## Q-21. What is inheritance?

ANS: Inheritance in object-oriented programming (OOP) is a mechanism that allows a new class, known as a subclass or derived class, to acquire the properties and behaviors (attributes and methods) of an existing class, known as a superclass or base class. This mechanism enables the creation of a hierarchical relationship between classes, promotes code reuse, and allows for the extension and customization of existing functionality without modifying the original class.

**Types of Inheritance:**

1. **Single Inheritance:** This inheritance occurs when a class inherits a single-parent class.
2. **Multiple Inheritance:** This inheritance occurs when a class inherits more than one parent class. Dart doesn't support this**.**
3. **Multi-Level Inheritance:** This inheritance occurs when a class inherits another child class.
4. **Hierarchical Inheritance:** More than one classes have the same parent class.

## Q-22. Which inheritance is not supported by Dart? Why? . What is advantage of inheritance?

ANS:- Multiple Inheritance is not supported by Dart. This means that a class cannot inherit from more than one class directly. Instead, Dart provides a way to achieve similar functionality through the use of mixins and interfaces.

Multiple inheritance, where a class can inherit from more than one parent class, is not supported in Dart due to several reasons related to complexity, ambiguity, and the design philosophy of the language.

**Advantages of Inheritance**

1. **Code Reusability**: Inheritance allows for code defined in one class to be reused in another, reducing redundancy.
2. **Extensibility**: New functionality can be added to a subclass without modifying the superclass, making it easier to extend and maintain the code.
3. **Polymorphism**: Inheritance supports polymorphism, where a subclass can be treated as an instance of its superclass, allowing for more flexible and generic code.
4. **Logical and Clear Structure**: Inheritance helps to organize code in a clear hierarchy, making it more logical and easier to understand and maintain.

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

ANS:

(i)

**Inheritance**: Inheritance is a concept in object-oriented programming that allows a class (subclass or derived class) to inherit attributes and methods from another class (superclass or base class). It establishes a hierarchical relationship between classes, where subclasses can extend or modify the behavior of the superclass. Inheritance promotes code reusability and

facilitates the creation of hierarchical structures in software design.

**Encapsulation**: Encapsulation is a principle of object-oriented programming that bundles the data (attributes) and methods (functions) that operate on the data into a single unit called a class. It hides the internal state of objects from the outside world and only exposes selected data and behaviors through public interfaces (methods). Encapsulation helps in achieving data abstraction, modularity, and protection of data integrity by preventing unauthorized access and modification of internal state.

(ii)

**Conceptual Focus:**

- **Inheritance:** Focuses on establishing a hierarchical relationship between classes to promote code reuse.
- **Abstraction:** Focuses on defining a blueprint for classes, often using abstract classes or interfaces, to specify methods that must be implemented by subclasses.

**Implementation:**

- **Inheritance:** Implemented using keywords like extends in Java or class Subclass(BaseClass) in Python to inherit properties and behaviors from a superclass.
- **Abstraction:** Implemented using abstract classes (in languages like Java) or interfaces (in languages like Java and C#) to define abstract methods that must be overridden by subclasses.

**Usage:**

- **Inheritance:** Used to extend the functionality of existing classes and establish a "is-a" relationship between classes.
- **Abstraction:** Used to define a common interface for a group of related classes, promoting code reusability and design flexibility.

## Q-24. Difference between inheritance and polymorphism. ANS:-

**Inheritance**: Inheritance is a fundamental concept in object-oriented programming where a class (subclass or derived class) can inherit attributes and methods from another class (superclass or base class). It allows for the reuse of code and facilitates the creation of hierarchical relationships between classes. Through inheritance, subclasses can extend or modify the behavior of the superclass, inheriting its properties and methods.

**Polymorphism**: Polymorphism refers to the ability of different objects to respond to the same message or method call in different ways. It allows objects of different classes to be treated as objects of a common superclass. Polymorphism enables flexibility and extensibility in object-oriented systems by enabling methods to be implemented in different ways based on the objects they operate on. This is often achieved through method overriding in subclasses, where methods with the same signature as those in the superclass provide specialized implementations.

## Q-25. Can we override static method in Dart?

ANS: In Dart, static methods cannot be overridden because they are associated with the class itself rather than with instances of the class.

## Q-26. Can we overload static method in Dart?

ANS: In Dart, you cannot overload static methods directly. Dart does not support method overloading by defining multiple methods with the same name but different parameter lists within the same class. Each method must have a unique name.

## Q-27. Can a class implement more than one interface? Can a class extend more than one class in Dart?

ANS: Yes, a class in many object-oriented programming languages can implement more than one interface. Implementing multiple interfaces allows a class to inherit behaviour and contracts from each interface independently, enabling greater flexibility in how it interacts with different parts of a system or with other classes.

## Q-28. Can an interface extend more than one interface in Dart?

ANS: No, in Dart, an interface cannot extend more than one interface. Dart follows a single inheritance model,

which means a class or interface can extend only one other class or interface.

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

ANS: If a class implements two interfaces that both have a method with the same name and signature, the class must provide a single implementation of that method. This implementation will serve both interfaces.

## Q-30. Can we pass an object of a subclass to a method expecting an object of the super class? Are static members inherited to sub classes? ANS:

Yes, you can pass an object of a subclass to a method that expects an object of the superclass in Dart. This is a

common feature in object-oriented programming languages known as polymorphism. When a method expects an object of a superclass, you can pass an object of any subclass because a subclass object is considered to be an instance of the superclass.

