CR TP1_TodoList

FANG Zhengjie
YANG Liu

1. Introduction

Ce projet a pour objectif d'établir une application TodoList qui peut satisfaire les besoins de faire une liste de chose à faire pour de différents utilisateurs avec différents pseudos, et nous pouvons gérer les pseudos avec le menu setting.

Pour construire une Todolist ayant les fonctions nécessaires, nous avons besoin de quatre activités en total, qui s'en chargent respectivement les quatre écrans qu'on utilise ici. On a réalisé avec fonctionnalités comme ci-dessous parmis les fonctionnalités proposé:

MainActivity

- Permet de saisir son *pseudo* et *password*.
- Les informations pour le dernier utilisateur sont automatiquement renseignées dans le champ de saisie.
- Lors du clic sur le bouton setting dans le menu, une activité **SettingsActivity** s'affiche.
- Lors du clic sur le bouton OK, les informations sont sauvegardées dans les préférences partagées de l'application et une activité **ChoixListActivity** s'affiche.

SettingsActivity

- Permet d'afficher toutes les informations des utilisateurs
- D'autres champs utiles peuvent être ajoutés, pour débogage par exemple

ChoixListActivity

- Affiche la liste des listes de l'utilisateur dont les informations ont été saisi.
- Lors du clic sur le bouton NOUVELLE, une nouvelle liste est ajoutée dans la liste des listes de chose à faire.
- Lors du clic sur une liste, une activité **ShowListActivity** s'affiche.

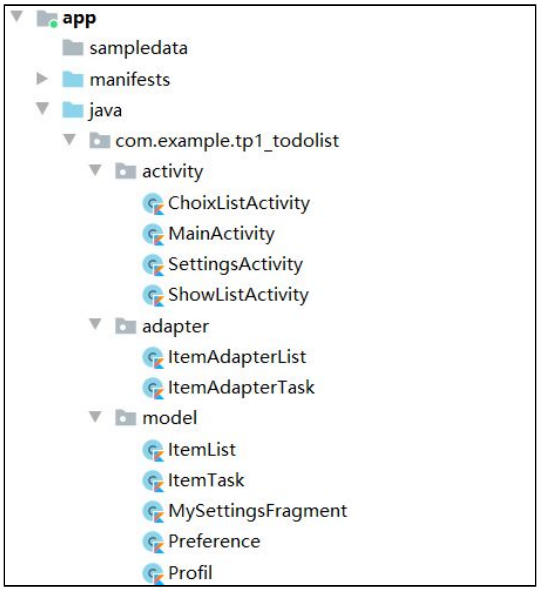
ShowListActivity

- Affiche les items des listes de l'utilisateur dont les informations ont été saisi.
- Permet de cocher et décocher un item.
- Lors du clic sur le bouton NOUVELLE, un nouvel item est ajouté dans la liste des items.

Améliorations facultatives :

