République Tunisienne
Ministère de
l'Enseignement
Supérieur
et de la Recherche
Scientifique

الجمهورية التونسية
وزارة التعليم العالي
والبحث العلمي

Compte rendu TP Service

Auteur
Karim Dhafer
Ahmed Khammesi

Matière
Développement Mobile sur Android

Classe
L3-DSI3

Année universitaire
2024-2025

Sommaire

Liste des figures

Introduction

Ce rapport se concentre sur le concept des services dans le cadre du développement d'une application Android, en spécifiant leur intégration au sein de l'application et leurs fonctionnements avec un exemple pratique sur Android Studio.

Objectifs

- Apprendre le concept du service et ses méthodes sur Android.
- Apprendre à créer un service.
- Apprendre l'implémentation et le fonctionnement des méthodes de base.

1. Les services sur Android

Un service est l'un des composants les plus importants et les plus fondamentaux dans le développement des applications Android. Contrairement aux activités Android, les services n'ont pas d'interface utilisateur et n'effectuent pas d'opérations au premier plan. Ils gèrent plutôt des tâches en arrière-plan, telles que le téléchargement de fichiers, la lecture de musique, etc.

1.1 Les types de services

Les services Android peuvent être de deux types : soit **"Started"**, soit **"Bounded"**.

- **"Started"** :
  Un service "Started" est initié par un appel d'un autre composant, tel qu'une activité, à l'aide de la méthode `startService()`. Une fois démarré, le service continue à fonctionner en arrière-plan indéfiniment, même si le composant qui l'a démarré est détruit.
  Généralement, un service démarré effectue une seule opération sans renvoyer de résultat au composant appelant. Une fois sa tâche terminée, le service doit s'arrêter lui-même en appelant `stopSelf()` ou par un autre composant qui appelle `stopService()`.

- **"Bounded"** :
  Un service "Bounded" est lancé par un appel d'un autre composant, tel qu'une activité, à l'aide de la méthode `bindService()`. Ce type de service fournit une interface client-service, permettant à d'autres composants d'interagir avec lui en envoyant des requêtes, en recevant des résultats et même en effectuant une communication inter-processus (IPC).
  Un service "Bounded" ne fonctionne que si au moins un composant lié à lui. Si aucun composant n'est lié, le service est détruit.

1.2 Le cycle de vie d'un service

Chaque service a un cycle de vie qui commence lors de sa création et se termine lors de sa destruction. Ce cycle de vie passe par des états prédéfinis. Le diagramme ci-dessous illustre les cycles de vie des deux types de services et leurs états au cours de leur durée de vie.

