Cas EDF : Développement Android - Concepts avancés – Partie 1

Cette publication comporte cinq parties dont l'ordre est dicté par la logique du développement.
Les parties 2 et 3 sont facultatives.

- **Partie 1 : Gestion des clients**
  Partie 2 : Géolocalisation de l'agent et géocodage du client sélectionné
  Partie 3 : Signature Client
  Partie 4 : Communication avec le serveur
  Partie 5 : Identification, import et export des données.

Description du thème

Propriétés	Description
Intitulé long	**Cas EDF : Développement Androïd - Concepts avancés - Partie 1 : Gestion des clients**
Formation concernée	BTS Services Informatiques aux Organisations
Matière	SLAM 4
Présentation	Développement permettant d'aborder des concepts de la programmation Android d'une application embarquée, communiquant avec un serveur. Il aborde les notions : ➢ d'affichage de liste / d'adapter, ➢ de GEOLOCALISATION / GEOCODER, ➢ de graphisme (canvas) et d'encodage JPG, ➢ d'échange avec un serveur WEB (THREAD / JSON / GSON), ➢ d'utilisation d'un SGBDO DB4o.
Notions	Savoirs • D4.1 - Conception et réalisation d'une solution applicative • D4.2 - Maintenance d'une solution applicative Savoir-faire • Programmer un composant logiciel • Exploiter une bibliothèque de composants • Adapter un composant logiciel • Valider et documenter un composant logiciel • Programmer au sein d'un framework
Transversalité	SLAM5
Pré-requis	Développement d'une application Android sous un environnement Eclipse. (Exemple : Cas AMAP Jean-Philippe PUJOL)
Outils	Eclipse, DB4o, OME, Gson, Google play services, Apache, Mysql
Mots-clés	Application mobile, Android, SGBDO, DB4o, Géolocalisation, Géocodage, Thread, json, Gson, MVC, canvas, encodage JPG
Durée	24 heures (8,4,4,4,4) (Temps divisé par 2 si utilisation du squelette application)
Auteur	Pierre-François ROMEUF avec la relecture et les judicieux conseils de l'équipe CERTA
Version	v 1.0
Date de publication	Juin 2014

Contexte

Application embarquée sur une solution technique d'accès (STA) sous Android, permettant à un agent EDF d'effectuer sa tournée journalière de relevés des compteurs EDF.

Les principales fonctionnalités sont :

- Identification de l'agent sur le device avec contrôle sur un serveur web,
- Import des clients depuis un serveur web,
- Affichage des clients,
- Saisie des informations clients,
- Aide au déplacement via géolocalisation de la position de l'agent EDF et géocodage de l'adresse client,
- Enregistrement de la signature client validant les informations saisies,
- Export des données sur le serveur web.

Le SGBD embarqué est un SGBDO DB4o. Le serveur distant est un serveur de type LAMP installé sur la ferme de serveurs ou via un hébergement gratuit (ex : http://www.hostinger.fr/)

Cette application peut être dérivée pour de multiples besoins : ceux des livreurs, des commerciaux, des visiteurs, des contrôleurs ...

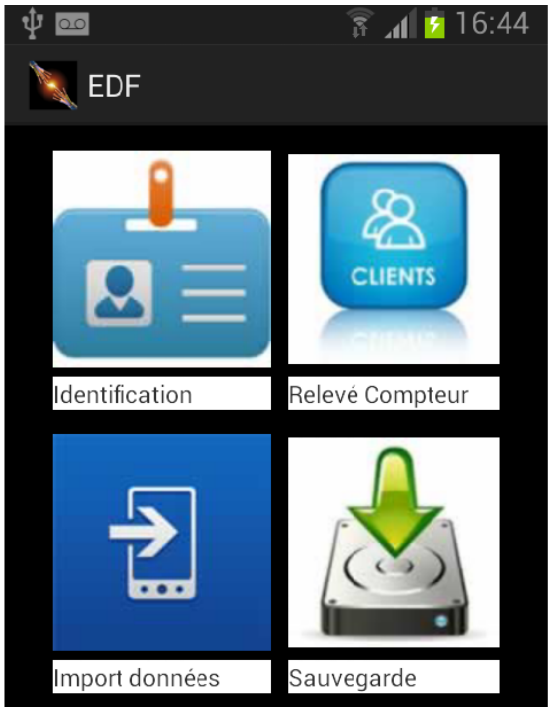
Le code fourni en annexe nécessite de votre part une compréhension.
Il représente normalement votre travail de programmeur, de fouille sur internet, avec tests, compréhension et modifications du code.

Il vous est fourni afin que ce développement ne représente qu'un travail raisonnable et pour vous présenter les différentes facettes du développement sur Android.

Conseils : consultez notamment developer.android.com**,** le tutoriel du zéro (http://uploads.siteduzero.com/pdf/554364-creez-des-applications-pour-android.pdf), etc.

Création du projet - MainActivity

- Créez votre projet ANDROID EDF-VOTRENOM.
- Importez les images du dossier IMG dans le dossier drawable (à créer).
- Dans le layout de votre MainActivity, créez un pseudo menu en image ressemblant à :



Solutions possibles :

<TableLayout> avec 4 <TableRow> ***Le + simple***	LinearLayout vertical LinearLayout horizontal LinearLayout vertical ImageView TextView /LinearLayout LinearLayout vertical ImageView TextView /LinearLayout /LinearLayout et bis repetita pour un 2° LinearLayout horizontal	Avec un menu déroulant de type "navigation drawer", via un glisser depuis le bord gauche de l'écran ou en appuyant sur l'icône de l'application dans la barre d'action et une image pour le background et un texte d'accueil. Plus complexe, implique de comprendre la gestion récente des menus sous Android.(navigation drawer, fragment)

La conception d'une vue à partir d'un RelativeLayout est complexe. Il est plus aisé d'utiliser des LinearLayout ou des TableLayout.

