Object Oriented Programming language (BCSC0002)

Software

System Software
- It works with machine.
- windows
- Linux
- MAC.

Application Software.
- Local
  - Window → Desktop. (run on particular System).
- Global
  - Server application (run on servers).
  - web application.

Language:- Medium of communication.
- Source language
- computer language.

Translators :- Three types of translators.
- Assembler :→ Single character by single character (low)
- Interpreter :→ Single row by single row. (medium)
- Compiler :- whole order. (Highest)

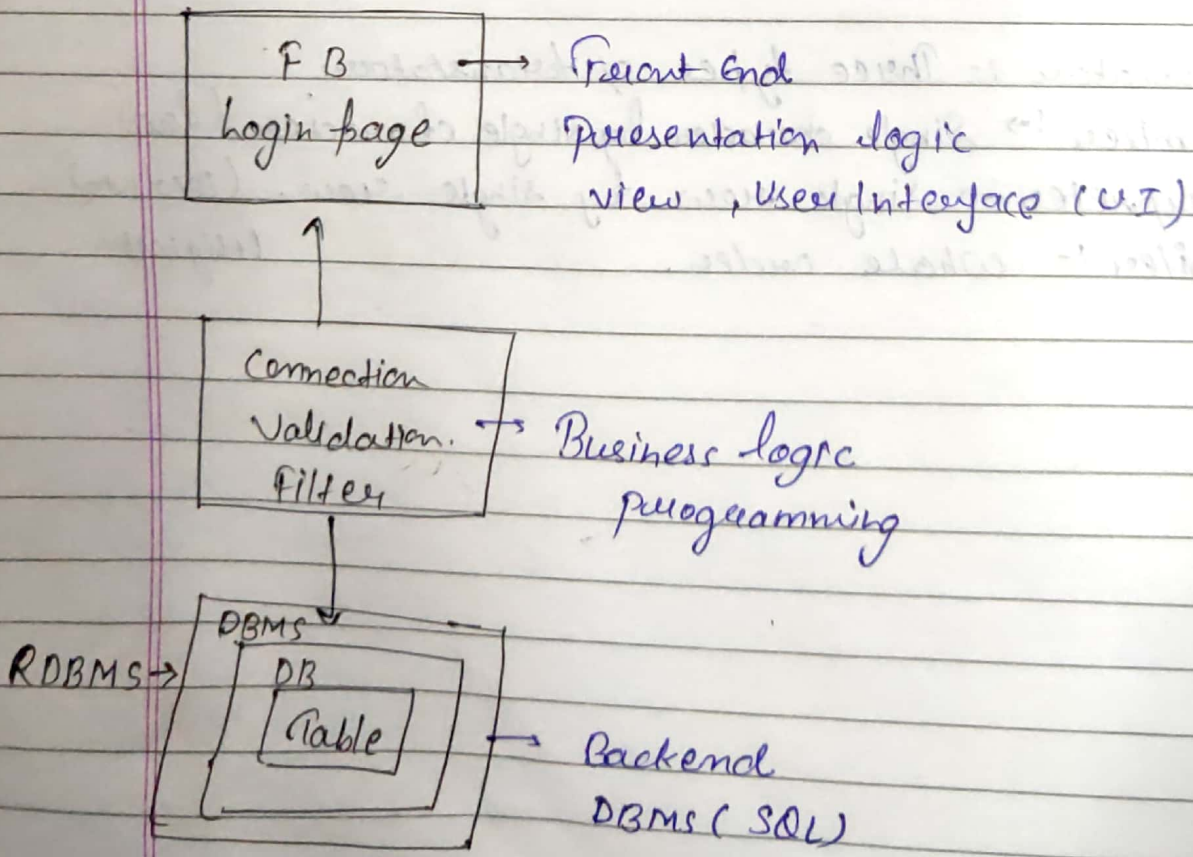
Date / /
Page

Computer Related language.

Front end.

