**1. What is meant by the term OOPs?**

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.

**2. What is the need for OOPs?**

There are many reasons why OOPs is mostly preferred, but the most important among them are:

* OOPs helps users to understand the software easily, although they don’t know the actual implementation.
* With OOPs, the readability, understandability, and maintainability of the code increase multifold.
* Even very big software can be easily written and managed easily using OOPs.

**3. What are some major Object Oriented Programming languages?**

The programming languages that use and follow the Object-Oriented Programming paradigm or OOPs, are known as Object-Oriented Programming languages. Some of the major Object-Oriented Programming languages include:

* [**Java**](https://www.interviewbit.com/java-interview-questions/)
* [**C++**](https://www.interviewbit.com/cpp-interview-questions/)
* [**Javascript**](https://www.interviewbit.com/javascript-interview-questions/)
* [**Python**](https://www.interviewbit.com/python-interview-questions/)
* [**PHP**](https://www.interviewbit.com/php-interview-questions/)

And many more.

**You can download a PDF version of Oops Interview Questions.**

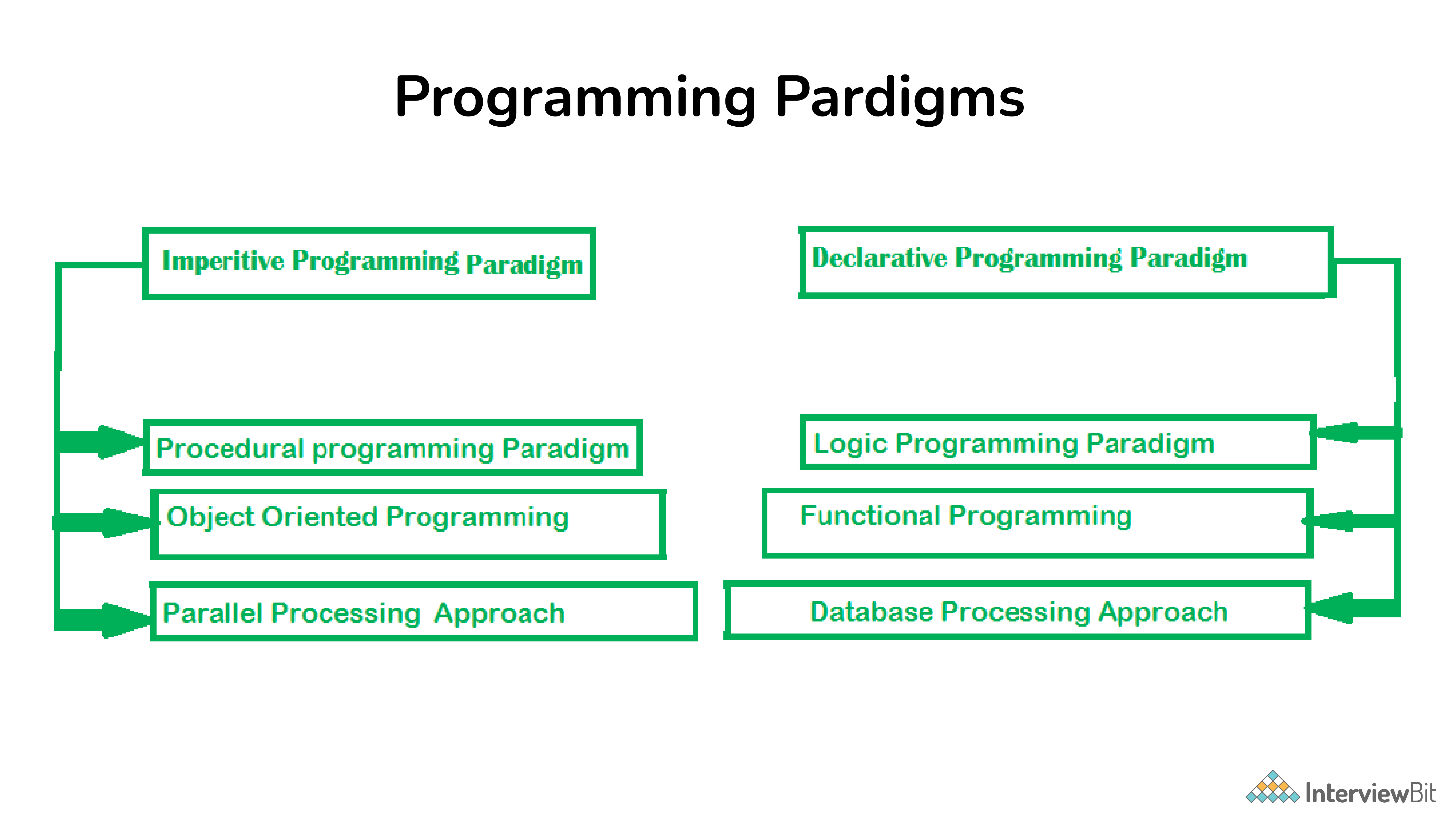
**Download PDF**

**4. What are some other programming paradigms other than OOPs?**

Programming paradigms refers to the method of classification of programming languages based on their features. There are mainly two types of Programming Paradigms:

* Imperative Programming Paradigm
* Declarative Programming Paradigm

Now, these paradigms can be further classified based:  
  
**1. Imperative Programming Paradigm**: Imperative programming focuses on **HOW to execute program logic and defines control flow** as statements that change a program state. This can be further classified as:  
a) Procedural Programming Paradigm: Procedural programming specifies the steps a program must take to reach the desired state, usually read in order from top to bottom.  
b) Object-Oriented Programming or OOP: Object-oriented programming (OOP) organizes programs as objects, that contain some data and have some behavior.  
c) Parallel Programming: Parallel programming paradigm breaks a task into subtasks and focuses on executing them simultaneously at the same time.  
  
**2. Declarative Programming Paradigm**: Declarative programming focuses on **WHAT to execute and defines program logic, but not a detailed control flow.** Declarative paradigm can be further classified into:  
a) Logical Programming Paradigm: Logical programming paradigm is based on formal logic, which refers to a set of sentences expressing facts and rules about how to solve a problem  
b) Functional Programming Paradigm: Functional programming is a programming paradigm where programs are constructed by applying and composing functions.  
c) Database Programming Paradigm: Database programming model is used to manage data and information structured as fields, records, and files.



**5. What is meant by Structured Programming?**

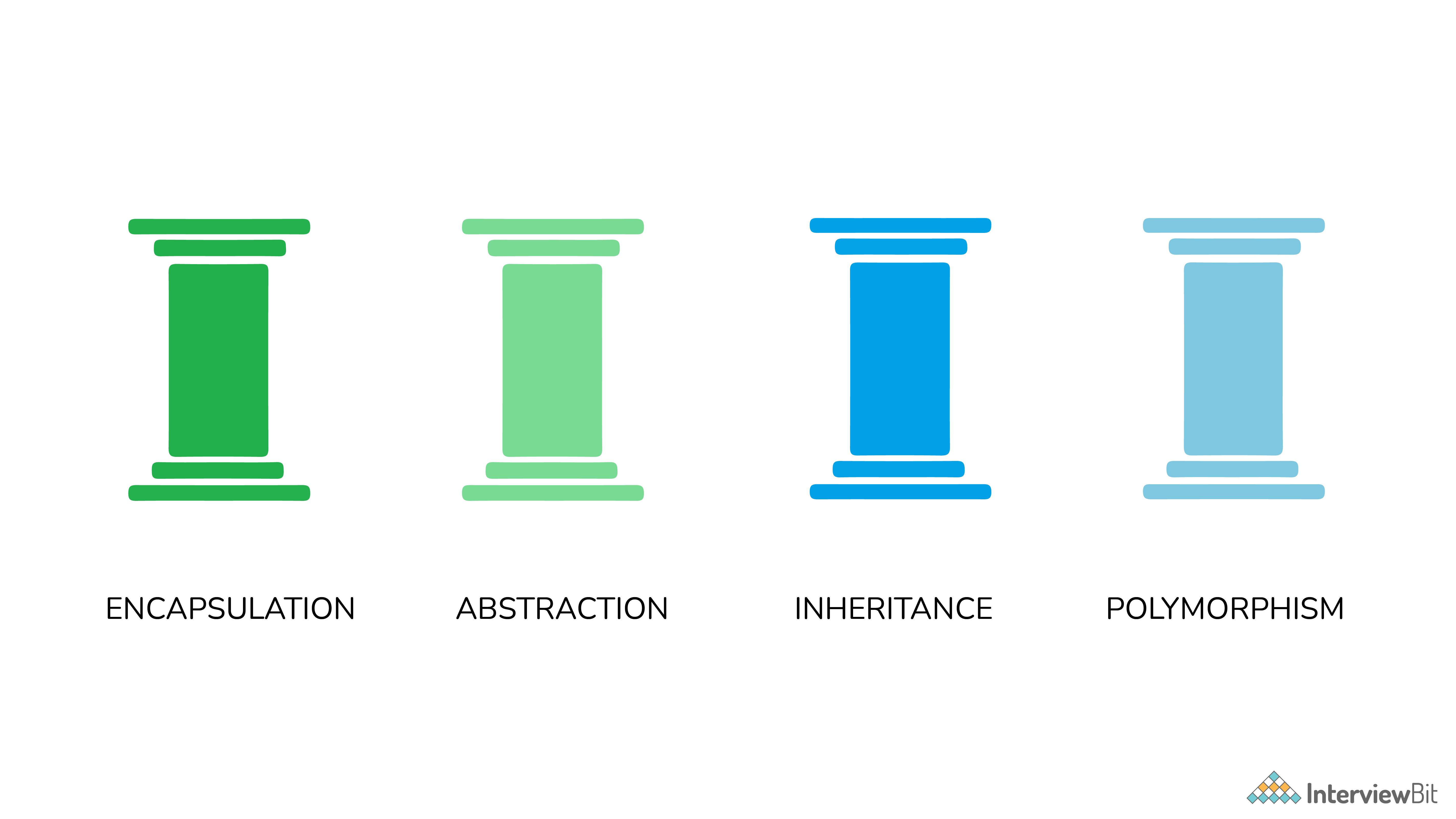
**Structured Programming** refers to the method of programming which consists of **a completely structured control flow.** Here structure refers to a block, which contains a set of rules, and has a **definitive control flow, such as (if/then/else), (while and for), block structures, and subroutines.**

Nearly all programming paradigms include Structured programming, including the OOPs model.

**6. What are the main features of OOPs?**

OOPs or Object Oriented Programming mainly comprises of the below four features, and make sure you don't miss any of these:

* Inheritance: Class Person is inherited by Student Class
* Encapsulation:
* Polymorphism
* Data Abstraction



**7. What are some advantages of using OOPs?**

* OOPs is very helpful in solving very complex level of problems.
* Highly complex programs can be created, handled, and maintained easily using object-oriented programming.
* OOPs, promote code reuse, thereby reducing redundancy.
* OOPs also helps to hide the unnecessary details with the help of Data Abstraction.
* OOPs, are based on a bottom-up approach, unlike the Structural programming paradigm, which uses a top-down approach.
* Polymorphism offers a lot of flexibility in OOPs.

**8. Why is OOPs so popular?**

OOPs programming paradigm is considered as a better style of programming. Not only it helps in writing a complex piece of code easily, but it also allows users to handle and maintain them easily as well. Not only that, the main pillar of OOPs - Data Abstraction, Encapsulation, Inheritance, and Polymorphism, makes it easy for programmers to solve complex scenarios. As a result of these, OOPs is so popular.

**Advanced OOPs Interview Questions**

**9. What is a class?**

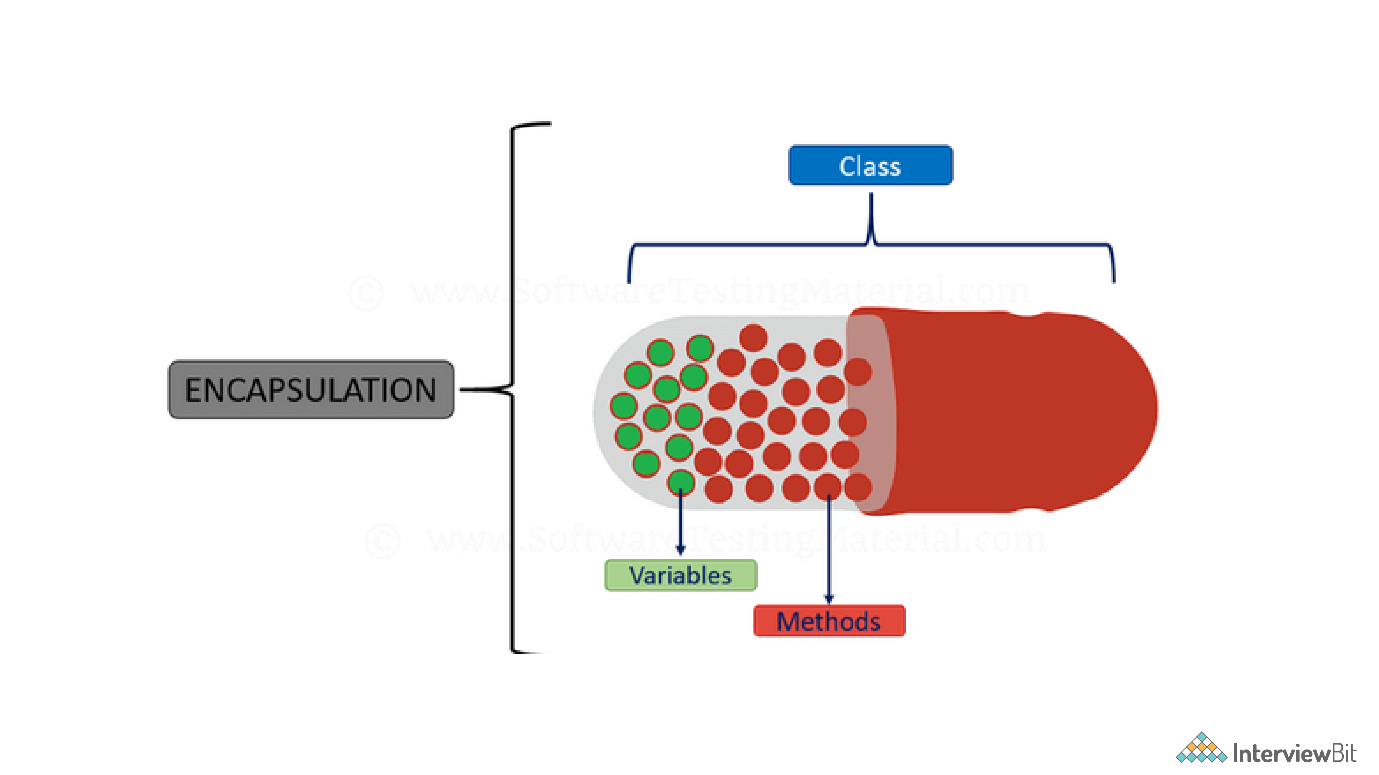
A class can be understood as a template or a blueprint, which contains some values, known as member data or member, and some set of rules, known as behaviors or functions. So when an object is created, it automatically takes the data and functions that are defined in the class.  
Therefore the class is basically **a template or blueprint for objects.** Also one can create as many objects as they want based on a class.

For example, first, a car’s template is created. Then multiple units of car are created based on that template.

**10. What is an object?**

An object refers to the instance of the class, which contains the **instance of the members and behaviors defined in the class template.** **In the real world, an object is an actual entity to which a user interacts, whereas class is just the blueprint for that object.** So the objects consume space and have some characteristic behavior.  
For example, a specific car.

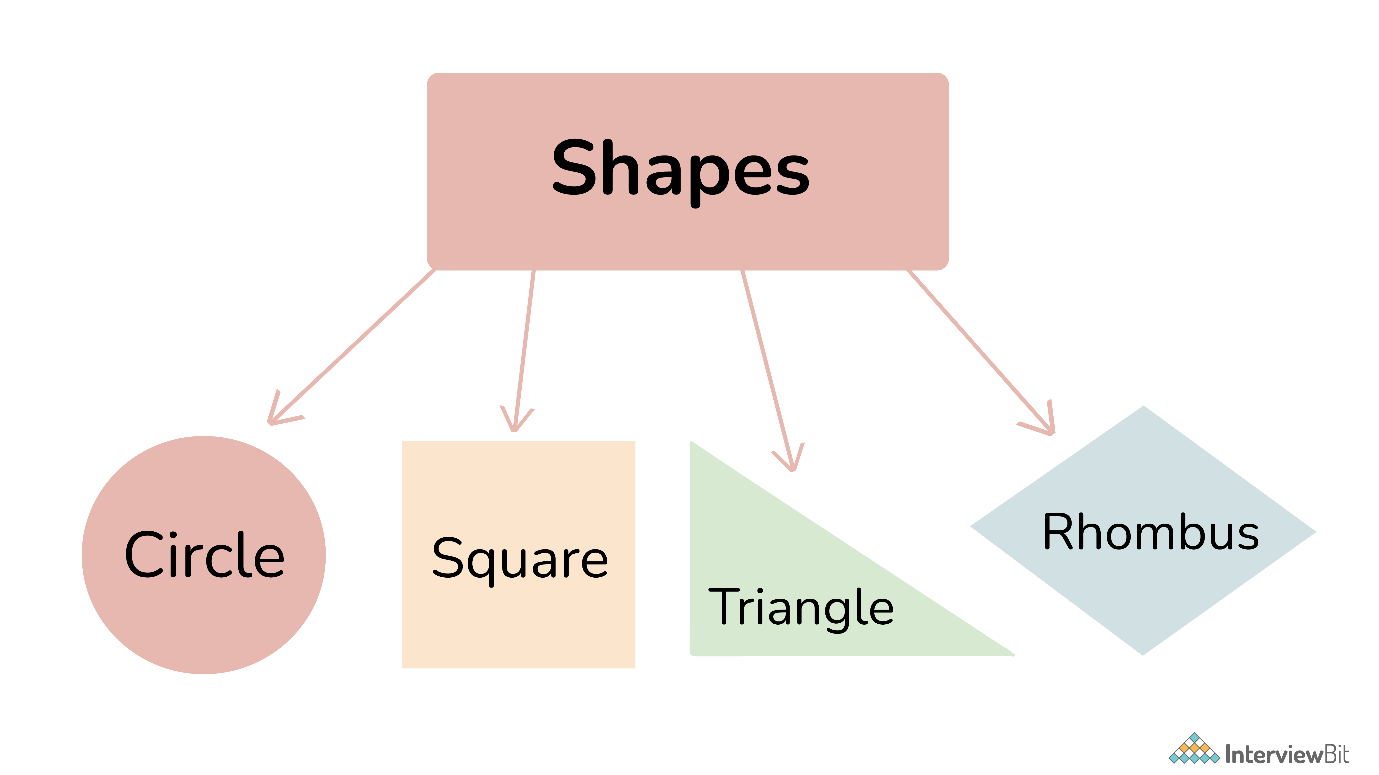
**11. What is encapsulation?**



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.