Pour modifier le layout de base d'une activity, par défaut un 'RelativeLayout', dans la fenêtre 'outline' clic droit sur le layout et choisir 'Change Layout'.

- Gérez le onClick sur les 4 images.

Exemple :

```
private ImageView imageViewAfficheListeClient;
...

imageViewAfficheListeClient = (ImageView) findViewById(R.id.????);
imageViewAfficheListeClient.setOnClickListener(imageClick);
private OnClickListener imageclick = new OnClickListener() {
        public void onClick(View v) {
        Intent i;
        switch (v.getId()) {
        case R.id.???:
              // appel de l'activity ??
                    break;
        case R.id.???:
              // appel de l'activity ??
                    break;
        }
        }};
```

Affichage de la liste des clients / Sélection d'un client / Affichage et saisie des informations

Classe Client

- Créez la classe Client

A partir du répertoire *src* , votre package de *base com.example.edf_votrenom*, clic droit et *new Class*.
Les attributs de la class Client sont présentés ci dessous.

```
/*
 * Données ne pouvant être modifiées cf service commercial/financier
 */
private String identifiant,nom,prenom,adresse,codePostal,ville,telephone;
// etc etc...
private String idCompteur;
private Double ancienReleve;
private Date dateAncienReleve;
/*
 * Données à saisir
 */
private Double dernierReleve;
private Date dateDernierReleve;
private String signatureBase64;
private int situation;
/* Exemples de situation client ------
 *  0 client non traité par défaut
 *  1 Absent 2 Absent mais relevé possible sans signature client
 *  3 Present, relevé ok mais pas de signature car pas représentant légal
 *  4 present et tout ok
 *  5 déménagé / logement vide 6 démanagé / nouveaux locataires
 *  72,73,74 idem 2,3,4 mais dysfonctionnement
 *  82,83,84 idem 2,3,4 mais dysfonctionnement / dégradation
 *  ... etc etc
 */
```

- Générez les getter et setter pour tous les attributs

Sélectionnez les attributs puis clic droit et choisir Source / Generate Getters and Setters.

- Créez un constructeur à vide.

- Surchargez le constructeur avec en paramètre les données ne pouvant être modifiées.

Les données modifiables seront initialisées (`dernierReleve` =0, `dateDernierReleve` =date du jour(new Date()),`signatureBase64`="", `situation` =0)

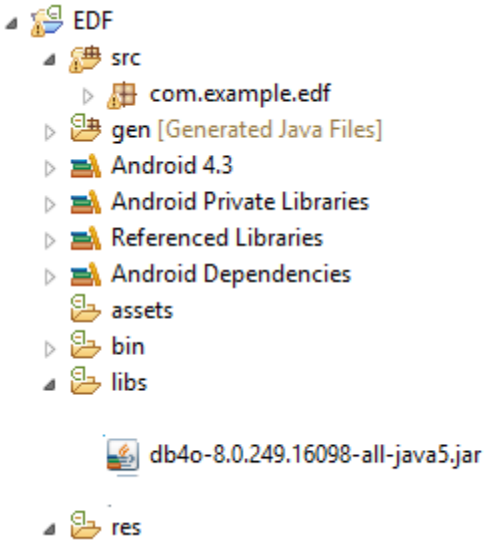
- Créez une méthode recopieClient(Client client)

Cette méthode permet de recopier toutes les données de l'objet client passé en paramètre vers this.
Cette méthode est utile pour l'update des informations de la classe Client à partir de DB4o.

