Date:
Page:

Programming Paradigm :-

A paradigm can be termed as a method to solve a problem or accomplish a task.

→ A programming paradigm is an approach to solve a problem using a specific programing language.

Two of the most common programming paradigms are :-

1. Object oriented programming
2. Procedural programming

(1) Procedural Programming

Procedural programming follows the concept of calling procedures also called as functions / Routines or subroutines, which consist of a series of steps that need to be carried out.

→ Procedural Programming language :- BASIC, FORTRAN, COBOL, PASCAL, C.

→ There is no concept of class & objects.
→ No concept of Inheritence, Encapsulation etc.
→ We only use functions.

Date:
Page:

(2) Object oriented programming :- It is a programming paradigm based on the concept of Object which may contain data often K/A attributes and code often K/A methods.

→ The aim of OOP is to implement real life concepts like
→ Inheritence
→ Encapsulation
→ Abstraction
→ Polymorphism
etc. in programing.

Examples of OOP language :- Java, Python, C++, C#, PHP, scala, Dart etc

Procedural Prg.	object oriented prog.
① In procedural programing the program is divided into small parts called function.	① In OOP the program is divided into small parts called objects.
② It follows a top-down approach.	② It follows a bottom-up approach.
③ Under procedural programing there is no access specifier.	③ OOPs has access specifiers like - Private, Public, Protected.
④ It does not have any proper way of hiding data so, it is less secure.	④ OOP provide data hiding so, it is more secure.
⑤ overloading is not posible	⑤ overloading is possible.
⑥ There is no concept of Data hiding and Inheritence	⑥ The concept of data hiding and inheritence is used.
⑦ Code reusibility is not present.	⑦ Code reusability is present in OOPs.
⑧ Procedural Prg. is based on unreal world.	⑧ Based on real world.

Date:
Page:

Advantages of OOPs over procedural programming :-

① → By using objects and classes we can create reusable components leading to less duplication & more efficient development.

② → It provides a clear & logical structure.

③ → OOPs support DRY (Don't Repeat Yourself) Principle.

Disadvantages of OOPs :-

→ ① Since OOPs uses the concepts like (classes, objects, inheritance etc.) for beginner this can be confusing & takes time to learn)

② If we have to write small program using OOP it can feel too heavy.

③ → OOPs creates a lot of objects, so it can use more memory compare to simple programs.

History of OOPs :-

→ 1960 :- First concept related to OOP began to emerge in 1960s with the creation of simula language developed by de-Johan Dahl & Kristen Nygaard at Norwegian computing Centre.

→ Simula67 introduced concepts of classes, objects & inheritence.

1970s :- Alan Kay, a researcher at Xerox PARC coined term "object oriented" to describe a programming approach based on simulating

Teacher's Signature

C++ → C with classes + ed

Date: 4 Feb
Page:

biological systems. Kay developed Smalltalk prog. language.

→ 1980s :- Witnessed the growth & popularization of OOP.
Bjarne Stroustrup at AT&T Bell Labs developed C++ (a language that evolved from C language with addition of object-oriented features).

→ Brad Cox and Tom have developed objective-C combining object-oriented capabilities of Smalltalk with C-programming language.
Primary language for software development on Apple platforms.

→ 1990s :- Java came, developed by Sun Microsystems (currently part of oracle) and designed by James Gosling & team.

→ From same era comes Python, created by Guido van Rossum.

→ 2000s C# (C sharp) developed by Microsoft as part of its .NET platform.
Influenced by C++ and Javascript. It has been the primary language for developmen on .NET platforms including desktop, web & mobile apps.

Teacher's Signature

Date:
Page:

2015 :- with introduction of ES6 (ECMA script 2015), Javascript become more object-oriented increasing its use in frontend & Backend development.

Present :- Go, Dart, Rust languages are constantly emerging & craining popularity.

Object oriented programming :- OOP in C++ was introduced for organizing codes into objects and classes, thus making programs easier to understand, reuse and maintain.

↳ ~~It uses classes and objects~~

(1) Structured code into logical units by using objects and classes.

(2) Makes code Modular, Reusable and Scalable. (Chunky)

(3) It prevents unauthorized access to data.

