ScopeTreasure Courses

DART COURSES · LESSON D.8 · CODE SCOPE SERIES

OOP in Dart — Class & Object

A class is a blueprint, an object is the house built from it — fields, constructors, methods and getters.

Introduction

So far your data lives in loose variables — a name here, marks there. Real apps group related data *and* the actions on it into one unit: a student has a name and marks, *and* can show its result. That unit is an **object**, and the plan it is built from is a **class**. This is the doorway to Flutter, because in Flutter everything on screen — every widget — is an object built from a class. *(Roman Urdu: Class naqsha hai, object us naqshe se bani hui cheez — aik naqshe se sau cheezein ban sakti hain.)*

Key points

- A **class** is a blueprint: it declares which data (fields) and which actions (methods) its objects will have. An **object** is one real thing built from that blueprint. One class, many objects — one house plan, a whole street of houses.
- **Fields** are the variables inside the class (`String name;`). **Methods** are the functions inside the class (`void showResult() { ... }`). Data and behaviour travel together.
  (Roman Urdu: Class ke andar ki variable ko field, function ko method kehte hain.)
- The **constructor** is the special method that runs when an object is born: `Student('Ali', 92)`. Dart's shortcut `Student(this.name, this.marks);` means "take these values and put them straight into my fields" — the `this.` says "this object's own field".
- You make an object with the class name like a function call: `Student ali = Student('Ali', 92);`. Then reach its insides with a dot: `ali.name`, `ali.showResult()`.
- Every object has its **own copy** of the fields. Changing `ali.grade` never touches `sara.grade` — the blueprint is shared, the houses are separate.
- A **getter** looks like a field but computes its value on the spot: `double get area => length * width;` — you read it as `r.area`, with no `( )`.

Code examples

Example 1 — Your first class: blueprint to object

```
class Student {
  String name;
  int marks;

  Student(this.name, this.marks);

  void showResult() {
    print('$name ke marks: $marks');
  }
}

void main() {
  Student ali = Student('Ali', 92);
  ali.showResult();
}
```

What it does: The class declares two fields, a shortcut constructor that fills them at birth, and one method. In `main`, the object `ali` is built with real values, and the method prints them. Read it as: the plan (class) decided *what a student is*; the object decides *which student this is*.

What you will see: one line — `Ali ke marks: 92`. Change the values in `Student('Ali', 92)` to your own name and marks and run again — same blueprint, different person. That reuse is the entire point of classes.

Example 2 — Two objects: shared plan, separate lives

```
class Student {
  String name;
  String grade;

  Student(this.name, this.grade);

  void introduce() {
    print('I am $name, grade $grade');
  }
}

void main() {
  Student ali = Student('Ali', 'A');
  Student sara = Student('Sara', 'B');

  ali.grade = 'A+'; // sirf Ali badla
  ali.introduce();
  sara.introduce();
}
```

What it does: Two objects are built from one class. Then only `ali.grade` is changed. If objects shared their fields (they do not), Sara's grade would change too — watch the output to prove they are separate.

What you will see: two lines — `I am Ali, grade A+` and `I am Sara, grade B`. Sara kept her B. In Flutter, this is why ten copies of a card widget can each show a different student, exactly like Lesson F.5 will do.

Example 3 — A getter: a field that calculates

```
class Rectangle {
  double length;
  double width;

  Rectangle(this.length, this.width);

  double get area => length * width;
}

void main() {
  Rectangle r = Rectangle(5, 3);
  print(r.area); // 15.0 — area aik field ki tarah parhi gayi
}
```

What it does: `area` is not stored anywhere — the getter computes `length * width` fresh every time you read `r.area`. If length changes later, area is automatically right, because nothing old was saved. Notice

`area` also uses the arrow-function style from Lesson D.6.

What you will see: one line — `15.0`. Try `print(r.area)` after adding `r.length = 10;` above it: the answer becomes `30.0` without you touching any area code. Getters keep one source of truth.

Quick recap

- Class = blueprint (fields + methods). Object = one real thing built from it. One class → many objects.
- Constructor runs at birth; `this.name` means "this object's own field".
- Dot reaches inside: `ali.name`, `ali.showResult()`.
- Each object owns its field values — changing one object never changes another.
- A getter computes a value on read: `double get area => ...`, used like a field.

Common mistakes

1. **Using the class where the object belongs** — writing `Student.name` makes no sense: the *blueprint* has no name, a *student* does. First build the object (`Student ali = Student(...)`), then use `ali.name`.
2. **Forgetting the constructor values** — `Student()` with no arguments fails when the constructor expects name and marks. The constructor's `( )` is a shopping list: everything in it must be handed over at birth.
3. **Expecting two objects to share changes** — beginners sometimes change `ali.marks` and wonder why `sara.marks` did not move. That separation is not a bug; it is the safety that lets one app hold a thousand students at once.

Practice

1. Make a `Book` class with fields `title` (String) and `pages` (int), a constructor, and a method `describe()` that prints both. Create two different books in main and describe both.
2. Add a getter to your Book class: `bool get isLong => pages > 300;` — then print `book.isLong` for a 150-page book and a 450-page book, and check the answers.

Practice quiz

1. What is the difference between a class and an object?

- They are the same thing
- A class is the blueprint, an object is one real thing built from it
- A class is one thing, an object is the blueprint
- A class holds data only, an object holds code only

Answer: A class is the blueprint, an object is one real thing built from it

2. What does this.name mean inside a constructor like Student(this.name, this.marks)?

- The name of the class

- This object's own field, put the value straight into it
- A global variable
- The next object's field

Answer: This object's own field, put the value straight into it

3. If you change ali.marks, what happens to sara.marks?

- It changes too, objects share fields
- Nothing, each object owns its own copy of the fields
- It becomes zero
- The program crashes

Answer: Nothing, each object owns its own copy of the fields

Watch the video lesson: https://youtu.be/QDuVubnqT5k

Read online: https://scopetreasure.com/notes/dart-classes-and-objects · More free tools and notes at https://scopetreasure.com