Programming	structured	Scripting	Markup
→ Business logic	→ Backend	→ client site	→ tag based < >
→ code construction	→ Data storage in logical format	→ validation on the web page.	→ HTML
→ Validation.			→ XML
local global	→ DBMS.	• javascript	
C –	• Oracle	• VB script	
C++ —	• My sql	• jquery	
Java Java	• MS SQL server	• Angular JS	
python python	• MS Access	• Note JS	
– PHP	• Sqlite		

Example:-



My Write is My Future

Scanned by CamScanner

→ Evolution of Programming language

1957	Fortran	(Formula translation)	Assembler Based
1960	Algol	(Algorithim Based)	
1963	BCPL	(Basic combine prog. language)	
1967	CPL	(combine prog. language)	
1970	B lang	developed by Martin Richard. → Interpreter	
1972	C lang	developed by Denis Ritchie → compiler Based	

1980 C with Advance OR C++ (Bjarne Stroustrup)
1991 Java developed by James Gosling.
OAK → Real name.
renamed in 1995 as Java by SUN microsystem which is later hieried by oracle in 2008.

Drawbacks of C or C++	Objective
→ local	→ Global
→ Platform Independent	→ platform Independent
→ Insecure	→ Secure.

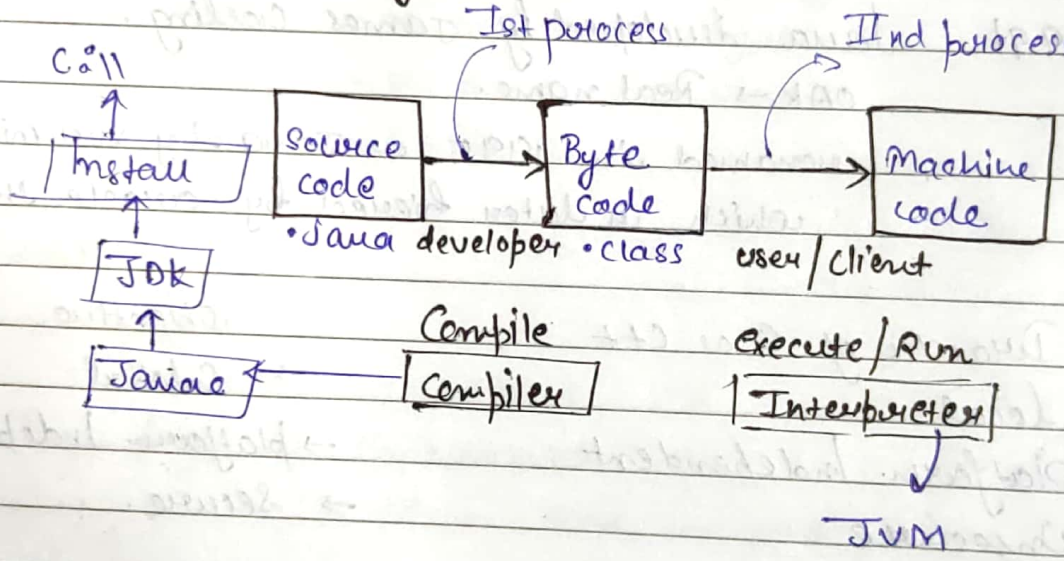
J2SE	J2EE	J2ME	EJB	Framework
→ Java 2 Standard Editon	Java 2 Enterprises Edition.	Java 2 micro Edition	Enterprises Java Bean	JSF presentation
→ core java	• Advanced java	→ Wireless	• session Bean	Hibernate Backend
→ local app.	• Global app.	• Broad Band	• Entity Bean	Spring
↙ ↘	→ servlet	• Data Card	• MDB message Driven Bean	Business logic
CUI app. (Desktop app.)	GUI app. (Mobile)	→ JSP → java bean	• mobile app.	

MVC

→ Model View Controller.
↙ ↓ ↘ Business
Data presentation logic.

→ Struts
It is also a framework based on MVC.
Apache Tomcat Vendor

⇒ Architecture of Java



Java is a two process

Java Application Development

By using Manual
• Note pad
• Command prompt

By Using Id.
• Intelli j Idea
• Net Beans
• Eclipse etc

WORA → Write Once Run anywhere.