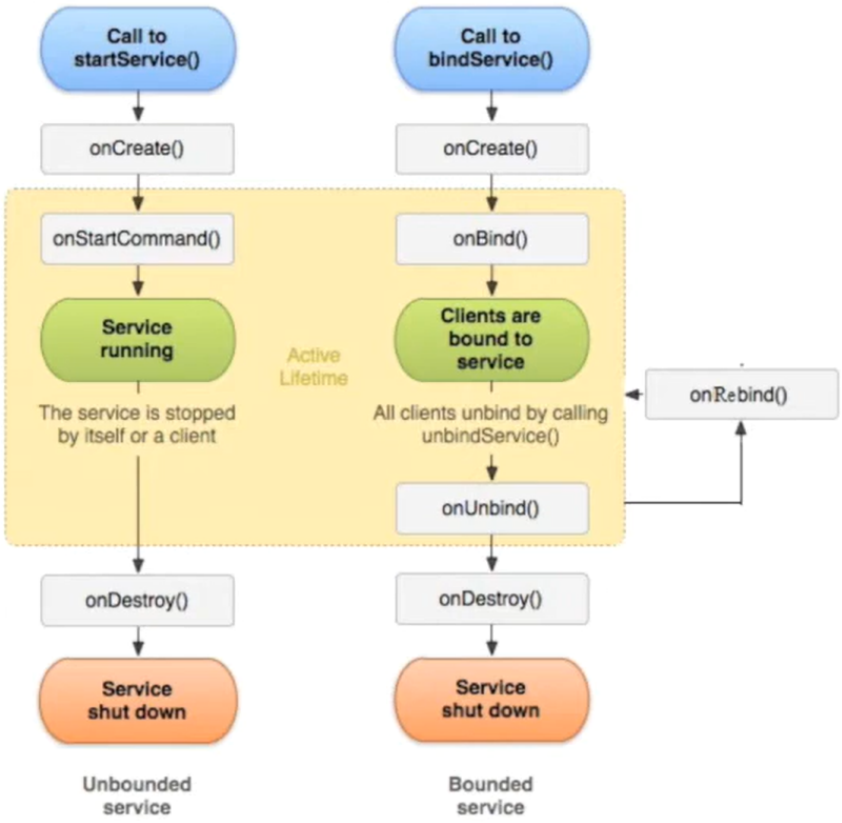

Fig. 1 : Diagramme de cycle de vie d'un service Android

1.3 Les méthodes basiques du service

Comme il est indiqué dans le diagramme de cycle de vie, le service possède des méthodes basiques qui seront appelés pratiquement dans toutes les cas de son développement.

- **`onCreate()` :**
  C'est la méthode essentielle lors de la création du service, avant d'appeler les autres méthodes `onStartCommand()` ou `onBind()`. Cette fonction n'est pas appelée à nouveau si le service est déjà en cours d'exécution.

- **`onStartCommand()` :**
  C'est la méthode invoquée lorsqu'un autre composant appelle `startService()`. Le service démarre et s'exécute en arrière-plan indéfiniment. Puis c'est au développeur d'arrêter le service une fois sa tâche terminée en appelant `stopSelf()` ou `stopService()`. Cette méthode est facultative si le service n'assure que la liaison.

Fig. 2 : La méthode onStartCommand()

- **`onBind() :`**
  C'est la méthode appelée lorsqu'un composant appelle `bindService()` pour se lier au service. Cette méthode fournit une interface que les appelants utilisent pour communiquer avec le service en renvoyant un **IBinder**. Si la liaison n'est pas autorisée, elle doit renvoyer null.

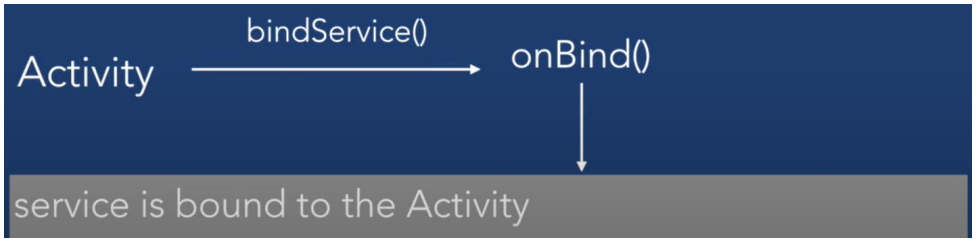

Fig. 3 : La méthode onBind()

- **`onDestroy() :`**
  C'est la méthode invoquée lorsque le service n'est plus utilisé et qu'il est en cours de destruction pour le but de nettoyer les ressources telles que les threads, les listeners ou les récepteurs. C'est la dernière méthode du cycle de vie d'un service.

2. Créer un service sur Android

Pour créer un service sur Android, il est nécessaire de créer une classe qui hérite de la classe préexistante `Service` dans Android, puis implémenter les classes abstraites de la classe héritée. Enfin, il faut déclarer ce service dans le `AndroidManifest.xml`. Cependant, afin de faciliter cette procédure et permettre aux développeurs de se concentrer sur les aspects métier du projet, Android Studio propose une méthode simplifiée pour créer un service en utilisant l'interface du IDE en suivant ces étapes :

1. Accéder à la fenêtre de création d'un service avec un clic droit sur le projet ou bien à partir de la barre d'outils dans l'onglet Fichier, puis suivez le chemin **New > Service > Service**, comme les images ci-dessous l'indiquent.

