#7

Android Studio

Android Studio

“ MVVM - State handling, what about the future...? ”

INNEHÅLLSFÖRTECKNING

01 Översikt

02 Recap, Toasts & Snackbars

03 ViewModel #2

04 Uppgifter & Övningar

01

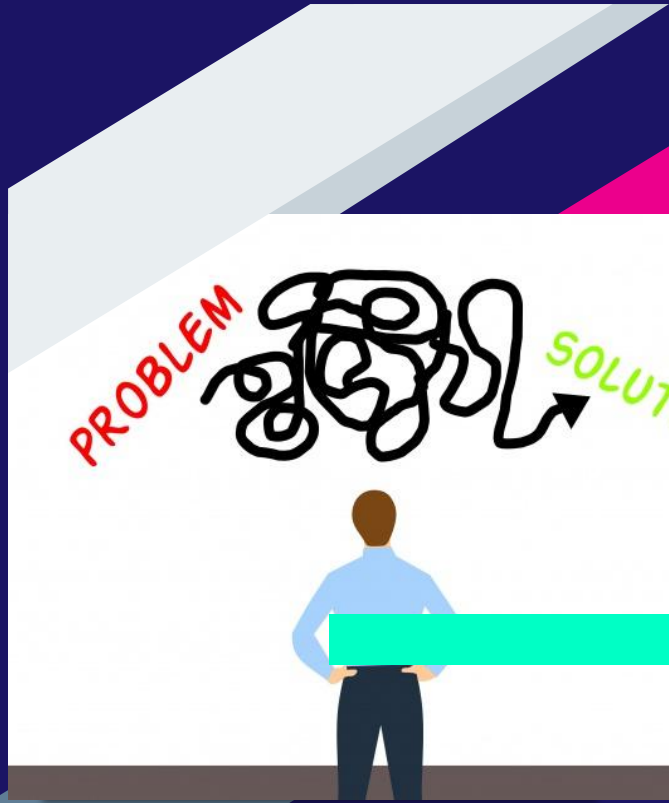
ÖVERSIKT


Problems…

Intents?

Navigering, vidareskickande av data…
Inte så bra!

- Inconsistent…
- Business Logic in Activities…
- Applications grow into a mess!!!!



02

Recap & Toasts + Snackbars


TOASTS

TOASTS

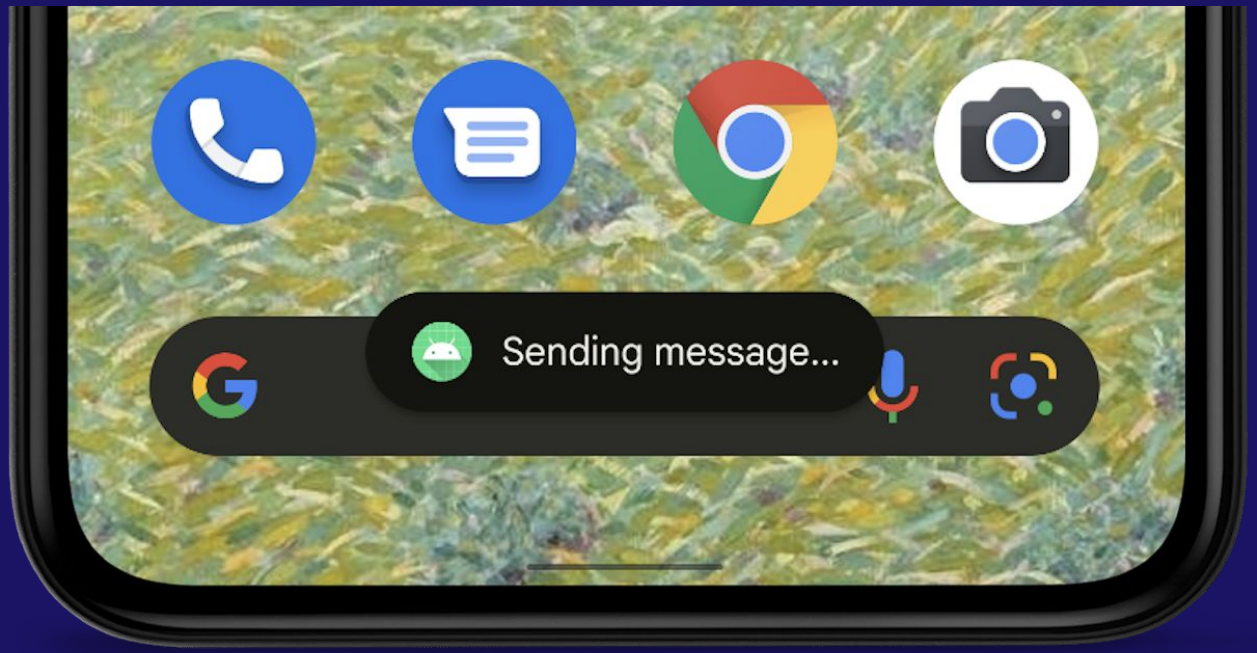
Alternatives to using toasts

If your app is in the foreground, consider using a snackbar instead of using a toast. Snackbars include user-actionable options, which can provide a better app experience.

If your app is in the background, and you want users to take some action, use a notification instead.

TOASTS

Mini pop-up!



TOASTS

31 >

If your app targets Android 12 (API level 31) or higher, its toast is limited to two lines of text and shows the application icon next to the text. Be aware that the line length of this text varies by screen size, so it's good to make the text as short as possible.

Toast

Skapa Toasts!

Här definierar vi texten, längden och bygger sedan vår ‘toast’ för att sedan visa upp den!

```
val text = "Hello toast!"
val duration = Toast.LENGTH_SHORT

val toast = Toast.makeText(applicationContext, text, duration)
toast.show()
```

TOASTS

```
val text = "Hello toast!"
val duration = Toast.LENGTH_SHORT

val toast = Toast.makeText(applicationContext, text, duration)
toast.show()
```

android.widget.Toast? ⌥⏎

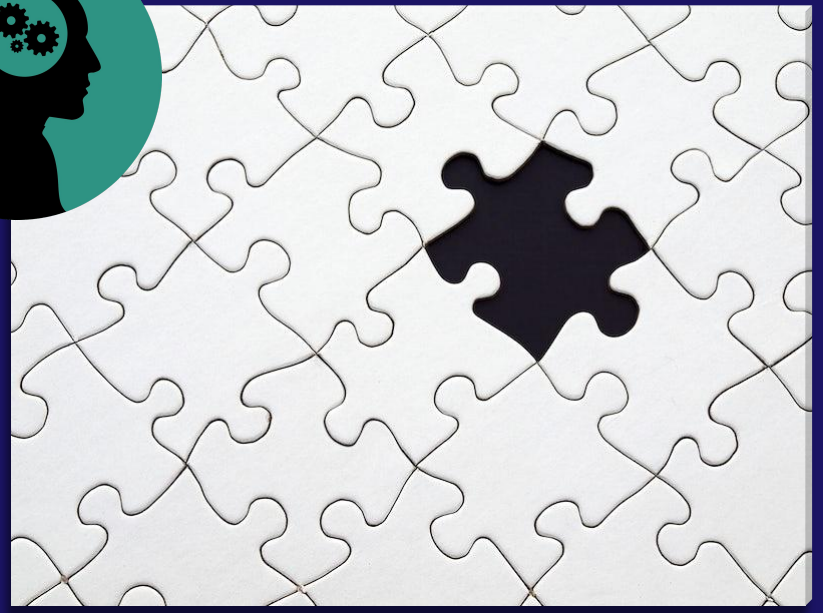
Problem

Hur visar jag upp en liten ruta med text?

Solution

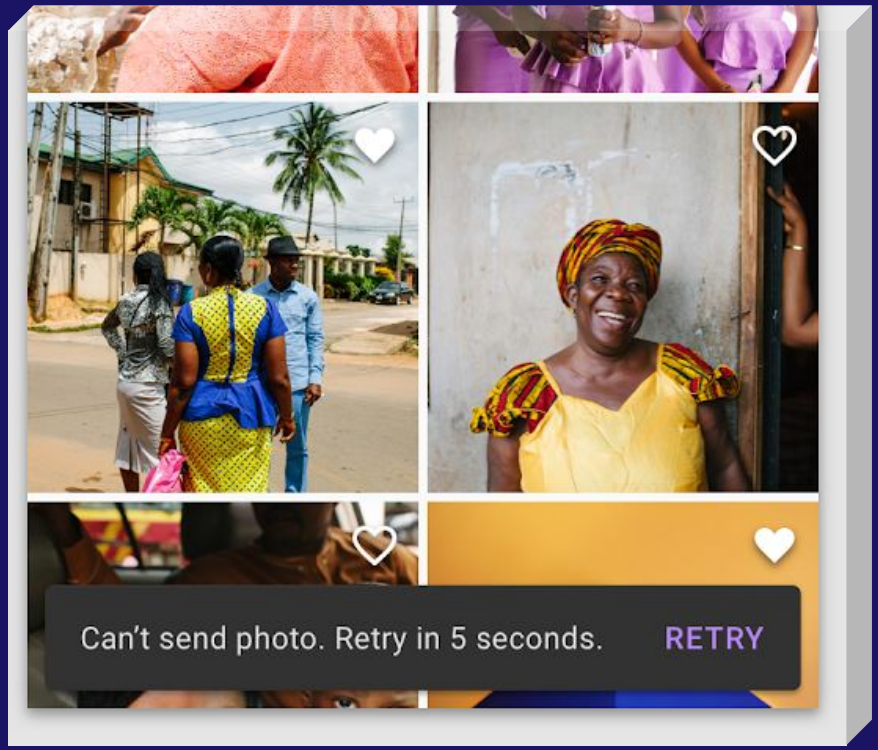
Om appen är i bakgrunden och du vill skicka en enkel text, prova Toasts!

