JAVA

__/__/__

1) JDK, JRE, JVM :-

JDK

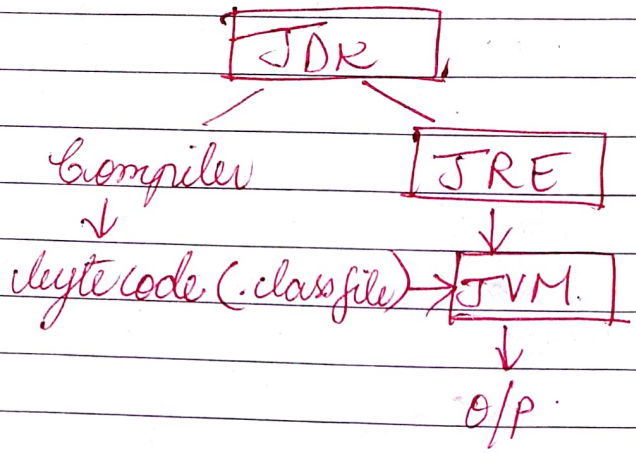
* Java Development Kit
* JDK provides an environment to compile, develop & execute Java apps
* JDK is a combo of Compiler & JRE
* JRE provides an environment to execute Java apps
* The major task of JVM is to execute the code line by line
* JVM contains platform dependent code
* The code for the JVM depends on OS
* The byte code will be given as input to JVM.
* Then the JVM produces the output
* That's why we can say that Java is interpreter based lang
* Java is an example for both compiler & interpreter based lang.

Also, Java is platform independent because once we get the byte code it can run on any OS

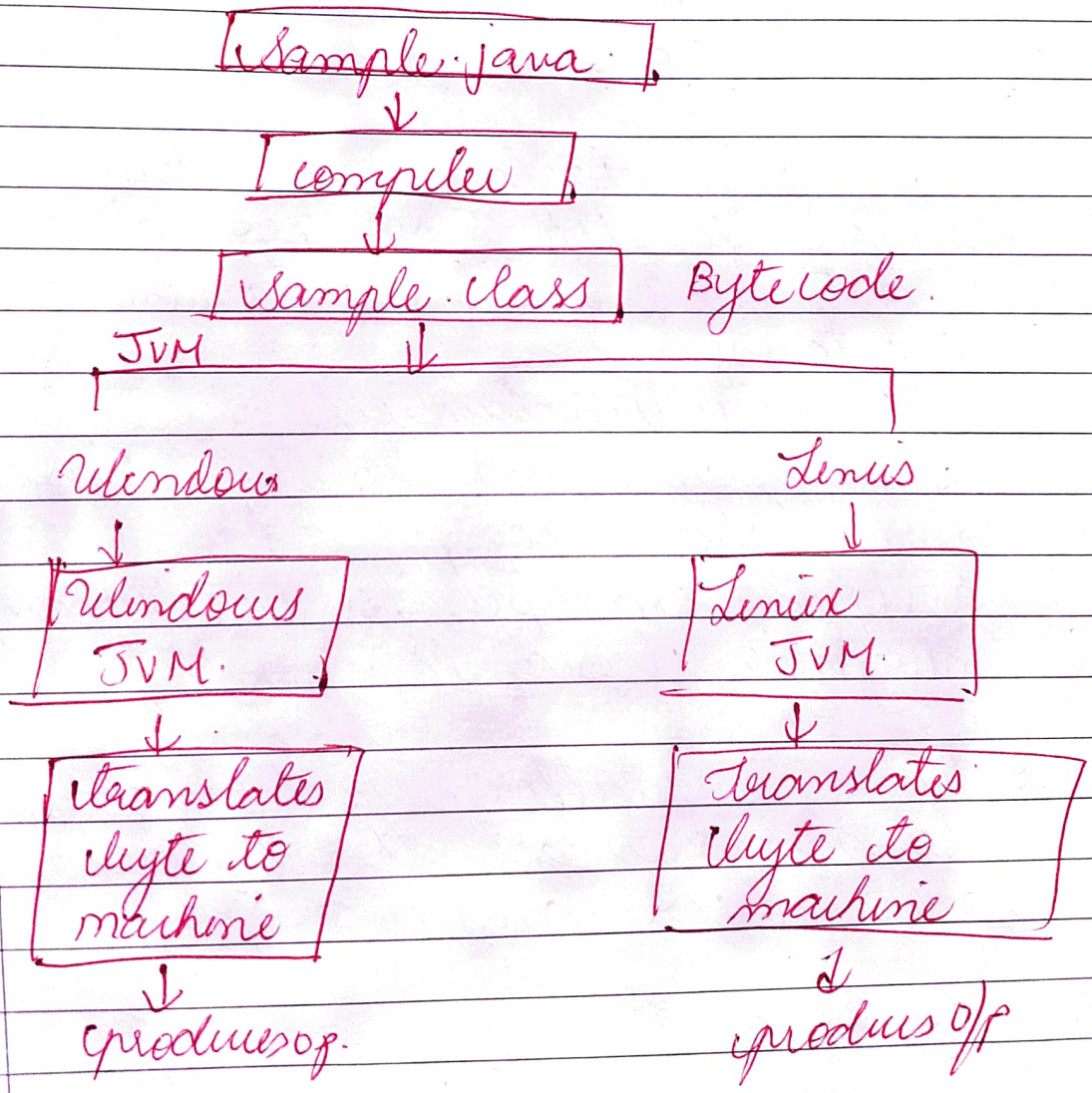
* JDK - Java Dev Kit
* JRE - Java Runtime Env
* JVM - Java Virtual Machine

__/__/____

i)


JDK
Compiler
JRE
byte code (.class file)
JVM.
O/P.

