Taller Dart 2: Programación asíncrona con Dart

Nombre y apellidos:	José Teodosio Lorente Vallecillos
Rol (director de proyecto/arquitecto/experto/programador):	Director de proyecto

Objetivos

- Aprender a programar tareas asíncronas en Dart: uso de async, await y la clase Future

NOTA: El punto de partida de este taller es el taller primero de Dart.

Material disponible

En sección Entornos de trabajo y lenguajes→Dart de PRADO hay un esquema que puede ser de gran ayuda para comprender Dart a partir de Java y Ruby.

Descripción del problema

- Vamos a poner en marcha los relojes creados en el taller anterior y a sincronizarlos, de forma que uno de ellos será usado por los otros dos para marcar un nuevo segundo. Realizaremos primero unos pasos previos:
  - [Clase Reloj] Añade el atributo *segundo* a la clase Reloj, con valor 0 por defecto, y haz las modificaciones correspondientes en el resto de la clase (añadir un consultor básico, añadir modificadores básicos, añadir el parámetro correspondiente en el constructor y añadir los segundos en el método *toString*
  - [Método main] Corrige la construcción de los relojes en el método main para añadir los segundos que consideres (fichero main.dart).
  - [Clase Reloj] Método *void addSegundo.()*- Define este método para que añade un segundo a la hora total (ten en cuenta que no se pase de 60, ni tampoco los minutos. Si se pasara la hora de 24, lanza una excepción (para hacerlo, piensa cómo lo harías en java).

- Necesitamos hacer uso de tres conceptos para manejar tareas asíncronas en Dart. Empezaremos por el de Future:
  - Future:
    - Clase Future<T>.- Una clase parametrizable, con el parámetro del tipo que devolvería el método si fuera síncrono. Si no devuelve nada, sería Future<void>.

- Future<T>.delayed(Duration duration, [FutureOr<T> computation()?]).- Es un constructor de un Future que realiza cualquier tarea de computación después de un tiempo dado.

¿Qué significa la “?” al final de la llamada al método computation? ¿Qué significa en general la “?” (cuando está al final de una clase o tipo primitivo)?

En Dart, la marca "?" se utiliza para indicar que un objeto o tipo puede ser nulo (o tener un valor "null").

Cuando se usa al final de una clase o tipo primitivo, se conoce como operador de nulabilidad. Esto significa que la variable de ese tipo puede contener un valor nulo además de los valores regulares de ese tipo.

En el caso del método computation() en el constructor Future.delayed(), la "?" se usa para indicar que el parámetro es opcional, lo que significa que puede ser nulo o tener un valor. Si no se proporciona un valor para este parámetro, se asignará el valor nulo por defecto.

- Uso de Future: Define los siguientes métodos en la clase Reloj:
  - Método *Future<String> marcarSegundo()*.- El método debe hacer uso del constructor de Future antes visto para crear un Future que marque un segundo y después llame al método *addSegundo()* y muestre algún mensaje para saber que ya ha pasado el segundo.
  - Método *String refreshToString(Reloj refReloj)*.- Este método debe hacer lo siguiente:
    1. Esperar a que el reloj del argumento marque un segundo (por ahora, basta llamar al método, imprimiendo el resultado),
    2. Actualizar la hora a partir del argumento, y
    3. mostrar el estado del reloj (método *toString*).

  - Imprime el resultado de llamar al método *refreshToString* para cada uno de los relojes (el argumento será siempre el primer reloj), desde el *main.*
  - Pruébalo.

¿Se han cambiado las horas? Explica por qué

Se han actualizado las horas en los relojes debido a la implementación del método `refreshToString`. En este método, primero se espera a que el reloj del argumento marque un segundo (usando el método `marcarSegundo`) y después se actualiza la hora a partir del argumento. Por lo tanto, cuando se llama al método `refreshToString` para cada uno de los relojes, se espera un segundo antes de actualizar y mostrar la hora en cada reloj, lo que resulta en la actualización correcta de las horas en cada reloj.

- async: Palabra clave justo antes del cuerpo de un método asíncrono. Exige que lo que devuelva el método sea de tipo Future<t> siendo t el

tipo que devolvería si el método fuera síncrono.

- await: Palabra clave solo de uso en métodos asíncronos. Se pone delante de la llamada a un método que devuelve un Future al que necesitamos esperar (es decir, el futuro debe estar completado, de o no error[1]).

  - Modifica todos los métodos que sean asíncronos porque deban esperar el resultado de un Future para hacer algo, añadiendo async y await donde convenga, teniendo en cuenta que hay que llegar hasta el main.

¿Qué métodos has necesitado cambiar?

Los métodos que han necesitado cambiar para añadir async y await son marcarSegundo() y refreshToString(). Además, en el método main() se hace uso de await para esperar la finalización de las llamadas a refreshToString().

¿Es necesario declarar de forma explícita como asíncrono un método que no haga uso de la palabra clave *await*? Haz pruebas para responder

No es necesario declarar explícitamente un método como asíncrono si no se hace uso de la palabra clave await. Sin embargo, si el método realiza operaciones que pueden tardar mucho tiempo en completarse, es recomendable utilizar la palabra clave async para indicar que el método se ejecutará de forma asíncrona y evitar así bloquear el hilo principal de la aplicación.
Aquí hay un ejemplo:

```
void printNumeros() {
  for (int i = 1; i <= 10; i++) {
    print(i);
  }
}

void main() {
  printNumeros();
  print("Terminado de imprimir numeros.");
}
```

Este código imprimirá los números del 1 al 10 y luego imprimirá " Terminado de imprimir numeros.". Sin embargo, si el método

[1]. Un Future tiene dos estados: completado o incompleto.

printNumeros() hiciera algo que tardara mucho tiempo, como por ejemplo leer un archivo grande, la aplicación se bloquearía durante ese tiempo y no se imprimiría nada hasta que la operación se completara. Para evitar esto, se puede declarar el método como asíncrono:

```
Future<void> printNumerosAsync() async {
  for (int i = 1; i <= 10; i++) {
    await Future.delayed(Duration(seconds: 1));
    print(i);
  }
}

void main() async {
  await printNumerosAsync();
  print("Terminado de imprimir numeros.");
}
```

En este ejemplo, el método printNumerosAsync() espera un segundo después de imprimir cada número para simular una operación que tarda tiempo en completarse. El método es declarado como asíncrono y devuelve un Future<void>, lo que permite que se utilice la palabra clave await dentro del bucle para esperar cada segundo. En el método main(), se utiliza await para esperar a que se complete la ejecución de printNumerosAsync() antes de imprimir " Terminado de imprimir numeros.".

- Pruébalo todo

Resultados:

Inserta aquí una captura de pantalla con la salida resultante de ejecutar el programa final:

```
Restarted application in 128ms.
I/flutter (17024): constante
I/flutter (17024): Instance of 'Object'
I/flutter (17024): Segundo pasado
I/flutter (17024): Reloj{hora(s): 3, minuto(s): 30, segundo(s): 34, total de horas en minutos: 210
I/flutter (17024): Segundo pasado
I/flutter (17024): Reloj{hora(s): 3, minuto(s): 30, segundo(s): 35, total de horas en minutos: 210
I/flutter (17024): Segundo pasado
2 I/flutter (17024): Reloj{hora(s): 3, minuto(s): 30, segundo(s): 36, total de horas en minutos: 210
I/flutter (17024): Total horas en minutos: 210
I/flutter (17024): Horas totales (con parte decimal): 3.5
I/flutter (17024): Horas completas (division entera): 3
```

Se subirá este fichero a Prado con el cuadro anterior relleno.