PROGRAMMATION MOBILE ANDROID

TD 2

INTERFACE AVANCÉE

charles.meunier@u-bourgogne.fr

OBJECTIFS

- Manipuler les événements
- Utiliser les Intents pour démarrer une nouvelle activité
- Créer une liste personnalisée

EXERCICE 1 - INTERFACE

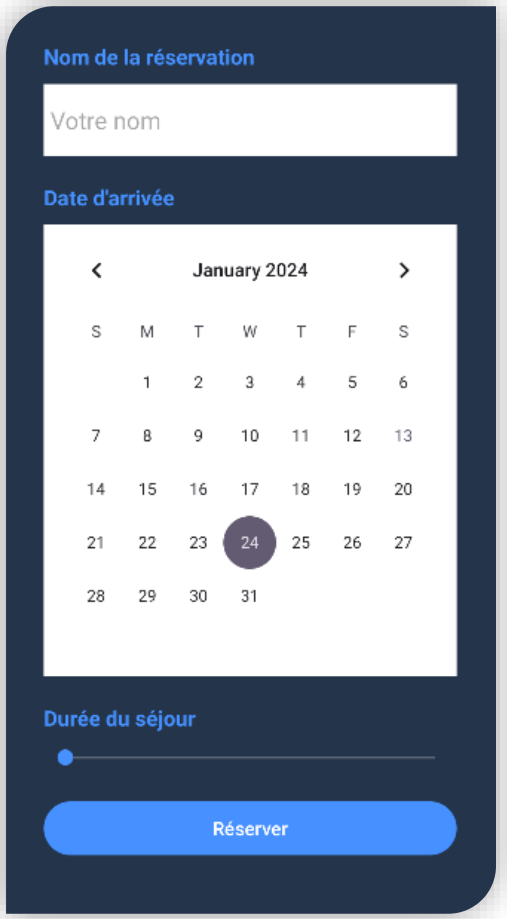
Nouveau projet

➢ Créez un nouveau projet selon la méthode vue lors du TD 1.

Interface

➢ Reproduisez l'interface ci-contre à l'aide des composants suivants :

- TextView pour les libellés
- PlainText pour la saisie du nom
- CalendarView pour le calendrier
- SeekBar pour la durée du séjour
- Button pour le bouton



Note :
Le fichier ***`res/values/colors.xml`*** *vous permet de définir les couleurs de votre thème.*
Créez une teinte ***`primary`*** *ayant la valeur* ***`#4691FF`*** *et appliquez là à tous les éléments bleus.*

Cliquez ici pour sélectionner une teinte prédéfinie

➢ Démarrez l'application et vérifiez que tout est bien en place.

EXERCICE 2 - ÉVÉNEMENT ONDATECHANGE

Date de réservation

➢ Ajoutez à la classe `MainActivity` un attribut `bookingDate` de type `String`.

```
private var bookingDate: String = ""
```

Récupération de la date

Lorsque l'utilisateur sélectionne une date à l'aide du `CalendarView`, un événement `onDateChange` est déclenché.

➢ Ajoutez la méthode ci-contre à la classe `MainActivity`

➢ Remplacez le commentaire par le code adéquat, le but étant de conserver la date au format `jj/mm/aaaa`

➢ N'oubliez pas d'appelez la méthode `initCalendarEvent` à la fin de la méthode `onCreate` de `MainActivity`

```
private fun initCalendarEvent()
{
    val calendar = findViewById<CalendarView>(R.id.calendarView)

    calendar.setOnDateChangeListener(object: CalendarView.OnDateChangeListener {
        override fun onSelectedDayChange(view: CalendarView, year: Int, month: Int, dayOfMonth: Int)
        {
            // Affecter la nouvelle date à l'attribut bookingDate
        }
    })
}
```

➢ Ajoutez une méthode `updateBookingDateLabel` qui affichera la date sélectionnée à la fin du libellé "Date d'arrivée".

