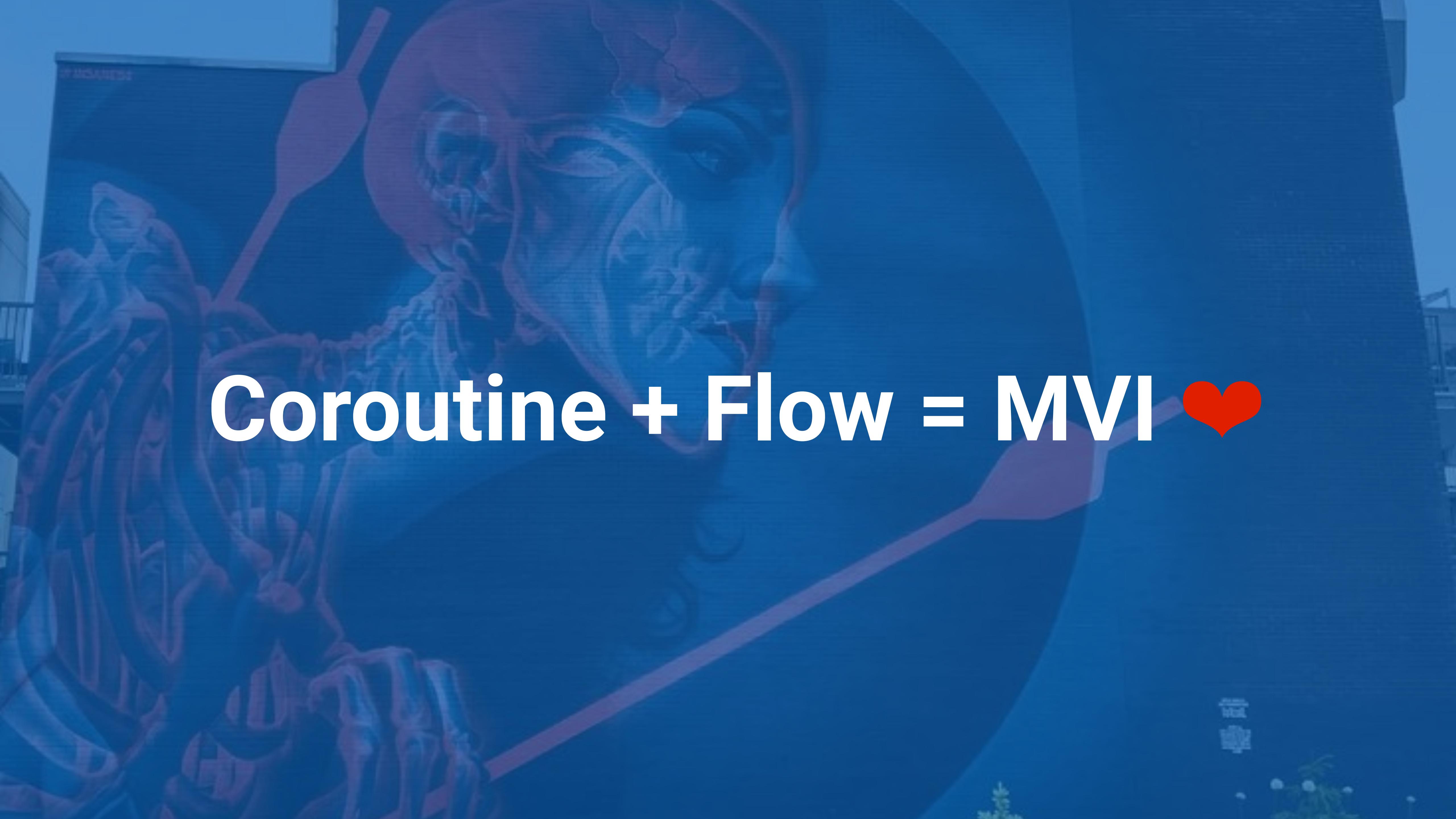
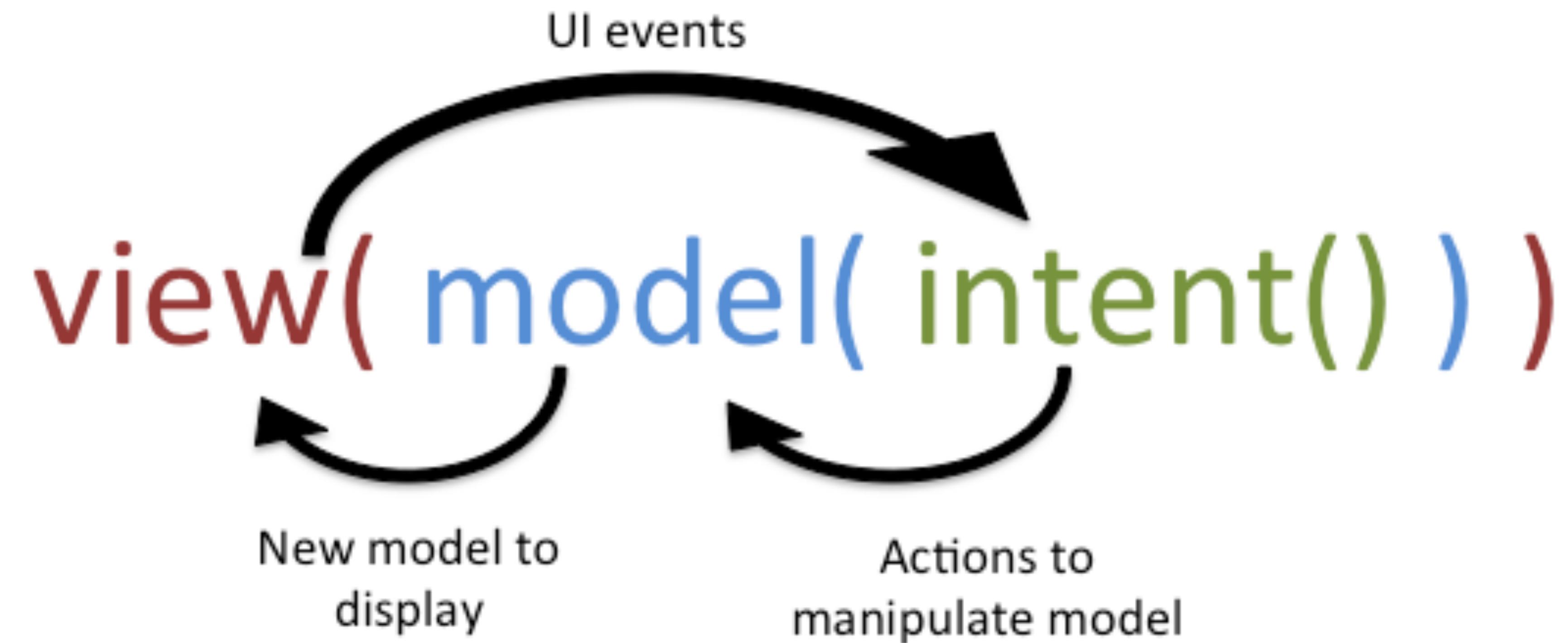




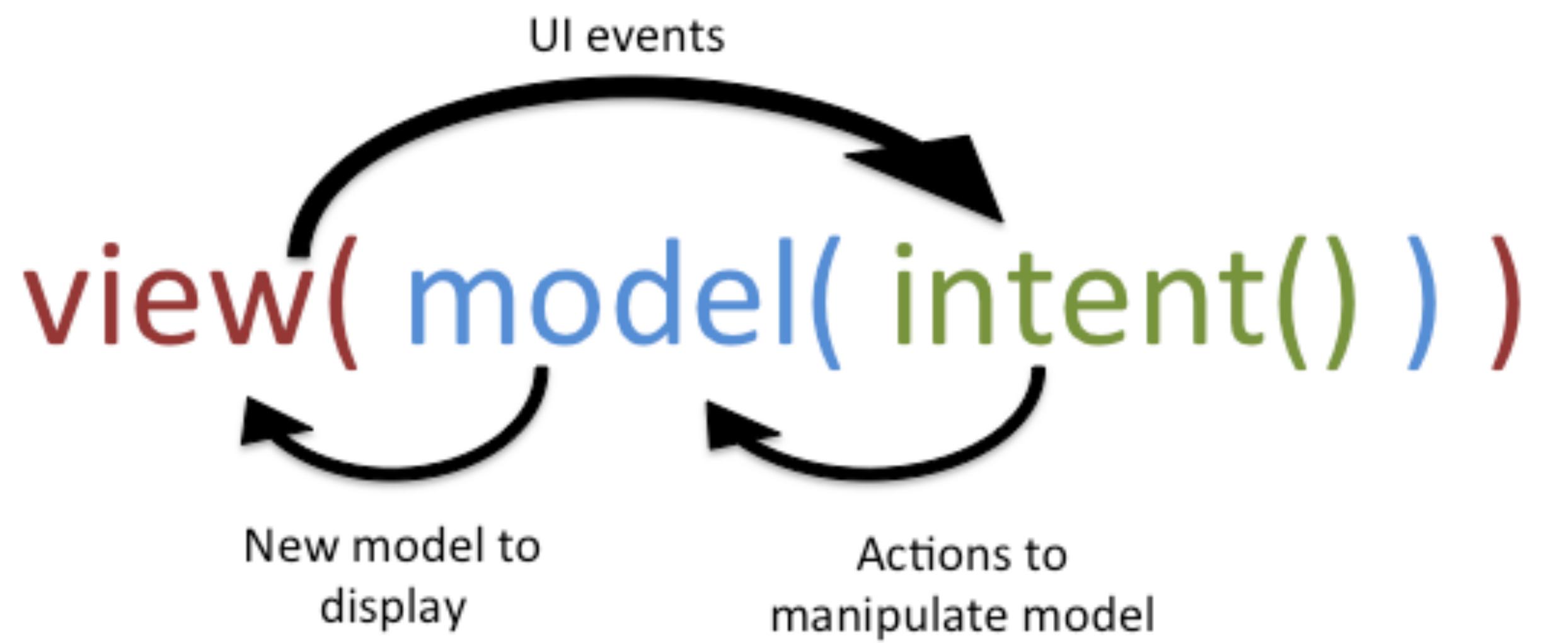
# Coroutine + Flow



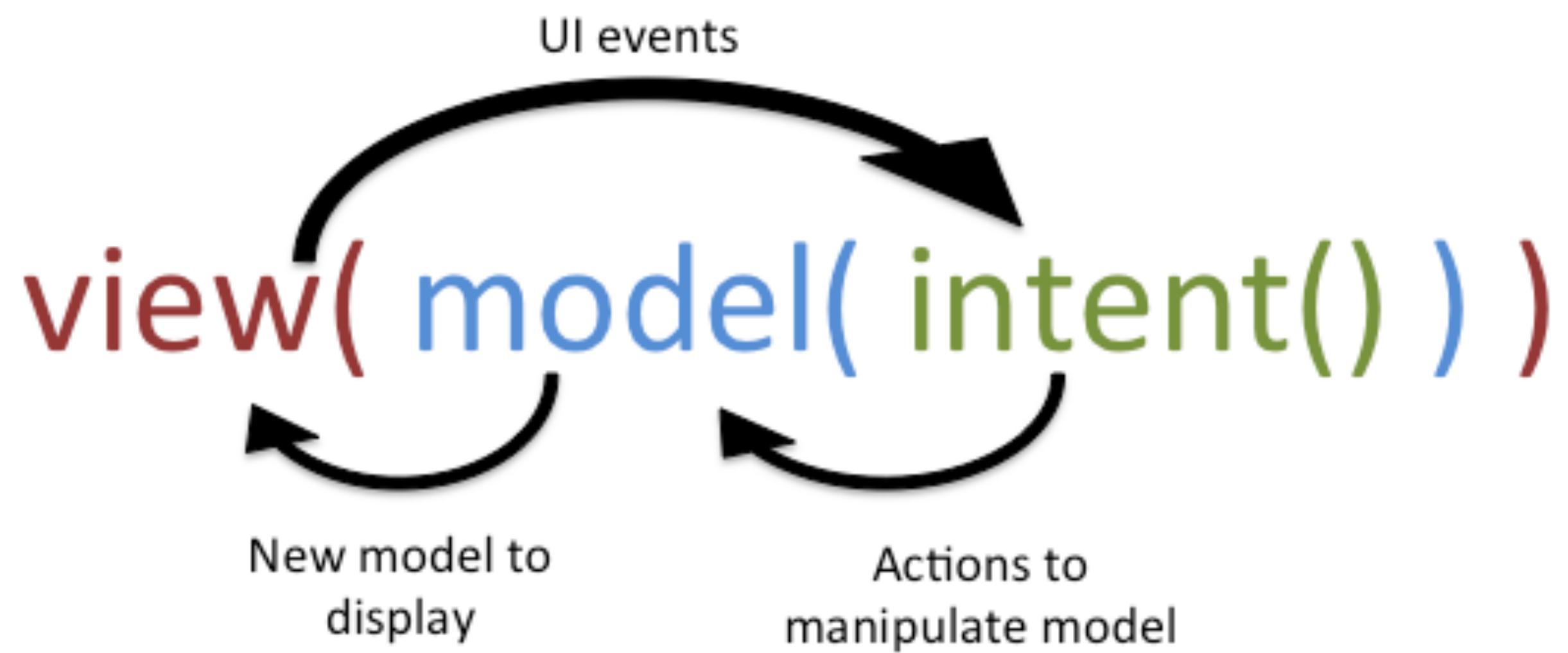
Coroutine + Flow = MVI ❤



<http://hannesdorfmann.com/android/mosby3-mvi-1>



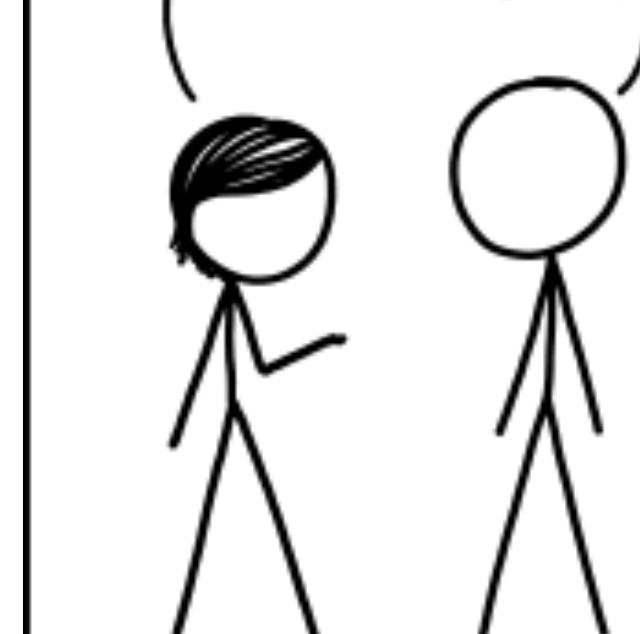
$$f(x)$$

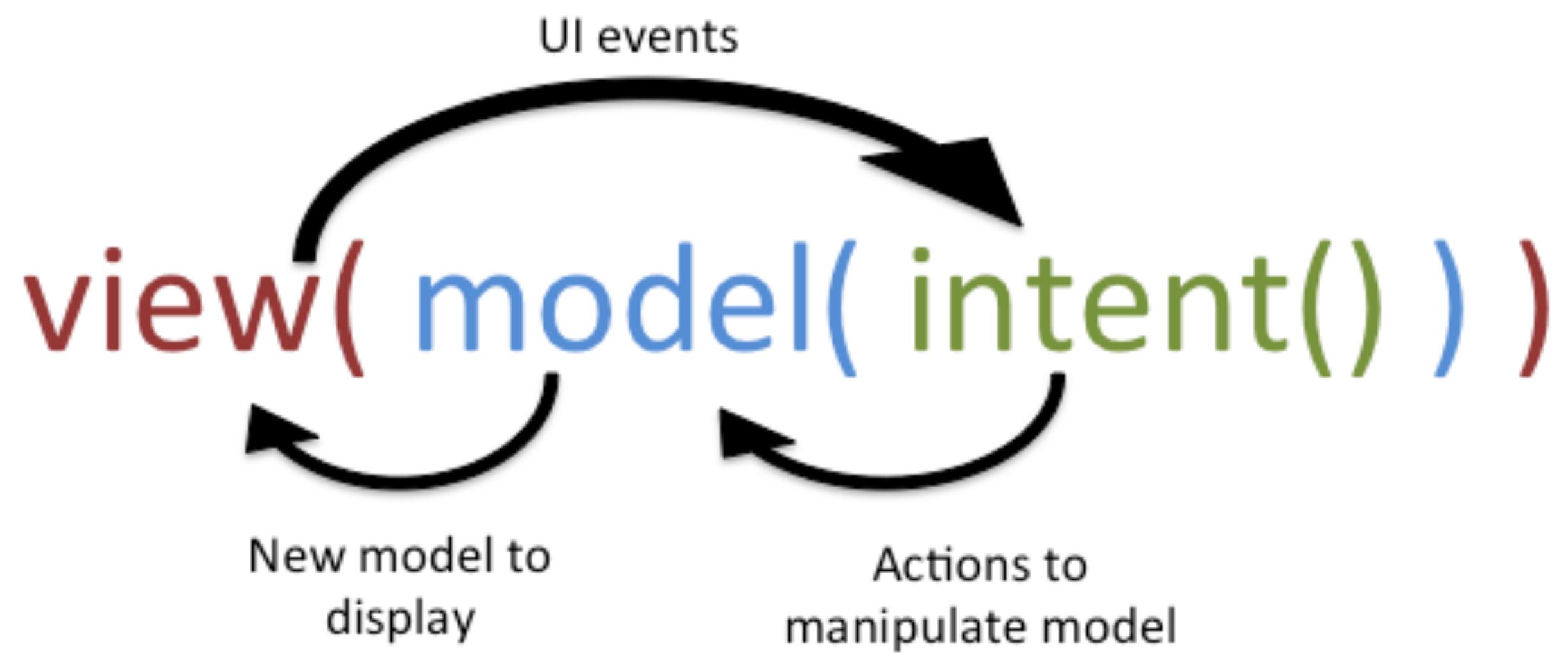


$f(x)$

CODE WRITTEN IN HASKELL  
IS GUARANTEED TO HAVE  
NO SIDE EFFECTS.

...BECAUSE NO ONE  
WILL EVER RUN IT?

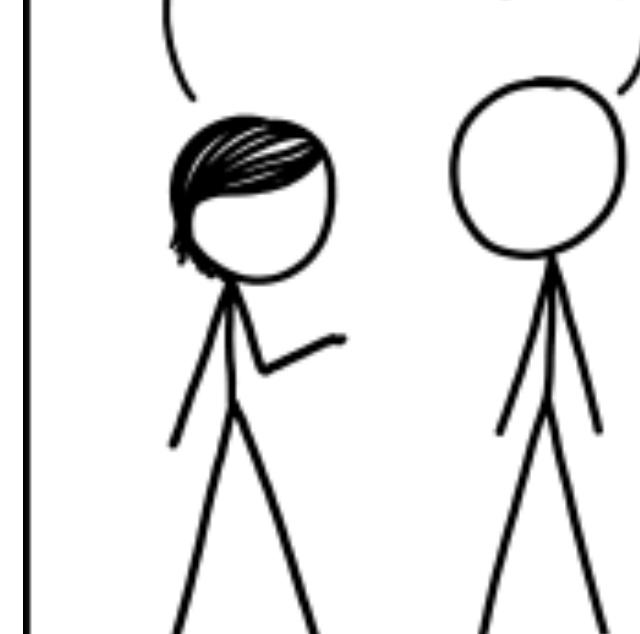


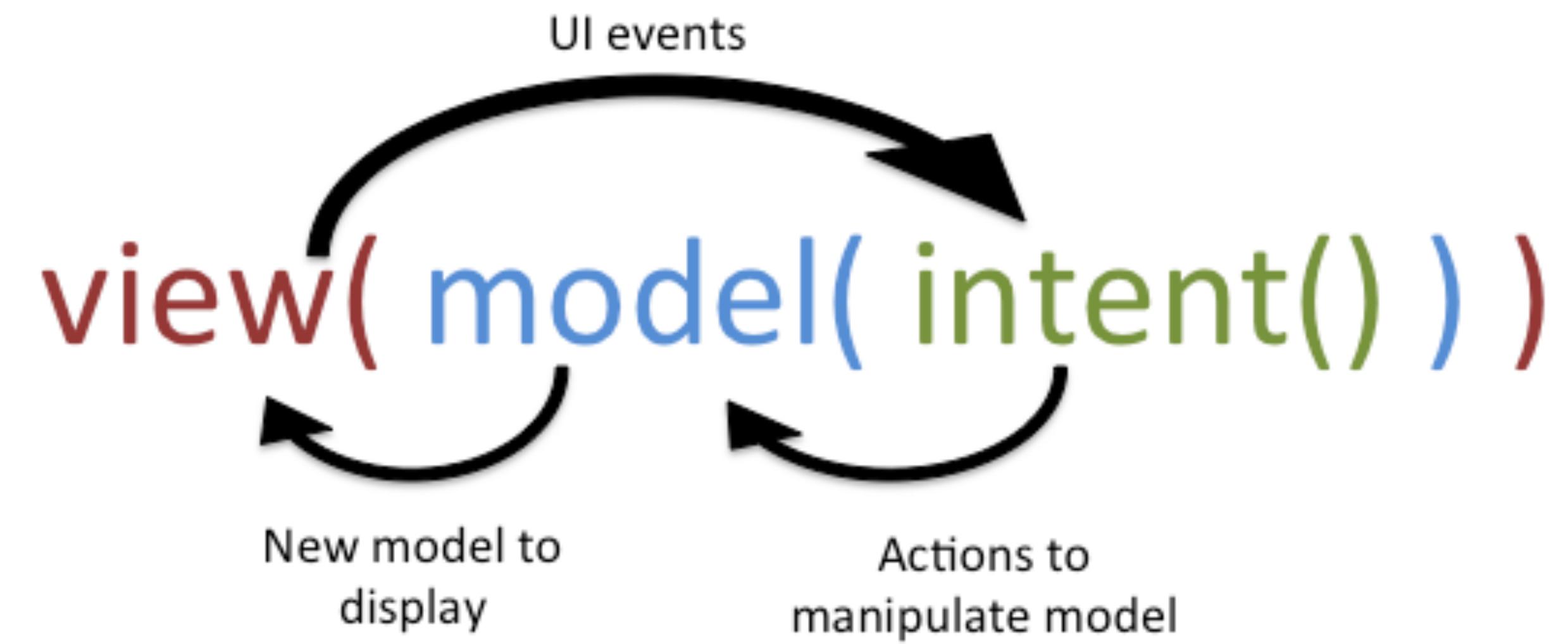


$f(x)$

CODE WRITTEN IN HASKELL  
IS GUARANTEED TO HAVE  
NO SIDE EFFECTS.

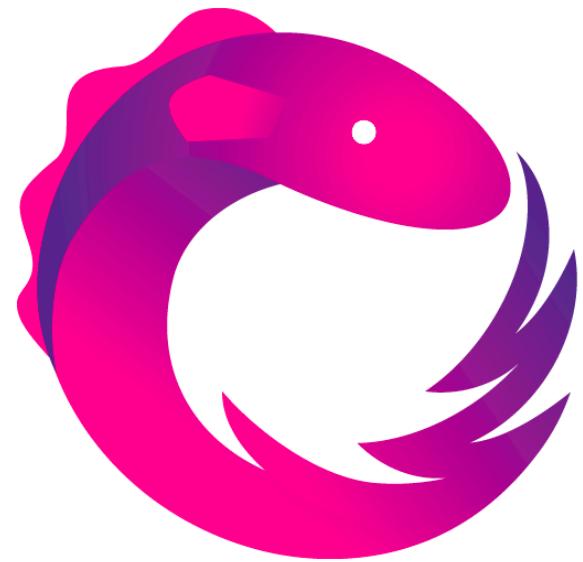
...BECAUSE NO ONE  
WILL EVER RUN IT?











```
private val disposables = CompositeDisposable()  
  
disposables += streamA().subscribe( )  
disposables += streamB().subscribe( )  
  
disposables.clear()
```

A collage of three images: a woman with curly hair smiling, a man in a dark suit, and a man wearing round-rimmed glasses.

# Coroutines & Flow

# Coroutines & Flow



# Coroutines & Flow

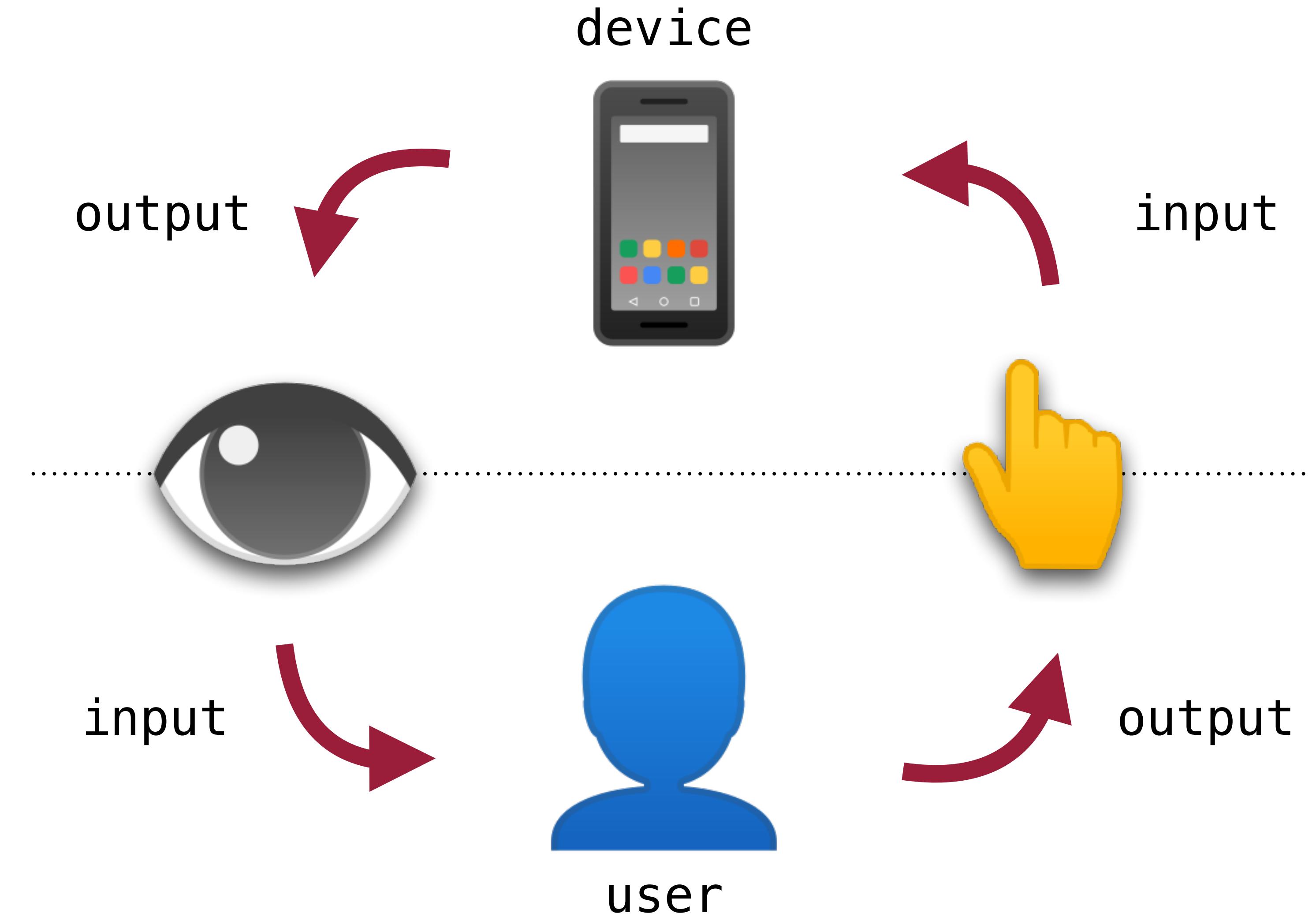


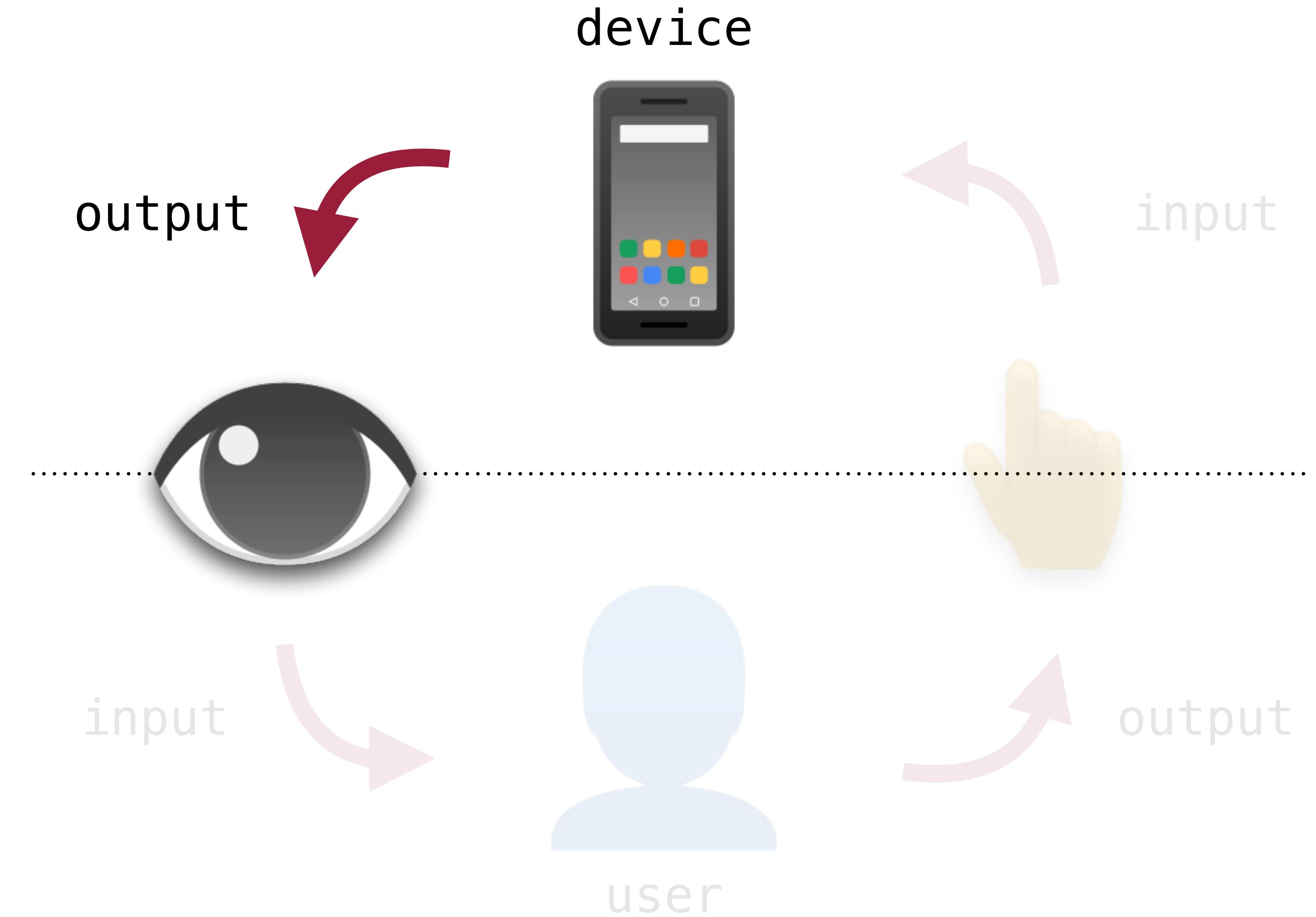
kotlinx-coroutines-android-1.3.0.jar 21 KB  
 kotlinx-coroutines-core-common-1.3.0.jar 137 KB