- Utiliser des fragments à la place d'une PreferenceActivity

2.Implémentation Globale

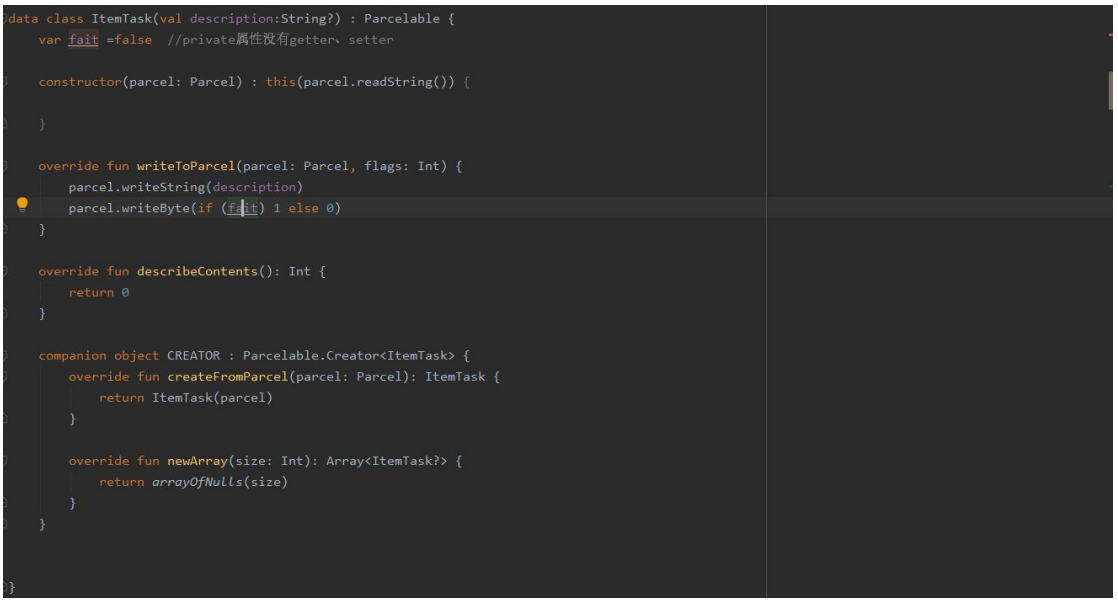


Globalement, nous avons trois packages, un pour les activités, un pour les adaptateurs, un pour les modèles. Le package activity correspond aux activités mentionnées ci-dessus, qui sont en effet les différents écrans de l'utilisation. Le package adapter contient les adaptateurs pour l'utilisation du *RecyclerView* , qui a besoin de l'adaptateur de faire les mises à jours sur les *Views* portés par les *ViewHolders.* Dans le package modèle, il y a non seulement les objets basiques (la liste, la tâche, le profil) dans notre cas, et aussi une classe de l'outillage *Preference* qui sauvegarde nos paramètres partagés.

3.Analyse

Donc pour analyser tout le processus de la réalisation de cette application, les détails vont être précisés à la suite avec la démonstration du code. Nous allons tout d'abord nous concentrer sur les objets qu'on a besoin ici, c'est que les classes *ItemList, ItemTask* et *Profil*.

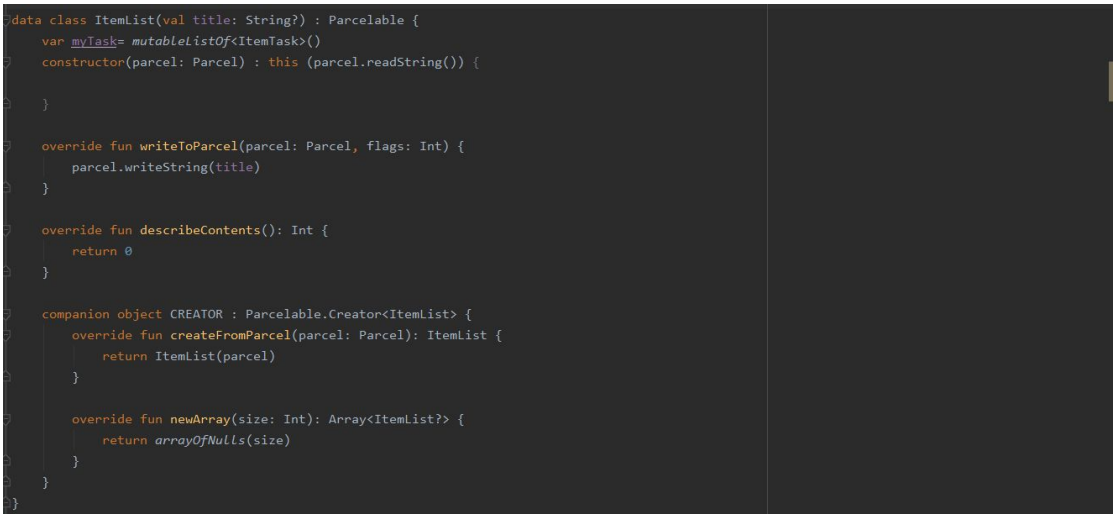
ItemTask



```
data class ItemTask(val description:String?) : Parcelable {
    var fait =false  //private属性没有getter、setter

    constructor(parcel: Parcel) : this(parcel.readString()) {

    }

    override fun writeToParcel(parcel: Parcel, flags: Int) {
        parcel.writeString(description)
        parcel.writeByte(if (fait) 1 else 0)
    }

    override fun describeContents(): Int {
        return 0
    }

    companion object CREATOR : Parcelable.Creator<ItemTask> {
        override fun createFromParcel(parcel: Parcel): ItemTask {
            return ItemTask(parcel)
        }

        override fun newArray(size: Int): Array<ItemTask?> {
            return arrayOfNulls(size)
        }
    }


}
```