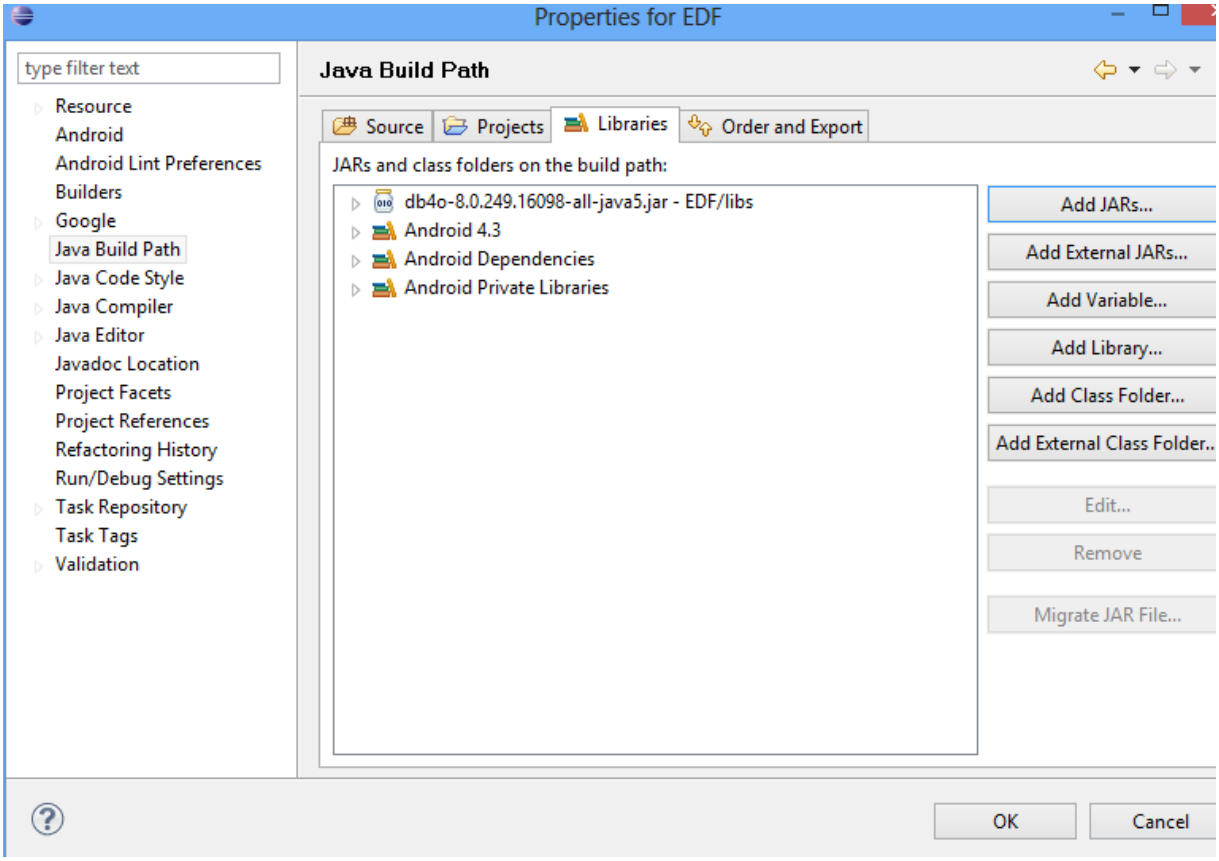
InstallationDB4o

- Importez dans le dossier libs de db4o-8.0.249.16098-all-java5.jar

Dernière version disponible au jour de la publication :
http://www.db4o.com/(S(tqvrulmp3ybmtw55i3mgop55))/Android/default.aspx



- Ajoutez le .jar à votre projet via *Add JARs*...



Attention pour pouvoir écrire sur la carte SD de votre téléphone ou tablette vous devez rajouter, dans le fichier AndroidManifest.xml de votre projet, la permission d'écriture, ce après la fermeture de la balise <uses-sdk :

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

Classe Modele

- Créez la class Modele

Le projet tente de respecter l'approche MVC. (Rappel : dans Android une Activity = une vue + un contrôleur)
Attributs :

```
private String db4oFileName;
private ObjectContainer dataBase;
```

private File appDir;

- Ajoutez la méthode open()

```
db4oFileName = Environment.getExternalStorageDirectory() + "/baseDB4o"
                    + "/BaseEDF.db4o";
dataBase = Db4oEmbedded.openFile(Db4oEmbedded.newConfiguration(),
            db4oFileName);
```

- Ajoutez la méthode createDirectory()

Cette méthode est utile pour stocker la BDD dans un dossier accessible.

```
appDir = new File(Environment.getExternalStorageDirectory()+"/baseDB4o");
if(!appDir.exists() && !appDir.isDirectory())
    {
        appDir.mkdirs();
                        }
```

Exemple d'utilisation du SGBD DB4o (cf db4o-8.0-java\db4o-8.0\doc\tutorial)

- Pour trouver tous les objets d'une classe (exemple Client) :

```
ObjectSet<Client> result = dataBase.queryByExample(Client.class);
   while (result.hasNext()) {
                        //TODO
   }
```

- Pour trouver un objet d'une classe (exemple Client) ou tous les objets d'une classe avec une spécificité (ici le client ayant l'identifiant id) :

```
Client vretour = new Client();
vretour.setIdentifiant(id);
ObjectSet<Client> result = dataBase.queryByExample(vretour);
   vretour= (Client) result.next();
```

Remarques : S'il y a plusieurs clients, utilisez un while et une ArrayList.

- Pour sauvegarder un objet d'une classe (exemple Client, objet cli)

```
open();
dataBase.store(cli);
dataBase.close();
```

Attention :
L'insertion et la mise à jour des objets d'une classe utilisent le même ordre : store.
DB4o utilise ses propres pointeurs pour différencier une mise à jour, d'une création.
Le contexte STA de notre projet (c.f. les états possibles d'une Activity sous Android) impose que toutes modifications de la base de données s'accompagnent d'un open et d'un close de la base de données, ce pour assurer la persistance des données.
La fermeture de la base de données s'accompagne de la perte des pointeurs, et impose pour une mise à jour d'objet, une recherche préalable de cet objet puis une mise à jour.

- Ajoutez et écrivez les 4 méthodes suivantes :

  - listeClient qui permet de renvoyer une ArrayList de Client ;
  - trouveClient qui permet de renvoyer une instance de Client à partir de son identifiant ;
  - saveClient avec comme argument une instance de Client qui permet de sauvegarder cette instance (c.f. attention : si ce client existe déjà il faut mettre à jour celui-ci dans la base via un appel à la méthode recopieClient de Client) ;
  - chargeDataBase qui permet de charger la base de données.

chargeDataBase est ici une méthode fictive qui va nous permettre de tester notre application, car nous n'avons pas encore codé le transfert de données. Faire un appel à la méthode listeClient, si l'ArrayList retournée est vide, il faut créer 5 clients (adresses réelles sur votre ville pour la géolocalisation) avec des appels à saveClient.

Exemple : pour la création d'un client

```
try { vcli=new Client("1001", "Dupont", "paul","10 rue Anne Frank",
"49000", "angers","0624553212", "19950055123", 1456.24,new
SimpleDateFormat("dd/MM/yyyy").parse("15/03/2012")));
                    } catch (ParseException e) {}
```

Le *try* est obligatoire pour le *parse* de date.

