Week 12: Lab 9

CS2030S Lab 16B

Chrysline & Alissa

Overview

1. Recap
2. Mock PE
3. Mock PE Solution

1: Recap

Unit 38 - Threads

- A single flow of execution
- Used to run multiple processes in separate threads at the same time
- `new Thread(Runnable)`
- Runnable does some task and returns nothing (kind of like a producer)
- Threads do not start running until you do `.start()` on them.
- `.start()` returns immediately, not only when the task/thread has finished executing
- Asynchronous execution

Unit 38 - Threads

- `Thread.sleep(milliseconds)`
    - Pauses execution for a period of time (artificial delay)
    - Can be used to repeat a task at time intervals
- `thread.isAlive()`
    - Checks if a thread is still running

Unit 39 - Async

- CompletableFuture
    - A monad that encapsulates a value that is there or not there yet.
    - A "promise"
    - Helps you to specify asynchronous tasks without worrying about details like catching exceptions, communicating between threads etc.
    - `.get()` or `.join()` blocks/waits for all concurrent tasks to finish and return the final value
    - Synchronous!

Unit 39 - Async (CF)

Creating a CF:

- `static <U> CF<U> completedFuture(value)`.
- `static CF<void> runAsync(Runnable)`
- `static <T> CF<T> supplyAsync(Supplier<T>)`

Using CFs:

- `CF<void> allOf(CF…)`
- `CF<Object> anyOf(CF…)`
- thenApply (map), thenCompose (flatmap), thenCombine (combine)
- thenApplyAsync, thenComposeAsync, thenCombineAsync

Unit 39 - Async (CF)

Handling exceptions:

- `.handle(BiFunction<value, exception, return value)`.
- Either value or exception will be null.
- Execution succeeds -> have value, null exception.
- Execution fails -> null value, have exception.
- Example of usage:

```
.handle((t, e) -> (e == null) ? t : 0)
```

Mock PE

That’s all for today! Thanks for coming!

Feedback