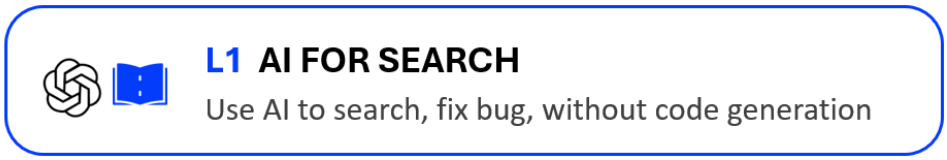
W3 *PRACTICE*

LAYERED ARCHITECTURE

Learning objectives

- Extend the **quiz app** with new **features** while keeping clear layer separation.
- Update the **UML class diagram** to reflect the new design (composition, aggregation, multiplicity).
- Replace the fake data source with a **file-based implementation** in the data layer.
- Improve **testability** by writing and running unit tests for the domain layer.



How to submit?

- ✔ **Push** your final code on **your GitHub repository**
- ✔ Then **attach the GitHub path** to the MS Team assignment and **turn it in**

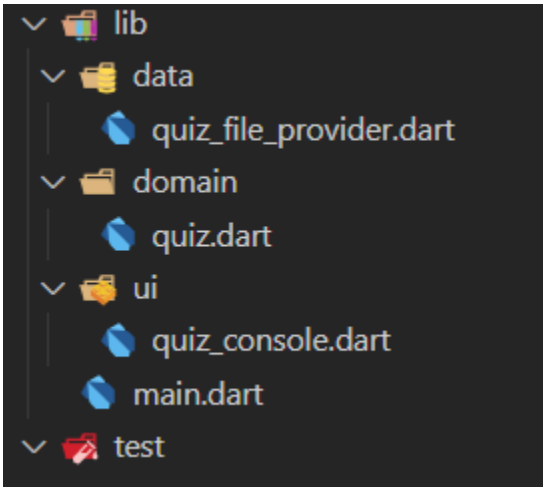
Are you lost?

Read the following documentation to be ready for this practice:

- ✓ Classes
- ✓ Constructors
- ✓ Methods

*EX 1 – **Understand** start code*

We provide a start code, composed of 3 layers (DATA, DOMAIN, UI).



Run the `main.dart` and ensure you get the bellow results (for now the data is FAKE) :

```
--- Welcome to the Quiz ---

Question: Capital of France?
Choices: [Paris, London, Rome]
Your answer: d

Question: 2 + 2 = ?
Choices: [2, 4, 5]
Your answer: d
```

Q1- Draw on paper the UML diagram, related to the **DOMAIN** layer

Q1- Write your test cases for this first version (`quiz_test.dart`):

- Create a quiz test with 2 questions and the answers to the 2 questions
- Check the score is the expected score (as example: 100, 50 or 0)

*EX 2 – Add **points** per question*

Now each question **has a number of points** to get.

- If not specified, the number of points is 1 by default.

```
--- Welcome to the Quiz ---

Question: Capital of France? - ( 10 points)
Choices: [Paris, London, Rome]
Your answer: Paris

Question: 2 + 2 = ? - ( 50 points)
Choices: [2, 4, 5]
Your answer: 4

--- Quiz Finished ---
Your score in percentage: 100 %
Your score in points: 60
```

We get 60 points (10 from Q1, and 50 from Q2)

Q1 – Update the **DOMAIN**

What classes attributes, methods do you want to change?

Q2 – Update the **UI**

Follow the screenshot

Q3 – Update the **TESTS**

*Check the score is the expected percentage but also **expected points***

*EX 3 – Add **multiple players***

Now we want many **players to play this quiz**.

<u>The game flow is as bellow:</u>

1 – The player enters his/her name

2 – The player starts the quiz

3 – Show the player score, and display the last players scores.