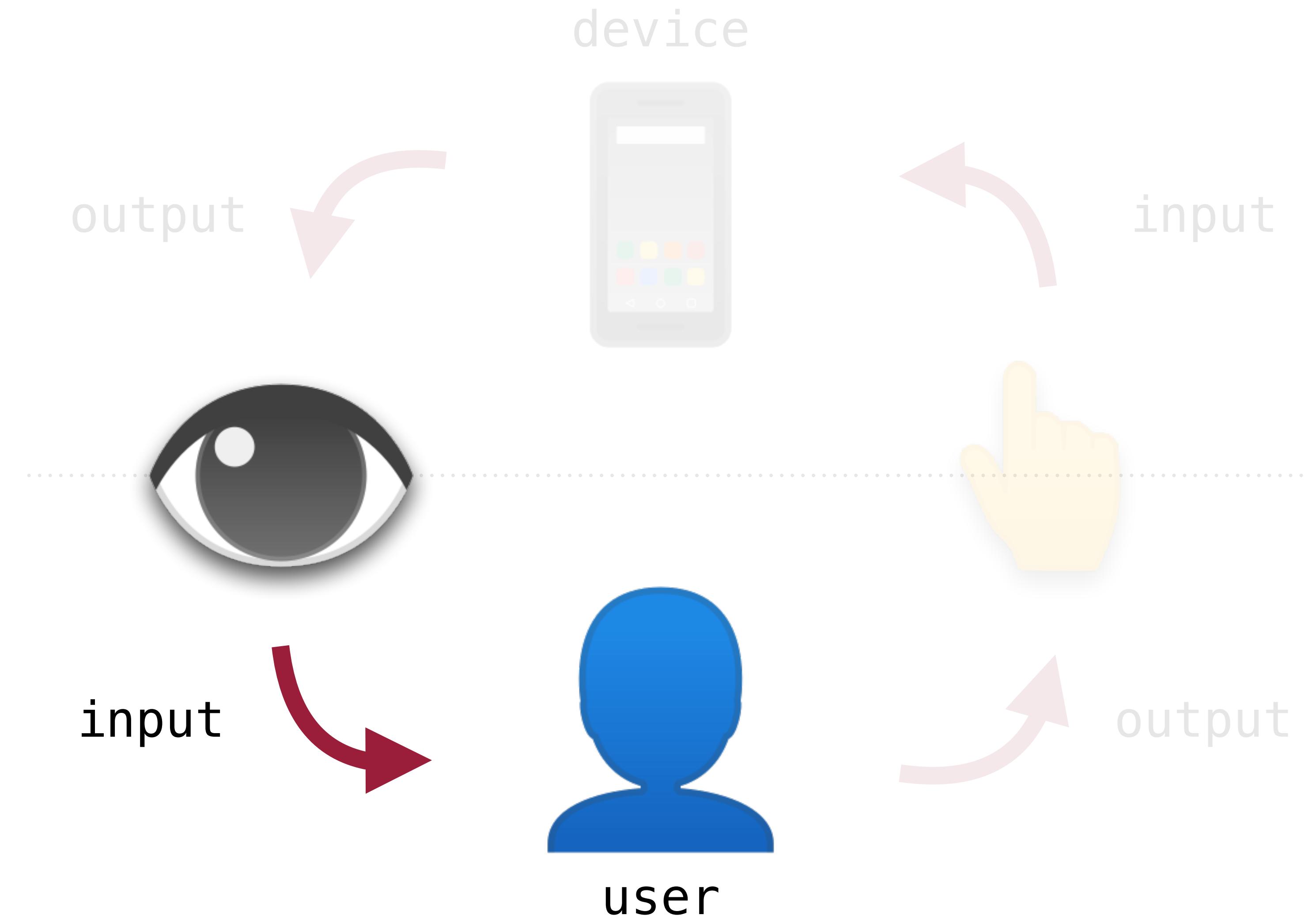


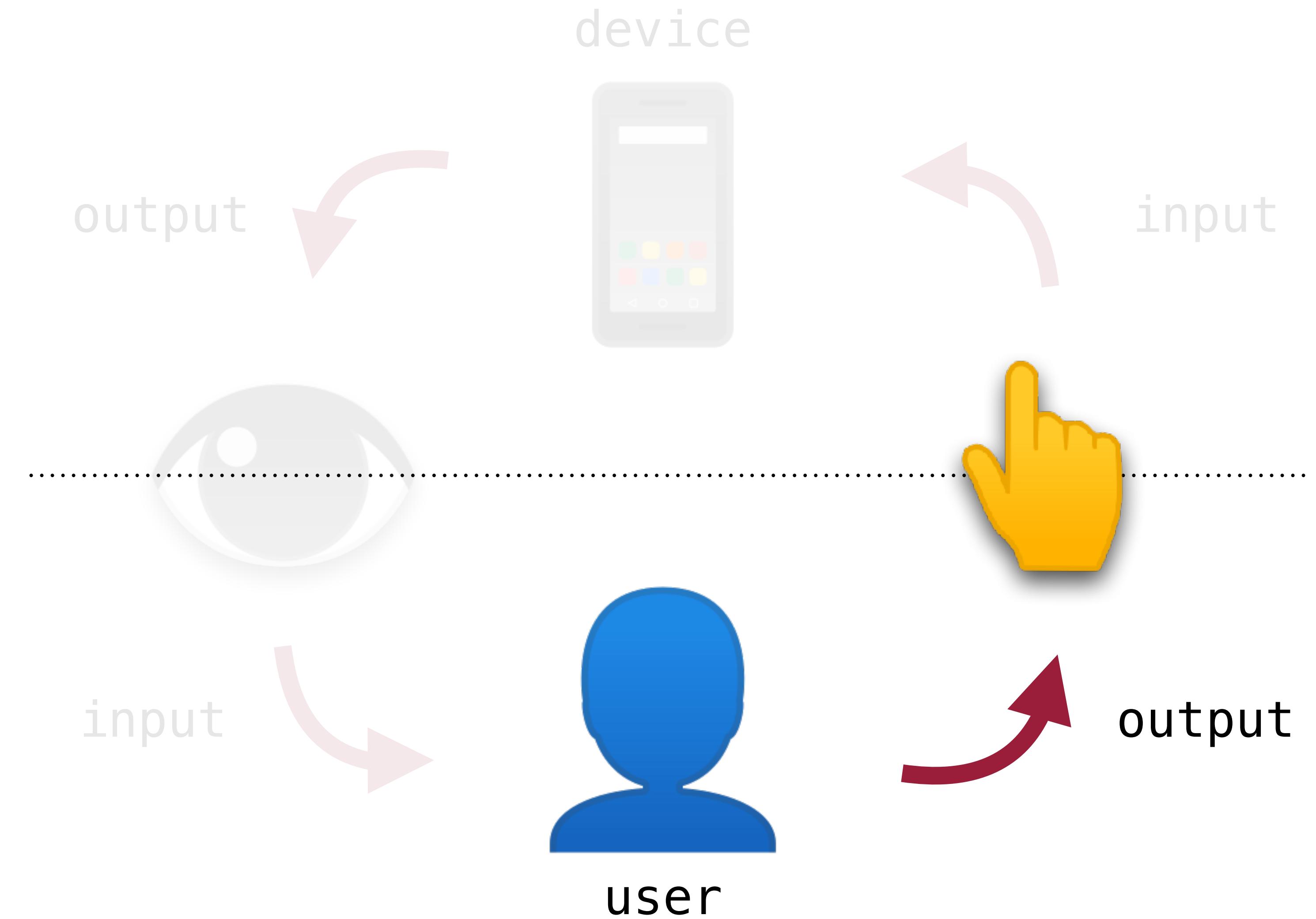
# Coroutines & Flow







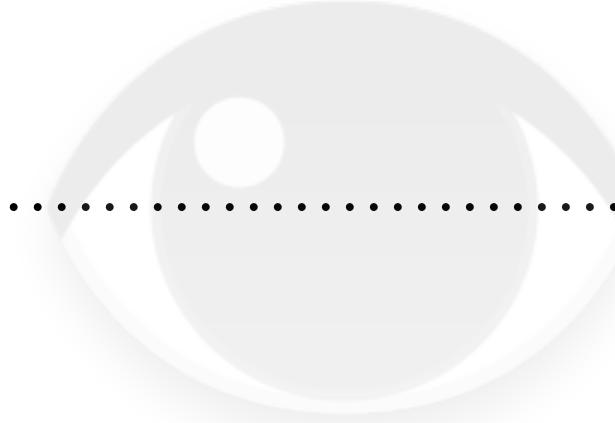




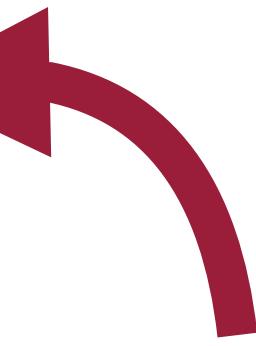
device



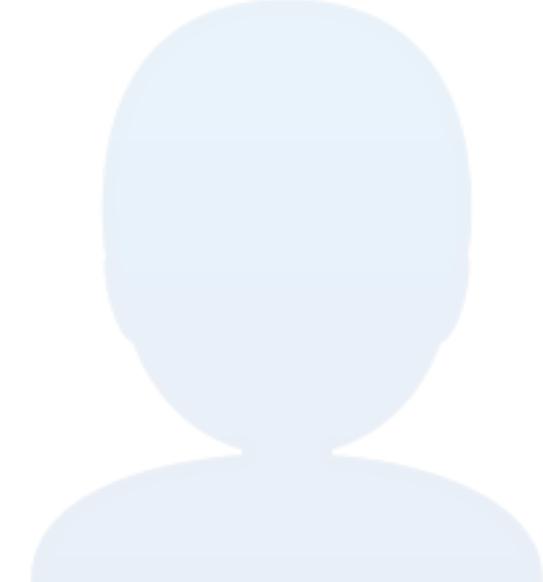
output



input

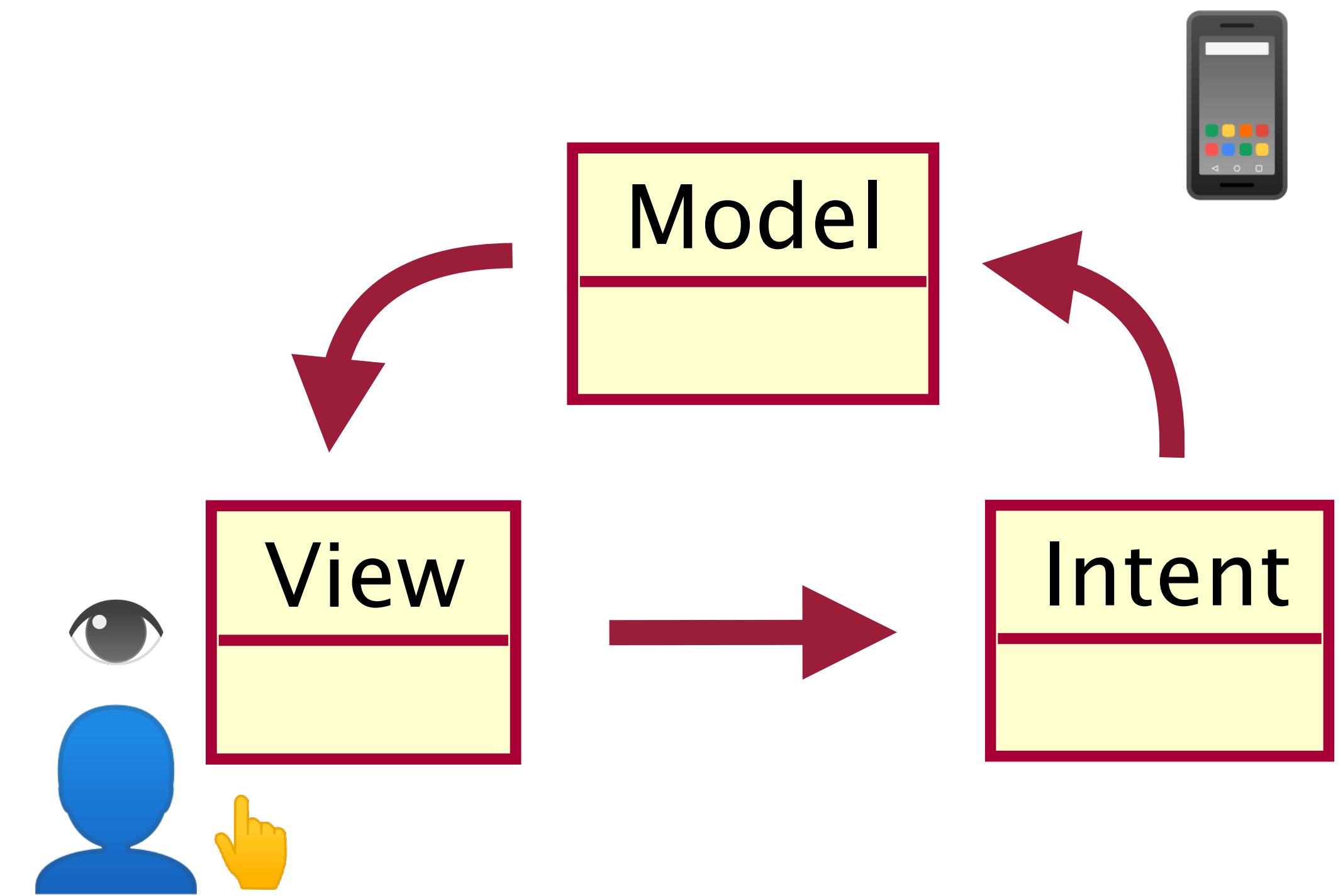


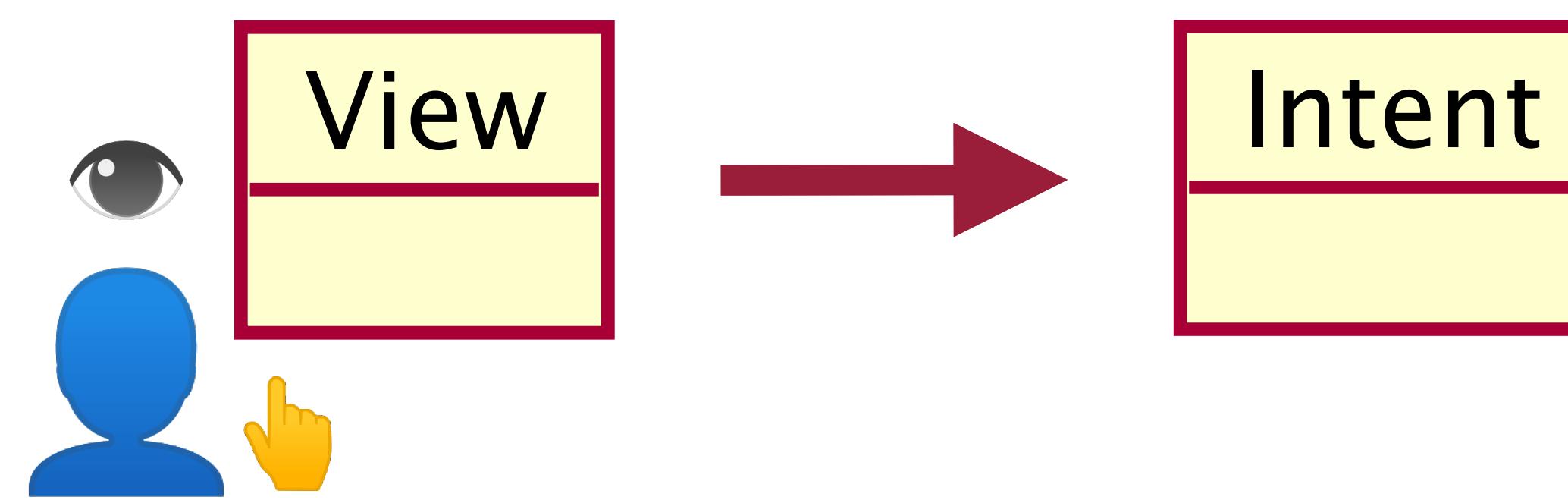
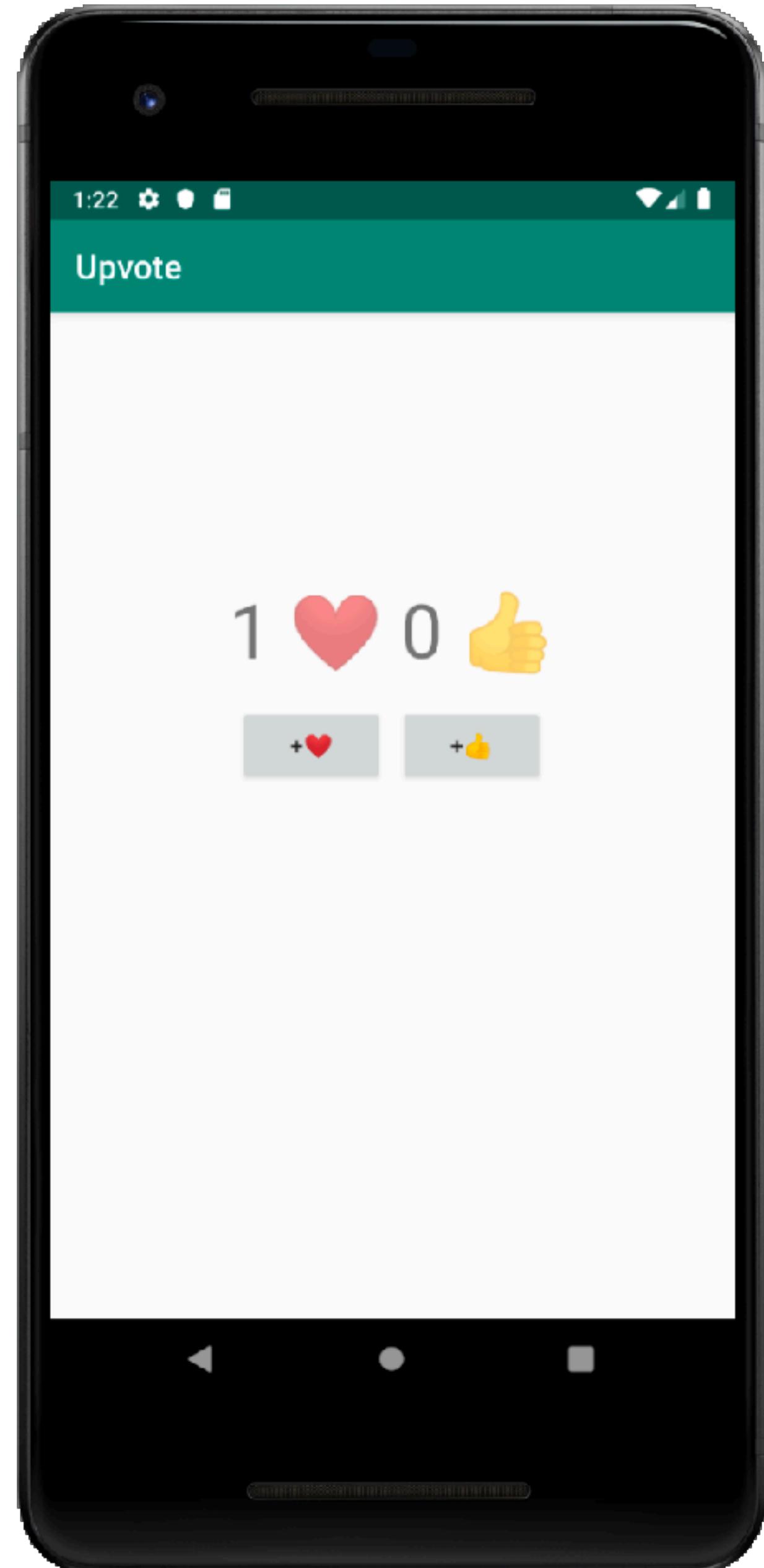
input

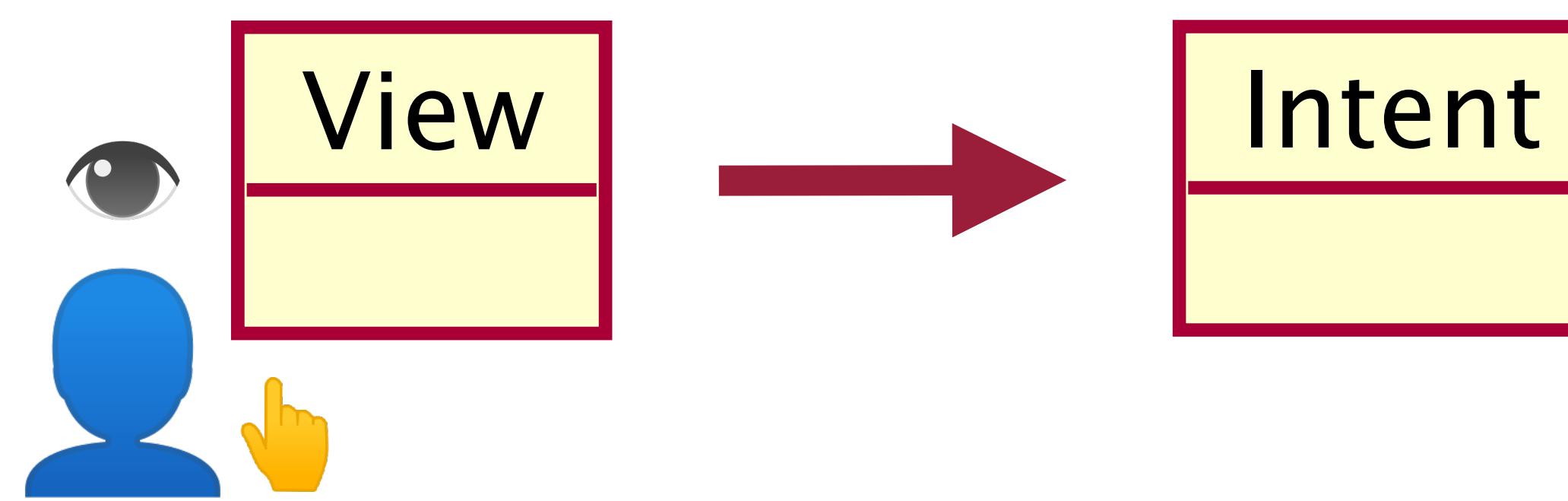
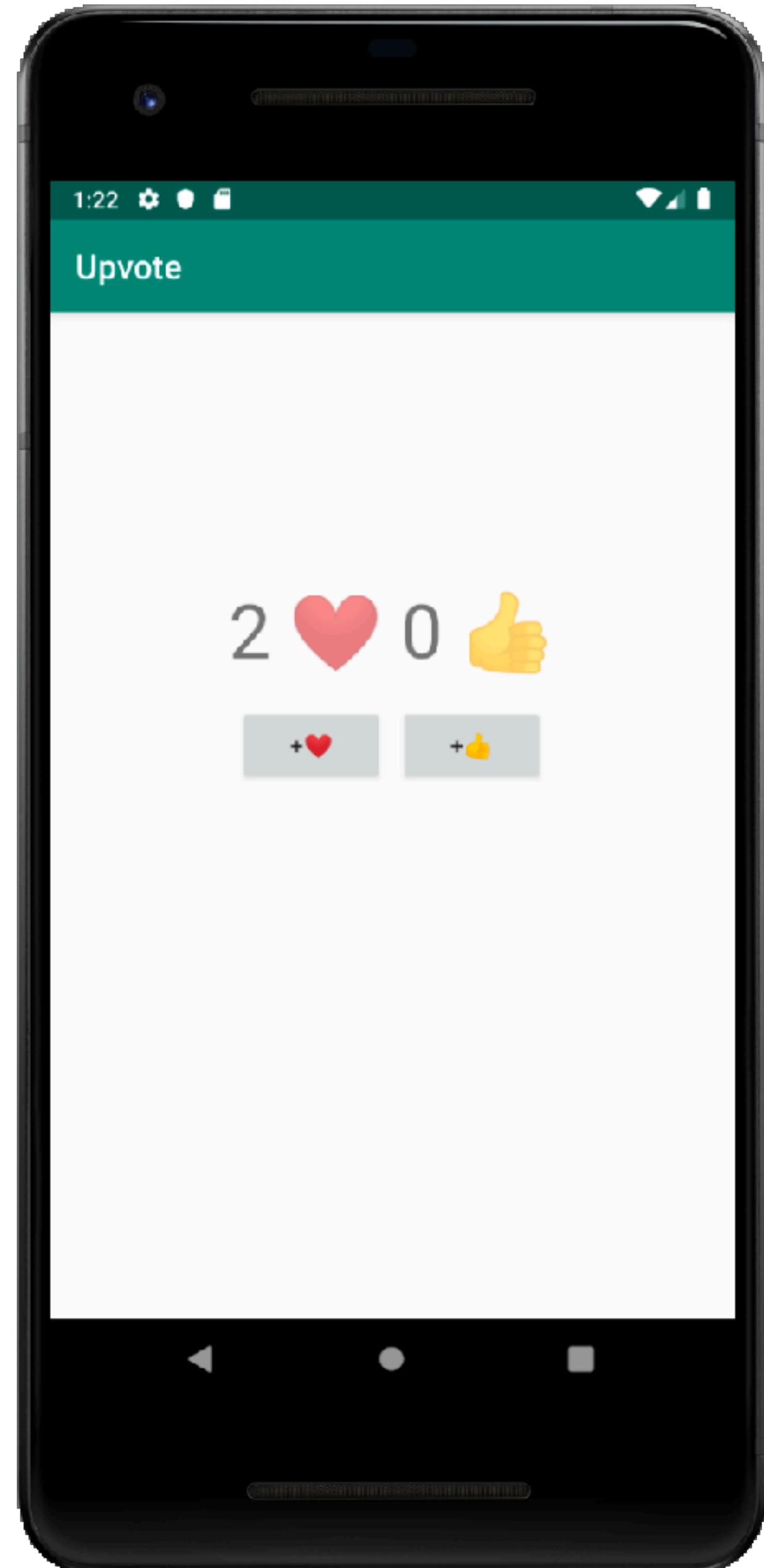


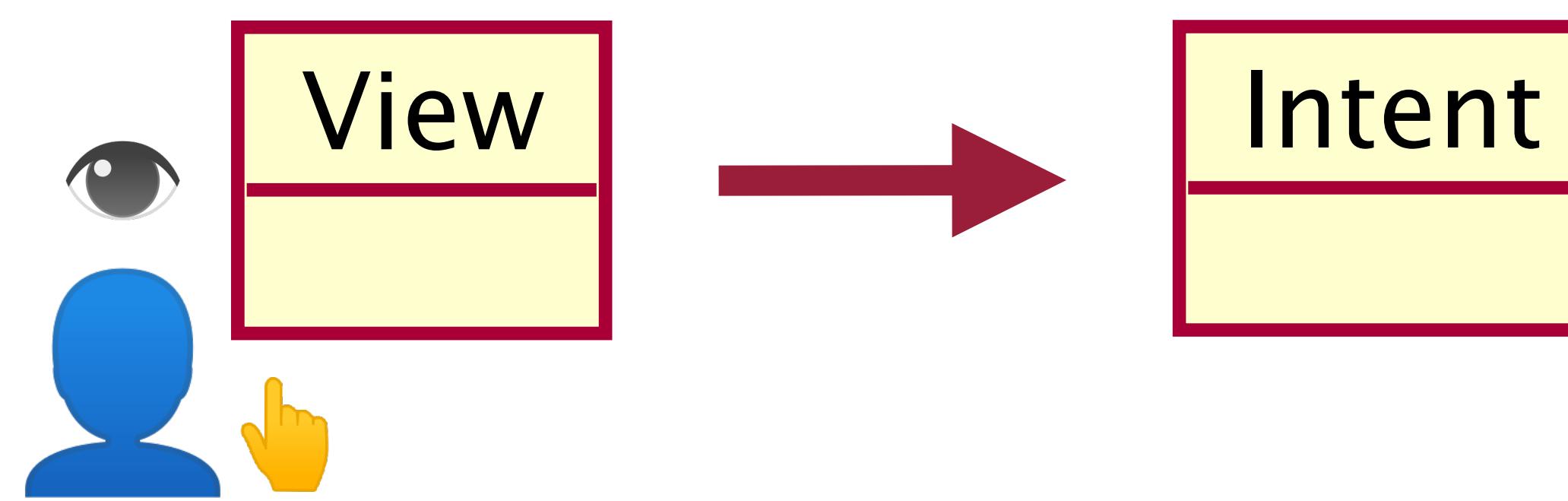
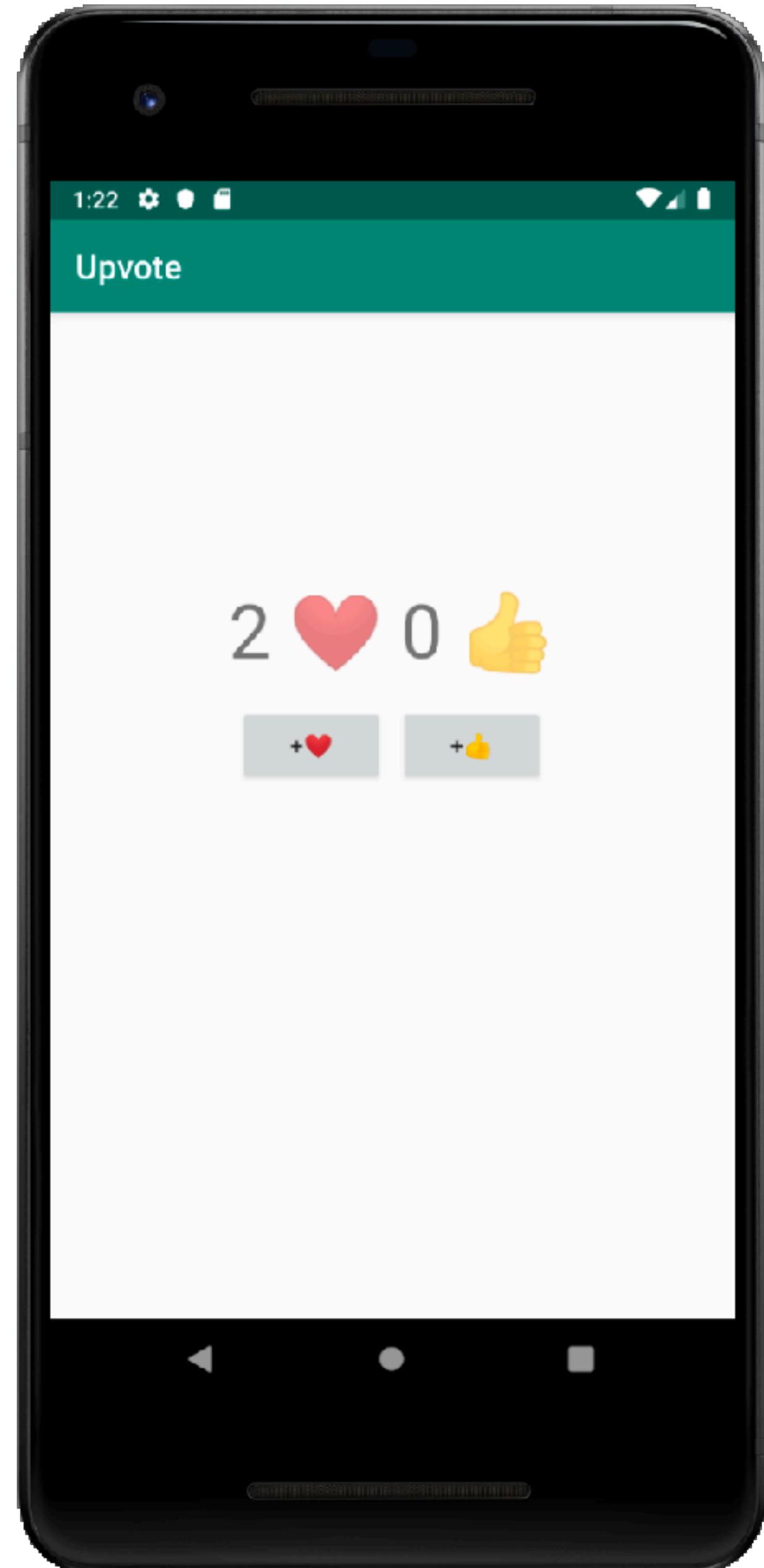
user

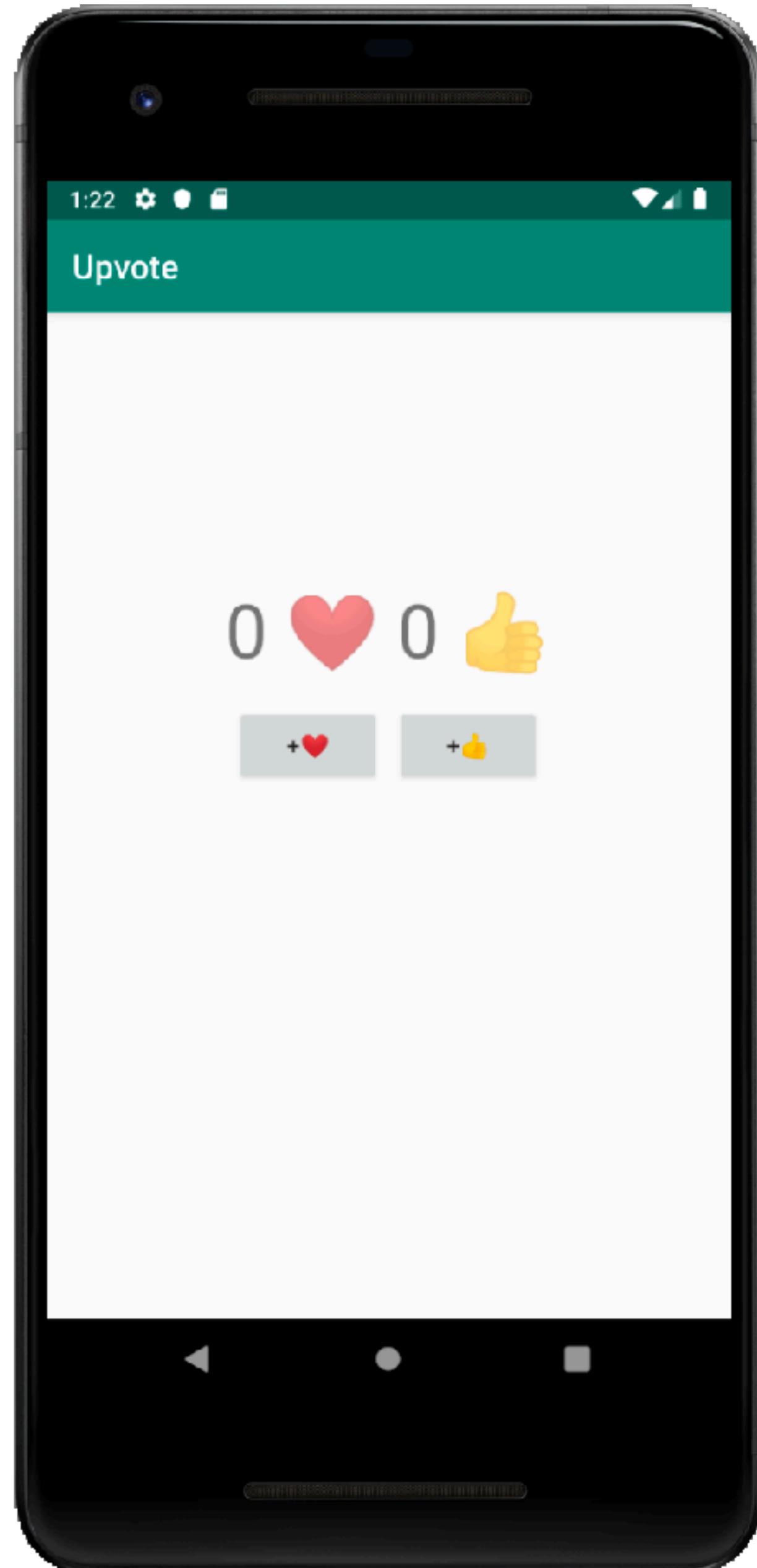
output



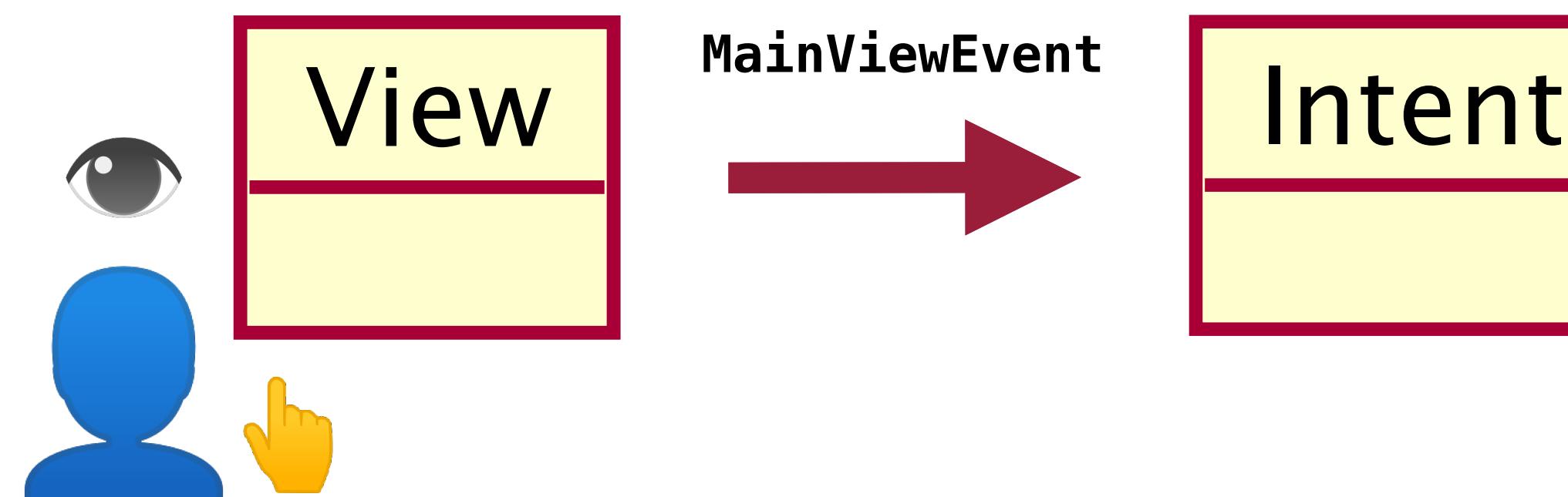


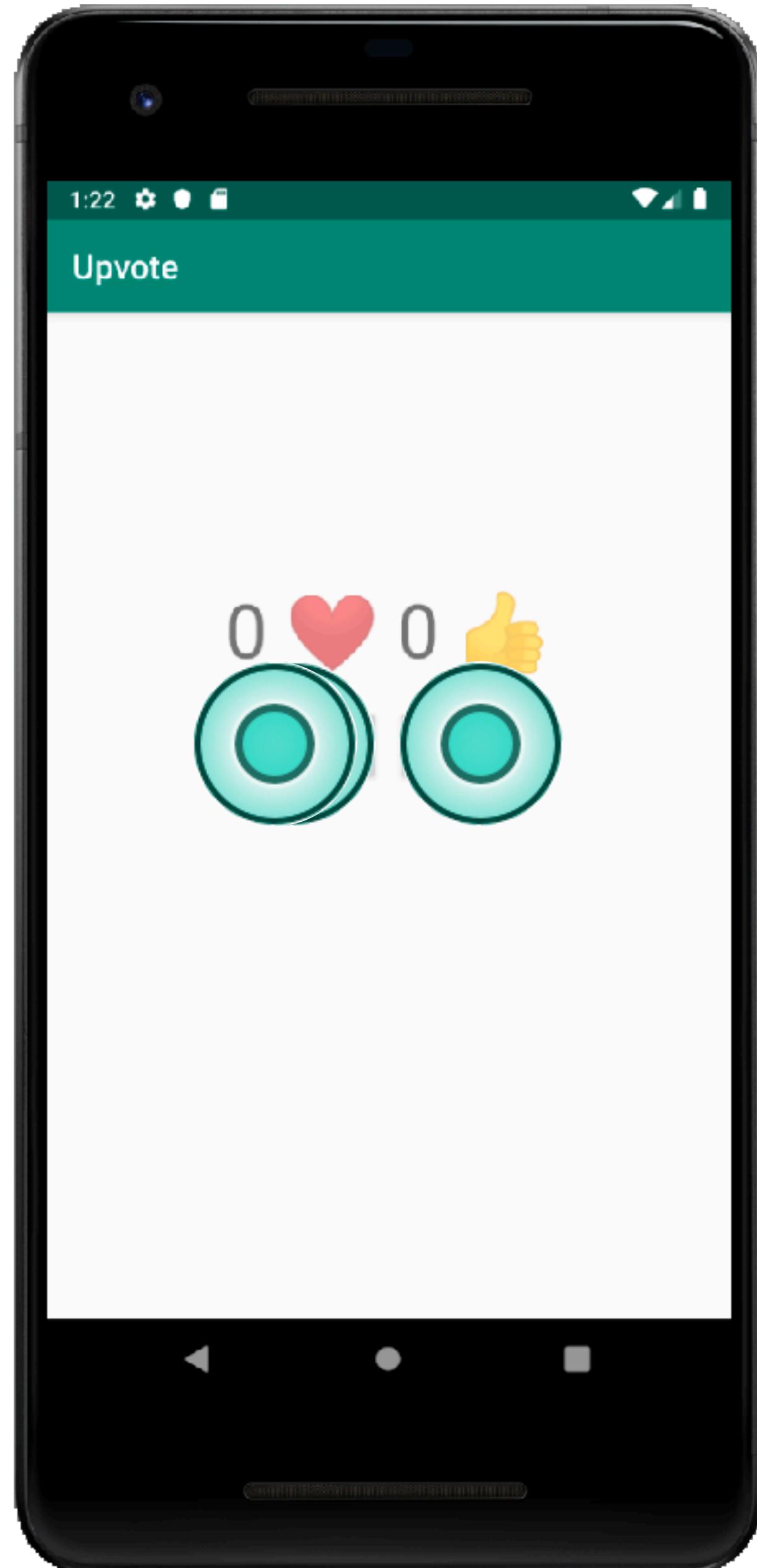






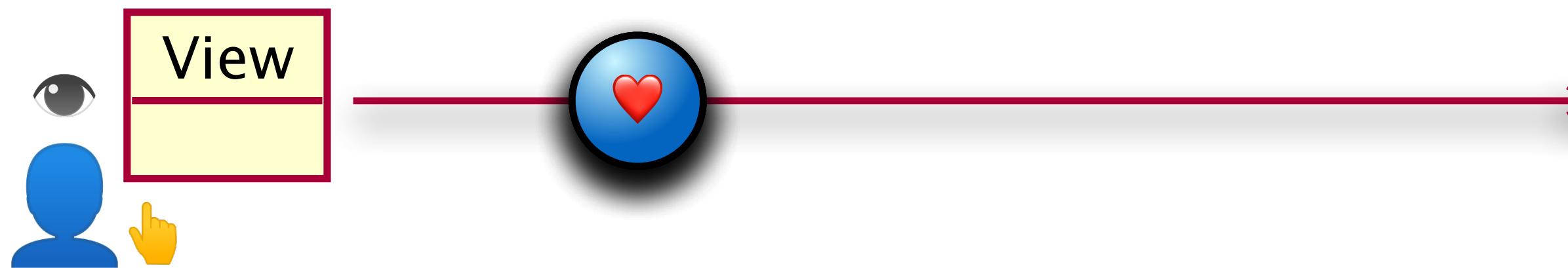
```
sealed class MainViewEvent {  
    object ThumbsUpClick : MainViewEvent()  
    object LoveItClick : MainViewEvent()  
}
```

