

C

⇒ 'C' developed by Dennis
M Ritchie at Bell Labs in
1972

C++

⇒ C++ developed by Bjarne
Stroustrup in 1980's

⇒ It doesn't support OOPS

⇒ It supports OOPS

Concept

⇒ It is a Top down approach

⇒ It is bottom up approach

⇒ Operator overloading is not possible

⇒ Operator overloading is possible

⇒ Multiple declaration of

Global variable are allowed

⇒ Multiple declaration of
Global variable are not
allowed.

⇒ Input & output functions

⇒ Input & output functions

are printf(), scanf()

are cin & cout.

⇒ Collocate, malloc() functions

⇒ To allocate memory we

are used to allocate memory

Use new operator, To

& free() is used to delete

deallocate memory we

⇒ 'C' is not a secure

Use delete operator.

⇒ It supports union & structure.

⇒ It is secure provide more

security compare with 'C'.

⇒ It supports class.

C++

→ C++ developed by

→ C++ is a platform dependent.

→ it support the operator overloading

→ Here we use compilers only

→ it support call by value & call by reference.

→ it support pointer concept.

→ methods ^{are} defined inside the class & outside the class in C++.

→ static memory - memory allotted at compile time

→ C++ is procedural & oops lang.

→ Character set include the 8-bit ASCII characters.

Java

→ Java developed by James Gosling and 1st jdk 1.0 released on 1996.

→ Java is a platform independent.

→ it not support the operator overloading

→ we use compilers & interpreter

→ it support only call by value.

→ it will not support pointers.

→ methods are defined inside the class in java.

→ Dynamic memory - memory allotted at runtime by JVM.

→ Java is object oriented programming language.

→ Character set includes the 16-bit unicode character.

Note: ASCII - American standard code for information interchange it restricted to use 256 char (0-255). each char use 1 byte. where as in unicode - universal standard for character encoding it allows 65536 char each char occupies 2 bytes of memory in unicode.

Types of java applications:

- Standalone Applications: it also known as desktop appⁿ or window-based appⁿ. these are traditional software that we need to install on every machine.
eg: media player, antivirus. (awt & swings are used to creating standalone appⁿ).
- Web applications: An application that runs on the server side and create dynamic page, is called web appⁿ currently, servlet, jsp, struts, spring, hibernate, JSF are used for creating web appⁿ in java.
- Enterprise application: An application that is distributed in nature, such as banking application etc. EJB is used for creating Enterprise appⁿ.
- Mobile Application: An application that is created for mobile devices. currently android & java me are used for creating mobile Appⁿ.

Java platforms / Editions: (4 platforms)

1. java SE (java standard Edition):

it is a java programming platform. it includes java programming APIs such as java.lang, java.io, java.net, java.util, java.sql, java.math, etc.

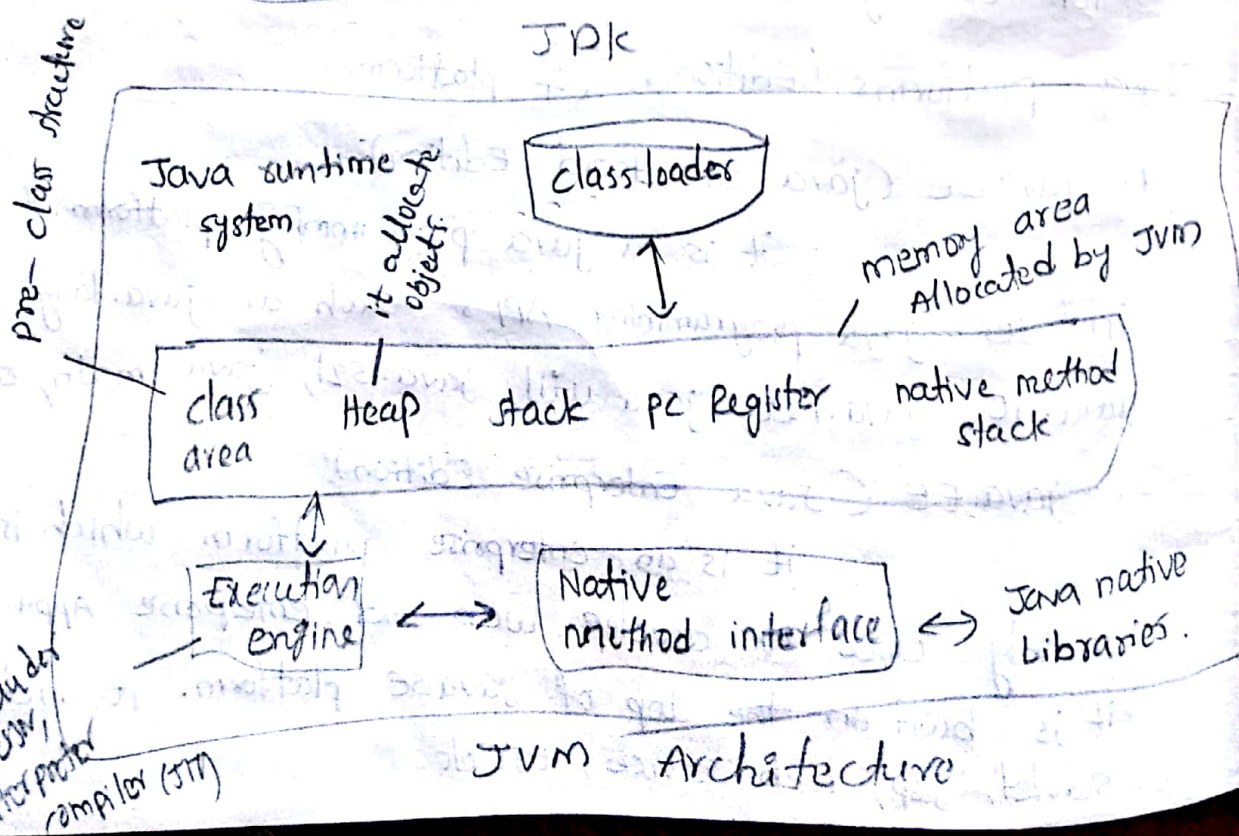
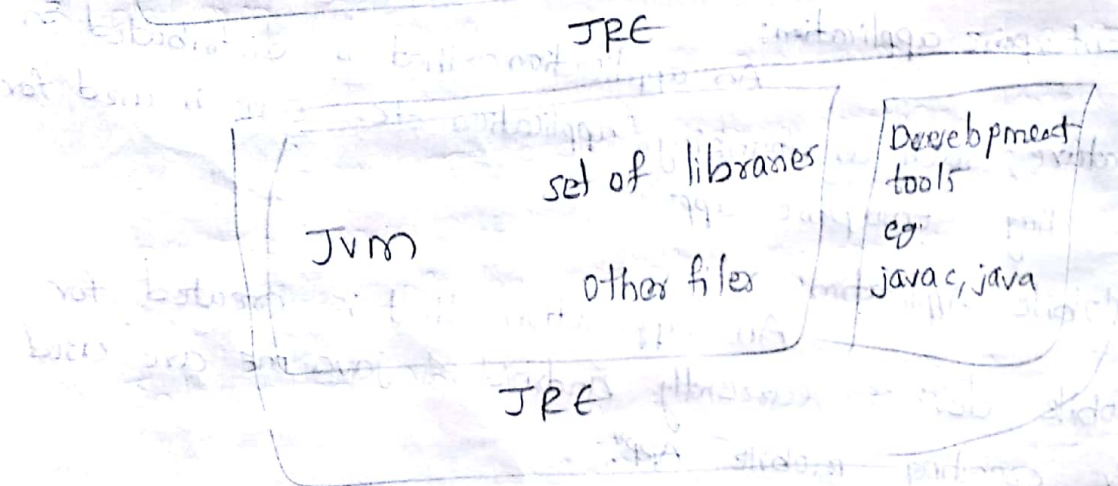
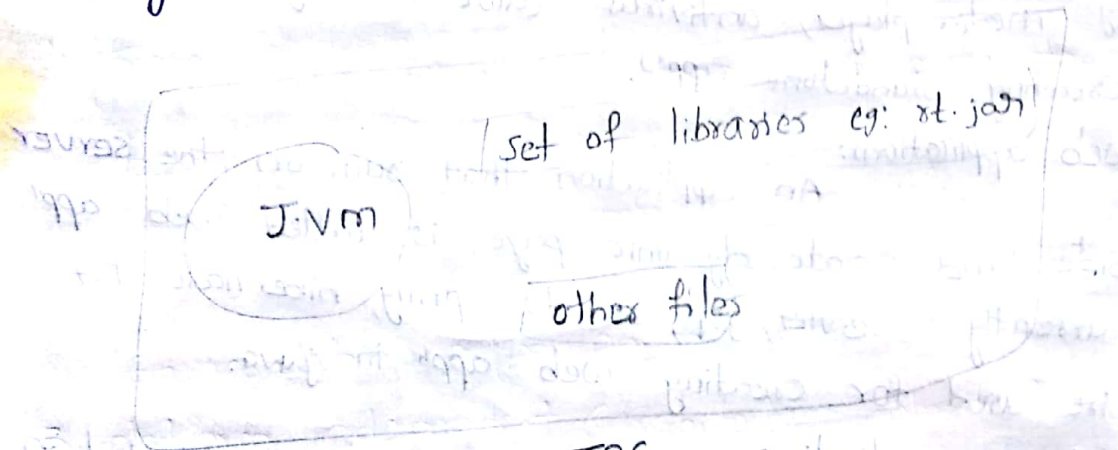
2. java EE (Java Enterprise Edition):

it is an enterprise platform which is mainly used to develop web and enterprise Appⁿ it is built on the top of java SE platform. it include servlet, jsp, web services, EJB etc.

3. Java ME (Java micro or mobile Edition):

It is mainly used to develop mobile app

4. JavaFx:
to develop rich internet app, it uses light-weight user interface APT.



Java:

Java is a high level programming language and it support the concepts of object oriented programming system. java is a platform providing an API (application programming interface) or library to reuse code and Java is a platform independent. java technologies are widely used in the designing of the web app & internet programming system.

History of JAVA: Ch James Gosling
The green project began in 1991

↓
named it as oak

↓
changed to java

↓
first public release in 1995

↓
in 1996 they release jdk 1.0

↓
it follow write once, Run anywhere

→ 1995

public releas

→ 1996

jdk 1.0

→ 1997

jdk 1.1 (java beans, JDBC)

→ 1998

j2se 1.2 (swing graphical API)

→ 2000

j2se 1.3

→ 2002

j2se 1.4 (XML)

→ 2004

j2se 5.0 (1.5)

→ 2006

J2SE 6

→ 2010

oracle buy sun microsystem.

2011

J2SE 7

2014

J2SE 8

2017

Java SE 9

Java buzzwords / characteristics / features :-

1. **Simplicity:** java is simple and easy to use because its syntax resembles c/c++.
2. **Object oriented programming approach:**
Java support the oops concept like Encapsulation, polymorphism, inheritance.
3. **platform independent (Architecture Neutral):**
Java is a platform independent or machine independent programming language. because java source code is independent of the architecture of the operating system of the computer.
4. **Interpreted Ability:** Java is a intermediary language which exhibits the features of interpretation and compilation. it uses JIT compiler to execute the java byte code.
5. **High performance:** Java is an interpreted language, so it will never be as fast as a compiled lang like c or c++. But, java enables high performance with the use of JIT compiler.
6. **portability:** Java is portable because it facilitates you to carry the java bytecode to any platform it doesn't require any type of implementation.

7. Secured:

java is secured because

- No explicit pointer
- java programs run inside virtual machine.

8. Robust:

Robust means strong. java is robust because

- it uses strong management
- There is exception handling and type checking mechanism in java.

→ There is automatic garbage collection in java.

9. Distributed:

java is distributed because it facilitates users to create distributed applications in java.

10. Multi-threaded:

Thread is a light-weight program. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, web applications.

11. Dynamic: java is a dynamic language it supports dynamic loading of classes

OOPS concept

Object

Class

Inheritance

polymorphism

Abstraction

Encapsulation

Object: Object is an instance of a class

- Object is a real world entity

- Object is a run time entity

- Object is an entity which has state & behaviour

eg: chair, bike, marker, car etc.

Object has three characteristics:

State: represents data of an object

behaviour: represents the behaviour (functionality) of an object such as deposit, withdraw.

class:

Class is a group of objects which have common properties. It is a template or blueprint from which objects are created.

Syntax: `class < class-name >`

{

 // data members

 // member functions

}

Note: instance variable: ^{variable} which is created inside the class but outside the method.

Inheritance:

it is a mechanism in which one object acquires all properties and behaviours of parent object

syntax: class subclass-name extends superclass-name

{

// methods & fields

}

↓
increase the functionality

polymorphism: polymorphism is derived from 2 greek words poly and morphs. the word poly means many and morph means forms. so it means many forms. 2 types of polymorphism in java

- ism in java

1. compile time polymorphism

2. runtime polymorphism | dynamic method dispatch

Abstraction:

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Encapsulation:

it is a process of wrapping code and data together into a single unit.

eg: capsule

Java Installation process: it specifies the size & type of values that can be stored in an identifier.

→ path setting

Data types:

primitive data types

- byte (8-bits, -2^7 to 2^7-1)
- short (16-bits, -2^{15} to $2^{15}-1$)
- int (32-bit, -2^{31} to $2^{31}-1$)
- long (64-bit, -2^{63} to $2^{63}-1$)
- float (32-bit)
- double (64-bit)
- boolean (it store one bit of information) default - f
- char (16-bit, 0 to 65535)

Reference / object data type / non-primitive

To find size → Byte (size 8)

- class
- array
- interface

Variables:

A variable can be defined as a place to hold the value of data in order to represent the state of an object. It should be declared before using the program, and it can be initialized during the declaration or execution.

Syntax: datatype <variable name>
<variable name> = value;

eg: int a, b, c; // declaration
int a=10, b=10; // initialization
float b=1.4f;

Operators:

Arithmetic operators

Relational "

Bitwise operator ($\gg$ shift right zero fill)

Logical "

Assignment "

Misc "

↓ if there is '-' sign in msb it will change value

eg:- $A = 60$, 00111100

$A \gg 2 = 00011100$ (15) ^{filled with zero}

$A \gg 2 = 1111$ (15)

$A \ll 2 = 1110000$ (240)

new operator: it is used to create an object or instance to a particular class.

Syntax: `classname variablename = new classname();`

eg:- `student s1 = new student();`

Sample program:

```
// simple java program
```

```
class Simple {
```

```
public static void main (String args[])
```

```
{  
    System.out.println ("Hello, it's java program");
```

```
}  
}
```

- class keyword is used to declare a class in java
- public keyword is an access modifier which represents visibility, it means it is visible to all.
- static is a keyword
- void is the return type, it doesn't return any value.
- main represent the starting point of program.
- `String [] args / String args[] / String ... args` - is used for command line arguments.

→ `System.out.println()` - is used to print stmt. here
System is class, out is object & println is method.

And we can write the `main` function in different ways.

```
public static void main (String args[])
```

```
static public void main ( " " )
```

```
public static void main (String [] args)
```

```
public static void main (String[] args)
```

```
public static void main (String... args)
```

// simple arithmetic operator program

```
class Arithmetic {
```

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

```
double n1 = 12.5, n2 = 5.3, res;
```

```
res = n1 + n2;
```

```
System.out.println ("addition is: " + res);
```

```
res = n1 - n2;
```

```
System.out.println ("sub: ", +res);
```

```
res = n1 * n2;
```

```
System.out.println ("mul: ", +res);
```

```
} }
```

// simple class program

```
class A {
```

```
int id;
```

```
String name;
```

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

```
A a1 = new A(); // creating object of 'A' class
```

```
S.O.P(a1.id); // accessing data member through  
reference variable.
```

```
} }
```


✓ // object & class Example

```
class student {
```

```
    int id;
```

```
    String name;
```

```
}
```

```
class student2 {
```

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

```
        student s1 = new student(); // object creation
```

```
        student s2 = new student(); // creating multiple objects
```

```
//initializing objects
```

```
s1.id = 501;
```

```
s1.name = "XXX";
```

```
s2.id = 502;
```

```
s2.name = "YYY";
```

```
//printing data.
```

```
s.o.p(s1.id + " " + s1.name);
```

```
s.o.p(s2.id + " " + s2.name);
```

```
}
```

✓ // object & class example: initialization through method.

```
class student {
```

```
    int rollno;
```

```
    String name;
```

```
    void insertRecord(int r, String n) {
```

```
        rollno = r;
```

```
        name = n;
```

```
}
```

```
    void display() {
```

```
        s.o.p(rollno + " " + name);
```

```
}
```

```

class student2 {
    public static void main(String args[]) {
        student s1 = new student();
        s1.insertRecord(501, "latha");
        s1.display();
    }
}

```

Arrays: Array is collection of similar data types. Array is a container object that hold values of homogenous type. it also known as static data structure because size of an array must be specified at the time of declaration. array occupies a contiguous memory location and array index starts from zero to zero size-1.

Syntax: datatype[] identifier; or
datatype identifier[];

eg: int[] a;
int a[];

→ new operator is used to initialize an array

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

```
int a[] = {10, 20, 30, 40, 50};
```

// simple array program

```
class Array {
```

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

```
        int[] arr; // Array declaration
```

```
        arr = new int[5]; // allocating memory for 5 int.
```

```
        arr[0] = 10; arr[1] = 20; arr[2] = 30;
```

```
        arr[3] = 40; arr[4] = 50;
```

// accessing the elements of array


```

for (int i=0; i < arr.length; i++)
    s.o.p ("element at index " + i + " : " + arr[i]);
}
}

```

// 2D array ; addition of 2 matrices.

```

class Addition {

```

```

    public static void main (String args[]) {

```

```

        int a[][] = { {1,3,4}, {3,4,5} }; // creating matrices

```

```

        int b[][] = { {1,3,4}, {3,4,5} };

```

// to store sum of matrices create another matrix

```

        int c[][] = new int [2][3];

```

```

        for (int i=0; i < 2; i++) {

```

```

            for (int j=0; j < 3; j++) {

```

```

                c[i][j] = a[i][j] + b[i][j];

```

```

                s.o.p (c[i][j] + " ");

```

```

            }

```

```

            s.o.p (); // new line

```

```

        }

```

```

    }
}

```

Control statements:

The execution can be sequential or random depending upon the control statements. The classification of control stmts is as follows.

- Conditional stmts — if, if-else, if-else if ~~else~~, nested if
- iteration stmts — while, do-while, for, nested for, foreach
- selection stmts — switch stmt
- Branching stmts — return, continue, break.

Conditional stmts:

if: syntax: if (condition)

{ statements;

}

if statement executes the stmts only if the condition is true.

// Example: if-condition

```
class even {  
    public static void main (String[] args) {  
        int n = 10;  
        if (n % 2 == 0)  
        {  
            S.O.P ("This is even number");  
        }  
    }  
}
```

if-else: if-else an extension of if stmt that provide another option when if stmt evaluates to false. i.e, else block is executed if 'if' stmt is false

syntax: if (condition) {

statements; }

else {

statements; }

// if-else example

```
int n = 15;
if (n % 2 == 0) {
    S.O.P ("This is even no");
}
else {
    S.O.P ("This is odd no");
}
```

if-elseif-else:

it executes stmts1 if condition1 is true. if condition1 is false, verifies condition2. if condition2 is true, execute the stmts2, if not, passes to the next elseif condition. if all conditions are false then executes the stmts in the last else block.

Syntax: if (condition1) {
 stmts1;
} else if (condition2) {
 stmts2;
} else {
 stmts;
}

// if-elseif example

```
int n = 25;
if (n > 10) {
    S.O.P ("n value is " + n); // cond1 - true
}
elseif (n > 20) {
    S.O.P ("n value is " + n); // cond2 - true
}
else {
    S.O.P ("all conditions are false");
}
```

nested if: if you are going to include multiple if stmts then we use nested if stmt.

Syntax: if (condition1)

{
stmts 1; // cond1 is true

if (condition2)

{
stmts 2; // cond2 is true

if (condition)

{
stmts n;

}
// example

int i = 10;

if (i == 10)

{
s.o.p ("condition is true");

if (i < 10)

{
s.o.p ("cond2 is true");

if (i > 10)

{
s.o.p ("cond3 is true");

s.o.p ("condition is false");

Iteration stmts:

iteration is nothing but repetitive loops

while:

syntax: while(condition)

```
{  
  statements;  
}
```

// example

```
int a = 25;  
while(a >= 25)  
{  
  s.o.p(+a);  
  a--;  
}
```

do-while:

syntax: do

```
{  
  stmts;
```

```
while(condition);
```

In do-while if condition is true or false it execute stmts atleast once, but while, if condition is true then only it execute stmts once.

// example

```
int i = 1;  
do {  
  s.o.p("Num!" + i);  
  i++;  
} while(i <= 10);
```

for loop:

syntax: for(initialization; condition; inc/dec)

```
{  
  stmts;  
}
```

Syntax: for (exp1: exp2)

det type Σ

stands;

3

p2) Larray collection

Note: exp_1 & exp_2 var exp_1 's declaration variable must be of the same return type of the exp_2 .

/* program for for-each loop */

class Foreach

$$\{$$

p.s.v.m ()

५

```
int a[] = {0, 1, 2, 3};
```

for (let x : α)

$$\{$$
$$s.o.p (+\infty);$$

3

String $S[0] = \{ \text{"java"}, \text{"python"} \}$

for (String name: S₁)

S.O.P(+name).

५५

Selection strat:

To select particular block of struts we are going to selection struts.

Switch:

Switch:

The keyword switch is followed by an expression that should be primitive data type only. In switch block there can be one or more labeled cases. The expression that create labels for the case must be unique. if expression is matched with any labeled case, that case is executed, if it is not match it will go to default block.

Syntax: switch (expression)

{

case 1:

stmts 1;

break;

case 2:

stmts 2;

break 2;

...

default:

stmts;

}

// Example for switch case.

char child = 'B';

switch(child) {

case 'A': s.o.p("girl"); break;

case 'B': s.o.p("baby"); break;

case 'C': s.o.p("boy"); break;

default: s.o.p("no body");

}

Branching stmts:

1. Break stmt:

break stmt is used for breaking the execution of a loop. it also terminates the switch stmts.

Syntax: break;

2. Continue stmt:

To skip the current iteration of the loop and resume the next iteration.

Syntax: continue;

3. return stmt:

Syntax: return; // this returns nothing. so this can be used when method is declared with void return type.

Syntax: return exp; // it returns the value evaluated from the exp.

// example for branching stmts

```
int num[] = {2, 3, 6};
```

```
for (int x : num)
```

```
{
```

```
    if (x > 3)
```

```
    {
```

```
        continue;
```

```
    }
```

```
    S.O.P(+x);
```

```
}
```

Note: `System.exit(0);` - stmt terminates in a normal way
`System.exit(1);` - stmt terminates with an error.

Methods:

Method is a function to express the behavior of an object by influencing its members.

Syntax: `specifier returntype methodname (parameter list)`

```
{
```

```
    // stmts
```

```
}
```

(or)

```
returntype methodname()
```

```
{
```

```
    // stmts
```

```
}
```

// example program for passing of arguments to a method

```
public class Argpass
```

```
{
```

```
    int a = 12, b = 15;
```

```
    void demo(int x, int y)
```

```
    {
```

```
        x = 3, y = 5;
```

```
        S.O.P("x: " + a + ", y: " + y);
```

```
    }
```

```
    public static void main (String args[])
```

```
    {
```



```
Argpass obj = new ArgPass();
```

```
s.o.p (" a and b : " + a + " , " + b);
```

```
obj.demo(a,b); // calling method to invoke  
the demo method using call-by-value.
```

```
s.o.p (" after calling method values of a & b : "  
+ a + " , " + b);
```

```
}  
}
```

Method overloading:

Method overloading can be defined as the process of defining various methods with same name but with different datatypes of parameters resulting in different signatures of methods.