(ii)


Sample.java
compiler
Sample.class
Bytecode.
JVM
Windows
Linux
Windows JVM.
Linux JVM.
translates byte to machine
translates byte to machine
produces op.
produces o/p

JAVA FEATURES (OR) JAVA BUZZWORDS

(1) Simple

* It follows syntaxes of C as well as OOPS from C++
* It eliminates complexes from C like pointers && multiple inheritance && operator overloading from C++
* If the object is no longer used, garbage collector automatically removes it
* ∴ less memory && less execution

(2) Object Oriented

* Data is represented in the form of objects
* An object is a real world entity
* Java is partial OOP because it does not support operator overloading multiple inheritance && primitive data types such as Integer, Byte, Char

(3) Platform Independent

* Platform means O.S
* The Java compiler compiles our Java program to byte code, then we can execute this byte code on any O.S like ....

(4) Architecture Neutral

* We can develop Java on any hardware or on any system configurations
* Bcoz it works on the principle of

"write Once, run-anywhere, anytime & forever"

(5) portable
* We can move the Java program from one OS to the other
* portability means moving
* Here we move without any modifications

(6) Robust
* Robust means strong
* We can say that it is strong bcoz it is excellent in memory managment & exception handling

(7) Secure
* In JVM no have security management
* Also it is secure because of JAAS

(8) Dynamic
* Java compiler produces byte code; that byte code will be loaded into the memory during execution. If anything happens during runtime ie execution time then it is called Dynamic
* And for primitive data types, the memory will be allocated during runtime

(9) Multithreading
* Thread means task
* It executes multiple threads simultaneously in less amt of time

(10) Distributed :-

* It is distributed bcoz with Java we can develop network apps by using RmI & EJB & webservices.

(11) Interpretive :-

* Java is both compiled & Interpretive based lang
* JVM accepts first line of byte code & executes it, second " " " " " etc
* line by line execution

(12) High performance

* It is high performance bcoz of the above reasons

BASIC concepts of OOPS in Java.

(1) Class :-

* A class is a blueprint, from which object is created
* Class is a collection of variables

fields,
methods
}

* For classname we have to follow naming conventions (upper case lower)
Eg:- TotalMarks
Eg:- fields like :- int rollno;
double marks;
Eg:- methods like :- void total();
void avg();

* Ex:-

```
class Students
{
    int rollno;          } fields
    double mark          }
    void total();        } methods
    void avg();          }
}
```

② Objects

* Objects is an instance of a class.
* The memory will be allocated only once the Object is created
* Syntax :- classname obj = new classname();
* Example :- Students obj = new Students
* Here, the new operator allocates memory for the object
* With the help of objects we can

access variables and methods of a class
* These variables can be also called as properties of a class
* We use the . (dot) operator to access variables & methods
* Object is a real world entity
* Object is a physical entity

③ Abstraction :-
* Abstraction means representing essential features & hiding unimplementation
* Eg website :- we can work with website but dk which tech stack, the db used in building the website

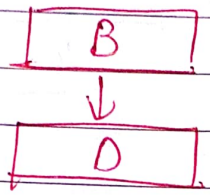
④ Encapsulation :-
* The process of wrapping (or) binding (or) combining variables & methods under a single class
* The major advantage of encapsulation is data hiding
* Data hiding is possible with the help of private access specifiers

⑤ Inheritance :-
* The process of creating a new class from old class
* Here the new class would be called as derived class, sub class or child class
* Old class is called parent class, Base class or super class
* Major adv of Inheritance is reusability

__/__/____

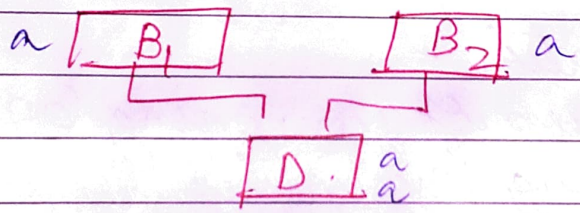
* We have several types of inheritance like multiple Inheritance, hierarchial I, single I, Multilevel I, hybrid I.

① Single I.



process of getting a single derived class from a single base class

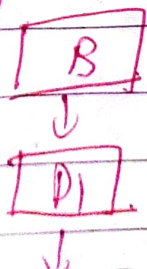
② Multiple I.



process of deriving a single derived class from multiple base classes

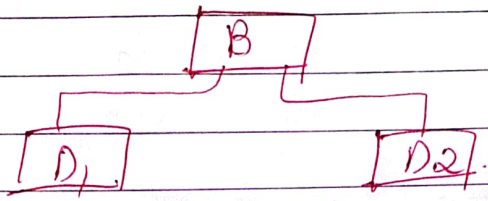
But Java does not support multiple I because its take variable a is both B_1 & B_2 then derived class also has both a's which creates a problem to compiler.