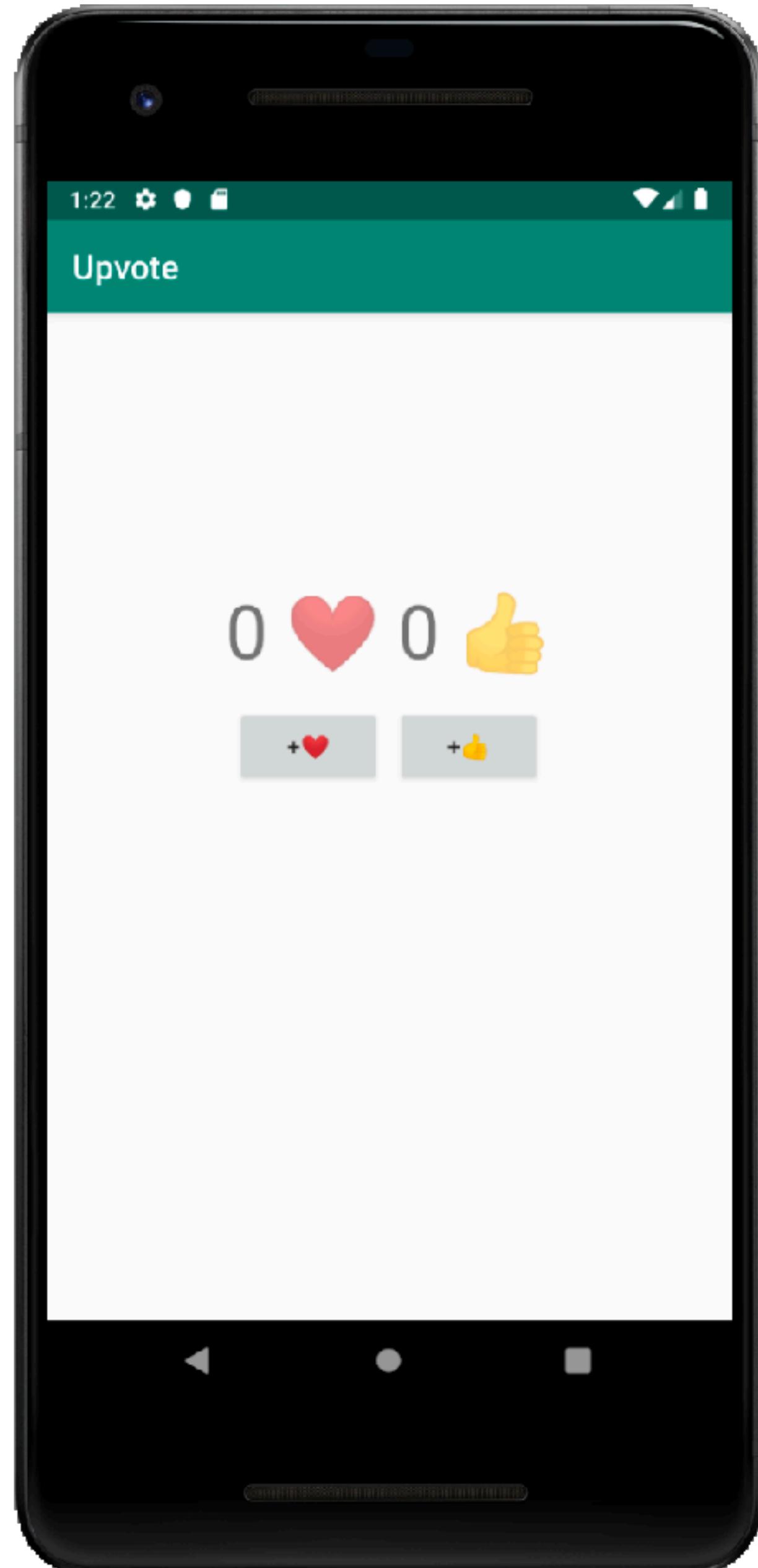




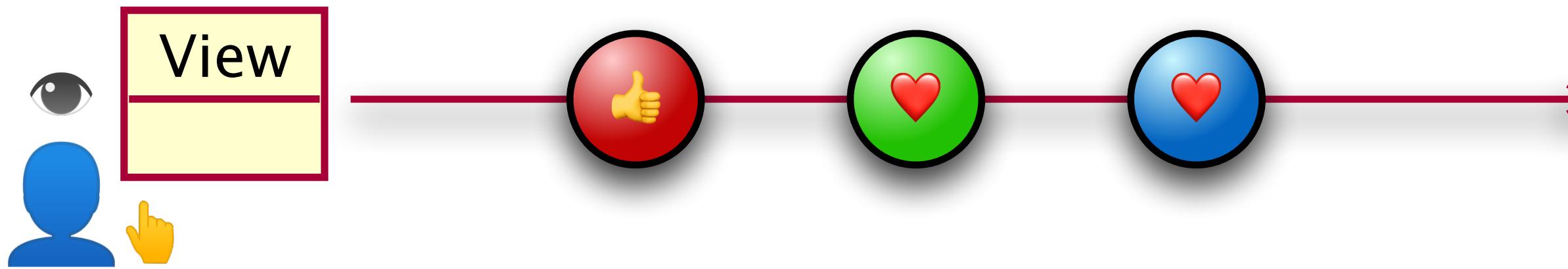
```
sealed class MainViewEvent {  
    object ThumbsUpClick : MainViewEvent()  
    object LoveItClick : MainViewEvent()  
}
```

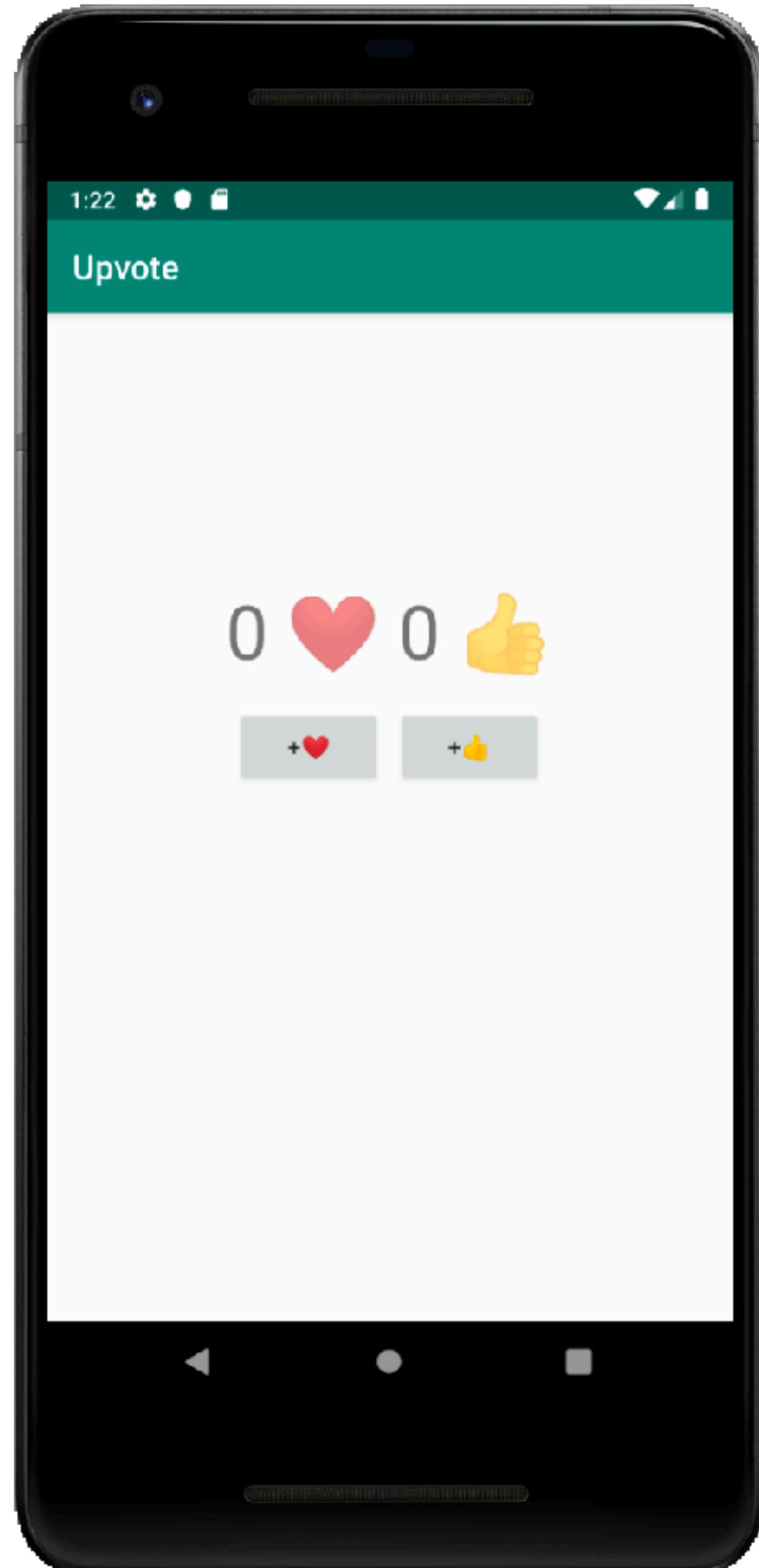
- MainViewEvent
- +👍 ThumbsUpClick
- +❤️ LoveItClick



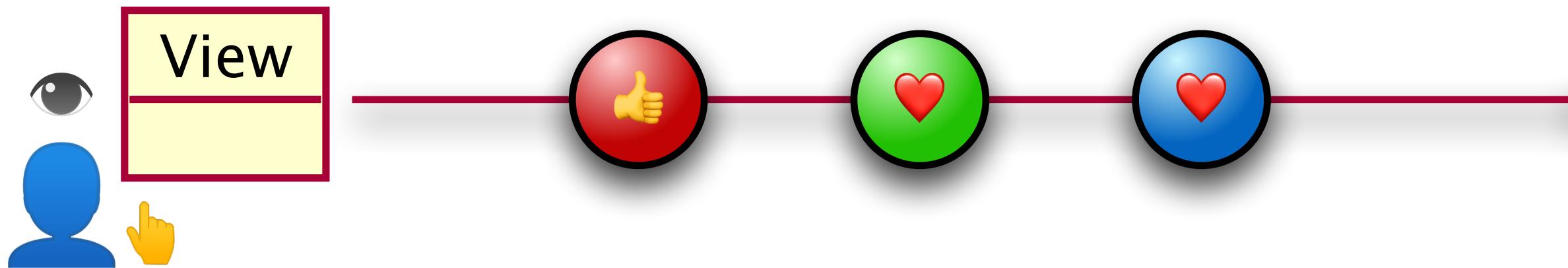


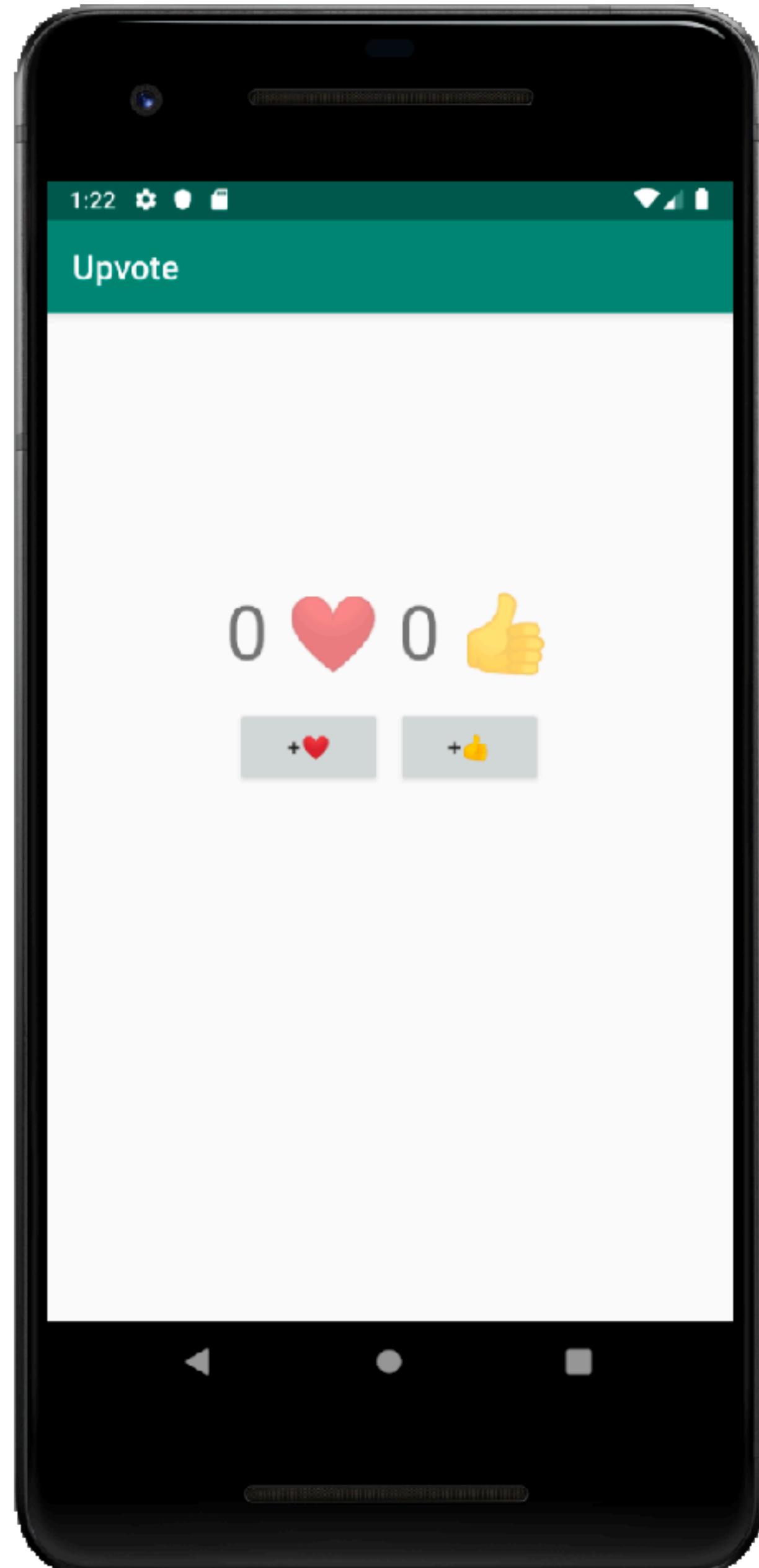
```
fun View.clicks(): Flow<Unit> = callbackFlow { this: ProducerScope<Unit>
    val listener = View.OnClickListener { offer(Unit) }
    setOnClickListener(listener)
    awaitClose {
        setOnClickListener(null)
    }
}
```



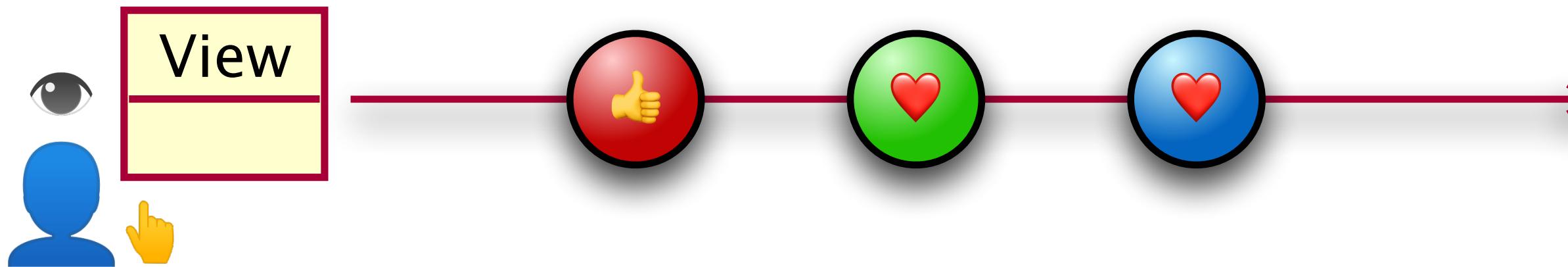


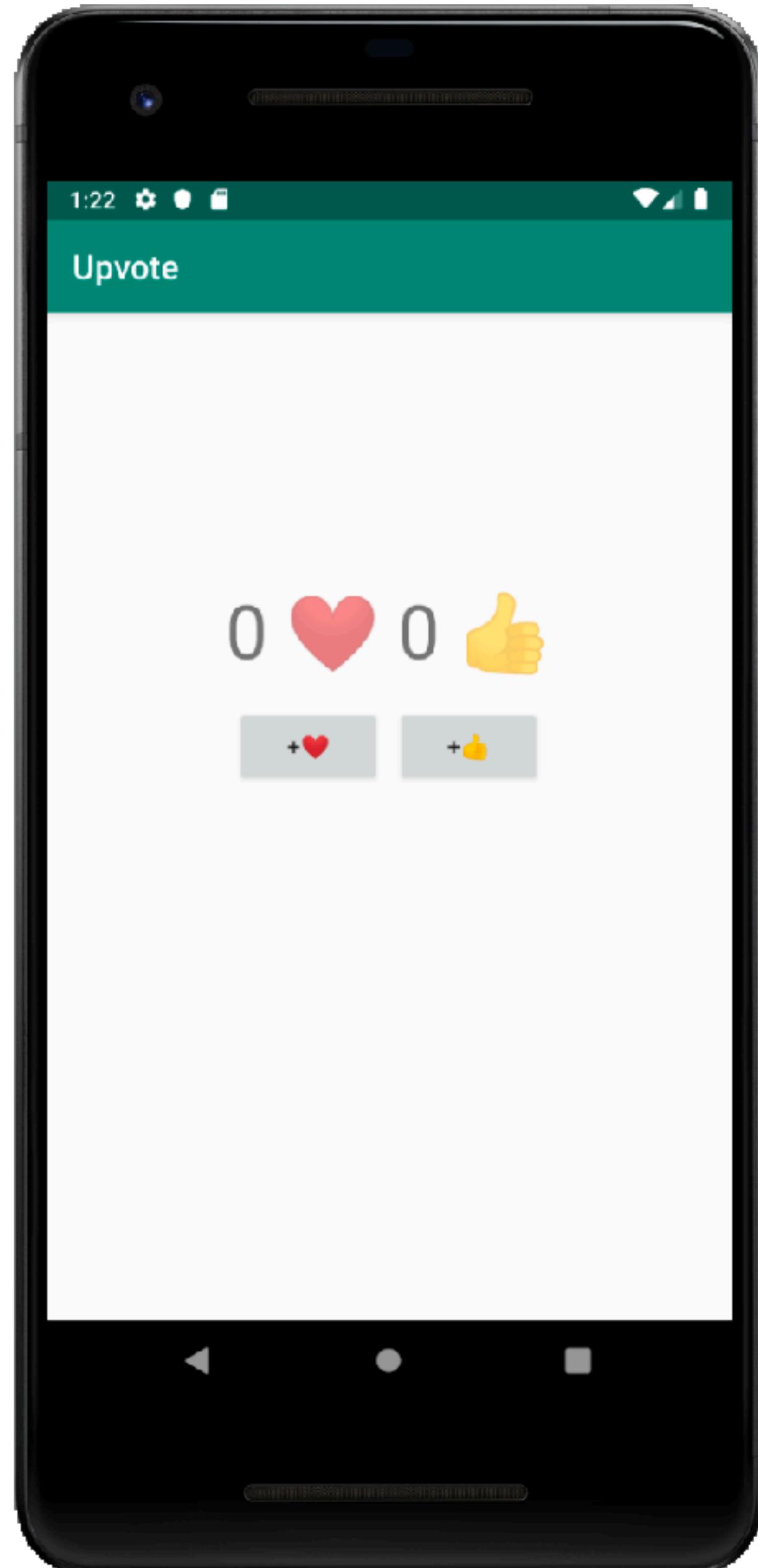
```
fun View.clicks(): Flow<Unit> = callbackFlow { this: ProducerScope<Unit>
    val listener = View.OnClickListener { offer(Unit) }
    setOnClickListener(listener)
    awaitClose {
        setOnClickListener(null)
    }
}
```



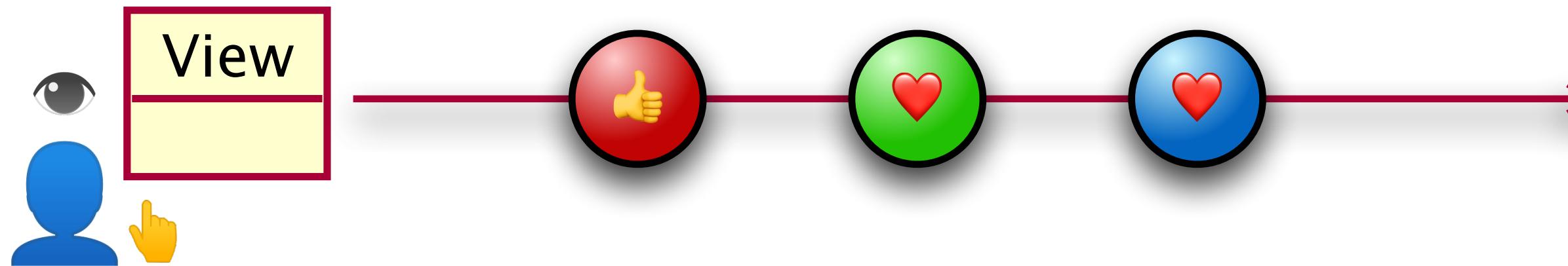


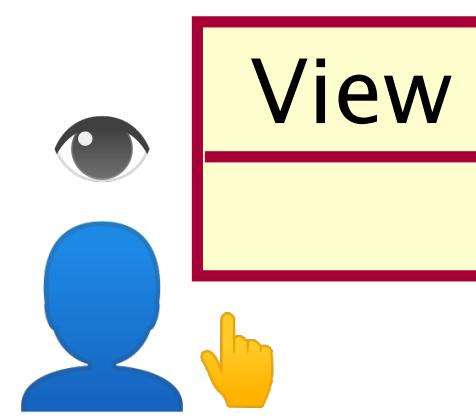
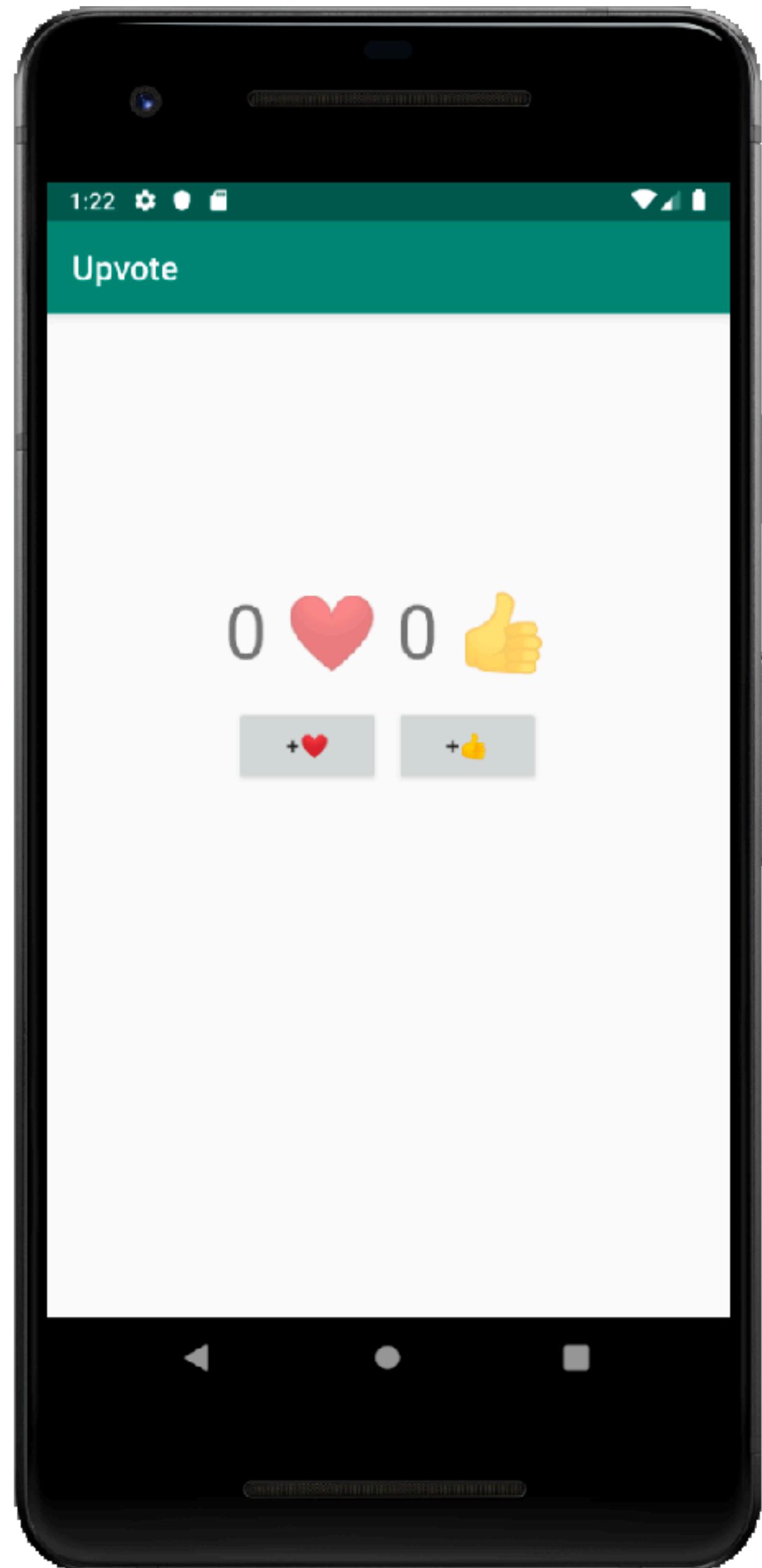
```
fun View.clicks(): Flow<Unit> = callbackFlow { this: ProducerScope<Unit>
    val listener = View.OnClickListener { offer(Unit) }
    setOnClickListener(listener)
    awaitClose {
        setOnClickListener(null)
    }
}
```



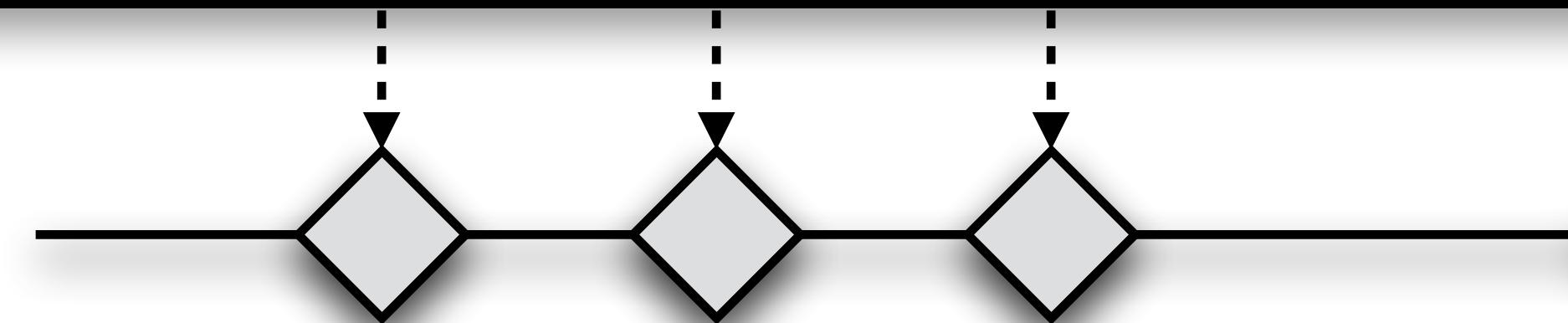


```
fun View.clicks(): Flow<Unit> = callbackFlow { this: ProducerScope<Unit>
    val listener = View.OnClickListener { offer(Unit) }
    setOnClickListener(listener)
    awaitClose {
        setOnClickListener(null)
    }
}
```

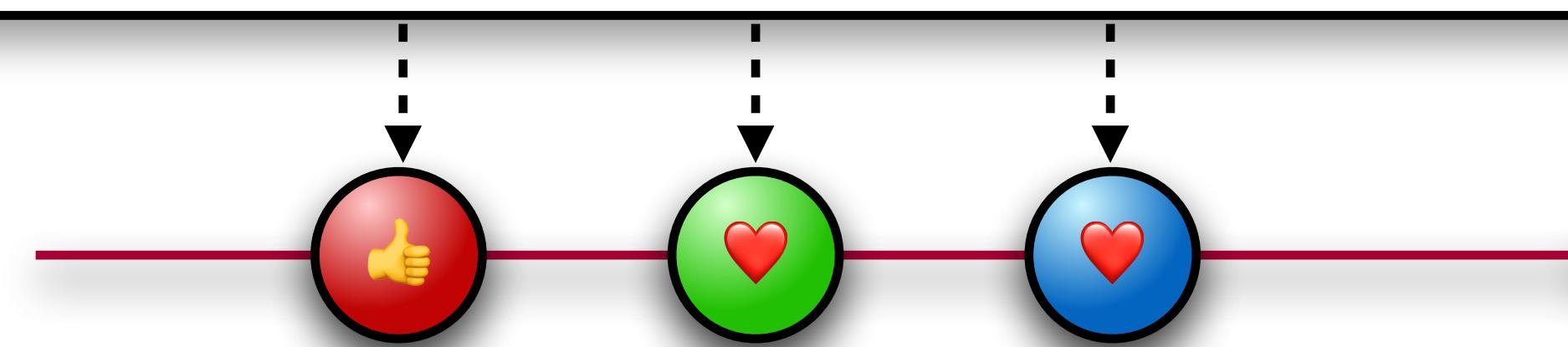


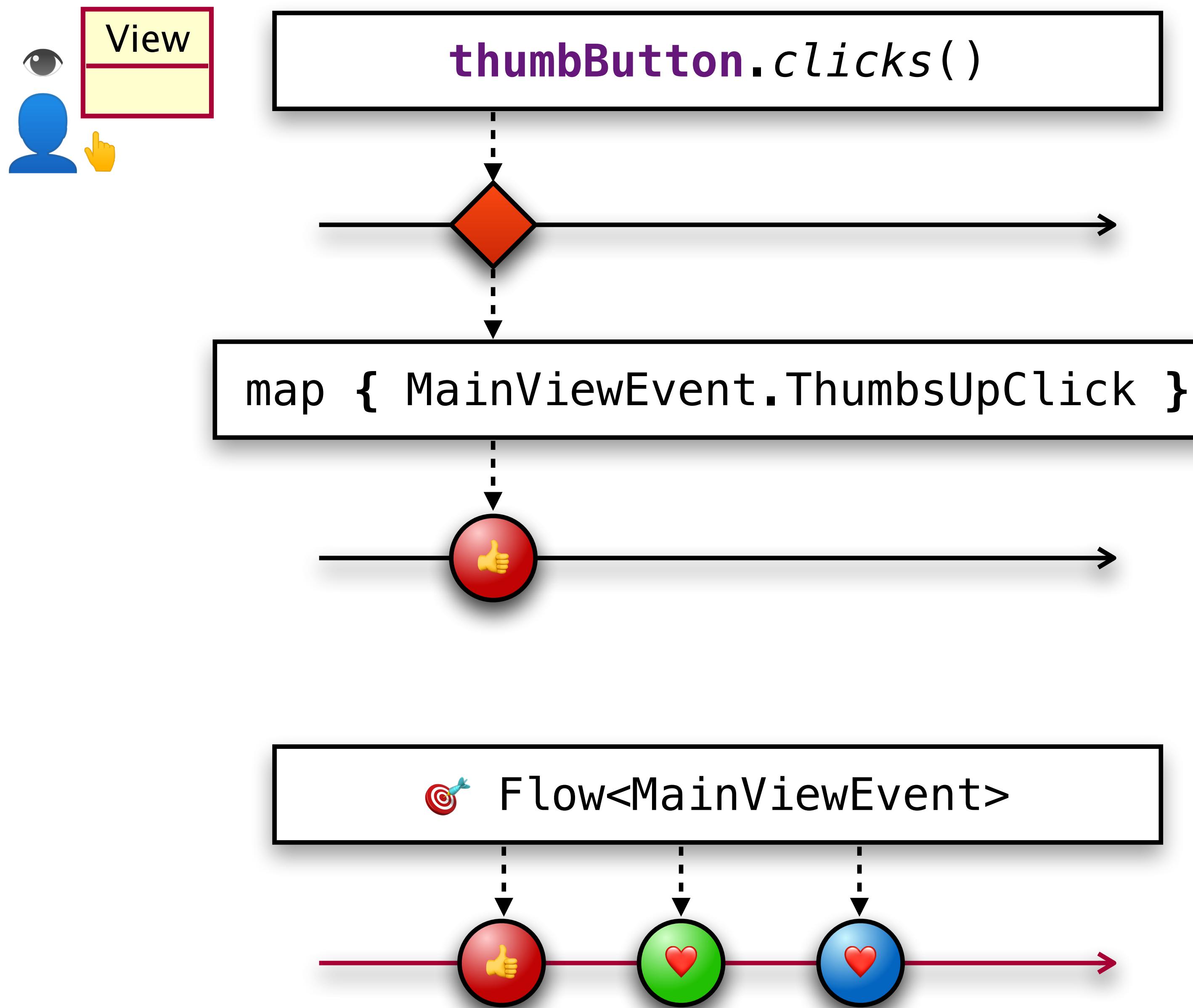
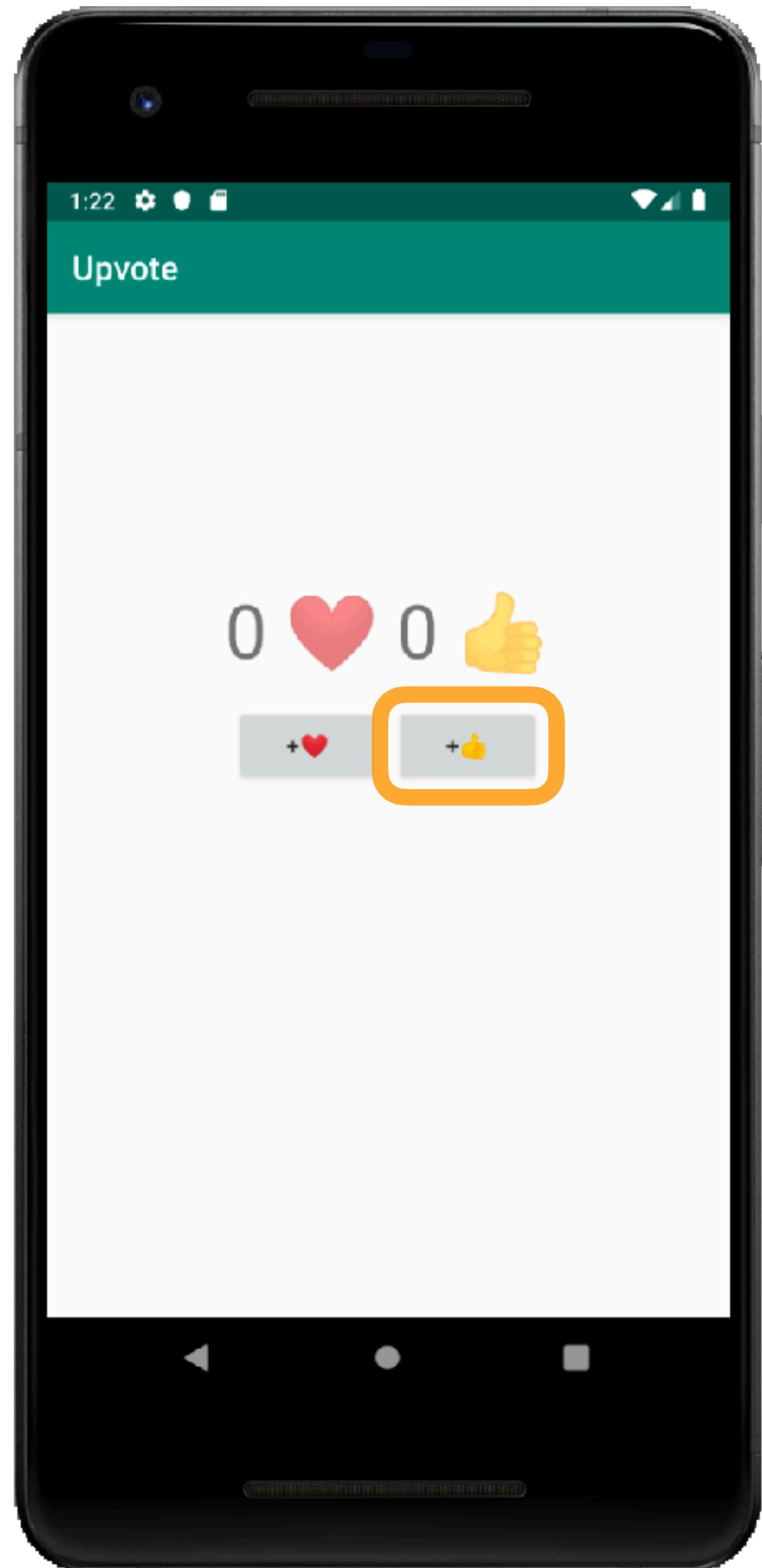