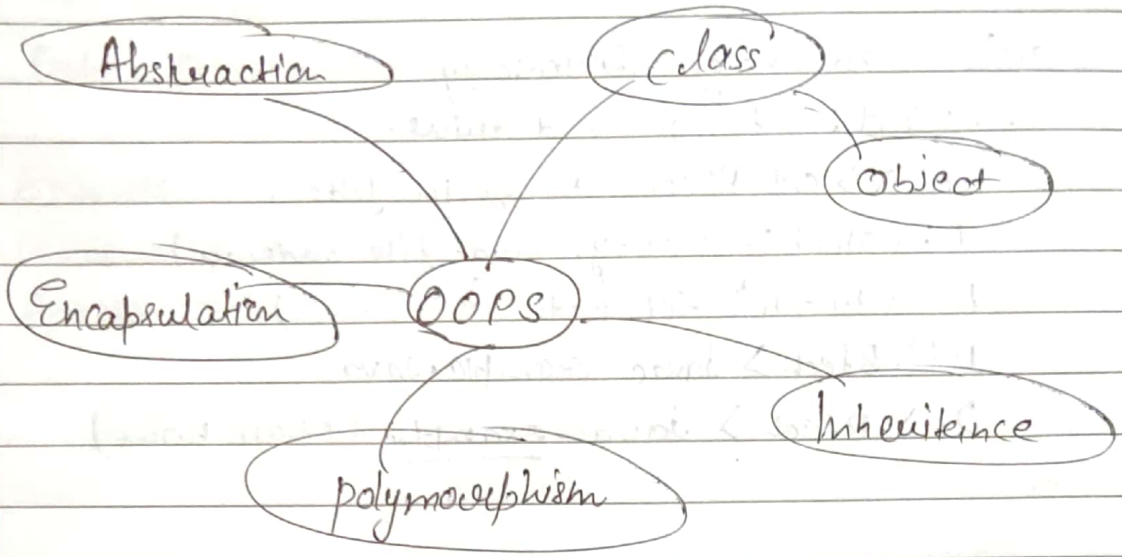
Date __/__/__
Page _____

Compile → Javac filename.java
execute → java filename

cd\ → to change directory.
e:\> d: → to go in d drive.
D:\> cd Btech to go in folder.
D:\Btech> (copy, Javac file address)
D:\Btech> set path (↓).
D:\Btech> javac example.java
D:\Btech> java example (class name)

Date / /
Page

Concept Of OOPs :-



- Abstraction :- Focus on the essential facts that is called Abstraction.
- Encapsulation :- Hiding of the non-essential detail is known as encapsulation.
- Polymorphism :- One thing many forms is called polymorphism.
- Inheritance :- When a generation use the features of another generation is known as Inheritance.

OR

When a class use the features of another class is called Inheritance.

My Write is My Future

→ Class

A group of different kind of entity, in a single unit is called class.

OR

Collection of variables, functions/methods and keywords in a single unit is known as a class.

→ Object

It is an instance of a class or reference of a class

→ Function

It is a set of instructions.

★ An object represent abstraction.

★ A class represents encapsulation.

```
Class GLA
{
Void Btech(
 {
 S.o.p(" Btech from CS");
}
 Void Btech ( )
{
S.o.p(" Btech from EC");
}
Public Static Void main (string args[])
{
GLA obj = new GLA();
 obj. Btech();
 }
}
```

Error :- Method Btech is already defined in class GLA.

