

**BILLOT – LUDWIG – LATTRECHE Lycée La Briquerie STIR2 STIR2**

HoneyBee

Pour Particulier

## Table des matières :

Contenu

[Table des matières : 48](#_Toc420605303)

[2 Introduction 50](#_Toc420605304)

[3 Présentation D’Android 51](#_Toc420605305)

[4 Les Environnements de Développements 52](#_Toc420605306)

[4.1 Eclipse 52](#_Toc420605307)

[4.2 Android Studio 53](#_Toc420605308)

[5 Protocole UDP 54](#_Toc420605309)

[5.1 Définition 54](#_Toc420605310)

[5.2 Structure d’un datagramme UDP 54](#_Toc420605311)

[5.3 Utilisation de l’UDP 55](#_Toc420605312)

[6 Test unitaire 56](#_Toc420605313)

[6.1 Connexion UDP entre deux PC 56](#_Toc420605314)

[6.2 Compréhension de l’univers Android 58](#_Toc420605315)

[6.2.1 Général 58](#_Toc420605316)

[6.2.2 Android manifest .xml 59](#_Toc420605317)

[6.2.3 Code java 60](#_Toc420605318)

[6.2.4 Layout 61](#_Toc420605319)

[6.3 Connexion UDP PC-ANDROID 61](#_Toc420605320)

[6.4 Récupérer les données 63](#_Toc420605321)

[7 APPLICATION FINAL 65](#_Toc420605322)

[7.1 Le Service 65](#_Toc420605323)

[7.2 Fonctionnement du service : 65](#_Toc420605324)

[7.3 Utilisation du Service 66](#_Toc420605325)

[7.4 Service UDP 66](#_Toc420605326)

[7.4.1 Méthode onCreate() 66](#_Toc420605327)

[7.4.2 Méthode onStartCommand() 67](#_Toc420605328)

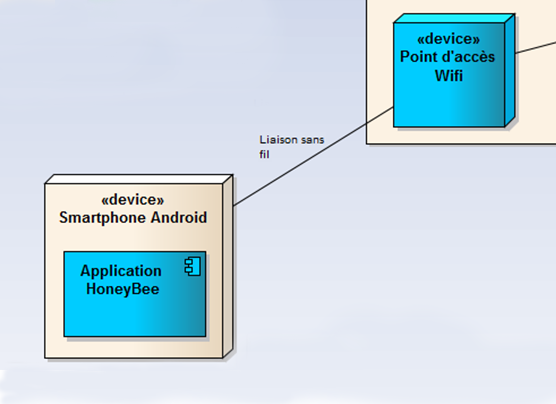
[7.5 WakeLock 69](#_Toc420605329)

[7.6 Les notifications 69](#_Toc420605330)

[7.7 Le bouton Retour 70](#_Toc420605331)

[7.8 Graphisme final 71](#_Toc420605332)

# Introduction



Ma partie du projet consiste à envoyer des données telles que le poids, la température de la ruche et la tension du calculateur Panda pour vérifier son niveau de batterie. La transmission des données se fait par wifi en utilisant le protocole UDP (***User Datagram Protocol).*** L’utilisateur pourra alors voir les différentes données sur son Smartphone du jour, c’est-à-dire les données du matin dans les environs de 11h et 12h et aussi les données du soir dans les environs de 19h et 20h. L’utilisateur recevra aussi des mails en cas d’alerte importante de la ruche, par exemple si le poids est beaucoup trop faible ou si la batterie est faible.

# Présentation D’Android



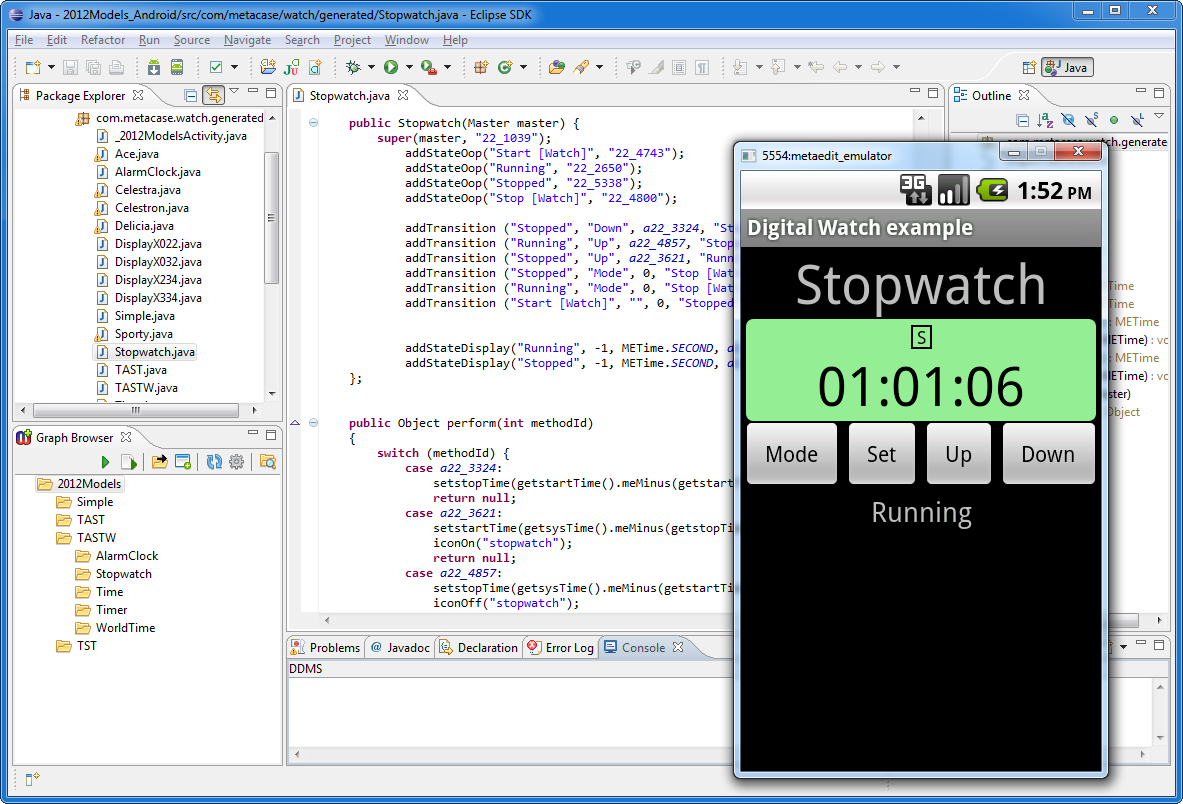
Android est un système d’exploitation mobile pour Smartphones, tablettes tactiles, PDA, montre connectée et terminaux mobiles. C’est un système open source utilisant le noyau Linux. Il a été lancé par une startup du même nom rachetée par Google en 2005. D’autres types d’appareils possédant ce système d’exploitation existent, par exemple des téléviseurs, des radioréveils, des autoradios et même des voitures.



