**Assignment Module-2**

**1.What is inheritance?**

Inheritance is a class capability to drive properties and behaviors from another class Dart allows one class inheritance from another and allows it to generate a new class from an existing one. To do this we use extend keyword.

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

In Dart, multiple inheritance is not supported. Multiple inheritance refers to a programming language feature that allows a class to inherit from multiple classes simultaneously. The reason Dart does not support multiple inheritance is to avoid the "diamond problem."

The diamond problem occurs when a class inherits from two or more classes that have a common ancestor. This can lead to ambiguity when calling methods or accessing properties from the common ancestor, as the compiler may not know which version of the method or property to use.

To avoid this complexity and confusion, Dart uses single inheritance, where a class can only inherit from one superclass. However, Dart supports mixin-based composition, which allows classes to inherit behavior from multiple mixins. Mixins are a way to reuse code in multiple class hierarchies without actually inheriting from multiple base classes.

Advantages of inheritance:

1. Code Reusability: Inheritance allows a subclass to inherit properties and methods from its superclass. This promotes code reuse, as common functionality can be defined in a base class and inherited by multiple subclasses.

2. Modularity: Inheritance promotes a modular design approach. By defining a class hierarchy, you can separate different levels of functionality, making it easier to understand and maintain the codebase.

3. Polymorphism: Inheritance enables polymorphism, a fundamental object-oriented programming concept. Polymorphism allows objects of different classes to be treated as objects of a common superclass, which simplifies code and makes it more flexible.

4. Extensibility: It allows you to extend the functionality of existing classes without modifying their code directly. You can create new subclasses that inherit from a base class and add additional features or behaviors to suit specific requirements.

5. Method Overriding: Inheritance allows subclasses to override methods defined in their superclass. This allows for specialized behavior in subclasses while still retaining the common functionality from the superclass.

Overall, inheritance is a powerful tool in object-oriented programming that promotes code organization, reusability, and flexibility, but it should be used judiciously to avoid creating deep and complex class hierarchies that may lead to maintenance challenges.

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

Difference between Inheritance and Encapsulation:

1. \*\*Definition\*\*:

- Inheritance: Inheritance is an object-oriented programming concept that allows a class (subclass/derived class) to inherit properties and behaviors from another class (superclass/base class). The subclass can access and reuse the members of the superclass.

- Encapsulation: Encapsulation is another object-oriented programming concept that involves bundling the data (attributes) and methods (behaviors) that operate on the data within a single unit (i.e., a class). It is about hiding the internal details of an object and providing a well-defined interface to interact with it.

2. \*\*Purpose\*\*:

- Inheritance: The main purpose of inheritance is to promote code reusability and create a hierarchical relationship between classes. It allows a subclass to acquire the attributes and methods of its superclass, reducing code duplication.

- Encapsulation: The main purpose of encapsulation is to provide data protection and control access to the data and methods of a class. It helps in creating a clear separation between the implementation details and the public interface of the class.

3. \*\*Relationship\*\*:

- Inheritance: It establishes an "is-a" relationship between classes. For example, if class B inherits from class A, we can say "B is a subclass of A" or "B is a specialized version of A."

- Encapsulation: It establishes a "has-a" relationship between classes. For example, a class may have data members and methods, but it may also "have" or "encapsulate" other classes as part of its attributes.

4. \*\*Access Control\*\*:

- Inheritance: Subclasses can access the non-private members (attributes and methods) of their superclass. In some cases, the accessibility may depend on the access modifiers applied to the members in the superclass.

- Encapsulation: It involves access modifiers like public, private, protected, etc., to control the visibility of the class members. By setting appropriate access modifiers, you can hide sensitive data and provide a limited set of methods for interaction.

Difference between Inheritance and Abstraction:

1. \*\*Definition\*\*:

- Inheritance: As explained earlier, inheritance is about creating a relationship between classes where one class (subclass) inherits the properties and behaviors of another class (superclass).

- Abstraction: Abstraction is an object-oriented principle that focuses on representing the essential features of an object while hiding the unnecessary details. It involves defining the interface of an object without providing the implementation.

2. \*\*Level of Detail\*\*:

- Inheritance: Inheritance deals with the relationship between classes and how a subclass inherits from a superclass, gaining access to its members.

- Abstraction: Abstraction deals with the design of individual classes and how they present a simplified view of their functionality to the outside world, hiding the internal complexities.