③ multilevel I



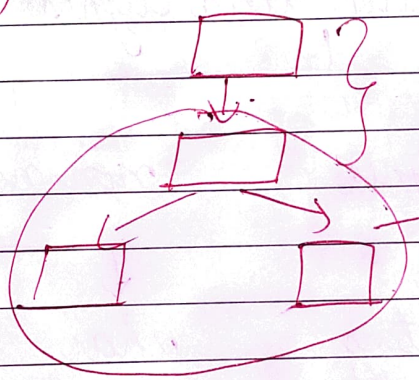
here D1 acts as a base class for D2

(5) Hierarchial I



process of creating several derived classes from a single base class

(4) Hybrid



Combination of more than 1 type of I

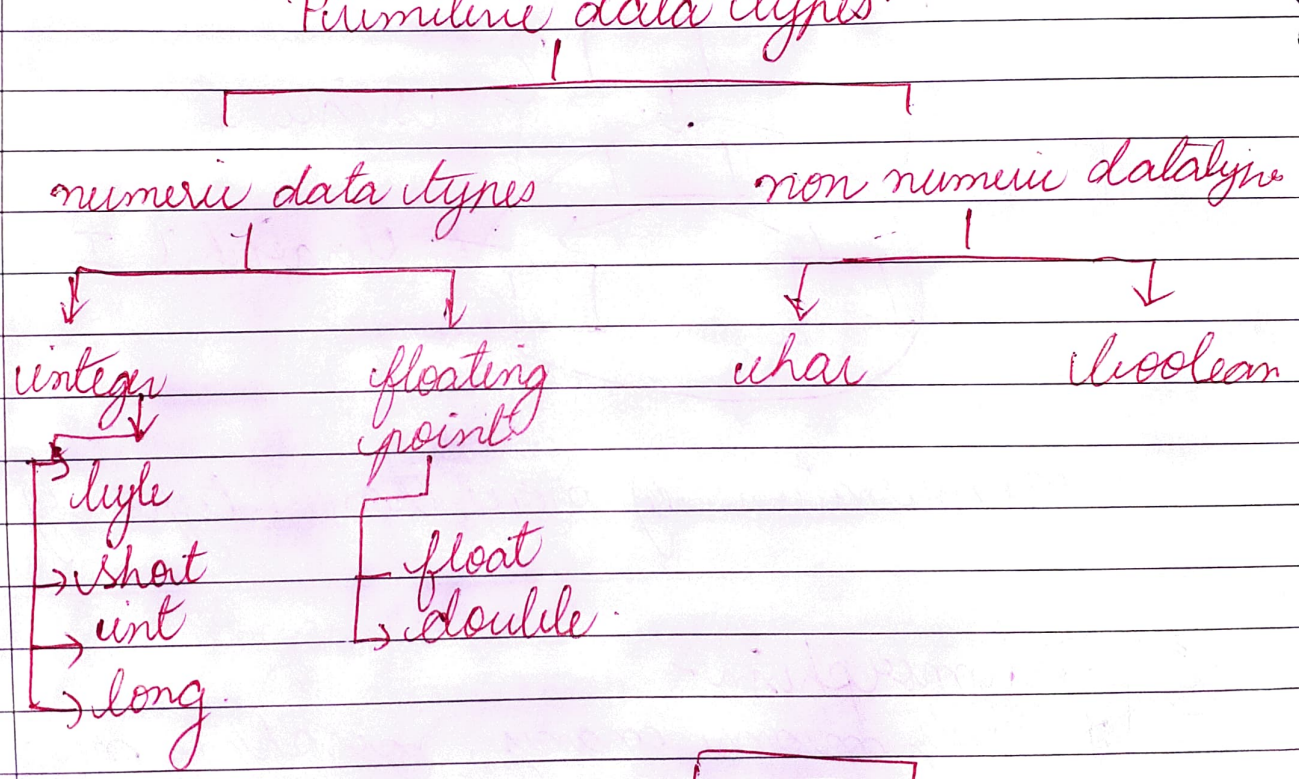
(6) Polymorphism:

* Poly means many, morphism means forms
* Representing 1 thing in many forms is called as polymorphism
* Compile time polymorphism: The decision abt which method will be called is decided during compile time
eg: Method overloading

* Run time polymorphism :-
The decision abt which method will be called, binding is decided during run time
Eg :- method over riding

DATA TYPES IN JAVA

① primitive data types
② non primitive data types.



Unsigned data type = $2^n - 1$; only +ve.
Eg 2 bits = $2^2 - 1 = 3$.