\*Exemple d’un Smartphone fonctionnant sous Android, le Google Nexus 5.

# Les Environnements de Développements

## Eclipse



**Eclipse** est un projet, décliné et organisé en un ensemble de sous-projets de développements logiciels, de la Fondation Eclipse visant à développer un environnement de production de logiciels libres qui soit extensible, universel et polyvalent, en s'appuyant principalement sur [Java](http://fr.wikipedia.org/wiki/Java_(technique)).

Son objectif est de produire et fournir des outils pour la réalisation de logiciels, englobant les activités de programmation (notamment environnement de développement intégré et frameworks) mais aussi d'AGL recouvrant modélisation, conception, testing, gestion de configuration, reporting… Son EDI, partie intégrante du projet, vise notamment à supporter tout langage de programmation à l'instar de Microsoft Visual Studio.



Pour développer sur Android avec Eclipse il nous faut ajouter l’ADT ( Android Developer Tools) pour que le développement d’application puisse se dérouler

## Android Studio



En 2014 est sorti l’environnement de développement créé par Google, intégrant directement tout ce qu’il faut pour programmer sur Android, c’est de ce fait que j’ai choisi d’utiliser ce logiciel de développement.

Android Studio représente la plateforme officielle, soutenue par Google, pour le développement d’applications Android. Il repose sur IntelliJ (Community Edition) de JetBrain et devrait permettre aux développeurs d’être plus rapides et plus productifs, puisqu’il intègre directement tout ce qu’il faut pour programmer sur Android.

# Protocole UDP

## Définition

Le ***User Datagram Protocol*** (**UDP**, en français **protocole de datagramme utilisateur**) est un des principaux protocoles de télécommunication utilisé par Internet. Il fait partie de la couche transport de la pile de protocole TCP/IP : dans l'adaptation approximative de cette dernière au modèle OSI, il appartiendrait à la couche 4, comme TCP.

Le rôle de ce protocole est de permettre la transmission de données de manière très simple entre deux entités, chacune étant définie par une adresse IP et un numéro de port. Contrairement au protocole TCP, il fonctionne sans négociation : il n'existe pas de procédure de connexion préalable à l'envoi des données. Donc UDP ne garantit pas la bonne livraison des datagrammes à destination, ni leur ordre d'arrivée. Il est également possible que des datagrammes soient reçus en plusieurs exemplaires.

L'intégrité des données est assurée par une somme de contrôle sur l'en-tête. L'utilisation de cette somme est cependant facultative en IPv4 mais obligatoire avec IPv6. Si un hôte n'a pas calculé la somme de contrôle d'un datagramme émis, la valeur de celle-ci est fixée à zéro. La somme de contrôle inclut les adresses IP source et destination.

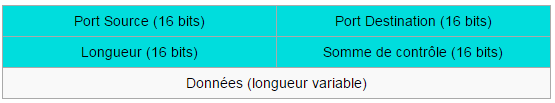
La nature de UDP le rend utile pour transmettre rapidement de petites quantités de données, depuis un serveur vers de nombreux clients ou bien dans des cas où la perte d'un datagramme est moins gênante que l'attente de sa retransmission. Le DNS, la voix sur IP ou les jeux en ligne sont des utilisateurs typiques de ce protocole.

## Structure d’un datagramme UDP

Le paquet UDP est encapsulé dans un paquet IP. Il comporte un en-tête suivi des données proprement dites à transporter.



L'en-tête d'un datagramme UDP est plus simple que celui de [TCP](http://fr.wikipedia.org/wiki/Transmission_Control_Protocol)



Il contient les quatre champs suivants :

**Port Source**

Indique depuis quel port le paquet a été envoyé.

**Port de Destination**

Indique à quel port le paquet doit être envoyé.

**Longueur**

Indique la longueur totale (exprimée en octets) du segment UDP (en-tête et données). La longueur minimale est donc de 8 octets (taille de l'en-tête).

**Somme de contrôle**

Celle-ci permet de s'assurer de l'intégrité du paquet reçu quand elle est différente de zéro. Elle est calculée sur l'ensemble de l'en-tête UDP et des données, mais aussi sur un pseudo en-tête (extrait de l'en-tête IP)

## Utilisation de l’UDP

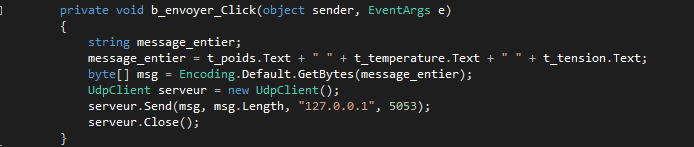
L’UDP est utilisé quand il est nécessaire soit de transmettre des données très rapidement et où la perte d'une partie de ces données n'a pas grande importance, soit de transmettre des petites quantités de données, là où la connexion TCP serait inutilement coûteuse en ressources. Par exemple, dans le cas de la transmission de la voix sur IP, la perte occasionnelle d'un paquet est tolérable dans la mesure où il existe des mécanismes de substitution des données manquantes, par contre la rapidité de transmission est un critère primordial pour la qualité d'écoute.

