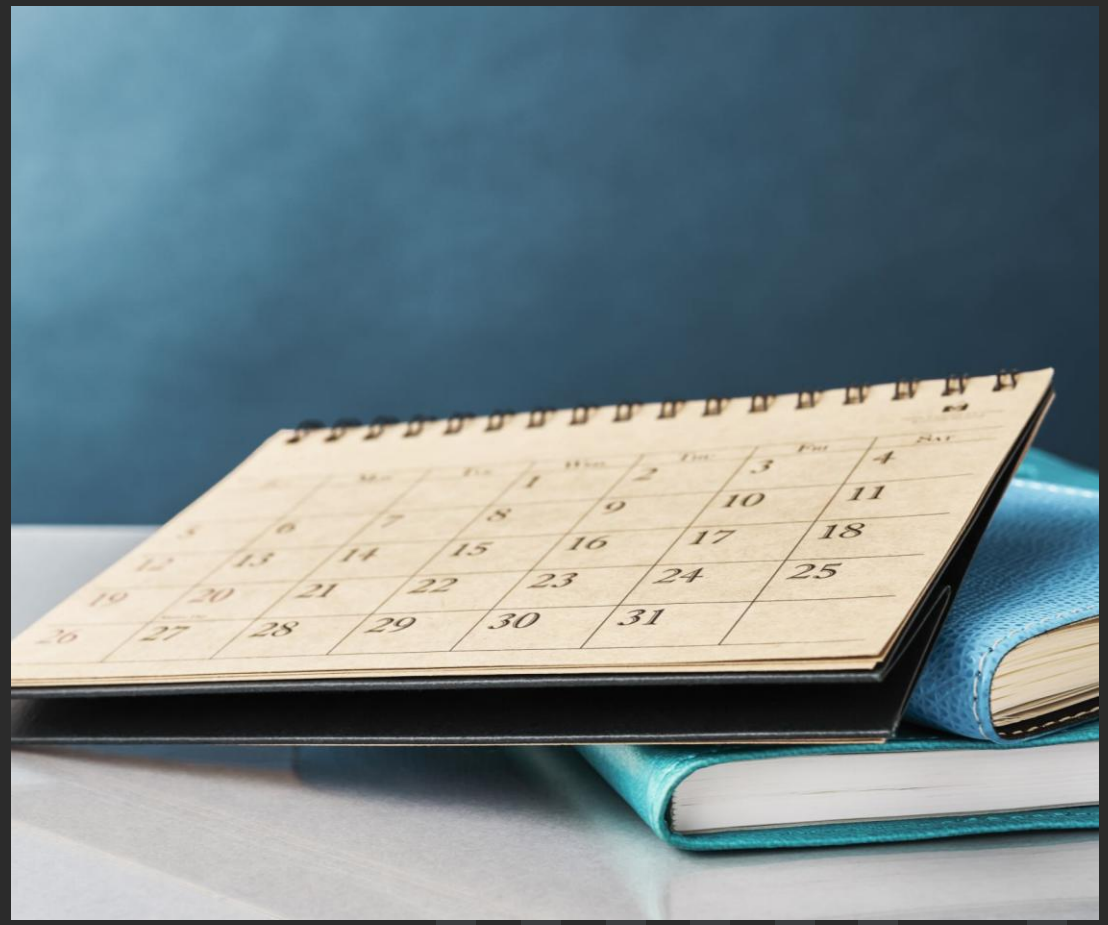
COLLÈGE DE
BOIS-DE-BOULOGNE

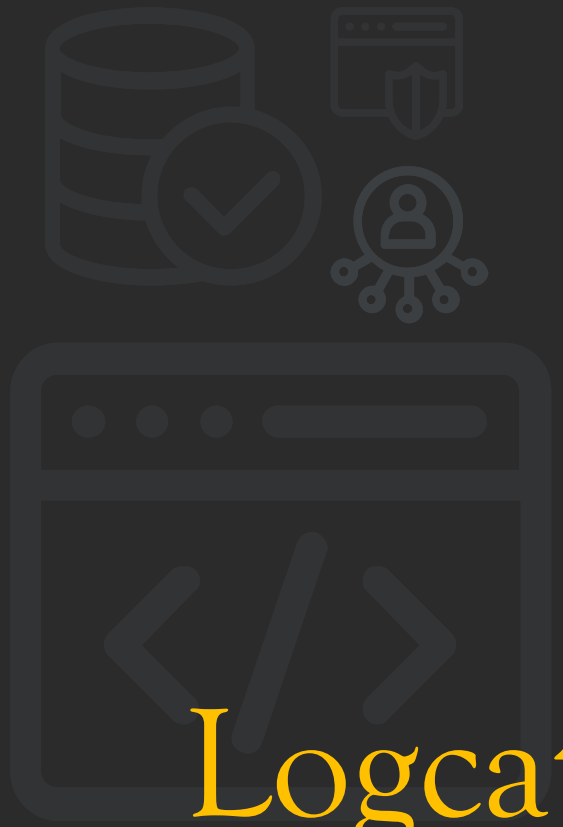
420-5GM-BB -Cours #5 Événements

Pierre Prades et Mathieu Brodeur-Béliveau

Agenda de la séance

- Présences
- Logcat
- Toast
- Gestion événements simples
- Gestion d'événements complexes
- Débogage





Logcat

Logcat

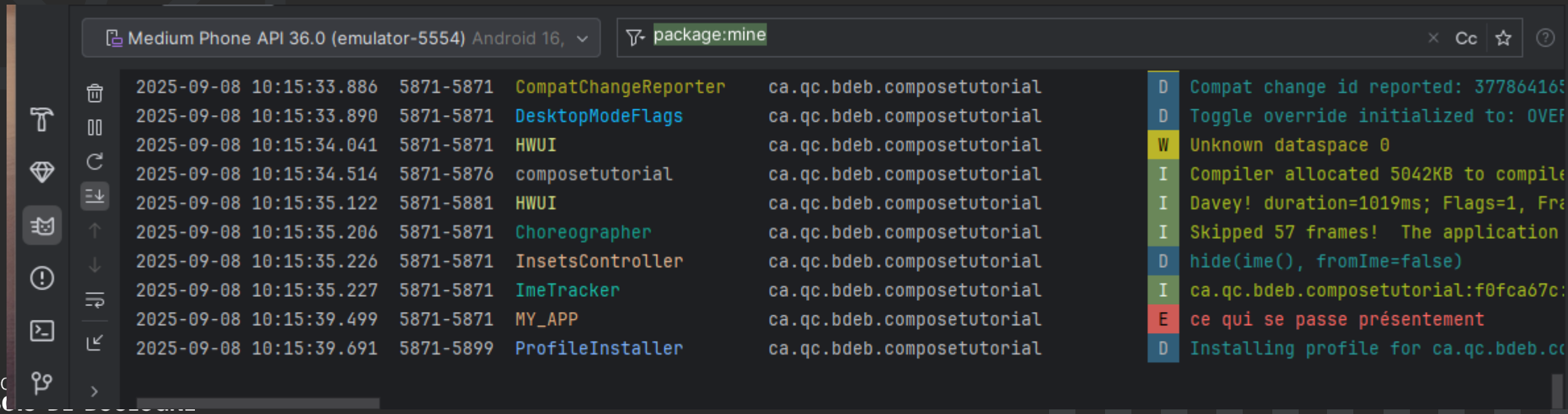
- Il est possible de « logger » des messages pendant l'exécution de notre app.
  - Log.e(String, String) (error)
  - Log.w(String, String) (warning)
  - Log.i(String, String) (information)
  - Log.d(String, String) (debug)
  - Log.v(String, String) (verbose)
  - Log.wtf(String, String) (what a terrible failure)
- S'assurer de ne jamais compiler avec des Log.v, sauf évidemment pendant le développement.
- Les Log.d devraient être ignorés en mode Release.
- Dans la fenêtre Logcat, on peut filtrer quels messages seront affichés

