Laboratoire 7

Persistance de données

Objectifs d'apprentissage

- Créer et utiliser les fichiers de préférences
- Manipulation des fichiers standards d'Android
- Créer et manipuler une base de données SQLite
- Comprendre les fournisseurs de contenus

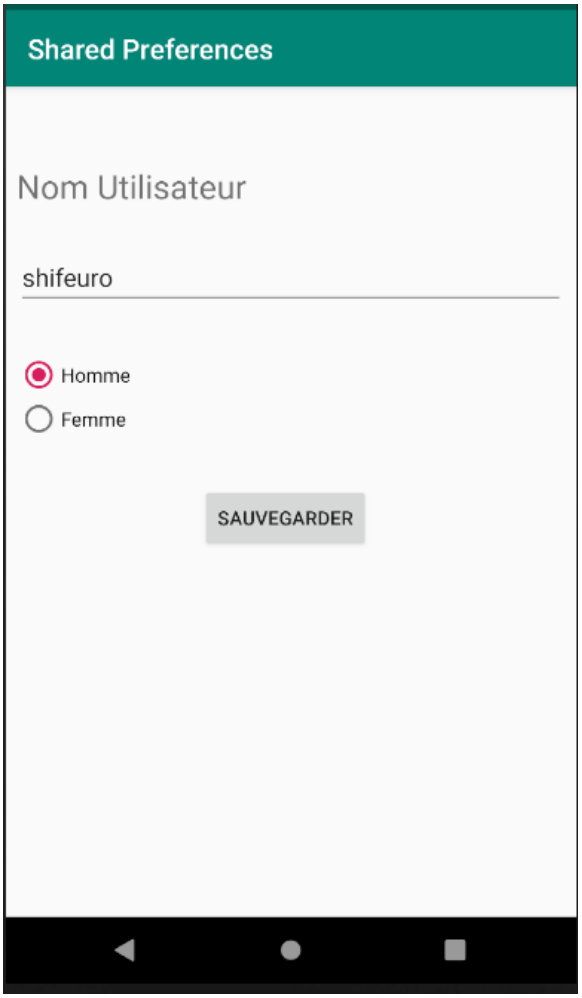
Les fichiers de préférences, les fichiers standards et les bases données sont des mécanismes offerts par Android pour assurer la persistance de données.

1- Les fichiers de préférences

Cette technologie est semblable à la base de registre de Windows ou java.util.pref de Java. Des données de configuration de type primitif sauvegardées sous forme de clés/valeurs, sont récupérées lors du chargement de l'application et sont mises à jour dans le programme à l'aide d'une interface graphique appropriée.

- Créer un nouveau projet Laboratoire 6 avec le template Empty Activity. La vue *activity_main* doit ressembler à ce qui suit :

- Créer l'activité *PreferencesActivity* dont l'interface graphique contient une zône de saisie, deux boutons radio (utiliser le composant RadioGroup pour les regrouper ensemble) et un bouton *sauvegarder.* Dans le fichier *strings.xml*, changer la valeur de *title_activity_preferences* (propriété label de l'activité) pour avoir ce qui suit :



- Programmer le clic du bouton *Activité préférences* de *MainActivity* qui affiche l'activité *PreferencesActivity*.
- Compléter la classe *PreferencesActivity* avec le code suivant :

```
class PreferencesActivity : AppCompatActivity() {
    private val PREF_USERNAME = "USERNAME"
    private val PREF_SEX = "SEX"
    private val GROUPE_HOMME = 0
    private val GROUPE_FEMME = 1

    public override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_preferences)
        restaurer()
    }
```

```
    fun onClickBtnSauvegarder(v: View) {
        val editPrefs = getPreferences(Context.MODE_PRIVATE).edit()

        val choice = when(radioGroupSexe.checkedRadioButtonId){
            R.id.radioHomme -> GROUPE_HOMME
            else -> GROUPE_FEMME
        }

        editPrefs.putInt(PREF_SEX, choice)
            .putString(PREF_USERNAME, etUsername.text.toString())
            .apply()

        etUsername.setText("")
        Toast.makeText(this, "Données sauvegardées!",
Toast.LENGTH_SHORT).show()
    }

    private fun restaurer() {
        val prefs = getPreferences(Context.MODE_PRIVATE)
        val choice = prefs.getInt(PREF_SEX, 0)
        radioGroupSexe.check(
            when(choice){
                GROUPE_HOMME -> R.id.radioHomme
                else -> R.id.radioFemme
            }
        )
        etUsername.setText(prefs.getString(PREF_USERNAME, ""))
    }
}
```

