**OOPS BASIC**

* **what is java:**

-Java is high-level object orient platform independent, robust programming language developed by Sun Micro Systems in year 1995. James Gosling

-The main difference between **high-level language** and **low-level language** is that Programmers can easily understand or interpret or compile the high-level language in comparison of machine. On the other hand, machines can easily understand low-level language in comparison of human beings.

Java Features:

1) **Simple**: It is a simple programming language and it remove the complicated topic of pointer and multiple inheritance.

2) **Platform independent**: SourceCode to Bytecode

3) **Portable:** Migrating from one system to another used of platform independent

4) **Secure:** Java program never communicate directly with machine first it converted into

byte code and then converted into machine code by jvm. If the byte code contains any problem or it is

not formatted properly which is check by byte code verifier jvm dont allow that

byte code to run and raise verify Error internally byte code verifier check. hence code won't

cause any problems.

5) **OOPS:**

6)**Multithreaded:** By default execution is sequentially but if you want you can do parallel by

explicitly writing the code

7) **Robust:**(The chance of failing is very very less)

-Java is strongly typed programming language.Ex:we can not write a variable without its type.compiler

-Gc becz of this memory problem don't get.

-inbuilt exception handling

-platform independent.

* Variables

1. **Instance variable**
2. **Static variable**
3. **Local variable**

* Data types

1. **Primitive(int,float,double)**
2. **Non-Primitive(String,array,collections)**

* Operators

1. **Unitary(++pre,post++)**
2. **Arithmetic(+,-,\*,/,%)**
3. **Shift (<<,>>)**
4. **Relational(<,>,<=,>=,==,===)**
5. **Bitwise(&,|)**
6. **Logical(&&,||)**
7. **Ternary(?:)**
8. **Assignment(+=,-=,\*=,\=)**

* **System.out.println()**

-System is class present inside java.lang.pkg

-out is a static variable of type printstream present in the system class.

-println() it is a method present inside the printstream class.

**OOP’S**

* What is OOPS concept in JAVA.

-OOPs is an programming concept which work on the principle of Abstraction, Encapsulation, polymorphism and inheritance. The basic concept of oops is to Create an object and reuse that object throughout the program and manipulate this object to get the result.

* CLASS

-Class is Ntg but an entity to determine how a object can behave and what an object contain.

-In other word it is class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

* OBJECT

-Any entity that has states and behaviour is known as an object.

For example,

chair, pen, table, keyboard, bike, etc. It can be physical or logical.

EX:

-A dog is an object because it has states like colour, name, breed, etc.

as well as behaviours like wagging the tail, barking, eating, etc.

* java is not a fully OOP's lang. becz it use primitive data types.

**Constructor**

* It is a special type of method which is used to initialize the object and it does not have implicite return type.
* Every time an object is created using the new() keyword, at least one constructor is called.
* It calls a default constructor if there is no constructor available in the class. In such case, **Java compiler** provides a default constructor by default.
* Instance block execute before the constructor
* There are two types of constructors in Java: no-argument constructor, and parameterized constructor.
* The only applicable access modifier to constructor are public, private protected, default.

**Default Constructor and Parameterized**

* The access modifier to default constructor is same as that of parent class.
* This default constructor contain only one line super() which is no–arg. call to super class constructor.
* If default constructor created by user then compiler will check whether the first line will be the super() or this() otherwise compiler give it by default.
* Override and inheritance is not applicable for constructor.
* In interface we don’t have constructor becz all the variable present inside the interface are by default public static final and as we can not create the object of interface and as static variable belong to class not belong to object.
* Static block always execute before the constructor first static block and the instance block and then constructor is call.
* Every time object create instance block is execute.

**Private Constructor in JAVA**

* **The benefit of private constructor is you can not create object of that class**
* **You can not inherit class if your class contain private constructor**
* **Many time we don’t want to create instance of certain class Utility, Helper classes.**
* **If our Class contain all static member(Variable) then always go for the private constructor**
* **A utility class, that only contains static methods so in that case you use a private constructor.**
* **in java many classes have only private constructor best example**
* **java.lang.Runtime contains only the private constructor**
* **Java.lang.Class also contain only private constructor.**
* **Factory method to create an object of private constructor class.**

**Singleton Class**

* For any java **class** **if** we are allowed to create only one object then that we called as singleton **class**
* EX:Runtime,BussinessDeligate,ServiceLocator,java.lang.Class
* If several people have the same requirement then it is not recommended to create a separate object **for** every requirement same object can be reused whenever similar requirement due to that memory utilization as well as performance improved

**THIS KEYWORD**

* This keyword is reference variable that refer the current object.
* We cannot use this is static method.
* This() is used for constructor chaining
* this can be used to refer current class instance variable.
* this can be used to invoke current class method (implicitly)
* this() can be used to invoke current class constructor.(this will be first statement

in the constructor its compulsory).

