OOP Concepts in Dart – Mastering Dart Summary

Polymorphism

Polymorphism allows objects of different classes to be treated through the same interface, with each class providing its own implementation.

Key Points:
- Use of @override to redefine methods in child classes.
- Enables flexible and reusable code structures.
- Promotes abstraction by allowing base class references to access child class behavior.

Example:
```
class Animal {
  void makeSound() => print("Some sound");
}
class Dog extends Animal {
  @override
  void makeSound() => print("Bark");
}
```

Static Variables

Static variables are shared across all instances of a class. They belong to the class itself rather than individual objects.

Key Points:
- Declared using the static keyword.
- Useful for storing class-level data like counters, constants, or configurations.
- Accessed directly via the class name.

Example:

```
class Counter {
  static int count = 0;

  Counter() {
    count++;
  }
}
```

Static Functions

Static functions are methods that can be accessed without creating an instance of the class.

Key Points:

- Cannot access instance variables or methods (this is not allowed).
- Ideal for utility or helper functions.

Example:

```
class MathUtils {
  static int square(int x) => x * x;
}
```

Enum

Enums are used to define a collection of named constant values.

Key Points:

- Helps avoid magic strings and enhances code readability.
- Enums in Dart are objects and can have properties and methods.

Example:

```
enum Status { loading, success, error }
```

Abstract Class

An abstract class defines a base structure that cannot be instantiated directly. It provides a blueprint for subclasses.

Key Points:

- Includes abstract methods (no body) and concrete methods.
- Forces child classes to implement abstract methods.

Example:

```
abstract class Shape {
 void draw();
}
class Circle extends Shape {
 @override
 void draw() => print("Drawing circle");
}
```

Implements and Interface

Using implements, a class agrees to implement all properties and methods defined by another class or interface.

Key Points:

- Dart does not have a separate interface keyword — every class can act as an interface.
- Multiple interfaces can be implemented.

Example:

```
class Printable {
 void printData();
}
class Report implements Printable {
 @override
 void printData() => print("Printing report...");
}
```

Mixins (with Keyword)

Mixins allow code reuse in multiple classes without traditional inheritance.

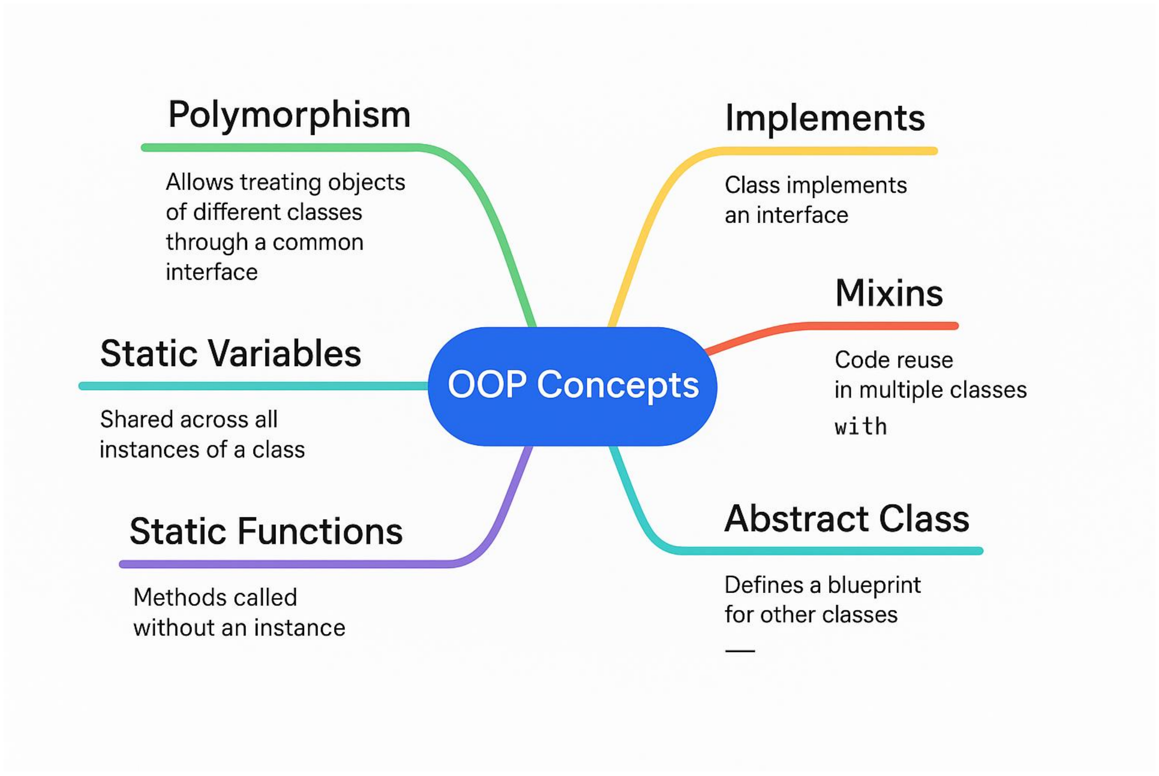
Key Points:

- Declared using mixin or regular classes.
- Used with with keyword.
- Cannot have constructors.

Example:

```
mixin CanFly {
  void fly() => print("Flying...");
}

class Bird with CanFly {}
```



OOP Concepts
Polymorphism
Allows treating objects of different classes through a common interface
Static Variables
Shared across all instances of a class
Static Functions
Methods called without an instance
Implements
Class implements an interface
Mixins
Code reuse in multiple classes with
Abstract Class
Defines a blueprint for other classes