Programmers write instructions in various programming languages to perform their computation

tasks such as:

(i) Machine level Language : Set of instruction executed by CPU

(ii) Assembly level Language : Low level Language

(iii) High level Language : High-level language is any programming language that enables development of a program in much simpler programming context and is generally independent of the computer's hardware architecture.

PROGRAMING PARDIGMS : The programming paradigm is referred to the technique or approach of writing a program.

The high-level programming languages are broadly categorized in to two categories:

- Procedure oriented programming(POP) language.
- Object oriented programming(OOP) language.

PROCEDURE ORIENTED LANGUAGE : In the procedure oriented approach, the problem **is viewed as sequence of things** to be done such as reading , calculation and printing. Procedure oriented programming basically **consist of writing a list of instruction or actions for the computer to follow and organizing these instruction into groups known as functions**.

The **disadvantage o**f the procedure oriented programming languages is:

1. Global data access

2. It does not model real word problem very well

3. No data hiding

OBJECT ORIENTED PROGRAMMING : Object oriented programming as an approach that provides a way of modularizing programs by creating partitioned memory area for both data and functions that can be used as templates for creating copies of such modules on demand"

OOP's

OOP's : OOPs refers to Object-Oriented Programming. It is the **programming paradigm** that is defined using objects. Objects can be considered as real-world instances of entities like class, that have some characteristics and behaviors.

OOP stands for object-oriented programming. It is a programming paradigm that revolves around the object rather than function and procedure. In other words, it is an approach for developing applications that emphasize on objects. An object is a real word entity that contains data and code. It allows binding data and code together.

OOPs needs to be used for:

1. making programming clearer and problem-solving more concise
2. reusing code with the help of inheritance
3. reducing redundancy
4. encapsulation
5. data hiding
6. the division into subproblems
7. program flexibility using polymorphism

There are various OOP languages but the most widely used are:

- Python
- Java
- Go
- Dart
- C++
- C#
- Ruby

Advantages of OOP

- It follows a bottom-up approach.
- It models the real word well.
- It allows us the reusability of code.
- Avoids unnecessary data exposure to the user by using the abstraction.
- OOP forces the designers to have a long and extensive design phase that results in better design and fewer flaws.
- Decompose a complex problem into smaller chunks.
- Programmer are able to reach their goals faster.
- Minimizes the complexity.
- Easy redesign and extension of code that does not affect the other functionality.

Disadvantages of OOP

- Proper planning is required.
- Program design is tricky.
- Programmer should be well skilled.
- Classes tend to be overly generalized.

LIMITATIONS OF OOPS

- Solving problems takes more time as compared to Procedure Oriented Programming.
- The size of the programs created using this approach may become larger than the programs written using the procedure-oriented programming approach.
- Software developed using this approach requires a substantial amount of pre-work and planning.
- OOP code is difficult to understand if you do not have the corresponding class documentation.
- In certain scenarios, these programs can consume a large amount of memory.
- Not suitable for small problems.
- Takes more time to solve problems.

PURE OBJECT ORIENTED LANGUAGE : The programming language is called pure object-oriented language that treats everything inside the program as an object. The primitive types are not supported by the pure OOPs language

Some other features that must satisfy by a pure object-oriented language:

- Encapsulation
- Inheritance
- Polymorphism
- Abstraction
- All predefined types are objects
- All user-defined types are objects
- All operations performed on objects must be only through methods exposed to the objects.

Java is not a pure object-oriented programming language because pre-defined data types in Java are not treated as objects. Hence, it is not an object-oriented language.

CLASS : A **class** is a building block of Object Oriented Programs. It is a user-defined data type that contains the data members and member functions that operate on the data members.

A group of objects that share common properties for data part and some program part are collectively called as class.

What is the difference between a class and a structure?

Class: User-defined blueprint from which objects are created. It consists of methods or set of instructions that are to be performed on the objects.

Structure: A structure is basically a user-defined collection of variables which are of different data types. All Members are public

Class	Structure
Class is a group of common objects that shares common properties.	The structure is a collection of differe types.
It deals with data members and member functions.	It deals with data members only.
It supports inheritance.	It does not support inheritance.
Member variables cannot be initialized directly.	Member variables can be initialized direct
It is of type reference.	It is of a type value.
It's members are private by default.	It's members are public by default.
The keyword class defines a class.	The keyword struct defines a structure.
An instance of a class is an object.	An instance of a structure is a structure va
Useful while dealing with the complex data structure.	Useful while dealing with the sma structure.

