**Important interview questions and answers for Object Oriented programming?**

1. What is object-oriented programming?

Answer: Object-oriented programming (OOP) is a programming paradigm that organizes software design around objects that represent real-world entities. It focuses on the concepts of encapsulation, inheritance, and polymorphism.

2. What are the four fundamental principles of object-oriented programming?

Answer:

a. Encapsulation: It is the mechanism of hiding data and methods within a class, allowing controlled access through public interfaces.

b. Inheritance: It allows creating new classes based on existing classes, inheriting their properties and behaviors.

c. Polymorphism: It enables objects of different classes to be treated as objects of a common super class, providing flexibility and reusability.

d. Abstraction: It refers to simplifying complex systems by breaking them down into smaller, more manageable objects.

3. What is a class?

Answer: A class is a blueprint or template for creating objects. It defines the attributes (data) and behaviors (methods) that objects of that class will possess.

4. What is an object?

Answer: An object is an instance of a class. It represents a specific entity based on the class's blueprint and has its own unique state and behavior.

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

Answer: A class is a blueprint or template for creating objects, while an object is an instance of a class. A class defines the properties and behaviors that objects will have, whereas objects are specific instances that possess the defined properties and behaviors.

6. What is inheritance?

Answer: Inheritance is a mechanism in OOP that allows a class (derived or child class) to inherit properties and behaviors from another class (base or parent class). It promotes code reuse and supports the concept of hierarchical relationships between classes.

7. What is encapsulation?

Answer: Encapsulation is the concept of bundling data and methods together within a class and providing controlled access to them. It allows for data hiding, preventing direct access to the internal implementation details of a class.

8. What is polymorphism?

Answer: Polymorphism is the ability of objects of different classes to respond to the same message or method invocation. It allows objects to be treated as objects of a common superclass, while each class can have its own implementation of the method.

9. What is method overloading?

Answer: Method overloading is the ability to define multiple methods in a class with the same name but different parameters. The compiler determines which method to execute based on the number, type, and order of the arguments provided.

10. What is method overriding?

Answer: Method overriding is the process of defining a method in a subclass that already exists in its super class. The overridden method in the subclass provides a different implementation while having the same name, return type, and parameters.

11. What is a constructor?

Answer: A constructor is a special method in a class that is used to initialize objects of that class. It is called automatically when an object is created and is used to set initial values to the object's attributes.

12. What is method overloading?

Answer: Method overloading is the ability to define multiple methods in a class with the same name but different parameters. The compiler determines which method to execute based on the number, type, and order of the arguments provided.

13. What is method overriding?

Answer: Method overriding is the process of defining a method in a subclass that already exists in its superclass. The overridden method in the subclass provides a different implementation while having the same name, return type, and parameters.

14. What is the difference between method overloading and method overriding?

Answer: Method overloading occurs within the same class and involves defining multiple methods with the same name but different parameters. Method overriding occurs in a subclass and involves providing a different implementation of a method that already exists in the superclass.

15. What is the difference between composition and inheritance?

Answer: Composition and inheritance are two ways of achieving code reuse and creating relationships between classes. Composition involves creating objects of other classes as attributes within a class. Inheritance involves creating a new class based on an existing class, inheriting its properties and behaviors.

16. What is a static method?

Answer: A static method is a method that belongs to the class rather than an instance of the class. It can be called directly on the class itself, without the need to create an object of the class. Static methods are commonly used for utility functions or operations that do not require access to instance-specific data.

17. What is the difference between an abstract class and an interface?

Answer: An abstract class is a class that cannot be instantiated and is intended to be subclassed. It can contain both concrete and abstract methods. An interface, on the other hand, is a collection of abstract methods that define a contract for the classes implementing it. A class can implement multiple interfaces, but it can only inherit from a single class.

18. What is the diamond problem in inheritance?

Answer: The diamond problem is a name given to the ambiguity that arises when a class inherits from two or more classes that have a common base class. It occurs when there is a method defined in the base class, and the derived class does not override it. In such cases, it's unclear which version of the method should be called.

19. What is a virtual function?

Answer: A virtual function is a function declared in a base class that is intended to be overridden by derived classes. It allows polymorphic behavior, where the appropriate version of the function is called based on the actual type of the object at runtime.

20. What is the SOLID principle in OOP?

Answer: The SOLID principle is a set of five design principles that promote software design that is modular, maintainable, and flexible:

- Single Responsibility Principle (SRP)

- Open-Closed Principle (OCP)

- Liskov Substitution Principle (LSP)

- Interface Segregation Principle (ISP)

- Dependency Inversion Principle (DIP)

How can we implement SOLID principle in OOP c++?

Implementing the SOLID principles in C++ involves following certain guidelines and design patterns. Here's how you can apply each principle:

1. Single Responsibility Principle (SRP):

- Define classes with a single responsibility or purpose.

- Split large classes into smaller, more focused classes.

- Avoid creating classes that handle multiple unrelated tasks.

2. Open-Closed Principle (OCP):

- Design classes to be open for extension but closed for modification.

