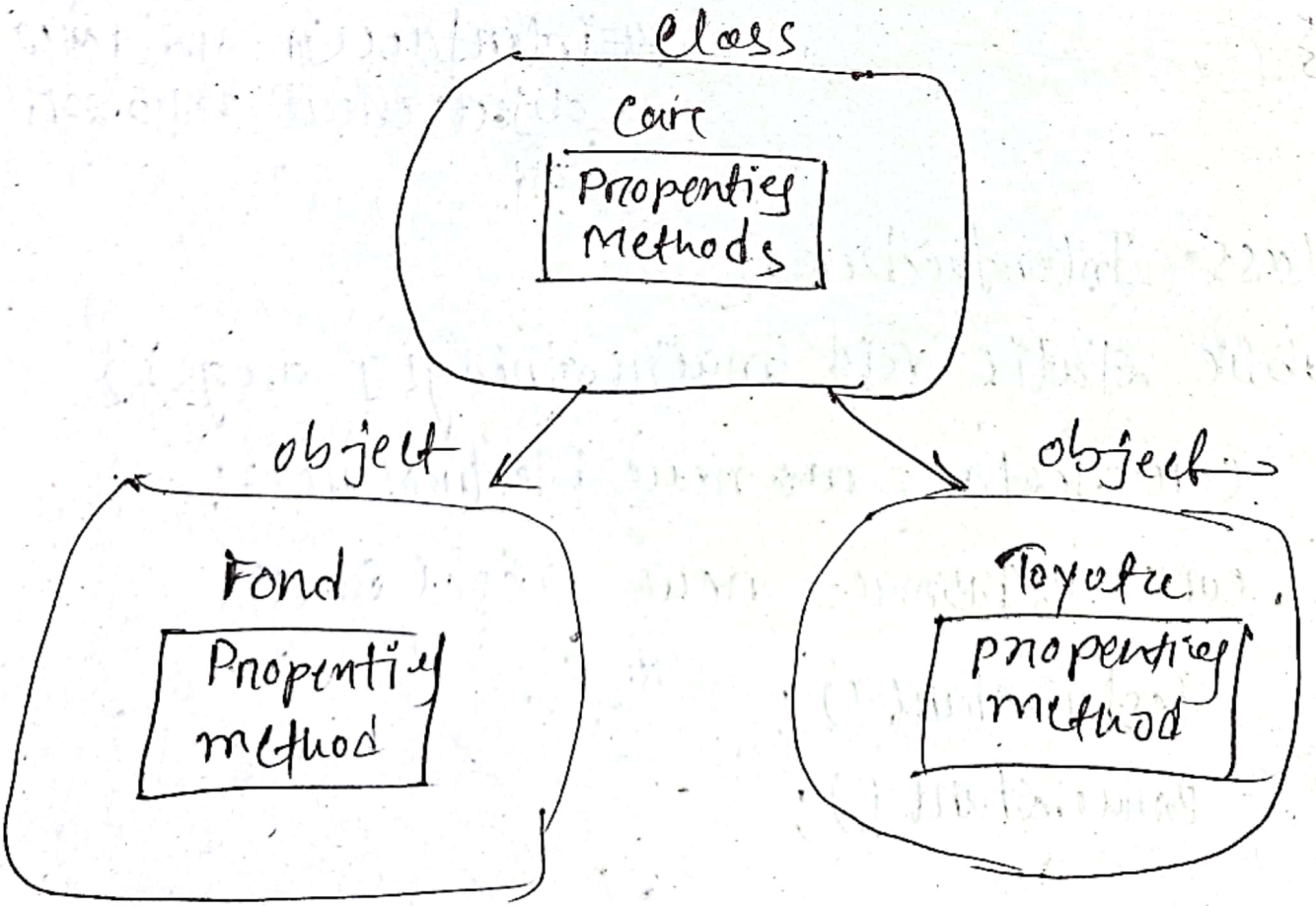
class_object

1. What is class? Briefly explain with an example:

⇒

Class are a a blueprint of or a set of instructions to build a specific type of object. It is a basic concept of object-oriented programming which revolve around real-life entities. Class in Java determine how an object will behave and what the object will contain.



Syntax of class in Java:-

```
class class_name{
    // variable
    // methods
}
```

```
access_modifier class class_name
{
}
```

→ Properties:

1. class is not a real-world entity. It is just a template or blueprint or prototype from which objects are created.
2. class does not occupy memory.
3. class is a group of variables of different data types and a group of methods.
4. A class in Java can contain:
   - Data member
   - Method
   - constructor
   - Nested class
   - Interface

Objects: objects are the instances of a class that are created to use the attributes and method of a class:

An object consists of:-

- State
- Behaviour
- Identity

Difference between class & object

class	object
1. class is the blueprint of an object.	1. An object is an instance of the class.
2. No memory is allocated when a class is declared.	2. Memory is allocated as soon as the object is created.
3. A class is a group of similare bject.	3. An object is a real-world entity such as a book, car, et

4. class is a logical entity.	4. An object is a physical entity.
5. A class can only be declared once.	5. Objects can be created many times as per requirement.
6. An example of class can be a car.	6. objects of the class can be BMW, Ferrari, etc

Access modifiers:-

1. Default
2. Private
3. Protected
4. Public

Constructors: A constructor in Java is similar to a method that is invoked when an object of the class is created.

properties:-

— special type of method (parameters / without parameters)

- Same names as class name
- Does not any returne type
- Enables an objects to initialize itself when it is created

Method overreloading :-

- having two or more methods in a class with the same name and different argumen

ex:

```
int add( int a , int b){
    return a+b;
}
int add ( int a, int b, int c){
    return a+b+c;
}
double add ( double a , double b){
    return a+b;
}
```

Constructor:

Constructor is a special type of method that is used to initialized the object.

properties of constructor:

1. constructor has the same name as that of the class it belongs.
2. It is special type of method.
3. It has no return type. not even void.
4. It is called automatically in program.
5. There are two type of constructor.
   - default
   - parameterized

example:

```
public class Person {
    private string name;
    private int age;
    public Person(string name, int age){
        this.name = name;
        this.age = age;
    }
    public void display(){
```

```
        S.O.P("Name:" +name);
        S.O.P("Age:" +age);
    }
    P.S.V.M(String[] args){
        Person person1 = new Person("Israt", 30);
        person1.display;
    }
}
```

Output: Name: Israt
Age: 30

No, in Java it is not possible to have multiple constructors with different name inside a class.