OBJECT : (Basic run time entity)An **object** is an instance of a class. Data members and methods of a class cannot be used directly. We need to create an object (or instance) of the class to use them. In simple terms, they are the actual world entities that have a state and behavior.

Object	Class
A real-world entity which is an instance of a class	A class is basically a template or a blueprint within which objects can be created
An object acts like a variable of the class	Binds methods and data together into a single unit
An object is a physical entity	A class is a logical entity
Objects take memory space when they are created	A class does not take memory space when created
Objects can be declared as and when required	Classes are declared just once

A Class can exist without an Object but an Object need a class to exist

```
# include<iostream.h>

class person {

char name[30];

int age;

public:

void getdata(void);

void display(void);

};
```

```
void person :: getdata ( void ) {

cout<<"enter name";

cin>>name;

cout<<"enter age";

cin>>age;

}

void display() {

cout<<"\n name:"<<name;

cout<<"\n age:"<<age;

}

int main( )

{

person p;

p.getdata();

p.display();

return(0);

}
```

- **ENUMERATIONS :** An enumerated data type is another user defined type which provides a way for attaching names to number. The enum keyword automatically enumerates a list of words by assigning them values 0,1,2 and soon.

Example:

enum shape { circle,square,triangle}

enum colour{red,blue,green,yellow}

enum position {off,on}

By default, the enumerators are assigned integer values starting with 0 for the first

enumerator, 1 for the second and so on. We can also write

enum color {red, blue=4,green=8};

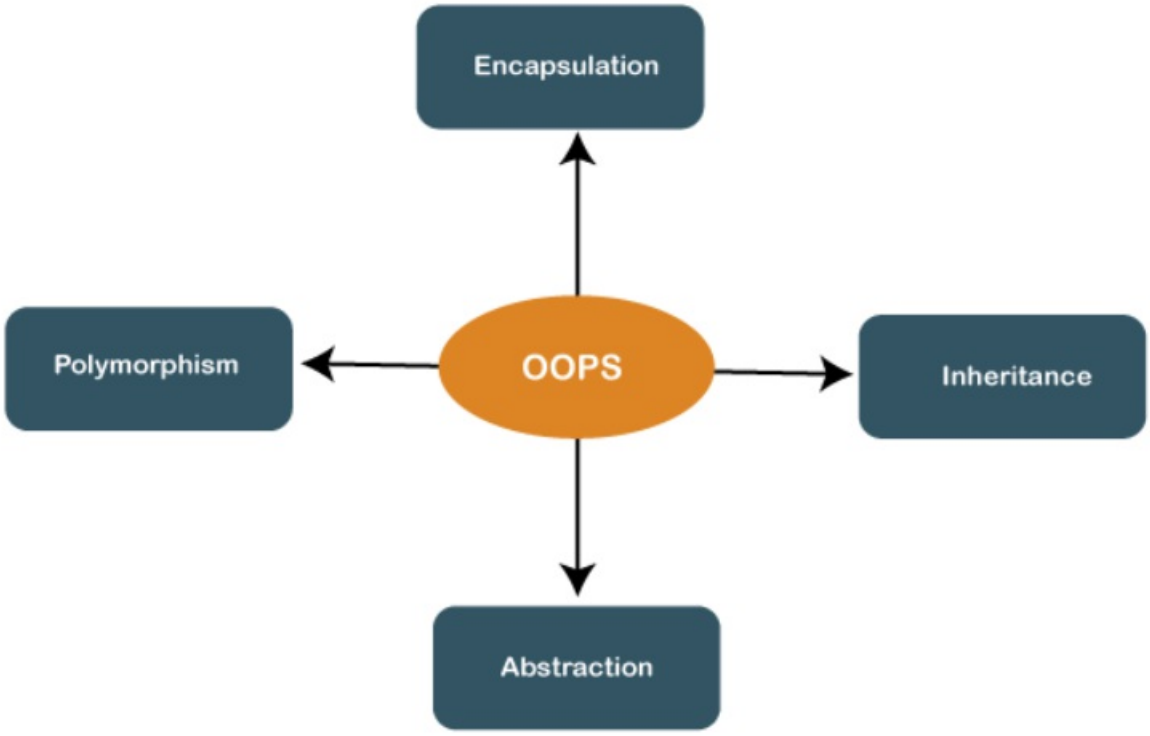
enum color {red=5,blue,green};