Signed = -2^{n-1} to $2^{n-1}-1$ where n = 8

1 byte = 8 bits

char = 2 size bcoz yaha uses unicode

	datatype	size (in bytes)	range	default	Ex.
signed	byte	1	-128 to 127	0	byte a = 10
	short	2	-64 to 64	0	short a = 25000
	int	4	-16 to 16	0	int a = 65765
	long	8	-16 to 4	0	lon
unsigned	float	4	3.4e-38 to	0.0	float a =
	double	8	$1.7e^{3.4e+38}$ to		a = 1.2f
	char	2		0	-
	bool	1	+a E	—	

Rough

Ex -2^{8-1} to $2^{8-1} = -128$

$-2^{8(2)-1} = -32768$ to -32767

$-2^{8(4)-1} = -2,147,483,648$

-2^{31} to $2^{31}-1$.

-2^{63} to $2^{63}-1$.

Non primitive Data types

Strings arrays class

LITERALS IN JAVA

* Literals is a constant value which is assigned to a value
* Literals are divided into 5 types
  a) Integer literal

a) Integer literal
1. decimal literal (base 10) - 0 to 9 [Ex 278]
2. octal literal (base 8) - 0 to 7 [Ex 020]
3. hexadecimal literal (base 16) - 0 to 9 & a to f

Ex :- 0x79
zero

b) Floating point literal
1. fractional notation :- Eg 123.48 | a = 1.25 | a = 1.25d
   after decimal 4 digits use float
   " " 10 digits use double.

2. exponential notation
   [m x b^e]
   Eg 2.2 x 10^-5
   Eg -0.5e^-5

c) Character literal
1. alphabet
2. digit
3. symbols

d) String literal
s = "abc"
inside double quotes
String is a class in Java

e) Boolean literal
a = true
b = false
T & F are diff, 0 & 1 r diff

__/__/____

Variables in Java

* It is used to store a value.
* The value may change during program execution

Declaration of a variable: datatype variable_name;

int a;

Eg int a=10;
int a=20;
int a=30;

Dynamic Initialization of variable.

If init is performed during runtime.

Eg double pi=3.14.
double area=pi*r*r //
Area is dynamically init

Scope & lifetime of a Variable.

* Scope specifies where the variable is going to be accessed.
* Lifetime specifies how long the value will be available

Demo of Scope of a variable.

```
Class DemoScope
{
  public static void main (String args[])
  {
    int x; // accesable to all the stats in main
    x = 10;
    if (x==10)
    {
      // starts new scope.
      int y = 20; // accesable only to this block
      S.O.println ("x and y:" +x +" " + y);
      x = y * 2;  // x = 20x2 = 40  ∴ x = 40
    }
    y = 100; // error, y cant be accessed here
    S.O.println ("x =" +x);
  }
}
```

here we can access 'x' throught the main method & can access y only in the if block

→ Demo of lifetime of a Variable.

```
Class DemoLifitime
{
  p s v m (S a[])
  {
    int x;
    for (x=0; x<3; x++)
    {
      int y=-1
```

```
S.O.println("y is" +y);
y=100;
S.O.println("y now is" +y);
}
}
}
```

Here as long as the control is in for loop the y = -1
And whenever control comes out of the for loop the y is no longer available

TYPES OF VARIABLES

① Instance Variables

Def: * The variables which are declared inside the class but outside of all the methods
* Also called as member variables or fields

memory allocation:
Whenever an object is created, the memory will be allocated for the instance variable

Scope & lifetime of Instance Variable:
whenever an object is created, the corresponding instance variables will be allocated for the memory. So whenever the object is destroyed then only the instance variables will be destroyd

When can we use:
If the value of the instance variable varies from one obj to another obj

Default Initial value
It is provided by the compiler
eg '0' for integer, 0.0 for float,

How we can Access Instance Variable.
We can access IV with the help of objectname.variable

Ex program:

```
class Test
{
 int i;
 P S V m (S a[])
 {
  Test t = new Test();
  S.O.p ln (t.i);
 }
}
```

∴ i = 0.

② Static Variables

Def: The variables that are declared inside the class & out of all methods

* Diff is we use Static keyword

Memory Allocation:
Whenever a class is loaded into the memory then the memory will be

static = global

__/__/____

allocated for the static variable
* static variables are known as class variables
* The memory will be allocated only once and all the objects of a class will use the same memory

Scope and lifetime of a variable
→ For static variable, the memory will be allocated while the class is loaded into the memory
→ whenever the class is unloaded the so the memory for static variables will be deallocated

When can we use
→ If we want to use a common variable for all objects of a class then we have to go for static variable
→ i.e if the value of the variable is fixed then we use static vaeriable

Default Initial Value
The Default Initial Value of a static variable is 0

How to access static Variable
→ static variables are assosiated with class
→ classname.variablename

Eg:
```
Class Test {
    int i;
    static int i;
    PSVM (S a[])
    {
```

```
{
  Test t = new Test();
  t.i = 10; //accessing instance v
  Test.j = 20//accessing static v
  S.O.P("Initalies i = " +i); //10
  S.O.P("static j = " + Test.j); //20
  }
}
```

(or) j

If static method declaration & main method is in the same class then there is no need to use class name directly we can access

(3) Local Variables

Def:- The variables which are declared inside the method

Memory Allocation :- As local variables are declared inside the method, so whenever we declare a local variable, the corresponding memory will be allocated

Scope & lifetime

Whenever the control is inside the block we can use the local variables

When can we use

We can use it inside the method only

Default Initialization

We have to input any value as default or else the compiler will throw error message

Access:

We can access the local variable directly with the help of variable name.