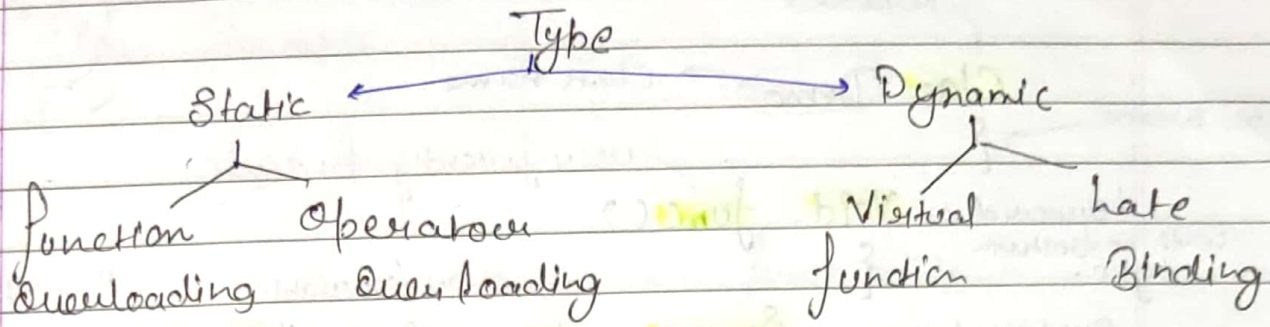
→ If we write. Void Btech(int acc)

```
obj. Btech(1500);
obj. Btech( );
```

o/p → Btech from Ec
Btech from CS

Date / /
Page

Polymorphism



Function Overloading

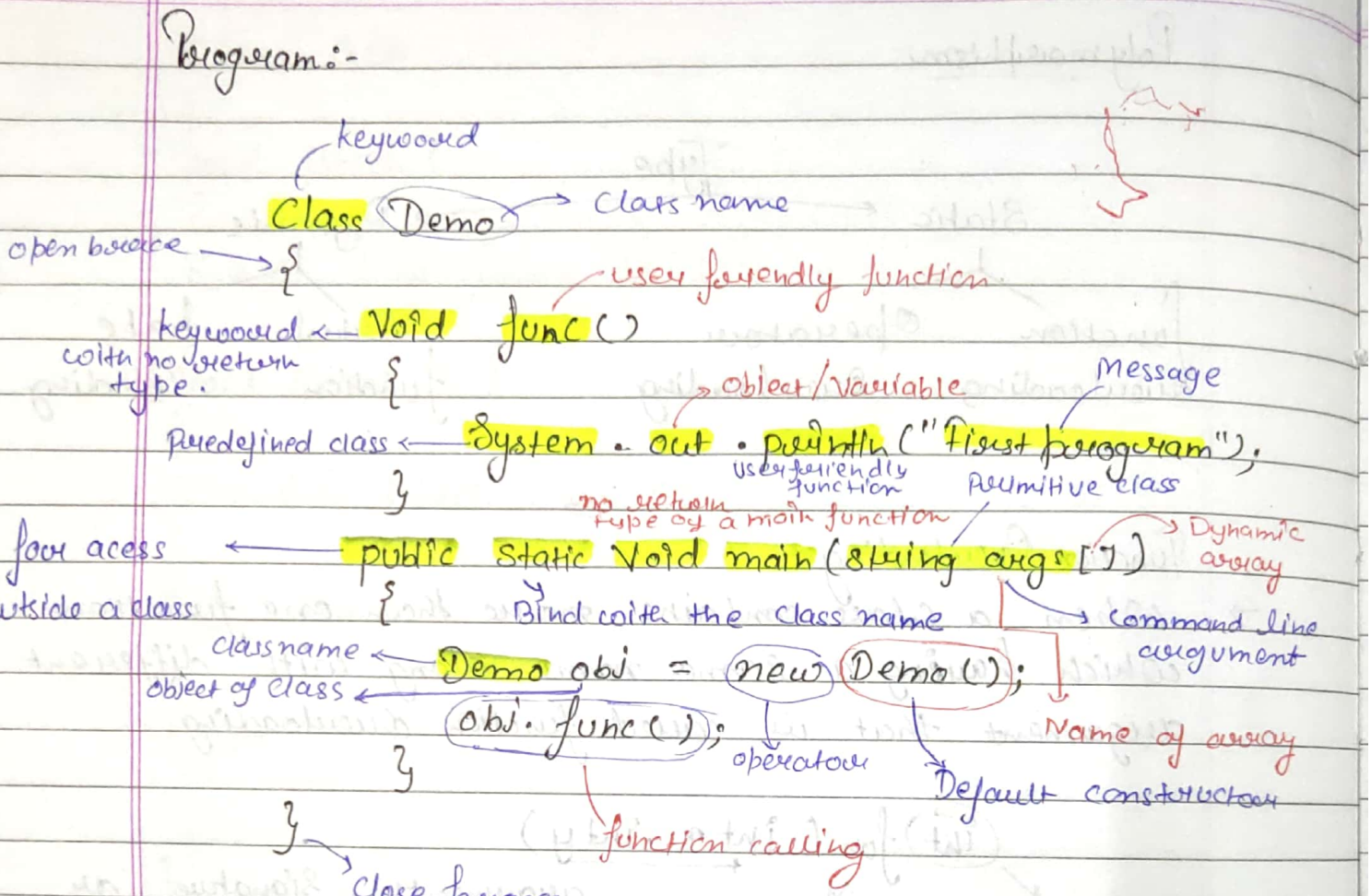
→ When a class contains more than one function which having a same name along with different argument that is called function overloading.

```
int fun ( int x , int y )
```

