TP4 : Les Services.
Créer un service de téléchargement

Objectifs :

- créer un service (classe **Service**) dédié à une activité,
- activer un service dans un processus dédié avec démarrage automatique,
- créer un service d'intention (classe **IntentService**) exportable pour exécuter des téléchargements en tâche de fond,
- utiliser la diffusion d'intention (broadcast) pour signaler l'avancement et la fin de l'opération du service.

Au final, deux applications seront mises en place :

- ✓ une application de service, assurant à la demande d'un client des téléchargements en tâche de fond,
- ✓ une application client utilisant le service précédent lui signalant l'évolution du téléchargement ainsi que sa fin à l'aide de notifications.

Premier Service

Autant vous prévenir tout de suite, ce premier service n'apportera aucun service !! Pour l'instant l'objectif étant juste de mettre en place le mécanisme de création, de démarrage et d'arrêt d'un service.

Mise en place de l'activité principale

1. Créer un nouveau projet **Android Application** nommé **Service1**, l'application portera le nom du projet et le paquetage sera nommé **net.votrenom.votreprénom.service1** :
   - ✓ utiliser **l'API 14** comme sdk mini et l'**API 19** pour le développement,
   - ✓ vous pouvez utiliser l'icône fournie dans le dossier **AndroidResources** pour « agrémenter » votre application,
   - ✓ conserver les noms par défaut de l'activité (**MainActivity.java**) et du layout (**activity_main.xml**) générés.

Voici le layout que vous devez obtenir pour l'activité **activity_main.xml**, celle-ci servant uniquement à démarrer et arrêter le futur service :

Vous aurez reconnu en partant du haut :

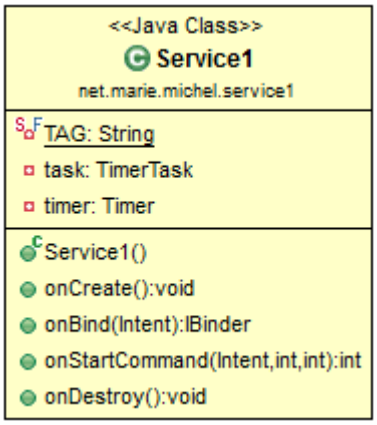
- ✓ un contrôle de type « **TextView** » pour le titre de la vue,
- ✓ deux contrôles de type « **Button** » qui serviront à démarrer/arrêter le service,
- ✓ et c'est tout !

2. A vous de jouer, faire en sorte d'obtenir la vue présentée ci-dessus.

Mise en place du service

3. Ajouter à votre projet une nouvelle classe **Service1** héritant de la classe **Service**.
4. Surcharger les méthodes suivantes de la classe service :
   - ✓ **onCreate()** : appelée par le système à la création du service,
   - ✓ **onBind()** : appelée pour créer un canal de communication entre le client et le service,
   - ✓ **onStartCommand()** : appelée par le système lorsque le client démarre le service,
   - ✓ **onDestroy()** : appelée par le système lorsqu'il décide de détruire le service : le service est resté trop longtemps sans être utilisé ou le système a besoin de ressources.
5. Ajouter à la classe **Service1** les deux attributs privés suivants :
   - ✓ **task** de type **TimerTask**, pour effectuer une tâche qui « postera » un message **Toast** au client du service,
   - ✓ **timer** de type **Timer**, pour activer la tâche précédente périodiquement, toutes les 5 secondes.

Ci-dessous le diagramme de la classe **Service1** :



6. Coder la méthode **onCreate()** afin qu'elle instancie l'objet **timer**.
7. Pour l'instant nous n'utiliserons pas de canal de communication entre le client et le service, la méthode **onBind()** se contentera donc de retourner null.
8. Coder la méthode **onStartCommand()** afin qu'elle affiche un **Toast** toutes les 5 secondes (période de déclenchement du timer) via la tâche timer **task**. Cet affichage périodique nous permettra de constater le fonctionnement du service.
9. Coder la méthode **onDestroy()** afin qu'elle libère le timer et l'éventuelle tâche associée.

