Object Oriented Programming.

Date: / / Page no:

⇒ Need of OOP, features

i) Programming Techniques

- i) Unstructured Programming ⇒ Only one program i.e. main program
- ii) Procedural / Functional Programming ⇒ For each task procedure (function) is created
- iii) Modular Programming ⇒ Procedures with some common functionality are grouped together into separate modules.
- iv) Object-Oriented Programming ⇒ Works on objects which is considered smallest unit of OOP.

⇒ Limitation of Functional Programming

i) Poor real world model
ii) No privacy, no true reuse.

⇒ Functional vs OOP	Functional	OOP
i) Divided Into ⇒	i) Functions	objects
ii) Importances ⇒	i) data as well as sequence of action to be done	data ; works on real world
⇒ Approach ⇒	Top Down	Bottom Up.

Functional OOP

Data Hiding ⇒ less secure more security

⇒ Generic Programming
- code should be generic as possible
- Data types are not specified at the time of implementation.
- Use tem writing templates or standard template ~~library~~ library

1) Object Oriented Features

i) Class ⇒ User defined data types which has ~~its~~ its own data member & member function (method)
A c++ class is like a blueprint for an object.

ex ⇒
```
class Box {
        int length;
    Public: getLength();
};
```

ii) Object ⇒ Object is variable of class type. or object is an instance of the class

Note ⇒ When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

iii) Encapsulation :- Wrapping up of data & function into a single unit (class class) is known as encapsulation

Encapsulation leads to data abstraction or hiding

iv) Abstraction :- Abstraction means hiding displaying only essential information features and hiding the details. Data abstraction refers to providing only essential information about the data to the hiding the background details or implementation.

v) Inheritance :- Capability of a class to derive properties & characteristics from another class is called inheritance.

OR

Process by which object of one class acquired the properties of ~~one~~ objects of another class.

⇒ Supports the concept of reusability.

vi) Polymorphism ⇒ Means "Many forms". ~~Ability~~ Ability of a ~~me~~ to take more than one form.

- Operator Overloading - The process of making an operator to exhibit different behaviour in different instances is known as operator Overloading

Function Overloading ⇒ Using a single function name to perform different types of tasks.

vii) Message Passing ← Object communicate with one another by sending and receiving information to each other. A message for an object is a request for execution of a procedure & therefore will invoke

a function on the receiving objects that generates the desired results.

2) Constructors in C++

Special ~~data~~ type of member function that has same name as the ~~da~~ class name & helps to initialize the object of a class.

- Don't have return type
- If we don't specify a constructor, C++ generate a default constructor for object (expects no parameter and has empty body) implicitly.

Types of Constructor

- i) ~~Dea~~ Default Constructor (No arugment or ~~p~~parameters)
- ii) Parameterized Constructor (with arugment)
- iii) Copy Constructor (initialize object using another object of same class)

⇒ More than one constructor means Constructor Overloading

```
class point {
        double x, y
    public:
        point () { x = 0; y = 0; }
        point (double px, double py) { x = px,
                                       y = py; }
        point (const point &p1)
            { x = p1.x, y = p1.y; }
```

3) Destructor :- Member function that has same name as the class name preceded by a (~) operator, automatically called whenever object is going to destroy.

→ It helps to deallocate the memory of an object
→ It is always called on the reverse order of the constructor
→ Only single destructor with no parameter.

~point ()

4) Static Keyword

⇒ static variable in C

- A static variable remain in memory while the program is running (still if its scope ends)

- Static variable allocated in data segment, not in stack segment of memory

- Static variable (like global variable) are initialized as 0 if not initialized explicity

- In C, static variable can only be initialized using constant literals.

In C++

i) static variables in a Function :- It gets allocated for the lifetime of the program.

Java doesn't allow static local variables in functions

- static variable initialized only once

Date: / / Page no:

ii) Static variable in a class :-
- In a class static variable are shared by the object
- Also because of the reason that static variable can not be initialized using constructors

datatype Classname :: variablename = "value";

iii) Class object as static :- Object also when declared as static have a scope till the lifetime of program.

iv) Static functions in a class :-
static member function are allowed to access only the static data member or other static member function.

Non-static function can accessed static data member but static function not allowed to access non-static data member of class.

5) Access Modifiers
Access specifiers/Modifiers Access Modifiers used to implement data hiding.

Access specifiers are used to assign the accessibility to the class members. That is, it sets some restrictions on the class members not to directly accessed by the outside functions.
Type of Access Modifiers :-
1) Public :- members are accessible to anywhere everyone (outside the class also) in the program using (.) operator with the object of that class.

ii) Private :- Only the member function or the friend functions are allowed to access the private data member of a class.

iii) Protected :- Similar to Private but ~~access by~~ accessed by any subclass (derived class) of that class as well. & also by friend class.