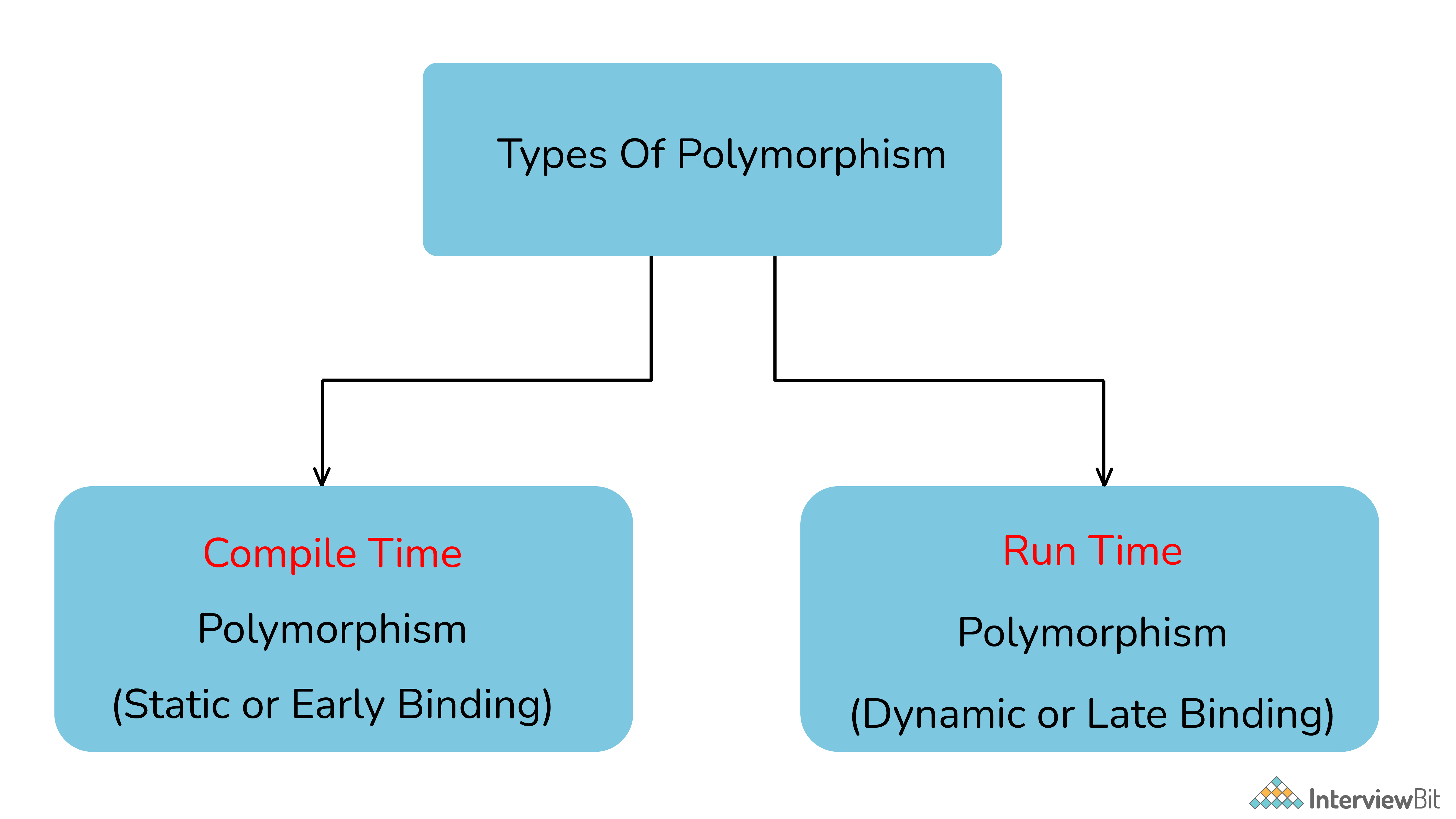
**12. What is Polymorphism?**

Polymorphism is composed of two words - “poly” which means “many”, and “morph” which means “shapes”. Therefore Polymorphism refers to something that has many shapes.



In OOPs, Polymorphism refers to the process by which some code, data, method, or object behaves differently under different circumstances or contexts. Compile-time polymorphism and Run time polymorphism are the two types of polymorphisms in OOPs languages.

**13. What is Compile time Polymorphism and how is it different from Runtime Polymorphism?**



Compile Time Polymorphism: Compile time polymorphism, also known as Static Polymorphism, refers to the type of Polymorphism that happens at compile time. What it means is that the compiler decides what shape or value has to be taken by the entity in the picture.

Example:

// In this program, we will see how multiple functions are created with the same name,

// but the compiler decides which function to call easily at the compile time itself.

**class** **CompileTimePolymorphism**{

// 1st method with name add

**public** **int** **add**(**int** x, **int** y){

**return** x+y;

}

// 2nd method with name add

**public** **int** **add**(**int** x, **int** y, **int** z){

**return** x+y+z;

}

// 3rd method with name add

**public** **int** **add**(**double** x, **int** y){

**return** (**int**)x+y;

}

// 4th method with name add

**public** **int** **add**(**int** x, **double** y){

**return** x+(**int**)y;

}

}

**class** **Test**{

**public** **static** **void** **main**(String[] args){

CompileTimePolymorphism demo=**new** CompileTimePolymorphism();

// In the below statement, the Compiler looks at the argument types and decides to call method 1

System.out.println(demo.add(2,3));

// Similarly, in the below statement, the compiler calls method 2

System.out.println(demo.add(2,3,4));

// Similarly, in the below statement, the compiler calls method 4

System.out.println(demo.add(2,3.4));

// Similarly, in the below statement, the compiler calls method 3

System.out.println(demo.add(2.5,3));

}

}

In the above example, there are four versions of add methods. The first method takes two parameters while the second one takes three. For the third and fourth methods, there is a change of order of parameters. The compiler looks at the method signature and decides which method to invoke for a particular method call at compile time.  
  
Runtime Polymorphism: Runtime polymorphism, also known as Dynamic Polymorphism, refers to the type of Polymorphism that happens at the run time. What it means is it can't be decided by the compiler. Therefore what shape or value has to be taken depends upon the execution. Hence the name Runtime Polymorphism.

Example:

**class** **AnyVehicle**{

**public** **void** **move**(){

System.out.println(“Any vehicle should move!!”);

}

}

**class** **Bike** **extends** **AnyVehicle**{

**public** **void** **move**(){

System.out.println(“Bike can move too!!”);

}

}

**class** **Test**{

**public** **static** **void** **main**(String[] args){

AnyVehicle vehicle = **new** Bike();

// In the above statement, as you can see, the object vehicle is of type AnyVehicle

// But the output of the below statement will be “Bike can move too!!”,

// because the actual implementation of object ‘vehicle’ is decided during runtime vehicle.move();

vehicle = **new** AnyVehicle();

// Now, the output of the below statement will be “Any vehicle should move!!”,

vehicle.move();

}

}

As the method to call is determined at runtime, as shown in the above code, this is called runtime polymorphism.

**14. 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.**

**15. What is meant by Inheritance?**

The term “inheritance” means “**receiving some quality or behavior from a parent to an offspring**.” In object-oriented programming, inheritance is the mechanism by which an object or class (referred to as a child) is created using the definition of another object or class (referred to as a parent). Inheritance not only helps to keep the implementation simpler but also helps to facilitate code reuse.

**16. What is Abstraction?**

If you are a user, and you have a problem statement, you don't want to know how the components of the software work, or how it's made. You only want to know how the software solves your problem. Abstraction is the method of hiding unnecessary details from the necessary ones. It is one of the main features of OOPs.   
**For example, consider a car. You only need to know how to run a car, and not how the wires are connected inside it. This is obtained using Abstraction.**

**17. How much memory does a class occupy?**

**Classes do not consume any memory.** They are just a blueprint based on which objects are created. Now when **objects are created, they actually initialize the class members and methods and therefore consume memory.**

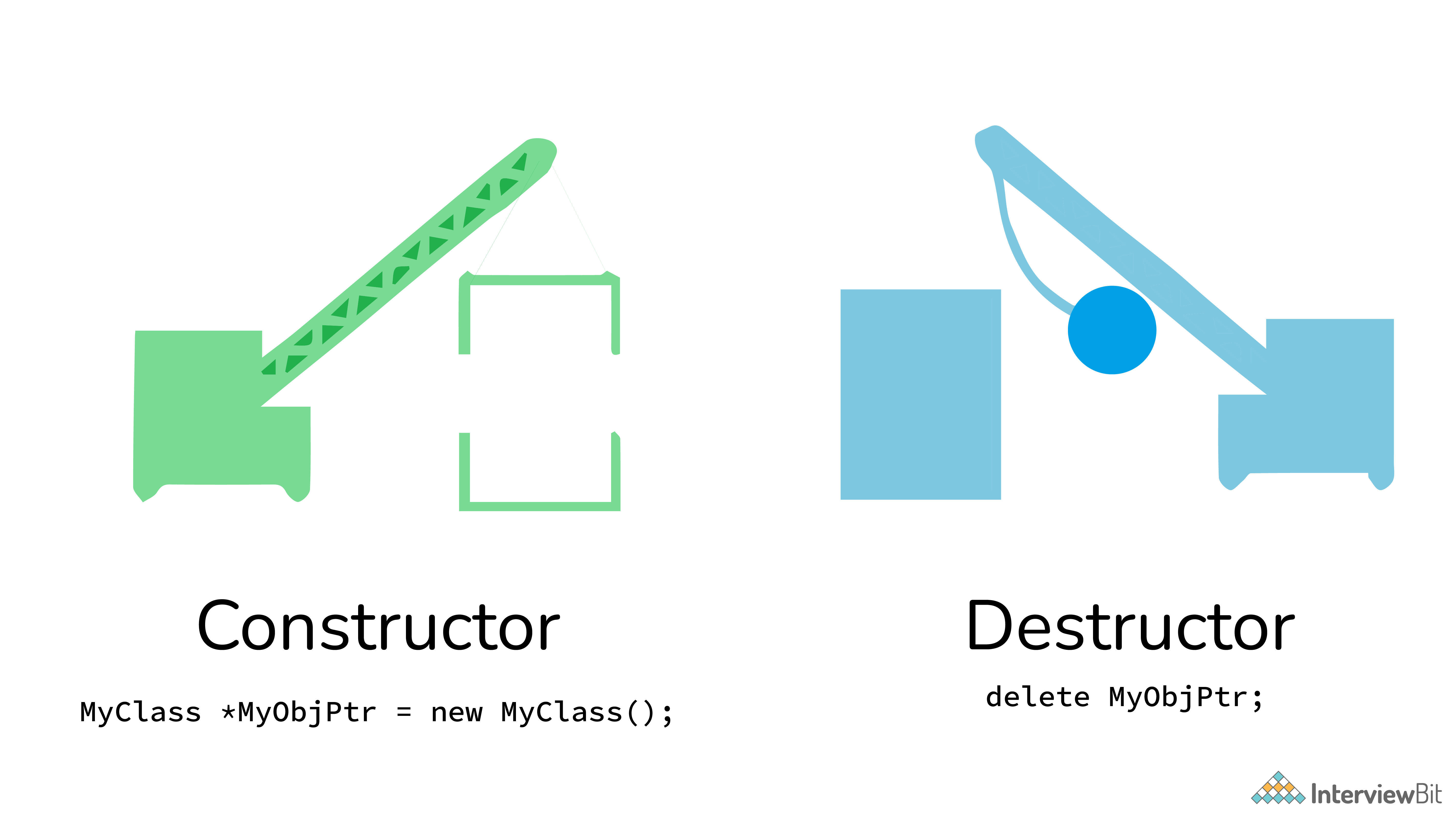
**18. Is it always necessary to create objects from class?**

No. An **object is necessary to be created if the base class has non-static methods.** But if the **class has static methods, then objects don’t need to be created.** You can call the class method directly in this case, using the class name.

**19. What is a constructor?**

Constructors are special methods whose name is the same as the class name. The constructors serve the special purpose of **initializing the objects.**  
For example, suppose there is a class with the name “MyClass”, then when you instantiate this class, you pass the syntax:  
MyClass myClassObject = new MyClass();

Now here, the method called after “new” keyword - MyClass(), is the constructor of this class. **This will help to instantiate the member data and methods and assign them to the object myClassObject.**



**20. What are the various types of constructors in C++?**

The most common classification of constructors includes:

Default constructor: The default constructor is the constructor which doesn’t take any argument. It has no parameters.

**class** **ABC**

{

**int** x;

ABC()

{

x = 0;

}

}

Parameterized constructor: The constructors that take some arguments are known as parameterized constructors.

**class** **ABC**

{

**int** x;

ABC(**int** y)

{

x = y;

}

}

Copy constructor: A copy constructor is a member function that initializes an object using another object of the same class.

**class** **ABC**

{

**int** x;

ABC(**int** y)

{

x = y;

}

// Copy constructor

ABC(ABC abc)

{

x = abc.x;

}

}

**21. What is a copy constructor?**

Copy Constructor is a type of constructor, whose purpose is to copy an object to another. What it means is that a copy constructor will clone an object and its values, into another object, is provided that both the objects are of the same class.

**22. What is a destructor?**

Contrary to constructors, which initialize objects and specify space for them, Destructors are also special methods. But destructors free up the resources and memory occupied by an object. Destructors are automatically called when an object is being destroyed.

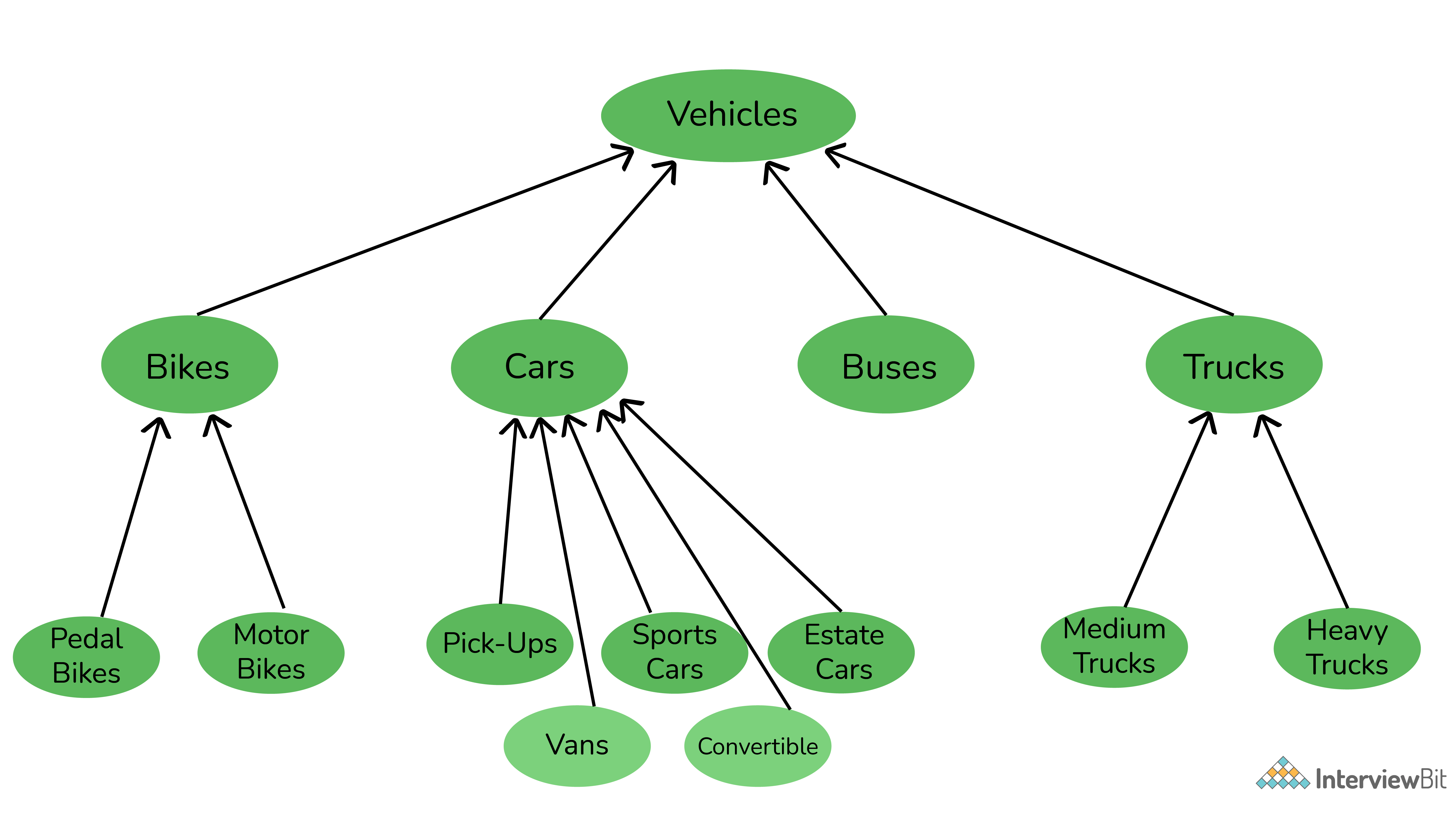
**23. Are class and structure the same? If not, what's the difference between a class and a structure?**

No, class and structure are not the same. Though they appear to be similar, they have differences that make them apart. For example, the **structure is saved in the stack memory, whereas the class is saved in the heap memory.** Also, **Data Abstraction cannot be achieved with the help of structure, but with class, Abstraction is majorly used.**

**24. Explain Inheritance with an example?**

Inheritance is one of the major features of object-oriented programming, by which an entity inherits some characteristics and behaviors of some other entity and makes them their own. **Inheritance helps to improve and facilitate code reuse.**

Let me explain to you with a common example. Let's take three different vehicles - a car, truck, or bus. These three are entirely different from one another with their own specific characteristics and behavior. But. in all three, you will find some common elements, like steering wheel, accelerator, clutch, brakes, etc. Though these elements are used in different vehicles, still they have their own features which are common among all vehicles. This is achieved with inheritance. The car, the truck, and the bus have all inherited the features like steering wheel, accelerator, clutch, brakes, etc, and used them as their own. Due to this, they did not have to **create these components from scratch, thereby facilitating code reuse.**



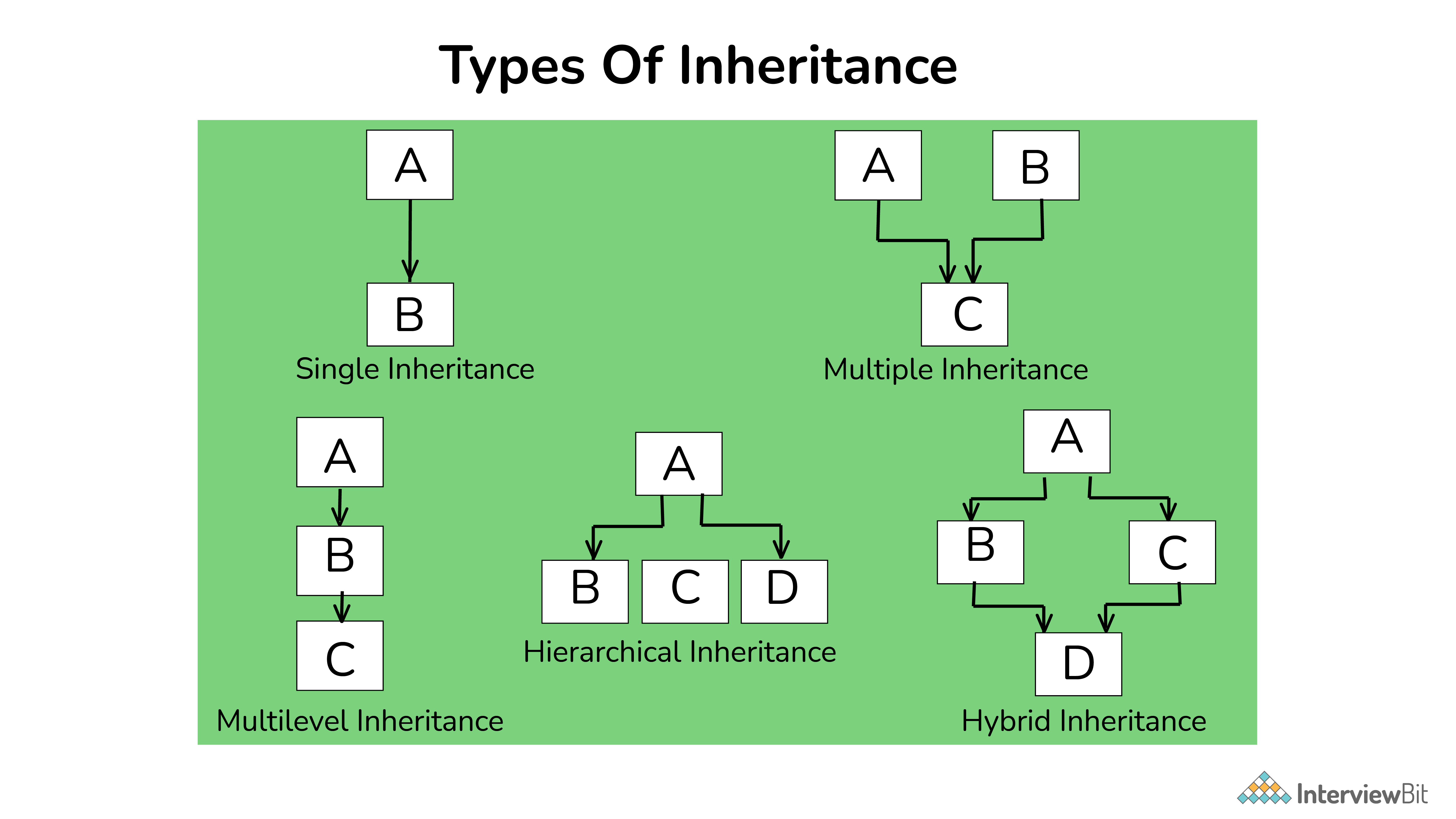
**25. Are there any limitations of Inheritance?**