(*) simple example for method overloading

```
Class Math
```

```
{
```

```
// method with same name but different datatypes  
& parameters.
```

```
int add(int x, int y)
```

```
{
```

```
int z = x + y;
```

```
return z;
```

```
}
```

```
void add(int p, float q)
```

```
{
```

```
float r = p + q;
```

```
s.o.p (" addition : " + r);
```

```
}
```

```

float add ( )
{
    int m = 6, n = 5; float s = 1.6;
    float p = m + n + s;
    return p;
}

```

```

public class overload
{
    p.s.v. m ( )

```

```

{
    int a = 10, b = 15;
    float c = 13.1;
    Math m1 = new Math();

```

```

    s.o.p ("addition of integers: " + m1.add(a, b));

```

```

    @ m1.add(b, c);

```

```

    s.o.p ("addition of integers & float: "
        + m1.add());

```

```

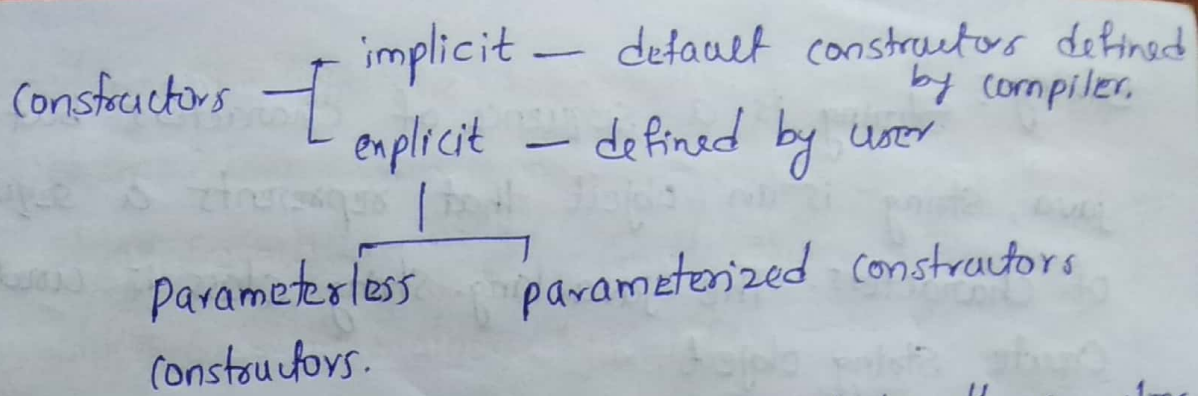
}
}

```

Constructors:

Constructor can be defined as a special type of method used to initialize the instance variables of a class when an object is created. At least one constructor must be defined in every class to construct an object, otherwise the compiler will automatically create a default constructor.

Syntax: Constructorname (parameters) {
 // statements
}



Default constructor: it is implicitly defined by the compiler

and also named as implicit constructor.

It's body is empty without any variables, and it has no arguments in its parameter list.

Syntax: `ConstructorName()`
{
 // empty body
}

parameterless constructor:

it can be initialized by the user to instance members of an object of a class, it consists of no parameters in its declaration and constitutes the body in its definition.

Syntax: `ConstructorName()`
{
 // constructor body
}

parameterized constructor:

The structure of parameterized constructor consists of parameters in its declaration and constitutes body in its definition.

Syntax: `ConstructorName(parameters)`
{
 // constructor body
}

String: String is a sequence of characters. but in java, string is an object that represents a sequence of characters. The java.lang.String class is used to create string object.

There are two ways to create string object

- By string literal
 - By new keyword
- ↓ char[] ch = {'j', 'a', 'v', 'a'};
String s = new String(ch);

String Literal: java string literal is created by using double quotes.

eg: String s = "java";

To make java more memory efficient it uses string literal because no new objects are created if it exists already in string constant pool.

Note: string objects are stored in a special memory area known as string constant pool.

new keyword:

JVM will create a new string object in heap memory and the literal 'welcome' will be placed in the string constant pool the variable 'a' will refer to the object in heap.

String a = new String("welcome") // create two objects & one reference variable.

// Example for strings

```
public class Stringeg  
{  
    p.s.v.m()   
}
```


String s1 = "java";

char ch[] = {'s', 't', 'r', 'i', 'n', 'g'};

String s2 = new String(ch); // converting char array to String

String s3 = new String("javab");

s.o.p(s1);

s.o.p(s2);

s.o.p(s3);

3 3

toUpperCase

toLowerCase

replace('i', 'j')

trim

equals

length

String built-in class

String class

StringBuffer class

String builder

String Handling functions:

① String length: it returns length of string

Syntax: object.length()
String

② String Concatenation:

Combination of one string with another string it can be done by using 2 ways

'+' operator

concat() method.

Syntax for '+' operator:

String obj = String obj1 + String obj2 + ...

Syntax for concat():

String obj = String object1.concat(String object2);

eg:- String s = "java"; String s2 = "Lab";

String p = s2.concat(s)

s.o.p(p) // o/p: java Lab java

③ `substring()`: it is a part of the existing string that is created by invoking the `substring()` method.

Syntax: `String ObjectName.substring(parameters)`.

eg:- `String s = "java lab"`.

`String s2 = s.substring(2);`

// creates a substring starting from the index 2 to the end of string. o/p: `valab`

`String s2 = s.substring(2, 5);`

// creates a substring starting from the index 2 to 5.
o/p: `vala`

④ Comparing strings:

Strings are compared using the `compareTo()` method. The value returned by this method is an integer.

Syntax: `variable name = String obj1.compareTo(String obj2)`

Note: mutable objects: it can be defined as the objects whose values can be changed after their creation.
eg:- `StringBuilder`, `StringBuffer`.

Immutable objects: the objects whose values cannot be changed once initialized.

- ⑤ boolean equals()
- ⑥ equalsIgnoreCase()
- ⑦ boolean contains()
- ⑧ boolean isEmpty()
- ⑨ string toUpperCase()
- ⑩ string toLowerCase()
- ⑪ boolean startsWith()
- ⑫ boolean endsWith()

⑬ char charAt(int index):

Returns char value for the particular index

Syntax: object name . charAt(int index);

eg:- string s₁ = "javab";

string s₂ = s₁. charAt(4);

⑭ string replace(char old, char new)

It returns a string replacing all the old char to new char.

Syntax: string name = objectname . replace(old, new);

string s = s₁. replace('B', 'D');

⑮ string replace (Sequence of char old, sequence of char new)

string s = s₁. replace("java", "paa");

⑯ string split (): split based on whitespace

Syntax: string name = object name . split (String regex);

string s₁ = "java string split";

string s = s₁. split ("\\s");

(17) String indexOf (char ch)

returns index of given character value or substring. if not found it returns -1.

(18) String trim(): removes beginning & ending spaces of this string.

Syntax: objectname.trim();

$S_1 = \text{" java "}$

String $S = S_1.trim()$

(19) String comparison:

① String equals()

② ==

③ String compareTo()

④ String compareToIgnoreCase()

String $S_1 = \text{"Hello world"}$

$S_2 = \text{"hello world"}$

String $S = S_1.compareTo(S_2)$

$S = S_1.compareToIgnoreCase(S_2)$

↓ returns 0

if (String 1 > String 2) - +ve

if (String 1 == String 2) - 0

if (String 1 < String 2) - -ve

$S_1 = \text{Ram}, S_2 = \text{Ram}, S_3 = \text{shyam}, S_4 = \text{ABC}$

$S_1.compareTo(S_2)$ - 0

$S_1.compareTo(S_3)$ - -1

$S_1.compareTo(S_4)$ - 17

returns ASCII value of 1st odd char

Typecasting:

Typecasting is the process of converting the data types declared in one type to a specially specified datatype. Except boolean, all the primitive datatypes are converted from one type to another.

Type conversion two types those are

- Implicit or automatic or widening
- Explicit or Narrowing

Implicit: the range of the destination is larger than the source type.

- byte to short, int, long, float, double
- short to int, long, float, double
- int to long, float, double
- long to float, double
- float to double

Explicit casting:- datatypes as the precision of the datatype to be converted is more than the converting type.

- byte to char
- short to byte, char

Syntax:

Input in java :

① Scanner class in java:

Scanner is a class in java.util package used for obtaining the input of the primitive types like int, double etc. and strings.

→ To create an object of scanner class, we usually pass the predefined object 'system.in', which represents the standard input stream.

→ To read numerical values of a certain datatype xyz, the function to use is nextxyz().

eg:- read value of int, we use nextInt().

→ To read strings, we use nextLine().

→ To check next value we use hasNextxyz().

(eg: hasNextInt(), hasNextIntLine() - strings)

// Example for scanner class.

```
import java.util.Scanner;
```

```
public class ScannerEg
```

```
{
```

```
    P.S.V.M.C )
```

```
{
```

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

```
    // Declare an object and initialize with predefined
```

```
    // standard input object.
```

```
    int sum = 0, count = 0;
```

```
    // check if an int value is available.
```

```
    while (s.hasNextInt())
```

```
    {
```

```
        // Read an int value
```

```
        int num = s.nextInt();
```

```
        sum = sum + num;
```

```
        count++;
```

```
}
```


$\text{int mean} = \text{sum} / \text{count};$

$\text{s.o.p}(" \text{mean} : " + \text{mean});$

Note: we use `nextInt()`, `nextFloat()`, `nextDouble()` functions in scanner class. for char we use

/ scanner example */*

`next()`, `charAt(0)`; - returns first position of char.

Singleword - `next()`
`nextLine()` - Line of string
`nextBoolean()` - Boolean
`Long()`, `Short()`

```
import java.util.Scanner;
```

// This will import just the scanner class.

```
class Input
```

```
{
```

```
    p.s.v.mc)
```

```
}
```

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

```
String s1 = s.nextLine();
```

```
System.out.println("String is: " + s1);
```

```
int a = s.nextInt();
```

```
System.out.println("Int: " + a);
```

```
float b = s.nextFloat();
```

```
System.out.println("float: " + b);
```

② Buffer Reader class:

This class introduced in `jdk1.0` this method is used by wrapping the '`System.in`' in an 'Input Stream Reader' ^{class} which is wrapped in a Buffer Reader ^{class}, we can read input from the user in the command line.

This class present in '`io`' package

/* Example for BufferedReader class */

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

```
public class Test
```

```
{
```

```
    p.s.v.m()
```

```
{
```

```
    BufferedReader br = new BufferedReader (new  
class InputStreamReader (System.in)).
```

```
    String s1 = br.readLine();
```

```
    s.o.p(s1);
```

```
} }
```

object

constructor

constructor

predefined object

to read string from keyboard.

Note: To read other types, we use functions like
Integer.parseInt(), Double.parseDouble()

③ console class:

It has been becoming a preferred way for
reading user's input from the command line. it will
not support for IDE.

```
public class sample
```

```
{
```

```
    p.s.v.m()
```

```
{
```

```
    String name = System.console().readLine();
```

```
    s.o.p(name);
```

```
} }
```


PART-II Inheritance

→ Inheritance is an important pillar of oop (Object oriented programming). it is the mechanism in java by which one class is allow to inherit the features of another class.

Important Terminology:

- superclass: the class whose features are inherited is known as super class / base class / parent class.
- sub class: The class that inherits the other class is known as sub class / derived class / extended class / child class.

the subclass can add its own fields and methods in addition to the superclass fields & methods

- Reusability: it support the concept of 'reusability' i.e, when we want to create a new class and there is already a class that includes some of the code that we want, by extending the class we access all data.

Syntax:

```
class derivedclass extends baseclass  
{  
    // methods & fields  
}
```

// Example it promotes the code reusability. it promotes polymorphism by allowing method overriding.

→ To inherit properties of one class to another class we use extends keyword.

Types of inheritance:

Single inheritance

multilevel inheritance

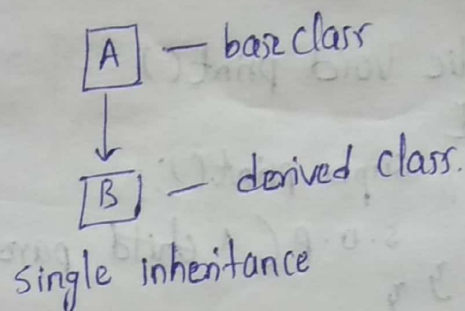
Hierarchical inheritance

multiple inheritance (Through Interface)

Hybrid inheritance

1. Single level inheritance:

Subclass inherit the features of one superclass.

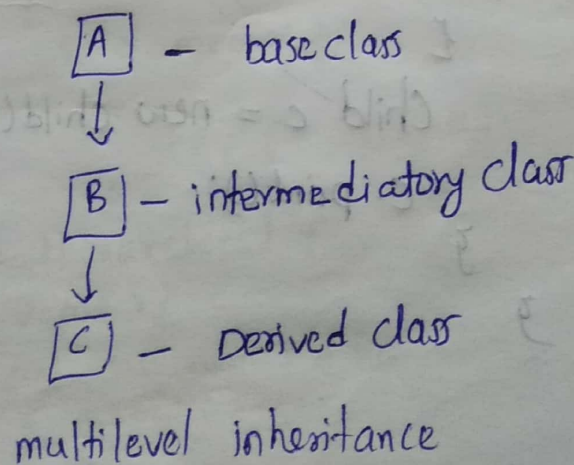


Syntax:

```
class derivedclass extends baseclass  
{  
    // methods & fields  
}
```

2. Multilevel inheritance:

A derived class will be inheriting a base class and as well as the derived class also act as the base class to other class.



Syntax:

```
class A  
{  
}  
class B extends A  
{  
}  
class C extends B  
{  
}
```


// Example for multilevel inheritance.

```
class GrandParent
```

```
{  
    public void print()
```

```
{  
    s.o.p ("Grandparent");  
}
```

```
class parent extends GrandParent
```

```
{  
    public void print()
```

```
{  
    // super.print();  
    s.o.p ("child parent");  
}
```

```
class child extends parent
```

```
{  
    public void print()
```

```
{  
    super.print();  
    s.o.p ("child");  
}
```

```
public class main
```

```
{  
    p.s.v.m()
```

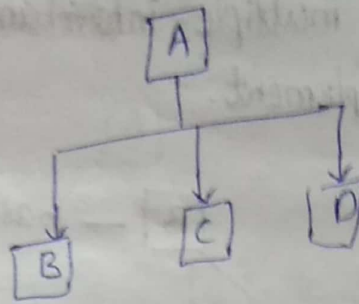
```
{  
    child c = new child();
```

```
    c.print();  
}
```

```
}
```

3. Hierarchical inheritance:

In one class serves as a superclass for more than one sub class.



Hierarchical inheritance

Syntax:

```

class A
{
  // code
}
class B extends A
{
  // code
}
class C extends A
{
  // code
}
  
```

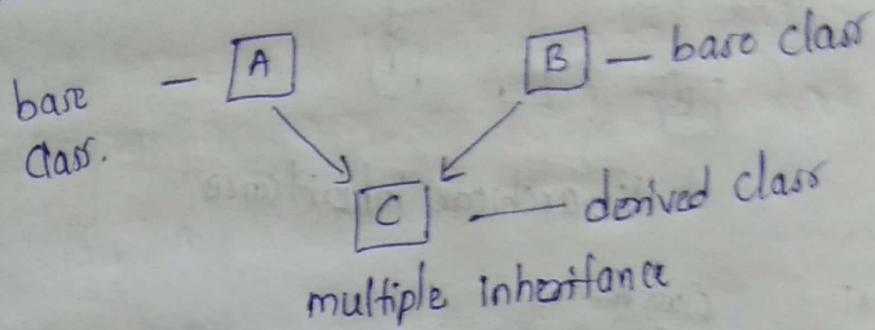
// Example

```

class Animal {
  void eat()
  {
    s.o.p("eating");
  }
}
class Dog extends Animal
{
  void bark()
  {
    s.o.p("barking");
  }
}
class cat extends Animal
{
  void meow()
  {
    s.o.p("meow");
  }
}
class Hierarchical {
  p.s.v.m() {
    cat c = new cat();
    c.meow();
    c.eat();
    c.bark(); // compilation error.
  }
}
  
```


4. Multiple inheritance:

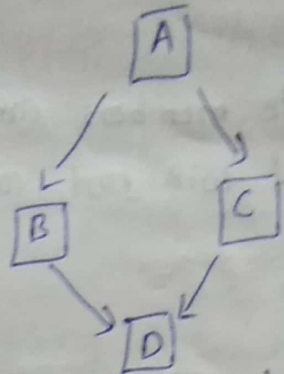
One class can have more than one superclass and inherit features from all parent classes. java does not support multiple inheritance, but by using interface we can implement.



Syntax:

5. Hybrid inheritance (Through interfaces):

it is a mix of two or more of the above types of inheritance. it is also not support by java classes, but by using interface it can be implemented



Hybrid inheritance

Member Access and Inheritance:

→ A class member that has been declared as private will remain private to its class. It is not accessible by any code outside its class, including subclasses.

/x Example for private members can't access: it contains an error and will not compile */

class A // superclass

```
{  
    int i; // public by default  
    private int j; // private to class A
```

```
void set(int x, int y)
```

```
{  
    i = x;  
    j = y;  
}
```

class B extends A // is a child class B

```
{  
    int total;
```

```
void sum()
```

```
{  
    total = i + j; // it shows error, because i  
                  // is private to class A
```

class Main

```
{  
    p.s.v.m()
```

```
{
```

```
    B obj = new B();
```

```
    obj.set(10, 20);
```

```
    obj.sum();
```

```
    s.o.p("total is: " + obj.total);
```

```
}
```

Method overriding: Methods of different classes have same signature. method overridden can be defined as the allowing sub class to override the method implementation of its super class, which is already provided.

// Example for overriding.

```
class A
{
    String s1 = "java";
    void display()
    {
        s.o.p ("s1: " + s1);
    }
}
```

```
class B extends A
{
    String s2 = "labh";
    void display()
    {
        s.o.p ("s2: " + s2);
    }
}
```

```
class override
{
    p.s.v.m()
    {
        A a1 = new A();
        a1.display();
        B b1 = new B();
        b1.display();
    }
}
```


Types of Access modifiers/specifiers/access controls

- public

- default

- protected

- private

public: The member with public modifiers can be accessed by any classes. The public methods, variables or class have the widest scope.

default: When we do not mention any access modifier it is treated as default, it is accessible only within same package.

protected:

The protected modifier is used within same package it lies between public & default access modifier. It can be accessed outside the package but through inheritance only.

Note: A class can't be protected.

eg:-

```
class student
```

```
{
```

```
    protected int id = 101;
```

```
    protected String name = "Jeevan";
```

```
}
```

```
p.s.N public class protected1 extends student
```

```
{
```

```
    private String course = "cse1";
```

```
    public void display()
```

```
{
```

```

s.o.p("student id:" + id);
s.o.p("student name:" + name);
s.o.p("student course:" + course);

```

```

}
public static void main(String args[])
{
    protected1 p1 = new protected1();
    p1.display();
}
}

```

O/P: student id: 101
 student name: jeevan
 student course: cse

private:

The private methods, variables and constructors are not accessible to any other class. it is the most restrictive access modifier. A class except a nested class cannot be private.

eg:-

```

public class privateDemo
{
    private int a = 101;
    private String s = "student";
    public void show()
    {
        s.o.p(a + s);
    }
}

P.S.V.M.C()
{
    privateDemo p1 = new privateDemo();
    p1.show();
    s.o.p(p1.a + " " + p1.s);
}
}

```


visibility of fields in a class:

access
modifiers/
access location

public protected default

private

same class	YES	YES	YES	YES
sub class in same package	YES	YES	YES	NO
other classes in same package	YES	YES	YES	NO
sub classes in other packages	YES	NO	NO	NO
non-sub class in other packages	YES	NO	NO	NO

Super key word in JAVA:

- super keyword is similar to this keyword in java
- it is used to refer to the immediate parent class of the object. it may be only used within a subclass constructor / method.
- parameter in super call must match the order and type of instance variable declared in the super class.

Uses of super keyword:

1. super() calls the parent class constructor with no arguments.

2. Super is used to invoke immediate parent class method. [super.methodname],
3. By super is used to call the parent class variable. [super.variablename]

Eg for case 1:

```
class parent
{
```

```
    public parent(String str)
```

```
    {
        S.O.P("constructor of parent class:" + str);
    }
}
```

```
class child extends parent
{
```

```
    public child()
```

```
    {
        super("super call child class constructor");
```

```
        S.O.P("constructor of child");
    }
}
```

```
public class Test
```

```
{
    p.s.v.m (String[] args)
```

```
{
    child c = new child();
}
```

O/P: constructor of parent class: super call
child class constructor.
constructor of child.

eg for case 3:

```
class Base
{
    int a = 50;
}
public class Derive extends Base
{
    int a = 100;
    void display()
    {
        s.o.p(super.a);
        s.o.p(a);
    }
}
P.S.V.m()
{
    Derive d1 = new Derive();
    d1.display();
}
```

O/p: 50

Refer case ¹⁰⁰ example: multilevel inheritance.

Final keyword in JAVA:

The final keyword in java indicates that no further modification is possible. final can be variable, method or class

final variable:

final variable is a constant we cannot change the value of final variable after initialization.

final method:

When we declare any method as final it cannot be override.

final class:

When a class is declared as a final it cannot be extended.

// Example for final keyword [final variable, final method, final class].

final class Example // final class

{

final int i = 2; // final variable

final sum() // final method

{

i = i + 10; // we can't change final var

s.o.p("value of i: " + i);

}

final void method() // final method

{

s.o.p("final method").

}

} // class close // can't extend final class

class child extends final Example

{

void method() // can't override final method

{

s.o.p("Normal method").

}

} // child class close

→ java.lang.String class provides many useful methods to perform operations on sequence of char values.

Method	Syntax	Description
1. char charAt (int index)		returns char value for the

1. int length()	String s1 = "java"; eg: s1.length() // 4	Returns length of char
-----------------	--	------------------------

2. boolean equals (Object)	eg: s1 = "java"; s2 = "Java"; s1.equals(s2) returns false	checks the equality of string with object
----------------------------	--	---

3. static String equalsIgnoreCase()	eg: s1.equalsIgnoreCase(s2) return True	compares another string it doesn't check case
-------------------------------------	--	---

4. boolean contains (CharSequence)	eg: name = "xxx"; return name.contains("x"); T name.contains("y"); return F	returns T or F after matching the sequence of char value
------------------------------------	---	--

5. boolean isEmpty()	eg: s1.isEmpty(); returns false s2 = ""; s2.isEmpty(); returns true	checks if string is empty
----------------------	---	---------------------------

6. String concat(String str)

eg:-

~~S1~~ S1.concat(" is immutable")
S.O.P(S1)
[o/p: java

S1 = S1.concat(" is immutable")

S.O.P(S1)
[o/p: java is immutable

concatenates specified string at end of string.

7. String toUpperCase()

S1. toUpper

S2 = S1.toUpperCase()

S.O.P(S2)

O/P: JAVA

returns string in upper case.

8. String toLowerCase()

S1 = S1.toLowerCase()

S.O.P(S1)

O/P: java

returns string in lower case

X9. String toUpperCase(Locale locale)

S1 = "Hello"

S1 = S1.toUpperCase()

returns specified chars in upper case

9. boolean startsWith()

eg:-

S1 = S1.startsWith("Lab")

S.O.P(S1)

O/P: Lab.java yes

starts with specified word or char then it returns true otherwise false.

This keyword:

→ this keyword in java can be used inside the method or constructor.

→ this works as a reference to the current object whose method or constructor is being invoked.

Eg: // Example for this keyword

```
class student
```

```
{
```

```
// int rollno = 101 // int var = 10 // instance var
```

```
void method (int var)
```

```
{
```

```
var = 20; // local variable
```

```
it print local var value ← s.o.p ("value of var" + var);
```

```
it print instance var value ← s.o.p ("value of var" + this.var);
```

```
}
```

```
}
```

```
class student 2
```

```
{
```

```
void student2 ()
```

```
{
```

```
s.o.p ("inside the student 2 method")
```

```
}
```

```
void student3 ()
```

```
{
```

```
s.o.p ("inside the student's method")
```

```
without calling ← this.student2();
```

```
student2() method }
```

by using this keyword we

can directly access student2().

class main

P.S.V.M ()

Student P1 = new Student(2)

P1 = method(1)

Student2 P2 = new StudentP2()

P2 = Student3()

3 3

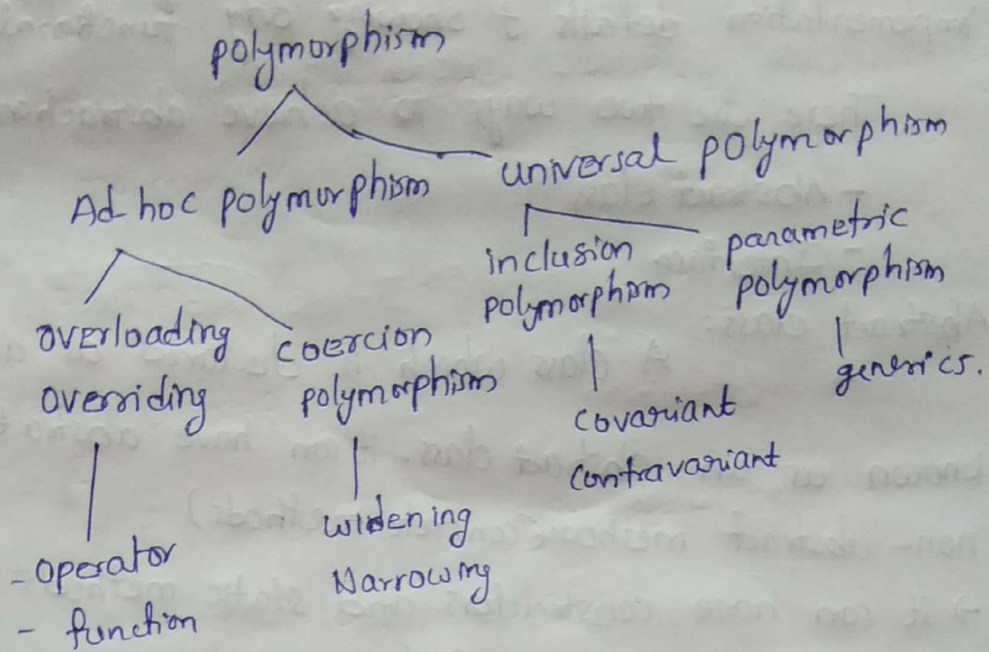
O/P: value of var: 20

value of var: 10

Inside the student3 method
inside the student2 method.

polymorphism:

polymorphism in computer science was introduced in 1967 by Christopher Strachey.