- **SWITCH STATEMENT :** This is a multiple-branching statement where, based on a condition, the control is transferred to one of the many possible points
- **INLINE FUNCTIONS :**

To eliminate the cost of calls to small functions C++ proposes a new feature called inline function. An inline function is a function that is expanded inline when it is invoked .That is **the compiler replaces the function call with the corresponding function code**.

```
inline double cube (double a)

{

return(a*a*a);

}

C=cube(3.0);
```

FEATURES OF OOPS



FUNCTION OVERLOADING : Overloading refers to the use of the same thing for different purposes . **We can use the same function name to creates functions that perform a variety of different tasks.** This is known as function polymorphism in oops.

CLASS : Class is a group of objects that share common properties and relationships.

Member can be defined in two places

• Outside the class definition

```
void item :: getdata (int a , float b )
{
number=a;
cost=b;
}
void item :: putdata ( void)
{
cout<<"number=:"<<number<<endl;
cout<<"cost="<<cost<<endl;
}
```

• Inside the class function

```
class item
{
Intnumber;
float cost;
public:
void getdata (int a ,float b);
void putdata(void)
{
cout<<number<<endl; cout<<cost<<endl;
```

}

};

STATIC DATA MEMBERS : A data member of a class can be qualified as static . The properties of a static member variable are similar to that of a static variable.

They Contains special characteristics.

Variable has contain special characteristics:-

- It is initialized to zero when the first object of its class is created. No other initialization is permitted.
- Only one copy of that member is created for the entire class and is shared by all the objects of that class, no matter how many objects are created.
- It is visible only with in the class but its life time is the entire program. Static variables are normally used to maintain values common to the entire class.

STATIC MEMBER FUNCTION : A member function that is declared static has following properties :-

- A static function can have access to only other static members declared in the same class.
- A static member function can be called using the class name as follows:-

class - name :: function - name;

class item

{

static int count; //count is static

int number;

public:

void getdata(int a)

. {

number=a;

count++;

}

void getcount(void)

{

cout<<"count:";

cout<<count<<endl;

}

static void showcount(void)

{

cout<<"count="<<count<<endl; }

};

};

// CALLING OF STATIC FUNCTION

Item :: showcount();

// TO INITIALISE STATIC DATA MEMBER

Int item :: count=7;

OBJECTS AS FUNCTION ARGUMENTS

Like any other data type, an object may be used as A function argument. This can cone in two ways

1. A copy of the entire object is passed to the function. (Call by Vaue)

2. Only the address of the object is transferred to the function (Call by Reference)

FRIEND FUNCTIONS : A friend function is a friend of a class that is allowed to access to Public, private, or protected data in that same class. If the function is defined outside the class cannot access such information.

A friend can be declared anywhere in the class declaration, and it cannot be affected by access control keywords like private, public, or protected.

WHEN two classes manager and scientist, have been defined we should like to use a function income tax to operate on the objects of both these classes.

In such situations, c++ allows the common function lo be made friendly with both the classes

A friend function processes certain special characteristics:

-
    - **It is not in the scope of the class to which it has been declared as friend.**
    - **Since it is not in the scope of the class, it cannot be called using the object of that class. It can be invoked like a member function without the help of any object.**
    - **Unlike member functions.**

class ABC

{ -------- --------

public:

----

friend void xyz(ABC x);

};

A function can be a friend of more than one class to access both class data private.

CONSTRUCTOR : A constructor is a method used to initialize the state of an object, and it gets invoked at the time of object creation. Rules for constructor are:

- Constructor Name should be the same as a class name.
- A constructor must have no return type.

A constructor that accept no parameter is called the default constructor. If no such constructor is

defined, then the compiler supplies a default constructor .

