DAA - Développement d'applications Android

Résumé Interface graphique Android

16 November 2025

Table des matières

1 Composants de base

1.1 Visibilité

États : `VISIBLE` (défaut), `INVISIBLE` (caché, espace réservé), `GONE` (caché, pas d'espace)

```
view.visibility = View.GONE
```

> ⚠ **Warning**
>
> Attention avec GONE dans `ConstraintLayout` : utiliser **Guidelines** ou **Barriers** pour éviter les problèmes de positionnement.

1.2 Vues principales

TextView : affiche du texte. Attributs : `text`, `textSize`, `textColor`, `textStyle`

EditText : saisie de texte. `getText().toString()`, `inputType` (clavier), `TextWatcher` (changements)

Button : `setOnClickListener { }`, `setOnLongClickListener { }`

ImageView : affiche image. `scaleType` : `fitCenter`, `centerCrop`, `fitXY`

1.3 Sélection et progression

CheckBox/Switch/ToggleButton : choix binaires. `isChecked`, `setOnCheckedChangeListener { _, isChecked -> }`

RadioGroup : choix unique. `setOnCheckedChangeListener { _, checkedId -> }`

Spinner : liste déroulante. Utiliser `ArrayAdapter` + `onItemSelectedListener`

ProgressBar : indéterminée (animation) ou déterminée (valeur 0-max)

SeekBar : slider. `setOnSeekBarChangeListener` pour capturer les changements

1.4 WebView

Navigateur embarqué. Permission `INTERNET`, `settings.javaScriptEnabled = true`, `webViewClient = WebViewClient()` pour intercepter navigation.

2 UI-Thread

Règle : opérations longues → thread séparé, modifications UI → UI-Thread uniquement (sinon ANR).

```
thread {
    val result = downloadData() // Thread séparé
    runOnUiThread { textView.text = result } // UI-Thread
}
```

Alternatives : Coroutines, RxJava, WorkManager

3 Vues personnalisées

Extension : hériter vue existante. Définir attributs XML dans `attrs.xml`, utiliser `@JvmOverloads`, récupérer avec `obtainStyledAttributes()`

From scratch : hériter `View`. Méthodes : `onDraw(canvas)` (dessin), `onMeasure()` (dimensions), `onTouchEvent()` (touch), `invalidate()` (redessiner)

4 Material Design

TextInputLayout : container EditText avec label flottant, erreurs, compteur. `inputLayout.error = "message"`

Autres : `MaterialButton`, `Chip`, `BottomNavigationView`, `TabLayout`, `CardView`

5 Feedback utilisateur

Toast : message temporaire. `Toast.makeText(this, "msg", Toast.LENGTH_SHORT).show()`

Snackbar : comme Toast mais Material + action. `Snackbar.make(view, "msg", LENGTH_SHORT).setAction("Annuler") { }.show()`

Dialog : fenêtre modale. `AlertDialog.Builder(this).setTitle().setMessage().setPositiveButton().show()`

Variantes : `.setItems()` (liste), `.setSingleChoiceItems()` (radio), `.setMultiChoiceItems()` (checkbox), `.setView()` (custom)

DatePickerDialog / TimePickerDialog : sélection date/heure

DialogFragment : dialogues complexes réutilisables (survit rotation)

6 Notifications

Canal requis (SDK 26+) : `NotificationChannel(id, name, importance)`, `createNotificationChannel()`

Création :

```
val notif = NotificationCompat.Builder(this, CHANNEL_ID)
    .setSmallIcon(R.drawable.icon)
    .setContentTitle("Titre")
    .setContentText("Message")
    .setContentIntent(pendingIntent) // Action au clic
    .setAutoCancel(true)
    .addAction(icon, "Action", pendingIntent) // Max 3 actions
    .build()

NotificationManagerCompat.from(this).notify(id, notif)
```

PendingIntent : `PendingIntent.getActivity(context, 0, intent, FLAG_IMMUTABLE)`

Styles : `BigTextStyle()`, `BigPictureStyle()`, `InboxStyle()` avec `.setStyle()`