Eg:-

```
Class Test
{
  PSVM(S a[])
  {
    int i=10;
    S.O.P(i);   //10
  }
}
```

Ex:-

```
Class Test
{
int a=10; // Instance Variable
Static int b = 20; // Static Variable
PSVm(S a[])
{
int c=30, // local variable
Test obj = new Test();
S.O.P("Instance V is" + obj.a);
S.O.P("Static V is" + Test.b);
S.O.P("Local V is" + c);
  }
}
```

Output: 10
20

Static members in Java

1. Static variable
2. Static method
3. Static block

Method Overloading in Java

* It is the concept of defining multiple methods with same name but diff signature
* Signature specifies no of arguments or order of arguments

Eg:

```
Class Test                                  10      20.
{
  int Sum (int a, int b).
  {
    return (a+b);
  }                                         1       2       3.
  int sum (int a, int b, int c).
  {
    return (a+b+c);                                 5       3-3.    8
  }
  double sum (int a, double b, int c)
  {
    return (a+b+c);                                 1.1     2.2
  }
  double sum (double a, double b).
  {
```

```
{return (a+b);
}

PSVM (SA[])
{
  Test obj = new Test();
  S.O.P (obj.sum(10,20));      //30.
  S.O.P (obj.sum(1,2,3));      //6
  S.O.P (obj.sum(1.1, 2.2));
  S.O.P(obj.sum(5, 3.3, 8);
}
}
```

here control goes to sum method which accepts 2 integer values.

* Method overloading is an example for compile time polymorphism or early binding / Static binding
* During compile time which method will be called is decided

CONSTRUCTORS IN JAVA.

* Constructor is a special method which is used inorder to initialize an object.
* The variables that are declared inside the class are called IV or fields.
* A constructor initializes the values to instance variables or fields

* A constructor constructs an obj
* The constructor contains a block of statements which performs a specific task.

Rules to define a constructor -

(1) Constructor name must be same as class name
(2) Constructor do not return any value so there is no need of return type before the constructor, not even void.
(3) We can give any access specifier before the constructor
(4) Constructors can't be final, abstract, synchronized (all these modifier can't be used)
(5) A constructor should be called at the time of object creation.

Types of Constructors

a) Default constructor
b) Parameterized constructor

a) * A constructor without args
* Also be called as no argument constructor
* If we didn't define any constructor inside a class then the compiler will define a default constructor & defines initializes default value to the corresponding IV

Eg:-

```
Class Rectangle
{
    double l;
    double b;
    Rectangle()  // constructor (default)
    {
        l = 10;
        b = 20;
    }
    double area()  // area method
    {
        return l * b;
    }
    PSVM(Sa())
    {
        Rectangle obj1 = new Rectangle();
        Rectangle obj2 = new Rectangle();   //200.0
        S.O.P("Area of 1st obj: " + obj1.area());
        S.O.P("Area of 2nd obj: " + obj2.area());
    }                                       //200.0
}
```

obj 1
l - 10
b - 20

Here the output is same for all the objects, we can overcome this problem by using parameterized constructor.

D) A constructor with parameters is called as parameterized constructor

Eg :-

```
class Rectangle
{
    double length;
    double breadth;
    Rectangle (double l, double b)        // param constructor
    {
        length = l;
        breadth = b;
    }
    double area ()                        // area method
    {
        return length * breadth;
    }
    PSVM (SAC)
    {
        Rectangle obj = new Rectangle (10, 20);
        Rectangle obj2 = new Rectangle (30, 40);
        S.O.P ("A of obj1" = +obj1.area());
        S.O.P ("A of obj2 = +obj2.area());
    }
}
```

obj	obj2
l [10]	l [30]
b [20]	b [40]

output

// 200

// 1200

Constructor Overloading

* Constructor Overloading means defining multiple constructors with same name in Java but with different signatures
* Sometimes there is a need of initializing object in several ways

```
Class Rectangle
{
  double length;
  double breadth;
  Rectangle()
  {
    length = 10;
    breadth = 20;
  }
  Rectangle(double a)
  {
    length = a;
    breadth = b;
  }
  Rectangle(double l, double b).
  {
    length = l;
    breadth = b;
  }
  double area()
  {
    return length * breadth;
  }
```

```
PS vm (S a[])
{
Rectangle obj1 = new Rectangle();
Rectangle obj2 = new Rectangle (30);
Rectangle obj3 = new Rectangle (40, 50);
S.O.P ("0 Args area is " +obj1.area(1));
S.O.P ("1 Args area is " +obj2.area(1));
S.O.P ("2 Args are ars " +obj3.area(1));
}
```

Strings class VS String Buffer class.

String Class.

* String objects are immutable
  i.e not changable.

Ex: String s = new String ("Java");
s.concat ("prog");
S.O.P (s);

* String class overrides equals() of object class.

Eg: String s1 = new String ("hi");
String s2 = new String ("hi");
S.O.P (s1.equals (s2)); // true

String Buffer Class.