3. \*\*Implementation\*\*:

- Inheritance: Inheritance is implemented using keywords like "extends" (for class inheritance) or "implements" (for interface inheritance) in object-oriented languages like Java, C++, and Dart.

- Abstraction: Abstraction is achieved through abstract classes and interfaces in many object-oriented languages. Abstract classes can have both concrete and abstract methods, whereas interfaces only have abstract method signatures.

4. \*\*Purpose\*\*:

- Inheritance: The primary purpose of inheritance is code reuse and building hierarchical relationships between classes to represent different levels of specialization.

- Abstraction: The main purpose of abstraction is to manage complexity by providing a simplified view of objects and their interactions, making the system easier to understand and maintain.

In summary, inheritance deals with the relationship between classes and code reuse, while encapsulation is about bundling data and methods within a class to provide data protection. Abstraction, on the other hand, is about simplifying the representation of objects to manage complexity and provide a clear interface for interaction.

**4. Difference between inheritance and polymorphism.**

In Dart, inheritance and polymorphism are both fundamental concepts in object-oriented programming (OOP). They are closely related but serve different purposes. Let's explore the differences between the two:

1. Inheritance:

Inheritance is a mechanism that allows a class (the subclass or derived class) to inherit the properties and behaviors (fields and methods) of another class (the superclass or base class). The subclass can extend the functionality of the superclass by adding new methods or overriding existing ones. The main purpose of inheritance is to promote code reuse and create a hierarchical relationship between classes.

2. Polymorphism:

Polymorphism is the ability of a class to take on multiple forms. It allows objects of different classes to be treated as objects of a common superclass. Polymorphism enables you to write code that can work with objects of various subclasses without knowing their specific types, making the code more flexible and reusable.

In Dart, polymorphism is achieved through method overriding. When a subclass overrides a method from its superclass, it allows objects of the subclass to be used wherever objects of the superclass are expected.

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

In Dart it is not possible Override a static method.

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

Dart does not support method overloading in the traditional sense for both static and instance methods. Method overloading is a feature that allows defining multiple methods with the same name but different parameter lists. In other programming languages like Java or C#, you can have multiple methods with the same name but different parameter types or numbers.

However, in Dart, only instance methods can be overloaded. This means you can't have multiple methods with the same name and different parameters, whether they are static or instance methods. If you try to define multiple methods with the same name and different parameter lists in a Dart class, you will get a compilation error.

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

Any class is allowed to implement one or more interfaces. The only difference is that the classes we are using as interfaces don’t have the abstract keyword so one() and two() must have a body( it’s an empty but still there.)

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

Dart has the ability to implement server interfaces. The keyword implements allows a class adhere to multiple interface and increase an objects polymorphic range. The keyword implement is followed by an existing name class whose public fields then become implementation requirements for the current class.

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

Interfaces only proscribe a method name and signature. If both interfaces have a method of exactly the same name and signature, the implementing class can implement the both interfaces method with a single concert method.

**10. 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?**

Yes, in Dart, you can pass an object of a subclass to a method expecting an object of the superclass. This concept is known as "polymorphism" and is a fundamental feature of object-oriented programming. Dart supports subtype polymorphism, which means that an object of a subclass can be treated as an object of its superclass.

Regarding static members, they are not inherited by subclasses in Dart. Static members belong to the class itself rather than individual instances of the class. Therefore, they cannot be overridden or inherited by subclasses. Each class, including subclasses, has its own separate copy of static members.

**11. 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?**

In Dart, if both the parent class and the child class have a field with the same identifier, the child class's field will override the parent class's field. This behaviour is known as "field hiding."

Regarding constructors and initializers, they are not inherited in Dart. However, constructors in the child class can call the constructors of the parent class using the `super` keyword. This allows you to reuse the parent class's constructors while also initializing the child class's specific fields.

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

We can restrict a member of a classes from inheriting to its sub classes by declaring the member as a private. Because , private members are not inherited to sub classes.

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

No, Dart does not support multiple implementation inheritance. Dart has interfaces and like most other similar language it has multiple interface inheritance. For implementation there is only a single-super-class chain that a class can inherit number implementations from.

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

No, a class cannot extend itself in Dart or any other object-oriented programming language. Inheritance is a relationship between two different classes where one class (the subclass or derived class) inherits properties and behaviours from another class (the superclass or base class).