Groupes : `.setGroup(GROUP_KEY)`, `.setGroupSummary(true)`

7 ActionBar et Menu

Configuration : `setSupportActionBar(toolbar)`, `supportActionBar?.title = "Titre"`, `setDisplayHomeAsUpEnabled(true)` (bouton retour)

Menu XML (`res/menu/`) : `<item android:id android:title android:icon app:showAsAction="ifRoom|never" />`

Gestion :

```
override fun onCreateOptionsMenu(menu: Menu): Boolean {
    menuInflater.inflate(R.menu.main_menu, menu)
    return true
}

override fun onOptionsItemSelected(item: MenuItem): Boolean {
    return when(item.itemId) {
        R.id.action -> { /* ... */; true }
        android.R.id.home -> { onBackPressedDispatcher.onBackPressed(); true }
        else -> super.onOptionsItemSelected(item)
    }
}
```

Modification dynamique : `onPrepareOptionsMenu()`, `invalidateOptionsMenu()`

SearchView : `app:actionViewClass="androidx.appcompat.widget.SearchView"`, `setOnQueryTextListener`

Menu contextuel : `registerForContextMenu()`, `onCreateContextMenu()`, `onContextItemSelected()`

8 Listes

8.1 ListView (legacy, déconseillé)

Affichage vertical de liste avec Adapter.

Problèmes :

- Recyclage optionnel (performances)
- ViewHolder non standardisé
- Seulement vertical
- Animations limitées

Exemple avec ViewHolder :

```
class MyAdapter(private val items: List<String>) : BaseAdapter() {

    override fun getCount() = items.size
    override fun getItem(position: Int) = items[position]
    override fun getItemId(position: Int) = position.toLong()

    override fun getView(position: Int, convertView: View?, parent: ViewGroup): View {
        val view: View
        val holder: ViewHolder

        if (convertView == null) {
            // Première création de la vue
            view = LayoutInflater.from(parent.context)
                .inflate(R.layout.item_list, parent, false)
            holder = ViewHolder(view)
            view.tag = holder
        } else {
            // Réutilisation d'une vue existante
            view = convertView
            holder = view.tag as ViewHolder
        }

        // Bind des données
        holder.title.text = items[position]

        return view
    }

    private class ViewHolder(view: View) {
        val title: TextView = view.findViewById(R.id.title)
    }
}

// Utilisation
listView.adapter = MyAdapter(myItems)
```

8.2 RecyclerView (recommandé)

Version moderne et optimisée des listes.

Avantages :

- Recyclage obligatoire (performances garanties)
- ViewHolder standardisé
- Types d'items multiples
- Animations intégrées
- DiffUtil pour mises à jour optimales
- Layouts flexibles (Linear, Grid, Staggered)
- ItemDecoration pour les séparateurs

Dépendance : `androidx.recyclerview:recyclerview`

8.2.1 Configuration de base

```
val recyclerView = findViewById<RecyclerView>(R.id.recycler_view)

// Adapter
recyclerView.adapter = MyAdapter()

// LayoutManager (définit la disposition)
recyclerView.layoutManager = LinearLayoutManager(this) // Liste verticale
// Autres options :
// LinearLayoutManager(this, LinearLayoutManager.HORIZONTAL, false) // Horizontale
// GridLayoutManager(this, 2) // Grille 2 colonnes
// StaggeredGridLayoutManager(2, StaggeredGridLayoutManager.VERTICAL) // Grille irrégulière

// Décorateurs (séparateurs, espacements)
recyclerView.addItemDecoration(
    DividerItemDecoration(this, DividerItemDecoration.VERTICAL)
)
```