* String buffer objects are mutable
  i.e changable

__/__/____

Ex StringBuffer S= new StringBuffer("Java").
S.append("programming");
S.O.P(S)

* String buffer class doesn't override equals() of object class.

Ex: StringBuffer S1 = new StringBuffer("Ian")
StringBuffer S2 = new StringBuffer("Ian")
S.O.P(S1.equals(S2)); // False.

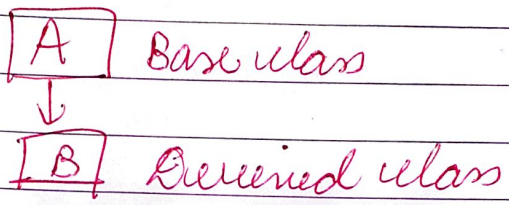
INHERITANCE IN JAVA.

* The process of creating a new class from an already existing class
* The old class can be called as base, parent, Super class
* The new class can be called as derived, child, Sub class
* The base class contains its members where as the derived class contains its members and also the members of parent class
* The best example of Inheritance is human beings. As we have inherited the features of our parents.
* The major advantage of inheritance is reusability.

Types Of Inheritance — (5)

① Single Inheritance

*The process of deriving a single derived class from a single base class is called single inheritance



Eg:-

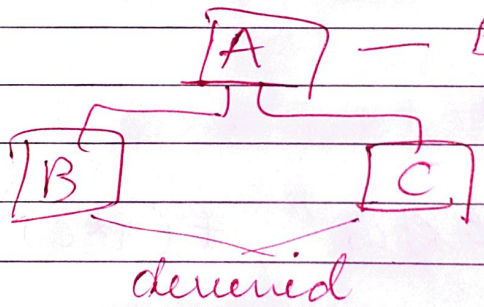
```
Class Base
{
  int a;
}
Class Derived extends Base
{
  int b;
  Derived().  // default constructor
  {
    a=10;
    b=20;
  }
  void sum()  // sum method
  {
    sop S.O.P("sum is :" +(a+b));
  }
```

```
Class Test          // main method
{
  P S V m (S a[])
  {  Derived
    Test obj = new Test(); Derived ();
    obj. show ();
  }
}
```

O/p = 30.

② Hierarchial Inheritance

* The process of creating several derived classes from a single base class is




```
Class Base
{
 int a;
}

Class Derived extends Base
{
  int b;
   Derived1 ()
   {
```

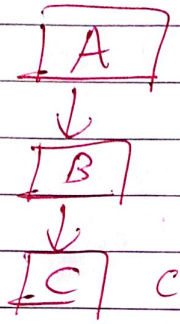
```
    a=10
    b=20;
  }
  void sum()
  {
  S.O.P ("sum is"  +(a+b);
  }
}
```

```
class Derived2 extends Base
{
  int c;
  Derived2 ()
  {
    a=1.
    c=2;
  }
  void mul()
  {
  S.O.P ("mul is "  +(a*c));
  }
}
```

```
class Test
{
  PSVM (S a[])
  {
  Derived1 obj = new Derived1();
  obj1.sum() // 30
  Derived2 obj2= new Derived2();
  obj2.mul() // 2
  }
}
```

(3) Multilevel Inheritance

The process of creating a class from a derived class




```
Class Grandparent
{
  int a;
}

Class Parent extends Grandparent
{
  int b;
}

Class Child extends Parent
{
  int c;
  Child()
  {
    a=10;
    b=20;
    c=30;
  }
  void sum()
```

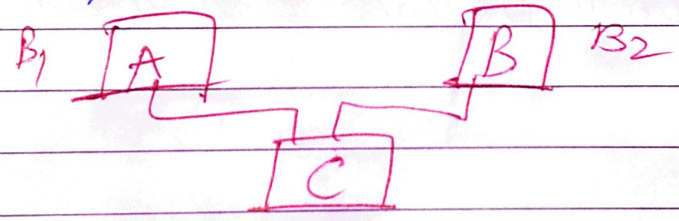
```
        S.O.P("sum is: " +(a+b+c));
    }
}

Class Test
{
    psvm(sa[])
    {
        Child obj = new Child();
        obj.sum(); // 60
    }
}
```