Yes, with more powers comes more complications. Inheritance is a very powerful feature in OOPs, but it has some limitations too. Inheritance needs more time to process, as it needs to **navigate through multiple classes for its implementation.** Also, the classes involved in Inheritance - the base class and the child class, are very tightly coupled together. So **if one needs to make some changes, they might need to do nested changes in both classes.** Inheritance might be complex for implementation, as well. So **if not correctly implemented, this might lead to unexpected errors or incorrect outputs.**

**26. What are the various types of inheritance?**

The various types of inheritance include:

* Single inheritance
* Multiple inheritances
* Multi-level inheritance
* Hierarchical inheritance
* Hybrid inheritance



**27. What is a subclass?**

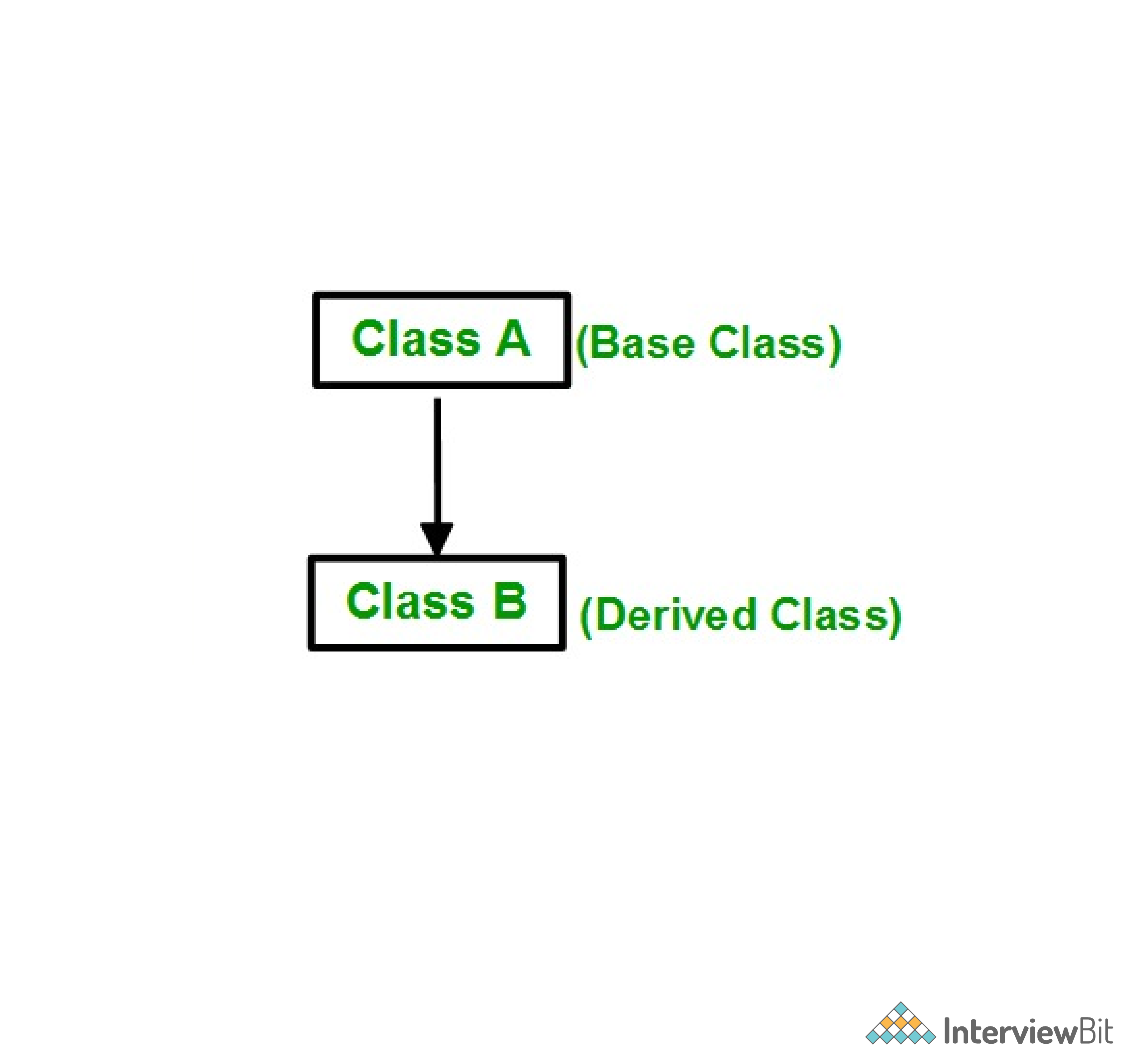
The subclass is a part of Inheritance. The subclass is an entity, which inherits from another class. It is also known as the child class.

**28. Define a superclass?**

In object-oriented programming, a superclass is a class that is inherited by one or more subclass(es). A superclass provides common attributes and methods to its subclasses, which can then extend or override them as needed.

For example, suppose we have a class called "Animal" that defines basic characteristics of animals such as "name", "age", "color", and "sound". We can then create subclasses like "Dog", "Cat", and "Bird" that inherit these common characteristics from the "Animal" class. These subclasses can also add their own unique characteristics or behaviors specific to their type of animal.

In this example, "Animal" is the superclass and "Dog", "Cat", and "Bird" are the subclasses. The subclasses inherit the attributes and methods of the superclass, but can also add or modify their own attributes and methods.

–

**29. What is an 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.

**30. 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.**

**31. 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.**

**32. 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**

**33. How is data abstraction accomplished?**

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

**34. 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.

**35. How is an abstract class different from an interface?**

Interface and abstract class 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 when an abstract class is inherited, the subclass does not need to provide the definition of its abstract method, until and unless the subclass is using it.**

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

Abstract Class:

abstract class Shape {

protected String color;

public Shape(String color) {

this.color = color;

}

public abstract double getArea();

}

class Rectangle extends Shape {

private double length;

private double width;

public Rectangle(String color, double length, double width) {

super(color);

this.length = length;

this.width = width;

}

public double getArea() {

return length \* width;

}

}

class Circle extends Shape {

private double radius;

public Circle(String color, double radius) {

super(color);

this.radius = radius;

}

public double getArea() {

return Math.PI \* radius \* radius;

}

}

Interface Example:

interface Shape {

String getColor();

double getArea();

}

class Rectangle implements Shape {

private String color;

private double length;

private double width;

public Rectangle(String color, double length, double width) {

this.color = color;

this.length = length;

this.width = width;

}

public String getColor() {

return color;

}

public double getArea() {

return length \* width;

}

}

class Circle implements Shape {

private String color;

private double radius;

public Circle(String color, double radius) {

this.color = color;

this.radius = radius;

}

public String getColor() {

return color;

}

public double getArea() {

return Math.PI \* radius \* radius;

}

}

**36. 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.

**37. 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.

**38. 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.

**39. 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.**

**40. Can we run a Java application without implementing the OOPs concept?**

No. Java applications are based on Object-oriented programming models or OOPs concept, and hence they cannot be implemented without it.

However, on the other hand, C++ can be implemented without OOPs, as it also supports the C-like structural programming model.

**OOPs Coding Problems**

**41. What is the output of the below code?**

#**include**<iostream>

**using** **namespace** std;

**class** **BaseClass1** {

**public**:

BaseClass1()

{ cout << " BaseClass1 constructor called" << endl; }

};

**class** **BaseClass2** {

**public**:

BaseClass2()

{ cout << "BaseClass2 constructor called" << endl; }

};

**class** **DerivedClass**: **public** BaseClass1, **public** BaseClass2 {

**public**:

DerivedClass()

{ cout << "DerivedClass constructor called" << endl; }

};

**int** **main**()

{

DerivedClass derived\_class;

**return** 0;

}

**Output:**

BaseClass1 constructor called

BaseClass2 constructor called

DerivedClass constructor called

**Reason:**  
The above program demonstrates Multiple inheritances. So when the Derived class’s constructor is called, it automatically calls the Base class's constructors from left to right order of inheritance.

**42. What will be the output of the below code?**

**class** **Scaler**

{

**static** **int** i;

**static**

{

System.out.println(“a”);

i = 100;

}

}

**public** **class** **StaticBlock**

{

**static**

{

System.out.println(“b”);

}

**public** **static** **void** **main**(String[] args)

{

System.out.println(“c”);

System.out.println(Scaler.i);

}

}

**Output:**

b

c

a

100

**Reason:**  
Firstly the static block inside the main-method calling class will be implemented. Hence ‘b’ will be printed first. Then the main method is called, and now the sequence is kept as expected.

**43. Predict the output?**

#**include**<iostream>

**using** **namespace** std;

**class** **ClassA** {

**public**:

ClassA(**int** ii = 0) : i(ii) {}

**void** **show**() { cout << "i = " << i << endl;}

**private**:

**int** i;

};

**class** **ClassB** {

**public**:

ClassB(**int** xx) : x(xx) {}

**operator** **ClassA**() **const** { **return** ClassA(x); }

**private**:

**int** x;

};

**void** **g**(ClassA a)

{ a.show(); }

**int** **main**() {

ClassB **b**(10);

g(b);

g(20);

getchar();

**return** 0;

}

**Output:**

i = 10

i = 20

**Reason:**  
ClassA contains a conversion constructor. Due to this, the objects of ClassA can have integer values. So the statement g(20) works. Also, ClassB has a conversion operator overloaded. So the statement g(b) also works.

**44. What will be the output in below code?**

**public** **class** **Demo**{

**public** **static** **void** **main**(String[] arr){

System.out.println(“Main1”);

}

**public** **static** **void** **main**(String arr){

System.out.println(“Main2”);

}

}

**Output:**

Main1

**Reason:**  
Here the main() method is overloaded. But JVM only understands the main method which has a String[] argument in its definition. Hence Main1 is printed and the overloaded main method is ignored.

**45. Predict the output?**

#**include**<iostream>

**using** **namespace** std;

**class** **BaseClass**{

**int** arr[10];

};

**class** **DerivedBaseClass1**: **public** BaseClass { };

**class** **DerivedBaseClass2**: **public** BaseClass { };

**class** **DerivedClass**: **public** DerivedBaseClass1, **public** DerivedBaseClass2{};

**int** **main**(**void**)

{

cout<<**sizeof**(DerivedClass);

**return** 0;

}

**Output:**

If the size of the integer is 4 bytes, then the output will be 80.

**Reason:**  
Since DerivedBaseClass1 and DerivedBaseClass1 both inherit from class BaseClass, DerivedClass contains two copies of BaseClass. Hence it results in wastage of space and a large size output. It can be reduced with the help of a virtual base class.

**46. What is the output of the below program?**

#**include**<iostream>

**using** **namespace** std;

**class** **A** {

**public**:

**void** **print**()

{ cout <<" Inside A::"; }

};

**class** **B** : **public** A {

**public**:

**void** **print**()

{ cout <<" Inside B"; }

};

**class** **C**: **public** B {

};

**int** **main**(**void**)

{

C c;

c.print();

**return** 0;

}

**Output:**

Inside B

**Reason:**  
The above program implements a Multi-level hierarchy. So the program is linearly searched up until a matching function is found. Here, it is present in both classes A and B. So class B’s print() method is called.

**1. What is the difference between OOP and SOP?**

|  |  |
| --- | --- |
| Object-Oriented Programming | Structural Programming |
| Object-Oriented Programming is a type of programming which is based on objects rather than just functions and procedures | Provides logical structure to a program where programs are divided functions |
| Bottom-up approach | Top-down approach |
| Provides data hiding | Does not provide data hiding |
| Can solve problems of any complexity | Can solve moderate problems |
| Code can be reused thereby reducing redundancy | Does not support code reusability |

**2. What is Object Oriented Programming?**

Object-Oriented Programming(OOPs) is a type of programming that is based on objects rather than just functions and procedures. Individual objects are grouped into classes. OOPs implements real-world entities like inheritance, polymorphism, hiding, etc into programming. It also allows binding data and code together.

**3. Why use OOPs?**

* OOPs allows clarity in programming thereby allowing simplicity in solving complex problems
* Code can be reused through inheritance thereby reducing redundancy
* Data and code are bound together by encapsulation
* OOPs allows data hiding, therefore, private data is kept confidential
* Problems can be divided into different parts making it simple to solve
* The concept of polymorphism gives flexibility to the program by allowing the entities to have multiple forms

**4. What are the main features of OOPs?**

* Inheritance
* Encapsulation
* Polymorphism
* Data Abstraction

**5. What is an object?**

An object is a real-world entity which is the basic unit of OOPs for example chair, cat, dog, etc. Different objects have different states or attributes, and behaviors.

**6. What is a class?**

A class is a prototype that consists of objects in different states and with different behaviors. It has a number of methods that are common the objects present within that class.

**7. 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.

**8. Can you call the base class method without creating an instance?**

Yes, you can call the base class without instantiating it if:

* It is a static method
* The base class is inherited by some other subclass

**9. What is the difference between a class and an object?**

|  |  |
| --- | --- |
| 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 |

To know more about objects and classes in JAVA, Python, and C++ you can go through the following blogs:

* [Objects in **Java**](https://www.edureka.co/blog/java-object/)
* [Class in **Java**](https://www.edureka.co/blog/java-objects-and-classes/)
* [Objects and classes in **Python**](https://www.edureka.co/blog/python-class/)
* [Objects in **C++**](https://www.edureka.co/blog/object-oriented-programming-in-cpp/#Objects)

## ****OOPs Interview Questions – Inheritance****

#### **10. What is inheritance?**

Inheritance is a feature of OOPs which allows classes inherit common properties from other classes. For example, if there is a class such as ‘vehicle’, other classes like ‘car’, ‘bike’, etc can inherit common properties from the vehicle class. This property helps you get rid of redundant code thereby reducing the overall size of the code.

#### **11. What are the different types of inheritance?**

* Single inheritance
* Multiple inheritance
* Multilevel inheritance
* Hierarchical inheritance
* Hybrid inheritance

#### **12. What is the difference between multiple and multilevel inheritance?**

|  |  |
| --- | --- |
| Multiple Inheritance | Multilevel Inheritance |
| Multiple inheritance comes into picture when a class inherits more than one base class | Multilevel inheritance means a class inherits from another class which itself is a subclass of some other base class |
| Example: A class defining a child inherits from two base classes Mother and Father | Example: A class describing a sports car will inherit from a base class Car which inturn inherits another class Vehicle |

#### **13. What is hybrid inheritance?**

Hybrid inheritance is a combination of multiple and multi-level inheritance.

#### **14. What is hierarchical inheritance?**

Hierarchical inheritance refers to inheritance where one base class has more than one subclasses. For example, the vehicle class can have ‘car’, ‘bike’, etc as its subclasses.

#### **15. What are the limitations of inheritance?**

* Increases the time and effort required to execute a program as it requires jumping back and forth between different classes
* The parent class and the child class get tightly coupled
* Any modifications to the program would require changes both in the parent as well as the child class
* Needs careful implementation else would lead to incorrect results

To know more about inheritance in Java and Python, read the below articles:

* [Inheritance in Java](https://www.edureka.co/blog/inheritance-in-java/)
* [Inheritance in Python](https://www.edureka.co/blog/inheritance-in-python/)

#### **16. What is a superclass?**

A superclass or base class is a class that acts as a parent to some other class or classes. For example, the Vehicle class is a superclass of class Car.

#### **17. What is a subclass?**

A class that inherits from another class is called the subclass. For example, the class Car is a subclass or a derived of Vehicle class.

**18. What is polymorphism?**

Polymorphism refers to the ability to exist in multiple forms. Multiple definitions can be given to a single interface. For example, if you have a class named Vehicle, it can have a method named speed but you cannot define it because different vehicles have different speed. This method will be defined in the subclasses with different definitions for different vehicles.

#### **19. What is static polymorphism?**

Static polymorphism (static binding) is a kind of polymorphism that occurs at compile time. An example of compile-time polymorphism is method overloading.

#### **20. What is dynamic polymorphism?**

Runtime polymorphism or dynamic polymorphism (dynamic binding) is a type of polymorphism which is resolved during runtime. An example of runtime polymorphism is method overriding.

#### **21. What is method overloading?**

Method overloading is a feature of OOPs which makes it possible to give the same name to more than one methods within a class if the arguments passed differ.

#### **22. What is method overriding?**

Method overriding is a feature of OOPs by which the child class or the subclass can redefine methods present in the base class or parent class. Here, the method that is overridden has the same name as well as the signature meaning the arguments passed and the return type.

#### **23. What is operator overloading?**

Operator overloading refers to implementing operators using user-defined types based on the arguments passed along with it.

#### **24. Differentiate between overloading and overriding.**

|  |  |
| --- | --- |
| Overloading | Overriding |
| Two or more methods having the same name but different parameters or signature | Child class redefining methods present in the base class with the same parameters/ signature |
| Resolved during compile-time | Resolved during runtime |

To know more about polymorphism in Java and Python, read the below articles:

* [Polymorphism in Java](https://www.edureka.co/blog/polymorphism-in-java/)
* [Polymorphism in Python](https://www.edureka.co/blog/object-oriented-programming-python/#Polymorphism)

## ****OOPs Interview Questions – Encapsulation****

#### **25. What is encapsulation?**

Encapsulation refers to binding the data and the code that works on that together in a single unit. For example, a class. Encapsulation also allows data-hiding as the data specified in one class is hidden from other classes.

#### **26. What are ‘access specifiers’?**

[Access specifiers or access modifiers are keywords](https://www.edureka.co/blog/access-modifiers-in-java/) that determine the accessibility of methods, classes, etc in OOPs. These access specifiers allow the implementation of encapsulation. The most common access specifiers are public, private and protected. However, there are a few more which are specific to the programming languages.

#### **27. What is the difference between public, private and protected access modifiers?**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Accessibility from own class | Accessibility from derived class | Accessibility from world |
| Public | Yes | Yes | Yes |
| Private | Yes | No | No |
| Protected | Yes | Yes | No |

To know more about encapsulation read along:

* [Encapsulation in Java](https://www.edureka.co/blog/object-oriented-programming/#encapsulation)
* [Encapsulation in C++](https://www.edureka.co/blog/encapsulation-in-cpp/)
* [Encapsulation in Python](https://www.edureka.co/blog/object-oriented-programming-python/#Encapsulation)

### ****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](https://www.edureka.co/blog/data-abstraction-in-cpp/) 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](https://www.edureka.co/blog/java-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 |

To know more about data abstraction, below articles might help you:

* [Abstraction in **Java**](https://www.edureka.co/blog/java-abstraction/)
* [Abstraction in **Python**](https://www.edureka.co/blog/object-oriented-programming-python/#Abstraction)

**Methods and Functions OOPs interview questions**

#### **34. What are virtual functions?**

Virtual functions are functions that are present in the parent class and are overridden by the subclass. These functions are used to achieve runtime polymorphism.