Therefore a statement such as :-

A a ;//invokes the default constructor of the compiler to create the object "a" ;

The constructor functions have some characteristics:-

- They should be declared in the public section .
- They are invoked automatically when the objects are created.
- They don't have return types, not even void and therefore they cannot return values.
- They cannot be inherited , though a derived class can call the base class constructor

```
class abc

{

private:

char nm[];

public:

abc ( )

{ cout<<"enter your name:";

cin>>nm; }

void display( )

{ cout<<nm; }

};

int main( )
```

```
{
clrscr( );
abc d;
d.display( );
getch( );
return(0);
}
```

The constructors that can take arguments are **called parameterized constructors**. Using parameterized constructor we can initialize the various data elements of different objects with different values when they are created.

```
integer int 1 = integer(0,100); // explicit call
integer int 1(0,100); //implicite call
```

We can overload a constructor too.

COPY CONSTRUCTOR : A copy constructor is used to declare and initialize an object from another object.

```
class code
{
int id;
public
code ( ) { } //constructor
code (int a) { id=a; } //constructor
code(code &x)
{
Id=x.id;
}
void display( )
{
cout<<id;
}
};
int main( )
{
code A(100);
code B(A); // Way 1 of Coppying constructor
code C=A); // Way 2 of Coppying constructor
code D;
D=A;
cout<<" \n id of A :"; A.display( );
cout<<" \nid of B :"; B.display( );
cout<<" \n id of C:"; C.display( );
cout<<" \n id of D:"; D.display( );
```

}

CALL SEQUENCE : DERIVED → BASE

EXECUTION SEQUENCE : BASE → DERIVED

DESTRUCTOR : A destructor, us the name implies is used to destroy the objects that have been created by a constructor. Like a constructor, the destructor is a member function whose name is the same as the class name but is **preceded by a tilde**.

A **destructor never takes any argument nor does it return any value.** It will be invoked implicitly by the compiler upon exit from the program to clean up storage that is no longer accessible.

CALL SEQUENCE : DERIVED → BASE

EXECUTION SEQUENCE : DERIVED → BASE

OPERATOR OVERLOADING : To define an additional task to an operator, we must specify what it means in relation to the class to which the operator is applied . This is done with the help of a special function called o**perator** function, which describes the task.

Syntax:-

return-type class-name :: operator op(arg-list) // op is operator that is overloaded

{

function body

}

The process of overloading involves the following steps:-

1. Create a class that defines the data type that is used in the overloading operation.

2. Declare the operator function operator op() in the public part of the class

3. It may be either a member function or friend function.

4. Define the operator function to implement the required operations.

```
class complex
{
float real,img;
public:
complex()
{
real=0;
img=0;
}
complex(float r,float i)
{
real=r;
img=i;
}
void show()
{
cout<<real<<"+"<<img;
}
complex operator+(complex &p)
{
complex w;
w.real=real+q.real;
w.img=img+q.img;
return w;
}
};
void main()
{
complex s(3,4);
complex t(4,5);
complex m;
m=s+t;
s.show();
t.show();
m.show();
}
```

```
class complex
{
float real,img;
public:
complex()
{
real=0;
img=0;
}
complex(float r,float i)
{
real=r;
img=i;
}
void show()
{
cout<<real<<"+"<<img;
}
friend complex operator+(complex &p,complex &q);

};

complex operator+(complex &p,complex &q)
{
complex w;
w.real=p.real+q.real;
w.img=p.img+q.img;
return w;
}
void main()
{
complex s(3,4);
complex t(4,5);
complex m;
m=operator+(s,t);
s.show();
t.show();
m.show();
}
```

VIRTUAL BASE CLASSES : Let us say the 'child' has two direct base classes 'parent1' and 'parent2' which themselves has a common base class 'grandparent'. The child inherits the traits of 'grandparent' via two separate paths.

The grandparent is sometimes referred to as 'INDIRECT BASE CLASS'. Now, the inheritance by the child might cause some DUPLICACY problems**. The duplication of the inherited members can be avoided by making common base class as the virtual base class:**

```
class parent1: virtual public g_parent
{
        // Body
};

class parent2: public virtual g_parent
{
        // Body
};
class child : public parent1, public parent2
{
        // body
};
```

When a class is virtual base class, C++ takes necessary care to see that only one copy of that class is inherited, regardless of how many inheritance paths exists between virtual base class and derived class.