Le nom du fichier XML utilisé pour la sauvegarde des préférences est par défaut celui de l'activité concernée. Si on veut en créer plusieurs, il faut préciser le nom à la méthode *getPreferences* juste avant le mode de création (*Context.MODE_PRIVATE*).

- Pour tester la sauvegarde des données dans le fichier de préférences. Exécutez votre application. Entrez un nom d'utilisateur, choisissez un sexe. Sauvegardez.
- Quittez l'application puis relancez-la. Observez l'état des composants de l'activité *PreferencesActivity*.
- Refaites la même manipulation mais sans sauvegardez cette fois-ci. Observez le résultat.

2- Les fichiers standards

Les fichiers de préférences constituent la solution pour des données de type primitif. Mais si on veut stocker des données plus complexes ou des données brutes, il faut pouvoir créer, lire et écrire dans des fichiers. Android permet d'enregistrer des fichiers sur le stockage interne de l'appareil ou sur un stockage externe (une carte SD par exemple). L'exemple suivant consiste à créer, lire et écrire un fichier sur le stockage interne.

- Ajouter une classe *FichierActivity* à votre projet dont l'interface graphique contient : une zone de saisie, un bouton *Ecrire dans un Fichier* et un *bouton Lire le Fichier :*

- Programmer le clic du bouton *Activite Fichier* de la classe *MainActivity* pour afficher l'activité *FichierActivity.*
- Compléter la classe *FichierActivity* comme suit :

```
class FichierActivity : AppCompatActivity() {
    private val FILENAME = "donnees.dat"

    public override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_fichier)
    }

    /*
    * Clic du bouton Ecrire dans un Fichier
    */
    fun onClickBtnEcrireFichier(v: View) {
        lateinit var fos: FileOutputStream
        lateinit var osw: OutputStreamWriter
        try {
            fos = openFileOutput(FILENAME, Context.MODE_APPEND)

            osw = OutputStreamWriter(fos)
            osw.write(edTextFich.text.toString())
            osw.flush()

            Toast.makeText(this, "Données insérées dans le fichier.",
Toast.LENGTH_SHORT).show()
        } catch (e: Exception) {
            Toast.makeText(this, "Données non insérées",
Toast.LENGTH_SHORT).show()
        } finally {
            try {
                edTextFich.setText("")
                osw.close()
                fos.close()
            } catch (e: IOException) {
                Toast.makeText(this, "Données non insérées ",
Toast.LENGTH_SHORT).show()
            }
        }
    }

    /*
    * Clic du bouton Lire le fichier
    */
    fun onClickBtnLireFichier(v: View) {
        lateinit var fis: FileInputStream
        lateinit var isr: InputStreamReader
        val inputBuffer = CharArray(255)
        lateinit var data: String
        try {
            fis = openFileInput(FILENAME)
            isr = InputStreamReader(fis)
            isr.read(inputBuffer)
            data = String(inputBuffer)

            Toast.makeText(this, data.toString(), Toast.LENGTH_SHORT).show()
        } catch (e: Exception) {
            Toast.makeText(this, "Fichier non lu", Toast.LENGTH_SHORT).show()

        }finally {
            edTextFich.setText("")
        }
    }
}
```

3- Les bases de données

Android fournit *SQLiteOpenHelper*, une classe abstraite, pour la création et la mise à jour des bases de données.