Remarque : *vous pouvez ajouter un message de debug spécifique (fonction **Log.d()**) avec le **tag Service1** dans chacune des méthodes précédentes, ce qui permettra de visualiser leur déclenchement avec **LogCat**.*

10. Compléter le manifeste de l'application afin de déclarer le service **Service1**.

Finalisation de l'activité

11. Compléter la méthode surchargée **Activity.onCreate()** afin :
    - ✓ d'instancier les objets boutons de démarrage et d'arrêt du service,
    - ✓ d'associer un écouteur sur ces boutons pour capturer l'événement **onClick** et déclencher la méthode associée,
    - ✓ de coder les méthodes **onClick** pour démarrer et arrêter le service.
12. Vérifier le bon fonctionnement de l'ensemble.

Exécution du service dans un processus dédié

13. Modifier le manifeste afin que le service s'exécute dans un processus dédié. Vérifier le bon fonctionnement de l'ensemble et vérifier le processus créé avec DDMS. Vous pouvez le tuer et le recréer par l'activité.

Faire démarrer le processus automatiquement avec le démarrage du système

14. Ajouter à votre projet une nouvelle classe **AutoStartService** héritant de la classe **BroadcastReceiver**.
15. Surcharger la méthode **onReceive()** afin qu'elle démarre le service **Service1**.
16. Compléter le manifeste afin de capturer l'intention **BOOT_COMPLETED** et de l'associer à la classe **AutoStartService**.
17. Vérifier le bon fonctionnement de l'ensemble.

Deuxième service

Et non, ce deuxième service n'apportera pas vraiment de service non plus !! Cette fois l'objectif visé consiste à utiliser la méthode **onBind()** laissée de côté pour créer une connexion au service afin, une fois démarré, de lui demander l'exécution de méthodes spécifiques pour récupérer des données de ce service par exemple.

Repartir du projet existant pour créer un nouveau service (afin de ne pas tout casser dans le Service1)

1. A « l'extérieur » de Eclipse, réaliser une copie de votre dossier de projet **Service1** et le renommer **Service2**.
2. Depuis Eclipse importer ce nouveau projet **File->Import->Existing projects into Workspace** (puis sélectionner le dossier **Service2**).
3. Au sein de ce nouveau projet **Service2**, modifier le nom du paquetage en remplaçant **service1** par **service2** (clic droit puis **refactor** sur le paquetage dans le dossier **src**).
4. Renommer la classe **Service1** en **Service2** (clic droit puis **refactor** sur le fichier **Service1.java** dans le dossier **src**).
5. Supprimer la classe **AutoStartService**, nous n'utiliserons pas cette fonction dans cette partie.
6. Dans le fichier de ressources **strings.xml** modifier le nom de l'application en **service2** ainsi que le titre affiché dans la vue.
7. Dans le fichier **Manifest** :
    - ✓ en général il faut y corriger le nom du paquetage (petit bogue qui sera corrigé un jour…),
    - ✓ supprimer ce qui fait référence au démarrage automatique,

- ✓ supprimer l'exécution du service dans un processus dédié (non supporté pour la connexion au service).

8. Faire un clean du projet pour supprimer toutes les références à l'ancien paquetage.

Voilà, on se retrouve avec un clone du **service1** nommé **service2** que l'on va pouvoir torturer !

Modification de l'activité principale

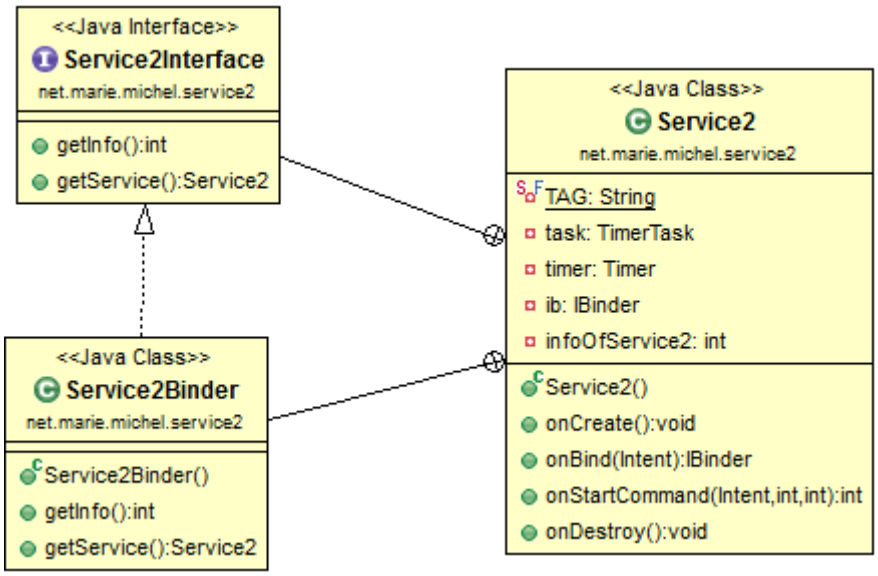
9. Compléter le layout de l'activité, **activity_main.xml**, afin de lui ajouter un troisième bouton comme ci-dessous, celui-ci servira à appeler une méthode exposée par le service.