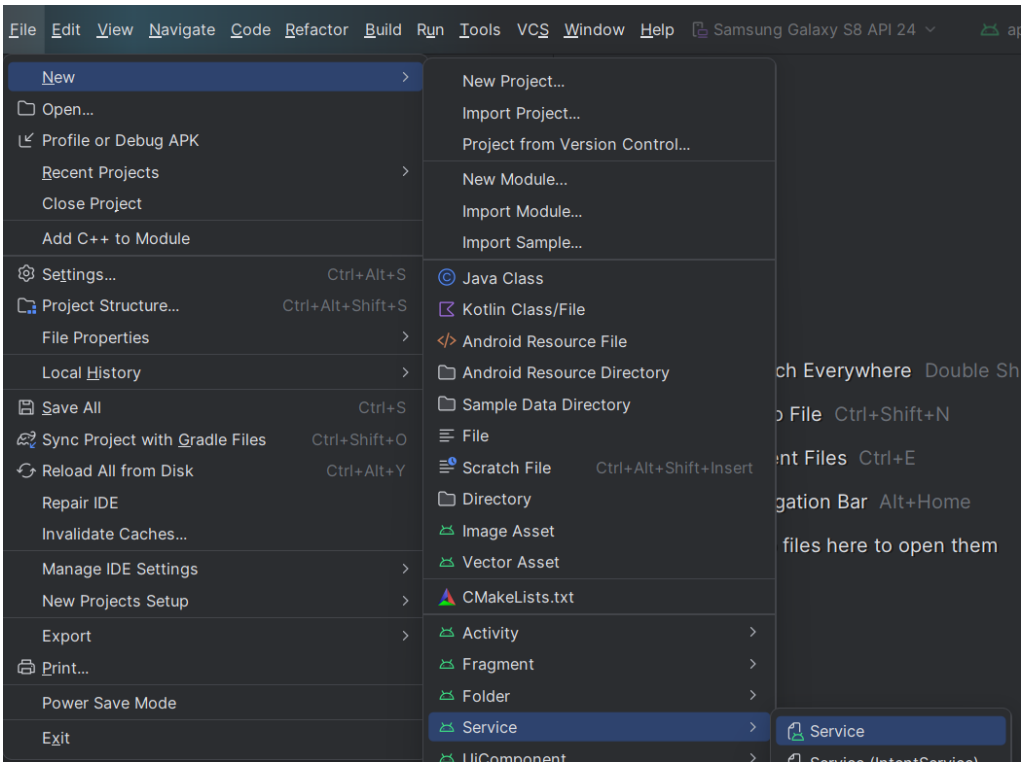

FIG. 4 : Création du service à partir de la barre d'outils

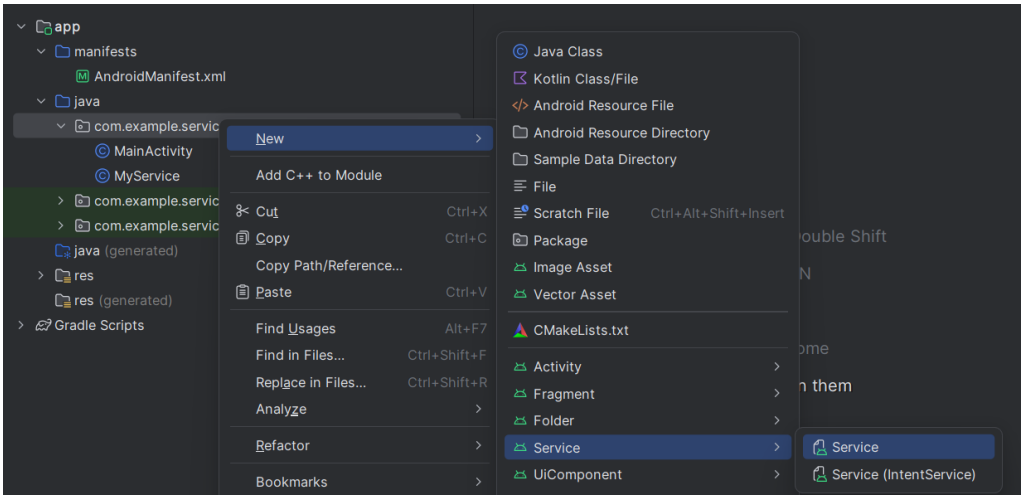

FIG. 5 : Création du service à partir du clic droit sur le projet

2. À ce moment, une interface apparaîtra, où nous pourrons modifier le nom du service selon notre choix et définir si celui-ci est *exportable* ou non. Un service exportable signifie qu'il peut être accessible et utilisé par d'autres applications installées sur l'appareil, en fonction des autorisations définies.

