# Callbacks – Async Task

Functions de rappel – Tâches synchrone et asynchrone

# Objectifs

Comprendre le principe d’une fonction de rappel.

Aborder les tâches synchrones et asynchrones.

Se préparer pour la suite Android : thread en background asynchrone.

# Fonction de rappel

Nous avons déjà abordé les fonctions de rappel lorsque nous avons vu le framework JavaScript Vue.js, notamment avec la notion de ***Promesse*** (promise) lorsque nous avons étudié le composant / librairie axios qui permet d’effectuer des requêtes asynchrones http vers des API.

Les fonctions de rappel sont au cœur des framework Javascript, cf par exemple JQuery (<https://www.tutorialrepublic.com/jquery-tutorial/jquery-callback.php>).

**Définition** :

Une fonction de rappel est une fonction qui est passée comme argument à une autre fonction et qui doit s’exécuter après qu’un évènement particulier s’est produit.

**Usage en Java** :

Pour la gestion d’évènement, lorsque par exemple un bouton est cliqué et que l’utilisateur est informé via la fonction de rappel associée au bouton.

Ce type de pattern est utilisé par exemple pour le design pattern Observer. Cf <https://www.baeldung.com/java-observer-pattern>

En java , les Callbacks (fonctions de rappel) peuvent être implémentés en utilisant une interface.

**Procédure** d’implémentation:

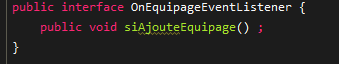
1. Dans une interface, définir le ou les méthodes que l’on souhaite appeler après le rappel (callback)
2. Définir une classe qui va implémenter les méthodes de callback de l’interface
3. Définir une référence dans la classe qui va émettre l’évènement et la méthode de callback.
4. Enregistrer la référence
5. Utiliser cette référence pour appeler la méthode de callback (rappel)

# Synchrone - Fonction de rappel Synchrone

Un appel Synchrone s’effectue dans l’ordre.

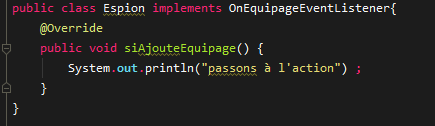
**Exemple :**

Etape 1 de la procédure : L’interface définit une méthode siAjouteEquipage , qui sera implémentée à l’ étape 2 et appelée à l’étape 5

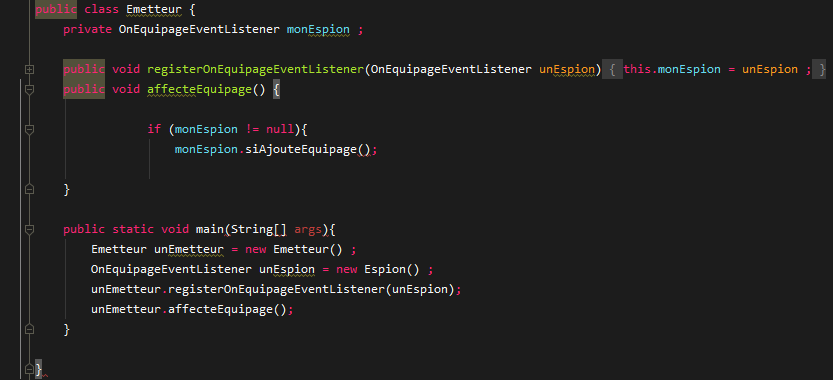


Etape 2 de la procédure :

La classe Espion ici implémente l’interface. Elle se contente d’afficher un petit message …



Etape 3, 4 et 5



La méthode registerOnEquipageEventListener réceptionne et initialise l’interface.

La méthode *affecteEquipage* se charge d’appeller la méthode de callback (étape 5 de la procédure) après avoir éventuellement effectué ce qu’elle avait à faire (ce qui n’est pas le cas dans l’exemple ici).

Ici l’exécution (méthode main) se fait dans la même classe : elle pourrait très bien avoir lieu dans une autre classe, ou une classe de test pour valider que la fonction de rappel est bien exécutée quand la méthode *affecteEquipage* s’exécute.

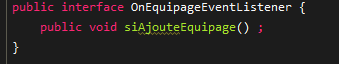
On remarquera que l’attribut monEspion de type OnEquipageEventListener est instanciée comme classe concrète Espion qui implémente l’interface correspondante c’est-à-dire la méthode de callback qui va être appelée.

# Asynchrone - Fonction de rappel asynchrone

Un appel asynchrone ne bloque pas le programme.

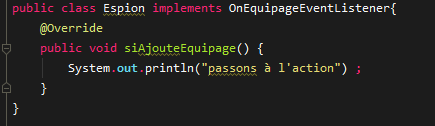
**Exemple :**

Etape 1 de la procédure : L’interface définit une méthode siAjouteEquipage , qui sera implémentée à l’ étape 2 et appelée à l’étape 5



Etape 2 de la procédure :

La classe Espion ici implémente l’interface. Elle se contente d’afficher un petit message …



Etape 3, 4 et 5



Ce qui change ici par rapport à l’appel synchrone est la création d’un Thread et d’une classe anonyme Runnable avec une méthode run dans laquelle est appelé le Callback.

Sinon mêmes remarques et étapes que pour l’appel synchrone.

La méthode registerOnEquipageEventListener réceptionne et initialise l’interface.

La méthode affecteEquipage s’occupe de lancer un nouveau Thread asynchrone et appelle la méthode de callback (étape 5 de la procédure).

Ici l’exécution se fait dans la même classe : elle pourrait très bien avoir lieu dans une autre classe, ou une classe de test pour valider que la fonction de rappel est bien exécutée quand la méthode affecteEquipage s’exécute.

On remarquera que l’instance de OnEquipageEventListener est instanciée comme classe concrète Espion qui implémente l’interface correspondante c’est-à-dire la méthode de callback qui va être appelée.

Les appels asynchrones via un Thread et une classe anonyme comme dans l’exemple ci-dessous sont très largement utilisés sous Android.