* this can be passed as an argument in the method call.
* this can be passed as argument in the constructor call.

* this can be used to return the current class instance from the method.

**SUPER KEYWORD**

* Super keyword is reference variable its meant for to refer immediate parent class Object
* super can be used to refer immediate parent class instance variable.
* super can be used to invoke immediate parent class method.
* super() can be used to invoke immediate parent class constructor.
* SUPER IS ALWAYS PROVIDED BY JVM AS A FIRST STATEMENT IN CASE OF DEFAULT CONSTRUCTOR IF YOU FORGOT TO GIVE Jvm AUTOMATICALLY GAVE .

**ACCESS MODIFIER**

-There are **12** Access modifier in java

-**5** are applicable to outer level/top level class

-**7** are applicable to inner class

-**2** only variable applicable modifier are volatile,transient.

-2 only method specific modifier are synchronized and native

-outer level modifiers

**1)public**

**2)<Default>**

**3)final**

**4)abstract**

**5)Strictfp**

* PKGS-A java package is a group of similar types of classes,interfaces and sub-packages.
* There are two types of pkgs in java

1)Build in pkg(lang,sql,util,awt etc)

2)User defined pkg

* Advantage of Java Package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision.

Ex how we can access pkg from another class

package a

class Test1

{

public void m1()

{

s.o.p("HII");

}

}

i want to access pkg a m1 method from test 2 class

Package B

import a.\*;

class Test2

{

public static void main(String[] args)

{

Test1 t1=new Test1();

t1.m1();//o/p HII

}

}

1) **Public**: If a class is public then we can access this class from anywhere within the pkg and outside the pkg also.

2**) <Default>**:We can access this class anywhere inside the same pkg but not outside this pkg.

3) **Final:** final FOLDER IN OOPS CONCEPT

4) **Abstract**: It is used to show only the functionality and hide the internal implementation.

* abstract is applicable to class and method
* if any method is abstract even if we don't know the internal implementation still we can declare a method as abstract child class is responsible to provide the implementation if child class extending this class

* If a class is abstract that mean its partially implemented class which does not have full concrete methods into it.if we make class abstract so we can not create the object of that class. Becz if we can create the object as

this abstract class have partially implemented method which are not concrete then we get some problem if any class trying to access this methods.

* If a class doesn’t contain any abstract method still we can make that class as

a abstract such class we called as Adapter class Ex:http servlet is abstract class which doesn’t contain any abstract method inside.

**@Adapter class:**

-A class which doesn’t contain any abstract method inside it that we called adapter class.

**@@@ What is adapter class?**

-An adapter class provides the default implementation of all methods in an event listener interface. Adapter classes are very useful when you want to process only few of the events that are handled by a particular event listener interface. You can define a new class by extending one of the adapter classes and implement only those events relevant to you.WindowAdapter-provide implementation to WindowListener interface.

**5)Strictfp**:

-It is applicable to class as well as the method and interface but not with static method

-It is introduced in 1.2 v

-If you want the result in floating point arithmetic which is platform dependent but we want independent so we used strictfp.

EX:10.0/3

in 16 bit processor we get 3.3333 for time only

but in case of 32 it processor we get 3.333333

6 time

so if we want every thing same use strictfp which follow IEEE stds.

**2) Inner level access Modifiers**

**1) Private**: The private access modifier is applicable to the method,variable as well and constructor only not to the class

-if your method is private that means you can not access this method outside this class.

-private constructor we can not create the object but by using the factory method we can do this.

**2)Protected:** This applies to the method, variable, and constructor.

-you can access within the same pkg as well as in diff.pkg but only in child class.

EX: if you have two pkg. The first one will be

Package p1;

Public class A

{

Protected void m1()

{

S.O.P(“HII”);

}

}

-Here we have second pkg

Package p2;

Import A.\*;

Public class B extends A

{

B b=new B()//we can only use chid reference only

b.m1();

}

**3) Static:** Static FOLDER IN OOPS CONCEPT

**STATIC KEYWORD**

-Static keyword is used for memory management

-The static method cannot use non static data member or call non-static method directly.

