Les Application mobiles

Silem Abdelheq

1.0

Mars 2023

Table des matières

Objectifs

la finalité de cette matière est d'apporter à l'étudiant des connaissances en matière de développement d'application et système informatique dans des environnements mobiles. Avec l'arrivée des smartphones les applications mobiles sont omniprésents que l'on soit client (BtoC), fournisseur (BtoB) ou collaborateur (BtoE). Le but de ce cours est aussi d'apprendre la programmation sous Android, sa plate-forme de développement et les spécificités du développement embarqué sur smartphone[

I Chapitre 3 : Activités et ressources

Pour créer une application Android, on utilise différents composants tels que des activités, des vues, des fragments, des services et des fournisseurs de contenu. Ces composants sont organisés et gérés par le système d'exploitation Android. Ce chapitre présente ces différents composants et leurs interactions au sein d'une application Android.

1. Objectives

Fondamental

L'objectif de ce chapitre est de fournir une vue d'ensemble des concepts clés et des meilleures pratiques liés aux activités, à la classe R, au fichier manifeste et aux ressources dans le développement Android :

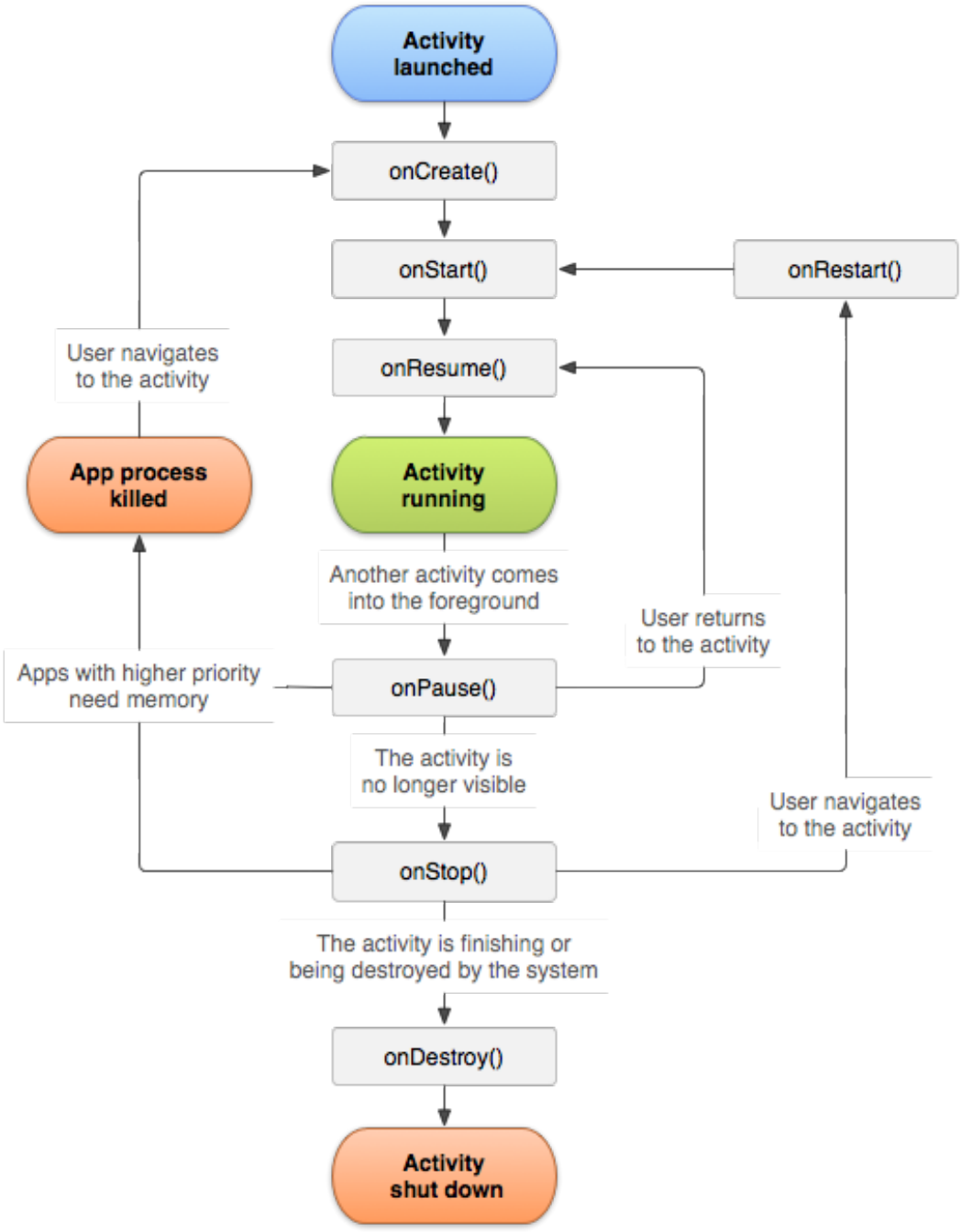
- Expliquer le but, la fonctionnalité, la structure et l'organisation des activités dans une application Android.
- Démontrer comment la création, le lancement et la gestion du cycle de vie des activités, y compris les meilleures pratiques pour une performance optimale.
- Fournir une vue d'ensemble du fichier manifeste dans le développement Android, notamment son objectif, sa fonctionnalité et les différents éléments qui peuvent être inclus.
- Décrire comment le fichier est utilisé par le système Android pour identifier et lancer l'application, et fournir les bonnes habitudes pour la gestion des permissions et des caractéristiques de l'application dans le fichier.
- Expliquer l'importance des ressources dans le développement Android, incluant leur but et les différents types utilisés dans une application.
- Démontrer comment organiser et accéder aux ressources de manière efficace pour une performance optimale.
- Expliquer la classe R dans le développement Android, son rôle et comment elle est générée dans le processus de construction d'une application.
- Démontrer comment utiliser la classe R pour accéder efficacement aux différents types de ressources dans une application Android.