VIRTUAL FUNCTIONS : A virtual function (also known as virtual methods) is a member function that is declared within a base class and is re-defined (overridden) by a derived class.

class point {

```
intx;

inty;

public:

virtual int length ( );

virtual void display ( );

};
```

A virtual function cannot be a static member since a virtual member is always a member of a particular object in a class rather than a member of the class as a whole.

Rules for Virtual Functions

The rules for the virtual functions in C++ are as follows:

1. Virtual functions cannot be static.
2. A virtual function can be a friend function of another class.
3. Virtual functions should be accessed using a pointer or reference of base class type to achieve runtime polymorphism.
4. The prototype of virtual functions should be the same in the base as well as the derived class.
5. They are always defined in the base class and overridden in a derived class. It is not mandatory for the derived class to override (or re-define the virtual function), in that case, the base class version of the function is used.
6. A class may have a virtual destructor but it cannot have a virtual constructor.

```
class base {

public:

   virtual void print() { cout << "print base class\n"; }


   void show() { cout << "show base class\n"; }

};


class derived : public base {

public:

   void print() { cout << "print derived class\n"; }


   void show() { cout << "show derived class\n"; }

};


int main()

{

   base* bptr;

   derived d;

   bptr = &d;


   // Virtual function, binded at runtime

   bptr->print();


   // Non-virtual function, binded at compile time

   bptr->show();
```

```
    return 0;
}
```

print derived class

show base class

Runtime polymorphism is achieved only through a pointer (or reference) of the base class type. Also, a base class pointer can point to the objects of the base class as well as to the objects of the derived class. In the above code, the base class pointer 'bptr' contains the address of object 'd' of the derived class.

Late binding (Runtime) is done in accordance with the content of the pointer (i.e. location pointed to by pointer) and Early binding (Compile-time) is done according to the type of pointer since the print() function is declared with the virtual keyword so it will be bound at runtime (output is *print derived class* as the pointer is pointing to object of derived class) and show() is non-virtual so it will be bound during compile time (output is *show base class* as the pointer is of base type).

C++ Function Overriding

If base class and derived class have member functions with same name and arguments. If you create an object of derived class and write code to access that member function then, the member function in derived class is only invoked,

ABSTRACT CLASS : Abstract Class is a class which contains atleast one **Pure Virtual function** in it. Abstract classes are used to **provide an Interface for its sub classes.** Classes inheriting an Abstract Class must provide definition to the pure virtual function, otherwise they will also become abstract class.

Pure virtual Functions are virtual functions with no definition. They start with virtual keyword and ends with = 0. Here is the syntax for a pure virtual function,

virtual void f() = 0;

```
class Base
//Abstract base class
{
//Pure Virtual Function
public:
virtual void show() = 0;
};
class Derived:public Base
{
public:
void show()
{ cout << "Implementation of Virtual Function in Derived class"; }
};
int main()
{
Base obj;
Base *b;
Derived d;
b = &d;
b->show();
}
//Compile Time Error
```

Output : Implementation of Virtual Function in Derived class

What is encapsulation?

Wrapping up of data to form a single entity. Encapsulation is the process of binding data members and methods of a program together to do a specific job, without revealing unnecessary details.

One can visualize Encapsulation as the method of putting everything that is required to do the job, inside a capsule and presenting that capsule to the user. What it means is that by Encapsulation, all the necessary data and methods are bind together and all the unnecessary details are hidden to the normal user. So Encapsulation is the process of binding data members and methods of a program together to do a specific job, without revealing unnecessary details.

Encapsulation can also be defined in two different ways:

1) **Data hiding:** Encapsulation is the process of hiding unwanted information, such as restricting access to any member of an object.

2) **Data binding:** Encapsulation is the process of binding the data members and the methods together as a whole, as a class.

6. How does C++ support Polymorphism?

C++ is an Object-oriented programming language and it supports Polymorphism as well:

- Compile Time Polymorphism: C++ supports compile-time polymorphism with the help of features like templates, function overloading, and default arguments.
- Runtime Polymorphism: C++ supports Runtime polymorphism with the help of features like virtual functions. Virtual functions take the shape of the functions based on the type of object in reference and are resolved at runtime.

Data abstraction

28. What is data abstraction?

Data abstraction is a very important feature of OOPs that allows displaying only the important information and hiding the implementation details. For example, while riding a bike, you know that if you raise the accelerator, the speed will increase, but you don't know how it actually happens. This is data abstraction as the implementation details are hidden from the rider.

