Object Oriented Programming

* Every prog is divided into data & functions.
* Memory area is created for data & functions
* Data & functions can be worked with objects
* Data moves from one function to other with the help of objects

OOPS Concepts

* class :- * Entire set of data & code of an object.
  * It is a user defined data type
  * It creates no of objects.
  * Ex:- fruit is a class.

```
class sample.
{
  data members;
  member functions;
}.
```

* Object * It is an instance of class
  * It is basic runtime entity.
* * Encapsulation :- * The wrapping up of data & fun into a single unit.
* * Inheritance :- * Acquiring properties of one class to another class

```
class A          | class B extends A
{                |  {
}                |  }
```

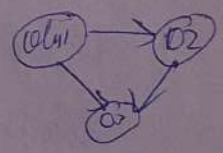
Polymorphism :-

* ability to take more than one form.
* Ex :- function overloading.

```
sum()            | sum(int x, int y)  | sum();
{                |  {                 | sum(5,2);
}                |  }                 |
```

Dynamic Binding :-

* 2 types :-
* Static binding :- * compile time.
  Dynamic binding :- * Run time



Message Communication

* Communications of objects to one & other.

Data Hiding

* In this data can be caused only in the functions of some class

Data Abstraction : (ADT) | The technique of creating new datatype.
↓ class | Eg :- ATM.

Java pure Object Oriented or Not.

* Java is a OOP but not purely OOP
* Because it [supports primitive] data type
* Whenever we represent Data & functions based on only objects then it is called pure OOP.
* Bcoz we represent data & functions with & without functions

Java Features

* Compile & Interpreter

Java code — source code.
↓ compiler.
Byte code.
↓ Interpreter.
Executable code

* Platform Independent & portable

Java programs can be easily moved from one computer to another.

* Object Oriented

Java language is truly OOP lang.

* Robust & Secure

Strong language with exception Handling

* Distributed:
In Java there is a facility to share a single project to different programmers

* Simple, small & familiar

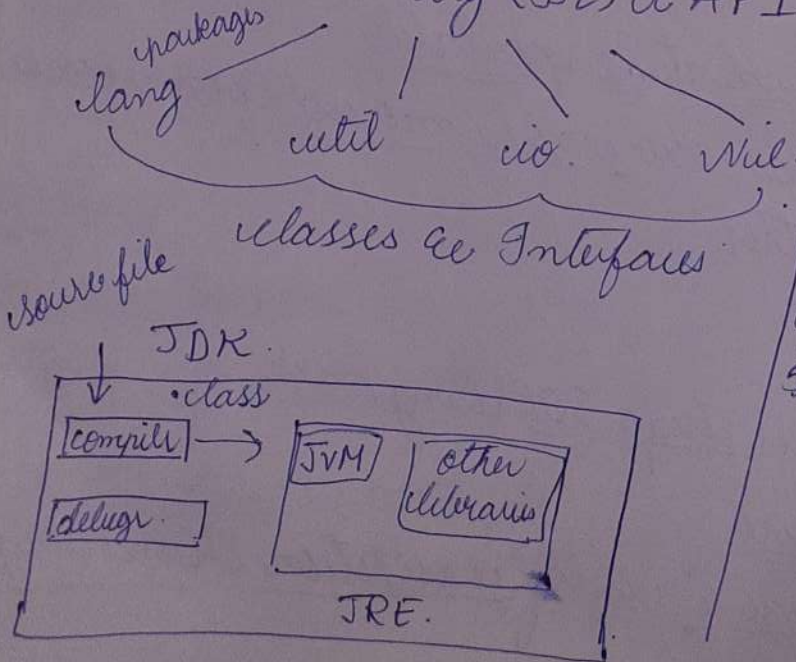
* Multi threading :- more than 1 task / handling multiple tasks simultaneously

* High Performance:- Java programs execution speed is increased.

* Dynamic & Extensible
class libraries & methods are linkable.

Java Environment

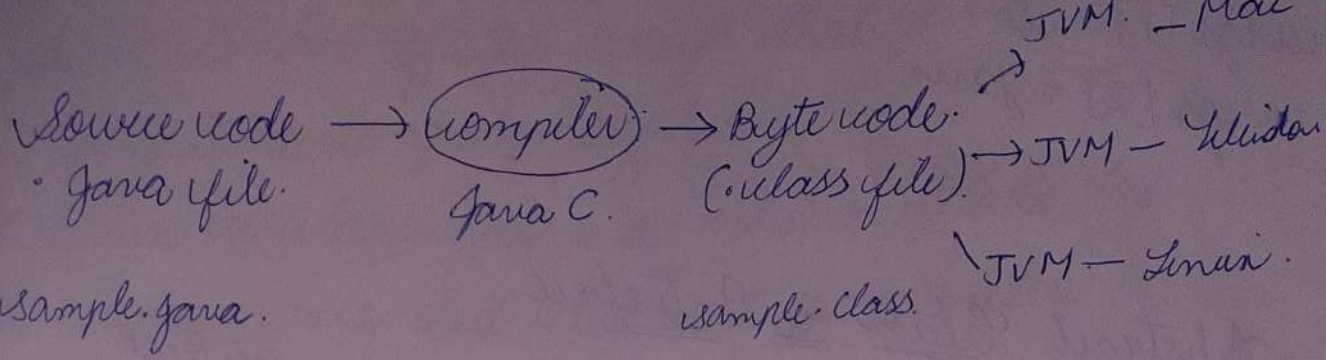
→ Java Standard Library (JSL) a API.



1) Applet Viewer
2) Java - Java Interp.
3) Javac - Java compile
4) Javadb - find errors
5) Javap - dissimbler.

JVM : Java Virtual Machine

OS



Eg sample.java. sample.class

JVM : platform independent
Java : high level.
Javac is used.

Command Line Arguments

* It is an info that directly follows the program's name when on the CL when it is executed.

Eg:

```
public class CommandLinePrg.
{
    public static void main (String args[]).
    {
        for(int i=0; i<args.length; i++)
        {
            System.Out.Println ("args["+i+"]: "+args[i]);
        }
    }
}
```

Java commadLineprg.java.

G:\> javac commandLineprg.java.