6) Inheritance

Capability of a class to derive properties and characteristics from another class is called inheritance

⇒ Base Class :- The class whose properties are inherited by sub class is called Base class or ~~Derived~~ Super class

⇒ Derived class :- The class the inherits properties from another class is called Sub class or Derived Class

Syntax :- class sub_class_name : access_mode base_class_name
{
//body of subclass
};

⇒ Mode of inheritance

i) ~~Pool~~ i) Public mode :- public of base class becomes public & protected become protected of derived class

ii) Protected mode :- public & protected member of base class became protected in derived class

iii) Private mode :- public & protected of base class became private in derived class.

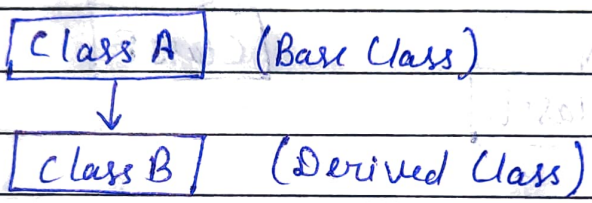
⇒ ~~Base class~~ derived class does not inherit :-
i) Base class's constructor & destructor, ii) base class's friend functions.
iii) Overloaded operator of the base class.

Date: __/__/____ Page no: _____

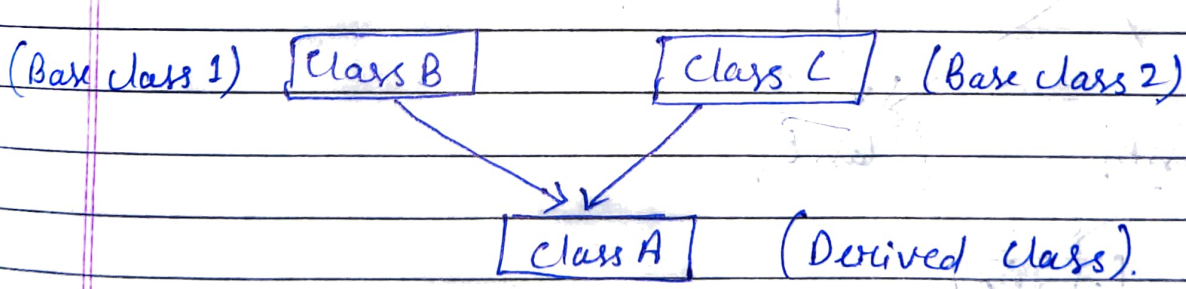
⇒ Note :- Private member can not be accessed by in the derived class. However, it does inherit a full parent object, which contains any private member which that class declares.

⇒ Types of Inheritance

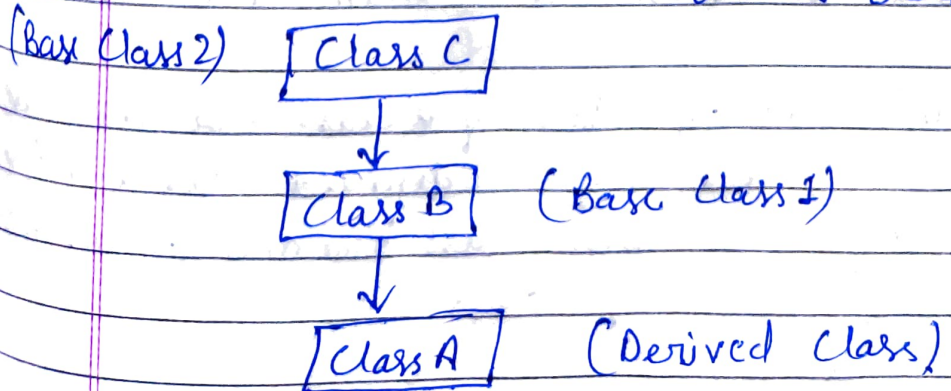
i) Single Inheritance :- One ~~to~~ sub class is inherited by one base class only



