- Read the following articles and summarize the key points:

    - **Dart Programming – Classes**

1. Class: A blueprint for creating objects, containing attributes and methods.

2. Object: An instance of a class, created using the new keyword (optional in Dart).

3. Constructors: Special methods used to initialize objects, which can be default, parameterized, or named.

4. Inheritance: Allows a class to inherit properties and methods from another class using the extends keyword.

5. Method Overriding: A subclass can redefine a parent class's methods using the @override annotation.

6. Interfaces: Used to enforce specific methods in a class for consistent implementation.

7. Dart provides robust tools for building organized, reusable, and maintainable code through classes.

    - **OOP in Flutter**

1. Understanding OOP: Object-Oriented Programming involves creating objects with attributes (data) and methods (behavior) to organize code flexibly and reusable.

2. Importance of OOP in Flutter: OOP simplifies building complex apps using reusable components, making the code more structured and maintainable.

3. Four Principles of OOP: Abstraction, Encapsulation, Inheritance, and Polymorphism are key to designing OOP-based code.

4. Abstraction in Flutter: Focuses on hiding complex details and highlighting essential features using Abstract Classes and Interfaces.

5. Encapsulation: Hides object data using private properties and controls access through methods.

6. Inheritance: Allows reusing code by inheriting classes, reducing redundancy and enhancing efficiency.

7. Polymorphism: Enables implementing the same function in different ways, increasing flexibility when working with objects.

8. Widgets as Objects: In Flutter, widgets are objects, showcasing the importance of OOP in designing user interfaces.

9. State Management: Relies on OOP principles to organize and handle dynamic data in applications.

10. Practical Application: The article provides practical examples of using OOP in Flutter, like creating custom classes and managing inheritance and encapsulation for better code structure.