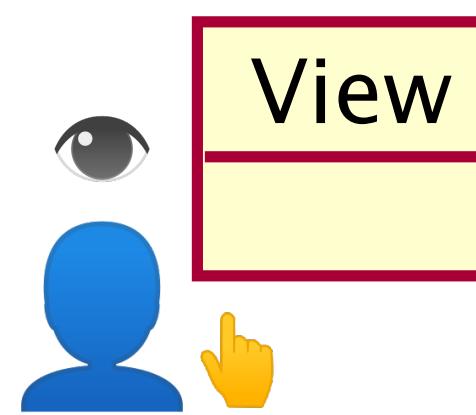
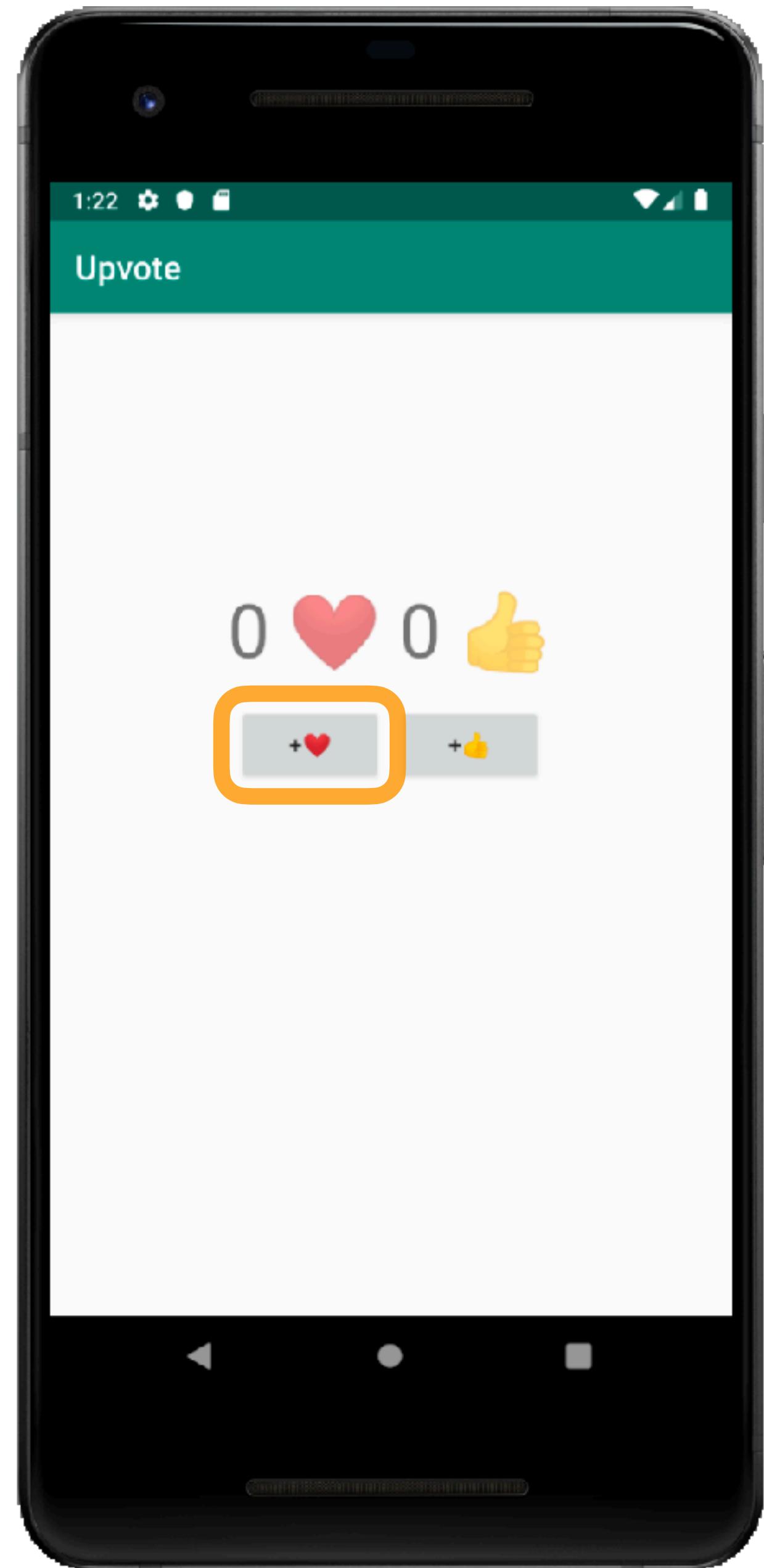
`View.clicks():Flow<Unit>`



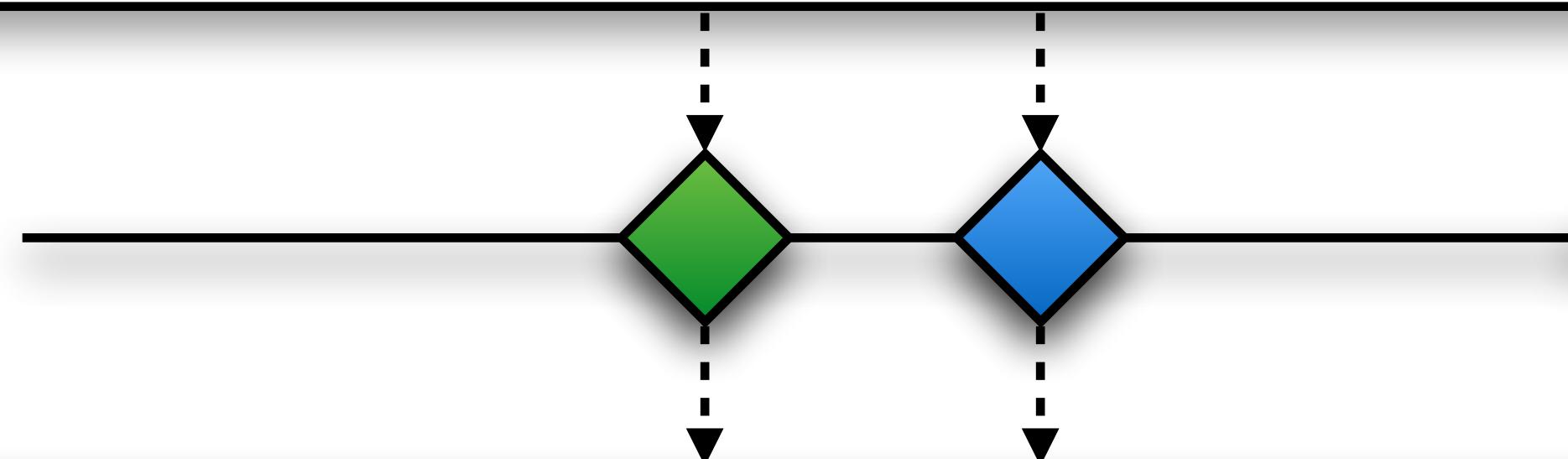
 `Flow<MainViewEvent>`



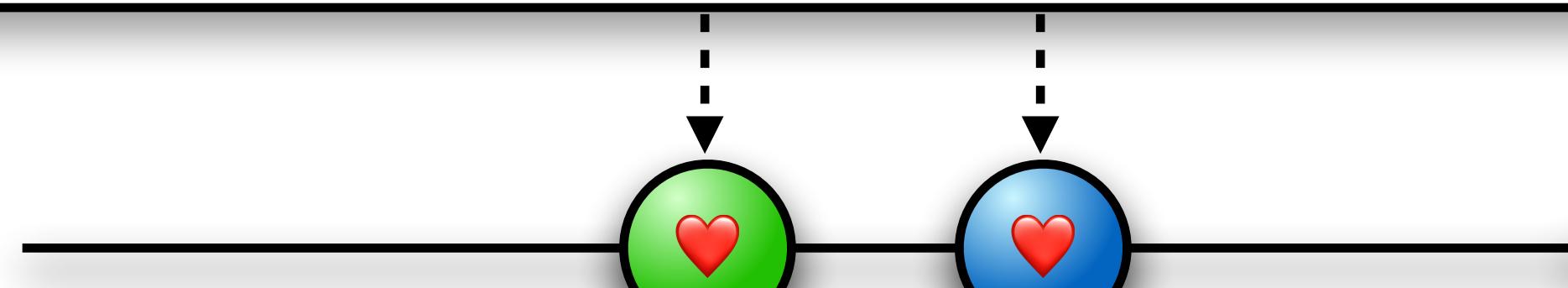




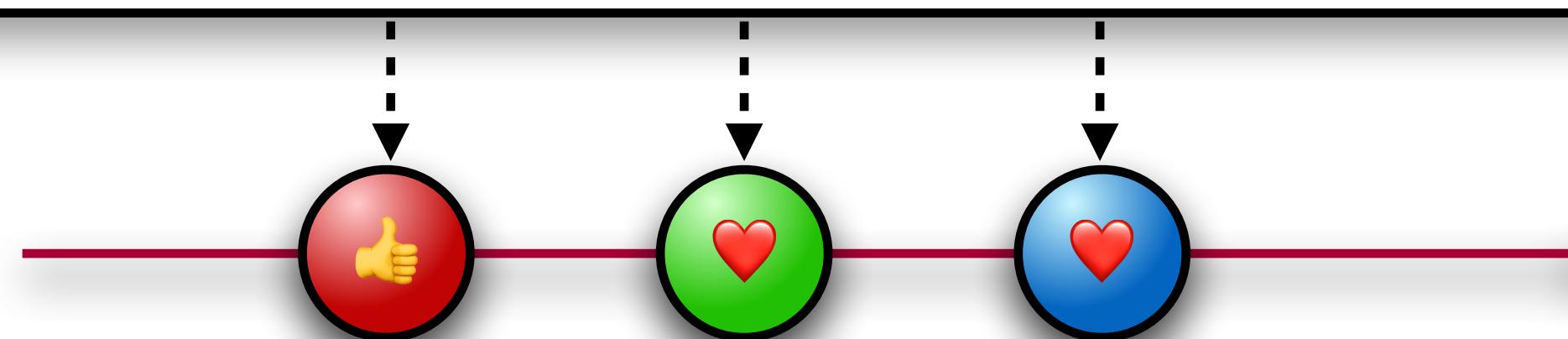
**heartButton.clicks()**

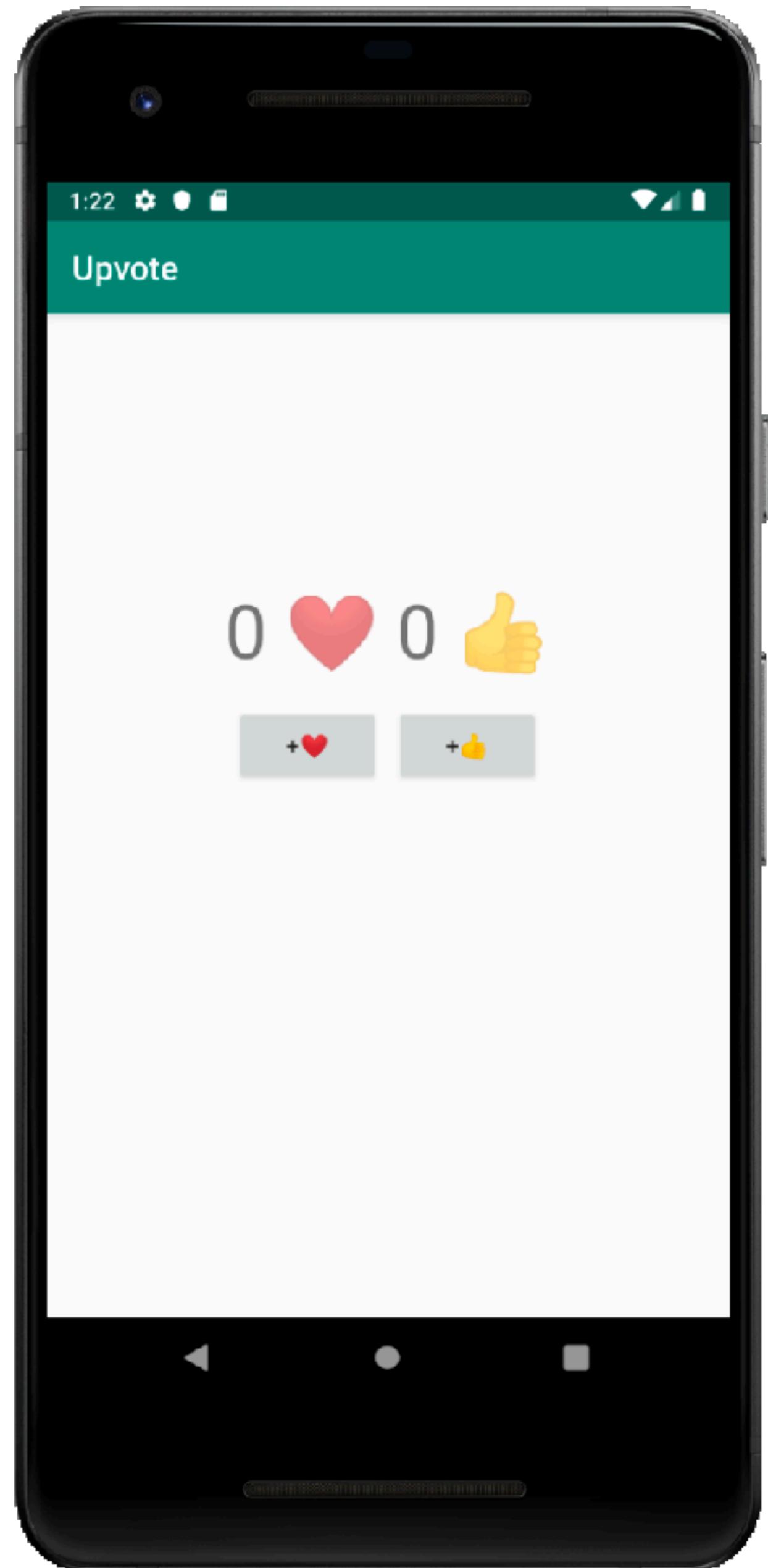


`map { MainViewEvent.LoveItClick }`



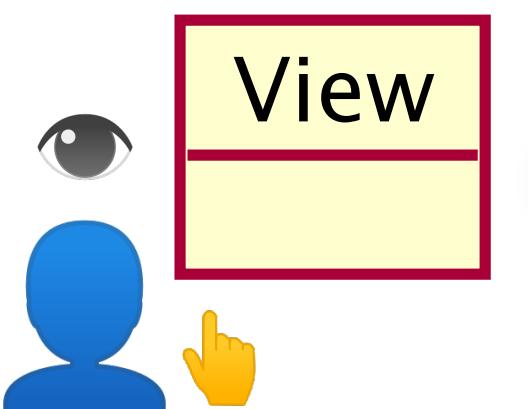
 **Flow<MainViewEvent>**





```
val flows = listOf(  
    heartButton.clicks().map { MainViewEvent.LoveItClick },  
    thumbButton.clicks().map { MainViewEvent.ThumbsUpClick }  
)  
  
return fl
```

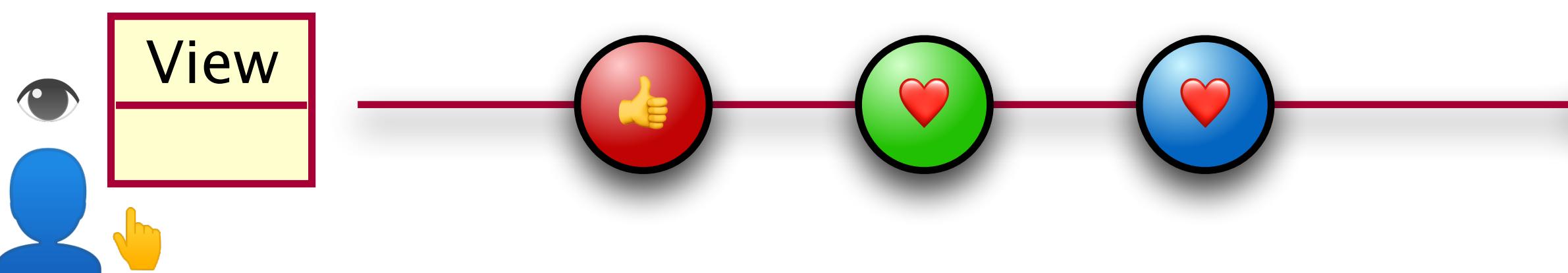
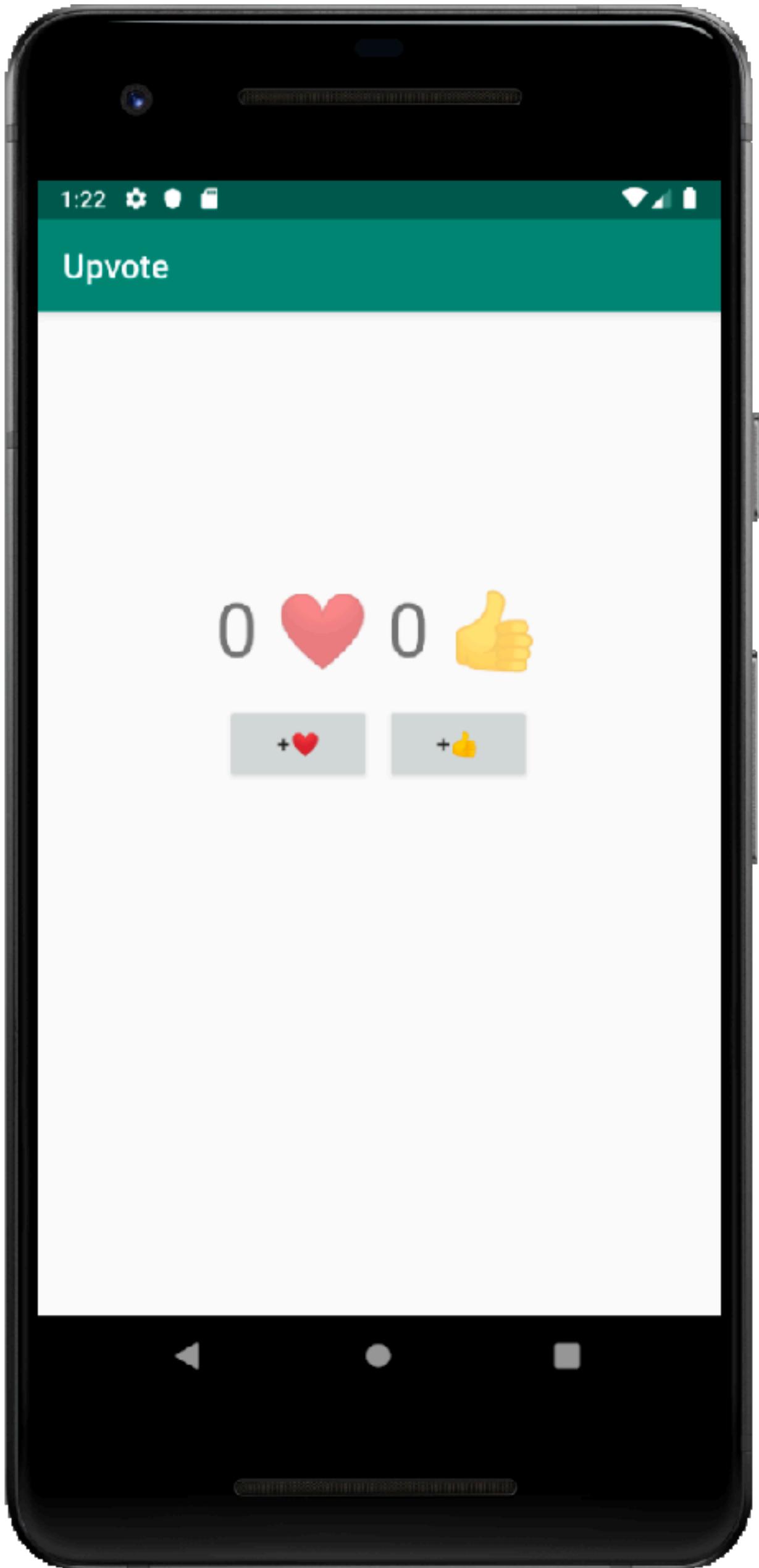
🎯 Flow<MainViewEvent>



```
class MainActivity {

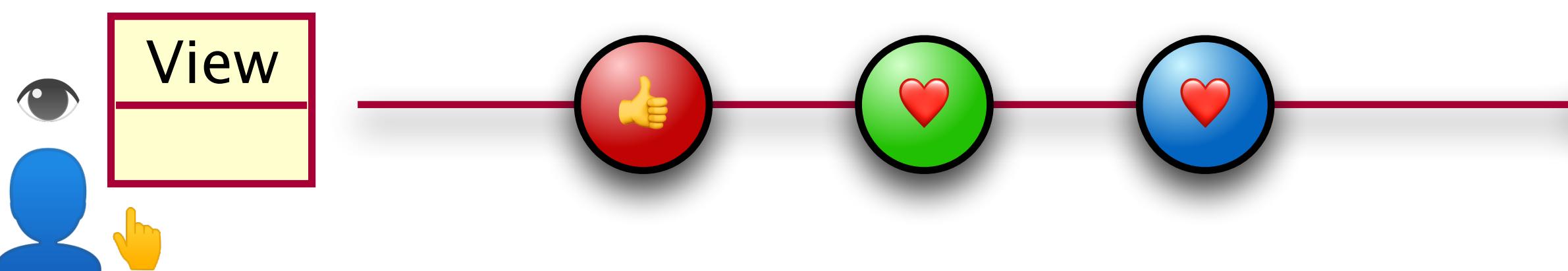
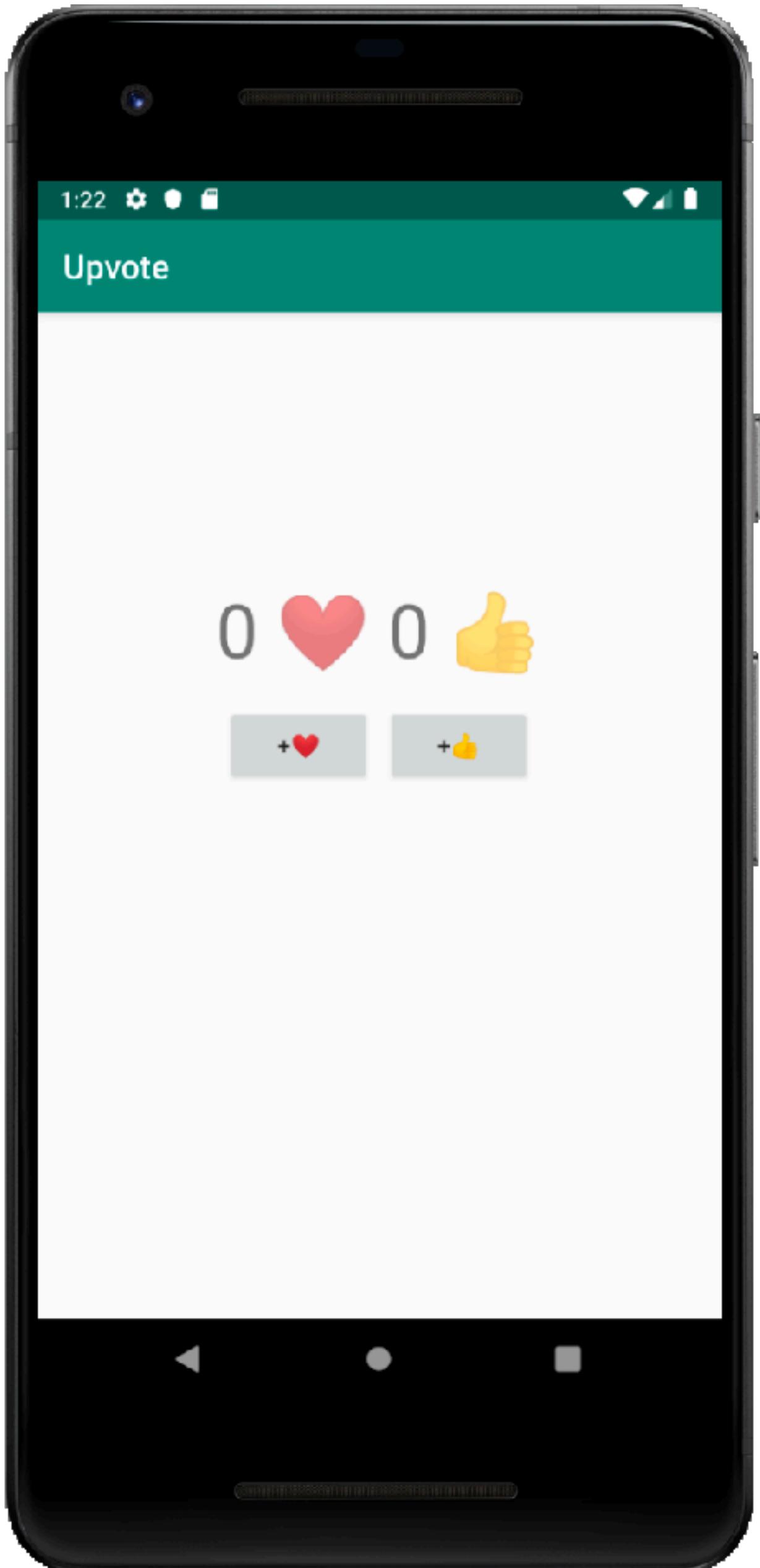
    override fun viewEvents(): Flow<MainViewEvent> {
        val flows = listOf(
            heartButton.clicks().map { MainViewEvent.LoveItClick },
            thumbButton.clicks().map { MainViewEvent.ThumbsUpClick }
        )

        return flows.asFlow().flattenMerge(flows.size)
    }
}
```



```
class MainActivity {

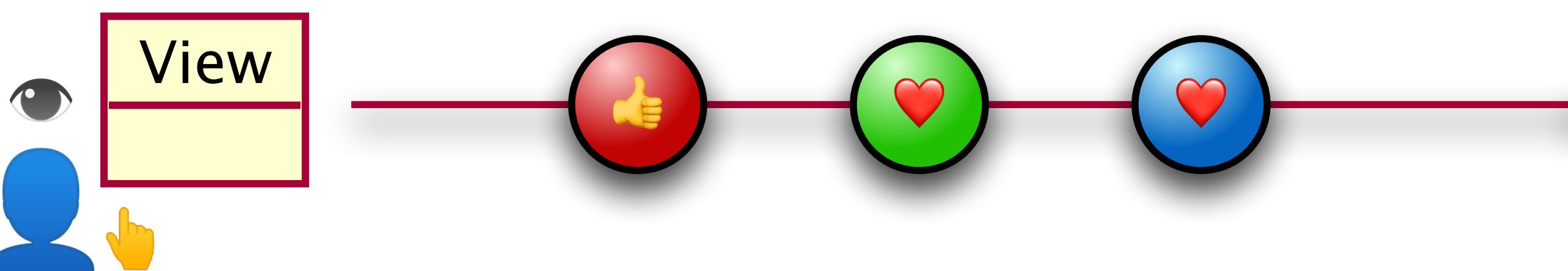
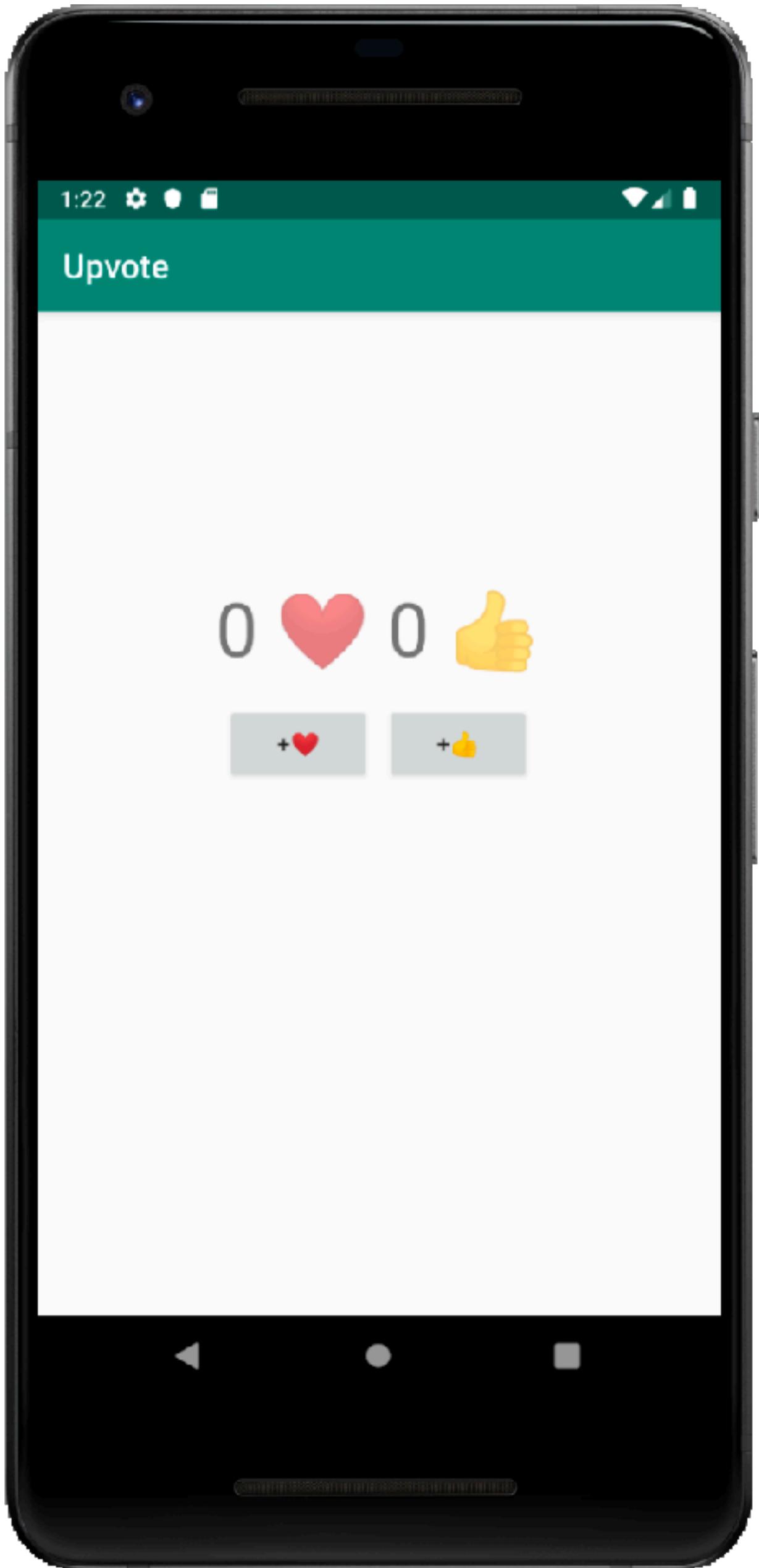
    override fun viewEvents(): Flow<MainViewEvent> {
        val flows = listOf(
            heartButton.clicks().map { MainViewEvent.LoveItClick },
            thumbButton.clicks().map { MainViewEvent.ThumbsUpClick }
        )
        return flows.asFlow().flattenMerge(flows.size)
    }
}
```



```
class MainActivity {

    override fun viewEvents(): Flow<MainViewEvent> {
        val flows = listOf(
            heartButton.clicks().map { MainViewEvent.LoveItClick },
            thumbButton.clicks().map { MainViewEvent.ThumbsUpClick }
        )

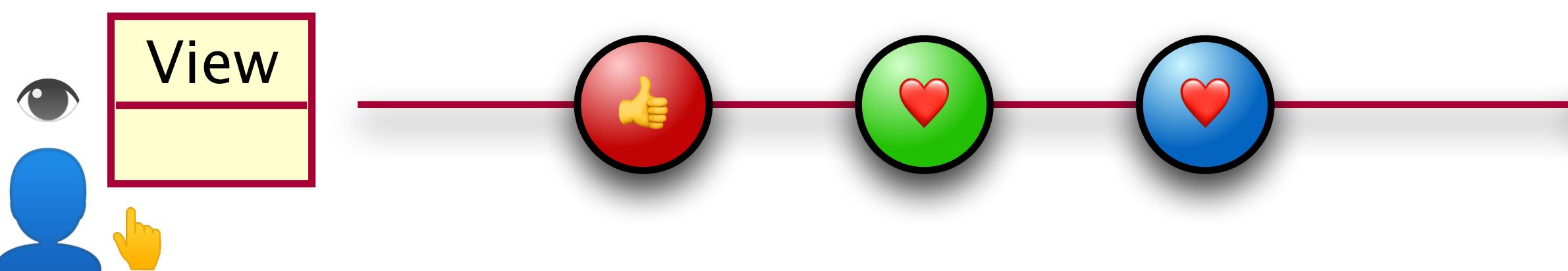
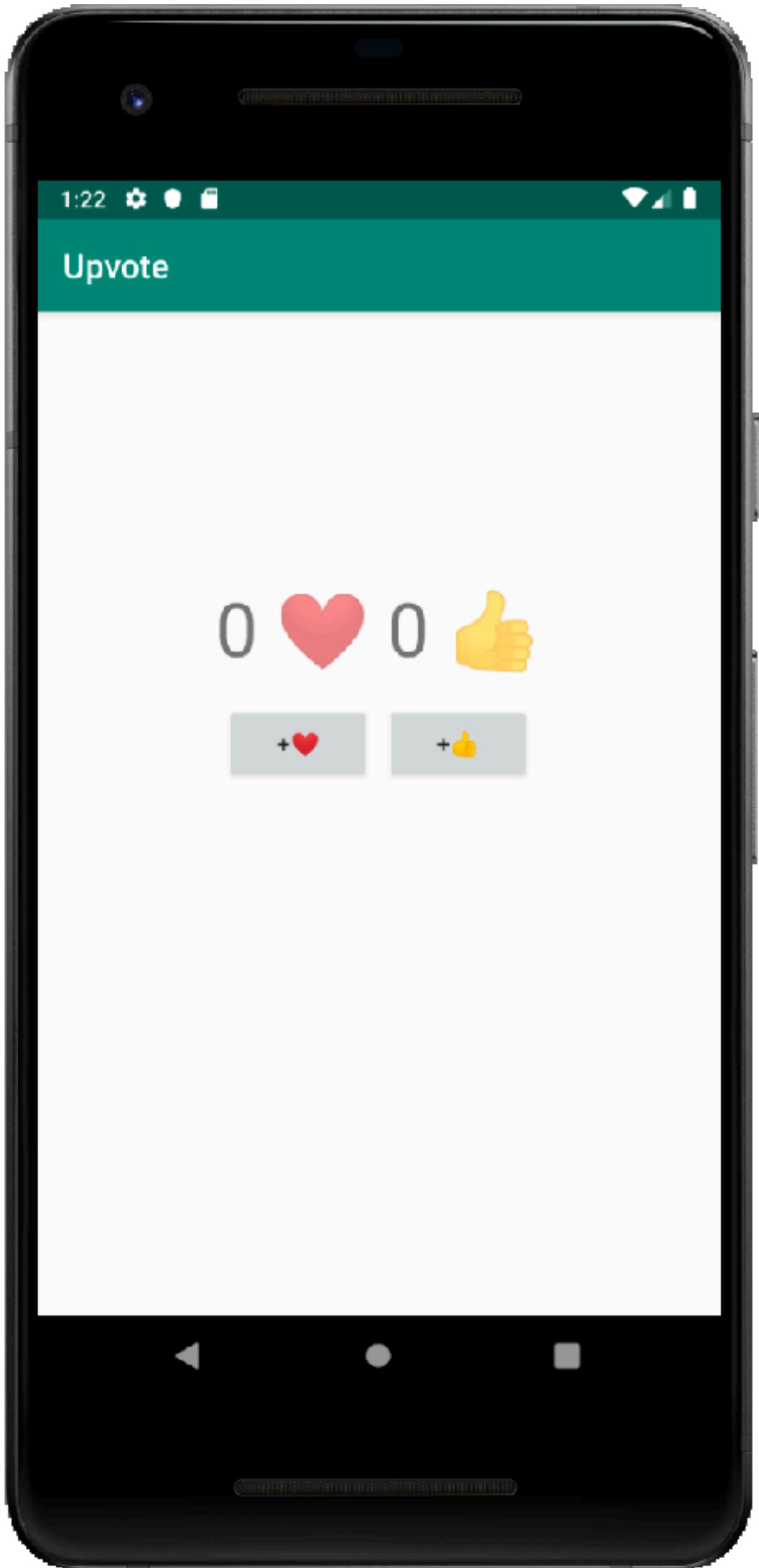
        return flows.asFlow().flattenMerge(flows.size)
    }
}
```

