21). What is inheritance?

Ans : Inheritance one class derived properties to another class inheritance which is provided resualibility.

22). Which inheritance is not supported by Dart? Why? B3. What is advantage of inheritance.

Ans: Dart does not support multiple inheritance. Multiple inheritance refers to a scenario where a class can inherit from multiple superclasses. Dart avoids multiple inheritance to maintain simplicity and avoid potential conflicts that may arise from inheriting conflicting properties or methods from multiple sources.

23). Difference between inheritance and encapsulation. B5. Difference between inheritance and abstraction.

Ans:

1). Inheritance : Inheritance is one of the fundamental principles of object-oriented programming. It allows a new class (subclass/derived class) to inherit properties and behaviors (methods) from an existing class (superclass/base class). The subclass can also override or extend the inherited methods. Inheritance promotes code reusability and establishes a relationship between classes, where a subclass can inherit attributes and methods from one or more superclasses.

2).Encapsulation:-Encapsulation is the concept of bundling data (attributes) and methods that operate on the data into a single unit known as a class. It restricts direct access to some of an object's components and prevents the accidental modification of data. Encapsulation helps in data hiding, where the internal representation of an object is hidden from the outside world, and access to the data is provided through public methods (getters and setters).

3).Abstract Class:- Abstraction is a concept that deals with hiding the complex reality while exposing only the essential parts. In object-oriented programming, abstraction is implemented through abstract classes and interfaces. An abstract class is a class that cannot be instantiated and may contain abstract methods (methods without implementation). Abstract classes serve as blueprints for concrete (sub)classes, providing a way to define methods that must be created within any child classes built from the abstract class.

24). Difference between inheritance and polymorphism

Ans:

1).Inheritance :- Inheritance is one of the fundamental principles of object-oriented programming. It allows a new class (subclass/derived class) to inherit properties and behaviors (methods) from an existing class (superclass/base class). The subclass can also override or extend the inherited methods. Inheritance promotes code reusability and establishes a relationship between classes, where a subclass can inherit attributes and methods from one or more superclasses.

2).Polymorphism:- Polymorphism is the ability of different classes to be treated as objects of a common superclass. It allows objects of different classes to be treated as objects of a common type. Polymorphism enables the same method or operator to behave differently based on the object it is operating upon. It allows flexibility and extensibility in the code.

25). Can we override static method in Dart?

Ans: In Dart, you cannot directly override static methods because static methods are associated with the class itself rather than with instances of the class. Static methods belong to the class, and they can be called using the class name without creating an instance of the class.

26). Can we overload static method in Dart?

Ans: Method overloading refers to the ability to define multiple methods with the same name in the same class but with different parameter lists. In Dart, if you try to declare multiple methods with the same name, regardless of whether they are static or instance methods, you will get a compilation error.

27). Can a class implement more than one interface? B10. Can a class extend more than one class in Dart?

Ans: Yes, a class can implement more than one interface. Dart supports multiple interface inheritance. When a class implements multiple interfaces, it is required to provide concrete implementations for all the methods declared in those interfaces.

No, Dart does not support multiple inheritance with classes. A Dart class can only extend one class.

28). Can an interface extend more than one interface in Dart?

Ans: Yes, in Dart, an interface can extend more than one interface. Dart supports multiple interface inheritance, allowing you to create new interfaces that inherit methods and properties from multiple existing interfaces.

29). What will happen if a class implements two interfaces and they both have a method with same name and signature?

Ans: In Dart, if a class implements two interfaces, and both interfaces have a method with the same name and signature, the class is required to provide a concrete implementation of that method. When a class implements multiple interfaces with conflicting method names and signatures, Dart doesn't allow implicit resolution of the conflict. Instead, the implementing class must explicitly provide an implementation for the conflicting method, thereby resolving the ambiguity.

30). Can we pass an object of a subclass to a method expecting an object of the super class? B14. Are static members inherited to sub classes?

Ans: Yes, in most object-oriented programming languages, including Dart, you can pass an object of a subclass to a method that expects an object of the superclass. This is a fundamental principle of polymorphism. When a subclass inherits from a superclass, it is considered to be of the superclass type, and therefore, you can use objects of the subclass wherever objects of the superclass are expected. This is known as subtype polymorphism.

No, static members are not inherited by subclasses in Dart. Static members (static fields and static methods) belong to the class itself, not to instances of the class. Subclasses do not inherit static members from their superclasses. Each class, including subclasses, has its own independent copy of static members.

31). What happens if the parent and the child class have a field with same identifier? B16. Are constructors and initializers also inherited to sub classes?

Ans: If both the parent and the child class have a field with the same identifier, the child class field shadows the parent class field. This means that the child class field takes precedence over the parent class field within the scope of the child class. The child class field essentially hides the parent class field with the same name.

•In Dart, constructors are not inherited by subclasses. However, a subclass can call a constructor from its superclass using the super keyword.Initializers, like constructors, are also not inherited by subclasses. When you initialize an object using a constructor, the initializers of the parent class are executed first, followed by the initializers of the child class.

32). How do you restrict a member of a class from inheriting by its sub classes?

Ans: In object-oriented programming, you might encounter situations where you want to restrict a member of a class from being inherited by its subclasses. In Dart, you can achieve this by making the member private, using an underscore (\_) before its name. Private members are not accessible from outside the class, including subclasses.

33). How do you implement multiple inheritance in Dart?

Ans: Dart does not support multiple inheritance in the traditional sense, where a class can directly inherit from more than one class. However, Dart provides a way to achieve similar behavior using a concept called mixins.