Permettre la connexion au service pour appeler des méthodes

Plutôt que d'afficher un Toast toutes les 5 secondes nous allons modifier le service afin, une fois démarré, qu'il incrémente une variable chaque seconde. Ensuite nous allons lui ajouter une interface exposant une méthode permettant au client du service de lire le nombre de secondes écoulées depuis le démarrage du service. Nous aurons ainsi créé un service chronomètre, indispensable n'est-ce pas…

10. Compléter la classe **Service2** avec :
    - ✓ un attribut **nbrSecondes** de type **int** pour compter les secondes,
    - ✓ un attribut **ib** de type **Binder** pour la connexion au service.
11. Modifier la méthode **onStartCommand()** afin qu'elle active le comptage des secondes dans un thread activé périodiquement par le timer.
12. Ajouter à la classe **Service2** une interface intégrée publique, nommée **Service2Interface**, imposant les deux méthodes suivantes :
    - ✓ **public Service2 getService()** : qui retournera la référence au **Service2**,
    - ✓ **public int getInfo()** : qui retournera l'attribut **nbrSecondes**.
13. Ajouter à la classe **Service2** une classe intégrée, nommée **Service2Binder**, héritée de la classe **Binder** et implémentant l'interface précédente. Coder les deux méthodes imposées de cette classe.
14. Compléter la méthode **onCreate()** afin qu'elle initialise le compteur des secondes à 0 et qu'elle instancie l'objet **ib** avec une instance de **Service2Binder**.
15. Compléter la méthode **onBind()** afin qu'elle retourne l'objet **ib**, canal de communication avec le service.

Le diagramme des classes du service devrait donner ceci :



Finalisation de l'activité

16. Déclarer un attribut nommé **monService**, du type **Service2Interface** pour la connexion et l'appel des méthodes du service.
17. Ajouter à la classe **MainActivity** une instance, nommée **maConnexion**, de la classe « callback » **ServiceConnexion** et coder ses deux méthodes abstraites afin de se connecter et se déconnecter du service.
18. Compléter la méthode surchargée **Activity.onCreate()** afin :
    - ✓ de déclarer et d'instancier le bouton « Lire Info… »,
    - ✓ d'associer un écouteur sur ce bouton pour capturer l'événement **onClick** et déclencher la méthode associée.
19. Modifier la méthode **onClick()** du bouton « Connexion » afin qu'elle établisse la connexion au service et démarre le service.
20. Modifier la méthode **onClick()** du bouton « Deconnexion » afin qu'elle stoppe le service puis déconnecte l'activité du service.
21. Coder la méthode **onClick()** du bouton « Lire Info… » afin qu'elle appelle la méthode **getInfo()** du service pour afficher le nombre de secondes écoulées dans un **Toast**.
22. Vérifier le bon fonctionnement de l'ensemble.

Troisième (et dernier) service

Comme je sens bien que vous en avez assez des services qui ne servent à rien, nous allons passer à un service qui sert à pas grand-chose…

Et pour éviter de passer du temps sur des choses répétitives et maîtrisées je vous fournis pour cette partie un projet (prêt à l'emploi) dans lequel l'activité principale est définie et en partie codée, et le service bien dégrossi… Le projet en question se nomme **DownloadService** et se trouve dans le dossier de ressources **AndroidRessources**.

Le service à compléter servira à télécharger un fichier en arrière-plan. Une fois téléchargé le fichier sera stocké dans l'espace de stockage externe principal du périphérique. Une notification sera diffusée pour signaler la fin du téléchargement, charge à l'activité client de la traiter. Le service recevra de la part de l'activité client :

- ✓ l'URL du fichier à télécharger (via la clé **URLPATH**),
- ✓ le nom du fichier à écrire dans l'espace de stockage (via la clé **FILENAME**).

