UNIVERSITY OF RWANDA
COLLEGE OF SCIENCE AND TECHNOLOGY
COMPUTER ENGINEERING
YEAR 3
GAKO CAMPUS

DART FOR FLUTTER MOBILE PROMMING

Reg n°: 223026685
223027476
223026674

Names:

Jerome HAKIZIMANA
Clement NZABAMWITA
Lewis NGENDAHIMANA

Dart Programming Lab 1

REPORT

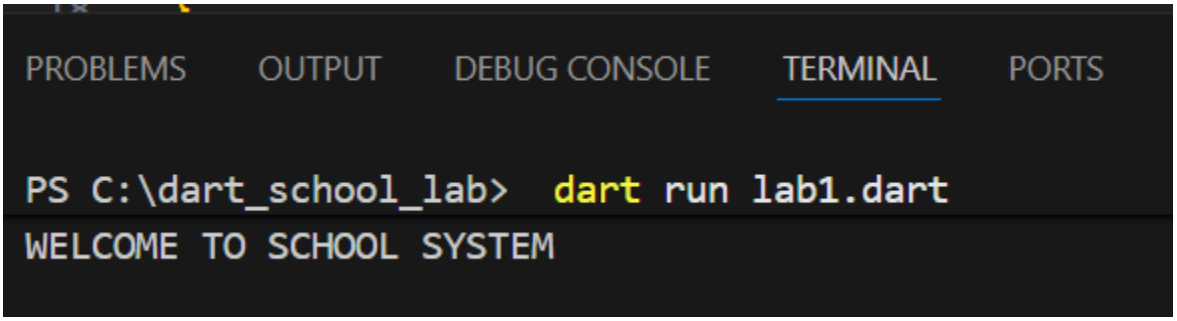
Contents

Lab Report: Dart Programming for Flutter

Part 1: Functions

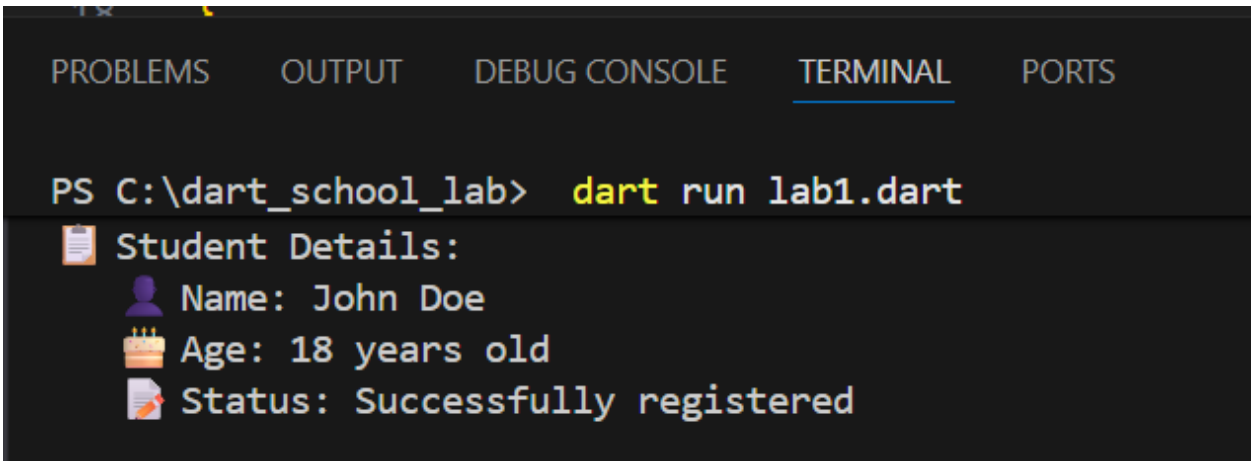
Q1: welcomeMessage Function

Output:



Q2: createStudent with Named Parameters

Output:



Q3: createTeacher with Optional Parameters

Output:

```
PS C:\dart_school_lab>  dart run lab1.dart
👨‍🏫 Teacher Registration:
   Name: Mr. Anderson
   Subject: Mathematics

--- Testing without subject ---

👨‍🏫 Teacher Registration:
   Name: Ms. Garcia
   ⚠️Subject not assigned
```

