Technical College of Zakho

Computer Information Systems

Mobile Application Development I

2. Class and object

Lecturer:

Sipan M. Hameed

www.sipan.dev

2025-2026

Contents

2.pure Dart classes and Flutter-specific classes (Widgets)

2.1. 1️⃣ What is a Class in Dart?

A **class** is a blueprint for creating objects.

It groups **data (fields)** and **behavior (methods)** together.

Think:

Class = Design

Object = Real thing created from that design

🔹 Basic Dart Class

```
class Person {
  String name;
  int age;

  void introduce() {
    print('Hi, my name is $name and I am $age years old.');
  }
}
```

- **Create an object**

```
void main() {
  Person p = Person();
  p.name = 'Sipan';
  p.age = 22;

  p.introduce();
}
```

📌 Output:

```
Hi, my name is Sipan and I am 22 years old.
```

2.2. 2️⃣ Constructors in Dart Classes

- 🔹 **Default Constructor**

```
class Person {
  String name;
  int age;

  Person(this.name, this.age);
}
void main() {
  Person p = Person('Sipan', 22);
}
```

✓ this.name automatically assigns values

- 🔹 **Named Constructors**

```
Used when you want multiple ways to create objects
class Person {
  String name;
  int age;

  Person(this.name, this.age);

  Person.child(this.name) : age = 0;
}
void main() {
  Person baby = Person.child('Ali');
}
```

2.3. 3️⃣ final vs const in Classes

- 🔹 **final**

Value assigned **once**, at runtime

- 🔹 **const**

Compile-time constant

```
class Car {
  final String model;
  final int year;

  Car(this.model, this.year);
}
```

2.4. 4️⃣ Private Fields & Methods (Encapsulation)

In Dart, **underscore _ means private to file**

```
class BankAccount {
  double _balance = 0;

  void deposit(double amount) {
    _balance += amount;
  }

  double get balance => _balance;
}
void main() {
  var acc = BankAccount();
  acc.deposit(100);
  print(acc.balance); // ✓ allowed
  // acc._balance ❌ not allowed
}
```

2.5. 5 Getters and Setters

```
class Temperature {
  double _celsius = 0;

  double get fahrenheit => _celsius * 9 / 5 + 32;

  set celsius(double value) {
    if (value >= -273.15) {
      _celsius = value;
    }
  }
}
```

2.6. 6 Inheritance (extends)

```
class Animal {
  void sound() {
    print('Animal sound');
  }
}

class Dog extends Animal {
  @override
  void sound() {
    print('Bark');
  }
}
void main() {
  Dog d = Dog();
  d.sound(); // Bark
}
```

2.7. 7 Abstract Classes

Used when a class **must be implemented**

```
abstract class Shape {
  double area();
}
class Circle extends Shape {
  double radius;

  Circle(this.radius);

  @override
  double area() => 3.14 * radius * radius;
}
```

2.8. 8️⃣ Interfaces in Dart

Dart doesn't have interface keyword.

Every class can be used as an interface using implements.

```
class Flyable {
  void fly() {}
}

class Bird implements Flyable {
  @override
  void fly() {
    print('Bird is flying');
  }
}
```

📌 Difference:

- extends → inherit code
- implements → must re-write everything

2.9. 9️⃣ Static Members

Belong to the **class**, not object

```
class MathUtils {
  static int add(int a, int b) => a + b;
}
void main() {
  print(MathUtils.add(2, 3));
}
```

2.10. 🔟 Mixins (Very Important in Flutter)

Used to **reuse behavior without inheritance**

```
mixin Logger {
  void log(String msg) {
    print('LOG: $msg');
  }
}

class User with Logger {
  void save() {
    log('User saved');
  }
}
```

🧱 Now: Flutter Classes (Widgets)

This mixin:

- **is not a class**
- **just contains reusable methods or variables**

In Flutter, **EVERYTHING IS A CLASS**.

2.11. 1️⃣ StatelessWidget

Used when **UI does NOT change**

```
class MyText extends StatelessWidget {
  const MyText({super.key});

  @override
  Widget build(BuildContext context) {
    return Text('Hello Flutter');
  }
}
```

✓ No state

✓ Faster

✓ Immutable

2.12. 2️⃣ StatefulWidget (VERY IMPORTANT)

Used when UI **changes over time**

- **Step 1: Widget class**

```
class Counter extends StatefulWidget {
  const Counter({super.key});

  @override
  State<Counter> createState() => _CounterState();
}
```

Step 2: State class

```
class _CounterState extends State<Counter> {
  int count = 0;

  void increment() {
    setState(() {
      count++;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Column(
      children: [
        Text('Count: $count'),
        ElevatedButton(
          onPressed: increment,
          child: Text('Add'),
        ),
      ],
    );
  }
}
```

📌 setState() → tells Flutter **rebuild UI**

2.13. 3️⃣ Why Two Classes in StatefulWidget?

Class	Role
StatefulWidget	Configuration
State	Mutable data & logic

This separation improves **performance**.

2.14. 4 Flutter Constructor + Key

```
class MyButton extends StatelessWidget {
  final String title;

  const MyButton({super.key, required this.title});

  @override
  Widget build(BuildContext context) {
    return ElevatedButton(
      onPressed: () {},
      child: Text(title),
    );
  }
}
```

2.15. 5 Inheritance in Flutter Widgets

```
class CustomText extends Text {
  CustomText(String data)
      : super(
          data,
          style: TextStyle(fontSize: 20, color: Colors.blue),
        );
}
```

2.16. 6 Abstract Classes in Flutter

Example: Repository Pattern

```
abstract class UserRepository {
  Future<String> getUserName();
}
class ApiUserRepository implements UserRepository {
  @override
  Future<String> getUserName() async {
    return 'Sipan';
  }
}
```

2.17. 7️⃣ Lifecycle Methods (StatefulWidget)

```
@override
void initState() {
  super.initState();
}

@override
void dispose() {
  super.dispose();
}
```

Method	When
initState	Once when widget created
build	Every UI update
dispose	Cleanup

🔑 Key Takeaways

✓ Dart classes = logic & data

✓ Flutter widgets = UI classes

✓ StatelessWidget → no change

✓ StatefulWidget → dynamic UI

✓ mixin, abstract, implements = architecture tools

✓ Flutter heavily relies on **OOP**