Date: Page No.:

OBJECT ORIENTED PROGRAMMING (OOP)

→ Oops is a Mechanism to solve real time problems with the help of class & object. The 4 Pillars of an Object Oriented language is:

1. Encapsulation
2. Inheritence
3. Polymorphism
4. Abstraction

There are two more pillars
- class
- Object.

Q. Is Java 100% of object Oriented language?

→ Java is not 100% Object Oriented language because of:

(i) Primitive datatype: Java Support's primitive type data which is not an Object type data so Java is not a 100% OOL.

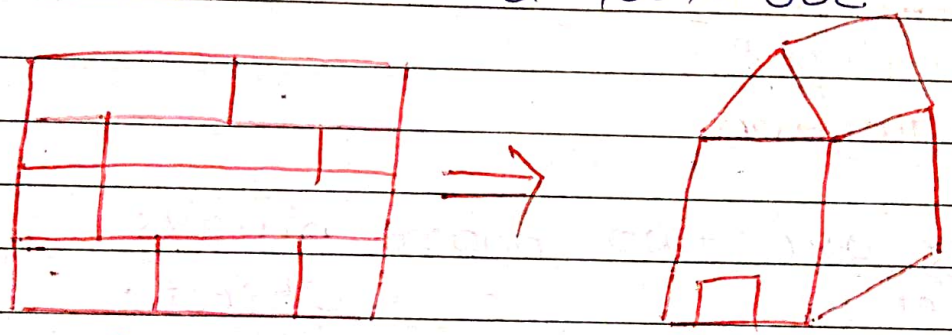
(ii) Static keyword:
Java Supports static type members with the help of static type keyword A static member can we

Date: Page No.:

accessed only by using class name by request so this make java not a 100% OOL

iii) Wrapping class :

A 100% should not support wrapper class but Java provides wrapper class which makes it not a 100% OOL



Blue print class

House Object.

* CLASS :- It is a blueprint or master copy or propotype from which an object is created.
A class is a logical entity which doesn't occupy space in memory
A class in Java is non-primitive type data by using one class Multiple Object can be created.

Date: Page No.:

→ **Object** :- It is a physical implementation of a class project is a physical entity which will occupy space in memory object will have all the details defined by the class Object is created at run time dynamically inside a special area called heap area.

Characteristics of class

- It have two type of characteristics.
- State
- Behaviours.

State : These are the details or attribute of the object. The state of a class for static or non-static variable of class

Eg :- name, price, empId, sal, age.

Behaviours :-

These are the action performed by the Object. It forms static or non-static Methods of the class location performed

eg:- start, stop, accelarate, drive, work, etc.

Q. Design a class/blueprint for employ.

→ Class employ
{
String names;
int id;
double salary;
String address;
} States.

Public void work(){
sop("employ is working");

Public void eat(){
sop("employ is eating");
}
}

→ Object creation in java:

Object is created in java by using new keyword and constructor of the class. inside Object

- Name of constructor must be same as the class.

Date: Page No.:

Object is created at runtime dinamically inside heap area.

→ New() keyword:

New is a keyword which is responsible to create an empty object inside heap-area.

whenever "New()" key word is used then memory allocation takes places for object inside heap area.

→ Constructor :-

constructor is special member of class which is used to load and initialized all the non-static member of the class inside Object.

- Name of constructor must be same as the class.

→ Reference type data:-

The data which reference to an object is called reference type data.

- A non-primitive data always reference to an object so it is also called reference type data.

* The variable which reference to an object called reference type variable.

Note :-
The default value of reference type data is "Null"

The actual value for the reference type data is an "Object".

← default value

```
Employ e1 = null;

Employ e2 = new Employ();
```

↑ Actual value.

→ Static member :-

If a member (variable, Methods) is declared by using static keyword then such member is called a static member.

- A static member is loaded inside class static area and shared with all the object in heap area.

Date: Page No.:

- A static member can be access inside same class directly or by using class reference.

  Inside other class a static member can be accessed by using Object reference.

  Inside other class a static member can be accessed by using Object reference.

→ Non-static member:

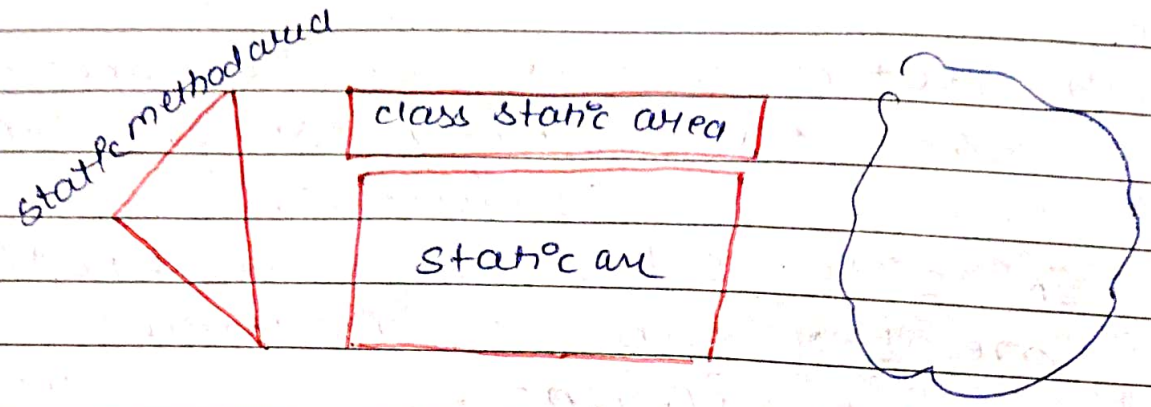
If a member (variable, method) is not declared by using static keyword then such member is called non-static members.

- A non-static member is loaded inside object only if when object gets created

- A static member will be same for every object but a non-static member will be different for different object

→ Memory structure:

when ever programm start execution then some memory block is created by JVM



→ **Static method area :**

When program start execution then all the static method are loaded inside static method at very first place.

→ **Class static area :**

In this area all the different type of static member such as method variable and blocks are loaded

→ **Stack area :**

Ones class static area is loaded then stack area is created where main starts its execution inside the method frame.

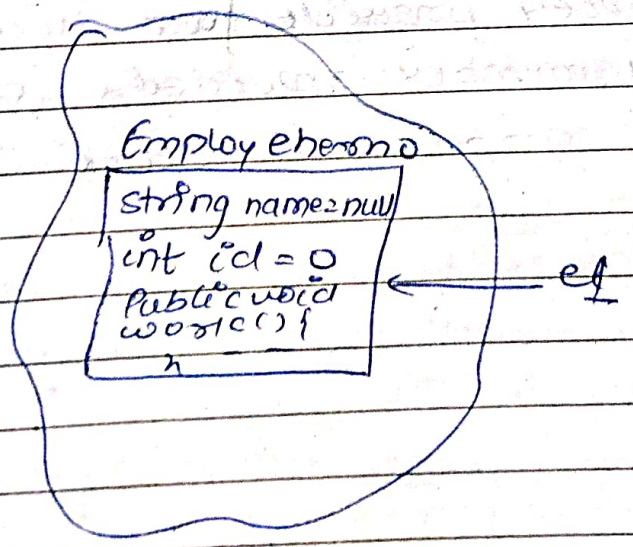
Date: Page No.:

→ Heap area:

Heap area is a located for object creation object gets created and all the non-static members are loaded inside object.

```
class driver
{
    Public static void main (String[] args)
    {
        Employ e1 = new Empby ();
```

- Employ → non-primitive type data OR Reference type data
- e1 → variable name or object reference or reference variable
- new Empby () → Object or instance or Empty class.
- It is used to create object
- It loads all non-static members of the class inside object.



Heap Area.

Date: Page No.:

→ Difference b/w stack area and heap area.

(I). Main method gets executed inside stack area but object is created at runtime inside heap area.

(II) Inside stack area their will be local variable and object reference but non-static members are loaded inside object inside heap area.

(III) stack area is fixed in size so, stack overflow error occurs inside stack area but heap area is growable so out of memory error occurs in heap area

(IV) Stack area member provide fast execution but heap area member provides slow execution as compare to stack area member.

Date: Page No.:

* * * *

→ 4 important point about static or non-static members.

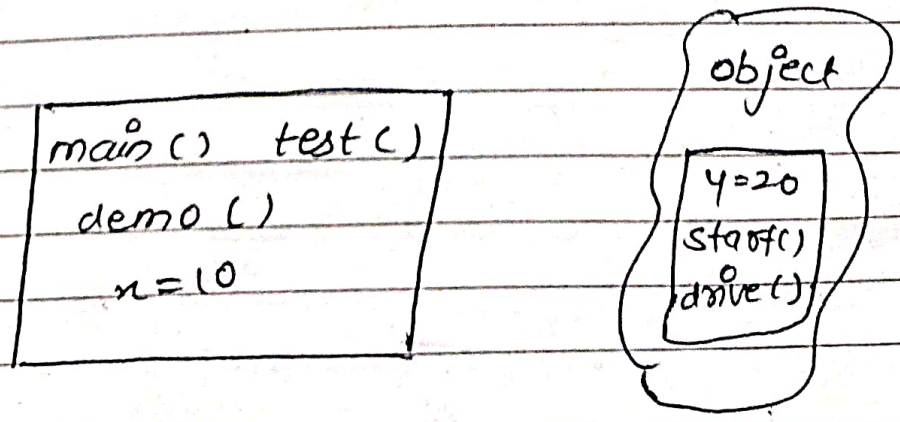
(i) All static members are loaded at one place inside class static area so a static method can access any other static members directly.

(ii) All the non-static members are loaded at one place inside object so a non-static method can directly access any other non-static member

(iii) The details of static members will be shared with objects so a non-static method can acess any static member directly

(iv) Any details of non-static members are not shared in class static area with static members so a static method cannot access and non-static method directly.

A static method can access any non-static member only throw object reference.



Date: Page No.:

* Variable Hiding :-

If a local variable and the globle variable [Static or non-static] will be exist with the same name then perefrence is given to the local variable and global variable get's hiding which is called variable hiding or Shadowing

1) Static variable shadowing :- class first{
static in

code --

② Initialization of an object :- Object can be initialized. in 3 ways reference

(i) By using Object reference
(ii) By using non-static Method
(iii) By using constructor.