Abstract classes & methods:

Abstraction: Abstraction is a process of hiding the implementation details & showing only functionality ^{to the user}.

There are two ways to achieve abstraction in java

- Abstract class
- Interface

Abstract class: A class which is declared as abstract is known as an abstract class. it can have abstract and non-abstract methods (concrete methods)

- it can have constructors and static methods also.
- it cannot be instantiated.
- it can have final methods which will force the subclass not to change the body of the method.

Syntax: `abstract class classname { - - - }`

Abstract method:

A method which is declared as abstract and does not have implementation is known as an Abstract method.

Syntax: `abstract method name ( ) { } ;`

|| Example for Abstract class & method.

```
abstract class shape {  
    abstract void draw ( )  
}
```

```
class Rectangle extends shape  
{
```



```

void draw()
{
    s.o.p ("Rectangle class");
}

class Circle extends Shape
{
    void draw()
    {
        s.o.p ("Circle class");
    }
}

```

Class Abstraction

```

{
    p.s.v.m()
    {
        Shape s = new Circle();
        s.draw();
    }
}

```

or

```

Shape s;
s = new Rectangle();
s = new Circle();

```

Note: if there is an abstract method in a class, that class must be abstract.

→ if you are extending an abstract class that has an abstract method, you must either provide the implementation of the method or make this class abstract.

// Example for abstract class has abstract & non-abstract methods.

abstract class Student

```

{
    Student () // constructor

```

```

    {
        s.o.p ("constructors in abstract class");
    }

```

```

    abstract void run();

```

```

    void display() {

```

S.O.P ("non-abstract method")

// creating a child class which inherits abstract class.

```
class student2 extends student
```

```
{
```

```
void run()
```

```
{
```

```
S.O.P ("Inside child class")
```

```
}
```

```
}
```

```
class Abstraction
```

```
{
```

```
P.S.V.M()
```

```
{
```

```
Student s = new student2()
```

```
s.run()
```

```
s.display()
```

```
}
```

→ Reference of an abstract class can point to objects of its sub-classes thereby achieving run-time polymorphism.

UNIT-II

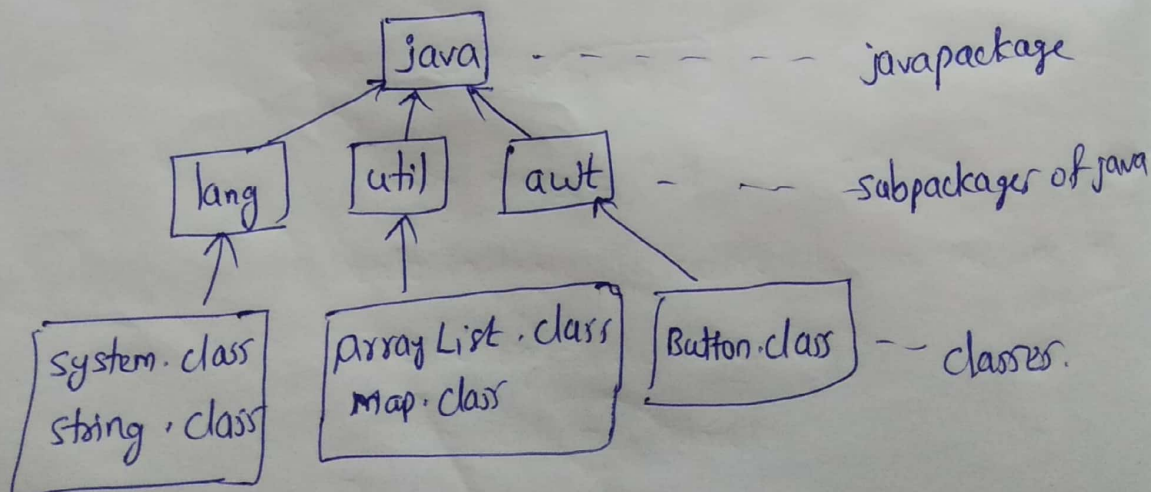
Packages - Defining a package, classpath, access protection
importing packages.

Package: package is a group of similar type of classes,
interfaces and sub-packages

Advantages of packages:

- java package is used to categorize the classes & interfaces so that they can be easily maintained.
- java package removes naming collision.
- java package provides access protection
- packages can be considered as data encapsulation.

we need to put related classes into packages.
after that we can simply write a import a class from
existing package & use it in our program.



Types of packages

- user defined packages
- Built in packages

Built in packages:

These packages consist of a large number of classes which are a part of java API.

→ Java.lang: it contains lang support classes. by default this package included.

eg:- primitive data types, math operations

→ java.io: contains classes for supporting i/p & o/p operations

→ java.util: contains utility classes which implement data structures like Linked List, Dictionary & support. for date/time operations.

→ java.applet: it contains classes for creating applets.

→ java.awt: it contains classes for implementing the components for graphical user interfaces (like button, menus).

→ java.net: contains classes for supporting networking operations.

→ java.string: it contains string methods & classes.

→ java.sql: for database storage.

user-defined packages:

These packages are defined by the user.

Syntax: `package <packagename>`.

↓
package is a keyword used to create userdefined package.

Steps for creating user defined packages:

- Create a directory 'mypackage'. this directory name should be same as name of package.
- create class inside the directory.

// Example for package (student.java)

```
package mypackage;
```

```
public class student
```

```
{  
    public void getNames(String s)
```

```
{  
    s.o.p(s);  
}
```

```
}
```

Now we can use the student class in our program.

```
// import student class from mypackage //
```

```
public class printName
```

```
{  
    p.s.v.m (String args[])
```

```
{  
    String name = "II - CSE - B";
```

// Create instance of student class in the package

```
Student obj = new student();
```

```
obj.getNames(name);
```

```
}
```

```
}
```

Compile java package: → destination where to put class files.

Syntax: `javac -d. directory javafilename`

eg: `javac -d. student.java`

To run: `java mypackage.student`

How to access package from another package.

- `import package.*`
- `import package.classname`
- fully qualified name.

`import package.*` :-

It include all classes & interfaces of this package will be accessible but not subpackages.

The `import` keyword is used to make the classes & interfaces of another package accessible to the current package.

`using package.classname` :-

if we import `package.classname` then only declared class of this package will be accessible.

`using fully qualified name` :-

if we use fully qualified name then only declared class of this package will be accessible. Now there is no need to import, but we need to use fully qualified name.

Generally we use these two packages have

Same class name eg: `java.util` & `java.sql` contains `Date` class.

CLASSPATH:

the classpath is an environment variable that tells the java compiler where to look for class files to import. classpath is generally set to a directory or a JAR (java archive) file.

Access protection:

// class access level: Example 1
package one;

public class A
{

private int pri = 1;

int pak = 2; // package access

protected int pro = 3;

public int pub = 4;

private void primethod()
{

s.o.p ("I am private method");

}

void packmethod()
{

s.o.p ("I am package method");

}

protected void promethod()
{

s.o.p ("I am protected method");

}

public void pubmethod()
{

s.o.p ("public method");

}

p.s.v.m()

{

A a1 = new A();

a1.primethod();

a1.packmethod();

a1.promethod();

a1.pubmethod();

5.0. p(a₁.pri+ " " + a₁.pub+ " " + a₁.prot+ " " + a₁.pack).

// Example 2: package level access

package one;

public class B

{

p.s.v.m()

{

A a₁ = new A();

a₁.primethod(); // illegal access

a₁.packmethod();

a₁.promethod();

a₁.pubmethod();

}

}

// Example 3: subclass access level

package two; // new package

import one.*; // import all class of one package.

public class AA extends A

{

p.s.v.m()

{

A a₁ = new A();

a₁.primethod(); // illegal access

a₁.packmethod(); // default - outside package

→ a₁.promethod(); // protected - "

a₁.pubmethod();

protected not access in
superclass

protected access in
subclass level.

AA a₂ = new AA();

→ a₂.promethod();

}

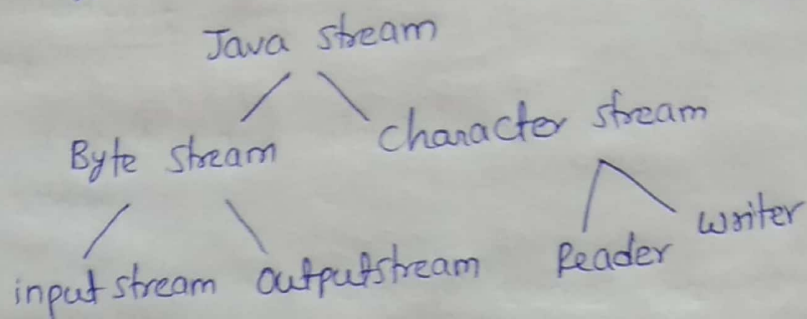
Java io Package

Streams:

Stream means continuous flow of data from source to destination.

Java encapsulates stream under java.io package

There are types of streams.



Byte stream:

The streams that are used to handle the data in the form of bytes and that can store or handle the data of size 8-bit ^{bytes} data. at a time are defined as the byte streams.

Byte stream class include the different classes. but mainly it include can be defined by using two abstract class at the top of hierarchy.

- InputStream
- OutputStream

These two abstract classes have several concrete classes that handle various devices such as disk, files, network etc.

Some important Byte stream classes

BufferedInputStream:

BufferedOutputStream

DataInputStream - it contain methods for reading java standard type.

DataOutputStream

FileInputStream - i/p stream that reads from a file

FileOutputStream - o/p stream that write to a file

InputStream - Abstract class that describe stream i/p

OutputStream - Abstract class that describe stream o/p

PrintStream - o/p stream that contain print() & println();

Character stream class:

it can handle the data in the form of characters & that can store or handle the data of 16-bit ^{unicode} data at a time can be defined as the character stream.

Some of the sub classes of the Reader/Writer such as the FileReader, FileWriter,

- Buffered Reader - handles buffered i/p stream

BufferedWriter - " " o/p "

FileReader - i/p stream that reads from file

FileWriter - o/p " " writer to "

InputStreamReader - it translates byte to character.

OutputStreamReader - " " char to byte

PrintWriter - it contain print() & println() method.

Reader - Abstract class that defines char stream i/p

Writer - Abstract class that defines char stream o/p

methods provided by InputStream class:

read() - read i/p stream data byte by byte

skip() - it returns the number of bytes that are actually skipped from the input stream.

available() -

close()

methods provided by OutputStream class:

write()

flush() - this method cleanses the o/p stream

close()

methods in Reader class:

read() - the chars are read into the specified buffer

skip() - the specified chars are skipped from char stream

mark() - it indicates the stream's current position.

reset() - Reset the stream to its previous position before mark() operation.

close()

methods in Writer class:

write()

append()

flush()

close()

creating a file using file output stream

it belongs to byte stream & store the data in the form of individual bytes. it can be used to create text files.

→ 1st we should read the data from the keyboard. ^{for} so that we include `DataInputStream` class for reading data from the keyboard.

```
DataInputStream d1 = new DataInputStream(  
    System.in);
```

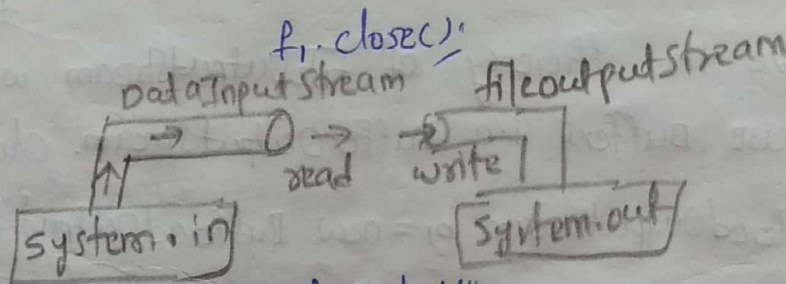
→ To attach file where the data is to be stored to some output stream. for that we use `FileOutputStream`.

```
FileOutputStream f1 = new FileOutputStream("abc.txt");
```

→ read data from `d1` object & write it into `f1` object.

```
ch = (char) d1.read(); // read one char into ch  
f1.write(ch); // write ch into file
```

→ after performing i/p & o/p operations close the file.



creating text file

// write a program which shows how to read data from the

// keyboard and write it to abc.txt file.

// create text file using `FileOutputStream`

```
import java.io.*;  
class CreateFile
```

```
{  
    P.S.V.MC) throws IOException
```

```
{
```

```
DataInputStream d1 = new DataInputStream(  
    System.in);
```

```
FileOutputStream f1 = new FileOutputStream("abc.txt")
```

```
S.O.P ("Enter text");
```

```
char ch;
```

```
while ((ch = (char) d1.read()) != '\n')
```

```
f1.write(ch);
```

```
f1.close();
```

```
}  
}
```

O/p: Enter text: Hi this is xxx \

C:\> abc.txt

Hi this is xxx

In above program user enter the text upto condition place i.e. if enters \ then only it reach the end of the file, then condition is false. (i.e. if you calculate 1st, 2nd, 3rd, 4th, it store all data).

To store data permanently we use

```
FileOutputStream f1 = new FileOutputStream("abc.txt", true);
```

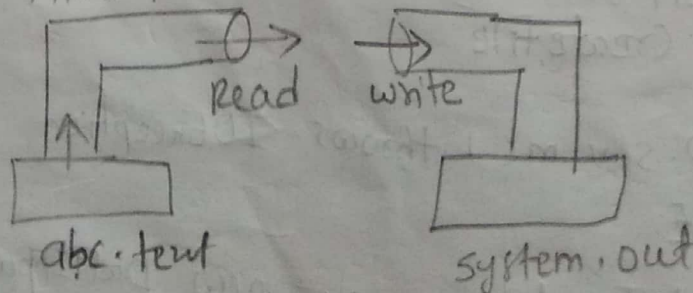
→ To improve the efficiency of FileOutputStream we are going to use buffers i.e. BufferedOutputStream class.

```
eg:- BufferedOutputStream b1 = new BufferedOutputStream(f1, 24);
```

here buffer size is 24, but buffer default size is 1024 bytes.

Reading data from a file using FileInputStream:

FileInputStream



Reading data from file.

// Reading text file using FileInputStream ✓

```
import java.io.*;
class ReadFile
{
    p.s.v.m() throws IOException
    {
        FileInputStream f1 = new FileInputStream("abc.txt");
        s.o.p("File contents ");
        int ch;
        while ((ch = f1.read()) != -1)
            s.o.p((char)ch);
        f1.close();
    }
}
```

// Creating a file using FileWriter. ✓

```
import java.io.*;
class File1
{
    p.s.v.m() throws IOException
    {
        String s1 = "Java Lab";
        FileWriter fw = new FileWriter("text.txt", filename);
        for (int i = 0; i < s1.length(); i++)
            fw.write(s1.charAt(i));
        fw.close();
    }
}
```

o/p: c:\> text a.txt
Java Lab

1 Reading a file using FileReader.

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

```
class FileReader1
```

```
{
```

```
    P.S.V.M.C() throws IOException
```

```
{
```

```
    int ch;
```

```
    FileReader fr = null;
```

```
    try {
```

```
        fr = new FileReader("text");
```

```
    } catch (FileNotFoundException e)
```

```
    {
```

```
        S.O.P("file not found");
```

```
    } return;
```

```
    while((ch = fr.read()) != -1)
```

```
        S.O.P((char) ch);
```

```
    fr.close();
```

```
}
```

Serialization:

Serialization is the process of object contents into a file. The class whose objects are stored in the file should implement serialization interface of java.io package.

Serializable interface is an empty interface without any members in it. it does not contain any methods also such interface is called 'marking interface' or 'tagging interface'. this interface is created to mark the objects of a class for special purpose.

if serializable interface not implemented by the class, then writing that class objects into a file will lead to 'NotSerializableException'.

→ static & transient variables of the class cannot be serialized.

class A implements Serializable

eg: static int x=10;

transient String s="xxx";

These two variables not implemented by class A.

// Serialization in java

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

```
class Employee implements Serializable
```

```
{ String name;
```

```
int id; double sal;
```

```
public Employee(String name, int id, double sal)
```

```
{ this.name = name;
```

```
this.id = id;
```

```
this.sal = sal;
```

```
} public String toString()
```

```
{ return "name=" + name + "; id=" + id + "; sal=" + sal;
```

```
}
```

```
public class Test
```

```
{ p.s.v.m() {
```

```
try { Employee e = new Employee("emp1", 101, 56.0);
```

```
s.o.p("Emp object " + e);
```

```
FileOutputStream fout = new FileOutputStream("abc.txt");
```

```
ObjectOutputStream t2 = new ObjectOutputStream(fout);
```

```
t2.writeObject(e);
```

```
t2.flush();
```

```
}
```

```
catch (Exception ex)
```

```
{ s.o.p("Error " + ex.toString());
```

```
}
```

O/p: emp object name=emp1; id=101; sal=56.0

& type: abc.txt

↓
it contains encrypted format of object data

De-serialization:

Once the objects are stored into a file, they can be latter retrieved and used as ^{we} and when needed this is called de-serialization.

// Deserialization

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

```
class Employee implements Serializable
```

```
{
```

```
    String name;
```

```
    int id; double sal;
```

```
    public Employee(String name, int id, double sal)
```

```
    {
```

```
        this.name = name;
```

```
        this.id = id;
```

```
        this.sal = sal;
```

```
    }  
    public String toString()
```

```
    {
```

```
        return "name=" + name + ", id=" + id + ", sal=" + sal;
```

```
    }  
}
```

```
public class Deserialization
```

```
{
```

```
    public static void main()
```

```
    {
```

```
        try
```

```
        {
```

```
            FileInputStream f1 = new FileInputStream("abc.txt");
```

```
            ObjectInputStream f2 = new ObjectInputStream(f1);
```

```
            Employee e = (Employee) f2.readObject();
```

```
            f2.close();
```

```
            System.out.println("Employee object " + e);
```

```
        }  
    }  
    catch (Exception e)
```

```
    {  
        System.out.println("Error " + e.toString());
```

```
    }  
}
```

O/P: Employee object name = emp, id = 101, sal = 50.0

console class:

java console class is used to get i/p from console. it provides methods to read texts & passwords. if you read password using console class, it will not be displayed to the user.

java io.console class is attached with system console internally.

Eg:- import java.io.Console;

class ReadTest

{

public static void main ()

{

Console c1 = System.console();

System.out.println(" Enter your name");

String s1 = c1.readLine();

System.out.println(" Hi " + s1);

System.out.println(" Enter pwd");

char[] ch = c1.readPassword();

String s2 = String.valueOf(ch);

// converting char array into string

System.out.println(" pwd " + s2);

}

}

o/p: Enter your name: padma

Hi padma

Enter pwd:

pwd : 1523

Files:

To store data we are using files in io package
io package provides some methods to know the properties
of file or dir.

boolean mkdir(File newDirectory)

File creation:

File obj = new File(filename);

File obj = new File(directoryname);

File obj = new File("path", filename);

File obj = new File("path", directoryname);

Eg:- // Creating file and check physical existence.

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

```
class FileProperties
```

```
{
```

```
    p.s.v.m()
```

```
{
```

```
    String name = args[0];
```

```
    File f = new File(name);
```

```
    s.o.p("filename:" + f.getName());
```

```
    s.o.p("path" + f.getPath());
```

```
    s.o.p("absolute path" + f.getAbsolutePath());
```

```
    s.o.p("parent" + f.getParent());
```

```
    s.o.p("exists" + f.exists());
```

```
    if (f.exists())
```

```
    {
```

```
        s.o.p("writable" + f.canwrite());
```

```
        s.o.p("readable" + f.canread());
```

```
        s.o.p("directory" + f.isDirectory());
```

```
        s.o.p("size" + f.length());
```

```
    }
```


o/p: java FileProperties abc.txt
Filename: abc.txt
path : abc.txt
absolutePath: D:\padman\abc.txt
parent: null
Exists: true
writable: true
Readable: true
directory: false
Size: 20

// writing text into file

```
import java.io.*;  
class WriteData  
{  
    p.s.v.m ( )  
    {  
        String filename = "temp.txt";  
        try  
        {  
            String bytes = "Hello";  
            byte[] buffer = bytes.getBytes();  
            FileOutputStream fow = new FileOutputStream(filename);  
            fow.write(buffer);  
            fow.close();  
            s.o.p ( buffer.length + " bytes" );  
        }  
        catch (IOException e)  
        {  
            s.o.p ( "Error writing file" + filename );  
        }  
    }  
}
```

Random access files:

we can read from a file & write to the file. Reading & writing can be done by using the file input & output streams. we can read & write file at any position by using modes.

mode:

r - The file is opened in a read-only mode

rw - The file opened in a read-write mode

The file is created if it does not exist.

rws - The file is opened in read-write mode
any modifications done at meta data it
will be stored into storage device immediately

rwd - The file opened in read-write mode.
& modifications done.

w - write mode.

Constructors

Syntax: `RandomAccessFile(String name, String mode)`

or

`RandomAccessFile(File file, String mode)`

Eg:- `import java.io.*;`

`public class Random`

`{`

`int i;`

`p.s.v.m()`

`{`

`try`

`{`

`Random R1 = new Random();`

`R1.Creation();`

`}`


```
catch (IOException e)
```

```
{
```

```
    s.o.p(e);
```

```
}
```

```
}
```

```
void creation() throws IOException
```

```
{
```

```
    File f1 = new File("abc.txt");
```

```
    RandomAccessFile fr = new RandomAccessFile(f1, "rw");
```

```
    RandomAccessFile (f1, "rw");
```

```
    for (i=1; i<=5; i++)
```

```
    {
```

```
        fr.writeInt(i);
```

```
    }
```

```
    fr.close();
```

```
}
```

Enumerations: java.lang.Enum

Enumeration: it is similar to a class that contains only constant values. Constant represents a fixed value and it does not change.

Eg:- there are fixed no. of days.

Enum Days

² Sunday, Monday, Tuesday, Wednesday, Thursday,
Friday, Saturday

3

→ enum is a keyword it mainly used for to represent sequence of constant values.

→ To retrieve the values we use `Days`. `sunday` or
by using `values` method. `enum type`

using values method.

eg: Days allDays[] = Days.values();

↓

To store all constants like Monday.

↓ days enumeration

To store all constants like Monday, ... return

→ one more predefined method `valueOf()`. it retrieve the constant value corresponding to string passed.

eg:- $f_1 = \text{days-values of ("sunday")}$

$S.O.P(f_1) \rightarrow \text{Sunday}$. $\perp$ string

11 Enumeration with constructor, instance variable & method.

Enum Student

5

John(11), Bella(10), Sam(13), Varj(9)

```
private int age;
```

```
int getage() { return age; } // method
```



```

public Student(int age) { // constructor
    {
        this.age = age;
    }
}

class EnumDemo
{
    p, s, v, m()
    {
        Student s; // enum variable.
        s.o.p("age of sam" + student.sam.getAge() +
            "years");
    }
}

```

o/p: age of sam is years.

- Enumerations can have constructors, instance variables, methods and can even implement interfaces.
- Enumerations are not instantiated using new keyword.

Auto boxing

- Autoboxing is a process of by which primitive type is automatically encapsulated (boxed) into its equivalent type wrapper.
- Auto-unboxing is a process by which the value of an object is automatically extracted from a 'type wrapper' class.
- Type wrapper class: if you want to create object for primitive data types, not possible in java, but in case of data structures java operate on object for that purpose java provides type wrappers which provide classes that encapsulate a primitive type with an object.

Eg:- character (char ch) - it encapsulates primitive type char within object.

→ Auto unbboxing is a process by which the value of an object is automatically extracted from a type wrapper class.

Benefits of autoboxing /unboxing:

→ we don't have to perform explicit typecasting

→ it helps prevent errors.

// Eg: program for autoboxing / auto-unboxing.

```
Class autoboxing
```

```
{
```

```
    P. S. V. m ()
```

```
    {
```

```
        Integer i = 100; // autoboxing
```

```
        int i2 = i; // auto-unboxing
```

```
        S.o.p (i + " " + i2);
```

```
        Character c = 'a';
```

```
        char c2 = c;
```

```
        S.o.p (c2 + " " + c);
```

```
        Double d = 33.3 // no need to typecasting.
```

```
        d = d + 1;
```

```
        S.o.p ("value of d is " + d);
```

```
    }
```

```
}
```

O/P: 100 100

a a

value of d is 133.3

Generics:

A generic type represents a class or an interface that is type-safe. It can act on any data type.

