Kotlin 4.1 - Coroutines

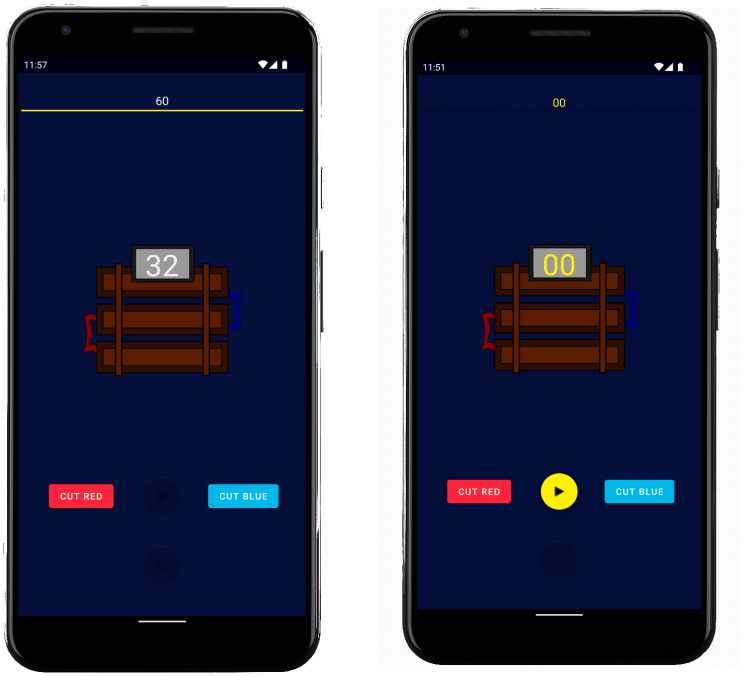
Aufgaben

1. Zeitbombe - Coroutines

In dieser Aufgabe erstellen wir eine Zeitbombe App. Die App besteht aus einem **TimerFragment**, welches sich ausschließlich um die UI Ein- und Ausgaben kümmert. Dem **TimerFragmentViewModel,** in dem die Logik für den Counter implementiert wird und der **MainActivity** als Fragmenthalter.

Der Timer kann über das Inputfeld (oberstes Element mit 00) gesetzt werden.
Die Zeit kann zwischen 0-60 Sekunden gestellt werden.
Der Timer soll mit Hilfe des “CUT BLUE“ Buttons angehalten werden können.
Der “CUT RED” Button lässt den Timer schneller ablaufen.
Mit dem Reset Button ist es möglich alle Funktionen zurück zu setzten.

Für diese Aufgabe muss lediglich das ViewModel bearbeitet werden. Erläuterungen zu den bestehenden Attributen und Funktionen des ViewModel findet Ihr auf Seite 3.



Ergänze die Aufgabe an den mit TODO markierten Stellen.

- Erstelle eine lateinit Variable *currentJob* vom Datentypen ***Job***
- Erstelle eine Variable *delay* vom Datentypen ***Long*** mit einem default Wert 1000
- Implementiere die Funktion **countDownTime**(timeString: String)
    - Die Funktion bekommt die Zeit vom Inputfeld übergeben und zählt diese runter
    - Setze die value der LiveData Variable **_stringTime** auf die übergebene Zeit (timeString)
    - konvertiere die übergebene Zeit vom Datentypen **String** → zum Datentyp **Long** mithilfe der ***convertTimeToMillis(timeString)*** *Funktion*
    - setzte die **_countDownInitiated** Variable auf true
    - Damit das UI während des Countdowns aktualisiert werden kann, muss der Countdown in einer Coroutine implementiert werden
    - Speichere den Coroutine Job in der Variable **currentJob**
    - In der Coroutine muss zunächst die **_countDownActive.value** auf true gesetzt werden
        - Diese Variablenzuweisung funktioniert nur im Main Thread, deshalb muss der Dispatcher auf Main gesetzt werden siehe Hinweis.
    - Jetzt kann eine **while** Schleife in der Coroutine die Sekunden herunterzählen
        - Solange timeInMillis größer ist als 0:
            - Soll hier eine Verzögerung um den Wert der _delay Variable stattfinden
            - timeInMillis soll um -1000 reduziert werden
            - Die übrige Zeit soll mithilfe der **convertTimeToString()** Funktion in einen String umgewandelt werden
                - Achtung: convertTimeToString() beinhaltet eine Variablenzuweisung, wie zuvor Hinweis beachten
            - Nach dem die Schleife durchgelaufen ist, soll in der Coroutine noch die **_countDownActive.value** wieder zu false geändert werden und der _delay auf 1000 gesetzt werden. - Hinweis beachten
- Implementiere die Funktion **stopCurrentJob()**
    - Jobs / Coroutines können mit hilfe Aufruf der cancel() Funktion angehalten werden
- fastRunCurrentJob()
    - verändere _delay auf den Wert 10

HINWEIS: Um den Thread zu verändern, in dem eine Coroutine läuft, ruft folgendes in der Coroutine auf:

withContext(Dispatchers.<Thread>){ variable = <WERT> }

TimerFragmentViewModel

ATTRIUBUTE

_timeInMillis: Long - speichert die Zeit

_stringTime: MutableLiveData<String> - LiveData, in dem die Zeit als String gespeichert wird

stringTime: LiveData<String> - TimerFragment beobachtet diese Variable und passt das UI an

_countDownInitiated: Boolean - Wurde ein countDown gestartet

countDownInitiated: Boolean - Read Zugriff auf private _countDownInitiated Variable von außen

_countDownActive: MutableLiveData<Boolean> - Läuft der Timer noch?

countDownAcivte: LiveData<Boolean> - TimerFragment beobachtet diese Variable und passt das UI an

FUNKTIONEN

priavte **convertTimeToString (millis: Long)** - konvertiert Millisekunden zu sekunden und speichert sie als String in der _stringTime Variable

priva**te convertTimetoMillis (timeString: String)** - konvertiert Sekunden vom Typ String zu Millisekunden vom Typ Long und speichert sie in der Variable _timeInMillis

resetCountDown() - setzt ***_countDownInitiated*** *auf false zurück*

Viel Erfolg! 😉