EXERCICE 3 - ÉVÉNEMENT ONSEEKBARCHANGE

Durée du séjour

➢ Ajoutez à la classe `MainActivity` un attribut `stayDuration` de type `Int`.

Récupération de la durée

Lorsque l'utilisateur modifie la position du curseur de la SeekBar, un événement `onSeekBarChange` est déclenché.

➢ Ajoutez la méthode ci-contre à la classe `MainActivity`

➢ Placez le curseur à l'endroit désigné par la flèche verte et appuyez sur les touches `Ctrl+Espace`.

➢ Sélectionnez les trois fonctions proposées à l'override.

➢ Modifiez le code de `onProgressChanged` pour récupérer la valeur **`progress`** de la SeekBar.

```
private fun initSeekBarEvent()
{
    val seekbar = findViewById<SeekBar>(R.id.seekBar)

    seekbar.setOnSeekBarChangeListener(object: SeekBar.OnSeekBarChangeListener
    {
        ➡
    })
}
```

Note :
Ne laissez pas de TODO("Not yet implemented") dans le code. Ils provoqueront une exception qui arrêtera l'application.

➢ Ajoutez une méthode `updateStayDurationLabel` qui affichera la durée du séjour à la fin du libellé "Durée du séjour".

EXERCICE 4 - INTENT

Nouvelle activité

- Ajoutez une nouvelle activité (de type `Empty Views Activity`) au projet

Interface de la seconde activité

- Placez un simple TextView au centre de l'écran de cette activité.

Démarrage de la seconde activité

- Ajoutez un événement `onClick` sur le bouton de la première activité qui démarrera la seconde activité à l'aide d'un Intent explicite (voir le cours).

- Testez le bon fonctionnement.

EXERCICE 5 - INTENT AVEC PASSAGE DE DONNÉES

Transmission des données

- Modifiez l'Intent précédent de manière à transmettre les données saisies par l'utilisateur de l'activité 1 vers l'activité 2 (voir le cours).

Récupération des données

- Ajoutez une méthode `extractBookingDataFromIntent` à la classe de la seconde activité. Cette méthode se chargera d'extraire les données transmises via l'Intent.

Affichage des données

- Ajoutez une méthode `displayBookingData` à la classe de la seconde activité. Cette méthode mettra à jour le TextView de la seconde activité afin d'afficher les informations de la réservation.

EXERCICE 6 - LISTE PERSONNALISÉE

Nouveau projet

➢ Créez un nouveau projet comme vu lors du TD 1.

ListView

➢ Placez une ListView (rubrique Legacy) et assurez vous qu'elle couvre bien toute la surface de l'écran.

Fonctionnement

Une ListView a besoin de trois éléments pour fonctionner :

- La liste des données à afficher
- Un layout décrivant l'apparence d'un élément de la ListView
- Un adaptateur permettant de faire le lien entre les données et le layout.

EXERCICE 6.1 - LES DONNÉES

Classe Contact

➢ Ajoutez au projet une nouvelle classe `Contact`.

Data Class

En Kotlin, il est possible de faire des `data class` qui ont un rôle similaire à celui des `struct` en C.

*Note : Dans l'exemple ci-contre, chaque attribut est défini avec le mot clé **`val`**. De cette façon, les attributs sont initialisés à la création de l'instance et ne peuvent plus être modifiés par la suite.*

➢ Créez la data classe `Contact` comme présentée ci-contre.

```
data class Contact(
    val lastName: String,
    val firstName: String,
    val phoneNumber: String,
    val photo: Int
)
```

Les contacts

➢ Ajoutez à la classe `MainActivity` un attribut contacts de type `ArrayList<Contact>`