- Ecrivez le constructeur de Modele :

```
createDirectory();
open();
// si parties identification et import non développées
chargeDataBase();
dataBase.close();
```

Test DB4o & Installation de OME Manager

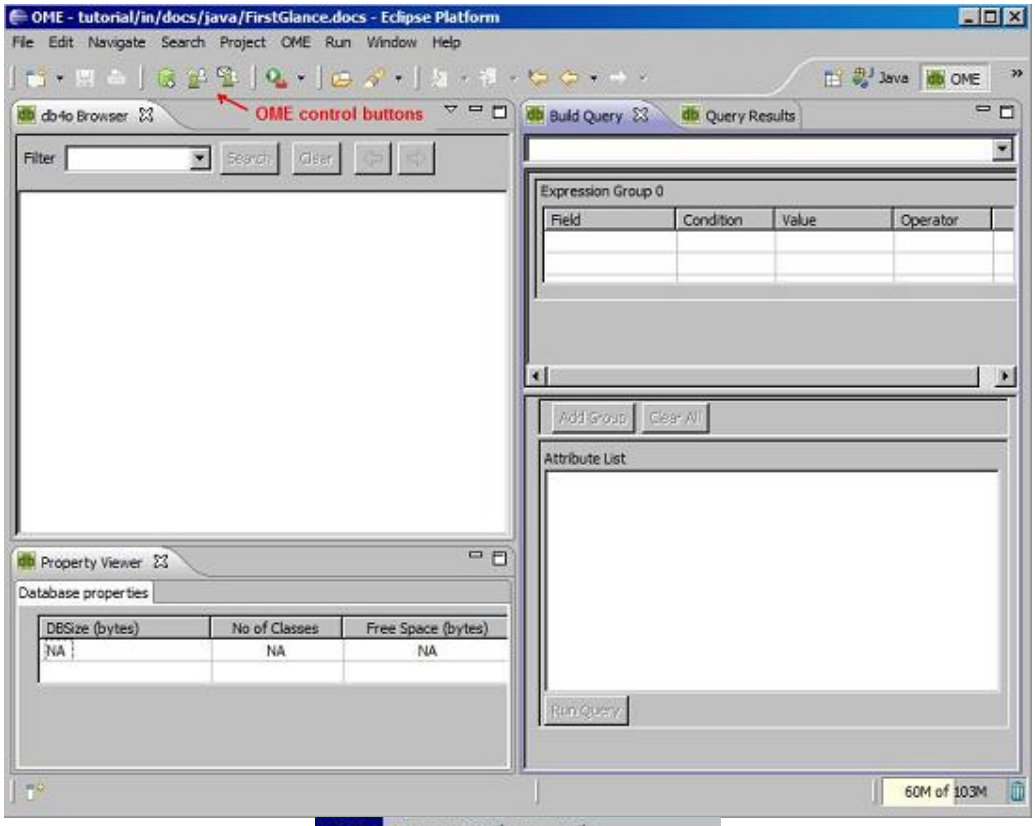
- Modifiez la class MainActivity,
  - déclarez un attribut de la class Modele
  - instanciez cet objet dans la méthode onCreate.

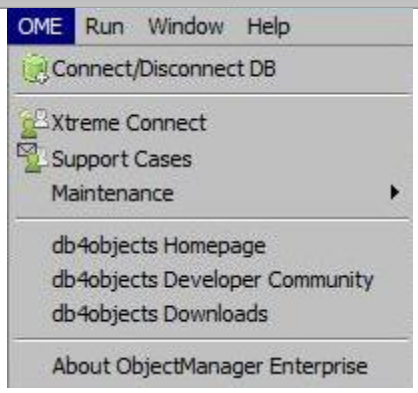
- Exécutez votre application

- Installez OME manager

"OME manager" est le plugin d'Eclipse permettant de visualiser le contenu d'une base de donnée DB4o et de faire des requêtes.

'Help' -> 'Software Updates...' -> 'Available Software
Add Site...' -> 'Local...' et EDF\db4o-8.0-java\db4o-8.0\ome\ObjectManagerEnterprise-Java-8.0.0
Window->Open Perspective->Other choisir "OME"





Choisir *Connect.*
Parcourir votre portable ou tablette / émulateur Android pour ouvrir votre base de donnée DB4o en mode *read only.*
Dans *filter* inscrire *Client.*
Avec clic droit *View all object*s (exemple pour 5 clients)

Build Query | Query Results

com.example.edf.Client

Row Id	adresse	ancien_releve	cp	date_ancien_re...	date_dernier_r...	dernier_releve
1	10 rue Anne Fr...	1456.24	49000	Thu Mar 15 00...	null	null
2	10 Avenue des...	1456.24	49000	Thu Mar 15 00...	null	null
3	10 rue Boisnet	1456.24	49000	Thu Mar 15 00...	null	null
4	25 rue du quin...	5656.26	49000	Wed Aug 15 0...	null	null
5	20 rue des lutins	57356.24	49000	Tue May 15 00...	null	null

Vous ne pouvez pas, via OME manager, modifier les enregistrements de DB4o.
Vous pouvez visualiser l'ensemble des classes gérées et, pour chacune, les instances d'objets.

Affichage de la liste des clients à traiter

Exemple d'écran d'affichage de la liste des clients à traiter que nous souhaitons créer :

Explications :

Nous allons donc utiliser une classe d'Android, la ListView, qui comme son nom l'indique est une visualisation sous forme d'une liste comprenant des items ; pour afficher une liste d'items dans celle-ci, il lui faut un adaptateur de données.

Ici chaque élément de la liste est une représentation complexe d'une instance de la classe client.