G:\> java commandLineprg welcome to java.

args[0] = welcome
args[1] = to
args[2] = java

Abstract Class & Interface

* Abstract class can have access modifiers for members. Interface can have only public members
* Abstract class may or may not contain abstract methods. Interface can't have defined methods
* Abstract class can have static or non static members. Interface can have only static members
* Abstract class can have final / non final members. Interface can have only final members
* Abstract class has constructor. Interface has no constructor
* AC doesn't support multiple inheritance. Interface supports
* Abstract keyword is used to declare abstract class. Interface keyword " " " interface
* AC can be extended using keyword extends. I, can be implemented using keyword implements

Method Overloading

* It is one of the ways that Java implements polymorphism.
* When a class has 2 or more methods by the same name but different parameters

* Eg:-

```
public class overloadex
{
public void sum(int a, int b)
{
System.out.println(a+b);
}
public void sum(int a, int b, int c)
{
System.out.println(a+b+c);
}
public static void main(String args[])
{
overloadex oe = new overloadex();
oe.sum(10,30);
oe.sum(2,4,6);
}
}
```

* Method name is same but arguments are diff

Method Overriding

* Overriding means defining a method in subclass by using same signature of the same method in super class

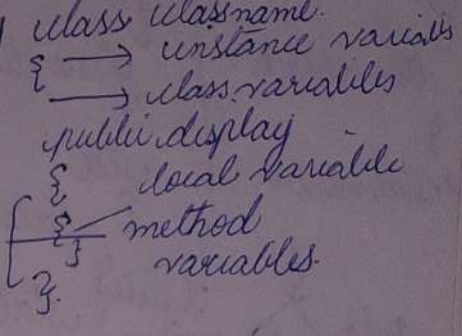
Eg:-

```
class one
{
public void display().
 {
  System.out.println("I'm from class one");
 }
}

class two extends one
{
public void display().
 {
  System.out.println("I'm from the class two");
 }
}

public static void main(String args[]).
{
two t = new two();
   t.display();
}
}//class close
```

Scope of a Variable

* Scope means the area of program where the variable is accessible.

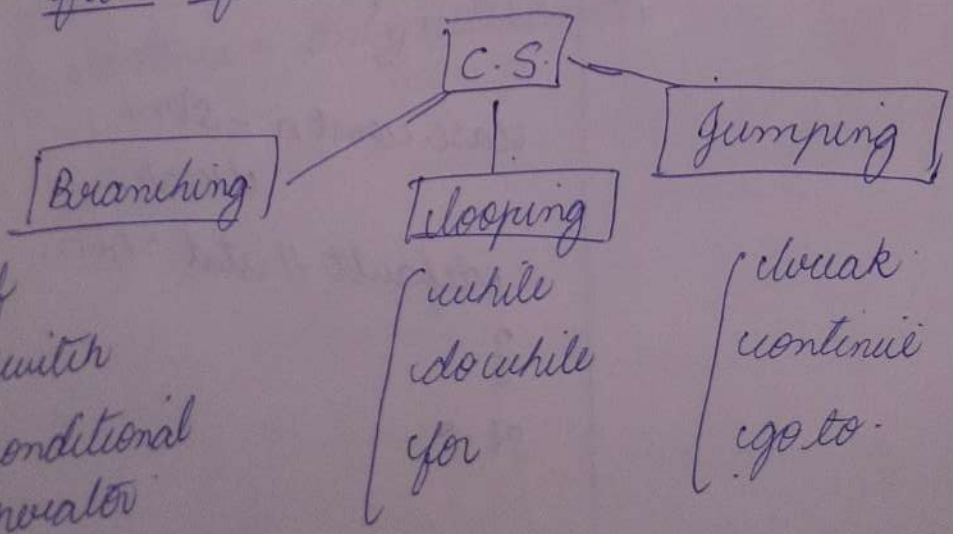
1) Instance variables - objects
2) class variables - class
3) Local variables - method.



1) Instance variables are created & associated with objects.
2) Class variables are global to class.
3) variables which are declared in a method are called local variables.

Control Statements

* Set of instructions is called prog.
* To handle some statements we follow some controls
* flow of control are called control statements.

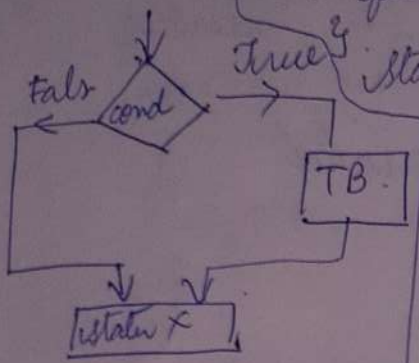


if statements

simple if | if-else | nested if | else if

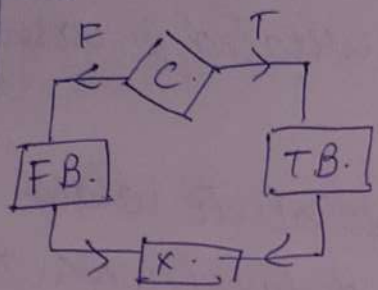
simple if

```
if(condition)
{
true block
}
statement-x
```



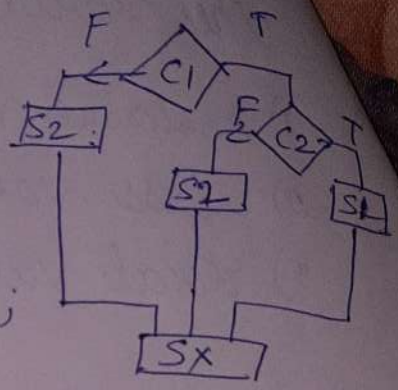
if-else

```
if (condition)
{
true block
}
else
{
false block
}
statement x
```



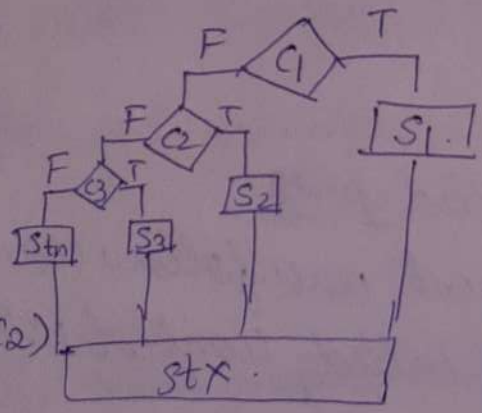
nested if

```
if (condition 1)
{
if (condition 2)
{
statm 1;
}
else
{
False block;
}
}
else
statement 2;
statement x;
```



else if

```
if (C1)
{
S1;
}
else if (C2)
{
S2;
}
else if (C3)
{
S3;
}
else
stn;
statement X
```



Switch

```
switch (expression)
{
case const 1: st1
              break;
case const 2: st2
              break;
      :
case const n - stn;
              break;
default // stat - n+1;
}
st-1
```

Conditional Operator
?:

(exp1) ? exp2 : exp3.
T
F

Looping :- No of iterations
↳ body of loop.
entry exit
↳ control of loop

While :-

```
initialization;
while(cond)
{
  block;
  ++, --;
}
SX;
```

do while

```
initialisation;
do
  {
    block;
    ++, --;
  }
while(cond);
```

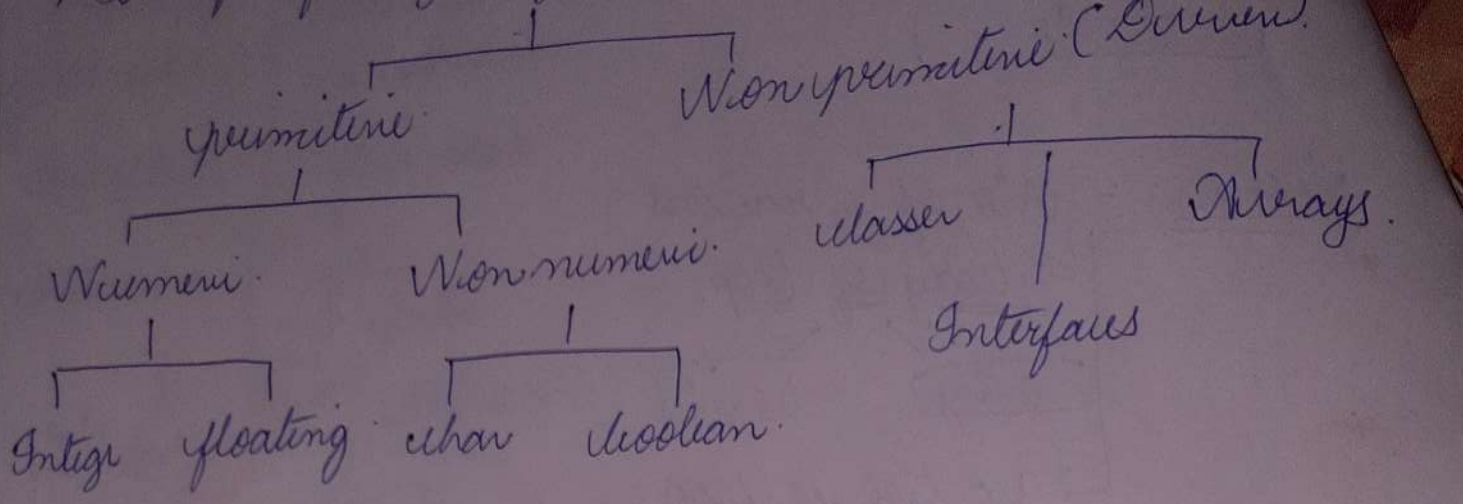
for :- large iterations
1) simple for
2) for each - unchanged
3) Labelled / Nested

```
for(init, cond, incl/dec)
  {S;
  }
SX;
```

break = break
continue = only, iteration

Data types

* It specifies type of a variable



Boolean - 1 bit
char - 2 bytes (unicode)
byte - 1 byte
short - 2 bytes
int - 4 bytes
long - 8 bytes
float - 4 bytes
double - 8 bytes

Syntax :

```
Datatype variablename = value;
int x=10;
float y = 3.14f;
char a;
Boolean c0 = "false";
```

Constructor

* It is a special type of function.
* A constructor is invoked whenever object is created.
* Constructor name is same name as class name.
* if we haven't defined any constructor then java will automatically define a constructor
* The complement (reverse) of constructor is destructor.
* Local objects r created when their block is entered

* When the block is left, Destructor is invoked.

* Constructors (User defined).

1) No argument constructor

2) parameterized constructor

* Constructors can only used for initialization
* " " not used for I/O operations.

1). public class conex.
{
int n;
conex()
{
n=10;
}
}
public class conmain
{
public static void main(String args[]).
{
conex e1 = new conex();
conex e2 = new conex();
System.out.println(e1.n + " " + e2.n");
}
}

2) parameterized constructor.

```
public class conex
{
 int x;
 conex(int j)
  {
    x=j;
  }
}

public class conmain
{
 public static void main(String args[])
  {
   conex e1 = new conex(100);
   conex e2 = new conex(200);
  }
}
```

Constructor Overloading

* Java supports constructor overloading.
* We can create multiple constructors with same name but diff parameters.

Ex: class Employee

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

Q.9.

```
Employee (int x, String n).  // constructor
{
uid = x;
name = n;
}

Employee (int x, String n, int y)  // con 2.
{
uid = x;
name = n;
age = y;
}
// display method
public void display()  // to print values
{
System.out.println("id is" + id);
"    "    "  ("name is" + name);
"    "    "  ("age is"  + age);
}

// main method
public static void main(String args[]).
{
Employee e1 = new employee (1234, "Tina");
Employee e2 = new employee (1234, "Tina", 20);
e1.display();
e2.display();
}
}
```

Super Keyword

* It invokes immediately super class members.
  1) Variables
  2) Methods
  3) Constructors

```
class one
{
  public void display()
  {
    S.O.P("I'm from class one");
  }
}
class two extends one
{
  public void display();
  {
    super.display();
    S.O.P("I'm from class 2");
  }
}
public class third
{
  public static void main(String args[])
  {
    one o = new two(); // one ref & two obj
    o.display();
  }
}
```

Final Keyword

* Used to restrict the user.
* Can be applied on
1) Variables, 2) Methods 3) Class.
* The final modifier is used for finalising the implementation of classes, Methods & Variables

Ex:-
```
public class example.
{
final int x=10;
String name = "Tina";
public void change().
{
x=20;  //will give an error
}
}
```

* A final method can't be overriden by any subclass

Ex:-
```
public class example
{
public final void change()
{
}
}
```

* Final Class :- Main use - A class being declared as final is to prevent the class from being subclass

public final class example
{

}

Inheritance

* Acquiring the properties of one class to another class
* reusability is the main use

class A.
{
}
- super class
- parent class
- base class

class B extends A.
{
}
- sub class.
- child class.
- derived class.

1) A - Base
   ↓
   B - Derived.

2) A B - base
   ↘ ↙
   C - derived

(we can't access multiple inheritance so we use Interface)

Types

1) Single - one base class inherit to one derived class
2) Multiple - more than 1 base class inherit to one derived class
3) Hierarchical - one base class inherit to many DC.
4) Multilevel.

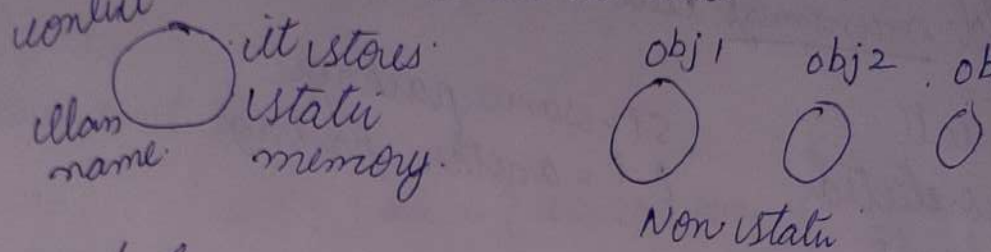
3) A - super
   B C D
   1 1 1
   sub class.

4) A - super
   ↓
   B - Intermediate super
   ↓
   C - sub class

Static Keyword

1) Static - It access only static data.
2) Non-static - It access both static & non static.

* Static members are common to all objects.
* Static data is stored in context.



* We can declare static variables & static methods

Abstract Keyword (Classes & Methods)

Abstract class :- A class is said to be AC if it contains one or more Abstract methods.