-This and super cannot be used in static context.

-In case of method hiding if you creating object with the help of parent reference still we get the parent object not child object becz of method hiding of static belong to class only.

**Inner Classes**

-Java inner class or nested class is a class which is declared inside the class or interface.

-We use inner classes to logically group classes and interfaces in one place so that it can be more readable and maintainable.

-it can access all the members of outer class including private data members and methods.

There are two types of nested classes

**1)non-static nested class(Inner class)**

**i)Member inner class**

**ii)Anonymous inner class**

**iii)local inner class**

**2)Static nested class**

**1)Member inner class**

-A non-static class that is created inside a class but outside a method is called member inner class.

IMP

-how to call inner class:

Create outerclass object first;

outer\_class\_name.inner\_class\_name object = outer\_class\_object.new inner\_class\_name();

**2)Anonymous inner class**

-A class that have no name is known as anonymous inner class in java.

-This class declare inside the main and when you create interface or abstract class object reference before closing semicolon open bracket this is anonymous class.

-We can also create outside the main method as well but when you have to call it you have to create main class object .then outer class object .method name

-**Without extending the abstract class we can create** -It should be used if you have to override method of class or interface. Java Anonymous inner class can be created

by two ways:

i)using class

ii)Interface

IMP

-How to call inner anonymous class

By crating abstract or interface object then { inside initialize} and then call abstract method object.method

**3)Local inner Class**

-a class which is declared inside class inside the method of that class

EX:

class A

{

void m1()

{

class B//local inner class

{

Void m2();

}

B b=new B();

b.m2();

}

}

A a=new A(); a.m1();

**1)Static nested class**

-A static class i.e. created inside a class or interface is called static nested class in java. It cannot access non-static data members and methods.

It can be accessed by outer class name.

IMP

how to call instance method inside static nested class

outer\_class\_name.inner\_class\_name object=new outer\_class\_name.inner\_class name;

inner\_class\_object.method\_name();

how to call static method inside static nested class

outer\_class\_name.Inner\_class\_name.Method\_name();

**2)Nested Interface**:

-A interface which is declared inside another interface or another class.

IMP point

-Nested interface must be public if it is declared inside the interface but it can have any access modifier if declared within the class.

-Nested interfaces are declared static implicitly.

-if u only accesing inner interface then we don’t need to provide the implementation for outer interface method.

VIMP

-we can create a class inside the interface also that we treat as an a static nested class

-inside class also we have interface we can call by doing upcasting with outer.inner =new currentclassname();

-a interface can only extend another

interface not class and interface cant implements another interface

**SERALIZATION**

* The process of saving state of an object to a file that we called it as serialization (**marshalling)**.
* OR Strictly speaking
* It is the process of converting an object from normal java supported form to file supported or network supported form
* The process of reading the state of object from file is called deserialization
* How we can write data into file(Serialization)

1)First you have to create the file output stream(fos) object it will save your byte stream into file but no object directly.

2)for converting your object into byte stream we have to use object output stream(oos).this oos contain the write object method which convert object into byte stream.

Actually oos take object and convert into binary data .fos take binary data and write into file.

* How to read data from file**(Deserialization)**

1) for reading data from file we required file Input Stream(FIS).

2) After reading you want data in the form of object then we use OIS which has readObject() method to get the object from file and it will return object so create object of class and do typecasting with that class.

**Externalization:**

* In serialization every thing takes care by jvm and programmer doesn’t have any control.
* In serialization it is always possible to save total object to the file and it is not possible to save part of the object, which may create performance problems.
* To overcome this problem we should go for externalization.
* The main advantage of externalization over serialization is every thing is take care by programmer and jvm doesn’t any control.
* Based on our requirement we can save wither total object part of object, which improves performance of system.
* Externalizable interface is not a marker interface it contain 2 methods
* **1)writeExternal()**
* **2)readExternal()**

-transient keyword is not required in case of externalization concept if you don’t want to save don’t mention the variable in writeExternal();

Marker Interface: If a interface doesn’t contain any method or filed and by implementing this interface our object get some special ability that we called as marker interface.

-actually this marker interface act as a tag which tell the information to compiler to add special behaviour to class who implementing it.

EX: serializable, cloneable, Random Access.

By implementing this serializable interface our object can save into file or travel across the n/w.it is present in java.io.pkg.