Frågor?




Snackbar

Snackbar



Här har vi ett alternativ mot Toasts!
Men denna dialog har en knapp!

Snackbar

Usage

Snackbars inform users of a process that an app has performed or will perform. They appear temporarily, towards the bottom of the screen. They shouldn't interrupt the user experience, and they don't require user input to disappear.

Frequency

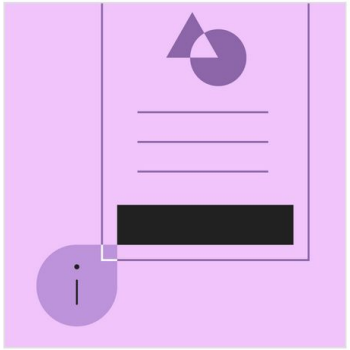
Only one snackbar may be displayed at a time.

Actions

A snackbar can contain a single action. "Dismiss" or "cancel" actions are optional.

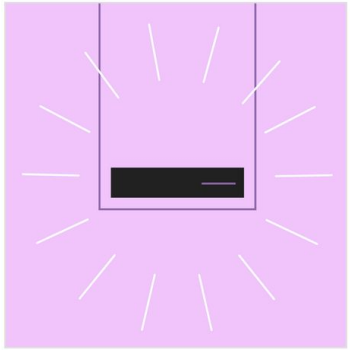
Snackbar

Principles



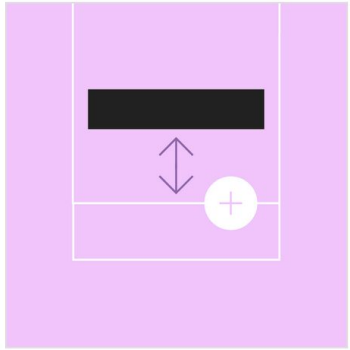
Informational

Snackbars provide updates on an app's processes.



Temporary

Snackbars appear temporarily, and disappear on their own without requiring user input to be dismissed.



Contextual

Snackbars are placed in the most suitable area of the UI.

Snackbar

When to use

Snackbars communicate messages that are minimally interruptive and don’t require user action.

Component	Priority	User action
Snackbar	Low priority	Optional: Snackbars disappear automatically
Banner	Prominent, medium priority	Optional: Banners remain until dismissed by the user, or if the state that caused the banner is resolved
Dialog	Highest priority	Required: Dialogs block app usage until the user takes a dialog action or exits the dialog (if available)

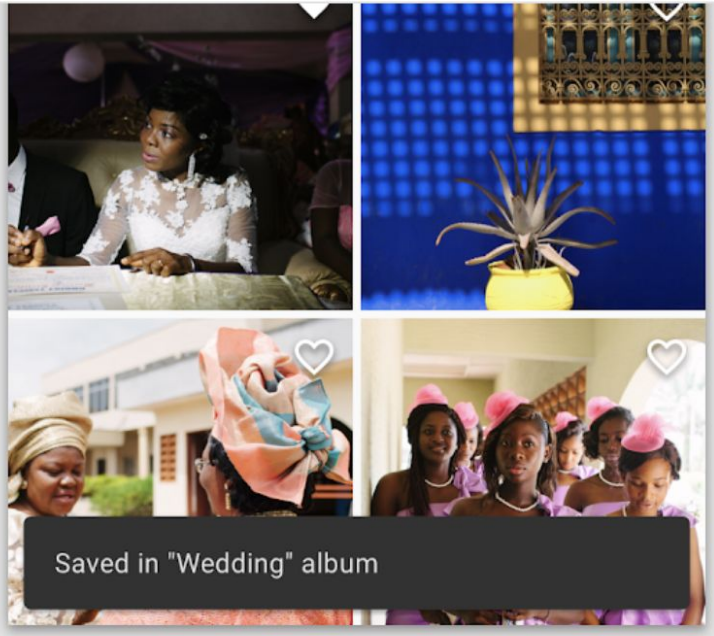
Snackbar

Anatomy

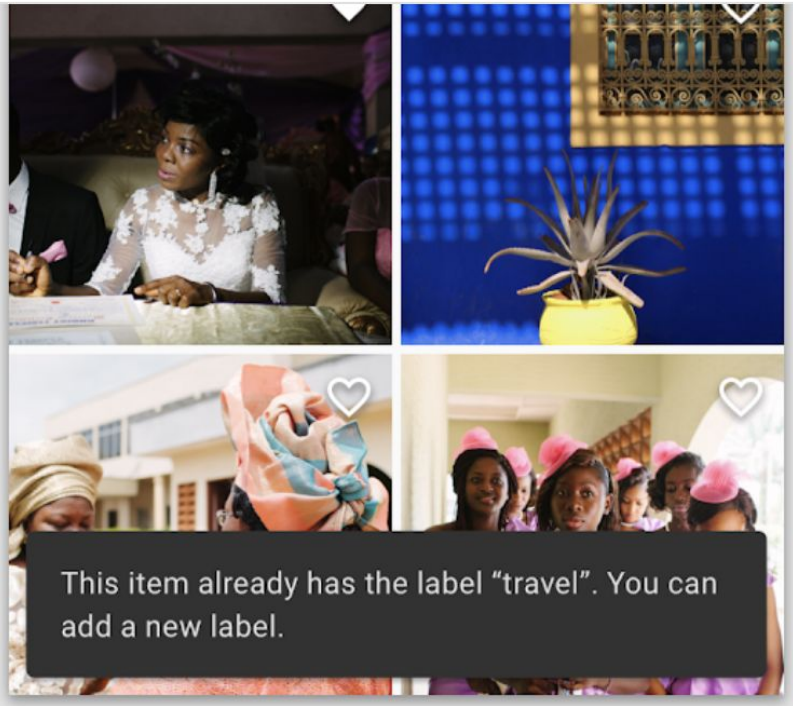
1. Text label
2. Container
3. Action (optional)

Text label

Snackbars contain a text label that directly relates to the process being performed. On mobile, the text label can contain up to two lines of text.

