ANDROID ET JEE

Par

FELOUACH Khalid

RABEH Tarik

LABRIJI Amine

Rapport entrant dans le cadre d'un exposé

2012

Encadre par :

BEN LAHMAR El habib

Date:
_____18/06/2012_____

TABLE DES MATIERES

REMERCIEMENTS

Les auteurs tiennent à exprimer sa sincère reconnaissance à Monsieur le Professeur Ben Lahmar El habib pour leur aide lors de la préparation de ce manuscrit. Tant sur le plan des besoins que des idées exposées, a été d'une grande aide pour la mise en route de ce expose. Enfin, merci également aux membres du comité étudiant pour leur soutien.

GLOSSAIRE

Android : est un système d'exploitation open source3 utilisant le noyau Linux, pour smartphones, PDA et terminaux mobiles conçu par Android.

SQLite :est une bibliothèque écrite en C qui propose un moteur de base de données relationnelles accessible par le langage SQL. SQLite implémente en grande partie le standard SQL-92 et des propriétés ACID.

Kernel : est aussi appelé noyau, il s'agit de la partie fondamentale d'un système d'exploitation. Le kernel permet de simplifier et sécuriser l'utilisation des différents composants et périphériques de l'ordinateur. Il détermine également quel programme doit s'exécuter et pendant combien de temps grâce à une méthode appelée l'ordonnancement.

licence *BSD* ***:*** est une licence libre utilisée pour la distribution de logiciels. Elle permet de réutiliser tout ou une partie du logiciel sans restriction, qu'il soit intégré dans un logiciel libre ou propriétaire.

Activity : le concept d'activity repose sur la notion d'interaction utilisateur.
Une activity la fenêtre ou tout simplement l'écran qui sera affiché a l'utilisateur Elle permet également de gérer des fonctionnalités telles que l'appui sur la touche ou l'affichage de message d'alerte.

Introduction

Android est système d'exploitation base sur le noyau Linux qui fournit l'interface bas niveau avec le HW, la gestion de la mémoire, le contrôle des processus, le tout optimise pour les terminaux mobiles

• Un ensemble de bibliothèques Open Source pour le développement d'applications incluant SQLite, WebKit, OpenGL, et la gestion des media.

• Android, incluant la VM Dalvik et les librairies principales qui fournissent la fonctionnalité Android. Cet exécutable est conçu pour être efficace sur les terminaux mobiles et de petit taille.

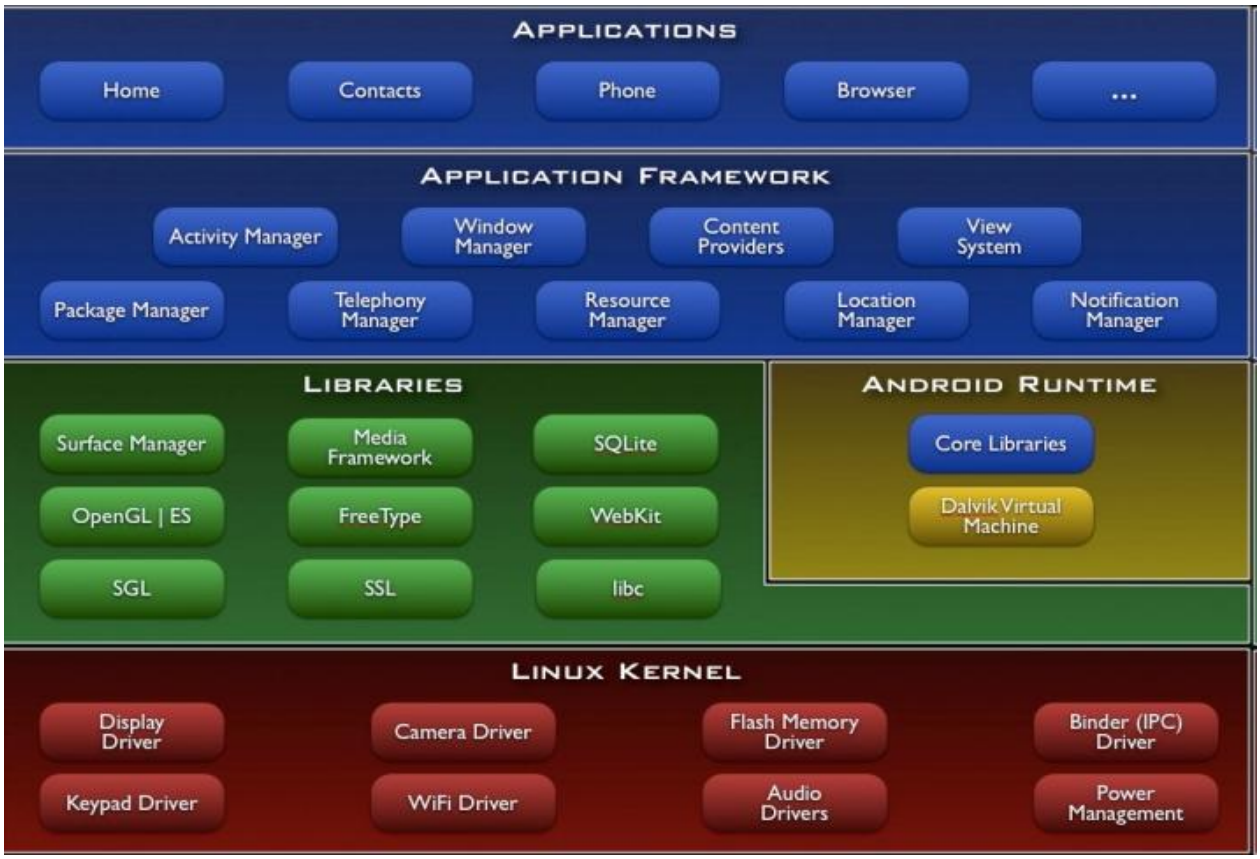
Un Framework applicatif exposant les services systèmes a la couche application y compris "window manager, location manager, content providers, telephony, sensors"