Nous devrions commencer par la classe *ItemTask*, car c'est l'objet le plus basique dans cette application. Dans cette classe, il y a deux attributs auxquels on va faire attention, où l'attribut *fait* est pour illustrer l'état d'être coché ou pas, et l'attribut *description* est exactement les choses à faire en forme de *String*.

ItemList



```
data class ItemList(val title: String?) : Parcelable {
    var myTask= mutableListOf<ItemTask>()
    constructor(parcel: Parcel) : this (parcel.readString()) {

    }

    override fun writeToParcel(parcel: Parcel, flags: Int) {
        parcel.writeString(title)
    }

    override fun describeContents(): Int {
        return 0
    }

    companion object CREATOR : Parcelable.Creator<ItemList> {
        override fun createFromParcel(parcel: Parcel): ItemList {
            return ItemList(parcel)
        }

        override fun newArray(size: Int): Array<ItemList?> {
            return arrayOfNulls(size)
        }
    }
}
```

Cette classe a pour objectif de construire un modèle d'une liste des tâches à faire, où l'attribut *myTask* contient tous les items de type *ItemTask*, et l'attribut *title* est le nom de cette liste qu'on a créée auparavant.

Profil

```
data class Profil(private val pseudoName:String?, private val password :String?): Parcelable {
    var myList= mutableListOf<ItemList>()

    constructor(parcel: Parcel) : this(
        parcel.readString(),
        parcel.readString()
    ) {

    }

    override fun writeToParcel(parcel: Parcel, flags: Int) {
        parcel.writeString(pseudoName)
        parcel.writeString(password)
    }

    override fun describeContents(): Int {
        return 0
    }

    companion object CREATOR : Parcelable.Creator<Profil> {
        override fun createFromParcel(parcel: Parcel): Profil {
            return Profil(parcel)
        }

        override fun newArray(size: Int): Array<Profil?> {
            return arrayOfNulls(size)
        }
    }


}
```

Étant donné que l'on voudrait un objet qui peut contenir tous les informations de l'utilisateur, la classe *Profil* est créée pour générer ce type d'objet, dont l'attribut *myList* contient de différents listes de tâches créées par l'utilisateur spécifié. De plus, en tant qu'un utilisateur, il faudrait taper son *pseudoName* et *password.*

Ayant les bases sur les modèles, on peut ensuite commencer à analyser les fonctionnalités de différents activités concernantes dans cette application.