- int → Return type
- fun → Function name
- (int x , int y) → arguments, Signature or parameters.

- Function name must be same
- Function argument must be different
- function return type may be same or not.

Program:-



Static function:-

It calls from class name.

Non-static function calls from object.

Static

→ Static function Example

```
Class Stexample
{
    Void func()
    {
        System.out.println("function");
    }
    public static void main(String args[])
    {
        Stexample obj = new stexample();
        obj.func();
```

Date / /
Page

Sum of two Variable which input by command line argument with numeric conversion.

```
class cmdexample
{
    public static void main(String args[])
    {
        System.out.println(args[0]+args[1]);
        int var1 = Integer.parseInt(args[0]);
        int var2 = Integer.parseInt(args[1]);

        int res = var1+var2;
        System.out.println("total"+res);
    }
}
```

→ Print all elements of array which input by command line argument with using for loop along with length variable.

```
class cmdexample2
{
    PSVM(String args[])
    {
        int size = args.length;
        S.o.pln("no of elements:"+size);

        for(int i=0; i<size; i++)
        {
            S.o.pln(args[i]);
        }
    }
}
```

My Write is My Future

Date / /
Page

length Vs length()

↓ ↓

Variable . function

predefined in java

- length :- it is predefined variable in java. It is used for finding length (no. of elements) of an array.
- length() :- It is a predefined function in the java. It is used for finding length (no. of characters) of String.

→ Sum of elements of array which input by commandline argument with using for loop 'arg' with length variable.

```
class Jha
{
  public static void main(String args[])
  {
     int size = args.length;
     S.o.pln("no of elements:"+size);

     int sum=0;
     for(int i=0; i<size; i++)
     {
      int var = Integer.parseInt(args[i]);
       sum = sum + var;
       S.o.pln(var);
     }
     S.o.pln("total:"+sum);
  }
}
```

My Write is My Future

=> For each loop.

Print all elements of array which input by cmd line argument with using for each loop.

```
Class Loop
{
    P s v m (string args[])
    {
        // foreach loop
        for (String var : args)
        {
            S.o.pln (var);
        }
    }
}
```

2). Sum all elements of array which input by command line argument with using for each loop.

```
Class Summation
{
    public static void main (String args[])
    {
        int sum = 0;
        for (String var : args)
        {
            int d = Integer.parseInt (var);
            sum = sum + d;
        }
        System.out.println (sum);
    }
```

My Write is My Future

- Print all elements of Numeric Array with using for loop.

```
Class Array
{
  public static void main(String args[])
  {
     int ar[] = new int[5];
      ar[0] = 10;
      ar[1] = 12;
      ar[2] = 14;
      ar[3] = 16;
      ar[4] = 18;
    for(int i=0; i<5; i++)
    {
      System.out.println(ar[i]);
    }
  }
}
```

- Print all elements of Numeric array which are using for each loop;

```
class Array
{
   P S V m (String args[])
   {
      int ar[] = {10,20,30,40,50,70};
      for(int var : ar)
      {
        S.o.pln(var);
      }
   }
}
```

