Basics of Kotlin Flow

Basics of Kotlin Flow

- What is a Flow?
- Reactive Programming
- Flow Builders
- Flow Operators
  - Lifecycle Operators
  - Terminal Operators
  - Intermediate Operators

What is a Flow?

What is a Flow?

"a stream of values

that are computed asynchronously"

What is a Flow?

"a stream of values

that are computed asynchronously"

asynchronous data stream

Converting a function that returns a single value to an asynchronous data stream

return type	# of return values	when	characteristic
BigInteger	single value	once	blocking
List<BigInteger>	multiple values	once	blocking
Sequence<BigInteger>	multiple values	continuously	blocking and synchronous
Flow<BigInteger>	multiple values	continuously	suspending and asynchronous

Suspend function

vs

Flow

Reactive Programming


Observer
Pattern

LiveData

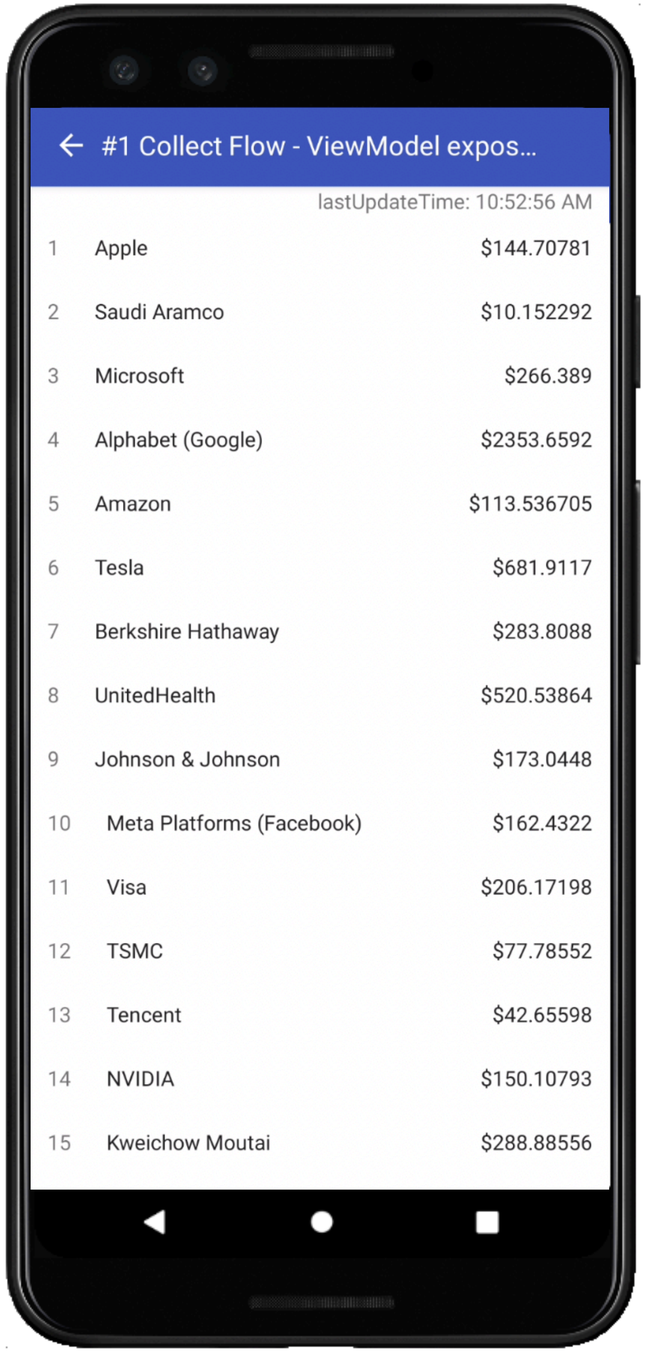

Activity
observe LiveData property
& react to changes
ViewModel

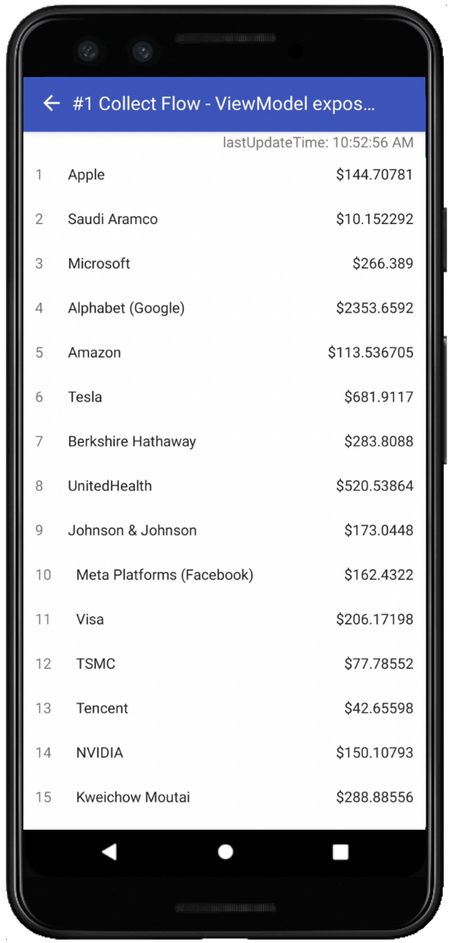
Reactive Approach	Imperative Approach
Expose a Flow	Expose a Suspend Function
other components can subscribe & receive changes	other components need to periodically check for data changes