Nous utiliserons pour cette partie un service de type **IntentService**.

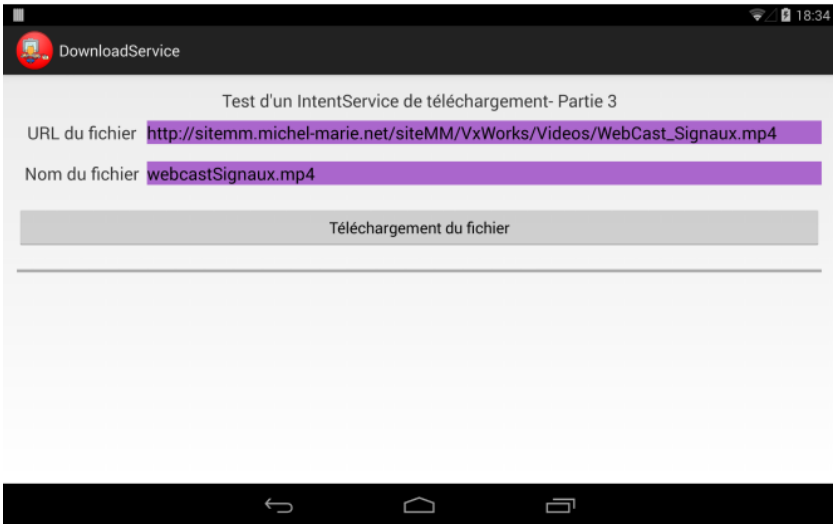
Mise en place du projet

1. Copier le dossier du projet **DownloadService** dans le dossier de votre **WorkSpace** puis l'importer.
2. Si le nom du paquetage ne vous convient pas (ce que je peux comprendre compte tenu de votre ego) modifiez-le (sans oublier la correction du **Manifest**).

Vous avez à présent 5mn pour vous approprier ce code et demander d'éventuels éclaircissements.

Démarrage du service

La vue suivante représente l'activité client du service que vous devez avoir, à l'URL et au nom du fichier près.



3. Faire en sorte de compléter l'activité afin que le clic sur le bouton « Téléchargement du fichier » démarre le service en lui passant les paramètres attendus.
4. Compléter le manifeste afin de déclarer le service avec un filtre sur l'intention déclenchée par l'activité.
5. Vérifier le bon fonctionnement du démarrage du service et le téléchargement du fichier spécifié. Vous pouvez pour cela utiliser l'outil **DDMS->File Explorer**.

Signaler la fin du téléchargement

Pour signaler l'arrêt du téléchargement suite à son succès ou suite à une erreur, le service va diffuser (broadcast) une intention avec deux paramètres :

- ✓ le chemin absolu du fichier sauvegardé (utiliser la clé **FILEPATH**),
- ✓ un entier : **Activity.RESULT_CANCEL** si échec et **Activity.RESULT_OK** si succès (utiliser la clé **RESULT**).

La méthode **publishResults()** a été prévue à cette fin et la variable **NOTIFICATION** a été prévue pour signer l'action de l'intention diffusée.

6. Coder la méthode **publishResults()** afin qu'elle diffuse l'intention signalant l'arrêt du téléchargement.
7. Compléter la méthode réalisant le téléchargement **onHandleIntent()** afin qu'elle appelle la méthode précédente avec les bons paramètres.

8. Compléter le manifeste afin que l'application déclare un récepteur d'intention « receiver » avec un filtre sur l'intention diffusée par **publishResults()**. La classe déclarée pour le traitement de cette intention (**BroadcastReceiver**) sera nommée **EndDownloadReceiver**.
9. Ajouter à votre projet une classe nommée **EndDownloadReceiver** héritant de **BroadcastReceiver** et surcharger la méthode **onReceive()**.
10. Coder la méthode précédente pour qu'elle traite l'intention et affiche dans un **Toast** un message signalant le succès ou l'échec du téléchargement. Vous disposez pour cela dans la classe **R.string** des deux chaînes suivantes prévues à cette fin :
    - ✓ **R.string.messageToastDownloadFini**,
    - ✓ **R.string.messageToastEchecDownload**.

    (si besoin voir ***solutionTP4_q10.txt***)
11. Vérifier le bon fonctionnement de l'ensemble.

Signaler la progression du téléchargement

Une seconde version de la fonction de téléchargement du service **onHandleIntent()** est fournie dans le projet initial. Cette version permet de suivre l'évolution du téléchargement par pas de 1ko.