But, Java allows constructor overloading, where we can define multiple constructor with the same name but with different parameter lists.

Constructor overloading Example:-

```
Public class Person{
    private String name;
    private String city;
```

```
private int age;

public Person(String name, String city){
    this.name = name;
    this.city = city;
    this.age = 30; //default age
}
public Person(String name, String city, int age){
    this.name = name;
    this.city = city;
    this.age = age;
}
public void display(){
    S.o.p("Name:"+name);
    S.o.p("City:"+city);
    S.o.p("age:"+age);
}
}
class Main{
    P.S.v.m(String[] arg){
        Person person1 = new Person("Tama","ABC");
        Person p2 = new Person("Isnat","XYZ",40);
    }
}
```

Output:

```
Name: Tama
City: ABC
Age: 30

Name: Isnat
City: XYZ
Age: 40.
```

— same names as class name
— Does not any returne type
— Enables an objects to initialize itself when it is created

Method overloading :—
— having two or more methods in a class with the same name and different argumen

eg:

```
int add( int a , int b){
    return a+b;
}
int add ( int a, int b, int c){
    return a+b+c;
}
double add ( double a , double b){
    return a+b;
}
```

Method overriding in Java :-

- a method in a subclass has the same name same parameters and the same return type as a method in its supper-class, then the method in the subclass is said to override the method in the supper-class.

Ex:

Static :

Keyword : used fore memory managment mainly

Static variable :

- used to refer to the common property of all objects.

- Gorbd memory no,
  gets memory only once in the class area at the time of class loading.

Advantage :-

- makes program memory efficient.

13 october, 2

Method overriding:-

1. Declaring a method in subclass which is already present in superclass is known as method overriding

Why use:
- code reuse
- One interface, multiple implement
- run time polymorphism

eg: public class person {
    String name;
    int age;
    void display() {
        sout("Name: "+name);
        sout("Age: "+Age);
    }
}

Public class Teacher extends person {
    string qualification;

    @override
    void display() {
        sout("---");
        sout("---");
        sout("qualification: "+ qualification);

Q. can we use static method?
- no, static method can't be overriden.

Because static method is bound to class on the other hand method is bound to object

→ can we override java main method?

= NO, because main is a static method.

overloading	overriding
→ parameters must be different.	→ parameters must be different.
→ It occurs within the same class	→ between two classes – sub class and – supper class
→ Inheritance is not involved	→ Inheritance is involved.
→ Return type may or may not be same	→ return type must be same
→ One method does not hide another	→ child method hides parent another

* Method overriding Vs overloading with example.

=>

example of overriding :-

```
class Car{
    void name(){
        S.O.P("Toyota");
    }
}

class colour extends Car{
    @Override void name(){
        S.O.P("green");
    }
}

class Main{
    P.S.V.m(String[] args){
        Car obj1 = new Car();
        obj1.name;

        Car obj2. = new colour();
        obj2.name;
    }
}
```

Out:

Toyota
green

Example of overloading:-

```
Public class Sum {

    public int sum (int x, int y) {
        return (x+y);
    }

    public int sum (int x, int y, int z) {
        return (x+y+z);
    }

    public double sum (double x, double y) {
        return (x+y);
    }

    public s.v.m (String args[]) {
        Sum s = new Sum();
        S.O.P.(S.sum(10,5));
        S.O.P(S.sum(10,5,20));
        S.O.P(S.sum(10.5, 20.5));
    }
}
```

Output:
15
35
31.0

Method overriding in Java:-

- a method in a subclass has the same name same parameters and the same return type as a method in its supper-class, then the method in the subclass is said to override the method in the supper-class.

Eg:

Static:

Keyword: used for memory managment mainly

Static variable:

- used to refer to the common property of all objects.

- Gorgia memory no,
  gets memory only once in the class area at the time of class loading.

Advantage:-

- makes program memory efficient.

static String college = "XYZ" ; // এখানে না দিয়ে → 1 বার দিলেই হবে।

Static method:

restrictions:

- cannot use this & super in static content;
- cannot use non static data member or call non-static method directly.
- → only call other static methods & access static data.

block: — Is used to initialize the static data member

— It is executed before the main method at the time of class loading.

The mechanism of deriving a new class from an old one is called Inheritance

1.| Static variable;

The static variable can be used to refer to the common property of all object.
static variable gets memory only once in the class area at the time of class loading.
Static variables are, essentially, global variables.
All instance of the classes share the same static variable.
Static variable can call by directly with the help of class only, we do not need to creat object for the class in this.

example of Static variable:-

```
class Student{
    static String collegeName = "University of Barisal";
    String studentName;
    int studentID
    Student(String name, int id){  //constructor
        this.name = name;
        this.studentID = id;
    }
    void display(){}
```

```
        S.O.P ("Student name: " + studentName);
        S.O.P("Student ID : " + studentID);
        S.O.P(" College Name : " + collegeName);
    }
}

Public class Main {
    public static void main (String[] args) {
        Student student1 = new Student("Alice", 101);
        Student student2 = new Student("Bob", 103);
        Student1.display();
    }   Student2.display();
}
```

Output: Student name : Alice

Student ID : 101

College Name : University of Barisal.

Student Name : Bob

Student ID : 103

College Name : University of Barisal

Static method:

Static keyword is used mainly for memory management mainly. Any method that uses the static keyword is referred to as a static method. A static method in Java is a method that is part of a class rather than an instance of that class. static method have access to class variables without using the class's object.

In static method cannot use this & supper static content, cannot use non-static data members, only call other static method and access static data.

Syntax of static method:

```
Access_modifier static void methodName(){
    // method body
}
```

example of static method:-

```
class Greeter{
    static void greet(){
```

```
        S.O.P.("Hello, welcom to Java programming!");
    }
}
public class Main {
    public static void main (String[] args) {
        Greeter.greet();
    }
}
```

Output: Hello, welcome to Java programming!

Q What if main method is not public static in Java?

⇒ In java the main method is required to be public, static & have a specific signature:-

public static void main (String[] args)

If main method is not public static:

1. Not public: if you omit public, the java virtual machine won't be able to access the method since it requires the main method to be pubic.