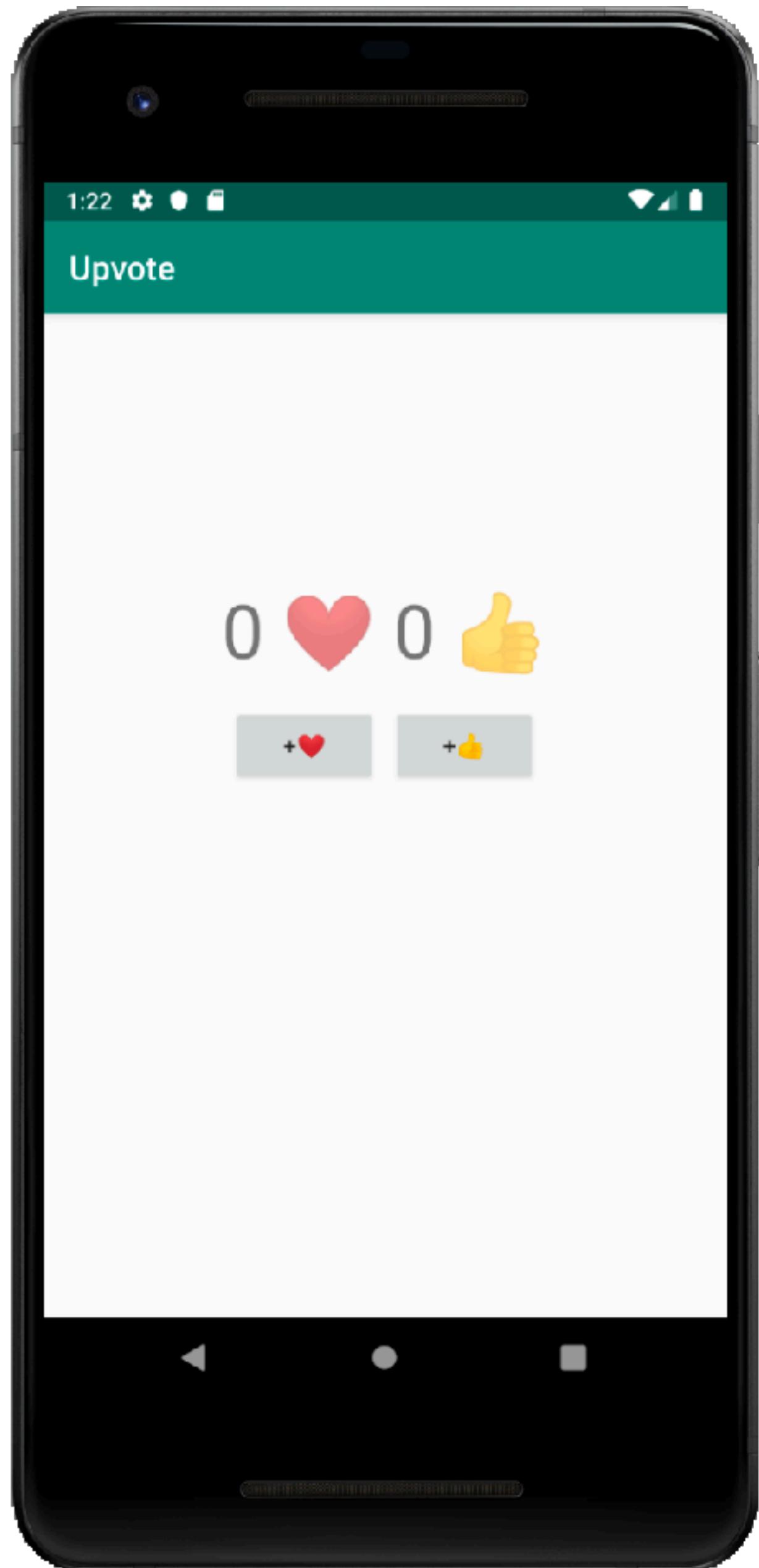


```
class MainActivity {

    override fun viewEvents(): Flow<MainViewEvent> {
        val flows = listOf(
            heartButton.clicks().map { MainViewEvent.LoveItClick },
            thumbButton.clicks().map { MainViewEvent.ThumbsUpClick }
        )

        return flows.asFlow().flattenMerge(flows.size)
    }
}
```



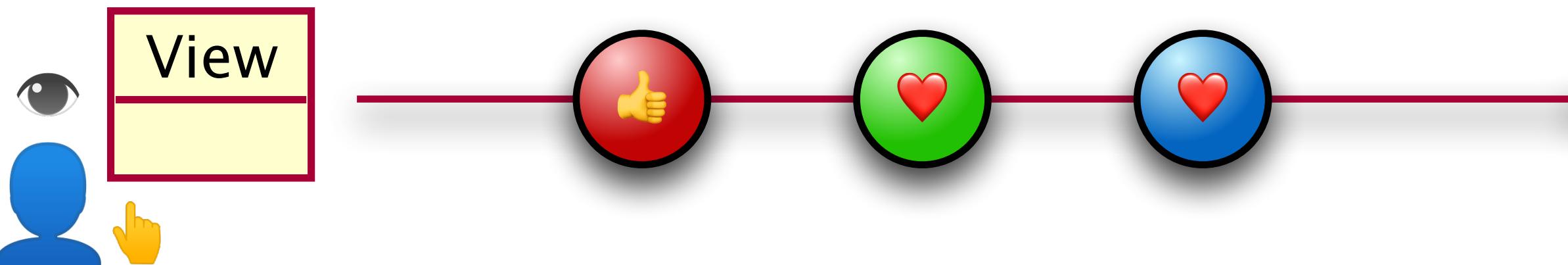


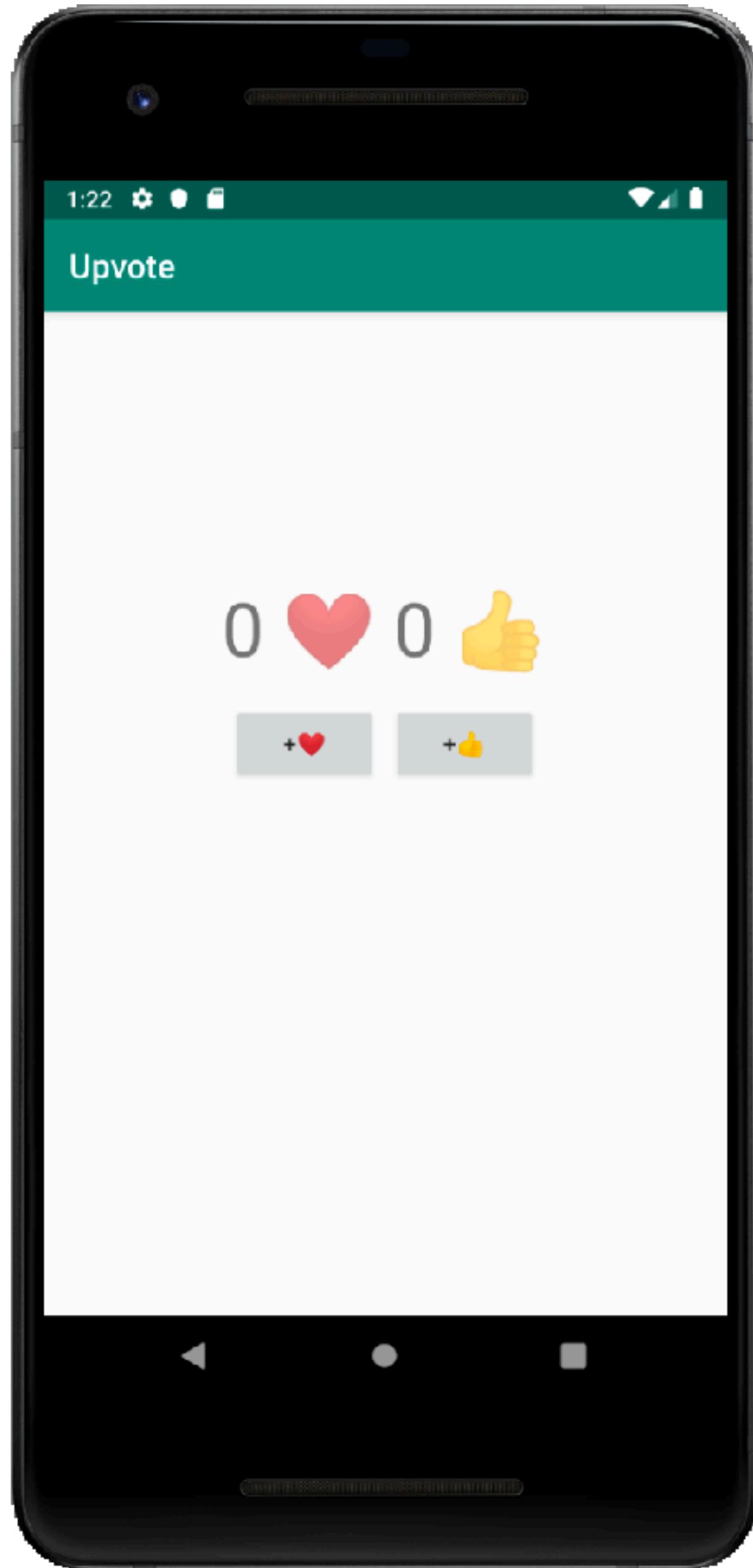
```
/*
 * This allows us to group all the viewEvents from
 * one view in a single Flow.
 */
interface ViewEventFlow<E> {
    fun viewEvents(): Flow<E>
}

class MainActivity : ViewEventFlow<MainViewEvent> {

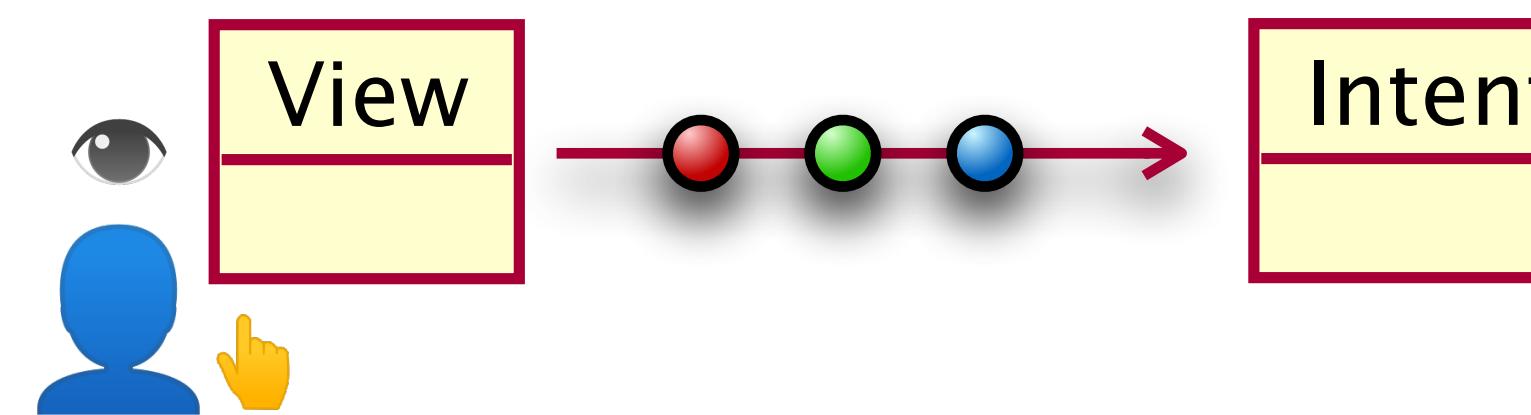
    override fun viewEvents(): Flow<MainViewEvent> {
        val flows = listOf(
            heartButton.clicks().map { MainViewEvent.LoveItClick },
            thumbButton.clicks().map { MainViewEvent.ThumbsUpClick }
        )

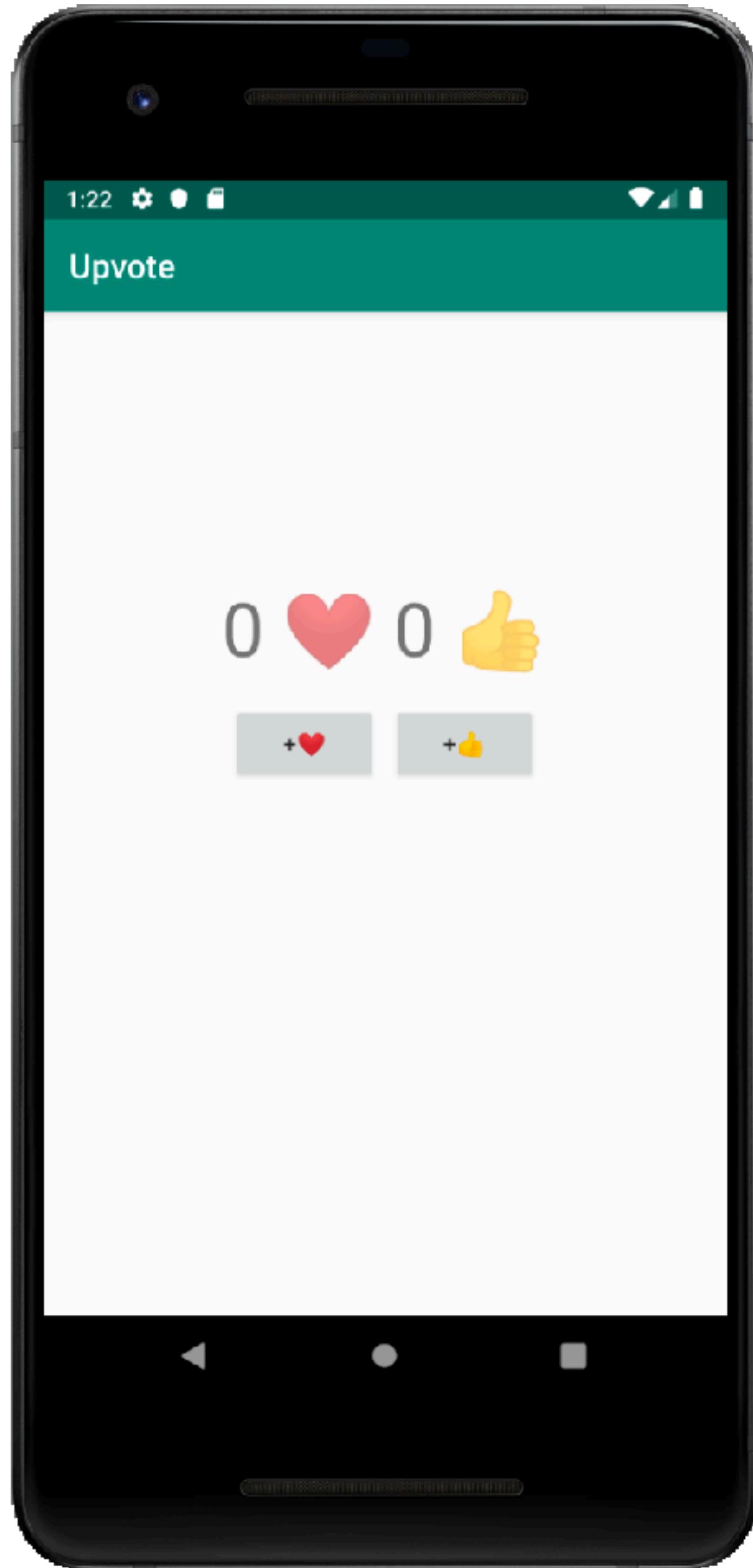
        return flows.asFlow().flattenMerge(flows.size)
    }
}
```



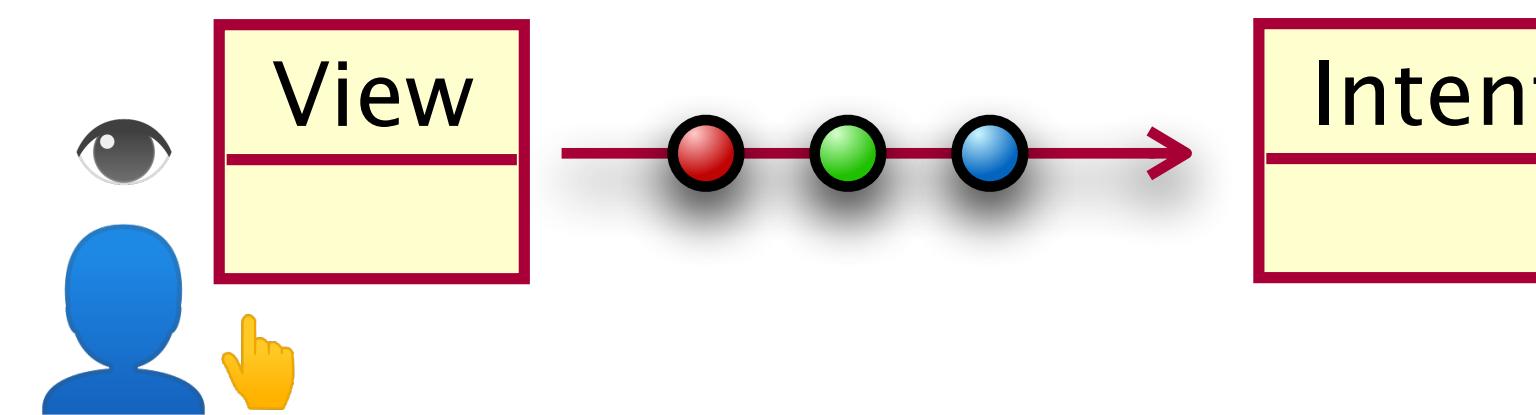


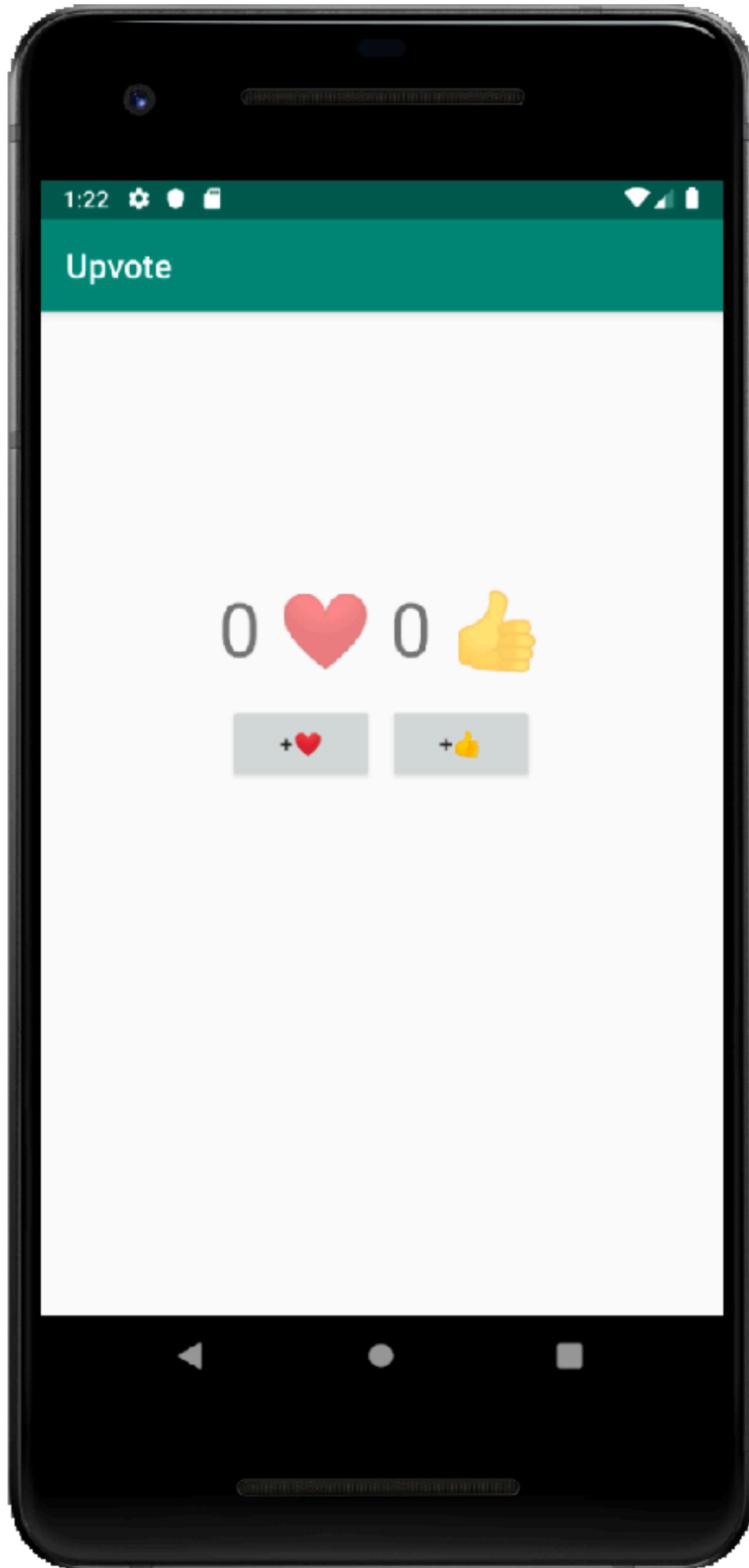
```
class MainActivity : ViewEventFlow<MainViewEvent> {  
  
    val scope: CoroutineScope = MainScope()  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        // ...  
    }  
  
    override fun onDestroy() {  
        super.onDestroy()  
        scope.cancel()  
    }  
  
    override fun viewEvents(): Flow<MainViewEvent> {  
        // ...  
    }  
}
```



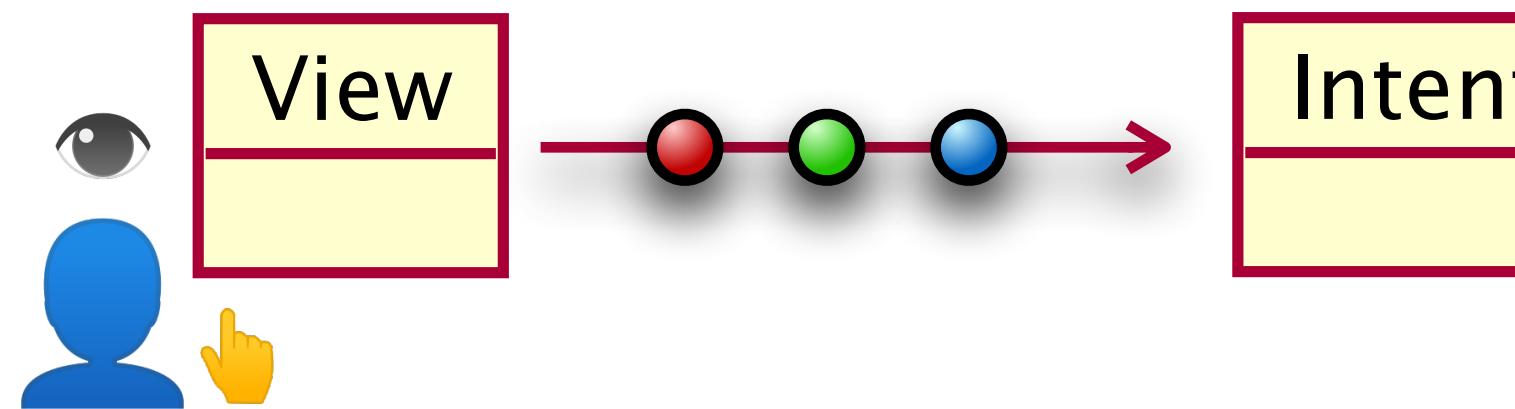


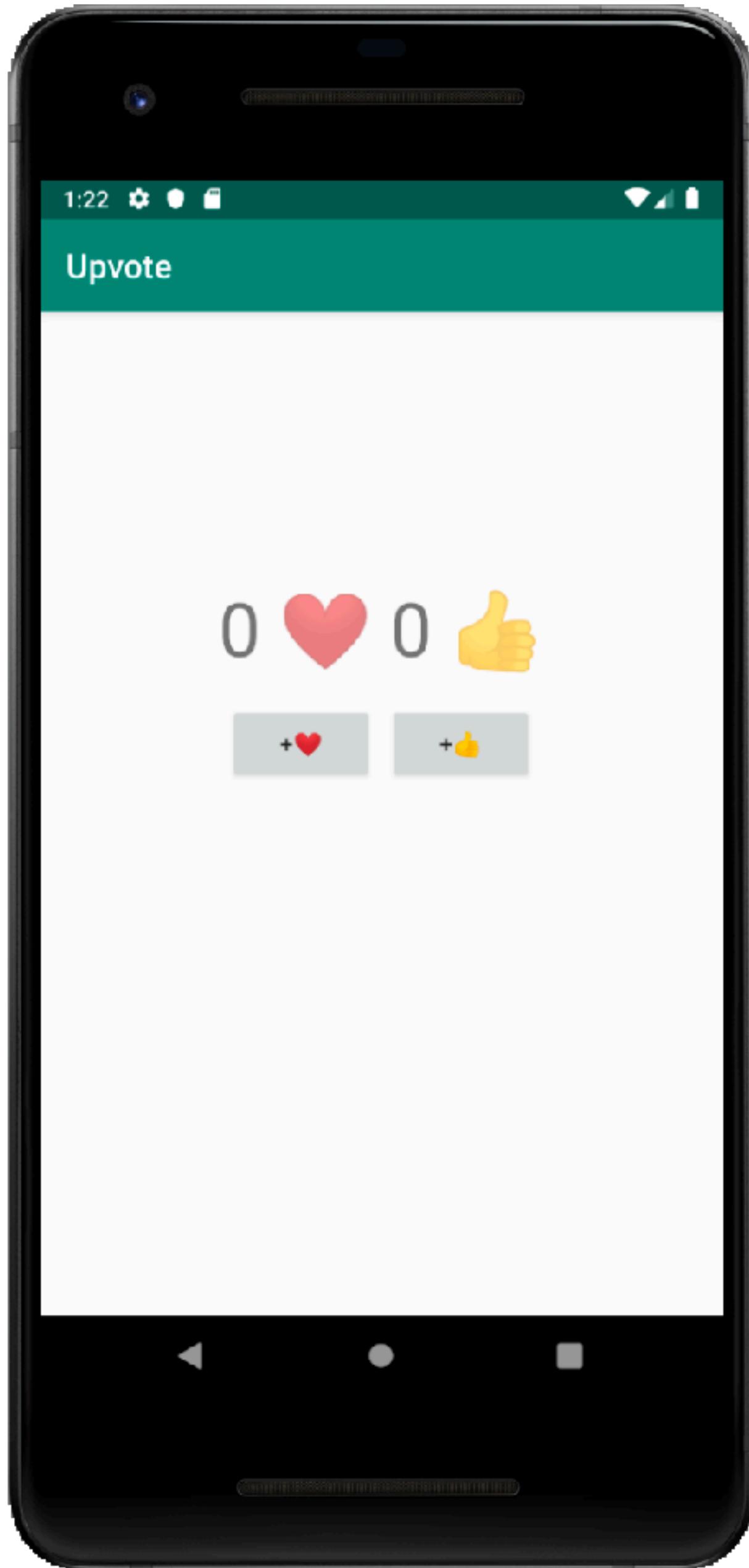
```
class MainActivity : ViewEventFlow<MainViewEvent> {  
  
    val scope: CoroutineScope = MainScope()  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        // ...  
    }  
  
    override fun onDestroy() {  
        super.onDestroy()  
        scope.cancel()  
    }  
  
    override fun viewEvents(): Flow<MainViewEvent> {  
        // ...  
    }  
}
```



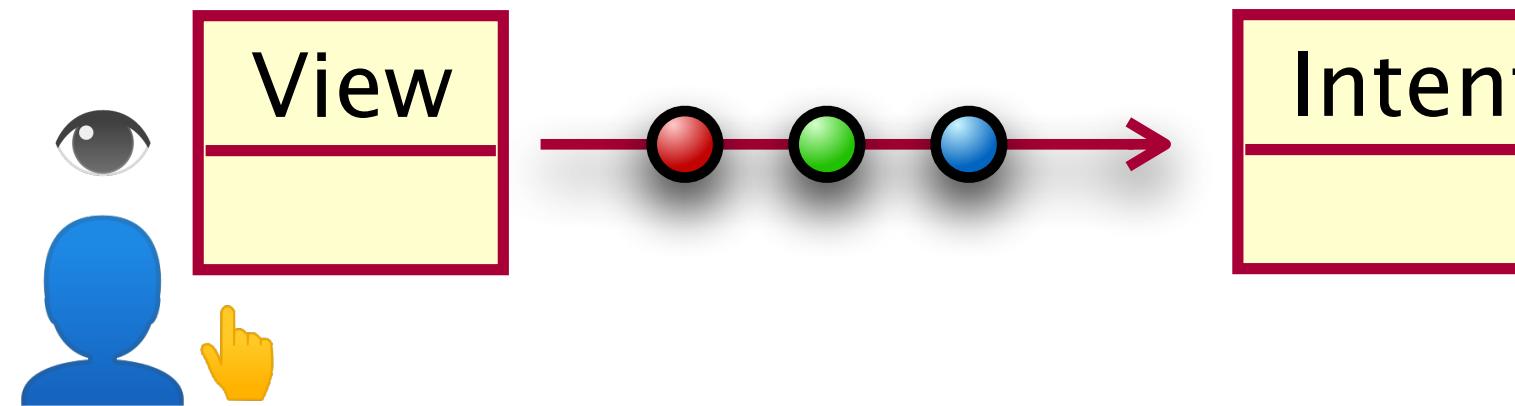


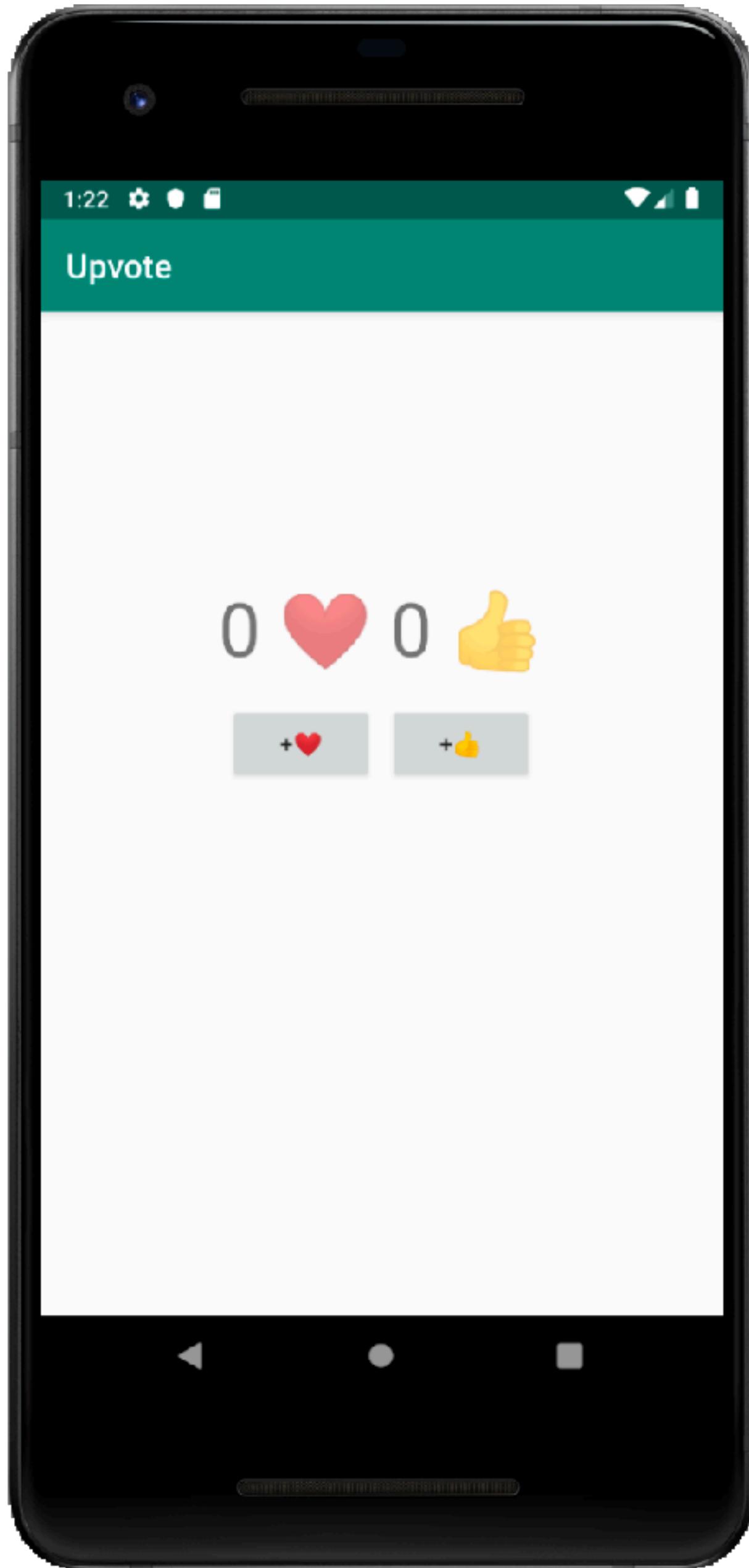
```
class MainActivity : ViewEventFlow<MainViewEvent> {  
  
    val scope: CoroutineScope = MainScope()  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        // ...  
        scope.launch_ { this: CoroutineScope  
            viewEvents()  
                .onEach { ?_ }  
                .collect()  
        }  
    }  
  
    override fun onDestroy() {  
        super.onDestroy()  
        scope.cancel()  
    }  
  
    override fun viewEvents(): Flow<MainViewEvent> {  
        // ...  
    }  
}
```



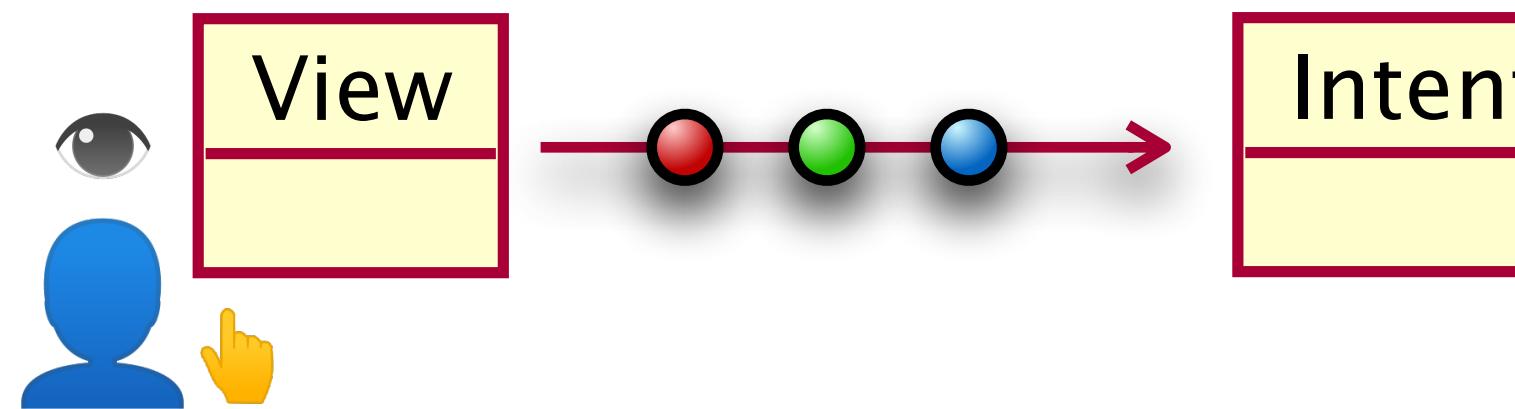


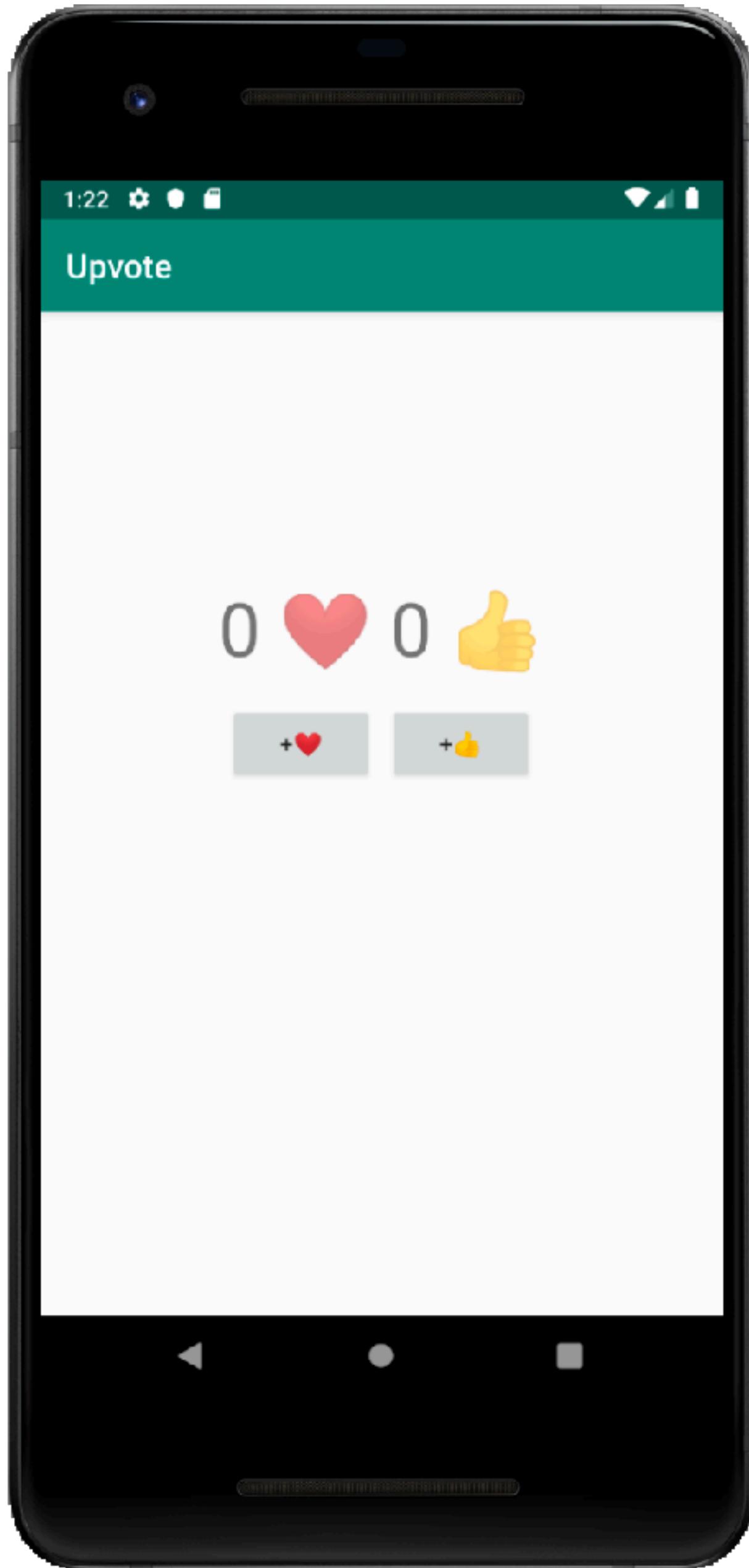
```
class MainActivity : ViewEventFlow<MainViewEvent> {  
  
    val scope: CoroutineScope = MainScope()  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        // ...  
        scope.launch { this: CoroutineScope  
            viewEvents()  
                .onEach { ? }  
                .collect()  
        }  
    }  
  
    override fun onDestroy() {  
        super.onDestroy()  
        scope.cancel()  
    }  
  
    override fun viewEvents(): Flow<MainViewEvent> {  
        // ...  
    }  
}
```