#### **35. What are pure virtual functions?**

Pure virtual functions or [abstract functions](https://www.edureka.co/blog/virtual-function-in-cpp/) are functions that are only declared in the base class. This means that they do not contain any definition in the base class and need to be redefined in the subclass.

#### **36. What is a constructor?**

A constructor is a special type of method that has the same name as the class and is used to initialize objects of that class.

#### **37. What is a destructor?**

A destructor is a method that is automatically invoked when an object is destroyed. The destructor also recovers the heap space that was allocated to the destroyed object, closes the files and database connections of the object, etc.

#### **38. Types of constructors**

[Types of constructors](https://www.edureka.co/blog/python-constructors/) differ from language to language. However, all the possible constructors are:

* Default constructor
* Parameterized constructor
* Copy constructor
* Static constructor
* Private constructor

#### **39. What is a copy constructor?**

A [copy constructor](https://www.edureka.co/blog/constructor-in-java/) creates objects by copying variables from another object of the same class. The main aim of a copy constructor is to create a new object from an existing one.

#### **40. What is the use of ‘finalize’?**

Finalize as an object method used to **free up unmanaged resources and cleanup** before Garbage Collection(GC). It performs memory management tasks.

#### **41. What is Garbage Collection(GC)?**

GC is an implementation of automatic memory management. The Garbage collector frees up space occupied by objects that are no longer in existence.

#### **42. Differentiate between a class and a method.**

|  |  |
| --- | --- |
| Class | Method |
| A class is basically a template that binds the code and data together into a single unit. Classes consist of methods, variables, etc | Callable set of instructions also called a procedure or function that are to be performed on the given data |

#### **43. Differentiate between an abstract class and an interface?**

|  |  |  |
| --- | --- | --- |
| Basis for comparison | Abstract Class | Interface |
| Methods | Can have abstract as well as other methods | Only abstract methods |
| Final Variables | May contain final and non-final variables | Variables declared are final by default |
| Accessibility of Data Members | Can be private, public, etc | Public by default |
| Implementation | Can provide the implementation of an interface | Cannot provide the implementation of an abstract class |

#### **44. What is a final variable?**

A variable whose value does not change. It always refers to the same object by the property of non-transversity.

## ****OOPs Interview Questions – Exception Handling****

### ****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.

### ****47. 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.

## ****OOPs Interview Questions – Limitations of OOPs****

### ****50. What are the limitations of OOPs?****

* Usually not suitable for small problems
* Requires intensive testing
* Takes more time to solve the problem
* Requires proper planning
* The programmer should think of solving a problem in terms of objects

**3) What is a friend function?**

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.

## Association in object oriented programming

Association is a semantically weak relationship (a semantic dependency) between otherwise unrelated objects. An association is a “using” relationship between two or more objects in which the objects have their own lifetime and there is no owner.

As an example, imagine the relationship between a doctor and a patient. A doctor can be associated with multiple patients. At the same time, one patient can visit multiple doctors for treatment or consultation. Each of these objects has its own life cycle and there is no “owner” or parent. The objects that are part of the association relationship can be created and destroyed independently.

In UML an association relationship is represented by a single arrow. An association relationship can be represented as one-to-one, one-to-many, or many-to-many (also known as cardinality). Essentially, an association relationship between two or more objects denotes a path of communication (also called a link) between them so that one object can send a message to another. The following code snippet illustrates how two classes, IDGBlogAccount and IDGBlogEntry, are associated with one another.



1 second of 27 secondsVolume 0%

**public** **class** **IDGBlogAccount**  
   {  
       **private** **IDGBlogEntry**[] blogEntries;  
       *//Other members of the IDGBlogAccount class*  
   }  
**public** **class** **IDGBlogEntry**  
   {  
       **Int32** blogId;  
       **string** caption;  
       **string** text;  
       *//Other members of the IDGBlogEntry class*  
   }

## Aggregation in object oriented programming

Aggregation is a specialized form of association between two or more objects in which each object has its own life cycle but there exists an ownership as well. Aggregation is a typical whole/part or parent/child relationship but it may or may not denote physical containment. An essential property of an aggregation relationship is that the whole or parent (i.e. the owner) can exist without the part or child and vice versa.

As an example, an employee may belong to one or more departments in an organization. However, if an employee’s department is deleted, the employee object would not be destroyed but would live on. Note that the relationships between objects participating in an aggregation cannot be reciprocal—i.e., a department may “own” an employee, but the employee does not own the department. In the following code example, an aggregation relationship is evident between the IDGBlogAuthor and IDGBlogAccount classes.

**public** **class** **IDGBlogAuthor**  
   {  
       **private** **Int32** authorId;  
       **private** **string** firstName;  
       **private** **string** lastName;  
       *//Other members of the IDGBlogAuthor class*  
   }  
**public** **class** **IDGBlogAccount**  
   {  
       **private** **IDGBlogEntry**[] blogEntries;  
       *//Other members of the IDGBlogAccount class*  
   }

Aggregation is usually represented in UML using a line with a hollow diamond. Like association, aggregation can involve a one-to-one, one-to-many, or many-to-many relationship between the participating objects. In the case of a one-to-many or many-to-many relationship, we may say that it is a redundant relationship.

## Composition in object oriented programming

Composition is a specialized form of aggregation. In composition, if the parent object is destroyed, then the child objects also cease to exist. Composition is actually a strong type of aggregation and is sometimes referred to as a “death” relationship. As an example, a house may be composed of one or more rooms. If the house is destroyed, then all of the rooms that are part of the house are also destroyed. The following code snippet illustrates a composition relationship between two classes, House and Room.

**public** **class** **House**  
{  
   **private** **Room** room;  
   **public** **House**()  
   {  
       room = **new** **Room**();  
   }  
}

Like aggregation, composition is also a whole/part or parent/child relationship. However, in composition the life cycle of the part or child is controlled by the whole or parent that owns it. It should be noted that this control can either be direct or transitive. That is, the parent may be directly responsible for the creation or destruction of the child or the parent may use a child that has been already created. Similarly, a parent object might delegate the control to some other parent to destroy the child object. Composition is represented in UML using a line connecting the objects with a solid diamond at the end of the object that owns the other object.

I hope this discussion of association, aggregation, and composition relationships has helped you understand how these three concepts differ. Remember that aggregation and composition are both subsets of association. In both aggregation and composition, an object of one class can be the owner of an object of another class. And in both aggregation and composition, the child objects belong to a single parent object, i.e., they may have only one owner.

Finally, in an aggregation relationship, the life cycles of parent objects and child objects are independent. In a composition relationship, the death of a parent object also means the death of its children.

**Most Popular Database Interview Questions And Answers**

Given below is a list of most popular Database interview questions and answers for your reference.

**Q #1) What do you understand by ‘Database’?**

**Answer:** Database is an organized collection of related data where the data is stored and organized to serve some specific purpose.

**For Example**, A librarian maintain a database of all the information related to the books that are available in the library.

**Q #2) Define DBMS.**

**Answer:**DBMS stands for Database Management System. It is a collection of application programs which allow the user to organize, restore and retrieve information about data efficiently and as effectively as possible.

Some of the popular DBMS’s are MySql, Oracle, Sybase, etc.

**Q #3) Define RDBMS.**

**Answer:**Relational Database Management System(RDBMS) is based on a relational model of data that is stored in databases in separate tables and they are related to the use of a common column. Data can be accessed easily from the relational database using Structured Query Language (SQL).

**Q #4) Enlist the advantages of DBMS.**

**Answer: The advantages of DBMS includes:**

* Data is stored in a structured way and hence redundancy is controlled.
* Validates the data entered and provide restrictions on unauthorized access to the database.
* Provides backup and recovery of the data when required.
* It provides multiple user interfaces.

**Q #5) What do you understand by Data Redundancy?**

**Answer:**Duplication of data in the database is known as data redundancy. As a result of data redundancy, duplicated data is present at multiple locations, hence it leads to wastage of the storage space and the integrity of the database is destroyed.

**Q #6) What are the various types of relationships in Database? Define them.**

**Answer: There are 3 types of relationships in Database:**

* **One-to-one:** One table has a relationship with another table having the similar kind of column. Each primary key relates to only one or no record in the related table.
* **One-to-many:** One table has a relationship with another table that has primary and foreign key relations. The primary key table contains only one record that relates to none, one or many records in the related table.
* **Many-to-many:** Each record in both the tables can relate to many numbers of records in another table.

**Q #7) Explain Normalization and De-Normalization.**

**Answer:**

**Normalization** is the process of removing redundant data from the database by splitting the table in a well-defined manner in order to maintain data integrity. This process saves much of the storage space.

**De-normalization** is the process of adding up redundant data on the table in order to speed up the complex queries and thus achieve better performance.

**Q #8) What are the different types of Normalization?**

**Answer: Different types of Normalization are:**

* **First Normal Form (1NF):** A relation is said to be in 1NF only when all the entities of the table **contain unique or atomic values.**
* **Second Normal Form (2NF):**A relation is said to be in 2NF only if it is in 1NF and all the **non-key attribute of the table is fully dependent on the primary key.**
* **Third Normal Form (3NF):**A relation is said to be in 3NF only if it is in 2NF and every non-key attribute of the **table is not transitively dependent on the primary key.**

**Q #9) What is BCNF?**

**Answer:**BCNF is the Boyce Code Normal form. It is the higher version of 3Nf which does not have any multiple overlapping candidate keys.

**Q #10) What is SQL?**

**Answer:**Structured Query language, SQL is an ANSI(American National Standard Institute) standard programming language that is designed specifically for storing and managing the data in the relational database management system (RDBMS) using all kinds of data operations.

**Q #11) How many SQL statements are used? Define them.**

**Answer:**SQL statements are basically divided into three categories, DDL, DML, and DCL.

**They can be defined as:**

* **Data Definition Language (DDL)**commands are used to define the structure that holds the data. These commands are auto-committed i.e. changes done by the DDL commands on the database are saved permanently.
* **Data Manipulation Language (DML)** commands are used to manipulate the data of the database. These commands are not auto-committed and can be rolled back.
* **Data Control Language (DCL)**commands are used to control the visibility of the data in the database like **revoke access permission for using data in the database.**

**Q #12) Enlist some commands of DDL, DML, and DCL.**

**Answer: Data Definition Language (DDL) commands:**

* CREATE to create a new table or database.
* ALTER for alteration.
* TRUNCATE to delete data from the table.
* DROP to drop a table.
* RENAME to rename a table.

**Data Manipulation Language (DML) commands:**

* INSERT to insert a new row.
* UPDATE to update an existing row.
* DELETE to delete a row.
* MERGE for merging two rows or two tables.

**Data Control Language (DCL) commands:**

* COMMIT to permanently save.
* ROLLBACK to undo the change.
* SAVEPOINT to save temporarily.

**Q #13) Define DML Compiler.**

**Answer:**DML compiler translates DML statements in a query language into a low-level instruction and the generated instruction can be understood by Query Evaluation Engine.

**Q #14) What is DDL interpreter?**

**Answer:**DDL Interpreter interprets the DDL statements and records the generated statements in the table containing metadata.

**Q #15) Enlist the advantages of SQL.**

**Answer: Advantages of SQL are:**

* Simple SQL queries can be used to retrieve a large amount of data from the database very quickly and efficiently.
* SQL is easy to learn and almost every DBMS supports SQL.
* It is easier to manage the database using SQL as no large amount of coding is required.

**Q #16) Explain the terms ‘Record’, ‘Field’ and ‘Table’ in terms of database.**

**Answer:**

**Record:** Record is a collection of values or fields of a specific entity. **For Example,** An employee, Salary account, etc.

**Field:** A field refers to an area within a record that is reserved for specific data. **For Example,** Employee ID.

**Table:** Table is the collection of records of specific types. **For Example,** the Employee table is a collection of records related to all the employees.

**Q #17) What do you understand by Data Independence? What are its two types?**

**Answer:**Data Independence refers to the ability to modify the schema definition in one level in such a way that it does not affect the schema definition in the next higher level.

**The 2 types of Data Independence are:**

* **Physical Data Independence**: It modifies the schema at the physical level without affecting the schema at the conceptual level.
* **Logical Data Independence:**It modifies the schema at the conceptual level without affecting or causing changes in the schema at the view level.

**Q #18) Define the relationship between ‘View’ and ‘Data Independence’.**

**Answer:**View is a virtual table that does not have its data on its own rather the data is defined from one or more underlying base tables.

Views account for logical data independence as the growth and restructuring of base tables are not reflected in views.

**Q #19) What are the advantages and disadvantages of views in the database?**

**Answer: Advantages of Views:**

* As there is no physical location where the data in the view is stored, it generates output without wasting resources.
* Data access is restricted as it does not allow commands like insertion, updation, and deletion.

**Disadvantages of Views:**

* The view becomes irrelevant if we drop a table related to that view.
* Much memory space is occupied when the view is created for large tables.

**Q #20) What do you understand by Functional dependency?**

**Answer:**A relation is said to be in functional dependency when one attribute uniquely defines another attribute.

**For Example,** R is a Relation, X and Y are two attributes. T1 and T2 are two tuples. Then,

T1[X]=T2[X] and T1[Y]=T2[Y]

Means, the value of component X uniquely define the value of component Y.

Also, X->Y means Y is functionally dependent on X.

**Q #21) When is functional dependency said to be the fully functional dependent?**

**Answer:**To fulfill the criteria of fully functional dependency, the relation must meet the requirement of functional dependency.

A functional dependency ‘A’ and ‘B’ are said to be fully functional dependent when removal of any attribute say ‘X’ from ‘A’ means the dependency does not hold anymore.

**Q #22) What do you understand by the E-R model?**

**Answer:**E-R model is an Entity-Relationship model which defines the conceptual view of the database.

The E-R model basically shows the real-world entities and their association/relations. Entities here represent the set of attributes in the database.

**Q #23) Define Entity, Entity type, and Entity set.**

**Answer:**

**Entity** can be anything, be it a place, class or object which has an independent existence in the real world.

**Entity Type** represents a set of entities that have similar attributes.

**Entity Set** in the database represents a collection of entities having a particular entity type.

**Q #24) Define a Weak Entity set.**

**Answer:**Weak Entity set is the one whose primary key comprises its partial key as well as the primary key of its parent entity. This is the case because the entity set may not have sufficient attributes to form a primary key.

**Q #25) Explain the terms ‘Attribute’ and ‘Relations’**

**Answer:**

**Attribute** is described as the properties or characteristics of an entity. **For Example**, Employee ID, Employee Name, Age, etc., can be attributes of the entity Employee.

**Relation** is a two-dimensional table containing a number of rows and columns where every row represents a record of the relation. Here, rows are also known as ‘Tuples’ and columns are known as ‘Attributes’.

**Q #26) What are VDL and SDL?**

**Answer: VDL**is View Definition Language which represents user views and their mapping to the conceptual schema.

**SDL**is Storage Definition Language which specifies the mapping between two schemas.

**Q #27) Define Cursor and its types.**

**Answer:**Cursor is a temporary work area that stores the data, as well as the result set, occurred after manipulation of data retrieved. A cursor can hold only one row at a time.

**The 2 types of Cursor are:**

* **Implicit cursors** are declared automatically when DML statements like INSERT, UPDATE, DELETE is executed.
* **Explicit cursors** have to be declared when SELECT statements that are returning more than one row are executed.

**Q #28) What is the Database transaction?**

**Answer:**Sequence of operation performed which changes the consistent state of the database to another is known as the database transaction. After the completion of the transaction, either the successful completion is reflected in the system or the transaction fails and no change is reflected.

**Q #29) Define Database Lock and its types.**

**Answer:**Database lock basically signifies the transaction about the current status of the data item i.e. whether that data is being used by other transactions or not at the present point of time.

There are two types of Database lock: **Shared Lock**and**Exclusive Lock.**

**Q #30) What is Data Warehousing?**

**Answer:**The storage as well as access to data, that is being derived from the transactions and other sources, from a central location in order to perform the analysis is called Data Warehousing.

**Q #31) What do you understand by Join?**

**Answer:**Join is the process of deriving the relationship between different tables by combining columns from one or more tables having common values in each. When a table joins with itself, it is known as Self Join.

**Q #32) What do you understand by Index hunting?**

**Answer:**Index hunting is the process of boosting the collection of indexes which helps in improving the query performance as well as the speed of the database.

**Q #33) How to improve query performance using Index hunting?**

**Answer: Index hunting help in improving query performance by:**

* Using a query optimizer to coordinate queries with the workload.
* Observing the performance and effect of index and query distribution.

**Q #34) Differentiate between ‘Cluster’ and ‘Non-cluster’ index.**

**Answer:**Clustered index alters the table and re-order the way in which the records are stored in the table. Data retrieval is made faster by using the clustered index.

A Non-clustered index does alter the records that are stored in the table but creates a completely different object within the table.

**Q #35) What are the disadvantages of a Query?**

**Answer: Disadvantages of a Query are:**

* Indexes are not present.
* Stored procedures are excessively compiled.
* Difficulty in interfacing.

**Q #36) What do you understand by Fragmentation?**

**Answer:**Fragmentation is a feature that controls the logical data units, also known as fragments that are stored at different sites of a distributed database system.

**Q #37) Define Join types.**

**Answer:**Given below are the types of Join, which are explained with respect to the tables as an **Example.**

**employee table:**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2017/04/employee-table.jpg)

**employee\_info table:**

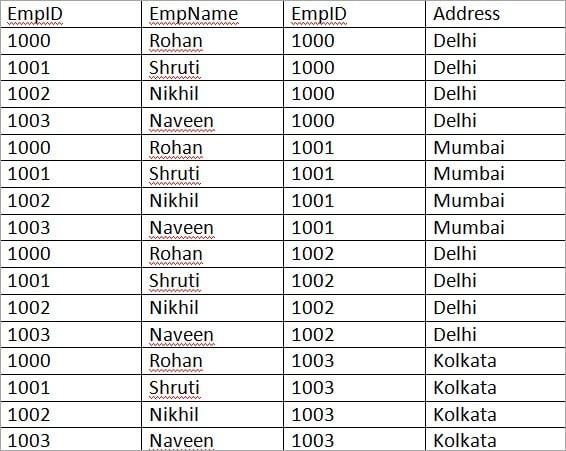
[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2017/04/employee_info-table.jpg)

**a) Inner JOIN:** Inner JOIN is also known as a simple JOIN. This SQL query returns results from both the tables having a common value in rows.

**SQL Query:**

|  |
| --- |
| **SELECT** \* **from** employee, employee\_info **WHERE** employee.EmpID = employee\_info.EmpID ; |

**Result:**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2018/02/Inner-Join-Example.jpg)

**b) Natural JOIN:** This is a type of Inner JOIN that returns results from both the tables having the same data values in the columns of both the tables to be joined.

**SQL Query:**

|  |
| --- |
| **SELECT** \* **from** employee NATURAL JOIN employee\_info; |

**Result:**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2017/04/Natural-JOIN.jpg)

**c) Cross JOIN:** Cross JOIN returns the result as all the records where each row from the first table is combined with each row of the second table.

**SQL Query:**

|  |
| --- |
| **SELECT** \* **from** employee CROSS JOIN employee\_info; |

**Result:**

**Let us do some modification in the above tables to understand Right JOIN, Left JOIN, and Full JOIN.**

**employee table:**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2017/04/employee-table-new.jpg)

**employee\_info table:**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2017/04/employee_info-table-new.jpg)

**a) Right JOIN:** Right JOIN is also known as Right Outer JOIN. This returns all the rows as a result from the right table even if the JOIN condition does not match any records in the left table.

**SQL Query:**

|  |
| --- |
| **SELECT** \* **from** employee RIGHT OUTER JOIN employee\_info **on** (employee.EmpID = employee\_info.EmpID); |

**Result:**

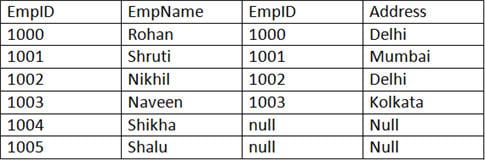
[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2018/02/Right-Join-Example.jpg)

**b) Left JOIN:** Left JOIN is also known as Left Outer JOIN. This returns all the rows as a result of the left table even if the JOIN condition does not match any records in the right table. This is exactly the opposite of Right JOIN.