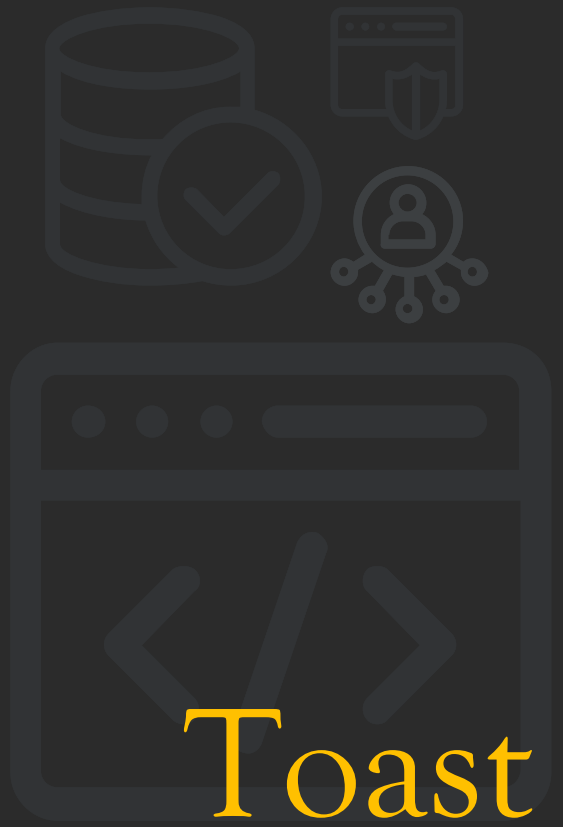
Logcat

- Par convention on déclare une constante TAG que l'on utilise comme premier argument:

```
val TAG = "MY_APP"
Button(onClick = {
   Log.e(TAG, "ce qui se passe présentement")
}
```

- Dans la fenêtre Logcat:



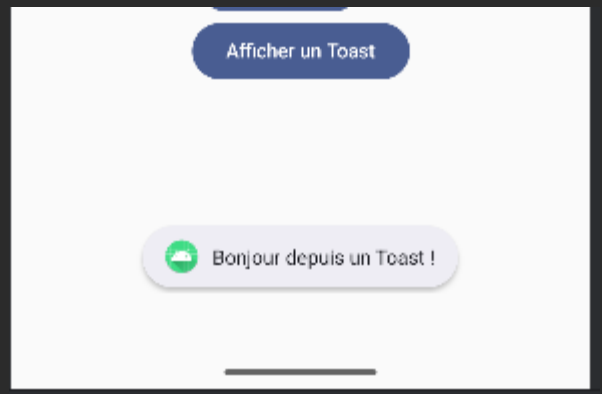


Toast

Qu'est-ce qu'un Toast ?

- Petit message temporaire qui apparaît en bas de l'écran.
- Sert à informer l'utilisateur rapidement sans interrompre son interaction.
- Exemple d'utilisation : confirmation après une action (ex. sauvegarde réussie).

⚠️ Les Toasts ne sont pas destinés à des messages critiques (préférer Snackbar ou Dialog pour ça).



- Par exemple, après l'envoi d'un courriel, vous pouvez utiliser un toast pour indiquer que l'opération s'est produite.
- Par défaut, le message apparait en bas, centré horizontalement.

Utilisation de Toast

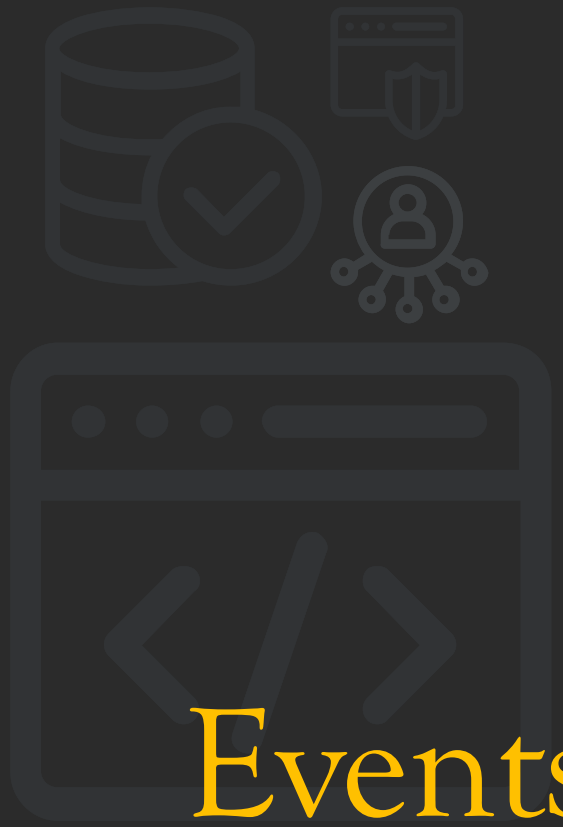
```
@Composable
fun ExempleToast() {
    val context = LocalContext.current

    Button(onClick = {
        Toast.makeText(context, "Bonjour depuis un Toast !", Toast.LENGTH_SHORT).show()
    }) {
        Text("Afficher un Toast")
    }
}
```

- Explications :
  - *LocalContext*.current donne le Context de l'Activité.
  - Toast.makeText(...) crée le toast.
  - .show() l'affiche immédiatement.

