TP n°3
Menu d'Options & Intents

Objectif

L'objectif de ce TP est de vous permettre de manipuler les intents explicites avec transmission de données ainsi que les intents implicites pour l'envoi des SMS et pour le passage des appels téléphoniques. Vous allez aussi vous baser sur les menus d'options comme outil de navigation.

Sous Android Studio, créez un nouveau projet Android. Choisissez "**Basic Activity**", puis cliquez sur « Next ». Remplissez le formulaire avec les informations nécessaires suivantes : Nom « TP3X » où *X* est votre nom, Nom du package « eniso.gte2.tp3X », Kotlin comme langage et ciblez une API de votre choix comme version minimale du SDK. Une fois c'est fait, cliquez sur « Finish ».

NB :
*Suite au choix de "Basic Activity", vous remarquerez la présence de quatre (4) fichiers xml sous le dossier **res/layout** : "**activity_main.xml**" qui inclut un Toolbar, le layout "**content_main.xml**" et un FloatingActionButton (bouton avec image d'enveloppe).*
*A son tour, "**content_main.xml**" contient un fragment pouvant afficher l'un des deux fragments définis par "**fragment_first.xml**" et "**fragment_second.xml**".*

Dans ce TP, nous n'allons pas nous focaliser sur tous ces détails.

*De plus, vous remarquerez la présence d'un fichier xml dans le dossier **res/menu**.*

- *"**menu_main.xml**" qui représente un menu d'options accessible via la barre d'actions de l'activité "MainActivity".*

Partie I : Menu d'options

a- Suite au clic sur la barre d'actions (les trois points en haut à droite), un menu d'options composé de trois choix s'affichera : ***SMS***, ***Appels*** et ***Quitter***.

Si on clique sur "SMS", un sous-menu s'ouvre et affiche deux options : "***SMS Normal***" et "***SMS d'Urgence***".
De même, si on clique sur "Appels", un sous-menu s'ouvre et affiche deux options : "***Appel Normal***" et "***Appel d'Urgence***".
Sinon, si on clique sur "***Quitter***", on quitte l'application grâce à la méthode **finish()**.

NB :

- Ce menu est à développer dans le fichier *res/menu/main.xml* par des simples « drag and drop » de *Menu Items* et de *Menus* de la partie "**Palette**" vers la partie "**Component Tree**", comme le montre la figure 1.
- Il doit donner un affichage presque équivalent à celui montré par la figure 2 (ici, l'option des appels n'est pas montrée).

