B1. What is inheritance?

Encapsulation is the process of hiding the implementation details of an object from the outside world and only exposing necessary information through public methods. This allows for better control over the access to an object's data and behavior and promotes data integrity, security, and simplicity in the code.

B2. Which inheritance is not supported by Dart? Why?

The reason why Dart does not support multiple inheritance is to avoid the complications that can arise from inheriting conflicting or ambiguous properties and methods from multiple sources. Instead, Dart supports a concept called mixins, which allow classes to reuse code from multiple sources without actually inheriting from multiple classes.

B3. What is advantage of inheritance?

Encapsulation: Inheritance enables encapsulation, which means that the implementation details of a class can be hidden from other classes. This reduces the complexity of code, since it makes it easier to focus on the interface or public methods of a class without worrying about the underlying implementation details.

Hierarchical organization: Inheritance allows for a hierarchical organization of classes, which can make code easier to understand and manage. This can help to organize and structure large codebases, making it easier to navigate and maintain over time.

B4. Difference between inheritance and encapsulation.

Inheritance is used to create a hierarchy of classes that share common properties and behaviors, while encapsulation is used to protect the internal state of an object from unwanted modification. Inheritance promotes code reuse and reduces redundancy, while encapsulation promotes data security and maintains code stability.

B5. Difference between inheritance and abstraction.

Inheritance and abstraction are both important concepts in object-oriented programming, but they serve different purposes.

Inheritance is a mechanism that allows a new class to be based on an existing class, inheriting its properties and methods. It enables code reuse, as you can create a new class that inherits the characteristics of an existing class and extends or modifies its behavior as needed. Inheritance establishes a hierarchical relationship between classes, with the child classes inheriting the properties and methods of their parent classes.

Abstraction, on the other hand, is a mechanism that allows you to represent complex systems in a simplified manner, by focusing on the essential features while ignoring the implementation details. It helps you to define a set of behaviors or properties that are common to a set of objects, and group them together in an abstract class or interface. Abstraction enables you to create modular and extensible code, by providing a clear separation between the interface and implementation.

In short, inheritance is about reusing and extending existing code, while abstraction is about creating simplified, high-level representations of complex systems.

B6. Difference between inheritance and polymorphism.

Inheritance and polymorphism are both important concepts in object-oriented programming, but they serve different purposes.

Inheritance is a mechanism that allows a new class to be based on an existing class, inheriting its properties and methods. It enables code reuse, as you can create a new class that inherits the characteristics of an existing class and extends or modifies its behavior as needed. Inheritance establishes a hierarchical relationship between classes, with the child classes inheriting the properties and methods of their parent classes.

Polymorphism, on the other hand, is a mechanism that allows objects of different classes to be treated as if they are of the same class, by providing a common interface. Polymorphism enables you to write code that can work with objects of different classes, as long as they implement the same interface or behave in the same way. This makes your code more flexible and reusable.

In short, inheritance is about reusing and extending existing code, while polymorphism is about writing code that can work with objects of different classes in a uniform way, by providing a common interface.

B7. Can we override static method in Dart?

No, we cannot override a static method in Dart. Static methods are associated with the class and not with an instance of the class, and they are not inherited by subclasses. Therefore, it is not possible to override a static method in the same way that instance methods can be overridden.

B8. Can we overload static method in Dart?

No, Dart does not support method overloading, including for static methods. In method overloading, you define multiple methods with the same name but with different parameters or argument lists. However, in Dart, every method, including static methods, must have a unique name. If you try to define multiple methods with the same name in a class, you will get a compile-time error. Therefore, you cannot overload a static method in Dart.

B9. Can a class implement more than one interface?

Yes, a class in Dart can implement more than one interface. This is known as multiple interface implementation. When a class implements multiple interfaces, it must provide implementations for all the methods declared in each of the interfaces it implements.

Multiple interface implementation allows you to define a class that has multiple behaviors, as it can inherit and implement features from different interfaces. This provides greater flexibility in creating complex and modular systems.

B10. Can a class extend more than one class in Dart?

No, Dart does not support multiple inheritance, which means a class cannot extend more than one class. However, a class can implement multiple interfaces.

B11. Can an interface extend more than one interface in Dart?

Yes, an interface can extend multiple interfaces in Dart using the extends keyword followed by a comma-separated list of interface names. This is known as interface inheritance or multiple interface inheritance.

B12. What will happen if a class implements two interfaces and they both have a method with same name and signature?

If a class implements two interfaces and they both have a method with the same name and signature, it will not cause any conflict or issue as the class is required to implement both the methods separately, and they will be treated as distinct methods with the same name and signature.

B13. Can we pass an object of a subclass to a method expecting an object of the super class?

Yes, we can pass an object of a subclass to a method expecting an object of the super class because the subclass inherits all the methods and attributes of the super class. This means that an object of the subclass is also an object of the super class and can be used wherever the super class object is expected. This is known as polymorphism in object-oriented programming.

B14. Are static members inherited to sub classes?

Yes, static members are inherited to sub classes. However, they are accessed using the class name rather than through an instance of the class.

B15. What happens if the parent and the child class have a field with same identifier?