Abstract method :- * Any method is said to be abstract method. That method has no body or no implementation.
* can be declared by using the keyword abstract.

Note :- Abstract class can't be instantiated by using new operator.

Visibility control

(Access Modifiers / Access Specifiers)

* To access classes, constructors, methods & fields
* To take advantage of encapsulation, we should minimize access

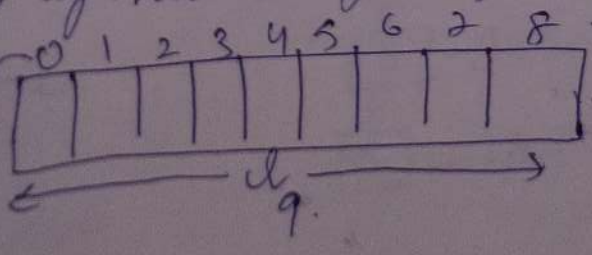
1) private
2) protected
3) default
4) public

S.P = same package
A.P = another package

class/ access	private	Default	protected	public
Own class	✓	✓	✓	✓
subclass (S.P)	x	✓	✓	✓
class (S.P)	x	✓	✓	✓
subclass (A.P)	x	x	✓/x	✓
class (A.P)	x	x	x	✓

Arrays

* Array is similar type of data collection name
* An array is a collection of homogeneous elements.
* An array is a container object that holds a fixed no. of values of single type.



* Syntax : datatype arrayname[ ];
* Ex : int arr[] arrayname;
* 2 types — Linear [1D]
  \ Non Linear [2D, 3D]

* Creating & Initializing

```
int [] a;
a = new int [10];
```

```
a[1] = 100;
a[1] = 200;
a[2] = 300;
```

```
int [] a = {100, 200, 300...};
```

2D Array (Double Dimensional, Tabular, rectangular)

Syntax: `Datatype [][] array = new Datatype [n][m];`
(array name; n = rows, m = cols)

Ex: `int[][] a = new int [3][4]`

Eg: `int [][] a = {{1,2,3}, {4,5,6,9}, {7}};`

rows \ cols	0	1	2	3
0	1	2	3	
1	4	5	6	9
2	7			

3D Array

Syntax: `Datatype [][][] array name = new Datatype [3][4][2];`

- [3] — array dimension
- [4] — row size
- [2] — column size

Strings.

* Sequence of characters
* In Java, String is an object that represent sequence of characters
* java.lang.String is used to create string object

* How to create a string ?
  1) By using string literal
  2) By using "new" operator

Serializable, Comparable
String → charsequence.

1) String s = "Good";
→ JVM checks String constant pool".

2) String s = new String ("welcome");
→ JVM creates new string object in Heap.
→ "welcome" will be placed in the "String constant pool";

Eg:

```
public class Stringex
{
public static void main (String args[]).
{
String s1 = "welcome";
char name = {'K', 'I', 'T', 'E'};
String s2 = new String (name);
String s3 = new String ("Tina");
S.O.P (s1);
" " (s2);
" " (s3);
}
}
```

Immutable Strings (Unchangable)

1) charAt (index).
2) length ().
3) substring (beginIndex).
4) substring (beg, end).
5) equals ( ).
6) isEmpty ( ).
7) concat (str).
8) split ( ).
10) toLowerCase ( ).
11) toUpperCase ( )

* In java, String objects r unchanged.
* Eg:- class immstring
{
public static void main (String args[]).
{
String S = "Good";
S.concat ("Morning");
S.o.p (S);
}
}

String Buffer. (Mutable / Changable)

* This class is used to create mutable strings.
* This StringBuffer class in Java is same as String class except it is mutable.
* It can be changed.

Constructor

1) StringBuffer ()
2) StringBuffer (String str).
3) StringBuffer (int capacity).

StringBuffer Methods

1) append ()
2) insert ()
3) replace ().
4) delete ().
5) reverse ().
6) capacity ()
7) length ()
8) charAt ()

Ex:-

```
class StringBuffEx
{
  psvm (Sa[]).
  {
    StringBuffer sb = new StringBuffer ("Good");
    Sb.append ("Morning");
    S.O.P (Sb);
  }
}
```

Diff b/w StringBuffer & StringBuilder

→ StringBuffer, StringBuilder } mutable (changed)

Constructor :- 1) StringBuilder ().
2) StringBuilder (String str).
3) StringBuilder (int capacity).

StringBuffer	StringBuilder
1) less efficient	2) more efficient
2) thread safe (synchronized)	2) thread not safe (non synchronized)

Interface

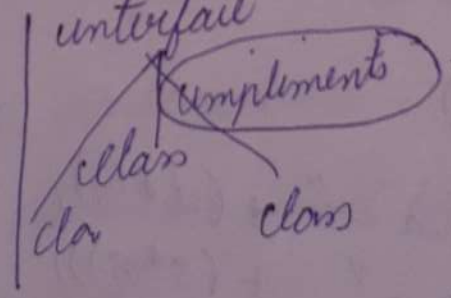
* it is a blue print of a class.
* mechanism to achieve Abstraction.
* In the place of Multiple Inheritance we use Interface

* Use
  1) Abstraction
  2) Multiple Inheritance.
  3) Used to achieve loose coupling.

* Syntax:

```
interface <interface name>
{ // declare fields ——— public static final by default.
  // abstract methods
}
```

* interface —(extends)— interface



Ex:

```
interface sample
{ void print();
}
class example implements sample
{ public void print()
  { SOP("Hello, GM");
  }
public static void main()
{ example e = new example();
  e.print();
}
}
```

Multiple Inheritance Ex.

```
interface measurement.
{
int l = 10, b = 20;
int side = 100;
public void x();
}
interface c1 extends measurmt.
{
  public void area();
}
interface c2 extends
 {
  public void perimeter();
 }
class ex implements c1, c2.
{
 public void area()
  {
   S.O.P("Rectangle area = " + (l*b));
   S.O.P("Square area = " + (s*s));
  }
 public void perimeter()
 {
   S.O.P("Rectangle perimeter = " + (2*(l+b));
   S.O.P("Square perimeter = " + (4*s));
  }
 public void x()
 {
```

```
s.o.p("x method");
}
public static void main(String args[]).
{
ex e = new ex.
e.area();
e.perimeter();
}
}
```

Package

* Package is a keyword.
* package is a collection of classes & Interfaces.

i) User defined
2) predefined - import java.util.*

Syntax package packagename;.

* The package statement must be 1st statement
* It terminate with ';'
* Package & classes are must be in same path.

Pac
cls
Int

ex: package one;
public class first
{
public void first()
{ s.o.p("Im from 1st package");
}
}

compilation

javac -d . first.java ↵

* . represents current directory

How to declare & import user define Package

* package mypack
public class example
{
public void msg()
{
S.O.P("my first package");
}
}

* save as example.java
* my pack folder is created which contains class example.

