**21. What is inheritance?**

* In Dart, the mechanism by which a class can inherit properties and behaviors from another class is called **inheritance**.

There are five types of inheritance:

1. **Single Inheritance** - In this type of inheritance, a class can inherit from only one class. In Dart, we can only extend one class at a time.
2. **Multilevel Inheritance** - In this type of inheritance, a class can inherit from another class and that class can also inherit from another class. In Dart, we can extend a class from another class which is already extended from another class.

**Ex, A**

**|**

**B**

**|**

**C**

1. **Hierarchical Inheritance** - In this type of inheritance, parent class is inherited by multiple subclasses.

Ex, A

/ | \

D B C

1. **Multiple Inheritance** - In this type of inheritance, a class can having more then one parent class . it known as multiple inheritance.

Ex, A B C

\ | /

D

1. **Hybrid Inheritance** – Two or more combination of multiple inheritance.

Ex, A A

/ \ |

B C B D

\ / | /

D C

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

* Dart does not support multiple inheritance because it can lead to ambiguity. For example, if class **Apple** inherits class **Fruit** and class **Vegetable**, then there may be two methods with the same name **eat**. If the method is called, then which method should be called? This is the reason why Dart does not support multiple inheritance.

**Advantages Of Inheritance In Dart**

* It promotes reusability of the code and reduces redundant code.
* It helps to design a program in a better way.
* It makes code simpler, cleaner and saves time and money on maintenance.
* It facilitates the creation of class libraries.
* It can be used to enforce standard interface to all children classes.

**23. Difference between inheritance and encapsulation.**

1. Inheritance: Inheritance is a mechanism in which a new class (subclass or derived class) is created based on an existing class (superclass or base class). The subclass inherits attributes and methods from the superclass, allowing it to reuse and extend the functionality of the superclass. Inheritance promotes code reusability and allows for the creation of hierarchical relationships between classes.
2. Encapsulation: Encapsulation is the concept of bundling the data (attributes or properties) and methods (functions or procedures) that operate on the data into a single unit known as a class. It hides the internal implementation details of a class from the outside world and only exposes a public interface for interacting with the class. Encapsulation helps in controlling access to the data and ensuring data integrity by enforcing access restrictions through access modifiers (public, private, protected).

**B5. Difference between inheritance and abstraction.**

1. Inheritance: Inheritance is a mechanism in which a new class (subclass or derived class) is created based on an existing class (superclass or base class). The subclass inherits attributes and methods from the superclass, allowing it to reuse and extend the functionality of the superclass. Inheritance establishes an "is a" relationship between classes, where the subclass is a specialized version of the superclass. It promotes code reuse and facilitates the creation of class hierarchies.
2. Abstraction: Abstraction is the process of hiding the complex implementation details of an object and exposing only the essential features or properties to the outside world. It allows you to focus on what an object does rather than how it does it. In object-oriented programming, abstraction is achieved through abstract classes and interfaces. Abstract classes may have abstract methods (methods without an implementation) that must be defined by any class inheriting from the abstract class. Interfaces define a contract of methods that must be implemented by classes that implement the interface. Abstraction helps in reducing complexity, increasing reusability, and promoting a clear separation of concerns.

**24. Difference between inheritance and polymorphism**.

1. Inheritance: Inheritance is a mechanism in object-oriented programming where a new class (subclass or derived class) is created based on an existing class (superclass or base class). The subclass inherits attributes, methods, and behaviors from the superclass, allowing it to reuse and extend the functionality of the superclass. Inheritance establishes an "is a" relationship between classes, indicating that the subclass is a specialized version of the superclass. It promotes code reuse, helps in developing class hierarchies, and facilitates the organization of code in a hierarchical manner.
2. Polymorphism: Polymorphism is the ability of different classes to be treated as instances of a common superclass through the use of a common interface. In other words, objects of different classes can be addressed using a common interface or a base class type. Polymorphism allows methods to operate on objects of various classes, each implementing the method in its way. There are two main types of polymorphism: compile-time polymorphism (method overloading) and runtime polymorphism (method overriding). Method overriding enables a subclass to provide a specific implementation of a method that is already provided by its superclass. Polymorphism enables flexibility, extensibility, and code reusability.