10 20 30

(24) Multiple Inheritance

* The process of deriving a class from multiple base classes



C

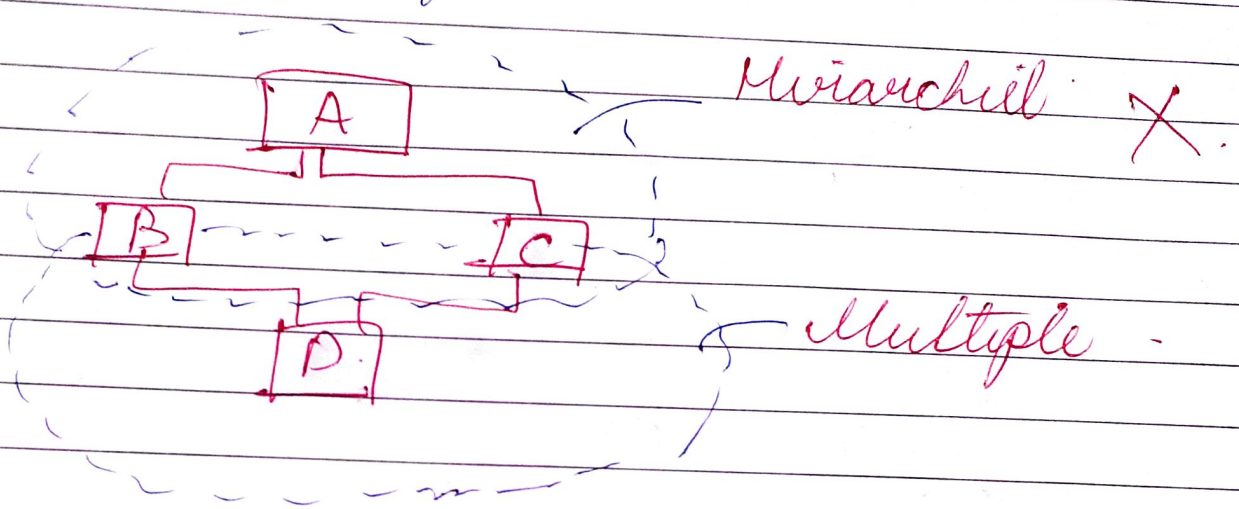
* Java does not support Multiple I
* This is mainly bcoz of ambiguity problem.
* Suppose A has a ee B has a then the C produces of both the copies of a á
* So Java ee M C# doesnt support M I.
* If we still want to implmit this

we use the concept of Inheritance

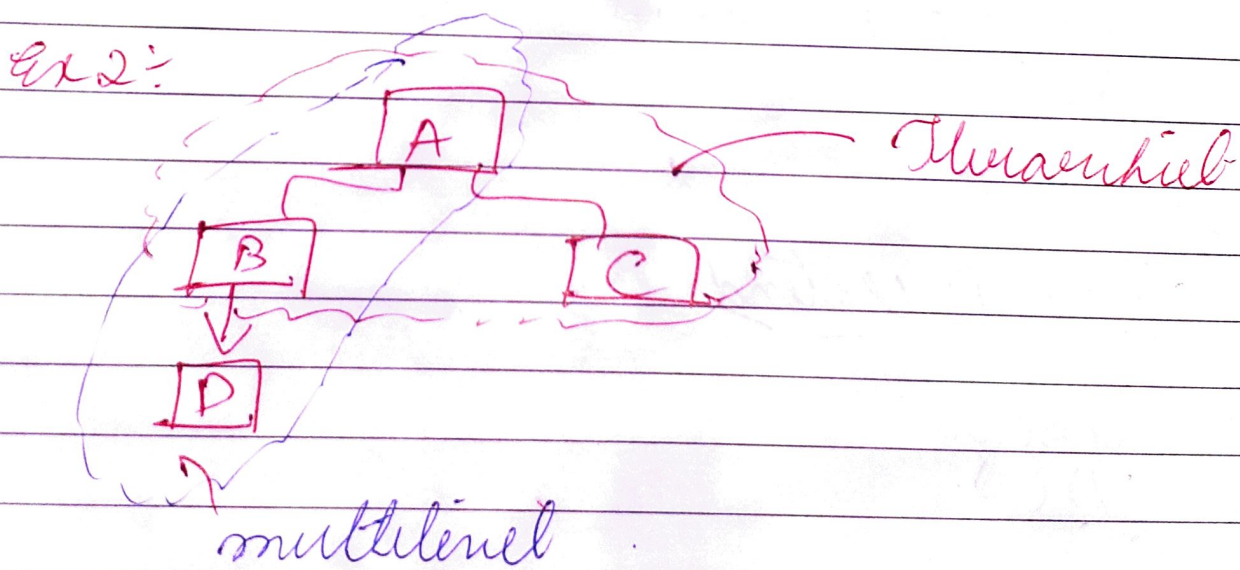
(5) Hybrid Inheritance

* It is a combo of more than 1 type of I
* If it performs multiple I then it must perform MI

Ex 1



Ex 2:




```
Class A
{
  int a;
}
```

```
Class B extends A
{
  int b;
}

Class C extends A
{
  int c;
  C()
  {
    a=10;
    b=20;
  }
  void sum()
  {
    S.O.P ("Sum is" +(a+b));
  }
}

Class D extends B
{
  int d;
  D()
  {
    a=10
    b=20.
    C=30.
  }
  void mul()
  {
    S.O.P ("mul is" +(a*b*c);
  }
}
```

```
Class Test.
{
  P S V m (Str args[]),
  {
    C obj1 = new C ();
    obj1. sum (),          //30
    D obj2 = new D ();
    obj2. mul (),          //600.
  }
}
```

METHOD OVERRIDING IN JAVA.