29. How to achieve data abstraction?

Data abstraction can be achieved through:

- Abstract class
- Abstract method

30. What is an abstract class?

An abstract class is a class that consists of abstract methods. These methods are basically declared but not defined. If these methods are to be used in some subclass, they need to be exclusively defined in the subclass.

31. Can you create an instance of an abstract class?

No. Instances of an abstract class cannot be created because it does not have a complete implementation. However, instances of subclass inheriting the abstract class can be created.

32. What is an interface?

It is a concept of OOPs that allows you to declare methods without defining them. Interfaces, unlike classes, are not blueprints because they do not contain detailed instructions or actions to be performed. Any class that implements an interface defines the methods of the interface.

33. Differentiate between data abstraction and encapsulation.

Data abstraction	Encapsulation
Solves the problem at the design level	Solves the problem at the implementation level
Allows showing important aspects while hiding implementation details	Binds code and data together into a single unit and hides it from the world

21 What is interface
An interface refers to a special type of class, which contains methods, but not their definition. Only the declaration of methods is allowed inside an interface. To use an interface, you cannot create objects. Instead, you need to implement that interface and define the methods for their implementation.

An interface in C++ is a class-like structure that contains only pure virtual functions and no member variables. Interfaces only contain pure virtual functions and no member variables.

Interfaces define a contract of behaviors that derived classes must implement.

Classes can inherit from multiple interfaces.

Interfaces cannot be instantiated directly; they are used to define common behaviors for classes.

```
class Drawable {
public:
// Pure virtual functions
virtual void draw() = 0;
virtual void setPosition(int x, int y) = 0;
};
```

26. What is an abstract class?

An abstract class is a special class containing abstract methods. The significance of abstract class is that the abstract methods inside it are not implemented and only declared. So as a result, when a subclass inherits the abstract class and needs to use its abstract methods, they need to define and implement them.

An abstract class in C++ is a class that cannot be instantiated on its own. It serves as a blueprint for other classes and can contain both concrete (implemented) and abstract (unimplemented) member functions. An abstract class cannot be instantiated directly; it must be subclassed (derived) to create objects.

```
class Animal {
public:
// Pure virtual function
virtual void makeSound() = 0;

// Normal member function
void sleep() {
cout << "Zzz..." << endl;
}
};
```

27. How is an abstract class different from an interface?

Interface and abstract classes both are special types of classes that contain only the methods declaration and not their implementation. But the interface is entirely different from an abstract class. **The main difference between the two is that when an interface is implemented, the subclass must define all its methods and provide its implementation.** Whereas in object-oriented programming, when a subclass inherits from an abstract class with abstract methods, the subclass is generally required to provide concrete implementations for all of those abstract methods in the abstract class unless the subclass itself is declared as abstract.

Also, an abstract class can contain abstract methods as well as non-abstract methods.

22. What is meant by static polymorphism?

Static Polymorphism is commonly known as the Compile time polymorphism. Static polymorphism is the feature by which an object is linked with the respective function or operator based on the values during the compile time. Static or Compile time Polymorphism can be achieved through Method overloading or operator overloading.

23. What is meant by dynamic polymorphism?

Dynamic Polymorphism or Runtime polymorphism refers to the type of Polymorphism in OOPs, by which the actual implementation of the function is decided during the runtime or execution. The dynamic or runtime polymorphism can be achieved with the help of method overriding.

24. What is the difference between overloading and overriding?

Overloading is a compile-time polymorphism feature in which an entity has multiple implementations with the same name. For example, Method overloading and Operator overloading.

Whereas Overriding is a runtime polymorphism feature in which an entity has the same name, but its implementation changes during execution. For example, Method overriding.
Image

25. How is data abstraction accomplished?

Data abstraction is accomplished with the help of abstract methods or abstract classes.

28. What are access specifiers and what is their significance?

Access specifiers, as the name suggests, are a special type of keywords, which are used to control or specify the accessibility of entities like classes, methods, etc. Some of the access specifiers or access modifiers include “private”, “public”, etc. These access specifiers also play a very vital role in achieving Encapsulation - one of the major features of OOPs.

29. What is an exception?

An exception can be considered as a special event, which is raised during the execution of a program at runtime, that brings the execution to a halt. The reason for the exception is mainly due to a position in the program, where the user wants to do something for which the program is not specified, like undesirable input.