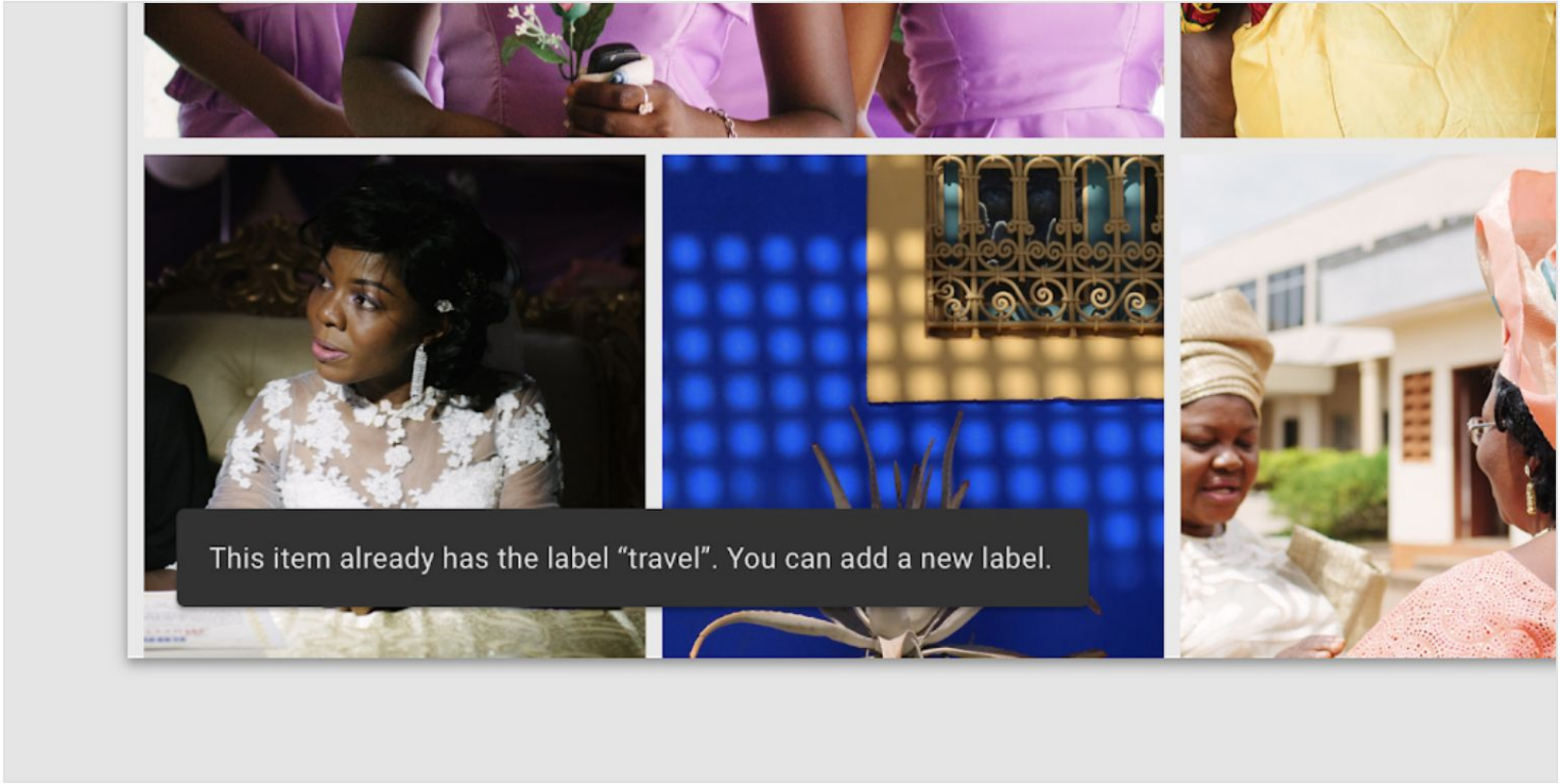


Text labels are short, clear updates on processes that have been performed.



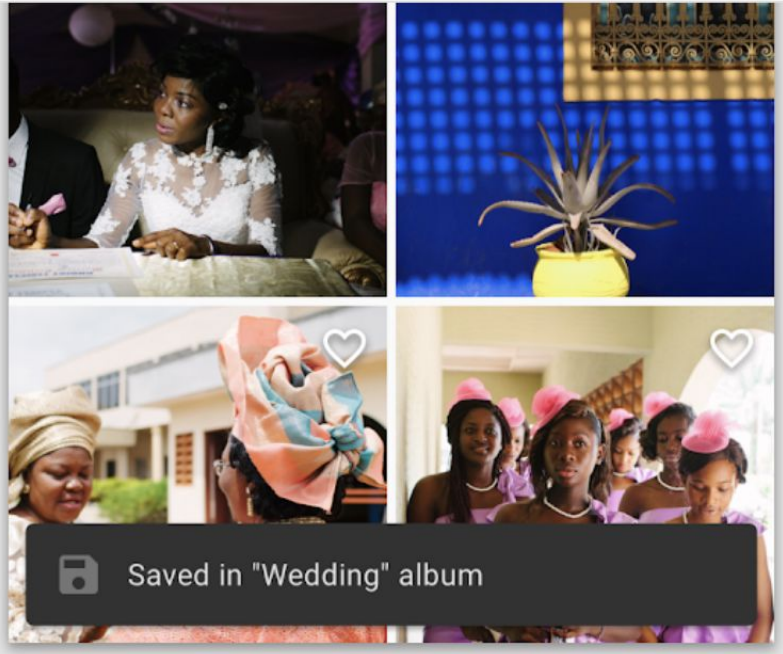
Do

On mobile, use up to two lines of text to communicate the snackbar message.



Do

In wide UIs like desktop and tablet, snackbars should have only a single line of text.

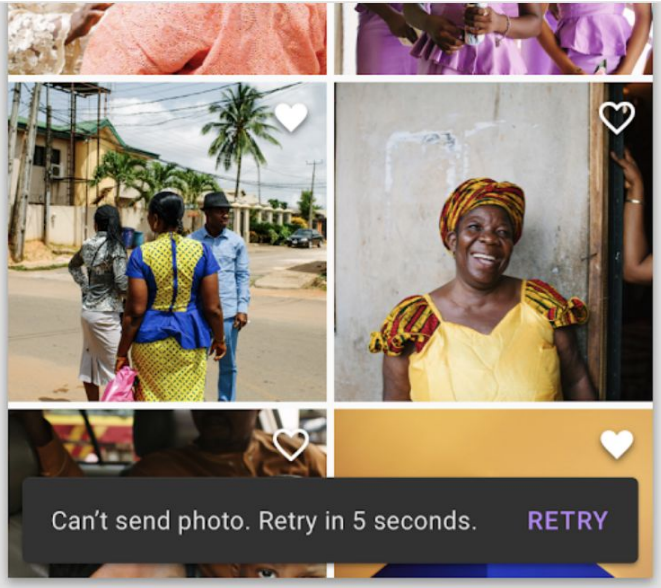


Don't

Don’t use icons in snackbars. If your message needs an icon, consider using a different component.

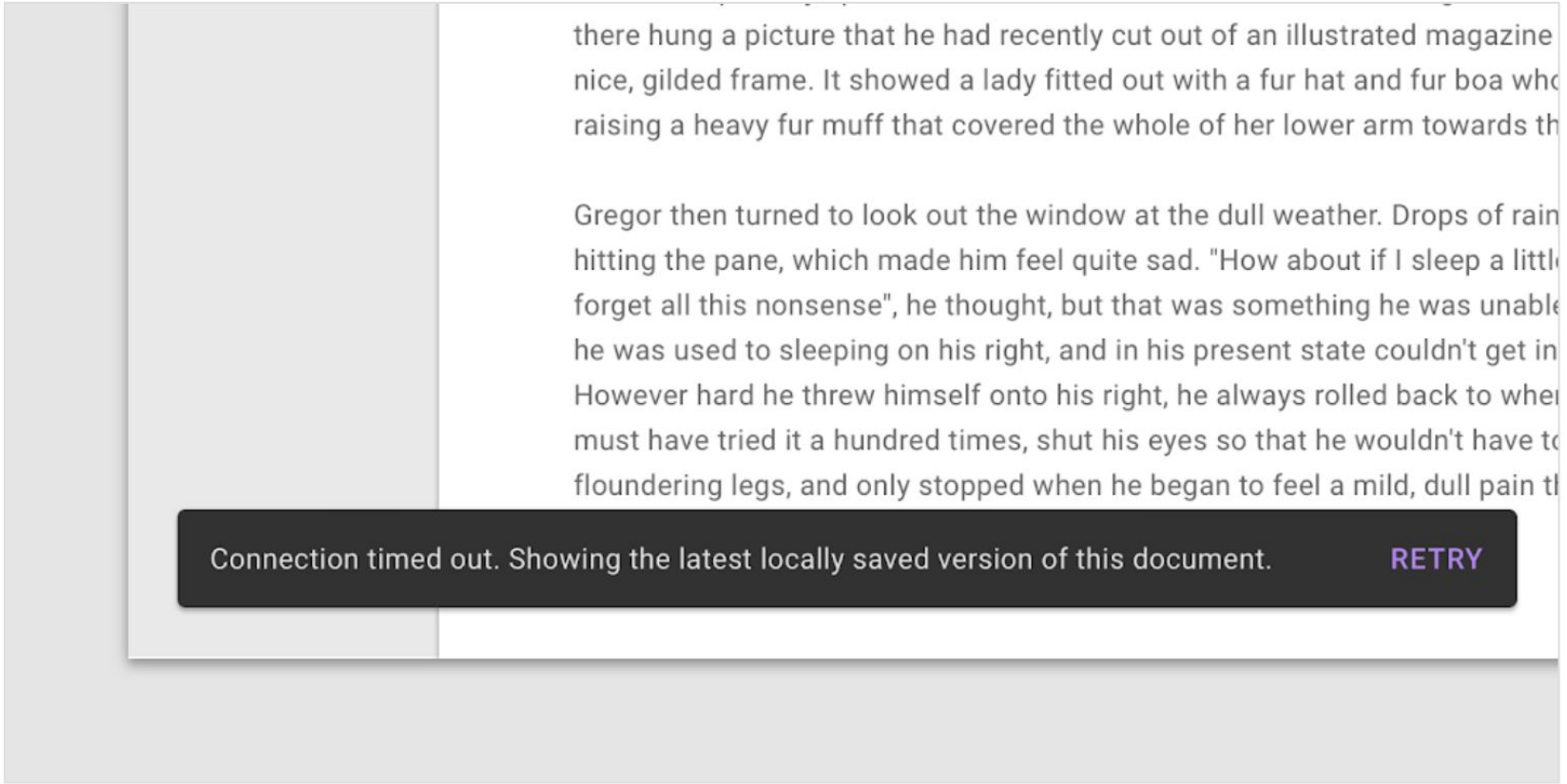
Container

Snackbars are displayed in rectangular containers with a grey background. Containers should be completely opaque, so that text labels remain legible.