Part 2: Constructors and Classes

Q4: Student Class with Constructor

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\dart_school_lab>  dart run lab1.dart
Object created: Student("Michael Chen", 17)
```

Q5: Object Creation and Usage

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\dart_school_lab>  dart run lab1.dart
Student: Michael Chen | Age: 17
```

Part 3: Inheritance

Q6: Person Class

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\jerom\Desktop\dart-school-lab> dart run .\test.dart
[Q6 & Q7] Inheritance (Person → Student2):
Hello, my name is Benjamin.
```

Q7: Student Inherits from Person

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\dart_school_lab>  dart run lab1.dart
Hello, I am John Smith
```

Part 4: Interfaces

Q8: Abstract Class Registrable

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\dart_school_lab>  dart run lab1.dart
Abstract class Registrable defined with registerCourse method
```

Q9: Student Implements Registrable

Q8 & Q9

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\dart_school_lab>  dart run lab1.dart
✅ Robert Johnson registered for: Physics
   Total courses: 1
```

Part 5: Mixins

Q10: AttendanceMixin

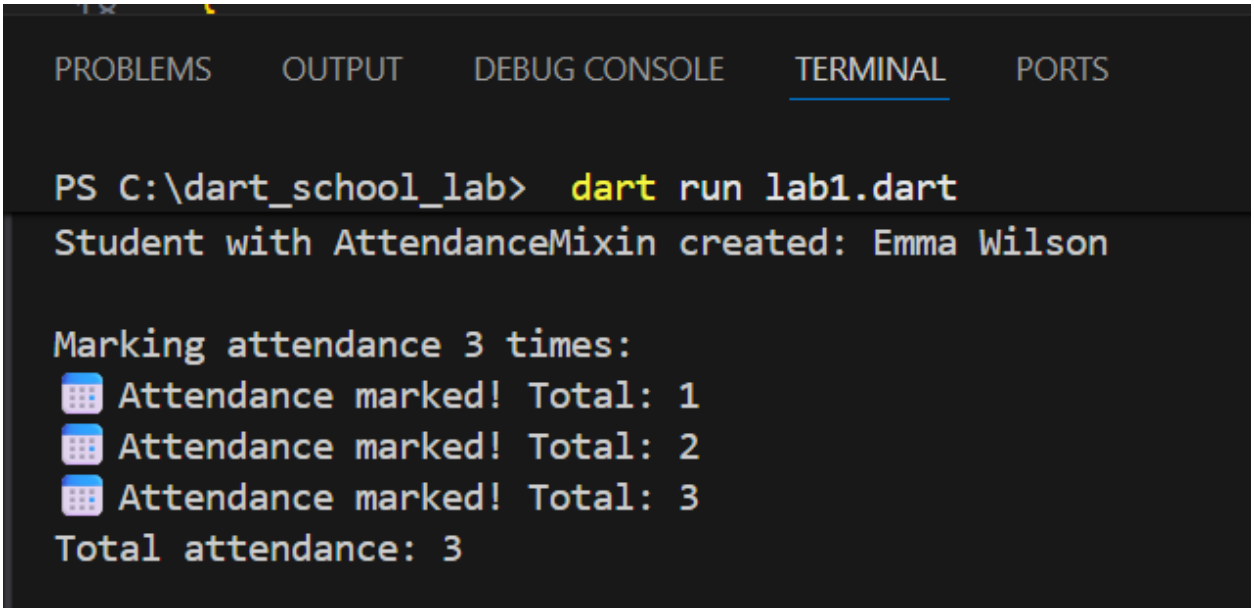
Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\dart_school_lab>  dart run lab1.dart
AttendanceMixin created with markAttendance() method
```

Q11: Applying AttendanceMixin to Student

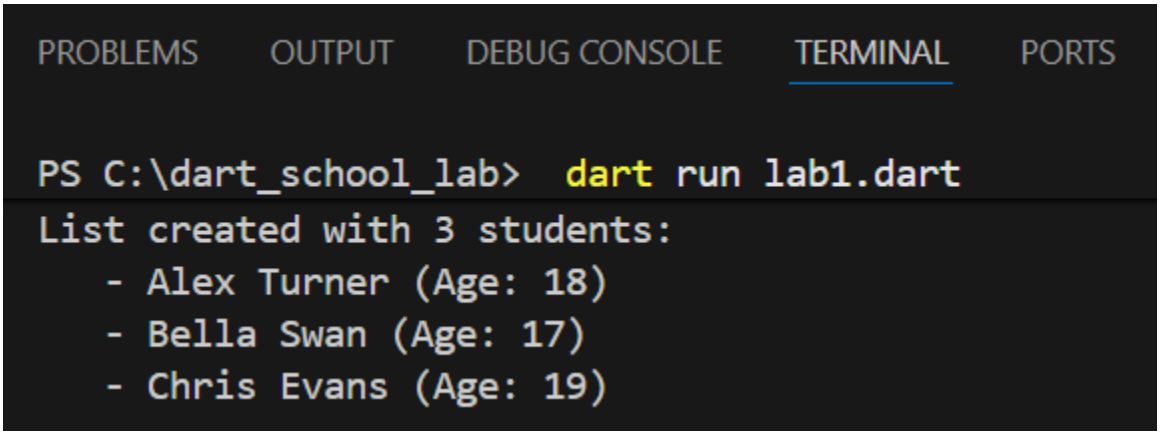
Output:



Part 6: Collections

Q12: List of Student Objects

Output:



Q13: Map of Students

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\dart_school_lab>  dart run lab1.dart
Map created with 3 entries:
   S201: David Miller
   S202: Elena Gilbert
   S203: Frank Ocean
```