Date: Page No.:

→ Initialization by object reference:
when an object is created then all the details of object are stored on object reference to initialized the object

Figure + code

→ "This" keyword :- "This" is a keyword which is used to reference non-static methods of the same class.

"This" keyword holds the reference of current object under execution.

Date: Page No.:

→ **Constructor:**

Constructor is special member of class which is use to load and initialized non-static member of a class inside the object.

→ difference b/w a method and constructor.

(i) A method can have any name but the name of constructor same as the class name.

** (ii) A method will have a return type but a constructor will not have return statement

(iii) A method can be static, final, abstract or synchronised but these keywords are not allowed for constructor so a constructor can never be static, final, abstract or synchronised.

(iv) A method is envoked by using method call but a constructor is envoked at the time of object creation.

(v) A method is used to perform general purpose task but a constructor is used only for initialization of object.

Date: Page No.:

```
class Fruit
{
 Public Fruit
{
 sop("This is constructor");
}
Public static void test()
 {
    sop("This is test method");
}
Public void demo()
 {
    sop("This is demo method");
 }
```

```
class FruitDriver
{
Public static void main
    (string[] args) {
 Fruit f1 = new Fruit();
  Fruit test();
  f1.demo();
}
}
```

→ Type of constructor

(I) Default constructor

(II) user define constructor → No argument constructor
→ Paramitize constructor

(I) Default constructor :

A compiler add constructor in class file at the time of compilation is called default constructor.

→ a default constructor is add only if . java file will not have any user define

Date: Page No.:

constructor if .java file will ~~xxx~~ not have any user defined constructor if java file will have any user defined constructor then default constructor will not be added by compiler

→ Use of default constructor :

① default constructor help in object creation even in case their is no user define constructor available in .java file.

→ User defined constructor :

A constructor which is designed by programmer is called user define constructor

1) No - Argument constructor :

If a user define constructor does not take any argument at its parameter then it is called no argument constructor.

Date: Page No.:

e.g →

```
class car
{
    car()
    {
        sop("car-object is created")
    }
}

class Driver
{ Public static void main(String[] args){
    car c1 = new car();
    car c2 = new car();
    car c3 = new car();
}
}
```

c1 → car() { sop( ) }

c2 → car() { sop( ) }

c3 → car() { sop( ) }

O/P

car-object is created
car-object is created
car-object is created

Javac Driver.Java
Java Driver.

Q Design a program to have object counter.

```
class Employ
    static int count = 0
    Employ() {
        count++;
        sop("object" + count + " is created");
    }
}
```

Date: Page No.:

→ Parametriz constructor :

If a constructor take different argument at its parameter then it is called parametriz constructor.

A parametriz constructor is used to initialized the instance member at the time of object creation.

→ Class Empty {
String name;
int id; double salary;
Public Employ (String name, int id, double salary).
{ this.name = name;
this.id = id;
this.salary = salary; }
Public void get details() {
sop("Name is " + Name);
sop("Id is " + Id);
sop("Salary is " + salary);

→ Class Driver.
Public static void main
(String[] args) {
Employ e1 = new Employ("Mohan",
121, 234567);
e1.get details();
Employ e2 = new Employ("Sohan",
122, 234787).

Javac Driver.Java
Java Driver.
Employ@ —

String name = null
int id = 0
double sal = 0.0
Employ(String name,
int id, double sal)
this.name = name;
this.id = id;
this.sal = salary;
Public void get details
}

Date: Page No.:

→ constructor overloading: It is mechanism to have multiple constructor inside a class with multiple argument

- we can achive constructor by 3 ways

① By changing no. of arguments.

Eg:

```
Employ (string name){
    this.name = name;
}
Employ (string name, int id){
    this.name = name;
    this.id = id;
}
```

(ii) By changing the datatype.

Eg

```
Employ (string name){
    this.name = name;
}
Employ (int id){
    this.id = id;
}
```

(iii) By changing the sequence of data type.

Eg.:

```
Employ (string name, int id)
{ this.name = name;
  this.id = id; }
Employ (int id, string name){
    this.id = id;
    this.name = name; }
```

Date: Page No.:

→ Advantage of constructor Overloading.

- By the help of constructor overloading we can ini°u°zed the object as per our requirement

- we can decide which is intiu°zed or not intiu°zed.

Date: Page No.:

* Constructor chaining :-

It is the process to connect or chain one constructor to another constructor

constructor chaining is achieve in two ways :-

(I) By using this () statement
(II) By using super() statement.

This call statement :-

It is a statement used in java used to call one constructor from another constructor inside the same class.

A constructor call must be 1st statement of constructor so this call must be first statement inside constructor

→

```
class car {
  car() {
    this (12.8);
    sop("car no arg const"); }
  car (int x) {
    this ();
    sop("car int-arug const"); }
  car (double x) {
    sop("car double arg const"); }
}
```

```
class driver {
  Psvm (s[]args){
    car c1 = new car();
  }
```

O/P

car double=argconst
car no-arg const
car int-arg const

Date: Page No.:

Syntax:

```
this (var_name1, var_name2 ---);
```

→

```
class Employ {
  String name;
  int id;
  String address;
  Employ () {
    Sop("Employ ob is created");
  }
  Employ(String name) {
    this();
    this.name = name }
  Employ (String name, int id) {
    this (name)
    this.id = id; }
  Employ (String name, int id, double salary) {
    this.(name, id);
    this.salary = salary, 27456,7;
Employ (String name, int id, double salary, String add)
    this (name, id, salary)
    this.add = add.

class Driver {
  Psvm () {
    Employ e1 = new employ("mohan", 238);
    Employ e2 = new employ("sohan", 300);
    Employ e3 = new employ("Rohan", e11, 23456,
                "Noida"); }}
```

Date: Page No.:

* Advantage of constructor chaining:-
constructor chaining helps to decrease length of code by eliminating line of code.

Copy constructor:

- copy constructor is a special type of constructor which is used to copy details of one objects for the current objects

```
Public class fruit {
    string name;
    int weight;
    fruit ();
    }
```

Date: Page No.:

Relationships :-

Connection b/w two classes or b/w two objects are called relationship.

Relationship are two-type

(i) Is-A relationship / Inheritence
(ii) HAS-A relationship.

→ Inheritence:

Inheritence is a mechnism to represent parent child relationship where a child class inherits the property from parent class.

- Inheritence is unidirectional so only child has property of parent class.
- Parent class will not have any properiter of child class
- Inheritence in java is achived by using keyword "Extends".
- A child class alway extends to the parent class to inherit the Property

Date: Page No.:

- Inheritance represent IS-A relationship.

eg:- child. Parent

car is a vehicle
Mango is a fruit
circle is a shape
chrome is a browser
dog is a animal.

Parent class
or
super class
or
Base class
or
generalized class

| car |

child class
or
sub class.
or
Derived class
or
Specized class.

eg:-

```
class vehicle {

}

class car extends vehicle {

}
```

child class

parent class.

Date: Page No.:

→ Types of Inheritence:

(i) Single level Inheritence
(ii) Mutilevel Inheritence
(iii) Hierachical Inheritence
(iv) Mutiple Inheritence
(v) Hybrid Inheritence

• Single-level Inheritence :-

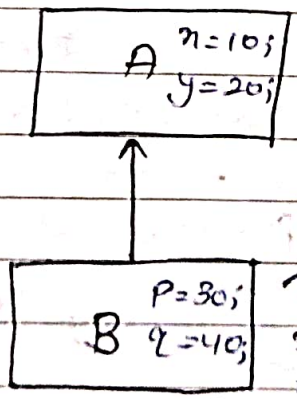
when their is a parent class and a child class then it is called single-level Inheritence.

Class A {

}

Class B Extents A {

}



Total members = 2
both are declared.

Total members = 4
2 => are declared
2 => are inherite member.

Date: Page No.:

- Parent and child relation is created by using "super call" statement.
- "Supercall" is constructor call which it is used to chain constructors of child class & parent class.
- "Supercall" is a constructor call so it must be first statement of the constructor

Q Why we cannot use "this call" & "super call" together

Ans This call and supercall are both constructor call and they need to be the first statement of constructor so if they are used together then one will be the first statement other will the second statement which will cause compilation error so it cannot be used together.

Date:

Page No.:

→ Implict Constructor():

whenever parent child relation is achived by compiler internally by adding super() inside child class constructor then it is called implicity super call.

* Explicit constructor():

when parent and child relationship is achived by using super call from child class constructor then it is called Explicit constructor call.

Note: In Implicit constructor call always no. argument constructor of parents class is called by complex.

In Explicit constructor call programmer can call any constructor of parent class.

Date: Page No.:

Static block :-

If a block in java is declared by using static keyword then such block is called a static block.

Syntax :-

```
static {
    // static block:
}
```

A static block is loaded in class static area and it gets executed at the time. We can have multiple static block inside a class and it will be executed in the sequence it is written in the class.

We can initialise a static variable inside this block so. this block is also called static initializer block.

Example for static block.

```
Public class Fruit{
  // Parent class
  static {
    Sop("This is static fruit"); }
  Fruit() {
  sop("This is no arguments");
    }
  }
```

Date: Page No.:

```
Public class Mango extends fruits.{
        //child class.
    Static {
        Sop("This is static
            Mango");
    }  Mango(){
        Sop("This is no-argoMango");
        }}.

Public class Mango Driver.{
    Public static void main (String[] args){
            // main class
            Mango m1 = new Mango();
            Sop ("==========");}}.
```

Output:- This is static fruit.
This is static Mango
This is no args mango
==========.

→ Non-static Block:-

It is a block is not declared by using static keyword than such block is called non-static block or non-static initizer block.

```
Syntax {
        // Non-static block:
        }.
```

Date: Page No.:

A non-static block is loaded and executed only when object is created whenever object is created then every time the non-static block gets executed.

Package static-non-static-block:

```
public class laptop {
    {
        System.out.println("This is non-static block - 1");
    }}
    System.out.println("This is no.")
```

→ Object Loading Steps :

when a program start execution then different types of members are loaded in following sequence.

Step1 :- All the static members are ~~loaded~~ loaded first from parent to child class in their class static area.

Step2 :- New main method of the class starts the execution.

Date: Page No.:

Steps 3: If Objects are created then non-static member from parents to child class are needed inside Object.

Step 4: from parent to child class non-static members are loaded as the below sequence.