# Test unitaire

## Connexion UDP entre deux PC

Mon premier test unitaire constituait à réaliser une communication UDP entre deux PC pour comprendre comment fonctionne le protocole UDP.

Code du serveur :

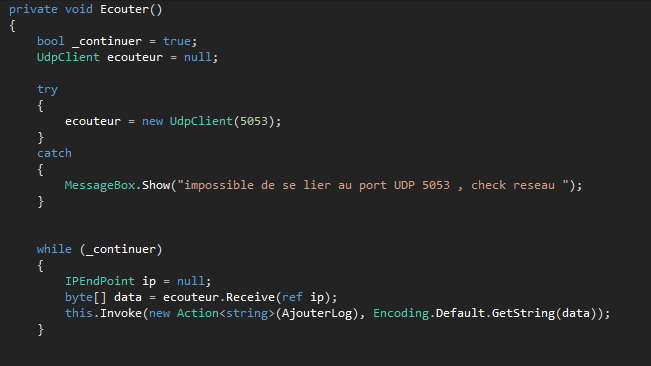


Pour envoyer des trames UDP il faut que les données soient des bytes, pour ce faire je regroupe toute mes données en un string et l’insère dans un tableau de byte. Je crée ensuite un objet de type UdpClient qui permettra de créer une connexion UDP. La méthode send() permet d’envoyer le paquet à l’adresse choisit (ici 127.0.0.1) , mais la méthode prend différent paramètre comme le tableau de byte a envoyé , sa taille , l’adresse IP et le port sur lequel elle est émise.

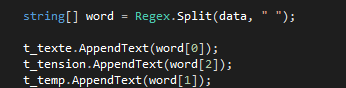
Code du récepteur :



On créer un Thread pour ne pas bloquer le processus principal

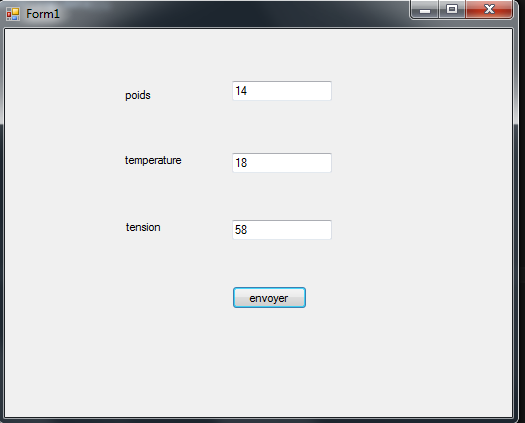


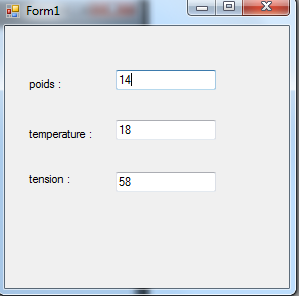
Dans cette méthode écouter() , on crée un objet UdpClient qui écoutera sur le port 5053 pour recevoir des données , dès qu’il entend quelque chose sur le port 5053, il le récupère.



On affiche ensuite ces données sur l’interface principale.

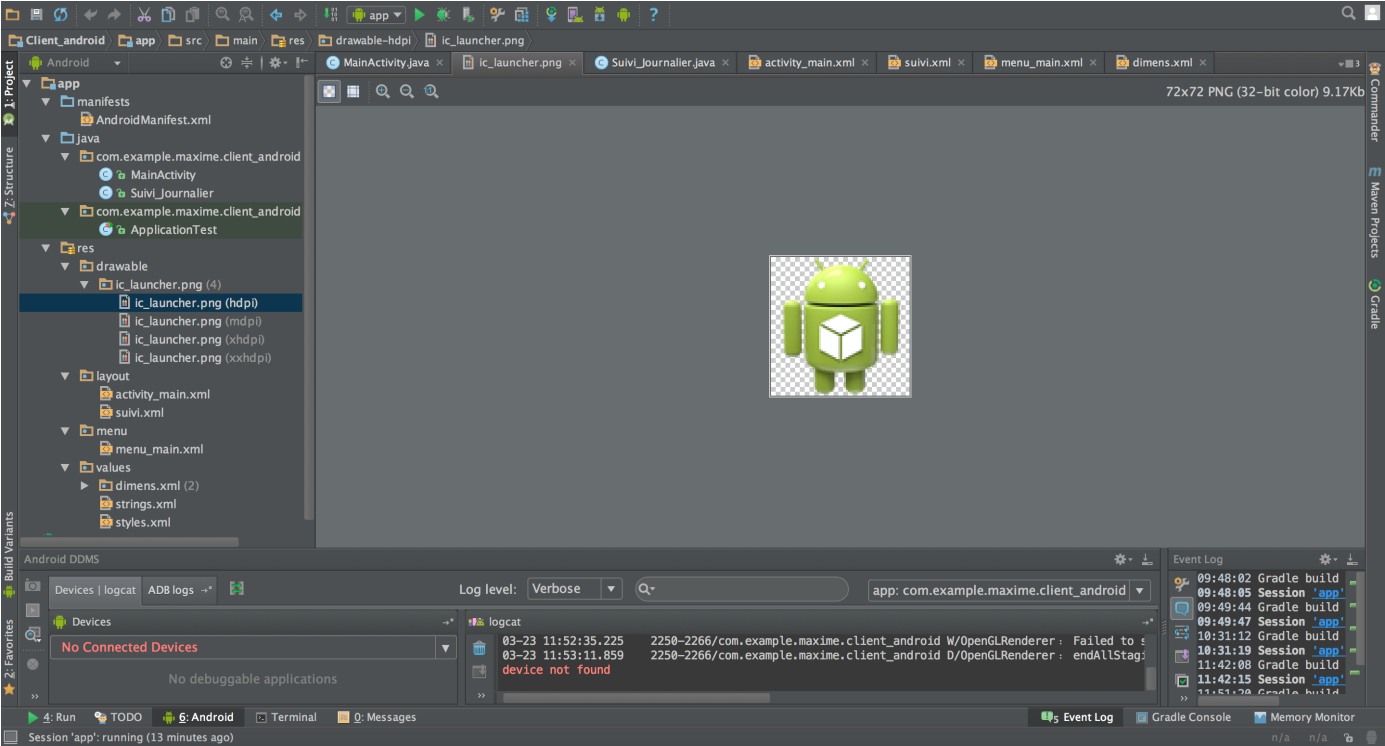
Exemple :