If the parent and the child class have a field with the same identifier, then the child class's field will override the parent class's field. This means that when accessing the field in the child class, it will refer to the child class's field rather than the parent class's field. However, the parent class's field can still be accessed using the super keyword in the child class. This concept is known as field hiding or shadowing.

B16. Are constructors and initializers also inherited to sub classes?

Yes, constructors and initializers are inherited to sub classes. When a sub class is created, it automatically inherits all the members of the parent class, including the constructors and initializers. This means that a sub class can use the same constructors and initializers as its parent class, as well as define its own constructors and initializers if needed. However, it is important to note that the behavior of constructors and initializers in sub classes may be different from their behavior in the parent class, depending on how the sub class is designed.

B17. How do you restrict a member of a class from inheriting by its sub classes?

To restrict a member of a class from inheriting by its sub classes, the access modifier "private" can be used for that member. This will make the member only accessible within the class where it is defined, and it cannot be inherited by any sub classes.

For example, consider a class "Parent" with a private member variable "privateVar". If a sub class "Child" tries to inherit this private variable, it will result in a compilation error.

B18. How do you implement multiple inheritance in Dart?

Dart does not support multiple inheritance directly, but we can achieve it using interfaces. In Dart, an interface is a contract that a class can implement. We can create multiple interfaces and a class can implement multiple interfaces.

B19. Can a class extend by itself in Dart?

No, a class cannot extend itself in Dart. It would create a circular dependency and result in a compilation error. A class can, however, implement an interface that it defines, allowing it to provide its own implementation for the interface's methods.

B20. How do you override a private method in Dart?

It is not possible to override a private method in Dart as it is not accessible outside of the class it is declared in.

B21. When to overload a method in Dart and when to override it?

In Dart, you should overload a method when you want to provide multiple variations of the same method that accept different arguments. This is useful when you want to give more flexibility to the users of the class to choose the most appropriate method for their needs.

On the other hand, you should override a method in a subclass when you want to change the behavior of the method defined in the parent class. This is useful when you want to add specific functionality or change the logic of the method to fit the requirements of the subclass.

B22. What is the order of extends and implements keyword on Dart class declaration?

In Dart, the order of the 'extends' and 'implements' keywords on a class declaration matters. The 'extends' keyword should always come first, followed by the 'implements' keyword. This is because 'extends' is used to inherit properties and methods from a superclass, whereas 'implements' is used to specify that a class implements certain interfaces.

B23. How do you prevent overriding a Dart method without using the final modifier?

One way to prevent overriding a Dart method without using the final modifier is to make the method private by prefixing it with an underscore (\_). This will limit access to the method within the same class or library, making it less likely to be accidentally overridden by a subclass or external code

B24. What are the rules of method overriding in Dart?

1.The method in the subclass must have the same name as the method in the superclass.

2.The method in the subclass must have the same parameters as the method in the superclass.

3.The return type of the method in the subclass must be a subtype of the return type of the method in the superclass.

B25. Difference between method overriding and overloading in Dart.

Method overriding is when a subclass provides a different implementation of a method that is already defined in its superclass. Overloading, on the other hand, is when a class has multiple methods with the same name but different parameters. In Dart, method overriding is achieved using the @override annotation, while method overloading is not supported.

B26. What happens when a class implements two interfaces and both declare field (variable) with same name?

The class will have to provide an implementation for both fields with the same name, and it will need to specify which interface each field belongs to by using the interface name followed by a dot and the field name. This is necessary to avoid ambiguity and ensure that each field is used appropriately by the class and any other code that interacts with it.

B27. Can a subclass instance method override a superclass static method?

No, a subclass instance method cannot override a superclass static method. Instance methods belong to an instance of a class while static methods belong to the class itself. Therefore, they are separate and cannot be overridden by each other.

B28. Can a subclass static method hide superclass instance method?

No, a subclass static method cannot hide a superclass instance method.

B29. Can a superclass access subclass member?

No, a superclass cannot access subclass member directly.

B30. Difference between object oriented and object based language.

Object-oriented languages support encapsulation, inheritance, and polymorphism. Object-based languages do not support inheritance, which is one of the major features of object-oriented languages. Object-based languages include JavaScript, Python, and Ruby. Object-oriented languages include Java, C++, and Smalltalk.

B31. Explain Diamond problem.

The diamond problem occurs in object-oriented programming when a class inherits from two separate classes that share a common ancestor. This leads to ambiguity in the inheritance hierarchy and creates difficulty for the compiler in determining which method to use, causing errors and complications in the program.

B32. Why Dart does not support operator overloading?

Dart does not support operator overloading because it can make the code less readable and harder to understand, and it can also lead to unexpected behavior and errors. Instead, Dart provides a set of predefined operators that work with built-in types and classes, and it encourages the use of methods and functions for more complex operations.

B33. What is Encapsulation in Dart?

Encapsulation is the process of hiding implementation details from users of a class, and only exposing a limited set of methods and properties that can be used to interact with the object. In Dart, encapsulation is achieved by using private variables and methods, which can only be accessed within the same library or file, and cannot be accessed from outside the class. This helps to ensure that the internal state of the object is not accidentally or maliciously modified, and makes it easier to maintain and update the code without affecting other parts of the program.

B34. Which of the Dart OOPS feature promotes access protection or data hiding?

The Encapsulation feature in Dart OOPS promotes access protection and data hiding.