- Programmer le bouton ***Activite Base de données*** pour afficher l'activité suivante (BDActivity)

- Vous devez construire le code de l'activité correspondante comme suit.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".BDActivity">

    <Button
            android:text="Creer une base donnee et y inserer des donnees"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/btnDBCreation" android:layout_marginTop="8dp"
            app:layout_constraintTop_toTopOf="parent"
app:layout_constraintStart_toStartOf="parent"
            android:layout_marginStart="8dp"
app:layout_constraintEnd_toEndOf="parent" android:layout_marginEnd="8dp"
            android:layout_marginBottom="8dp"
app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintHorizontal_bias="0.498"
app:layout_constraintVertical_bias="0.119"
            android:onClick="onClickBtnCreer"/>
    <Button
            android:text="Afficher les donnees"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/btnDisplayData" android:layout_marginTop="8dp"
            app:layout_constraintTop_toBottomOf="@+id/btnDBCreation"
app:layout_constraintStart_toStartOf="parent"
            android:layout_marginStart="8dp"
app:layout_constraintEnd_toEndOf="parent" android:layout_marginEnd="8dp"
            android:layout_marginBottom="8dp"
app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintHorizontal_bias="0.498"
app:layout_constraintVertical_bias="0.114"
            android:onClick="onClickBtnAfficherDonnees"/>
    <TextView
            android:text="Donnees"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/tvData"
app:layout_constraintStart_toStartOf="parent"
android:layout_marginStart="8dp"
            app:layout_constraintEnd_toEndOf="parent"
android:layout_marginEnd="8dp"
            app:layout_constraintTop_toBottomOf="@+id/btnDisplayData"
android:layout_marginBottom="8dp"
            app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintVertical_bias="0.203"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Remarque : Voici le code de la classe de la classe ***BDActivity.*** *tvData* est l'identicateur du *textView* où seront affichées les données lues.

```
class BDActivity : AppCompatActivity() {
    class BDAssistant(context: Context) : SQLiteOpenHelper(context, NOM_BASE, null,
VERSION) {

        override fun onCreate(arg0: SQLiteDatabase) {
            arg0.execSQL(
                String.format(
                    "CREATE TABLE %s (%s INTEGER PRIMARY KEY AUTOINCREMENT, %s
TEXT, %s TEXT)",
                    NOM_TABLE, BaseColumns._ID, CHAMP_NOM, CHAMP_PRENOM
                )
            )
        }

        override fun onUpgrade(arg0: SQLiteDatabase, arg1: Int, arg2: Int) {}

        companion object {
            val VERSION = 1
            val NOM_BASE = "maBase"
            val NOM_TABLE = "maTable"
            val CHAMP_NOM = "nom"
            val CHAMP_PRENOM = "prenom"
        }
    } // fin de la classe BDAssistant

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

    private fun insererDonnees() {
        val bdAssistant = BDAssistant(this)
        val bd = bdAssistant.writableDatabase

        var valeurs = ContentValues()

        valeurs.put(BDAssistant.CHAMP_NOM, "Nom1_" + System.currentTimeMillis())
        valeurs.put(BDAssistant.CHAMP_PRENOM, "Prénom1_" +
System.currentTimeMillis())
        bd.insertOrThrow(BDAssistant.NOM_TABLE, null, valeurs)

        valeurs = ContentValues()

        valeurs.put(BDAssistant.CHAMP_NOM, "Nom2_" + System.currentTimeMillis())
        valeurs.put(BDAssistant.CHAMP_PRENOM, "Prénom2_" +
System.currentTimeMillis())
        bd.insertOrThrow(BDAssistant.NOM_TABLE, null, valeurs)

        bd.close()
        bdAssistant.close()
    }

    private fun lireBD() {
        val bdAssistant = BDAssistant(this)
        val bd = bdAssistant.readableDatabase
        val noms = StringBuffer("Noms et Prénoms :\n\n")
```

```
        val curseur = bd.query(
            BDAssistant.NOM_TABLE,
            arrayOf(BDAssistant.CHAMP_NOM, BDAssistant.CHAMP_PRENOM), null, null,
null, null,
            "${BDAssistant.CHAMP_NOM} desc"
        )

        while (curseur.moveToNext()) {
            val nomEtPrenom = curseur.getString(0) + " " + curseur.getString(1)
            noms.append(nomEtPrenom + "\n\n")
        }

        bd.close()
        bdAssistant.close()

        val tvNoms = findViewById<View>(R.id.tvData) as TextView
        tvNoms.text = noms
        curseur.close()
    }

    fun onClickBtnCreer(v: View) {
        insererDonnees()
        Toast.makeText(this, "Données insérées avec succès",
Toast.LENGTH_SHORT).show()
    }

    fun onClickBtnAfficherDonnees(v: View) {
        lireBD()
        Toast.makeText(this, "Données lues avec succès", Toast.LENGTH_SHORT).show()
    }
}
```

