Dart Mini Project — Student Marks App

The whole Dart series in one real program: a Student class, a list of students, grades, totals and a safe average.

Introduction

Time to stop practising pieces and build a thing. This mini project is a **Student Marks App** in pure Dart: it stores students in a list, prints each student's marks and grade, survives a student whose marks have not arrived yet, and finishes with the class total and average. Every line uses something from this series — classes (D.8), lists (D.7), loops (D.5), a function (D.6) and null safety (D.9) — assembled the way real programs are assembled: in steps. *(Roman Urdu: Yeh project poori Dart series ko aik asli program mein jorta hai.)*

Key points

- **Step 1 — the model:** a `Student` class with a name and *nullable* marks (`int? marks`). Marks can genuinely be missing (a student was absent), so the honest type is `int?`, not `int` — Lesson D.9's thinking, applied before any problem appears.
- The class carries a `grade()` method. Inside it, the marks are checked for null *first*; after that check, Dart promotes them to safe, and the grade chain (A ≥ 90, B ≥ 75, C ≥ 50) is the if / else ladder from Lesson D.4.

  (Roman Urdu: Pehle model (class), phir data (list), phir report — asli program isi tartib se banta hai.)
- **Step 2 — the data:** students live in a `List<Student>` — a list whose items are objects. This is the exact shape Flutter apps use for lists of anything: products, chats, posts.
- **Step 3 — the report:** a `for-in` loop walks the list once. A student with null marks prints a polite line and is **skipped for the total** — not counted as zero. Zero and missing are different facts; good programs keep them different.
- **The average function:** `averageOf` adds only the students who *have* marks and divides by how many they are. Dividing the total by *all* students (including the absent one) would quietly lie — the function exists so this decision lives in exactly one place.

Code examples

Example 1 — Step 1: the Student class

```
class Student {
  String name;
  int? marks; // null ho sakta hai — marks abhi nahi aaye

  Student(this.name, this.marks);

  String grade() {
    if (marks == null) {
      return 'No grade';
    }
    if (marks! >= 90) return 'A';
    if (marks! >= 75) return 'B';
    if (marks! >= 50) return 'C';
    return 'F';
  }
}
```

What it does: The class holds the name and the maybe-missing marks. In `grade()`, the null case leaves immediately ('No grade'); after that check, `marks!` is provably safe, because the only way to reach those lines is with marks present. The `!` here is justified — the check above it is the proof.

What you will see: nothing yet — a class alone prints nothing. That is normal in real projects: you build the parts first, then the program that uses them. Keep this code; Steps 2 and 3 stand on it.

Example 2 — Step 2: the data + the average function

```
List<Student> students = [
  Student('Ahmed', 92),
  Student('Sara', 85),
  Student('Bilal', null), // Bilal ke marks abhi nahi aaye
];

double averageOf(List<Student> list) {
  int total = 0;
  int count = 0;

  for (Student s in list) {
    if (s.marks != null) {
      total = total + s.marks!;
      count++;
    }
  }
  return total / count;
}
```

What it does: Three objects are created from the Step 1 blueprint — one per student, Bilal's marks honestly null. The function loops over any student list it is given, adds up only the present marks, counts only those students, and returns the true average.

What you will see: still nothing on the console — data and a function, both waiting. Notice what the function does *not* do: it never divides by 3. Bilal is absent from both the total and the count, so the average will describe the students who actually sat the exam.

Example 3 — Step 3: the full program, run

```
void main() {
  int total = 0;

  for (Student s in students) {
    if (s.marks == null) {
      print('${s.name}: marks not available');
    } else {
      total = total + s.marks!;
      print('${s.name}: ${s.marks} marks, grade ${s.grade()}');
    }
  }

  print('TOTAL (jin ke marks hain): $total');
  print('AVERAGE: ${averageOf(students)}');
}
```

What it does: Put Steps 1 and 2 above this `main()` in one file and run it. The loop prints one line per student — a full report line when marks exist (adding them to the total on the way), a polite line when they do not — and then the summary uses the function from Step 2.

What you will see:

```
Ahmed: 92 marks, grade A
Sara: 85 marks, grade B
Bilal: marks not available
TOTAL (jin ke marks hain): 177
AVERAGE: 88.5
```

Five lines, no crash, no lie: 92 + 85 = 177, and 177 ÷ 2 students = 88.5. You just built — and can now explain line by line — a program that uses every Dart lesson in this series. Keep this file: in the Flutter finale (F.5), this same data becomes a phone app.

Quick recap

- Build in steps: model (class) → data (list of objects) → behaviour (functions) → report (main).
- Nullable marks were a *design decision*, made in the class, before any null could crash anything.
- Missing is not zero: absent students leave both the total and the count.
- A class + a list of its objects is the standard data shape of real apps.

Common mistakes

1. **Counting the absent student as 0** — if Bilal's null were treated as 0 marks, the average would fall to 59.0 and the report would punish a student who never sat the exam. Always ask: "is this value missing, or is it really zero?"
2. **Writing the average logic twice** — once in the loop and once in a function, "to save time". Two copies drift apart the first time one is edited. One job, one place: that is why `averageOf` exists.
3. **Putting everything in main()** — it works for three students and becomes unreadable at thirty. The class / function / main split feels slower for one day and faster for the whole career.

Practice

1. Add two more students — one with marks, one without — and run the program again. Predict the TOTAL and AVERAGE lines *before* you run, then check your prediction.
2. Add a `topper()` function that returns the name of the student with the highest marks (ignoring nulls), print it in main, and test it by changing Sara's marks to 95.

Practice quiz

1. In what order is the Student Marks App built?

- Report first, then data, then model
- Model (class), then data (list of objects), then behaviour (functions), then report (main)
- Data first, then report, then model
- All in main() at once

Answer: Model (class), then data (list of objects), then behaviour (functions), then report (main)

2. How are students with null (missing) marks treated in the total?

- Counted as zero marks
- Skipped for the total, missing is not zero
- Given the class average
- Removed from the list forever

Answer: Skipped for the total, missing is not zero

3. What does the averageOf function divide by?

- All students, including absent ones
- Only the students who have marks
- The number 100
- The highest marks in the class

Answer: Only the students who have marks

Watch the video lesson: https://youtu.be/Olp2xhXnX8M

Read online: https://scopetreasure.com/notes/dart-mini-project-student-marks · More free tools and notes at https://scopetreasure.com