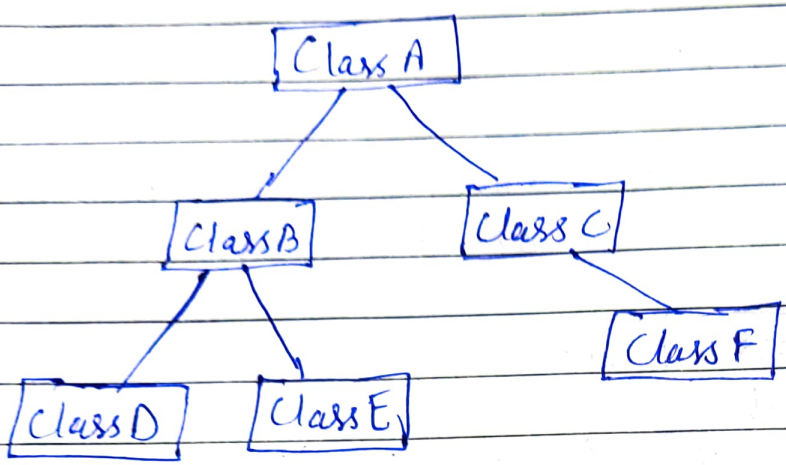
ii) Multiple Inheritance :- One sub class is inherited from more than one base classes.



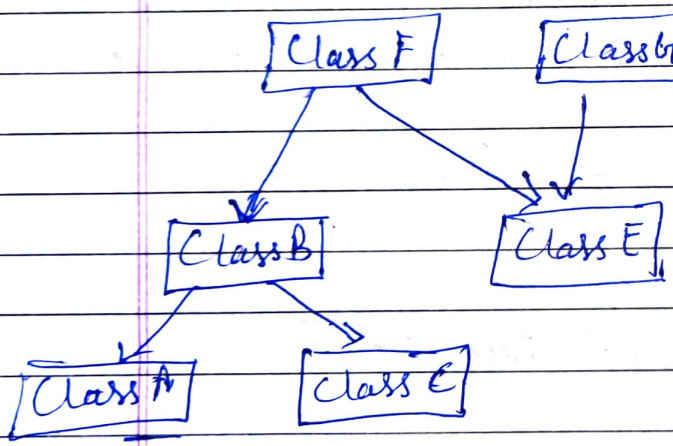
iii) Multilevel Inheritance :- A derived class is created from another derived class.



iv) Hierarchical Inheritance :- More than one sub class is inherited from a single base class.
OR// more than one derived class is created from a single base class.

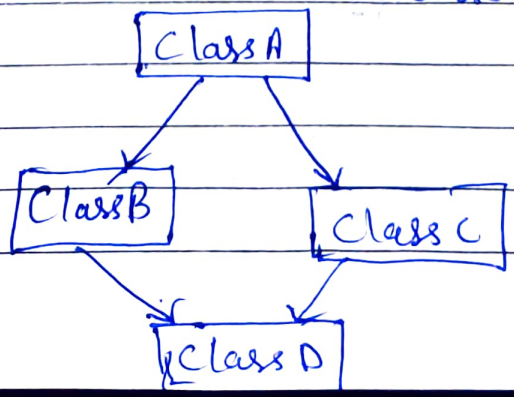


v) Hybrid (Virtual) Inheritance :- Combining more than one type of inheritance, ex- multiple + hierarchical



⇒ Ambiguity in Multiple Inheritance :- diamond Problems.

Diamond Problem :- Two superclass of a class have same base class. OR A derived class with two base class have one common base class.



Class D gets two copies of all attributes of Class A, this cause ambiguities.

⇒ Two ways to avoid this ambiguity.

1) Using scope resolution,

Note :- still there are two copies of class A in class D.

Ex → class D obj;

obj.B::a = 10; obj.C::a = 100;

2) Avoiding ambiguity using virtual base class.

Ex → class B : virtual public class A { } class C : virtual public class A { }

Here class D has only one copy of class A.

7) Polymorphism

└ Provision of single interface to entities of different types or use of single symbol to represent multiple different types.

Types — i) Compile time

├ Function Overloading

Operator Overloading.

ii) Runtime

├ Virtual Function or Overriding function

i) Compile time :- Whenever an object is bound with their functionality at the compile time, this is known as the compile time polymorphism.

a) Function Overloading ↳ Multiple function with same name but different parameter.

Function can be overloaded by change in no. of arugments or/and change in type of arugments.

Note ⇒ In C++ & Java, functions can not be overloaded if they differ only in the return type.

↓ It gives error.

Java does not support operator overloading.

Date: / / Page no:

⇒ C++ allows functions to be overloaded on the basis of const-ness of parameter only if the const parameter is a reference or a pointer.

⇒ b) Operator Function Overloading :-

└ The ability to provide the operators with a special meaning for a data type

Syntax :- return_type classname :: operator op (argument list)
{ //body }
[op can be +, -, x ----]

Rules :- i) The overloaded operator contains atleast one operand of the user defined data types.

ii) In case of normal member function, the binary operator have only argument & unary do not have an argument

iii) In case of friend function (global), the binary operator should have two argument & unary have one argument not

* iv) Operator that can be overloaded are . (dot), ::, ?:, .*, sizeof

v) Operator than can not be overloaded when defined as friend are Assignment (=), subscript ([ ]), funcal call ("()") & member selection (→)

vi) Some operator like (assignment) =, (address) &, (Comma), are by default overloaded.