## Compréhension de l’univers Android

### Général



Gestion des fichiers :

* Manifest : contient l’AndroidManifest.xml : définit le comportement de l’application au système android, ce fichier définit le nom, la version du SDK, les activités et les services
* MainActivity et Suivi\_Journalier sont mes principales activités, c’est ici que j’écris le code.
* Dossier res : ce dossier comporte les ressources de l’application (images, vidéo, styles)
* Drawable-hdpi : contient les images en hautes résolution
* Drawable-ldpi : contient les images en basse résolution
* Drawable-mdpi : contient les images en moyenne résolution
* Dossier layout : dossier décrivant les différentes interfaces
* Activity\_main.xml : le fichier principal de mon interface
* Dossier values : c’est un dossier contenant les valeurs de notre application, on peut par exemple y mettre des chaines de caractères ou des tableaux.
* String.xml : contient les déclarations des chaines de caractères.

### E:\Projet 2015\capture d'ecran client_android\Capture d’écran 2015-03-23 à 13.49.31.pngAndroid manifest .xml

- Le package (identifiant) de notre application (ici com.example.maxime.client\_android) est défini dans l'attribut **package**de la balise manifeste.

- L'icône et le nom de notre application sont précisés par les attributs **icon**et **label**de la balise application ainsi que le thème utilisé par l'application (cf. chapitre Personnalisation et gestion d'événements - Personnalisation).

- La description de l'activité principale (balise **activity**) :  
 Le nom de la classe qui implémente Activity.  
 Le titre pour l'activité.  
 Des filtres sur l'activité :  
  - **MAIN**: indique qu'il s'agit de l'activité principale de l'application.  
  - **LAUNCHER**: indique que cette activité est présente dans le lanceur d'application.

### Code java

Android a la particularité de programmer en JAVA.

Le langage Java est un langage de programmation informatique orienté objet créé par James Gosling et Patrick Naughton, employés de Sun Microsystems, avec le soutien de Bill Joy (cofondateur de Sun Microsystems en 1982), présenté officiellement le 23 mai 1995 au SunWorld.

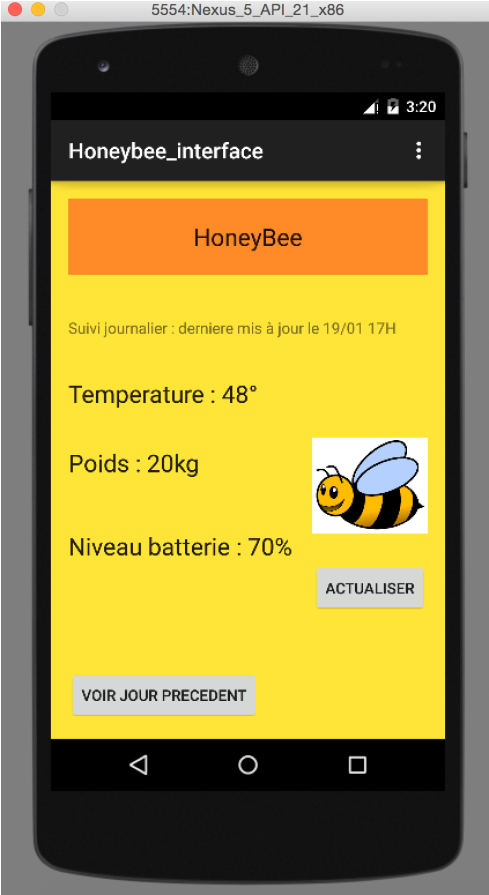
La société Sun a été ensuite rachetée en 2009 par la société Oracle qui détient et maintient désormais Java.

La particularité et l'objectif central de Java est que les logiciels écrits dans ce langage doivent être très facilement portables sur plusieurs systèmes d’exploitation tels que UNIX, Windows, Mac OS ou GNU/Linux, avec peu ou pas de modifications. Pour cela, diverses plateformes et frameworks associés visent à guider, sinon garantir, cette portabilité des applications développées en Java.



### Layout

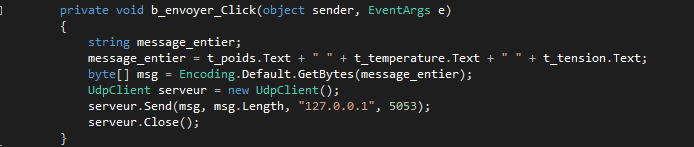
Les layouts permettent de créer des interfaces graphiques sur Android , il permet par exemple d’afficher des boutons , des textboxs ou encore des images .