Method Overriding is possible only by implementing Inheritance

```
Class Base.
{
  void display ()
  {
    S. O. P ("parent class"),
  }
}

Class Derived extends Base.
{
  void display ()
  {
    S. O. P ("child class"),
```

```
    }
}

class Test
{
    p v s m (S a[]).
    {
        Derived obj1 = new Derived();
        obj1.display();
        Base obj2 = new Base();
        obj2.display();
    }
}
```

O/P = Child class

* Here the derived class display method is overriding base class display method
* It is an ex for run time polymorphism Dynamic poly / binding, late binding

Rules for Method Overriding

1. The method name in the base class & derived class should be same
2. The signature of the corresponding methods must be same (args)
3. The return type must be same for Base class & Derived class
4. private methods can't be overloaded
5. static methods can't be overrided

⑥ final methods can't be overruled

SUPER KEYWORD.

Advantages of Super Keyword.

① To access super class variables
② " " " " methods
③ " " " " constructors

① Demo of SC variables to class.

* If the base class & derived class contain same variable then we use super keyword

```
Class Base
{
    int a = 10;
}

Class Derived extends Base
{
    int a = 20;
    void display()
    {
        S.O.P ("a is" + a);        //20
        S.O.P ("super class a is" + super.a);
    }                              //10
}
```

__/__/____

```
class Test
{
    psvm (S a[])
    {
        Derived obj = new Derived();
        obj.display();
    }
}
```

② Demo to access super class methods.

```
class Base
{
    void show()          // method1
    {
        S.o.p("Base class");
    }
}

class Derived extends Base
{
    void show()          // method1
    {
        S.o.p("DC");
    }

    void display()       // method2
    {
        show();          // DC
        super.show();    // Base class
    }
}
```

```
Class Test
{
  P S V m (St args())
  {
    Derived obj = new Derived();
    obj.display();
  }
}
```

③ Demo to cause Super class constructor

```
Class Base          ↓100.
{
  Base(int i)
  {
    S.O.P("Base class constr value is" + i);
  }
}

Class Derived extends Base
{
  Derived()  // default constructor
  {
    super(100);
    S.O.P("Derived CC");
  }
}

Class Test {
  P S V m (s a)
  {
    Derived obj = new Derived();
```

__/__/____

FINAL KEYWORD

We can apply final keyword on variable, on method, on class

① Demo to access final variable → to create (constant variables)

```
Class Base
{
final int a=10; // const.
    Base()
    {
      a = 20;
    }
}

Class Test.
{
  PSVM (Sa[]).
  {
    Base obj = new Base();
  }
}
```

(we cant change)

O/P = compilation time error.

② Demo for final method

The final methods prevents method overriding.

Overriding is not possible in Final :-

```
Class Base.
{
final void show().
   {
     S.O.P ("Base").
   }
}

Class Derived extends Base.
{
   void show().
   {
     S.O.P ("derived").
   }
}

Class Test
{
   P S VM (S a[])
   {
     Derived class = new Derived ();
     Obj.show
   }
}
```

o/p = compile time error.

(3) Demo for Final Class
The final class prevents Inheritance

__/__/____

final Class

Eg:

```
final Class Base
{
    void show()
    {
        S.o.p ("show in Base");
    }
}

Class Derived extends Base
{
    void show()
    {
        S.o.p ("show derived");
    }
}

Class Test
{
    P S V M (S a[])
    {
        Derived obj = new Derived ()
        obj.show ()
    }
}
```

o/p compile time error

PACKAGES

* Every package is a collection of classes & interfaces
* Packages are classified into
  a) pre defined package
  b) user defined package

a) * java . lang (package) — String (class) — isEmpty() (methods), toUpperCase()
   — StringBuffer — length(), capacity()
   — Math — max(), floor(), sqrt(), cos()

* java util — Scanner — nextInt, nextFloat

* java.io — data input stream, data output stream

* java.awt
* java.applet

* Each class has a collection of methods
* When we use import . java.lang.* then all the classes will be implimented
* java.lang.Str means only string class will be implimented

b) * The packages which are defined by the user is called as user defined packages

* Syntax : Package package-name.
* The access specifier of the package must be public.
* We need to define a method inside the package.
* Even the access specifier of the method must be public.
* -d means in which directory you want to store the class file.
* If we want to store the class file in the present directory then we use .

Eg :
Hello.java.

```
Package myPack1;
Public Class Hello
{
  Public void display()
  {
   S.O.P ("Hello world");
  }
}
```

Compilation : javac -d . Hello.java.
Eg C → progams → mypack1 → Hello.class

But this doesn't contain main method
So the package program wont execute.
* Now we import this package.

Test.java (filename)

```
import mypack1.Hello;
Class Test
{
  P S V M (Sa[]).
  {
    Hello obj = new Hello();
    obj.display();
  }
}
```

Compilation: javac Test.java
Run: java Test

Example: Java program to calculate the factorial of a given no using package.

```
Package mypack2.
public class Factorial
{
  public void fact(int n)
  {
    int i;
    int f=1;
    for(i=1; i<=n; i++)
      f= f*i
    S.O.P ("factorial is" +f);
```

```
    }
}
```

compile :- javac -d . Factorial.java

```
import mypack.*;
class Test
{
  PSVM (SA[])
  {
    Factorial obj = new Factorial();
    obj.fact(5);
  }
}
```

Compilation :- Javac Test.java
Run :- java test

o/p = 15.

COMMAND LINE ARGUMENTS -