**25. Can we override static method in Dart?**

* In Dart, static methods cannot be overridden in the same way that instance methods can be overridden. Static methods are associated with the class itself rather than with instances of the class, so they belong to the class rather than to specific objects. Therefore, it does not make sense to override static methods in Dart because they are not inherited by subclasses in the same manner as instance methods.
* Static methods are called directly on the class, and the method that gets executed is based on the class where the method is defined rather than the class of the object. Subclasses cannot provide alternative implementations for static methods defined in their superclass. Instead, subclasses can define their own static methods with the same name, but this is not considered method overriding.

**26. Can we overload static method in Dart?**

* In Dart, method overloading is not supported, whether the method is static or instance method. Method overloading refers to the ability to define multiple methods in a class with the same name but a different number or type of parameters. The decision on which method to call is based on the number and types of arguments passed during the method call.
* In Dart, if you define multiple methods with the same name in a class, they would conflict with each other, and Dart will not be able to determine which method to call based on the arguments passed.
* However, you can achieve similar behavior by using optional and named parameters in Dart. Optional parameters allow you to define methods with default values for some parameters, and named parameters allow you to specify arguments by their names when calling a method. By using optional and named parameters, you can emulate method overloading to some extent in Dart.
* In conclusion, method overloading is not natively supported in Dart, including for static methods. It is important to structure your code in a way that avoids naming conflicts and ambiguity when defining methods with similar functionalities.

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

1. A class can implement more than one interface: Yes, a class in Dart can implement multiple interfaces. This means that a class can specify that it guarantees to implement the behaviors defined by multiple interfaces. When a class implements multiple interfaces, it needs to provide concrete implementations for all the methods defined in each of the interfaces it implements. This allows for a class to exhibit multiple types of behavior specified by the different interfaces it implements.
2. A class cannot extend more than one class: Unlike some other programming languages like Java, in Dart, a class can only extend one superclass. Dart does not support multiple inheritance, which means a class can have only one direct superclass. This single inheritance model helps in avoiding issues related to method conflicts and ambiguity that can arise from multiple inheritance.

In summary, in Dart, a class can implement multiple interfaces to guarantee implementation of different types of behaviors specified by those interfaces. However, a class can only extend one superclass, as Dart does not support multiple inheritance.

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

* In Dart, an interface can extend multiple interfaces. This means that an interface can inherit behaviors and method signatures from multiple parent interfaces, allowing for a hierarchical structure of interfaces. When an interface extends multiple interfaces, it combines the behaviors and method signatures defined in each of the parent interfaces.
* Dart allows interfaces to extend multiple interfaces, enabling the creation of complex type structures and promoting code reuse and flexibility in designing interfaces.

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

* If a class in Dart implements two interfaces, and both interfaces declare a method with the same name and signature, the class must provide a concrete implementation for that method once. Dart will not allow the class to implement the same method twice with conflicting implementations.
* When a class implements multiple interfaces with a method of the same name and signature, Dart requires the class to provide a single implementation of that method that satisfies the requirements of both interfaces. This means that the class must reconcile the potentially different implementations specified by the two interfaces into a single coherent implementation.

**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?**

* In Dart, you can pass an object of a subclass to a method expecting an object of the superclass. This is possible due to the concept of subtype polymorphism, which allows a subclass instance to be treated as an instance of its superclass. This means that a subclass object can be substituted wherever a superclass object is expected.
* Regarding static members in Dart, they are not inherited by subclasses. Static members belong to the class itself and are not inherited by subclasses. Subclasses do not have access to the static members of their super classes. Each class, including its subclasses, has its own 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?**

* When both a parent class and a child class have a field with the same identifier, the child class field will hide the field in the parent class. This is known as field hiding. In Dart, when a child class declares a field with the same name as a field in its superclass, the child class's field will shadow or hide the parent class's field.
* Regarding constructors and initializers in Dart, constructors are not inherited by subclasses. Each class, including its subclasses, must define its own constructors. However, Dart does provide the ability to automatically invoke constructors from the superclass using the super keyword.
* When a subclass is created, and if the superclass has a default constructor (no parameters), the subclass's default constructor will automatically call the superclass's default constructor. If the superclass has a parameterized constructor, the subclass must explicitly call a superclass constructor using the super keyword with the appropriate parameters.
* Initializers, which are part of the constructors, are also not inherited by subclasses. Subclasses must initialize their own fields using their constructors or by calling the superclass's constructor to initialize inherited fields.