(4) It follows Dry principle.

Teacher's Signature

Date:
Page:

* Classes :- Class is a user-defined blueprint or prototype from which objects are created. It Represents set of properties or methods that are common to all objects of one type.

Eg:- Consider the class of car.
There may be many cars with diff. names and brands but all of them will share some common properties like all of them will have 4 wheels, speed limit, mileage, Range etc., and have functions like Start, accelerate, forward, backward etc.

→ Declaration structure of a class :-

```
class className
{
    Access specifier :   (public, private or protected)
    {
        Data members / Attributes
        Member function / Methods
    };
```

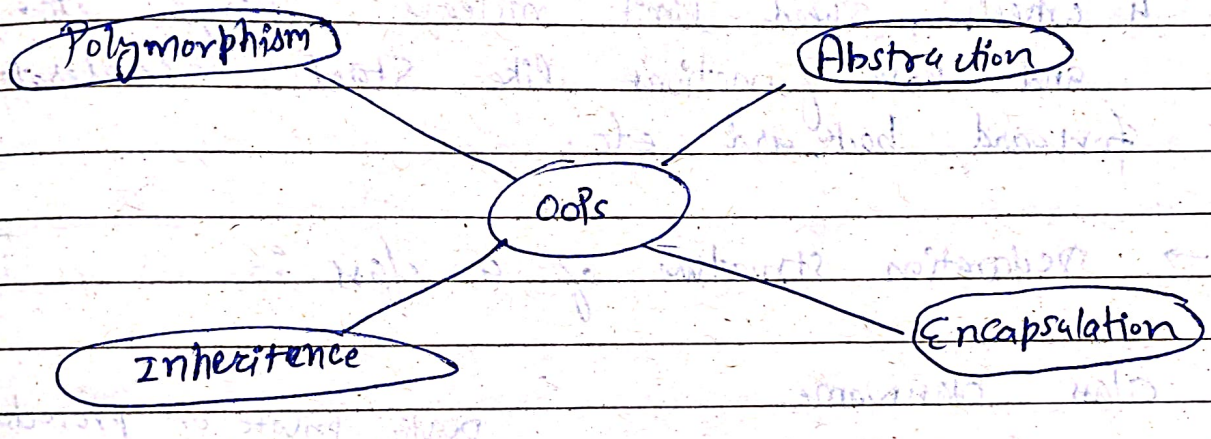
Objects :- Objects Represent the Real life Entities, it is an instance of a class.
When a class is defined no memory is allocated but when it is instanciated, i.e. an object is created then a memory is allocated.

Teacher's Signature

Date:
Page:

An object has identity, state, and behaviour.
Eg:- Dog is an object (real life example).
Identity :- Name of Dog.
State :- Breed, Age, Color.
Behaviour :- Bark, sleep, Eat.

4 Pillars of OOP :-



(1) Abstraction :- Data Abstraction Refers to providing only essential information about an object to the outside world, hiding the background details or implementation.

→ It allows to focus on 'what an object does' rather than 'how it does'.

→ In C++, abstraction is achieved using abstract classes (Abstract classes are those that have atleast one pure virtual function).

Teacher's Signature

Date:
Page:

for Example :- A Man driving a car, the man only knows that pressing the accelerator will only inc. the speed of car or applying break will stop it.
But the man does not know, how on pressing the accelerator, the speed is increasing, the inner implementation (mechanism) is hidden from the man, this is what Abstraction is.

(2) Encapsulation :- It is defined as wrapping up of data and methods into a single unit (class), thus restricting direct access to its components (data members).

→ It ensures data integrity and controlled access

for Example :- Suppose in a company there are different sections, like account, finance and sale. The finance section handles all the financial transactions and keeps all data related to finance.
Similarly sales and account section maintains all the data.
Now, when someone from finance section needs the data about sales in a particular month then, the person can not directly access the sales data.
→ He will first contact some officers in sales and request him to save him data. this is what Encapsulation is.