Syntax: `classname <data-type> referencename = new classname`
(or) `<data-type> ()`

`classname <datatype> referencename = new classname <T> ()`

diamond notation

// Example of Generic class

```
class Generic<T>  // generic type
```

```
{  
    T object; // object of type T
```

```
    Generic(T ob) // constructor
```

```
{
```

```
    this.object = ob;  
}
```

```
    public T getObject()
```

```
{  
    return this.object;  
}
```

```
}  
class Test
```

```
{  
    p.s.v.m()
```

```
{
```

```
        Generic<Integer> g = new Generic<T>(100);
```

```
        int x = g.getObject();
```

```
        S.o.p(x) g.getObject();
```

```
        Generic<String> s = new Generic<T>("Hello");
```

```
        String str = s.getObject();
```

```
        S.o.p(str) g.getObject();  
    }  
}
```

Generic method:

we can also create generic method with different types of arguments.

Syntax:

```
<type-parameter> returntype methodname(parameters)
```

```
{
```

```
}
```

Generic Interface:

Syntax: interface <name of interface> <T>

```
{
```

```
}
```

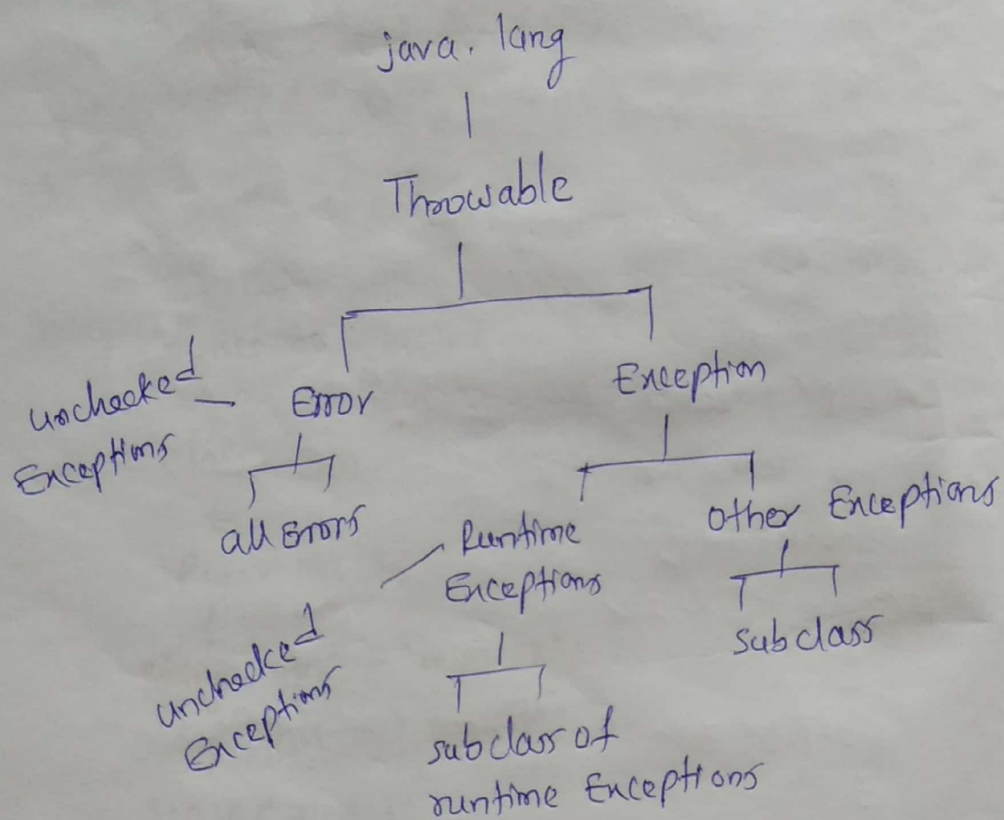

→ To design the code or developing the code we commit several errors. These errors are called bugs. And the process of removing them is called debugging.

Errors are compile time errors: syntactical errors

Runtime errors: inefficiency of the computer system to execute a particular stmt.

Logical errors: flaws in the logic of the program.

→ Exceptions: Exception is a runtime error.



The Exception hierarchy in java.

- Throwable is a class that represents all errors and Exceptions which may occur in java.
- Exception is a super class of all Exceptions in java.

- An Exception can be handled by using try catch and finally blocks.
- To handle multiple Exceptions we use multiple catch blocks.
- A single try block can be followed by several catch blocks.
- we can write a catch without try block, but we can write a try without any catch block.
- it is not possible to insert some stmts between try & catch.
- we can write try with in try, is called nested try block.

Eg: use of throws clause

```
import java.io.*
```

```
class Example
```

```
{
    private String name; // instance variable
```

```
void accept() throws IOException
```

```
{
    BufferedReader br = new BufferedReader (
        new InputStreamReader (System.in));
```

```
System.out.println("Enter name: ");
```

```
name = br.readLine();
```

```
}
```

```
void display()
```

```
{
    S.O.P ("name: " + name);
```

```
}
```


class Test

{

p.s.v.m() throws IOException

{

Example e = new Example();

e.accept();

e.display();

}

}

Eg:- use of throw clause:

class Example

{

static void demo()

{

try

{ s.o.p("inside demo");

throw new NullPointerException("Exp data");

}

catch (NullPointerException e)

{ s.o.p(e);

}

}

}

class ThrowDemo

{

p.s.v.m()

{

Example demo();

}

}

User Defined Exceptions:

① The user should create an exception class as a subclass to Exception class.

class myException extends Exception

② his own exception class

```
myException ()
```

```
{  
    ...  
}
```

③ The user can create parameterized constructor with a string parameter, to store details.

```
myException (String str)
```

```
{
```

```
    super(str); // call Exception class
```

```
}
```

constructor & store str here.

④ when the user wants to raise his own exception, he should create an object to his Exception class & throw it using throw clause.

```
myException s = new myException ("exp details");
```

```
throw s;
```

& throw it
out of method

Note: the programmer does not want to handle the exception we use throws, To throw exception explicitly & wants to handle it using catch blocks we use throw block.

Re-throwing an Exception:

```
try {
```

```
    throw Exception;
```

```
}
```

```
catch (Exception obj)
```

```
{
```

```
    throw exception; // re-throw exception
```

```
}
```

Re-throw an Exception from catch block to another class where it can be handled.

// Rethrowing an Exception

```
class A
```

```
{  
    void method ()
```

```
{
```



```

try
{
    String str = "Hello";
    char ch = str.charAt(5);
}
catch (StringIndexOutOfBoundsException e)
{
    s.o.p ("index out of bound");
    throw e; // rethrow an exception
}
}

class B
{
    p.s.v.m()
    {
        A a = new A();
        try {
            a.method1();
        }
        catch (StringIndexOutOfBoundsException e)
        {
            s.o.p ("I caught rethrown exception");
        }
    }
}

```

→ rethrown exception can be caught by this catch block

O/p: index out of bound
I caught rethrown exception.

PART-II

Multithreading: Differences between thread-based multithreading and process based multithreading, java thread model, creating threads, thread priorities, synchronizing threads, inter thread communication.

Thread: Thread is a light-weight process because they utilize minimum resources of the system. This means they take less memory and less processor time.

In java program if we write a group of stmts, then these stmts are executed by JVM one by one. This execution is called a thread, because JVM uses a thread to execute these stmts. That means by internally in every java program main thread is running.

A thread represents execution of stmts. The way the stmts are executed is of two types.

- single tasking
- multi tasking

Single tasking: In single tasking, only one task is given to the processor at a time. This means we are waiting a lot of processor time & microprocessor has to sit idle without any job for a long time. This is drawback of single tasking.

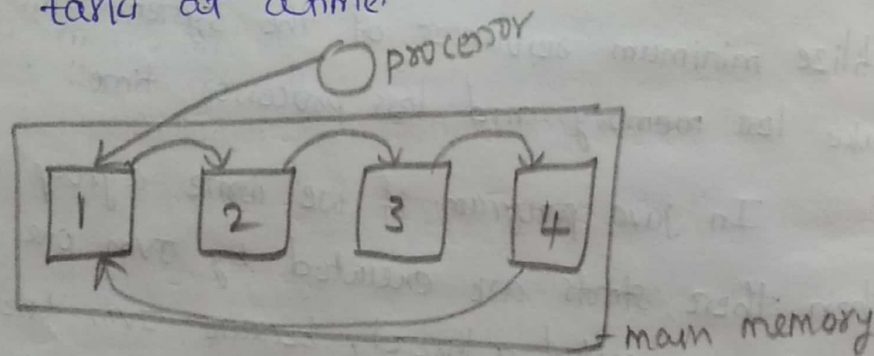
Multi-tasking: We can execute several tasks at a time is called multitasking.

Multitasking is of two types

- process-based multi tasking
- Thread-based multi tasking.

process based multitasking:

In process based multitasking, execute several tasks at a time.



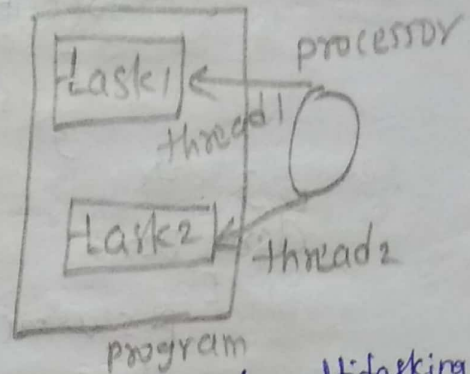
process based multitasking.

Suppose 4 tasks are ready to execute, we load them into the memory. memory divided into 4 parts & the jobs are loaded there. Now the microprocessor has to execute them all at a time. the small part of the processor is called time slice.

Thread-based multitasking:
in process-based multi-tasking, several programs are executed at a time, by the microprocessor. in thread based multitasking, several parts of the same program is executed at a time by the microprocessor.

thread uses:

- Threads are mainly used in server side programs.
- Threads are also used to create games and animations.



Thread based multitasking

Creating a Thread:

In every java program main thread available.
→ create a class that extends thread class or implements Runnable interface. there are present in lang package.

class classname extends Thread (or)
class classname implements Runnable

→ Now include run() method.

```
public void run()
```

```
{ // stmts;
```

```
}
```

→ create object for thread class

```
Thread t = new Thread(obj);
```

→ pass class object name

```
(or) Thread t = new Thread(obj, "threadname");
```

→ To start thread

```
t.start();
```


// To create a thread & run it

```
class MyThread extends Thread
{
    public void run()
    {
        for(int i=1; i<=50000; i++)
        {
            s.o.p(i);
        }
    }
}
```

```
class Demo
{
    p.s.v.m()
    {
        myThread m1 = new MyThread();
        Thread t = new Thread(m1);
        t.start();
    }
}
```

// To find currently running thread

```
class Current
{
    p.s.v.m()
    {
        s.o.p("find current thread");
        Thread t = Thread.currentThread();
        s.o.p("current thread: " + t);
        s.o.p("its name: " + t.getName());
    }
}
```

O/p:

find current thread

current thread: Thread [main, 5, main]

its name: main

↳ by default running thread

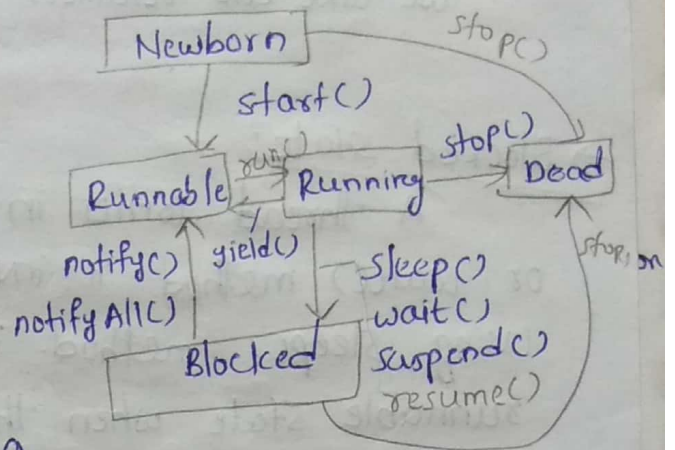
default
priority
thread group
name

(current thread name)

Thread Life cycle:

During execution, a thread can be in one of the following five states.

- Newborn state
- Runnable state
- Running state
- Blocked state
- Dead state.



→ Newborn state: When we create a new thread object using new operator thread life cycle thread state is new thread.

→ Runnable state: A thread will be called runnable when the start() method is invoked. After calling the start method, the thread joins the thread queue & waits for its turn to get execute.

→ Running state: Running state means that the processor has given its time to thread for its execution. A running thread can give its control in one of the following reasons.

sleep(): method is invoke and the thread goes into blocked state for specific time period.

yield(): it is invoke to give control from the current thread & put it back to the thread queue for execute.

wait(): it invoke & the thread goes into blocked state until some other thread doesn't invoke notify() or notifyAll() method.

suspend():

this method puts a thread in the suspended state & can be resumed using resume() method.

resume(): this method resumes a thread, which was suspended using suspend() method.
we can't use resume() method without suspend() method.

→ Blocked state:

A thread comes in blocked state when sleep() or wait() method is invoked. when a thread blocks using sleep() method, it will automatically come in runnable state when the sleep time over.

notify(): it wakes up only one thread waiting on the object and that thread starts execution. if there are multiple threads waiting for single object it allows only one thread.

notifyAll(): if a thread it wakes up all the threads waiting on the object, although which one will process first depends on the OS implementation.

→ Dead state:

if a thread has finished its execution or its run() method & when stop() method is invoked. the stop() method kills the thread & thread doesn't work further.

// Example for thread

```
public class A extends Thread
```

```
{ public void run()
```

```
{ s.o.p(" Thread A");
```

```
s.o.p(" i in thread A");
```

```
for(int i=1; i<=5; i++)
```

```
{ s.o.p(" i = " + i);
```

```
try {
```

```
Thread.sleep(1000);
```

```
} catch (InterruptedException e)
```

```
{ e.printStackTrace();
```

```
} s.o.p(" Thread A completed");
```

```
}
```

```
public class B extends Thread
```

```
{ public void run()
```

```
{ s.o.p(" Thread B");
```

```
s.o.p(" i in thread B");
```

```
for(int i=1; i<=5; i++)
```

```
{ s.o.p(" i = " + i);
```

```
} s.o.p(" Thread B completed");
```

```
}
```

```
public class main
```

```
{ p.s.v.m()
```

```
{ A a1 = new A();
```

```
B b1 = new B();
```

} both threads are runnable state

a₁.start(); // Running state of thread A
a₁.yield(); // To move control to another thread
try {
 a₁.sleep(1000);

}
catch (InterruptedException e₁)

{
 e₁.printStackTrace();

}
b₁.start();

s.o.p ("main thread end")

}

// Single tasking using thread
// program for execution of multiple tasks with a single thread

class MyThread implements Runnable

```
{  
    public void run()  
    {  
        task1();  
        task2();  
        task3();  
    }  
    void task1()  
    {  
        s.o.p("This is task 1");  
    }  
    void task2()  
    {  
        s.o.p("This is task 2");  
    }  
    void task3()  
    {  
        s.o.p("This is task 3");  
    }  
}
```

class main

```
{  
    p.s.v.m()  
    {  
        MyThread m1 = new MyThread();
```

// Create thread t1 & attach it to that object

```
Thread t1 = new Thread(m1);
```

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

O/p: This is task1

This is task2

This is task3

- Threads implemented by using Runnable interface.
- extends Thread and implements Runnable - both are functionally same. But when we write extends Thread there is no scope to extend another class, as multiple inheritance is not supported in Java.

class myclass extends Thread, anotherclass // invalid

- if we write implements Runnable interface, then still there is scope to extend another class.

class myclass extends anotherclass implements Runnable // valid.

// Multi tasking using threads

// Write a program showing two tasks at a time

```
class MyThread implements Runnable
```

```
{
    String s1; // it represent task.
```

```
    MyThread(String s1)
```

```
    {
        this.s1 = s1;
```

```
    }
    public void run()
```

```
    {
        for (int i = 1; i <= 10; i++)
```

```
        {
            s.o.p(s1 + " : " + i);
```

```
            try {
```

```
                Thread.sleep(2000);
```

```
            } catch (InterruptedException e) {
```

```
                {
                    e.printStackTrace();
```

```
                }
            }
        }
    }
}
```

```
class Theatre
```

```
{
```

```
    p.s.v.m()
```

```
{
```

`MyThread m1 = new MyThread("cut the ticket");`
`MyThread m2 = new MyThread("show the seat");`
To represent two tasks.

`Thread t1 = new Thread(m1);`
`Thread t2 = new Thread(m2);` } create 2 threads & attach two objects.

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

y
y

Thread priority / Thread priorities:

Thread priorities can change from 1 to 10. we can use the following constants to represent priorities.

	Thread.MAX_PRIORITY	value is 10
	Thread.MIN_PRIORITY	value is 1
default priority	Thread.NORM_PRIORITY	value is 5

Methods in thread class:

- ① Thread t = Thread.currentThread();
To know the current running thread
- ② to start thread : t.start();
- ③ To stop execution of a thread for specific time
Thread.sleep(milliseconds);
- ④ To get the name of thread
String name = t.getName();
- ⑤ To set a new name to a thread
t.setName("New Name");
- ⑥ To set priority of a thread
t.setPriority(int priorityno); // t.setmaxPriority();
- ⑦ To test if a thread still alive
t.isAlive() - return true/false
- ⑧ To wait till a thread dies:
t.join();

// Write a program to understand the thread priorities
The thread with higher priority will complete its execution first.

```
class ThreadPriority extends Thread
{
    int count = 0;
    public void run()
    {
        for(int i = 1; i <= 10000; i++)
            count++;
        S.o.p("completed Thread "
            + Thread.currentThread().getName());
        S.o.p("Its priority " + Thread.currentThread().getPriority());
    }
}
```

33

class prior

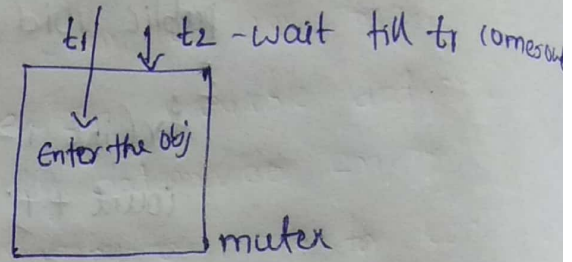
```
{
    p.s.v.m()
    {
        ThreadPriority t1 = new ThreadPriority();
        Thread a1 = new Thread(t1, "one");
        Thread b1 = new Thread(t1, "two");
        a1.setPriority(2);
        b1.setPriority(Thread.NORM_PRIORITY);
        a1.start();
        b1.start();
    }
}
```

3

Synchronizing Threads:

When a thread is already acting on an object preventing any other thread from acting on the same object is called 'Thread synchronization' or 'thread safe'. The object on which the threads are synchronized is called 'synchronized object'. it is used in multithreading.

thread t_1 locks the object after entering and t_2 waits till t_1 leaving this condition is called mutex or exclusion mutually exclusive lock.



Thread synchronization

→ To synchronize the object we use synchronized block inside run method.

```
synchronized (Object)
{
    stmts;
}
```

→ using synchronized keyword, we can synchronize an entire method

Eg:- synchronized void display()

```
{
    stmts;
}
```

// program for synchronize the threads acting on the same object the synchronized block in the program can be executed by only one thread at a time.

class Reserve implements Runnable

```
{
    int available = 1, wanted;
```

```

Reserve (int i)
{
    wanted = i;
}

public void run()
{
    synchronized (this) // synchronize the current object
    {
        s.o.p("available berths = " + available);
        if (available >= wanted)
        {
            String name = Thread.currentThread().getName();
            s.o.p(wanted + " Berths reserved for " + name);
            try {
                Thread.sleep(1500);
                available = available - wanted;
            }
            catch (InterruptedException e)
            {
            }
        }
        else s.o.p("no berths");
    }
}

```

```

}
}
class Hospital

```

```

{
    p.s.v.m.c()

```

```

    {
        Reserve r1 = new Reserve();

```

```

        Thread t1 = new Thread(r1);

```

```

        Thread t2 = new Thread(r1);

```

```

        t1.setName("first person");

```

```

        t2.setName("second person");

```

```

        t1.start();

```

```

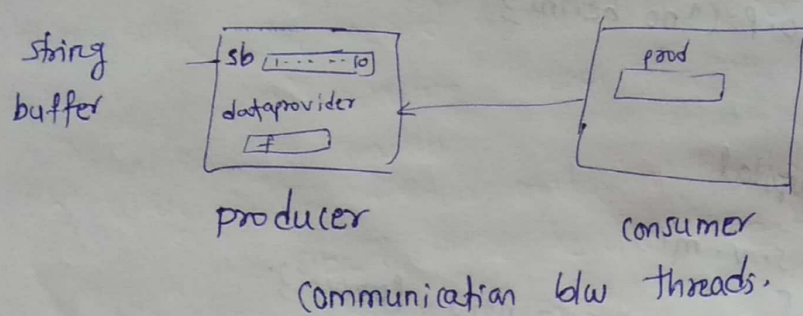
        t2.start();
    }
}

```


Inter Thread Communication:

In some cases, two or more threads should communicate with each other.

- Eg:- A consumer thread is waiting for a producer the data. When the producer thread completes production of data, then consumer thread should take that data & use it.
- To store data in producer class we use StringBuffer object. we take another boolean variable dataproducer & initialize to false (initially dataproducer empty)
 - producing data append to StringBuffer using for loop. we come out of for loop & store true into dataproducer.
 - if dataproducer is true, then consumer takes data from StringBuffer, if it is false, then consumer will sleep for some time & again check the condition.



// write a program where the consumer thread checks whether the data production is over or not every 10 milliseconds.

Class Communicate

{ p. s. v. m() throws Exception

{ producer p1 = new Producer();

Consumer C1 = new Consumer(); // consumer C1 = new Consumer(A)

Thread t1 = new Thread(p1);

Thread t2 = new Thread(C1);

```
t2.start(); // consumer waits  
t1.start(); // producer starts production
```

```
}  
}  
class producer extends Thread  
{
```

```
boolean dataproducer = false;
```

```
producer()
```

```
{
```

```
StringBuffer sb = new StringBuffer(); // allot memory  
}
```

```
public void run()
```

```
{
```

```
// synchronized (sb)
```

```
for(int i=1; i<=10; i++)
```

```
{
```

```
try {
```

```
sb.append(i + " ");
```

```
Thread.sleep(100);
```

```
s.o.p("appending");
```

```
}
```

```
catch (Exception e)
```

```
{ }
```

```
} dataproducer = true;
```

```
}
```

```
}
```

```
class Consumer extends Thread
```

```
{
```

```
// create producer reference object from consumer class
```

```
Producer prod;
```

```
Consumer(Producer prod)
```

```
{
```

```
this.prod = prod;
```

```
}
```

```
public void run()
```

```
{
```

```
try {
```

```
// check for data production  
while(!prod.dataprovider)
```

```
Thread.sleep(10);
```

```
catch
```

```
(Exception e) { }
```


S.o.p (prod.s.b) // To print data production

3

3

→ To improve the efficiency of communication b/w threads we use three methods.

Obj. notify(): This method release an object & sends a notification to a waiting thread that the object is available.

Obj. notifyAll(): This method is useful to send notification to all waiting threads at once that the object is available.

Obj. wait(): This method makes a thread wait for the object till it receives a notification from a notify() or notifyAll() methods.

→ Sometimes, a thread has to continuously execute without any interruption to provide services to other threads, such threads are called daemon thread. it generally provides a background processing.

To make thread as daemon thread

t.setDaemon(true).

To check if a thread is daemon thread or not

boolean x = t.isDaemon();

Deadlock of Threads:

When a thread has locked an object and waiting for another object to be released by another thread, and the other thread is also waiting for the 1st thread to release the 1st object, both the threads will continue waiting forever. This is called thread deadlock.

To avoid the deadlocks we are using synchronized keyword.

// Create deadlock between 2 threads.

```
class myThread
```

```
{
```

```
String s1 = "java";
```

```
String s2 = "unix";
```

```
Thread t1 = new Thread("my thread1");
```

```
{
```

```
public void run()
```

```
{
```

```
while (true) {
```

```
synchronized (s1)
```

```
{
```

```
s.o.p(s1 + s2);
```

```
}
```

```
}  
}  
}
```

```
Thread t2 = new Thread("my thread2");
```

```
{
```

```
public void run()
```

```
{
```

```
synchronized (s2)
```

```
{
```

```
s.o.p(s2 + s1);
```

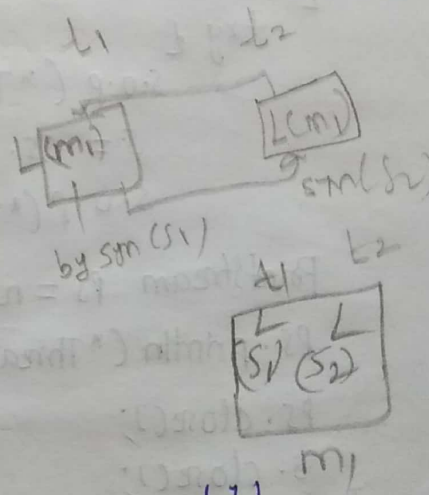
```
}
```

```
}
```

```
myThread m1 = new myThread();
```

```
m1.t1.start();
```

```
m1.t2.start();
```



Applications of Threads:

// Write a program to create a server with 2 threads to communicate with several clients.