8.2.2 Adapter simple

```
class MyAdapter : RecyclerView.Adapter<MyAdapter.ViewHolder>() {

    private var items = listOf<String>()

    // Mise à jour des données
    fun updateItems(newItems: List<String>) {
        items = newItems
        notifyDataSetChanged() // Force le rafraîchissement
    }

    // Nombre d'éléments
    override fun getItemCount() = items.size

    // Création d'un ViewHolder
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
        val view = LayoutInflater.from(parent.context)
            .inflate(R.layout.item_list, parent, false)
        return ViewHolder(view)
    }

    // Bind des données
    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        holder.bind(items[position])
    }

    // ViewHolder
    inner class ViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
        private val title: TextView = itemView.findViewById(R.id.title)
        private val subtitle: TextView = itemView.findViewById(R.id.subtitle)

        fun bind(item: String) {
            title.text = item
            subtitle.text = "Item #$adapterPosition"

            // Gestion du clic
            itemView.setOnClickListener {
                // adapterPosition donne la position actuelle
                Toast.makeText(itemView.context,
                    "Cliqué sur $item",
                    Toast.LENGTH_SHORT).show()
            }
        }
    }
}
```

8.2.3 Types d'items multiples

Pour afficher différents layouts selon le type de données.

```
class MultiTypeAdapter : RecyclerView.Adapter<RecyclerView.ViewHolder>() {

    private val items = mutableListOf<Any>()
```

```
    companion object {
        const val TYPE_HEADER = 0
        const val TYPE_ITEM = 1
        const val TYPE_FOOTER = 2
    }

    override fun getItemCount() = items.size

    // Déterminer le type de chaque position
    override fun getItemViewType(position: Int): Int {
        return when (items[position]) {
            is Header -> TYPE_HEADER
            is Footer -> TYPE_FOOTER
            else -> TYPE_ITEM
        }
    }

    // Créer le bon ViewHolder selon le type
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): RecyclerView.ViewHolder {
        return when (viewType) {
            TYPE_HEADER -> HeaderViewHolder(
                LayoutInflater.from(parent.context)
                    .inflate(R.layout.item_header, parent, false)
            )
            TYPE_FOOTER -> FooterViewHolder(
                LayoutInflater.from(parent.context)
                    .inflate(R.layout.item_footer, parent, false)
            )
            else -> ItemViewHolder(
                LayoutInflater.from(parent.context)
                    .inflate(R.layout.item_normal, parent, false)
            )
        }
    }

    override fun onBindViewHolder(holder: RecyclerView.ViewHolder, position: Int) {
        when (holder) {
            is HeaderViewHolder -> holder.bind(items[position] as Header)
            is ItemViewHolder -> holder.bind(items[position] as Item)
            is FooterViewHolder -> holder.bind(items[position] as Footer)
        }
    }

    class HeaderViewHolder(view: View) : RecyclerView.ViewHolder(view) {
        fun bind(header: Header) { /* ... */ }
    }

    class ItemViewHolder(view: View) : RecyclerView.ViewHolder(view) {
        fun bind(item: Item) { /* ... */ }
    }

    class FooterViewHolder(view: View) : RecyclerView.ViewHolder(view) {
        fun bind(footer: Footer) { /* ... */ }
    }
}
```

8.2.4 DiffUtil : mises à jour optimales

Calcule automatiquement les différences entre deux listes et applique uniquement les changements nécessaires (animations incluses).

```
class MyAdapter : RecyclerView.Adapter<MyAdapter.ViewHolder>() {

    private var items = listOf<Item>()

    // Setter avec DiffUtil
    fun updateItems(newItems: List<Item>) {
        val diffCallback = ItemDiffCallback(items, newItems)
        val diffResult = DiffUtil.calculateDiff(diffCallback)

        items = newItems
```

```
        diffResult.dispatchUpdatesTo(this) // Applique les changements
    }

    // Reste de l'adapter...
}

// DiffUtil.Callback
class ItemDiffCallback(
    private val oldList: List<Item>,
    private val newList: List<Item>
) : DiffUtil.Callback() {

    override fun getOldListSize() = oldList.size
    override fun getNewListSize() = newList.size

    // Les items représentent-ils la même entité ?
    override fun areItemsTheSame(oldPos: Int, newPos: Int): Boolean {
        return oldList[oldPos].id == newList[newPos].id
    }

    // Le contenu est-il identique ?
    override fun areContentsTheSame(oldPos: Int, newPos: Int): Boolean {
        return oldList[oldPos] == newList[newPos]
    }

    // Optionnel : payload pour mises à jour partielles
    override fun getChangePayload(oldPos: Int, newPos: Int): Any? {
        val oldItem = oldList[oldPos]
        val newItem = newList[newPos]

        return if (oldItem.name != newItem.name) {
            "name_changed"
        } else null
    }
}
```