- Use inheritance or interfaces to allow for easy extension without modifying existing code.

- Apply the "Dependency Inversion Principle" to decouple high-level modules from low-level implementation details.

3. Liskov Substitution Principle (LSP):

- Ensure that derived classes can be used as substitutes for the base class without altering the correctness of the program.

- Follow the "is-a" relationship between base and derived classes.

- Override methods in derived classes while maintaining the same preconditions, postconditions, and invariants.

4. Interface Segregation Principle (ISP):

- Create cohesive interfaces that are specific to the needs of the clients.

- Avoid bloated interfaces that force clients to depend on methods they don't use.

- Split large interfaces into smaller, more specialized interfaces.

5. Dependency Inversion Principle (DIP):

- Depend on abstractions, not concrete implementations.

- Use dependency injection to provide dependencies to classes instead of creating them internally.

- Use interfaces or abstract classes as dependencies to enable flexibility and easy substitution of implementations.

To implement these principles in C++, you can utilize various design patterns, such as:

- Factory Pattern: Encapsulates object creation, allowing for flexible instantiation without exposing the concrete class.

- Dependency Injection (DI) Pattern: Inverts the responsibility of creating dependencies by injecting them into classes, promoting loose coupling and testability.

- Observer Pattern: Defines a one-to-many dependency relationship between objects, ensuring that changes in one object trigger updates in dependent objects.

- Strategy Pattern: Defines a family of algorithms or behaviors encapsulated in separate classes, allowing them to be selected at runtime.

Remember, the SOLID principles are guidelines, not rigid rules. Their implementation may vary depending on the specific requirements and complexity of your codebase. Applying these principles promotes better code organization, maintainability, and extensibility.

How we implement the abstraction in OOP using c++?

In C++, abstraction can be implemented using abstract classes and pure virtual functions. Here's how you can implement abstraction in OOP using C++:

1. Create an abstract class:

- Define a class that contains at least one pure virtual function.

- Use the "= 0" syntax to indicate a pure virtual function, which means it has no implementation in the base class.

- This abstract class cannot be instantiated.

class AbstractClass {

public:

virtual void pureVirtualFunction() = 0;

// Other members and functions...

};

2. Derive concrete classes from the abstract class:

- Create derived classes that inherit from the abstract class.

- Implement the pure virtual function in each derived class.

- The derived classes must provide an implementation for the pure virtual function to become concrete and instantiable.

class ConcreteClass1 : public AbstractClass {

public:

void pureVirtualFunction() override {

// Implementation specific to ConcreteClass1

}

// Other members and functions...

};

class ConcreteClass2 : public AbstractClass {

public:

void pureVirtualFunction() override {

// Implementation specific to ConcreteClass2

}

// Other members and functions...

};

3. Use abstraction in your code:

- Declare variables or function parameters of the abstract class type.

- You can assign instances of concrete classes to these variables.

- Access the common interface defined by the abstract class.

```cpp

void someFunction(AbstractClass\* object) {

// Call the pure virtual function

object->pureVirtualFunction();

}

int main() {

ConcreteClass1 obj1;

ConcreteClass2 obj2;

someFunction(&obj1); // Pass an instance of ConcreteClass1

someFunction(&obj2); // Pass an instance of ConcreteClass2

return 0;

}

```

By using abstract classes and pure virtual functions, you achieve abstraction in C++. The abstract class defines a common interface, and derived classes provide specific implementations. This allows you to write code that operates on the abstract class type, providing flexibility and allowing for easy extension by adding new derived classes in the future.

The main difference between a virtual function and a pure virtual function is that a virtual function has an implementation in the base class and can be overridden in the derived classes, while a pure virtual function has no implementation in the base class and must be overridden in the derived classes.

Here are the key distinctions:

Virtual Function:

- A virtual function is a function declared in a base class that can be overridden in derived classes.

- It has an implementation in the base class, but it can be overridden in the derived classes to provide different functionality.

- The base class provides a default implementation that can be used if the derived classes do not override the virtual function.

- The derived classes can choose to override the virtual function to provide their own implementation.

Example:

class Base {

public:

virtual void virtualFunction() {

// Base class implementation

}

};

class Derived : public Base {

public:

void virtualFunction() override {

// Derived class implementation

}

};

Pure Virtual Function:

- A pure virtual function is a function declared in a base class that has no implementation in the base class.

- It is denoted by "= 0" at the end of the function declaration.

- The base class with a pure virtual function is called an abstract class, and it cannot be instantiated.

- The derived classes must override the pure virtual function and provide their own implementation.

- If a derived class does not override the pure virtual function, it remains an abstract class and cannot be instantiated either.

Example:

```cpp

class AbstractBase {

public:

virtual void pureVirtualFunction() = 0;

};

class Derived : public AbstractBase {

public:

void pureVirtualFunction() override {

// Derived class implementation

}

};

```

In summary, a virtual function provides a default implementation in the base class and can be overridden in derived classes, while a pure virtual function has no implementation in the base class and must be overridden in derived classes, making the base class abstract.