(a) First all non-static variables are loaded.

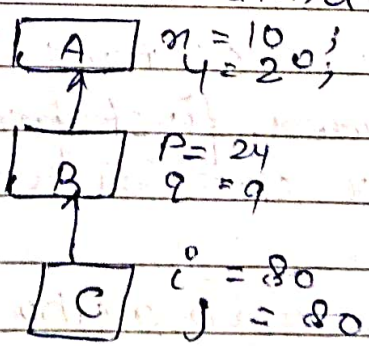
(b) Next non static blocks are executes

(c) Next method and constructors are loaded in the sequence in which they are written.

→ Multi-level Inheritence :-

when there is a parent class and child class and child class is parent for another class then it is called as multilevel inheritence.

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



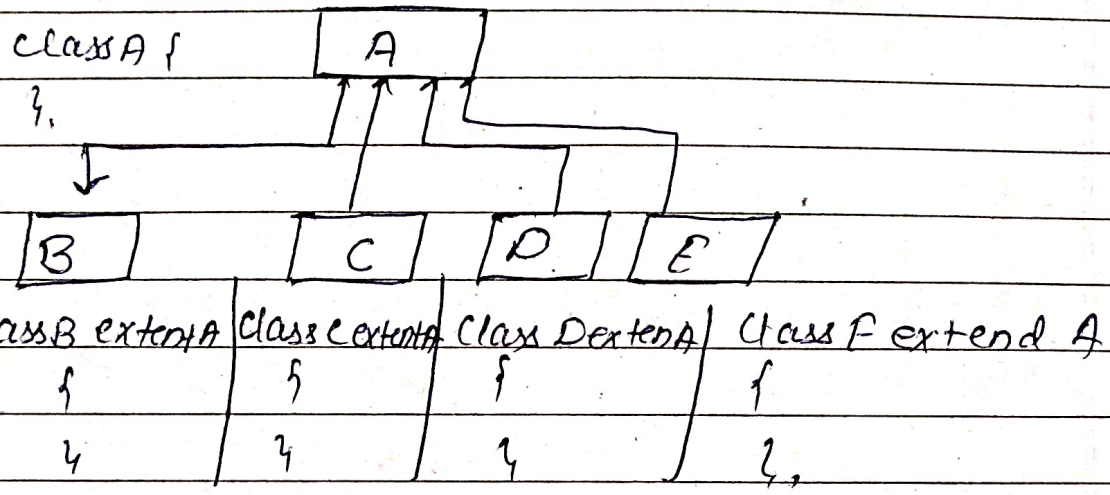
x = 10 ; y = 20 ; total member = 2 (2) declared

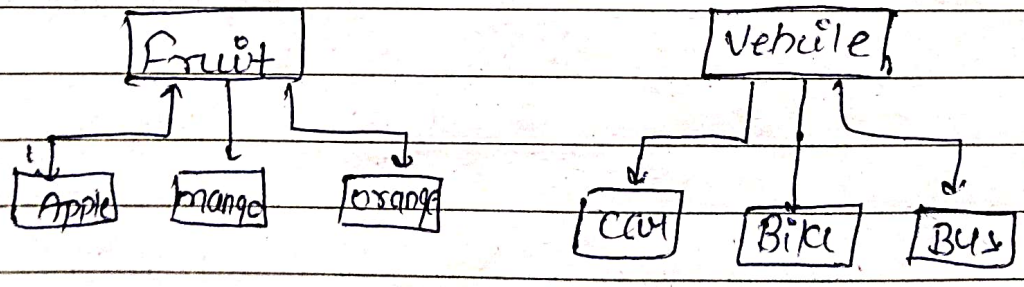
p = 24 q = 9 total member = 4 (2) declared (2) Inheritence

i = 80 j = 80 total member = 6 (02) declared (04) Inheritence

superclass.

Subclass

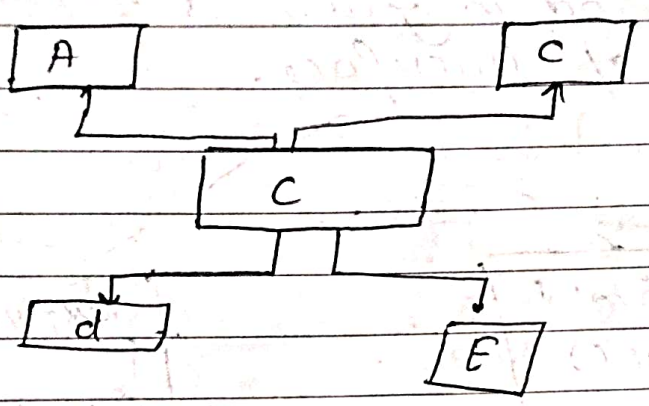

class A {
}.
A
B
C
D
E
class B extend A
{
}
class C extend A
{
}
class D extend A
{
}
class E extend A
{
}.


Fruit
Apple
mango
orange
Vehicle
Car
Bike
Bus

Date: Page No.:

* Multiple Inheritence :

when there are multiple parent class and one child class then such Inheritence is called multiple Inheritence.




```
class c extend A B
{                 ↑
                  error

}
```

* multiple Inheritence is not Support in Java with class.

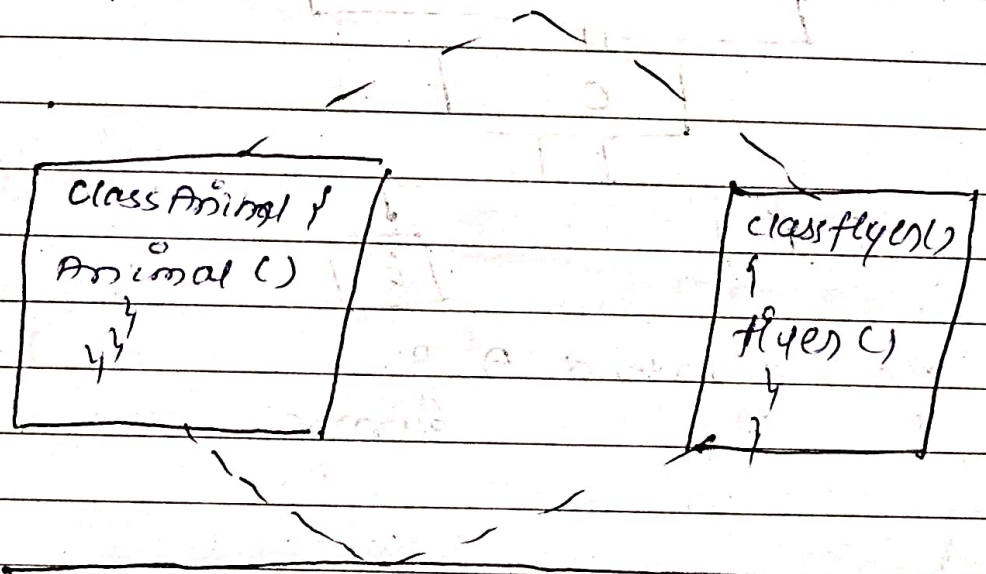
① why multiple inheritence is not supported in java with class.

→ In multiple inheritence multiple parent class will have one child. at the time of constructor call from child class to parent class it creats ambity problem because one super call cannot call multiple constructor at a time. This create

Date: Page No.:

an ambity problem at compilation so multiple inheritence is not supported in class because of ambity problem

The problem of multiple Inheritence is also called diamond shape problem. The diamond in java is resolved by the help of Interface.




```
class Hen extends Animal, Flyer
{ Hen () {
    super ()
    }
}
```

Error: one class can't have more than one parent class

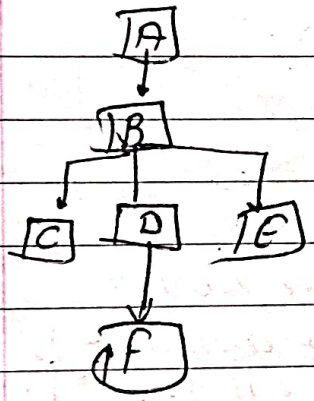
Ambignity Problem for compiler which causes compilation error

Date: Page No.:

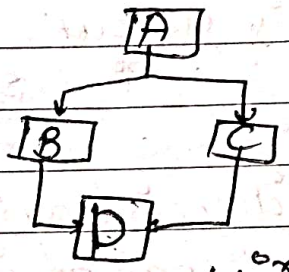
Hybrid Inheritence:

The combination of two or more inheritence is called hybrid inheritence.

* It is supported if multiple inheritence is not used with class.



multiple + hirerical Inheritence



Hirerical + multiple Inheritence.

Date: Page No.:

```
class loan {
 int loan no;
 double amount;
 double roi;
   loan () {
   }
 loan (int loan.no, double amount, double roi) {
        this. loan.no = loan.no;
        this amout = amount;
        this. roi = roi;
      }
Public void displayloan() {
      Sopl("loan no is :" + loan no);
      Sopl("Loan Amount is :" + amount);
      Sopl("Rate of Instrest is :" + roi);
    }}

class PersonalLoan extends loan {
      string name;
      string address;
      Presonal loan() {
        }
  Presonal loan(int loan no
```

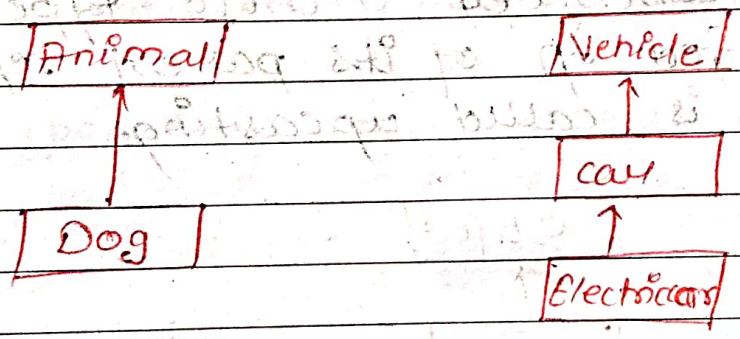
Date: Page No.:

→ Non-primitive type casting :-

If one non-primitive data is converted into another non-primitive data then it is called non-primitive type casting

In another word if a class type data is converted into another class type data then it is called non-primitive type casting.

To achive non-primitive type casting their must be Is-A relationship or Inheritance



Here vechile, car, Electric car can be converted into one another but it cannot be converted into animal or dog because they don't have parent and child relationship.