2. Introduction aux activités

Les activités sont les éléments de construction d'une application Android. Elles représentent des interfaces individuelles ou des composants d'interface utilisateur d'une application avec lesquels les utilisateurs peuvent interagir. Chaque activité a généralement un objectif ou une fonction spécifique, comme l'affichage d'une liste d'éléments, la modification des informations de l'utilisateur ou la lecture d'une vidéo. Les activités sont organisées et gérées par le système Android afin d'offrir à l'utilisateur une expérience transparente.

2.1. Cycle de vie d'une activité

Le cycle de vie d'une activité Android comporte plusieurs étapes qui correspondent à différents événements qui se produisent pendant l'exécution de l'activité . Le cycle de vie de l'activité commence lorsqu'elle est lancée par l'utilisateur et se termine lorsque l'activité est fermée ou détruite par le système d'exploitation. Chaque étape du cycle de vie d'une activité Android correspond à l'appel d'une méthode spécifique que l'on peut surcharger dans notre code pour définir le comportement correspondant à cette étape.



Dans Android, les activités sont gérées à l'aide d'une pile, ce qui signifie que l'application en cours est la première de la pile. Une activité peut se trouver dans l'un des quatre états suivants, également connus sous le nom d'états d'activité :

- Actif ou en cours d'exécution : L'activité est au premier plan, c'est-à-dire qu'elle se trouve en haut de la pile d'activités et qu'elle est en train d'interagir avec l'utilisateur.
- En pause : L'activité est toujours visible pour l'utilisateur mais n'est plus au premier plan. Cela se produit lorsqu'une autre activité couvre partiellement l'activité en cours ou lorsque l'utilisateur appuie sur le bouton Accueil.
- Arrêtée : L'activité n'est plus visible pour l'utilisateur et se trouve en arrière-plan. Cela se produit lorsque l'utilisateur s'éloigne de l'activité ou lorsque l'activité n'est plus nécessaire mais n'a pas encore été détruite.
- Détruite : L'activité a été retirée de la pile d'activités et ses ressources ont été libérées. Cela se produit lorsque l'activité se termine ou qu'elle est tuée par le système pour libérer des ressources.

En général, le passage entre les états de cycle de vie d'une activité est assuré par les méthodes de callback suivantes:

- onCreate() - Cette méthode est appelée lors de la création de l'activité. Elle est généralement utilisée pour effectuer les tâches initiales de configuration de l'activité, telles que la configuration de l'interface utilisateur, l'initialisation des variables ou le chargement des données à partir d'une base de données. C'est également à ce moment-là que vous appelez setContentView() pour définir la mise en page de l'activité. Afin de surcharger cette méthode, il suffit d'ajouter le code ci-dessous à votre classe d'activité et d'implémenter la logique requise.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // perform necessary setup tasks here
}
```

- onStart() - Cette méthode est appelée lorsque l'activité devient visible pour l'utilisateur. Elle est souvent utilisée pour effectuer les tâches de configuration nécessaires qui doivent se produire lorsque l'activité est affichée pour la première fois. Par exemple, vous pouvez utiliser cette méthode pour démarrer une animation ou vous connecter à un service réseau. Pour surcharger cette méthode, ajoutez ce code à votre classe d'activité et mettez en œuvre la logique requise.