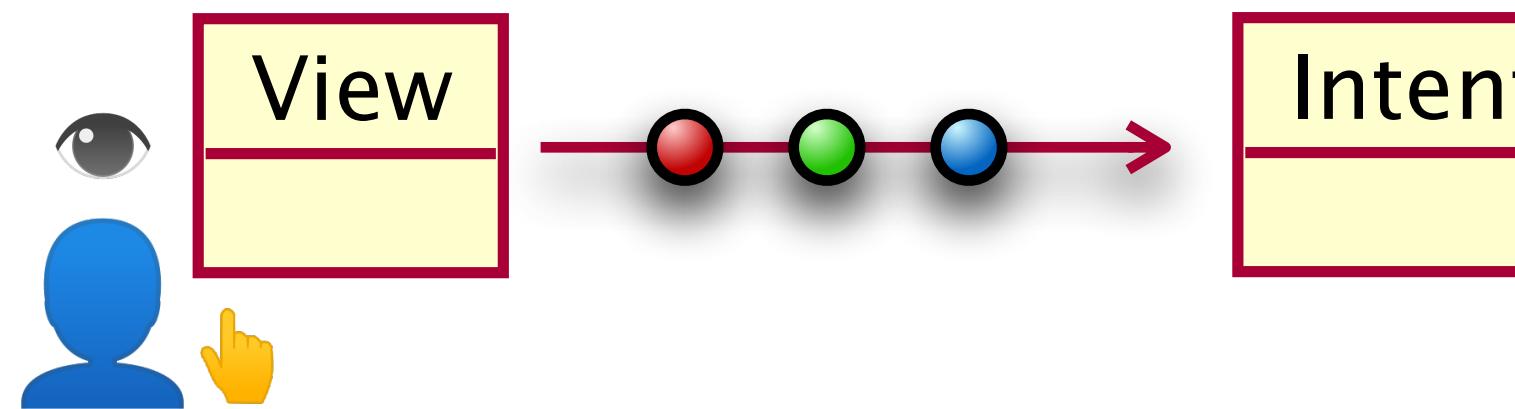


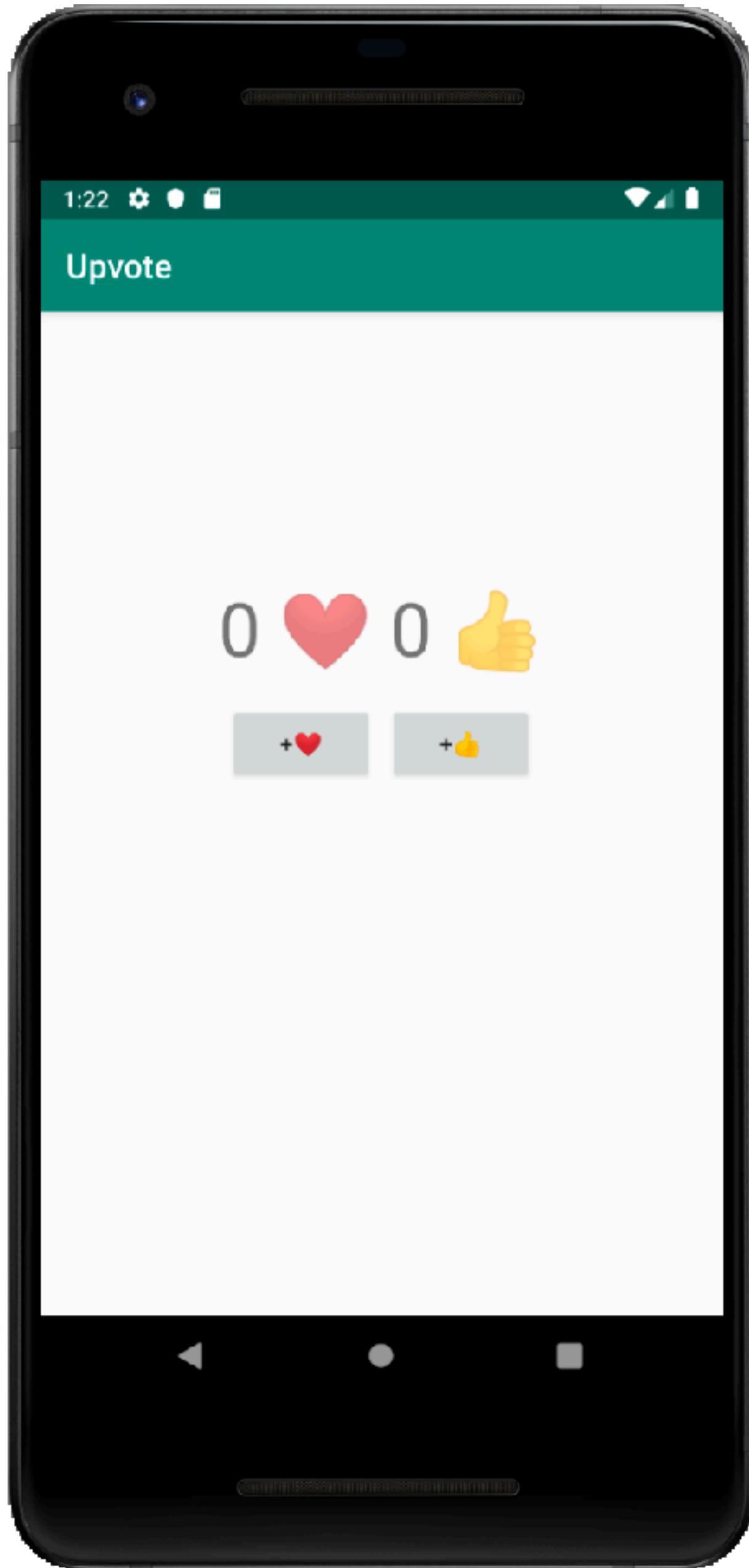
```
class MainActivity : ViewEventFlow<MainViewEvent> {  
  
    val scope: CoroutineScope = MainScope()  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        // ...  
        scope.launch { this: CoroutineScope  
            viewEvents()  
                .onEach { ?_ }  
                .collect()  
        }  
    }  
  
    override fun onDestroy() {  
        super.onDestroy()  
        scope.cancel()  
    }  
  
    override fun viewEvents(): Flow<MainViewEvent> {  
        // ...  
    }  
}
```



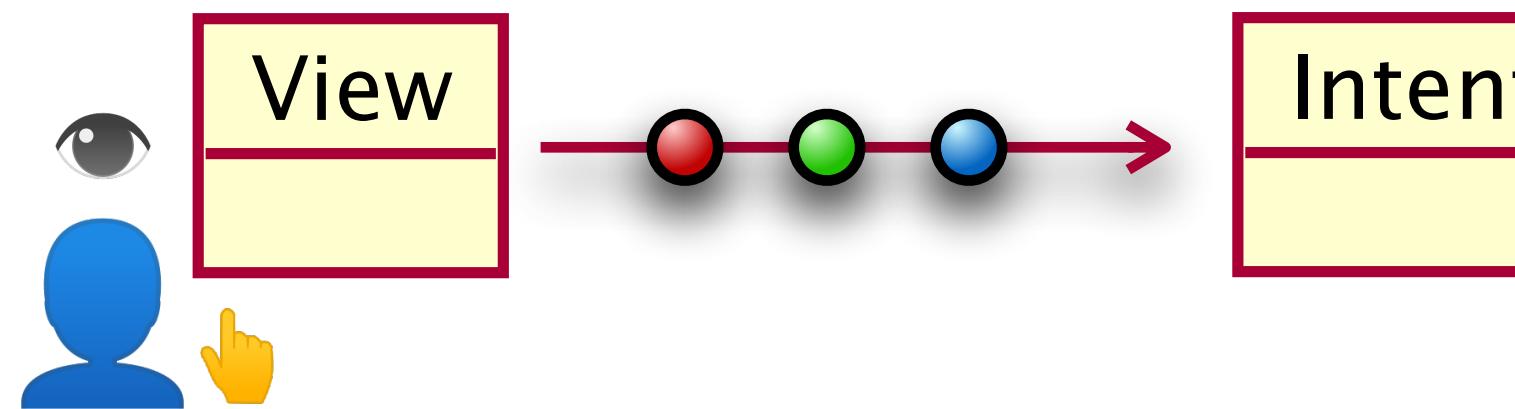


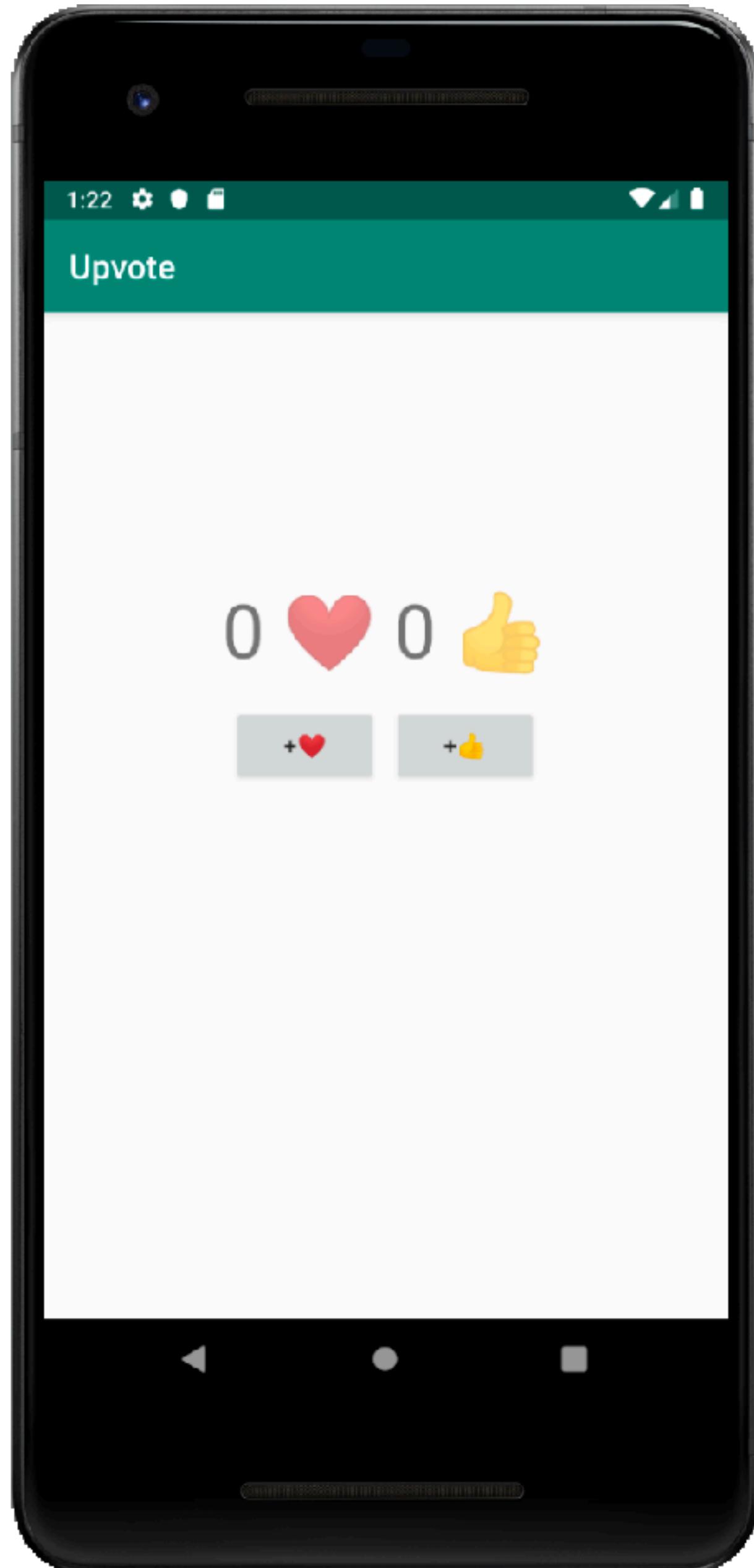
```
class MainActivity : ViewEventFlow<MainViewEvent> {  
  
    val scope: CoroutineScope = MainScope()  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        // ...  
        scope.launch { this: CoroutineScope  
            viewEvents()  
                .onEach { ? }  
                .collect()  
        }  
    }  
  
    override fun onDestroy() {  
        super.onDestroy()  
        scope.cancel()  
    }  
  
    override fun viewEvents(): Flow<MainViewEvent> {  
        // ...  
    }  
}
```



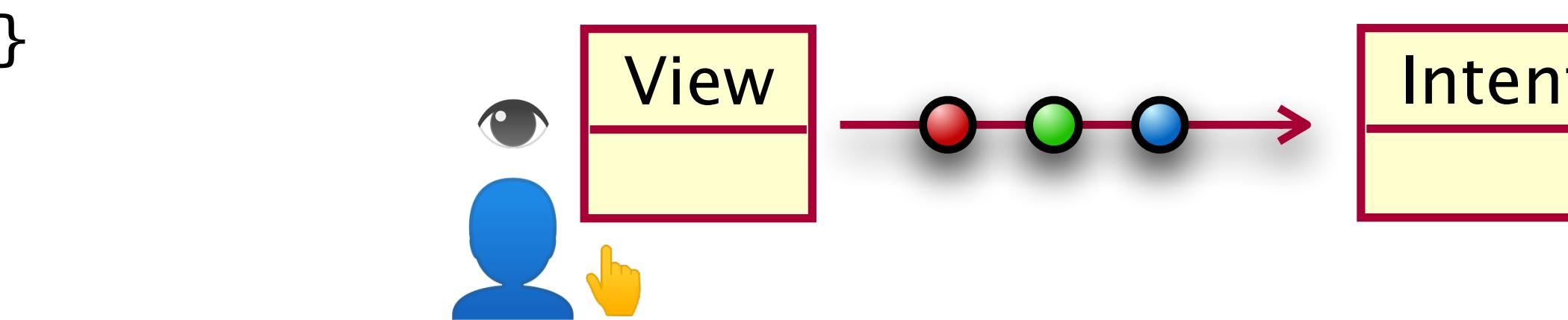


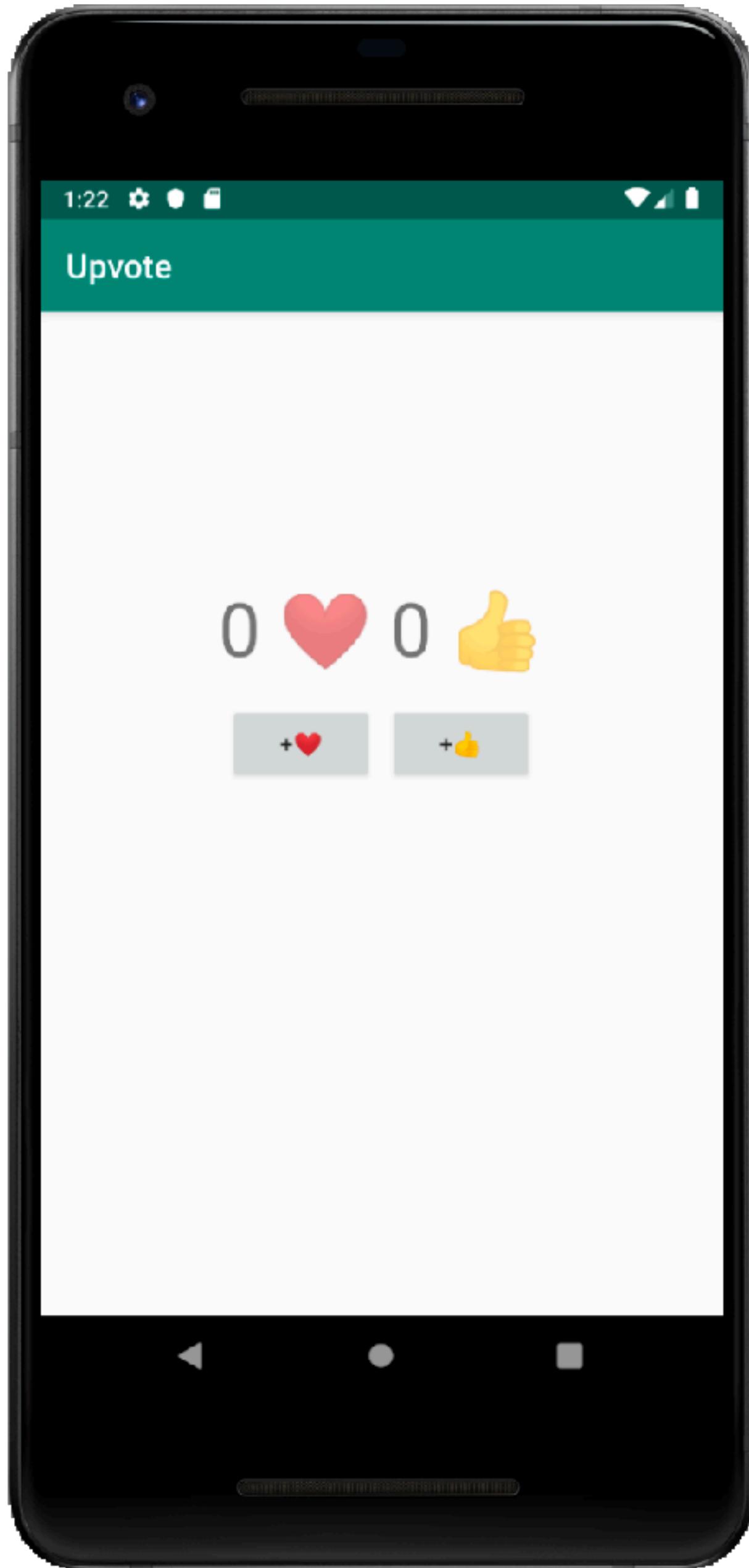
```
class MainActivity : ViewEventFlow<MainViewEvent> {  
  
    val scope: CoroutineScope = MainScope()  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        // ...  
        scope.launch_ { this: CoroutineScope  
            viewEvents()  
                .onEach { ?_ }  
                .collect()  
        }  
    }  
  
    override fun onDestroy() {  
        super.onDestroy()  
        scope.cancel()  
    }  
  
    override fun viewEvents(): Flow<MainViewEvent> {  
        // ...  
    }  
}
```



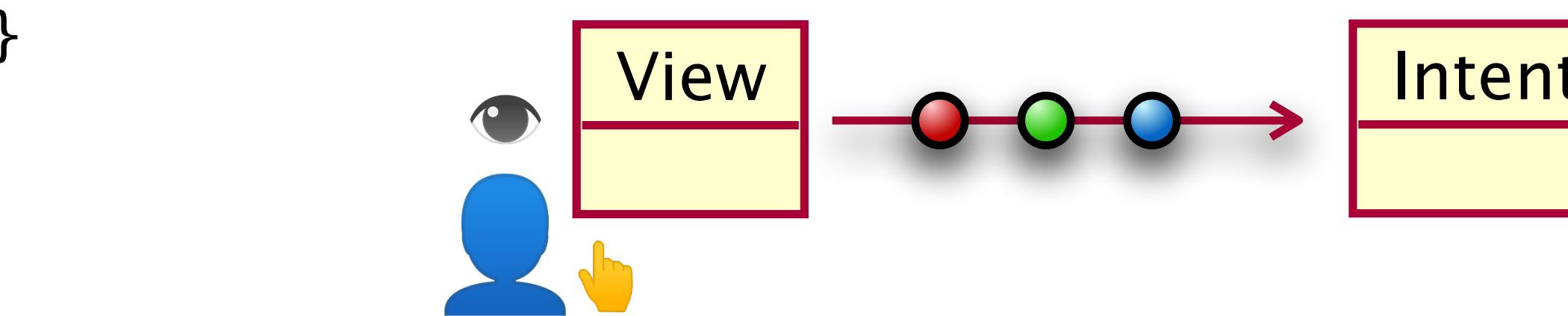


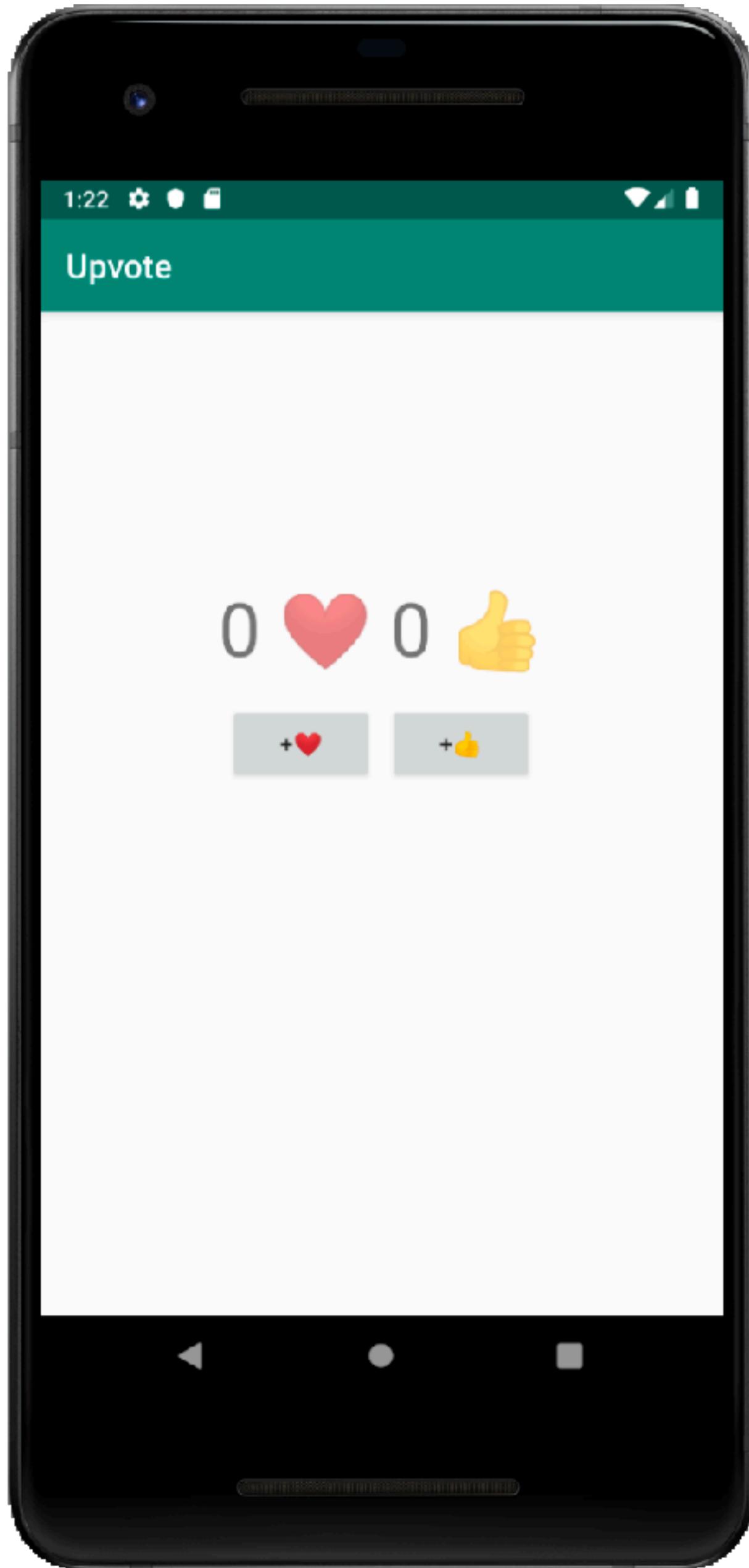
```
class MainActivity : ViewEventFlow<MainViewEvent> {  
  
    val scope: CoroutineScope = MainScope()  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        // ...  
        viewEvents()  
            .onEach { ? }  
            .launchIn(scope)  
    }  
  
    override fun onDestroy() {  
        super.onDestroy()  
        scope.cancel()  
    }  
  
    override fun viewEvents(): Flow<MainViewEvent> {  
        // ...  
    }  
}
```



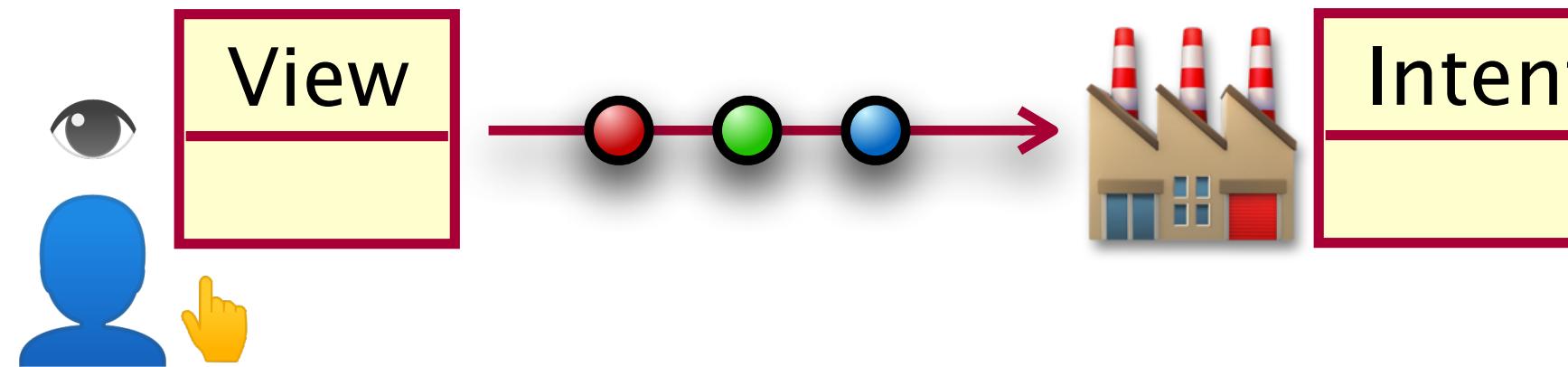


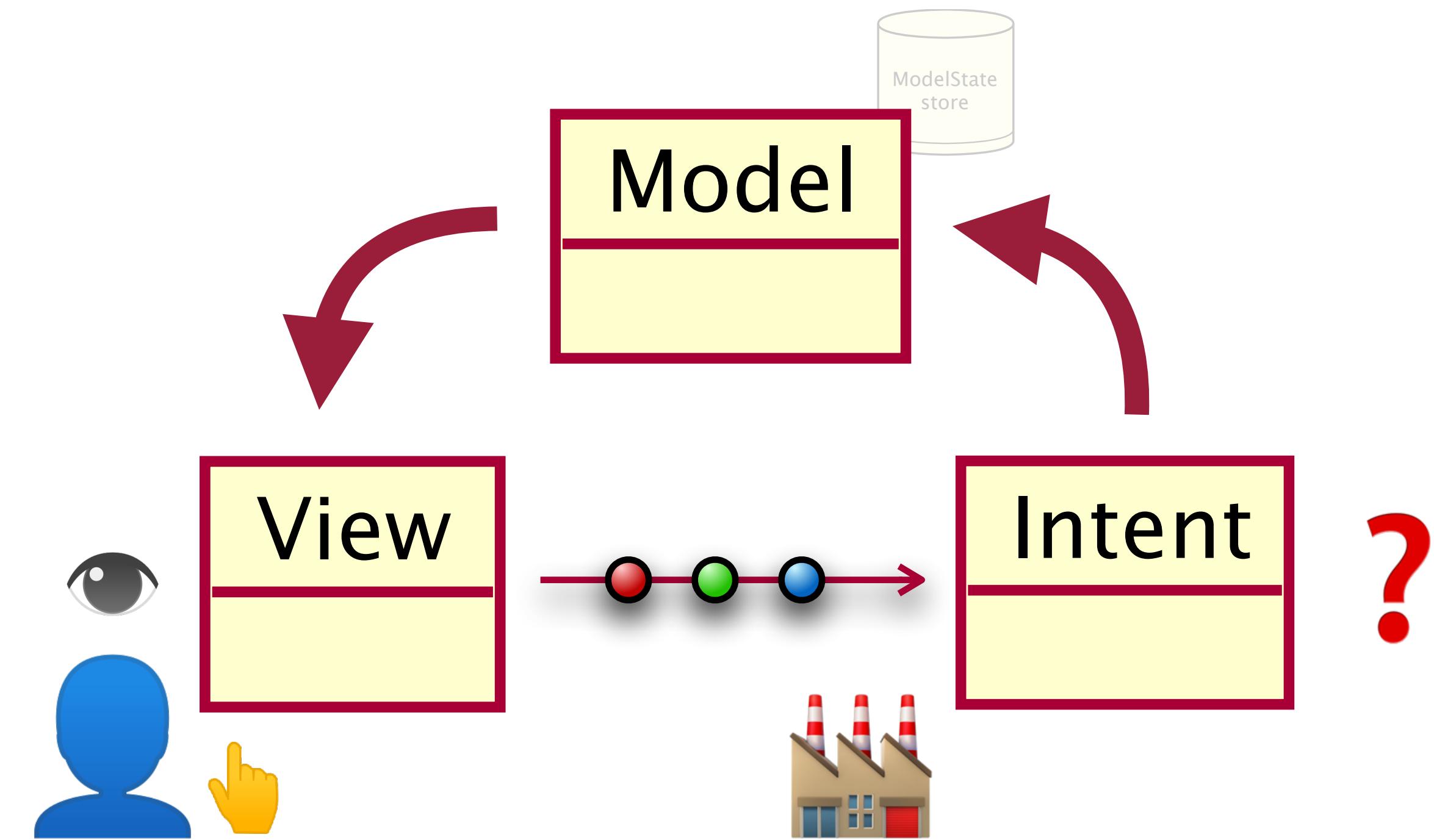
```
class MainActivity : ViewEventFlow<MainViewEvent> {  
  
    val scope: CoroutineScope = MainScope()  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        // ...  
        viewEvents()  
            .onEach { event -> ? }  
            .launchIn(scope)  
    }  
  
    override fun onDestroy() {  
        super.onDestroy()  
        scope.cancel()  
    }  
  
    override fun viewEvents(): Flow<MainViewEvent> {  
        // ...  
    }  
}
```





```
class MainActivity : ViewEventFlow<MainViewEvent> {  
  
    val scope: CoroutineScope = MainScope()  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        // ...  
        viewEvents()  
            .onEach { event ->  
                S→ MainViewIntentFactory.process(event)  
            }  
            .launchIn(scope)  
    }  
  
    override fun onDestroy() {  
        super.onDestroy()  
        scope.cancel()  
    }  
  
    override fun viewEvents(): Flow<MainViewEvent> {  
        // ...  
    }  
}
```