Date: Page No.:

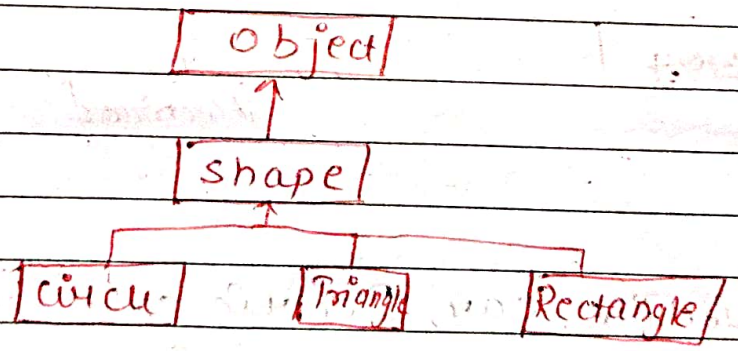
* Non-primitive type casting of two types

(i) Upcasting / Generalization
(ii) Downcasting / specilization

Upcasting / Generalization :-

Upcasting is a process to store child type object on parent type reference.

In another word whenever a child type object is stored on of its parent reference then it is called upcasting.



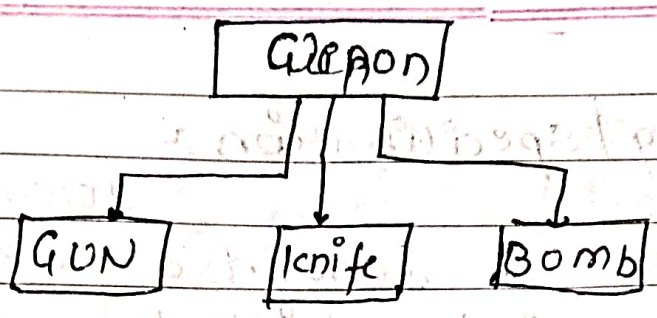
eg:

upcasting.

```
Shape S1 = new circle ();
Shape S2 = new Triangle ();
Shape S3 = new Rectangle ();
Object O1 = new circle ();
Object O2 = new Triangle ();
Object O3 = new Rectangle ();
```

Date: Page No.:

eg:→



(i) weapon w_1 = new Gun();
(ii) Weapon w_2 = new Knife();
(iii) Weapon w_3 = new Bomb();

By using upcasting only details of parent can be accessed as per reference given.

On upcasting child details cannot be accessed.

Shape s_1 → [method of object, n=10, a=30;]

Here, s_1 can access details only from object class and shape class.

→ Upcasting is done by compiler automatically so it is also called auto casting.

Date: Page No.:

* downcasting / specilization :-

downcasting is a process to convert a parent type reference into child type. when a parent type reference on child type reference then it is called downcasting

* downcasting can not be achive directly by compiler. it is achived by using type cast operator. so it is also called expelict conversion

Note:

Upcasting is called implicit conversion.

downcasting is called explicit conversion

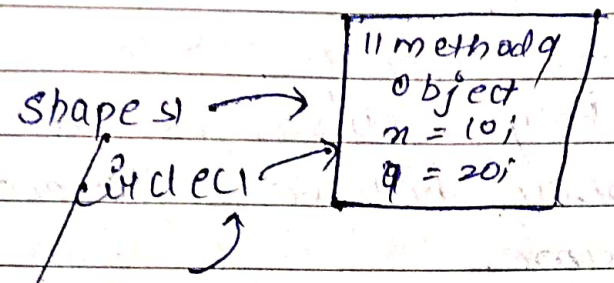
* Upcasting downcasting can be performed only after upcasting is done

* by performing downcasting we can access details of parent as well as child class

```
Shape s1 = new circle();      → upcasting
circle c1 = (circle) s1;      → downcasting
```

typecast operator.

Date: Page No.:

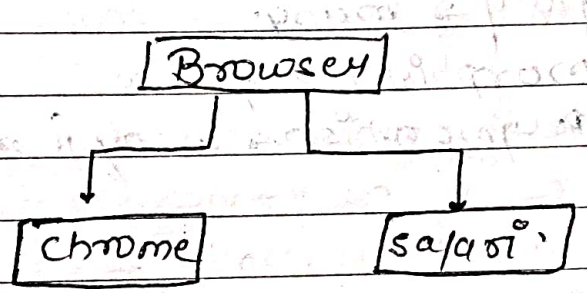


only parent details are accessible by s1.

Parent as well as child details can be accesed by using c1

```
Shape s2 = new Rectangle();
Rectangle r1 = (Rectangle)s2;

Shape s3 = new Triangle();
Triangle T1 = (Triangle) s3;
```




```
Browser b1 = new chrome();
chrome c1 = (chrome) b1;

Browser b2 = new safari();
safari s1 = (safari) b2;
```

Date: Page No.:

→ Polymorphism :-

Polymorphism is a mechanism by which member behave differently with same name.

Same method name ⇒ different implementation
Same object reference ⇒ different forms

poly ⇒ many.
morphism ⇒ forms
polymorphism ⇒ many forms.

* Polymorphism is two type

(i) compile time / static polymorphism.
(ii) Run time / Dynamic polymorphism

* compile time / static polymorphism :

If the execution of the member remains same as per the connection provided by compiler then it is called compile time polymorphism.

what compiler sees/Binds ⇒ Same get executed by JVM at runtime.

Date: Page No.:

* different ways to achive compile-time polymorphism

(i) methodoverloading
(ii) constructor overloading
(iii) operator Overloading [not supported in java]

* Runtime/ Dynamic polymorphism:-
If the execution of member doesn't remains same as per the connection given by compiler then it is called runtime polymorphism.

In another word if execution of member is decided by JVM at runtime as per object created then it is called runtime polymorphsim.

what compiler sees/ Binds => same may not ~~get execut~~ get executed by JVM at runtime.

* different ways to achive run-time polymorphism

(i) method overriding
(ii) Upcasting.

Date: Page No.:

Q why operator overloading is not supported in java ?

→ Operator Overloading by which an operator can be assined different task to perform.

It is supported in C++ but it creates confusion and complexity in the code.

To keep Java simple & easy its not allowed in Java.

Date: Page No.:

* Method overloading :

It is a mechanism to have multiple methods inside a class with a same name but different argument.

In method overloading modifier and return type doesn't play any role it is decided by only method signature where name must be same and argument must be different.

A static as well as non-static method can be overloaded

→ We can achieve overloading in 3 ways.

(i) By changing no. of argument
(ii) By changing the data type
(iii) By changing the sequence of data.

```
class calculator{
    Public static void add (int a, int b) {
        Sop("Add method int, int);
        sop("sum is" + (a+b));

Public static void add (int a, double b){
        sop("Add meth int, double);
        sop("sum is" + (a+b));
```

Date: Page No.:

```
Public static void add(double a, int b){
        sop("Add method double, int");
        Sop("sum is" + (a+b));
    }}.

class driver{
    Public static void main(String[] args){
        (Calculator.add(12, 18, 15)
                    // Add method int double
                    // sum is 30.5

    Calculator.add(20, 30);
                    // Add method int, int
                    // Sum is 50.
    Calculator.add(12.5, 30);
                    // Add ~~method~~ double, int
                        method
                    // sum is 42.5 }}.
```

Javac Driver.Java // connection is provided by compiler

Java Driver // same gets executed by Jvm at runtime.

Date: Page No.:

* Method overloading is an example of compile time polymorphism because execution depends as per connection provided by compiler.

Method overloading is also called by different name such as ~~compile time polymorphism~~

(i) compile-time polymorphism
(ii) static polymorphism
(iii) static method despatch
(iv) Early binding

Advantage of method overloading:

(i) Method overloading help to improve code readability by decreasing complexity of the code

(ii) By method overloading with same name different action is achived so a user will not have to member different names
eg:- Print / println are the overloaded method are available in print stream class of java.

Date: Page No.:

Q Can we overload main method of java?

→ Yes, we can overload main method of java. we can have multiple main method inside a class with different argument but at runtime JVM calls main method with argument "String[]"

imp

* Method Overriding:

If there are methods in parent and child class with same name, same arguments, same return type and non-static type method then it is called method overriding.

* In method overriding execution of a method depends as per object created. it will not depend as per the reference provided.

· By method overriding we update or change the old implimentation of a method from child class by providing new implimantation.

```
class shape {
Public void area(){
  Sop("Area of shape");
```

```
//methods of objects
Public void main(){
  Sop("Area of circle is pi*r*r");
```

shape s_1 →

```
class circle extends shape{
  @overrid
Public void test area() {
  Sop("Area of circle is: pi*r*r")
  }}
  @ override
class Rectangle extends shape{
Public void area() {
  Sop("Area of rectangle is: l*b");
  }}

class square extends shape{
  @ override.
Public void area(){
  Sop("Area of square is: side*side");
  }
}
```

```
class Driver {
Public sm {
  Shapes1 = new circle()
  S1.area()// Area of circle is pi*r*r
  shapes2 = new rectangle()
  S2.area()
  // Area of rectangle is l*b
  shapes3 = new triangle()
  S3.area();
  // Area of triangle is side*side;
```

Java Driver javac ⇒ ~~item~~ Here connection is provided by compiler as per reference given

Java Driver ⇒ Same is not executed by Jvm at runtime execution is provided by Jvm as per objects created

Date Page No.:

* Method overriding is also example of runtime polymorphism because execution is decided by JVM at runtime as per object created.

Method overriding is also called by different name such as.

(i) Runtime polymorphism
(ii) Dynamic polymorphism
(iii) Dynamic method dispatch
(iv) Late Binding

→ Difference b/w method Overloading and method overriding.

(i) Method overloading is achived from same class but method overriding is achived through parent and child class.

(ii) Method overloading can be done for static as well as non static method but method overriding is allowed only for non-static method.

(iii) Method overloading name must be same but argument must be different but in method overriding name, argument and return type must be all same

Public > Protected > Default > private
↑ (No keyword) ↑
least Protected — most Restricted

(iv) Method Overloading is example of compile time polymorphism but method overriding is example of runtime polymorphism.

Q. can be override the main method of java?

→ No, Method overriding is applicable only for non-static method in parent as well as child class.
The main method of java is alway static in nature so method overriding is not allowed for it..

