1. What is a class?

A **class** is a blueprint or template in object-oriented programming (OOP) that defines the

properties (attributes) and behaviors (methods) of objects. It encapsulates data for the object and the methods that manipulate the data. In many programming languages like C++, Python, and SystemVerilog, classes are used to define objects.

Example:

class Car;

string make;

string model;

// Constructor

function new(string make, string model);

this.make = make;

this.model = model;

endfunction

endclass

1. What are virtual classes?

A **virtual class** is a class that is used as a base for other classes but cannot be instantiated directly. It's a concept used in languages like **SystemVerilog** to define base classes for inheritance, where a class provides a common interface but expects derived classes to provide the actual implementation of methods.

Example:

virtual class Animal;

function void speak(); // Virtual method to be implemented by subclasses

$display("Animal makes a sound");

endfunction

endclass

1. What is inheritance and give the basic syntax for it?

**Inheritance** is a feature in OOP where a class can inherit properties and behaviors (methods) from another class. This allows for reusability and extension of existing code.

Example:

class Vehicle;

string brand;

function new(string brand);

this.brand = brand;

endfunction

endclass

class Car extends Vehicle; // Inherit from Vehicle class

string model;

function new(string brand, string model);

super.new(brand); // Call base class constructor

this.model = model;

endfunction

endclass

1. What are super, abstract and concrete classes?

* **Super**: The super keyword refers to the parent class or base class. It allows you to call methods or constructors from the base class.
* **Abstract Class**: An abstract class cannot be instantiated directly and may contain abstract methods (methods without implementation). It provides a common interface for derived classes.
* **Concrete Class**: A concrete class can be instantiated and may have fully implemented methods.

1. What is the significance of super and this keyword?

* **super**: Refers to the parent class and allows calling methods or constructors from the parent class.
* **this**: Refers to the current instance of the class. It can be used to access instance variables and methods within the class.

Example:

class Parent;

function void sayHello();

$display("Hello from Parent");

endfunction

endclass

class Child extends Parent;

function void sayHello();

super.sayHello(); // Call Parent's sayHello

$display("Hello from Child");

endfunction

endclass

1. What is inheritance and its advantages?

**Inheritance** allows one class (child) to inherit the properties and methods of another class (parent). It promotes code reusability, hierarchical relationships, and polymorphism.

Advantages:

* **Code Reusability**: Child classes inherit functionality from parent classes.
* **Extensibility**: Child classes can extend or override methods.
* **Maintainability**: Changes in the parent class propagate to child classes, making it easier to maintain common functionality.

1. What is polymorphism and its advantages?

**Polymorphism** allows methods to behave differently based on the object they are acting upon. It enables objects of different classes to be treated as objects of a common superclass.

Advantages:

* **Flexibility**: One interface can work with objects of different classes.
* **Extensibility**: New functionality can be added with minimal changes to existing code.

Example:

class Animal;

virtual function void speak();

$display("Animal sound");

endfunction

endclass

class Dog extends Animal;

function void speak();

$display("Woof");

endfunction

endclass

class Cat extends Animal;

function void speak();

$display("Meow");

endfunction

endclass

// Usage

Animal a;

a = new Dog();

a.speak(); // Outputs "Woof"

1. What is encapsulation?

**Encapsulation** is the concept of hiding the internal state and implementation details of an object and exposing only the necessary functionality to the outside world. It is achieved using access modifiers like private, protected, and public.

1. What is the difference between private, public and protected members of a class?

* **Private**: Members are accessible only within the same class.
* **Public**: Members are accessible from any part of the program.
* **Protected**: Members are accessible within the class and its subclasses.

1. Can a constructor be qualified as protected or local in System Verilog?

Yes, in SystemVerilog, constructors can be protected or local. A protected constructor can only be called from the class itself or its subclasses. A local constructor can only be used within the class.

1. Difference between structure and class.

* **Structure (SystemVerilog)**: A structure is a value type that holds related data but does not support methods or inheritance.
* **Class**: A class is a reference type that can hold both data and methods and supports inheritance

1. Difference between static and automatic methods in class.

* **Static Methods**: Belong to the class itself, not instances. They do not have access to instance variables and can be called without creating an object.
* **Automatic Methods**: These are instance methods, which operate on the instance's data.

1. Difference between new[] and new().

* **new[]**: Used to create an array of objects.
* **new()**: Used to create a single object.

1. Difference between shallow and deep copy.

* **Shallow Copy**: Copies the reference of an object, not the object itself. Changes to the original object affect the copied object.
* **Deep Copy**: Creates a new copy of the object and all objects referenced by it.

1. How does the OOP concept add benefit to Verification?

In verification, OOP concepts like **encapsulation**, **inheritance**, and **polymorphism** improve modularity, reusability, and flexibility. Testbenches can be more scalable and maintainable, as classes can represent different components and verification scenarios. Furthermore, **abstraction** allows complex verification logic to be hidden behind simpler interfaces.

1. How can we use one class inside another? Explain with an example.

class Engine;

string model;

function new(string model);

this.model = model;

endfunction

endclass

class Car;

Engine eng; // Using Engine class inside Car class

function new(string engineModel);

eng = new(engineModel); // Create an Engine object

endfunction

endclass