Object - Oriented Programming (OOPs)

Q1). What is object - oriented programming (OOPs)?

Ans) OOPs is a programming paradigm that organizes code into objects. Objects are self-contained units that contain both data and behavior. OOPs promotes code reuse, modularity and encapsulation.

Eg: A car can be an object, because it has both attributes (or) data (like: brand, model, colour, Year....) and methods (or) functions (like: accelerate, brake, turn...).

Q2). What is encapsulation?

Ans) Encapsulation is the concept of bundling data (attributes) and functions (methods) that operate on that data into a single unit known as class. It hides the internal details of an object and Provides controlled access through methods.

Eg: Encapsulation in a car for example hides the internal details of a car (object) like car's engine, Transmission,... and provides controlled access to user through methods like steering wheel, gear box for shifting gear, Accelator, brake, etc....

Q3). What is Inheritance?

Ans) Inheritance is a mechanism where a new class (subclass or derived) inherits properties and behaviours from an existing class (superclass or base class). It promotes code reuse and supports hierarchical relationships.

Eg:- we can create sport car class that inherits from the car class, it will automatically have all of the properities and behaviors of the car class like: engine, gears box, brake, streeing, etc.... and also have the additional properities and behaviors that are specific like top speed, spoiler,

Q4). What is polymorphism? ("Poly" → many ; "Morphs" → forms)

Ans) Polymorphism allows objects of different classes to be treated as instances of a common super class. In other words, polymorphism allows us to define one interface and have multiple implementations.

Eg.:- creating a base class 'Car' that defines the common behavior and attributes shared by all types of cars (like: acceleration, brake, teaning), while subclasses (sedan, suv, Truck) implement these methods differently for each car type (like acc() may be more for sedan than truck, etc)

- Polymorphism is of two types :- i) compile time & ii) Run time.

Q5). What is method overloading?

Ans) Method overloading is also known as static (or) compile time polymorphism is the ability to define multiple methods in a class with the same name but different parameter lists.

Eg.:- In a 'Car' class, it could have a method called "drive()" which could be overloaded to accept different parameters, such as the distance to drive; speed to drive at, etc....

Q6). What is method overriding?

Ans) Method overriding is the process by which a subclass provides a specific implementation for a method that is already defined in its superclass.

Eg:- Consider a 'Car' class and a 'sportCar' class. Here the 'sportCar' class inherits from 'Car' class and it could override the drive() method from the 'Car' class to provide a specific implemenation for 'sport cars'.

Q7) What is a constructor?

Ans) A constructor is a special method that is automatically called when an object is created. It initializes the object's attributes and prepare them for use in a consistent and reliable way. Constructor often have the same name as the class.

Eg.:- The __init__() method is the constructor for the car class & it is called when the car object is created. The constructor initializes the car's make, model, colour and speed attributes.

Q8) What is an abstract class?

Ans) An abstract class is a class that cannot be instantiated on its own. Instead, it serves as a blueprint or template for its concrete subclasses. Abstract class may contain abstract methods (methods without body) that must be implemented by its concrete subclasses to provide common interface.

Eg.:- The "vehicle class" is a abstract class and "car class" is a concrete subclass of vehicle class and it implements all the abstract methods of vehicle like acc, brake, turn.

Q9) What is an interface?

Ans) An interface is a contract that defines a set of methods that a class must implement. It allows multiple classes to adhere to the same interface, promoting a form of multiple inheritance. Interface only declare method signatures, not implementations.

Eg.:- The "drivable" interface defines three methods: start(), drive(), stop(). The car and truck classes implement the "Drivable" interface by implementing the three methods.

Q10) What is static method?

Ans) A static method belongs to the class itself, not to instances of the class. It can be called using the class name and it's often used for utility functions or operations that don't require instance-specific data.

Eg:- The get_no_of_wheels() method is a static method. It can be called using the class name(car) without having to create an instance of the car class.

Q11) What is a final Class?

Ans) A final class is a class that cannot be subclassed. It prevents other classes from extending it and inheriting its behavior.

Final classes can also have final methods. Final methods cannot be overridden by subclasses. This is useful for methods that implement core functionality that should not be changed. (Eg: String class in Java)

Q12) What is Composition?

Ans) Composition is a design principle where a class contains objects of other classes as part of its attributes. It allows creating complex structures by combining simpler classes.

Eg.:- The "car" class contains objects of other classes, such as an "engine" class, a "transmission" class, and a "wheel" class. The "car" class uses these objects to implement its own functionality. For example, the start() method calls the start() method on the "engine" object.

Q13) What is super keyword?

Ans) The super keyword is used to refer to the parent class or superclass

It can be used to call methods and constructors from the superclass.

Q14) What is method hiding?

Ans) Allow us to hide methods from other classes and prevent them from modifying the behavior of our class, or for the encapsulating the implementation of our class's methods.

Q15) What is constructor chaining?

Ans) Constructor chaining is the process of calling one constructor from another within the same class or between a superclass and a subclass.

Q16) What is a shallow copy and a deep copy?

Ans) A shallow copy copies the references of objects contained within an object, while a deep copy creates new instances of the objects contained.

A deep copy results in a completely independent copy of the original object and its contained objects.

Q17) What is virtual method?

Ans) A method that is declared in a base class and can be overriden by its subclasses. The actual implementation invoked is determined by the runtime type of the object, supporting polymorphism.

Q18) Purpose of the final keyword?

Ans) final keyword ~~class~~ can be applied to classes, methods and variables. A final class cannot be subclassed, a final method cannot be overridden, and a final variable cannot be reassigned after its initial assignment.

Q19) What is the SOLID Principle?

Ans) The SOLID Principle is an acronym for a set of design Principles that promote maintainable and scalable code:

S - Single Responsibility Principle

O - open closed principle

L - Liskov Substitution Principle

I - Interface Segregation Principle

D - Dependency Inversion Principle

Q20) What is the difference between an abstract class and an interface?

Ans) An abstract class can have both abstract and concrete methods, while an interface only has method signatures (no method implementation)

A class can ~~inhes~~ implement multiple interfaces, but it can inherit from only one abstract class.

Q21) How is encapsulation related to data hiding?

Ans) Encapsulation is the process of bundling the and methods together into a single unit. Data hiding is the process of making the internal data of an object inaccessible to the outside world, except through well-defined methods.

Q22) What is the difference between composition and inheritance?

Ans) Inheritance creates a relationship between a subclass and a superclass, allowing the subclass to inherit properties and methods

Composition involves creating an object within another object to achieve complex behavior without the tight coupling of inheritance