- This is runtime
the result in a runtime error:

Error: Main method not found in class <classNam>,
please define the main method as:

P.S.V.M (String[], args)

2. Not static: If main is not static, the JVM would be need an instance of class to call main.

3. Incorrect parameter type:
The JVM expects main to take a single String[] argument. if you use any other parameter type or add a return type other than void, the JVM won't recognize it as a valid main method. & show error.

Access Modifiers / Specifiers:

Java has four main access specifiers that define the visibility and accessibility of classes, methods & variables.

1. public: Accessible from anywhere in the program. ~~examples~~ classes, methods or data members that are declared as public are

Example:

```
public class Test{
    public int value;
    public void publicMethod(){
        // access from any class
    }
}
```

2. Protected: the method or data members declared as protected are accessible within the same package or subclass in different packages.

Example:

Program :01

```
package P1;
public class A{
    protected void display(){
        S.O.P("Hello!");
    }
}
```

Program :02

```
package P2;
import P1.*;
class B extends A{
    ps.s.v.m(string[] anges){
        B obj = new B();
        obj.display();
    }
}
```

Output: Hello!

3. Default: The data members, classes or method that are not declared using any access modifiers having default access modifiers are accessible only within the same package.

Example:

```
class Car {
    String color; // Default variable
    void stunt() { // Default method
        S.O.P("Car stunted.");
    }
}

class Main {
    P.S.V.M(String[] args) {
        Car car = new Car();
        car.color = "Red"; // Accessible within the same package
        car.stunt(); // Accessible within the same package
        S.O.P("Car color:" + car.color);
    }
}
```

output:

Car stunted.
Car color: Red

4. Private: The methods or data members declared as private are accessible only within the class in which they are declared.

example:

```
class Car{
    private String name;
    public void setName(String name){
        this.name = name;
    }
    public void startCar(){
        S.o.p("Car Name:" + name);
    }
}
class Main{
    P.S.V.M(String[] args){
        Car car = new Car();
        car.setName("TOYOTA");
        car.startCar(); // Access the private 'name'
                        through public method.
    }
}
```

Output: TOYOTA

Inheritance:

- Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object.

→ Single: when a class inherits another class, it is known as a single Inheritance.

```
class Animal {
    void eat(){
        System.out.println("eating...");
    }
}

class Dog extends Animal {
    void bank(){
        S.o.p.("banking...");
    }
}

class Test inheritance {
    public static void main(String args[]){
        Dog d = new Dog();
        d.bank();
        d.eat();
    }
```

Why multiple inheritance is not supported in Java?

→ To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

When one class extends more than one class then this is called multiple inheritance.

example:

```
public class A{ ----}
public class B{ ----}
public class c extends A,B{ ---}
```

class c extends A and B then this type of inheritance is known as multiple inheritance.

Java doesn't allow multiple inheritance. Java doesn't allow multiple inheritance to avoid the ambiguity.

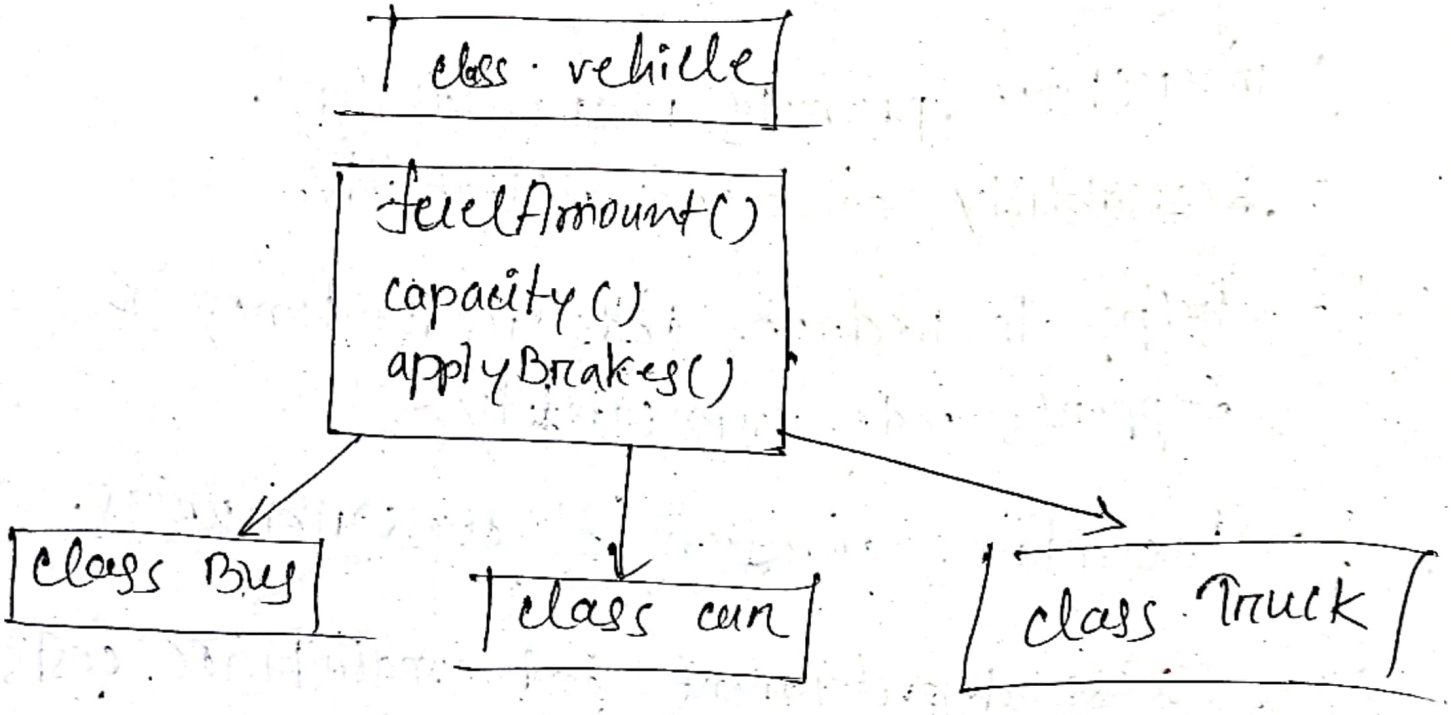
Advantage:—

- inheritance promotes reusability.
- Reusability enhanced reliability.
- helps to reduce code redundancy & supports code extensibility.
- facilitates creation of class libraries.
- less development and maintenance costs.

