

## W2-S2 PRACTICE

### OOP MANIPULATION

#### 💡 Learning objectives

- Implement a class in Dart with specific **attributes and methods**
- Define and use default constructors.
- Implement **named constructors** and **parameterized constructors**.
- Implement **Value Objects, immutable objects**
- Manipulate **enumerations**
- Handle **Exceptions**
- Identify **const** and **final** attributes
- Be able to **encapsulate** data (private members, getters)
- Implement Operator **Overloading**
- Implement **aggregation** and **composition** of classes



No AI tools allowed to solve this practice



#### ⬇ 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 – Employee Class



In this exercise, you will be working on an existing Employee class. **Read the start code!**

We provide the start code of the `Employee` class and the main that creates a few instances of `Employee`. We also provide the `Skill` enum with various values.

Employee	< enum > Skill
+ String name + double baseSalary	FLUTTER DART OTHER

**Q1-** Add the following **new attributes to the Employee class**:

- skills: A list of skills
- address: An Address class that contains street, city, and zipCode attributes
- yearsOfExperience: An integer representing the employee's experience in years

**Q2 - Update the constructor** to initialize the new attributes.

Add [named constructors](#) for specific types of employees

Example: *mobileDeveloper* constructor that assigns FLUTTER and DART skills

**Q3 -** Make all attributes **private** and **provide getter** methods for accessing them.

**Q4 –** Add a method to **compute the salary** of the employee (*you can create your own rules!*):

Salary Specification:

Base salary: \$40,000.

Each year of experience adds \$2,000.

Each skill adds a different bonus:

- FLUTTER: \$5,000.
- DART: \$3,000.

# EX

- OTHER: \$1,000.

*Note: don't forget to define attributes const or final whenever possible!*

## 2 – Bank System



In this exercise, you will be working on a Bank account and a Bank class. **Read the start code!**

### Bank Account

**Q1** - Decide which attributes make sense for a bank account

**Q2** – Implement the following methods

`balance():`

- Returns the current balance
- `withdraw(double amount)`

  - Deducts the given amount from the account balance.
  - If the balance goes below 0, it should throw an exception.

`credit(double amount)`

- Adds the given amount to the account balance.

### Bank

**Q3** - Decide which attributes make sense for a bank

- The Bank class manages a list of accounts and ensures the uniqueness of each account ID.