- Sum of array elements with using for each loop

```
Class Array
{
  public static void main (String args[])
  {
     int ar[] = {10, 20, 30, 40, 50, 60}
     int sum = 0;
     for (int var : ar)
     {
        S.o.println(var);
        sum += var;
     }
     S.o.println ("total !! " + sum);
  }
}
```

- Count no. of elements of Numeric array with using for each loop without using length variable.

```
Class Array
{
  P S V m (String args[])
  {
     int ar[] = {10, 20, 30, 40, 50, 60};
     int count = 0;
     for (int var : ar)
     {
        S.o.println (var);
        count ++;
     }
```

My Write is My Future

```
        S.o.println("no of elements : count);
    }
}
```

#. Inheritance :- (Single Inheritance)

```
//base // parent // super class
 class SM
{
  Void java()
   {
     System.out.println("Java Prog---");
   }
}
//derived // child // sub class.
class Oracle extends SM
{
    Void dbms()
    {
      System.out.println("sql---");
    }
    public static Void main(String ar[])
     {
      Oracle obj = new oracle();
        obj.dbms();
        obj.java();
    }
}
```

file will be save with name oracle.

My Write is My Future

⇒ Single

SM
Java

Oracle
dbms

dbms()
Java()

Multiple

SM
Java

Google
Android

Oracle SM Google
dbms

dbms()
Java()
android()

Multilevel.

SM
Java

Google SM
Android

android()
Java().

Oracle Google
dbms

dbms()
android()
Java()

Date / /
Page

Function Overloading

- When a class contains more than one function which having a same name along with different arguments, it is known as function overloading.
- It exist in polymorphism.

Function Overriding

- When an object contains reference of more than one function which having a same name along with same arguments, it is known as function over-riding.
- It exist in Inheritance.
- Derived class function overrides the base class function, therefore, Java provides final keyword for breaking overriding.

→ Function Overriding Program

```
class SM
{
    void java()
    {
        System.out.println("Java Prog---");
    }
    // Overridden
    void clang()
    {
        System.out.println("C lang by SM");
    }
class Oracle extends SM
{
    // Overridden
    void clang()
    {
    System.out.println("C lang by oracle");
    }
    void dbms()
    {
        System.out.println("sql---");
    }
    public static void main(String args[])
    {
        Oracle obj = new Oracle();
        obj.clang();
    }
}
```

o/p → Clang by oracle.

My Write is My Future

⇒ Function Overriding with using final keyword :-

```
class SM
{
  // Overridden
  Final void clang()
   {
    System.out.println("c lang by SM");
   }
}
class Oracle extends SM
{
    // Override
  void clang(int a)
   {
     System.out.println("clang by Oracle");
  }
   public static void main(String args[])
    {
      Oracle obj = new Oracle();
        obj.clang();
    }
}
```

o/p :- c lang by SM
c lang by Oracle.

Date / / Page

Final keyword

- It is used for making constant.

→ Final class can't be Inherited.
→ Final function cannot be overoride.
→ Final variable value can't be changed.

Constructor:-

Constructor is a special member of class because its name is same as a class name.

- Constructor invokes automatically when the object of class is created.
- Constructor doesn't have return type even void.

Ex:-
```
Class GLA
{
GLA()
 {
    System.out.println("Welcome in GLA");
 }
 public static void main(String args[]);
 {
   GLA obj = new GLA();
 }
```

- Constructor is used to first invoke in a class rest of the other function.
- Constructor invokes only one time from a single object.

(It's a disadvantage of the constructor).

My Write is My Future

Date / / Page

D. class GLA
{
GLA ()
{
System.out.println ("Welcome to GLA");
}
Void Btech()
{
s.o.pln ("Btech ---");

⇒ There are 3 types of a constructor :-

(i) Non-parameterized
It doesn't contains argument of itself.

Syntax :-

```
class name()
{
    // Statements
}
```

My Write is My Future

(ii) Parameterized

- It can contain no. of arguments of any Data type such as parameterized function.