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

* In Dart, you can restrict a member of a superclass from being inherited by its subclasses by using the final keyword. When a member (field or method) is marked as final, it cannot be overridden by subclasses, thus restricting its inheritance.
* Here's an example demonstrating how you can use the final keyword to restrict a member from being inherited by its subclasses:

class Parent {

final String name = 'Parent'; // Final field that cannot be overridden

final void introduceYourself() {

print('Hello, I am $name');

}

}

class Child extends Parent {

// This class cannot override the final members in the Parent class

}

void main() {

var child = Child();

// Uncommenting the lines below will result in a compilation error

// print(child.name);

// child.introduceYourself();

}

In this example, both the name field and introduce Yourself method in the Parent class are marked as final, indicating that they cannot be overridden by subclasses. The Child class cannot access or override these final members from the Parent class.

By using the final keyword, you can effectively restrict a member in a superclass from being inherited or overridden by its subclasses in Dart.

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

Dart does not support multiple inheritance, where a class can inherit from more than one class at a time. However, Dart provides a way to achieve code reuse through mixins. Mixins allow you to "mix in" reusable pieces of code into a class without the need for sub classing.

A mixin is a class that provides a set of methods or fields that can be added to other classes. To use a mixin in Dart, you define a class that extends Object (or another class) and use the with keyword to apply the mixin to another class.

Here's an example demonstrating how to implement mixins in Dart:

// Mixin defining a method speak

mixin Speaker {

void speak(String message) {

print(message);

}

}

// Mixin defining a method run

mixin Runner {

void run() {

print('Running');

}

}

// Class implementing Speaker and Runner mixins

class Person with Speaker, Runner {

String name;

Person(this.name);

}

void main() {

var person = Person('Alice');

person.speak('Hello, Dart!'); // Output: Hello, Dart!

person.run(); // Output: Running

}

In this example, the Speaker and Runner mixins define the speak and run methods, respectively. The Person class uses the with keyword to apply both mixins. As a result, instances of the Person class gain access to the speak and run methods defined in the mixins.

By using mixins, you can achieve code reuse and behavior composition in Dart without the need for multiple inheritance. Mixins provide a flexible way to add functionality to classes and avoid some of the issues associated with traditional multiple inheritance.

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

In Dart, a class cannot directly extend itself, as it would result in a cyclic inheritance which is not allowed. Cyclic inheritance occurs when a class tries to inherit from itself either directly or indirectly through a chain of other classes.

If you attempt to define a class that directly extends itself, you will encounter a compilation error.

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

In Dart, private methods (methods prefixed with an underscore \_) are not meant to be overridden by subclasses. Private methods are only accessible within the same library where they are defined, and they are not visible to subclasses.

If a private method is defined in a superclass, it cannot be accessed or overridden by subclasses outside the library where the superclass is defined. This helps in encapsulation and ensuring that the behavior of a class remains consistent regardless of its subclass implementations.

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

In Dart, overloading and overriding are two different concepts related to methods in classes:

1. Overloading: Overloading refers to defining multiple methods in the same class with the same name but different parameters. Dart does not support method overloading based on the number or types of parameters. Therefore, in Dart, you cannot have multiple methods with the same name but different parameter lists like in some other languages such as Java or C++.
2. Overriding: Overriding refers to providing a new implementation for a method in a subclass that is already defined in its superclass. When a subclass provides its own implementation of a method that is already defined in its superclass, it's called method overriding. This allows the subclass to provide specific behavior for that method while still maintaining the method signature defined in the superclass.