Part 7: Anonymous and Arrow Functions

Q14: Anonymous Function

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\dart_school_lab>  dart run lab1.dart

📋 Student List (using anonymous function):
   👤 Alex Turner
   👤 Bella Swan
   👤 Chris Evans
```

Q15: Arrow Function

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\dart_school_lab>  dart run lab1.dart
Arrow function: String welcomeStudent(String name) => "Welcome, $name!"
Testing arrow function:
    Welcome, John!
    Welcome, Mary!
```

Part 8: Asynchronous Programming

Q16: Async Function loadStudents()

Output:

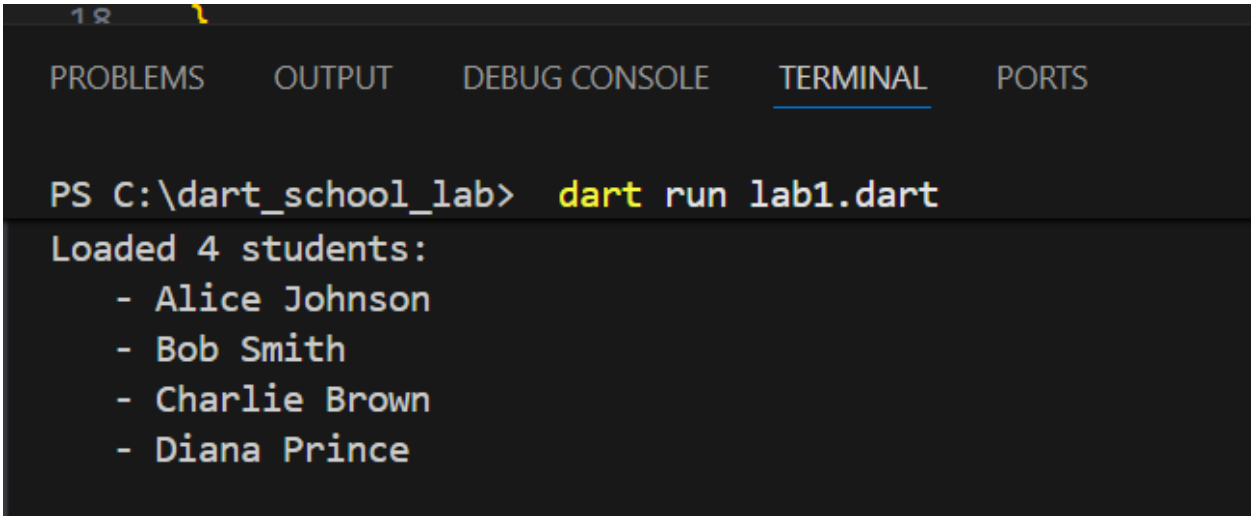
```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\dart_school_lab>  dart run lab1.dart
Async function that waits 2 seconds:

⏳ Loading student data...
```

