FLUTTER ASSIGNMENT

MODULE 2

21. What is inheritance?

Ans.

* Inheritance : When One Object Acquire All The Properties and Behaviour of Parent class That is called an Inheritance.

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

Ans.

* Dart does not support multiple inheritance, but you can use mixins to achieve a similar effect.
* Multiple Inheritance: This inheritance occurs when a class inherits more than one parent class. Dart doesn't support this.
* 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

23. Difference between inheritance and encapsulation. Difference between inheritance and abstraction.

Ans.

* Encapsulation allows us to hide implementation details and provide controlled access to the data and behavior of an object. Inheritance facilitates code reuse and supports hierarchical relationships between classes.
* These are two different concept and selections are based on the requirements. Abstraction hide the implementation details and only show the functionality. It reduce the codecomplexity. Inheritance create a class using a properties of another class.

24. Difference between inheritance and polymorphism

* Definition: Inheritance is a mechanism by which one class (the child or subclass) can inherit properties and methods from another class (the parent or superclass). This allows the subclass to reuse code from the superclass and extend or modify it as needed.
* Purpose: It enables one interface to be used for a general class of actions, making it easier to manage and scale code.
* **Definition**: Inheritance is a mechanism by which one class (the child or subclass) can inherit properties and methods from another class (the parent or superclass). This allows the subclass to reuse code from the superclass and extend or modify it as needed
* **Purpose**: It promotes code reusability and establishes a hierarchical relationship between classes.

25. Can we override static method in Dart?

No, you cannot override static methods in Dart. In Dart, static methods belong to the class itself rather than to any instance of the class. Since static methods are associated with the class, not with instances, they cannot be overridden in the same way that instance methods can.

However, you can hide static methods using the static keyword. This involves defining a static method with the same name in a subclass, but this does not override the method in the parent class. Instead, it hides it. When you call the static method from the subclass, it will use the method defined in the subclass, but the method in the parent class is still accessible using the class name.

26. Can we overload static method in Dart?

No, Dart does not support method overloading, including for static methods. In Dart, method overloading—where you have multiple methods with the same name but different parameters—is not allowed. Each method in a class must have a unique name.

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

Yes, a class in Dart can implement multiple interfaces. Dart uses a concept called "implements" for interfaces, and a class can implement multiple interfaces by separating them with commas.

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

Conflict Resolution: The class provides a single implementation for the method that satisfies the contract of both interfaces. The implementation of this method in the class will override the method definitions from both interfaces.

Interface Method Contracts: The class must adhere to the contract specified by both interfaces for that method. As long as the method implementation in the class conforms to the specifications (e.g., return type, parameters), the class will correctly fulfill the requirements of both interfaces.

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?

Object-Oriented Programming (OOP) Theory

1. Inheritance

Inheritance is a fundamental concept in OOP where a class (known as a subclass or child class) inherits properties and behaviors (methods) from another class (known as a superclass or parent class). This allows for code reuse and the creation of hierarchical relationships between classes.

* Single Inheritance: A class inherits from a single superclass.
* Multiple Inheritance: A class can inherit from multiple superclasses. (Not supported directly in Dart but can be achieved through interfaces or mixins.)

2. Field Shadowing

When a child class declares a field with the same name as a field in its parent class, the child’s field shadows the parent’s field. This means that:

* Access in Child Class: References to the field in the child class will use the child’s field.
* Access in Parent Class: The parent’s field can still be accessed using the super keyword (or equivalent) in many languages.

3. Constructors

Constructors are special methods used to initialize objects. They are not inherited but can be called using super() to initialize the parent class.

* Default Constructor: Provided by default if no constructors are explicitly defined.
* Parameterized Constructor: Takes arguments to initialize the object with specific values.
* Named Constructor: Allows multiple constructors to be defined with different names for clarity or different initialization logic.

4. Initialization Lists

In languages like Dart, initialization lists allow you to set up initial values for fields before the constructor body runs. This is particularly useful for initializing final fields or fields that depend on the constructor parameters.

5. Instance vs. Static Initializers

* Instance Initializers: Run each time an instance of the class is created. They are used to initialize non-static fields.
* Static Initializers: Run once when the class is first loaded. They are used to initialize static fields or perform one-time setup.

6. Polymorphism

Polymorphism allows objects to be treated as instances of their parent class rather than their actual class. This includes:

* Method Overriding: A subclass provides a specific implementation for a method that is already defined in its superclass.
* Method Overloading: Defining multiple methods with the same name but different parameters. (Not supported directly in Dart, but similar concepts can be achieved through named parameters.)

7. Encapsulation

Encapsulation refers to the bundling of data (fields) and methods that operate on the data into a single unit (class). It also involves restricting direct access to some of the object’s components, which is typically achieved using access modifiers like private, protected, and public.

8. Abstraction

Abstraction means hiding the complex implementation details and showing only the necessary features of an object. This is usually achieved through abstract classes or interfaces, which define methods that must be implemented by subclasses.

34. Can a class extend by itself in Dart?

No, in Dart, a class cannot extend itself. This would create a circular dependency and is not allowed. Each class can only extend one other class (or implement one or more interfaces) and must be a distinct class.

If you need to reuse functionality, you might consider using mixins or composition instead. Mixins allow you to define methods and properties that can be reused across multiple classes, while composition involves creating classes that use instances of other classes to achieve the desired behavior.

