**B1. What is inheritance?**

**Ans -** Encapsulation is a fundamental principle of object-oriented programming, which refers to the practice of combining data and methods that operate on that data in a single unit, known as a class. It involves restricting access to an object's data and methods, thereby protecting it from external interference and ensuring that it is accessed only in a controlled way. Encapsulation is often described as "information hiding," as it allows the object's internal workings to be kept hidden from the user.

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

**Ans -** Dart does not support multiple inheritance. Multiple inheritance is a feature in some programming languages where a subclass can inherit properties and methods from more than one superclass.

**B3. What is advantage of inheritance?**

**Ans -** Polymorphism: Inheritance enables the concept of polymorphism, which means that objects of different classes can be treated as if they were of the same class. This allows for greater flexibility in code design and implementation, since it allows for more generalized code that can work with different types of objects.

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.

**B4. Difference between inheritance and encapsulation**.

**Ans -** Inheritance is a mechanism for building class hierarchies, while encapsulation is a mechanism for achieving data abstraction and information hiding. Inheritance allows subclasses to inherit properties and behaviors from a superclass, while encapsulation allows objects to expose a public interface for interacting with their internal state, while hiding the implementation details.

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

**Ans -** Inheritance is a mechanism in which one class inherits properties and behaviors from another class. The class that inherits is called the subclass or derived class, and the class that is inherited from is called the superclass or base class. The subclass can access all the public and protected members of the superclass, which can save time and reduce code duplication. Inheritance promotes code reuse, and it allows for the creation of new classes that are modified versions of existing classes.

Abstraction is a concept that helps manage complexity by focusing on essential features of an object or system while ignoring irrelevant details. Abstraction is achieved by creating abstract classes or interfaces that define a set of methods or properties that must be implemented by any class that uses them. This allows the programmer to work at a higher level of abstraction, which can make the code easier to understand and modify.

**B6. Difference between inheritance and polymorphism.**

**Ans -** Inheritance is a mechanism in which one class inherits properties and behaviors from another class. The class that inherits is called the subclass or derived class, and the class that is inherited from is called the superclass or base class. The subclass can access all the public and protected members of the superclass, which can save time and reduce code duplication. Inheritance promotes code reuse, and it allows for the creation of new classes that are modified versions of existing classes.

Polymorphism is the ability of an object to take on many forms. In object-oriented programming, polymorphism allows objects of different classes to be treated as if they are objects of the same class. This means that objects of a subclass can be used wherever objects of the superclass are expected. Polymorphism is often achieved through method overriding, which is when a subclass provides a different implementation of a method that is already defined in its superclass. This allows for flexible and extensible code, since new classes can be created that conform to an existing interface or contract.

**B7. Can we override static method in Dart?**

**Ans -** In Dart, static methods are bound to the class and not to an instance of the class. This means that the same method is shared among all instances of the class and cannot be overridden by a subclass. Static methods are typically used for utility functions or for performing tasks that do not depend on the state of an object. Therefore, there is no concept of overriding a static method in Dart. However, we can hide a static method in a subclass by defining a method with the same name in the subclass. When we call the method on an instance of the subclass, the method in the subclass will be called instead of the one in the superclass. But , this is not overriding as the method in the superclass is still available to be called from other instances of the class or other subclasses.

**B8. Can we overload static method in Dart?**

**Ans -** No, we cannot overload static methods in Dart. In Dart, methods are resolved at compile-time based on their name and the number and types of their parameters. Since static methods are associated with the class rather than an instance of the class, it is not possible to have multiple static methods with the same name in the same class with different parameters or return types.

**B9. Can a class implement more than one interface?**

**Ans -** Yes, a class can implement more than one interface in Dart. This is known as multiple interface inheritance. When a class implements multiple interfaces, it must provide implementations for all the methods defined in each interface.

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

**Ans -** No, Dart does not support multiple inheritance. A class can only extend a single class in Dart. However, a class can implement multiple interfaces, which allows it to define and adhere to multiple contracts or behaviors.

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

**Ans -** Yes, an interface can extend more than one interface in Dart using the "extends" keyword followed by a comma-separated list of interface names.

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

**Ans -** If the class implements two interfaces and they both have a method with the same name and signature, then the class will not compile unless it provides an implementation for the method that satisfies both interfaces. This is because the compiler needs to ensure that the class adheres to the contract specified by both interfaces, and having two methods with the same name and signature would violate this contract.

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

**Ans -** Yes, it is generally possible to pass an object of a subclass to a method that expects an object of the super class. This is because a subclass inherits all the attributes and methods of its super class, so it can be treated as an instance of its super class. However, if the method in the super class is expecting specific behavior or properties of the super class object that are not present in the subclass object, then it may result in unexpected behavior or errors.