Q. Criteria's to override a method.

(i) Method must be non-static.

(ii) Method signature (name, argument) must be same in parent & child class

(iii) Return type must be same.

(iv) child class method can't decrease the visibility of method through access modifier.
Example if parent class is "Protected then child class method can be public or protected.

Date: Page No.:

Q which type of details are not inherited from parent to child class.

→ (i) A private member is visible and accesable only same class so any private members of parent class will not be inherited into child class.

Imp (ii) A constructor of parent class is not inherited by child class. The basic rule of constructor is that its name must same as the class name so if child will inherit the constructor then child class will have constructor with different name which will break the rule of constructor.

However child class can call parent class constructor by using ~~use~~ super call statement

Date: Page No.:

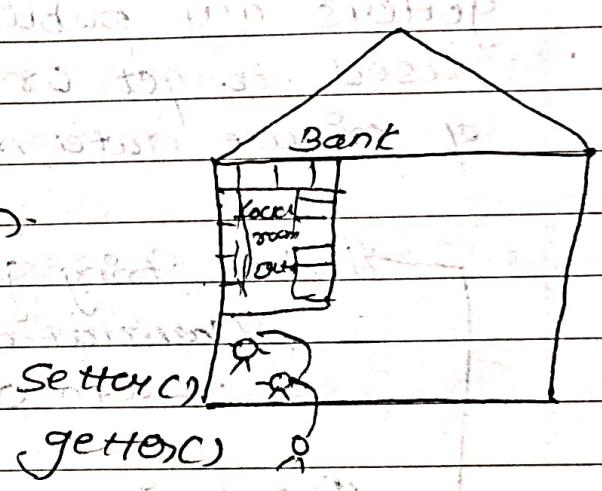
Imp

→ Encapsulation :-

Encapsulation is a process to restrict direct access of data member but provide indirect and validated access through public "getters" and "setters" methods.

In another word Encapsulation is a process to ~~wa~~ wrappe data member inside a single unit called class.

Encapsulation.



→ Steps to achive Encapsulation :-

Step I

(i) We have to declare all data members as private.

(ii) A private member or accessable only inside a same class.

(iii) It cannot be accessed by directly

Date: Page No.:

Step II

We have to design public type get & setter method to provide validate access of private data members. For every private data member their should be one getter and one setter

(a) Design of getter's

getter's are public type method which are used to get or read or access the details of private data members.

```
Public String getName() {
    // validation
    return Name; }

Public int getId() {
    // validation
    return Id; }

Public double getSal() {
    // validation
    return Sal; }
```

getter()

Date: Page No.:

(b) Design of setter

setter's are public type method which are used to set or write or update the details of private data member

```
Public void setName(String[] args){
    // validation
    this.name = name; }

Public void setId (String[] args){
    // validation
    this.Id = Id; }

Public void setsalary( String[] args){
    // validation
    this.salary = salary; }
```

setter()

Step III

we should have only no argument constructor inside encapsulated class.

Q. Design a Facebook user class and active Encapsulation

Q// Design a laptop class and active encapsulation

Q// Design a fruit class and active encapsulation

Date: Page No.:

→ Data Hiding :-

Data Hiding is a process by which we declared a data member private so that it will be visible or accesable only inside the same class it will be not visible inside other class directly.

Inside another class it can be access by the help of getter's and setter's method.

→ Bean class :-

A fully encasulated class called Java Bean class. If :-

(i) All its data members are private
(ii) Their are public getter and setter available
(iii) Only no argument construction is designed
(iv) It implements serializable interface.

The Object created for a bean class is called bean object. The main advantage o bean ~~cal~~ class is code ~~reusa~~ reuseability

Date: Page No.:

Bean class

class Employ implements serializable
{
All data members are Private
+
Public getters() and setters()
+
No argument constructor

Employ e1 = new Employ()
bean object

Date. Page No.:

* Abstraction:-

Abstraction is mechanism to show functionality to user but hide the internal implimantation.

In another words abstraction is mechanism by which we show "what it does to user" but hide's "How it does from user"

Abstraction is achived in two ways:

(i) By using Abstract class (to achive 0% to 100% abstraction)

(ii) By using Interface. (to achive 100% abstraction)

→ Abstract class:

If a class in java is declared by using a key word abstract then such class is called abstract class

Normal class

```
class Employ
{
}
```

```
Abstract class Employ
{
}
```

Date: Page No.:

→ Features of Abstract class.

(i) If a class is abstract then we cannot create object of such class.

```
Abstract class shape()
{
}
```

```
shape s1 = new shape();
```
Error.

(ii) Inside an abstract class we can have two types of methods such as:

(i) Abstract method
(ii) concrete method.

→ Abstract method ⇒ If a method is declared by using abstract keyword then such methods is called abstract method
An abstract method do not specify method body.
∴ Abstract method is allowed only inside abstract class.

eg: Public abstract void test();
Public abstract void demo();
} → abstract method

Date: Page No.:

→ Concrete method :-

If a method is not declared by using abstract keyword then such method is called concrete method.

- A concrete method will always have method body.

eg
```
Public void start() {
    sop ("start method")}
```
} concrete method.

```
abstract class Vehicle {
    Public void start() {
        sop("start the vehicle"); }
    Public void stop() {
        sop("stop the vehicle"); }
    Public abstract void drive();
    Public abstract void accelarate();
}
```

% abstraction = $\frac{\text{Total no. of abstract method}}{\text{Total no. of instance method (non-static)}}$ * 100%

$= \frac{2}{4} \times 100 = 50\%$ abstraction

Date: Page No.:

(iii) we can have static as well as non-static variable inside abstract class

(iv) we can have constructor inside abstract class which are used to initialize instance variables of abstract class.

(v) We can have static as well as non-static block inside abstract class.

(vi) one abstract class can extend another abstract class

(vii) In the end there should be normal class we should provide body to all the abstract method by method overriding

→
```
abstract class vechicle {
        Public void start() {
            sop("start the vehicle"); }
        Public void stop() {
            sop("stop the vehicle"); }
    Public abstract void drive();
        Public abstract void accelarate();
    }.

abstract class car extend vehicle {
        Public void playmusic() {
            sop("play music in car"); }
        Public abstract void openGate();
    }.
```

Date: Page No.:

→ Difference between class and abstract class.

class	abstract class
(i) It is declared by key word class	(i) It is declared by keyword abstract class
(ii) Object can be created	(ii) object can't be executed
(iii) Only concrete method are allowed	(iii) concrete as well as abstract method are allowed.
(iv) It is not used to achive hiding	(iv) It is used to achieve hiding from 0% to 100%
(v) It implements the features define by abstract class	(v) It defines the general features for a class.

Date: Page No.:

→ Interface :

It is a mechanism to achieve complete abstraction or 100% abstraction. Interface is designed by using keyword interface.

```
class vehicle
{
}
```

⇒ A class

```
interface vehicle
{
}
```

⇒ An interface

→ Features of interface :-

(i) Interface is a blueprint similar as a class which defines the features or contracts for a class

(ii) Objects creation of interface is not allowed

(iii) Inside an interface a variable is always by default public, static and final type variable

(iv) Inside interface a non-static method is by default is abstract.

Date: Java file Page No.:

```
interface valu
{
int x = 10;
Public void start();
}
```

After compilation →

• class file

```
interface valu
{
Public static final
int x = 10;
Public abstract
void Static();
}.
```

Imp

(v) we can have static and default type method inside interface since Java 8 update.

(vi) we can have private method inside interface since Java 9 update.

Imp

(vii) Interface can't have constructor because interface will not have instance variable

(viii) Interface can't have static or non-static intilizer blocks. that's why constructor or object are not allowed.

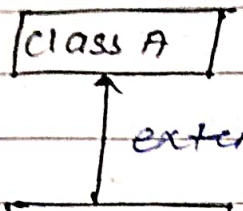
(ix) One Interface can extend another interface so Interface can have "Is-A Relationship"

(x) In the end their should be a class which will implement Interface and will provied body to all the abstract methods.

by method overriding
@override

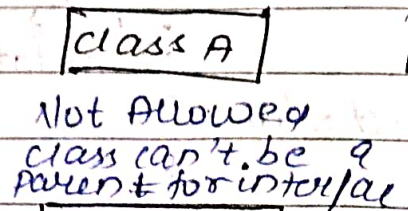
Date: Page No.:

(I)

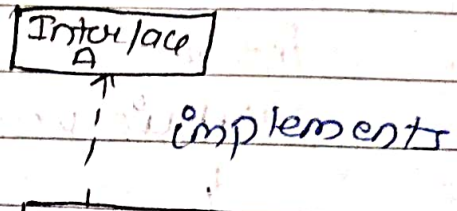


class B extends A
{
}

(II)

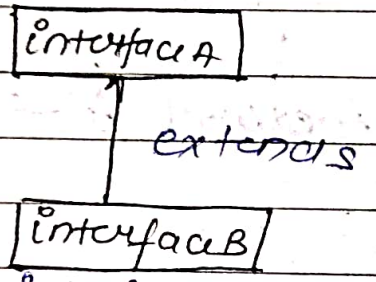


(III)



~~Interface~~
class B implements A
{
}

(IV)



interface B extends A
{
}

→ interface Vehicle
{
Public void start();
Public void stop();
}

interface car extends Vehicle {
Public void open gate();
}

Electric car @

Vehicle →
car G →

```
// methods of object
Public void start() {
}
Public void stop() {
}
Public void open gate()
{
}
```

Date: Page No.:

```
class Electric car implements car
{ @override
Public void start () {
    Sop("Start the Electric car");}
  @ Override
  Public void stop() {
    Sop("stop the Electric car");}
  @ override
  Public void openGate() {
    Sop("open Gate in Electric car");}
```

→ class Driver {

```
Public static void main (String[] args) {
    Vehicle v1 = new Electric car();
        v1.start();
        v1.stop();
        car c1 = (car) v1;
        c1.openGate(); }}
```

o/p: Start the E1 car
stop the E1 car
open Gate in Elect car

Date: Page No.:

→ Some special type of ~~collection~~ Interface.

① Functional interface:

If an interface has only one abstract method then such interface is called functional interface.

functional interface can have multiple static and default method it should have only one abstract type method.

Ex:

Runnable, comparable
compatariable supplier are the example of functional interface.

@ Functional Interface
interface Runnable
{
Public void run();
}.