Teacher's Signature

Date:
Page:

(3) Inheritance :-

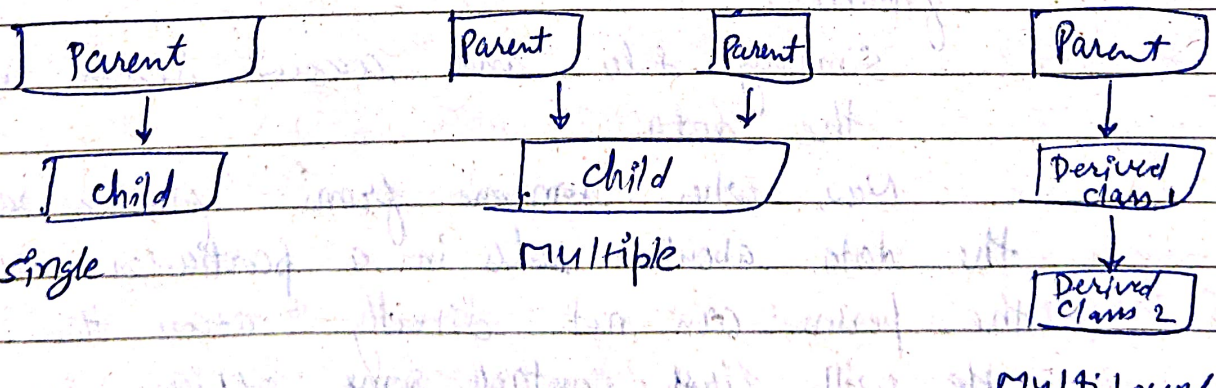
→ The capability of a class to derive properties & characteristics from another class is ~~called~~ class is called Inheritence.

→ Whenever we write or create we don't need to write function again & again as these can be inherited from another class that posses it.

→ Inheritence allows user to reuse the code whenever possible & reduce its redundancy.

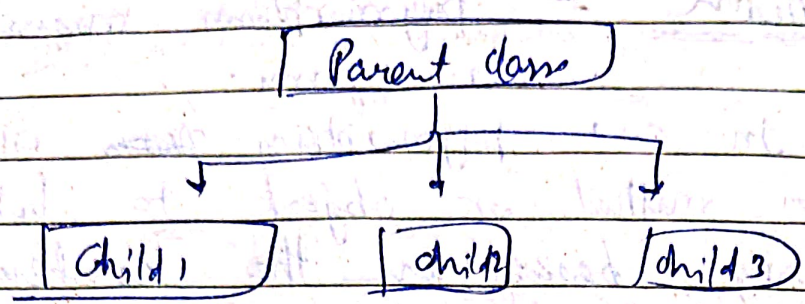
For Examples :- There is one base class car. Derived class could be a petrol car, Diesel car, or EV car, but all can have same number of seats and other same features inherited from parent (base) car class.

Types of Inheritance

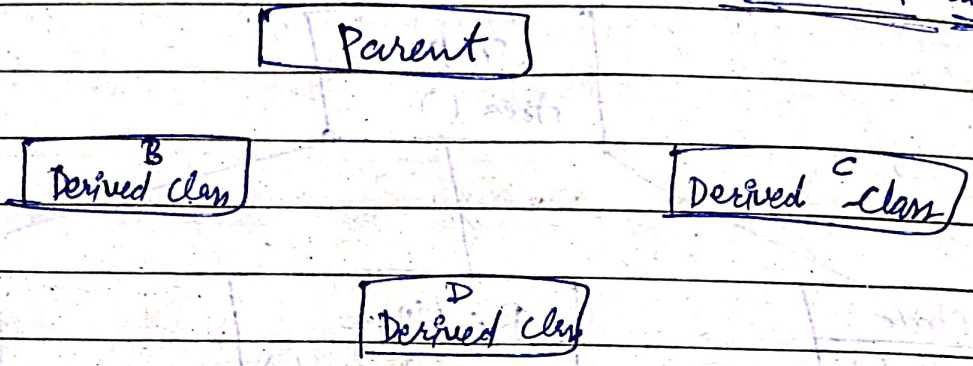


Teacher's Signature _____

Date:
Page:



Hierarchical clustering Inheritance



Hybrid

① Single Inheritence :-
A Derived class inherits from only one base class.

② Multiple Inheritence :- A Derived class inherits from multiple base class.