8.2.5 *ListAdapter : DiffUtil intégré*

Version simplifiée avec DiffUtil automatique.

```
class MyAdapter : ListAdapter<Item, MyAdapter.ViewHolder>(
    object : DiffUtil.ItemCallback<Item>() {
        override fun areItemsTheSame(old: Item, new: Item) = old.id == new.id
        override fun areContentsTheSame(old: Item, new: Item) = old == new
    }
) {

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
        val view = LayoutInflater.from(parent.context)
            .inflate(R.layout.item_list, parent, false)
        return ViewHolder(view)
    }

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        holder.bind(getItem(position))
    }

    class ViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
        fun bind(item: Item) { /* ... */ }
    }
}

// Utilisation
val adapter = MyAdapter()
recyclerView.adapter = adapter

// Mise à jour (DiffUtil automatique)
adapter.submitList(newItems)
```

8.2.6 *Click listeners*

Approche recommandée : callback dans le constructeur

```kotlin
class MyAdapter(
    private val onItemClick: (Item) -> Unit
) : RecyclerView.Adapter<MyAdapter.ViewHolder>() {

    // ...

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        val item = items[position]
        holder.bind(item)
        holder.itemView.setOnClickListener { onItemClick(item) }
    }
}

// Utilisation
val adapter = MyAdapter { item ->
    Toast.makeText(this, "Cliqué: ${item.name}", Toast.LENGTH_SHORT).show()
}
```

9 Composants avancés

FAB (Floating Action Button) : bouton flottant. `FloatingActionButton` / `ExtendedFloatingActionButton` (avec texte)

Animations XML (`res/anim/`) : `<alpha>`, `<translate>`, `<scale>`, `<rotate>`. Charger : `AnimationUtils.loadAnimation()`, lancer : `view.startAnimation()`

Animations code

`view.animate().alpha().translationX() .scaleX().rotation().setDuration().start()`, `ObjectAnimator`, `AnimatorSet`

Gestures : `GestureDetectorCompat` + `SimpleOnGestureListener` (onDown, onSingleTapUp, onLongPress, onScroll, onFling). Détecter swipe : comparer velocityX/Y avec threshold

Pinch-to-zoom : `ScaleGestureDetector` + `ScaleListener`, `onScale()` avec `scaleFactor`

9.1 Animations

Définies en XML dans `res/anim/` ou programmatiquement.

9.1.1 Animations XML

Créer `res/anim/fade_in.xml` :

```
<set xmlns:android="http://schemas.android.com/apk/res/android"
     android:fillAfter="true">

    <!-- Fade in -->
    <alpha
        android:fromAlpha="0.0"
        android:toAlpha="1.0"
        android:duration="300" />

    <!-- Translation -->
    <translate
        android:fromYDelta="100%"
        android:toYDelta="0%"
        android:duration="300" />

</set>
```

Types d'animations :

`<alpha>` : transparence

- `fromAlpha` / `toAlpha` : 0.0 (transparent) à 1.0 (opaque)

`<translate>` : déplacement

- `fromXDelta` / `toXDelta` : déplacement horizontal
- `fromYDelta` / `toYDelta` : déplacement vertical
- Valeurs : pixels (50), % de la vue (50%), % du parent (50%p)

`<scale>` : redimensionnement

- `fromXScale` / `toXScale` : échelle horizontale (1.0 = taille normale)
- `fromYScale` / `toYScale` : échelle verticale
- `pivotX` / `pivotY` : point de pivot (50% = centre)

`<rotate>` : rotation

- `fromDegrees` / `toDegrees` : angle de rotation
- `pivotX` / `pivotY` : point de pivot