34). Can a class extend by itself in Dart?

Ans: In Dart, a class cannot directly extend itself. Attempting to declare a class that extends itself will result in a compilation error. This restriction is in place to prevent infinite recursion and ensure that class hierarchies are well-formed and maintainable.

35). How do you override a private method in Dart?

Ans: In Dart, private methods (and fields) are denoted by an underscore (\_) before their names. Private methods cannot be accessed directly from outside the class, including in subclasses. Therefore, you cannot override a private method in the traditional sense.

36). When to overload a method in Dart and when to override it?

Ans: In Dart, method overloading and method overriding serve different purposes in the context of object-oriented programming.

37). What the order is of extends and implements keyword on Dart class declaration?

Ans: In Dart, when declaring a class, the extends keyword is used to specify the superclass (the class that the new class inherits from), and the implements keyword is used to specify the interfaces that the class implements.

38). How do you prevent overriding a Dart method without using the final modifier?

Ans:- In Dart, if you want to prevent a method from being overridden by subclasses without using the final modifier, you can achieve this by making the method private. By prefixing the method name with an underscore \_, you make it private to the current library. Private methods cannot be accessed or overridden by subclasses outside of the same library.

39). What are the rules of method overriding in Dart?

Ans:- In Dart, method overriding allows a subclass to provide a specific implementation for a method that is already defined in its superclass. Here are the key rules for method overriding in Dart:

**(i).** Method Signature

**(ii).** Keyword @override

**(iii).** Visibility

**(iv)**. Covariant Parameters

40). Difference between method overriding and overloading in Dart.

Ans: In Dart, method overriding and method overloading are two different concepts used to achieve polymorphism and handle different method signatures. Let's explore the differences between method overriding and overloading in Dart:

**(i)**.Method Overriding: Method overriding occurs when a subclass provides a specific implementation for a method that is already defined in its superclass. The overriding method in the subclass must have the same name, return type, and parameter list as the method in the superclass. It is used for achieving runtime polymorphism and is a fundamental concept in object-oriented programming.

**(ii)**.Method Overriding: It allows a subclass to provide a specialized implementation of a method inherited from its superclass. This enables dynamic method invocation, where the appropriate method implementation is determined at runtime based on the object's actual type.

41). What happens when a class implements two interfaces and both declare field (variable) with same name?

Ans: In Dart, if a class implements two interfaces, and both interfaces declare a field (variable) with the same name, the implementing class is required to provide an implementation for that field. This can be done using a getter and setter method or by directly defining the field in the implementing class. The implementing class should specify which interface's field it is implementing by explicitly referencing the interface's name followed by the field name.

42). Can a subclass instance method override a superclass static method?

Ans: No, in Dart, a subclass instance method cannot override a superclass static method. Method overriding in Dart applies to instance methods, not static methods. When you define a static method in a superclass, it belongs to the class itself and not to any specific instance. Subclasses cannot override static methods from their superclasses.

43). Can a subclass static method hide superclass instance method?

Ans: No, in Dart, a subclass static method cannot hide a superclass instance method. Static methods are associated with the class itself, not with instances of the class. They operate at the class level, not at the instance level. Instance methods, on the other hand, are associated with instances of the class and operate on the state of those instances.

44). Can a superclass access subclass member?

Ans: No, in object-oriented programming, the superclass cannot access members (fields or methods) of a subclass directly. The principle of encapsulation, which is one of the core principles of object-oriented programming, dictates that the internal details of a class, including its private and protected members, should not be directly accessible from outside the class, including its superclasses.

45). Difference between object oriented and object based language.

Ans: "Object-oriented" and "object-based" are terms used to describe programming languages based on their support for object-oriented programming (OOP) principles. While these terms are sometimes used interchangeably, they have distinct differences:

Object-Oriented Programming (OOP) Languages:

(1). Classes and Objects: Object-oriented programming languages have support for classes and objects. Classes are user-defined data types that encapsulate data (attributes) and behaviors (methods) into a single unit. Objects are instances of classes.

(2). Inheritance: OOP languages support inheritance, allowing one class (subclass/derived class) to inherit properties and behaviors from another class (superclass/base class). Inheritance promotes code reusability and establishes a relationship between classes.

(3). Encapsulation: OOP emphasizes encapsulation, where data (attributes) and methods that operate on the data are bundled together within a class. Access to data is controlled through getter and setter methods, allowing data hiding and protection.

(4). Polymorphism: Polymorphism enables objects of different classes to be treated as objects of a common superclass. This allows methods to be called on objects without knowing their specific types, facilitating flexibility and extensibility in the code.

(5). Abstraction: Abstraction allows developers to define abstract classes and interfaces, providing a blueprint for concrete classes. Abstract classes cannot be instantiated and can have abstract methods that must be implemented by subclasses.

• Languages like Java, C++, Python, and Ruby are considered truly object-oriented because they support all these principles.

Object-Based Programming Languages:

• Object-based programming languages, on the other hand, support most of the features of object-oriented programming but may lack one or more key features. Commonly, object-based languages lack support for inheritance or polymorphism:

(1). Classes and Objects: Object-based languages have support for classes and objects similar to OOP languages.

(2). Inheritance: Some object-based languages might not support inheritance, meaning that classes cannot inherit properties and methods from other classes. Each class operates as a standalone entity.

(3). Polymorphism: Polymorphism might be limited or absent in object-based languages. Method names must be known and explicitly called on specific object types.