MainActivity

```
class MainActivity : AppCompatActivity() {

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

        val editTextPseudo:EditText=findViewById(R.id.editTextPseudo)
        val editTextPassword:EditText=findViewById(R.id.editTextPassword)
        editTextPassword.inputType = InputType.TYPE_TEXT_VARIATION_VISIBLE_PASSWORD


        val btnMainOK:Button=findViewById(R.id.btnMainOK)
        btnMainOK.setOnClickListener(){ it: View!
            var pseudo by Preference( context: this, name: "pseudo", default: "pseudoDefault")
            pseudo=editTextPseudo.getText().toString()
            var password=editTextPassword.getText().toString()
            Log.i( tag: "MainActivity", msg: "pseudo: $pseudo   password: $password ")
            val profil: Profil =Profil(pseudo,password)
            var jsonProfil by Preference( context: this,pseudo,  default: "jsonProfilDefault")
            if(jsonProfil=="jsonProfilDefault") {
                jsonProfil = Gson().toJson(profil)
            }
            Log.i( tag: "MainActivity", msg: "jsonProfil $jsonProfil")
            Log.i( tag: "MainActivity", msg: "btnMainOK clicked")
            val intentToChoixList:Intent=Intent( packageContext: this@MainActivity,ChoixListActivity::class.java)
            startActivity(intentToChoixList)
        }
    }
    override fun onCreateOptionsMenu(menu: Menu?): Boolean {
        getMenuInflater().inflate(R.menu.main, menu);//这里是调用menu文件夹中的main.xml，在登陆界面label右上角的三角里显示其他功能
        return true;
    }
```

Dans cette classe, les attributs *pseudo* et *password* sont délégués en utilisant la classe *Preference*, ce qui nous permet d'enregistrer les informations dans le *setting*. En plus, quand on clique sur le bouton OK, un écouteur de l'action est mis en place pour le détecter et le changement de l'activité est réalisé par l'utilisation de Intent.

SettingsActivity

Dans la class SettingActivity, on a utilisé un fragment qui est implementé dans la class MyMySettingsFragment dans la module fragment.

ChoixListActivity

```
class ChoixListActivity:AppCompatActivity(),ItemAdapterList.ActionListener {
    var dataSet=mutableListOf<ItemList>()//? ?  列表总是同一个，变化的是里面的元素


    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.choix_list_activity)
        val tvListCtivity:TextView=findViewById(R.id.tvListCtivity)

        var pseudo by Preference( context: this, name: "pseudo", default: "pseudoDefault")
        tvListCtivity.text="$pseudo's List"
        var jsonProfil by Preference( context: this,pseudo, default: "jsonProfilDefault")
        val myProfil: Profil = Gson().fromJson(jsonProfil, Profil::class.java)
        dataSet=myProfil.myList
        Log.i( tag: "ChoixListActivity", msg: "dataSet is $dataSet")
        Log.i( tag: "ChoixListActivity", msg: "myProfil is $myProfil")
        Log.i( tag: "ChoixListActivity", msg: "jsonProfil is $jsonProfil")
        val rvList:RecyclerView = findViewById(R.id.rv_list)
        val btnNouvelleList: Button =findViewById(R.id.btnNouvelleList)
        val etNouvelleList:EditText=findViewById(R.id.etNouvelleList)
        rvList.adapter=newAdapter(dataSet)
        rvList.layoutManager = LinearLayoutManager( context: this, RecyclerView.VERTICAL,  reverseLayout: false)
```

```
        btnNouvelleList.setOnClickListener(){ it: View!
            Log.i( tag: "ChoixListActivity", msg: "btnNouvelleList is clicked")
            val listTitle:String=etNouvelleList.getText().toString()
            Log.i( tag: "ChoixListActivity", msg: "listTitle is $listTitle")

            if(listTitle!=""){
//                dataSet.add(ItemList(listTitle))? ?
                myProfil.myList.add(ItemList(listTitle))
                jsonProfil= Gson().toJson(myProfil)
                Log.i( tag: "ChoixListActivity", msg: "aftrer click btnNouvelle, dataSet is $dataSet")
                Log.i( tag: "ChoixListActivity", msg: "aftrer click btnNouvelle, myProfil is $myProfil")
                Log.i( tag: "ChoixListActivity", msg: "aftrer click btnNouvelle, jsonProfil is $jsonProfil")
                (rvList.adapter as ItemAdapterList).notifyDataSetChanged()
                etNouvelleList.setText("")
            }else{
                val t = Toast.makeText( context: this,  text: "A list name is need", Toast.LENGTH_SHORT)
                t.show()
            }
        }
```

```
    private fun newAdapter(dataSet:MutableList<ItemList>): ItemAdapterList {
        val adapter = ItemAdapterList(
            dataSet=dataSet,
            actionlistener=this//将ChoixListActivity implementer为ItemAdapter.ActionListener
        )
        return adapter
    }


    override fun onItemClicked(listPosition: Int,listTitle:String) {
        Log.i( tag: "onItemClickedList", msg: "listPosition is $listPosition")
        val bundle:Bundle= Bundle()
        bundle.putInt("listPosition",listPosition)
        bundle.putString("listTitle",listTitle)
        val intentToShowListActivity:  Intent= Intent( packageContext: this@ChoixListActivity,ShowListActivity::class.java)
        intentToShowListActivity.putExtras(bundle)
        startActivity(intentToShowListActivity)
    }



}
```