// A server with 2 threads to contact multiple clients

```
import java.io.*;
import java.net.*;
class MultiServer implements Runnable
{
    static ServerSocket ss;
    static Socket s;
    public void run()
    {
        String name = Thread.currentThread().getName();
        for(;;) // server runs continuously
        {
            try {
                s.o.p("Thread "+name+" ready to accept");
                s = ss.accept();
                s.o.p("Thread "+name+" accepted a connection");
                PrintStream ps = new PrintStream(s.getOutputStream());
                ps.println("Thread "+name+" contacted you");
                ps.close();
                s.close();
            }
            catch (Exception e) { }
        }
    }
}
p.s.v.m()
{
    MultiServer m = new MultiServer();
    ss = new ServerSocket(999);
    Thread t1 = new Thread(m, "one");
    Thread t2 = new Thread(m, "Two");
    t1.start();
    t2.start();
}
```

// Program for to create client that receives msg from server.

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

```
import java.net.*;
```

```
class Multiclient
```

```
{
```

```
    p.s.v.m ( ) throws Exception
```

```
    {
```

```
        Socket s = new Socket("localhost", 999)
```

```
        BufferedReader br = new BufferedReader(new InputStreamReader  
            ( s.getInputStream() ));
```

```
        String str;
```

```
        while ((str = br.readLine()) != null)
```

```
            s.o.p(str);
```

```
            br.close();
```

```
            s.close();
```

```
    }  
}
```


UNIT-IV

The collections Framework (java.util)

collection overview, collection interfaces, The collection classes - Array List, Linked List, Hash set, Tree set, priority queue, Array Deque. Accessing a collection via an iterator, using an iterator, The for-each alternative, map interfaces & classes, comparators, collection Algorithms, Arrays, the Legacy classes and interfaces - Dictionary, hashtable, properties, stack, vector, more utility classes, string Tokenizer, BitSet, Date, calendar, Random, formatter, scanner.

Collection Framework:

A collection framework is a class library to handle group of objects. collection framework is implemented in java.util package.

Collection object or container object is an object which can store a group of other objects. A collection object has a class called 'collection class' or 'container class'.

All the collection classes in java.util package are the implementation classes of different interfaces.

Interface type

implementation classes

Set <T>

HashSet <T>, LinkedHashSet <T>

List <T>

Stack <T>, Vector <T>
ArrayList <T>, LinkedList <T>

Queue <T>

LinkedList <T>, PriorityQueue <T>

map <K, V>

HashMap <K, V>

Hashtable <K, V>

Sets:

A set represents a group of elements arranged just like an array. The set will grow dynamically when the elements are stored into it. A set will not allow duplicate elements.

Lists: Lists are like sets. They store a group of elements. But lists allow duplicate values to be stored.

Queues: A queue represents arrangements of elements in FIFO order. This means that an element is stored as a first element into queue will be removed first.

Maps: It stores the elements in the form of key & value pairs. If the key is provided then its corresponding value can be obtained. The keys should have unique values.

Retrieving elements from collections:

- using for-each loop
- using Iterator interface
- using ListIterator interface
- using Enumeration interface

For-each loop: which repeatedly executes a group of statements for each element of the collection.

```
for (variable : collection-object)
{
    statements;
}
```

Iterator Interface:

Iterator is an interface that contains methods to retrieve the elements ^{forward direction} one by one from a collection object. It has 3 methods.

boolean hasNext(): returns true if it has more elements

element Next(): it returns next element of iterator

void remove(): It removes from the collection last element returned by the iterator.

ListIterator Interface:

it is an interface that contains methods to retrieve the elements one by one from a collection object, both in forward & reverse directions.

- ① boolean hasNext(): it returns true if the listiterator has more elements when traversing the list in the forward direction.
- ② boolean hasPrevious(): it returns true, if the listiterator has more elements when traversing the list in the reverse direction.
- ③ element next(): This returns the next element in the list
- element previous(): This returns the previous element in the list
- ④ void remove(): it removes last element that was returned by the next() or previous() methods.

Enumeration Interface:

This interface is useful to retrieve one by one the elements like the iterator. it has 2 methods

- ① boolean hasMoreElements(): this method tests if the Enumeration has any more elements or not.
- ② element nextElement(): it returns next element that is available in Enumeration.

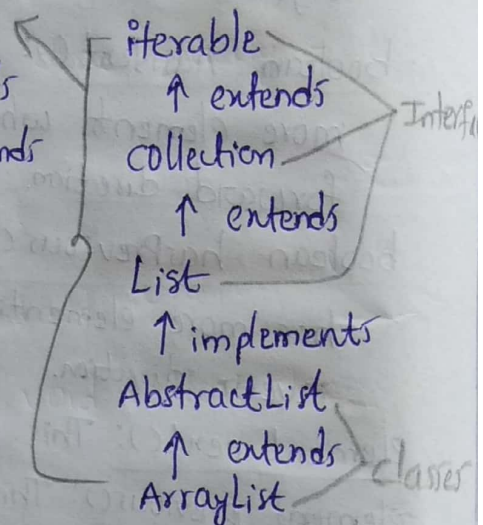
Iterator is preferable compare to enumeration because it include the 3 methods and easy to remember compare to enumeration.

Collection classes: ① ArrayList:

Java ArrayList class uses a dynamic array for storing the elements. it inherits AbstractList class & implements List interface.

- Java ArrayList class can contain duplicate elements
- manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list.

ArrayList class extends AbstractList class which implements ListInterface. the list interface extends collection & iterable interfaces.



ArrayList Class methods:

- boolean add(element obj): it add the element at the end of the List, if successfully added it returns True.
- void add(int position, element obj): To add specific position
- boolean remove(object obj): it removes the 1st occurrence of the specified element obj from the ArrayList
- void clear(): it removes all elements from ArrayList.
- int size(): it returns no. of elements present in ArrayList
- element set(int pos, element obj): To replace obj to specified position.

// WP. to create an ArrayList with strings and perform various operations on it.

```
import java.util.*;
class ArrayListDemo
{
    p. s. v. mc )
    {
```



```
ArrayList<String> a1 = new ArrayList<String>();
```

```
a1.add("Apple");
```

```
a1.add("Mango");
```

```
a1.add("Grapes");
```

```
a1.add("Guava");
```

```
S.O.P ("contents : " + a1);
```

```
a1.remove(3);
```

```
a1.remove("mango");
```

```
S.O.P ("contents after removing : " + a1);
```

```
S.O.P ("size of ArrayList : " + a1.size());
```

```
S.O.P ("Extract elements using iterator:");
```

```
Iterator t = a1.iterator();
```

```
while (t.hasNext())
```

```
{ S.O.P (t.next());
```

```
}
```

```
}
```

```
}
```

O/P: Contents: [Apple, Mango, Grapes, Guava]

contents after removing: [~~mango~~^{Apple}, Grapes]

Size of ArrayList: 2

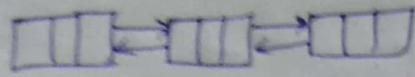
Extract elements using iterator:

~~mango~~ Apple

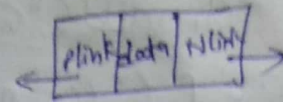
Grapes.

Linked List:

→ A linked list contains a group of elements in the form of nodes. each node contains 3 fields i.e, data field, link field (To store previous node & next node references).



a linked list



node structure

→ it is very convenient to store data. inserting & removing of elements from linked list is done quickly & takes the less amount of time.

LinkedList methods:

- boolean add(element obj), void add(int pos, element obj)
- void addFirst(element obj), void addLast(element obj)
- element removeFirst()
- " " Last(), element remove(int pos)
- void clear()
- element get(int position)
- element getFirst()
- element getLast()
- int size(), element set(int pos, element obj)

Eg:- // WP for that shows use of LinkedList class

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

```
    p.s.v.m ( ) throws IOException
```

```
{
```

```
    LinkedList<String> l1 = new LinkedList<String> ( )
```

```
    l1.add ("India")
```

```
    l1.add ("Sri Lanka")
```



```
l1.add("Japan");  
s.o.p("List = " + l1);  
BufferedReader br = new BufferedReader(new InputStreamReader  
    (System.in));
```

```
String e;
```

```
int position, choice = 0;
```

```
while (choice < 4)
```

```
{
```

```
    s.o.p("In Linked List operations");
```

```
    s.o.p("In 1. add an element in 2. Remove an element
```

```
        in 3. change an element in 4. exit");
```

```
    s.o.p("Enter your choice");
```

```
    choice = Integer.parseInt(br.readLine());
```

```
    switch (choice)
```

```
    {
```

```
        case 1: s.o.p("Enter element");
```

```
            e = br.readLine();
```

```
            s.o.p("At what position?");
```

```
            position = Integer.parseInt(br.readLine());
```

```
            l1.add(position-1, e);
```

```
            break;
```

```
        case 2: s.o.p("Enter position");
```

```
            position = Integer.parseInt(br.readLine());
```

```
            l1.remove(position-1);
```

```
            break;
```

```
        case 3: s.o.p("Enter position");
```

```
            position = Integer.parseInt(br.readLine());
```

```
            s.o.p("Enter new element");
```

```
            e = br.readLine();
```

```
            l1.set(position-1, e); // To replace specified position  
            break; // with element object.
```

```
        default: return;
```

```
    }
```

Iterator t = l.iterator(); // To retrieve elements use Iterator
while (t.hasNext())
S.o.p("List = " + t.next() + " ");

}

}

}

O/p: List = [India, srilanka, japan]

LinkedList operations

1. add an element
2. Remove an element
3. change an element
4. Exit

Your choice: 1

Enter element: UK

At what position? 1

List = UK, India, srilanka, japan

HashSet:

A HashSet represents a set of elements (objects). It does not guarantee the order of elements. & it does not allow the duplicate elements to be stored.

```
class HashSet <T>  
    ↳ generic type
```

constructors:

```
HashSet();
```

```
HashSet(int capacity);
```

```
HashSet(int capacity, float load-factor);
```

default capacity 16 & load-factor 0.75

HashSet class methods:

- int size();
- boolean isEmpty();
- void clear();
- boolean remove(obj);
- boolean add(obj);
- boolean contains(obj)

Eg:- // WP which shows the use of HashSet & iterator

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

```
class HashSetDemo
```

```
{
```

```
    p. s. v. m ()
```

```
{
```

```
    HashSet<String> h = new HashSet<String> ();
```

```
    h.add ("Indra");
```

```
    h.add ("Japan");
```

```
    h.add ("Srilanka");
```

```
    s.o.p ("HashSet = " + h);
```

```
    Iterator t = h.iterator();
```

```
    s.o.p ("Elements using iterator");
```

```
while (t.hasNext())
```

```
{
```

```
    String s = (String) t.next();
```

```
    s.o.p(s);
```

```
}
```

```
}
```

```
}
```

O/p: Hash set = [India, Sri Lanka, Japan]

Elements using iterator:

India

Japan

Sri Lanka

- HashSet is faster than TreeSet and should be preferred choice if sorting of element is not required
- HashSet doesn't guarantee any order while TreeSet maintains objects in sorted order.

Tree set:

TreeSet class implements the set interface that uses a tree for storage. It inherits AbstractSet class & implements NavigableSet interface. The objects of TreeSet class are stored in ascending order.

→ it contains unique elements only like HashSet.

Methods:

- boolean isEmpty()
- boolean contains(Object obj)
- boolean addAll(Collection c)
- Object clone()

eg: // WP for TreeSet

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

```
class TreeSetDemo
```

```
{  
    p.s.v.m()
```

```
{  
    TreeSet<String> t = new TreeSet<String>();
```

```
    t.add("inorder");
```

```
    t.add("postorder"); t.add("preorder");
```

```
    t.add("postorder");
```

```
    Iterator<String> t1 = t.iterator();
```

```
    while (t1.hasNext())
```

```
    {  
        s.o.p(t1.next());  
    }  
}
```

o/p: inorder
postorder
preorder

priorityQueue: [Queue interface]

priorityQueue uses the queue. but it does not orders the elements in FIFO manner. it inherits Abstract Queue class.

PriorityQueue < E >

Eg:-

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

```
class priorityQueueDemo
```

```
{
```

```
    p.s.v.m()
```

```
    {  
        priorityQueue <String> q = new priorityQueue <String> ();
```

```
        q.add("Tom");
```

```
        q.add("Jerry");
```

```
        q.add("Bheem"); q.add("motu"); q.add("patlu");
```

```
        s.o.p("head" + q.element());
```

```
        s.o.p("head" + q.peek());
```

```
        s.o.p("iterating the queue elements");
```

```
        Iterator i = q.iterator();
```

```
        while (i.hasNext())
```

```
        {
```

```
            s.o.p(i.next());
```

```
        }
```

```
        q.remove();
```

```
        q.poll();
```

```
        s.o.p("after removing two elements");
```

```
        Iterator <String> i2 = q.iterator();
```

```
        while (i2.hasNext())
```

```
        {  
            s.o.p(i2.next());
```

```
        }
```

```
    }
```

```
}
```


Methods in Priority Queue:

- boolean add(Object)
- Object element() - it return top element of queue
- Object peek() - it returns head of queue, if queue is empty returns null.
- Object remove() - it removes head of queue.
- Object poll() - it removes head of queue, if queue is empty it returns null.

O/P: head Bheem

head Bheem

Iterating the queue elements

Bheem

Motu

Tom

Terry

Pathulu

It holds the elements & objects by their priority
doesn't store null

After removing two elements

Pathulu

Terry

Tom

ArrayDeque class: (implements Deque interface)

java Deque interface is a linear collection that supports elements insertion & removal at both ends & we can call it as double ended deque.

ArrayDeque class uses the deque & resizable-array

- unlike queue, we can add or remove elements from both sides.
- Null elements are not allowed in the ArrayDeque.
- it is faster than stack & LinkedList.

Methods:

- `offer()`;
- `boolean add()`;
- `offerFirst()`;
- `pollLast()` - ~~removes~~ Last element

Eg:- // WP for ArrayDeque

```
import java.util.*;
public class DequeDemo
{
    p.v.m.v()
    {
        Deque<String> d = new ArrayDeque<String>();
        d.offer("List");
        d.offer("Single linked list");
        d.add("double linked list");
        d.offerFirst("Circular linked list");
        s.o.p("after offerFirst Traversal");
        for(String s:d)
        {
            s.o.p(s);
        }
        d.poll();
        d.pollLast();
    }
}
```


S.O.P ("after pollLast() Traversal")

for (String s : d)

{ S.O.P(s);

}

O/p: After offerList traversal

Circular linked list

list

Single linked list

Double linked list

After pollLast() traversal

list

Single linked list

Deque : add elements both ends

double-ended-queue

it is faster than ArrayList & stack

no capacity restriction

// WRP for iterator interface

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

```
class Test
```

```
{
```

```
    p.s.v.m()
```

```
    {
```

```
        ArrayList<String> a = new ArrayList<String>();
```

```
        a.add("ab");
```

```
        a.add("cd");
```

```
        a.add("bc");
```

```
        Iterator t1 = a.iterator();
```

```
        while (t1.hasNext())
```

```
        {
```

```
            s.o.p(t1.next() + " ");
```

```
        }
```

// WRP for ListIterator

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

```
class Test
```

```
{
```

```
    p.s.v.m()
```

```
    {
```

```
        ArrayList<String> a = new ArrayList<String>();
```

```
        a.add("ab");
```

```
        a.add("bc");
```

```
        ListIterator L = a.listIterator();
```

```
        while (L.hasNext())
```

```
        {
```

```
            s.o.p(L.next() + " ");
```

```
        }
```

```
        while (L.hasPrevious())
```

```
        {
```

```
            s.o.p(L.previous() + " ");
```

```
        }
```

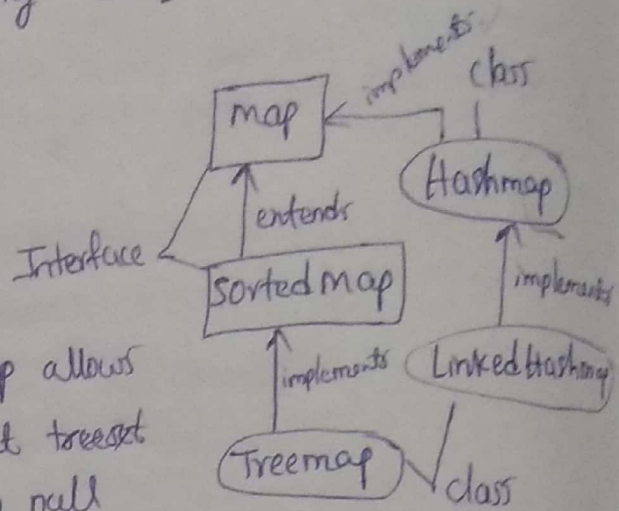

Map interface & classes:

A map contains values on the basis of key i.e., key and value pair. Each key and value pair is known as an entry. map contains only unique keys.

Map hierarchy:

Map doesn't allow duplicate keys, but you can have duplicate values.

→ HashMap & LinkedHashMap allows null keys and values but ~~tree~~ TreeMap doesn't allow any null key or value.



HashMap class:

- A HashMap contains values based on the key
- it contains only unique elements
- it may have null key and multiple null values
- it maintains no order.
- default capacity of hashmap is 16 and load factor is 0.75. Load factor represents at what level the hashmap capacity should be doubled.

Methods in hashmap:

keySet(): when applied on a hashmap converts it into a set where only keys will be stored.

value put(key, value): to store key-value pair into the hashmap.

value get(Object key): it returns corresponding value whose key is given. if key not available it returns null.

void clear()

boolean isEmpty()

int size()

// WAP to create telephone book using hashmap

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

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

```
class HashmapDemo
```

```
{  
    p.s.v.m() throws IOException
```

```
{
```

```
HashMap<String, Long> h = new HashMap<String, Long>();
```

```
String name, str;
```

```
Long phno;
```

```
BufferedReader br = new BufferedReader(new
```

```
InputStreamReader(System.in));
```

```
while(true)
```

```
{
```

```
    s.o.p("1. Enter phone entries\n2. Lookup in the  
    book\n3. Display names in book\n4. Exit");
```

```
    s.o.p("Enter your choice");
```

```
    int n = Integer.parseInt(br.readLine());
```

```
    switch(n)
```

```
{
```

```
        case 1: s.o.p("Enter name");
```

```
                name = br.readLine();
```

```
                s.o.p("Enter phno:");
```

```
                str = br.readLine();
```

```
                phno = new Long(str);
```

```
                h.put(name, phno) // store name & phno in hashmap
```

```
                break;
```

```
        case 2: s.o.p("Enter name");
```

```
                name = br.readLine();
```

```
                name = name.trim(); // remove unnecessary spaces
```



```
phno = h.get(name);
```

```
s.o.p("phno: " + phno);
```

```
break;
```

→ create hashtable object to store & defer it by 'set'

```
case 3: set<String> set = new HashSet<String>();
```

```
s = h.keySet(); // to display names
```

```
s.o.p(s);
```

```
break;
```

```
case 4: return;
```

```
}  
}  
}  
}
```

O/P: 1. Enter phno entries

2. Lookup in the book

3. Display Names in book

4. Exit

Your choice: 1

Enter name: padma

Enter phno: 9999999990

1. Enter phno entries

2. Lookup in the book

3. Display names in book

Your choice: 2

Enter name: padma

phno: 9999999990

1. Enter pho entries

2. Lookup in the book

3. Display names in book.

4. Exit

Your choice: 4

Hashtable class:

It is similar to HashMap which can store elements in the form of key-value pairs. But Hashtable is synchronized, ensuring proper results even if multiple threads act on it simultaneously.

// WAP to create hashtable that stores cricket players & scores

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

class HashtableDemo
{
    P.S.V.M() throws IOException
    {
        Hashtable<String, Integer> h = new Hashtable<String, Integer>();

        h.put("padma", 100);
        h.put("sachin", 102);
        h.put("dhoni", 50);

        S.O.P("The player names: ");
        Enumeration e = h.keys(); // to store keys into enumeration object
        while(e.hasMoreElements())
            S.O.P(e.nextElement());

        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

        S.O.P("Enter player names");
        String name = br.readLine();
        name = name.trim(); // remove unnecessary space
        Integer score = h.get(name);
        if(score != null)
        {
            int sc = score.intValue();
            S.O.P(name + " scored: " + sc);
        }
        else S.O.P("player not found");
    }
}
```


Difference between hashmap & hashtable

HashMap

- hashmap object is not synchronized by default.
- In case of single thread, using hashmap is faster than the hashtable.
- HashMap allows null keys and null values to be stored.
- Iterator in the hashmap is fail fast.

Hashtable

- hashtable object is synchronized by default.
- in case of multiple threads, using hashtable is advisable. with a single thread, hashtable becomes slow.
- Hashtable does not allow null keys or values.
- Enumeration for the hashtable is not fail-fast.

Set

- A set represents a collection of elements order of the elements may change in the set.
- Set will not allow duplicate values to be stored.
- Accessing elements by their index is not possible in case of sets.
- Sets will not allow null elements.

List

- List stores the order of elements in which they are entered.
- List will allow duplicate values.
- Accessing elements by index is possible in lists.
- Lists allow null elements to be stored.

using comparator class to sort an array:

Comparator is an interface, that is useful to impose a total ordering on a collection of elements. it can be used to sort the elements of an array into ascending order or descending order.

→ To compare two objects we use `compareTo()` method.

// WAP to demonstrating how to sort objects into ascending & descending order using comparator.

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

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

```
Class Ascend implements Comparator<Integer>
```

```
{ public int compare (Integer i1, Integer i2)
```

```
{ return i1.compareTo(i2);
```

```
}
```

```
}
```

```
Class Descend implements Comparator<Integer>
```

```
{ public int compare (Integer i1, Integer i2)
```

```
{ return i2.compareTo(i1);
```

```
}
```

```
}
```

```
Class Arrays
```

```
{
```

```
    p.s.v.m() throws IOException
```

```
{
```

```
    BufferedReader br = new BufferedReader (new  
        InputStreamReader(System.in));
```

```
    S.O.P(" how many elements ");
```

```
    int size = Integer.parseInt (br.readLine());
```

```
    Integer a[] = new Integer[size]
```



```

for(int i=0; i < size; i++)
{
    S.o.p("Enter int: ");
    a[i] = Integer.parseInt(br.readLine());
}
Arrays.sort(a, new Ascend()); // sort array in ascending order
S.o.p("sorted ascending order: ");
display(a); // To display sorted array
Arrays.sort(a, new Descend());
S.o.p("sorted descending order: ");
display(a);
}
static void display(Integer a[])
{
    for(Integer i: a)
        S.o.p(i + " ");
}
}

```

O/P: How many elements: 4

Enter int: 10

Enter int: 90

Enter int: 2

Enter int: 30

sorted in ascending order:

2 10 30 90

sorted in descending order:

90 30 10 2

Collection Algorithms:

The collection 's' and arrays classes, available as a part of the collection framework, support various algorithms. it mainly support the sorting & search Alg.

Sorting Alg: The sort algorithm reorders a List such that its elements are in ascending order according to an ordering relationship. TreeSet & TreeMap sort the elements according to sort() method.

// To print the arguments of a List in an alphabetical order

```
import java.util.*;  
public class SortDemo  
{  
    p.s.v.m()  
    {  
        List<String> l1 = Arrays.asList(args);  
        Collections.sort(l1);  
        s.o.p(l1);  
    }  
}
```

o/p: > java SortDemo this is a cmdline arguments
[a, arguments, cmdline, is, this]

Searching Algorithm:

Before searching an element, the List must be sorted, to sort the list we use collection.sort() method.

// print sorted arguments list & then search the position of a specified key value

```
import java.util.*;  
public class Search  
{  
    p.s.v.m() {
```


try {

List<String> l1 = Arrays.asList(args);

Collections.sort(l1);

S.o.p("The sorted list is " + l1);

int pos = Collections.binarySearch(l1, l1.get(2));

S.o.p("The position of the searched element is " + pos + " and the element is " + l1.get(2));

}

catch (Exception e)

{

}

}

o/p: > java Search this is cmdline args

The sorted list is: [a, args, cmdline, is, this]

The position of the searched element is: 2 and the element is cmdline.

Arrays class:

Arrays class provides methods to perform certain operations on any one dimensional array. All the methods of the Arrays class are static, so it can be called `Arrays.methodname()`.

Methods in Arrays class:

- static void `sort(array)`: to sort elements in ascending order
- static void `sort(array, int start, int end)`: to sort the elements with in range
- static int `binarySearch(array, element)`: it search an element & returns its position. if element not found it return negative value.

// Write a program for sorting & searching an array