Pointer ambiguity issue is solved by virtual keyword.

Date: / / Page no:

⇒ ii) Runtime Polymorphism :-

Whenever an object is bound with functionality at run time, this is known as runtime polymorphism, achieved by method overriding.

a) Function (Method) Overriding :- Redefinition of base class function in its derived class with same signature i.e. return type & parameter.

Override keyword :- It will make the compiles to check the base class to see if there is a virtual function with this exact signature.

Virtual function :- Virtual function is member function declared with a base class using the keyword virtual and is redefined (overriden) by a derived class.

When you refer to a derived class object using pointer or reference of the base class, you can call virtual function for that object and execute the derived class version of the function

Use to achieve runtime polymorphism.

Virtual function can not be static

⇒ Pure Virtual Function ⇒ For which we need not to write any function definition & only we have to declare it. It is declare by assigning 0 in the declaration.

Abstract Class ~~only~~ ~~3. Only defi~~

↳ We cannot create objects of abstract class.

~~1) Some Some Points~~
~~→ Only A abstract~~

Some Points

1) A class is abstract if it has at least one pure virtual function
2) We can have pointer & references of abstract class type.
3) If we do not override the pure virtual function in derived class, then derived class also becomes abstract class
4) A abstract class can have constructors.

Java has abstract keyword for abstract class & also ~~pure~~ function made pure virtual by abstract keywords.

~~7) Fri~~

8) Friend Keyword

1) Friend Class ⇒ Can access private & protected members of other class in which it is declared as Friend.
2) Friend Function ⇒ Can be given special grant to access private and protected members.

Points :- 1) Friendship is not mutal. ~~If class A~~
2) Should use only for limited purpose (reduce value of encapsulation)
3) Friendship is not inherited. If base class has a friend function, then the function doesn't become a friend of the derived class(es).
4) The concept of friend is not there in Java.

Date / / Page no:

9) Some Important Points.

i) Early Binding (Compile time) ⇒ Early binding means an object is bound to its function call at compile time.

OR// The overloaded member function are selected for invoking based on their type & no. of arug arguments at compile time

Also known as static binding or static lnking.

ii) Late Binding (Runtime) ⇒ In run time polymorphism the function call is linked with the appropriate object much later after compilation process is called late binding or dynamic binding.

iii) Virtual Destructors ⇒ It ensure that the derived class destructor is called when a base class pointer is used while deleting a dynamically created derived class object.

Constructors cannot be virtual.

iv) When a object of a derived class is created, the base class's constructor is ~~execut~~ executed first, followed by the derived class constructor. Also called in same order in which they are inherited.

v) When an object of a derived class is destroyed, its destructor is called first, then of the base class.

10) Exception Handling

```
try {
    throw ...
}
catch {
}  catch(...) {
}
```

```
try {
    throw ...
}
catch(argument)
{
} catch(...) { }
```

... is used as a default catch block.

⇒ If an exception is thrown and not caught anywhere, the program terminates abnormally.

⇒ A derived class exception should be caught before a base class exception.

⇒ Like Java, C++ library has a standard exception class which is base class for all standard exceptions.

⇒ Unlike Java, in C++, all exception are unchecked.

⇒ Try : catch blocks can be nested. & comes in pair

⇒ When an exception is thrown, all objects created inside the enclosing try block are destructed before the control is transferred to catch block

⇒ Template Points

1) We can pass more than one data types as arguments to templates
2) We can specify default value to template
3) When static member has template class/function, then each instance contains its own static variable
4) Template Specialization allows us to have different code for particular data type.

5) We can pass non-type argument to templates but it must be const. Mainly use for min/max or any constant value;

Date: / / Page no:

11) Generic Programming & Template

└ Data types are not specified at the time of implementation of code logic.

Advantages – 1) Code Reusability
2) Avoid Function Overloading
3) One written it can be used for multiple time & cases.

Template ⇒ Allow to pass data type as a parameter.

i) Function Template

Syntax :- template <typename T>
T myMax (T x, T y)
{ //body
}

Can use class also in place of typename

Call :- myMax<int>(3,7);

ii) Class Templates

Syntax :- template <typename T, typename U>
class MyClass {
T x;
U y;
//body
};

template <typename T>
void MyClass<T,U>::Print { cout<<x; }

template <typename T, typename U>
void Myclass<T,U>::getx() { cout<<x; }.