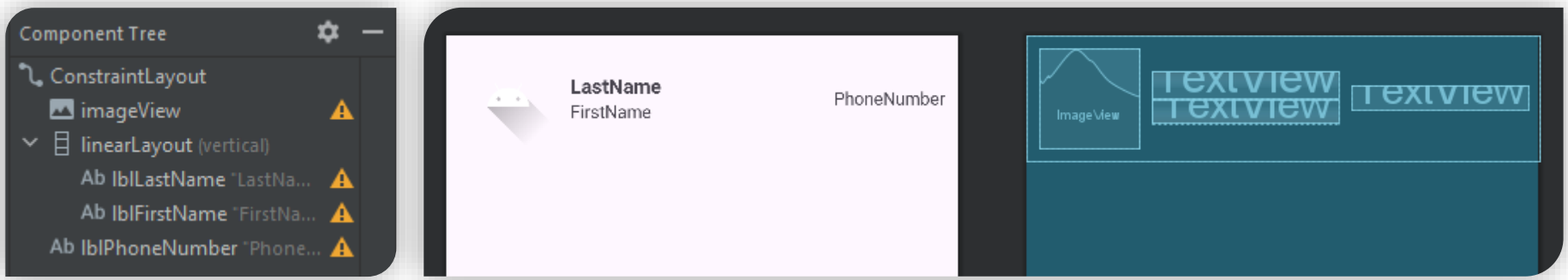
➢ Ajoutez une fonction `initContacts` qui remplit le tableau contacts avec les informations suivantes :

- Meunier, Charles, 06 06 06 06 06, 1
- Serier, Karine, 07 07 07 07 07, 2
- Heyrman, Barthélémy, 08 08 08 08 08, 3

EXERCICE 6.2 - LE LAYOUT

Nouveau layout

- Ajoutez au projet un nouveau layout nommé contacts_list_item
- Ajoutez les éléments suivants en respectant les id présents dans l'arborescence des composants :



EXERCICE 6.3 - L'ADAPTATEUR

Nouvelle classe

➢ Ajoutez au projet une nouvelle classe `ContactAdapter` qui héritera de BaseAdapter et aura trois attributs initialisés par le constructeur :

- context de type Context (passé en paramètre au constructeur)
- dataSource de type ArrayList<Contact> (passé en paramètre au constructeur)
- inflater de type LayoutInflater

```
class ContactAdapter(
    private val context: Context,
    private val dataSource: ArrayList<Contact>
) : BaseAdapter()
{
    private val inflater: LayoutInflater =
        context.getSystemService(Context.LAYOUT_INFLATER_SERVICE) as LayoutInflater


}
```

Notes :

- ***`context`*** *représente l'activité courante,*
- ***`dataSource`*** *est la collection de contacts à afficher,*
- ***`inflater`*** *permet de générer dynamiquement un layout pour chaque élément du ListView*

EXERCICE 6.3 - L'ADAPTATEUR

Implémenter les fonctions de l'adaptateur

➢ Redéfinissez les quatre fonctions issues de la classe abstraite `BaseAdpter` :

- `getItemId` qui retourne l'ID de l'élément dont la position est indiquée. Nos contacts n'ayant pas d'attribut ID, vous retournerez la position demandée sous la forme d'un entier long (`toLong`).
- `getItem` qui retourne le contact dont la position est indiquée.
- `getCount` qui retourne la taille de la collection de données à afficher.
- `getView` qui retourne le layout à afficher pour la position indiquée :

```
override fun getView(position: Int, convertView: View?, parent: ViewGroup?): View {
    val rowView = inflater.inflate(R.layout.contacts_list_item, parent, false)

    // Code nécessaire pour affecter les données du contact n°position aux
    // différents champs du layout.
    // Vous ne vous occuperez pas de la photo dans un premier temps.

    return rowView
}
```