**SQL Query:**

|  |
| --- |
| **SELECT** \* **from** employee LEFT OUTER JOIN employee\_info **on** (employee.EmpID = employee\_info.EmpID); |

**Result:**

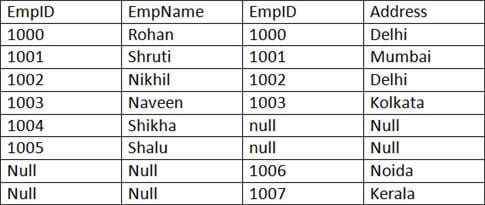
[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2017/04/Left-JOIN.jpg)

**c) Outer/Full JOIN:** Full JOIN return results in combining the result of both the Left JOIN and Right JOIN.

**SQL Query:**

|  |
| --- |
| **SELECT** \* **from** employee **FULL** OUTER JOIN employee\_info **on** (employee.EmpID = employee\_info.EmpID); |

**Result:**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2017/04/Outer-Full-JOIN.jpg)

**Q #38) What do you understand by ‘Atomicity’ and ‘Aggregation’?**

**Answer: Atomicity** is the condition where either all the actions of the transaction are performed or none. This means, when there is an incomplete transaction, the database management system itself will undo the effects done by the incomplete transaction.

**Aggregation** is the concept of expressing the relationship with the collection of entities and their relationships.

**Q #39) Define Phantom deadlock.**

**Answer:**Phantom deadlock detection is the condition where the deadlock does not actually exist but due to a delay in propagating local information, deadlock detection algorithms identify the deadlocks.

**Q #40) Define checkpoint.**

**Answer:**Checkpoint declares a point before which all the logs are stored permanently in the storage disk and is the inconsistent state. In the case of crashes, the amount of work and time is saved as the system can restart from the checkpoint.

**Q #41) What is Database partitioning?**

**Answer:**Database partitioning is the process of partitioning tables, indexes into smaller pieces in order to manage and access the data at a finer level.

This process of partitioning reduces the cost of storing a large amount of data as well as enhances the performance and manageability.

**Q #42) Explain the importance of Database partitioning.**

**Answer: The importance of Database partitioning are:**

* Improves query performance and manageability.
* Simplifies common administration tasks.
* Acts as a key tool for building systems with extremely high availability requirements.
* Allows accessing a large part of a single partition.

**Q #43) Explain the Data Dictionary.**

**Answer:**Data dictionary is a set of information describing the content and structure of the tables and database objects. The job of the information stored in the data dictionary is to control, manipulate and access the relationship between database elements.

**Q #44) Explain the Primary Key and Composite Key.**

**Answer: Primary Key** is that column of the table whose every row data is uniquely identified. Every row in the table must have a primary key and no two rows can have the same primary key. Primary key value can never be null nor can it be modified or updated.

**Composite Key**is a form of the candidate key where a set of columns will uniquely identify every row in the table.

**Q #45) What do you understand by the Unique key?**

**Answer:**A Unique key is the same as the primary key whose every row data is uniquely identified with a difference of null value i.e. Unique key allows one value as a NULL value.

**Q #46) What do you understand by Database Triggers?**

**Answer:**A set of commands that automatically get executed when an event like Before Insert, After Insert, On Update, On Delete of row occurs in a table is called as Database trigger.

**Q #47) Define Stored procedures.**

**Answer:**A Stored procedure is a collection of pre-compiled SQL Queries, which when executed denotes a program taking input, process and gives the output.

**Q #48) What do you understand by B-Trees?**

**Answer:**B-Tree represents the data structure in the form of a tree for external memory that reads and writes large blocks of data. It is commonly used in databases and file systems where all the insertions, deletions, sorting, etc., are done in logarithmic time.

**Q #49) Name the different data models that are available for database systems.**

**Answer: Different data models are:**

* Relational model
* Network model
* Hierarchical model

**Q #50) Differentiate between ‘DELETE’, ‘TRUNCATE’ and ‘DROP’ commands.**

**Answer:**After the execution of **‘DELETE’** operation, COMMIT and ROLLBACK statements can be performed to retrieve the lost data.

After the execution of**‘TRUNCATE’** operation, COMMIT, and ROLLBACK statements cannot be performed to retrieve the lost data.

**‘DROP’** command is used to drop the table or key like the primary key/foreign key.

**Q #51) Based on the given table, solve the following queries.**

**Employee table**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2017/04/Employee-table-1.jpg)

**a)** Write the SELECT command to display the details of the employee with empid as 1004.

**SQL Query:**

|  |
| --- |
| **SELECT** empId, empName, Age, Address **from** Employee **WHERE** empId = 1004; |

**Result:**

[SELECT command](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2017/04/SELECT-command.jpg)

**b)** Write the SELECT command to display all the records of table Employees.

**SQL Query:**

|  |
| --- |
| **SELECT** \* **from** Employee; |

**Result:**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2017/04/display-all-records.jpg)

**c)** Write the SELECT command to display all the records of the employee whose name starts with the character ‘R’.

**SQL Query:**

|  |
| --- |
| **SELECT** \* **from** Employee **WHERE** empName LIKE ‘R%’; |

**Result:**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2017/04/name-starts-with-character-R.jpg)

**d)** Write a SELECT command to display id, age and name of the employees with their age in both ascending and descending order.

**SQL Query:**

|  |
| --- |
| **SELECT** empId, empName, Age **from** Employee  **ORDER** **BY** Age; |

**Result:**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2017/04/employees-with-their-age-in-ascending.jpg)

|  |
| --- |
| **SELECT** empId, empName, Age **from** Employee  **ORDER** **BY** Age **Desc**; |

**Result:**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2017/04/employees-with-their-age-in-descending.jpg)

**e)** Write the SELECT command to calculate the total amount of salary on each employee from the below Emp table.

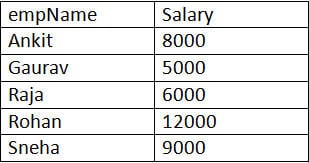
**Emp table:**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2017/04/Emp-table-1-e1492776699935.jpg)

**SQL Query:**

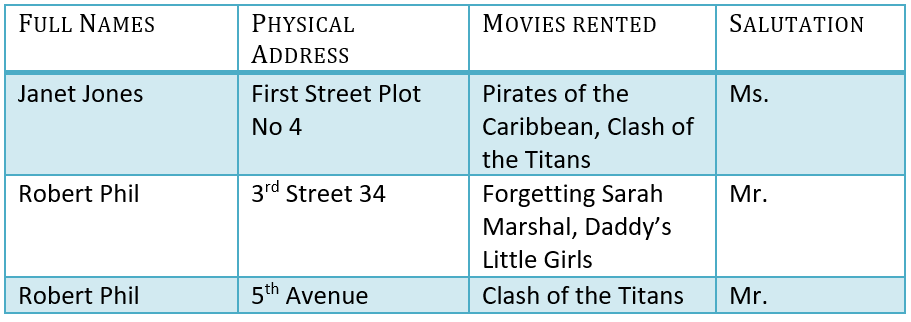
|  |
| --- |
| **SELECT** empName, SUM(Salary) **from** Emp **GROUP** **BY** empName; |

**Result:**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2017/04/Result.jpg)

**Database Normalization With Examples**

Database **Normalization Example** can be easily understood with the help of a case study. Assume, a video library maintains a database of movies rented out. Without any normalization in database, all information is stored in one table as shown below. Let’s understand Normalization database with normalization example with solution:



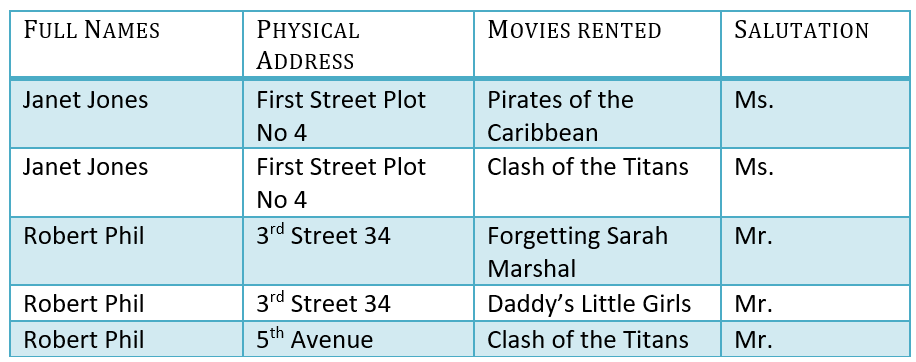
Here you see **Movies Rented column has multiple values.** Now let’s move into 1st Normal Forms:

**1NF (First Normal Form) Rules**

* Each table cell should contain a single value.
* Each record needs to be unique.

The above table in 1NF-

**1NF Example**



Example of 1NF in DBMS

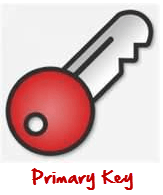
Before we proceed let’s understand a few things —

**What is a KEY in SQL?**

A **KEY in SQL** is a value used to identify records in a table uniquely. An SQL KEY is a single column or combination of multiple columns used to uniquely identify rows or tuples in the table. SQL Key is used to identify duplicate information, and it also helps establish a relationship between multiple tables in the database.

Note: Columns in a table that are NOT used to identify a record uniquely are called non-key columns.

**What is a Primary Key?**



Primary Key in DBMS

A primary is a single column value used to identify a database record uniquely.

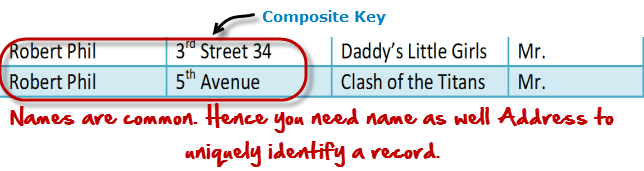
It has following attributes

* A [primary key](https://www.guru99.com/dbms-keys.html) cannot be NULL
* A primary key value must be unique
* The primary key values should rarely be changed
* The primary key must be given a value when a new record is inserted.

**What is Composite Key?**

A composite key is a primary key composed of multiple columns used to identify a record uniquely

In our database, we have two people with the same name Robert Phil, but they live in different places.



Composite key in Database

Hence, we require both Full Name and Address to identify a record uniquely. That is a composite key.

Let’s move into second normal form 2NF

**2NF (Second Normal Form) Rules**

* Rule 1- Be in 1NF
* Rule 2- Single Column Primary Key that does not functionally dependant on any subset of candidate key relation

It is clear that we can’t move forward to make our simple database in 2nd Normalization form unless we partition the table above.



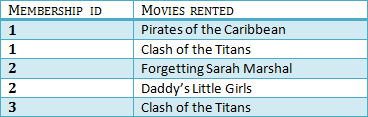


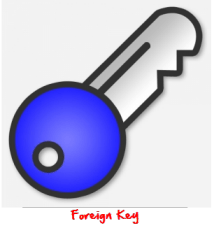
We have divided our 1NF table into two tables viz. Table 1 and Table2. Table 1 contains member information. Table 2 contains information on movies rented.

We have introduced a new column called Membership\_id which is the primary key for table 1. Records can be uniquely identified in Table 1 using membership id

**Database – Foreign Key**

In Table 2, Membership\_ID is the Foreign Key

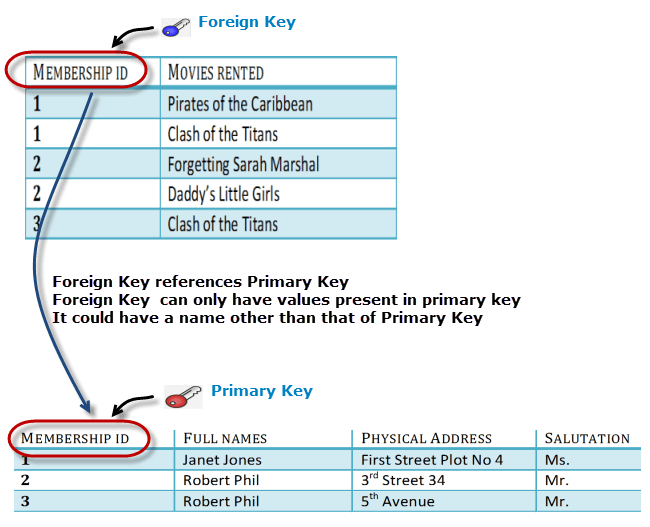




Foreign Key in DBMS

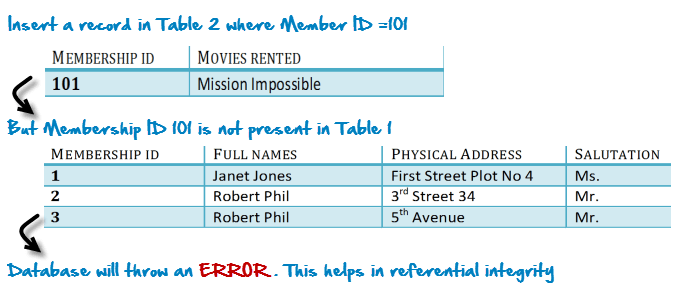
Foreign Key references the primary key of another Table! It helps connect your Tables

* A foreign key can have a different name from its primary key
* It ensures rows in one table have corresponding rows in another
* Unlike the Primary key, they do not have to be unique. Most often they aren’t
* Foreign keys can be null even though primary keys can not



**Why do you need a foreign key?**

Suppose, a novice inserts a record in Table B such as



You will only be able to insert values into your foreign key that exist in the unique key in the parent table. This helps in referential integrity.

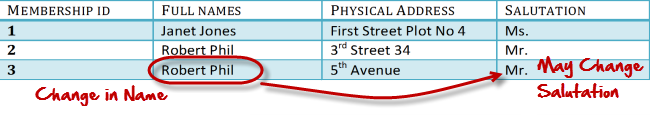
The above problem can be overcome by declaring membership id from Table2 as foreign key of membership id from Table1

Now, if somebody tries to insert a value in the membership id field that does not exist in the parent table, an error will be shown!

**What are transitive functional dependencies?**

A transitive [functional dependency](https://www.guru99.com/dbms-functional-dependency.html) is when changing a non-key column, might cause any of the other non-key columns to change

Consider the table 1. Changing the non-key column Full Name may change Salutation.



Let’s move into 3NF

**3NF (Third Normal Form) Rules**

* Rule 1- Be in 2NF
* Rule 2- Has no transitive functional dependencies

To move our 2NF table into 3NF, we again need to again divide our table.

**3NF Example**

Below is a 3NF example in SQL database:







We have again divided our tables and created a new table which stores Salutations.

There are no transitive functional dependencies, and hence our table is in 3NF

In Table 3 Salutation ID is primary key, and in Table 1 Salutation ID is foreign to primary key in Table 3

Now our little example is at a level that cannot further be decomposed to attain higher normal form types of normalization in DBMS. In fact, it is already in higher normalization forms. Separate efforts for moving into next levels of normalizing data are normally needed in complex databases. However, we will be discussing next levels of normalisation in DBMS in brief in the following.

**BCNF (Boyce-Codd Normal Form)**

Even when a database is in 3rd Normal Form, still there would be anomalies resulted if it has more than one **Candidate**Key.

Sometimes is BCNF is also referred as **3.5 Normal Form.**

**4NF (Fourth Normal Form) Rules**

If no database table instance contains two or more, independent and multivalued data describing the relevant entity, then it is in 4th Normal Form.

**5NF (Fifth Normal Form) Rules**

A table is in 5th Normal Form only if it is in 4NF and it cannot be decomposed into any number of smaller tables without loss of data.

**6NF (Sixth Normal Form) Proposed**

6th Normal Form is not standardized, yet however, it is being discussed by database experts for some time. Hopefully, we would have a clear & standardized definition for 6th Normal Form in the near future…

That’s all to SQL Normalization!!!

**Summary**

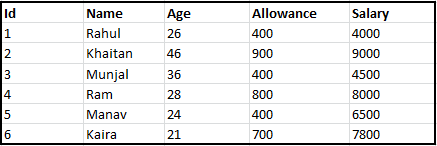
* Database designing is critical to the successful implementation of a database management system that meets the data requirements of an enterprise system.
* Normalization in DBMS is a process which helps produce database systems that are cost-effective and have better security models.
* Functional dependencies are a very important component of the normalize data process
* Most database systems are normalized database up to the third normal forms in DBMS.
* A primary key uniquely identifies are record in a Table and cannot be null
* A foreign key helps connect table and references a primary key

Transaction, Commit & Rollback

**1.** **COMMIT-**  
COMMIT in SQL is a transaction control language that is used to permanently save the changes done in the transaction in tables/databases. The database cannot regain its previous state after its execution of [commit.](https://www.geeksforgeeks.org/sql-transactions/)

**Example:** Consider the following STAFF table with records:

**STAFF**



sql>

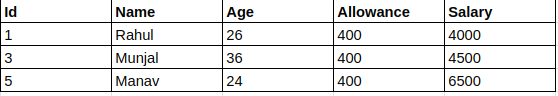
SELECT \*

FROM Staff

WHERE Allowance = 400;

sql> COMMIT;

**Output:**



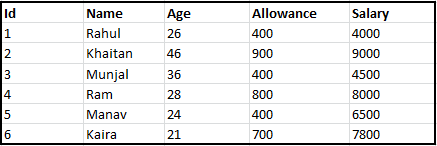
So, the SELECT statement produced the output consisting of three rows.

**2. ROLLBACK**

ROLLBACK in SQL is a transactional control language that is used to undo the transactions that have not been saved in the database. The command is only been used to undo changes since the last COMMIT.

**Example:** Consider the following STAFF table with records:

**STAFF**



sql>

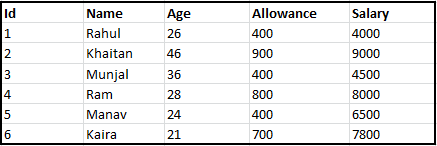
SELECT \*

FROM EMPLOYEES

WHERE ALLOWANCE = 400;

sql> ROLLBACK;

**Output:**



So, the SELECT statement produced the same output with the ROLLBACK command.

## ****Difference between COMMIT and ROLLBACK****

|  | COMMIT | ROLLBACK |
| --- | --- | --- |
| 1. | COMMIT permanently saves the changes made by the current transaction. | ROLLBACK undo the changes made by the current transaction. |
| 2. | The transaction can not undo changes after COMMIT execution. | Transaction reaches its previous state after ROLLBACK. |
| 3. | When the transaction is successful, COMMIT is applied. | When the transaction is aborted, incorrect execution, system failure ROLLBACK occurs. |
| 4. | COMMIT statement permanently save the state, when all the statements are executed successfully without any error. | In ROLLBACK statement if any operations fail during the completion of a transaction, it cannot permanently save the change and we can undo them using this statement. |
| 5. | **Syntax of COMMIT statement are:**  COMMIT; | **Syntax of ROLLBACK statement are:**  ROLLBACK; |

# ACID Properties in DBMS

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 05 Jul, 2022

A [**transaction**](https://www.geeksforgeeks.org/sql-transactions/) is a single logical unit of work that accesses and possibly modifies the contents of a database. Transactions access data using read and write operations.   
In order to maintain consistency in a database, before and after the transaction, certain properties are followed. These are called **ACID** properties.

### Timeline Description automatically generated with medium confidence

### ****Atomicity:****

By this, we mean that either the entire transaction takes place at once or doesn’t happen at all. There is no midway i.e. transactions do not occur partially. Each transaction is considered as one unit and either runs to completion or is not executed at all. It involves the following two operations.   
—**Abort**: If a transaction aborts, changes made to the database are not visible.   
—**Commit**: If a transaction commits, changes made are visible.   
Atomicity is also known as the ‘All or nothing rule’.

