CMPS 312 Mobile Application Development
Lab 3-Dart OOP and Lambdas

Objective

1. Practice **Object-Oriented Programming (OOP)** in Dart.
2. Apply a **simple Clean Architecture** structure to separate domain and data layers.
3. Read and parse **JSON** data into objects.
4. Use **lambdas** and **extensions** for collection processing.

PART A – OOP

In section of the lab, you will build a small Library application using Dart. The goal is to practice OOP and Clean Architecture concepts in a console app. You will model books, store them in JSON, and manage them through a repository.

Step 1 : Project Setup

1. Create a Dart console project named **library** with the following folders:
2. By the end of this exercise your folder structure should look like this.

```
assets/data/          // JSON lives here
bin/                  // console entry point
lib/
  domain/             // pure business layer (no IO)
    contracts/        // interfaces (abstract classes)
    entities/         // Book, AudioBook, PaperBook
    mixins/           // optional shared behaviors
  data/               // implementations (JSON repository)
  core/utils/         // optional extensions/helpers
```

3. Figure 1 shows the library application class hierarchy that you will be implementing.

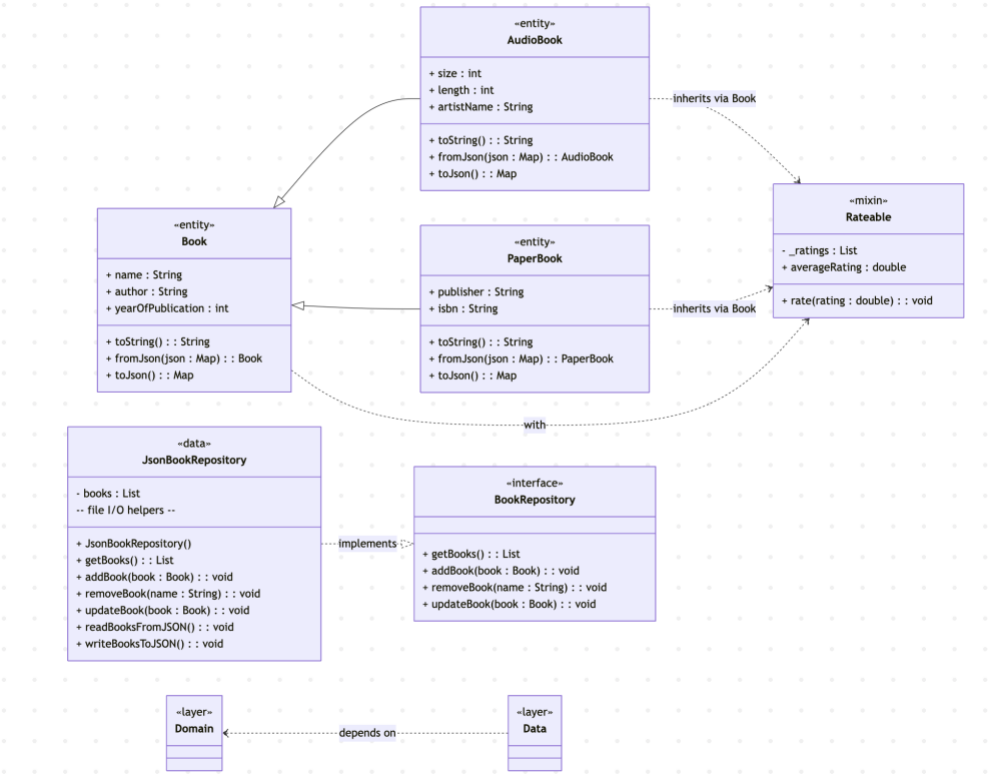


Figure 1 Library Class Diagram

4. Inside **assets/data/,** create a file called **books.json** with some initial book data.

Step 1 : Domain Layer : Defines the core business logic of the app (entities, contracts, mixins). It should not depend on any other layer.