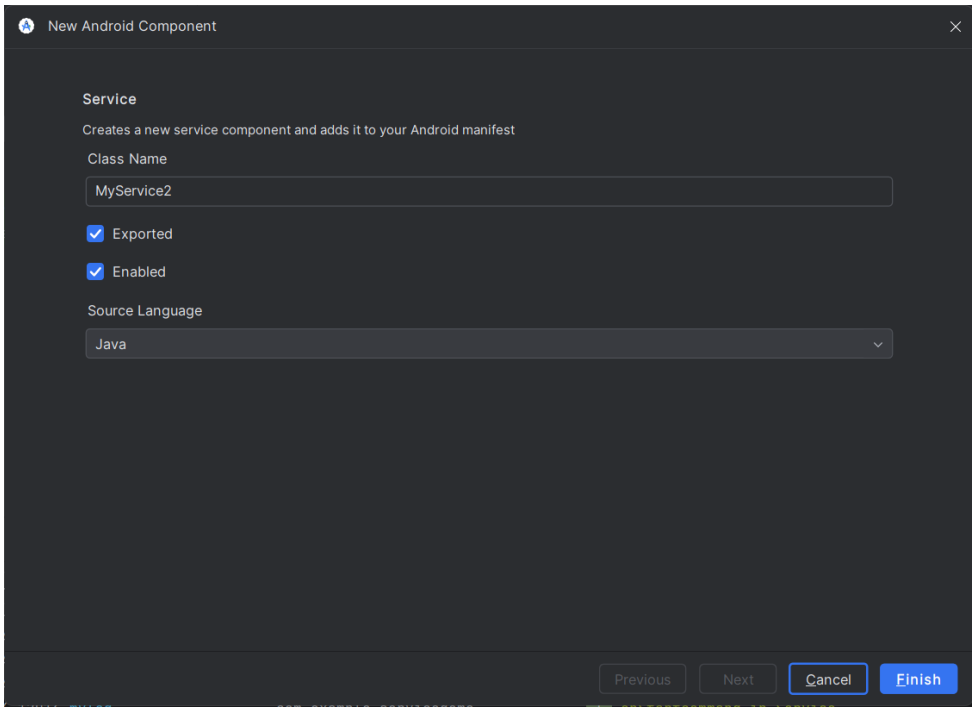

FIG. 6 : Fenêtre de création du service

3. Implémentation et fonctionnement des méthodes du service

Dans cette partie, nous allons explorer l'implémentation des méthodes principales du service, ainsi que leur fonctionnement au sein du cycle de vie en utilisant les journaux LogCat. Nous verrons aussi l'importance des flags retournés par la méthode onStartCommand(), qui déterminent le comportement du service dans le système.

3.1 Implémentation des méthodes

Pour mettre en œuvre un service dans Android, il est essentiel d'implémenter les méthodes nécessaires au cycle de vie du service dans la classe dédiée, ainsi que de permettre leur invocation depuis une activité via des interactions utilisateur.

3.1.1 Redéfinir des méthodes dans la classe du service

Dans cette étape, nous allons redéfinir (override) les principales méthodes de la classe Service, à savoir :

- **`onCreate()` :** C'est la méthode appelée lors de création pour initialiser les ressources nécessaires du service. Pour cela, nous appelons la méthode `onCreate()` de la classe mère. Puis, nous ajoutons un message qui apparaît à chaque appel de la méthode pour suivre son exécution.

```
1      @Override
2      public void onCreate() {
3          Log.i("mylog", "onCreate in service");
4          super.onCreate();
5      }
```

- **`onStartCommand()` :** C'est la méthode invoquée directement aprés `onCreate()` utilise un `Intent` passé en paramètre. Elle gère les actions spécifiques du service et retourne un code indiquant comment le système doit gérer le service après l'exécution. Pour cela, nous appelons la méthode `onStartCommand()` de la classe mère. Puis, nous ajoutons un autre message pour suivre l'exécution.

```
1       @Override
2       public int onStartCommand(Intent intent, int flags, int
            startId) {
3           Log.i("mylog", "onStartcommand in Service");
4           Log.i(
5                   "mylog",
6                   "thread id in service is " + Thread.
                        currentThread().getId()
7           );
8           // stopSelf();
9           return super.onStartCommand(intent, flags, startId);
10      }
```

- **`onDestroy()` :** C'est la méthode invoquée lorsque le service est détruit, pour libérer les ressources. Nous allons implémenter le même logique en appelant `onDestroy()`.

```
1       @Override
2       public void onDestroy() {
3           Log.i("mylog", "onDestroy in Service");
4           super.onDestroy();
5       }
```

Attention

→ Il est important de ne pas oublier d'appeler `stopSelf()` dans la méthode `onStartCommand()`. Cela permet d'arrêter le service automatiquement après l'exécution d'une tâche. Dans cet exemple simple, nous ne l'avons pas implémenté pour illustrer un cas de base.