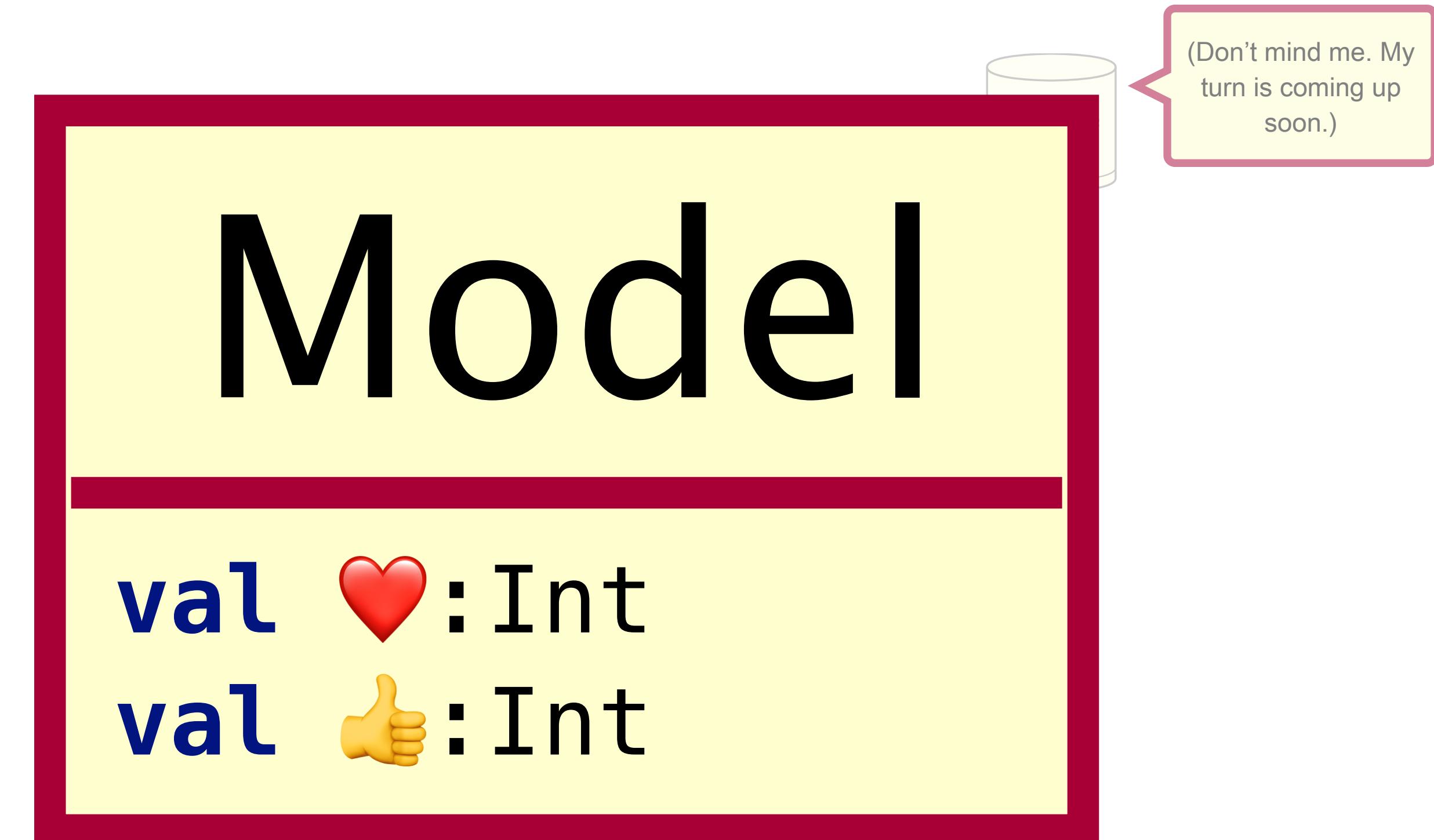


# Model

---

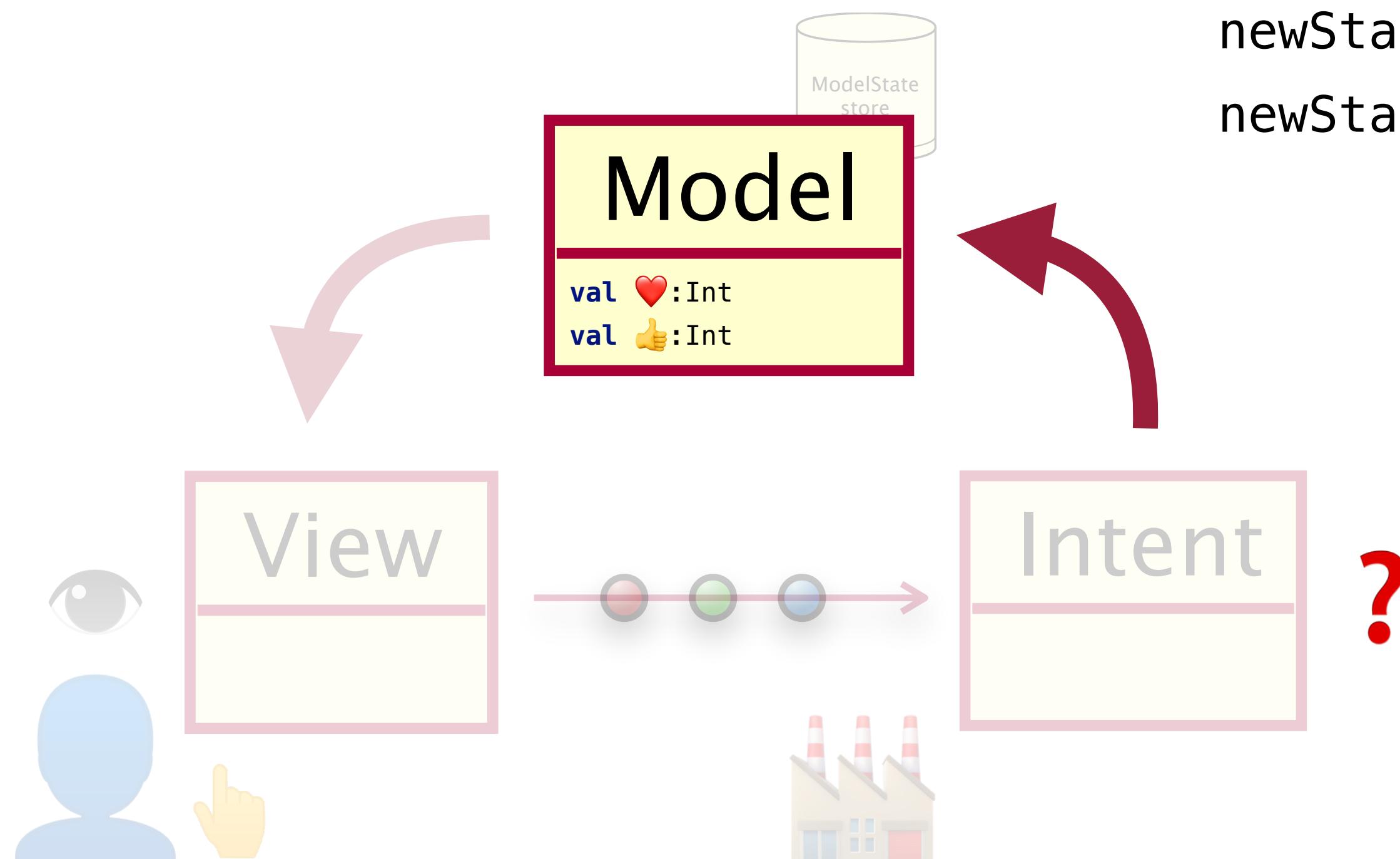
val ❤️: Int  
val 👍: Int



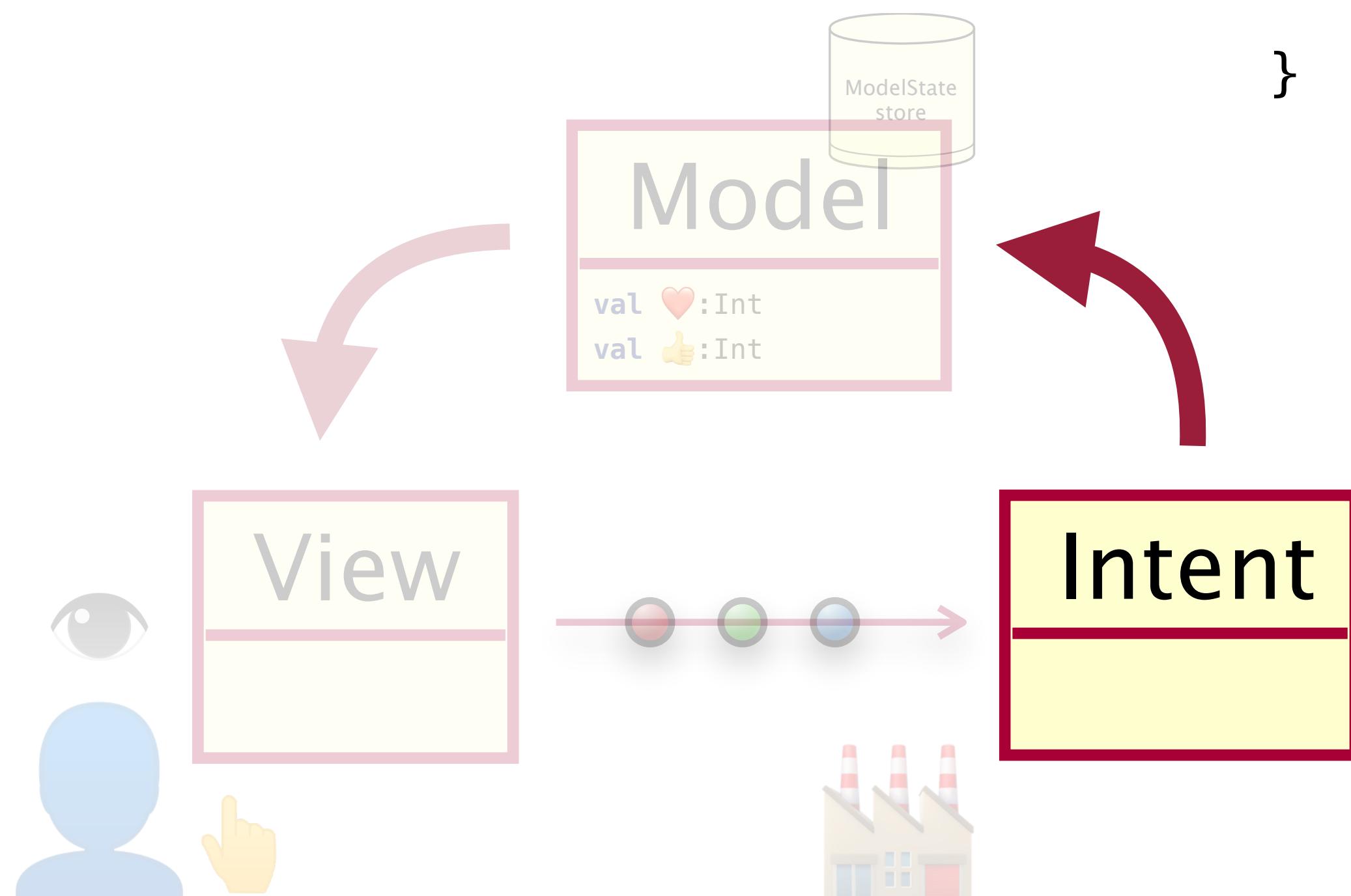


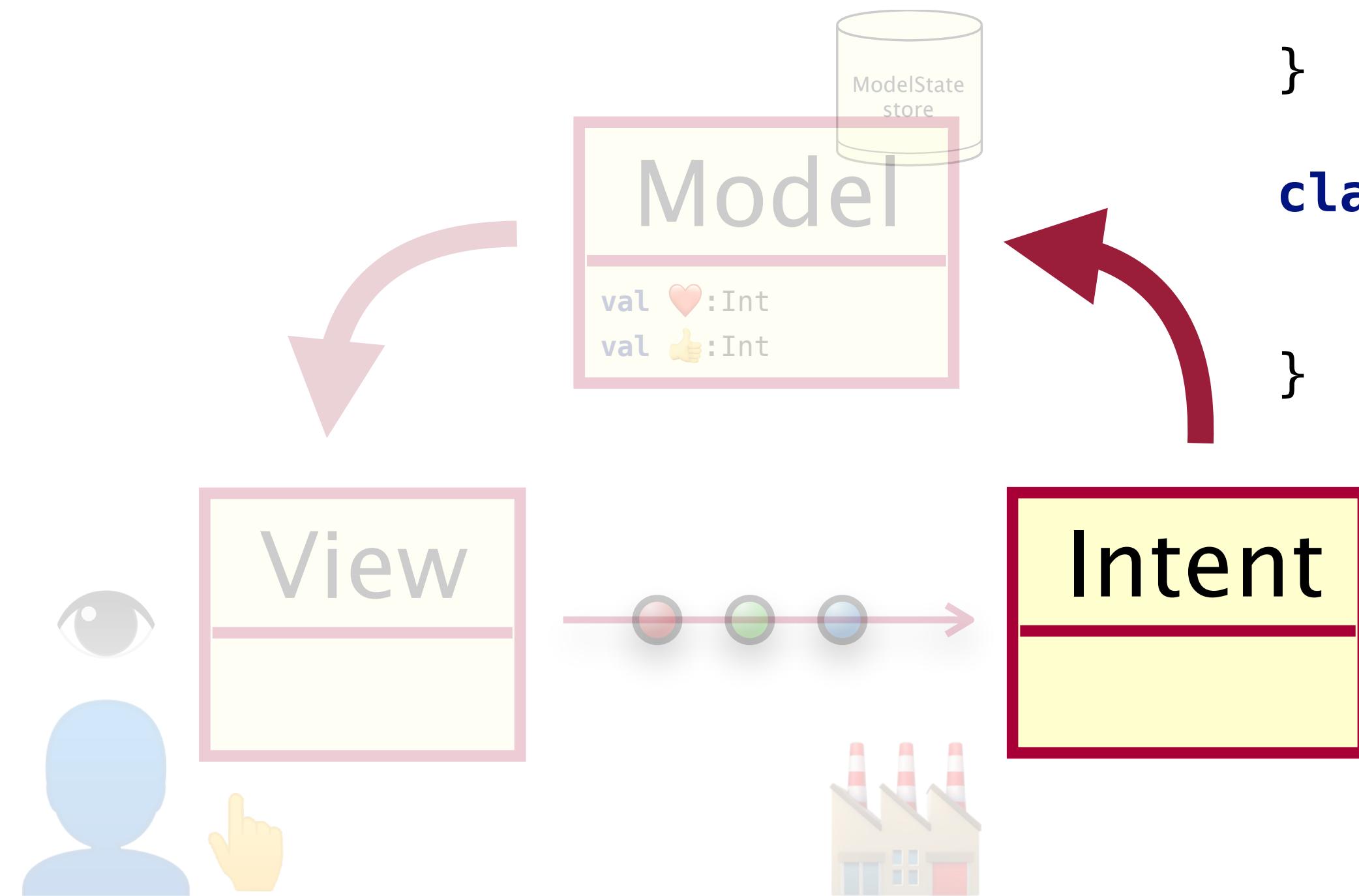
```
data class UpvoteModel(val hearts:Int, val thumbs:Int)
```

```
data class UpvoteModel(val hearts:Int, val thumbs:Int)  
  
newState = oldState.copy(thumbs = thumbs + 1) // 0, 1  
newState = newState.copy(hearts = hearts + 1) // 1, 1  
newState = newState.copy(hearts = hearts + 1) // 2, 1
```



```
data class UpvoteModel(val hearts:Int, val thumbs:Int)  
  
interface Intent<T> {  
    fun reduce(oldState: T): T  
}
```





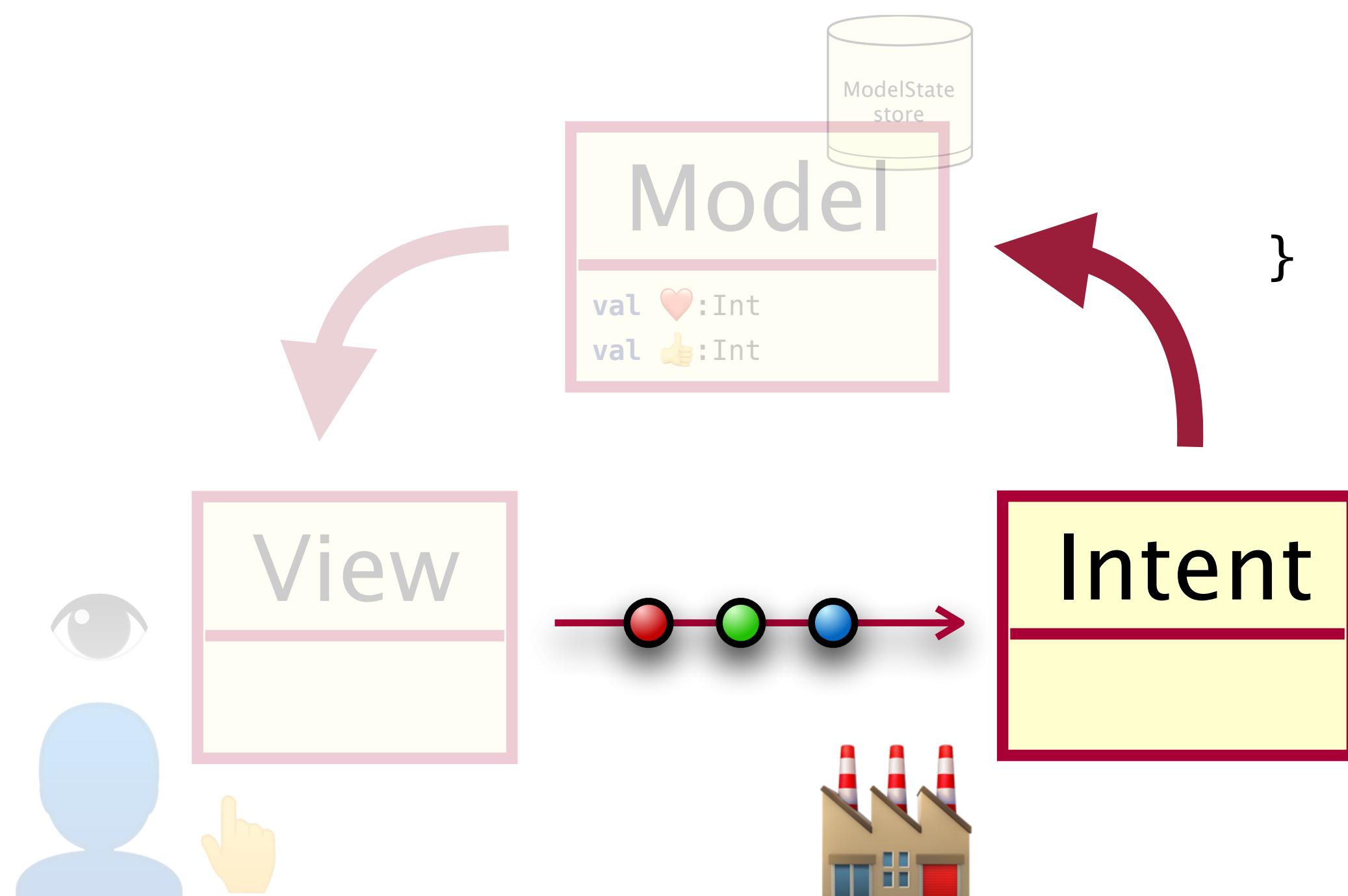
```
data class UpvoteModel(val hearts:Int, val thumbs:Int)

interface Intent<T> {
    fun reduce(oldState: T): T
}

class AddHeart():Intent<UpvoteModel> {
    override fun reduce(oldState: UpvoteModel) =
        oldState.copy(hearts = oldState.hearts + 1)
}
```

```
data class UpvoteModel(val hearts:Int, val thumbs:Int)

fun toIntent(viewEvent: MainViewEvent):Intent<UpvoteModel> {
    return when (viewEvent) {
        MainViewEvent.LoveItClick -> AddHeart()
        MainViewEvent.ThumbsUpClick -> AddThumb()
    }
}
```



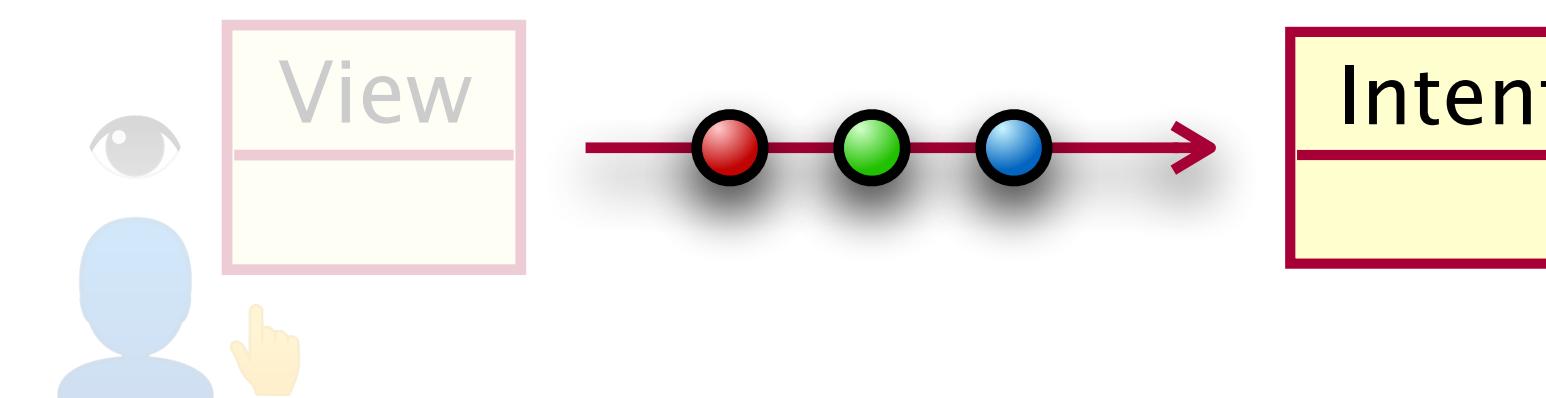


```
interface IntentFactory<E> {
    suspend fun process(viewEvent:E)
}

object MainViewIntentFactory : IntentFactory<MainViewEvent> {

    override suspend fun process(viewEvent: MainViewEvent) {
        UpvoteModelStore.process(toIntent(viewEvent))
    }

    private fun toIntent(viewEvent: MainViewEvent):Intent<UpvoteModel> {
        return when (viewEvent) {
            MainViewEvent.LoveItClick -> AddHeart()
            MainViewEvent.ThumbsUpClick -> AddThumb()
        }
    }
}
```



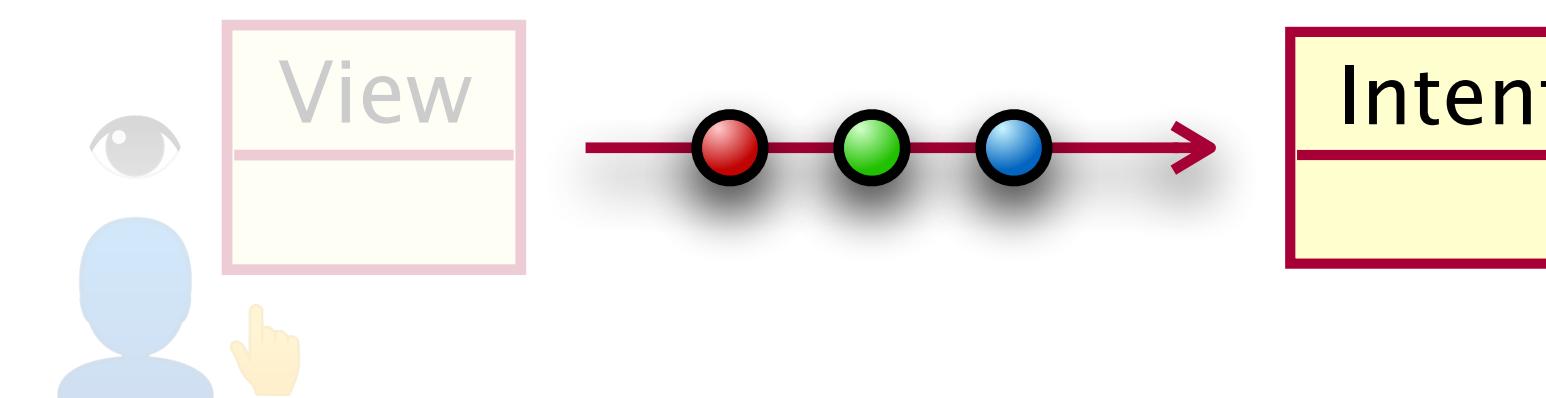


```
interface IntentFactory<E> {
    suspend fun process(viewEvent:E)
}

object MainViewIntentFactory : IntentFactory<MainViewEvent> {

    override suspend fun process(viewEvent: MainViewEvent) {
        UpvoteModelStore.process(toIntent(viewEvent))
    }

    private fun toIntent(viewEvent: MainViewEvent):Intent<UpvoteModel> {
        return when (viewEvent) {
            MainViewEvent.LoveItClick -> AddHeart()
            MainViewEvent.ThumbsUpClick -> AddThumb()
        }
    }
}
```



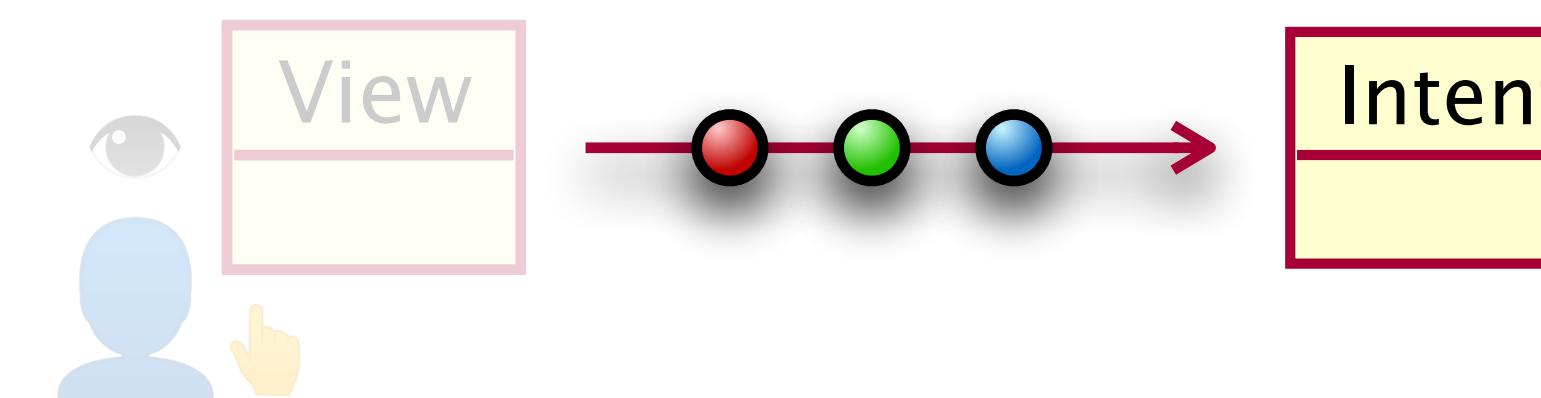


```
interface IntentFactory<E> {
    suspend fun process(viewEvent:E)
}

object MainViewIntentFactory : IntentFactory<MainViewEvent> {

    override suspend fun process(viewEvent: MainViewEvent) {
        UpvoteModelStore.process(toIntent(viewEvent))
    }

    private fun toIntent(viewEvent: MainViewEvent):Intent<UpvoteModel> {
        return when (viewEvent) {
            MainViewEvent.LoveItClick -> AddHeart()
            MainViewEvent.ThumbsUpClick -> AddThumb()
        }
    }
}
```



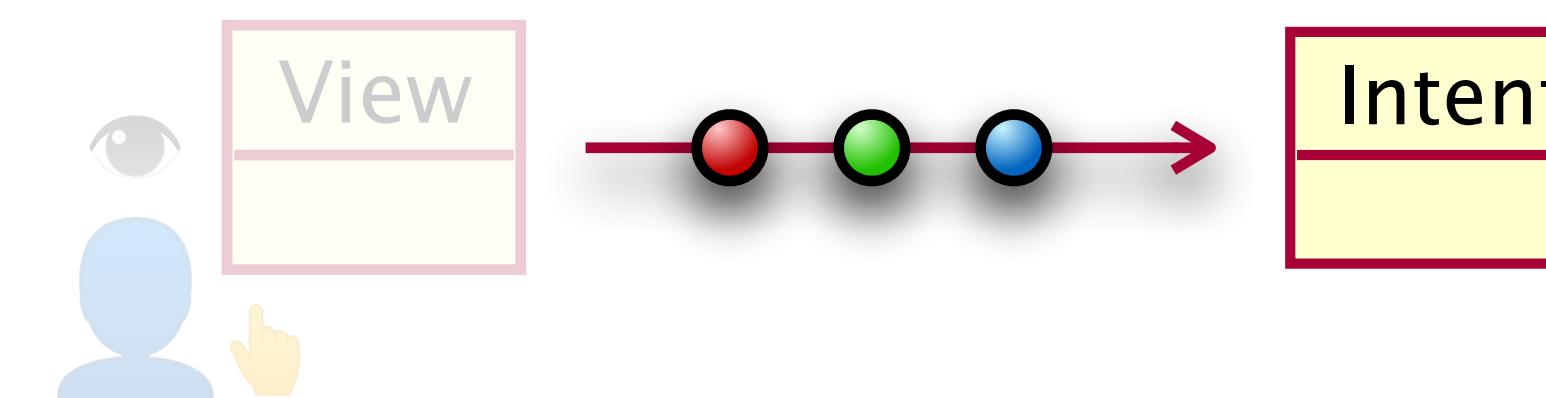


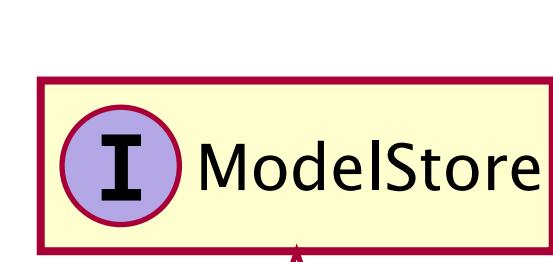
```
interface IntentFactory<E> {
    suspend fun process(viewEvent:E)
}

object MainViewIntentFactory : IntentFactory<MainViewEvent> {

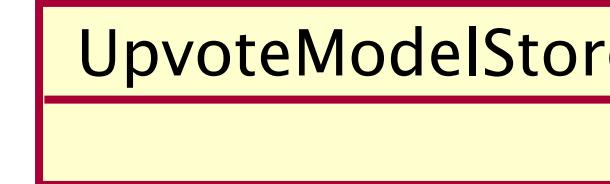
    override suspend fun process(viewEvent: MainViewEvent) {
        UpvoteModelStore.process(toIntent(viewEvent))
    }

    private fun toIntent(viewEvent: MainViewEvent):Intent<UpvoteModel> {
        return when (viewEvent) {
            MainViewEvent.LoveItClick -> AddHeart()
            MainViewEvent.ThumbsUpClick -> AddThumb()
        }
    }
}
```

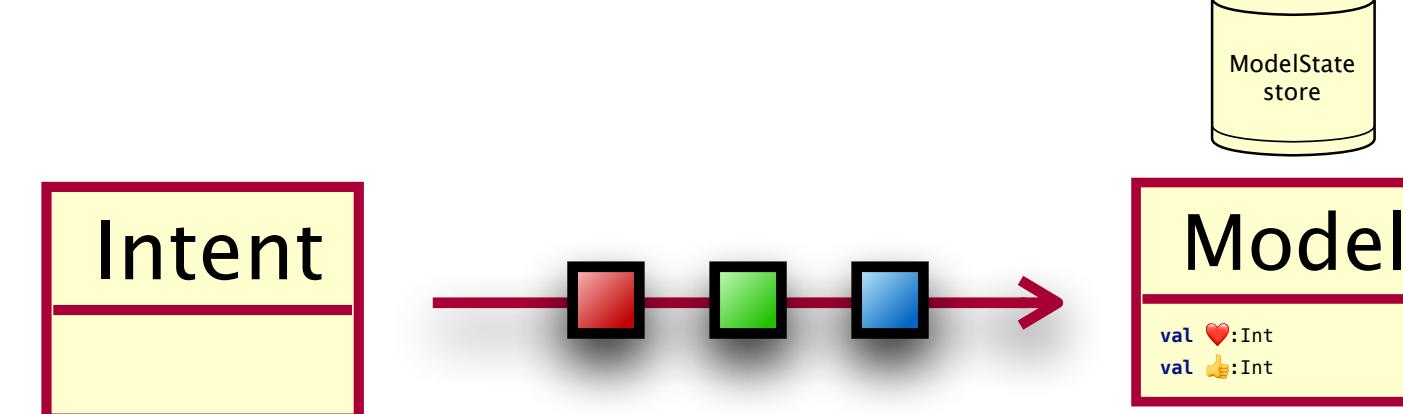


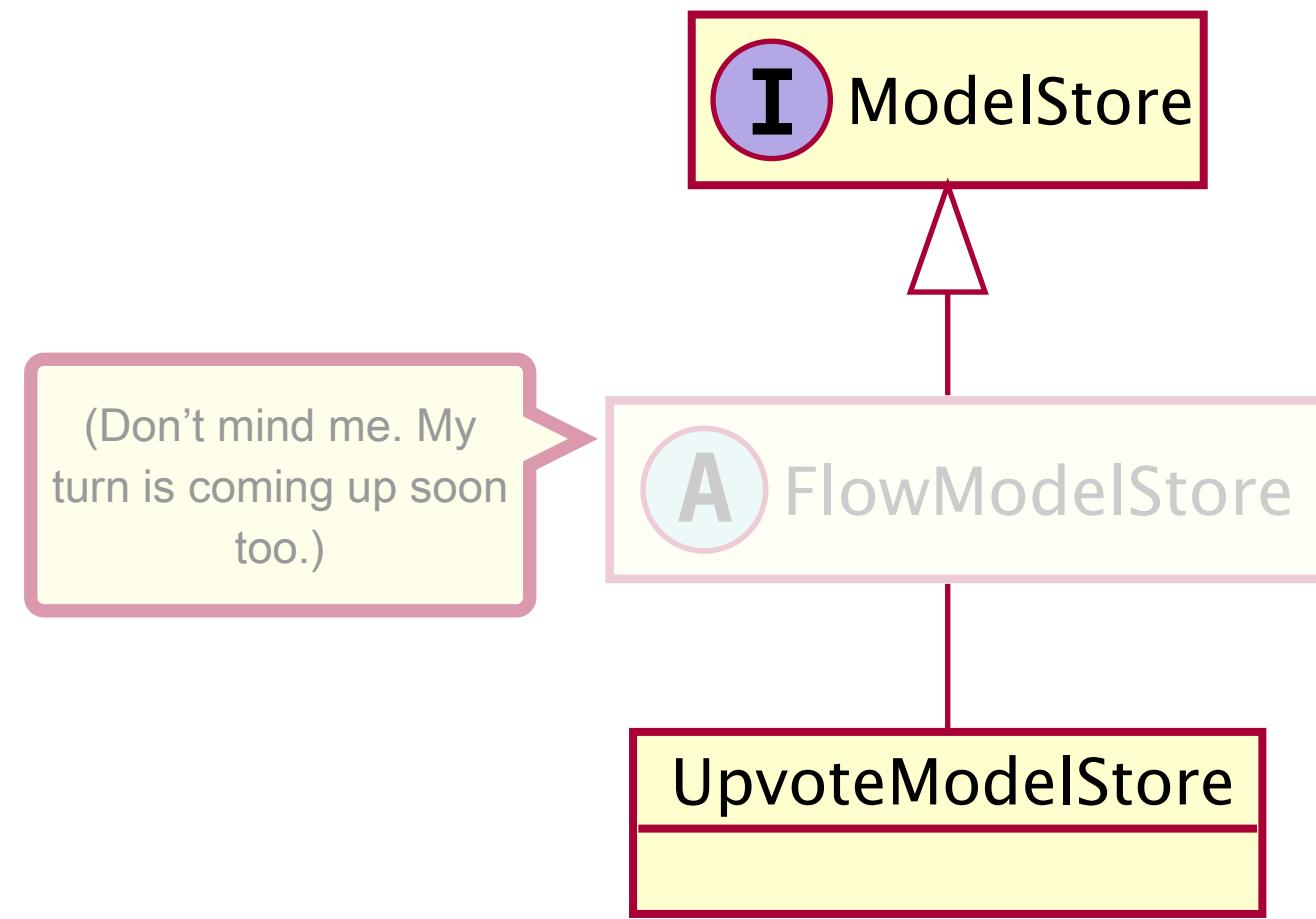


```
interface ModelStore<S> {  
    suspend fun process(intent: Intent<S>)  
    fun modelState(): Flow<S>  
}
```



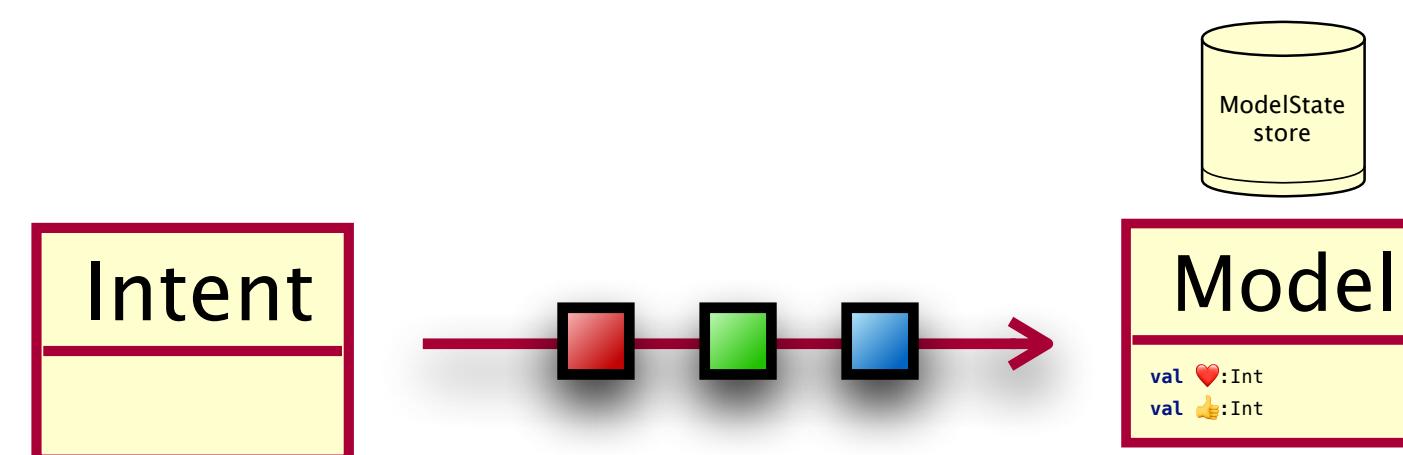
```
override suspend fun process(viewEvent: MainViewEvent) {  
    UpvoteModelStore.process(toIntent(viewEvent))  
}
```

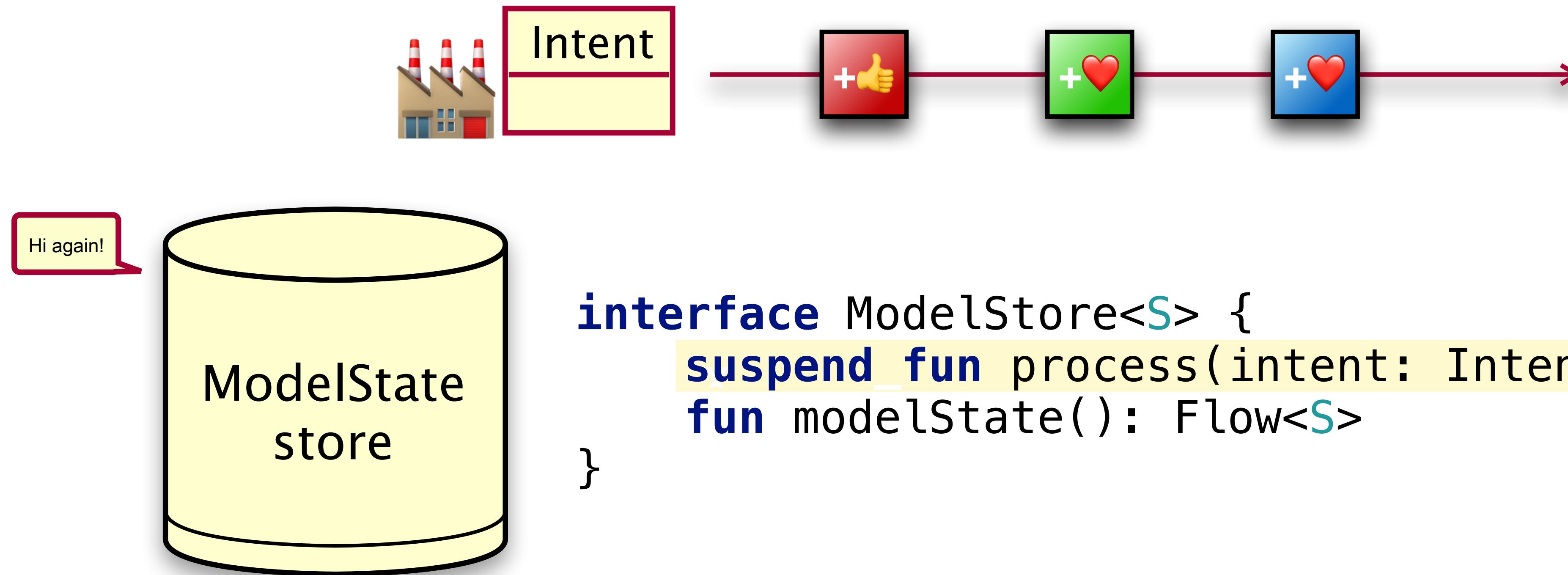




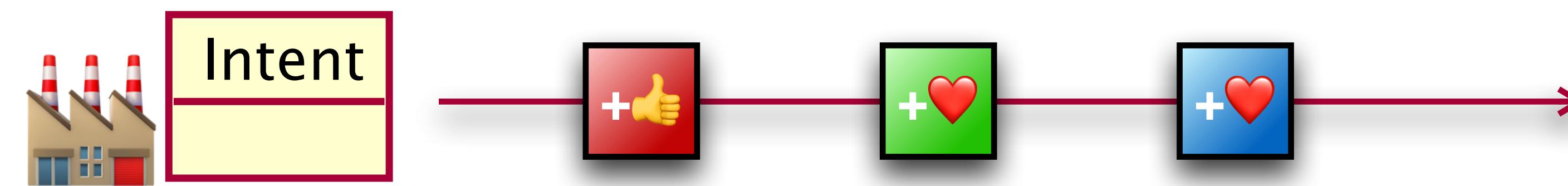
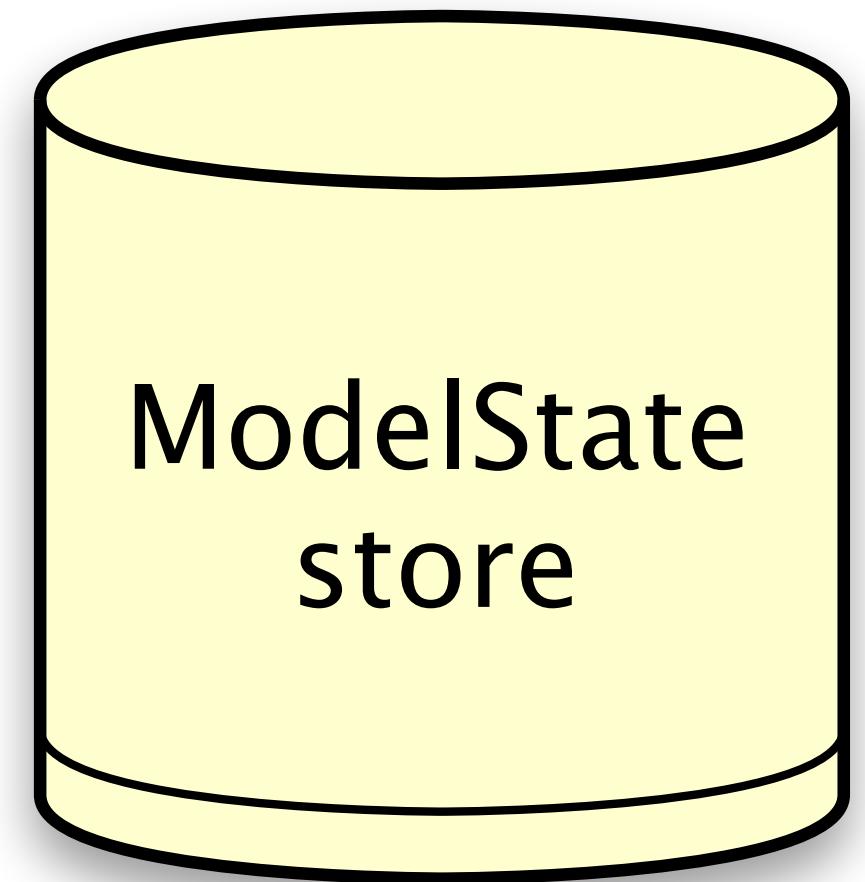
```

interface ModelStore<S> {
    suspend fun process(intent: Intent<S>)
    fun modelState(): Flow<S>
}
  