**B14. Are static members inherited to sub classes?**

**Ans -** Yes, static members are inherited to sub classes.

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

**Ans -** If the parent and the child class have a field with the same identifier, it creates ambiguity in the program. The child class may override the field of the parent class or may have a separate field. The behavior depends on the programming language and the specific implementation. Generally, it is recommended to avoid such conflicts and ensure that each field has a unique identifier to maintain the code's clarity and avoid errors.

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

**Ans -** Yes, constructors and initializers are inherited to sub classes.

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

**Ans -** To restrict a member of a class from inheriting by its sub-classes, you can make that member private or use the final keyword to make the class or method final. Making a member private will prevent it from being accessible by any subclass. Similarly, making a class or method final will prevent it from being extended or overridden by any subclass.

**B18. How do you implement multiple inheritance in Dart?**

**Ans -** Dart does not support multiple inheritance, but you can use interfaces to achieve a similar result. To do this, you can define multiple interfaces, each with their own set of methods and properties. Then, you can implement these interfaces in your class to inherit the behavior from all of them. This approach is called interface-based multiple inheritance.

**B19. Can a class extend by itself in Dart?**

**Ans -** No, a class cannot extend by itself in Dart.

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

**Ans -** It is not possible to override a private method in Dart. Private methods are only accessible within the same class and cannot be accessed or overridden by any subclass.

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

**Ans -** Overloading a method in Dart is done when you want to define multiple methods with the same name but different parameters. This is useful when you want to provide different ways to call a method, for example, passing different types or numbers of arguments. Overriding a method in Dart is done when you want to provide a different implementation for a method that is already defined in a superclass. This is useful when you want to customize the behavior of a method for a subclass, without changing the method signature or interface. In short, you overload a method when you want to provide multiple ways to call it, and you override a method when you want to provide a different implementation for it.

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

**Ans -** The order of extends and implements keywords on Dart class declaration is "extends" followed by "with" and then "implements".

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

**Ans -** 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?**

**Ans -** --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.

**B25. Difference between method overriding and overloading in Dart.**

**Ans -** Method overloading in Dart is when you have multiple methods with the same name, but different parameters. The compiler determines which method to call based on the number and type of arguments passed to it. Method overriding, on the other hand, is when a subclass provides a different implementation of a method that is already defined in its superclass. The subclass method must have the same name, return type, and parameters as the superclass method.

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

**Ans -** If a class implements two interfaces and both declare a field with the same name, the class must implement the field and provide a concrete implementation for it. The field should have the same data type and should be declared with the same visibility modifier as in the interfaces. This is called field or variable shadowing, where one field shadows or hides the other. The class can access both fields using the interface names as a prefix.

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

**Ans -** No, a subclass instance method cannot override a superclass static method as they are two different types of methods with different functionalities and scopes.

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

**Ans -** No, a subclass static method cannot hide a superclass instance method.

**B29. Can a superclass access subclass member?**

**Ans -** No, a superclass cannot access subclass members. It is the subclass that can access and inherit members of the superclass.

**B30. Difference between object oriented and object based language**.

**Ans -** Object-oriented languages support all the features of object-oriented programming such as inheritance, encapsulation, and polymorphism. Examples include Java, C++, and Python.

Object-based languages only support some of the features of object-oriented programming such as encapsulation and polymorphism but do not support inheritance. Examples include JavaScript and Visual Basic.

**B31. Explain Diamond problem**.

**Ans -** The Diamond problem is an issue in multiple inheritance where two parent classes share a common method or attribute, and a child class that inherits from both parents calls the common method or attribute. This results in ambiguity and an error in the program. It is named the Diamond problem because the inheritance hierarchy forms a diamond shape. It can be resolved using virtual inheritance or other techniques like using an interface.

**B32. Why Dart does not support operator overloading?**

**Ans -** Dart does not support operator overloading because it can lead to confusion and potential errors in the code. Instead, Dart promotes the use of methods and functions to accomplish the same tasks as operator overloading. This approach ensures that code is clear, concise, and easily maintainable.

**B33. What is Encapsulation in Dart?**

**Ans -** Encapsulation in Dart is the practice of hiding the internal workings of a class from outside entities, and only exposing necessary functionality through well-defined interfaces. This improves code organization, maintainability, and security. It is achieved by using access modifiers such as public, private, and protected to control the visibility of class members.

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

**Ans -** sIn Dart, the feature that promotes access protection and data hiding is the concept of encapsulation through the use of classes and their associated access modifiers. The access modifiers are used to control the visibility and accessibility of class members, such as variables and methods, from outside of the class. By using access modifiers such as private or protected, developers can limit the visibility of data and implementation details, promoting encapsulation and data hiding.