Exemple complet :

```
<set xmlns:android="http://schemas.android.com/apk/res/android"
     android:fillAfter="true"
```

```
     android:duration="500">

    <!-- Rotation 360° -->
    <rotate
        android:fromDegrees="0"
        android:toDegrees="360"
        android:pivotX="50%"
        android:pivotY="50%" />

    <!-- Zoom in -->
    <scale
        android:fromXScale="0.5"
        android:toXScale="1.0"
        android:fromYScale="0.5"
        android:toYScale="1.0"
        android:pivotX="50%"
        android:pivotY="50%" />

    <!-- Fade in -->
    <alpha
        android:fromAlpha="0.0"
        android:toAlpha="1.0" />

</set>
```

9.1.2 Application des animations

```
// Charger et démarrer
val animation = AnimationUtils.loadAnimation(this, R.anim.fade_in)
view.startAnimation(animation)

// Listener d'événements
animation.setAnimationListener(object : Animation.AnimationListener {
    override fun onAnimationStart(animation: Animation?) {
        // Début de l'animation
    }

    override fun onAnimationEnd(animation: Animation?) {
        // Fin de l'animation
        view.visibility = View.VISIBLE
    }

    override fun onAnimationRepeat(animation: Animation?) {}
})

// Interpolateurs (accélération)
animation.interpolator = AccelerateInterpolator() // Accélère
animation.interpolator = DecelerateInterpolator() // Décélère
animation.interpolator = BounceInterpolator() // Rebond
```

9.1.3 Animations programmatiques

```
// Fade in
view.animate()
    .alpha(1f)
    .setDuration(300)
    .setListener(object : AnimatorListenerAdapter() {
        override fun onAnimationEnd(animation: Animator) {
            // Fin
        }
    })
    .start()

// Translation + rotation
view.animate()
    .translationY(100f)
    .rotation(360f)
    .setDuration(500)
    .start()

// Chaîner plusieurs animations
```

```
val animator1 = ObjectAnimator.ofFloat(view, "alpha", 0f, 1f)
val animator2 = ObjectAnimator.ofFloat(view, "translationY", 0f, 100f)

AnimatorSet().apply {
    playSequentially(animator1, animator2) // L'une après l'autre
    // ou playTogether(animator1, animator2) // En parallèle
    duration = 300
    start()
}
```