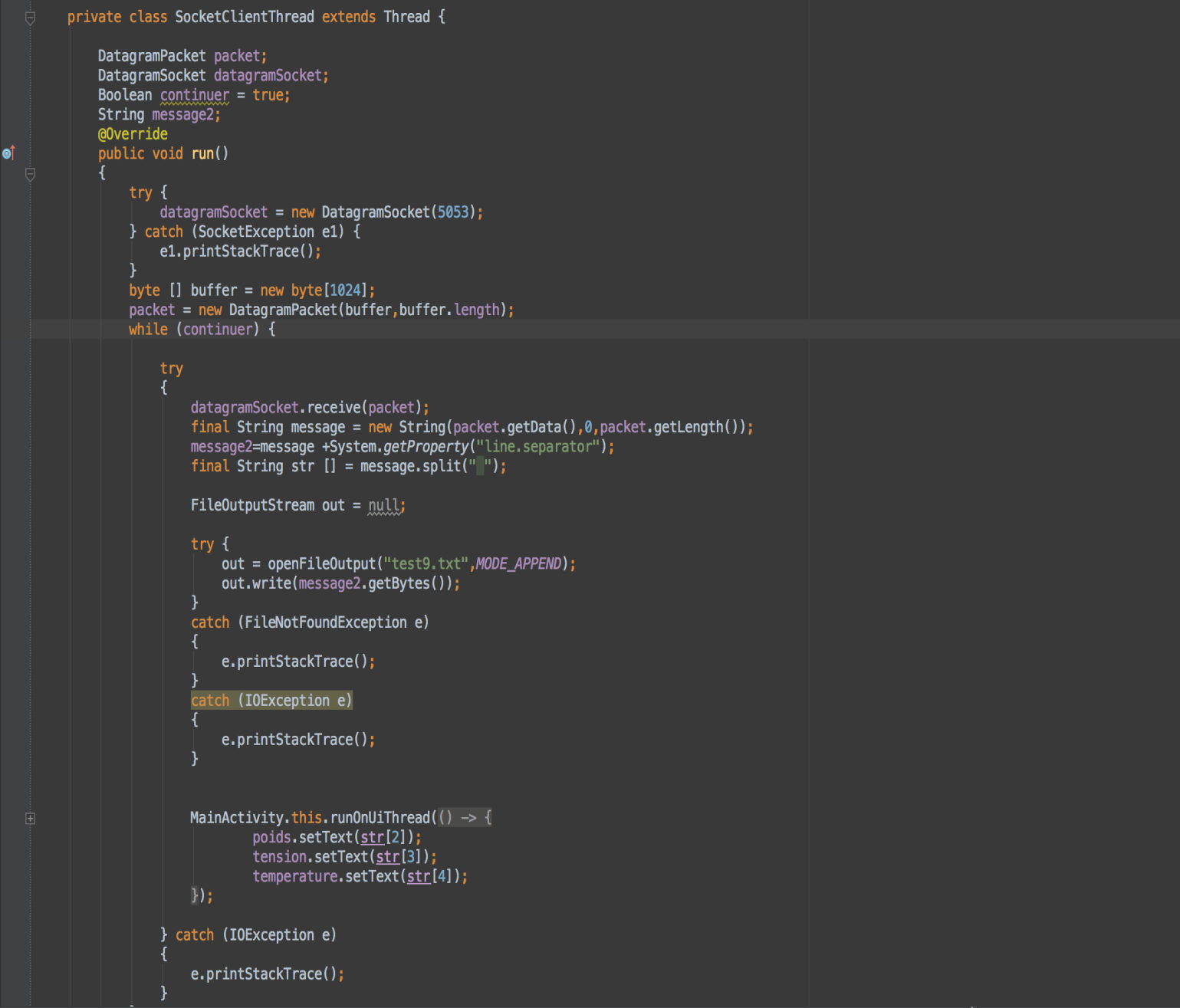
## Connexion UDP PC-ANDROID

Le test unitaire que j’ai effectué ensuite est la connexion entre Un pc et un Smartphone Android dans le même réseau wifi.

Je reprends alors le code de l’émetteur précédant



Le changement se fait dans le code Android



Je crée une classe publique Thread me permettant à n’importe quel moment de recevoir les informations du PC. J’enregistre dans un fichier texte la ligne qui vient d’être envoyée. Chaque donnée est séparée d’un espace pour pouvoir ensuite splité au bon endroit et récupérer les données que je souhaite.

J’affiche ensuite les données sur l’interface du Smartphone :



## Récupérer les données

La suite de mon application est de récupérer les valeurs des jours précédents, pour se faire je lis les informations dans mon fichier texte je vérifie que la date corresponde à la date que l’utilisateur a choisi, puis s’il voulait les données de 12h ou de 19h. L’utilisateur devra appuyer sur le bouton rechercher pour afficher les résultats

.



# APPLICATION FINAL

## Le Service

Contrairement aux threads, les services sont conçus pour être utilisés sur une longue période de temps. En effet, les threads sont des éléments sommaires qui n'ont pas de lien particulier avec le système Android, alors que les services sont des composants et sont par conséquent intégrés dans Android au même titre que les activités. Ainsi, ils vivent au même rythme que l'application. Si l'application s'arrête, le service peut réagir en conséquence, alors qu'un thread, qui n'est pas un composant d'Android, ne sera pas mis au courant que l'application a été arrêtée si vous ne lui dites pas. Il ne sera par conséquent pas capable d'avoir un comportement approprié, c'est-à-dire la plupart du temps de s'arrêter.

Le service nous sera donc utile quand l’utilisateur aura l’application en fond de tâche, il permettra de recevoir les données à n’importe quel moment même si l’application est tuée, le service ne le sera pas.

J’ai donc créé une classe ServiceUDP :

## Fonctionnement du service :



OnCreate() : Cette méthode est appelée à la création du service et est en général utilisée pour initialiser ce qui sera nécessaire à votre service.

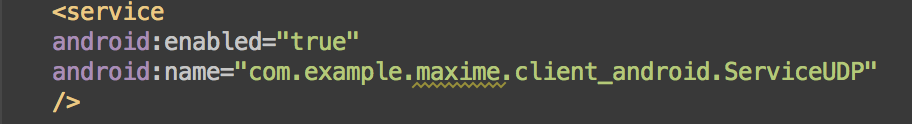
OnStart(Intent i) : Le service démarre. Valable uniquement pour les versions du SDK inférieur à 2.0.

OnStartCommand(Intent i, int flags, int startId) : Le service démarre. Valable uniquement pour les versions duSDK supérieur à 2.0.