Dans cette activité, on a principalement deux actions à faire, d'où la première est de créer une nouvelle liste en cliquant sur le bouton NOUVELLE, et la deuxième est de changer l'activité à la *ShowListActivity* en cliquant sur le nom de la liste. Alors pour réaliser le premier point, on met un écouteur de l'action pour réagir. En ce qui concerne le deuxième point, la classe *ChoixListActivity* devrait hériter l'interface *ActionListener* créée dans la classe *ItemAdapter,* et une fonction *onItemClicked* doit être créée pour trouver les tâches correspondantes en vérifiant l'attribut *listPosition* et *listTitle*.

ShowListActivity

```
class ShowListActivity :AppCompatActivity(),ItemAdapterTask.ActionListener {
    var dataSetTask=mutableListOf<ItemTask>()//? ?  列表总是同一个，变化的是里面的元素
    var listPosition:Int=0
    var listTiltle:String="ShowListActivity"
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.show_list_activity)

        val bundle=intent.extras
        listPosition= bundle?.getInt( key: "listPosition")!!
        listTiltle=bundle?.getString( key: "listTitle")!!
        val tvShowListCtivity:TextView=findViewById(R.id.tvShowListCtivity)
        tvShowListCtivity.text=listTiltle

        var pseudo by Preference( context: this, name: "pseudo", default: "pseudoDefault")
        var jsonProfil: String by Preference( context: this,pseudo, default: "jsonProfilDefault")
        var myProfil: Profil = Gson().fromJson(jsonProfil, Profil::class.java)
        dataSetTask=myProfil.myList[listPosition].myTask
        Log.i( tag: "ShowListActivity", msg: "listPosition is $listPosition")
        Log.i( tag: "ShowListActivity", msg: "myList is ${myProfil.myList[listPosition]}")
        Log.i( tag: "ShowListActivity", msg: "dataSetTask is $dataSetTask")
        Log.i( tag: "ShowListActivity", msg: "myProfil is $myProfil")
        Log.i( tag: "ShowListActivity", msg: "jsonProfil is $jsonProfil")
        val rvTask: RecyclerView = findViewById(R.id.rv_task)
        val btnNouvelleTask: Button =findViewById(R.id.btnNouvelleTask)
        val etNouvelleTask: EditText =findViewById(R.id.etNouvelleTask)
        rvTask.adapter = newAdapter(dataSetTask)
        rvTask.layoutManager = LinearLayoutManager( context: this, RecyclerView.VERTICAL,  reverseLayout: false)
```

```
        btnNouvelleTask.setOnClickListener(){ it: View!
            Log.i( tag: "ShowListActivity", msg: "btnNouvelleTask is clicked")
            val taskDescription:String=etNouvelleTask.getText().toString()
            Log.i( tag: "ShowListActivity", msg: "Task description is $taskDescription")

            if(taskDescription!=""){
                myProfil.myList[listPosition!!].myTask.add(ItemTask(taskDescription))
                jsonProfil= Gson().toJson(myProfil)
                Log.i( tag: "ShowListActivity", msg: "aftrer click btnNouvelle, dataSetTask is $dataSetTask")
                Log.i( tag: "ShowListActivity", msg: "aftrer click btnNouvelle, myList is ${myProfil.myList[listPosition!!]}")
                Log.i( tag: "ShowListActivity", msg: "aftrer click btnNouvelle, myProfil is $myProfil")