③ Multi-level Inheritence :- Inheritance forms a chain where a class ~~derived~~ Inherits from a derived class, which in turns inherit from another class.

④ Hierarchical clustring Inheritence :- Multiple derived classes Inherits from a single base class.

⑤ Hybrid Inheritence :- A combination of two or more of the above types.

Teacher's Signature

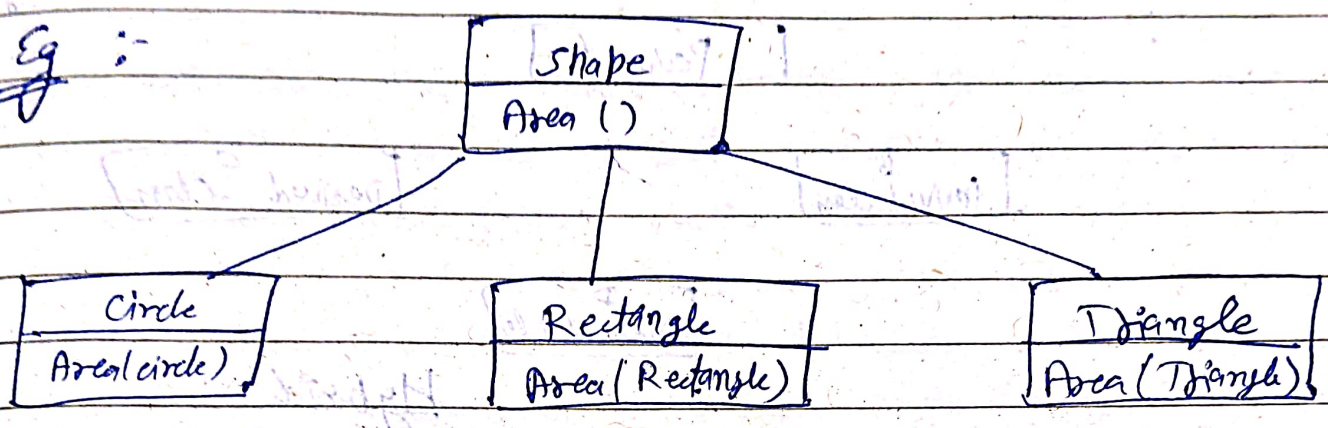
Date:
Page:

many → forms

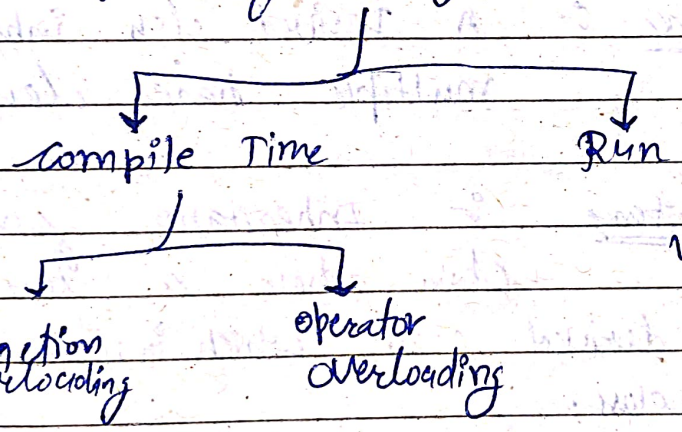
(4) Polymorphism :- polymorphism means having more forms.

→ In C++ polymorphism allows the same method or object to behave differently, based on the context.

for Eg :-



Types of Polymorphism



Compile time :- In compile time the method call is resolved during program compilation.

Run time :- In Run time the method call is resolved during program Execution.

Teacher's Signature

Date:
Page:

Diff. kinds of behaviours (we can change behaviours of operators like +, -, *, / etc.)

overloading :- Term overloading have two or more distinct meanings.

Virtual function :- A virtual function is a member function that is declared within a base class, using the keyword 'virtual', and is redefined in the derived class.

→ Also called method overwriting.

Need of OOPs :-

① To make development and maintainence of project more Effortless.
② To provide the feature of data hiding, that is good for security concerns.
③ Ensures code reusability.
④ We can solve real world problems easily.