Personnalisation de Toast

- La durée peut être soit Toast.*LENGTH_SHORT* ou Toast.*LENGTH_LONG*
- Il est possible de positionner notre Toast en utilisant .setGravity(gravity:Int, xOffset:Int, yOffset:Int)

```
val toast = Toast.makeText(context, "Message personnalisé", Toast.LENGTH_LONG)
toast.setGravity(Gravity.CENTER, 0, 0) // Centré à l'écran
toast.show()
```



Events

Créer un bouton avec `Button`

- Syntaxe de base :

```
Button(onClick = { /* action à exécuter */ }) {
    Text("Clique-moi")
}
```

- Explication :
  - Button : composable qui représente un bouton.
  - onClick : lambda appelée lorsqu'on clique sur le bouton.
  - Text("...") : contenu du bouton (on peut aussi mettre une icône, etc.).

Exemple simple : afficher un message dans la console

```
@Composable
fun MonBouton() {
    Button(onClick = {
        println("Le bouton a été cliqué !")
    }) {
        Text("Appuie ici")
    }
}
```

- Ce que fait ce code:
  - Quand l'utilisateur clique (event onClick), le message est affiché dans la console (Logcat).

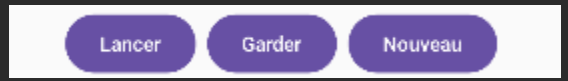
Gérer l'état avec remember et *mutableStateOf*

```
@Composable
fun CompteurClics() {
    var nombreClics by remember { mutableStateOf(0) }

    Column {
        Text("Nombre de clics : $nombreClics")
        Button(onClick = {
            nombreClics++
        }) {
            Text("Clique-moi")
        }
    }
}
```

- Explication :
  - var nombreClics by remember { mutableStateOf(0) } : crée une variable observable (patron observateur).
  - À chaque clic, nombreClics est incrémenté.
  - Jetpack Compose redessine automatiquement le composant Text quand la valeur change.

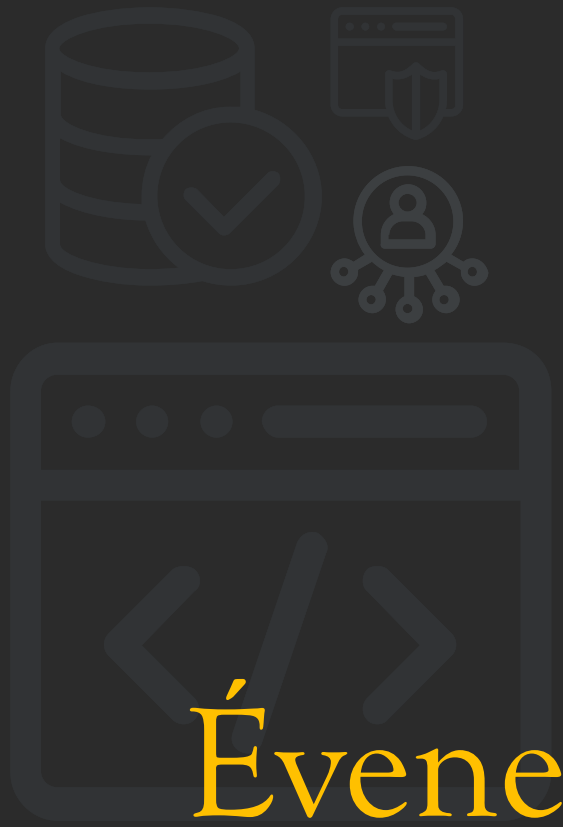
Exercices

- **Exercice 1 : Bouton simple**
  - Créer un bouton qui affiche "Bonjour!" dans Logcat lorsqu'on clique dessus.
- **Exercice 2 : Compteur**
  - Créer un bouton qui incrémente un nombre affiché à l'écran.
- **Exercice 3 : Changement d'état**
  - Créer deux boutons : un pour augmenter un compteur, l'autre pour le réinitialiser.
- **Exercice 4 : Labo**
  - Compléter le labo : https://developer.android.com/codelabs/basic-android-kotlin-compose-build-a-dice-roller-app?hl=fr
  - Modifier l'interface pour ajouter 2 boutons et l'information des joueurs et du score:



Règles du jeu de Dé

- Le joueur lance le dé en cliquant sur le bouton « ROULER »
- S'il lance un 1, il perd les points du tour courant et perd aussi son tour.
- S'il lance autre chose, il accumule les points
  - Il peut soit rouler une nouvelle fois et le pointage du tour augmente dépendant du résultat.
  - Ou il peut cliquer sur le bouton « GARDER » et les point du tour courant s'additionne au total de ses points. S'il clique sur « GARDER », c'est la fin de son tour.
- Le premier qui arrive à 25 gagne.



Évenements complexes

Pression longue (Long Press)

- Utilisation de Modifier.*pointerInput* + detectTapGestures

```
Box(
    modifier = Modifier
        .fillMaxSize()
        .pointerInput(Unit) {
            detectTapGestures(
                onLongPress = { offset ->
                    println("Pression longue à $offset")
                }
            )
        }
        .background(Color.Gray)
)
```

- Cas d'utilisation : afficher un menu contextuel, sélectionner un élément.

Tap simple et double tap

```
Modifier.pointerInput(Unit) {
    detectTapGestures(
        onTap = { println("Tap simple") },
        onDoubleTap = { println("Double tap") }
    )
}
```

Gestes de glissement (swipe, drag)

- Exemple avec detectDragGestures

```
Modifier.pointerInput(Unit) {
    detectDragGestures { change, dragAmount ->
        change.consume()
        println("Glissement de $dragAmount")
    }
}
```

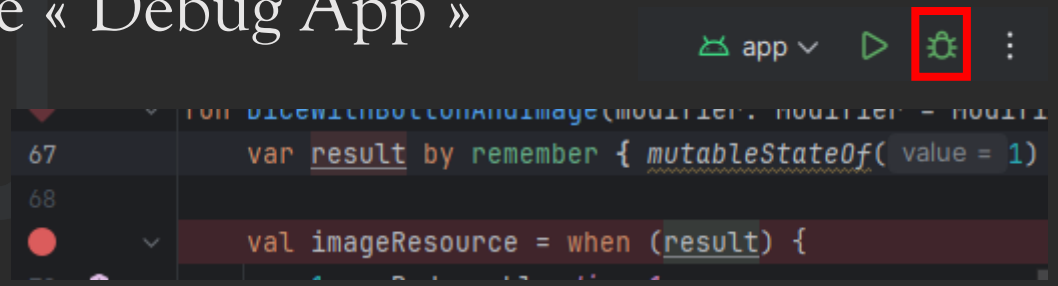
- Cas d'utilisation : réorganisation d'éléments, suppression par swipe.

Exercice

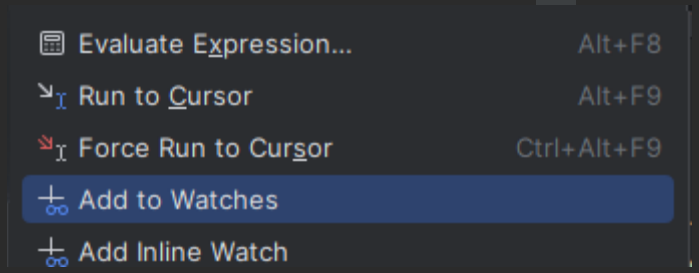
- Créer une app avec un composable image :
  - Tap simple → affiche un Toast.
  - Double tap → change l'image affichée.
  - Long press → affiche un toast long avec le message : menu contextuel.
  - Glisser → déplace l'image dans l'écran.
- Astuces:
  - Stocker les id des images dans une liste et utiliser la gestion d'état (remember et mutableStateOf) pour changer l'index de la liste.
  - On peut modifier la position d'un composable en utilisant Modifier.offset{IntOffset(x.roundToInt(), y.roundToInt())}

Débogage

- Si vous avez débogué avec un autre IDE (IntelliJ IDEA par exemple), vous serez vite à l'aise à déboguer avec Android Studio.
- Il est très facile d'ajouter un point d'arrêt (toggle breakpoint) avec Ctrl-F8 et faire « Debug App »



- Ou de regarder l'état d'une variable grâce au clic droit:



Débogage

- On peut voir la pile d'appels et l'état des variables