- When an object implements a marker interface, it indicates to the runtime environment or framework that the object should be treated in a certain way or that it possesses specific characteristics. The runtime environment or framework can then use reflection or other mechanisms to determine if an object implements a particular marker interface and apply the corresponding behavior.

**4) Transient:** transient is a variables modifier used in serialization. At the time of serialization, if we don’t want to save value of particular variable in a file for security constraints then we use transient keyword. When JVM comes across transient keyword, it ignores original value of the variable and save default value of that variable data type.

-serialization is all about to save state of an object to a file.

-if you apply static variable as .transient there is no impact at all.

-final transient

// A sample class that uses transient keyword to

// skip their serialization.

class Test implements Serializable

{

// Making password transient for security

private transient String password;

// Making age transient as age is auto-

// computable from DOB and current date.

transient int age;

// serialize other fields

private String username, email;

Date dob;

// other code

}

**5) Native**: The method which are implemented at different language.

**6) Volatile:** This is applicable only to variable

-If the value of variable is keep on changing by multiple thread then there may be chance of data inconsistency problem(not thread safe) we can solve this problem by using volatile keyword.

-if variable declared as volatile then for every thread jvm will create a separate local copy.

-every modification performed by the thread will take place in local copy so there is no effect on copy so reaming thread don’t get affected and before the termination of thread the local finally updated values is updated to master copy.

Disadvantage: creating and maintaining separate copy for every thread increase complexity and reduces performance.

7)**Synchronized**: further chapter

**OOPS Four Pillars**

* **Encapsulation**

-It is a mechanism of binding or wrapping data member and member method in a single unit called encapsulation.

-Encapsulation is a combination of data hiding and abstraction.

-The basic thing you have to do to make the class bean or pojo is to make the variable private and make the getter and setter for every variable.

-Hence If you would have understood till now you can say that abstraction is just a subset of ENCAPSULATION.i.e.Every entity that performs abstraction is encapsulated internally but every thing that shows encapsulation need not be abstraction always.

-In General words,Abstraction is Just Hiding the complex things behind a particular Procedure to make the procedure look simple.  
Example:**Monitor ON/OFF**::--The user doesn't need to know much about all the chips functioning that happens when Monitor is switched ON or OFF..All he needs to know is On Function ON-Monitor is On and on function OFF-Monitor is off...

-Example: We can consider The HR in a company as a person who works on the principle of Encapsulation. i.e. we cannot talk to other departments directly we need to communicate through them through HR. This ensures security and better maintenance of the company's records.

**-Diff between Pojo and Bean is**

-Require private data member (variable)

-for every variable separate getter and setter.

-make default no argument constructor

-but diff is Bean class can be Serializable but pojo cannot serialize.

-Advantage of Encapsulation

1) Security

2) Control over data

3) Easy to teste using unit testing

4) Read only write only

**-Disadvantage of Encapsulation**

1) Length of code increases becz of separate getter and setter for each variable.

2) Performance is reduce becz every time we have to do validation

* Inheritance

1. Is-A Relation
2. Has-A Relation

Inheritance: It is a mechanisms of acquiring all the properties and behaviour of parent class.

Except (constructor and private method);

-when we use inheritance classes are tightly couple that means if we trying to perform modification it will affect the extending classes also

**Q Why to do inheritance**

1) You can achieve runtime polymorphism/method overriding.

2)Code reusability going to achieve.

3) Cost cutting

4) Reduce redundancy

Types Of Inheritance

1)Single inheritance

2)Multilevel inheritance

3)Multiple inheritance

4)Hierarchical

5)hybrid

How many types of relationships in between classes

**1)IS-A relationship(Inheritance)**

**2)Has-A relationship(association)**

**i) Aggregation**

**ii) Composition**

1)Is a relationship

The main disadvantage is make the classes tightly coupled

EX:

class Vehicle

{

}

class Car extends Vehicle

{

This is a is a relation ship

car is a vehicle

}

**2) Association (Has-A relationship)**

-Association is relation between two separate classes which establishes through their Objects. Association can be one-to-one, one-to-many, many-to-one, many-to-many.

In Object-Oriented programming, an Object communicates to other Object to use functionality and services provided by that object.

- Composition and Aggregation are the two forms of association.

-If a class have an entity reference, it is known as Aggregation.

Aggregation represents HAS-A relationship.

-It is very commonly used relationship

-association can achieve directly as well as through new keyword.

Because of has a relationship you can access the method which is required for your project not all property so because of this there is no tightly coupled.