Nous devons créer :

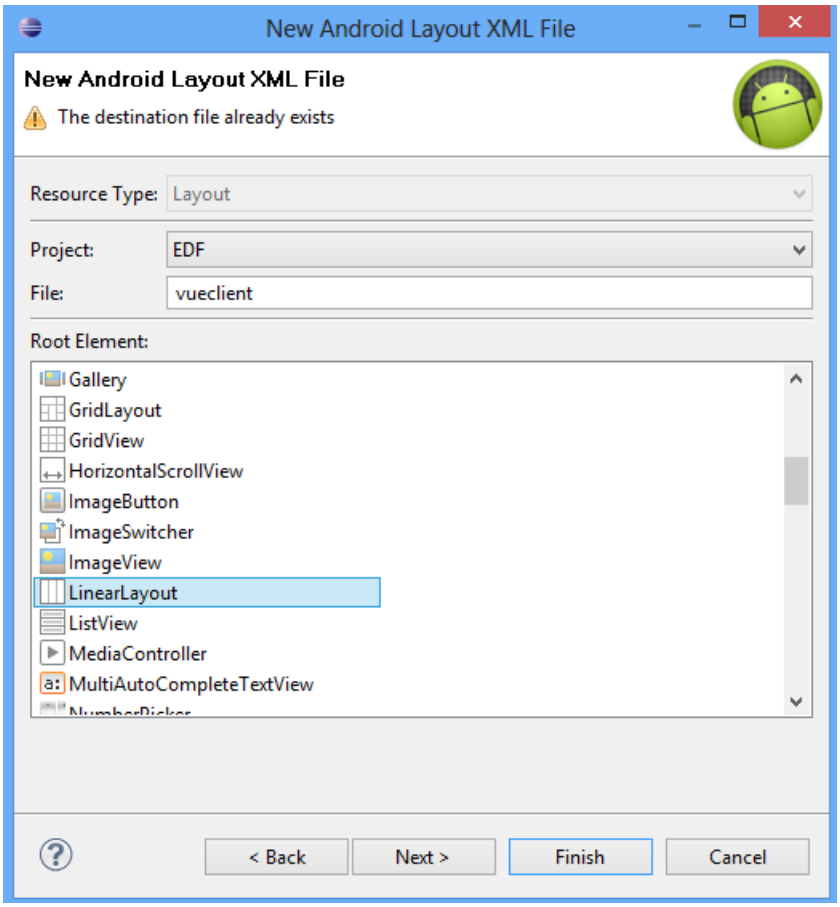
- une vue qui va nous permettre d'afficher un élément de la liste ;
- une classe de type *Adapter* qui va permettre de remplir la vue d'un élément de la liste et de gérer les événements de l'utilisateur (clic,...) ;
- l'*Activity* qui permettra d'afficher la ListView et à partir d'une collection, d'appeler l'*Adapter* pour remplir chaque élément de la liste.

Création de la vue d'un élément de la liste

- Créez un *Layout* vueclient (*res / layout* clic droit *new android layout xml file*)

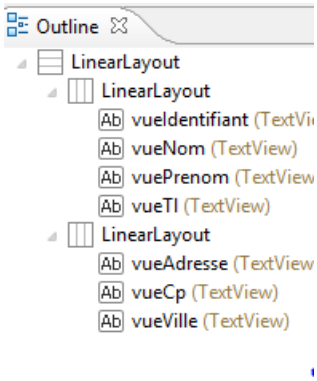
Ce Layout permet d'afficher un client tel que décrit au dessus.

Identifiant Nom Prénom Téléphone
Adresse CP Ville



Conseil : votre *LinearLayout* de base est vertical et, à l'intérieur, 2 *LinearLayout* horizontaux correspondant à vos 2 groupes de données ou utiliser un *Tablelayout*.

Exemple :



Rappel :

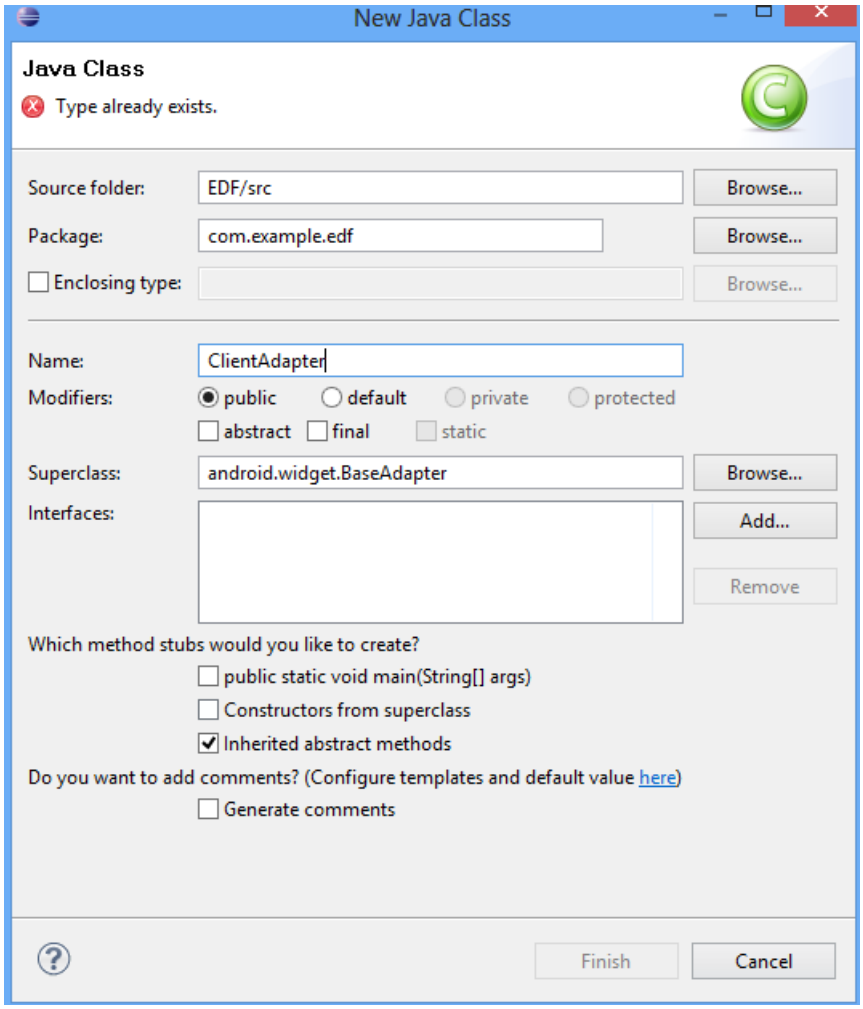
match_parent	L'objet concerné prend toute la place de son conteneur
wrap_content	L'objet concerné prend uniquement la place nécessaire

