Module – 2

Flutter – Dart Programming

Q. Which inheritance is not supported by Dart?

Ans. Dart does not support multiple inheritance. This is because multiple inheritance can lead to a number of problems, such as the diamond problem. The diamond problem occurs when two parent classes of a class have a common method, and that method is called by the child class. In such cases, it becomes unclear which parent class's method should be called. To avoid such issues, Dart promotes the use of mixins, which allow a class to inherit functionality from multiple sources without creating a hierarchy of classes.

Q. Difference between inheritance and encapsulation.

Ans.

Inheritance	Encapsulation
- Inheritance is a mechanism that allows you to define a new class based on an existing class.	- Encapsulation is a mechanism that allows you to hide the implementation details of a class from the outside world.
- The new class inherits all the properties and methods of the existing class.	- This is achieved by making the properties and methods of the class private, so that they can only be accessed from within the class.
- Inheritance promotes code reuse and helps you organize your code into a hierarchy of related classes.	- Encapsulation helps you to create more robust and secure code, by preventing external code from accessing or modifying the internal state of your objects.

Q. Difference between inheritance and polymorphism

Ans.

Inheritance	**Polymorphism**
- Inheritance is a mechanism that allows you to define a new class based on an existing class.	- Polymorphism is a mechanism that allows you to use the same interface to represent different types of objects.
- The new class inherits all the properties and methods of the existing class.	- This means that you can write code that works with a variety of different objects, without having to know the specific details of each object.
- Inheritance promotes code reuse and helps you organize your code into a hierarchy of related classes.	- Polymorphism is typically achieved through inheritance, by defining a common interface in a superclass and then implementing that interface in a variety of subclasses.

Q. How do you implement multiple inheritance in Dart?

Ans. Dart does not support multiple inheritance. However, Dart provides an alternative to multiple inheritance called mixins.

A mixin is a way to reuse a class's code in multiple class hierarchies. we can define a mixin as a class that provides a set of methods and properties that can be added to other classes without the need to extend them.

Here's an example:

```
class Animal {
  void move() {
    print('Animal is moving.');
  }
}
mixin Flyer {
  void fly() {
    print('Flyer is flying.');
  }
}
class Bird extends Animal with Flyer {
  // Bird now has both move() and fly() methods
}
void main() {
  var bird = Bird();
  bird.move(); // Output: Animal is moving.
  bird.fly(); // Output: Flyer is flying.
}
```

In this example, `Animal` is a base class that defines the `move` method. The `Flyer` mixin defines the `fly` method. The `Bird` class extends `Animal` and includes the `Flyer` mixin using the `with` keyword.

When we create an instance of `Bird`, we can call both the `move` and `fly` methods, even though they are defined in different parts of the class hierarchy.

Mixins can be a powerful tool for creating reusable code and adding functionality to existing classes. However, like multiple inheritance in general, they can also be complex and difficult to use in some cases.

Q. How do you restrict a member of a class from inheriting by its subclasses?

Ans. In Dart, you can prevent a subclass from inheriting a member by marking that member as `final` or `private`.

If you mark a member as `final`, it cannot be overridden by any subclass. Here's an example:

```
class BaseClass {
  final int value = 10;
}
class SubClass extends BaseClass {
  // This will cause a compile-time error:
  // 'value' can't be overridden because it's final.
  // int value = 20;
}
```

In this example, `BaseClass` has a `final` member called `value`. When you try to override `value` in `SubClass`, you get a compile-time error because `value` is marked as `final`.

If you mark a member as `private`, it cannot be accessed by any subclass. Here's an example:

```
class BaseClass {
  int _value = 10;
}

class SubClass extends BaseClass {
  // This will cause a compile-time error:
  // '_value' is not defined for the subclass.
  // int value = 20;
}
```

In this example, `BaseClass` has a private member called `_value`. When you try to access `_value` in `SubClass`, you get a compile-time error because `_value` is marked as private.

By using `final` or `private`, you can prevent subclasses from inheriting certain members of a class.