```
@Override
protected void onStart() {
    super.onStart();
    // start background tasks or connect to services here
}
```

- onResume() - Cette méthode est appelée lorsque l'activité est au premier plan et qu'elle a le focus. Elle est souvent utilisée pour reprendre les opérations qui ont été mises en pause dans onPause(), telles que le démarrage d'animations, la lecture de médias ou la création de récepteurs pour les entrées de l'utilisateur. C'est également à ce moment-là que l'on démarre les threads d'arrière-plan nécessaires à la mise à jour de l'interface utilisateur ou à l'exécution d'autres tâches. Ajoutez ce code pour surcharger cette méthode.

```
@Override
protected void onResume() {
    super.onResume();
    // resume animations or update UI here
```

```
}
```

- onPause() - Cette méthode est appelée lorsque l'activité perd le focus et n'est plus au premier plan. Elle est souvent utilisée pour mettre en pause les opérations en cours qui doivent être arrêtées lorsque l'activité n'est plus visible, comme l'arrêt des animations, la mise en pause de la lecture des médias ou la suppression des récepteurs d'entrée de l'utilisateur. Ajoutez ce code pour surcharger cette méthode.

```
@Override
protected void onPause() {
    super.onPause();
    // stop animations or release resources here
}
```

- onStop() - Cette méthode est appelée lorsque l'activité n'est plus visible par l'utilisateur. Elle est souvent utilisée pour arrêter les threads ou les services d'arrière-plan qui ont été démarrés dans onStart(), ainsi que pour libérer les ressources système qui ne sont plus nécessaires. Par exemple, vous pouvez utiliser cette méthode pour arrêter un service réseau ou libérer une ressource caméra. Ajoutez ce code pour surcharger cette méthode.

```
@Override
protected void onStop() {
    super.onStop();
    // release resources or stop background tasks here
}
```

- onRestart() - Cette méthode est appelée lorsque l'activité est redémarrée après avoir été arrêtée. Elle est souvent utilisée pour effectuer les tâches de configuration nécessaires au redémarrage de l'activité, telles que la réinitialisation de l'interface utilisateur ou le rechargement des données d'une base de données. Ajoutez ce code pour surcharger cette méthode.

```
@Override
protected void onRestart() {
    super.onRestart();
    // reset UI components or reload data here
}
```

- onDestroy() - Cette méthode est appelée lorsque l'activité est sur le point d'être détruite. Elle est souvent utilisée pour libérer les ressources qui ne sont plus nécessaires, comme la fermeture des connexions à la base de données ou l'arrêt des services d'arrière-plan. C'est également à ce moment-là que vous effectuez les dernières tâches de nettoyage nécessaires avant la destruction de l'activité, comme la libération des ressources système ou la destruction des récepteurs de diffusion. Ajoutez ce code pour surcharger cette méthode.

```
@Override
protected void onDestroy() {
    super.onDestroy();
    // release any remaining resources or unregister listeners here
}
```

Remarque

- Les méthodes onPause() et onResume() sont très importantes pour la gestion des ressources du système, telles que l'autonomie de la batterie et l'utilisation de la mémoire. Par exemple, vous devez toujours libérer les ressources dont vous n'avez plus besoin dans la méthode onPause() afin d'économiser les ressources du système, et les réacquérir dans la méthode onResume() lorsque l'activité reprend.
- Il est également important de tenir compte de l'impact des appels du cycle de vie sur l'expérience de l'utilisateur. Par exemple, des opérations de longue durée dans onResume() peuvent entraîner un manque de réactivité de l'application et donc une mauvaise expérience utilisateur. Pour éviter cela, vous devez toujours vous efforcer de maintenir votre code aussi efficace que possible et éviter d'effectuer des opérations susceptibles de provoquer des retards ou des blocages.
- Il est intéressant de noter que les rappels du cycle de vie Android sont un élément clé de l'architecture globale du développement d'applications Android. En comprenant et en utilisant efficacement ces rappels, vous pouvez créer des applications efficaces, réactives et offrant une expérience utilisateur de haute qualité.

Attention

- Soyez conscient des risques liés à l'utilisation d'opérations de longue durée dans onStart() ou onStop(), car ces méthodes peuvent être appelées fréquemment au cours de la durée de vie d'une activité. Par exemple, le démarrage d'une opération réseau de longue durée dans onStart() peut entraîner l'envoi de plusieurs requêtes réseau si l'activité est arrêtée et démarrée à plusieurs reprises
- Soyez conscient du risque de bugs ou de comportements inattendus lorsque vous utilisez des bibliothèques complexes ou des bibliothèques tierces qui peuvent ne pas être entièrement compatibles avec le cycle de vie d'Android. Veillez à lire attentivement la documentation et à effectuer des tests complets pour vous assurer que votre application se comporte correctement.

2.2. Navigation entre les activités

Dans Android, une activité est un écran autonome doté d'une interface utilisateur. Pour créer une application fonctionnelle, il peut être nécessaire d'utiliser plusieurs activités ou une seule activité avec plusieurs fragments (nous les verrons dans les chapitres suivants). Lorsque vous utilisez plusieurs activités, vous devez naviguer entre elles pour fournir les fonctionnalités requises. Cette navigation s'effectue généralement à l'aide d'intents.

Les intentions sont des objets qui vous permettent de spécifier une action à effectuer, telle que le démarrage d'une nouvelle activité. Il existe deux types d'intentions : explicites et implicites. Les intentions explicites sont utilisées pour spécifier la classe d'activité cible et démarrer une nouvelle activité au sein de la même application. Les intentions implicites sont utilisées pour effectuer une action qui peut être gérée par une activité dans une application différente, telle que l'ouverture d'une page web dans le navigateur préféré de l'utilisateur.

Démarrer une nouvelle activité en utilisant une intent explicite et la méthode startActivity

Pour démarrer une nouvelle activité à l'aide d'une intention explicite, vous devez d'abord créer une instance de la classe Intent et passer le contexte de l'activité en cours et la classe de l'activité cible en tant que paramètres. Ensuite, vous pouvez appeler la méthode startActivity() sur l'activité en cours et passer l'intention en paramètre. Voici un exemple de code :