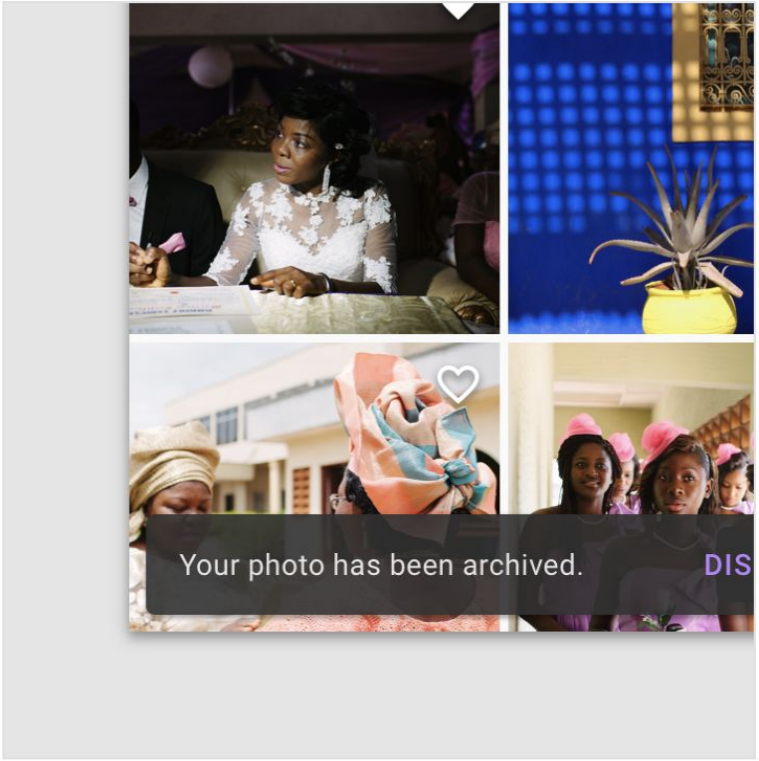
Do

Snackbar containers use a solid background color with a shadow to stand out against content.



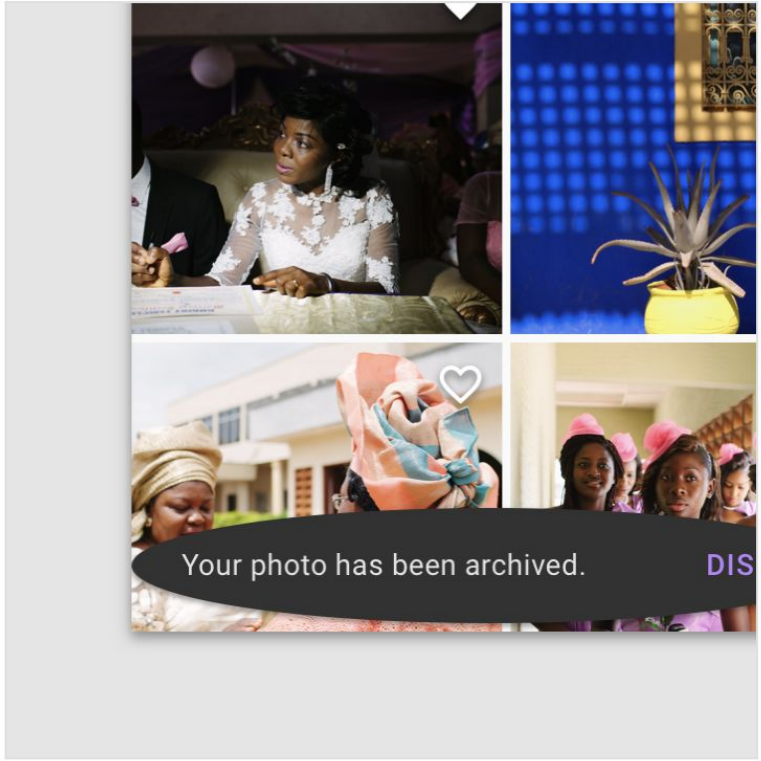
Do

In wide layouts, extend the container width to accommodate longer text labels.



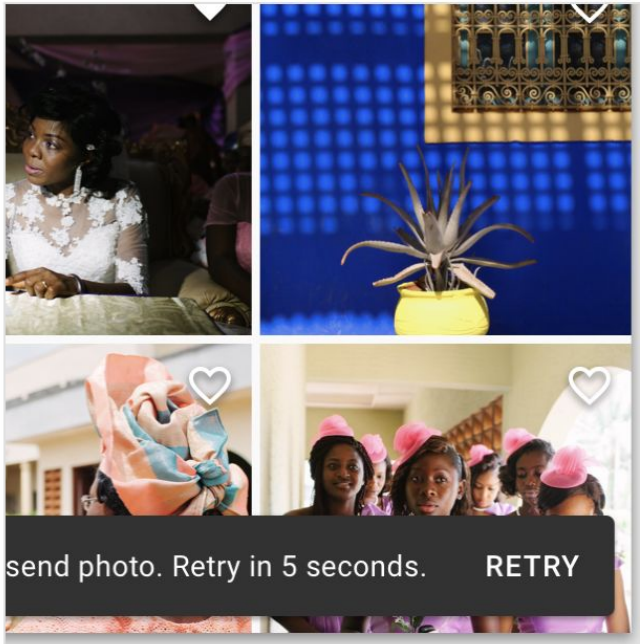
Caution

An app can apply slight transparency to the container background, as long as text remains clearly legible.



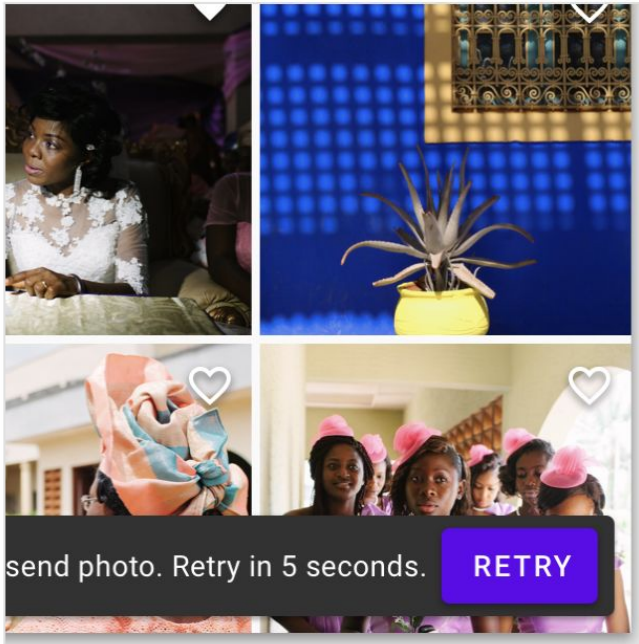
Don't

Avoid significantly altering the shape of a snackbar container.



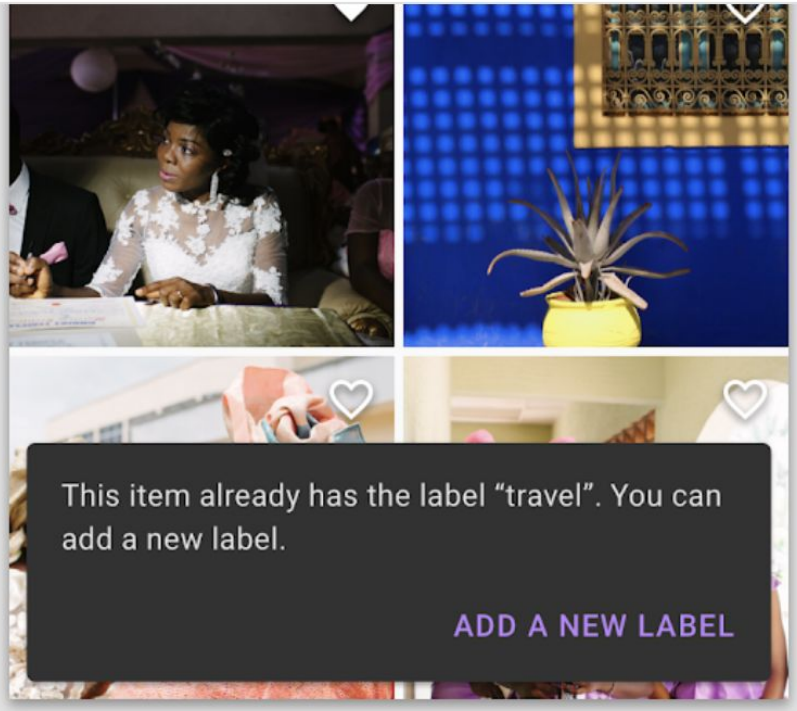
Don't

The text label shouldn't share the same color as the text button.