Disadvantage:-

- Improper use of inheritance may lead to wrong solutions.
- functions work slower.

Example: A child inherits the traits of his/her parents. With inheritance, we can reuse the fields and methods of the existing class. Hence, inheritance facilitates Reusability and is an important concepts of oops.



Ij when using private inheritance,

- all public and protected members of the base class will become private member in derived class.
- only member functions of the derived class can access them.
- Moreover, it is a stop to further inheritance as private members don't get futher inherited.

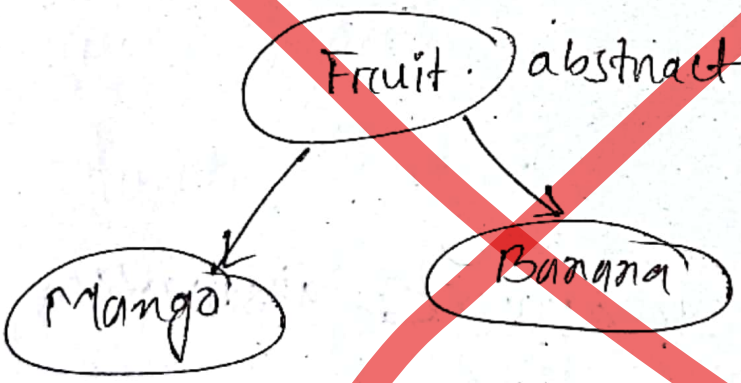
When using protected:-

- your public & protected members of base class become protected members in derived class. It isn't a stop to inheritance as you can go further.

Abstract class; → object এবং এর মধ্যে

→

Abstract class is a technique of hiding unnecessary details from the user.

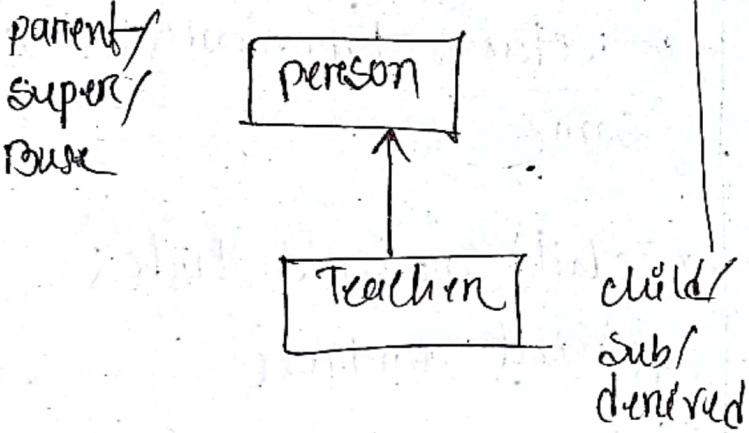


Data abstraction is defined as the process of reducing the object to its essence so that only the necessary characteristic are exposed to the users.

Inheritance: উত্তরাধিকার

```
Public class person{
    String name;
    String age;
    void display1(){
        Sout("Name: "+name);
        Sout("Age: "+Age);
    }
}
```

```
Public class Teacher extends person{ // String name; age;
    String qualification;
    void display(){
        display1();
        Sout("Qulifiecation: "+ qualificaiton);
    }
}
```



→ Inheritance can be defined as the process where one class acquires the properties of another

→ Inheritance allows a class to inherit properties & methods from another class.

Why use:-

- For code reusability
- For method overriding
- To implement parent-child relationship.

How to inherite private members?

- using setter & getter method.

Example:

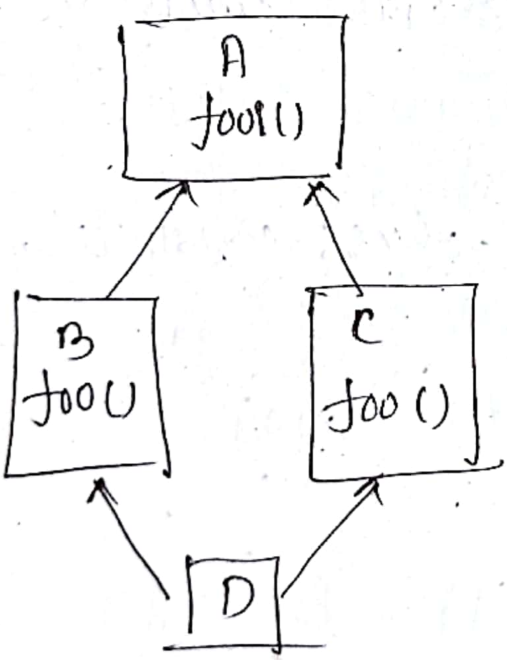
```
Public class person{
    private string name;
    private int age;
    public String getName(){
        return name;
    }
    public void setName(String name){
        this.name = name;
    }
    public int getAge(){
        return age;
    }
```

```
    Public void setAge (int age){
        this.age = age;
    }
}

public class Teacher extends Person{
// person -এর public member গুলো extends করে নিয়ে আসবে;

// getName(), setName(), getAge(); getsetAge()

privatestring qualification;

    public string getqualification(){
        return qualification;
    }
    public void setQualification(string qualification){
        this.qualification = qualification;
    }
}
public class Test{
    public static void main (String[] args){
        Teacher t1 = new Teacher();
        t1.SetName("Asrat");
```

```
t1.setAge(22);
t1.setQulification("Bsc in CSE");

sout(t1.getName());
sout(t1.getAge());
sout(t1.getQulification());
}
}
```

এখানে Teacher class এ display method create না করে call করা হলো main class থেকে,

Java Doesn't Support Multiple inheritance :—



Here, Both classes are in same level with same priority.
If we want to use foo() method that form D class object leads to ambiguity.
This is called diamond problem.

→ To avoid the ambiguity
→ To reduct the complexity and simplify language.