Essayez de comprendre le code qui vous est fourni et complétez l'application pour qu'elle fonctionne.

- Un autre exemple de manipulation de base de données SQLite StudentDBManagement vous est fourni dans le cadre du laboratoire vous permettant de manipuler des étudiants. Ouvrez ce programme et essayez de comprendre le code.
- Modifiez ce programme pour permettre à un utilisateur de pouvoir insérer des données saisies dans une activité additionnelle que vous devez ajouter à l'application.

4- Les fournisseurs de contenus

Les fournisseurs de contenus permettent de partager des données entre toutes les applications. Il existe par défaut plusieurs fournisseurs de contenus pour les données audio, vidéo, images et contacts personnels. Il est aussi possible d'en créer. Quel que soit la solution de stockage utilisée par les fournisseurs de contenus, ces derniers exposent la même interface. Les données sont retournées sous forme de tables de base de données. Chaque ligne est un enregistrement et chaque colonne est une valeur correspond au champ concerné.

Chaque enregistrement (ligne) est identifié de façon unique à l'aide d'un champ numérique _ID.
Chaque table contient une URI unique. Il y aura donc autant d'URI que de tables de données.
Synatxe générale d'une URI :
Content:// autorité de l'URI / sous-chemin/id

Autorité de l'URI : identifie le content provider correspondant à l'URI
Sous-chemin : permet au fournisseur de déterminer le type de données. Vide, s'il un seul type de données est géré.
id : identifiant unique de l'enregistrement demandé (champ _ID). Vide si la requête concerne plus d'un enregistrement.

URI pour obtenir les données des contacts **: ContactsContract.Contacts.CONTENT_URI**

Pour pouvoir utiliser les contacts, il faut ajouter les permissions nécessaires dans le fichier **manifest.xml.**
android.permission.READ_CONTACTS : pour accéder aux données des contacts et
android.permission.WRITE_CONTACTS : pour ajouter, modifier ou supprimer des contacts.
Utiliser l'aide en ligne pour bien comprendre la mise en œuvre des opérations d'ajout, de suppression, d'interrogation et de mise à jour des données du fournisseur de contenus **ContactsContract.Contacts**

Parler de ContactProviderOperation , ContactsContract et de Contacts.Contract.Data

5- Travail à faire

Concevoir et réaliser une application Android qui offre les opérations suivantes :

- **Ajout d'un contact**
  - En utilisant une intention implicite
  - En ajoutant directement le contact dans la base de données des contacts (nom, prénom, courriel et type de courriel (ex « travail »)