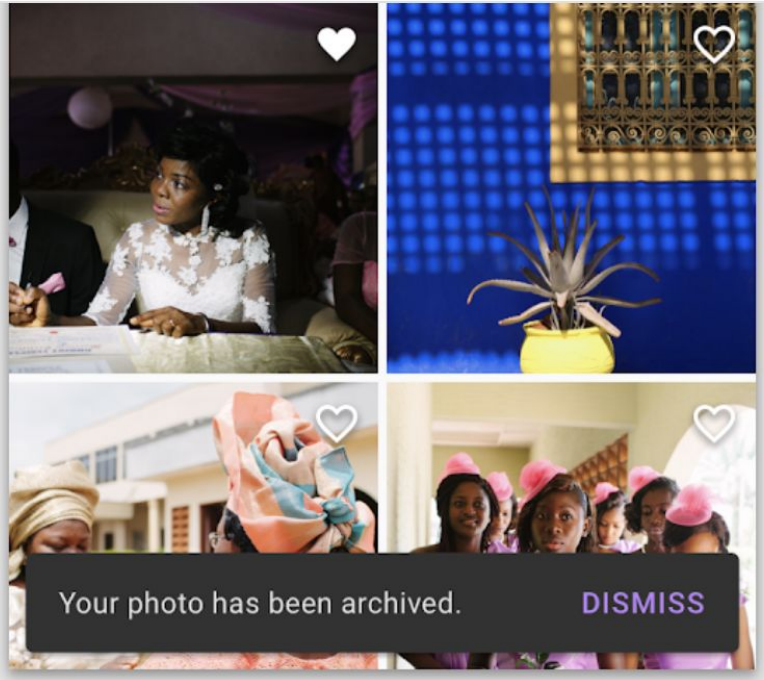
Don't

Don't use a filled or elevated button in a snackbar, as it draws too much attention.



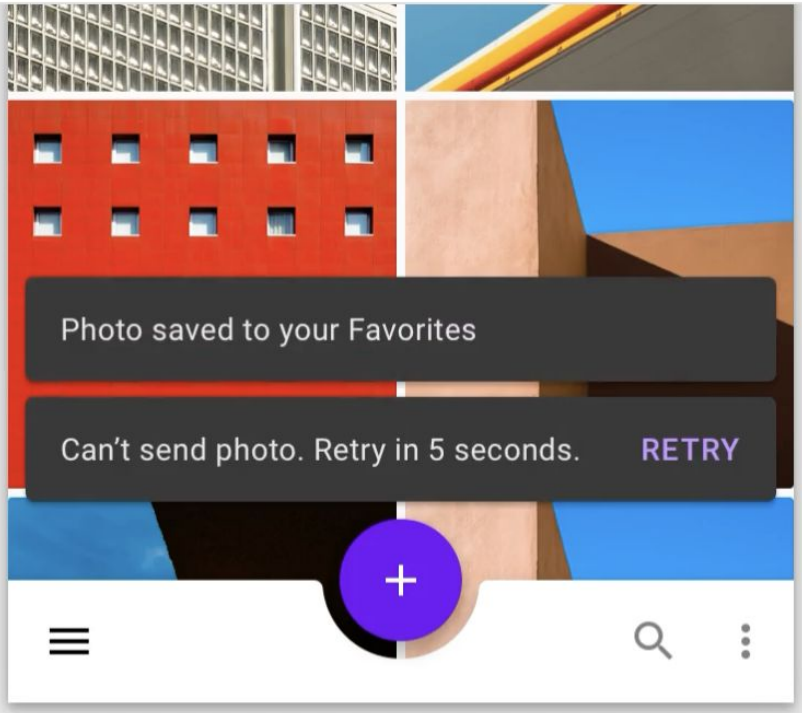
Do

If an action is long, it can be displayed on a third line.



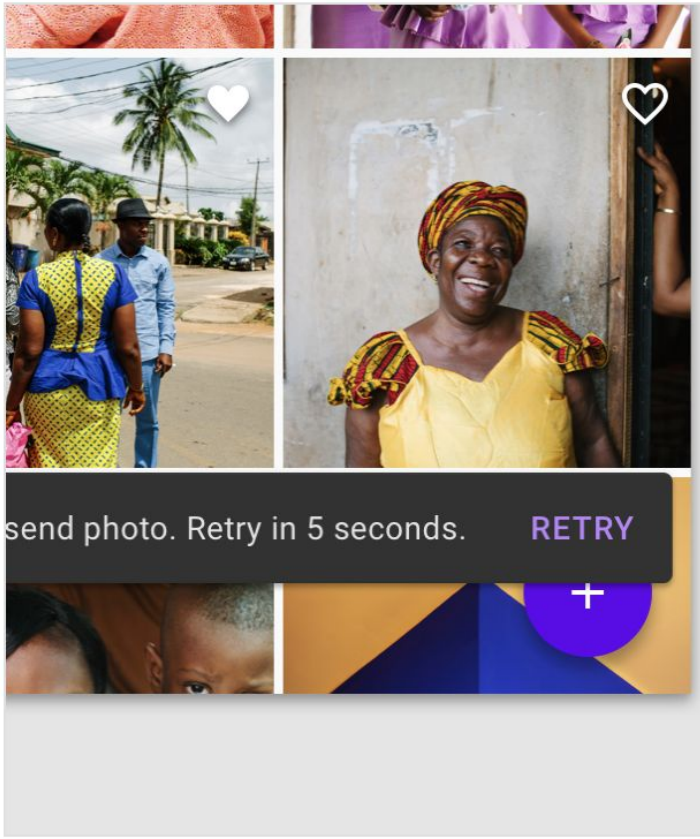
Caution

A dismiss action is unnecessary, as snackbar disappears on their own by default.



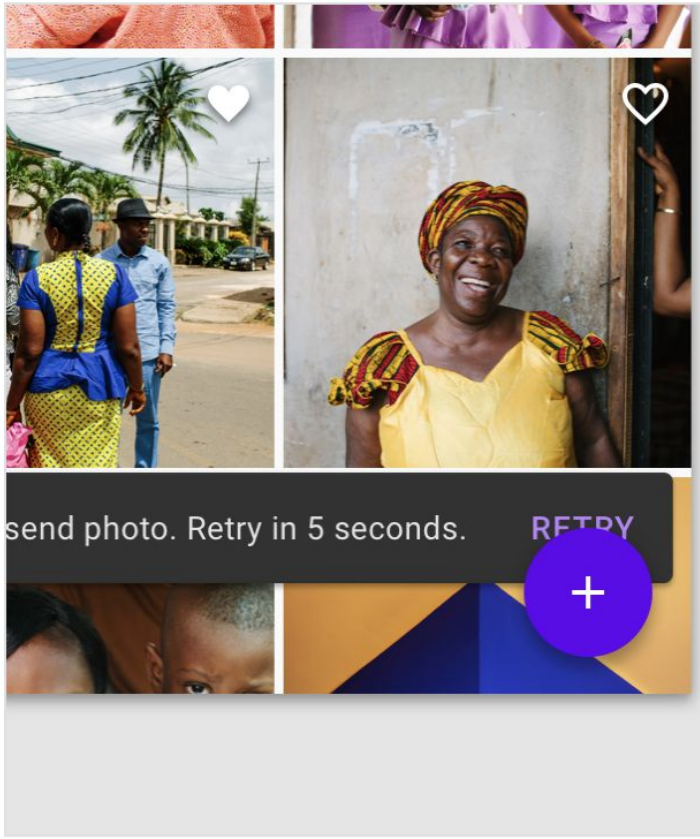
Don't

Avoid stacking snackbars on top of one another.



Don't

Don’t place a snackbar in front of a FAB.



Don't

Don’t place a snackbar behind a FAB.

Snackbar

Skapa Snackbar!

Context - Vilken klass?
Vy - Vart ska den visas?
Text - på Snackbar
Långvarighet - Längd

```
val snack = Snackbar.make(
      this,
      binding.root,
      "Button is clicked",
      Snackbar.LENGTH_LONG
      )
```

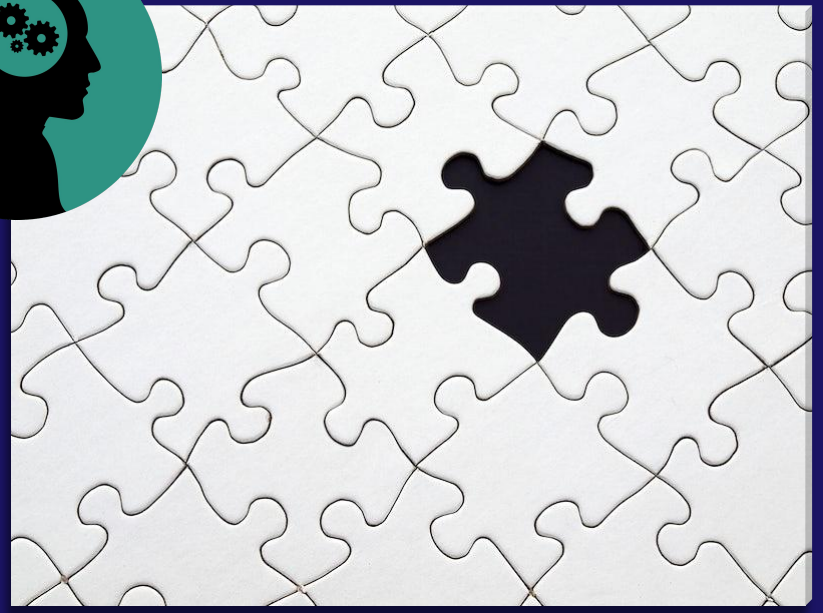
Problem

Snackbar...?