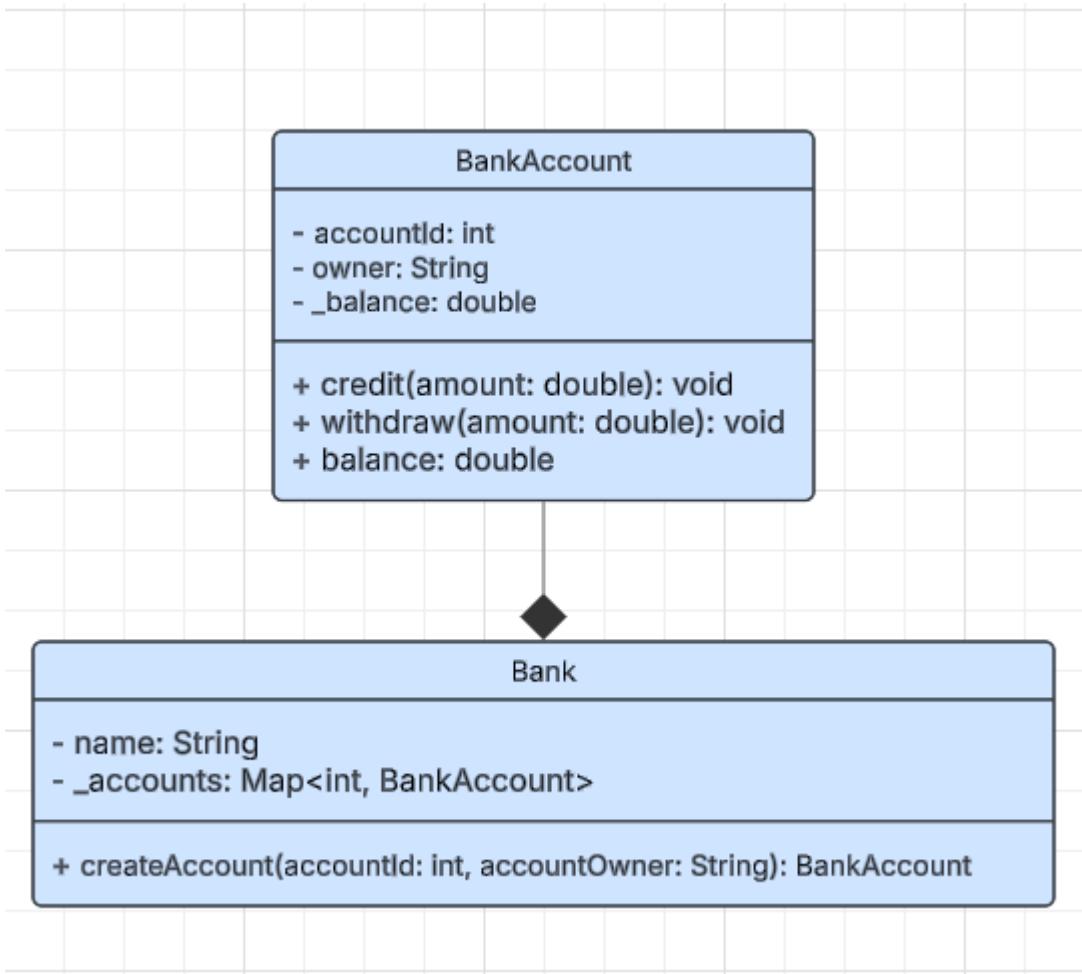
**Q4** – Implement the following method

`Account createAccount(int accountId, String accountOwner)`

- Create a new bank account
- Ensure that the account ID is unique. Otherwise, throw an exception - Add the account to the bank list and return it

**Q5 –** Draw a UML class diagram that reflects your implementation.

- Attributes and methods of the BankAccount class.
- The relationship between Bank and BankAccount (composition or aggregation)



*EX*

## 3– Duration



In this exercise, you need to create a `CustomDuration` class, similar to Dart's built-in `Duration`.

<https://api.dart.dev/stable/3.5.3/dart-core/Duration-class.html>

You need to understand:

- the concepts of **immutability**
- operator **overloading**
- custom **methods**

### Q1 –Attribute and constructors

- Internally store the duration as a number of milliseconds (private field).
  - o **Duration shall always be greater or equal to 0**
- Constructor:
  - o `fromHours(int hours)`: Constructs a duration from a number of hours
  - o `fromMinutes(int minutes)`: Constructs a duration from a number of minutes
  - o `fromSeconds(int seconds)`: Constructs a duration from a number of seconds

### Q2- Overloaded Operators:

- `>`: Compare two durations, returning true if one duration is greater than the other.
- `+`: Add two durations, returning a new `CustomDuration` object.
- `-`: Minus two durations, returning a new `CustomDuration` object (*if possible*)

*EX*

## 4 – Shop Management



A shop sells **products** that customers can **order online**.

An **order** can be **delivered** to an address or **picked up** at the shop.

We must compute each order's **total amount**.

### Learning Objectives

Design a simple **Online Shop system** using Dart concepts:

- Constructors (named, required, nullable)
- Enums
- UML class diagrams
- Object relations and methods

### Questions

#### Q1 - UML

- Draw a **UML diagram** showing classes, attributes, and relationships. *Be ready to explain your choices to other students or the classroom*

#### Q1 - Implementation

- Implement the Dart classes and methods to solve this problem.  
*How can we get the order total amount?*
- In main(), create sample data and test your API

### Deliverables

## *EX*

- UML diagram
- Dart code file