It is used to create Thread Object.

Date: Page No.:

(ii) Marker/Tagged Interface :

If an Interface doesn't have any member then such ~~m~~ Interface is called marker or Tagger Interface

In other word an empty Interface is called marker or Tagged Interface

Ex:- serilizable, clonable.
Remote are the example of marker or tagged Interface

- marker interface is used to pass some important msg to JVM

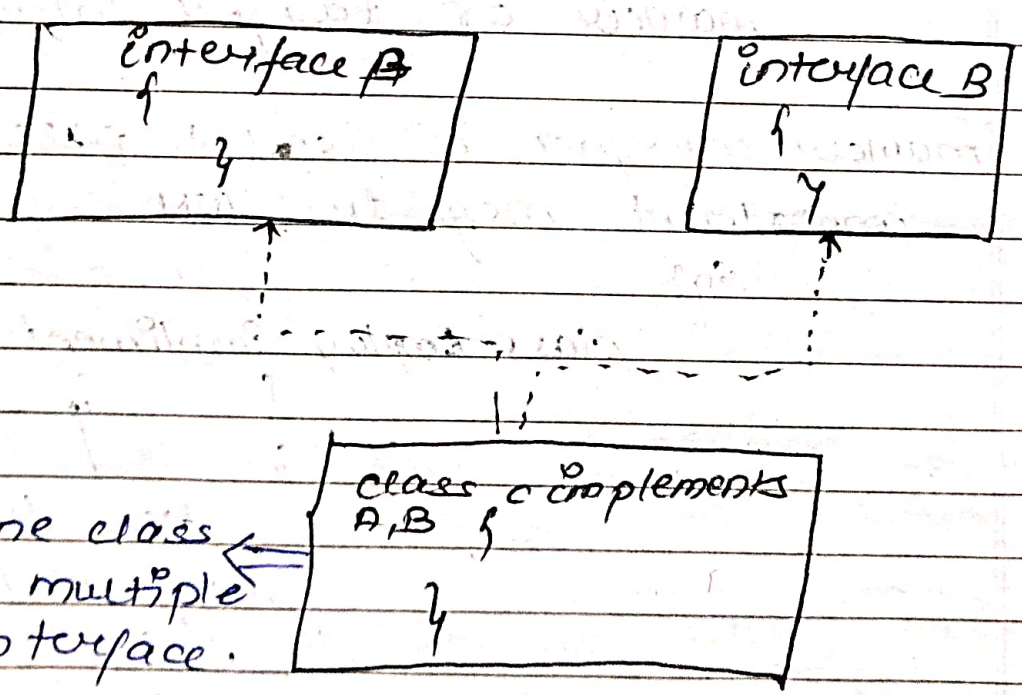
Eg:-

class Employ implement serialization
{
}

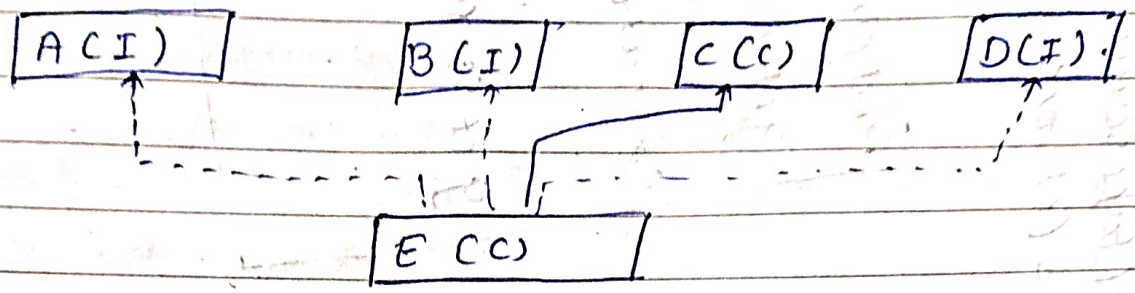
Here, Jvm will get msg to perform serialization.

Q. How multiple inheritence is supported by interface?

In case of class multiple inheritence is not supported because it creates ambiguity problem.
As Interface doesn't have constructors so ambiguity problem doesn't occur with interface, so multiple inheritence is supported by interface.

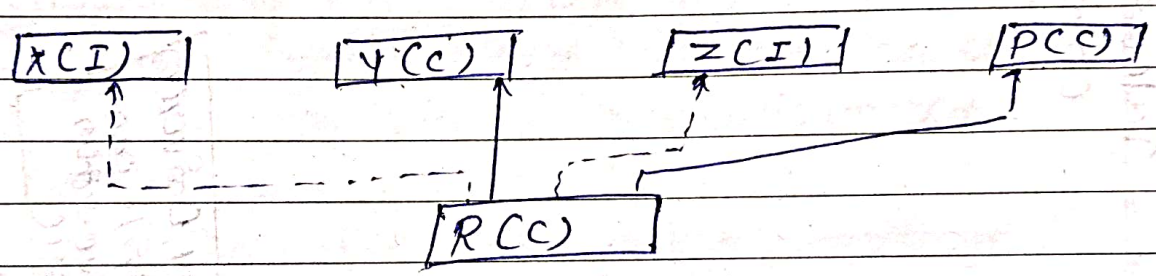


Date: Page No.:



class E extends C implements A, B, D.
{

}



Class R extends Y, P implements X, Z
{

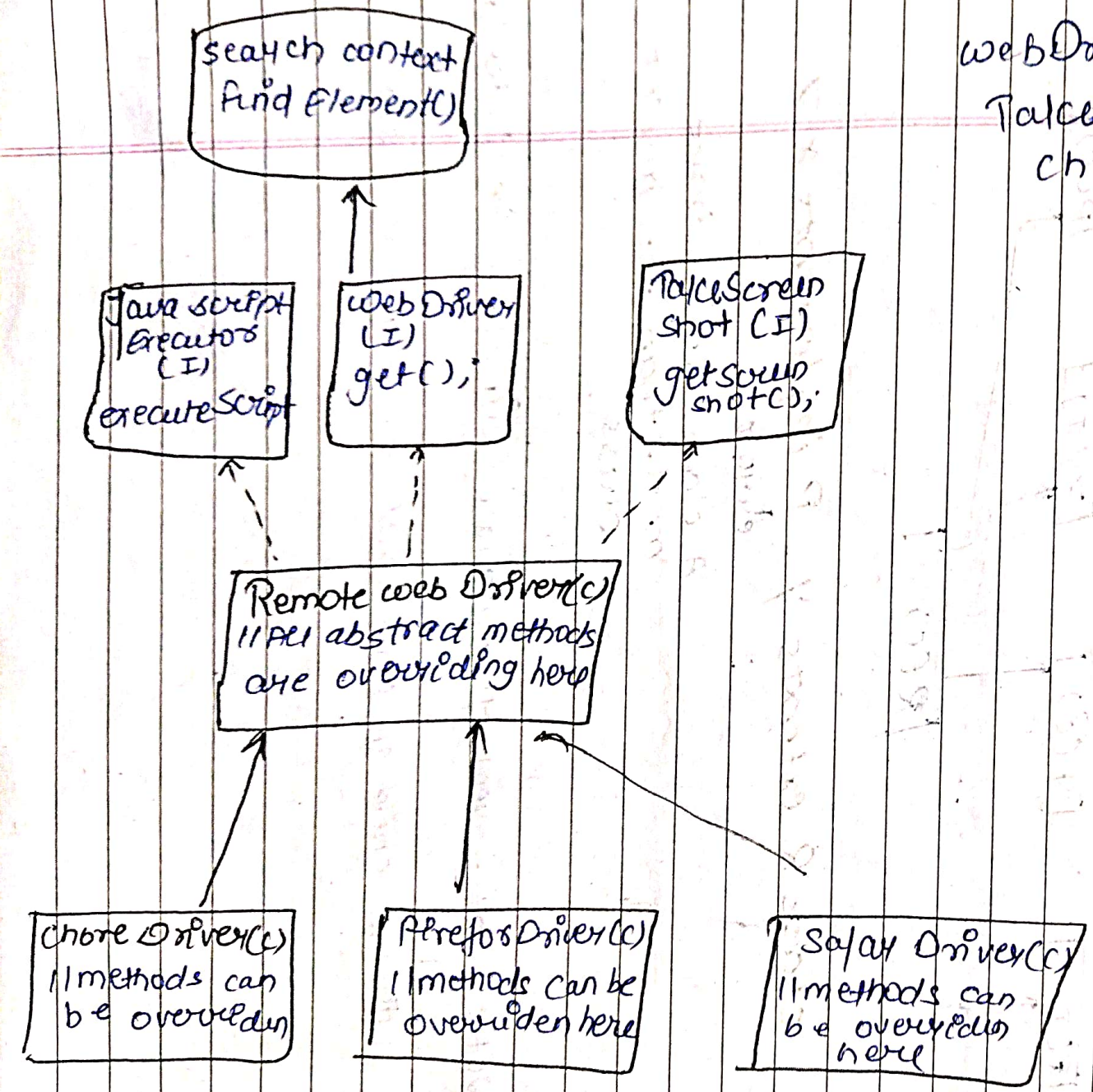
}

error

A class can't have more than one parent class.

Note:- (i) The parent of Interface will alway be Interface

(ii) The child of class will alway be a class.




```
webDriver wd = new chromeDriver
TakeScreenshot ts = (takescreenshot)wd;
chromeDriver cd = (chromeDriver)wd;
```

```
Find Element() {}
get() { }
executeScript() {
}
get screenshot().
```

Date: Page No.:

* Default method :-

Default method is a special type method. which is allowed inside interface since Java 8 update.

Default method created by using a keyword. "default" and it must have default type body. By the use of default type method we can add new features. in a project without distrubing our existing project.
Default type method should be overridden at later the stages of development.

* Private method :-

- Private type method is introduced in interface. Since Java 9 update.
- Private method is a concrete type method which is used inside default method to run or execute a group of statement.

Q. Difference b/w class and Interface.

class	Interface.
(I) Designed by keyword class	(I) Designed by keyword Interface.
(II) object can be created	(II) object can't be created
(III) Variables can be static or non-static	(III) Variable are by default static

Date: Page No.:

(iv) constructor are allowed	(iv) constructors are not allowed
(v) Non-static method are always concrete	(v) Non-static method are by default abstract
(vi) static and non-static blocks are allowed	(vi) static and non-static blocks are not allowed.
(vii) multiple inheritance is not allowed	(vii) multiple inheritance to allowed.