Consider the following transaction **T** consisting of **T1** and **T2**: Transfer of 100 from account **X** to account **Y**.

![Table

Description automatically generated](data:image/jpeg;base64,/9j/4AAQSkZJRgABAQEAYABgAAD/4RDmRXhpZgAATU0AKgAAAAgABAE7AAIAAAAJAAAISodpAAQAAAABAAAIVJydAAEAAAASAAAQzOocAAcAAAgMAAAAPgAAAAAc6gAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAE1pdGhsZXNoAAAABZADAAIAAAAUAAAQopAEAAIAAAAUAAAQtpKRAAIAAAADMDkAAJKSAAIAAAADMDkAAOocAAcAAAgMAAAIlgAAAAAc6gAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADIwMTg6MDI6MTQgMTI6MjM6MzUAMjAxODowMjoxNCAxMjoyMzozNQAAAE0AaQB0AGgAbABlAHMAaAAAAP/hCxtodHRwOi8vbnMuYWRvYmUuY29tL3hhcC8xLjAvADw/eHBhY2tldCBiZWdpbj0n77u/JyBpZD0nVzVNME1wQ2VoaUh6cmVTek5UY3prYzlkJz8+DQo8eDp4bXBtZXRhIHhtbG5zOng9ImFkb2JlOm5zOm1ldGEvIj48cmRmOlJERiB4bWxuczpyZGY9Imh0dHA6Ly93d3cudzMub3JnLzE5OTkvMDIvMjItcmRmLXN5bnRheC1ucyMiPjxyZGY6RGVzY3JpcHRpb24gcmRmOmFib3V0PSJ1dWlkOmZhZjViZGQ1LWJhM2QtMTFkYS1hZDMxLWQzM2Q3NTE4MmYxYiIgeG1sbnM6ZGM9Imh0dHA6Ly9wdXJsLm9yZy9kYy9lbGVtZW50cy8xLjEvIi8+PHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9InV1aWQ6ZmFmNWJkZDUtYmEzZC0xMWRhLWFkMzEtZDMzZDc1MTgyZjFiIiB4bWxuczp4bXA9Imh0dHA6Ly9ucy5hZG9iZS5jb20veGFwLzEuMC8iPjx4bXA6Q3JlYXRlRGF0ZT4yMDE4LTAyLTE0VDEyOjIzOjM1LjA4ODwveG1wOkNyZWF0ZURhdGU+PC9yZGY6RGVzY3JpcHRpb24+PHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9InV1aWQ6ZmFmNWJkZDUtYmEzZC0xMWRhLWFkMzEtZDMzZDc1MTgyZjFiIiB4bWxuczpkYz0iaHR0cDovL3B1cmwub3JnL2RjL2VsZW1lbnRzLzEuMS8iPjxkYzpjcmVhdG9yPjxyZGY6U2VxIHhtbG5zOnJkZj0iaHR0cDovL3d3dy53My5vcmcvMTk5OS8wMi8yMi1yZGYtc3ludGF4LW5zIyI+PHJkZjpsaT5NaXRobGVzaDwvcmRmOmxpPjwvcmRmOlNlcT4NCgkJCTwvZGM6Y3JlYXRvcj48L3JkZjpEZXNjcmlwdGlvbj48L3JkZjpSREY+PC94OnhtcG1ldGE+DQogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgIDw/eHBhY2tldCBlbmQ9J3cnPz7/2wBDAAcFBQYFBAcGBQYIBwcIChELCgkJChUPEAwRGBUaGRgVGBcbHichGx0lHRcYIi4iJSgpKywrGiAvMy8qMicqKyr/2wBDAQcICAoJChQLCxQqHBgcKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKir/wAARCACSAWQDASIAAhEBAxEB/8QAHwAAAQUBAQEBAQEAAAAAAAAAAAECAwQFBgcICQoL/8QAtRAAAgEDAwIEAwUFBAQAAAF9AQIDAAQRBRIhMUEGE1FhByJxFDKBkaEII0KxwRVS0fAkM2JyggkKFhcYGRolJicoKSo0NTY3ODk6Q0RFRkdISUpTVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXqDhIWGh4iJipKTlJWWl5iZmqKjpKWmp6ipqrKztLW2t7i5usLDxMXGx8jJytLT1NXW19jZ2uHi4+Tl5ufo6erx8vP09fb3+Pn6/8QAHwEAAwEBAQEBAQEBAQAAAAAAAAECAwQFBgcICQoL/8QAtREAAgECBAQDBAcFBAQAAQJ3AAECAxEEBSExBhJBUQdhcRMiMoEIFEKRobHBCSMzUvAVYnLRChYkNOEl8RcYGRomJygpKjU2Nzg5OkNERUZHSElKU1RVVldYWVpjZGVmZ2hpanN0dXZ3eHl6goOEhYaHiImKkpOUlZaXmJmaoqOkpaanqKmqsrO0tba3uLm6wsPExcbHyMnK0tPU1dbX2Nna4uPk5ebn6Onq8vP09fb3+Pn6/9oADAMBAAIRAxEAPwD6Rorm/Hpk/wCEXSOK4uLfz9T06B5Lad4ZNkl7CjgOhDLlWYZBBwaP+ED0j/n88Qf+FHqH/wAfoA6Siub/AOED0j/n88Qf+FHqH/x+j/hA9I/5/PEH/hR6h/8AH6AOkorm/wDhA9I/5/PEH/hR6h/8fo/4QPSP+fzxB/4Ueof/AB+gDpKK8yF78Pjq8mm/274kWaO+OnPI2r6usC3P/PIzGTyw3oN3Paup/wCED0j/AJ/PEH/hR6h/8foA6SivPNAsfDviXVNXtNNPisw6TdvYz3kniO8ETzpt3xqPtO8kbhyVA64Jrf8A+ED0j/n88Qf+FHqH/wAfoA6Siub/AOED0j/n88Qf+FHqH/x+j/hA9I/5/PEH/hR6h/8AH6AOkorgfsPg37LNcf2z4g2w34011/t7U94uSwUR7PN3E/MDwMbTu+7zW1/wgekf8/niD/wo9Q/+P0AdJRXBaxp3hPQr+3sb698WPdXETzRw2eqavdOUQqGYiJ2wAXUZOOoqDVYvBWivGmo6r4ojd7b7WyJq2ryNBD/z0lCyEwrweZNo+Vv7pwAeiUV51qC+BtLvfst5rfiHcFieSWLW9UlhgWQ4RpZUlKRBuoLsoxz05rof+ED0j/n88Qf+FHqH/wAfoA6Siub/AOED0j/n88Qf+FHqH/x+j/hA9I/5/PEH/hR6h/8AH6AOkorm/wDhA9I/5/PEH/hR6h/8fo/4QPSP+fzxB/4Ueof/AB+gDpKK8/8ABXhOy1fwD4f1LUNQ8QTXd5pltPPJ/wAJFfrvd4lZjgTADJJ4AxW5/wAIHpH/AD+eIP8Awo9Q/wDj9AHSUVzf/CB6R/z+eIP/AAo9Q/8Aj9H/AAgekf8AP54g/wDCj1D/AOP0AdJRXN/8IHpH/P54g/8ACj1D/wCP0f8ACB6R/wA/niD/AMKPUP8A4/QB0lFc3/wgekf8/niD/wAKPUP/AI/R/wAIHpH/AD+eIP8Awo9Q/wDj9AHSUVx8nhnw3DrUGky6l4gS9uIXnhibxDqI8xEKhyG87BILrkZzznGKNJ8M+G9c0yLUNL1LxBPaTFhHL/wkOoqH2sVJGZgSMg4PQjBGQQaAOworz/VLTwbouoiy1HVfEsUmYw8i6zqrwwmRtqeZKshSPJ/vsOoPQ03UYfBOlas2nX2seI0njMYmdda1V4bcyHCCWVZCkRbjAdhwQehGQD0KivPJIfBMWt/2U+seIxcfaFtC41rVTAs7AFYmnEnlrIcrhC2csBjJFb//AAgekf8AP54g/wDCj1D/AOP0AdJRXN/8IHpH/P54g/8ACj1D/wCP0f8ACB6R/wA/niD/AMKPUP8A4/QB0lFc3/wgekf8/niD/wAKPUP/AI/R/wAIHpH/AD+eIP8Awo9Q/wDj9AHSUVzf/CB6R/z+eIP/AAo9Q/8Aj9H/AAgekf8AP54g/wDCj1D/AOP0AdJRXN/8IHpH/P54g/8ACj1D/wCP0eAjJ/wi7xy3FxceRqeowJJczvNJsjvZkQF3JZsKqjJJOBQB0lFFFAHN+PP+Rdtf+w1pX/pwt66SuW+IrTr4SiazjjluBq2mGKOWQojN9vgwGYBioJ6kA49D0qX7d43/AOhe8P8A/g+n/wDkOgDpKK5v7d43/wChe8P/APg+n/8AkOj7d43/AOhe8P8A/g+n/wDkOgDpKK5v7d43/wChe8P/APg+n/8AkOj7d43/AOhe8P8A/g+n/wDkOgDxm58Pa42oeI4oLHW5b8+Mv7S0/TJdNcafdqJFxNJOIxhQu4/60D5R8pzWqnhTVbr4halJrss1lqH/AAkgurDUE8PXN5I1rlfKjW8RvLjiKEoyMMD5i2a9R+3eN/8AoXvD/wD4Pp//AJDo+3eN/wDoXvD/AP4Pp/8A5DoA8gtPBV7p+g6+bbw1NBaHx47XtvBYFZLrRwy/IiKu6SHJBCrkEBsA8il1rwpqN9BqC6LoWoW3hm88UabJY6fHZy27RRLGVuZRDgNEhbnJC+vFevfbvG//AEL3h/8A8H0//wAh0fbvG/8A0L3h/wD8H0//AMh0AeW+MPBlxJ441m1uLf7Bo0ljbR6HcWvhy41H7BtB8zyPs7L9mcPhskHdx6YPp/hy1sbfxZrzRaXqEWptDZrfarcRMkOolYiEMWWIyoyGAAwSM5p/27xv/wBC94f/APB9P/8AIdH27xv/ANC94f8A/B9P/wDIdAHJzadqLfElvGo0ic6bFdrYGz8mTznwpiOoeX3KljGPlyYdzegHa61DYya7oD3ml3l5cR3UhtbiBWMdm/kuC8hBAClcqMg8sPrVb7d43/6F7w//AOD6f/5Do+3eN/8AoXvD/wD4Pp//AJDoAzfHMGjy3yS6hYeJDqEdlIlpeaGt2G+cjdHutzgHciH97henPBxhWUmuaAdRbxTpl/qOpatoVpHvsbN50muo45FkiZowVjJZwdzbUO4kEAHHX/bvG/8A0L3h/wD8H0//AMh0fbvG/wD0L3h//wAH0/8A8h0Aed2mi6n4c8E+JfCF5pd9e6jq+n29vZTQWkksErGwitTvlUFY9skbE72X5cEda9is4GtrG3gdt7RRKhb1IGM1g/bvG/8A0L3h/wD8H0//AMh0fbvG/wD0L3h//wAH0/8A8h0AdJRXN/bvG/8A0L3h/wD8H0//AMh0fbvG/wD0L3h//wAH0/8A8h0AdJRXN/bvG/8A0L3h/wD8H0//AMh0fbvG/wD0L3h//wAH0/8A8h0AHw4/5JZ4U/7Atn/6ISukrifDlt438P8AhXStG/sTw/cf2dZQ2nnf23OnmeWgTdt+yHGcZxk1pfbvG/8A0L3h/wD8H0//AMh0AdJRXN/bvG//AEL3h/8A8H0//wAh0fbvG/8A0L3h/wD8H0//AMh0AdJRXN/bvG//AEL3h/8A8H0//wAh0fbvG/8A0L3h/wD8H0//AMh0AdJRXN/bvG//AEL3h/8A8H0//wAh0fbvG/8A0L3h/wD8H0//AMh0AUvidos2r+FFewW6GoW9zF5EtkP30aSOIZsYB48qSTPpjPbNaEWnaTYeLNLgtdFuUntdLkgtbyJGFtbwBox5JOcBjhSBjOEPI7s+3eN/+he8P/8Ag+n/APkOj7d43/6F7w//AOD6f/5DoAxfHWr/AG3VIvDNxYatHpZ2T6he2+k3NysyBsi3jMUbDJIG9jjavAyWymPr1teNpPjnw0uk6hNeeIrkyafMllI8EiTQRR7nlA2RlGRtwcqcKMA5Gey+3eN/+he8P/8Ag+n/APkOj7d43/6F7w//AOD6f/5DoA4uaxv20HUvBzabqD6ld+Imuo7k2chtjA14tz5xnA8sbUBG0tu3KFxyK9Yrm/t3jf8A6F7w/wD+D6f/AOQ6Pt3jf/oXvD//AIPp/wD5DoA6Siub+3eN/wDoXvD/AP4Pp/8A5Do+3eN/+he8P/8Ag+n/APkOgDpKK5v7d43/AOhe8P8A/g+n/wDkOj7d43/6F7w//wCD6f8A+Q6AOkorm/t3jf8A6F7w/wD+D6f/AOQ6Pt3jf/oXvD//AIPp/wD5DoA6Sub8B/8AIu3X/Ya1X/04XFH27xv/ANC94f8A/B9P/wDIdRfDpp28JSteRxxXB1bUzLHFIXRW+3z5CsQpYA9CQM+g6UAdTRRRQBzfjz/kXbX/ALDWlf8Apwt66SuU+JV5Bp3gxb27Zlt7bVdMmlZUZyFW/gJIVQSeB0AJNUf+FyeCf+ghff8AgnvP/jVK6REqkIfE0juaK4b/AIXJ4J/6CF9/4J7z/wCNUf8AC5PBP/QQvv8AwT3n/wAao5kR7el/MvvR3NFcN/wuTwT/ANBC+/8ABPef/GqP+FyeCf8AoIX3/gnvP/jVHMg9vS/mX3o7miuG/wCFyeCf+ghff+Ce8/8AjVH/AAuTwT/0EL7/AME95/8AGqOZB7el/MvvR3NFcN/wuTwT/wBBC+/8E95/8ao/4XJ4J/6CF9/4J7z/AONUcyD29L+ZfejuaK4b/hcngn/oIX3/AIJ7z/41R/wuTwT/ANBC+/8ABPef/GqOZB7el/MvvR3NFcN/wuTwT/0EL7/wT3n/AMao/wCFyeCf+ghff+Ce8/8AjVHMg9vS/mX3o7miuG/4XJ4J/wCghff+Ce8/+NUf8Lk8E/8AQQvv/BPef/GqOZB7el/MvvR3NcBr2gQS/ErRLb+0Nbjg1G3vZrmGHW7yNGdPJ24VZQEA3twuBz06VN/wuTwT/wBBC+/8E95/8aqpN8UPh7caraalNdXzXdmkkcEn9k3w2LJt3jHlYOdi9R24o5kHt6X8y+9FqDxRrKWfiHULm50m00zSLmWygE0UzyOylQrswYlj82BGqlnbADAtgZkHxB16a0u7eGOzmvodYtNPjnuNNubJGSdFO5oJW8xSpJ74YAYxnhsvjz4Yz6Zf6fLLfG21Cdri4UaZfgtKSDvDCPKkFQQVIwQCMGqtp4r+FNkG8mXVHd7qK8eWez1KaR5o/uOzuhZiOnJIwAOgFHMg9vS/mX3o1rnxf4otrXUbTybSe80vVks7y+tNOmnjjge3SZZRapIZGOZFQqrnHLdARVbxL8SLrSNNivtK1HTdSigsoLudbbS7qcXKv1O9GKWoIBK+YXz3+7kwXfjb4Z3k93M1/rdvLeTrcXElnBqls0jrGsYJMSrxsRRjpxnGeao3mufB++gEEy6jHb/ZktWt7ey1KGKSNM7A8aIFfbngsCRxg8CjmQe3pfzL70drBqviTWtd1VdFl0q2sNNvBYtHd28kksreWjtJuWQBQPMGE2ktt+8u7K8x4Z1XXY/Dfg2XWdQj1BtQ1l7cOonidUEVyx3nziJeYxgMNoH8JIDBv/CdfCy61oa6lzqDXErJMWisdREM7KPkkaJU8t2AxhmUkbVweBgtfGXwtsmhNvPqgFvetfwI9lqTpDMyupZFZCFGJX+UAL82cZxRzIftqS+0vvNbSPGWvXUHhvWr1dNOkeI5xFDaQxOLi1DxvJEzSlyshwmGARcFuCQvLPCnjHxFqJ8MXeuDS/sniKKUJBaQSLJbyJGZAxkZyHVlR8rtXaSBubGTl6f4v+Felaot/YyakksbSPDG1hqLw27SEl2ihZDHETlhlFBwzDoTme08efDGxh0mK1lvkTR932Ef2XfnycoUPWP5vlYj5s9aOZC9vS/mX3o0/HWjxP4i8NypfavbnUtXW1ukttYuoI3jFrO20IkgVfmjQ5UAnHuc9vaWyWdnFbQtK0cShFM0zSuQPV3JZj7kk1wF98T/AIe6jNZTXt1fSPYXH2m2P9k3w8uTYybuIuflkYYORz9Kt/8AC5PBP/QQvv8AwT3n/wAao5kHt6X8y+9Hc0Vw3/C5PBP/AEEL7/wT3n/xqj/hcngn/oIX3/gnvP8A41RzIPb0v5l96O5orhv+FyeCf+ghff8AgnvP/jVH/C5PBP8A0EL7/wAE95/8ao5kHt6X8y+9Hc0Vw3/C5PBP/QQvv/BPef8Axqj/AIXJ4J/6CF9/4J7z/wCNUcyD29L+ZfejuaK4b/hcngn/AKCF9/4J7z/41R/wuTwT/wBBC+/8E95/8ao5kHt6X8y+9Hc0Vw3/AAuTwT/0EL7/AME95/8AGqP+FyeCf+ghff8AgnvP/jVHMg9vS/mX3o7miuG/4XJ4J/6CF9/4J7z/AONUf8Lk8E/9BC+/8E95/wDGqOZB7el/MvvR3NFcN/wuTwT/ANBC+/8ABPef/GqP+FyeCf8AoIX3/gnvP/jVHMg9vS/mX3o7mub8B/8AIu3X/Ya1X/04XFZX/C5PBP8A0EL7/wAE95/8aq98NbyDUfBjXtozNb3Oq6nNEzIyEq1/OQSrAEcHoQCKLplxqQn8LTOroooplnDfGT/kl19/192P/pZDXite1fGT/kl19/192P8A6WQ14rXHiN0fGcR/xYen6hRRRXMfLhRRRQAUUUUAFFFFAFrTvshvFW/hmljf5QIJhGwJI5yVbjrxj8a6fUvDug2njb+wTPeW0cciBriSRZTLuVSEVVQbSS2MkkDr7VyET+XMj4ztYHHrW9feJLfUfGb69dWEgUuki28dwAQyBQMsUOR8vTA69auLVtTtozpKnadr8y3XTW+3y8yp4n0uHRfE17p1q0jwwOFVpCCx+UHnAHrWVWt4k1iDXtZl1GG0ktZJzmVXnEgzgDj5Vx075rJpStfQwr8ntZez+G7t6BRRRUmIUUUUAFFFFABRRRQBn6B/yLemf9ekX/oArQrP0D/kW9M/69Iv/QBWhVS3ZrX/AIsvV/mFFFFSZBRRRQAUUUUAFS2zQLcKbyOSWEZ3JFIEY8dmKsBz7GoqKYJ2dzqvEGkaBotzYReXqRW6tI7tn+0RsVDZ+QLsHpjdnjOcHGDV8SaHZabpej39ibhBqMLSNBcOrtHgjByAMgg8cdqW78TW2oa3pd7faWJoLC1jt2tjNxNszgk7eBkg4wemO9VvEWtQ69fNeeTdR3DHkzXKyKq9lVRGu0D8f61o3GzselWnh3GfLbpbS3bXb5f0jHooorI80KKKKACiiigAooooAK9q+Df/ACS6x/6+77/0smrxWvavg3/yS6x/6+77/wBLJq6cPuz6jhz+LP0/U7miiiuw+zOU+JVnBqPgxbK7Vmt7nVdMhlVXZCVa/gBAZSCOD1BBFUf+FN+Cf+gfff8Ag4vP/jtavjz/AJF21/7DWlf+nC3rpKVkyJU4T+JJnDf8Kb8E/wDQPvv/AAcXn/x2j/hTfgn/AKB99/4OLz/47Xc0Ucq7Eewpfyr7kcN/wpvwT/0D77/wcXn/AMdo/wCFN+Cf+gfff+Di8/8AjtdzRRyrsHsKX8q+5HDf8Kb8E/8AQPvv/Bxef/HaP+FN+Cf+gfff+Di8/wDjtdzRRyrsHsKX8q+5HmWt/D/4b+Hha/2nZatvvJTDbxW17qVzJI4UuQEidm4VWPToKn0j4Z/DrXdPW+0u2vpoGZkJOq3yMjKSGVlaUMrAggqQCD1Fa3jbSr7VPEHhEafPe2Yh1GZ5b2ziR2t1+yTgE70dACSFyy/xcc4rh7zRNXGh6Va6jp3mw2Oq3q6u11pE+ow3srDMd2bZHDSK+7Py7lR2xtGz5TlXYPYUv5V9yOv/AOFN+Cf+gfff+Di8/wDjtV9Q+FHgXTtNub6fTdQaK2heZwmsXhYhQScZm68VzGpaPOPC2kaXLp66vZGG8aCe88M3kyQMZR5dulmXDQYUkJLIflVMA4YmopdKnutCh/4S7w7qurahN4Vs4tMf7FJM9vdCJ/OG4D9xLvKEuxUnAGflxRyrsHsKX8q+5G/c/D74d2fhJvEVzp+pJZLaC7I/ta8L7Su4KAJuWOQMZ61HN4I+GFqup/bYb61fSIUnvo5tVvg0KOu5TxKQ+cEZQsCwKjkYrS8UwzQfBzTEmRohbtpbXSONpWNJ4DICD0wAcj0Bql420rU9Z8XDWNO0mWa38NpG80Do6nWDvWUxIMgOIgodSQQZSFGMNRyrsHsKX8q+5D9O+G3gDU7+/s4tL1SKaxaMOs2q3iF1dA6uB52dpyV5AO5GGOOdH/hTfgn/AKB99/4OLz/47VnTZBc/F7VZ7YMIv7BshLuQqdxmuGQEHBBC54IzyK7CjlXYPYUv5V9yOG/4U34J/wCgfff+Di8/+O0f8Kb8E/8AQPvv/Bxef/Ha7mijlXYPYUv5V9yOG/4U34J/6B99/wCDi8/+O0f8Kb8E/wDQPvv/AAcXn/x2u5oo5V2D2FL+Vfcjhv8AhTfgn/oH33/g4vP/AI7R/wAKb8E/9A++/wDBxef/AB2u5oo5V2D2FL+VfcjynwN8KPCGo/Dzw7e3djeNcXOlW00rLqt2gLNEpJCrKAOT0AAFbv8AwpvwT/0D77/wcXn/AMdrV+HH/JLPCn/YFs//AEQldJRyofsaT+yvuOG/4U34J/6B99/4OLz/AOO0f8Kb8E/9A++/8HF5/wDHa7mijlXYXsKX8q+5HDf8Kb8E/wDQPvv/AAcXn/x2j/hTfgn/AKB99/4OLz/47Xc0Ucq7B7Cl/KvuRw3/AApvwT/0D77/AMHF5/8AHao618Nfh34e0mTUtVtNRitY2RGaPUr+VizuEUBEkLElmUYAPWvR65T4lWF3qfgiS109bg3D39gVNtHvkQLeQszgEEfKAW5BAAyeKOVdg9hS/lX3I57Rvh78N9fW4/s201MyWriOeC41DULeWIkBhujkdWGQQQSMHtWn/wAKb8E/9A++/wDBxef/AB2ub1zQteisvEli6XepTyX1lczajLaNL9vsRgNE0URRXKbXDRR7C64OCXO6uNLa08NvDBYR3+mXeqq8tq3hO9htNPAgPzDTy2+ZWYLwDsV234yuaOVdg9hS/lX3I6z/AIU34J/6B99/4OLz/wCO1naL8N/h/rujjU7XTdSigMk0Z87WLtSDHI0bE4mIxlCRz0xXP6NpcsOjeG08Z+H7zUNDtZdUQ2P9kSzLDKbj/RnNqA7BBF5qpncFDKM8g103h/TNQs/gRf6etndWV99m1JYbe5J85C0sxTJJOTyp3ZOc5yc5o5V2D2FL+VfcjO0fwZ8MtaiiaC11GBprJdQRLnVb1M2zMQsufN27TjPXIDLuAyKfpHgT4ca3eJBYafqhWaxiv4JZNTvY1nhkLAFQ0obI2jIIGNy+tWNe0GDxJ4J8HTadHO8cn2W3kezXO+wnRVmjbA/1bLtJ6Y2g5Fbd9GB8XvD8dooQQaLfecFGAIzLahB7cqcfQ0cq7B7Cl/KvuRV/4U34J/6B99/4OLz/AOO0f8Kb8E/9A++/8HF5/wDHa7mijlXYPYUv5V9yOG/4U34J/wCgfff+Di8/+O0f8Kb8E/8AQPvv/Bxef/Ha7mijlXYPYUv5V9yOG/4U34J/6B99/wCDi8/+O0f8Kb8E/wDQPvv/AAcXn/x2u5oo5V2D2FL+Vfcjhv8AhTfgn/oH33/g4vP/AI7V74a2cGneDGsrRWW3ttV1OGJWdnIVb+cAFmJJ4HUkk11dc34D/wCRduv+w1qv/pwuKLJFxpwh8KSOkoooplnN+PP+Rdtf+w1pX/pwt66Sub8ef8i7a/8AYa0r/wBOFvXSUAFFFFABRRRQAUUV5R8SZPsPilpNIluIku9PVfE7Wy58jTxIAJ8hhtkAMqhhk7PMOD5aigD1eisTS0it9altbHV7V7GGxt1t9IhRAbVRuAkyDuKsAAARgbOO9YV54k1mHxFJaxajtt1uNgj/AOEM1GXC5xj7Qsnln/fA298YoA7iiuE+JxubVNG1Dw5H5viqK6aPS4cZE6sp85JBkfutg3E54ZE74B0fCQig+HdnN4TVNRlmQzFr+U27XE7PmZpmVHKyby+4bThhjjsAdFqFha6rptzp+oQrPaXUTQzRNnDowwRx7Gpo0EUSxqWIVQoLMWPHqTyT7msbTrrxTLfImraPpFraHO+W21aWeReOMI1sgPOP4h+PSsXW9Ntofiv4Yv0Rzc3C3au7ys+FEK4CgkhBxkhQATycnmgDq7XTLSyvL26todk99IstxIWLF2CKg6ngBVHAwOpxkkm1XjU1navoV14jWKM+LE8Xm2ivAAbhcXwhEAbqI/s3GzO3aScHJJdp1nZrpHhrxBbRxjxNe+I/JurxcfaJh50izRO3UosSt8h4UIMDgUAex0UUUAFFFFABRRRQBzfw4/5JZ4U/7Atn/wCiErpK5v4cf8ks8Kf9gWz/APRCV0lABRRRQAUUUUAFFcT8SI7zSrG28V6EYV1XTCbdVmJEc8U7LGUcgE4D+XIPePHc1f0TRl8NXWkaRa68nlRWkzS2Msaedfyl1Z7ktncMMzEgDGZOegoA6eiuL8R+IdX0/W5bexv/ACYVVSE/4RDUL/GRz++hkCN9AOOhqz42tZJvCUet27qupaGP7TtpCjRBmRDvjIOWVXQuhHUbvUUAdXRXM+DdNuNP8J/bX8u51bVc6jdMzlUknkUEJuwSEUbUBwSFUcZrM8E6l4ovPEniWLV7OxW1h1XYxTVJZmtj9kt2EcaNCoKEtuzleXb5TjLAHX6Zplpo+mw2GnReTawArFHvLbBnOASScDPA6AcDgUR6ZaR6tNqaQ/6ZPCkDyliSY0LFVAJwBl2PGM55zgVyPjDTrZfH3grUdrtdNqrw7nlZgifY7gkKpO1ckDO0DdtXOdoxyniCzs7nRvGWvSxxv4msNYWCwuzjz7cgxfZ4o26qrbh8owG3tkHJoA9jorxqaztX0K68RrFGfFieLzbRXgANwuL4QiAN1Ef2bjZnbtJODkk+y0AFFFFABRRRQAVzfgP/AJF26/7DWq/+nC4rpK5vwH/yLt1/2GtV/wDThcUAdJRRRQBzfjz/AJF21/7DWlf+nC3rpK5vx5/yLtr/ANhrSv8A04W9dJQAUUUUAFFFFABUC2Fos1zKtrAJLoAXDiMZmAGBuP8AFgcc9qnooAqWmk6dp8gewsLW1cQpbhoYVQiJM7I+B91cnA6DPFW64XxD4o1LQdY8R2rTby2lxXejKyqAJmYwGPgZP71oDzn/AFn4VV0bxTrOoJ4b02W7B1OGW7GtOsKjzVtMxPxghN8rxNxjgnFAHfNZ2z30d49vC11HG0aTlAXVGILKG6gEqpI74HpRbWdtZrItpbxQCWRpZBEgXe7HLMcdSTyT1NcNZeNtX1bSPCuqvpg06LXbuNbeGO/Vy6tazS4lzCdq5jGNh3EEElcFDn6Z8R9W0vwPbal4qi0tbm91Kays5pdS8mFiss27znMKiJUWIgEBy+BwCaAPT6xr3wf4Z1PUjqOo+HdJu74lSbqexjklJUAL85XPAAx6Yqt4T8YWvinQbrUIvs+bO4kt5zZ3IuYSyANmOUAb1KspzgdcEZFcafF2q+HPhzoHje+u5buzugbnUbFypd1ufniMRbkGMsi7QcFNxxkCgD0P/hHtF/tz+2v7IsP7Vxj7f9mTz8bdv+sxu+7x16cUQ+HtFt9Zl1e30iwi1OYESXqWyLM4OOC4G49B37VyyXusaVe+DbrWdS+1TarNNZ3iQuBbhpYnnTYBwwQw7Fc8lWJJ5ruqACiiigAooooAKKKKAOb+HH/JLPCn/YFs/wD0QldJXN/Dj/klnhT/ALAtn/6ISukoAKKKKACiiigCK6tbe9t2gvIIriFiCY5UDKSCCODxwQD9RTXsrWS+jvXtoWuokaOOcxgyIrEFlDdQCVGR3wPSp65jXNcn0LxnpH2u5CaNfW1zFIrKuI7iNRKjZxu5jSYEZx8ooA6emTQxXNvJBcRpLDKpSSORQyupGCCD1BHavLdH8Z+JL3wybG+uBb+IbzVbWK2byUzHb3CrcfdwVJSITrkjkxZPrWvfeN9Un0TWr6LTPsNlp1+1gtzFeoZ5ZEuliJVGhZQhUnJY7sggDo9AHeRxpDEscSKkaKFVFGAoHQAdhTq84j8d6zo83jS+8RW1m+laNfLDb+XeYdS0UHlx8xKu1jLuLs3yliMEKDW54N8c2/iu91GxD6Y91YCKRn0rURfW7RybtuJNiYbKMCpXgbTk54ANPVPCHhrW7z7XrXh7StQudoTzruyjlfaOg3MpOOanl8O6LPrEWrTaPYSalCAsV49qhmjAyAFfG4dT0PeuK/trWv7P1zxPZ3U0tvp+uSpNY7dytZW/7mVUBBIbcskoK4LEBTxwFk1vWRoOleKr66mhgutcgMFl9xUsrhhbosgwCx/eLMd2SG4GAKAO0/4R7Rf7c/tr+yLD+1cY+3/Zk8/G3b/rMbvu8denFaNFFABRRRQAUUUUAFc34D/5F26/7DWq/wDpwuK6Sub8B/8AIu3X/Ya1X/04XFAHSUUUUAct8RbmCz8JRXV5NHBbw6tpkkssrhUjUX8BLMTwAAMkmpf+Fj+CP+hy8P8A/g0g/wDiqPHn/Iu2v/Ya0r/04W9dJQBzf/Cx/BH/AEOXh/8A8GkH/wAVR/wsfwR/0OXh/wD8GkH/AMVXSUUAc3/wsfwR/wBDl4f/APBpB/8AFUf8LH8Ef9Dl4f8A/BpB/wDFV0lFAHN/8LH8Ef8AQ5eH/wDwaQf/ABVH/Cx/BH/Q5eH/APwaQf8AxVb17eQafYXF7eSCO3tomllc9FVRkn8hXn2hfFv+1PBOv61qOhXGm3mjQJctp4drh5IpYhJC+VTIDZIJxhdrEnAOACbXtZ+G3iLVtI1C/wDGmjLNpM/nQiLV4FWT5lba4zyu+ON8ccoppun6r8NtN8Taxrlv410c3OrqoljfV7cxw4UBjGueC2FLZJyQOlZnhX4wTX9x4ZsNasJ5rnXbVrl57TSbuGO3wqEIqsr+cuWOZVbaBgnArtLTx94avtYGmWuol7h3kjjb7PKIZnjGZFjlK+W7LzkKxPB9DQBzltqnw3tdI8OadH440kw+HJI5LRm1e23OUheEb+cH5ZCeMc4+lVVn+HkdibWD4j2MSxX739g66rZ7tPld3Z/KJXlT5rgiTfw2K6Oz+J/hLUNNj1C01OSSyluIbWO5+xTiOSWU4RFYpgnPBx908NitDxH4kXwzJp9xfwKNLnn8i6vmmCLZluI2YEY2M3ylsjBK9cnABk6f438G2mnta33xE0nVWYndNd6lZq5B/hxEEXH4ZrmvCuq+CLbSNDh13xfojSeH0mtLSBtXtWhdQTHHcYDElzEMDJ43vxnmvQfD2tvrmhJq0tm1lBOzvbq7Zd4MnZIwwNu5Ru284BGTnIFjTNa0/WNDg1nT7lZNPuIfPjnYFAUxncQwBAx64oA83XV/Bdlq3hmwsPGWitoujT3N6rTaxbHymKNHDAoDAlVEz4znAjAJNdl/wsfwR/0OXh//AMGkH/xVY9t8TludM1i8XRpR9kube20+IzANfG4KiBuR+7Dl1POcKcnn5akm+IF7Y3lxo+o6JCviATW8draQXxkguBPv2v5zRqVA8qXd8hICcBsigDU/4WP4I/6HLw//AODSD/4qj/hY/gj/AKHLw/8A+DSD/wCKqfw54hn1e41Kw1KxSw1LS5liuIYp/OjIdA6OjlVJBU91BBBGO53aAOb/AOFj+CP+hy8P/wDg0g/+Ko/4WP4I/wChy8P/APg0g/8Aiq6SigDm/wDhY/gj/ocvD/8A4NIP/iqP+Fj+CP8AocvD/wD4NIP/AIqukooA878A+PvB9n8NvDVreeLNDguIdJtY5YpdShV42EKgqwLZBBGCDXQf8LH8Ef8AQ5eH/wDwaQf/ABVHw4/5JZ4U/wCwLZ/+iErpKAOb/wCFj+CP+hy8P/8Ag0g/+Ko/4WP4I/6HLw//AODSD/4qukooA5v/AIWP4I/6HLw//wCDSD/4qj/hY/gj/ocvD/8A4NIP/iq6SigDm/8AhY/gj/ocvD//AINIP/iqxPFuufDjxloTaVqnjbSIoWcP5lrrECSDGQQDuPDKWU+zHp1rv6KAPPJdW+GkvjW28T/8JnoqXltbfZ0iTVrcQkDeFcrnJZVkkUHPAcjFNk1T4byaDf6Q3jjSfs99fPfSMNXtt4kaYTEA5xt3DHQnHfvWvrXjv+xLzVrK401nvrZIX02BJedS80hFCnb8pEvyt97aMMeDit+/1iz0mOyOqyi3e9uI7WIKrOGmfOFBA9jycD6UAcJd33w4vbvWpJvH2mC31rY9zaJq1qsazIsapOjffVwIkx823IztzzWppHjPwnpqyi8+J2n6v5mNpvdRsV8vGfu+Sqdc989BjHNdBrt/rFjHGdE0u2vjtd5nu777LFEqgcbgjksc8fLjAOWHAODbeO77WbbSB4b0NLm81HTF1N4r28+zpBE2Ao3qjlmYk4wuCATkcZAMbRvEvg63g8T6ReeLdBj0/UL2ae3nj1a2yyXCBpBjeSGEjSdRjBX3ANd8S+DrnTPDui2Hi7QZLKzvrWS4mk1e3ykVv+8XI3fMWeONcAdyeMVo2/xHu9btRN4V0Fb4waet9fJd3v2cwli6rCmEcPJmKTOdqjC/N83HYaPqltrmh2Oq2BY2t9bpcQlhg7HUMMjscHpQBj/8LH8Ef9Dl4f8A/BpB/wDFUf8ACx/BH/Q5eH//AAaQf/FV0lFAHN/8LH8Ef9Dl4f8A/BpB/wDFUf8ACx/BH/Q5eH//AAaQf/FV0lFAHN/8LH8Ef9Dl4f8A/BpB/wDFUf8ACx/BH/Q5eH//AAaQf/FV0lFAHN/8LH8Ef9Dl4f8A/BpB/wDFVF8OrmC88JS3VnNHPbzatqckUsThkkU385DKRwQQcgiuprm/Af8AyLt1/wBhrVf/AE4XFAHSUUUUAc348/5F21/7DWlf+nC3rpKo6xo9lr2mPYamkj27vHJ+6meF1ZHV0ZXQhlIZVIII6Vk/8IHpH/P54g/8KPUP/j9AHSUVzf8Awgekf8/niD/wo9Q/+P0f8IHpH/P54g/8KPUP/j9AHSUVzf8Awgekf8/niD/wo9Q/+P0f8IHpH/P54g/8KPUP/j9ADfiF4e1LxX4OuNC0q5itft8kcV1NIxDJb7wZdmFOWKgqAcDk8iuV1L4Y62muahd6XrpvoNV0G40m7GpeXGynYfIZRBCqkAkgkjIB4zwK6z/hA9I/5/PEH/hR6h/8fo/4QPSP+fzxB/4Ueof/AB+gDm7DwFrmk3PgK8tX0+4n8O6a2n3sUk7xq4eNFLxsI2JwUJwQM+oqj4T+FNz4eksrK+tLfUbTTbqeezvn127Vot5cq32PaYQ3z7ThhnJPXOey/wCED0j/AJ/PEH/hR6h/8fo/4QPSP+fzxB/4Ueof/H6AMjwX4X8TeDPhhoWgWUuky6jZzYu3laVoTC0zO+wgA79rDGRjPWtrxZ4en8UpZ6VcPGuhySF9TjDsslwqjKRDA4UtyxyDhduDuJDP+ED0j/n88Qf+FHqH/wAfo/4QPSP+fzxB/wCFHqH/AMfoAueGrDVNK0BdP1S6jvJbUtFb3O4lpYR/qzJkD59uA2M5Iz3wK97oV74m8CnR/E9ytreXcKpeyaSfkzkFlTzVbKHG0hgcgkVH/wAIHpH/AD+eIP8Awo9Q/wDj9H/CB6R/z+eIP/Cj1D/4/QBzc/w31rz9buV8Ry3k09zZXlgL1IgrS2xjdfN8qFNoJj2ZXICtkqSKmvfBviDVNabxPdf2bBrMFxaNaWSXMkluIoPNyjSmNWBf7RLyIztwvDc53v8AhA9I/wCfzxB/4Ueof/H6P+ED0j/n88Qf+FHqH/x+gB/hnRtQs77V9W1v7OmoarOjtBaStJFBHGgREDsqljwzE7V5bGOMnoa5v/hA9I/5/PEH/hR6h/8AH6P+ED0j/n88Qf8AhR6h/wDH6AOkorm/+ED0j/n88Qf+FHqH/wAfo/4QPSP+fzxB/wCFHqH/AMfoA6Siub/4QPSP+fzxB/4Ueof/AB+j/hA9I/5/PEH/AIUeof8Ax+gA+HH/ACSzwp/2BbP/ANEJXSVy1t8OtBs7WK1s5dcgt4UEcUUXiG/VI1AwFUCbAAAwAKl/4QPSP+fzxB/4Ueof/H6AOkorm/8AhA9I/wCfzxB/4Ueof/H6P+ED0j/n88Qf+FHqH/x+gDpKK5v/AIQPSP8An88Qf+FHqH/x+j/hA9I/5/PEH/hR6h/8foA6Siub/wCED0j/AJ/PEH/hR6h/8fo/4QPSP+fzxB/4Ueof/H6AJNa8OHVfF3hzV9lsy6RJcMxlGXAkiKDZwe+CeR0rT1L+1P8ARP7H+x/8fKfavtW7/Uc79m3+PpjPHXNZH/CB6R/z+eIP/Cj1D/4/R/wgekf8/niD/wAKPUP/AI/QAvi3SLrWEtYU0HQNetELNJa6yxQJJwFdG8qUcAuCNoPzDkYIOJp/hTxR4ek02+0+5sdZv49NaxuxqN1NED+8MiMsm2RmC7mXDDJGDkEYra/4QPSP+fzxB/4Ueof/AB+j/hA9I/5/PEH/AIUeof8Ax+gDndM8D+IvCiSr4cm02+a/02O2u5b2WSExXCtK3nIFR96kzt8hK42j5uTjttA0eLw/4b07R7Z2kisLWO2R26sEULk/XFZf/CB6R/z+eIP/AAo9Q/8Aj9H/AAgekf8AP54g/wDCj1D/AOP0AdJRXN/8IHpH/P54g/8ACj1D/wCP0f8ACB6R/wA/niD/AMKPUP8A4/QB0lFc3/wgekf8/niD/wAKPUP/AI/R/wAIHpH/AD+eIP8Awo9Q/wDj9AHSUVzf/CB6R/z+eIP/AAo9Q/8Aj9H/AAgekf8AP54g/wDCj1D/AOP0AdJXN+A/+Rduv+w1qv8A6cLij/hA9I/5/PEH/hR6h/8AH61tH0ey0HTEsNMSRLdHkk/ezPM7M7s7szuSzEszEkk9aAL1FFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAf/9k=)