Super:

super keyword is used to refer immediate super class object.

uses:

1. It is used to call super class instance variable
2. It is used to call super class method. (override method).
3. It is used to call super class construction

Example: (Pi)

```
public class A{
    void display(){
        sout("Inside A class");
    }
}
```

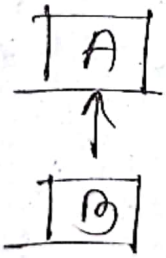
Inheritance: The mechanism of deriving a new class from an old one is called inheritance.

Inheritance allows a sub class to inherit properties and methods from another class.

Type of inheritance:

1. Single inheritance
2. Multilevel inheritance
3. Hierarchical inheritance
4. Multiple inheritance

Single inheritance: A sub class is derived from only one super class.



Example:

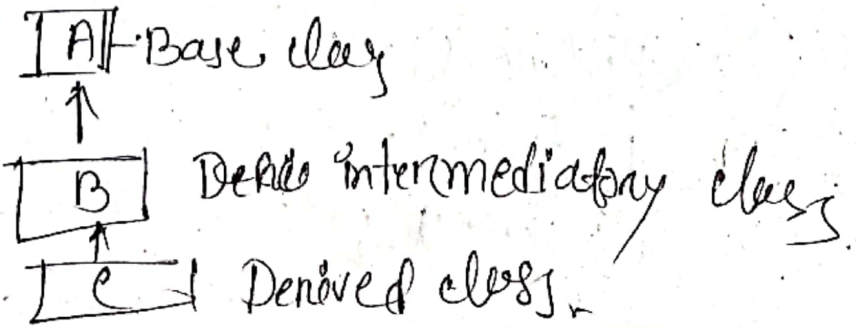
```
public class One {
    public void print_Java() {
        S.O.P("Java");
    }
```

```
preblic class two extends one {
    Public void print_P() {
        S.o.P ("Programming");
    }
}

Public class Main {
    P.S.V.M() {
        Two P = new Two();
        P.print_Java;
        P.print_P();
    }
}
```

output: Java
programming

2. Multilevel inheritance:

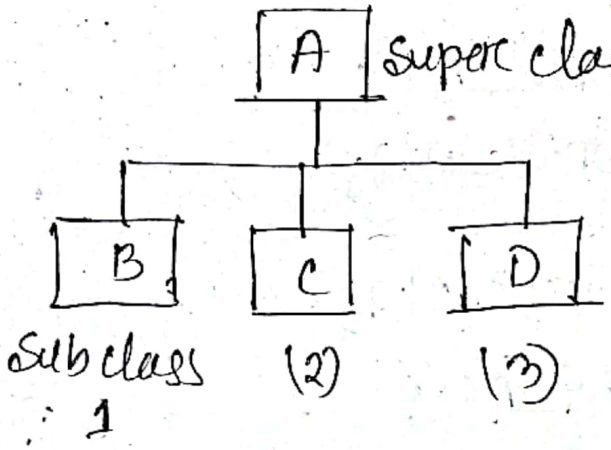
A derived class will be inheriting a base class and as well as a derived class also acts as the base class for another classes.



```
public class One {
    public void print_one() {
        S.O.P("One");
    }
}

class Two extends One {
    public void print_two() {
        S.O.P("Two");
    }
}

class Three extends Two {
    public void print_three() {
        S.O.P("Three");
    }
}

Public class Main {
    P.S.V.M() {
        Three p = new Three();
        p.One();
        p.Two();
        p.Three();
    }
}
```

output:
One
Two
Three

3. Hierarchical Inheritance: One super class for more than one sub class.



example:

```
class A{
    Public void Print_a(){ S.O.P("class A");}
}
    extends A
class B{
    public void print_b(){S.O.P(" class B");}
}
    extends A
class C{
    public void print_c(){S.O.P("          ");}
}

class D extends A{
    Public void print_D(){S.O.P("class D");}
}
```

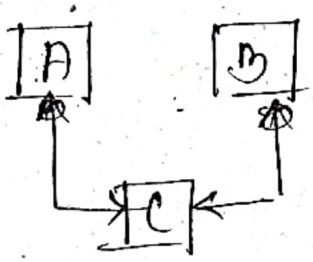
```
Public class main{
    public static void main(String [] args){

        B obj_B = new B();
        obj_B.print_A();
        obj_B.print_B();

        C obj_c = new C();
        obj_c.print_A();
        obj_c.print_C();

        D obj_d = new D();
        obj_d.print_A();
        obj_d.print_D();
    }
}
```

output:

```
class A
class B
class A
class C
class A
class D
```

Multiple : In multiple inheritance, several super class for one subclass.

In Java multiple inheritance is not supported.



~~using interface multiple inheritance~~

In Java, using interface we can achive multiple inheritance.

example :

```
interface One{
        p.v. print_one();
}
interface Two{
        P.V.print_two();
}
interface Three extends One, two{
        P.V.print_one();
}
```

```
class child implements Three {
    @Override
    public void print_one() {
        S.o.p("One");
    }
    public void print_two() {
        S.o.p("two");
    }
}

Public class Main {
    public static void Main (String[] args) {
        child c = new child();
        c.print_one();
        c.print_two();
        c.print_one();
    }
}
```

Output:
One
Two
One

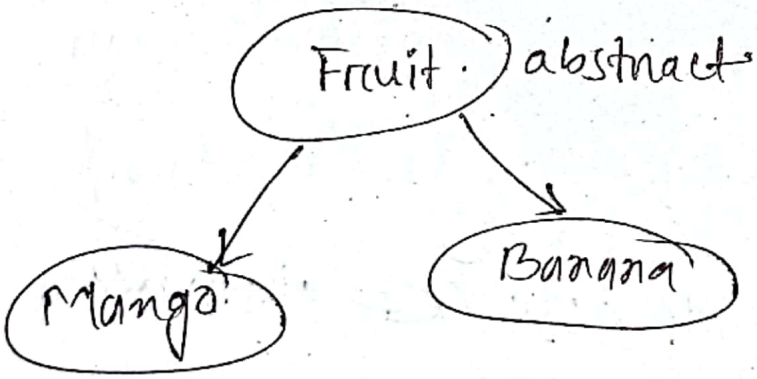
When using protected:-

- your public & protected members of base class become protected members in derived class. It isn't a stop to inheritance as you can go further.

Abstract class: → object তৈরি করা যাবে না

→

Abstract class is a technique of hiding unnecessary details from the user.