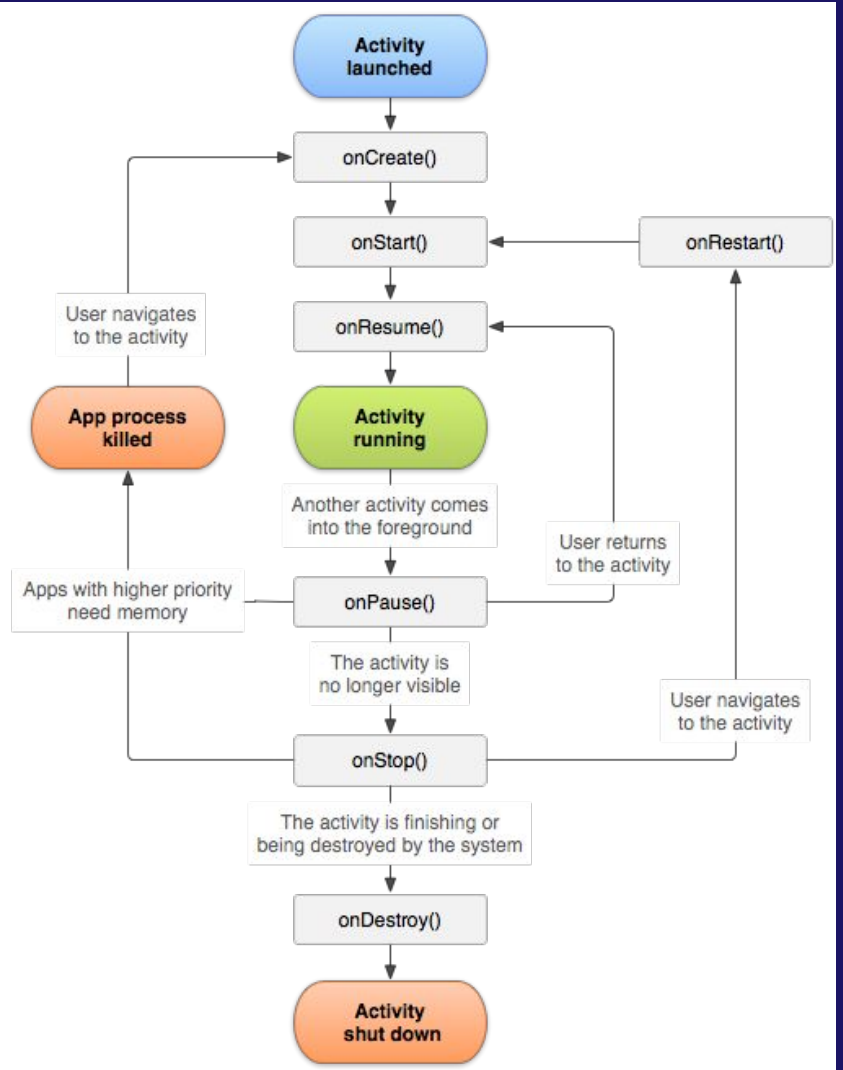
Solution

Snackbars fungerar när vi har applikationen inom ‘foreground’ då vi kan interagera med den!

Frågor?



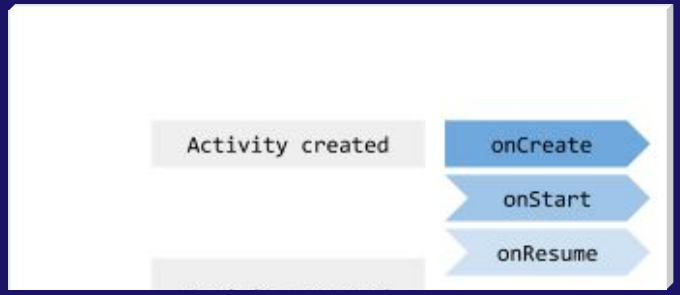

RECAP



Livscykeln är något vi diskuterar så fort vi pratar om vyer AKA ‘activities’

Varje vit rektangel symboliserar en livscykel!

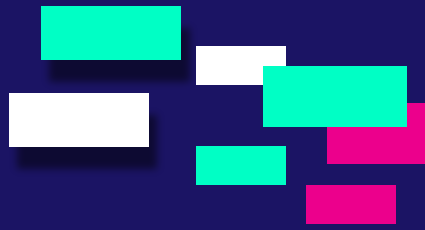
LifeCycle



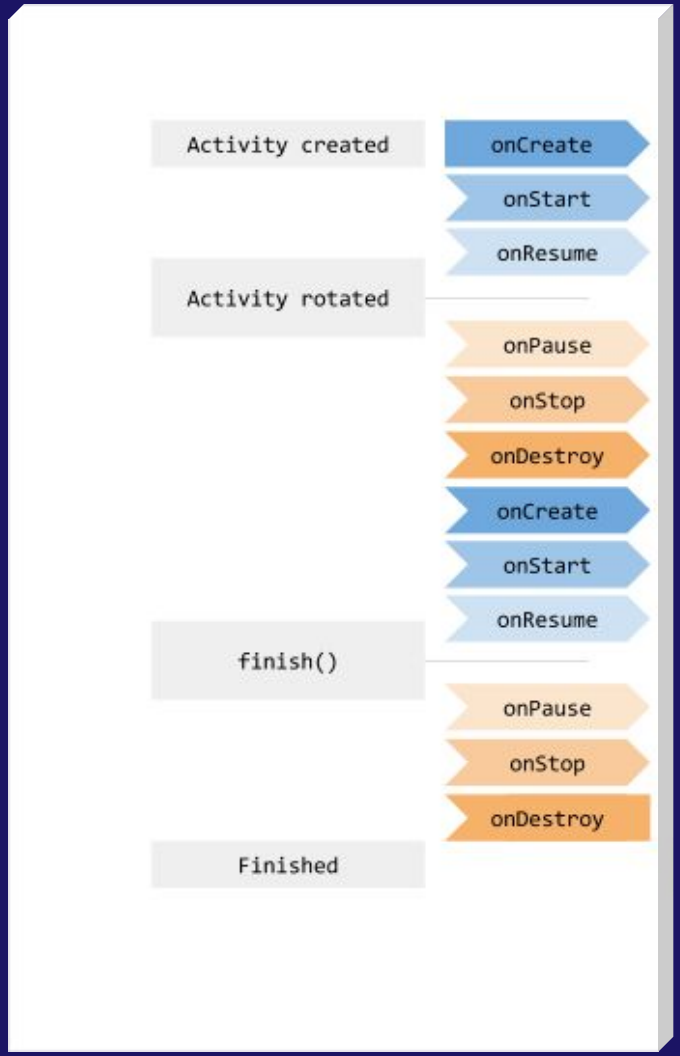

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
```

Så fort aktiviteten skapas:

```
onCreate( )
onStart( )
onResume( )
```



LifeCycle

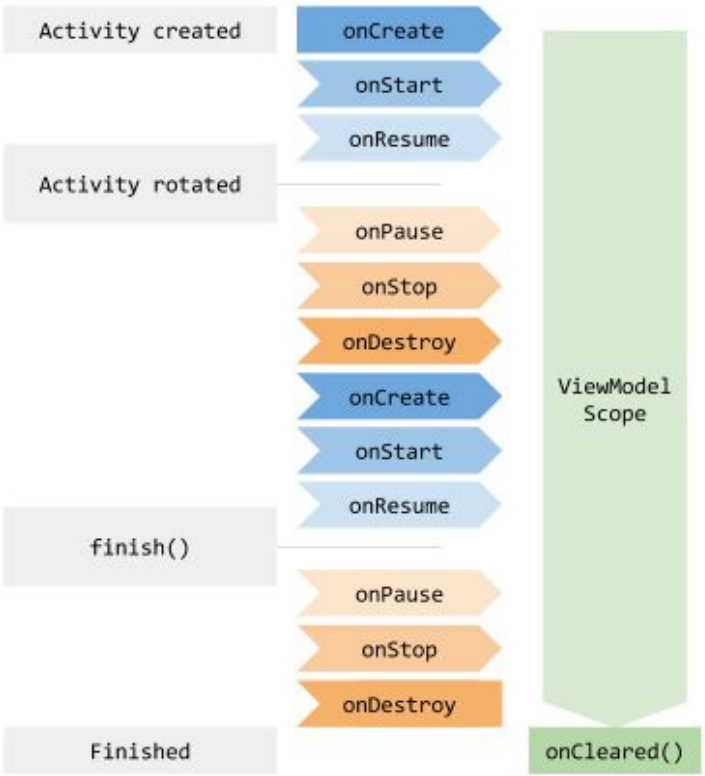


Activity rotated!

```
onPause()
onStop()
onCreate()
onStart()
onResume()
```

Activiteten återskapas!

LifeCycle



ViewModel ‘överlever’ och man kan t.om säga att den ligger ‘utanför’ livscykeln på detta sätt!

ViewModel

Extenda din klass med ‘ViewModel’

```
class Counter : ViewModel() {

    var value: Int = 0

    fun add() {
        value++
    }

}
```

NOTERA: Jag har tagit bort min konstruktor ()

ViewModel

Finslipa kod inom Main!