• Un Framework d'interface utilisateur pour installer et lancer les applications.

• Un software developpement kit utilise pour créer des applications, incluant des outils, plugins et documentation.

Chapitre 1

L'ARCHITECTURE ANDROID



Android est basé sur un kernel linux 2.6.xx, au-dessus du kernel il y a "le hardware abstraction layer" qui permet de séparer la plateforme logique du matériel.
Au-dessus de cette couche d'abstraction on retrouve les librairies C/C++ utilisées par un certain nombre de composants du système Android.
Au-dessus des librairies on retrouve l'Android Runtime, cette couche contient les librairies cœurs du Framework ainsi que la machine virtuelle exécutant les applications.
Au-dessus la couche "Android Runtime" et des librairies cœurs on retrouve le Framework permettant au développeur de créer des applications. Enfin au dessus du Framework il y a les applications.

Android est basé sur un kernel linux 2.6 mais ce n'est pas linux. Il ne possède pas de système de fenêtrage natif (X window system), la glibc n'est pas supporté, Android utilise une libc customisé appelé Bionic libc.

Enfin Android utilise un kernel avec différents patches pour la gestion de l'alimentation, le partage mémoire, etc. permettant une meilleurs gestion de ces caractéristiques pour les appareils mobiles.

Ces librairies sont écrites en C/C++. Elles fournissent les fonctionnalités de bas niveau d'Android par exemple :

Libc

Les ingénieurs d'Android ont développé une librairie C (libc) nommé Bionic libc.
Elle est optimisée pour les appareils mobiles et a été développé spécialement pour Android.
Cette libc est sous licence BSD, elle reprend une grande partie du code des glibc issue d'OpenBSD, FreeBSD et NetBSD.

WebKit

Le navigateur web présent dans Android est basé sur le moteur de rendu sous licence BSD WebKit.

WebKit est moteur de rendu, qui fournit une "fondation" sur lequel on peut développer un navigateur web. Il a été originellement dérivé par Apple du moteur de rendu KHTML pour être utilisé par la navigateur web Safari et maintenant il est développé par KDE project, Apple, Nokia, Google et d'autres. WebKit est composé de deux librairies : WebCore et JavascriptCore qui sont disponible sous licence GPL.

Core Libraires :

Les libraries Core fournissent le langage Java disponible pour les applications.

Dalvik :

La machine virtuelle Dalvik est basée sur une architecture de registre à l'instar de beaucoup de machine virtuel et de la machine virtuel Java qui ont une architecture de pile. Utilisé une architecture de pile ou de registre dépends des stratégies de compilation et d'interprétation choisit. Généralement, les machines basées sur une architecture de pile.