Nous venons donc de créer notre gabarit d'affichage (vue spécifique d'un élément de la liste) d'un objet de type Client pour notre liste.

Création de notre classe de type Adapter

Maintenant, nous allons nous pencher sur notre adaptateur personnalisé.

- Créez une classe nommée *ClientAdapter*, héritant de la classe *BaseAdapter*.



Le générateur de classes a ajouté directement les méthodes nécessaires pour le bon fonctionnement de l'adaptateur.

On retrouvera ces méthodes :

- **getCount**() qui retournera le nombre d'éléments dans notre liste.
- **getItem**() qui retournera notre objet client à la position indiquée.
- **getItemId**() qui retournera l'id du client.
- **getView()** qui retournera la vue de l'item pour l'affichage.

- Ajoutez un attribut listClient de type List<Client>

```
private List<Client> listClient;
```

- Ajoutez un attribut layoutInflater de type *LayoutInflater*

```
private LayoutInflater layoutInflater;
```

Cet attribut a pour mission de charger notre fichier XML de la vue pour l'item.

- Créez un constructeur ayant comme paramètres un objet de type Context et un objet de type List<Client>

Le Context est une référence à un objet qui permet d'accéder à des informations plus spécifiques de l'objet (état, ressources, préférences,...). Il est souvent lié à l'Activity en cours d'exécution.

```
public ClientAdapter(Context context, List<Client> vListClient) {
	layoutInflater = LayoutInflater.from(context);
	listClient = vListClient;
}
```

- Complétez les différentes méthodes :

  - getCount qui retournera la taille de la liste

```
@Override
public int getCount() {
	// TODO Auto-generated method stub
	return Listec.size();
}
```

  - getItem qui retournera l'item

```
@Override
public Object getItem(int arg0) {
	// TODO Auto-generated method stub
	return Listec.get(arg0);
}
```

  - getItemId qui retournera la position de l'item.

```
@Override
public long getItemId(int arg0) {
	// TODO Auto-generated method stub
	return arg0;

}
```

  - *getView* qui retournera la vue de l'item.

Avant de modifier la méthode *getView* nous devons créer une classe (une classe dans une classe !) qui sera nommée *ViewHolder*.
Elle nous servira à mémoriser les éléments de la liste en mémoire pour qu'à chaque rafraichissement l'écran ne scintille pas (c'est une sorte de buffer/cache comme en graphisme).

```
private class ViewHolder {
	TextView textViewIdentifiant;
	TextView textViewNom;
	TextView textViewPrenom;
	TextView textViewTelephone;
	TextView textViewAdresse;
	TextView textViewCodePostal;
	TextView textViewVille;
}
```

Le nom des *TextView* n'a pas de rapport avec votre *Layout vueclient*

La méthode *getView* utilise le *ViewHolder*, vérifie que la vue présente n'est pas *null* sinon elle la crée, et ensuite charge le XML en mémoire pour l'attribuer à notre objet, taggue notre objet pour pouvoir le récupérer à la prochaine mise à jour graphique, et pour finir, attribue les données et retourne la vue.

Exemple :

```
@Override
	public View getView(int position, View convertView, ViewGroup parent) {
		ViewHolder holder;
		if (convertView == null) {
			holder = new ViewHolder();
			convertView = layoutInflater.inflate(R.layout.vueclient, null);
			holder.textViewIdentifiant = (TextView) convertView
					.findViewById(R.id.vueIdentifiant);
			etc...
			convertView.setTag(holder);
		} else {
			holder = (ViewHolder) convertView.getTag();
		}

		holder.textViewIdentifiant.setText(listClient.get(position)
				.getIdentifiant());
		etc...
		return convertView;
	}
```

A adapter selon votre *Layout vueclient* et votre classe Client getter/setter.

Création de l'Activity AfficheListeClient

- Créez une nouvelle Activity AfficheListeClient

Depuis *src / com.example.edf-votrenom* clic droit *new Android Activity*.

- Ajoutez dans le *LinearLayout* une *ListView* d'id *lvListe*

Cette *ListView* permet d'afficher notre liste de client.

Vous pouvez améliorer votre *ListView* avec des séparateurs.

Exemple :

```
android:divider="#000000"
android:dividerHeight="1dp"
etc,etc
```

- Déclarez dans votre *Activity* un attribut listView de type ListView

```
private ListView listView;
```

- Modifiez la méthode onCreate(Bundle savedInstanceState), afin d'afficher notre liste

```
listView = (ListView)findViewById(R.id.lvListe);
ClientAdapter clientAdapter = new ClientAdapter(this, listeClient);
listView.setAdapter(clientAdapter);
```

La question est ici : comment récupérer `listeClient` ?
Il suffit :

- de déclarer `listeClient` comme une *List* de Client ;
- de déclarer une instance de la classe Modele ;
- à partir de l'instance de la classe Modele, d'appeler la méthode listeClient pour initialiser `listeClient`.