Here are some guidelines on when to use method overloading and when to use method overriding in Dart:

* Use method overriding when you want a subclass to provide a specific implementation for a method that is already defined in its superclass. This allows for polymorphic behavior and enables the subclass to customize the behavior of the inherited method.
* Use method overloading when you want methods with the same name but different parameters in other languages. While Dart does not support method overloading based on parameters, you can achieve similar behavior using named parameters or default parameter values to provide different functionalities within a single method

Overall, method overriding is more commonly used in Dart to implement polymorphism and to provide specific behaviors in subclasses, while method overloading is not supported in Dart in the traditional sense.

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

In Dart, when declaring a class, the extends keyword should always come before the with keyword if the class is extending a superclass and implementing mixins. The general order of keywords in a class declaration is as follows:

1. Class Name
2. extends (if extending a superclass)
3. with (if implementing mixins)
4. implements (if implementing interfaces)

Here's an example of the correct order of keywords in a Dart class declaration:

class Child Class extends Parent Class with Mixin1, Mixin2 implements Interface1, Interface2 {

// Class members and methods

}

In this example:

* Child Class is the name of the class.
* extends Parent Class indicates that Child Class is extending the Parent Class.
* with Mixin1, Mixin2 specifies that Child Class is using Mixin1 and Mixin2 as mixins.
* implements Interface1, Interface2 declares that Child Class is implementing Interface1 and Interface2.

It's important to follow this order when declaring a class in Dart to ensure the correct inheritance and composition relationships are established between classes and mixins. Paying attention to the order of keywords will help maintain code readability and avoid potential syntax errors.

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

* 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 static. By declaring a method as static, it becomes associated with the class itself rather than with instances of the class. Subclasses cannot override static methods because they are not inherited as instance methods.
* Here is an example demonstrating how to prevent overriding a method in Dart without using the final modifier:

class Parent {

static void staticMethod() {

print('This is a static method in the Parent class');

}

}

class Child extends Parent {

// Attempting to override the static method will result in a compilation error

// void staticMethod() {

// print('This is a static method in the Child class');

// }

}

void main() {

// Calling the static method from the Parent class

Parent.staticMethod();

// Attempting to call the static method from the Child class will result in a compilation error

// Child.staticMethod();

}

* In this example, the static Method() in the Parent class is a static method, and when the Child class tries to override or provide its implementation for this method, it will result in a compilation error.
* By using static methods, you can prevent subclasses from overriding specific methods while still allowing those methods to be called using the class name itself. This technique helps enforce specific behaviors in classes and maintain control over method implementations in the class hierarchy.

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

* In Dart, method overriding allows a subclass to provide a specific implementation for a method that is already defined in its superclass. When overriding a method in Dart, there are a few rules to keep in mind:

1. Method Signature:
   * The overriding method in the subclass must have the same name, return type, and parameters (including types and order) as the method in the superclass.
2. Visibility Modifier:
   * The visibility modifier of the overriding method in the subclass should be the same as or less restrictive than the overridden method in the superclass.
3. The @override Annotation:
   * While not required, it is a good practice to use the @override annotation above the overriding method in the subclass to indicate that it is intended to override a method from the superclass.
4. Super Keyword:
   * Inside the overriding method in the subclass, if you want to call the overridden method from the superclass, you can use the super keyword followed by the method name.
5. Covariant Parameter:
   * If you want to allow a subclass to override a method with more specific parameter types, you can use the covariant keyword before the parameter type in the superclass method signature.

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

Here are the key differences between method overriding and method overloading in Dart:

**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 parameters (including types and order) as the method in the superclass.
* The purpose of method overriding is to provide a specialized behavior in subclasses while maintaining the polymorphic behavior of object-oriented programming.
* Overriding is a way to achieve runtime polymorphism in Dart, where the specific implementation of a method is determined dynamically based on the object's actual type.

**Method Overloading**:

* Method overloading refers to the ability to have multiple methods in the same class with the same name but different parameters (or different number of parameters).
* In Dart, method overloading is not supported in the traditional sense because Dart does not allow multiple methods with the same name, even if they have different parameters.
* In Dart, if you declare multiple methods with the same name, the last method declaration will overwrite the previous ones. This is called shadowing, not overloading.