```
import java.io.*;
import java.util.*;
class ArraysDemo
{
    p.s.v.m() throws IOException
    {
        BufferedReader br = new BufferedReader(new InputStreamReader(
            System.in));

        int a[] = new int[5];
        for(int i=0; i<5; i++)
        {
            s.o.p("Enter an integer");
            a[i] = Integer.parseInt(br.readLine());
        }
        s.o.p("Contents of the array");
        display(a);
        Arrays.sort(a); // to sort element
        s.o.p("The sorted array");
        display(a); // to display sorted elements
    }
}
```



```

s.o.p("Which element to search ");
int element = Integer.parseInt(br.readLine());
int index = Arrays.binarySearch(a, element);
if (index > 0)
    s.o.p("Element not found");
else
    s.o.p("Element found at " + (index + 1));
}

```

```

}
static void display(int a[])
{
    for (int i = 0; i < a.length; i++)
        s.o.p(a[i]);
}
}

```

o/p: Enter an integer : 10
 " " : 5
 " " : 20
 " " : 50
 " " : 40

Contents of the array:

10
 5
 20
 50
 40

The sorted array:

5
 10
 20
 40
 50

Which element to search: 20
 Element found at 3

Legacy classes and interfaces:

Early version of java did not include the collections framework. it only defined several classes & interfaces that provide methods for storing objects. when collection framework was added in J2SE 1.2, the original classes were reengineered to support the collection interface. these classes are known as Legacy class.

Legacy classes are redesigned in jdk 5 to support Generics.

java.util package support the following Legacy classes.

1. Dictionary
2. HashTable
3. properties
4. stack.
5. vector.

All legacy classes are synchronized.

Stack class:

A stack represents a group of elements stored in LIFO order. This means that the element which is stored as a last element into the stack will be the first element to be removed from the stack.

Inserting elements into the stack is called push operation and removing elements from stack is called pop operation. Searching for an element in the stack is called 'peek' operation. Insertion and deletion of elements take place only from one side of the stack, called 'top of the stack'.

Eg: A pile of plates, A group of CDs.

Methods :

element peek(): it returns the top-most object from the stack without removing it.

element pop(): it pops the top-most element from the stack and return it.

element push(element obj): it pushes an element obj onto the top of the stack & returns that element.

boolean empty(): it checks whether stack is empty or not. if stack is empty it return T otherwise F.

int search(Object obj): it returns the position of an element obj from the top of the stack. if obj not found it returns -1.

// WAP for stack operations

import java.io.*;

import java.util.*;

class Stackmain

{
p.s.v.m (String args[]) throws Exception

{

// Create an empty stack to contain integer objects

Stack<Integer> s = new Stack<Integer> ();

int choice = 0;

int pos, ele;

BufferedReader br = new BufferedReader (new InputStreamReader
(System.in));

while (choice < 4)

{
s.o.p ("Stack operations" /n 1. push /n 2. pop
/n 3. search /n 4. exit ");

s.o.p ("Enter your choice: ");

choice = Integer.parseInt (br.readLine());

switch (choice)

{

Case 1: s.o.p ("enter element: ");

ele = Integer.parseInt (br.readLine());

s.push (element);

break;

Case 2:

Integer obj = s.pop(); // top most int obj is
popped.

s.o.p ("popped = " + obj);

break;

Case 3:

s.o.p ("which element to search");

ele = Integer.parseInt (br.readLine());

pos = s.search(ele);

if (pos == -1)

s.o.p("Element not found");

else

s.o.p("position : " + pos);

break;

default: return;

}

s.o.p("stack contents : " + s); // To view contents of stack

}

}

}

O/P: Stack operations

1. push

2. pop

3. search

4. exit

Enter your choice: 1

Enter element : 10

Stack contents : [10]

Stack operations

1. push

2. pop

3. search

4. exit

Enter your choice : 2

Stack contents:

Stack operations

1. push

2. pop

3. search

4. exit

Enter your choice : 4

Vector class:

A vector also stores elements (objects) similar to ArrayList, vector implements a dynamic array.

but the difference is vector is synchronized. vector contains many legacy methods that are not part of the collections framework.

We can write a vector class as

```
class Vector < E ?  
    ↳ Type of elements to be stored  
Vector < Integer > v = new Vector < Integer > (101);  
                                ↳ capacity of vector
```

```
Vector < Integer > v = new Vector < Integer > (101, 20);  
                                ↳ capacity      ↳ Increment value
```

Here, 101 is the capacity of vector and it can be increment by 20 i.e., 121.

Methods:

boolean add(element obj): append element at end of vector
void add(int position, element obj): it insert the specified element at specified position.

element remove(int position): it removes specified element.
boolean remove(Object obj): it removes the 1st occurrence of the specified element obj from vector

void clear(): To remove all elements from vector.

element set(int position, element obj): it replaces an element at the specified position in the vector with the specified element obj.

int capacity(): it returns the current capacity of vector.

// WAP that shows the use of vector class.

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

```
class VectorDemo
```

```
{
```

```
    p.s.v.m( )
```

```
    {
```

```
        Vector<Integer> v = new Vector<Integer>();
```

```
        int x[] = {10, 20, 50, 15, 22, 60}
```

```
        for (int i = 0; i < x.length; i++)
```

```
        {
```

```
            v.add(x[i]);
```

```
        }
```

```
        s.o.p(" retrieve the elements using get() : ").
```

```
        for (int i = 0; i < v.size(); i++)
```

```
        {
```

```
            s.o.p(v.get(i));
```

```
        }
```

```
        s.o.p(" Elements using List Iterator : ").
```

```
        ListIterator li = v.listIterator();
```

```
        s.o.p(" In forward direction ").
```

```
        while (li.hasNext())
```

```
            s.o.p(li.next() + " \t ");
```

```
        s.o.p(" In backward direction ").
```

```
        while (li.hasPrevious())
```

```
            s.o.p(li.previous() + " \t ");
```

```
    }
```

o/p: ^{3 retrieve} ^{using get()} vector elements: 10 20 50 15 22 60

Elements using List Iterator:

In forward direction: 10 20 50 15 22 60

In backward direction: 60 22 15 50 20 10

Hashtable class:

Hashtable is similar to HashMap which can store elements in the form of key-value pairs. But hashtable is synchronized assuring proper results even if multiple threads act on it simultaneously.

Syntax: `class Hashtable < K, V >`

`Hashtable < String, Integer > h = new Hashtable < String, Integer > ();`

`Hashtable < String, Integer > h = new Hashtable < String, Integer > (11, 0.75);`

initial capacity ↘ ↙ load factor

Methods:

value put(key, value): it stores key-value pair into the hashtable.

value get(Object key): it returns corresponding value when key is given, if key not found it return null.

void clear():

boolean isEmpty(): it return true no key-value pair are not found.

value remove(Object key): it removes the key and corresponding value from the hashtable.

// WAP for hashtable class

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

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

```
class HashtableDemo
```

```
{  
    p.s.v.m() throws IOException
```

```
{
```

```
    Hashtable<String, Integer> ht = new Hashtable<String, Integer>
```

```
    ht.put("java", 100);
```

```
    ht.put("os", 90);
```

```
    ht.put("cpp", 70);
```

```
    ht.put("cp", 80);
```

```
    s.o.p("subject names: ");
```

```
    Enumeration e = ht.keys();
```

```
    while (e.hasMoreElements())
```

```
    {  
        s.o.p(e.nextElement());
```

```
        BufferedReader br = new BufferedReader(new Input  
            StreamReader(System.in));
```

```
        s.o.p("Enter sub name: ");
```

```
        String name = br.readLine();
```

```
        name = name.trim(); // remove unnecessary spaces
```

```
        Integer score = ht.get(name);
```

```
        if (score != null)
```

```
        {
```

```
            int sc = score.intValue();
```

```
            s.o.p(name + " scored " + sc);
```

```
        }
```

```
        else
```

```
        {  
            s.o.p("sub not found");
```

```
        }
```

o/p: subject names: java, os, cpp, cp

Enter sub name: java

java scored: 100

Dictionary class:

it store the values in the form of key-value. it is subclass of hashtable class.

methods:

put(k, v): to add key & value.

elements(): returns value representation in dictionary.

get(Object key): get the mapped value.

isEmpty():

keys(): return key representation in dictio.

remove(Object key):

Size():

// program

```
import java.util. Dictionary;
```

```
public class Demo
```

```
{
```

```
    p.s.v.mc)
```

```
{
```

```
    Dictionary d1 = new Hashtable();
```

```
    d1.put("123", "code");
```

```
    d1.put("456", "prog");
```

```
    for(Enumeration i = d1.elements(); i.hasMoreElements();)
```

```
    {
```

```
        s.o.p(i.nextElement());
```


}

s.o.p ("key = 6" + d.get("6"))

s.o.p ("key = 456" + d1.get("123"))

s.o.p ("remove" + d1.remove("123"))

s.o.p ("size" + d1.size())

}

}

o/p:

code

prog

key = 6 : null

key = 456 : code

remove : code

size : 1

Properties class:

it contains key and value pair both as a string. it is a subclass of properties class.

The main advantage is the "recompilation is not required, if information is changed from properties file".

methods:

load(Reader r): Loads data from the Reader object.

getProperty(string key): return value based on key.

setProperty(String key, String value): sets the property in the properties object.

// Eg:

```
import java.util. properties;
```

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

```
public class Test
```

```
{
```

```
    p.s.v.m() throws Exception
```

```
{
```

```
    FileReader r = new FileReader("db.properties");
```



```
properties P = new properties();
```

```
P.load(x);
```

```
S.o.P(P, getProperty("user"));
```

```
S.o.P(P, getProperty("pwd"));
```

```
}
```

```
}
```

O/P: system
oracle

Note: in above program it displays the db properties i.e, database username & pwd. Suppose if you add the below stmt no need to recompile the program

```
P.setProperty("name", "padma");
```

```
P.setProperty("email", "p@gmail.com");
```

O/P: name = padma

email = p@gmail.com.

HashMap

- HashMap object is not synchronized by default
- In case of single thread, using hashmap is faster than the hashtable.
- it store null keys & null values
- Iterator interface is preferable

Hashtable

- Hashtable object is synchronized by default.
- in case of multiple threads hashtable is faster than the hashmap.
- it doesn't store null keys or values.
- Enumeration is preferable.

Set

- it represents collection of elements order of the elements may change in the set.
- it not allow duplicate values to be stored.
- Accessing elements by their index is not possible in sets.
- it not allow null elements

List

- it represents ordered collection of elements - it store elements which they are entered.

it allow duplicate values.

Accessing elements by index is possible in lists.

it allow null elements

ArrayList

- ArrayList object is not synchronized by default.
- In case of single thread, it is faster than the vector
- it increases ArrayList size every time by 50 percent.

vector

vector object is synchronized by default

In case of multiple threads it is faster than ArrayList.

vector increases its size every time by doubling it.

Utility classes:

StringTokenizer class:

it is useful to break a string into pieces, called 'tokens'. These tokens are then stored in the StringTokenizer object from where they can be retrieved

Syntax:

StringTokenizer st = new StringTokenizer(str, "delimiter");

actual string, it broken into pieces
which place the string broken in to pieces.

eg:- StringTokenizer st = new StringTokenizer(str, ",");

StringTokenizer st = " " (str, ":");

it can brake the string
whenever : find.

Methods:

int countTokens(): it count & return the number of tokens available in StringTokenizer object.

boolean hasMoreTokens(): it test if there are more tokens available in the StringTokenizer object or not. if next token is there then it returns true.

String nextToken(): it return next token from the StringTokenizer.

//

// WAP for String Tokenizer class.

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

```
class STDemo
```

```
{  
    p.s.v.m()
```

```
{  
    String str = "He is a gentle man";
```

```
StringTokenizer st = new StringTokenizer(str, " ");
```

// break into tokens at spaces, here delimiter is a space

```
s.o.p("The tokens are:");
```

```
while(st.hasMoreTokens())
```

```
{  
    String one = st.nextToken();
```

```
    s.o.p(one);
```

```
}
```

O/P: The tokens are:

He

is

a

gentle

man

Calendar class:

Calendar class is useful in two ways

- it helps in knowing the system date and time
- it helps in storing a date and time value so that it can be transported to some other application.

Syntax: To create calendar class object

```
calendar cl = calendar.getInstance();
```

↳ it stores current system date & time.

Methods:

int get(int field): it returns the value of the given calendar field.

calendar.DATE - gives the date numbers from 1 to 31

calendar.MONTH - it gives month no' from 0 to 11

(January is taken as 0).

calendar.YEAR - gives year number.

calendar.HOUR - gives the hour no' from 0 to 11

calendar.MINUTE, calendar.SECOND, calendar.AM-PM

Void set(int field, int value): sets the given field in calendar object to the given value.

eg:- cl.set(calendar.DATE, 15);

cl.set(calendar.MONTH, 2);

cl.set(calendar.YEAR, 2018);

11) WAP that shows the use of calendar class.

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

```
class calendarDemo
```

```
{
```

```
    p.s.v.m.c()
```

```
    {
```

```
        Calendar cl = Calendar.getInstance();
```

```
        S.O.P("current date: ");
```

```
        int dd = cl.get(Calendar.DATE);
```

```
        int mm = cl.get(Calendar.MONTH);
```

```
        int yy = cl.get(Calendar.YEAR);
```

```
        S.O.P(dd + " " + mm + " " + yy);
```

```
        S.O.P("current time");
```

```
        int h = cl.get(Calendar.HOUR);
```

```
        int m = cl.get(Calendar.MINUTE);
```

```
        int s = cl.get(Calendar.SECOND);
```

```
        S.O.P(h + " : " + m + " : " + s);
```

```
        int x = cl.get(Calendar.AM_PM);
```

```
        if(x == 0)
```

```
            S.O.P("Good Morning");
```

```
        else
```

```
            S.O.P("Good evening");
```

```
    }
```

```
}
```

o/p: current date: 20-10-2018

current time: 11: 9: 10

Good morning

Date class:

Date class is useful to display the date and time at a particular moment. When an object to date class is created, it contains the system date and time by default.

Syntax: To create object

Date d = new Date();

// WAP for date class

```
import java.util.*;
import java.text.*; // DateFormat class present in
                    // text package.

class MyDate
{
    p.s.v.m()
    {
        Date d = new Date();
        DateFormat d2 = DateFormat.getDateInstance(
            DateFormat.MEDIUM, DateFormat.SHORT,
            Locale.UK);
        String str = d2.format(d);
        s.o.p(str);
    }
}
```

O/P: 20-sep-2018 23:46

Bitset class: it creates an array of bits represented by boolean values

Bitset Constructors $\leftarrow$ Bitset()
Bitset(int no of bits)

methods:

set(): - set the bits at specified index to the specified value.

xor(): to perform logical xor operation on the bitsets

cardinality(): it is used to find the no of elements in bitset

clear(): To clear all elements.

length()

```
// program
import java.util.Bitset;
public class Demo
{
    p.s.v.m()
    {
        Bitset b1 = new Bitset();
        b1.set(7);
        b1.set(1);
        b1.set(2);
        Bitset b2 = new Bitset();
        b2.set(4);
        b2.set(2);
        S.O.P(b1 + b2);
        b1.xor(b2);
        S.O.P(b1);
    }
}
```


$b_2.\text{xor}(b_1);$

$\text{S.O.P}(\text{"use of xor()"} + b_2);$

$b_1.\text{clear}();$

$b_2.\text{clear}();$

$\text{S.O.P}(\text{"after clear"} + b_1 + b_2);$

O/P:

$b_1: \{1, 2, 7\}$

$b_2: \{2, 4\}$

$b_1.\text{xor}: \{1, 7, 4\}$

USE OF $\text{xor}()$: $\{1, 2, 7\}$

after clear: $\{ \}$

Random: To generate random numbers we are going to use Random class in java. it is going to generate different random datatypes such as float, double, int.

methods:

next Boolean()

next Byte()

nextDouble()

nextFloat()

nextInt()

nextInt(int bound)

nextLong()

// program

```
import java.util. Random;
```

```
public class Random1
```

```
{
```

```
    p.s.v.m()
```

```
{
```

```
    Random r = new Random();
```

```
    s.o.p(r.nextInt(10));
```

```
    s.o.p(r.nextBoolean());
```

```
    s.o.p(r.nextDouble());
```


Formatter:

formatter class o/p the formatted o/p.
it can format numbers, strings, and time & date.
it operates in a manner similar to the c/c++
printf() function.

method:

format(String fmtString, Object... args);

// Eg:

```
import java.util.Formatter;
```

```
public class Demo
```

```
{
```

```
    p.s.v.m ( )
```

```
{
```

```
    Formatter fm = new Formatter();
```

```
    fm.format (" %s gap java %d %f",
```

```
        "asString", 10, 12.3);
```

```
    s.o.p (fm.out());
```

```
}
```

```
}
```

o/p: asString gap java 10 12.300000

UNIT-V [Part-1]

GUI Programming with swing: Introduction, limitations of AWT, MVC Architecture, components, containers, understanding layout managers, Flow Layout, Border Layout, Card Layout, Grid Layout, Card Layout, Grid Bag Layout.

Introduction:

When ever an application is created, the user can work with it in two ways.

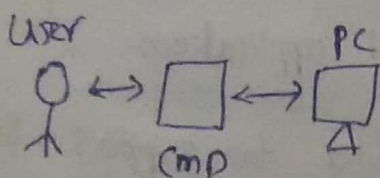
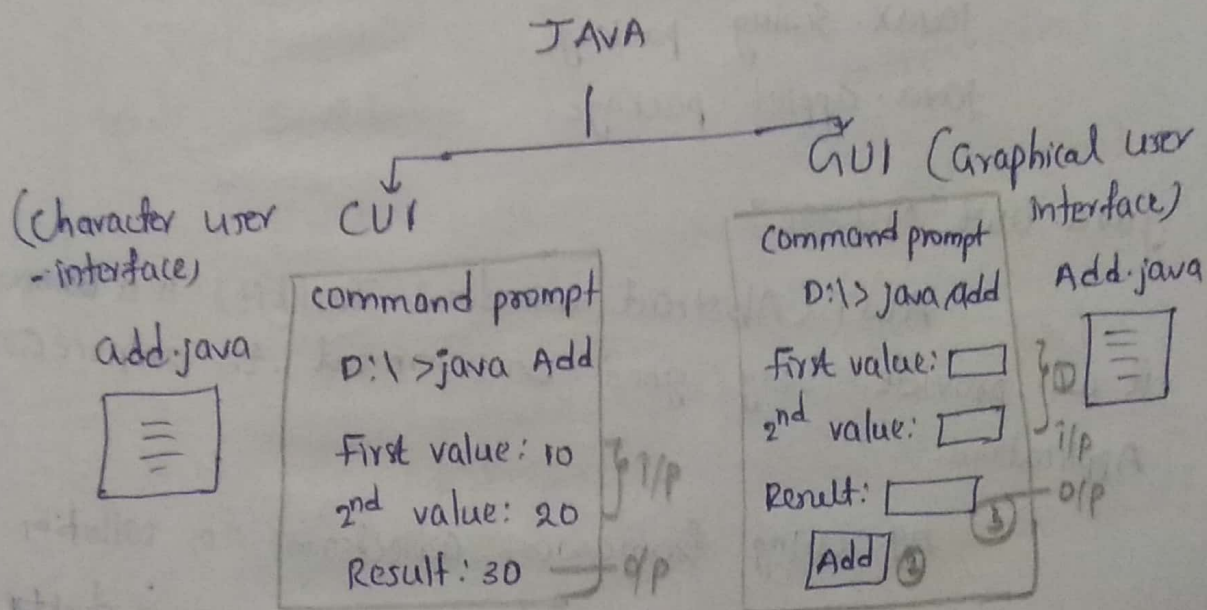
In first way user can remember some cmds on which the application is built and type those cmds to achieve the respective task. this is called CUI- (Character user Interface) since the user has to use characters or cmds to interact with the application.

The main disadvantage of CUI is the user has to remember several cmds and their use with correct syntax. Hence, CUI is not user-friendly. if a person doesn't know any thing about computers will find this CUI very difficult.

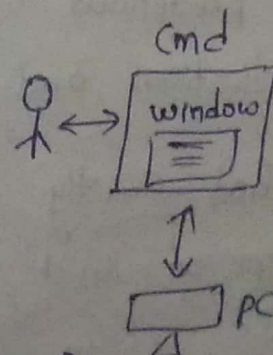
The 2nd way is where the user need not remember any cmds but interacts with any application by clicking on some images or graphics. This environment where the user can interact with an application through graphics or images is called GUI (graphics user Interface).

GUI Advantages:

- it is user-friendly. The user need not worry about any cmds. Even a layman will be able to work with the application developed using GUI.
- It adds attraction & beauty to any application by adding pictures, colors, menus, animations etc.
- It is possible to simulate the real life objects using GUI Eg:- calculator
- it helps to create graphical components like push button, radio button, check boxes etc.



In CUI, takes the i/p by using cmdprompt & gives the o/p on cmdprompt only. here cmdprompts has a interface between user & pc



In GUI, same runs it open new window that has some components. like labels, Textfield buttons. Here window act as an interface b/w user & system.

GUI application is a java application it is designed in such a way to take input through a collection of GUI components & o/p through a collection of GUI components.

Here the collection of GUI components act as user interface so, this supports only graphics data that's way we call it a GUI.

To prepare GUI applications, java has provided with 3 types of packages they are

java.awt package

javax.swing package

java.applet package.

java.awt package:

AWT (Abstract Window Toolkit) is a framework it will provide very good environment to prepare GUI Applications.

by using framework collections i.e, collection of predefined classes and interfaces to prepare desktop applications and window based applications.

What exactly AWT is providing to support for which Type of GUI components are supported by AWT.

The Type GUI components that are supported by AWT

1. Window/frame
2. menu
3. menu Item
4. TextField
5. passwordField
6. checkBox

7. Radio button
8. List
9. Choice Box
10. Text Area
11. Button
12. Label

Diagram illustrating a registration form layout with various UI components:

- Title Bar:** Title, - (minimize), O (maximize), X (close)
- Menu Bar:** File, Edit
- File Menu:** New, Save, Open
- Form Fields:**
 - User ID: (Text field)
 - Password: (passwordField) [it accept echo character]
 - Qualification: ☐ (checkbox) [multiple values]
 - Gender: male ☐ female ☐ (radiobutton) [single button]
 - Hobbies:

Cricket, music, books

 (List) [multiple values]
 - profession:

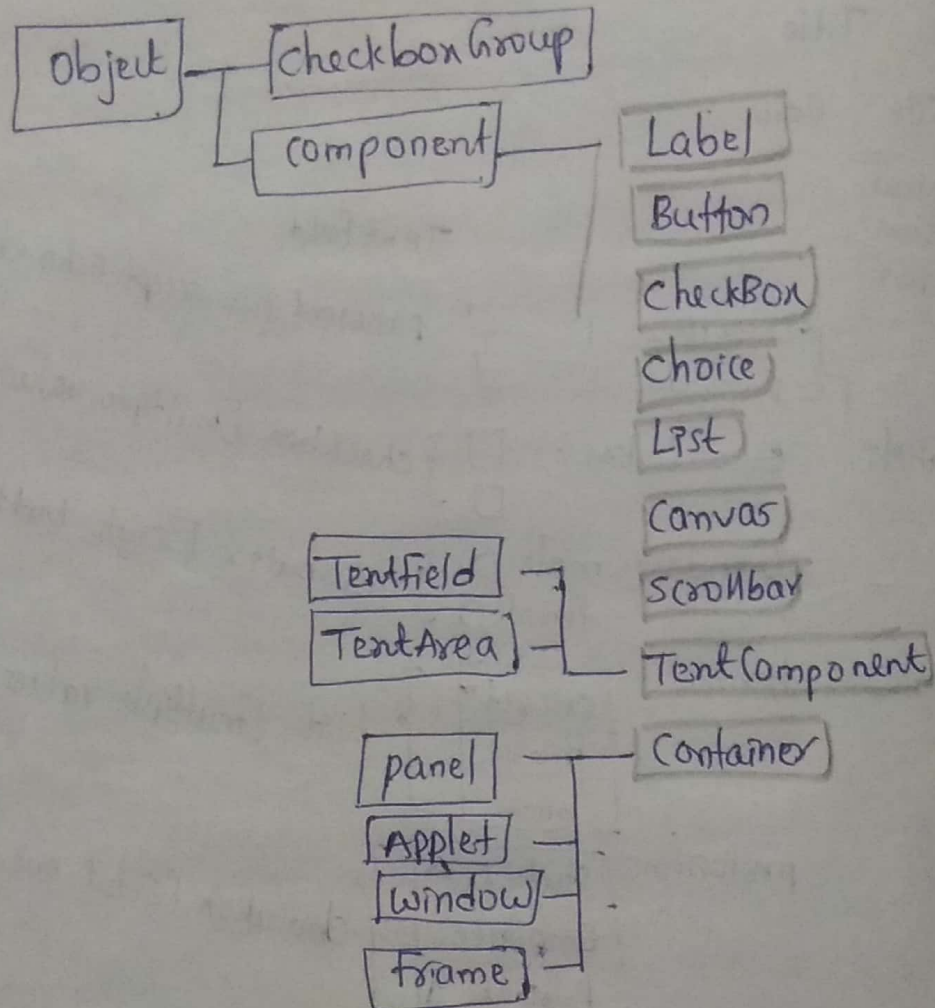
student, Employee, Business, Teacher

 (choiceBox) [select one choice]
 - User Address: (Text Area)
- Buttons:** Registration, Cancel (grouped as buttons)

Eg: Registration page

→ Given AWT supports only the basic GUI components it doesn't support some advanced components like calendar / Graphics.

Q: How these GUI components are represented, what exact predefined library was provided by AWT to represent these GUI components.



Classes of AWT

These are the predefined classes and interfaces which are supporting GUI components in our application.

container:

It is a GUI component its able to manage other GUI components.

eg:- frame, window, panel.

Window and frame:

A window represents any imaginary rectangular area on the screen without any borders or title bar.

A frame represents a single window with some title and borders. A frame become a basic component in AWT. the frame has to be created before any other components. the reason is all components can be displayed in a frame.

There are 3 ways to create frames

- `Frame f = new Frame();`
- `Frame f = new Frame("my frame");`
└ Title of frame
- `class MyFrame extends Frame`
`MyFrame f = new MyFrame();`

In all these cases, a frame with initial size of '0' pixels width & '0' pixels height will be created. then it is not visible on the screen. and by default frame is not visible to user. so we have to make it as visible to users by using `setVisible()` method.

`show()` method is deprecated method (outdated)

`public void show();`

`public void setVisible(boolean);`

if you provide `true` it set to visible mode
if you provide `false` it set to invisible mode

11 Write a program to create a frame.