                Log.i( tag: "ShowListActivity", msg: "aftrer click btnNouvelle, jsonProfil is $jsonProfil")
                (rvTask.adapter as ItemAdapterTask).notifyDataSetChanged()
                etNouvelleTask.setText("")
            }else{
                val t = Toast.makeText( context: this,  text: "A task name is need", Toast.LENGTH_SHORT)
                t.show()
            }
        }


    }
```

```
    private fun newAdapter(dataSetTask:MutableList<ItemTask>): ItemAdapterTask {
        val adapter = ItemAdapterTask(
            dataSetTask=dataSetTask,
            actionlistener=this//将ChoixListActivity implementer为ItemAdapter.ActionListener
        )
        return adapter
    }


    override fun onItemClicked(taskPosition: Int) {
        var pseudo by Preference( context: this, name: "pseudo", default: "pseudoDefault")
        var jsonProfil: String by Preference( context: this,pseudo, default: "jsonProfilDefault")
        val myProfil: Profil = Gson().fromJson(jsonProfil, Profil::class.java)
        val taskClickd:ItemTask=myProfil.myList[listPosition].myTask[taskPosition]
        Log.i( tag: "onItemClicked", msg: "mylistPosition is $listPosition")
        Log.i( tag: "onItemClicked", msg: "taskPosition is $taskPosition")
        Log.i( tag: "onItemClicked", msg: "myTask is ${myProfil.myList[listPosition].myTask}")
        Log.i( tag: "onItemClicked", msg: "dataSetTask is $dataSetTask")
        Log.i( tag: "onItemClicked", msg: "myProfil is $myProfil")
        Log.i( tag: "onItemClicked", msg: "jsonProfil is $jsonProfil")
        myProfil.myList[listPosition].myTask[taskPosition].fait = !taskClickd.fait
        jsonProfil=Gson().toJson(myProfil)
        Log.i( tag: "onItemClicked", msg: "after click mylistPosition is $listPosition")
        Log.i( tag: "onItemClicked", msg: "after click taskPosition is $taskPosition")
        Log.i( tag: "onItemClicked", msg: "after click myTask is ${myProfil.myList[listPosition].myTask}")
        Log.i( tag: "onItemClicked", msg: "after click dataSetTask is $dataSetTask")
        Log.i( tag: "onItemClicked", msg: "after click myProfil is $myProfil")
        Log.i( tag: "onItemClicked", msg: "after click jsonProfil is $jsonProfil")
    }

}
```

Dans cette classe-ci, nous avons aussi deux affaires à faire, la première est de créer une nouvelle tâche en cliquant sur le bouton NOUVELLE, et la deuxième est de distribuer une propriété décrivant l'état d'être coché ou pas par l'utilisateur. Du coup d'abord, avec la même stratégie, nous pouvons créer une nouvelle tâche. Cependant, il est un peu différent au niveau du clic sur la tâche. Il est de même obligatoire de hériter une interface créée dans la classe *ItemAdapterTask*, ensuite avec la fonction *onItemClicked*, l'état de cette tâche peut être changé en cliquant sur cette tâche.

À cause de l'utilisation de *RecyclerView*, les adaptateurs doivent être utilisés pour remplir les informations transmises de la base de données. Après c'est la démonstration de la réalisation de cette liaison sur ces deux adaptateurs concernants dans notre problématique.