If the transaction fails after completion of **T1** but before completion of **T2**.( say, after **write(X)** but before **write(Y)**), then the amount has been deducted from **X** but not added to **Y**. This results in an inconsistent database state. Therefore, the transaction must be executed in its entirety in order to ensure the correctness of the database state.

### Consistency:

This means that integrity constraints must be maintained so that the database is consistent before and after the transaction. It refers to the correctness of a database. Referring to the example above,   
The total amount before and after the transaction must be maintained.   
Total **before T** occurs = **500 + 200 = 700**.   
Total **after T occurs** = **400 + 300 = 700**.   
Therefore, the database is **consistent**. Inconsistency occurs in case **T1** completes but **T2** fails. As a result, T is incomplete.

### Isolation:

This property ensures that multiple transactions can occur concurrently without leading to the inconsistency of the database state. Transactions occur independently without interference. Changes occurring in a particular transaction will not be visible to any other transaction until that particular change in that transaction is written to memory or has been committed. This property ensures that the execution of transactions concurrently will result in a state that is equivalent to a state achieved these were executed serially in some order.   
Let **X**= 500, **Y** = 500.   
Consider two transactions **T** and **T”.**

Table

Description automatically generated

Suppose **T** has been executed till **Read (Y)** and then **T’’** starts. As a result, interleaving of operations takes place due to which **T’’** reads the correct value of **X** but the incorrect value of **Y** and sum computed by   
**T’’: (X+Y = 50, 000+500=50, 500)**   
is thus not consistent with the sum at end of the transaction:   
**T: (X+Y = 50, 000 + 450 = 50, 450)**.   
This results in database inconsistency, due to a loss of 50 units. Hence, transactions must take place in isolation and changes should be visible only after they have been made to the main memory.

### Durability:

This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even if a system failure occurs. These updates now become permanent and are stored in non-volatile memory. The effects of the transaction, thus, are never lost.

**Some important points:**

| **Property** | **Responsibility for maintaining properties** |
| --- | --- |
| Atomicity | Transaction Manager |
| Consistency | Application programmer |
| Isolation | Concurrency Control Manager |
| Durability | Recovery Manager |

The **ACID** properties, in totality, provide a mechanism to ensure the correctness and consistency of a database in a way such that each transaction is a group of operations that acts as a single unit, produces consistent results, acts in isolation from other operations, and updates that it makes are durably stored.

## How are Anomalies Caused in DBMS?

What exactly does “bad construction” imply? When the DB (database) designer constructs the database, he should identify the entities that rely on one other for existence, such as hotel rooms and the hotel, and then reduce the probability that one might ever exist independently of the other.

A database anomaly is a fault in a database that usually emerges as a result of shoddy planning and storing everything in a flat database. In most cases, this is removed through the normalization procedure, which involves the joining and splitting of tables. The purpose of the normalization process is to minimise the negative impacts of generating tables that would generate anomalies in the DB.

## Example

Consider a manufacturing firm that keeps worker information in a table called employee, which has four columns: w\_id for the employee’s id, w\_name for the employee’s name, w\_address for the employee’s address, and w\_dept for the employee’s department. The table will look like this at some point:

|  |  |  |  |
| --- | --- | --- | --- |
| **w\_id** | **w\_name** | **w\_address** | **w\_dept** |
| 201 | David | Delhi | F001 |
| 201 | David | Delhi | F002 |
| 223 | Mike | Agra | F890 |
| 266 | Berry | Chennai | F900 |
| 266 | Berry | Chennai | F004 |

The table above has not been normalized. We’ll look at the issues that arise when the table isn’t normalized.

## Type of Anomalies in DBMS

Various types of anomalies can occur in a DB. For instance, redundancy anomalies are a very significant issue for tests if you’re a student, and for job interviews if you’re searching for a job. But these can be easily identified and fixed. The following are actually the ones about which we should be worried:

1. Update

2. Insert

3. Delete

Anomalies in databases can be, thus, divided into three major categories:

Update Anomaly

Employee David has two rows in the table given above since he works in two different departments. If we want to change David’s address, we must do so in two rows, else the data would become inconsistent.

If the proper address is updated in one of the departments but not in another, David will have two different addresses in the database, which is incorrect and leads to inconsistent data.

### Insert Anomaly

If a new worker joins the firm and is currently unassigned to any department, we will be unable to put the data into the table because the w\_dept field does not allow nulls.

### Delete Anomaly

If the corporation closes the department F890 at some point in the future, deleting the rows with w\_dept as F890 will also erase the information of employee Mike, who is solely assigned to this department.

# Indexing in Databases | Set 1

* Difficulty Level : [Medium](https://www.geeksforgeeks.org/medium/)
* Last Updated : 15 Sep, 2021

Indexing is a way to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed. It is a data structure technique which is used to quickly locate and access the data in a database.

Indexes are created using a few database columns.

* The first column is the **Search key** that contains a copy of the primary key or candidate key of the table. These values are stored in sorted order so that the corresponding data can be accessed quickly.   
  *Note: The data may or may not be stored in sorted order.*
* The second column is the **Data Reference** or **Pointer** which contains a set of pointers holding the address of the disk block where that particular key value can be found.

Graphical user interface, text, application

Description automatically generated

The indexing has various attributes:

* **Access Types**: This refers to the type of access such as value based search, range access, etc.
* **Access Time**: It refers to the time needed to find particular data element or set of elements.
* **Insertion Time**: It refers to the time taken to find the appropriate space and insert a new data.
* **Deletion Time**: Time taken to find an item and delete it as well as update the index structure.
* **Space Overhead**: It refers to the additional space required by the index.

In general, there are two types of file organization mechanism which are followed by the indexing methods to store the data:

**1. Sequential File Organization or Ordered Index File:** In this, the indices are based on a sorted ordering of the values. These are generally fast and a more traditional type of storing mechanism. These Ordered or Sequential file organization might store the data in a dense or sparse format:

**(i) Dense Index:**

* For every search key value in the data file, there is an index record.
* This record contains the search key and also a reference to the first data record with that search key value.

[Table

Description automatically generated](https://media.geeksforgeeks.org/wp-content/cdn-uploads/20190812183521/Dense-Index.jpg)

**(ii) Sparse Index:**

* The index record appears only for a few items in the data file. Each item points to a block as shown.
* To locate a record, we find the index record with the largest search key value less than or equal to the search key value we are looking for.
* We start at that record pointed to by the index record, and proceed along with the pointers in the file (that is, sequentially) until we find the desired record.

[Diagram

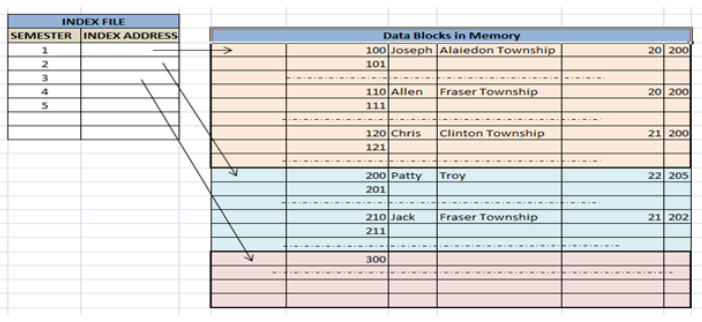
Description automatically generated with medium confidence](https://media.geeksforgeeks.org/wp-content/cdn-uploads/20190812183518/Sparse-Index.jpg)

**2. Hash File organization:** Indices are based on the values being distributed uniformly across a range of buckets. The buckets to which a value is assigned is determined by a function called a hash function.

There are primarily three methods of indexing:

* Clustered Indexing
* Non-Clustered or Secondary Indexing
* Multilevel Indexing

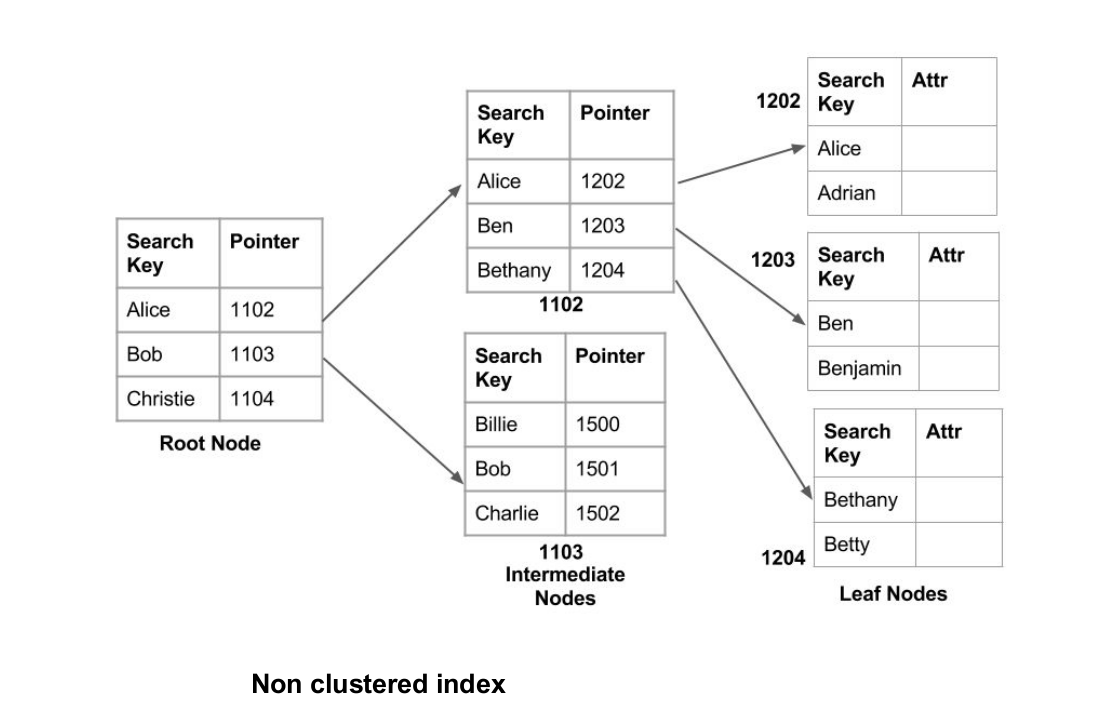
**1. Clustered Indexing**   
When more than two records are stored in the same file these types of storing known as cluster indexing. By using the cluster indexing we can reduce the cost of searching reason being multiple records related to the same thing are stored at one place and it also gives the frequent joining of more than two tables (records).   
Clustering index is defined on an ordered data file. The data file is ordered on a non-key field. In some cases, the index is created on non-primary key columns which may not be unique for each record. In such cases, in order to identify the records faster, we will group two or more columns together to get the unique values and create index out of them. This method is known as the clustering index. Basically, records with similar characteristics are grouped together and indexes are created for these groups.   
For example, students studying in each semester are grouped together. i.e. 1st Semester students, 2nd semester students, 3rd semester students etc. are grouped.



**Clustered index sorted according to first name (Search key)**

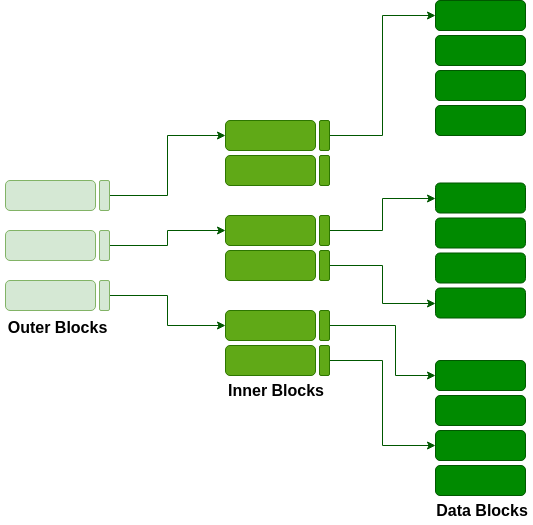
***Primary Indexing:***   
This is a type of Clustered Indexing wherein the data is sorted according to the search key and the primary key of the database table is used to create the index. It is a default format of indexing where it induces sequential file organization. As primary keys are unique and are stored in a sorted manner, the performance of the searching operation is quite efficient.

**2. Non-clustered or Secondary Indexing**   
A non clustered index just tells us where the data lies, i.e. it gives us a list of virtual pointers or references to the location where the data is actually stored. Data is not physically stored in the order of the index. Instead, data is present in leaf nodes. For eg. the contents page of a book. Each entry gives us the page number or location of the information stored. The actual data here(information on each page of the book) is not organized but we have an ordered reference(contents page) to where the data points actually lie. We can have only dense ordering in the non-clustered index as sparse ordering is not possible because data is not physically organized accordingly.   
It requires more time as compared to the clustered index because some amount of extra work is done in order to extract the data by further following the pointer. In the case of a clustered index, data is directly present in front of the index.



**3. Multilevel Indexing**

With the growth of the size of the database, indices also grow. As the index is stored in the main memory, a single-level index might become too large a size to store with multiple disk accesses. The multilevel indexing segregates the main block into various smaller blocks so that the same can stored in a single block. The outer blocks are divided into inner blocks which in turn are pointed to the data blocks. This can be easily stored in the main memory with fewer overheads.



# SQL | Join (Inner, Left, Right and Full Joins)

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 15 Jul, 2022

**SQL Join** statement is used to combine data or rows from two or more tables based on a common field between them. Different types of Joins are as follows:

* INNER JOIN
* LEFT JOIN
* RIGHT JOIN
* FULL JOIN

Consider the two tables below:

Pause

Unmute

Loaded: 6.19%

Remaining Time-21:23

1x

Playback Rate

Captions

Picture-in-PictureFullscreen

**Student**



**StudentCourse**



The simplest Join is INNER JOIN.

### ****A. INNER JOIN****

The INNER JOIN keyword selects all rows from both the tables as long as the condition is satisfied. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be the same.

**Syntax**:

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

INNER JOIN table2

ON table1.matching\_column = table2.matching\_column;

**table1**: First table.

**table2**: Second table

**matching\_column**: Column common to both the tables.

***Note****: We can also write JOIN instead of INNER JOIN. JOIN is same as INNER JOIN.*

Diagram, venn diagram

Description automatically generated

**Example Queries(INNER JOIN)**

This query will show the names and age of students enrolled in different courses.

SELECT StudentCourse.COURSE\_ID, Student.NAME, Student.AGE FROM Student

INNER JOIN StudentCourse

ON Student.ROLL\_NO = StudentCourse.ROLL\_NO;

**Output**:



### ****B. LEFT JOIN****

This join returns all the rows of the table on the left side of the join and matches rows for the table on the right side of the join. For the rows for which there is no matching row on the right side, the result-set will contain *null*. LEFT JOIN is also known as LEFT OUTER JOIN.

**Syntax:**

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

LEFT JOIN table2

ON table1.matching\_column = table2.matching\_column;

table1: First table.

table2: Second table

matching\_column: Column common to both the tables.

***Note****: We can also use LEFT OUTER JOIN instead of LEFT JOIN, both are the same.*

Diagram, venn diagram

Description automatically generated

**Example Queries(LEFT JOIN)**:

SELECT Student.NAME,StudentCourse.COURSE\_ID

FROM Student

LEFT JOIN StudentCourse

ON StudentCourse.ROLL\_NO = Student.ROLL\_NO;

**Output**:



### ****C. RIGHT JOIN****

RIGHT JOIN is similar to LEFT JOIN. This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of the join. For the rows for which there is no matching row on the left side, the result-set will contain *null*. RIGHT JOIN is also known as RIGHT OUTER JOIN.

**Syntax:**

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

RIGHT JOIN table2

ON table1.matching\_column = table2.matching\_column;

table1: First table.

table2: Second table

matching\_column: Column common to both the tables.

***Note****: We can also use RIGHT OUTER JOIN instead of RIGHT JOIN, both are the same.*



**Example Queries(RIGHT JOIN)**:

SELECT Student.NAME,StudentCourse.COURSE\_ID

FROM Student

RIGHT JOIN StudentCourse

ON StudentCourse.ROLL\_NO = Student.ROLL\_NO;

**Output:**



### ****D. FULL JOIN****

FULL JOIN creates the result-set by combining results of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both tables. For the rows for which there is no matching, the result-set will contain *NULL* values.

Venn diagram

Description automatically generated

**Syntax:**

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

FULL JOIN table2

ON table1.matching\_column = table2.matching\_column;

table1: First table.

table2: Second table

matching\_column: Column common to both the tables.

**Example Queries(FULL JOIN)**:

SELECT Student.NAME,StudentCourse.COURSE\_ID

FROM Student

FULL JOIN StudentCourse

ON StudentCourse.ROLL\_NO = Student.ROLL\_NO;

**Output:**

| NAME | COURSE\_ID |
| --- | --- |
| HARSH | 1 |
| PRATIK | 2 |
| RIYANKA | 2 |
| DEEP | 3 |
| SAPTARHI | 1 |
| DHANRAJ | NULL |
| ROHIT | NULL |
| NIRAJ | NULL |
| NULL | 4 |
| NULL | 5 |
| NULL | 4 |