```
Intent intent = new Intent(CurrentActivity.this, TargetActivity.class);
startActivity(intent);
```

Démarrer une nouvelle activité en utilisant une intent explicite et la méthode startActivityForResult

Si vous devez obtenir un résultat à partir d'une activité, vous pouvez utiliser la méthode startActivityForResult() au lieu de startActivity(). Cette méthode vous permet de transmettre une intention et un code de requête à l'activité cible, et l'activité cible renvoie un résultat à l'activité principale en appelant setResult() et finish(). Voici un exemple de code :

Code d'activité principale

```
Intent intent = new Intent(CurrentActivity.this, TargetActivity.class);
int REQUEST_CODE= 1;
startActivityForResult(intent, REQUEST_CODE);

// Override the onActivityResult() method to handle the result
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    if (requestCode == REQUEST_CODE) {
        if (resultCode == RESULT_OK) {
            // Handle the successful result
        } else if (resultCode == RESULT_CANCELED) {
            // Handle the canceled result
        }
    }
}
```

Code d'activité cible

```
.....
// perform necessary logic here
.....
Intent resultIntent = new Intent();
resultIntent.putExtra("parameter name", parameter value);
setResult(Activity.RESULT_OK, resultIntent);
finish();
```

Démarrer une nouvelle activité à l'aide de intent implicites

Pour démarrer une nouvelle activité à l'aide d'une intent implicite, vous devez créer une instance de la classe Intent et spécifier l'action à effectuer ainsi que toute donnée supplémentaire. Par exemple, pour ouvrir une page web dans le navigateur, vous pouvez créer une intention avec l'action ACTION_VIEW et l'URL comme données. Ensuite, vous pouvez appeler la méthode startActivity() sur l'activité en cours et passer l'intention en paramètre. Voici un exemple de code :

