

PRACTICA 11,12 y 13

Rodríguez Alvarez María José

November 2025

1 Marco Teórico

1.1 Archivos

En Dart, la manipulación del sistema de archivos se gestiona principalmente a través de la librería `dart:io`. Esta librería provee clases como `File` y `Directory` para interactuar con el sistema operativo.

1.2 Asincronía en I/O

Un concepto crítico en Dart es que las operaciones de Entrada/Salida (I/O) deben ser, preferentemente, **no bloqueantes**. Dado que Dart es *single-threaded* en su ejecución principal, leer un archivo grande de manera síncrona congelaría la interfaz de usuario.

- **Síncrono:** `file.readAsStringSync()` (Bloquea el hilo).
- **Asíncrono:** `file.readAsString()` (Retorna un `Future`).

El uso de las palabras reservadas `async` y `await` permite escribir código asíncrono que se lee como si fuera síncrono, facilitando el manejo de excepciones y el flujo lógico.

2 Hilos y Concurrencia (Isolates)

A diferencia de Java, donde se crean múltiples *Threads* que comparten el mismo espacio de memoria, Dart utiliza un modelo de aislamiento basado en el **Event Loop** y los **Isolates**.

2.1 El Event Loop vs. Threads

Dart ejecuta el código en un único hilo principal impulsado por un bucle de eventos (*Event Loop*). Para tareas concurrentes ligeras (como esperar una respuesta HTTP o leer un archivo), no se crean nuevos hilos del sistema operativo; simplemente se programan en el Event Loop.

2.2 Isolates: Paralelismo Real

Cuando se requiere procesamiento pesado (CPU-bound) que podría bloquear el Event Loop, Dart utiliza **Isolates**.

- Cada Isolate tiene su propio espacio de memoria (Heap).
- **No hay memoria compartida:** Esto elimina la necesidad de *locks* complejos y evita condiciones de carrera (*race conditions*).
- La comunicación entre el hilo principal y un Isolate se realiza mediante paso de mensajes (*message passing*) a través de puertos (*Ports*).

3 Patrones de Diseño en Dart

La sintaxis moderna de Dart permite implementar los patrones del *Gang of Four* (GoF) de manera más concisa, e incluso algunos están integrados en el lenguaje.

3.1 Patrón Singleton

El patrón Singleton asegura que una clase tenga una única instancia. En Dart, esto se logra elegantemente utilizando un constructor privado y un *factory constructor*.

```
[caption=Implementación de Singleton en Dart]
class Database {
  // 1. Instancia estática privada static final Database _instance = Database._internal();
  // 2. Constructor privado Database._internal();
  // 3. Factory constructor retorna la instancia existente factory Database()
  return _instance;
}
```

3.2 Patrón Observer (Streams)

Mientras que en otros lenguajes se implementan interfaces **Observer** y **Subject** manualmente, Dart incorpora este patrón de forma nativa a través de los **Streams**.

- **StreamController:** Actúa como el sujeto que emite eventos.
- **StreamSubscription:** Actúa como el observador que escucha los cambios.

Este mecanismo es la base de la arquitectura reactiva (BLoC, Provider) utilizada comúnmente en Flutter.