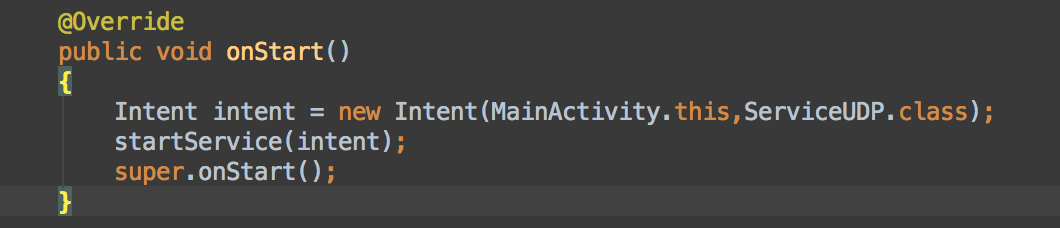
OnDestroy() : Appelé à la fermeture du service.

## Utilisation du Service

Il faut d’abord le déclarer dans le Androidmanifest.xml

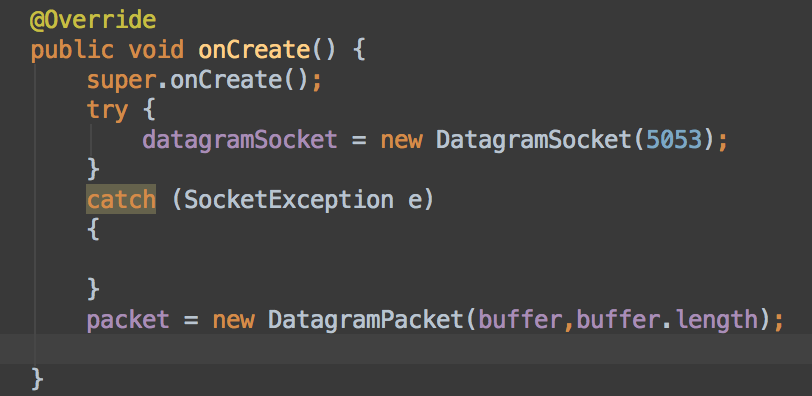


Je l’initialise dans le OnStart() de l’application principale , c’est-à-dire dès que l’application Honeybee se lance , le service sera lancé.



## Service UDP

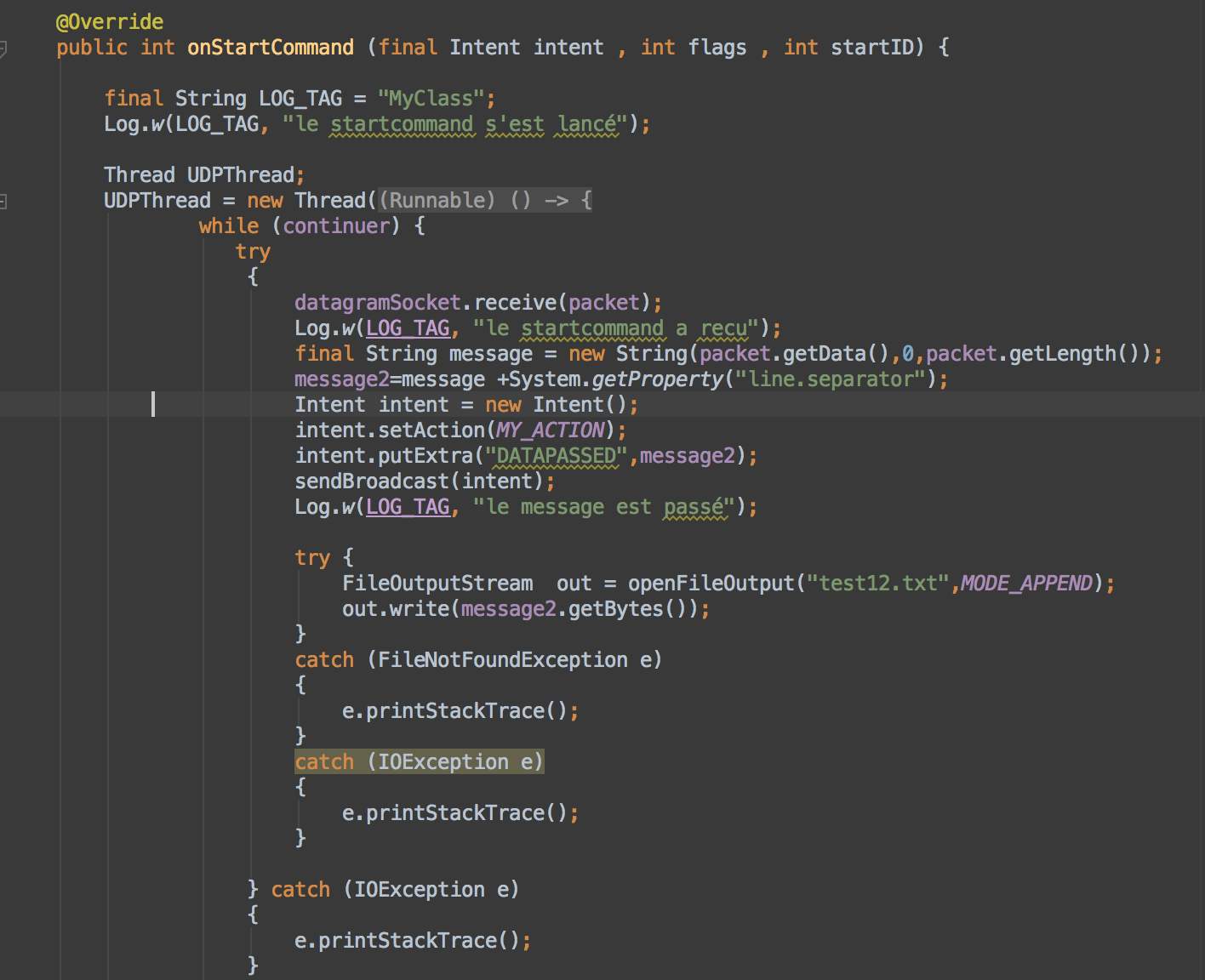
### Méthode onCreate()



.

Dans le onCreate() on initialise le DatagramSocket et le DatagramPacket.

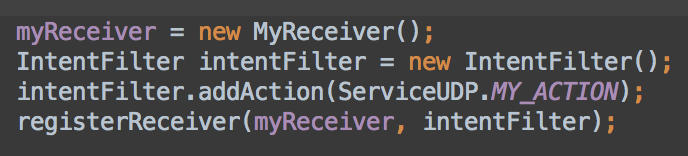
### Méthode onStartCommand()



On crée un Thread pour ne pas bloquer le processus principal, ensuite on récupère la trame envoyée avec datagramSocket.receive(packet).On stocke ensuite cette donnée dans un string.  
On crée ensuite un Intent pour faire passer la valeur du string « message2 » dans le MainActivity pour l’afficher à l’utilisateur les données reçues.



On crée une classe qui hérite de BroadcastReceveir qui permet à l’activité principale de récupérer les valeurs du Service.



On crée une instance de la classe MyReceiver , puis avec L’intentFilter on execute le code de l’objet myReceiver.

A la fin de OnStartCommand() dans notre cas nous avons un return « Service.StartSTicky » mais il y aussi des autres valeurs de retour possibles.

##### START\_NOT\_STICKY

Si le système tue le service, alors ce dernier ne sera pas recréé. Il faudra donc effectuer un nouvel appel à startService() pour relancer le service.

Ce mode vaut le coup dès qu'on veut faire un travail qui peut être interrompu si le système manque de mémoire et que vous pouvez le redémarrer explicitement par la suite pour recommencer le travail. Si vous voulez par exemple mettre en ligne des statistiques sur un serveur distant. Le processus qui lancera la mise en ligne peut se dérouler toutes les 30 minutes, mais, si le service est tué avant que la mise en ligne soit effectuée, ce n'est pas grave, on le refera dans 30 minutes.

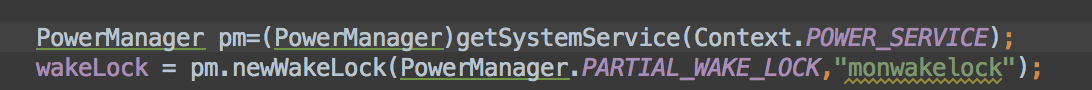
##### START\_STICKY

Cette fois, si le système doit tuer le service, alors il sera recréé mais sans lui fournir le dernier Intent qui l'avait lancé. Ainsi, le paramètre intent vaudra null. Ce mode de fonctionnement est utile pour les services qui fonctionnent par intermittence, comme par exemple quand on joue de la musique.

##### START\_REDELIVER\_INTENT

Si le système tue le service, il sera recréé et dans onStartCommand() le paramètre intent sera identique au dernier intent qui a été fourni au service. START\_REDELIVER\_INTENT est indispensable si vous voulez être certain qu'un service effectuera un travail complètement.

## WakeLock



Le WakeLock permet au service de tourner même quand l’écran du smartphone est éteint, il y a plusieurs options possible :

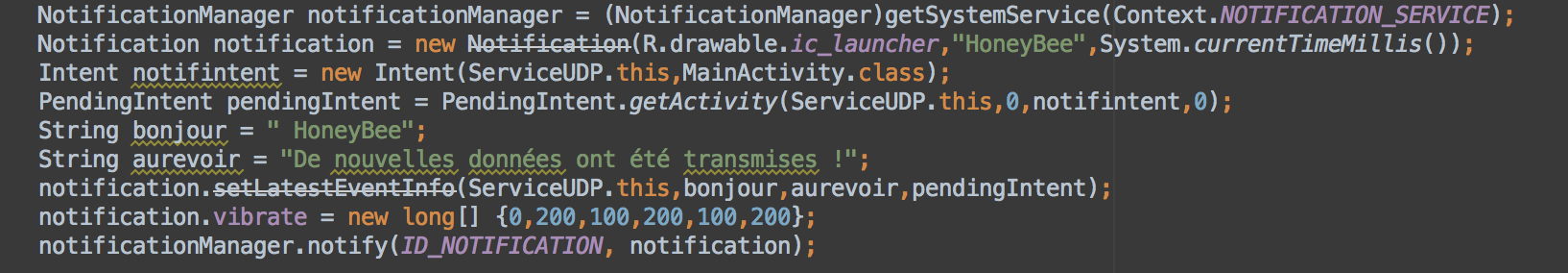
FULL\_WAKE\_LOCK - garde l’écran et le clavier allumé et le processeur travaille

SCREEN\_BRIGHT\_WAKE\_LOCK – garde l’écran allumé et le processeur travaille

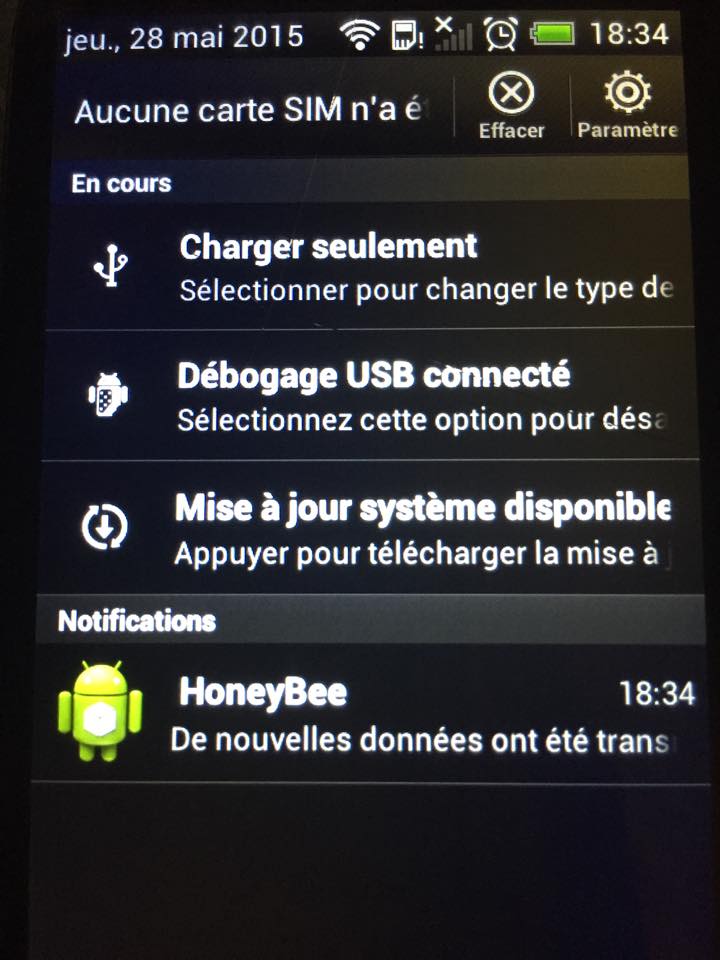
PARTIAL\_WAKE\_LOCK - seul le processeur travaille.

## Les notifications

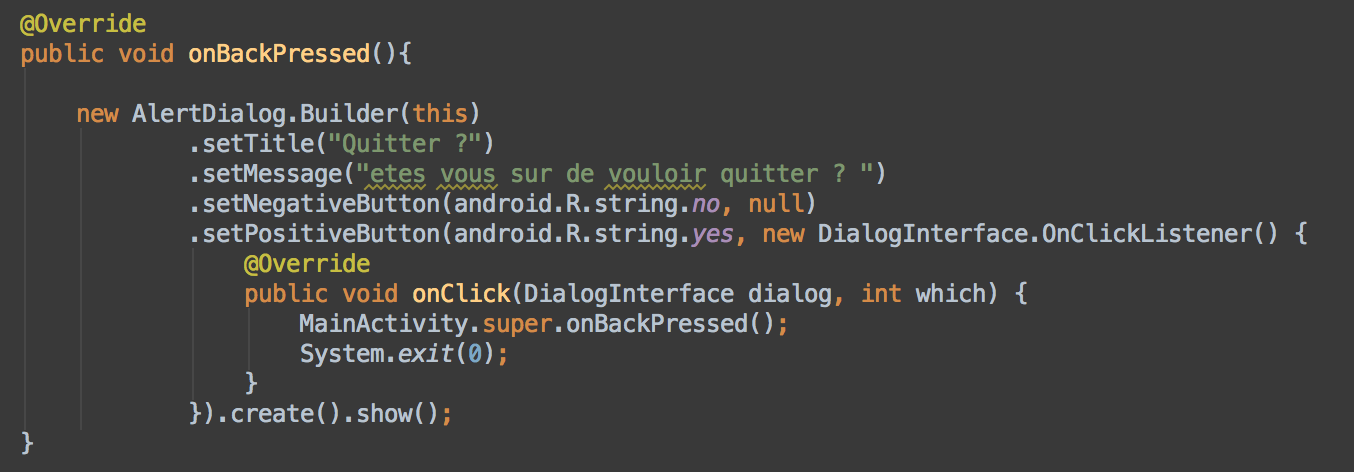
Une notification est une indication qui s'affiche sur la barre qui se situe en haut d'un téléphone Android. Cette notification sert à prévenir un utilisateur de certains évènements, comme la réception d'un message par exemple.

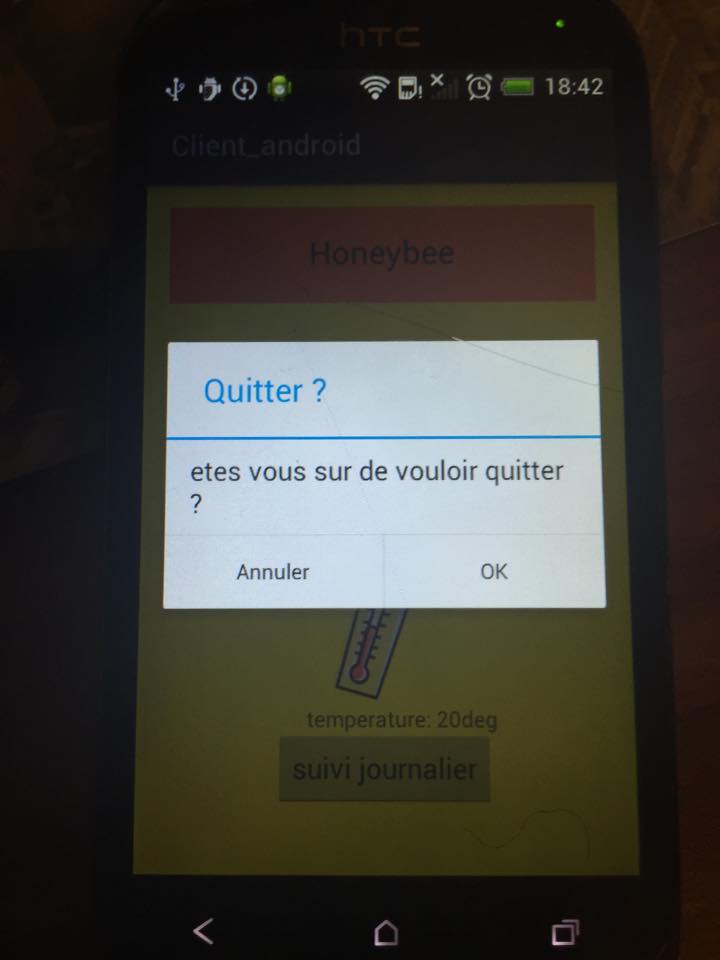


On crée une notification avec comme nom « honeybee » l’image ic.launcher , on crée aussi une vibration au moment de recevoir les données.



## Le bouton Retour



Quand l’utilisateur appuyera sur la touche retour de son smartphone, l’application affichera une boîte de dialogue , où il pourra ensuite decider s’il veut complètement quitter l’application ou pas.

## Graphisme final