Syntax:-

```
class_name (int x, int y)
{
    // statement
}
```

(iii) Default Constructor

- It is used for creating a new instance (object) with using new key word.

Syntax:-

```
class_name object_name = new class_name();
```

Default Constructor.

→ Constructor Overloading can be possible same as function Overloading.

```
class CLA
{
  CLA()
  {
    S.o.println("non para ....");
  }
  CLA(int a)
  {
    S.o.println("para ....");
  }
```

My Write is My Future

```
public static void main (String args[]).
 {
  GLA ob1 = new GLA (15);
  GLA ob2 = new GLA ();
 }
}
```

→ CONSTRUCTOR INHERITANCE (Constructor Chaining)

- Constructor cannot be inherited but derived class Constructor's responsibility is to call base class constructor.

- When a derived class constructor calls the base class constructor. Then first output will occur the base class constructor.

→ Non-parameterized Constructor Inheritance.

```
class SM
{
 SM ()
 {
  S.o.pln ("base ---- Non para SM");
 }
}
 Class Oracle extends SM
 {
    Oracle ()
    {
     S.o.pln ("derived --- non para");
    }
```

My Write is My Future

```
public static void main (String args[])
{
  Ocrade obj = new Ocrade();
}
}
```

o/p ⇒ base... non para SM
derived ... non para.

(1) Derived class parameterized cons. doesn't call the base class parameterized cons. therefore java provide super() method for calling base class parameterized cons.

→ Parameterized constructor Inheritance with use of super() method.

```
Class SM
{
  SM()
  {
    S.o.pln("base... Non para");
  }
  SM(int a)
  {
    S.o.pln("base... Single para SM");
  }
  SM(int a, int b)
  {
    S.o.pln("base... dual para");
  }
```

My Write is My Future

```
}
class Oracle extends SM
{
   Oracle()
   {
     S.o.pln("derived ... non para");
   }
   Oracle(int x)
   {
     super(x);
     S.o.pln("derived ... single para");
   }
   public static void main(String args[]).
   {
      Oracle obj = new Oracle(15);
   }
}
```

→ Note:- .super() method statement must be first statement in the derive class para. cons.

Abstract Class

→ An abstract is a class which can not create an object of itself.

→ An Abstract class is created with using abstract keyword.

Sample Program:-

```
abstract class DU
{
    Void Btech()
    {
      S.o.pln("Btech From DU");
    }
}
class college extends DU
{
   Void diploma()
   {
     System.out.println("Self course");
   }
   public static Void main(String args[])
    {
       College obj = new College();
        obj.Btech();
    }
}.
```

→ An abstract class can contains any type of method such as public, private and Abstract etc.

* Abstract Method :- An abstract method is a method which doesn't have a body of itself. It is declared by using abstracts keycoord.

→ An abstract method is declared only in an abstract class.
→ Defination of an abstract method will be defined in the derived class.

* Abstract Class With Abstract Method.

```
abstract class DU
{
    void Btech();
    {
        S.o.pln("Btech---");
    }
    // declare // overidden.
    abstract void manager();
}
class college extends DU
{
    // defination // override.
    void manager()
    {
        S.o.pln("DU Manager");
    }
    Void diploma()
    {
```

My Write is My Future

```
        S.o.p("Scy course");
    }
    public static void main(String args[])
    {
        college obj = new college();
        obj.Btech();
        obj.manager();
    }
}
```

→ When a class extends an abstract class then there is mandatory to provide the definition of all abstract methods of abstract class.

OR

→ All Abstract method of the abstract class must be override in the derived class.

INTERFACE

Java doesn't support multiple Inheritance therefore Java provides a concepts an Interface for Implementing multiple inheritance.

- An Interface is same as an abstract class which doesn't create an object of itself.
- An Interface is declared by using "Interface" keyword.
- An Interface contains only public and abstract method.
- An Interface provides only declaration of Method even public method also.
- When a class inherit a interface then there is mandatory to provide the definition of all methods of an interface.

My Write is My Future

OR.

→ All Methods of an Interface must be override in the derived class.