Q17: Using await in main()

Output:



Part 9: Integration Challenge

Q18: Mixins vs Inheritance

Mixins are a powerful Dart feature that allows reusable code to be shared across multiple class hierarchies without establishing an “is-a” relationship.

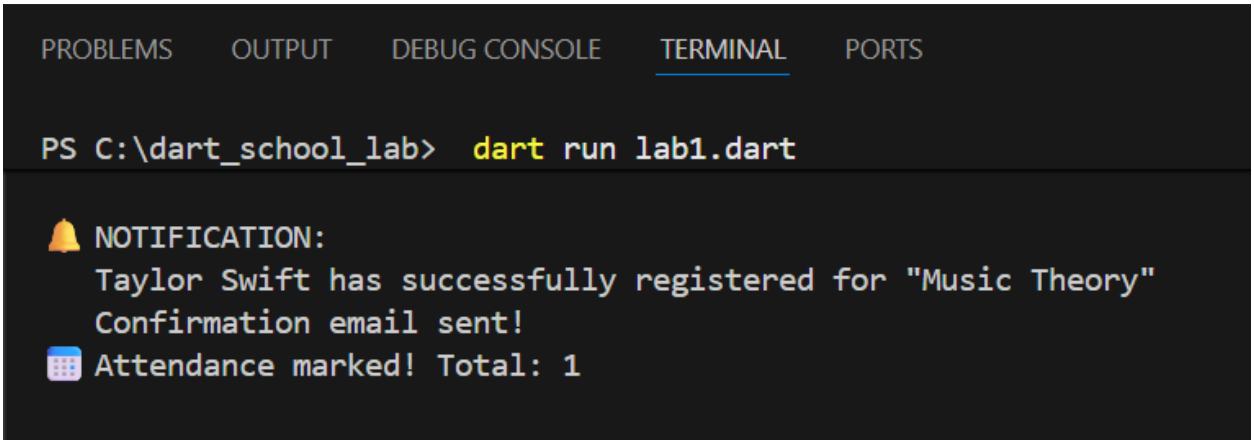
They are useful for adding common behavior such as logging, notification, or attendance tracking to classes that are otherwise unrelated.

Unlike inheritance, which creates a strict parent-child chain and is limited to a single superclass, mixins can be applied to any class using the with keyword, and a class can use several mixins at once.

Inheritance defines what an object *is* (e.g., a Student is a Person), while mixins define what an object *can do* (e.g., a Student can send notifications). Mixins promote composition over inheritance, keeping class hierarchies simple and avoiding the problems of deep or multiple inheritance.

They encapsulate cross-cutting concerns, making code more modular, maintainable, and reusable.

Q19: NotificationMixin



Q20: How Learning Dart Helps Understand Flutter

Learning Dart is essential for understanding Flutter because Flutter is built entirely on Dart—every Flutter application is fundamentally a Dart application.

The concepts I practiced in this lab translate directly to Flutter development: classes define widgets, constructors configure widget properties using named parameters, inheritance allows creating custom widgets that extend Flutter's built-in widgets, and mixins add lifecycle methods or common behaviors to widgets.

When I fetch data from an API in Flutter, I use the same async/await syntax I learned here. When I display lists of data, I work with Dart collections like List and Map. Anonymous functions become event handlers for button presses, and arrow functions provide concise callbacks.

Without understanding Dart, Flutter widgets would seem like magic; with Dart knowledge, I recognize that Flutter is simply a sophisticated framework leveraging Dart's object-oriented and asynchronous features.

This lab has given me the confidence to move from console programs to building real mobile apps, knowing that the same language rules and programming patterns apply seamlessly.

SOURCE CODE_👇

GITHUB LINK: https://github.com/sticker2024/dart-school-lab.git