Key application areas of OOPs :-

① GUI Base applications uses oops Extensively
eg:- Each component such as textbox, button, window, menus is treated as an object.

② Game development, creates Enemy, person, vechicles are treated as objects.

③ Mobile apps development.
④ Software development

Teacher's Signature

Date:
Page:

(5) Object oriented database - The database try to maintain direct ~~contact~~ correspondance between the real world and database object in order to ~~have~~ let the object retain its identity & ~~integrity~~ integrity.

Classes and objects :-

```
class className {
    Private :
        data-member
        member function
    Protected :
        —
        —
    Public :
        —
        —
};
```

Class :- The declaration of a class involves the declaration of its & associated attributes.

(i) Data members :- These are data type properties that describes the characteristics of a class.

Date:
Page:

(ii) Member function :- These are set of operations that can be applied to class objects.

→ They are also refer to class interface.

(iii) Access-specifiers :- These control access to members from within the program.

(iv) Class Tag Name :- It serves as a type-~~specifier~~ classifier for the class.

Eg :-

```
class Account {
    Public:
    int account no.;
    char type;
    float balance;

    float deposit (float amount) {
        balance += amount;
        return balance;
    }
    float withdraw (float amount) {
        balance -= amount;
        return balance;
    }
```

Teacher's Signature

Date:
Page:

Eg:- class student {
Public:
int Enrollment_number;
string branch;
string Department;
string Name;
int Marks;

float avg_marks (int m1, int m2, int m3)
marks = $\frac{m_1 + m_2 + m_3}{3}$;

return marks; }

class Method Definition :- Methods can be defined inside a class as well as outside the class.

Eg:- (i) the previous class & function definitions were inside.

(ii) outside class definition :-

return_type Classname :: function_name (Parameter list)
↓
scope resolution

{ body
}

Teacher's Signature

Date:
Page:

→ Scope resolution (::):- It resolves the identity and scope of a function that is restricted to class name.

Inside the class definition :-

When a member is defined inside the class it is known as inside the class definition

for creating an object :-

Class_name obj_name ;

Eg:- Student O_1, O_2, O_3 ——— ;

to access the member functions & attributes :-

obj_name . attribute = value ;

Eg:- O_1 . Sname = ___ ;
O_1 . marks = ___ ;
O_1 . age = ___ ;

Teacher's Signature

Date:
Page:

Global class and local class :-

① Global class :- A class is said to be global if its definition occurs outside the ~~body~~ bodies of all function in a program.

Eg:- class x
{ ——
——
}
x obj1; → Global obj

int main ()
{
x obj2; → local obj
}
void funct1 (void)
{ x obj3; → local obj.
};

Teacher's Signature _____

Date:
Page:

② Local class :-

Eg:-
```
int main() {
    class Y
    { ___
    };
Y obj1;
}
```

Eg:-
```
class X
{ Public:
    int a;
    Void fc(Void);
};

X obj1;
int main()
{
    obj1.a = 10;
    obj1.fc();
}

Void func1(Void)
{
    obj1.a = 20;
    obj1.fc();
}
```

In this all these are valid statement as the obj1 is global and class X is public so accessible to either side programmer.

Date:
Page:

Eg:-

```
class X
{ Public :
    int a;
    void fc (void);
  };
int main()
{ class Y
  { Public :
    int i;
    void fun(void);
  }

  X obj 1;      → local   } inside int main()
  Y obj 2;      → local
  obj1. a = 5;
  obj1. fc();
  obj 2. i = 15;
  obj 2. fun();
}

void func1 (void)
{
  X obj 3;      ✓
  Y obj 4;      ✗ (local class Y)
  obj 3. a = 25;    ✓
  obj 3. fc();      ✓
  obj 1. a = 10;    ✗ (local object obj1)
  obj 2. a func();  ✗ (local object obj2)
  obj 4. i = 15;    ✗ (local class attribute)
}
```

Teacher's Signature _____

Date:
Page:

Global & local objects :-

Acces Modifiers :- These control the visibity of class members, they help in implementing the data hiding by restricting or allowing the access to certain part of a class.

There are 3-types of Acces Modifiers.

① Public
② Private
③ Protected.