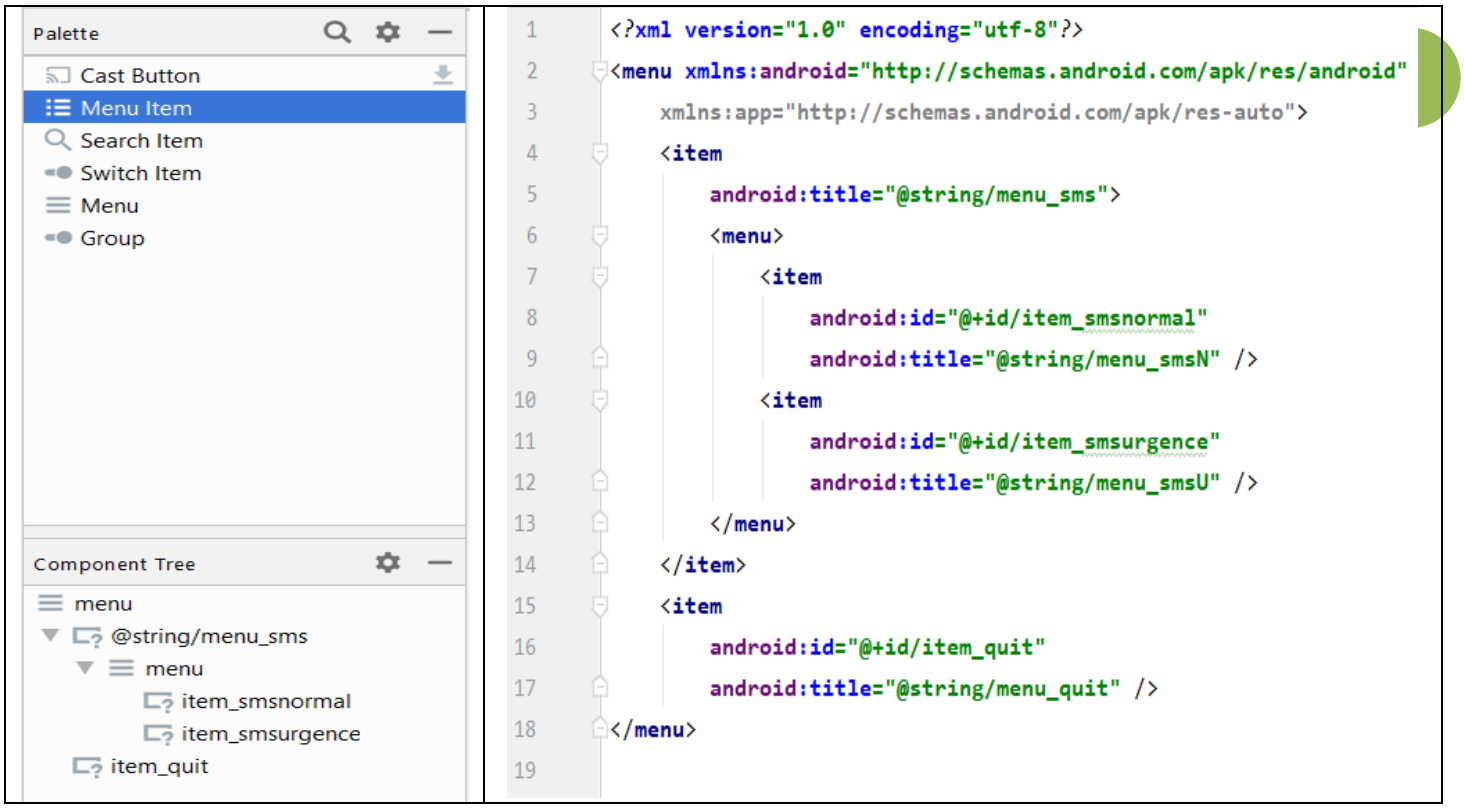


Fig 1. Méthode de création du menu (à gauche) et code xml généré (à droite).

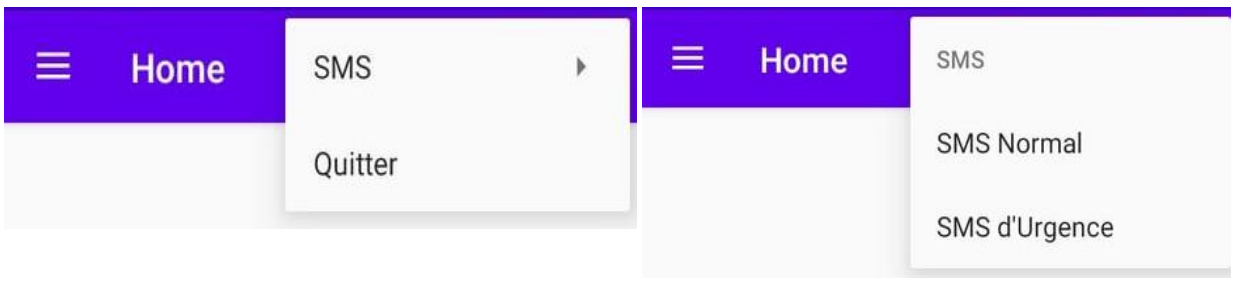
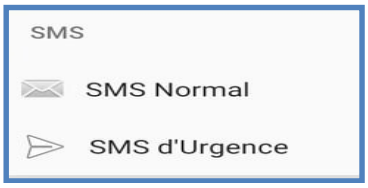


Fig 2. Menu (à gauche) et Sous-Menu (à droite).

NB :
Vous pouvez ajouter des icônes aux items du sous-menu. Pour ce faire, il faut attribuer une icône à la propriété « ***android:icon*** », comme par exemple (icône pour l'item 2):

```
android:icon="@android:drawable/ic_menu_send"
```



b- Dans le fichier "MainActivity.kt", il y a deux nouvelles méthodes :

- **onCreateOptionsMenu()** qui se lance dès qu'on clique sur les 3 points de la barre d'action pour charger le menu et l'ajouter à l'activité.

- **onOptionsItemSelected()** qui servira à gérer le choix d'un élément du menu. C'est dans cette méthode que vous allez coder les rôles des items "**SMS Normal**", "**SMS d'Urgence**", "**Appel Normal**", "**Appel d'Urgence**" et "**Quitter**" du menu.

Partie II : Gestion des SMS

a- Si on choisit l'option "*SMS d'Urgence*", on envoie directement un SMS dont le message et le numéro de téléphone sont à lire à partir des ressources chaines de caractères par la méthode :

```
resources.getString(R.string.....)
```

Pour que le SMS soit envoyé directement sans passer par l'application du téléphone, utilisez un objet ***SmsManager*** comme ceci est montré par le code ci-dessous :

```
val smsManager:SmsManager = SmsManager.getDefault()
smsManager.sendTextMessage(phone_number, scAddress: null, message, sentIntent: null, deliveryIntent: null)
```

NB :
- *Ceci nécessite une permission SEND_SMS qu'il faudra rajouter au fichier Manifest.*
- *« phone_number » et « message » sont à remplacer par vos données stockées parmi les ressources chaines de caractères.*
- *Activez la permission à la main sur votre téléphone.*

b- Si on choisit l'option "*SMS Normal*", on passe à une activité de type *Empty Activity* dite "**WriteSMS**" ayant :

