# Annexe 1– Exercice récapitulatif Intents /persistance avec fichiers texte / utilisation du langage Kotlin

**Pour démarrer Android Studio cette session : C:\travail\Android-studio\ studio64.exe ou raccourci sur le bureau**

**Pour créer un nouveau projet avec un activité initiale, choisir «  Empty Views Activity » :** Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

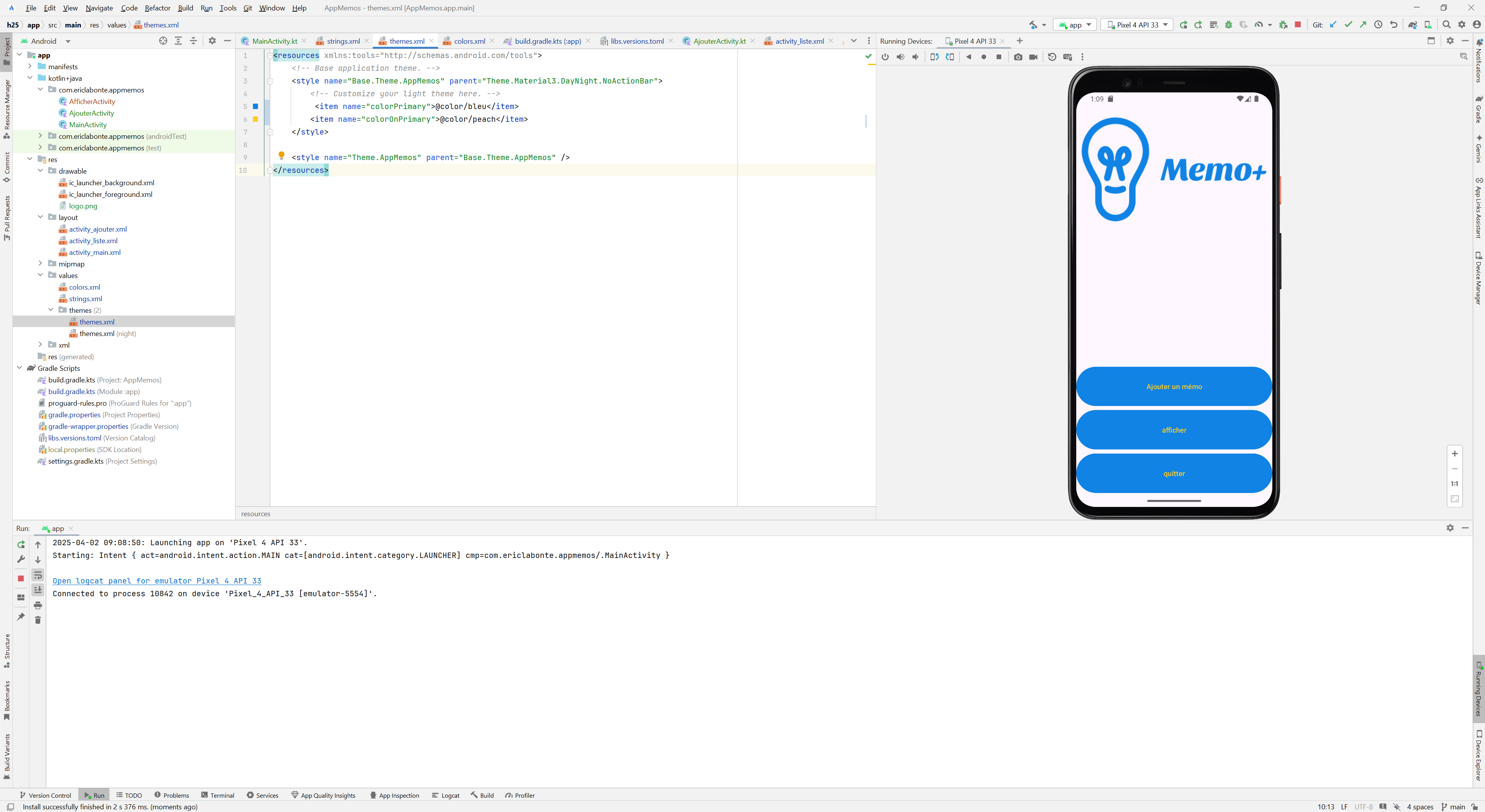
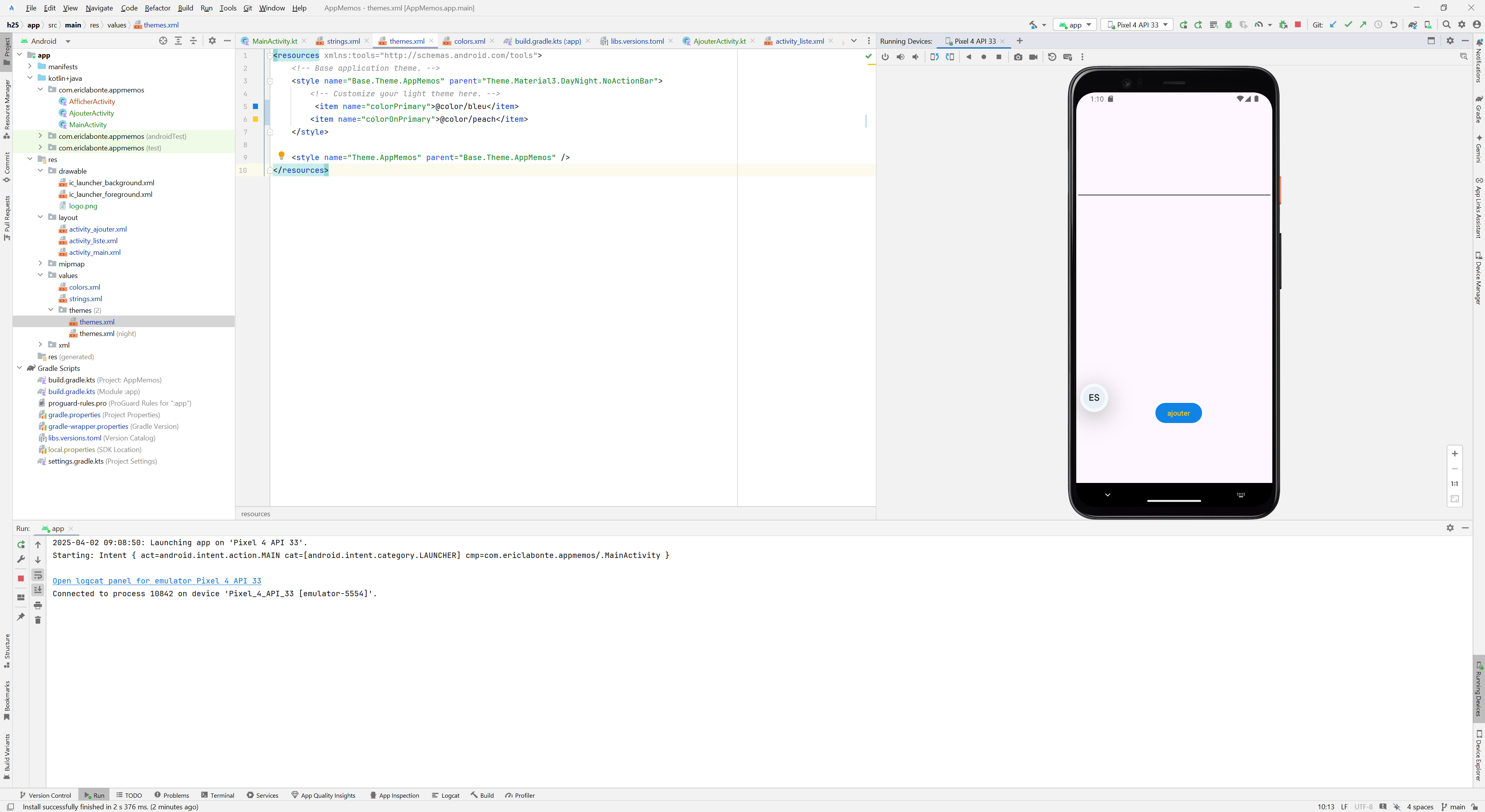
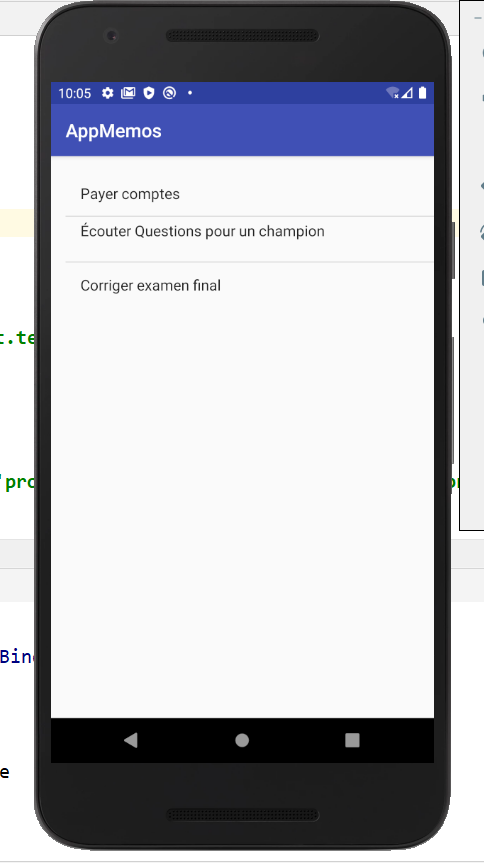
Description générée automatiquement