Entities [/domain]

1. Create a base class **Book** with:
    a. Properties: name, author, yearOfPublication.
    b. A toString() that prints these fields.
    c. Methods fromJson() and toJson().
2. Create a subclass **AudioBook** with:
    a. Extra fields: size, length, artistName.
    b. Override toString().
    c. Implement JSON conversion.
3. Create a subclass **PaperBook** with:
    a. Extra fields: publisher, isbn.
    b. Override toString().
    c. Implement JSON conversion.

Mixin [/mixins]

- Create a mixin named Rateable with:
    - A private list of ratings.
    - A method rate(double rating).
    - A getter averageRating.
    - Apply this mixin to your entities so that all book types can be rated.

Contracts [/contracts]

- In book_contracts.dart, create an abstract class BookRepository that declares:
    - getBooks()
    - addBook(Book book)
    - removeBook(String name)
    - updateBook(Book book)

Step 2 : Data Layer : provides the implementation details, in this case reading/writing books from a JSON file. It depends on the domain layer.

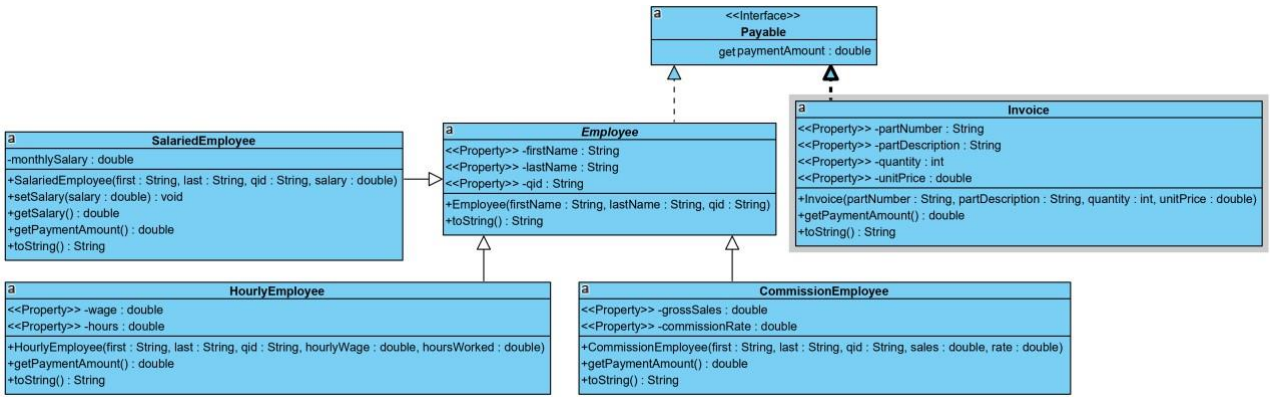
1. Implement **JsonBookRepository** in lib/data/repositories/.
2. It should:
    - Load books from assets/data/books.json.
    - Parse the "type" field to decide whether to create an AudioBook or a PaperBoo

Step 3 : Presentation Layer : For this console app, we will consider the bin as our presentation layer. As it is the entry point that wires everything together and interacts with the user.

1. Instantiates the JSON repository.
2. Loads and prints all books using forEach.
3. Adds a new book and saves it back.
4. Rates a book and displays its average rating.
5. Updates and removes a book to test all repository functions.
6. A Create an extension under libs/core/utils on List<Book> that can:
    a. Calculate the average publication year.
    b. Group books by type.
    c. Return the top N books by rating.
7. Demonstrate the extensions in your console program

Exercise 2

1. Create an project named **Payroll**
2. Create a package named **model**
3. Implement the following class hierarchy inside the **model** package



- Note that the amount to pay for HourlyEmployee is *wage * hours*. For CommissionEmployee, it is *grossSales * commissionRate*. For Invoice, it is *quantity * unitPrice*.
- Make sure the **salary**, **rate** and **sales** are all non-negative numbers otherwise display a warning message. [hint: for data validation using init or set methods]