```
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("https://www.example.com"));
startActivity(intent);

```

Autres exemples d'actions d'intentions implicites :

- ACTION_EDIT : Permet de modifier les données identifiées par l'intention.

- ACTION_SEND : permet d'envoyer des données à une autre application, telle qu'une application de messagerie ou de courrier électronique.
- ACTION_PICK : Permet de sélectionner un élément dans une liste, par exemple un contact ou une photo.
- ACTION_SEARCH : Permet d'effectuer une recherche sur les données identifiées par l'intention.
- ACTION_WEB_SEARCH : Permet d'effectuer une recherche sur le web en utilisant la requête donnée.
- ACTION_DIAL : Permet de composer un numéro de téléphone.
- ACTION_CALL : Permet de lancer un appel téléphonique.
- ACTION_GET_CONTENT : Permet de récupérer le contenu d'un fichier ou d'un document.
- ACTION_SENDTO : Permet d'envoyer un message à un numéro de téléphone ou à une adresse électronique spécifique.
- ACTION_VIEW_DOWNLOADS : Permet d'afficher la liste des téléchargements.
- ACTION_SYNC : Permet d'effectuer une opération de synchronisation.
- ACTION_INSTALL_PACKAGE : Permet d'installer un paquet.
- ACTION_UNINSTALL_PACKAGE : Permet de désinstaller un paquet.

Bouton de retour

Lorsque l'utilisateur appuie sur le bouton Précédent, l'activité en cours est terminée et retirée de la pile d'activités, et l'activité précédente de la pile est reprise. Il s'agit du comportement par défaut du bouton Précédent, et vous n'avez pas besoin d'écrire de code pour le mettre en œuvre. Si vous souhaitez personnaliser ce comportement, vous pouvez surcharger la méthode onBackPressed dans votre activité

Il est important de noter que lorsque vous démarrez une nouvelle activité, l'activité précédente est mise en pause mais reste dans la pile d'activités. Vous pouvez utiliser le bouton "retour" pour revenir à l'activité précédente ou appeler la méthode "finish()" de l'activité en cours pour la retirer de la pile.

2.3. Méthodes de la classe activité

Relative à	Déclaration de méthode	Description
Cycle de vie	onCreate(Bundle)	Appelé lorsque l'activité est créée.
	onStart()	Appelé lorsque l'activité est sur le point de devenir visible.
	onResume()	Appelé lorsque l'activité est devenue visible (premier plan).
	onPause()	Appelé lorsque l'activité est sur le point de passer en arrière-plan.
	onStop()	Appelé lorsque l'activité n'est plus visible.
	onRestart()	Appelé après que l'activité a été arrêtée et est sur le point de redémarrer.

	onDestroy()	Appelé avant que l'activité ne soit détruite.
	onSaveInstanceState(Bundle)	Appelé avant que l'activité ne soit arrêtée, permet de sauvegarder l'état de l'activité dans un bundle.
	onRestoreInstanceState(Bundle)	Appelé lorsque l'activité est en cours de restauration, récupère l'état enregistré dans le bundle.
Interface	setContentView(int)	Crée l'interface utilisateur à partir d'un fichier de mise en page XML spécifié par le paramètre d'identifiant de ressource.
	findViewById(int)	Récupère un élément de vue de l'interface utilisateur. Renvoie un objet View. Le paramètre est l'ID de l'élément de vue (normalement défini dans la classe R).
	showDialog(int)	Ouvre une fenêtre de dialogue. Le paramètre est l'ID de la fenêtre de dialogue.
	showDialog(int, Bundle)	Ouvre une fenêtre de dialogue. Le premier paramètre est l'ID de la fenêtre de dialogue, le second permet de transmettre des paramètres au dialogue.
	dismissDialog(int)	Ferme une fenêtre de dialogue spécifiée par son ID.
	onCreateDialog(int, Bundle)	Appelé lorsqu'une fenêtre de dialogue est ouverte. Le premier paramètre est l'ID de la fenêtre de dialogue, le second est les données transmises à celle-ci.
	onPrepareDialog(int, Bundle)	Appelé lorsqu'une fenêtre de dialogue qui est déjà ouverte est activée. Le premier paramètre est l'ID de la fenêtre de dialogue, le second est les données transmises à celle-ci.

Ressources	getResources()	Renvoie l'objet Resources associé à l'activité.
	getString(int)	Récupère une ressource de chaîne avec l'ID donné.
Démarrage d'activités	startActivity(Intent)	Démarre une nouvelle activité spécifiée par une intention explicite.
	startActivityForResult(Intent, int)	Démarre une nouvelle activité et attend un résultat. Le premier paramètre est l'intention utilisée pour démarrer l'activité, le second est un code qui identifie la demande.
	startActivityIfNeeded(Intent, int)	Démarre une nouvelle activité uniquement si elle ne fonctionne pas déjà dans la tâche en cours.
	startActivityFromChild(Activity, Intent, int)	Démarre une nouvelle activité à partir d'une activité enfant. Le premier paramètre est l'activité enfant qui démarre la nouvelle activité.

3. Fichier Manifeste sur Android

Le fichier manifeste est une partie essentielle de l'application Android car il fournit des informations cruciales sur l'application au système Android. Il agit comme un plan de l'application, dont il définit la structure et les caractéristiques. Le fichier manifeste énumère tous les composants de l'application, tels que les activités, les services, les récepteurs de diffusion et les fournisseurs de contenu, et définit leurs propriétés et leurs autorisations.

Ce fichier est également utilisé par le système Android pour identifier et lancer l'application. Lorsque l'application est installée sur un appareil, le système lit le fichier manifeste pour déterminer ses composants et ses autorisations. Le fichier manifeste permet également de s'assurer que l'application est sécurisée et ne dispose pas d'un accès à des ressources ou à des fonctionnalités qu'elle ne devrait pas avoir. Il est essentiel de gérer correctement les autorisations et les autres caractéristiques de l'application dans le fichier manifeste pour garantir la fonctionnalité et la sécurité de l'application.

Voici la structure générale de fichier AndroidManifest.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="com.example.myapp">
4
```

```
    <uses-permission android:name="android.permission.INTERNET" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/app_icon"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

        <activity
            android:name=".MainActivity"
            android:label="@string/app_name"
            android:theme="@style/AppTheme.NoActionBar">

            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <service
            android:name=".MyService"
            android:exported="false">
            <intent-filter>
                <action android:name="com.example.myapp.MyService" />
            </intent-filter>
        </service>

        <receiver
            android:name=".MyReceiver"
            android:enabled="true"
            android:exported="false">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED" />
            </intent-filter>
        </receiver>

        <provider
            android:name=".MyContentProvider"
            android:authorities="com.example.myapp.MyContentProvider"
            android:exported="false" />

    </application>

</manifest>