```
import java.awt.Frame;
```

```
class myFrame
```

```
{
```

```
    p.s.v.mc()
```

```
}
```

```
Frame f = new Frame("my awt frame")
```

```
f.setSize(300, 250);
```

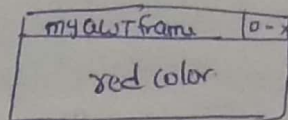
```
f.setVisible(true);
```

```
f.setBackground(Color.red);
```

```
}
```

```
}
```

O/p:



Note: To set the size for the frame `setSize` method is used. `setSize(int width, int height);`

`setBackground(Color.c)`

it is a class

it represent all colors like red, green...

if you want set title we use

`setTitle(String title);` method.

11 Write a program to create a frame [userdefined frame class object].

```
import java.awt.Frame;
```

```
class MyFrame extends Frame
```

```
{
```

```
    MyFrame(String str)
```

```
{
```

```
        super(str);
```



```
}  
p.s.v.m()
```

```
{
```

```
MyFrame f = new MyFrame("Frame")
```

```
f.setSize(300, 250)
```

```
f.setVisible(true)
```

```
f.setBackground(Color.red)
```

```
}  
}
```

→ if you want to provide any message on frame we must override paint() method.

```
public void paint(Graphics g)
```

```
{
```

```
Font f = new Font("arial", f.ITALIC + f.BOLD, 35)
```

```
g.setFont(f)
```

```
g.drawString("Hello", 100, 100)
```

```
this.setForeground(Color.red)
```

```
}
```

Layout Managers:

A layout manager is a class that is useful to arrange components in a particular manner in a frame or container.

We create several components like push buttons, checkboxes, radio buttons etc. in GUI after creating these components, they should be placed in the frame or container. to arrange these components in a particular order we are using layout managers.

Java.awt package provide various types of layouts to arrange the components.

1. Flow Layout
2. BorderLayout
3. CardLayout
4. GridLayout
5. GridBagLayout
6. BoxLayout

FlowLayout:

FlowLayout is useful to arrange the components in a line one after the other. When a line is filled with components, they are automatically placed in the next line. This is the default layout in applets & panels.

To create flowlayout we follow two ways

- i. `FlowLayout f1 = new FlowLayout();`
- ii. `FlowLayout f1 = new FlowLayout(int alignment);`

- i) by default, the gap between components will be 5 pixels and the components are centered in the 1st line.
- ii) the alignment of components can be specified. To arrange the components starting from left to right, we can use `FlowLayout.LEFT`. To adjust the components towards right, we can use `FlowLayout.RIGHT` & for center alignment, we can use `FlowLayout.CENTER`.

eg: `FlowLayout f1 = new FlowLayout(int alignment, int hgap, int vgap);`
 vgap - vertical gap in pixels
 hgap - horizontal gap } space b/w components.

// WAP to create a group of push buttons and arrange them in the container using `FlowLayout`. the buttons are right justified.

```

import java.awt.*; // FlowLayout
import javax.swing.*; // container, JButton, JFrame

class FlowLayoutDemo extends JFrame
{
    FlowLayoutDemo()
    {
        // create content pane
        Container c = getContentPane();
        FlowLayout f = new FlowLayout(FlowLayout.RIGHT,
                                     10, 10);
        // set the layout to content pane
        c.setLayout(f);
        // create 4 push buttons
        JButton b1, b2, b3, b4;
    }
}

```

b₁ = new JButton("Button1");

b₂ = new JButton("Button2");

b₃ = new JButton("Button3");

b₄ = new JButton("Button4");

c.add(b₁); // To add buttons to container

c.add(b₂);

c.add(b₃);

c.add(b₄);

}

P.S.V.m()

{

FlowLayoutDemo d₁ = new FlowLayoutDemo();

d₁.setSize(400, 400);

d₁.setTitle("FlowLayout");

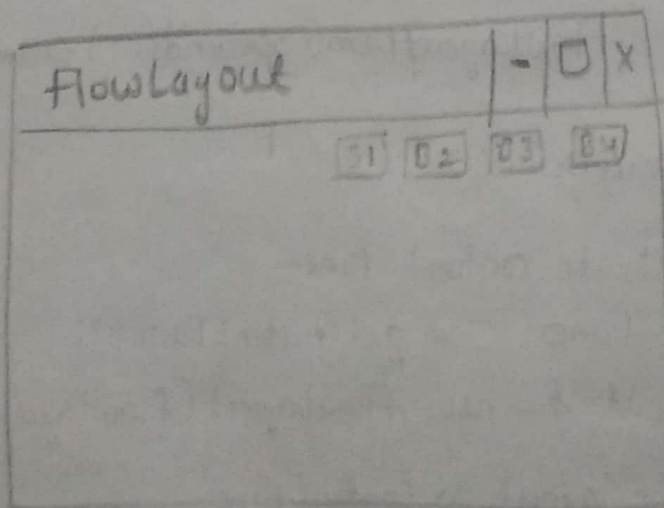
d₁.setVisible(true);

d₁.setDefaultCloseOperation(JFrame.EXIT-ON-CLOSE);

}

}

O/p:



BorderLayout:

It is useful to arrange the components in the 4 borders of the frame as well as in the center. The borders are identified with the names of directions. The top border is specified as 'North', the right side border as 'East', the bottom one as 'South' and left one as 'West'. The center is represented as 'center'.

BorderLayout obj = new BorderLayout() - with out any gaps bwn components.

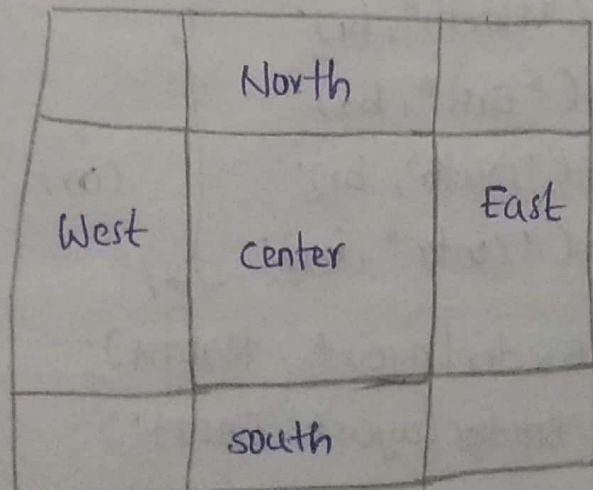
BorderLayout obj = new BorderLayout(int hgap, int vgap).
↙
gap between components.

c.add("North", component); // c - container.



to add container the direction should be specified.

c.add(component, BorderLayout.NORTH);



Direction of components in BorderLayout.

// Write a program to create a group of push buttons and add them to the container by using BorderLayout

```
import java.awt.*;
import javax.swing.*;

class BorderLayoutDemo extends JFrame
{
    BorderLayoutDemo()
    {
        Container c = getContentPane();
        BorderLayout b = new BorderLayout(10, 10);
        c.setLayout(b);

        JButton b1, b2, b3, b4;
        b1 = new JButton("Button1");
        b2 = "    " ("Button2");
        b3 = "    " ("Button3");
        b4 = "    " ("Button4");

        /*
        c.add("North", b1);
        c.add("East", b2);
        c.add("South", b3);
        c.add("Center", b4);
        */
        (Or)
        c.add(b1, BorderLayout.NORTH);
        c.add(b2, BorderLayout.EAST);
        "    (b3,    " "    SOUTH);
        "    (b4,    " "    CENTER);
    }
}

P.S.V.M ( )
{
    BorderLayoutDemo d1 = new BorderLayoutDemo();
}
```


d1. setSize(400,400);

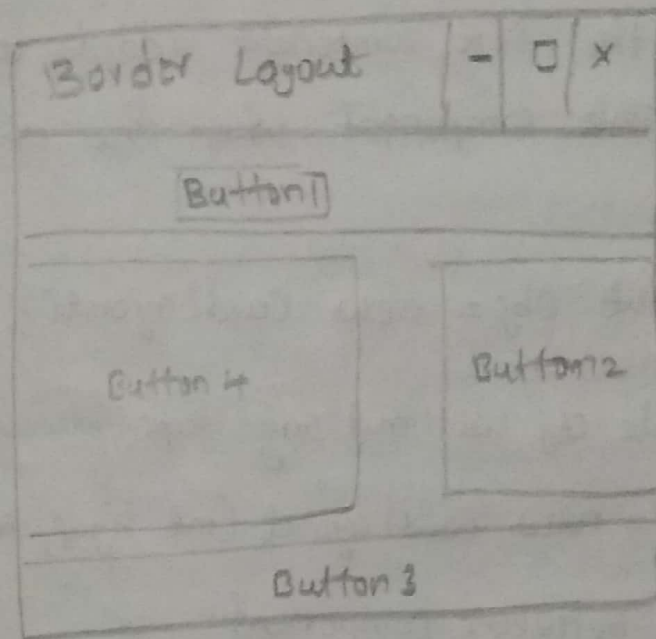
d1. setVisible(true);

d1. setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

3

3

O/P:



CardLayout:

A CardLayout object is a layout manager which treats each component as a card. Only one card is visible at a time & the container acts as stack of cards. the first component added to a CardLayout object is the visible component when the container is first displayed.

CardLayout obj = new CardLayout();

↓
it create obj without any gaps between ^{the} components

CardLayout obj = new CardLayout(int hgap, int vgap);

↓
gaps between components

→ To retrieve the cards one by one, the following methods can be used.

void first(container): to retrieve 1st card

void last(container): to retrieve the last card

void next(container): to go to the next card

void previous(container): to go to previous card

void show(container, "cardname"); to see a particular card with the name specified.

// WAP to create a group of push buttons & add them to the container using CardLayout.

```
import java.awt.*;
```

```
import java.awt.event.*;
```

```
import javax.swing.*;
```


class CardLayoutDemo extends JFrame implements

ActionListener

{

Container c;

CardLayout card;

JButton b1, b2, b3, b4;

CardLayoutDemo()

{

c = getContentPane();

card = new CardLayout(50, 10);

c.setLayout(card);

b1 = new JButton("Button 1");

b2 = new JButton("Button 2");

b3 = new JButton("Button 3");

b4 = new JButton("Button 4");

// add each button to 'c' on a separate card.

c.add("first card", b1);

c.add("2nd card", b2);

c.add("3rd card", b3);

c.add("4th card", b4);

// add action listeners to buttons

b1.addActionListener(this);

b2.addActionListener(this);

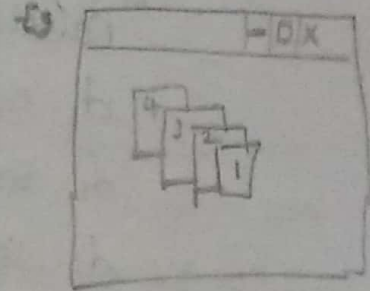
b3.addActionListener(this);

b4.addActionListener(this);

}

public void actionPerformed(ActionEvent ae)

{



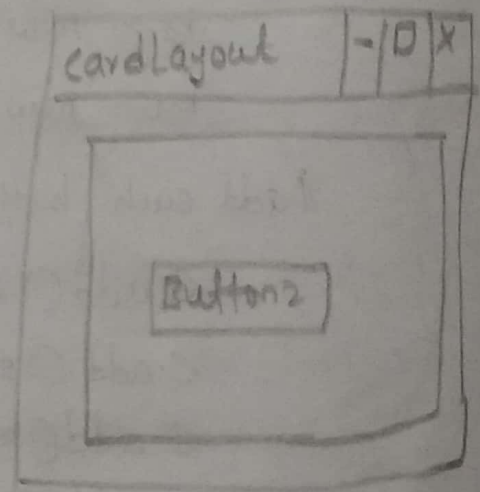
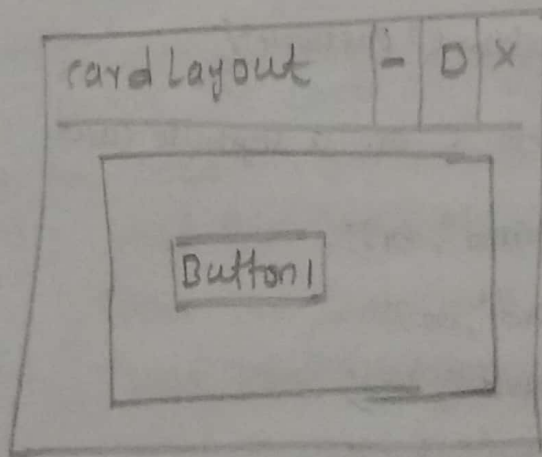
Card.next(C) // if button clicked it show next card.

// card.show(C, "ThirdCard") - To show particular card

}
p.s.v.m.c)

{
CardLayoutDemo d1 = new CardLayoutDemo();
d1.setSize(400, 400);
d1.setVisible(true);
d1.setTitle("CardLayout");
d1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

o/p:



if click button1 display ↗

GridLayout: it is useful to divide the container into a two-dimensional grid form that contains several rows & columns. The container is divided into equalized rectangles and one component is placed in each rectangle.

`GridLayout obj = new GridLayout();`

↓
it creates a default one column per component in a single row

`GridLayout obj = new GridLayout(int rows, int cols);`

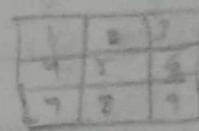
↓
it create specified no. of rows & cols.

`GridLayout obj = new GridLayout(int rows, int cols,`

↓ `int hgap, int vgap);`

gaps between components

// WAP to create 5 push buttons and add them to the content pane using BorderLayout. eg.



1	2	3
4	5	6
7	8	9

`import java.awt.*;`

`import javax.swing.*;`

`class GridLayoutDemo extends JFrame`

`{`
`GridLayoutDemo()`

`{`
`container c = getContentPane();`

`GridLayout g = new GridLayout(3, 3, 50, 50);`

`c.setLayout(g);`

JButton b1 = new JButton("Button1");

JButton b2 = new JButton("Button2");

JButton b3 = new JButton("Button3");

JButton b4 = new JButton("Button4");

JButton b5 = new JButton("Buttons");

c.add(b1);

c.add(b2);

c.add(b3);

c.add(b4);

c.add(b5);

}

p.s.v.m()

{

GridLayoutDemo g = new GridLayoutDemo();

g.setSize(500, 500);

g.setTitle("Grid");

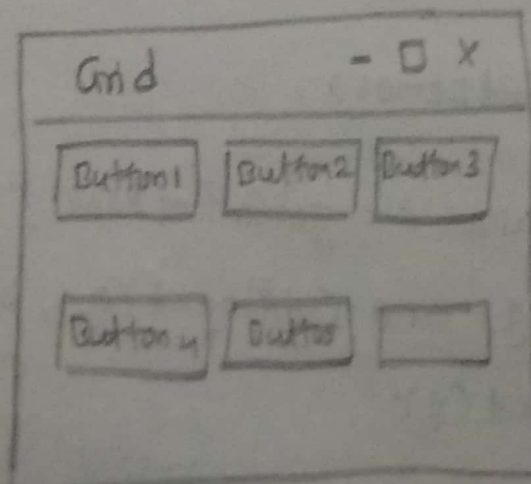
g.setVisible(true);

g.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

}

g

O/p:



GridBagLayout:

It represents gridbag layout manager where the components are arranged in rows and columns. This layout is more flexible as compared to other layouts since in this layout, "the components can span more than one row or column & the size of the components can be adjusted to fit the display area."

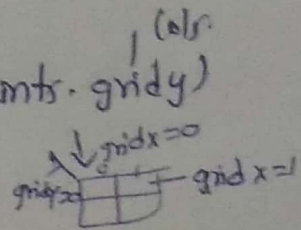
When positioning the components by using grid bag layout, it is necessary to apply some constraints or conditions on the components regarding their position, size & space in or around the components etc. such constraints are specified using GridBagConstraints class.

GridBagLayout obj = new GridBagLayout();

GridBagConstraints cns = new GridBagConstraints();

↓
methods:

- (GridBagConstraints.gridx, GridBagConstraints.gridy)



Eg:

		username
		pwd
		login

11

- GridBagConstraints gridWidth, GridBagConstraints gridHeight:
Specify no. of rows & cols.

- GridBagConstraints weightx, GridBagConstraints weighty:

To resize the frame

- GridBagConstraints.anchor: when the display area is larger than the component, anchor constraint will determine to place the component in the display area.

- GridBagConstraints.fill:

it is useful to resize the component according to the space available in the display area.

- GridBagConstraints.insets:

to leave some space around the component at the edges of component.

Event Handling

PART - 2

The Delegation event model - Events, Event sources, Event Listeners, Event classes, Handling mouse, & keyboard events, Adapter classes, Inner classes, Anonymous Inner classes.

Event Delegation Model:

When we create a component, generally the component is displayed on the screen but not capable of performing any action. A user wants the push button to perform some action. Hence user clicks on the button. Clicking like this is called event.

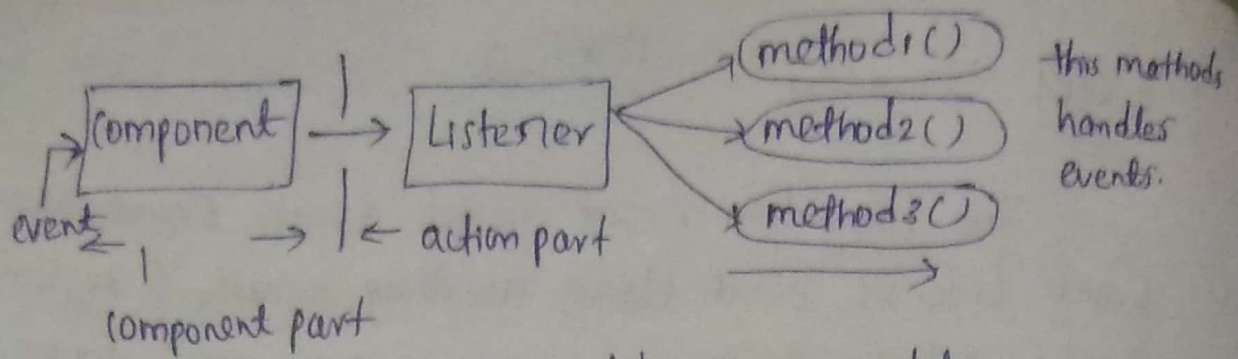
Event: An event represents a specific action done on a component.

Eg:- Clicking, double clicking, typing data inside the component etc.

→ An event is generated, the component will not know about it because it cannot listen to the event, so we should add some listener to the components.

→ A Listener is an interface which listens to an event coming from a component. A listener will have some abstract methods which need to be implemented by the programmer.

→ An event is generated by the user on a component. It is delegated (sends) to a listener interface and the listener calls a method in response to the event. Finally, the event is handled by the method. This is called event delegation model.



Event delegation model.

Following steps are involved in event delegation model:

- We should attach an appropriate listener to a component. This is done using `addXXXListener()` method. Similarly, to remove a listener from a component, we can use `removeXXXListener()` method.
- implement the methods of the listener, especially the method which handles the event.
- When an event is generated on the component, then the method in step 2 will be executed & the event is handled.
- The component & the action parts can be developed in two separate environments.

Listeners and Listener Methods:

For working with push buttons, `ActionListeners` is more suitable. All listeners are available in `java.awt.event` package.

component	Listener	Listener methods
1. Button	<code>ActionListener</code>	<code>public void actionPerformed (ActionEvent ae)</code>
2. check Box	<code>ItemListener</code>	<code>p.v itemStateChanged (ItemEvent ie)</code>

Component	Listener	Event Name	Listener Methods
3. CheckBoxGroup	ItemListener	ItemEvent	P.V. itemStateChanged (ItemEvent e)
4. Text Area	ActionListener	ActionEvent	P.V. actionPerformed (ActionEvent e)
	FocusListener	FocusEvent	P.V. focusGained (FocusEvent e) P.V. focusLost (FocusEvent e)
5. Text field	ActionListener	ActionEvent	P.V. actionPerformed (ActionEvent e)
	FocusListener	FocusEvent	P.V. focusGained (FocusEvent e) P.V. focusLost (FocusEvent e)
6. Choice	ActionListener	ActionEvent	P.V. actionPerformed (ActionEvent e)
	ItemListener	ItemEvent	P.V. itemStateChanged (ItemEvent e)
7. List	"	"	"
8. Scrollbar	AdjustmentListener	AdjustmentEvent	P.V. adjustmentValueChanged (AdjustmentEvent e)
	MouseMotionListener	MouseEvent	P.V. mouseDragged (MouseEvent e) P.V. mouseMoved (MouseEvent e)
9. Frame	WindowListener	WindowEvent	P.V. windowActivated (WindowEvent e) P.V. windowClosed (WindowEvent e) P.V. windowClosing (WindowEvent e) P.V. windowDeactivated (WindowEvent e) P.V. windowIconified (WindowEvent e) P.V. windowDeiconified (WindowEvent e) P.V. windowOpened (WindowEvent e)
10. Keyboard	KeyListener	KeyEvent	P.V. keyPressed (KeyEvent e) P.V. keyReleased (KeyEvent e) P.V. keyTyped (KeyEvent e)
11. Mouse	MouseListener	MouseEvent	P.V. mouseClicked (MouseEvent e) P.V. mousePressed (MouseEvent e) P.V. mouseReleased (MouseEvent e) P.V. mouseEntered (MouseEvent e) P.V. mouseExited (MouseEvent e)

// WAP for WindowListener

```
import java.awt.event.*;
```

```
class WindowListenerEx implements WindowListener
```

```
{  
    P.V. windowOpened(WindowEvent we)
```

```
{  
        S.O.P("Window opened");  
    }
```

```
    public void windowClosed(WindowEvent we)
```

```
{  
        S.O.P("Window close");
```

```
        // System.exit(0);  
    }
```

```
    public void windowClosing(WindowEvent we)
```

```
{  
        S.O.P("Window closing"); } System.exit(0);
```

```
    public void windowIconified(WindowEvent we)
```

```
{  
        S.O.P("Window Iconified");  
    }
```

```
    public void windowDeiconified(WindowEvent we)
```

```
{  
        S.O.P("Window Deiconified");  
    }
```

```
    public void windowActivated(WindowEvent we)
```

```
{  
        S.O.P("Window Activated");
```

```
    }  
    public void windowDeactivated(WindowEvent we)
```

```
{  
        S.O.P("Window Deactivated");  
    }
```

```
class MyFrame extends Frame
```

```
{  
    MyFrame()
```

```
{
```



```

this. setVisible (true);
this. setSize (500, 500);
this. setTitle ("Window Listener");
this. setBackground (Color. red);
this. addWindowListener (new WindowListenerEx());
}
}

```

```

class WindowEventsMain

```

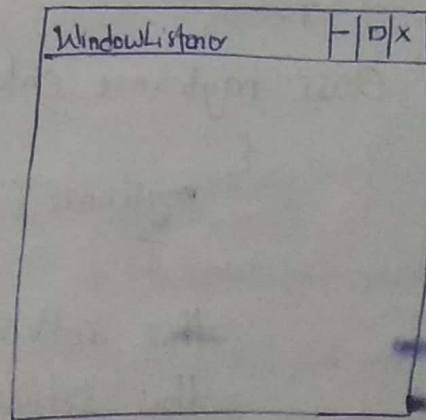
```

{
    p. s. v. m ( )
    {
        myFrame f = new myFrame();
    }
}

```

O/p:

Window Identified
 Window Deactivated
 Window Deiconified
 Window Activated
 Window Closing



Here we are ^{only creating} not frame but also close the frame

When the user click on the close button. for this purpose we use the WindowListener interface.

we had to mention all the methods of window Listener interface, just for the sake of one method i.e. windowClosing

→ There is another way to escape this. there is a class 'WindowAdapter' in java.awt.event package, that contains all the methods of the WindowListener interface with an empty implementation.

// Creating a frame & closing it. (an Adapter class eg)

```
import java.awt.*;
```

```
import java.awt.event.*;
```

```
class WindowListenerA extends WindowAdapter
```

```
{  
    public void windowClosing(WindowEvent we)
```

```
{  
    // S.O.P ("Window Closing");
```

```
    System.exit(0);
```

```
}
```

```
}
```

```
class myFrame extends Frame
```

```
{
```

```
    myFrame()
```

```
{
```

```
        this.setVisible(true);
```

```
        this.setSize(400,400);
```

```
        this.setTitle("Window Listener");
```

```
        this.setBackground(Color.green);
```

```
        this.addWindowListener(new WindowListenerA());
```

```
    }  
}
```

```
class WindowEventMam
```

```
{
```

```
    P.S.V.M()
```

```
{
```

```
        myFrame f = new myFrame();
```

```
    }
```

```
}
```


Adapter class:

An adapter class is an implementation class of listener which contains all methods implemented with empty body.

Ex: WindowAdapter is an adapter class of WindowListener interface. Adapter classes reduce overhead on programming while working with listener interfaces.

Anonymous inner class:

Anonymous inner class is an inner class whose name is not mentioned, and for which only one object is created.

In above Adapter class example program, that is not a correct approach, because to implement window closing method we are taking two classes i.e., WindowListener, WindowAdapter.

To avoid the using of two classes we are going to use anonymous inner class in above program.

1) Example for Anonymous inner class

```
import java.awt.*;
import java.awt.event.*;

class MyFrame extends Frame
{
    MyFrame()
    {
        this.setVisible(true);
        this.setSize(400,400);
        this.setTitle("WSL");
    }
}
```

```

        this.setBackground (color. pink);
    this.addWindowListener (new WindowAdapter ()
    {
        public void windowClosing (WindowEvent we)
        {
            System.exit(0);
        }
    });
}
}

```

```

class WindowEventsMain
{
    p.v.v.m()
    {
        MyFrame f1 = new MyFrame ();
    }
}

```

- upto now we are discussing ^{the} three ways to close the frame.
- By implementing all the methods of windowListener interface.
 - By using WindowAdapter class & by implementing only the required method.
 - By directly copying the code of Anonymous inner class.

push Buttons:

Button class is useful to create push buttons.
A push button is useful to perform a particular action.