Dans cette partie, vous pouvez utiliser une AlertDialog pour vous offrir les choix. Voici un exemple pour ajouter un contact duquel vous pouvez vous inspirez:

```
fun onAjouterClick(view: View){
    val builder = AlertDialog.Builder(this)

    builder.setItems(
```

```
        arrayOf(" Utiliser une intention implicite", "Ajouter directement dans
la BD")
    ) { _, item ->
        Toast.makeText(
            applicationContext,
            "item choisi : $item",
            Toast.LENGTH_LONG
        ).show()
        if (item == 0){
            val intent = Intent(ContactsContract.Intents.Insert.ACTION).apply {
                type = ContactsContract.RawContacts.CONTENT_TYPE
            }
            intent.apply {
                putExtra(ContactsContract.Intents.Insert.EMAIL,
emailAddress?.text)
                putExtra(ContactsContract.Intents.Insert.EMAIL_TYPE,
emailType?.text)
                putExtra(ContactsContract.Intents.Insert.PHONE,
phoneNumber?.text)
                putExtra(ContactsContract.Intents.Insert.PHONE_TYPE,
ContactsContract.CommonDataKinds.Phone.TYPE_MAIN)
                putExtra(ContactsContract.Intents.Insert.NAME,
prenom?.text.toString() + " " + nom?.text.toString())
            }
            startActivity(intent)
        } else {
            val hasWriteContactsPermission =
checkSelfPermission(Manifest.permission.WRITE_CONTACTS)
            if (hasWriteContactsPermission != PackageManager.PERMISSION_GRANTED)
{
                requestPermissions(
                    arrayOf(Manifest.permission.WRITE_CONTACTS),123)
            } else {

                val ops = ArrayList<ContentProviderOperation>()

                var op =
ContentProviderOperation.newInsert(ContactsContract.RawContacts.CONTENT_URI)
                    .withValue(ContactsContract.RawContacts.ACCOUNT_TYPE, null)
                    .withValue(ContactsContract.RawContacts.ACCOUNT_NAME, null)
                ops.add(op.build())

                op =
ContentProviderOperation.newInsert(ContactsContract.Data.CONTENT_URI)

.withValueBackReference(ContactsContract.Data.RAW_CONTACT_ID, 0)
                    .withValue(ContactsContract.Data.MIMETYPE,

ContactsContract.CommonDataKinds.StructuredName.CONTENT_ITEM_TYPE)

.withValue(ContactsContract.CommonDataKinds.StructuredName.DISPLAY_NAME,
prenom?.text.toString() + " " + nom?.text.toString())

.withValue(ContactsContract.CommonDataKinds.StructuredName.FAMILY_NAME,
nom?.text.toString())

.withValue(ContactsContract.CommonDataKinds.StructuredName.GIVEN_NAME,
prenom?.text.toString())
                ops.add(op.build())

                op =
```

```
ContentProviderOperation.newInsert(ContactsContract.Data.CONTENT_URI)

.withValueBackReference(ContactsContract.Data.RAW_CONTACT_ID, 0)
                    .withValue(ContactsContract.Data.MIMETYPE,

ContactsContract.CommonDataKinds.Email.CONTENT_ITEM_TYPE)
                    .withValue(ContactsContract.CommonDataKinds.Email.ADDRESS,
emailAddress?.text.toString())
                    .withValue(ContactsContract.CommonDataKinds.Email.TYPE,
courriel_type.text.toString())
                ops.add(op.build())

                op =
ContentProviderOperation.newInsert(ContactsContract.Data.CONTENT_URI)

.withValueBackReference(ContactsContract.Data.RAW_CONTACT_ID, 0)
                    .withValue(ContactsContract.Data.MIMETYPE,

ContactsContract.CommonDataKinds.Phone.CONTENT_ITEM_TYPE)
                    .withValue(ContactsContract.CommonDataKinds.Phone.NUMBER,
phoneNumber?.text.toString())
                    .withValue(ContactsContract.CommonDataKinds.Phone.TYPE,
ContactsContract.CommonDataKinds.Phone.TYPE_MAIN)
                ops.add(op.build())

                contentResolver.applyBatch(ContactsContract.AUTHORITY, ops)

            }
        }
    }

    builder.create().show()
}
```

- **Recherche d'un contact** dont on connaît le nom et le prénom

- **Suppression d'un contact** en fournissant le nom et le prénom

- **Modification d'un contact**