```

3.1. Intent Filter

Un "intent filter" est une fonctionnalité puissante d'Android qui permet à une application de déclarer les types de "intents" qu'elle peut gérer. Ce filtre est défini dans le fichier manifeste d'une application et précise le type de requête que le composant souhaite recevoir. On peut considérer ce filtre comme une description de la capacité d'un composant à gérer des types spécifiques de commandes.

Lorsqu'une intention implicite est déclenchée, le système Android vérifie le filtre d'intention de chaque composant pour voir si l'un d'entre eux peut gérer la requête. Si plus d'un composant peut traiter cette requête, l'utilisateur se voit proposer une liste d'options parmi lesquelles il peut choisir. Cela permet une flexibilité et une personnalisation au sein de l'écosystème Android.

Le "intent filter" suivant est utilisé pour spécifier que l'activité principale (MainActivity) est le point d'entrée principal de l'application

```
<activity android:name=".MainActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Exemple d'un "intent filter" qui traite les demandes de cartes géographiques

```
<activity android:name=".MapActivity">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:scheme="geo" />
    </intent-filter>
</activity>

```

3.2. Activité

La balise <activity> décrit chaque activité existant dans l'application Android. Les informations de base sur l'activité principale sont généralement définies comme suit :

- android:name spécifie le nom de la classe Java qui implémente l'activité.
- android:label définit le nom affiché de l'application.
- <intent-filter> décrit les types d'intentions que cette activité peut gérer, par exemple l'intention de lancer l'application en tant qu'activité principale

```
<activity
    android:name=".MainActivity"
    android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

```

3.3. Service

La balise <service> - définit un service au sein de l'application. Il contient des informations sur le service, telles que son nom et ses filtres d'intention. Voici un exemple :

```
<service
     android:name=".MyService"
     android:exported="false">
     <intent-filter>
         <action android:name="com.example.myapp.MY_ACTION" />
     </intent-filter>
</service>

```

3.4. Récepteur de diffusion

Un récepteur de diffusion est défini par la balise <receiver> - dans le fichier manifeste. Elle contient des informations sur le récepteur, telles que son nom et ses filtres d'intention. Voici un exemple :

```
<receiver
    android:name=".MyReceiver">
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED" />
    </intent-filter>
</receiver>
```

3.5. Fournisseur de continu

Un fournisseur de contenu est défini par la balise <provider> - dans le fichier manifeste. Elle contient des informations sur le fournisseur, telles que son nom et ses filtres d'intention. Voici un exemple :

```
<provider
    android:name=".MyProvider"
    android:authorities="com.example.myapp.provider"
    android:exported="false">
</provider>

```

3.6. Les permissions

Dans Android, les permissions sont utilisées pour protéger les informations sensibles et les ressources du système contre tout accès non autorisé. Les autorisations sont déclarées dans le fichier AndroidManifest.xml et doivent être demandées par l'application au moment de l'exécution. La plateforme Android utilise un système de permissions pour s'assurer que seules les applications de confiance peuvent accéder à certaines fonctionnalités ou données sur l'appareil.

Pour déclarer une permission dans le fichier AndroidManifest.xml, la balise <uses-permission> est utilisée. L'attribut android:name est utilisé pour spécifier le nom de la permission. Les autorisations les plus courantes sont INTERNET, CAMERA, READ_CONTACTS et WRITE_EXTERNAL_STORAGE.

Par exemple, pour demander l'autorisation d'utiliser l'appareil photo de l'appareil, le code suivant peut être ajouté au fichier AndroidManifest.xml :