9.2 Gestures Detection

Détection de gestes complexes (swipe, fling, double-tap, etc.).

```
class MyActivity : AppCompatActivity() {

    private lateinit var gestureDetector: GestureDetectorCompat

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        gestureDetector = GestureDetectorCompat(this, MyGestureListener())

        // Double-tap detection
        gestureDetector.setOnDoubleTapListener(MyDoubleTapListener())
    }

    override fun onTouchEvent(event: MotionEvent): Boolean {
        return if (gestureDetector.onTouchEvent(event)) {
            true
        } else {
            super.onTouchEvent(event)
        }
    }

    // Gestes simples
    private inner class MyGestureListener : GestureDetector.SimpleOnGestureListener() {

        override fun onDown(e: MotionEvent): Boolean {
            // Touch down (toujours retourner true)
            return true
        }

        override fun onSingleTapUp(e: MotionEvent): Boolean {
            // Tap simple
            Log.d("Gesture", "Single tap at ${e.x}, ${e.y}")
            return true
        }

        override fun onLongPress(e: MotionEvent) {
            // Appui long
            Log.d("Gesture", "Long press")
        }

        override fun onScroll(
            e1: MotionEvent?,
            e2: MotionEvent,
            distanceX: Float,
            distanceY: Float
        ): Boolean {
            // Scroll (glissement)
            Log.d("Gesture", "Scroll: $distanceX, $distanceY")
            return true
        }

        override fun onFling(
            e1: MotionEvent?,
            e2: MotionEvent,
            velocityX: Float,
            velocityY: Float
        ): Boolean {
```

```
            // Fling (balayage rapide)
            val threshold = 100
            val velocityThreshold = 100

            if (e1 != null) {
                val diffX = e2.x - e1.x
                val diffY = e2.y - e1.y

                if (Math.abs(diffX) > Math.abs(diffY)) {
                    // Swipe horizontal
                    if (Math.abs(diffX) > threshold &&
                        Math.abs(velocityX) > velocityThreshold) {
                        if (diffX > 0) {
                            onSwipeRight()
                        } else {
                            onSwipeLeft()
                        }
                        return true
                    }
                } else {
                    // Swipe vertical
                    if (Math.abs(diffY) > threshold &&
                        Math.abs(velocityY) > velocityThreshold) {
                        if (diffY > 0) {
                            onSwipeDown()
                        } else {
                            onSwipeUp()
                        }
                        return true
                    }
                }
            }
            return false
        }
    }

    // Double-tap
    private inner class MyDoubleTapListener : GestureDetector.OnDoubleTapListener {

        override fun onSingleTapConfirmed(e: MotionEvent): Boolean {
            // Tap simple confirmé (pas suivi d'un double-tap)
            Log.d("Gesture", "Single tap confirmed")
            return true
        }

        override fun onDoubleTap(e: MotionEvent): Boolean {
            // Double-tap
            Log.d("Gesture", "Double tap")
            return true
        }

        override fun onDoubleTapEvent(e: MotionEvent): Boolean {
            // Événements durant le double-tap
            return true
        }
    }

    private fun onSwipeLeft() { Log.d("Gesture", "Swipe left") }
    private fun onSwipeRight() { Log.d("Gesture", "Swipe right") }
    private fun onSwipeUp() { Log.d("Gesture", "Swipe up") }
    private fun onSwipeDown() { Log.d("Gesture", "Swipe down") }
}
```

Utilisation sur une vue spécifique :

```
val view = findViewById<View>(R.id.gesture_view)
view.setOnTouchListener { _, event ->
    gestureDetector.onTouchEvent(event)
}
```

9.3 Pinch to zoom

```
class ZoomView @JvmOverloads constructor(
    context: Context,
    attrs: AttributeSet? = null
) : View(context, attrs) {

    private val scaleDetector = ScaleGestureDetector(context, ScaleListener())
    private var scaleFactor = 1f

    override fun onTouchEvent(event: MotionEvent): Boolean {
        scaleDetector.onTouchEvent(event)
        return true
    }

    override fun onDraw(canvas: Canvas) {
        super.onDraw(canvas)
        canvas.save()
        canvas.scale(scaleFactor, scaleFactor)
        // Dessiner le contenu
        canvas.restore()
    }

    private inner class ScaleListener : ScaleGestureDetector.SimpleOnScaleGestureListener() {
        override fun onScale(detector: ScaleGestureDetector): Boolean {
            scaleFactor *= detector.scaleFactor
            scaleFactor = Math.max(0.1f, Math.min(scaleFactor, 5.0f))
            invalidate()
            return true
        }
    }
}
```

10 Bonnes pratiques

10.1 Performance

1. **RecyclerView + DiffUtil** : toujours préférer au ListView
2. **ViewHolder pattern** : éviter les `findViewById()` répétés
3. **Threading** : opérations longues hors UI-Thread
4. **Image loading** : utiliser Glide/Picasso/Coil pour les images
5. **Memory leaks** : attention aux références de Context dans les listeners

10.2 UI/UX

1. **Material Design** : suivre les guidelines Google
2. **Feedback visuel** : indiquer les actions (loading, succès, erreur)
3. **Accessibilité** : `contentDescription` sur les images
4. **Responsive** : tester différentes tailles d'écran
5. **Dark mode** : supporter le thème sombre

10.3 Code

1. **View Binding / Data Binding** : remplacer `findViewById()`
2. **Canaux de notifications** : appropriés et bien nommés
3. **PendingIntent** : toujours FLAG_IMMUTABLE sur Android 12+
4. **Lifecycle awareness** : utiliser ViewModel, LiveData, Flow
5. **Coroutines** : préférer aux threads pour l'asynchrone