```
Button b = new Button();
```

```
Button b = new Button("label"); // button with label
```

```
String s = b.getLabel(); // To get label of button
```

```
b.setLabel("label"); // To set label of button
```

CheckBox:

Check box is a square shaped box which displays an option to the user. the user can select one or more options from a group of check boxes.

```
checkbox c = new checkbox(); // with label.
```

```
checkbox c = new checkbox("label", state);
```

state is T then it is visible, if state is F checkbox will be not visible mode.

```
boolean b = c.getState(); // To get state
```

RadioButton:

it represent round shaped buttons, such that one can be selected from a group of buttons. Radio buttons can be created checkboxGroup class & checkbox class.

```
checkboxGroup cb = new checkboxGroup();
```

```
checkbox c = new checkbox("label", cb, state);
```

TextField: it represents a long rectangular box where the user can type a single line of text.

```
TextField tf = new TextField();
```

```
TextField tf = new TextField("default", 25);
```

To display default text & width is 25.

```
tf.setEchoChar('char');
```

```
tf.setText("text");
```

TextArea: it is similar to textfield, but it can accommodate several lines of text.

```
TextArea t = new TextArea();
```

```
TextArea t = new TextArea(rows, cols);
```

Label: To display constant text.

```
Label l = new Label();
```

```
Label l = new Label("text", alignment);
```

Choice: To display choice menu. it is a pop-up list of items & the user can select only one item from the available items.

```
Choice ch = new Choice();
```

```
ch.add("item");
```

List: To create list box. it represents the user with a scrolling list of text items.

```
List l = new List();
```

```
l.add("item");
```

Scrollbar: To create scrollbars that can be attached to a frame.

```
Scrollbar b = new Scrollbar(alignment, start, stop, mm, max);
```


Mouse Events:

// WAP for mouse-events.

```
import java.awt.*;
```

```
import java.awt.event.*;
```

```
class mouseEvents implements mouseListener
```

```
{  
    public void mouseClicked(mouseEvent m)
```

```
    {  
        s.o.p("mouse clicked [" + m.getX() + ", " + m.getY() + "]" );  
    }  
}
```

```
    public void mouseReleased(mouseEvent m)
```

```
    {  
        s.o.p("mouse released [" + m.getX() + ", " + m.getY() + "]" );  
    }  
}
```

```
    public void mousePressed(mouseEvent m)
```

```
    {  
        s.o.p("mouse pressed [" + m.getX() + ", " + m.getY() + "]" );  
    }  
}
```

```
    public void mouseEntered(mouseEvent m)
```

```
    {  
        s.o.p("mouse entered [" + m.getX() + ", " + m.getY() + "]" );  
    }  
}
```

```
    public void mouseExited(mouseEvent m)
```

```
    {  
        s.o.p("mouse exited [" + m.getX() + ", " + m.getY() + "]" );  
    }  
}
```

```
}  
class myFrame extends Frame
```

```
{  
    myFrame()
```

```
    {  
        this.setVisible(true);
```

```
        this.setSize(400, 400);
```

```
        this.setTitle("ME");
```

```
        this.addMouseListener(new mouseEvents());  
    }  
}
```

```
class MouseEventMain
```

```
{
```

```
    p.v.v.mc()
```

```
    {
```

```
        Myfram f = new myframe();
```

```
    }
```

```
}
```

O/p:

keyEvents

// WAP for keyEvents

```
import java.awt.*;  
import java.awt.event.*;  
class keyEvents implements KeyEvent  
{  
    public void keyPressed(KeyEvent k)  
    {  
        S.o.p("keyPressed" + k.getKeyChar());  
    }  
    public void keyReleased(KeyEvent k)  
    {  
        S.o.p("keyReleased" + k.getKeyChar());  
    }  
    public void keyTyped(KeyEvent k)  
    {  
        S.o.p("keyTyped" + k.getKeyChar());  
    }  
}
```

```
class myFrame extends Frame  
{  
    myFrame()  
    {  
        this.setVisible(true);  
        this.setSize(400,400);  
        this.setTitle("KE");  
        this.setBackground(color.red);  
        this.addKeyListener(new keyEvents());  
    }  
}
```

```
class KeyEventMain
```

```
{
```

```
    p.s.v.mc()
```

```
{
```

```
    myFrame f1 = new MyFrame()
```

```
}
```

```
}
```

o/p:

PART-3

A simple swing Application, Applets - Applets & HTML, security Issues, Applets & Applications, passing parameters to applet. Creating a swing Applet, painting in swing, A paint Example Exploring swing controls - JLabel & ImageIcon, JPasswordField, The Swing Buttons - JButton, JToggleButton, JCheckBox, JRadioButton, JToggleButton, JCheckBox, JTabbedPane, JScrollPane, JList JComboBox, swing menus, Dialogs.

Applet:

a html page wants to communicate with the user on Internet, it can use a special java program called applet to communicate and respond to the user.

Applet = java byte code + HTML page

"An applet represents java byte code embedded in a web page."

Creating an Applet:

To create an applet, we need to write a java program and compile it to get byte code. then we should embed it into a HTML page on a particular location whenever we want it to be displayed. this page is then stored in the webserver.

To create an applet, we have Applet class of java. applet package and JApplet class of javax.swing package.

These classes use the following methods, which are automatically run by any applet program.

public void init(): To initialize the variables, creating components & creating threads. & this method is executed by browser only once.

`Public void start();`

This method is called after `init()` method and each time the applet is revisited by the user.

`public void stop();`

it is called by the browser when the applet is to be stopped. if the user minimize the webpage, then this method is called & when the user again comes back to this page, then `start()` method is called. in this way, `start()` and `stop()` methods can be called repeatedly.

`public void destroy();`

this method is called when the applet is being terminated from memory. the `stop()` method will always be called before `destroy()`.

Note: `public static void main(String args[])` method is not available in case of applets. this means, we can compile the applet code but we can not run it using a JVM.

→ Applets are executed by a program called applet engine which is similar to virtual machine that exists inside the web browser at client side.

Uses of Applets:

→ Applets are used on Internet for creating dynamic web pages. there are two types of web pages static & dynamic. Static web page provide some information to the user but the user cannot interact with the web page other than viewing the information. dynamic web pages interact with the user at the runtime.

→ Another use of applets is for creating animation and games where the images can be displayed or moved giving a visual impression that they are alive.

A simple Applet:

→ `<Applet>` tag is used to insert an applet into html page

`<Applet code="classname.class" width="10" height="10"`

`Align="left, right, top, bottom, middle"`

`Alt="Alternate text to be displayed". >`

`<param name= parametername value= its values >`

`</applet>.`

→ To display msg we can take help of `paint()` method of `Component` class of `java.awt` package.

// A simple applet

```
import java.awt.*;
```

```
import java.applet.*;
```

```
public class myApplet extends Applet
```

```
{
```

```
// set a background color for the frame
```

```
public void init()
```

```
{
```

```
setBackground(Color.red);
```

```
}
```

```
// display msg in applet window
```

```
public void paint(Graphics g)
```

```
{
```

```
g.drawString("Hello", 50, 100);
```

```
}
```

```
}
```

→ save myApplet.java and compile the program

C:\> javac myApplet.java

→ next it generate the myApplet.class is created
this byte code should be embedded into a html
page using <Applet> tag.

→ <html>

<applet code = "myApplet.class" height = "100" width = "200">

</applet>

</html>

→ and save above program myApplet.html. this page
contains the applet which can be opened in the
browser.

→ open any browser & in the browser's address bar
type -html file name along with the directory path
or give the path in command prompt.

C:\> appletviewer myApplet.html

// Applet life cycle Example

```
import java.awt.*;
```

```
import java.applet.*;
```

```
public class Appl extends Applet
```

```
{
```

```
    String msg = " ";
```

```
    public void init()
```

```
{
```

```
        setBackground(Color.red);
```

```
        setForeground(Color.pink); // foreground for  
text in frame
```



```
// set font for text in applet  
Font f = new Font("Arial", Font.BOLD, 20);  
setFont(f);  
msg += "init"; // store method name in msg
```

```
}  
public void start()  
{  
    msg += "start"; // add this method name to msg  
}
```

```
}  
public void stop() // to stop applet.
```

```
{  
    msg += "stop";  
}
```

```
}  
public void destroy() // remove applet from memory
```

```
{  
    msg += "destroy";  
}
```

O/p: C:\> javac Appl.java

Now create the Appl.html file

C:\> appviewer Appl.html

Applet parameters:

Along with applet tag, we can use <param> tag to pass information to applets.

<param name="t1" value="AAA">

<param name="t2" value="15.0">

↓ ↓
name of parameter value of parameter

→ To receive the values of the parameters, an applet uses `getParameter()` method. This method takes the parameter name & returns its value as a string.

// program to pass name and salary to an applet
and get the tax calculated.

```
import java.awt.*;
```

```
import java.applet.*;
```

```
public class Tax extends Applet
```

```
{
```

```
String name, str;
```

```
float sal, tax;
```

```
public void init()
```

```
{
```

```
name = getParameter("t1"); // accept name from  
t1 parameter.
```

```
str = getParameter("t2");
```

```
sal = Float.parseFloat(str);
```

```
calculateTax(sal);
```

```
}
```

```
public void calculateTax(float sal)
```

```
{
```

```
if (sal <= 10000)
```

```
tax = 0.0f;
```

```
else if (sal <= 20000)
```

```
tax = sal * 0.1f;
```

```
else
```

```
tax = sal * 0.2f;
```

```
}
```

```
public void paint (Graphics g)
```

```
{
```

```
g.drawString("Hello" + name, 20, 100);
```

```
g.drawString("your sal" + sal, 20, 100);
```

```
g.drawString("pay the tax" + tax, 20, 140);
```

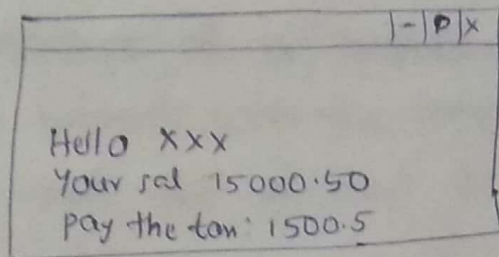
```
}
```

```
}
```


O/p: C:\> javac Tax.java

```
<html>  
<applet code="Tax.class" width=300 height=300>  
  <param name="t1" value="xxx">  
  <param name="t2" value="15000.50">  
</applet>  
</html>
```

C:\> appletviewer Tax.html



- When a component is created in AWT, it internally calls a native method (eg: c function) that creates the component internally. This component is called 'peer component'. This peer component is given back and displayed on the screen in AWT. This means AWT internally depends on code & this is not desirable as we know that 'c' is a system dependent language.
- The appearance of a component is called its 'look' & how the user interacts with the component is called its 'feel'. The look-and-feel of AWT components change depending on the platform.
- moreover, AWT components are heavy-weight it means these components take more system resources like more memory & more processor time.

All the classes of AWT are extended to form new classes and a new class library is created. This library is called JFC [java foundation classes].

Java foundation classes (JFC):

- JFC is an extension of the original AWT. It contains classes that are completely portable, since the entire JFC is developed in pure java.
- JFC components are light-weight. This means, they utilize minimum system resources.

- JFC components have same look & feel on all platforms. Once a component is created it looks same on any operating system.
- JFC offers 'pluggable look-and-feel' feature, which allows the programmer to change the look and feel as suited for a platform.
- JFC offers a rich set of components with lots of features.
- JFC does not replace AWT. JFC is an extension to AWT.

Packages of JFC:

`javax.swing`: this package contains push buttons, radio buttons, menus etc. the 'x' in `javax` represents that it is an 'extended package' whose classes are derived from `java.awt` package.

`javax.swing.plaf`: `plaf` stands for pluggable look and feel.

`java.awt.dnd`: `dnd` stands for drag & drop. this package helps to drag & drop the components such as text & data from one place to another place in the application.

`javax.accessibility`: it is used to develop app^s that physically challenged people are to use the Applications.

`java.awt.geom`: `geom` stands for geometrical shapes. & it helps to draw 2D graphs filling with colors, rotating the shapes etc.

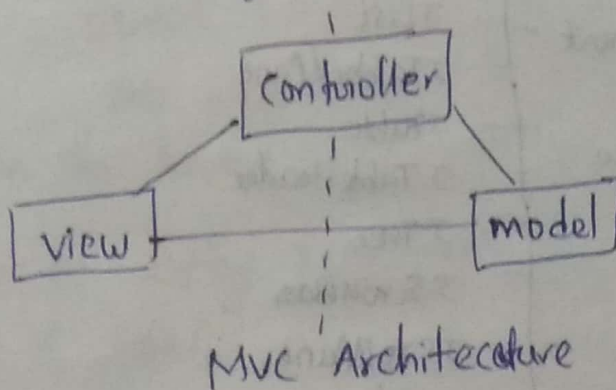
Java Swing & MVC:

Most popular package in JFC is swing package.

All the components in swing follow a model-view-controller (mvc) architecture.

Model represents the data that represents the state of a component. view represents visual appearance of the component based on the model.

MVC is a model used in swing components. model represents the data of the component. view represents its appearance and controller is a mediator between the model and the view. it represents the separation of model of an object from its view & how it is controlled.

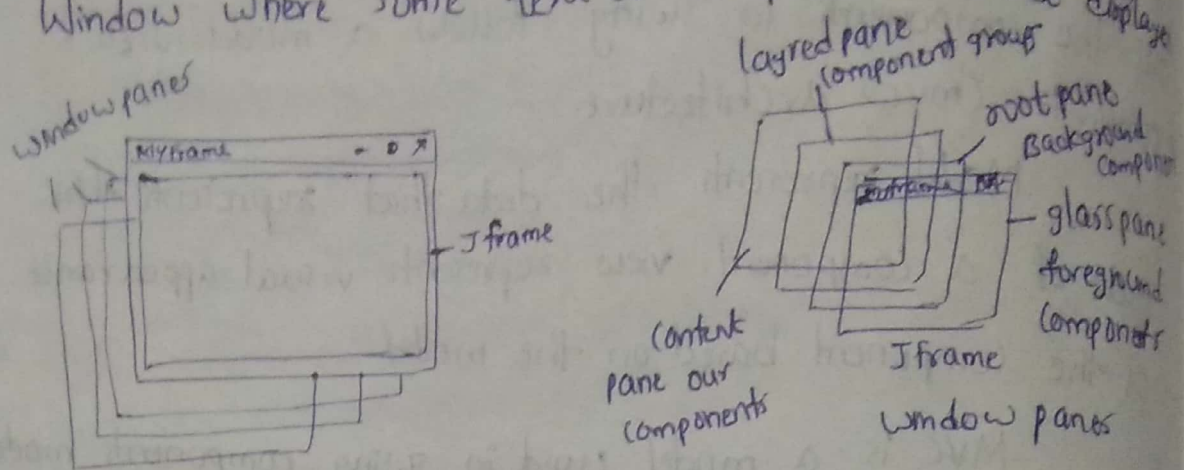


Advantages:

- Multiple views using the same model: the view & model are separated in mvc, it is possible to represent the same model data in several views.
- efficient modularity
- Easier support for new types of clients.

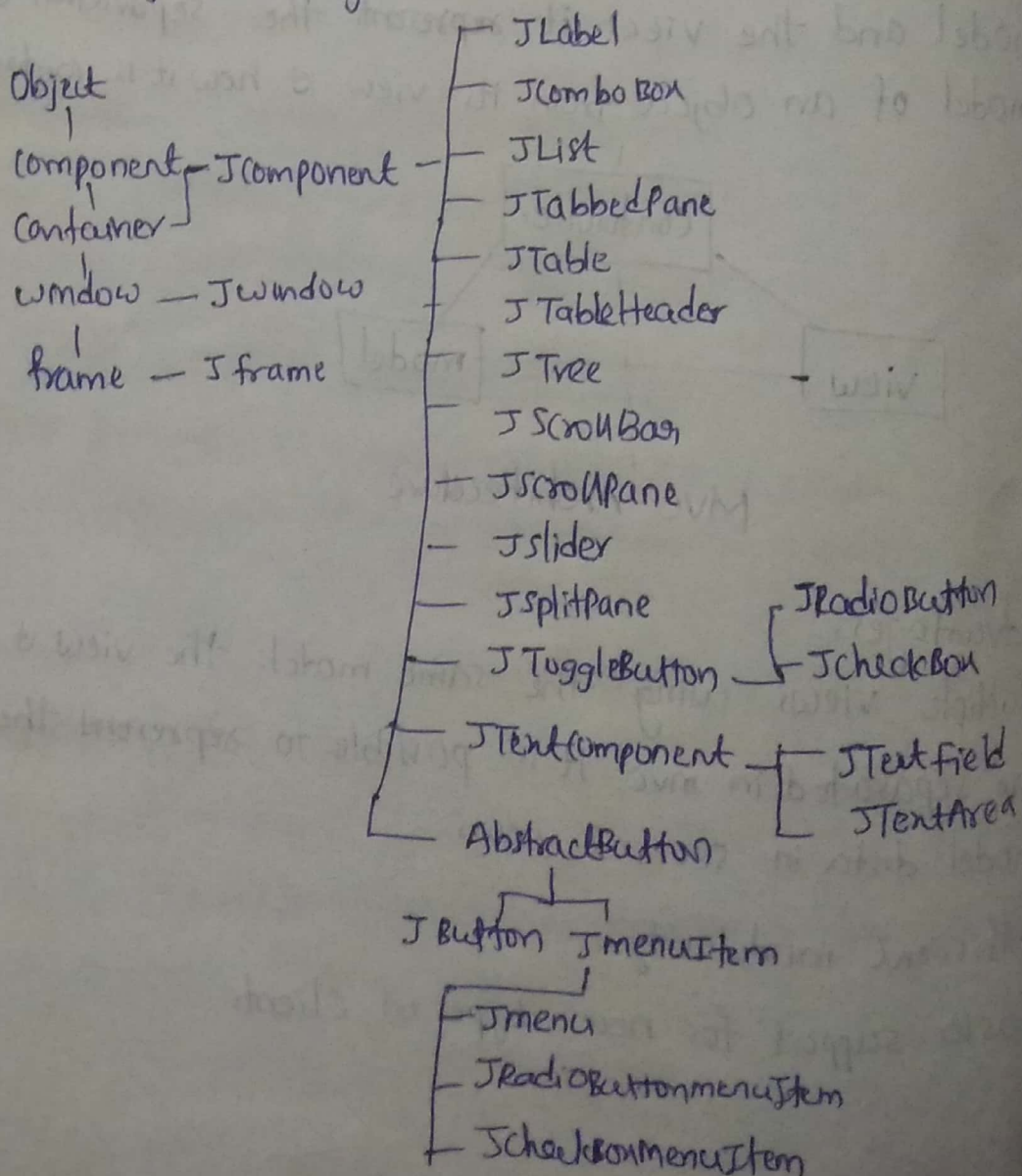
Window Panes:

A window pane represents a free area of a window where some text or components can be displayed.



Window panes are part of JFrame

Classes of java.swing:



creating a frame in swing:

A frame represent a window with a title bar and borders.

```
JFrame f = new JFrame();
```

```
JFrame f = new JFrame("title");
```

Creating Label:

To display text in an application swing package

follow the two ways.

→ paintComponent(Graphics g): JPanel class is used to paint the portion of a component in swing.

→ by using Label class.

```
JLabel l = new JLabel("text");
```

this label attach to contentpane to display the text in the frame.

// WAP to display some text in the frame with the help of label.

```
import javax.swing.*;
```

```
import java.awt.*;
```

```
class Framedemo extends JFrame
```

```
{
```

```
    JLabel l;
```

```
    Framedemo()
```

```
{
```

```
        Container c = this.getContentPane();
```

```
        c.setLayout(new FlowLayout());
```

```
        c.setBackground(Color.green);
```



```

l = new JLabel ("Hello students");
l.setFont (new Font ("Arial", Font.BOLD, 34));
l.setForeground (color.red);
c.add(l);
}
p.s.v.mc()
{
    JFrameDemo f = new JFrameDemo();
    f.setTitle ("myFrame");
    f.setVisible (true);
    f.setSize (400, 300);
    f.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
}
}

```

Displaying Image in swing:

To display image, 1st we create a empty

label.

```

JLabel l = new JLabel();
JLabel l = new JLabel ("text", ImageIcon(l));

```

Ex:- ImageIcon ll = new ImageIcon ("car2.gif");

it support gif, jpeg, png format.

Textfield:

```

JTextField tf = new JTextField();

```

```

JTextField tf = new JTextField ("text", 15);

```

width of 15 char

```

JTextField tf = new JTextField ("text", 5, 15);

```

5 row width of 15 char

// WAP for creating checkbox & radio button.

```
// import java.awt.*;
```

```
import javax.swing.*;
```

```
import java.awt.event.*;
```

```
class Checkbox extends JFrame implements ActionListener.
```

```
{
```

```
    JCheckBox cb1, cb2;
```

```
    JRadioButton rb1, rb2;
```

```
    JTextArea ta;
```

```
    ButtonGroup bg;
```

```
    String msg = "";
```

```
    Checkbox()
```

```
{
```

```
    Container c = getContentPane();
```

```
    c.setLayout(new FlowLayout());
```

```
    ta = new JTextArea(10, 20);
```

```
    cb1 = new JCheckBox("JAVAB", true);
```

```
    cb2 = new JCheckBox("J2EE");
```

```
    rb1 = new JRadioButton("M", true);
```

```
    rb2 = new JRadioButton("F");
```

```
    bg = new ButtonGroup();
```

```
    bg.add(rb1);
```

```
    bg.add(rb2);
```

```
    c.add(cb1);
```

```
    c.add(cb2);
```

```
    c.add(rb1);
```

```
    c.add(rb2);
```

```
    c.add(ta);
```


cb1.addActionListener(tho);

cb2. " " "

xb1. " " "

xb2. " " "

setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

3
p.v.actionPerformed(ActionEvent e)

{
if (cb1.getSelected().isSelected()) msg += "In JAVA";

if (cb2.getSelected().isSelected())

msg += "In J2EE";

if (xb1.getSelected().isSelected()) // which component
is selected by user.

msg += "In Male";

else

msg += "In female";

ta.setText(msg); // To display selected msg

msg = ""; // reset empty msg.

3

p.s.v.mc)

{

CheckBox c2 = new CheckBox();

c2.setTitle("bbb");

c2.setSize(200, 200);

c2.setVisible(true);

3

3

q.p.