First

Kotlin Flow

Usecase


#1 Collect Flow - ViewModel expos...
lastUpdateTime: 10:52:56 AM
1 Apple $144.70781
2 Saudi Aramco $10.152292
3 Microsoft $266.389
4 Alphabet (Google) $2353.6592
5 Amazon $113.536705
6 Tesla $681.9117
7 Berkshire Hathaway $283.8088
8 UnitedHealth $520.53864
9 Johnson & Johnson $173.0448
10 Meta Platforms (Facebook) $162.4322
11 Visa $206.17198
12 TSMC $77.78552
13 Tencent $42.65598
14 NVIDIA $150.10793
15 Kweichow Moutai $288.88556



Activity

renders UI

ViewModel

LiveData<UiState>

mocked Endpoint

Flow<Stock>

DataSource

Flow Builders

Basic Flow Builders

Flow Builder	Description	Example
flowOf()	create a Flow from a fixed set of values	`flowOf(1,2,3)`
.asFlow()	extension function on various types to convert them into Flows	`listOf("A", "B", "C")` `.asFlow()`

flow{}	builder function to construct arbitrary flows
	from sequential calls to the emit function
		`flow {`
		`emit("one")`
		`delay(100)`
		`emit("two")`
		`}`

Terminal Operators

Terminal Operators

- `collect { }`
- `first()`
- `last()`
- `single()`
- `toList(), toSet()`
- `fold(), reduce()`

Terminal operator

```
launchIn()
```

Lifecycle operators

Terminal operator

```
.asLiveData()
```

Basic

Intermediate

Operators

Basics of Kotlin Flow

Recap 🎓

What is a Flow?

"a stream of values

that are computed asynchronously"

```
val currentStockPriceAsLiveData: LiveData<UiState> = stockPriceDataSource
    .latestStockList
    .map { stockList: List<Stock> ->
        UiState.Success(stockList) as UiState
    }
    .onStart { this: FlowCollector<UiState>
        emit(UiState.Loading)
    }
    .asLiveData()
```

```
override val latestStockList: Flow<List<Stock>> = flow { this: FlowCollector<List<Stock>>
    while (true) {
        Timber.tag( tag: "Flow").d( message: "Fetching current stock prices")
        val currentStockList : List<Stock> = mockApi.getCurrentStockPrices()
        emit(currentStockList)
        delay( timeMillis: 5_000)
    }
}
```

Flow Builder

```
val currentStockPriceAsLiveData: LiveData<UiState> = stockPriceDataSource
    .latestStockList
    .map { stockList : List<Stock> ->
        UiState.Success(stockList) as UiState
    }
    .onStart { this: FlowCollector<UiState>
        emit(UiState.Loading)
    }
    .asLiveData()
```

Basic Flow Builders

Flow Builder	Description	Example
flowOf()	create a Flow from a fixed set of values	`flowOf(1,2,3)`
.asFlow()	extension function on various types to convert them into Flows	`listOf("A", "B", "C")` `.asFlow()`

flow{}	builder function to construct arbitrary flows
	from sequential calls to the emit function
		`flow {`
		`emit("one")`
		`delay(100)`
		`emit("two")`
		`}`

```
val currentStockPriceAsLiveData: LiveData<UiState> = stockPriceDataSource
    .latestStockList
    .map { stockList: List<Stock> ->
        UiState.Success(stockList) as UiState
    }
    .onStart { this: FlowCollector<UiState>
        emit(UiState.Loading)
    }
    .asLiveData()
```

Lifecycle Operator

Lifecycle Operators

- `onStart{}`
- `onCompleted{}`

```
val currentStockPriceAsLiveData: LiveData<UiState> = stockPriceDataSource
    .latestStockList
    .map { stockList: List<Stock> ->
        UiState.Success(stockList) as UiState
    }
    .onStart { this: FlowCollector<UiState>
        emit(UiState.Loading)
    }
    .asLiveData()
```

Terminal Operator

Terminal Operators

- `collect{}`
- `first(), last(), single()`
- `toList(), toSet()`
- `launchIn()`
- `asLiveData()`

```
val currentStockPriceAsLiveData: LiveData<UiState> = stockPriceDataSource
    .latestStockList
    .map { stockList: List<Stock> ->
        UiState.Success(stockList) as UiState
    }
    .onStart { this: FlowCollector<UiState>
        emit(UiState.Loading)
    }
    .asLiveData()
```

Intermediate Operator

Basic Intermediate Operators

- `map{}`
- `filter{}`
- `take(), drop()`
- `transform{}`
- `withIndex()`
- `distinctUntilChanged()`