EX:

class student

{

String name;

int roll\_no;

direct has a relationship

student has a name

student has a rollNO

}

Another example

class Engine

{

}

class car

{

Engine e=new Engine();//has a relation as car has a engine

//if we use extend keyword then we achieve inheritance

but here we achieve has a relationship so we don't get tightly coupling

only few method we can access based on project not all like inheritance

}

//in case of inheritance blood relation is created but in case of has a Relationship blood relation is not created

**-types of association**

-if there is weak bonding between two classes then we called it as Aggregation.

-If there is strong bonding between two classes that we called it as a composition

EX:

Suppose we have class car and that class we have a subclass is music player and engine.

-music player with car has weak bonding because a car can have music player

but its not compulsory a car without music player also valid And music player without car is also valid.(Aggregation)

-engine with car has strong bonding without engine car can not drive and without engine what engine can do .(Composition)

-If we don't have container object still we contained object it is aggregation

-if we don't have contained object and because of this the exsistent of container object is gone that is composition.

EX:

we have college class an two subclass is professor and branches.

if we don't have college still we can have professor but we don't have college then we don't

have branches.

when to use aggregation and inheritance

-when the relation is lifelong go for inheritance

-otherwise association is the best choice when we have to use the code readability is going to achieve.

* Polymorphism:

-To perform a particular task you have multiple way to do so

1. Compile time(Overloading)
2. Runtime(Overriding)

1) Method overloading is compile time polymorphism.

2) Method overriding is run time polymorphism

-If a class has multiple methods having same name but different in parameters, it is known as METHOD OVERLOADING.

-If we have to perform only one operation, having same name of the methods increases the READABILITY of the program.

There are two ways to overload the method in java

**1)By changing number of arguments.**

**2)By changing the data type.**

-In java we can not perform method overloading just by changing the return

type from 1.5v onward.

-TYPEPRAMOTION

-if one is integer and one is double then the integer type promoted into double.

-but if we have two integer and two double then type promotion not happen .

\*\*\*\*VVIMP

-runtime polymorphism you can’t achieve with the variables;

EX:

class Animal

{

String s="LION";

}

class Dog extends Animal

{

String s="Dog";

}

class Test

{

public static void main(String[]args)

{

Animal a=new Dog();//upcasting

System.out.println(a.s);//LION not dog

}

}

-becz with the help of parent class object reference we can hold child object but we can not access the child methods. But if parent class having same method that is override in child class and if you perform upcasting then child class object is print.

-we can not reduce the scope of the method.

-if we have static method in parent class and same static method in child class then we can override that method but this is not method overriding its method hiding.

-if parent class static method and child class also override that method then if you do upcasting then parent class o/p we get not child this is method hiding not overriding

**Abstraction**

1. Abstraction
2. Interface

Abstraction: abstraction is a process of hiding the internal implementation details from the user, and only show the functionality to the user.

i.e user only know what object do how it is doing user dont know.

-we can not create the object of abstract class but in case of anonymous inner class we can using the reference of class to override the method of abstract class without extending or implementing class.

-we can not crate object becz its partially implemented class which having the partially method which has no complete implementation like concreate method so that’s why jvm dont allows us to create the object;

-static method is also present inside abstract method.

Advantage:

- The advantage of using an abstract class is that you can group several related classes together as siblings. Grouping classes together is important in keeping a program organized and understandable.

-Abstract classes enable code reusability.

-Abstract class in Java is highly beneficial in writing shorter codes

-Abstraction in Java avoids code duplication.

-Changes to internal code implementation are done without affecting classes

Q. **why abstract class have constructor as we dont create the object.**

ANS-to initialize the instance variable and constructor chaining is always happens.

**INTERFACE:**

#interface is used to achieve the full abstraction

#If you dont know implementation only know the service requirement specification then we should go for interface.

# Interface is mediator between the subclass and user class

-interface tell subclass what to implements and interface tell user class what to call.

#You can not create object for interface.

# IMP

# you can not provide instance variable inside interface becz by default all the variable inside interface are public static final.

@ it is used to achieve the multiple inheritance

# it has loose coupling(if you do change in one it will not get change in another).

# **Diamond problem solution** is only available in the interface.

# interface can not extends any class becz class contain concrete method.

#interface can extends another interface but not implement

# Every method inside the interface will be have public Abstract Method();

**TILL 1.7 VERSION**

# Every field( int field) is public static final by default. becz those who implementing the interface not affected by any change made to this variable they make it is as final.