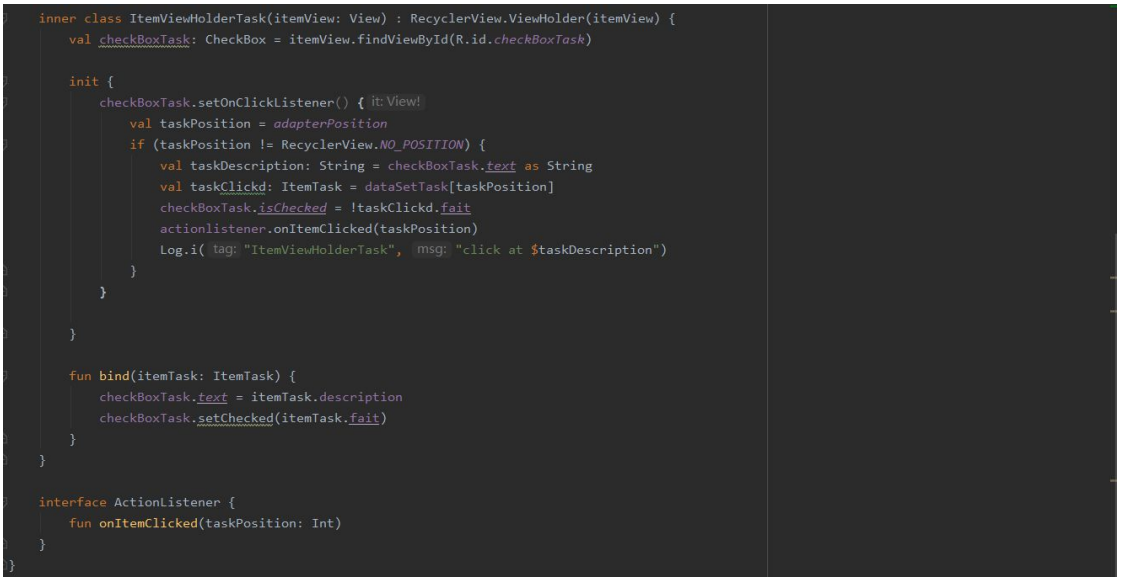
ItemAdapterList

```
class ItemAdapterList(private val dataSet:MutableList<ItemList>,private val actionlistener:ActionListener):
    RecyclerView.Adapter<RecyclerView.ViewHolder>() {
    override fun getItemCount(): Int {
        return dataSet.size
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): RecyclerView.ViewHolder {
        Log.i( tag: "ItemAdapterList",  msg: "createViewHolder")
        val inflater:LayoutInflater= LayoutInflater.from(parent.context)
        val itemView = inflater.inflate(R.layout.item_list,parent, attachToRoot: false)
        return ItemViewHolderList(itemView)
    }

    override fun onBindViewHolder(holder:RecyclerView.ViewHolder, position: Int) {
        Log.i( tag: "ItemAdapterList", msg: "onBindViewHolder")
        val itemList=dataSet[position]
        when (holder) {
            is ItemViewHolderList->{
                holder.bind(itemList)
        }
        }
    }
```

```
    inner class ItemViewHolderList(itemView: View):RecyclerView.ViewHolder(itemView){
        val tvList:TextView = itemView.findViewById(R.id.tvList)

        init {
            itemView.setOnClickListener(){ it: View!
                val listPosition=adapterPosition
                if(listPosition!=RecyclerView.NO_POSITION){
                    val listTitle:String= tvList.text as String
//                    val listClickd:ItemList=dataSet[listPosition]
                    actionlistener.onItemClicked(listPosition,listTitle)
                    Log.i( tag: "ItemViewHolderList", msg: "click at $listTitle")
                    Log.i( tag: "ItemViewHolderList", msg: "listPosition is $listPosition")

                }

            }

        }
        fun bind(itemList: ItemList){
            tvList.text=itemList.title
        }
    }

    interface ActionListener{
        fun onItemClicked(listPosition:Int,listTitle:String)
    }


}
```