3.1.2 Invocation des méthodes à partir de l'activité

L'invocation des méthodes du service est réalisée depuis l'activité principale à l'aide de boutons permettant de démarrer ou d'arrêter le service.

```
1       start = (Button) findViewById(R.id.start);
2       stop = (Button) findViewById(R.id.stop);
3       Log.i(
4               "mylog",
5               "thread id in main activity is " + Thread.currentThread()
                    .getId()
6       );
7
8       start.setOnClickListener(new View.OnClickListener() {
9           @Override
10          public void onClick(View view) {
11              // start service
```

```
12              Intent intent = new Intent(getApplicationContext(),
                    MyService.class);
13              startService(intent);
14          }
15      });
16
17      stop.setOnClickListener(new View.OnClickListener() {
18          @Override
19          public void onClick(View view) {
20              // stop service
21              Intent intent = new Intent(getApplicationContext(),
                    MyService.class);
22              stopService(intent);
23          }
24      });
```

Ces deux boutons permettent d'interagir avec le service :

- Le bouton Start initialise et démarre le service en appelant la méthode `onStartCommand()` via l'intent correspondant avec la méthode `startService()`.
- Le bouton Stop arrête le service et déclenche la méthode `onDestroy()` via la méthode `stopService()`

3.2 Fonctionnement des méthodes

Pour vérifier et comprendre le fonctionnement des méthodes du service, nous avons implémenté des journaux `Log.i` pour suivre l'exécution en temps réel à partir du LogCat, en spécifiant le nom de log afin de filtrer les journaux pertinents.

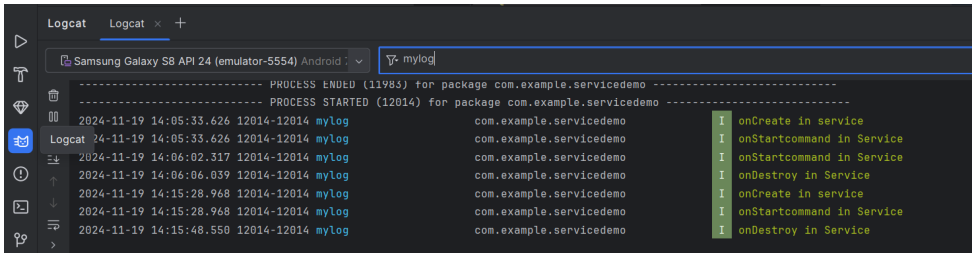

Fig. 7 : LogCat

Ci-dessous, un exemple d'exécution du programme dans LogCat, nous observons que le service s'exécute dans le thread main.

Fig. 8 : Exemple d'exécution sur LogCat

NB :
Il est important de souligner que, dans le cas de l'exécution du service sur le même thread de main, les tâches longues ou bloquantes doivent être déléguées à un autre distinct pour éviter de bloquer l'interface utilisateur et assurer une expérience fluide pour l'utilisateur.

3.3 Les flags de la méthode `onStartCommand()`

La méthode `onStartCommand()` dans un service retourne une valeur qui indique comment le système doit gérer le service après son exécution. En fonction de cette valeur, le système peut prendre différentes actions concernant le service, notamment décider de le redémarrer, de le laisser actif ou de le stopper. Cette valeur peut inclure différents flags (drapeaux) qui définissent le comportement du service, en particulier en ce qui concerne son redémarrage ou son arrêt. Ces flags sont spécifiés sous forme d'entiers. Le rôle du système ici est de garantir une gestion optimale des ressources en fonction des besoins de l'application et des conditions d'exécution, tout en respectant les priorités de performance et de stabilité de l'appareil.
Les trois principaux flags utilisés sont les suivants :

- **START_STICKY** : Ce flag, comme l'indique le nom, reste attaché au système même lors d'un arrêt inattendu dû à une insuffisance de mémoire. Il informe le système qu'il doit recréer une nouvelle instance du service lorsque de l'espace mémoire sera disponible. Mais les résultats et les opérations effectuées au cours de son exécution initial seront perdus.
- **START_NOT_STICKY** : Ce flag indique que le service ne doit pas être redémarré si le système le tue. Le service sera arrêté définitivement si un arrêt inattendu s'est produit.
- **START_REDELIVER_INTENT** : Ce flag se comporte comme le premier mais en plus il insiste sur la nécessité du retour des intents qui ont été présents lors de l'arrêt pour continuer les opérations lors du lancement du service à nouveau. Prenant l'exemple du téléchargement qui doit être continue du moment de l'arrêt.

Conclusion

Ce rapport décrit le rôle des services Android et leurs méthodes essentielles en mettant l'accent sur leur fonctionnement. Une bonne maîtrise de ces concepts est cruciale pour développer des applications Android réactives et performantes et de mieux gérer les ressources du système.