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

* In Dart, if a class implements two interfaces and both interfaces declare a field (variable) with the same name, it can lead to a conflict. Dart does not allow a class to inherit the same member (field or method) multiple times through interface implementation.
* When a class implements multiple interfaces with the same field name, Dart will raise a compile-time error indicating a naming conflict. The error message will inform you that the field is already defined in the class due to the implementation of multiple interfaces that specify the same field name.
* To resolve this naming conflict, you can explicitly define the field in the class and provide your own implementation that satisfies the requirements of both interfaces. By providing your own implementation, you can specify how the field should behave in the context of the class that implements the interfaces.

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

* No, in Dart, a subclass instance method cannot override a superclass static method. Method overriding in Dart is based on the concept of inheritance and polymorphism, which applies to instance methods of a class.
* Static methods in Dart belong to the class itself rather than to instances of the class. They are not inherited by subclasses and cannot be overridden by subclass instance methods. Subclasses can define their own static methods with the same name as the superclass static method, but this does not constitute method overriding.

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

a subclass static method can indeed "hide" a superclass instance method if both the methods have the same name. However, it's important to clarify what this means in Dart's context regarding static and instance methods.

### Explanation

1. **Static Methods in Dart**: Static methods belong to the class itself rather than to instances of the class. They are called on the class rather than on an instance of the class.
2. **Instance Methods in Dart**: Instance methods are associated with a specific object or instance and can access instance variables and other instance methods.

When a subclass defines a static method with the same name as an instance method from the superclass, if you call that method on the subclass, the static method will be used rather than the instance method from the superclass.

**44. Can a superclass access subclass member?**

a superclass cannot directly access members (attributes or methods) of its subclass. This concept is rooted in the principles of inheritance and polymorphism, where a superclass is designed to operate independently of its subclasses.

Here's how it works:

1. **Superclasses Do Not Know About Subclasses**: The superclass is not aware of any members or implementation details of its subclasses. It defines a structure that subclasses may extend or implement, but it does not have access to their specific implementations or members.
2. **Member Access**: When a method in a superclass tries to call a member that it expects to be in its own class, it can only access the members that are defined in that superclass or its own ancestors (parent classes). If you try to access subclass-specific members from a superclass, it will result in an error.

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

Many of us have a misconception that Java script is an object oriented language. But, the truth is Java Script is an Object Based Language.

Object Based languages are different from Object Oriented Languages:

## Object Based Languages

* Object based languages supports the usage of object and encapsulation.
* They does not support inheritance or, polymorphism or, both.
* Object based languages does not supports built-in objects.
* Javascript, VB are the examples of object bases languages.

## Object Oriented Languages

* Object Oriented Languages supports all the features of Oops including inheritance and polymorphism.
* They support built-in objects.
* C#, Java, VB. Net are the examples of object oriented languages.

**46. Create a program using List**

A very commonly used collection in programming is an **array**. Dart represents arrays in the form of **List** objects. A **List** is simply an ordered group of objects.

void main() {

var list = [210, 21, 22, 33, 44, 55];

print(list[0]);

print(list[1]);

print(list[2]);

print(list[3]);

print(list[4]);

print(list[5]);

}

**Output**: 210

21

22

33

44

55

**47. Create a program using Set**

* Set is a unique collection of items. You cannot store duplicate values in the Set.

void main() {

// declaring fruits as Set

Set<String> fruits = {"Apple", "Orange", "Mango", "Banana"};

// using different properties of Set

print("First Value is ${fruits.first}");

print("Last Value is ${fruits.last}");

print("Is fruits empty? ${fruits.isEmpty}");

print("Is fruits not empty? ${fruits.isNotEmpty}");

print("The length of fruits is ${fruits.length}");

}

**Output:**

First Value is Apple

Last Value is Banana

Is fruits empty? false

Is fruits not empty? true

The length of fruits is 4

**48. Create a program using Map**

* In a Map, data is stored as keys and values. In Map, each key must be unique.

**Ex,**

void main(){

Map<String, String> countryCapital = {

'USA': 'Washington, D.C.',

'India': 'New Delhi',

'China': 'Beijing'

};

print(countryCapital);

}

**Output:**

{USA: Washington, D.C., India: New Delhi, China: Beijing}