1) What is inheritance?

Ans:

Inheritance is dart is defined as the process in which one class derive the properties and characteristics of another class.

Parent class:

A class that is inherited by the other class is known as the parent class. Sometimes, we also refer to it as the base class.

Child class:

A class that inherits the properties and characteristics of the parent class is known as child class.

Whenever a child class wants to inherit the properties and characteristics of a parent class we make use of the extends keyword.

2) Which inheritance is not supported by Dart? Why? What is the advantage of inheritance?

Ans.

Multiple Inheritance is not supported by Dart.
This means that a class cannot inherit from more than one superclass.

Reasons why dart does not support multiple inheritance

1) Complexity and ambiguity:
Multiple inheritance can lead to ambiguity and complexity in the class hierarchy.

2) Simplification:
By avoiding multiple inheritance, the language and its use become simpler and more predictable, this simplifies both the language implementation and the mental model for developers.

3) Maintainability:
Codebases using multiple inheritance can become difficult to maintain because changes in one superclass can have unintended side effects in subclasses, especially when they are also inheriting from other superclasses.

Advantages:
1) Code reusability
2) Method overriding
3) Polymorphism
4) Hierarchical classification
5) Ease of maintenance
6) Enhanced readability

3)Difference between inheritance and encapsulation.
Difference between inheritance and abstraction

Inheritance in Dart

Inheritance in dart allows a class to inherit properties and methods from another class, enabling code reuse and the creation of class hierarchy.

Inheritance allows one class to inherit the properties and methods of another class

Ex:

```
Class firstclass{
      Body part
  }

Class secondclass extends  parentclass{
      Body part
  }
```

Encapsulation in Dart

Encapsulation in dart is achieved by restricting access to the internal state of an object and allowing access through public methods. In Dart, private members are marked with an underscore prefix.

Encapsulation hides the internal state and requires all interaction to be performed through an object's methods:

Ex:

```
Class myclass{
   Int number; // private field
   Int get xyz => _privateField; // public getter
    }
```

Abstraction in Dart

Abstraction in Dart is implemented using abstract classes and interfaces, which define a class structure without providing full implementation.

Abstraction provides a way to define the structure of a class without implementing all its methods, using abstract classes and interfaces:

Ex:

```
abstract class AbstractClass {
            void body();
 }
class mainclass extends AbstractClass {
       @override
        void abstractMethod() {
              // Implementation
          }
}
```

4) Difference between inheritance and polymorphism

inheritance :

1) **Hierarchy establishment** : inheritance establishes a hierarchical relationship between classes.

2) **Code reusability:** it promotes code reuse by allowing subclasses to inherit properties and methods from superclasses.

3) **Parent-child relationship:** inheritance creates a parent-child relationship between classes, where child classes inherit attributes and behaviors from parent classes.

Polymorphism:

1) **Interface uniformity:** polymorphism allows objects of different classes to be treated as instances of a common superclass.

2) **Method Overriding:** it involves method overriding, where a method in a subclass has the same name and signature as a method in its superclass, but with different implementation.

3) **Dynamic behavior:** polymorphism enables dynamic behavior, where the method to be invoked is determined at runtime based on the actual type of the object.

4) **Implementation:** it's achieved through method overriding,where a subclass provides a specific implementation of a method defined in its superclass.

5) Can we override static methods in Dart?

Ans: No, it's not possible to override static methods in Dart

6) Can we overload static methods in Dart?

Ans: Dart does not support method overloading.

7) Can a class implement more than one interface?

Ans: yes, in Dart, a class can implement more than one interface.

Q)Can a class extend more than one class in Dart?

Ans: no, in Dart, a class cannot extend more than one class.

8) Can an interface extend more than one interface in Dart?

Ans: yes, in Dart, a class can implement more than one interface. By using the 'implements' keyword, a class can promise to provide implementation for all the methods defined in the multiple interfaces it implements.

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

Ans: the class must provide a single implementation of that method. This implementation will fulfill the requirements of both interfaces.

10) Can we pass an object of a subclass to a method expecting an object of the superclass?

Ans: Yes, in Dart, you can also pass an object of a subclass to a method expecting an object of superclass. This is possible due to dart's support for polymorphism and inheritance.

Q) Are static members inherited to subclasses?

Ans: in Dart, static members are not inherited by subclasses. Static members belong to the class itself rather than to any instance of the class.

11) What happens if the parent and the child class have a field with the same identifier?

Ans: in Dart, if both the parent and the child class have a field with the same identifier, the field in the child class hides 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 child's field, not the parent's field.

Q) Are constructors and initializers also inherited to subclasses?

Ans: in Dart, constructors are not inherited by subclasses. Each class must define its own constructors. A subclass can call a superclass constructor using the 'super' keyword in its own constructor.

Initializers, which are used to initialize fields in a class, can be included in a subclass constructor but they must explicitly call the superclass constructor if necessary.

12) How do you restrict a member of a class from inheriting by its subclasses?

Ans: in Dart, you can use the 'final' and 'private' keyword to control access to class members and restrict inheritance:

Private Members: by making a class member private, you can restrict its visibility to the class it is defined in. In Dart, private members are denoted by a leading underscore '_' in their names, private members are not accessible to subclasses or other classes outside the library in which they are defined.

Private Members: The 'final' keyword is used to create immutable fields, but it does not prevent inheritance. It ensures that the value of the field can only be set once.

13) How do you implement multiple inheritance in Dart?

Ans: Dart does not support multiple inheritance directly, meaning a class cannot inherit from more than one class.

