Les connexions sans fil avec Android

Plan de l'exposé

- Wi-Fi : un rappel
- Démo 1 : trouver les réseaux Wi-Fi
- Démo 2 : repérer les appareils connectés sur notre réseau Wi-Fi
- Démo 3 : Par programmation, le smartphone devient un hotspot
- Démo 4 : Faire communiquer deux smartphones par Wi-Fi direct (tethering)

Rappel (?) Wi-Fi (1/2)

- Wi-Fi = ensemble de protocoles de communication sans fil régis par les normes du groupe IEEE 802.11
- couche physique et liaison
- = jeu de mots avec Hi-Fi ?
- Rayon de plusieurs dizaines de mètres en intérieur (généralement entre une vingtaine et une cinquantaine de mètres)
- Points d'accès Wi-Fi = bornes Wi-Fi = hot spots
- Nom de réseau Wi-Fi = SSID = Service Set IDentifier
- Le mode "Ad-Hoc" permet de connecter directement les ordinateurs équipés d'une carte Wi-Fi, sans utiliser de point d'accès. Utile pour échanger des données entre portables dans un train, dans la rue, au café, ... = Wi-Fi direct = Wi-Fi Peer-to-Peer = Wi-Fi P2P
- source : `http://fr.wikipedia.org/wiki/Wi-Fi`

Rappel (?) Wi-Fi (2/2)

- WPA et WPA2 (Wi-Fi Protected Access) sont des mécanismes de cryptage dans les réseaux Wi-Fi plus puissants que le WEP (Wired Equivalent Privacy)

- source : `http://fr.wikipedia.org/wiki/Wi-Fi`

Wi-Fi P2P : présentation

- The Wi-Fi peer-to-peer (P2P) APIs allow applications to connect to nearby devices without needing to connect to a network or hotspot (Android's Wi-Fi P2P framework complies with the Wi-Fi Direct™ certification program). Wi-Fi P2P allows your application to quickly find and interact with nearby devices, at a range beyond the capabilities of Bluetooth
- source : `http://developer.android.com/training/connect-devices-wirelessly/wifi-direct.html`
- C'est clair non ?
- Les appareils sous Android (au dessus des versions 4.1) peuvent communiquer entre eux et entre d'autres appareils lorsqu'ils sont proches
- ~ Bluetooth ou Wi-Fi

Démo 1

Trouver les réseaux Wi-Fi

Android = architecture de composants (rappel ?)

- Les principales classes développées sont des composants
- Composant = objet dont le cycle de vie, le lancement de certaines méthodes est pris en charge par l'environnement d'exécution
- = ce n'est pas l'utilisateur, ni le développeur qui décident quand certains codes, certains chargements sont lancés. C'est l'environnement d'exécution (= Android)
- => le développement doit suivre des règles de programmation : dériver de certaines classes, développer certaines méthodes, etc.
- = architecture de framework : cf. applet, servlet, EJB, ...
- Les composants fondamentaux sont : les `Activity`, les `Services`, les `BroadcastReceiver`, les `ContentProvider`

Découverte des réseaux Wi-Fi

- source : `http://www.androidsnippets.com/scan-for-wireless-networks`
- Démo dans `1DecouverteReseauWiFi` sur le téléphone Nexus S ou la tablette galaxy Tab
- 1°) On récupère le gestionnaire des services Wi-Fi
- 2°) On construit un `BroadcastReceiver` et on l'enregistre de sorte à être déclanché pour les événements "la recherche des réseaux Wi-Fi est terminé et le résultat de cette recherche est disponible" (="An access point scan has completed, and results are available from the supplicant.")
- 3°) On demande au gestionnaire des services Wi-Fi de lancer la recherche des réseaux Wi-Fi
- 4°) Le `BroadcastReceiver` est averti par l'environnement d'exécution et affiche la liste des réseaux Wi-Fi

Découverte des réseaux Wi-Fi : l'activité principale (1/2)

- Toutes ces étapes sont écrites dans la méthode `onCreate()` qui possède le code :

```
WifiManager mainWifi;
MonWifiReceiver receiverWifi;
...
public void onCreate(Bundle savedInstanceState) {
  ...
  mainWifi = (WifiManager) getSystemService(Context.WIFI_SERVICE);
  // indiquer qu'on veut être averti lorsque des réseaux WiFi auront été trouvés
  // (cf. BroadcastReceiver voir diapos suivantes)
  mainWifi.startScan();
  ...
}
```

- `startScan()` est une méthode de la classe `android.net.wifi.WifiManager`

Découverte des réseaux Wi-Fi : l'activité principale (2/2)

- Avoir le résultat de la liste des réseaux Wi-Fi proches ne peut pas être immédiat
- `startScan()` lance ce travail et n'est (heureusement) pas bloquant
- Lorsque le résultat de ce travail (obtenir la liste des réseaux Wi-Fi proches) sera connu, il faudra que l'application Android en soit informée
- On est en pleine programmation asynchrone !

Utiliser un `BroadcastReceiver`

- Souvent la communication entre ces composants est faite à l'aide d'`Intent`
- C'est l'environnement d'exécution (Android) qui reçoit et envoie les `Intent` : Android est un aiguilleur !
- C'est aux composants développés d'indiquer à quel `Intent` il sont sensibles
- Lorsque Android a reçu les indications justifiant la génération de l'`Intent`, il lance la méthode adaptée du composant sensible à cet `Intent`
- Ici, on écrit donc un `BroadcastReceiver` (= un objet d'une classe dérivée de `android.content.BroadcastReceiver`) et on l'enregistre dans l'application en indiquant pour quel `Intent` il est sensible

Découverte des réseaux Wi-Fi : le `BroadcastReceiver`

- Le `BroadcastReceiver` est une classe (`MonWifiReceiver`) développée par le programmeur :

```
import android.net.wifi.ScanResult;
...
class MonWifiReceiver extends BroadcastReceiver {
     public void onReceive(Context c, Intent intent) {
          sb = new StringBuilder();
          wifiList = mainWifi.getScanResults();

          for(int i = 0; i < wifiList.size(); i++){
               sb.append((i+1) + ".");
               sb.append((wifiList.get(i)).toString());
               sb.append("\n----------------------------");
               sb.append("\n");
          }
          mainText.setText(sb);
     }
}
```

- On utilise la méthode
`public List<ScanResult> getScanResults ()`
de la classe `WifiManager` qui retourne la liste des points d'accès au réseau Wi-FI

Enregistrement du `BroadcastReceiver`

- `registerReceiver()` est une méthode de `Context` (donc d'une `Activity`) permettant d'enregistrer des `BroadcastReceiver` sensibles à certains `Intent`

```
WifiManager mainWifi;
MonWifiReceiver receiverWifi;
...

  mainWifi = (WifiManager) getSystemService(Context.WIFI_SERVICE);
  receiverWifi = new MonWifiReceiver();
  registerReceiver(receiverWifi, new
         IntentFilter(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION));
  ...
```

Et pour terminer l'activité !

- L'activité a les méthodes :

```
protected void onPause() {
     unregisterReceiver(receiverWifi);
     super.onPause();
}

protected void onResume() {
  registerReceiver(receiverWifi, new IntentFilter(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION));
  super.onResume();
}
```

Découverte des réseaux Wi-Fi : le `Manifest`

- L'`AndroidManifest.xml` doit contenir les permissions (filles de l'élément `manifest`) :

```
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Démo 2

Repérer les appareils connectés sur notre réseau Wi-Fi

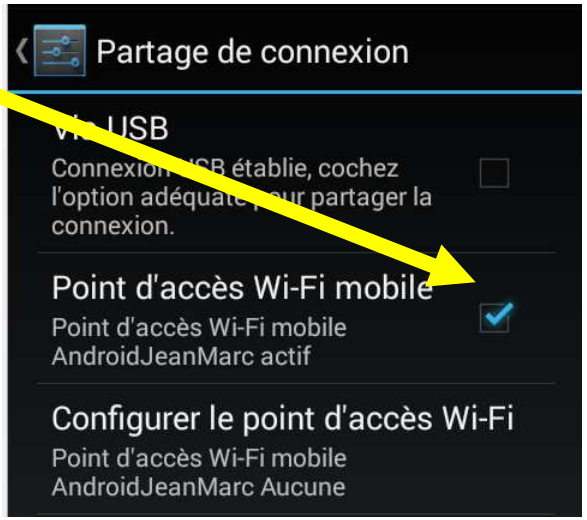
Repérer les appareils connectés sur notre réseau Wi-Fi

- On veut :
- 1°) Ajouter à un smartphone, la fonctionnalité d'être un hotspot (= une borne Wi-Fi = un émetteur Wi-Fi)
- 2°) Un second smartphone et récepteur Wi-Fi
- 3°) Une application lancée sur le "smartphone-hotspot" repère ce récepteur Wi-Fi

- source : `http://www.whitebyte.info/android/android-wifi-hotspot-manager-class` : WhiteByte (Nick Russler, Ahmet Yueksektepe)) et Fabrice Mourlin

Le smartphone devient hotspot

- Pour ajouter à un téléphone (pas possible pour une tablette ?) Nexus S , la fonctionnalité d'être un hotspot (= une borne Wi-Fi)
- Paramètres | Plus... | Partage de connexion | Point d'accès Wi-Fi mobile : cocher la case



- Le téléphone devient alors hotspot pour le réseau par défaut
- Si on veut choisir un autre réseau par défaut faire
- Paramètres | Plus... | Partage de connexion | Configurer le point d'accès Wi-Fi
- Le réseau choisi est ici, `AndroidJeanMarc`

Un second smartphone se connecte sur cet hotspot

- Pour qu'un smartphone se connecte sur une autre smartphone devenu hotspot, par exemple la tablette Galaxy Tab se connecte sur le hotspot (= téléphone)
- Sur la tablette, Paramètres | Wi-Fi
- La liste des réseaux Wi-Fi apparaît
- Choisir le réseau dont le téléphone est hotspot (`AndroidJeanMarc`)

Application Android affichant les smartphones qui sont connectés sur l'hotspot

- Démo dans `2AfficheSmartphonesConnectesSurNotreHotspot`
- A lancer sur le hotspot (le téléphone)
- Amène :

Affichage des smartphones connectés sur l'hotspot : le code

- L'appli lit le fichier `/proc/net/arp` (et c'est tout) ligne par ligne. Sur une ligne, on trouve, entre autre l'adresse IP, le HW adress (= le hardware adresse = l'adresse MAC), et le nom du réseau des machines connectées. Ces informations sont affichées sur le smartphone
- En gros on a un code comme :

```
public ArrayList<ClientScanResult> getClientList(...) {
     result = new ArrayList<ClientScanResult>();
     br = new BufferedReader(new FileReader("/proc/net/arp"));
     String line;
     while ((line = br.readLine()) != null) {
       String[] splitted = line.split(" +");
       // split() utilise les expressions régulières.
       // Ici découpe la ligne suivant des tokens séparés par une suite de caractères espace
          if ((splitted != null) && (splitted.length >= 4)) {
            String mac = splitted[3];
            ...
            result.add(new ClientScanResult(splitted[0], splitted[3], splitted[5], ...);
     }
     return result;
}
```

Menus de l'application

- Les menus de l'appli sont intéressants. Ils permettent d'arrêter la propriété de hotspot du smartphone et de la relancer
- Pour rechercher les smartphones connectés, il suffit de lancer la méthode `setWifiApEnabled()` du `WifiManager` : voir la méthode `setWifiApEnabled()` de la classe `WifiApManager` qui a été écrite
- Euh cette méthode n'est pas publique, il faut donc faire de l'instrospection :

```
Method method = mWifiManager.getClass().getMethod("setWifiApEnabled",
     WifiConfiguration.class, boolean.class);
return (Boolean) method.invoke(mWifiManager, wifiConfig, enabled);
```

- Pourquoi cela ? Je ne sais pas !

L'introspection : quezako ?

- Java est un langage de programmation qui modélise ses propres notions (de classe, de méthodes, de constructeurs) ...
- ... comme objet de classe : c'est vachement bien
- Par exemple une classe peut être modélise comme un objet de la classe `java.lang.Class` (si, si) (et cela depuis la version 1.0, 1995)
- De tels langages sont dits des langages réflexifs
- L'introspection est la possibilité qu'on certain langage de programmation de pouvoir connaître, à l'exécution, les caractéristiques internes d'un objet : ces champs mais aussi sa classe, les constructeurs et méthodes de sa classe ... et de pouvoir lancer ces méthodes !
- Euh l'introspection rompt l'encapsulation

L'introspection : code

- En Java on écrit évidemment :

```
// Sans utiliser la réflexion
MaClasse ref = new MaClasse();
ref.maMethode();
```

- Avec l'introspection on écrit :

```
// En utilisant la réflexion
Class cl = Class.forName("MaClasse");
// Instanciation de l'objet dont la méthode est à appeler
Object instance = cl.newInstance();
// Invocation de la méthode via réflexion
Method method = cl.getMethod("maMethode", null);
method.invoke(instance, null);
```

- La méthode `invoke()` permet aussi de lancer des méthodes avec des arguments. D'ailleurs sa signature est :
  `public Object invoke(Object obj, Object... args)`
- source :
  `http://fr.wikipedia.org/wiki/R%C3%A9flexion_(informatique)`
  et
  `http://docs.oracle.com/javase/tutorial/reflect/TOC.html`

L'introspection : conclusions

- Donc `ref.maMethode();`
  est équivalent à `method.invoke(instance, null);`
  où `method` modélise la méthode `maMethode()` (comme objet de la classe `Method`) et instance repère un objet de la classe `MaClasse`
- Et alors ?
- Ben on peut, à l'exécution, récupérer un objet inconnu (comme un objet qui modélise un gestionnaire de connexion Wi-Fi ;-)) et lui demander de lancer certaines de ces méthodes
- D'ailleurs l'objet repéré par une référence de la classe `android.net.wifi.WifiManager` n'est pas forcément un objet de cette classe, n'est ce pas ! ;-)
- source :
  `http://docs.oracle.com/javase/tutorial/reflect/TOC.html`

Démo 3

Par programmation, le smartphone devient un hotspot

Un smartphone devient hotspot par programmation : démo

- 1°) La tablette est initialement sur le réseau `AndroidJeanMarc` (cf. démo précédente). C'est visible dans Paramètres | Wi-Fi

- 2°) Lancer l'appli `3WiFiHotspotCreator` sur la tablette
- 3°) Elle se déconnecte alors de ce réseau "externe" (dont le hotspot n'était pas la tablette mais le téléphone) pour devenir hotspot du réseau `MyWifiAp`
- Cliquer icône Wi-Fi | Modem ou point d'accès activé
- On obtient :

Transformer un smartphone en hotspot par programmation (1/3)

- Il faut préparer une configuration Wi-Fi à l'aide de la classe `android.net.wifi.WifiConfiguration` par :

```
import android.net.wifi.WifiConfiguration;
...
WifiConfiguration netConfig = new WifiConfiguration();

netConfig.SSID = "MyWifiAP";  // Nom de ma connexion WIFI

// Positionne les algorithmes d'authentification à ceux de la norme 802.11, bref du Wi-Fi
netConfig.allowedAuthAlgorithms.set(WifiConfiguration.AuthAlgorithm.OPEN);

// autorise le protocole de sécurisation WPA2/IEEE 802.11i (= RSN)
netConfig.allowedProtocols.set(WifiConfiguration.Protocol.RSN);

// autorise le protocole WEP
netConfig.allowedKeyManagement.set(WifiConfiguration.KeyMgmt.NONE);

// autorise le protocole de sécurisation WPA2/IEEE WPA/IEEE 802.11i/D3.0 (= WPA)
netConfig.allowedProtocols.set(WifiConfiguration.Protocol.WPA);
```

- Le réseau Wi-Fi (son SSID) est `MyWifiAp`

Transformer un smartphone en hotspot par programmation (2/3)

- Il faut, là encore, utiliser l'introspection :

```
WifiManager wifiManager = (WifiManager) this.getSystemService(Context.WIFI_SERVICE);

if (wifiManager.isWifiEnabled()) {
     wifiManager.setWifiEnabled(false);
}
Method[] wmMethods = wifiManager.getClass().getDeclaredMethods();
...
for (Method method : wmMethods) {
     if (method.getName().equals("setWifiApEnabled")) {
     ...
     method.invoke(wifiManager, netConfig, true);
```

- Puis "insister" pour lancer la connexion Wi-Fi

```
while (!(Boolean) isWifiApEnabledmethod.invoke(wifiManager)) {
     ;
}
```

Transformer un smartphone en hotspot par programmation (3/3)

- On peut lancer d'autres méthodes par introspection :

```
for (Method method1 : wmMethods) {
     if (method1.getName().equals("getWifiApState")) {
          ...
          method1.invoke(wifiManager);
          for (Method method2 : wmMethods) {
               if (method2.getName().equals("getWifiApConfiguration")) {
                    ...
                    netConfig = (WifiConfiguration) method2.invoke(wifiManager);
                    ...
```

Démo 4

Faire communiquer deux smartphones par Wi-Fi direct (tethering)

Faire communiquer deux smartphones : démo (1/2)

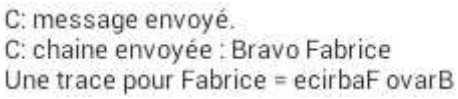
- Au début de la démo, la tablette est hotspot du réseau `MyWifiAP` (tablette = serveur Wi-Fi ?), le téléphone est connecté à ce réseau (téléphone = client Wi-Fi ?), le faire si ce n'est pas le cas (Paramètres | Wi-Fi | Partage de connexion | Configurer le point d'accès Wi-Fi, changer le SSID pour mettre `MyWifiAP`)
- Le service construit est un service de renversement de chaînes de caractères : on envoie une chaîne de caractères au service distant, il la retourne écrite à l'envers : c'est renversant ;-)
- On lance l'appli `4.1JeanMarcWiFiServer` sur le téléphone (téléphone = serveur applicatif ?)

Faire communiquer deux smartphones : démo (2/2)

- On lance l'appli `4.2JeanMarcWiFiClient` sur la tablette (tablette = client applicatif ?)
- Ajuster l'adresse IP du serveur
- Cliquer Send (par défaut le serveur écoute derrière le port 9000 et le client envoie sur ce port)

- Le serveur (applicatif) (= le téléphone) affiche :

- Le client (applicatif) (= la tablette) affiche :



Faire communiquer deux smartphones en Wi-Fi direct

- = Tethering (attacher à)
- Un des smar(télé)phones devient borne Wi-Fi (hotspot), les autres peuvent se connecter sur ce réseau. L'ensemble forme un réseau de machines connectées
- Voir à `http://en.wikipedia.org/wiki/Tethering`

Programmation réseau en général

- Un serveur est un programme qui rend un service (si, si). En général, un serveur est à l'écoute de requête qui lui sont adressées. Mais ce n'est pas systématique (si, si, exemple un serveur de temps). En général, un serveur est distant mais ce n'est pas systématique (si, si, exemple serveur X)
- Un client est un programme qui demande un service (si, si). En général, il envoie une requête, attend une réponse et est local. Mais ce n'est pas systématique (cf. exemples ci dessus)
- Qui dit programmation réseau, dit, si possible, programmation multithreadée, coté serveur comme coté client
- Pourquoi programmation multithreadée coté serveur ?
- Pourquoi programmation multithreadée coté client ?

Programmer les threads en Java

- Une technique est d'utiliser la classe `Thread`
- Un constructeur de la classe Thread est :
  `public Thread(Runnable target)`
- `Runnable` est une interface. Une classe implémentant un Runnable doit donner un corps à la méthode `public void run()`
- C'est ce code qui sera lancé lorsqu'on lancera la méthode ... `start()` sur la thread
- Bref on écrit (souvent) :

```
Runnable unRunnable = new Runnable() {
  public void run() {
    // code qui sera exécuté dans la thread
  }
};
Thread uneThread = new Thread(unRunnable);
uneThread.start();
```

- On écrit du code dans `run()`, on le lance par `start()`, étrange non ? Pas vraiment (programmation asynchrone)

La "UI Thread" : rappel (1/2)

- Lorsqu'une application Android est lancée, un seul processus est créé qui contient une seule thread pour l'application
- Cette thread est dite la thread principale
- Elle s'occupe, entre autre, de l'affichage et de l'interaction sur les divers écrans
- Voilà pourquoi cette thread principale est appelée la UI Thread (User Interface Thread) : "As such, the main thread is also sometimes called the UI thread."
- source : `http://developer.android.com/guide/components/processes-and-threads.html`

La "UI Thread" : rappel (2/2)

- Donc, dans une application Android, il existe une et une seule thread qui gère l'interface graphique : la UI Thread (User Interface Thread)
- Tout ce qui concerne l'affichage est (et doit être) géré par cette Thread. Si une autre Thread s'occupe de faire de l'affichage graphique, il y a erreur à l'exécution
- Lorsqu'un travail demandant du temps est lancé, il faut le faire dans une Thread autre que la UI Thread. Au besoin en créé une !
- Mais lorsque autre thread demande à afficher dans l'IHM, cette autre thread doit contacter l'UI Thread !

Code du serveur (1/2)

- En plus de la UI thread, on a au moins 2 autres threads :
  - une thread "réseau" qui écoute en qui est en attente d'une connexion par un client
  - chaque fois qu'un client se connecte, une thread de traitement pour ce client est lancée. Immédiatement l'application revient en écoute
- Par contre les affichages de trace doivent se faire dans la UI thread

Code du serveur (2/2)

```
protected void onCreate(Bundle savedInstanceState) {
  ...
  Thread fst = new Thread(new ServerThread());
  fst.start();
}

public class ServerThread implements Runnable {
  public void run() {
    ...
    mSS_serverSocket = new ServerSocket(NUM_PORT_ECOUTE_SERVEUR);
    ...
    while (true) {
      // Ecoute des clients et lancement d'une thread de traitement pour
      // chaque client, puis retour en écoute
      Socket socketClient = mSS_serverSocket.accept();
      Thread traiteClient = new Thread(new TraitementDUnClient(socketClient));
      traiteClient.start();
    }
  }
  ...
}
public class TraitementDUnClient implements Runnable {
  private Socket socketClient;
  public TraitementDUnClient(Socket socketClient) {
    this.socketClient = socketClient;
  }
  public void run() {
    // travail à faire pour un client
    ...
  }
}
```

android.os.Handler

- Dès qu'on lance la thread d'écoute réseau, on est ... dans la thread d'écoute réseau
- Quand on lance la thread de traitement d'un client, on est ... dans la thread de traitement d'un client
- Bref on n'est plus dans la seule thread permettant de faire des affichages d'interfaces graphiques : la UI thread
- Il **faut** pourtant faire ces affichages dans la UI thread
- Une première technique déjà vue sont les `AsyncTask`. Une autre sont les `Handler`

Mise à jour de l'IHM du serveur : `Handler` (1/2)

- La classe `Handler`, bien pratique, permet d'indiquer de lancer du code (= envoyer un `Runnable`) ou d'envoyer des données (des `Message`) à la thread qui lui a donné naissance dite thread associée au `Handler`
- Ainsi un objet de la classe `Handler` envoie du code à exécuter à la thread qui lui a donné naissance par les méthodes `postXXX(Runnable r, ...)`
- La thread associée au `Handler` exécutera le code du `Runnable` passé comme argument
- source : `http://developer.android.com/reference/android/os/Handler.html`

Mise à jour de l'IHM du serveur : Handler (2/2)

- D'où l'architecture :
  - dans la déclaration de la classe `Activity`, on écrit

```
private Handler mH_handler = new Handler();
```

  - après le travail du serveur :

```
mH_handler.post(new Runnable() {
     @Override
     public void run() {
          // mise à jour du TextView tv
          tv.append(chaine);
     }
});
```

- Finalement le mieux est de s'écrire une méthode

```
private void ajouteDansTextView(final TextView tv, final String chaine){
  mH_handler.post(new Runnable() {
    @Override
    public void run() {
      tv.append(chaine);
    }
  });
}
```

et de l'appeler

Code du client (1/2)

- Lorsqu'on clique sur le bouton du l'IHM du client, la chaîne de caractères à traiter (= à renverser) doit être envoyée au serveur
- => dans le code de la méthode `onClick()`, on écrit du code réseau d'envoi = une thread à créer et à lancer :

```
public void onClick(View v) {
  ...
  Thread cThread = new Thread(new ClientRunnable(mS_serverIpAddress, portEcouteDuServeur));
  cThread.start();
}
```

- Les mises à jour graphique seront faites par un `Handler` lié à la UI thread

Code du client (2/2)

- La thread ouvre une socket vers le serveur, écrit dedans, puis écoute cette même socket pour recevoir le résultat traité par le serveur :

```
public class ClientRunnable implements Runnable {
  private String mS_adresseIPDuServeur;
  private int mi_numeroDePortEcouteDuServeur;
  public ClientRunnable(String serverIpAddress, int portEcouteDuServeur){
    mS_adresseIPDuServeur = serverIpAddress;
    mi_numeroDePortEcouteDuServeur = portEcouteDuServeur;
  }
  public void run() {
    ...
    InetAddress serverAddr = InetAddress.getByName(mS_adresseIPDuServeur);
    ...
    Socket socket = new Socket(serverAddr, mi_numeroDePortEcouteDuServeur);
    ...
    final String chaineAEnvoyer = "Bravo Fabrice";

    PrintWriter out = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(socket.getOutputStream())), true);
    BufferedReader br = new BufferedReader(new InputStreamReader(socket.getInputStream()));

    out.println(chaineAEnvoyer);
    out.flush();

    ...
    s_chaineRecuperee = br.readLine();
  }
}
```

Bibliographie pour ce chapitre (1/2)

- Pour la démo 1 (découverte des réseaux Wi-Fi) : `http://www.androidsnippets.com/scan-for-wireless-networks`
- Démo 2 : repérer les appareils connectés sur le téléphone hotspot `http://www.whitebyte.info/android/android-wifi-hotspot-manager-class` : WhiteByte (Nick Russler, Ahmet Yueksektepe)) et Fabrice Mourlin
- Démo 3 : Par programmation, le smartphone devient un hotspot : Fabrice Mourlin
- Démo 4 : Fabrice et moi (si !)

Bibliographie pour ce chapitre (2/2)

- Sur le site `developer.android.com` on les tutoriaux commençant à : `http://developer.android.com/training/connect-devices-wirelessly/index.html`
- Par la suite il y a 4 cours que j'ai appelé
  - chapitre 0 (Wi-Fi Peer-to-Peer) à `http://developer.android.com/guide/topics/connectivity/wifip2p.html`
  - chapitre 1 (Using Network Service Discovery) à `http://developer.android.com/training/connect-devices-wirelessly/nsd.html`
  - chapitre 2 (Creating P2P Connections with Wi-Fi) à `http://developer.android.com/training/connect-devices-wirelessly/wifi-direct.html`
  - chapitre 3 (Using Wi-Fi P2P for Service Discovery) à `http://developer.android.com/training/connect-devices-wirelessly/nsd-wifi-direct.html`
- Euh, je ne suis pas arrivé à faire fonctionner correctement tous ces tutoriaux

Fin