EXERCICE 6.4 - AFFICHAGE DE LA LISTE

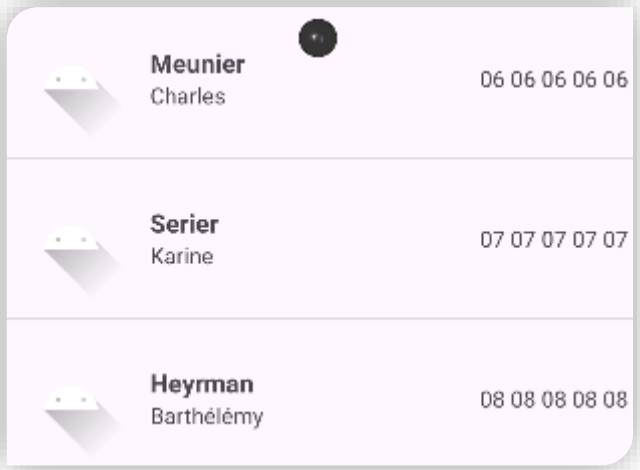
Retour à MainActivity

- Ajoutez la fonction initContactsListView à la classe MainActivity :

Note : la fonction initContactsListView s'occupe simplement d'indiquer à la ListView quel adaptateur utiliser et avec quelles données. N'oubliez pas de l'appeler depuis la méthode onCreate...

```
private fun initContactsListView()
{
    val listView = findViewById<ListView>(R.id.lstContacts)
    listView.adapter = ContactAdapter(this, contacts)
}
```

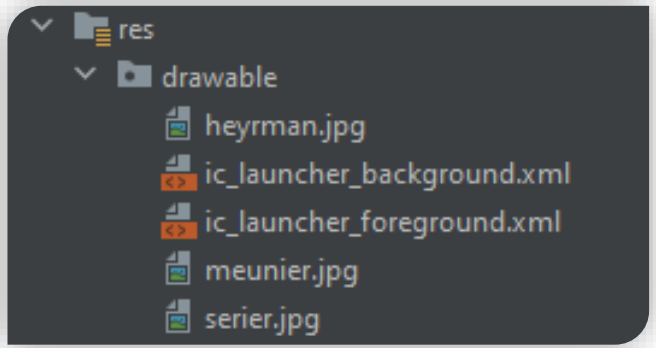
- Testez votre application. Vous devriez obtenir le résultat suivant :



EXERCICE 6.5 - ET LES PHOTOS ?

Téléchargement des photos

- Téléchargez les photos de vos contacts depuis ce lien :
  https://www.lamarmotte.info/wp-content/uploads/2024/01/photos.zip
- Placez les images téléchargées dans le dossier `res/drawable` de votre projet :

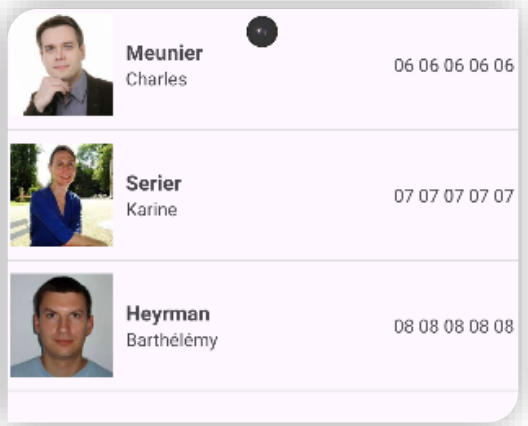


Ajout des photos aux contacts

- Dans la fonction `initContacts`, remplacez le dernier entier par `R.drawable.[nom_image_sans_extension]`

Modification de l'adaptateur

- Modifiez la fonction `getView` pour chargez la photo du contact dans l'ImageView du layout (`setImageResource`).
- Testez le résultat qui devrait s'approcher de l'image ci-contre :



EXERCICE 6.6 - BONUS

Appeler les contacts

➢ Ajoutez la fonctionnalité permettant d'appeler un contact lorsque l'on clique dessus dans la liste.

Piste 1 : Gérer l'événement click sur un élément de la liste

```
listView.setOnItemClickListener { parent, view, position, id ->

    // Appeler le contact n°position
}
```

Piste 2 : Passer un appel

```
val intent = Intent(Intent.ACTION_DIAL)
intent.data = Uri.parse("tel:xxxxxxxxxxxx")
```