Test de l'application

Les *Intents* sont des objets permettant de faire passer des messages contenant de l'information entre composants principaux. La notion d'*Intent* peut être vue comme une demande de démarrage d'un autre composant, d'une action à effectuer.

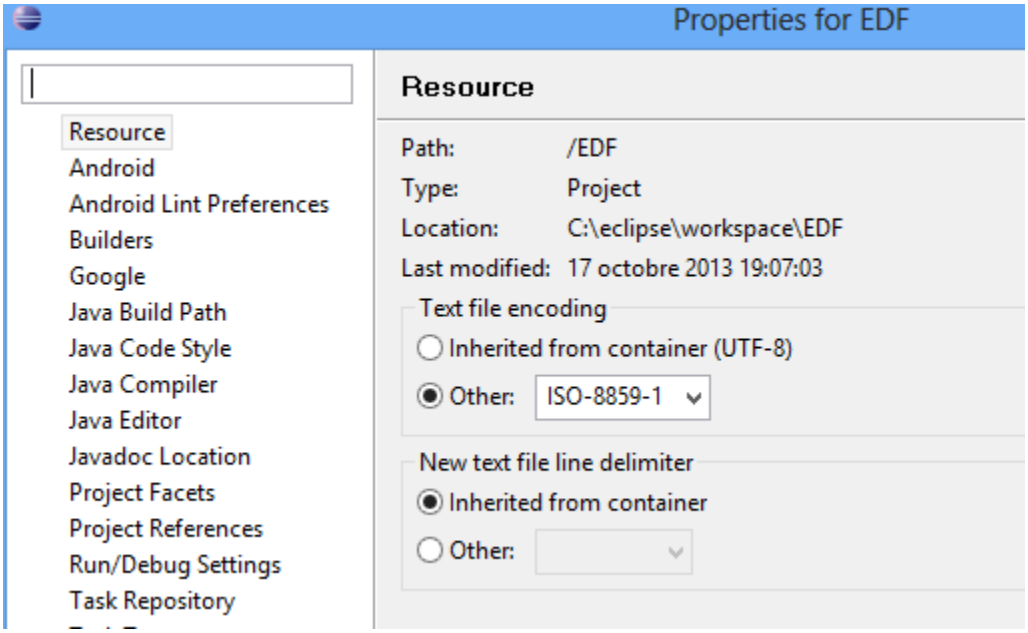
- Associez l'activity AfficheListeClient au clic de l'image 'Clients' (Relevé des compteurs EDF)

A partir de votre MainActivity faites appel à AfficheListeClient sur le clic de l'image Client afin de tester votre affichage de la liste de vos clients.

```
Intent myIntent = new Intent(this, AfficheListeClient.class);
startActivity(i);
```

Le *this* est souvent remplacé par *getApplicationContext*().

Attention aux caractères accentués, il faut modifier l'affichage :



- Testez l'affichage des clients

Vous pouvez améliorer l'affichage du téléphone dans ClientAdapter en formatant la chaine de caractères.

Exemple :

```
String s = listClient.get(position).getTelephone();
s = String.format("%s.%s.%s.%s.%s", s.substring(0, 2),
                  s.substring(2, 4), s.substring(4, 6), s.substring(6, 8),
                  s.substring(8, 10));
holder.textViewTelephone.setText(s);
```

et, avant "return convertView;", vous pouvez rajouter ici une modification de couleur selon les changements de lignes…

```
if(position % 2 == 0){
   convertView.setBackgroundColor(Color.rgb(238, 233, 233));
}
else {
  convertView.setBackgroundColor(Color.rgb(255, 255, 255));
}
```

… et colorier les clients déjà traités par une couleur particulière (exemple vert).

Sélection d'un client

Dans le contexte d'une interface graphique, les listeners permettent au programmeur de réagir suite aux actions de l'utilisateur (clic de souris, touche du clavier enfoncée, etc.).

Dans *l'Activity AfficheListeClient* nous allons donc créer un *Listener* qui écoute sur le clic d'un item de la listView.

- Ajoutez les imports suivants :

```
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
```

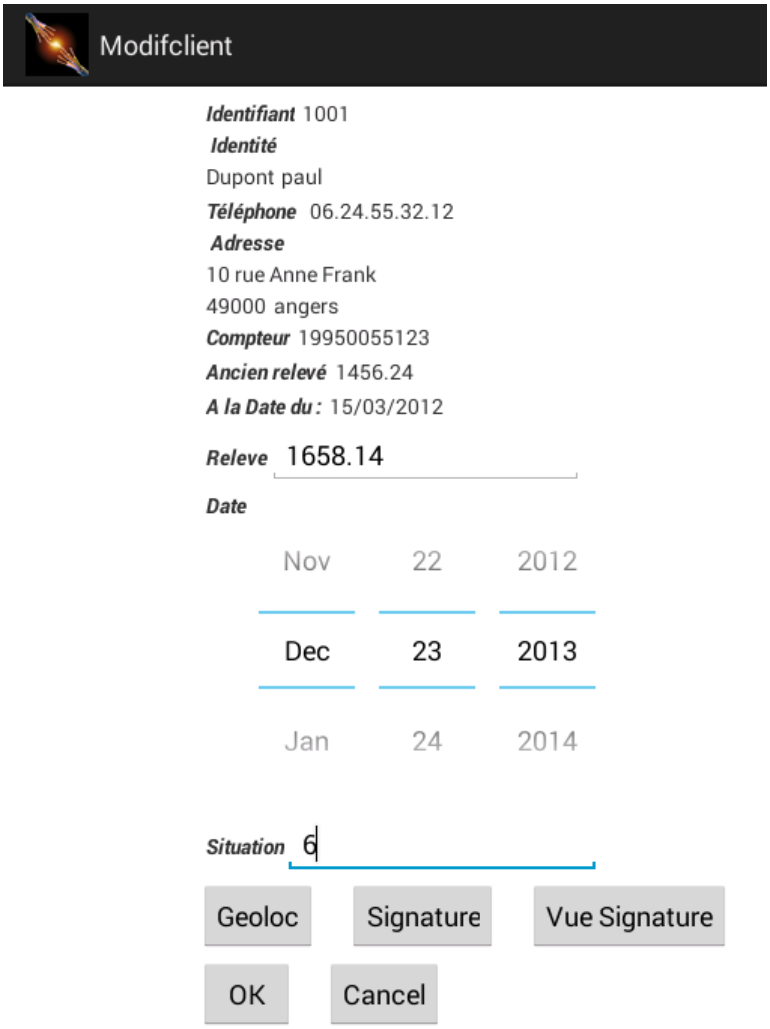
- Ajouter en fin de la méthode onCreate :

```
listView.setOnItemClickListener(new OnItemClickListener() {
@Override
 public void onItemClick(AdapterView<?> a, View v, int position, long id) {
Toast.makeText(getApplicationContext(),"Choix : "+listeClient.get(position).getIdentifiant(),
Toast.LENGTH_LONG).show();
}
});
```