Langage Kotlin

* Interopérable avec Java : En Kotlin, on peut créer des objects provenant de classes Java
* **Voir présentation Prezi**

**Faire une application comprenant 3 Activités permettant de gérer nos tâches à faire en se gardant des mémos en utilisant Kotlin *( 35 min )***

* 1. L’application possède 3 activités
  + MainActivity (page d’accueil)
    - Intégrer l’image logo et faites en sorte que le logo prenne le tiers de l’espace et les 3 boutons un autre tiers ( utiliser un widget Space pour le tiers central et un/des LinearLayouts )
    - Faites-en sorte que le fond des boutons et le texte écrit par l’usager soit de couleur #1083e5
  + AjouterActivity (page permettant l’ajout d’un mémo)
  + ListeActivity (page permettant de consulter la liste des mémos, présentés avec un ListView, la remplir avec un ArrayAdapter )

2. *( 15 min )* Faites la gestion d’événements sur les 3 boutons de l’activité initiale de manière à ce que :

* Clic sur Ajouter nous amène à l’activité Ajouter
* Clic sur Afficher nous amène à l’activité Afficher
* Clic sur Quitter ferme l’activité ( finish )
* Couleur pour les boutons et le texte sur les boutons 🡪 utiliser le thème, voir :

[Color roles - Material Design 3](https://m3.material.io/styles/color/roles)

## 3.Modèle de données

On cherche à conserver une liste de Strings,

Comment stocker les mémos ajoutés pour qu’ils soient conservés en mémoire après la fermeture de l’app ?

A) base de données SQLite : Oui, c’est une possibilité mais plusieurs autres options s’offrent à nous dans le système Android. C’est beaucoup de travail pour ne conserver que quelques strings, pour nous et pour l’appareil

B) ( *30 min* ) fichiers en stockage interne sur l’appareil Android

## Stockage interne vs Stockage externe des fichiers

Un peu plus complexe qu’il n’y paraît …

|  |  |  |  |
| --- | --- | --- | --- |
| Type de stockage | Format | | Accessibilité |
| **Pour toutes les versions du SDK / de l’API** | | | |
| Stockage interne | .xml ( SharedPreferences /DataStore )  .txt ( fichier texte ) | | Seulement l’app qui a accès à ces données-là |
| **Pour les versions antérieures à SDK 29 / API 10** | | | |
| Stockage externe primaire / stockage partagé | Photos, images, documents téléchargés, fichiers de données que tu veux rendre disponible à plusieurs apps | | Plusieurs app en autant qu’on donne les permissions dans le fichier AndroidManifest |
| Stockage externe secondaire ( carte SD ) | Photos, images, documents téléchargés, fichiers de données que tu veux rendre disponible à plusieurs apps | | Plusieurs app en autant qu’on donne les permissions dans le fichier AndroidManifest |
| **Depuis les versions SDK 29 / API 10 et suivantes** | | | |
| Scoped Storage / stockage étendu | | Beaucoup moins de liberté à pouvoir créer/ajouter des fichiers sur le téléphone : endroits précis seulement | Pas de permissions à écrire car moins d’endroits où l’on peut créer/ajouter/lire des fichiers |