- Un 1er *EditText* prêt à contenir un numéro de téléphone.
- Un 2ème *EditText* prêt à contenir un texte multi-ligne.
- Un bouton « *Confirmer SMS* » qui lancera une activité « **ConfirmSMS** » (*Empty Activity*) en lui passant les 2 informations saisies (numéro et texte).

c- 0L'activité « **ConfirmSMS** » a une interface pareille à celle de « **WriteSMS** », juste au lieu du bouton « *Confirmer SMS* », elle possède deux boutons « *Envoyer* » et « *Annuler* ».

- Elle récupère les deux chaines saisies dans « **WriteSMS** » et les affiche dans les zones appropriées.
- L'utilisateur pourrait éventuellement, rectifier le numéro de téléphone et/ou le message, si nécessaire.
- L'utilisateur pourrait cliquer sur l'un des deux boutons :
  - S'il clique sur le bouton d'envoi, ***un intent implicite*** est indispensable (dans ce cas, vous devez passer par l'application de messagerie).
  - S'il annule l'envoi, il y aura retour à l'activité « **WriteSMS** » et un toast signalant l'annulation sera affiché.

Ainsi, il vous est demandé de gérer les intents, le transfert d'informations et le retour du résultat en cas d'annulation ainsi que l'envoi du SMS en cas de confirmation.

NB :

– *Utilisez l'action "ACTION_SENDTO" avec une URI ayant le schéma "smsto".*

exp. ***"smsto:97xxxxxx?body=Au secours !!!"***

Cet intent va lancer l'application d'envoi des SMS de votre Smartphone ou émulateur.

– *Pour les tests, il est recommandé d'utiliser deux Smartphones, sinon il faut lancer deux émulateurs.*

- *Pour les émulateurs, il faut spécifier le numéro de l'un d'eux comme destinataire du SMS. Par défaut, le 1er a comme numéro « 5554 » alors que le second a comme numéro « 5556 ».*

Partie III : Gestion des Appels

a- Si on choisit l'option "*Appel d'urgence*", on effectue un appel direct à un numéro de téléphone à lire à partir des ressources chaines de caractères.

> ***NB :***
> *Ici, vous devez utiliser l'action ACTION_CALL qui passe directement l'appel. Dans ce cas, l'intent implicite nécessite la permission « CALL_PHONE » qu'il faudra rajouter au fichier Manifest.*

b- Si on choisit l'option "*Appel Normal*", on passe à "**CallActivity**" (*Empty Activity*).
Cette dernière doit contenir un EditText servant à saisir un numéro de téléphone et un bouton servant à passer l'appel via un **intent implicite**.

> ***NB :***
> *Ici, vous pouvez utiliser l'une des actions suivantes :*
> – *ACTION_CALL qui passe directement l'appel.*
> – *ACTION_DIAL qui lance le composeur téléphonique. Dans ce cas, aucune permission n'est requise.*

Partie IV : Gestion des autorisations au moment de l'exécution

Généralement, on ne demande pas aux utilisateurs d'accéder aux paramètres de l'application pour lui donner l'autorisation d'envoi des SMS ou de lancement des appels téléphoniques directs, etc. Plutôt, on doit les avertir par une simple boite de dialogue les informant de l'autorisation requise, puis c'est à eux de décider d'accepter ou non.

La figure 3 donne un exemple de code à écrire dans une activité pour demander l'autorisation de passage des appels téléphoniques (il peut changer si on l'écrit dans un fragment).

```
private fun appel(): Unit {
if (ActivityCompat.checkSelfPermission( context: this, Manifest.permission.CALL_PHONE) != PackageManager.PERMISSION_GRANTED)
{
    Toast.makeText(applicationContext,
            text: "Vous devez activer l'autorisation pour passer des appels téléphoniques",
                Toast.LENGTH_LONG).show()
    ActivityCompat.requestPermissions( activity: this, arrayOf(Manifest.permission.CALL_PHONE),  requestCode: 1)
} else
     {   val intent = Intent(Intent.ACTION_CALL, Uri.parse( uriString: "tel:12345678" ))
         startActivity(intent)
     }
}
override fun onRequestPermissionsResult(requestCode: Int, permissions: Array<out String>, grantResults: IntArray)
{
    if (requestCode == 1) {
        if (grantResults.isNotEmpty() && grantResults[0] == PackageManager.PERMISSION_GRANTED)
            appel()
        else Toast.makeText(applicationContext,  text: "Accès non autorisé aux appels téléphoniques", Toast.LENGTH_LONG).show()
    }
}
```

Fig 3. Vérification et demande de permission pour les appels téléphoniques.