```
<uses-permission android:name="android.permission.CAMERA" />
```

Lorsqu'une application demande une autorisation définie dans le fichier AndroidManifest.xml, l'utilisateur est invité à l'accorder ou à la refuser. Si l'utilisateur accorde la demande, l'application est autorisée à accéder à la fonctionnalité ou aux données demandées. Si l'utilisateur refuse la demande, l'application ne peut pas accéder à la fonctionnalité ou aux données et peut être amenée à fournir une fonctionnalité alternative ou à quitter l'application.

Le tableau ci-dessous contient une liste des différentes permissions.

Permission	Description
android.permission.CAMERA	Permet à l'application d'accéder à la caméra pour prendre des photos ou enregistrer des vidéos.

android.permission.ACCESS_FINE_LOCATION	Autorise l'application à accéder à la position précise de l'appareil à l'aide du GPS ou d'autres sources de localisation.
android.permission.READ_CONTACTS	Permet à l'application de lire les contacts stockés sur l'appareil.
android.permission.WRITE_EXTERNAL_STORAGE	Autorise l'application à accéder à l'espace de stockage externe de l'appareil pour lire, écrire et supprimer des fichiers.
android.permission.RECORD_AUDIO	Permet à l'application d'enregistrer de l'audio en utilisant le microphone de l'appareil.
android.permission.READ_CALENDAR	Autorise l'application à lire les événements du calendrier stockés sur l'appareil.
android.permission.SEND_SMS	Permet à l'application d'envoyer des messages texte.
android.permission.CALL_PHONE	Autorise l'application à passer des appels téléphoniques sans l'intervention de l'utilisateur.
android.permission.INTERNET	Autorise l'application à accéder à Internet.
android.permission.ACCESS_NETWORK_STATE	Permet à l'application de savoir si l'appareil est connecté à Internet et de vérifier l'état de la connexion réseau.

4. Ressource et la classe R

Dans le développement Android, les ressources désignent tous les assets non codés utilisés dans une application, tels que les images, les chaînes de caractères, les layouts, les couleurs, etc. Ces ressources sont fournies avec l'application et servent à améliorer l'expérience de l'utilisateur, par exemple en fournissant un texte localisé ou en s'adaptant à différentes tailles et densités d'écran.

L'utilisation des ressources de manière organisée et efficace peut faciliter et accélérer le développement, tout en optimisant les performances de l'application. Android propose différentes manières d'organiser et d'accéder aux ressources, comme l'utilisation de qualificateurs de ressources pour s'adapter aux différentes configurations des appareils et la création de fichiers XML distincts pour définir et référencer les ressources.

La classe R est un élément clé de l'accès aux ressources dans une application Android, car elle fournit une correspondance entre l'identifiant de la ressource et sa valeur. Il est important de comprendre comment générer et utiliser la classe R pour gérer efficacement les ressources dans le cadre du développement d'une application.

4.1. Les répertoires des Ressources

les ressources dans Android sont réparties dans plusieurs répertoires :

- */res/layout* : Ce répertoire contient les fichiers XML décrivant l'interface utilisateur de l'application.

- */res/values* : Ce répertoire contient les fichiers XML qui définissent les valeurs de ressources qui sont utilisées dans toute l'application, telles que les chaînes de caractères, les couleurs, les dimensions, les styles, les thèmes, etc.
- */res/drawable* : Ce répertoire contient les fichiers d'images et de graphiques qui sont utilisés dans l'application.
- */res/mipmap* : Ce répertoire contient les icônes de l'application qui sont utilisées dans le tiroir d'applications et d'autres endroits de l'interface utilisateur.
- */res/raw* : Ce répertoire contient les fichiers binaires, tels que les fichiers audio, vidéo ou autres types de fichiers non traitables par Android, qui sont utilisés dans l'application.
- */res/xml* : Ce répertoire contient les fichiers XML qui définissent les ressources personnalisées, telles que les fichiers de configuration ou les fichiers de définition de menu.
- */res/menu* : Ce répertoire contient des fichiers XML définissant les menus utilisés dans l'application. Ces menus peuvent être gonflés dans des activités ou des fragments et affichés dans la barre d'outils de l'application ou comme menus contextuels.

4.2. Ressource de type valeur

Les ressources de type valeurs sont utilisées pour définir divers types de données tels que des chaînes de caractères, des couleurs, des dimensions, des nombres entiers et des tableaux qui peuvent être utilisés dans l'ensemble de l'application.

Strings

Définition d'une ressource de type "String" en XML : Pour définir une ressource de type chaîne en XML, vous devez créer une nouvelle entrée dans le fichier strings.xml situé dans le répertoire values de votre projet Android. Voici un exemple de définition d'une ressource de type chaîne :

```
1 <!-- strings.xml -->
2 <resources>
3     <string name="hello_world">Hello World!</string>
4 </resources>
5
```

Utilisation d'une ressource de type chaîne de caractères en XML : Pour utiliser une ressource de type chaîne de caractères en XML, il suffit de référencer la ressource de type chaîne de caractères en utilisant la syntaxe "@string" suivie du nom de la ressource de type chaîne de caractères. Voici un exemple :