Test your implementation using the main method

```
void main() {

  // create payable list   List<Payable> payables = [];   payables.add(Invoice("01234",

"Textbook", 2, 375.00));   payables.add(Invoice("56789", "USB Disk", 3, 179.95));

payables.add(SalariedEmployee("Ahmed", "Ali", "111-11-1111", 15000.00));
```

```
payables.add(HourlyEmployee("Fatima", "Saleh", "222-22-2222", 160.75, 40.0));

payables

    .add(CommissionEmployee("Samir", "Sami", "333-33-3333", 100000.0, 0.06));


 print("Invoices and Employees processed polymorphically:\n");


 // generically process each element in the list using forEach

payables.forEach((payable) {

   // output current Payable and its appropriate payment

amount     print("$payable\n");


   // If SalariedEmployee, then increase the

salary by 10%     if (payable is

SalariedEmployee) {

    double oldBaseSalary =

payable.monthlySalary;

payable.monthlySalary = oldBaseSalary *

1.1;       print(

        "New salary with 10% increase is: QR ${payable.getPaymentAmount()}\n");}

 });
```

```
Invoices and Employees processed polymorphically:

Part Number      : 01234
Part Description : Textbook
Payment Amount   : 750.0

Part Number      : 56789
Part Description : USB Disk
Payment Amount   : 539.8499999999999


First Name :Ahmed
Last Name  :Ali
QID        :111-11-1111
            Payment Amount : 15000.0

New salary with 10%% increase is: QR 16500.0


First Name :Fatima
Last Name  :Saleh
QID        :222-22-2222
            Payment Amount : 6430.0


First Name :Samir
Last Name  :Sami
QID        :333-33-3333
            Payment Amount : 6000.0
```

PART B – Lambdas Warmup exercises

Create a lambda function that takes ,

a) a number as input and returns its square.
b) two numbers as input and returns their sum.
c) a list of numbers as input and returns the sum of all the numbers in the list.
d) a list of strings as input and returns the length of the longest string in the list.
e) a list of strings as input and returns a new list that contains only the strings that start with the letter "A".
f) Create a custom extension method for integer lists that enables users to directly calculate the sum

Dart JSON Serialization and Deserialization Exercises

1. Create an application named **CovidTracker**
2. Copy the **covid-data.json** from **Lab 3 folde**r into a subdirectory named **data** under the root directory of your project (create the **data** subfolder yourself)
3. The most common format used for serialization in Dart is JSON. Dart provides built-in support for JSON through the dart:convert library.
4. Create a class called **CovidStat** (in a Dart file named covid-stat.dart). Derive CovidStat properties from the JSON object shown below. Note that some of the statistics could be null for some countries.

```
class CovidStat {
  late final int totalCases;
  late final int totalDeaths;
  late final int newDeaths;
  late final int totalRecovered;
  late final int newRecovered;
  late final int activeCases;
  late final int population;

  CovidStat(
      {required totalCases,
      required this.totalDeaths,
      required this.newDeaths,
      required this.totalRecovered,
      required this.newRecovered,
      required this.activeCases,
      required this.population});
```

5. Create a factory method named "fromJson" inside the CovidStat class that converts **Map<String, dynamic>** json to Covid Stat Object

6. Add **CovidStatRepository** object (in a Dart file named covid-stat-repository).

The init function of this object should **load** the json data in **data/covid-data.json** file into covidStats list.

The CovidStatRepository should implement the following functions that return:

- The **total COVID deaths around the world.**
- The total **active cases** for a specific **continent**.
- The **top five countries** with the **highest number of COVID cases.**
- The **top five countries** with the **lowest number of COVID cases.**
- The total **critical cases** of the neighboring countries of a given country. For example, if the country is Qatar then you should return all countries having the same region as Qatar and their respective critical cases. Finally sort those countries by population. - The top three regions in a **continent** with the highest recovery - The country with the **lowest death** in a **continent**.

7. Add a Dart file named **main** to test the functions in CovidStatRepository object. Test as you implement the functions of CovidStatRepository object. Do NOT leave the testing to the end.