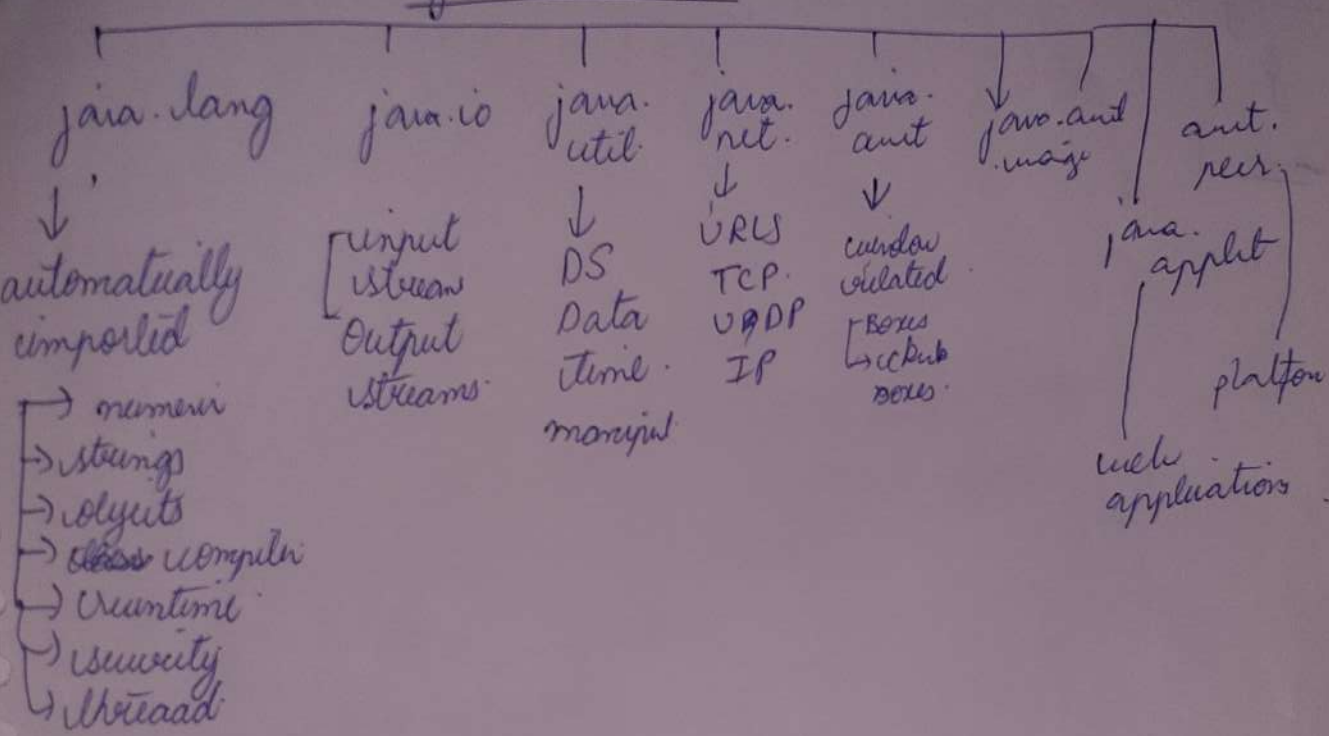
package mypack2;
import mypack.example;
public class second
{
public static void main(String args[]).
{
example e = new example();
e.msg();
}
}

* save as second.java.
* compile as javac -d . second.java

Interport.

java mypack2.Usend;

System Packages.



Hiding Classes.

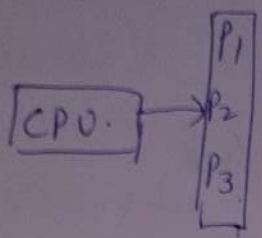
* When we import a package by using (*) all public classes are imported.
* When we want to hide a class from outside of package those r declared as "Not public"

Ex. package mypack;
public class myclass
{
public void display()
{

```
S.o.p("my 1st class");
}
}
class example  // (not public) can be unpublic
{
public void display()
{
S.o.p("example class");
}
}
```

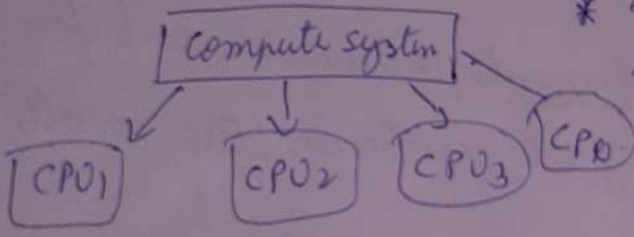
Multi programming, processing, Threading, tasking

Multi programming



* The concurrent residency of more than 1 program in the main memory is called multi program.
* In list multiprog currently executing program finished its execute.

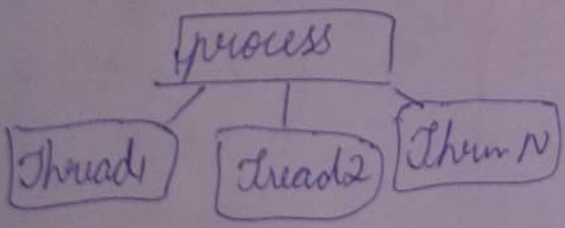
Multi processing



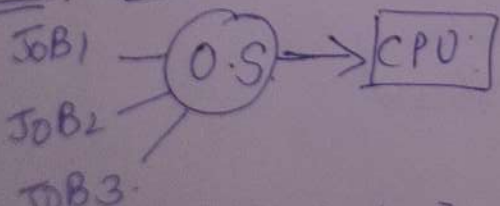
* When 1 system is connected to more than 1 processor which collectively work for the completion of the task.

Multi Threading

* It is a conceptual programs paradigm where a process is divided into no of sub process called Threads.



Multi Tasking



1) single user multitasking
2) multi user multi tasking

Thread

→ A thread is a similar to program that has single flow control.

Heart - run()

Creating Threads

1) By extending Thread class
2) By implementing Runnable interface

1) * Define a class as extends Thread class
   * Override run() it contains the code.