Q Difference b/w abstract class and Interface.

Abstract class	Interface.
(i) Declared by abstract keyword	(i) Declared by interface keyword.
(ii) Variable can be static or non-static	(ii) Variable are always static
(iii) constructor allowed	(iii) constructor not allowed
(iv) Non-static method can be concrete or abstract	(iv) Non-static methods are always abstract.

Date: Page No.:

(v) static and non-static blocks are allowed	(v) static and non-static blocks are not allowed
(vi) multiple inheritance are not allowed.	(vi) multiple inheritance are allowed
(vii) It is used to achieve 0% to 100% abstraction	(vii) It is used to achive 100% abstraction

Give some real time example of interface and abstraction.

- while using any ecommerce application we are shown only functionality such as buy add to cart, wishlist etc. but its internal implementation are not shown to users

- while performing a transaction at ATM machine a user is shown only functionality such as user is shown as withdraw, deposit, check statement etc. but the internal working of atm is not show to user.

- while using any bank application are shown only features such send or check balance but its internal working is not shown.

Date: Page No.:

Features of Java 8 :-

(I) Lambda Expression

Lambda Expression was introduced in Java 8.

(II) Stream API :-

Stream API was introduced in Java 8 to work on large amount of data

(III) Time and Date API :-

Time and Date API was introduced in Java 8 to work on time and date

(IV) Static method :-

Static method was allowed for interface Java 8 update.

(V) Default method :-

Default method was introduced for interface in Java 8 update.

(VI) Functional Interface :-

Functional Interface was introduced in Java 8 update.

Date: Page No.:

* Wrapper class :-

In Java for every primitive type data Their is a class available which is called wrapper class.
with the help of wrapper class we can convert a primitive data into non-primitive Data and non-primitive back into primitive type

Primitive Data		Wrapper class/Nonprimitive
(i) byte	→	Byte
(ii) short	→	short
(iii) int	→	Integer.
(iv) long	→	Long
(v). float	→	Float.
(vi) double	→	Double.
(vii) char	→	character
(viii) boolean	→	Boolean.

Date: Page No.:

```
int x = 212;
Integer x = 212;
   OR
Integer x = new Integer();
```

* Boxing or AutoBoxing :-

Boxing is a process to convert a primitive data into non-primitive data.

Boxing is achieved by compiler directly so also called auto Boxing.

```
ex-  int x = 358;
     Integer y = x; ---> boxing

     double p = 18.58;
     Double q = p; ---> boxing.
```

Unboxing or AutoUnboxing :-

Unboxing is a process to convert a non-primitive data block into primitive type. It is done by compiler automatically so it is called autounboxing.

Date: Page No.:

ex- Integer x = 856;
int y = x; —— unboxing.

Double d = 856.73;
double q = d; —— unboxing

Integer ← int ⇒ Boxing
int ← Integer ⇒ unboxing.

→ Advantage of wrapper class :-

(I) Every wrapper class has a method tostring() which is used to convert any data into string type.

```
tostring();
int x = 35687
string y = Integer.toString(x);
sop("total digit is:" +y.length);

double p = 183.456;

string q = Double.toString(P);
sop("Total digit is:" + (q.length()-1));
```

Date: Page No.:

(II) With the help of wrapper class we can convert a String value into its original form.

Wrapper class provids two different methods to achieve it which are

(I) valueOf();
(II) parse();

```
String s1 = "238";
sop(s1+100); // op - 238100

Integer x = Integer.valueOf(s1);
int y = Integer.parseInt(s1);

sop(x+100); // 338
sop(y*2); // 476.


String s2 = "28.5";
sop(s2+100); // op - 28.5100.;

Double a = Double.valueOf(s2);
double b = Double.parseDouble(s2);
sop(a+100); // 128.5
sop(b*2); 57.0;
```

Date: Page No.:

Note :

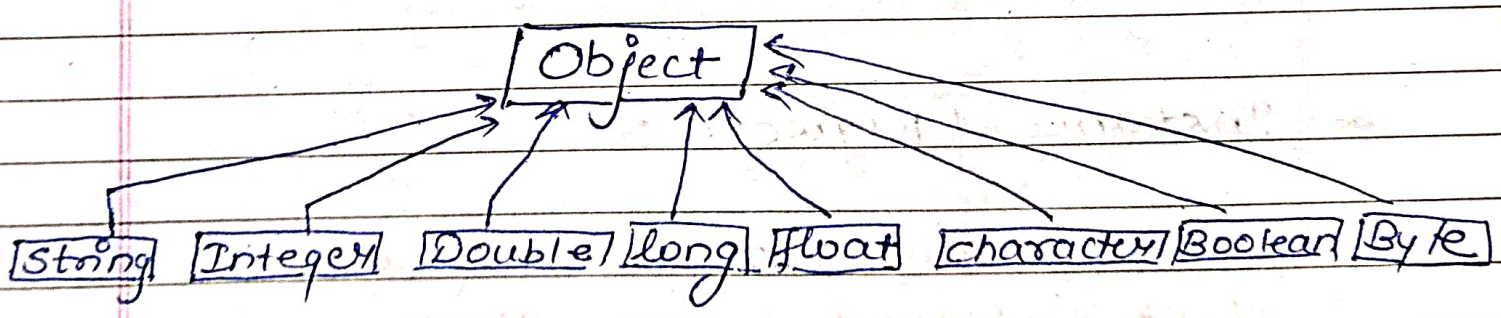
(I) value of method return's Integer while parse method return primitive data int.

(II) If a String will contain String type value then its will cause a exception called number format exception.

```
String s1 = "@aw4b";
Integer x = Integer.valueOf(s1);
```

Exception
NumberFormatException

(III) with the help of wrapper class we can store any type of data on object reference by using upcasting mechanism.




```
object o1 = "mohan";
Object o2 = 18;
Object o3 = 10.5;   Object → [18, 10.5, "mohan", '@']
object o4 = true;
Object o5 = 'a';
```

Date: Page No.:

(IV) With the help of wrapper class we can convert any object type data into its original form by to use its special features

example →

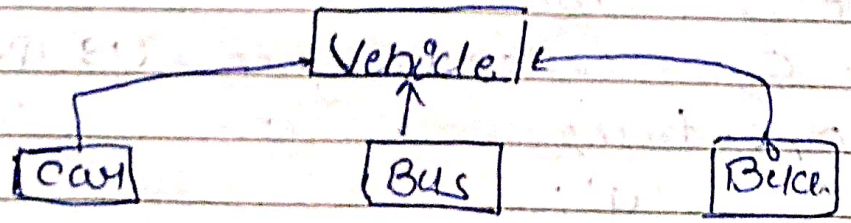
```
Object O1 = "mohan is here";
String S1 = (String)O1;

Sop("length is : " + S1.length());
Sop("length is : " + ((String)O1.length());

Object O1 = 234;
int x = (Integer)O1;
Sop(x+100);
Sop((Integer)O1 + 100);
```

* Instance of keyword :-

It is a keyword which is used to check whether a given reference holds a particular type of instance or not



Date: Page No.:

```
vehicle v1 = new Bus();
vehicle v2 = new car();
vehicle v3 = new Bike();

sop(v1 instance of car); false
sop(v1 instance of Bus); true
sop(v2 instance of Bike); false
sop(v3 instance of Bike); true

Public class Test {
  Public static void checkdata(object o) {
      if (o.instance of Integer) {
      sop(o+"is Integer Data);
      sop((Integer)o + 100);
      }
    else if (o.instance of Double) {
        sop(o+"is double Data");
        sop(Double)o +100);
      }
    else if (o. instance of String) {
      sop(o+"is string data");
      sop((string)o).length()); }}}.

Public class Driver {
    pvs m (string [] args) {
      test.checkData(34);
      sop("=====");
        test.checkdata("Mohan");
        test. checkData("34.8);
```

```
test.checkData('@');
test.checkData(true);
test.checkData(new car());
}}
```

Note :-

(I) At boolean constructor we can provide any value in form of string. In that case the provided string is first converted into lowercase and then it is compared with true. If it equal then it will be true otherwise it will be false.

```
Boolean b1 = new Boolean(true); => true
"       b2 = new    "     (false); => false
"       b3 = "      "     ("true"); => true
"       b4 = "      "     ("true"); => true
"       b5 = "      "     ("false"); => false
"       b6 = "      "     ("mohan"); => false
"       b7 = "      "     ("abc"); => false
```

Date: Page No.:

(II) If downcasting is not performed in right way then it will cause exception.

In another if two classes relationship then it it will give exception at upcasting or downcasting.

ex - object o = "mohan"
int x = (Integer) o;

class cast Exception because string and Integer doesn't have Is-A Relationship.

Date: Page No.:

* Final keyword :-

Final is a keyword which is used to restrict modifications of a member. Final can be a class, method, variable

a) Final class :-

If a class in java declared by using final keyword then it is called a final class.
If a class is final then it can't have any child class or it can't be extended.

example -

```
final class car {

}
```

```
class ElectricCar extends car
{
}
```

Error a final class can't be a parent for a class.

Example of final classes are String class and all wrapper class.
As class is final so, it can't have any child class and its method can't be overridden.

Date: Page No.:

b) Final Method :-

If a method is declared by using final keyword the such method will be called final method.

If a method is final it can't be overridden from child class.

example →

```
class vehicle {
    public final void start ()
    {
    // start method
    }.
    Public void drive ()
    {
    // drive method
    }}
```

final method can't be overridden

It is not a final method so, it can be overriden from child class.

c) Final Variable :-

If a variable is declared by using final keyword then such variable is called a final variable.

a) Local final variable :-

If a local variable is declared by final keyword then it can be initialised only once and later we can't change its value

final int x = 38; → final variable

final int y; → blank final variable.

y = 90;

b) Static final variable :-

If a static variable is declared by using final keyword then it is called static final variable either should be initialised at the time of declaration or it can be initialised from static initialiser block.

```
class vehicle {
    final static String brandName = "TATA";
    final static int numofGear;
    static {
        numofGear = 6;
    }
}
```

c) Non-static final variable :-

If a non-static variable is declared by using final keyword then it must be initialised from every constructor of the class otherwise it will give error.