Le Framework est situé au-dessus de l'Android Runtime et des librairies.
Il fournit des API permettant aux développeurs de créer des applications riches.
Android introduit la notion de services.
Un service est une application qui n'a aucune interaction avec l'utilisateur et qui tourne en arrière-plan pendant un temps indéfini.
Par exemple :
View System : fournit tous les composants graphiques : listes, grille, Text box, buttons et même un navigateur web embarqué.

Chapitre 2

COMMUNIQUEZ APPLICATION ANDROID AVEC UNE APPLICATION SERVEUR JEE VIA INTERNET

Aujourd'hui la plupart des terminaux Android, si ce n'est pas tous, intègrent un accès à Internet. Cet accès peut passer par le Wifi, les services de données cellulaires (EDGE, 3G, etc.) ou, éventuellement, un mécanisme totalement différent.
Il n'est donc pas étonnant qu'Android offre aux développeurs un large éventail de moyens leur permettant d'exploiter cet accès. Ce dernier peut être de haut niveau, comme le navigateur WebKit intégré. Mais il peut également intervenir au niveau le plus bas et utiliser des sockets bruts. Entre ces deux extrémités, il existe des API – disponibles sur le terminal ou via des JAR tiers – donnant accès à des protocoles spécifiques comme HTTP, XMPP, SMTP, etc.

1. Utilisation du protocole http

Envoyer la requête

Pour utiliser le protocole http on utilise le composant HttpClient qui permet de gérer les requêtes HTTP.
La première étape pour l'utiliser consiste évidemment à créer un objet. HttpClient étant une interface, on doit donc instancier une implémentation de celle-ci, comme DefaultHttpClient.
HttpClient client = new DefaultHttpClient();
Ces requêtes sont enveloppées dans des instances de HttpRequest, chaque commande
HTTP étant géré par une implémentation différente de cette interface (HttpGet pour les requêtes GET, par exemple). On crée donc une instance d'une implémentation de HttpRequest, on construit l'URL à récupérer ainsi que les autres données de configuration (les valeurs des formulaires si l'on effectue une commande POST via HttpPost, par exemple) puis l'on passe la méthode au client pour qu'il effectue la requête HTTP en appelant execute().

```
HttpGet request = new HttpGet(url);
```

```
HttpPost request = new HttpPost("http://10.0.2.2/android/andro.jsp");
```

Ce qui se passe ensuite peut être très simple ou très compliqué. On peut obtenir un objet `HttpResponse` enveloppant un code de réponse (200 pour OK, par exemple), des en-têtes HTTP, etc. Mais on peut également utiliser une variante d'execute() qui prend en paramètre un objet `ResponseHandler<String>` : cet appel renverra simplement une représentation String de la réponse. En pratique, cette approche est déconseillée car il est préférable de vérifier les codes de réponses HTTP pour détecter les erreurs.

```
HttpResponse response = client.execute(request);
```

Exemple :

Envoie d'un login et un password par GET :

```
String login= "admin";
String pass= "passadmin";
HttpClient client = new DefaultHttpClient();
HttpGet request = new
HttpGet("http://10.0.2.2/TEST/test.jsp?login="+login+"
&password="+pass);
// On récupère le résultat
HttpResponse response = client.execute(request);
```

Envoie d'un login et un password par POST :

```
String login= "admin";
String pass= "passadmin";
HttpClient client = new DefaultHttpClient();
HttpPost request = new
HttpPost("http://10.0.2.2/TEST/test.jsp");
List<NameValuePair> nvps = new
ArrayList<NameValuePair>();

nvps.add(new BasicNameValuePair("login", login));

nvps.add(new BasicNameValuePair("password", pass));
post.setHeader("Content-Type", "application/x-www-form-
urlencoded");
/* On passe les paramètres login et password qui vont être
récupérés par test.jsp en post*/
request.setEntity(new UrlEncodedFormEntity(nvps,
HTTP.UTF_8));
// On récupère le résultat
HttpResponse response = client.execute(request);
```

Récupérer la réponse de la requête

Pour récupérer la réponse de la requête il suffit de créer un flux `InputStreamReader` à partir de `HttpResponse`. Puis en passant ce flux a un `BufferedReader` pour lire chaque ligne de la réponse par la méthode `readLine`.

```
client.execute(request);
BufferedReader rd = new BufferedReader(new
InputStreamReader(response.getEntity().getContent()));

String line = "";
if ((line = rd.readLine()) != null)
{
Log.d("Toast", line);
}
```