- Testez la sélection d'un client

Modification des informations du client

Exemple d'écran de modification des informations du client que nous souhaitons obtenir :



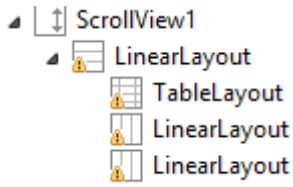
Activity ModificationClient

- Créez *l'Activity ModificationClient*

Cette Activity va permettre d'afficher les informations non modifiables de Client et de saisir les informations modifiables de Client.

Conseil pour le layout :

Dans la vue, clic sur *Graphical layout*, puis dans fenêtre *outline* clic droit sur *RelativeLayout* avec *change layout* en *ScrollView* afin que vous puissiez afficher cette vue sur de petits écrans :



L'arborescence décrite ci-dessus vous permettra dans la *TableLayout*, avec autant de *TableRow* que nécessaire, d'afficher toutes les informations, ainsi que les informations à saisir.

Les 2 *LinearLayout* horizontaux vous permettront d'afficher les 2 rangées de boutons.

Les informations modifiables sont aussi affichées car initialisées dans le constructeur de Client.

- Associez le lancement de l'Activity ModificationClient au clic d'un item de la liste des clients

Depuis *l'Activity AfficheListeClient*, à partir de la méthode *onItemClick* du *setOnItemClickListener* de `listeClient`, nous appelons *l'Activity ModificationClient* en lui passant l'identifiant du client sélectionné.

Exemple *d'Intent* avec paramètre :

```
Intent myIntent = new Intent(this, MaDeuxiemeActivite.class);
myIntent.putExtra("param1", maChaineDeCaractere);
startActivity(myIntent);
```

et réception

```
Bundle b = getIntent().getExtras();
String param1 = b.getString("param1");
```

Dans *l'Activity ModificationClient* : réception de l'identifiant, appel via la classe Modele de la méthode trouveClient en lui passant l'identifiant qui permet alors d'afficher les informations du client.

- Modifiez la méthode onCreate(Bundle savedInstanceState) pour afficher les informations du client

Par défaut cf. constructeur de Client
dernier_releve = 0 date_dernier_releve=Date du jour situation=0

Conseils

- Utilisation dans le *Layout* d'un *DatePicker* pour la saisie de la date
- Modification d'un *DatePicker* afin qu'il n'affiche pas les semaines et l'heure (à adapter)

```
DatePicker datePicker = (DatePicker) findViewById(R.id.dreleve);
datePicker.setCalendarViewShown(false);
```

- Modification de la date d'un *Datepicker* (à adapter)

```
Date date = client.getDateDernierReleve();
Calendar calendar = Calendar.getInstance();
calendar.setTime(date);
datePicker.updateDate(calendar.get(Calendar.YEAR),
                      calendar.get(Calendar.MONTH),
                      calendar.get(Calendar.DAY_OF_MONTH));
```

- Modification d'un entier en chaine de caractères : Integer.toString(value)
- Modification d'une chaine de caractères en entier : Integer.parseInt(value)
- Création d'une date à partir d'un *DatePicker* (à adapter)

```
int day = ((DatePicker)findViewById(R.id.dreleve)).getDayOfMonth();
int month = ((DatePicker)findViewById(R.id.dreleve)).getMonth();
int year =  ((DatePicker)findViewById(R.id.dreleve)).getYear();
Calendar calendar = Calendar.getInstance();
calendar.set(year, month, day);

La date est (calendar.getTime());
```

- Affichage d'une date en format jj/mm/aaaa

```
new SimpleDateFormat("dd/MM/yyyy").format(votredate)
```

- Ecrivez la méthode *save* qui utilise les *setter* de Client à partir des informations modifiées et qui appelle *saveClient* de Modele

- Gérez les clics des boutons

  - Sur le clic d'annulation, appel d'une méthode qui fait *finish*();
  - Sur le clic de validation, appel d'une méthode qui vérifie que :
    - Situation !=0
    - Relevé compteur > Relevé compteur initial sauf si situation = (1,5,6)
    - Date relevé > date relevé ancien compteur
    - Situation appartient au vecteur (1,2,3,4,5,6,...).

puis mise à jour de Client (appel à *save*) et *finish*() qui implique un retour à l'affichage des clients avec, normalement, une couleur différente si "situation" est différent de zéro.