4 – Continue with another player, **until the entered name is empty**.

```
--- Welcome to the Quiz ---

Your name: RONAN
Question: 3 + 3 = ? - ( 50 points)
Choices: [2, 3, 6]
Your answer: 6
Question: 2 + 2 = ? - ( 50 points)
Choices: [2, 4, 5]
Your answer: 4
RONAN, your score in percentage: 100 %
RONAN, your score in points: 100 %
Player: RONAN           Score:100
Your name: HUGO
Question: 3 + 3 = ? - ( 50 points)
Choices: [2, 3, 6]
Your answer: 2
Question: 2 + 2 = ? - ( 50 points)
Choices: [2, 4, 5]
Your answer: 1
HUGO, your score in percentage: 0 %
HUGO, your score in points: 0 %
Player: RONAN           Score:100
Player: HUGO            Score:0
Your name:
--- Quiz Finished ---
```

⚠️ Note: that if the player has already played, **the last score will be overwritten**

Q1 – Update the **UML diagram** to match with this requirement

Q2– Update the **DOMAIN** accordingly

What classes attributes, methods do you want to change?

⚠️ You might need to move code from class to class!

Q3 – Update the **UI**

Follow the screenshot

Q4 – Update the **TESTS**

Add players and check the submissions scores are valid

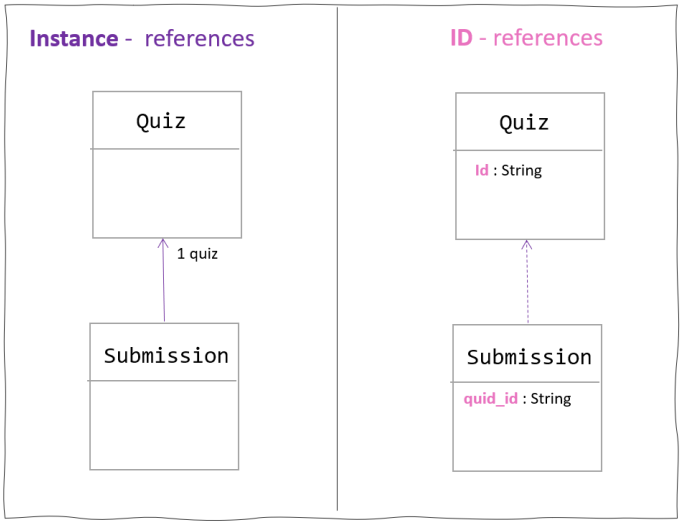
*EX 4 – **Read** the QUIZ from a JSON file*

Now we want to **save the list of questions** and the submissions to an ASCII file !

From Instance to ID references…

Having direct references to object instances works fine in memory.
But it becomes difficult to manage once the **model needs to be persisted**.

By introducing an **ID-based mechanism**, we can easily store and retrieve relationships when saving our model to JSON files, text files, or even SQL databases.



When we assign IDs manually (like 1, 2, 3), we risk collisions or duplication — especially when data comes from multiple sources or files.

To ensure that every object has a truly unique identifier, we can use the uuid package, which generates Universally Unique Identifiers (UUIDs).

https://pub.dev/packages/uuid

Q1 - Switch from Instance to ID references…

Update the **UML** to match with this requirement

- Quiz and Question and Answers should have now ID
- Add getter to retrieve a question from its ID
- Add getter to retrieve an answer from its ID

Update the **DOMAIN** accordingly

Import the UUID package and update the code accordingly

Update the **TESTS**

All the test should pass again

Read and process a JSON file

A JSON file is a map of key values, as represented bellow:

```
Map<String, dynamic> json
```

A JSON file can be read synchronously as shown bellow:

```
  File file = File(filePath);
  String content = file.readAsStringSync();
  Map<String, dynamic> data = jsonDecode(content);
```

In the data layer, create a file `QuizRepository` to read (and write later) a Quiz

Note: for now, we process the file **synchronously**

```
class QuizRepository {
  final String filePath;

  QuizRepository(this.filePath);

  Quiz readQuiz() {
    final file = File(filePath);
    final content = file.readAsStringSync();
    final data = jsonDecode(content);

    // Map JSON to domain objects
    var questionsJson = data['questions'] as List;
    var questions = questionsJson.map((q) {
      return Question(
        title: q['title'],
        choices: List<String>.from(q['choices']),
```

```
        goodChoice: q['goodChoice'],
        points: q['points'],
      );
    }).toList();

    return Quiz(questions: questions);
  }
}
```

Example of code to read a list of questions and create a Quiz

Q2 – Complete this above start code to **READ** the Quiz from a JSON file, including the ID, the player submissions…

Q3 – (BONUS) Add a method to **WRITE** a QUIZ into a JSON file