-you have to declare value to variable at the declaration time only becz they are public static final .

Q**.WHY INTERFACE CONTAIN VARIBLES AS THEY ARE FINAL WHAT IS THERE USE?**

ANS-interface contain variable for requirement level constant.

suppose you have an interface which having variable pi=3.14

and you are implementing this interface in calculator class so when ever

you are doing any PI calculation interface variable

provide you constant

**# advantage of java version 8**

we can also create the concrete method inside the interface by default keyword

-becz of this default method we does not require adapter class anymore

-but we have to explicitly provide the default keyword this method is introduce to provide

the dummy implementation so that it is not necessary for implementation class to implement it.

- we can also use static concrete method can also used but must be public access modifier.

-we can not have static method by default we have to call this method through interface name. Method name.

-this static method is introduce to provide security as it is not override in child implemented class and it is mainly use to provide the static utility method for all class.

# in java 9 version you can create the Private concrete method inside the interface and private static method

-one question arrives to your mind private method you cannot access outside the class then how we can use then you can call this private

-method inside default method so we can call outside the class.

-as well as inside static method also but you have to make that method static

-These private methods will improve code re-usability inside interfaces and will provide choice to expose only our intended methods implementations to users. These methods are only accessible within

That interface only and cannot be accessed or

Inherited from an interface to another interface or class.

-Static block inside the interface is not allowed and Instance variable is also not allowed

-from java 1.8 version main method can be inside the interface.

WHEN SHOLD WE GO FOR INTERFACS VS ABSTRACT VS CONCREATE CLASS

-When we don't know anything about the implementation but you know service requirement specification then we should go for Interface.

-When we are taking about implementation but not completely (Partially) then we should go for Abstract class.

ex- Generic servlet,HttpServlet

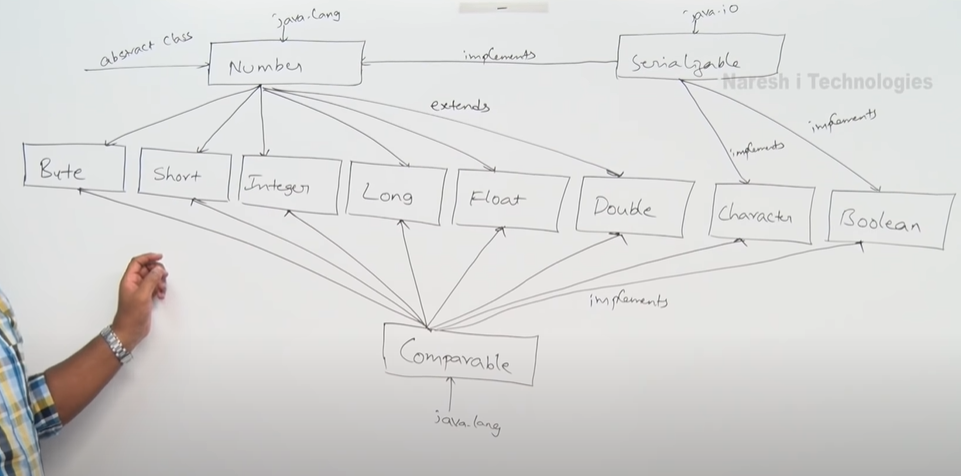
-If you know implementation completely and ready to provide the service then you should go for concrete method.

ex our own servlet

-If everything is abstract then always go by interface becz in interface we will get multiple inheritance and you can extends other class also at a same time.

**WRAPPER CLASS**

-This Wrapper class is use to convert the primitive datatype into wrapper objects and this object is required in collection class for that we use the java.lang.pkg class wrapper class



-To convert the primitive into wrapper object we have to perform autoboxing.

-AUTOBOXING :The process of binding primitive data type into wrapper class object is called binding.

-Unboxing: The process of taking primitive datatype out of wrapper object called as UnBoxing.

-when ever we have to perform boxing we have to use the constructor present inside the wrapper classes or we can use the valueOf() method also.

-Except float and char all the warapper classes has 2 constructor.

-This 2 constructor out of which

1)one constructor will bind its own data into object.

2)sec. constructor will bind its own data in the form of string.

-When you think about float it has three constructor first 2 is same but third is to bind double data.

-Char has only one constructor to bind its own data.

-When ever you have to perform the unboxing we have to use 8 method i.e. datatypename.value()

@intValue()