Dart provides a feature called "Mixins" that allows you to achieve similar functionality by reusing code from multiple classes.

14) Can a class extend by itself in Dart?

Ans: No, a class in Dart cannot extend itself, in OOP, including Dart, a class cannot inherit from itself.

Inheritance is a relationship where a class inherits properties and behaviors from another class.

15) How do you override a private method in Dart?

Ans: in Dart, private methods are those that start with an underscore and are only accessible within the library in which they are defined.

You can achieve similar functionality through other means, such as designing a new public or protected method in the subclass that mimics or extends the behavior of the private method.

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

Ans:

Method Overloading:
Method overloading is not directly supported in Dart.

When to use:
To provide multiple ways to call a method: when you want to give users of your class multiple ways to call a method with different sets of parameters.

Default values: when you need to provide default values for some parameters.

Method overriding:
Method overriding is used when you want to provide a specific implementation of a method that is already defined in a superclass.

When you need different implementations of the same method in different subclasses to provide specific behaviors.

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

Ans: in Dart, when declaring a class that both extends another class and implements one or more interfaces the 'extends' keyword must come before the 'implements' keyword.

extends : if the class is extending a superclass.
Implements : if the class is implementing one or more interfaces.

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

Ans:

1) Use Private Methods
2) Use Composition instead of inheritance

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

Ans:

1) Same method signature
2) Use the '@override' annotation
3) Accessing the superclass method
4) Visibility and accessibility
5) Covariant return types
6) Method parameters
7) Private methods cannot be overridden

20) Difference between method overriding and overloading in Dart.

Method overriding:

1) **Definition :**
   method overriding occurs when a subclass provides a specific implementation for a method that is already defined in its superclass.

2) **Purpose:**
   to provide a specific implementation of a method in a subclass to change or extend the behavior defined in the superclass.

3) **Method signature:**
   the method in the subclass must have the same name, return type, and parameters as the method in the superclass.

4) **Annotations:**
   typically uses the '@override' annotation to indicate that a method is being overridden.

5) **Inheritance:**
   requires inheritance. The subclass inherits from the superclass.

6) **Accessing superclass method :**
   The 'super' keyword can be used to call the superclass version of the method.

7) **Visibility :**
   the overriding method must have the same or less restrictive visibility compared to the method in the superclass.

Method overloading

1) **Definition :**
   Method overloading occurs when multiple methods in the same class have the same name but different parameter lists. **Dart does not support method overloading directly like some other languages.**

2) **Purpose :**
   to allow a method to handle different types or numbers of arguments.

3) **Method signature :**
   the overloaded methods must have the same name but different parameter lists

4) **Annotations :**
   no specific annotations are required.

5) **Inheritance :**
   overloading does not require inheritance, it can be done within the same class.

6) **Implementation in dart :**
   Dart achieves overloading through the use of optional positional parameters or named parameters instead of true method overloading.

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

Ans.
In dart, interfaces are implemented using classes, and fields in interfaces are treated as member variables of those classes.

Dart does not have a direct keyword for interfaces like java, but every class implicitly defines an interface. When a class implements multiple interfaces, and both interfaces declare a field with the same name, the situation is managed slightly differently compared to java.

What happens in dart when a class implements two interfaces that declare a field with the same name:

1) **Instance variable conflict:**
   If both interfaces declare an instance variable with the same name, Dart does not allow this ambiguity directly. You cannot have two instance variables with the same name in a class.
2) **Explicit implementation:**
   To resolve this, the implementing class must explicitly override the conflicting fields to ensure there is no ambiguity.

22) Can a subclass instance method override a superclass static method?

Ans.

No, in dart, a subclass instance method cannot override a superclass static method.

Static methods : static methods belong to the class itself, not to instances of the class rather than on an instance of the class.

Instance methods : instance methods are tied to instances of a class. They operate on the data contained in a specific object of the class.

23) Can a subclass static method hide a superclass instance method?

Ans.

no , in Dart, a subclass static method cannot hide a superclass instance method. The concept of"hiding" typically applies to static methods in both the superclass and the subclass, not between static and instance methods.

Static methods:
Belong to the class itself and are called on the class.

Instance methods:
Belong to instances of the class and are called on an object of the class.

24) Can a superclass access subclass member?

Ans.

In Dart, a superclass cannot directly access members that are defined only in its subclasses.

25) Difference between object oriented and object based language

Ans.

Object-Oriented Language:

1) **Class and objects :**
   supports the creation and use of classes and objects.

2) **Inheritance :**
   supports inheritance, allowing classes to inherit properties and methods from other classes.

3) **Polymorphism :**
   supports polymorphism, enabling a single function to handle different types of objects.

4) **Encapsulation :**
   supports encapsulation, allowing data and methods to be bundled within objects.

5) **Abstraction :**
   supports abstraction, enabling the creation of complex systems by modeling real-world entities and relationships.

Object-Oriented Language:

1) **Object :**
   Supports the creation and use of objects.

2) **No Inheritance :**
   Does Not support inheritance, Objects do not inherit properties or methods from other objects.

3) **Limited polymorphism** :
   Typically does not support Polymorphism.

4) **Encapsulation** :
   supports encapsulation to some extent, allowing data and methods to be bundled within objects.

5) **Limited Abstraction** :
   Provides less abstraction compared to object - oriented languages since it lacks features like inheritance and polymorphism.

—---------- THANK YOU —----------

REFERENCE:

1) Javatpoint
2) Geeksforgeeks
3) Dart.dev
4) Chat GPT