```
1 <!-- activity_main.xml -->
2 <TextView
3     android:id="@+id/hello_text"
4     android:layout_width="wrap_content"
5     android:layout_height="wrap_content"
6     android:text="@string/hello_world" />
7
```

Utilisation d'une ressource de type "String" en Java : Pour utiliser une ressource de type chaîne en Java, vous pouvez y accéder en utilisant la classe R générée. Voici un exemple :

```
1 // MainActivity.java
2 public class MainActivity extends AppCompatActivity {
3     @Override
4     protected void onCreate(Bundle savedInstanceState) {
5         super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_main);

        // Get the string resource
        String hello = getString(R.string.hello_world);

        // Use the string resource
        Toast.makeText(this, hello, Toast.LENGTH_SHORT).show();
    }
}

```

Tableaux de strings

La définition des tableaux de chaînes se fait dans le fichier strings.xml comme suit :

```
<resources>
    <string-array name="planets_array">
        <item>Mercury</item>
        <item>Venus</item>
        <item>Earth</item>
        <item>Mars</item>
        <item>Jupiter</item>
        <item>Saturn</item>
        <item>Uranus</item>
        <item>Neptune</item>
    </string-array>
</resources>

```

L'utilisation de tableaux de chaînes de caractères dans un fichier de layout XML se fait comme suit :

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@array/planets_array" />

```

L'utilisation de tableaux de strings comme référence dans le code Java :

```
String[] planets = getResources().getStringArray(R.array.planets_array);
```

Styles

Définir les Style dans le fichier styles.xml dans le répertoire res/values.

```
<style name="MyStyle">
    <item name="android:textSize">20sp</item>
    <item name="android:textColor">#FF0000</item>
</style>

```

Comment l'utiliser en XML :

```
<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    style="@style/MyStyle"
    android:text="Hello World!" />
```

Comment l'utiliser en Java :

```
1 TextView textView = findViewById(R.id.textView1);
2 textView.setStyle(R.style.MyStyle);
```

Couleurs

Définition des couleurs dans le fichier colors.xml :

```
1 <resources>
2     <color name="colorPrimary">#3F51B5</color>
3     <color name="colorAccent">#FF4081</color>
4 </resources>
5
```

Utilisation de couleurs dans un fichier de layout XML :

```
1 <TextView
2     android:layout_width="wrap_content"
3     android:layout_height="wrap_content"
4     android:textColor="@color/colorPrimary"
5     android:text="Hello World!" />
6
```

Référencement des couleurs dans le code Java :

```
1 int colorPrimary = getResources().getColor(R.color.colorPrimary);
2
```

4.3. La classe R

La classe R dans Android est une classe générée dynamiquement qui offre un moyen pratique d'accéder aux ressources telles que les layouts, les tables dessinées et les chaînes de caractères dans votre application Android. La classe R est générée automatiquement par les outils de construction d'Android lors de la compilation des ressources de votre application, et elle fournit un identifiant entier unique pour chaque ressource de votre application, ce qui vous permet d'y accéder facilement dans votre code.

La classe R est créée dans l'espace de noms de votre application (par exemple, com.example.myapp.R) est organisée par type de ressource, avec des sous-classes pour chaque type de ressource. Par exemple, les ressources de layout de votre application sont accessibles via la sous-classe R.layout, et les ressources de dessin sont accessibles via la sous-classe R.drawable.

En accédant aux ressources à l'aide de la classe R, votre application est en mesure de s'adapter dynamiquement aux différentes configurations de l'appareil et aux paramètres de l'utilisateur, tels que la langue et la densité de l'écran, ce qui garantit que votre application s'affiche et fonctionne correctement sur une grande variété d'appareils. Les meilleures pratiques pour gérer et optimiser l'accès aux ressources avec la classe R comprennent l'utilisation de qualificateurs de ressources pour fournir des ressources alternatives pour différentes configurations d'appareils, et l'utilisation de la mise en cache des ressources pour améliorer les performances de l'application.

Voici un exemple de la structure de la classe R

```
1 public final class R {
2     public static final class anim {
3         public static final int fade_in = 0x7f050000;
4         public static final int fade_out = 0x7f050001;
```

```
    }
    public static final class drawable {
        public static final int ic_launcher = 0x7f020000;
        public static final int logo = 0x7f020001;
    }
    public static final class id {
        public static final int button_ok = 0x7f060000;
        public static final int button_cancel = 0x7f060001;
    }
    // more inner classes for other resource types
}

```