## Sauvegarde dans un fichier texte en **stockage interne**

**Accéder à la mémoire interne**

On va utiliser les différents types de flux Java / Kotlin pour écrire les mémos dans un fichier binaire stocké dans la mémoire interne

On définit les entrées / sorties en termes de flots de données ( streams )

Flot de données : c’est une suite ordonnée d’infos qui provient d’une source ( input stream ) ou qui se dirige vers une destination ( output stream )

**Fichier 🡪 flot d’entrée / lecture 🡪 Programme / app Android**

**Programme/ app Android 🡪 flot de sortie/écriture 🡪 Fichier**

De plus, on peut séparer les flux de données par rapport à leur contenu. 2 grandes catégories existent :

* **Flots d’octets / binaires** ( File**InputStream**, File**OutputStream**, … ) pour travailler avec des Images, du Son ou faire de la sérialisation d’objets.
* **Flots de caractères** : ( File**Reader**, File**Writer**, StringReader…) pour travailler avec des caractères.

**\*\*\*IMPORTANT** : On utilise toujours ces flux de communication en collaboration avec un flux de traitement, un Buffer ( tampon )🡪 permet de régulariser l’accès aux données, évite que chaque lecture d’octet ou de caractère nécessite un accès au fichier texte ( beaucoup d’aller-retour ).

Se convaincre de son efficacité : démonstration avec exercice à l’annexe 2 ( MP3 )

L’utilisation de buffers permet donc la lecture de lignes de texte ( readLine )

Les principaux flux de traitement ( Buffers ) : BufferedReader, BufferedWriter, BufferedInputStream, BufferedOutputStream.

En écriture …

* Importance du flush , la méthode flush est appelée par la méthode close et vide le flux dans le fichier texte de destination. Il peut être nécessaire de l’appeler par elle-même lorsque fait de l’écriture et de la lecture simultanément.
* Importance du paramètre mode de la méthode openFileOutput: lire là-dessus dans la Javadoc

**Méthodes** **propres à notre situation :**

ANDROID

|  |  |  |
| --- | --- | --- |
| **En écriture** |  | **En lecture** |
| JAVA/KOTLIN openFileOutput  (String nomDuFichier, int mode ):FileOutputStream | Permet d’accéder à un fichier en mémoire interne : **retourne un flux binaire ( propre à Android )**  ***On ne fait jamais de new*** | openFileInput  (String nomDuFichier) : FileInputStream |
| OutputStreamWriter | Permet de convertir le flux binaire en flux de caractères | InputStreamReader |
| BufferedWriter | Tampon accélérant le processus d’écriture ( ne pas oublier de fermer le flux à la fin de l’utilisation ) | BufferedReader |

JAVA PUR ( PAS EN ANDROID )

|  |  |  |
| --- | --- | --- |
| **En écriture** |  | **En lecture** |
| new FileWriter ou new FileOutputStream | Permet d’accéder à un fichier en mémoire interne : **retourne un flux de caractères ou un flux d’octets** | new FileReader ou new FileInputStream |
| BufferedWriter ou BufferedOutputStream | Tampon accélérant le processus d’écriture ( ne pas oublier de fermer le flux à la fin de l’utilisation ) | BufferedReader ou BufferedInputStream |

**Exercice 1** *( 30 min )* : faites la gestion du bouton de l’activité « ajouter » dans le projet Mémos de manière à ce que le nouveau mémo soit sauvegardé dans un fichier texte en stockage interne

**Exercice 2** *( 40 min* ) : faire la gestion du bouton Afficher de l’activité de départ de manière à ce que les mémos affichés soient ceux indiqués dans le fichier en stockage interne de l’exercice 1

\*\*\* le fichier utilisé peut-être vu en tant que développeur dans le dossier data/data/nom du package/files

**Où sont disparus les try / catch en Kotlin ? :**

**En kotlin, les exceptions sont toutes non controller (RuntimeException)**

**Try/Catch facultatif**

**Methode use : Permet de fermer le flux de donnees a la fin de l’utilisation, qu’il y ait eu une exception ou non**

**Use pas de parametre car le parametre est une lambda (c’est une fonction) 🡪 on peut coder directement les lignes de code.**