Date: Page No.:

example :-

```
class Employ {
    string name;
  final int id;
    double salary;
Employ (int id) {
        this . id ;
    }
  Employ (string name, int id) {
        this. name = name;
        this . name = name;
    }
  Employ (string name, double salary, int id) {
        this . name = name;
        this. salary = salary
        this . id = id;
    } }
```

$e_1 \rightarrow$

name = n44
id = 234
Salary = 0.0

```
Employ e1 = new Employ (234);
e1 . name = "mohan",
e1 . salary = 45000;
```

$e_2 \rightarrow$

name = sohan
id = 364
Sal = 0.0

```
Employ e2 = new Employ
("sohan" 364);
```

Page No.:

Date:

* Mutable :-

If a class is designed in such a manner that the details of its object can be changed multiple times then such class is called mutable class.

Example :-

```
class Employ {  → mutable class.
    string name;
    int id;
    double salary;
}
```

* Immutable class :-

If a class is designed in such a manner that once its object is created then the detail of object can't be changed or modified.

Steps to create Immutable class :-

- we have to declare class a final so that it can't be extended and its method can't be overridden

- we have to declare variable as private and final so that it can't be modified

Date: Page No.:

- we have to design a parameterised constructor from where all the final variable must be initialised

- we have to design getters to get the details of all private members.

example →

```
Final class Employee {
    private final string name;
    private final int id;
    private final double salary;

Employee(string name, int id, double salary) {
        this.name = name;
        this.id = id;
        this.salary = salary;
    }
    public string getname() {
        return name;
    }
    public int getId() {
        return Id;
    }
    Public double get salary() {
        return salary;
    }
}
```

Date: Page No.:

```
Employee e1 = new Employee("mohan", 123, 45666.37);

sop("Name is" + e1.getName());
sop("Id is" + e1.getId());
sop("salary is" + e1.getsalary());
```

e1 →

```
string name = mohan;
int id = 123;
double salary = 45666.37;
Employ() { }
Public string getName() { }
       " int getId() { }
       " double getsalary() { }
```

* Has-A-Relationships :-

when one class is used inside another class then it is called has-A-relationship

example :-

car Has-A Engine → class Engine { }

mobile Has-A Processor

Fruit Has-A screen

Laptop Has-A Screen

Mouse Has-A kitchen

student Has-A Address.

```
class car
{
Engine e =
  new Engine
}.
```

Date:_____ Page No.:_____

In another word if the reference of one class is used inside another class then such relationship is called Has-A Relationship

```
class Engine.
{
 int hp ;
 int strokes ;
 string type;
 Engine ( ){
 }
Engine(int hp, int strokes, string type){
        this.hp=hp;
        this.strokes = strokes;
        this.type = type;
        }
    Public void getEngineDetails(){
            sop("Hp is : " + hp);
            sop("Strokes is :" + strokes);
            sop("Type is : " + type);
            }
            }
```

```
class car {
String name;
double price;
Engine e = new Engine(120, 8, "petrol");
car() {
}
car(String name, double price) {
    this.name = name;
    this.price = price;
    this.e = e
}

public void getCarDetails() {
    sop("Name is : "+name);
    sop("prime is : "+price);
        e.getEngineDetails();
```

```
class Demo {
public static void main(String[] args) {
car c1 = new car("TATA", 123456.7);
car x = new car("Audi", 23456.7, new Engine
                (200, 10, "Desel"));
```

new Engine

```
// methods
hp = 200
sp = 10
type = Diesel
```

car @

```
// methods of objects class
String name = null;
double price = 0;
Engine e @
Engine() {
}
getCarDetails() {
e.getEngineDetails();
}
```

Engine @

```
// methods of objects car
int hp = 100
int speed = 0.8
str type =
Engine(---) {
}
getEngineDetails() {
}
```

Date: Page No.:

Types of Has-a-Relationship :-

Has a relationship is of two type :-

I) composition.
II) Aggregation.

Composition :-

If has a relation is a strong relation then it is called composition. in composition. if the main object is deleted then the dependent object is also get deleted

```
ex - car c1 = new car("maruti", 8888.88,
                    new Engine(15,10,
                        "diesel");
```

Here if car is erased then engine object will also be deleted because engine doesn't have its own reference.

Aggregation :-

If has-A relation is a weak relationship then it is called aggregation In aggregation by deleting one object the other object will not be deleted

Date _____ Page No.: _____

```
Engine e = new Engine (180, 9, "CNG");
car c2 = new car ("BMW", 222.25, e);
c2 = null;
```

Here by deleting the car object the engine object will not be deleted because engine has its own reference.

- Explain System.out.println().statement of java.

System.out.println() is a standard output statement which is used to print some information in java.
This statement is written by using "has-a" relationship.
System is the class where printstream reference is used.

System :-
System is a class available in java.lang package where there are three static type reference which are out, in and err.

Date: Page No.:

Out :-
Out is a static type reference in System class of printstream type.

Println :-
Println is a overloaded method in printstream class of Java. Printstream is the class which is available in Java. io package.

Difference between Is-A and Has-A relationship:-

(i) Is-A relationship is used to represent parent and child relationship but has-A relationship is used to represent all the other types of relations which are not parent and child relationsh.

(ii) Is-A is of five types of but Has-A relationship is of two types which are composition and aggregation.

(iii) Is-A relationship is achieved by using keywords extends but Has-A relationship is achieved by providing reference of one class inside another class.

Date: Page No.:

(IV) Is-A relationship is used to achieve code reusable but Has-a relationship is used to achieve flexibility in the code.

```
class PrintStream {
public void println(int x) {
}
public void println(double x) {
}
public void println(String x) {
}
}
```

```
class System
{
static printstream out=
new printstream();
static printstream err = new
printstream();
static printstream in = new
printstream();
}
```

```
class Driver {
p.s. v main (String[] args)
s.out (21+30);
sout (28.5);
sout ("abc");
```

Date: Page No.:

* PACKAGES :-

PACKAGES means nothing but folder.

- They are physical folder which contains similar type of classes & Interface.

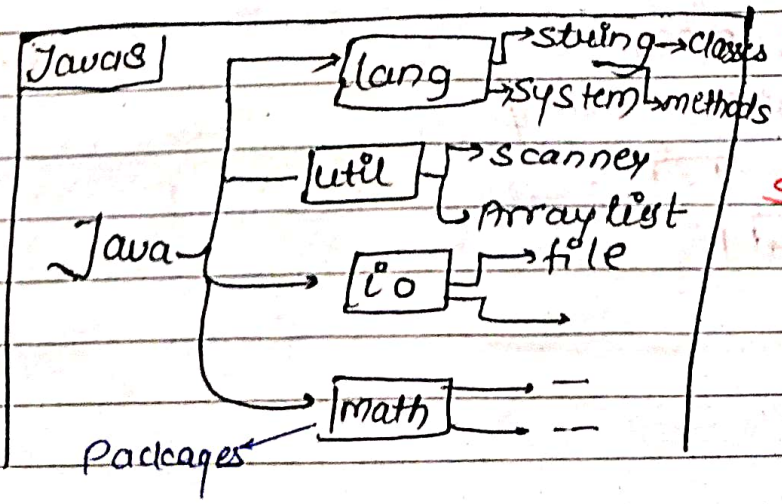
Predefined/Inbuilt Package

Userdefined Package

① Predefined Package :-

↳ 14 inbuilt method are present about 7 methods are available.

- Java provide total 14 predefined packages to perform different types of operational.
- Package is used as library which helps to execute a program.



Total packages : 14
sub packages 70
classes : 7000
methods : 7 lakhs.

Date: Page No.:

(2) Userdefined Packages :

A packages created by user or programmer is called undefined package.

- A package is created by using keyword "Package"
- we can use one package inside another package by using keyword import.

* Structure of package :

company specification

company specification Projectloan

Ex (i) package com.wipro.hdfc.loan.homeloan;

domain name — company name — client name — module name

(ii) Package com.wipro.icici.loan.personalloan;

terminator.

जिसमे package लिख दिया जाता है normally compile/run नहीं कर सकते।

Date: Page No.:

for compile :

जहा पर है वही folder बनेगा

package -d filename.java.

for Run/Execution :

Java com.unipo.hdfc.loan.homeloan.calculateInternet.

Package -d f:\\ filename.java.

जो name देंगे वही class file build होगी

f drive.

package -d f:\\ "foldname" filename.java

(उस particular folder मे जाकर build होगा)

* compile a package file.

- Javac -d calculateInternet.java.

filename.

- By this command package will be created at the same directory.

- Java -d f:\\ calculateInternet.java.

by this command package will be created inside not directly f drive.

Date: Page No.:

- Java -d f:\\ "July 2024" calculate Intrest.
  by this command package will be created inside 08 July 2024 folder of f-drive.

* Command to Run the package class.
- We have to provide fully qualified class name to Jvm to run the program

  Jvm com.wipro.hdfc.loan.homeloan.calculate Intrest;

* Advantages of Packages

(i) By having packages over file will be stored in proper Organised manner.

(ii) By having proper package it is easy to maintain and debug the program.

(iii) By using package we can control access of different types of class its details.

(iv) By having packages we can resolve naming conflict issues.

(v) we can have class with same name in different packages.

Date: Page No.:

* Access modifier.

Java provide different access modifier which are used to change or modify the accessibility of different types of members.

(i) Public
(ii) Protected
(iii) default (no keyword)
(iv) private

(i) Public : If a member declared by using public keyword then such member will be accessible inside same package as well as it can be accessed inside another package
(import वही class हो सकती जो public हो)

- Public member can be class, an integer, a variable, a method, a constructor etc,

- If a member is declared ~~protect~~ protected then such member can access by any class inside the same package but outside the package it can be accessed by child class

Date Page No.:

- A protected member can be a variable, a method, a constructor.

- Protected keyword not applicable for class or Interface so a class or Interface can never be protected.
  (class को protected नहीं बना सकते).

(iii) Default (No keyword).

- If a member not declared with any access modifier then such member is called default type member

- A default type method is available only accesable only inside the same package by any class.

- Any class outside the package can't access default type member.

- A default type member can be a class a interface, a variable, a method or a constructor.

Date: Page No.:

(iv) Private (same class. में ही access होगा बस).

- If a member is declared by using private keyword then such member will be accessable inside only same class.

- A private member can't be