ItemAdapterTask

```
class ItemAdapterTask(private val dataSetTask:MutableList<ItemTask>, private val actionlistener:ActionListener):
    RecyclerView.Adapter<RecyclerView.ViewHolder>() {
    override fun getItemCount(): Int {
        return dataSetTask.size
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): RecyclerView.ViewHolder {
        Log.i( tag: "ItemViewHolderTask",  msg: "createViewHolder")
        val inflater: LayoutInflater = LayoutInflater.from(parent.context)
        val itemView = inflater.inflate(R.layout.item_task, parent,  attachToRoot: false)
        return ItemViewHolderTask(itemView)
    }

    override fun onBindViewHolder(holder: RecyclerView.ViewHolder, position: Int) {
        Log.i( tag: "ItemViewHolderTask",  msg: "onBindViewHolder")
        val itemTask = dataSetTask[position]
        when (holder) {
            is ItemViewHolderTask -> {
                holder.bind(itemTask)
            }
        }
    }
```



```
inner class ItemViewHolderTask(itemView: View) : RecyclerView.ViewHolder(itemView) {
    val checkBoxTask: CheckBox = itemView.findViewById(R.id.checkBoxTask)

    init {
        checkBoxTask.setOnClickListener() { it: View!
            val taskPosition = adapterPosition
            if (taskPosition != RecyclerView.NO_POSITION) {
                val taskDescription: String = checkBoxTask.text as String
                val taskClickd: ItemTask = dataSetTask[taskPosition]
                checkBoxTask.isChecked = !taskClickd.fait
                actionlistener.onItemClicked(taskPosition)
                Log.i( tag: "ItemViewHolderTask",  msg: "click at $taskDescription")
            }
        }

    }

    fun bind(itemTask: ItemTask) {
        checkBoxTask.text = itemTask.description
        checkBoxTask.setChecked(itemTask.fait)
    }
}

interface ActionListener {
    fun onItemClicked(taskPosition: Int)
}
}
```

4. Persistance de données et Sérialisation/Désérialisation en JSON

JSON est un type de data de forme chaîne de caractère.

```
jsonProfil= Gson().toJson(myProfil)
```

```
val myProfil: Profil = Gson().fromJson(jsonProfil, Profil::class.java)
```

Donc en utilisant la première commande ci-dessus, nous pouvons réaliser la transformation de notre type Profil à la forme Json, ainsi la sérialisation, et en utilisant la seconde commande ci-dessous, nous pouvons réaliser la transformation inverse, c'est ainsi que la désérialisation.

On a implementé une class Preference qui est prend encharge de tous les getters et setters de tous paramètres partagés. En utilisant la phrase ci-dessous:

```
var jsonProfil by Preference(this,pseudo,"jsonProfilDefault")
```

On a déclarer la propriété de prdélégation de variable jsonProfil.

Parmi les plusieurs exemples de réalisation de la partage de donnée, en fait nous avons utilisé les fichiers Json pour stocker ces données intérieures en utilisant la commande ci-dessous, c'est ainsi que la persistence de données en préférences.

Conclusion

Nous avons réussi à construire une application TodoList qui peut ajouter des listes de choses, nous pouvons également générer les informations de plusieurs utilisateurs. Pendant cette période de travail, on a pu bien réviser les cours sur le TP qui nous a appris comment utiliser les Views, les Adapters, les Layouts etc., et nous avons aussi acquis pas mal de connaissances de pratique pour maîtriser Android Studio.

Perspectives

- Donner un attribut de temps pour lier les choses au calendrier.
- Ajouter une fonction chercher pour rapidement localiser les items ce qu'on veut trouver.
- Completer l'activity **SettingActivity** et les fragments
- Ajouter des longClickedListener sur les List et Item pour les supprimer.