Data abstraction is defined as the process of reducing the object to its essence so that only the necessary characteristic are exposed to the users.

Abstraction:

→ Is the process of hiding the implementation details & showing only the functionality to the user.

→ body নাই এমন method.

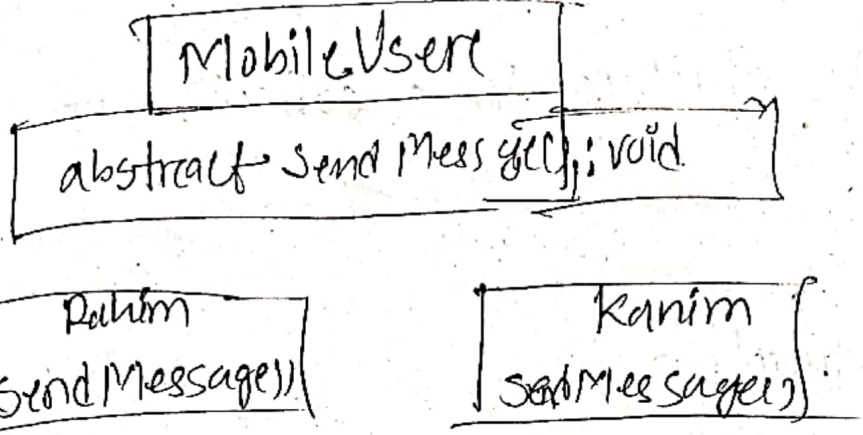
abstract void message();

Points:—

- has no body
- It must be ends with a semicolon.
- It must be in the abstract class.
- It must be overridden.
- It can never be final and static.

Ex: abstract class MobileUser {

```
    void call() {
        sout("Hello");
    }
    abstract void sendMessage();
}
```



Code:

```
abstract class MobileUser {
    abstract void sendMessage();
}

class Rahim extends MobileUser {
    @override
    void sendMessage() {
        sout("Hi!, This is Rahim");
    }
}

class Karim extends MobileUser {
    @override
    void sendMessage() {
        sout("Hi, This is Karim");
    }
}

public class Test {
    psvm() {
        MobileUser ms;
        ms = new Rahim();
        ms.sendMessage();
        ms = new Karim();
        ms.sendMessage();
    }
}
```

Interface: A Java interface is a collection of constants and abstract methods.

— alternate approach to support multiple inheritance

Syntax:

```
public interface InterfaceName {
    public static final String NAME;
    public void doThis();
    public int doThat();
    public void doThis2(float value, char ch);
    public boolean doTheOther(int num);
}
```

~~extends~~

why use:

— It is used to achieve abstraction

— By interface we can support the functionality of multiple inheritance.

— It can be used to achieve loose coupling.

class Vs Interface

class	Interface
The members of a class can be constant or variable.	Always declared as constant their values are final.
The class definition can contain the code for each of its method. That can be abstract or non-abstract	The methos of interface in an are abstract in nature.
It can be instantiated by declaring objects.	It cannot be used to declare objects. It can only ~~use~~ ~~they~~ be ~~public~~ acc inherited by a class.
It can use various access ~~modifier~~ specifier	It can only use the public access ~~m~~ specifier.

Abstract class.	Interface
1. can contain both abstract & non-abstract methods.	Interface contains only abstract methods.
2. An abstract class can have all four, — final, non-final, static non-static variables.	Only final & static variables are used.
3. To declared abstract class abstract keyword are used.	4. interface keyword are used.
4. ~~It does support~~ It doesn't supports multiple inheritance	4. It support multiple inheritance
5. ~~Fully abst~~ Partial abstraction	5. Fully abstract.

6. abstract class Animal {
    abstract void eat();
}

6. interface Animal {
    void eat();
}

abstract class Animal {
    abstract void sound();
}
→ cannot be instantiated directly and is meant to be subclassed.

class Dog extends Animal {
    void sound() {
        System.out.println("Bark");
    }
}

final class MathConstants {
    public static final double PI = 3.14159;
}

- To create class constant
- To prevent inheritance
- To prevent method overriding

* A class marked as final cannot be subclassed

Output: Name: John
Age: 30

* interface vs abstract class with example.

Example of interface:

```
interface Person3
    void setName(string name);
    string getName();
}
```

```
class Student implements Person {
    private String name;

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
}

public class Main {
    public static void main(String[] args) {
        Student student = new Student();
        student.setName("Alice");
        S.O.P("Name:" + student.getName());
    }
}
```

Output: Alice.

example of abstract class:

```
abstract class Person {
    string name;
    public Person(string name) { // constructor
        this.name = name;
    }
    abstract void displayInfo(); // abstract method.
    public string getName() { // non-abstract method.
        return name;
    }
}
class Student extends Person {
    public students (string name) { // constructor.
        super(name);
    }
    public void displayInfo() { // Implement the abstract method.
        S.O.P("Name:" + name);
    }
}
```

```
Public class Main {
    P.S.V.m() {
        Student student = new Student("Israt");
        student.displayInfo();
    }
}
```

Output: Israt

Abstraction & Encapsulation:—

In object oriented Programming (OOP)

Interface: An interface is a collection of abstract method

why need:—

→ fully abstraction

→ It supports multiple inheritance

Method এর public কেন

→ method in an interface are implicitly public

→ A class can inherit ~~from~~ from just one superclass, but can implement multiple interfaces.

→ class কেন class এর or interface & interface কে extends করে।

class কেন interface কে implement করে।

eg: interface animal {

void eat(); or public abstract void eat();

}

class Dog implements Animal {

public void eat() {

Sout("Dogs can get eggs");

}

}

Inheritance Vs Polymorphism

Inheritance	Polymorphism
1. Inheritance is one in which a new class is created (derived class) that inherits the features from the already existing class (Base class).	1. Where as polymorphism is that which can be defined in multiple forms.
2. It is basically applied to classes.	2. whereas it is basically applied to functions or methods.
3. Inheritance supports the concept of reuseability and reduces code length in OOP.	3. polymorphism allows the object to decide which form of the function to implement at compile-time as well as run-time.
4. Inheritance can be Single, hybrid, multiple, hierarchical and multilevel inheritance.	4. whereas it can be compiled-time polymorphism as well as run-time polymorphism.
5. It is used in pattern designing	5. while it is also used in pattern desinging.