2) Runnable
   * Define a class that implements runnable interface
   * run() method overriding it contains the code

1)
```
class threadex extends Thread.
{
Public void run()
{
 s.o.p("1st thread");
}
Public static void main(String args[]).
{
threadex t1 = new threadex();
t1.start(); // life cycle of thread
}
}
```

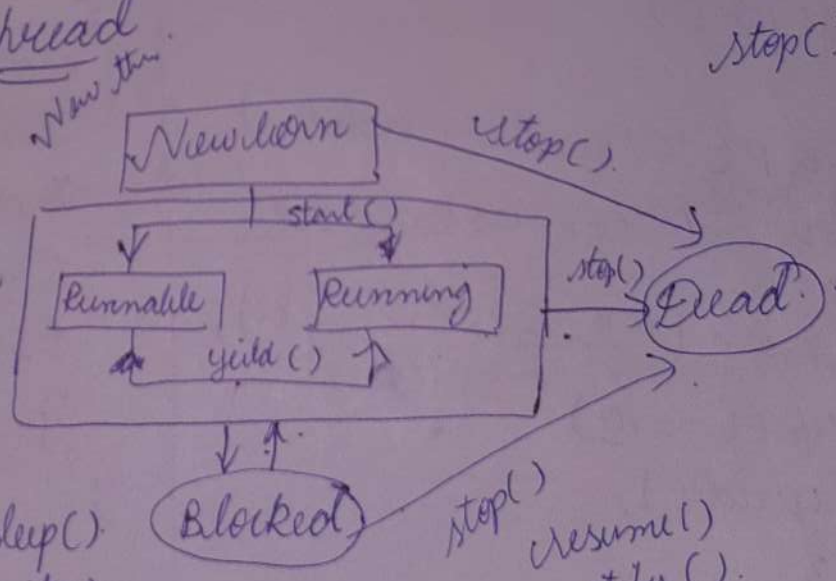
2) class example implements Runnable

```
{
  public void run()
  {
    p.s.o.p("1st Thread running");
  }
public s.v.m (s a[]).
{
  example e = new example();
  Thread t1 = new Thread (e); // extra
  t1.start().
}
}.
```

=

Life Cycle of Thread

1. New born
2. Runnable
3. Running
4. Blocked
5. Dead



Suspend() — resume()
wait() — Notify()

```
//class
class Thread1 extends Thread.
{
 public void run().
 {
  p.s.v.m S.o.p ("Thread1 is started").
   for (int i=0; i<20; i++).
   {
    S.o.p ("Thread1 : " +i).
    if (i==5)
     stop();
   }
 }
}
```

```
//class2
class Thread2 extends Thread
{
 P V M ()
 {
  S.o.P ("Thread2 started").
  for (int i=1, i<20, i++)
   {S.o.p ("Thread2" : +i);
   if (i==10)
    yield();
   }
 }
}
```

```
//main method
class example
{
 p.s.v.m (String a[]).
 {
  Thread t1 = new Thread1();
  Thread t2 = new Thread();
  t1.start().
  t2.start().
 }
}
```

Thread Priority

* Thread Priority is set b/w 1-10.
* MIN_PRIORITY - 1
  NORM_PRIORITY - 5
  MAX_PRIORITY - 10.

```
class Sample extends Thread
{
public void run()
{
S.O.P("Running thread");
S.O.P("welcome");
}
public static void main(String a[])
{
Sample S1 = new Sample();
Sample S2 = new Sample();
S1.setPriority(Thread.MIN_PRIORITY);
S2.setPriority(Thread.MAX_PRIORITY);
S1.start();
S2.start();
```

Synchronisation

* Synchronisation in Java is the capability to control the access of multiple threads to any shared resource

1) To prevent thread synchronisation interface
2) to prevent consistancy problem

1) process synchronisation
2) Thread synchronisation

2)
1) Mutual exclusive Thread synchronisation

Lock monitor

* synchronized method

```
* class Thsample
  {
   synchronized void print(int n)
   {
    for(int i=1; i<=5; i++)
    {
     S.O.P(i+n);
     try
     {
      Thread.sleep(200);
     }
     catch(Exception e)
     {
     }
    }
   }

class Thread1 extends Thread
{
 Thsample S;
 Thread1(Thsample S)
 {
  this.S = S;
 }
```

```
public void run()
{
  S.print(10);
  }
}
```

```
class Thread2 extends Thread
{
  Thsample S;
  Thread2(Thsample S)
  {
    this.S = S;      //this
  }
    p v r ()
    {
      S.print(100);
      }
    }
```

* After we get one thread ka output then we get other thread ka output

```
p c Test
{
  p s v m [s a[])
  {
    Thsample TS = new Thsample()
    Thread t1 = new Thread1();
    Thread t2 = new Thread2();
```

```
t1.start();
t2.start();
  }
}
```

2) Synchronized Block.

* It can be used to perform synchronisation on any specific resource of the method.
* Synchronized block is used to block can object for any shared resource.
* Scope of synchronised block is smaller than method. It is subpart of method.

Syntax:
```
synchronized (Object reference variable)
{
  //code
}
```

Ex:
```
class Sample
{
  void print(int n)
  {
    synchronized (this)
    {
      for (int i=1; i<=5; i++)
      {
        S.O.p("n" + i);
        try
        {
          Thread.sleep(200);
        }
        catch (Exception e)
        {
        }
      }
    }
  }
```

```
class Thread1 extends Thread
{
 Sample S;
 Thread1 (Sample S)
 {
   this.S = S;
 }
 public void run()
 {
   S.print(5);
 }
}
```

```
C Thrd2 e T.
{
  Sample S;
Thread2 (Sample S)
 {
   this.S = S;
 }
public void run()
 {
  S.print(10);
 }
}
```

```
//main m
p c example
{
p s v m(sa[])
{
Sample S = new sample();
Thread t1 = new Thread1(S);
Thread t2 = new Thread2(S);
 t1.start()
 t2.start()
}
```

Static Synchronization

* If you make any static method as synchronized the lock will be on class not an object
* Class fully gets locked.
* Static members accessed by class name.
* Non static members accessed by object name.

Ex: class Thsample.

```
{
synchronized static void print(int n).
 {
   for(int i=1; i<=5; i++).
    {
     s.o.p(n+i);
     try.
     {
      Thread.sleep(200);
     }
     catch(Expression e);
     {
     }
   }
 }
}
}
}
```

```
class Thread1 extends Thread
{
pvr()
{
Thsample.print(5);
}
}
```

```
class Thread2 extends Thread
{
pvr()
{
Thsample.print(10);
}
}
```

```
// main method
public class example
{
public static void main(String args[])
{
Thread1 t1 = new Thread1();
Thread2 t2 = new Thread2();
t1.start();
t2.start();
}
```

Types of Error

* A mistake might lead to an error causing the program to produce unexpected results.

1) Compile Time – Syntax problem (Ex; miss ";" mis)
2) Run time – a[5] – 0,1,2,3,4.
   but a[6] = 20 ✗
   Out of Bounds
   a/y = 3/0 = ∞ ✗

Exceptional Handling (Run time error) : Solution

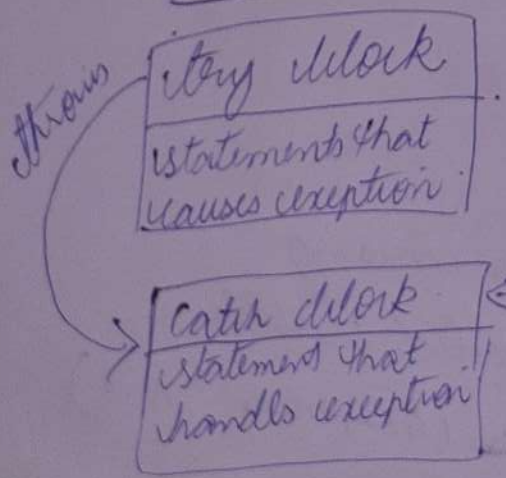
* Exception is a condition that is caused by a run time error in program

* 1. find the problem
  2. inform the error
  3. Reduce error info
  4. Take corrective action

* x/y
  3/0 = ∞ (infinity)

* Exception Types
  1. Arithmetic Exception
  2. Array store Exception
  3. IO store Exception
  4. Stack Overflow exception

* Mechanism



Ex.

```
class exceptionhandling
{
 p s v m (sa[])
 {
  int a=10, b=5, c=5;
  try
  {
  int x=a/(b-c);
  }
 catch (Exception e)
 {
  s.o.p ("Division by zero");
  }

 int y=a/(b+c);
 s.o.p ("y =" +y);
  }
```

Multiple Catch.