```
8  class MainActivity : AppCompatActivity() {
9
10     // Initialization
11     private lateinit var counterViewModel: Counter      // ViewModel
12     private lateinit var binding: ActivityMainBinding   // ViewBinding
13
```

Skriver koden snyggare!

ViewModel

Skapa koppling med ‘ViewModelProvider’

```
// ViewModel SETUP
counterViewModel = ViewModelProvider( owner: this)[Counter().javaClass]
textViewCounter.text = counterViewModel.value.toString()

// On click
textViewCounter.setOnClickListener() { it: View!
    counterViewModel.add()
    textViewCounter.text = counterViewModel.value.toString()
}
```

03

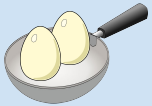
ViewModel #2


Asynchronous
VS
Synchronous

Synchronous

1

Brew Coffee

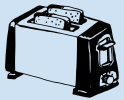


2

Fry Eggs

3

Fry Bacon



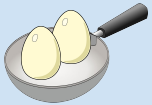
4

Toast Bread

Elapsed Time:
35 min SLOW

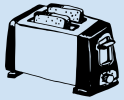
Asynchronous

1 Brew Coffee



2 Fry Eggs

3 Fry Bacon



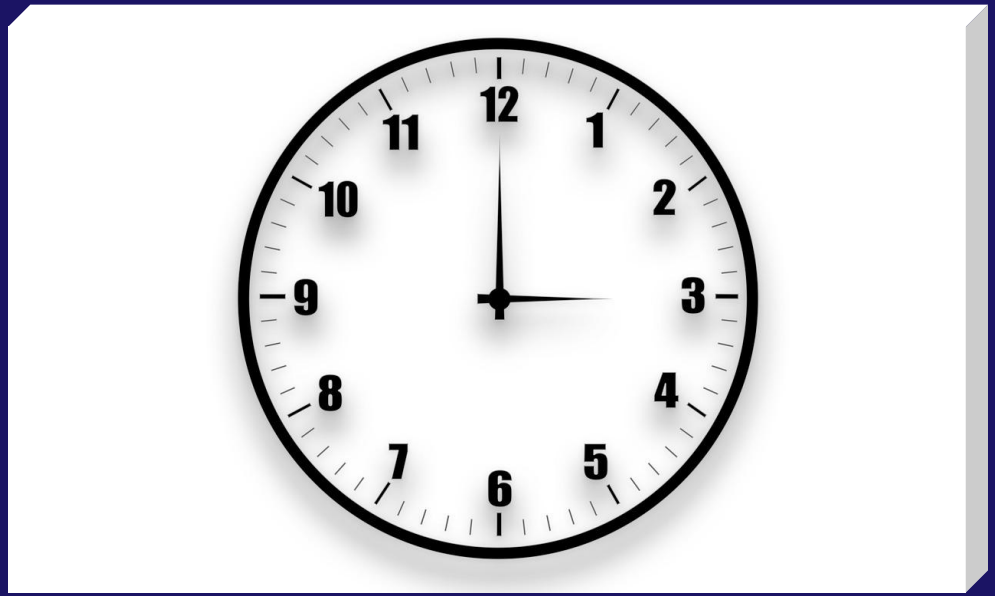
4 Toast Bread

Eggs: DONE

Bacon: DONE

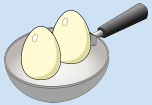
Elapsed Time:
20 min FAST

Asynchronous
Clear Winner…?



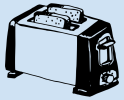
Asynchronous

1 Brew Coffee



2 Fry Eggs

3 Fry Bacon



4 Toast Bread

Eggs: DONE

Bacon: DONE

Eggs: BURNT

Bacon: BURNT

Elapsed Time:
20 min FAST

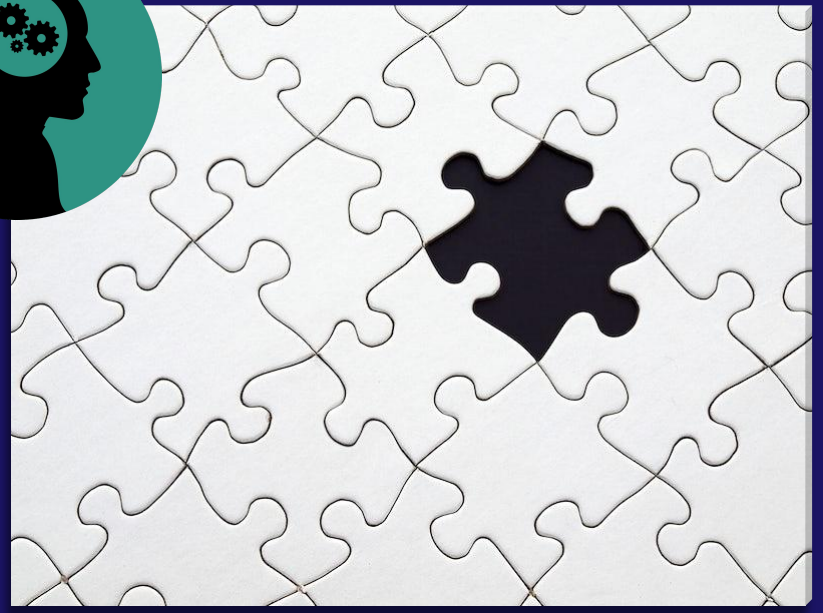
Problem

Vad är skillnaderna på Asynk/Synkron programmering?

Solution

Async sker parallelt, medan synkront sker ‘steg för steg’ och inväntar svar tills dessa blir klar.

Frågor?




ViewModel #2

INSTALL DEPENDENCY

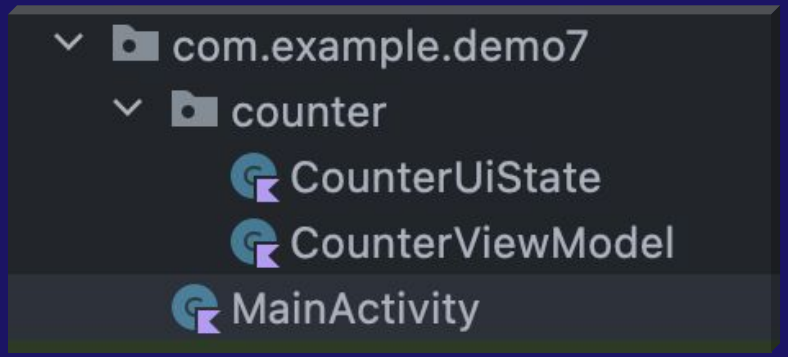
```
// Kotlin
implementation ("androidx.activity:activity-ktx:1.6.1" )
```

```
dependencies {


    // Kotlin
    implementation("androidx.activity:activity-ktx:1.6.1")

    implementation 'androidx.core:core-ktx:1.7.0'
    implementation 'androidx.appcompat:appcompat:1.6.1'
    implementation 'com.google.android.material:material:1.8.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.5'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'
}
```

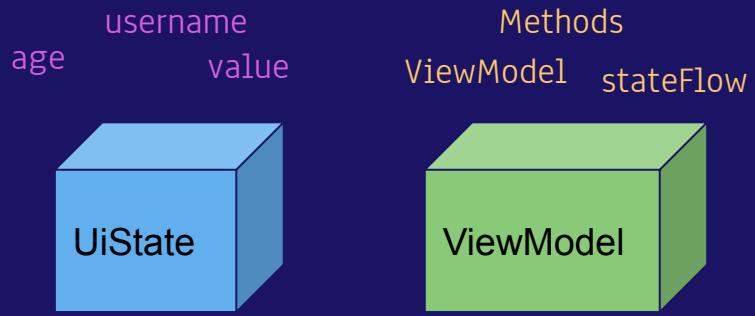
ViewModel #2



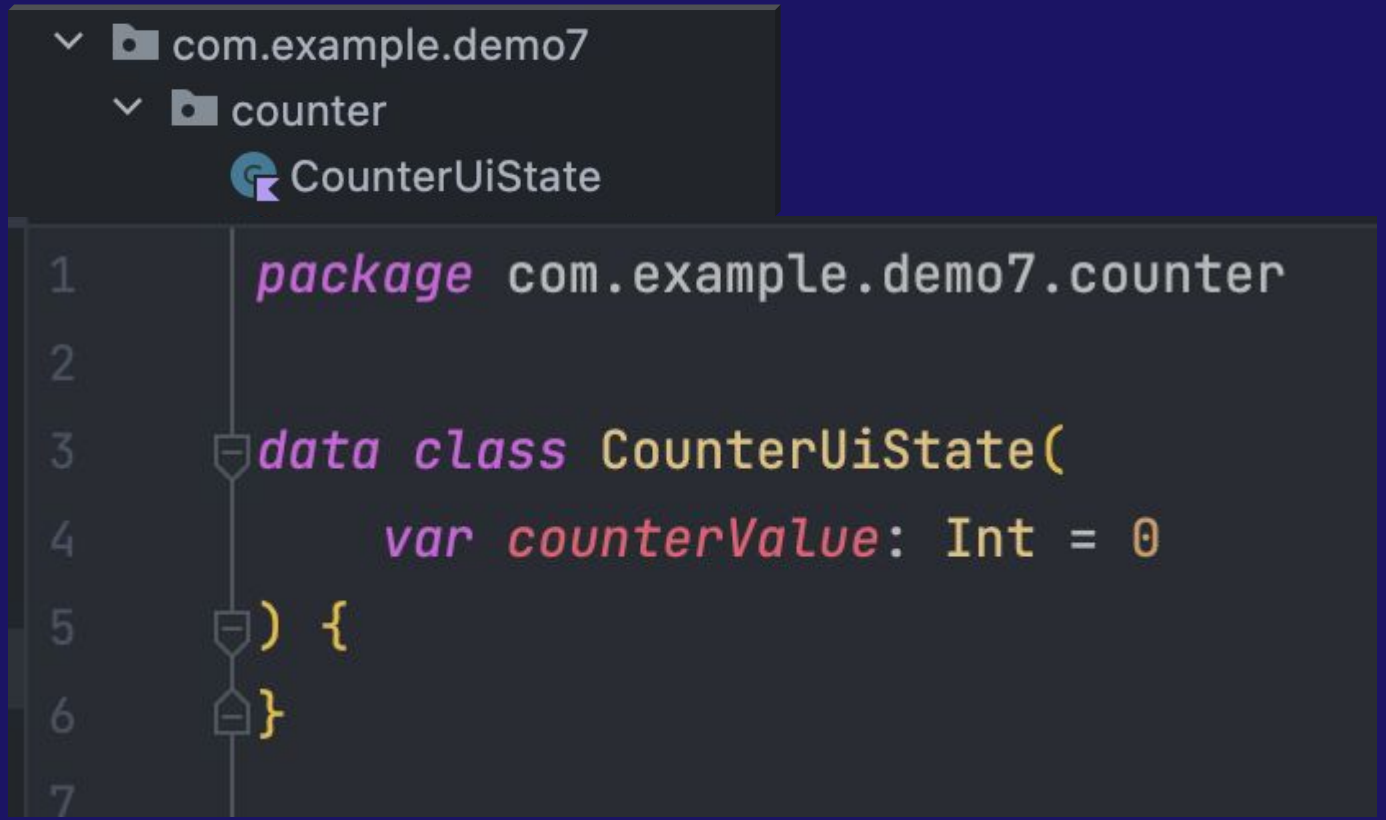
Vi kommer arbeta med ett ‘State’
och en ‘ViewModel’

Se ett ‘state’ som en useState hook!

ViewModel innehåller funktionalitet och ViewModel implementering (boilerplate kod)



ViewModel #2


com.example.demo7
counter
CounterUiState
1 package com.example.demo7.counter
2
3 data class CounterUiState(
4 var counterValue: Int = 0
5) {
6 }
7

ViewModel #2

CounterViewModel