```



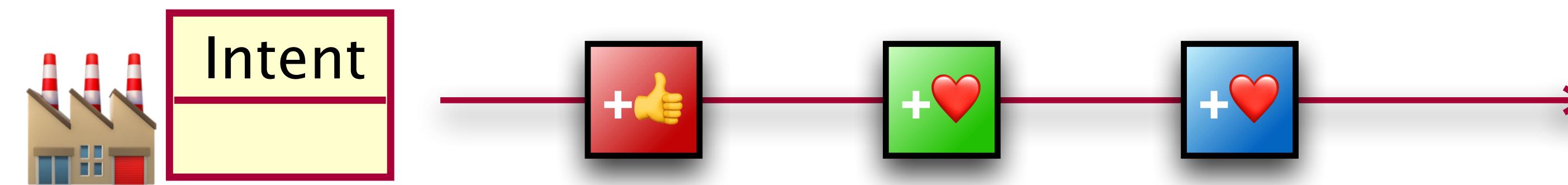
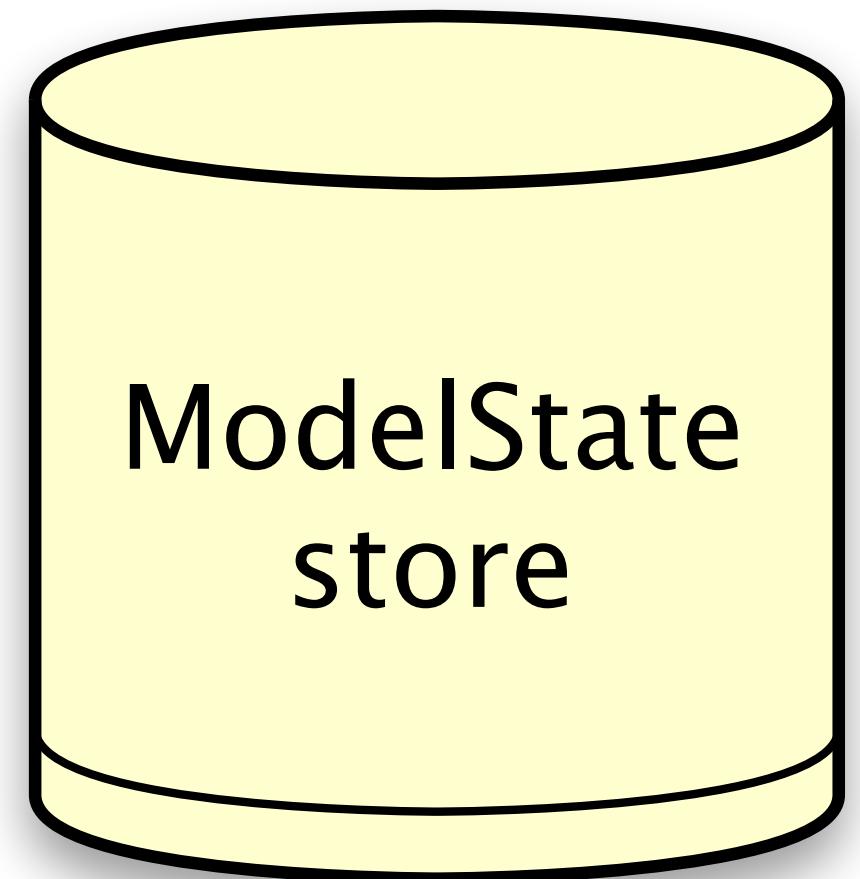


```
interface ModelStore<S> {
    suspend fun process(intent: Intent<S>)
    fun modelState(): Flow<S>
}
```



```
val intents = Channel<Intent<S>>()
```

```
suspend fun process(intent: Intent<S>) {  
    intents.send(intent)  
}
```

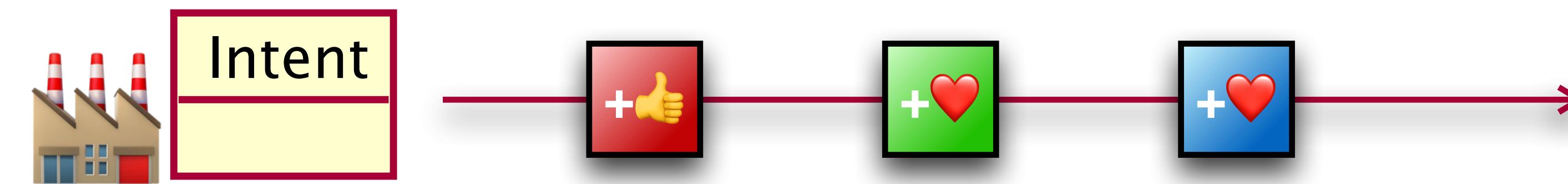


```
val intents = Channel<Intent<S>>()
```

```
suspend fun process(intent: Intent<S>) {  
    intents.send(intent)  
}
```

```
val store = ConflatedBroadcastChannel(startingState)
```

```
override fun modelState(): Flow<S> {  
    return store.asFlow()  
}
```



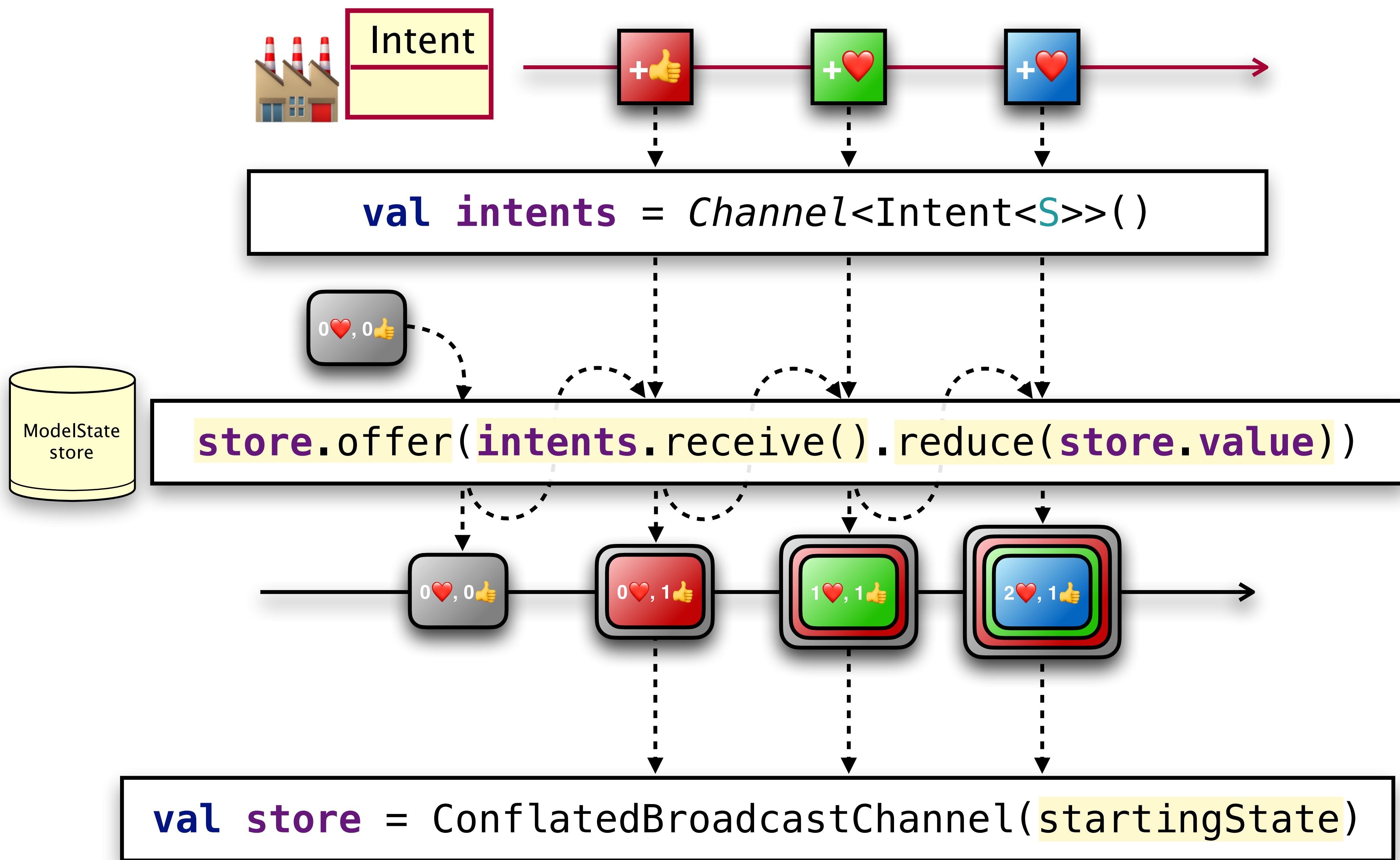
```
val intents = Channel<Intent<S>>()
```

?

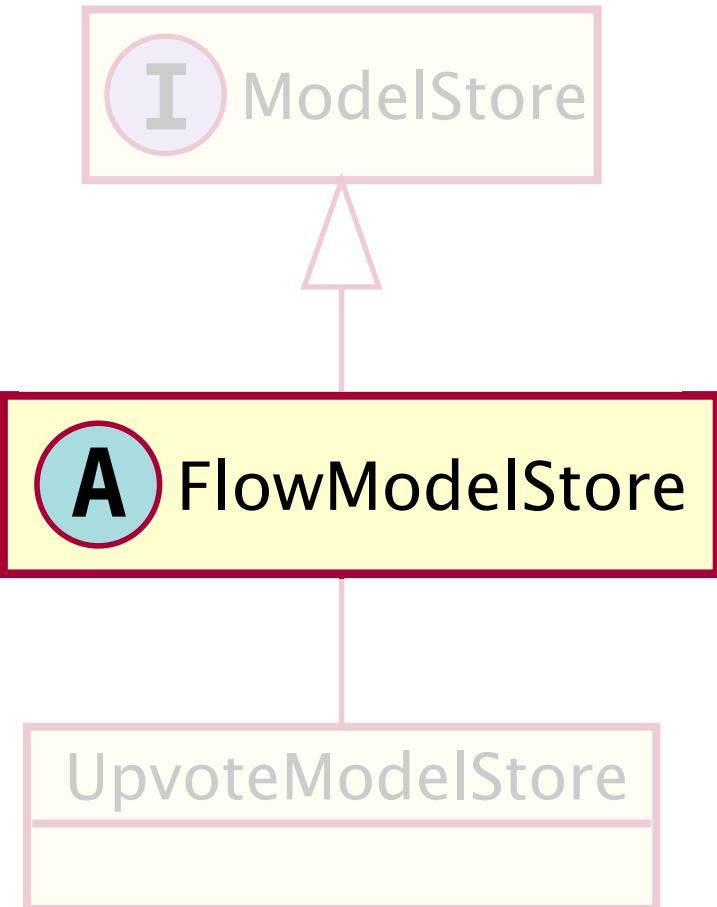
?

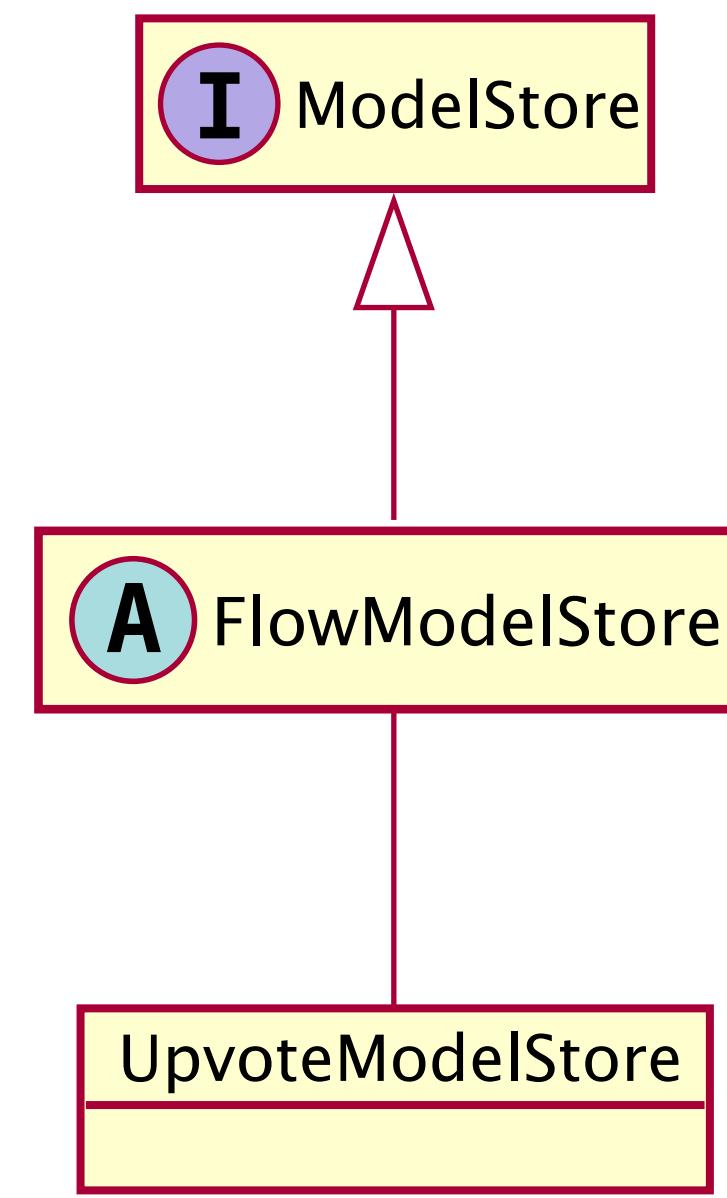
?

```
val store = ConflatedBroadcastChannel(startingState)
```



```
open class FlowModelStore<S>(startingState: S) : ModelStore<S> {  
    private val scope = MainScope()  
    private val intents = Channel<Intent<S>>()  
    private val store = ConflatedBroadcastChannel(startingState)  
  
    init {  
        // Reduce from MainScope()  
        scope.launch {  
            while (isActive) store.offer(intents.receive().reduce(store.value))  
        }  
    }  
  
    // Could be called from any coroutine scope/context.  
    override suspend fun process(intent: Intent<S>) {  
        intents.send(intent)  
    }  
  
    override fun modelState(): Flow<S> {  
        return store.asFlow()  
    }  
  
    fun close() {  
        intents.close()  
        store.close()  
        scope.cancel()  
    }  
}
```





```
interface ModelStore<S> {
    suspend fun process(intent: Intent<S>)
    fun modelState(): Flow<S>
}
```

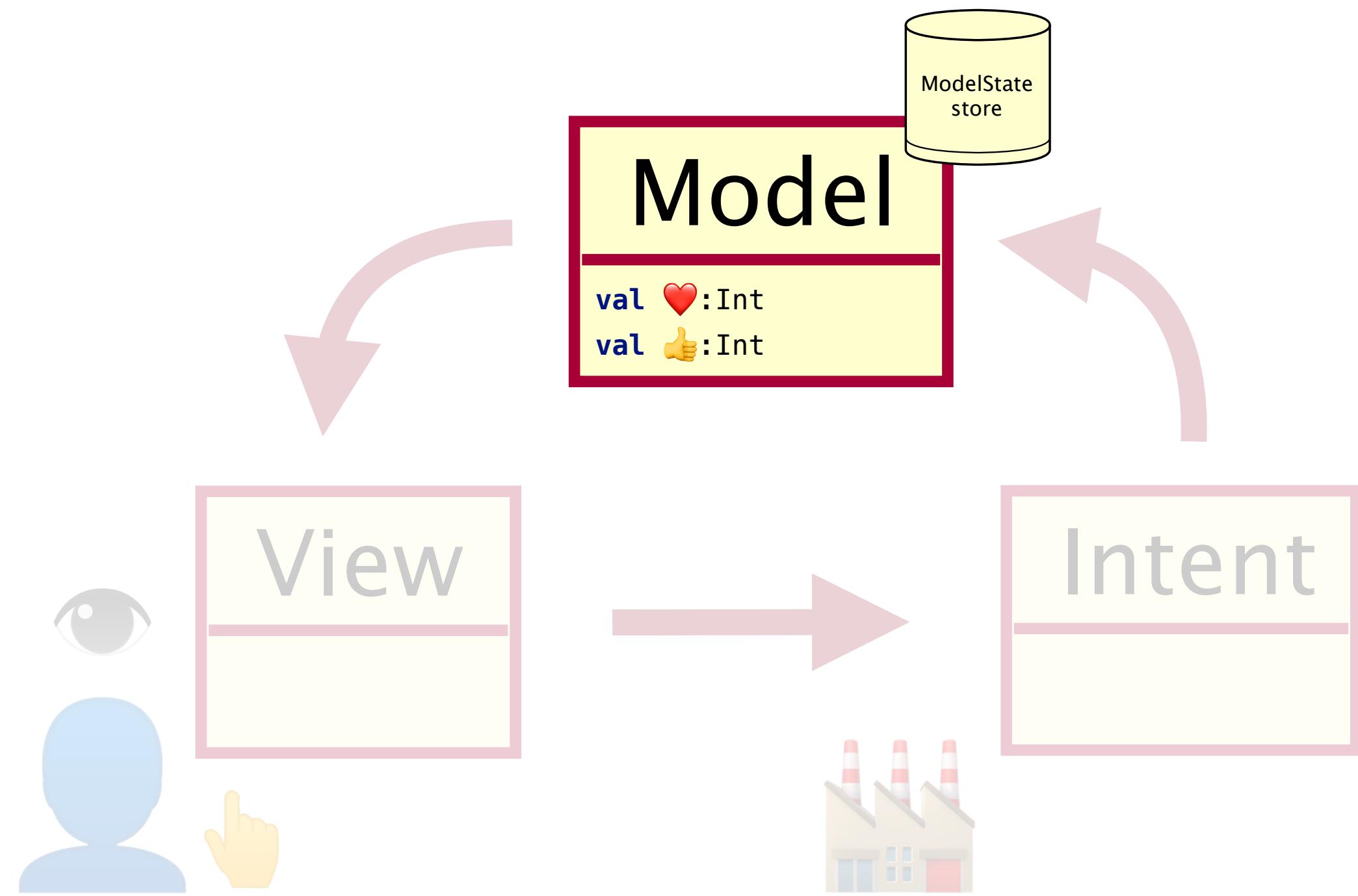
```
open class FlowModelStore<S>(startingState: S) : ModelStore<S> {
    // ...
}
```

```
object UpvoteModelStore :  
    FlowModelStore<UpvoteModel>(UpvoteModel(0, 0))
```

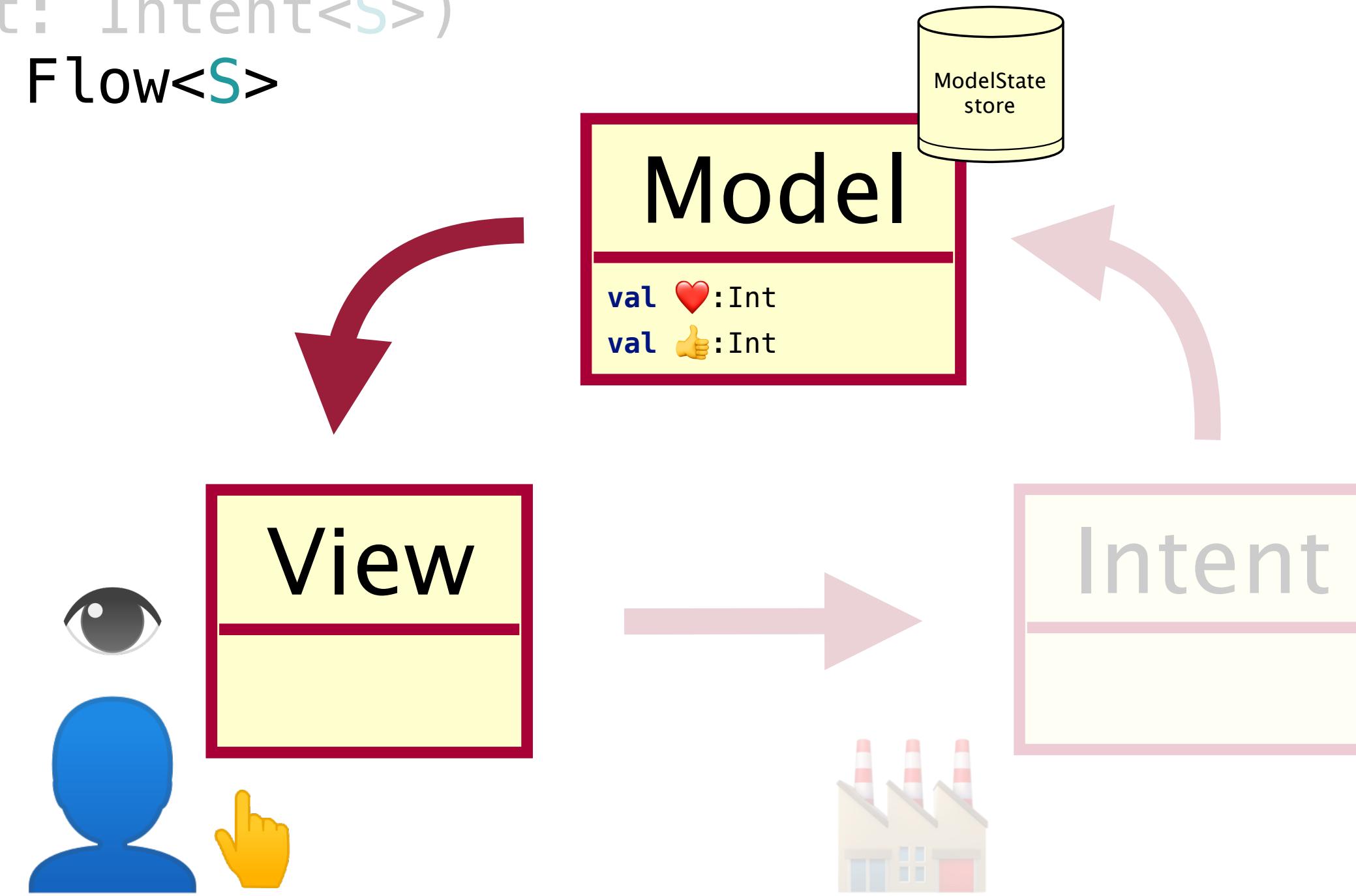
```
UpvoteModelStore
```

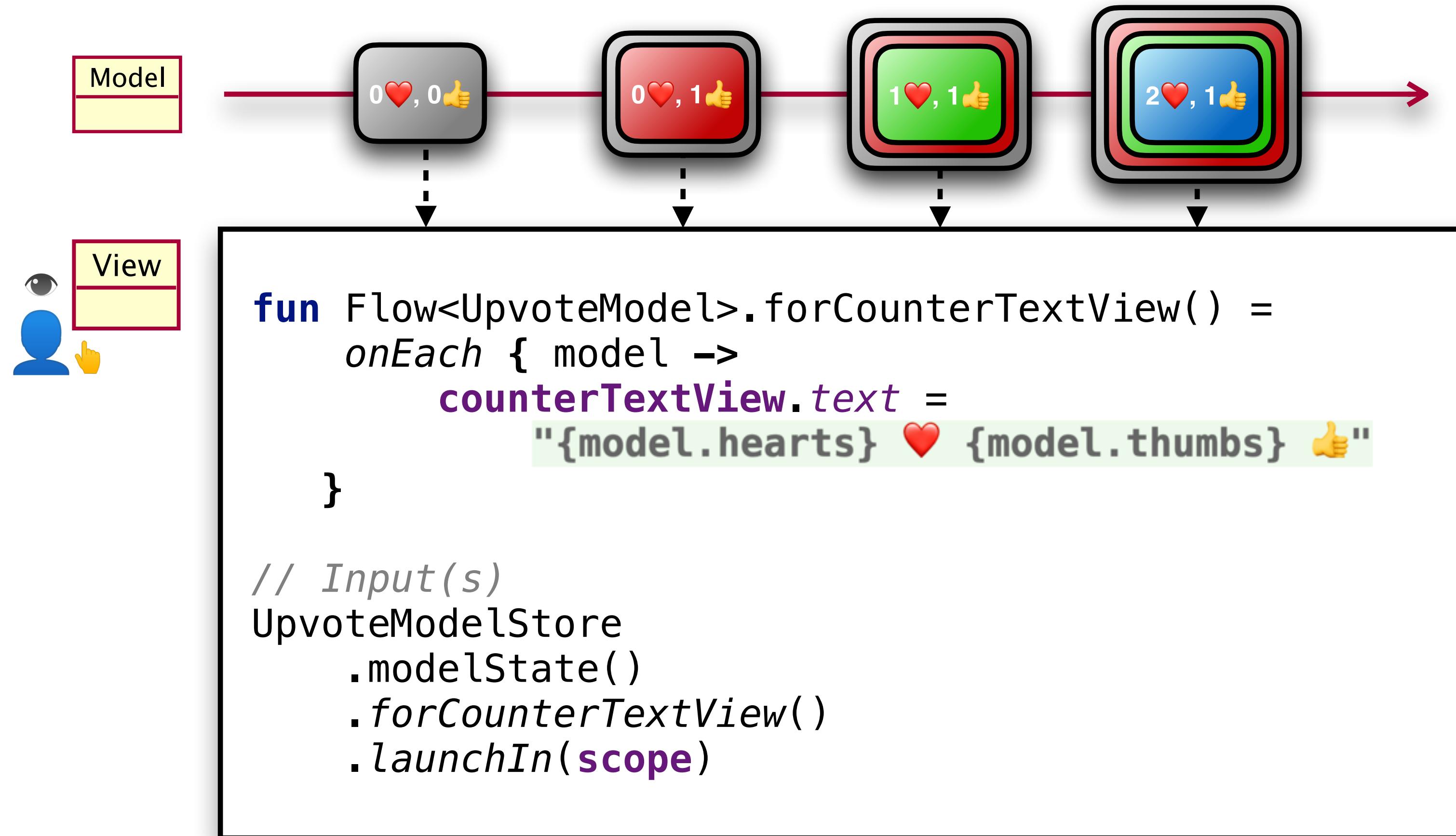
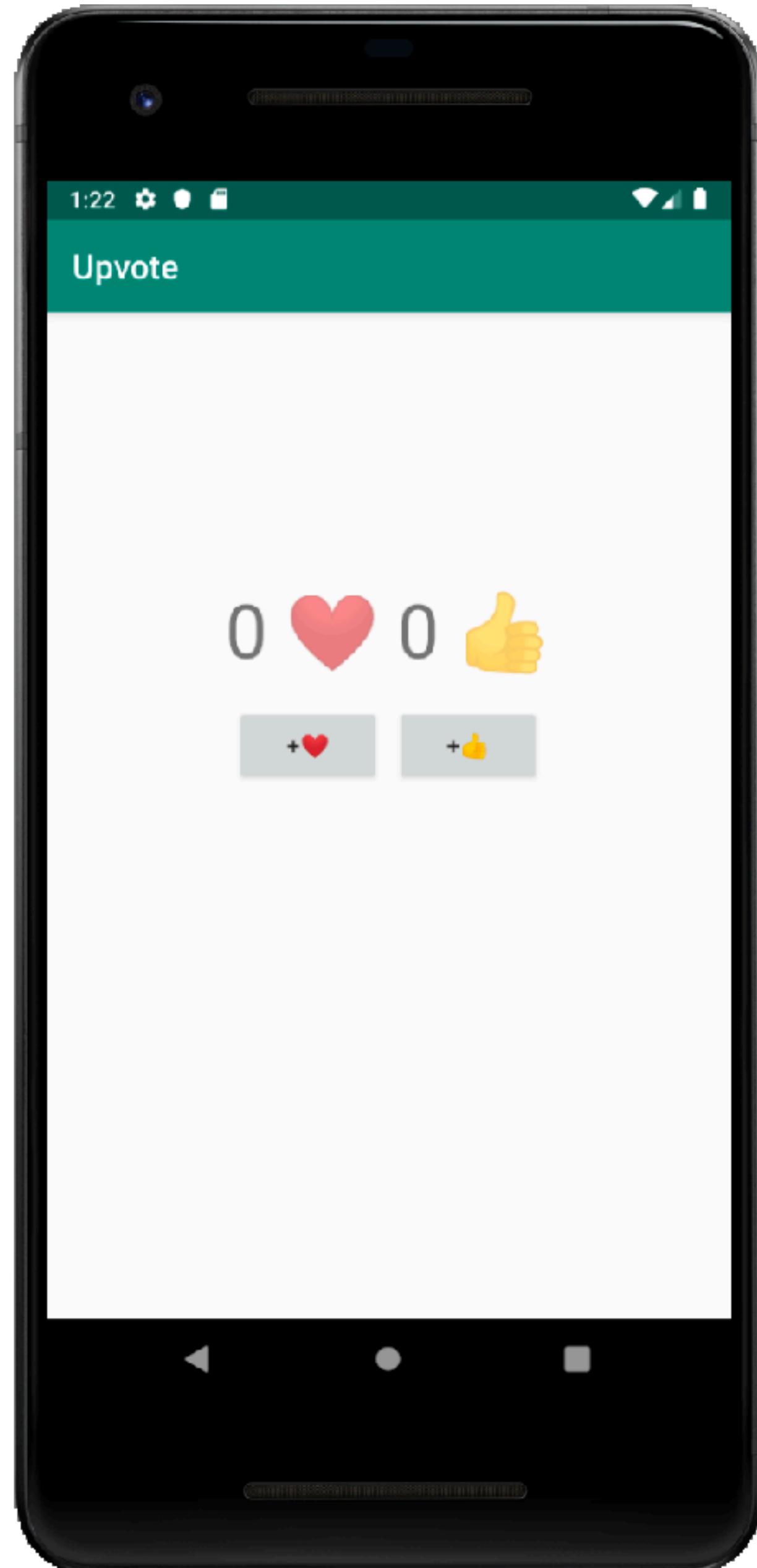
```
object UpvoteModelStore :  
    FlowModelStore<UpvoteModel>(UpvoteModel(0, 0))
```

## **object** UpvoteModelStore



```
object UpvoteModelStore  
  
interface ModelStore<S> {  
    fun process(intent: Intent<S>)  
    fun modelState(): Flow<S>  
}
```





# Live demo?



kanawish/upvote at flow

GitHub, Inc. [US] | [github.com/kanawish/upvote/tree/flow](https://github.com/kanawish/upvote/tree/flow)

20+ Notes GDE Caster Speaking Research 16-04-22 App Cli... ASG Archive Other Bookmarks

Search or jump to... Pull requests Issues Marketplace Explore

kanawish / upvote Watch 1 Star 13 Fork 1

Code Issues 0 Pull requests 0 Projects 0 Wiki Security Insights Settings

Simplest possible demo MVF App Edit

Manage topics

26 commits 2 branches 0 packages 0 releases 1 contributor

Branch: flow New pull request Create new file Upload files Find File Clone or download

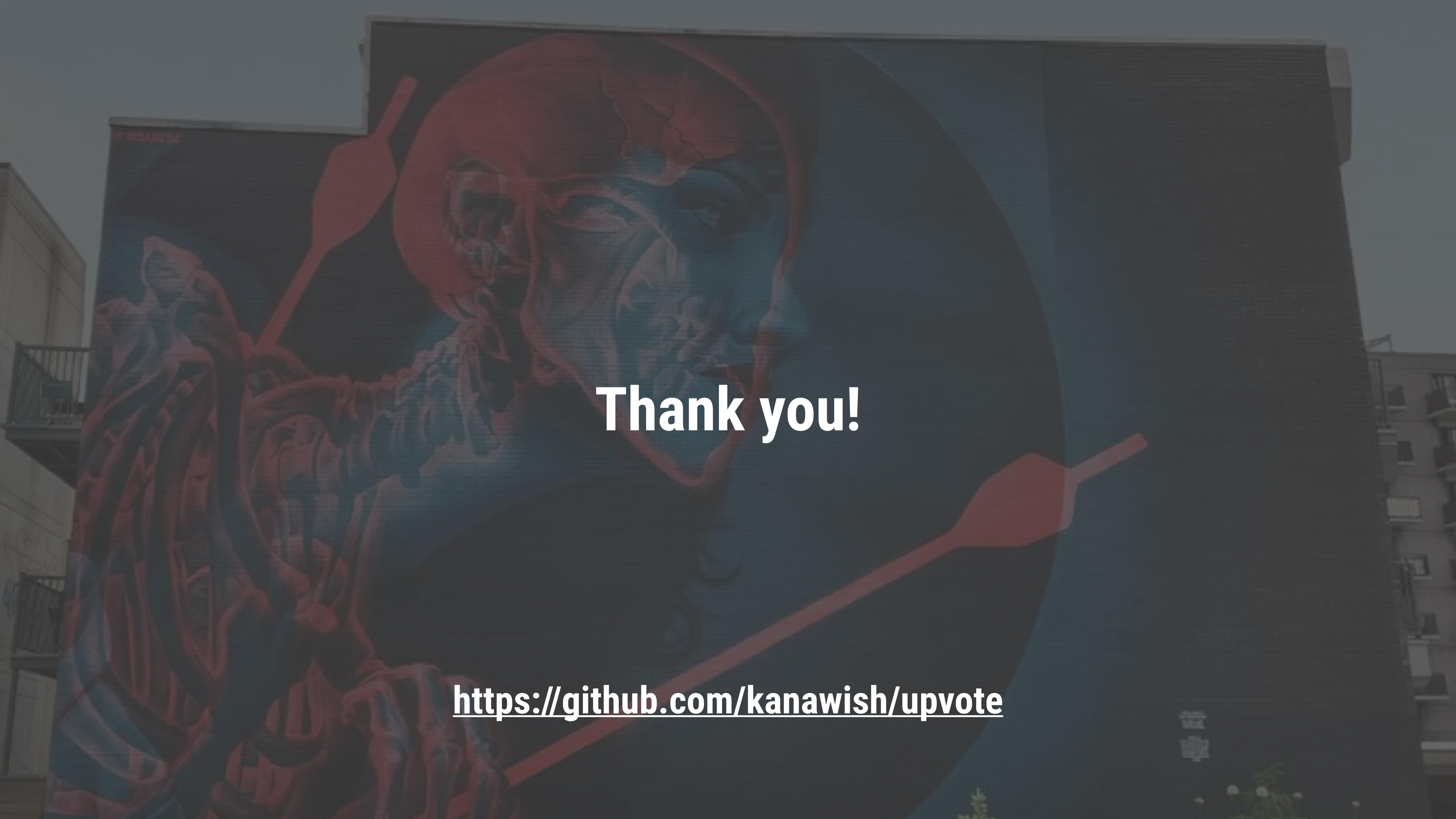
This branch is 20 commits ahead of master.

Pull request Compare

kanawish working basic version for talk Latest commit 50e5a07 yesterday

File	Message	Time
.idea	Some cleanups and adjustments.	2 months ago
app	working basic version for talk	yesterday
docs	Playing around with `hu.akarnokd.kotlin.flow`, store not yet usable.	17 days ago
gradle/wrapper	Some cleanups and adjustments.	2 months ago
.gitignore	Some cleanups and adjustments.	2 months ago
README.md	Update README.md	5 months ago
build.gradle	Playing around with `hu.akarnokd.kotlin.flow`, store not yet usable.	17 days ago
gradle.properties	First commit	5 months ago
gradlew	First commit	5 months ago

<https://github.com/kanawish/upvote>

A large mural of a red octopus with tentacles wrapped around a building facade. The octopus has a textured, reddish-brown body with darker spots and stripes. Its tentacles are draped over the windows and edges of the building. The building itself is a light-colored concrete structure with some architectural details visible through the tentacles.

Thank you!

<https://github.com/kanawish/upvote>