Les requete http et AsyncTask :

La documentation android conseille d'utiliser la classe `AsyncTask` pour toutes les taches longues, afin de ne pas bloquer le thread principal (utilisé pour mettre à jour l'interface graphique). Une requête HTTP doit toujours être considérée comme une tache longue car une requête rapide à un instant donné peut prendre plus de temps en production (parce que le réseau ou le serveur peuvent avoir des charges plus importantes à certains moments). Android 3 (API 11) force cette bonne pratique en levant l'exception `NetworkOnMainThreadException` si on utilise le thread principal pour exécuter une requête HTTP.

Pour utiliser AsyncTask, il faut :

- créer une sous-classe d'AsyncTask, généralement sous la forme d'une classe interne à celle qui utilise la tâche (une activité, par exemple) ;
- redéfinir une ou plusieurs méthodes d'AsyncTask pour réaliser le travail en arrièreplan ainsi que toute opération associée à la tâche et qui doit s'effectuer dans le thread de l'interface (la mise à jour de la progression, par exemple) ;
- le moment venu, créer une instance de la sous-classe d'AsyncTask et appeler execute() pour qu'elle commence son travail.

On n' pas besoin :

- De créer votre propre thread en arrière-plan ;
- De terminer ce thread au moment voulu ;
- D'appeler des méthodes pour que des traitements s'effectuent dans le thread de l'interface.

AsyncTask, généricité et paramètres variables :

La création d'une sous-classe d'AsyncTask n'est pas aussi simple que, par exemple, l'implémentation de l'interface Runnable car AsyncTask est une classe générique ;
Il faut donc lui indiquer trois types de données :

- Le type de l'information qui est nécessaire pour le traitement de la tâche (les URL à télécharger, par exemple).

- Le type de l'information qui est passée à la tâche pour indiquer sa progression.

- Le type de l'information qui est passée au code après la tâche lorsque celle-ci s'est terminée.

En outre, les deux premiers types sont, en réalité, utilisés avec des paramètres variables, ce qui signifie que notre sous-classe d'AsyncTask les utilise via des tableaux.
Pour parvenir à nos fins, on peut redéfinir quatre méthodes d'AsyncTask.
Mais la seule qu'on doit redéfinir pour que notre classe soit utilisable s'appelle doInBackground().
Elle sera appelée par AsyncTask dans un thread en arrière-plan et peut s'exécuter aussi longtemps qu'il le faut pour accomplir l'opération nécessaire à cette tâche spécifique. Cependant, il est déconseillé d'utiliser AsyncTask pour réaliser une boucle infinie.
La méthode doInBackground() recevra en paramètre un tableau variable contenant des éléments du premier des trois types mentionnés ci-dessus – les type des données nécessaires au traitement de la tâche. Si la mission de cette tâche consiste, par exemple, à télécharger un ensemble d'URL, doInBackground() recevra donc un tableau contenant toutes ces URL.
Cette méthode doit renvoyer une valeur du troisième type de données mentionné – le résultat de l'opération en arrière-plan.

Remarque :

1: Le fichier AndroidManifest.xml

Pour que l'application puisse fonctionner correctement, elle devra avoir accès à Internet. Il faut donc donner l'autorisation d'accès à Internet à l'application. Pour cela, il faut ouvrir le fichier AndroidManifest.xml et rajouter la ligne suivante :

```
<uses-permission
android:name="android.permission.INTERNET"/>
```

2: localhost :

L'utilisation de « localhost » comme adresse dans l'application android revient en fait à chercher un serveur local à l'émulateur (qui est normalement absent).
Heureusement il existe une adresse IP "magique" qui permet de contourner ce problème de double localhost : la 10.0.2.2. Lorsque vous accéderez à 10.0.2.2 depuis l'émulateur, vous accéderez en fait à votre adresse de boucle locale (la localhost de votre machine de développement).

```
HttpPost request = new HttpPost("http://10.0.2.2/android/andro.jsp");
```

Partie serveur :

Dans le partie serveur que ce soit JEE, PHP ou autre on traite la requête qui vient de client comme on fait avec une requête qui vient d'un navigateur web dans les 2 cas POST et GET.

Exemple :

On va fait un exemple simple d'application qui permet de calculer la somme de deux nombre

Principe de fonctionnement :

1. L'utilisateur entre deux nombre par 2 EditText
2. puis il clique sur un boutton
3. l'envoi des 2 nombres a un servlet par méthode GET
4. le servlet reçoit les 2 nombres et calcule son somme.
5. Le servlet envoi le la somme par out.print
6. Le client reçoit la réponse et il affiche dans un TextView

3-Partie client(ANDROID)

main.xml (interface):

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:orientation="vertical" >

<TextView
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="APPLICATION D'ADDITION" />
<EditText
android:id="@+id/a"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:hint="0"
android:inputType="numberDecimal"
android:layout_alignParentRight="true"
/>
<EditText
android:id="@+id/b"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:hint="0"
android:inputType="numberDecimal"
android:layout_alignParentRight="true"
/>
<Button
android:id="@+id/btnReg"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="SOMME" />



<TextView
android:id="@+id/resultat"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="RESULTAT" />


</LinearLayout>
```

Main.java (Activite) :

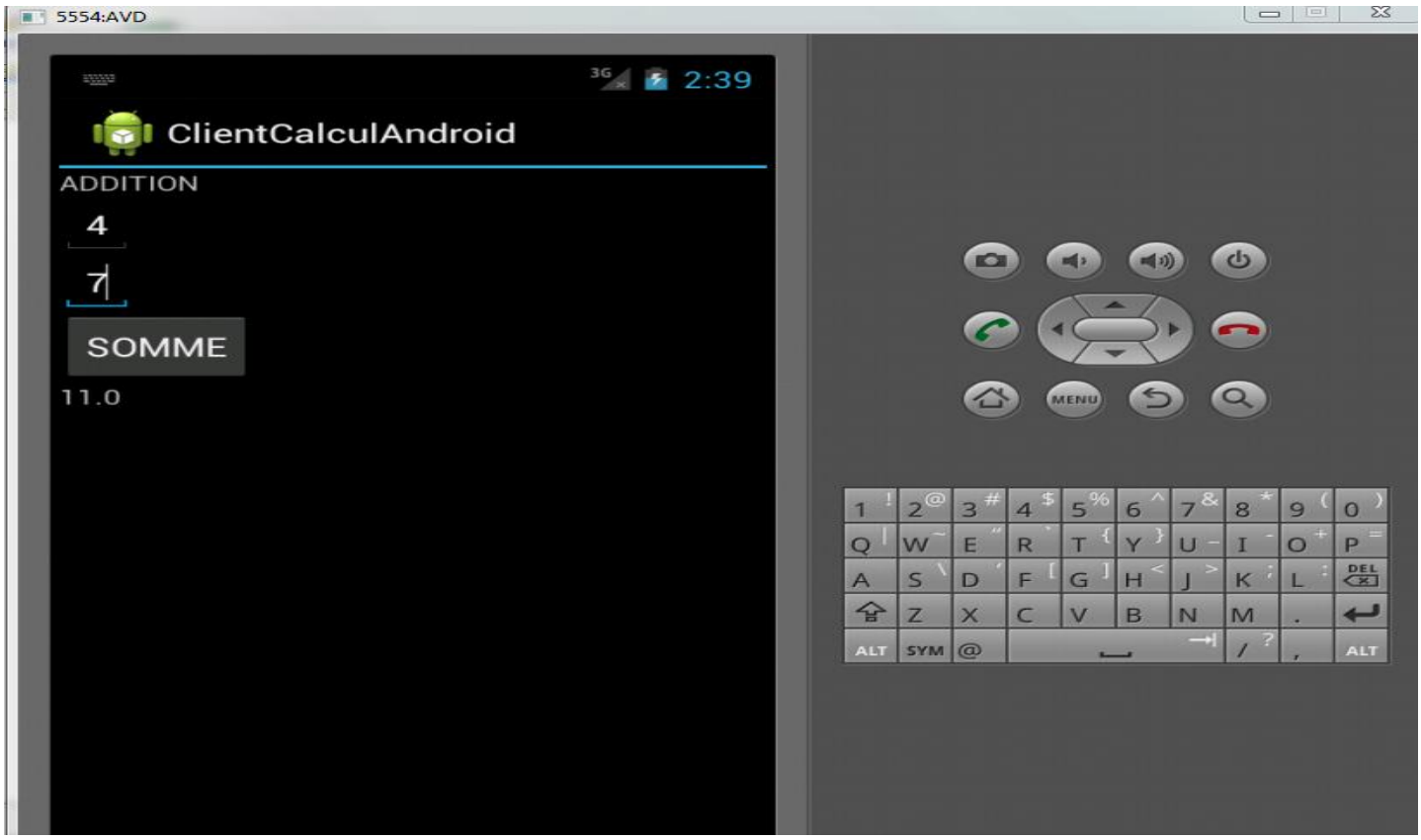
```
package khalid.calcul;


import java.io.BufferedReader;
import java.io.InputStreamReader;

import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;

import android.app.Activity;
import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;

public class Main extends Activity {
/** Called when the activity is first created. */
TextView a;
TextView b;
TextView resultat;
Button btnReg;
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
a=(TextView)findViewById(R.id.a);
b=(TextView)findViewById(R.id.b);
resultat =(TextView)findViewById(R.id.resultat);
btnReg=(Button)findViewById(R.id.btnReg);
btnReg.setOnClickListener(new OnClickListener() {

public void onClick(View v) {
// TODO Auto-generated method stub

Tache t=new Tache();
t.execute();
}
});
}
class Tache extends AsyncTask<Void, Void, Void> {
@Override
protected Void doInBackground(Void... params) {
try
{
String A=a.getText().toString();
String B=b.getText().toString();
```

```
String
                url="http://10.0.2.2:8082/ServeurCalculJEE/Calcul?
                a="+A+"&b="+B;
try
{
HttpClient client = new DefaultHttpClient();
HttpGet request = new HttpGet(url);
HttpResponse response = client.execute(request);
           BufferedReader    rd    =    new    BufferedReader(new
                             InputStreamReader(response.getEntity
                             ().getContent()));
String line = "";
if ((line = rd.readLine()) != null)
{
final String linee=line;
runOnUiThread(new Runnable() {
@Override
public void run() {
resultat.setText(linee);

}
});

}
}
catch(Exception c)
{
Log.e("Exception",c.getMessage());
}

}
catch(Exception c)
{
c.printStackTrace();
}
return(null);
}

}

}
```

Partie serveur(JEE)

Calcul.java (Servlet):

```
package Serveur.calcul;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
* Servlet implementation class Calcul
*/
public class Calcul extends HttpServlet {
private static final long serialVersionUID = 1L;

/**
* @see HttpServlet#HttpServlet()
*/
public Calcul() {
super();
// TODO Auto-generated constructor stub
}

/**
*      @see      HttpServlet#doGet(HttpServletRequest      request,
HttpServletResponse response)
*/
protected       void       doGet(HttpServletRequest       request,
HttpServletResponse     response)     throws     ServletException,
IOException {
// TODO Auto-generated method stub
double a= Double.parseDouble((String)request.getParameter("a"));
double b= Double.parseDouble((String)request.getParameter("b"));
double c=a+b;
PrintWriter out=response.getWriter();
out.print(c);
}

/**
*     @see     HttpServlet#doPost(HttpServletRequest     request,
HttpServletResponse response)
*/
protected       void       doPost(HttpServletRequest       request,
HttpServletResponse     response)     throws     ServletException,
IOException {
// TODO Auto-generated method stub
}
}
```


5554:AVD
2:39
ClientCalculAndroid
ADDITION
4
7
SOMME
11.0
MENU

R é s u m é

Android est un system d'exploitation pour le téléphone mobile,
Android il se compose en plusieurs couches (Application, application Framework, libraires, Kernel linux).
Les Applications Android peut se communique avec des applications PHP, ou bien JEE le sujet de notre expose ou avec n'importe quelle application web car Android offre a les développeurs des mécanismes qui permet l'accès au internet avec utilisation des protocoles http avec les méthodes POST ou GET

BIBLIOGRAPHIE

http://www-igm.univ-mlv.fr/~dr/XPOSE2008/android/index.html

http://fr.wikipedia.org/wiki/Licence_BSD

http://fr.wikipedia.org/wiki/Android

Site official :

http://www.android.com/

http://developer.android.com/index.html

Livre:
l'art du developpement android
Auteur : Mark L. Murphy

Traduit par Éric Jacoboni, avec la contribution d'Arnaud Farine

http://www.mti.epita.fr/blogs/2010/08/03/introduction-a-la-programmation-sous-android/