Encapsulation:

The Encapsulation in java is a process of wrapping code and data together into a single unit. Known as a class.

Advantage of encapsulation:

1. Data protection:

Encapsulation provide a mechanism for securing data within a class, as it prevents outside code from accessing the data directly.

2. code reusability:

By encapsulating data & methods within a class, it is easier to reuse the code in other parts of program.

3. Abstraction: Encapsulation allows the programmer to hide the implementation details of a class & provide a high-level abstraction of its functionality.

4. Flexibility: Encapsulation makes it easier to modify & update the implementation details of a class, as the interface remains the same.

5. code maintainability:

Encapsulation can improve the maintainability of the code by making it easier to locate and fix bugs or issues.

Example of encapsulation:

```
class person {
    private string name;
    private int age;

    public string getName() {
        return name;
    }
    public void setName(string name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
}
```

```
Public class Main {
    Public static void main(String[] args) {
        Person person = new Person();
        person.setName("John");
        person.setAge(30);

        S.O.P("Name:" + person.getName());
        S.O.P("Age:" + person.getAge());
    }
}
```

Output: Name: John
Age: 30

interface vs abstract class with example.

Example of interface:

```
interface Person {
    void setName(String name);
    String
```

Encapsulation Vs Abstraction

Encapsulation	Abstraction
1. It is the process or method for containing information.	1. It is a process or method of obtaining information.
2. During encapsulation, issues are resolved at the implementation level.	2. Abstraction solve problems at the design or interface level.
3. The focus of encapsulation is 'how' it should be done.	3. The focus of abstraction is 'what' to do.
4. It is a way of hiding data in a single entity and a way of protecting information from the outside world.	4. Abstraction is a way of hiding unnecessary information.
5. It can be implemented using access modifiers, private, protected, public	5. Abstract classes and interfaces can be used to implement abstractions.

6. It hides data from direct access by users	6. Abstractions allow access to specific pieces of data

Dinamic binding	Message passing
Dinamic binding is a mechanism where the specific method to be executed is determind at runtime rather then at compile time.	Message passing is a concept where the object communicate by sending message to each other.
It allows execating different codes using the same object at runtime	It allows devloping communication between object.

```
Public class Main {
    P.S.V.m() {
        Student student = new Student("Israt");
        student.displayInfo();
    }
}
```

Output: Israt

Abstraction & Encapsulation:—

⇒ In object oriented Programing (OOP), abstraction and encapsulation are both fundamental concepts that involve data hiding, but they serve different purpose and are implemented differently.

* Explanation & Example.

- Abstraction deals with the design-level representation, focusing on hiding the implementation complexity and showing only necessary features.
- Encapsulation deals with implementation, focusing on binding the internal states and requiring

all interactions to be performed through well-defined method.

Abstract class code:

```
abstract class Animal {
    private string name;
    public void setName(string name) {
        this.name = name;
    }
    public string getName() {
        return name;
    }
    public abstract void eat();
}

class Dog extends Animal {
    @override
    public void eat() {
        S.O.P("Dog is eating");
    }
}
```

```
Dublic class Main {
    P.S.V.M( ) {
        Dog dog = new Dog ();
        dog.setName ("Buddy");
        S.O.P("Dog Name: " + dog.setName());
        dog.eat();
    }
}
```

output: Dog's Name: Buddy
Dog is Eating

→ To avoid the ambiguity
→ To reduce the complexity and simplify language

Super:

super keyword is used to refer immediate super class object.

uses:

1. It is used to call super class instance variable
2. It is used to call super class method. (override method).
3. It is used to call super class constructo

Example (i)

```
public class A {
    void display() {
        sout ("Inside A class");
    }
}
```

```
public class B extends A{
    @override
    void display(){
        super.display();
        sout("Inside B class");
    }
}

public class Test{
    public svm(String[] args){
        B ob = new B();
        ob.display();
    }
}
```

Output: Inside A class
Inside B class

(ii) public class A{
```
    A(){
        sout(" A is constructor");
    }
}

public class B extends A{
    B(){
        super(parameter if any);
    }
```

```
        Sout("B is a constructor");
    }
}
public class Test{
    psvm(s->){
        B ob = new B();
    }
}
```

Output: A is cons - -
B is - - -

(i)
```
public class A { int x = 10; }
public class B extends A{
    int x = 5;
    void display(){
        Sout(super.x);
    }
}
public class Test{
    psvm(String[] args){
        B ob = new B();
        ob.display;
    }
}
```

output: 5

Polymorphism:

- compile Time/ static polymorphism
  - method overloading
  - constructor overloading
- run time/ dynamic
  - overriding

Polymorphism is a mechanism where a parent class reference variables can take many forms. (It can refer object from different classes)

Ex: main ( ){

```
Shape s1 = new Shape();
Shape s2 = new Rectangle(10,20);
Shape s3 = new Triangle(10,20);

Sout(s1.area());
Sout(s2.area());
Sout(s3.area());
```

```
Shape[] = new Shape[3];
s[0] =
s[1] =
s[2] =

Sout(s[0]. —
Sout(s[1]. —
Sout(s[2]. —
```

Polymorphism allows a single interface to represent different types. This achieved through method overriding ( ) or method overloading.