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

**Overloading**

Method overloading allows you to define multiple methods with the same name but different parameters within the same class. Dart does not support method overloading directly. Instead, you can achieve similar functionality by using named parameters or optional parameters.

When to use:

* When you want to provide multiple ways to call a method with different parameters.
* To offer flexibility in method signatures while keeping method names intuitive.

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

Same Method Name and Signature: The method in the subclass must have the same name and parameters as the method in the superclass.

@override Annotation: While not mandatory, it is recommended to use the @override annotation before the method in the subclass. This helps to ensure that you are actually overriding a method from the superclass and not accidentally creating a new method.

Access Level: The access level of the overridden method in the subclass should be the same or more permissive than in the superclass. For example, if the superclass method is public, the overriding method in the subclass must also be public or protected.

Return Type: The return type of the overriding method must be the same as the method in the superclass, or a subtype of the superclass method’s return type. This is known as covariant return types.

Inheritance: Method overriding can only be done if the class is inheriting from another class. If you are not extending a class, you are not overriding any methods.

Super Keyword: You can call the superclass method from the overridden method using the super keyword, allowing you to build on the functionality of the superclass method.

40. Difference between method overriding and overloading in Dart.

Method Overriding

Concept:

* Inheritance: Method overriding is a key concept in inheritance, where a subclass inherits methods from a superclass.
* Polymorphism: It enables polymorphism, where the method that gets called is determined at runtime based on the object's actual type.
* Purpose: Allows a subclass to provide a specific implementation of a method that is already defined in its superclass, tailoring the method's behavior to the subclass's needs.

How It Works:

* The subclass method must have the same name, return type, and parameters as the superclass method.
* The @override annotation in Dart (and similar annotations in other languages) indicates that the method is overriding a superclass method.
* At runtime, when the method is called on an object of the subclass, the overridden method in the subclass is executed.

Method Overloading

Concept:

* Single Class: Method overloading occurs within a single class, where multiple methods share the same name but differ in their parameter lists.
* Compile-time Polymorphism: It allows a class to have more than one method with the same name, distinguished by the number or types of their parameters.
* Purpose: Provides flexibility in method usage and can simplify code by allowing methods to handle different types or numbers of inputs.

How It Works:

* The method name is the same, but the parameter lists must differ. This can include varying the number of parameters, types of parameters, or both.
* The compiler determines which method to invoke based on the method's signature at compile-time.
* Since Dart doesn’t support method overloading directly, it uses optional or named parameters to achieve similar functionality.

Summary

* Method Overriding: Allows subclasses to provide a specific implementation for methods inherited from a superclass. It is crucial for polymorphism and runtime behavior modification.
* Method Overloading: Allows multiple methods with the same name but different parameters within the same class, providing compile-time flexibility. Dart achieves similar behavior using optional and named parameters.

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

Dart Interfaces and Fields: In Dart, interfaces are represented by classes and abstract classes. Interfaces in Dart do not support fields with different values in the same way that Java interfaces do. Dart's classes (including abstract classes) can have fields, and when you implement multiple classes (or abstract classes), Dart requires you to handle any conflicts.

Field Conflicts: If you try to implement two classes (or abstract classes) with fields of the same name, you'll face issues, as Dart will not allow field name conflicts. Fields in Dart are not inherited in the same way constants are in Java interfaces. Instead, if a field is declared in multiple inherited classes or mixins, Dart will not allow you to declare them with the same name in a subclass.

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

1. Static Methods: These belong to the class, not to instances of the class. They can be called using the class name and do not participate in dynamic dispatch. Static methods in a subclass can hide static methods in a superclass if they have the same name and parameter list, but they don’t override instance methods.
2. Instance Methods: These are associated with instances of the class. They can be overridden by subclass instance methods if they have the same signature (name and parameters).

In summary:

* A subclass can hide a superclass static method with another static method of the same name.
* A subclass instance method can override a superclass instance method if it has the same signature.

45. Difference between object oriented and object based language.

Object-Oriented Languages

1. Definition: Object-oriented languages are designed with the principles of object-oriented programming (OOP) in mind. They fully support the four main OOP concepts: encapsulation, inheritance, polymorphism, and abstraction.
2. Key Features:
   * Encapsulation: Bundling of data and methods that operate on the data within a class.
   * Inheritance: Ability to create new classes that reuse, extend, and modify the behavior defined in other classes.
   * Polymorphism: The ability for different classes to be treated as instances of the same class through a common interface.
   * Abstraction: Hiding complex implementation details and showing only the essential features of an object.
3. Examples: Languages like Java, C++, Python, and C# are considered object-oriented because they fully support these principles and encourage their use.

Object-Based Languages

1. Definition: Object-based languages support some aspects of object-oriented programming but do not necessarily support all OOP principles, particularly inheritance and polymorphism.
2. Key Features:
   * Encapsulation: Often supported, allowing the creation of objects with their own data and methods.
   * Inheritance: Not always supported. Object-based languages may not allow one object to inherit properties and methods from another.
   * Polymorphism: May not be supported. There may be limited or no support for methods to have multiple forms.
   * Abstraction: Typically supported.
3. Examples: JavaScript and VBScript are examples of object-based languages. They allow the creation and manipulation of objects but do not fully support inheritance in the way that classic object-oriented languages do.

In summary, object-oriented languages provide a full implementation of OOP principles, while object-based languages support some of these principles but may lack full support for inheritance and polymorphism.