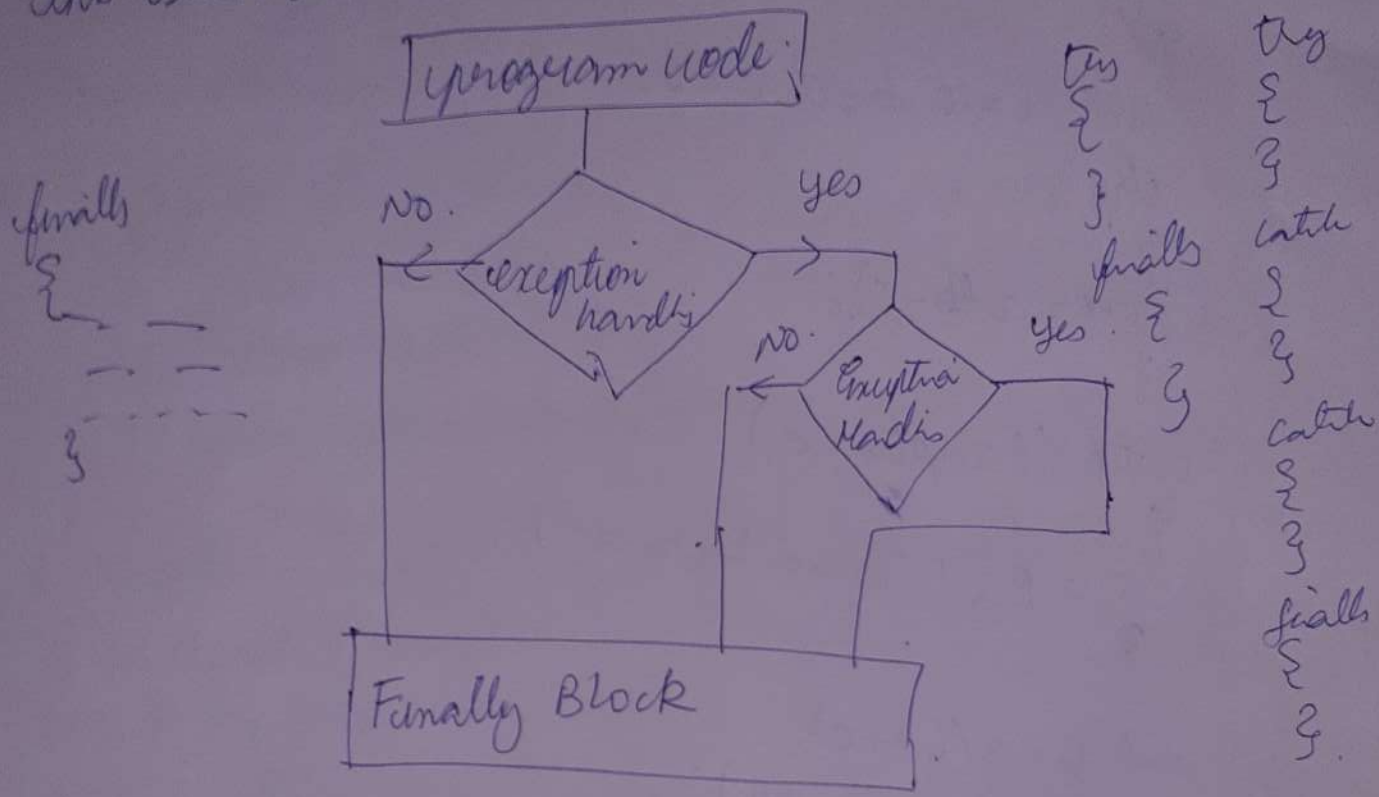
It is possible to create multiple catchs for single try.

```
try
 {
  }
catch (Exception1 obj)
 {
 }
```

```
catch (Exception2 obj)
{
}

catch (Exception3 obj)
 {
 }
```

Finally Statement

This statement uses to handle the exception



finally
{
}

try
{
}
finally
{
}

try
{
}
catch
{
}
catch
{
}
finally
{
}

Throwing out own Exception

"throw"

eg: throw new Exception name ();

import java.lang.Exception;

class myexception extends Exception

{ myexception(String message)

{ super(message);

}

```
class testmyexception
{
public static void main (String args[])
{
int x=5;
int y=100;
try
{
float z =(float)x/(float)y;
if (z<0.01)
{
throw new mymyexception("no is too small");
}
}
catch(Exception e)
{
S.o.p.( "Catch my exception");
S.o.p ("e.getmessage());

Finally
{
S.o.p("I am also more");
}
}
}
```

Applets

Embedded with Java + HTML.

* Applet is a small Java program.
* "applet viewer" or web browser.
* It can perform arithmetic operations, display graphics, play sounds, animations.

1) Local Applets
2) Remote Applets

1) Local Applets

* An applet developed locally & stored in a local system is known as local Applet
* This local system does not require internet connection

Steps involved in applet creation -

1) Builds an applet code (.Java file).
2) Creating an executable applet (.class file)
3) Designing a web page using HTML tags
4) preparing <Applet> tag.
5) Creating HTML file for a webpage Incorporating <Applet>

6) testing applet code

Applet General format

```
import java.awt.*;
import java.applet.*;
public class Hello extends Applet
{
public void paint(Graphics g)
  {
    g.drawString("Hello", 10,100);
  }
}
```