What is JVM ? Explain the internal architecture of JVM with neat sketch

⇒ JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

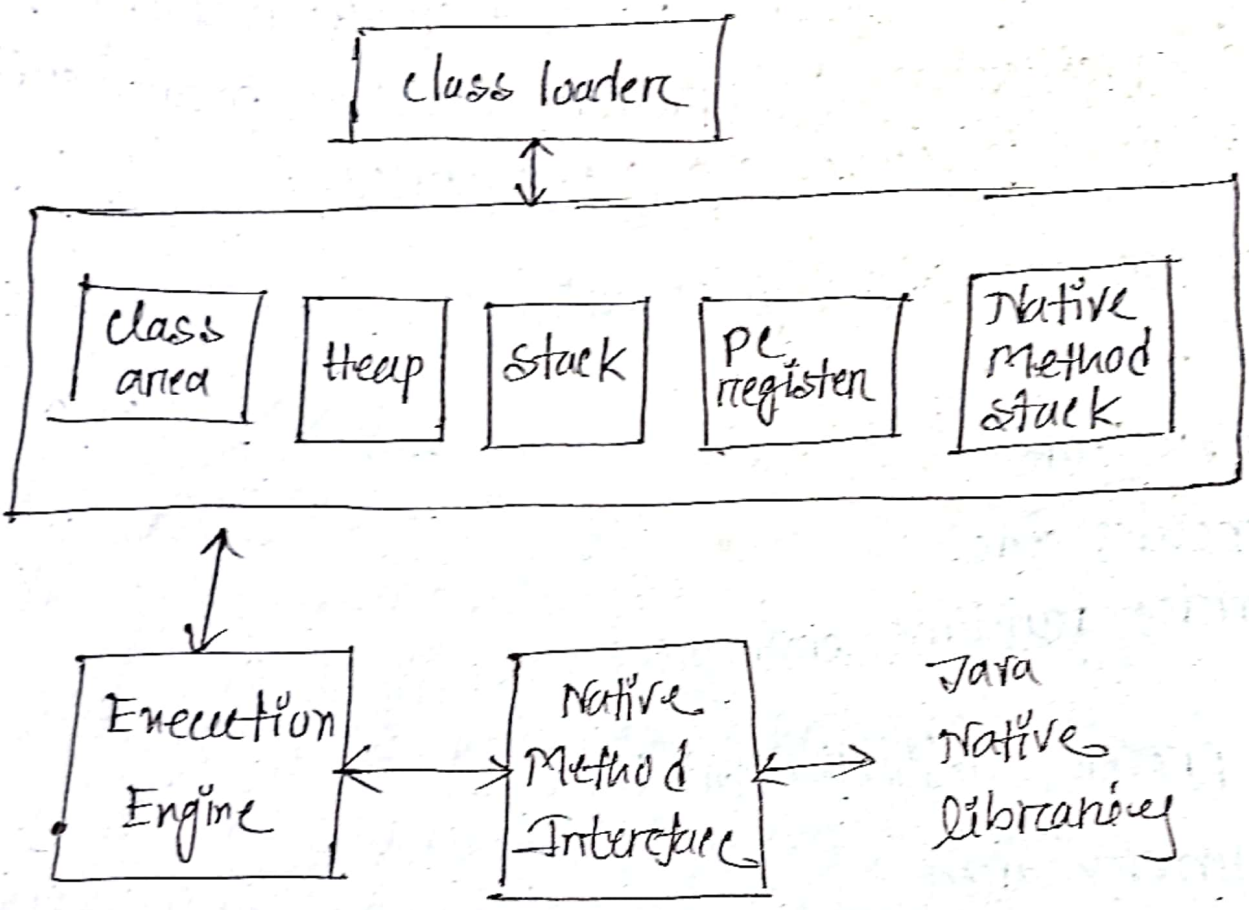
The JVM performs following operations:

1. loads code
2. Verifies code
3. Executes code
4. provides runtime environment

JVM provides definition for the:-

1. Memory area
2. class file format
3. Register set
4. Garbage-collected heap
5. Fatal error reporting etc.

JVM architecture:-



1. class loader: classloader is a subsystem of JVM which is used to load class files. whenever we run the java program, it is loaded first by the classloader. There are three built-in classloader in Java.

- Bootstrap
- Extension
- System/ Application.

2. Class Area: class area stores per-class structures such as the runtime constant pool, field & method data the code for methods.

3. Heap: It is the runtime data area in which objects are allocated.

4. Stack: Java stack stores frames. It holds local variables and partial results & plays a part in method invocation and return.

5. Program control register:

PC register contains the address of the JVM instruction currently being executed.

6. Native Method stack:

It contains all the native methods used in the application.

7. Execution Engine;

It contains,

- A virtual processor
- Interpreter
- Just in Time compiler

8. Java Native Interface:

Java native interface is a framework which provides an interface to communicate with another application written in another language like C, C++, etc.

Wrapper class:

wrapper class provide a way to use primitive data types as objects.

wrapper class provides the mechanism to convert primitive into object & object into primitive.

these wrapper classes are part of the java.lang package. Java provides a wrapper class for each of the eight primitive types:

~~Boolean wraps the boolean primiti~~

Primitive Data type	wrapper class
1. char	Character
2. byte	Byte
3. short	Short

4. int	Integer
5. long	Long
6. Float	Float
7. double	Double
8. boolean	Boolean

Autoboxing: Autoboxing is the automatic conversion of a primitive type to its corresponding wrapper class object.

Example: conversion of int to Integer, long to Long etc

```
import java.util.ArrayList;
class Autoboxing{
    P.S.V.M( ){
        char ch='a';
        Character a=ch;
        ArrayList<Integer> arrayList
            =new ArrayList<Integer>();
        arrayList.add(25);
```

```
        S.O.P(arrayList.get(0));
    }
}
```

output : 25

Unboxing: Unboxing is the automatic conversion of a wrapper class object to its corresponding primitive type.

Example: Long to long, Double to double etc,

```
import Java.util.ArrayList;
class Unboxing {
    P.S.V.M() {
        Character ch='a';
        char a=ch;
        ArrayList<Integer> arrayList = new ArrayList<Inte
        arrayList.add(24);
        int num = arrayList.get(0)
    } S.O.P(num);
```

output: 24

purpose of Java's wrapper class:

1. object storage in collections: Allow primitive values to be used in collections like ArrayList.
2. Utility Methods: Provide method for conversions & comparisons.
3. Handling Null values: can hold null, unlike primitive
4. Type saftey in Generics: Enable the use of generics with types corresponding to primitives.
5. Autoboxing/unboxing: automatic conversion between primitive to wrapper objects.
6. Enhanced Type Information:
7. Immutability: Wrapper object are immutable,
8. Compatibility: work with legacy code that requires objects.

ensuring the integrity of values.

* List of all the topics of Java discussed in the lab work:-

1. Basic of Java syntax & data types
2. Control structures (if-else, loops)
3. Object Oriented Programming concepts
   (i). class & objects
   (ii). Inheritance
   (iii). Encapsulation
   (iv). polymorphism
   (v). Abstraction
4. Method & constructor
5. Exception handling
6. File I/O Operations
7. package, Interface

* Oop paradiagram: