documentation string"
  ;body of function
)
```

- Ex:-

```
(defun add-no ( a b)
 " Addition of nos"
    ( + a b)
)
(write-line ( add-no( 10 20)))
```

Find average of five numbers:

```
( defun average ( a b c d e)
 " Addition of nos"
   (/(+ a b c d)/5)
)
```

(write (addmage (10 20 30 40 50)))

* Write a function to calculate area of circle when the radius of the circle is the given as argument.

```
(defun ( Area (a)
        " Area of circle"
        (* 3.14 * a * a)
 )
(write(Area(4)))
      'OR'
(defun ( Area-circle(rad))
        " find area of circle"
        (terpri)

    (Format t  "Radius : ~5f" rad)
    ( Format t "&Area : ~10f" (* 3.14 *ad rad)
 )
(write (Area_circle 5)
```