Hello.java

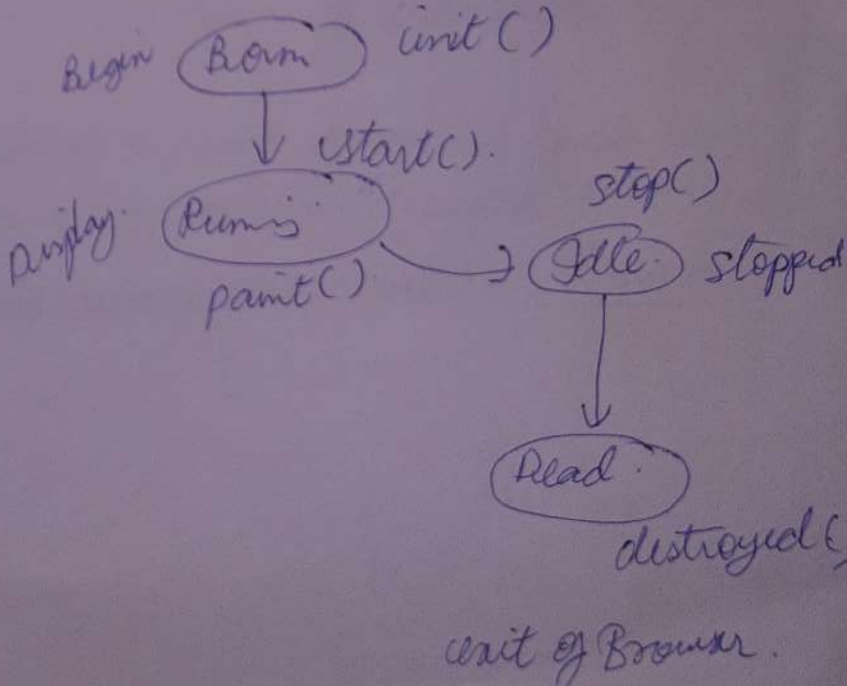
Component
↓
Container
↓
Panel
↓
Applet

```
<applet*
code = Hello.java
width = 400;
height = 200>
</applet>
```

First.html

Applet Life Cycle

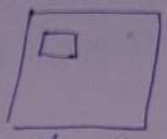
1) Born
2) Running
3) Idle
4) Dead

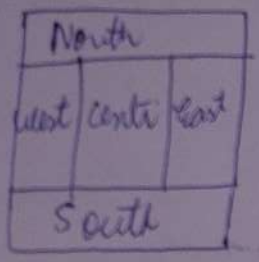


Java Layout Manager

* Used to arrange components in a particular manner.
* It is an interface i.e implemented by all classes of a layout manager.

① Border Layout Manager



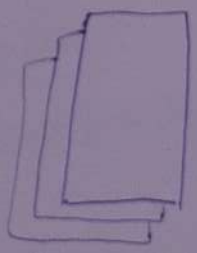


Each region can contain only one component.

Eg- Frame f = new Frame ("layout");
f.setLayout (new BorderLayout ());
f.add (b1, "North");
.....

② Card Layout Manager

- Arranges each component in the container as card.
- Only one card is visible, & the container acts like stack of cards

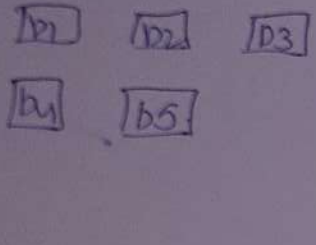


Frame f1 = new frame ();
Button b1, b2; //
CardLayout card;
Card = new CardLayout (40, 30);

f.add ("card1, b1);
f.add ("card 2, b2);
b1. addActionListener(this);
pub void actionPerfor(ActionEvent e)
{ card. next (f);
// used to display next.

Flow Layout

* used to arrange components in a line, one after the another.
* default layout.




```
f.set layout (new FlowLayout());
f.add f(b1);
   :
(b1)
```

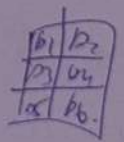
Grid Layout

* Container is divided into a grid of cells — rows, columns
* Each cell accomodate one component

3 rows 4 colums

1	2	3	4
5	6	7	8
9	10	11	12

```
f.set layout (new GridLayout(3,2));
f.add(b1);
   :
   :
b6.
```



Swings & AWT

AWT	SWING
* components r platform dependent	* components r platform independent
* heavy wt	* light wt
* dosnt support pluggable look & feel	* support pluggable look & feel
* less components	* more powerful components
* dosnt follow MVC Architecture	* follow MVC architecture

Event Handling

Event is the change in state of object or source
eg:

* Event Handling is the mechanism that controls the event & decide what should happen if a event occurs
* Delegation Event Model

Source
Is an obj on which event occurs

Listner
Generate response to event

Mouse Events

Mouse Listner interface contains 5 methods.

1) Mouse clicked
2) Mouse pressed
3) Mouse released
4) Mouse entered
5) Mouse Exited

Java Tokens

* Smallest individual units in a program
* 5 types of tokens
  1) Keywords - 60 [predefined words] int, float if, else
  2) Identifier - programmed defined Tokens
  3) Literals - Sequence of Chars (Integer, String, char, bool)
  4) Separators - Symbol ( ), { } [ ], . , ;
  5) Operators - a+b
     operator

Identifiers

→ They can have alphabets, digits - _ $
→ They must not begin with a digit
→ Upper case & lower case r diff
→ They can be any length

NC :- 1) All public member = Small letters
2) Input stream [use defin

Vectors

* Vector belongs to java.util.
* This class can be used to create a generic dynamic array.
* It holds objects of any type.

1) addElement(item)
2) elementAt(position)
3) size()
4) removeElement(item)
5) removeAllElement()

Eg:-
```
import java.util.Vector
class A
{
public void message()
{
S.O.P("HELLO");
}
}
class example
{
P S V m (S a[])
{
Vector v = new Vector();
A a = new A();
String S1 = "RAJU";
v.addElement(a);
v.addElement(S)
S.O.P(v.size());
A b = (A)v.elementAt(0);
b.message
}}
```

Wrapper class :

↓

java.lang

↓

& wrapper

* In Java, provides wrapper class as a mechanism to convert primitive types to object & object types to primitive

primitive type → object (auto boxing)
object → primitive (unboxing)

primitive type	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Serialization & Deserialization

* Serialization is a process of converting an object into a stream of bytes which can be persisted to a disk.
* A java object is serializable if its class or any of its superclass implements java.io.
* The entire process is JVM independent.

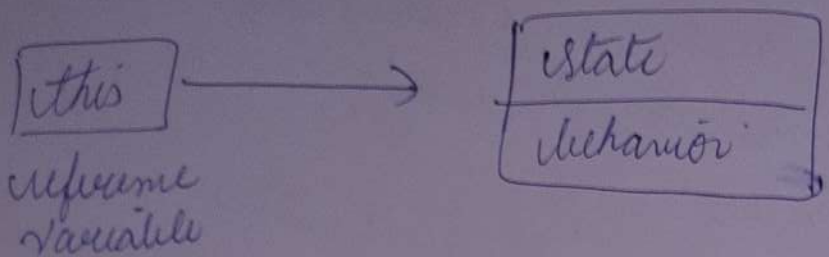
Enumeration

* The Enum in Java is a data type which contains fixed set of constants
  eg Mondas, ..., North ....
* All constants must be in capitals
* Enums are used to create our own data type.
  like classes
  eg enum Season{ WINTER, SPRING, FALL }
* It defins a class type

Iterator

* removeal Cursor
* Used for iterating collection object

This Keyword



* reference variable that refers to current object
* this keyword can be used to refer current class instance variable
* It resolves the problem of Ambiguity