**OOPS**

oops are building blocks for any programming

oops concets ------------

**1.Encapsulation**

**2.Abstraction**

* Abstract class
* Interface

**3.Inheritance**

**4.Polymorphism**

* compile time polymorphism (method-over-loading)

* run time polymorphism (method over-riding)

**oops are implemented based on DRY principle**

**DRY stands for don't repeat yourself**

advantages

\*\*\*\*\*\*\*\*\*\*

re-usability

reduce redundancy

security

**Encapusulation**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*

binding variables and methods into single unit is nothing but

encapsulation

wrapping of data(variables) and associated methods into single unit is

called as Encapusulation

class is example for encapusulation

Encapsulation main moto is to avhieve **security**

we can achieve security by **declaring variables as private**

whenever we will declare variables as private then we can access

private variables with in class only

don't direct allow one class variables to another class for modication

private

\*\*\*\*\*\*\*

'private' is access mofidier which is used at variable level or method

level

visibility for private variables is with in scope only

we can't access private variables out-side class

pojo class is eample for Encapsulation

declare variables as private inside pojo class

**setters are used to set the value to class variables**

**getters are used to get the value from property**

**Inheritance**

**\*\*\*\*\*\*\*\*\*\*\***

The process of deriving new class from existing class is called as

inheritance

new class-->child class

existing class-->parent class

**inheritance is also called as 'IS-A' relationship**

child class is called as sub class or derived class

parent class is called as super class/root class

project performance get down if duplicate code increases

remove duplicate code from multiple class,design as super class and

use where ever required

inheritance is used for to achieve re-usability

**we can use 'extends' keyword to achieve inheritance in java**

**Inheritance is divided into below categories o**

1. single level inheritance
2. multiple inheritance
3. multi level inheritance
4. hybrid inheritance
5. hierarchical inheritance

**one class extend only one class at a time**.

**this()-->**calling current class default constructor

**this(v1,v2)-->**calling current class parameterized constructor

**super()-->**calling super class default constructor

**super(v1,v2)-->**calling super class parameterized constructor

**Abstraction**

\*\*\*\*\*\*\*\*\*\*\*

The process of showing essential features to end-user by hiding

internal details is called as Abstraction

we will hide the implementation by using Abstraction

We will show project functionality to end-user by hiding

tools,technology and coding part

System.out.println("Welcome To java");

shows some response to end-user by hiding internal

details

we can achieve abstraction in 2 ways

by using abstract class

by using interface

suppose we want to achieve full abstraction then go for 'interface'

suppose we want to achieve partial abstraction then go for 'abstract

class'

what is difference between Encapsulation and Abstraction?

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

we can hide the data by using Encapsulation where as we can hide the

implementation by using Abstract class

Abstract class

\*\*\*\*\*\*\*\*\*\*\*\*\*\*

The class which is declared with 'abstract' keyword is called as abstract

class

abstract class contains both complete as well as incomplete methods

complete method

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

The method with body is called as complete method

public void disp()

{

body

}

incomplete method

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

incomplete method is called as abstract method

The method without body is called as incomplete method

we will define incomplete method **with 'abstract' keyword**

**Abstract class**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\***

suppose we want to achieve partial abstraction then go for abstract

class

The class which is declared with 'abstract' keyword is called as abstract

class

abstract class contains only complete methods or only incomplete

methods or both complete and incomplete methods

designing sub class is mandatory for abstract class

designing abstract class without sub class is meaning less

we will provide 'IS-A' relationship between sub class and abstract class

Abstarct class contains instance variables

Abstarct class allow constructor to initialize the instance variables

abstract class constructor get executed whenever we will create an

object to sub class

we can't achieve multiple inheritance aby using abstract class

complete method vs incomplete method

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

method with body is called as complete method or concrete method

method without body is called as incomplete method or abstract

method

we can define abstract method with keyword 'abstract'

what is difference between concrete class and abstract class?

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*

concrete class contains complete methods where as abstract class

contains both complete and incomplete methods

we can create an object for concrete class where as we can't create an

object to abstract class

sub class is not manadatory for concrete class where as sub class is

mandatory for abstract class

Interface

\*\*\*\*\*\*\*\*\*

suppose we want to achieve full abstraction then go for 'interface'

we will define interface by using 'interface' keyword

syntax

\*\*\*\*\*\*\*

interface <interface-name>

{

constants

incomplete methods

}

interface allow constants and incomplete methods before java8

interface allow default methods and static methods from java8 onwards

interface doesn't allow instance variables

Interface doesn't allow constructor due to there is no instance variables

concept

interface Cusomer

{

int cno=10;

public void disp();

}

suppose developer forget to define 'abstract' keyword before incomplete

method then compiler will add 'public abstract' keywords before the

method

suppsoe developer forget 'public static final' keywords before the

variable then compiler will add 'public static final' before variable

designing interface without implementation class is use-less

designing implementation class for interface is mandatory

we will provide 'IS-A' relationship between implemenattion class and

interface

we will use 'implements' keyword between implemenatation class and interface

can we create an object to interface?

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

we can't create an object to interface due to there is no complete

methods concept

can we design constructor as part of interface?

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

interface doesn't allow constructor due to there is no instance variables

what is difference between Abstract class and interface?

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

we will define abstract class with 'abstract' keyword where as we will

define interface with 'interface' keyword

abstract class contains complete methods as well as incomplete

methods where as interface contains only incomplete methods

Abstarct class allow instance variables where as interface doesn't allow

instance variables

Abstract class allow constructor where as Interface doesn't allow

constructor

we can't achieve multiple inheritance through abstract class where as

we can achieve multiple inheritance through interface.