```
class CounterViewModel: ViewModel() {

    // Expose screen UI state
    private val _uiState = MutableStateFlow(CounterUiState())
    val uiState: StateFlow<CounterUiState> = _uiState.asStateFlow()

}
```

Setup är viktigt...
MutableStateFlow innebär att vi kan förändra värdet!

ViewModel #2

CounterViewModel

```
class CounterViewModel: ViewModel() {

    // Expose screen UI state
    private val _uiState = MutableStateFlow(CounterUiState())
    val uiState: StateFlow<CounterUiState> = _uiState.asStateFlow()

    fun add() {

        _uiState.update {
            state -> state.copy(
                counterValue = state.counterValue + 1
            )
        }
    }
}
```

ViewModel #2

MainActivity

```
val viewModel: CounterViewModel by viewModels()

val example = findViewById<TextView>(R.id.textView_counter)
```

ViewModel #2

MainActivity

```
    val viewModel: CounterViewModel by viewModels()

    val example = findViewById<TextView>(R.id.textView_counter)

    lifecycleScope.launch {
        repeatOnLifecycle(Lifecycle.State.STARTED) {
            viewModel.uiState.collect() {

                // Update UI elements
                example.text = viewModel.uiState.value.counterValue.toString()
            }
        }
    }


}
```

ViewModel #2

MainActivity

```
    val viewModel: CounterViewModel by viewModels()

    val example = findViewById<TextView>(R.id.textView_counter)

    lifecycleScope.launch {
        repeatOnLifecycle(Lifecycle.State.STARTED) {
            viewModel.uiState.collect() {

                // Update UI elements
                example.text = viewModel.uiState.value.counterValue.toString()
            }
        }
    }

    example.setOnClickListener {
        viewModel.add()
    }
}
```

ViewModel #2

Resource: https://developer.android.com/topic/libraries/architecture/viewmodel

04

Uppgifter & Eget Arbete

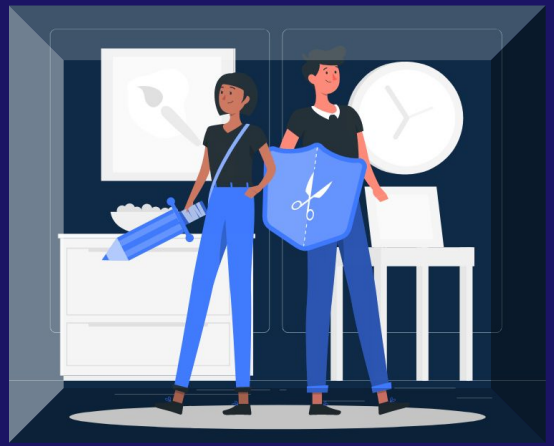
Uppgifter

Välkommen till första uppgiften!

Uppgifterna är till för att testa dina färdigheter och kunskaper för att både öva och repetera på det vi har arbetat med under föreläsningarna.

Dessa är **INTE** obligatoriska.

Men är starkt rekommenderat att arbeta med.



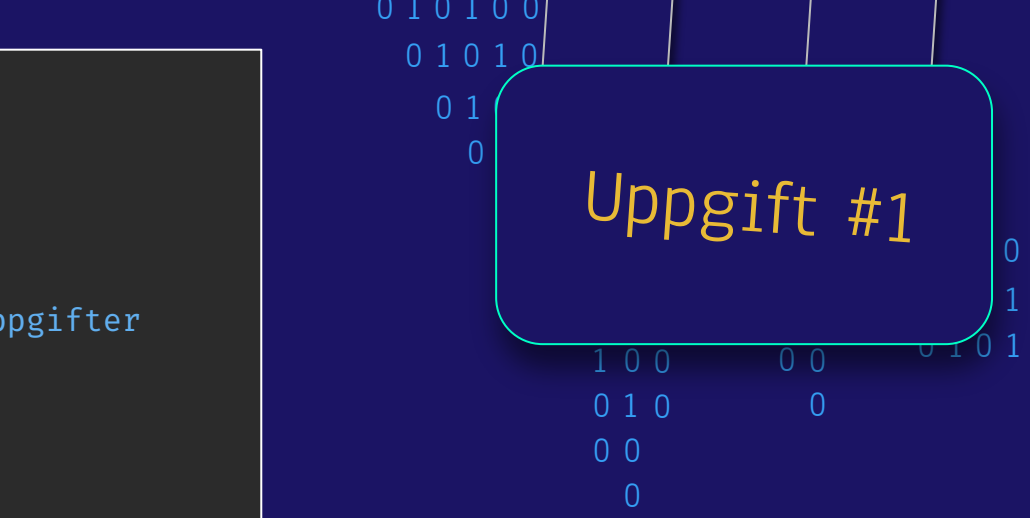
MINNS DU?

```
// Vad är skillnaden på en Toast och en
Snackbar?


// Varför arbetar vi med ‘ViewModels’?


// Varför behövs ett ‘uiState’ ?
```

```
1   // -Uppgift #1- //
2
3   /* INSTRUCTIONS
4
5       Skapa ett nytt projekt!
6
7       Döp projektet till: Lektion_7_uppgifter
8
9
10      Skapa en Toast!
11
12      Toast.makeText()
13  */
14
15  // HINT & Examples
16  hint(" Slide #7 ")
17
18
19
20
21
22
23
```



Kom igång enkelt med uppgift #1

```
1   // –Uppgift #2– //
2
3   /* INSTRUCTIONS
4
5       Skapa en enkel ‘Snackbar’
6
7       Glöm inte att den tar in 4 parametrar!
8         +   Context
9         +   View
10        +   Text
11        +   Längd
12
13  */
14
15  // HINT & Examples
16  hint(" Slide #15
17
18  Snackbar.make() ← skapar en Snackbar! ")
19
20
21
22
23
```

Glöm inte att du kan tilldela dessa till variabler!

```
1   // –Uppgift #3– //
2   The Toughest Nut
3   /* INSTRUCTIONS
4
5       Skapa en userUiState klass
6       Skapa en userViewModel klass
7
8       UiState ska bara tillhandahålla
9       en konstruktor med variabler.
10      Ge användaren en variabel för: 'points'
11
12      Försök nu att utöka points från MainActivity
13
14  */
15
16  // HINT & Examples
17  hint(" Slide #52
18
19  Points kommer bli det värdet som skall
20  manipuleras.
21
22  ViewModel är det svåra steget, följ sliden för
23  instruktioner: #55
     ")
```

ViewModel är tungt...

Därför är det bra att lära sig uppsättningen!

THANKS !

Do you have any questions?
kristoffer.johansson@sti.se

sti.learning.nu/

You can also contact me VIA Teams (quicker response)
Du kan också kontakta mig VIA Teams (Snabbare svar)