# OOP concepts in Dart

- Object-Oriented Programming (OOP) in Dart is a programming paradigm that structures code around objects, combining data and methods to model real-world entities.
- Dart, a language developed by Google, supports key OOP concepts such as classes, objects, inheritance, polymorphism, and encapsulation.
- Classes act as blueprints for objects, enabling code reuse and modularity. Inheritance allows the creation of hierarchies to share characteristics among classes, while polymorphism permits the same method to be applied to different objects.
- Encapsulation hides implementation details and exposes only necessary functionalities.
- OOP in Dart fosters organized and maintainable code, enhancing software development efficiency.

## Four Pillars of OOPs in Dart

- **Encapsulation:** This means that the internal representation of an object is hidden from the outside. In Dart, you can create private fields by prefixing an underscore to the name, and you can use getters and setters to control access to those fields. This way, you can control how data within an object is accessed or modified.

  Encapsulation refers to the bundling of data and the methods that operate on that data, restricting the direct access to some of an object's components. This means that the internal representation of an object is hidden from the outside, and access to it is provided only through specific methods, properties, or functions.

Example:

```
class MyClass {
  int _age; // Private field

  int get age => _age; // Getter

  set age(int value) => _age = value; // Setter
}
```

**Getters and Setters:** Getters and setters are used to access and modify the private fields of a class, respectively. They provide controlled access to the fields.

- **Inheritance:** Inheritance allows a class to be defined that has a certain set of characteristics (fields, methods) and then other classes to be created that are derived from that class. The derived class inherits the characteristics of the base class and typically adds to or modifies them. Dart supports single inheritance, meaning that a class can only inherit from one parent class.

  Inheritance is a key concept in object-oriented programming, allowing for code reusability, and Dart is no exception. In Dart, inheritance is used to create a new class that is a modified version of an existing class. The new class inherits properties and behavior (data members and methods) from the existing class.
  - **Base Class (Parent Class):** The class whose properties and functions are inherited by another class.
  - **Derived Class (Child Class):** The class that inherits the properties and functionality from another class

- **Derived Class (Child Class):** The class that inherits the properties and functionality from another class.
- **'extends' Keyword:** In Dart, you use the extends keyword to create a class that inherits from another class.
- **'override' Keyword:** If you need to redefine a method in a derived class, you can use the override keyword.
- ***Calling the Superclass Constructor:*** You can call the parent class's constructor using the super keyword.

Example:

```
class Animal {
  void eat() {
    print('The animal eats');
  }
}

class Dog extends Animal {
  @override
  void eat() {
    print('The dog eats');
  }

  void bark() {
    print('The dog barks');
  }
}
```

```
void main() {
  Dog myDog = Dog();
  myDog.eat(); // Outputs 'The dog eats'
  myDog.bark(); // Outputs 'The dog barks'
}
```

- **Polymorphism:** Polymorphism allows objects of different classes to be treated as objects of a common superclass. There are two types of polymorphism: compile-time (or static) and runtime (or dynamic). In Dart, you can make use of polymorphism through method overriding, where a method in a subclass can provide a different implementation for a method that is already defined in its superclass.

  Polymorphism in Dart refers to the ability of a class or data type to take on multiple forms or behave differently depending on the context.

  Example:

```
class Animal {
  void makeSound() {
    print("Animal makes a generic sound");
  }
}

class Dog extends Animal {
  @override
  void makeSound() {
```

```
    print("Dog barks");
  }
}

class Cat extends Animal {
  @override
  void makeSound() {
    print("Cat meows");
  }
```

```
}

void main() {
  Animal animal1 = Dog();
  Animal animal2 = Cat();

  animal1.makeSound(); // Output: "Dog barks"
  animal2.makeSound(); // Output: "Cat meows"
}
```

- **Abstraction:** Abstraction means dealing with the level of detail that is most appropriate to a task, and it often involves creating abstract classes that can't be instantiated on their own. These abstract classes are meant to be subclassed by other classes that provide concrete implementations of the abstract methods. Dart provides the abstract keyword to define abstract classes and methods.

Example:

```
// Abstraction: Shape is an abstract class.
abstract class Shape {
  // Encapsulation: area and perimeter are accessible only through
  double get area;
  double get perimeter;
}

// Inheritance: Rectangle inherits from Shape.
class Rectangle extends Shape {
  final double width;
  final double height;

  Rectangle(this.width, this.height);

  // Polymorphism: area and perimeter are implemented for Rectangle
  @override
  double get area => width * height;
```

```
  @override
  double get perimeter => 2 * (width + height);
```

```
}

void main() {
  Shape shape = Rectangle(5, 10);
  print('Area: ${shape.area}');
  print('Perimeter: ${shape.perimeter}');
}
```

This code defines an abstract Shape class and a Rectangle class that inherits from Shape. The Shape class encapsulates the details of how area and perimeter are calculated, and it provides an abstract interface for these properties. The Rectangle class provides a concrete implementation of the Shape interface, thus demonstrating polymorphism.

**Static variables** in Dart refer to the variables that are declared inside a class using the Static keyword.

Instead of a specific instance, these are members of the class. For all instances of the class, static variables are regarded the same. This means that a single copy of the static variable is shared by all instances of the class. It only allocates memory once, when classes are loading, and then uses it throughout the program.

**The following are the key points to know about static variables:**

- A **class variable** is also called a static variable.
- A **single copy** of the static variable is shared by all of a class's instances.
- The class name can be used to access static variables. We *don't* need to create an object from the class they belong to.
- Static variables can be easily accessed in static methods.

Example:

```
class Student {
  String roll_no;
  String level;

  // Declaring a static variable
  static var department;

  // Function to print details of the student
  displayDetails() {
    print("The student with roll number ${roll_no} is i
    print("The student studies ${department}");
  }
}

void main() {
  Student st1 = Student();

  // Accessing the static variable
  Student.department = "Computer Sci";

  // Assigning value to variables using the class objec
  st1.roll_no = '1238AS23';
```

```
    st1.level = "4001";

    // Invokeing the method displayDetails
    st1.displayDetails();
}
```