- A class inherit an Interface with using "implements" keyword.

#. MULTIPLE INHERITANCE USING INTERFACE PROGRAM:-

```
Interface DU
{
   abstract void Btech();
}
Interface UPTU
{
    Public void Mtech();
}
Class college implements DU, UPTU
{
   Public void Btech()
   {
       S.o.pln ("btech...");
   }
  Public void Mtech()
   {
       S.o.pln ("mtech...");
   }
  Public static void main (String args[])
   {
        college obj = new college();
          obj. Btech();
          obj. Mtech();
    }
```

My Write is My Future

STATIC FUNCTION IN INTERFACE

→ It has been implemented in Interface from jdk 7.

→ In Interface can contain static method with the defination from jdk 7.

→ Static function will not be overridden in the derived class.

→ It will be called by Interface name instead of derived class object name.

Example:-

```
interface DU
{
   abstract void Btech();
   static void func()
     {
       S.o.pln("Static fn in interface");
     }
}
class college implements DU
{
   public void Btech()
     {
       S.o.pln("Btech---");
     }
   public static void main(String args[])
     {
       college obj = new college();
       Obj.Btech();
     }
}
```

My Write is My Future

Date
Page

Packages

→ Collection of classes & Interfaces in a single unit is called a package.

- Variable
- Keyword
- function / method
- class
- Abstract class
- Interface

- Java ⇒ package → import
- C/C++ ⇒ headerfile → #include < >
- .net ⇒ namespace → using

Types of Packages

Predefined

- import java.lang.*;

→ It is a primary package. It is used for implementing basic operations for developing Java application. This package contains following classes and interfaces :-

i) System
ii) String
iii) StringBuffer
iv) Integer
v) Float
vi) Double
vii) Boolean
viii) Byte
ix) Exception
x) ArithmaticException
xi) ArrayIndexOutOfBoundsException
xii) NullPointerException
xiii) NumberFormatException
xiv) Thread
xv) Runnable
etc.

→ This package is also known as default package. Due to javac compiler automatically imports this package at a time of compilation. (Source code ⟶ Byte code)

- import java.util.*;

→ It is a utility package. It is used for utilizing the memory, system resources and other things.
This package contains following classes and Interfaces:-

- Scanner
- Set
- List
- HashSet
- SortedSet
- ArrayList
- LinkedList
- Stack
- Queue
- Date
- Calender etc.

- import java.io.*;

→ It is an input & output package. It is used for implementing input and output operation from Harddisk, from Server etc.
This package contains following classes & Interfaces:-

- File
- FileStream
- FileOutputStream
- FileReader
- FileWriter
- BufferReader
- BufferWriter
- DataInputStream
- IOException etc.

My Write is My Future

- import java.sql.*;

→ It is a structured query language Package. It is used for connecting java application with any DBMS. Such as :- MY SQL
ORACLE, MS SQL SERVER, MS ACESS, DB2, SQLITE etc.
This package contains following classes and Interfaces:-

- Connection
- DriverManager
- PreparedStatement
- Statement
- ResultSet
- SQLException etc.

⇒ Exception Handling.

Try/catch	Throws
• Inside function	• Outside function.
Syntax:-	Syntax:
Void func()	Void func() throws Exception
{	{
try()	}
{	
}	
catch(Exception e)	
{	
}	
}	

→ Exception Handling program with try/catch and finally blocks:-

```
Class expdemo
{
  void func1()
  {
    try
    {
      int var1 = 10
      int var2 = 0;
      int res = var1/var2;
      S.o.pln("try block.... Division :" +res);
    }
    catch (Exception e)
    {
      S.o.pln("catch block ... Problem occur");
      // S.o.pln(e);
      // S.o.pln(e.getMessage());
    }
    Finally
    {
      S.o.pln("Final block ... no impact");
    }
  }
void func2()
{
    S.o.pln("Second fn is working");
}
```

Date __/__/__
Page _____

```
public Static Void main (String args [])
{
    expdemo obj = new expdemo();
    obj. func1();
    obj. func2();
}
}
```