30. What is meant by exception handling?

No one wants its software to fail or crash. Exceptions are the major reason for software failure. The exceptions can be handled in the program beforehand and prevent the execution from stopping. This is known as exception handling.
So exception handling is the mechanism for identifying the undesirable states that the program can reach and specifying the desirable outcomes of such states. Try-catch is the most common method used for handling exceptions in the program.

31. What is meant by Garbage Collection in OOPs world?

Object-oriented programming revolves around entities like objects. Each object consumes memory and there can be multiple objects of a class. So if these objects and their memories are not handled properly, then it might lead to certain memory-related errors and the system might fail.

Garbage collection refers to this mechanism of handling the memory in the program. Through garbage collection, the unwanted memory is freed up by removing the objects that are no longer needed.

45. What is an exception?

An exception is a kind of notification that interrupts the normal execution of a program. Exceptions provide a pattern to the error and transfer the error to the exception handler to resolve it. The state of the program is saved as soon as an exception is raised.

46. What is exception handling?

Exception handling in Object-Oriented Programming is a very important concept that is used to manage errors. An exception handler allows errors to be thrown and caught and implements a centralized mechanism to resolve them.

. What is the difference between an error and an exception?

Error	Exception
Errors are problems that should not be encountered by applications	Conditions that an application might try to catch

48. What is a try/ catch block?

A try/ catch block is used to handle exceptions. The try block defines a set of statements that may lead to an error. The catch block basically catches the exception.

49. What is a finally block?

A finally block consists of code that is used to execute important code such as closing a connection, etc. This block executes when the try block exits. It also makes sure that finally block executes even in case some unexpected exception is encountered.

is it possible for a class to inherit the constructor of its base class?

No, a class cannot inherit the constructor of its base class.

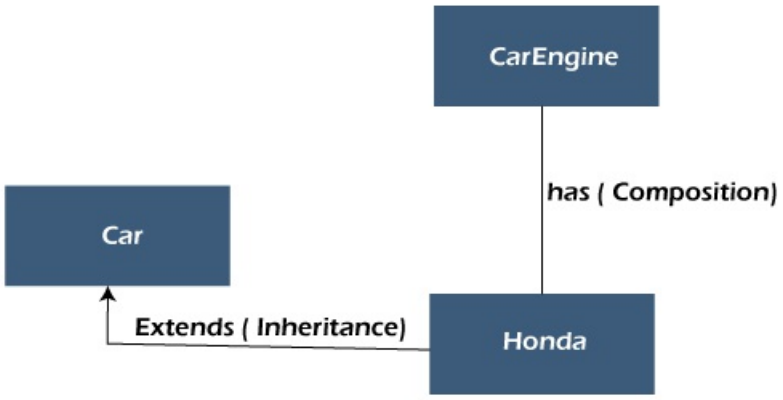
21) Identify which OOPs concept should be used in the following scenario?

A group of 5 friends, one boy never gives any contribution when the group goes for the outing. Suddenly a beautiful girl joins the same group. The boy who never contributes is now spending a lot of money for the group.

Runtime Polymorphism

22) What is composition?

Composition is one of the vital concepts in OOP. It describes a class that references one or more objects of other classes in instance variables. It allows us to model a has-a association between objects. We can find such relationships in the real world. For example, a car has an engine. the following figure depicts the same



The main benefits of composition are:

- Reuse existing code
- Design clean APIs
- Change the implementation of a class used in a composition without adapting any external clients.

23) What are the differences between copy constructor and assignment operator?

The copy constructor and the assignment operator (=) both are used to initialize one object using another object. The main difference between the two is that the copy constructor allocates separate memory to both objects i.e. existing object and newly created object while the assignment operator does not allocate new memory for the newly created object. It uses the reference variable that points to the previous memory block (where an old object is located).