A la lecture du code (que vous pouvez améliorer) vous remarquerez que deux variables permettent de suivre la progression : **tailleTotale** qui contient la taille du fichier à télécharger et **sommeLue** qui évolue au rythme du téléchargement par pas de 1ko max.

12. Commenter la précédente version de la méthode **onHandleIntent()** afin de la remplacer par cette nouvelle version avec prise en charge de la progression du téléchargement.
13. Coder la méthode **publishResultsProgress()** afin qu'elle diffuse l'intention signalant la progression. L'action correspondante est déclarée dans la variable String nommée **NOTIFICATION_PROGRESS**. Les paramètres attendus et transmis par l'intention sont :
    - ✓ un entier signalant si le téléchargement se déroule normalement ou non (clé **RESULT**),
    - ✓ la taille totale en octets du fichier en cours de téléchargement (clé **TAILLETOTALE**),
    - ✓ le nombre d'octets actuellement téléchargés (clé **SOMMELUE**),
    - ✓ le chemin local de stockage du fichier en cours de téléchargement (clé **FILEPATH**).

Capturer l'intention informant sur la progression du téléchargement

L'information de progression du téléchargement doit être capturée par l'activité afin de mettre à jour la barre de progression. Vous remarquerez dans la configuration du layout de l'activité que cette barre de progression est configurée pour évoluer entre 0 et 100, ce qui représentera le pourcentage du téléchargement réalisé.

Compte tenu que la classe **BroadcastReceiver** en charge de l'intention signalant la progression doit avoir accès à ce contrôle pour le mettre à jour, une solution consiste à la déclarer comme classe intégrée de la classe **MainActivity** et déclarer l'intention d'activation avec son filtre « programmatiquement ».

Pour ceci, dans la classe **MainActivity** :

14. Déclarer une variable membre privée **ProgressReceiver** du type **BroadcastReceiver**.
15. Compléter la méthode **onCreate()** afin :
    - ✓ d'instancier l'objet **ProgressReceiver** en surchargeant sa méthode **onReceive()**,

- ✓ de coder la méthode **onReceive()** afin qu'elle récupère les informations nécessaires à la mise à jour de la barre de progression et qu'elle mette à jour cette barre de progression,
- ✓ de créer un filtre d'intention (classe **IntentFilter**) avec pour action l'intention à capturer,
- ✓ d'enregistrer un récepteur d'intention (méthode **Activity.registerReceiver()**) pour ce filtre avec pour objet de traitement de l'intention l'objet **ProgressReceiver**.

Remarque : *si besoin le code de cette question se trouve dans le fichier* ***solutionTP4_q16.txt*** *dans le dossier* ***AndroidRessources****.*

16. Dans le fichier de ressources **strings.xml**, remplacer les ressources string **FILE** et **URL** par celles qui sont en commentaires, cela afin de télécharger un fichier plus volumineux que le précédent, ce qui devrait laisser le temps de visualiser la progression du téléchargement.
17. Vérifier le bon fonctionnement de l'ensemble.

Extension possible : signaler la fin du téléchargement par un effet sonore et utiliser le service depuis une application externe

Il peut être intéressant que la fin du téléchargement soit également signalée par un effet sonore. Pour ceci vous disposez d'un fichier son **effet.mp3** dans le dossier **AndroidRessources**.

Le lecteur média d'Android est accessible au travers de la classe **MediaPlayer**, méthode statique **create()** qui attend comme paramètres le contexte de l'activité et l'identificateur de la ressource à jouer.

18. Compléter la méthode **onReceive()** de l'objet **ProgressReceiver** afin qu'elle joue le fichier son lorsque le téléchargement est terminé. (si besoin voir **solutionTP4_q18.txt**)

Enfin pour tester la possibilité d'utiliser le service depuis une application indépendante du service, un projet **ClientService** est disponible toujours dans le dossier **AndroidRessources**. Vous pouvez cependant négliger cette aide et faire votre propre client à partir d'une copie du projet **DownloadService** dans lequel vous supprimerez la fonction de service.

19. Importer ce projet dans votre **WorkSpace** et effectuer les éventuelles modifications pour qu'il utilise le service que vous avez mis au point.
20. Vérifier le bon fonctionnement de l'ensemble.

Sujet, ressources et corrigé :