In Dart, static members (both methods and fields) are not inherited by subclasses. Static members belong to the class itself rather than to any instance of the class. Consequently, they cannot be accessed via an instance of the class or a

subclass. They must be accessed directly through the class they are defined in.

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

ANS: In Dart, if both the parent and the child class have a field with the same identifier, the field in the child class will hide the field in the parent class. This means that when you access the field using an instance of the child class, you will access the field defined in the child class. However, you can still access the field in the parent class using '**`super`**'.

In Dart, constructors and initializers are not inherited by subclasses. Each class needs to define its own constructors, but subclasses can call the constructors of their superclass using the **'`super`'** keyword.

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

ANS: In Dart, you can restrict a member of a class from being inherited by its subclasses by using the `private` convention. In Dart, privacy is enforced at the library level rather than the class level, and you can make a class member private to the library by prefixing its name with an underscore (_).

## Q-33. How do you implement multiple inheritance in Dart?

ANS: Dart does not support multiple inheritance directly through classes, but it provides a feature called mixins that allows you to achieve a similar effect. Mixins enable you to reuse code across multiple classes.

## Q-34. Can a class extend by itself in Dart?

ANS: **No**, a class cannot extend itself in Dart. Dart does not support self-extension or self-inheritance. A class can only extend one superclass, and it cannot extend itself.

## Q-35. How do you override a private method in Dart?

ANS: In Dart, you cannot directly override a private method from a superclass in a subclass because private members are not visible outside of the library where they are defined.

## Q-36. When to overload a method in Dart and when to override it?

ANS: Dart doesn't support method overloading, but it does support method overriding. Overriding allows a child class to provide its own implementation for a method that already exists in the parent class.

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

ANS: In Dart, when declaring a class, the **`'extends'`** keyword comes before the **`'with'`** keyword (used for mixins) and the **`'implements'`** keyword.

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

ANS: To prevent overriding a Dart method without using the final modifier, you can use the @override annotation on the child class method and the @protected annotation on the parent class method. The @override annotation ensures that the child class method is indeed an override of the parent class method, and the @protected annotation restricts access to the parent class method to only the child class and its subclasses.

## Q-39. What are the rules of method overriding in Dart?

ANS**: Rules of Method Overriding in Dart**

1. **Same Method Signature**:
    - The overriding method must have the same method signature as the method in the

superclass. This includes the method name, parameter types, and return type.

2. **Override Annotation**:
    - It is a good practice (though not strictly required) to use the @override annotation when overriding a method. This helps with code readability and ensures that the method is correctly overriding a method from the superclass.
3. **Accessing Superclass Method**:
    - The overridden method can call the method from the superclass using super.methodName(). This is useful if you want to extend the functionality of the superclass method rather than completely replacing it.
4. **No Final Methods**:
    - A method in a superclass cannot be overridden if it is marked as final.
5. **No Overriding Private Methods**:
    - Private methods (methods starting with an underscore _) cannot be overridden outside the file they are declared in, because they are not accessible outside their own library.
6. **Covariant Parameters**:
    - Dart allows you to specify that a parameter of an overridden method is covariant. This means

that the parameter can accept subtypes of the type specified in the superclass

## Q-40. Difference between method overriding and overloading in Dart.

ANS: **Method overriding** in Dart occurs when a subclass provides a specific implementation for a method that is already defined in its superclass. This allows the subclass to modify or extend the behavior of the inherited method. Overriding is indicated with the @override annotation, and the method in the subclass must have the same name, return type, and parameters as the method in the superclass.

**Method overloading** is about defining multiple methods with the same name but different parameters within the same class. Dart does not support traditional method overloading, as seen in languages like Java or C++. Instead, Dart achieves similar functionality through the use of optional positional parameters or named parameters, allowing methods to be called in various ways based on the parameters provided.

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

ANS: When a class in Dart implements two interfaces that both declare a field (variable) with the same name, Dart does not allow fields to be part of interfaces. This means you cannot have conflicting field names from interfaces because interfaces in Dart can only contain method signatures, not fields.

## Q-42. Can a subclass instance method override a superclass static method?

ANS In Dart, a subclass instance method cannot override a superclass static method. Static methods belong to the class itself and not to instances of the class.

## Q-43. Can a subclass static method hide superclass instance method?

ANS: In Dart, a subclass's static method cannot hide a superclass's instance method. This is because static methods belong to the class itself rather than an instance of the class, while instance methods belong to an instance of the class.

## Q-44. Can a superclass access subclass member?

ANS: In Dart, a superclass cannot directly access members (fields, methods) that are defined in its

subclasses. The inheritance relationship in objectoriented programming flows from the superclass to the subclass, meaning that the subclass inherits from the superclass, but the superclass does not inherit from the subclass.

## Q-45. Difference between object oriented and object based language.

ANS: **Object-Oriented Language**:

- Fully supports all principles of object-oriented programming (OOP) such as classes, objects, inheritance, polymorphism, and encapsulation.
- Examples include Java, C++, Python, Ruby, Dart.

**Object-Based Language**:

- Supports objects and their attributes but may lack full support for inheritance and polymorphism, limiting the application of OOP principles.
- Example includes JavaScript (to some extent), which supports objects but does not fully support classical inheritance and other advanced OOP features.