However, a class cannot extend itself, as it would lead to a circular dependency and infinite recursion, which is not allowed in object-oriented programming.

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

1) Copy the package folder in your current code and change it as per your need. 2) Create a public method in library class and pass that private method in it. You can have the access of that private method as defined public method.

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

In Dart, overloading and overriding are two important concepts related to class methods.

**1. Overloading:**

Overloading refers to the ability to define multiple methods with the same name but different parameter lists within the same class. These methods can have different numbers or types of parameters, providing flexibility and ease of use to the users of the class. Dart does not support method overloading in the traditional sense (like some other programming languages do), where you can define multiple methods with the same name and different parameter lists. Instead, you can use optional parameters and named parameters to achieve similar functionality.

**2. Overriding:**

Overriding refers to the ability of a subclass to provide a specific implementation for a method that is already defined in its superclass. When you override a method, the subclass replaces the implementation of the superclass's method with its own implementation. This allows you to customize the behaviour of the method for the specific subclass while still maintaining a common interface across related classes.

In summary, use method overloading (using optional parameters or named parameters) in Dart when you want to provide different ways of calling the same method with different parameter lists. Use method overriding when you want to provide a specialized implementation of a method in a subclass, replacing the implementation from the superclass.

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

In Dart, when declaring a class that extends another class and implements one or more interfaces, the order is as follows:

```dart

class MyClass extends SuperClass implements Interface1, Interface2 {

// Class body

}

```

Here's a breakdown of the order:

**1. `extends**`: This keyword is used to specify that the class `MyClass` is inheriting from the `SuperClass`. In Dart, a class can extend only one superclass. It establishes an "is-a" relationship, where `MyClass` is a more specialized version of `SuperClass`.

**2. `implements`**: This keyword is used to specify that the class `MyClass` is implementing one or more interfaces (`Interface1` and `Interface2` in this case). Dart allows a class to implement multiple interfaces, enabling it to inherit behavior from multiple sources. An interface defines a contract that the implementing class must adhere to.

So, the order is always `extends` followed by `implements`. However, keep in mind that you are not required to use `implements` if the class doesn't implement any interface. In that case, you can have just the `extends` keyword for class inheritance.

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

In Dart, the `final` modifier is commonly used to prevent method overriding. However, if you want to achieve the same effect without using the `final` modifier, you can follow an alternative approach by leveraging abstract classes. Abstract classes are classes that cannot be instantiated directly and can have abstract methods, which are methods without an implementation.

Here's how you can prevent method overriding without using the `final` modifier:

**1. Create an abstract class**: Define your base class as an abstract class using the `abstract` keyword. This prevents direct instantiation of the class.

**2. Declare the method as abstract**: Within the abstract class, declare the method that you want to prevent from being overridden as an abstract method. Abstract methods are declared using the `abstract` keyword, and they have no implementation in the abstract class.

**3. Provide a default implementation (optional):** If you want to provide a default implementation for the method that may be overridden by subclasses, you can provide a non-abstract method in the abstract class. Subclasses that don't want to override the method can use this default implementation, while subclasses that need a different implementation must override the method.

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

**Rules of method overriding**

The method which we want to override can only be written in the subclass not in the same class.

The return type should be the same as in the method present in the super class.

The argument list should be the same as in the method present in the super class.

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

The most basic difference here is that overloading is achieved in the same class whereas overriding requires a parent and a child class at the minimum. Another key concept to keep in mind is that overloaded methods are bound at compile time to the method calls.

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

If a type implements two interfaces, and each interface define a method that has identical signature, then in effect there is only one method, and they are not distinguishable. If, say, the two methods have conflicting return types, then it will be a compilation error.

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

Important Points about method overriding and static methods

An instance method cannot override a static method, and a static method cannot hide an instance method. For example, the following program has two compiler errors. In a subclass (or Derived Class), we can overload the methods inherited from the superclass.

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

Important Points about method overriding and static methods

An instance method cannot override a static method, and a static method cannot hide an instance method. For example, the following program has two compiler errors. In a subclass (or Derived Class), we can overload the methods inherited from the superclass.

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

No superclass has no knowledge of its subclasses.

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

Object-oriented languages do not have the inbuilt objects whereas Object-based languages have the inbuilt objects, for example, JavaScript has window object. Examples of object-oriented programming are Java, C#, Smalltalk, etc. whereas the examples of object-based languages are JavaScript, VBScript, etc.