Guide de Révision Android

Optimisé pour Examen QCM & Analyse de Code

Stratégie QCM : Ce guide met l'accent sur les *mots-clés*, les *différences* (Tableaux) et la *reconnaissance de code* (Code Radar). Pour le code, ne lisez pas tout, cherchez les méthodes spécifiques indiquées.

1. Évolution & Hardware (SoC)

Dates Clés (QCM Potentiel)

- **2007 :** Lancement iPhone (Tactile capacitif).
- **2008 :** Lancement Android & App Store.
- **Android :** Basé sur noyau **Linux** (Open Source). ~73% de parts de marché.

Le SoC (System on Chip)

Regroupe tous les composants sur une puce :

- **CPU :** Cerveau (Exécute instructions).
- **GPU :** Graphismes (Jeux, UI).
- **DSP :** Signal (Audio/Vidéo).
- **ISP :** Traitement Image (Caméra).

Comparatif Écrans

Technologie	Avantages	Inconvénients
LCD	Moins cher, Lumineux (bon pour extérieur)	Contraste faible, Consomme + (Rétroéclairage)
OLED / AMOLED	Contraste infini (Noirs profonds), Économe (pixels éteints)	Plus cher, Risque de marquage (Burn-in)

Mémoire :

- **RAM (LPDDR) :** Volatile, exécution applications.
- **ROM (Flash) :** Stockage OS + Données utilisateur.

2. Architecture Android

Android est un empilement de couches (Stack). De bas en haut :

1. **Noyau Linux (Kernel) :** Drivers (Hardware, Wifi, Caméra), gestion mémoire/processus.
2. **HAL (Hardware Abstraction Layer) :** Interface standardisée.
3. **Bibliothèques Natives (C/C++) & Runtime :** OpenGL, SQLite, WebKit + **ART/Dalvik**.
4. **Java API Framework :** Ce que les devs utilisent (Activity Manager, Notification Manager).
5. **Applications Système :** Contacts, Téléphone, Vos apps.

Dalvik vs ART (Question classique)

Machine Virtuelle	Compilation	Détails
DVM (Dalvik)	**JIT** (Just-In-Time)	Compile à l'exécution. Plus lent au lancement, prend moins de place. (Avant Android 5.0)
ART (Android Runtime)	**AOT** (Ahead-Of-Time)	Compile à l'installation. Lancement rapide, fluide, mais installation + longue.

3. Outils & Build

SDK (Software Dev Kit)
Contient les librairies, outils de compilation, émulateurs. Nécessaire pour coder en Java/Kotlin.

NDK (Native Dev Kit)
Pour coder en C/C++ (Jeux, haute performance).

Gradle
Outil d'automatisation de build. Gère les **dépendances** et la création de l'APK.

```
ADB (Android Debug Bridge) : Outil ligne de commande pour communiquer avec le téléphone.
> adb install monapp.apk
> adb logcat (pour voir les logs/erreurs)
```

4. Composants & Cycle de Vie

*Tous les composants doivent être déclarés dans **AndroidManifest.xml**.*

- **Activity :** Un écran interface utilisateur (UI).
- **Service :** Tâche de fond sans UI (Musique, Téléchargement).
- **Broadcast Receiver :** Écoute les événements système (Batterie faible, SMS reçu).
- **Content Provider :** Partage de données entre applications (ex: Accéder aux contacts).
- **Intent :** Message pour communiquer entre composants.
  Explicite : Lance une classe précise (Moi → MaPage2).
  Implicite : Demande une action (Moi → Ouvrir Caméra).

Cycle de Vie Activity (À SAVOIR PAR CŒUR)

1. **onCreate()** : Init (setContentView).
2. **onStart()** : Visible mais pas interactif.
3. **onResume()** : Interactif (Au premier plan).
4. **onPause()** : Partiellement visible (popup). Sauvegarder données ici.
5. **onStop()** : Plus visible.
6. **onDestroy()** : Fermeture finale.

Piège QCM : Rotation d'écran = L'activité est détruite (onDestroy) puis recréée (onCreate).

5. Interface Utilisateur & Événements

Layouts (Conteneurs)

- **LinearLayout :** Aligne enfants verticalement ou horizontalement.
- **RelativeLayout :** Position relative (à gauche de, sous...).
- **ConstraintLayout :** Complexe, flexible, plat (Recommandé).

Unités & Ressources

- **dp (density-independent pixel) :** Pour la taille des vues (boutons, marges).
- **sp (scale-independent pixel) :** Pour la taille du **texte** (respecte pref utilisateur).
- **R.java :** Classe générée auto contenant les ID des ressources (`R.id.btn`, `R.layout.main`).

CODE RADAR : Listeners

Si le QCM demande "Comment gérer un clic ?", cherchez ce pattern :

```
Button btn = findViewById(R.id.monBouton);
// Le mot clé est setOnClickListener
btn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Action ici (ex: Toast)
        Toast.makeText(context, "Click!", Toast.LENGTH_SHORT).show();
    }
});
```

6. Persistance des Données

Trois méthodes principales pour sauvegarder des données localement.

A. SharedPreferences (Clé-Valeur)

Pour petites données (paramètres, login). Fichier XML.

Pour ÉCRIRE (Mots clés) :

- `.edit()`
- `.putString("cle", "valeur")`
- `.apply()` (Asynchrone, rapide)
- OU `.commit()` (Synchrone, bloquant)

Pour LIRE (Mots clés) :

- `.getString("cle", "defaut")`
- `.getInt("cle", 0)`

B. Fichiers (Interne vs Externe)

- **Interne :** Privé à l'app. Supprimé si app désinstallée. `openFileOutput()`.
- **Externe :** Public (Photos, PDF). Nécessite permissions.

C. Base de Données SQLite

Base relationnelle structurée. Utilise la classe `SQLiteOpenHelper`.

CODE RADAR : SQLiteOpenHelper

Si on vous montre une classe héritant de `SQLiteOpenHelper`, regardez ces deux méthodes :

```
public class MyDB extends SQLiteOpenHelper {

    // 1. onCreate : Appelé 1 seule fois pour créer les tables
    public void onCreate(SQLiteDatabase db) {
        db.execSQL("CREATE TABLE users (id INTEGER, nom TEXT)");
    }

    // 2. onUpgrade : Appelé si la version de la DB change
    public void onUpgrade(...) {
        db.execSQL("DROP TABLE IF EXISTS users"); // Supprime l'ancienne
        onCreate(db); // Recrée la nouvelle
    }
}
```

CODE RADAR : Opérations CRUD

```
INSERT db.insert("table", null, contentValues);

READ Cursor c = db.rawQuery("SELECT *...", null);
// Utilise un Cursor pour parcourir les résultats

UPDATE db.update("table", values, "id=?", args);

DELETE db.delete("table", "id=?", args);
```