① Public :- Public makes class members accesible from anywhere, i.e. Inside the class, outside the class and even other files.

② Private :- This makes class members accesible only inside the class itself and is mainly used to hide data from outside. By default all class members in c++ are private.

→ Private members can only be used by member of that class or friend function of that class.

Teacher's Signature

Date:
Page:

```
class Employee {
    Private :
        double Salary ;
        int empID ;
        Void Min_salary ();

    Public :
        string name ;
        Void incs_projects ();
};
Employee obj ;
Int main ()
{
    obj . emp ID = 3412 ;        → Not valid (Private)
    obj . min Salary () ;        → not valid
    obj . name = " XYZ " ;       → valid (Public)
}
obj . inc_projects () ;          → valid (Public)
```

Teacher's Signature _____

Date:
Page:

③ Protected :- Protected Specifier makes members accessible inside the class itself and its derived classes (child classes), but not from outside the code.

It is mainly used in inheritence, allowing child to modify data and functions while keeping them hidden from outside code.

```
class CSE_4sem {
    Public :
        string Name;
        string Rollno;
        float m1, m2, m3;
        void avg (int m1, int m2, int m3){
            int avg. avg = (m1+m2+m3)/3;
            Cout << " avg. marks are " << avg;
        }
CSE_4sem S1, S2, S3, S4, S5;
int main () {
    Cout << " To Calculate avg. Marks ";
    for (int i=0; i<5; i++)
    { Cout << Enter marks of three subjects ";
        int m1, m2, m3;
        Cin >> m1 >> m2 >> m3;
        S[i].avg(m1, m2, m3); }
    return 0;
}
```

Teacher's Signature

Date:
Page:

```
class CSE_4sem {
    Private :
    string Name;
    string Rollno;
    float m1, m2, m3;
    Public :
        float avg (int m1, int m2, int m3)
        { float average = (m1 + m2 + m3)/3;
          return average;
        }
};

CSE_4sem S[5];

int main () {
    Cout << "To calculate average of 3 marks";
    float mani = INT_MIN;
    int m1, m2, m3; int flag = 0;
    for( int i=0 ; i<5 ; i++) {
        Cout << "Enter Marks of 3 subjects";
        cin >> m1, m2, m3;
    float avg = S[i].avg( m1, m2, m3) ;
    Cout << " The average marks of 5th
            are" << avg;

    mani = max( mani, avg ) ;
    if (mani == avg) flag ++;
    }
    Cout << mani ;
    Cout << S[flag].Name;
    Cout << S[flag].Roll No;
    return 0;
}
```

Teacher's Signature _____

Entity ID Project Project-name Department

arr.size()

Date:
Page:

Nested classes :- A class may be declared within another class, it is called nested classes.

The outer class is known as Enclosing class and the inner class is known as Nested class.

Eg:- Syntax :-

```
class X {

        class Y {

        };
};
class Z {

        class Y {

        };
};
```

scope resolution operator

```
X::Y func();

Z::Y func();
```

Teacher's Signature

Date:
Page:

```
class Employee {
    Private :
    string Name ;
    string ID ;
    string Department ;
    string prj[5] ;
    Public :
    int total_projects (string arr[])
    { return proj.size(); }
}

Employee E[5] ;

int main () {
    Cout << Enter Details according";

for( int i=0 ; i<5 ; i++)
    { Cout << Enter name of E[i] ;
      Cin >> E[i].Name ;
      Cout << "Enter ID" ;
      Cin >> E[i].ID ;
      Cout << "Enter Department" ;
      Cin >> E[i].Department ;
      Cout << " enter the projects" ;
      for( int j=0 ; j<5 ; j++)
        { Cin >> E[i].prj[j] ;
        }
}
```

Teacher's Signature

Date:
Page:

```
Cout << " Details of Employees " ;
for (int i=0; i<5; i++)
  {
    Cout << E[i].Name
    Cout << "ID:" << E[i].ID;
    Cout << Department: " << E[i].Department;
    for( int j=0; j<5; j++)
      { Cout << "Projects:" << E[i].Prj[j] << "," ;
      }
    cout << "total no. of Projects " << "5";
  }
  return 0;
  }
```

Teacher's Signature