Syntax of Copy Constructor

```
1. class_name (const class_name &obj)
2. {
3. //body
4. }
```

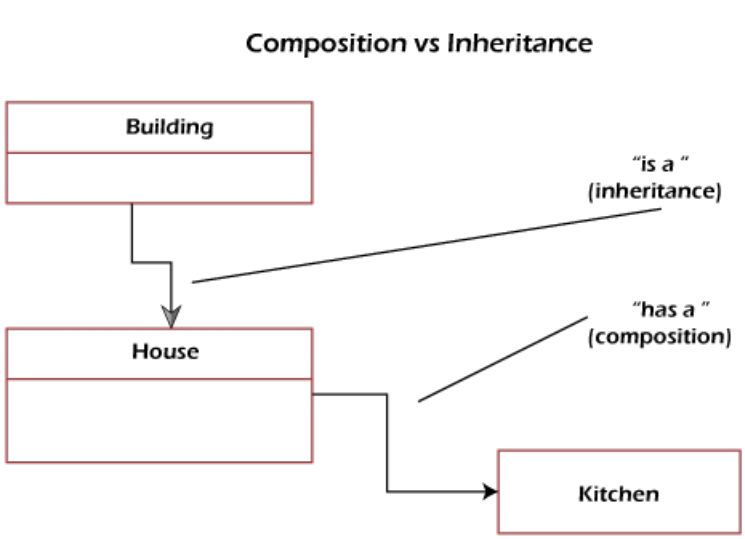
Syntax of Assignment Operator

```
1. class_name obj1, obj2;
2. obj1=obj2;
```

Copy Constructor	**Assignment Operator**
It is an overloaded constructor.	It is an operator.
It creates a new object as a copy of an existing object.	It assigns the value of one object to another object both of which already exist.
The copy constructor is used when a new object is created with some existing object.	It is used when we want to assign an existing object to a new object.
Both the objects use separate memory locations.	Both objects share the same memory but use the two different reference variables that point to the same location.
If no copy constructor is defined in the class, the compiler provides one.	If the assignment operator is not overloaded then the bitwise copy will be made.

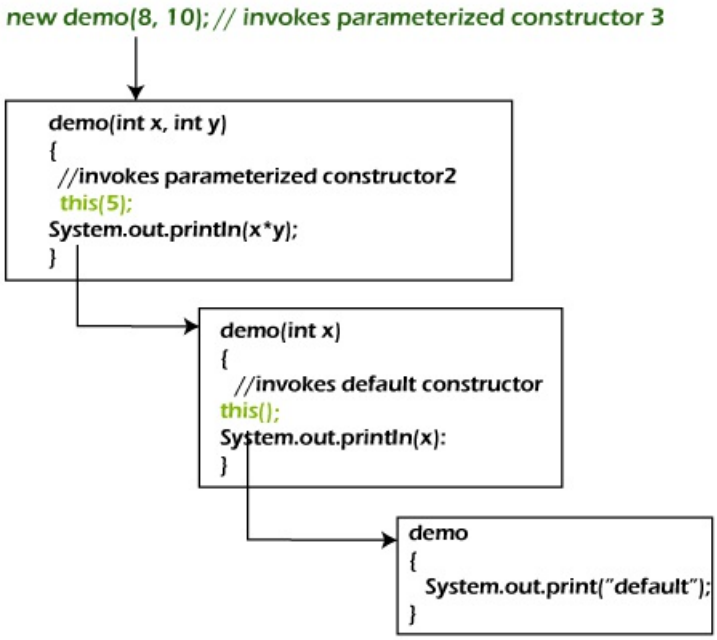
24) What is the difference between Composition and Inheritance?

Inheritance means an object inheriting reusable properties of the base class. Compositions mean that an object holds other objects. In Inheritance, there is only one object in memory (derived object) whereas, in Composition, the parent object holds references of all composed objects. From a design perspective, inheritance is "is a" relationship among objects whereas Composition is "has a" relationship among objects.



25) What is constructor chaining?

In OOPs, constructor chaining is a sequence of invoking constructors (of the same class) upon initializing an object. It is used when we want to invoke a number of constructors, one after another by using only an instance. In other words, if a class has more than one constructor (overloaded) and one of them tries to invoke another constructor, this process is known as constructor chaining. In C++, it is known as constructor delegation and it is present from C++ 11.



26) What are the limitations of inheritance?

- The main disadvantage of using inheritance is two classes get tightly coupled. That means one cannot be used independently of the other. If a method or aggregate is deleted in the Super Class, we have to refactor using that method in SubClass.
- Inherited functions work slower compared to normal functions.
- Need careful implementation otherwise leads to improper solutions.