TP_C2 :IMPORT DE DONNEES

Objectif du TP :

- Connexion à la BDD
- IMPORT de données avec le serveur

Rappel du thread AsynTask

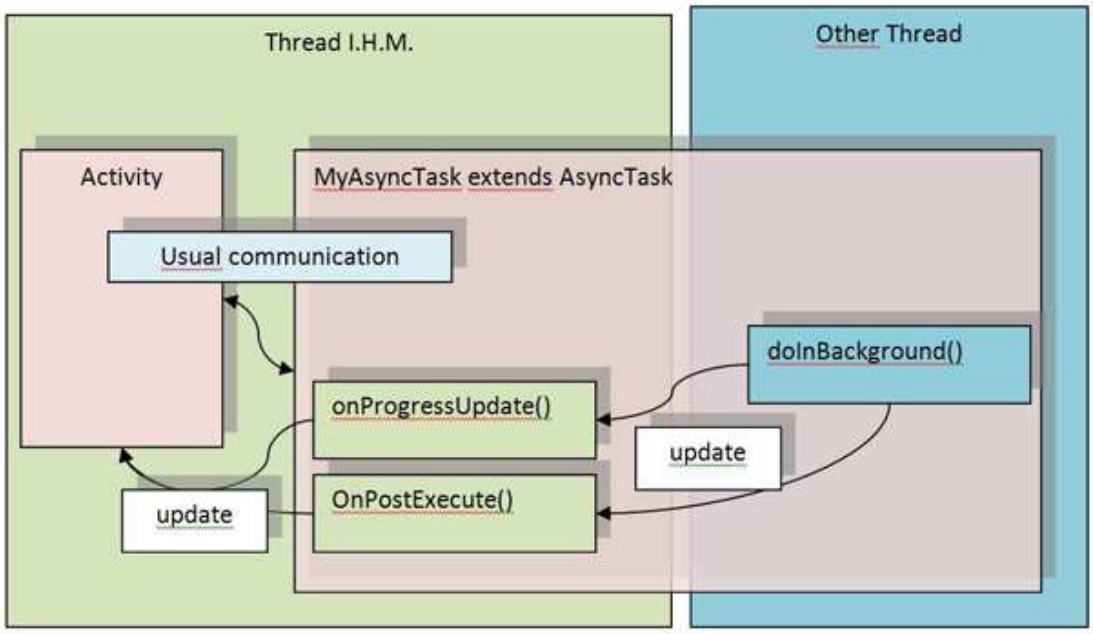
Les étapes d'AsynchTask :

doInBackground est la méthode qui s'exécute dans une autre Thread. Elle reçoit un tableau d'objets lui permettant ainsi d'effectuer un traitement en série sur ces objets. **Seule cette méthode est exécutée dans un Thread à part, les autres méthodes s'exécutent dans la Thread de l'IHM** .

onPreExecute est appelée par la Thread de l'IHM avant l'appel à *doInBackground* , elle permet de pré-initialiser les éléments de l'IHM.

onProgressUpdate est appelée par la méthode *publishProgress* à l'intérieur de la méthode *doInBackground.*

onPostExecute est appelée lorsque la méthode *doInBackground* est terminée.

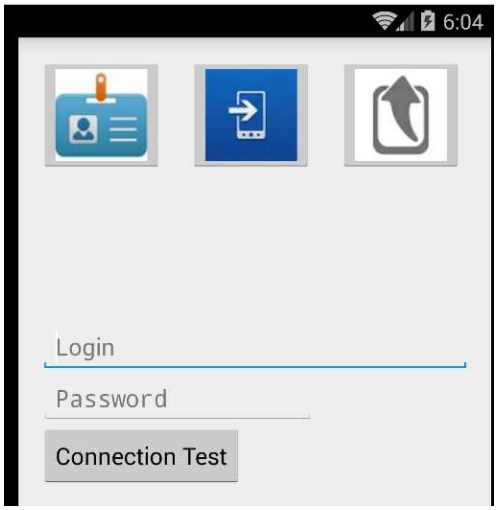


1. LAYOUT POUR ACTIVITY IDENTIFICATION

- Récupérer les images dans Partage pour les 3 boutons : identification / import / export

- Copier les Strings dans votre fichier Strings

```
<!-- ressources de l'activity Identification pour la communication avec le serveur-->
<string name="editTextLogin">Login</string>
<string name="editTextMdp">Password</string>
<string name="buttonTestConnexion">Connection Test</string>
<string name="identification">Identification</string>
<string name="importe">import</string>
<string name="exporte">export</string>
```



- Récupérer le code du Layout dans **Partage**

```
on utilisera des images Buttons comme pour l'identification ci-dessous :
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imageButtonIdentification"
    android:src="@drawable/identification"
    android:contentDescription="@string/identification"
    android:cropToPadding="false"
    android:layout_marginRight="20dp" />
```

- Stratégie d'identification :

  - Comme la **STA est personnelle**, l'utilisateur s'est déjà identifié avec son code PIN. Ici l'utilisateur est le moniteur qui va récupérer **ses** cours individuels dans un but de consultation dans un premier temps.

  - Si base de données DB4o est **vide** (liste des cours est vide)
    1. si OUI,
       alors seul le bouton identification est visible pour que l'utilisateur puisse vérifier sa connexion au SGBDR => réponse OK ou pas
       a. si connexion OK alors le bouton IMPORT est visible
       b. sinon, il peut ressaisir ses données pour retenter la connexion !
    2. si NON, cela signifie qu'il a déjà fait un import, donc pas besoin de le ré-identifier donc :
       - on peut lancer la MainActivity directement. Il pourra voir donc ses cours déjà enregistré (sur le clic du bouton Liste des Cours)
       - et le bouton EXPORT est visible.

  - Si le bouton IMPORT est visible (donc connexion SGBR OK)
    1. si OUI,
       alors on lancera la nouvelle activité nommée **ImportActivity** qui se chargera d'importer les données du SGBDR et de les retourner en JSON.

  - Si le bouton EXPORT est visible (donc base DB4o déjà remplie)
    2. si OUI,
       alors on lancera la nouvelle activité nommée **ExportActivity** qui se chargera d'exporter les objets (CoursIndividuels) stockés dans DB4o vers le SGBDR.

3. MODIFIER LE MODELE POUR DB40

Pour simplification, on considérera que seul un moniteur (participant) peut utiliser le S.T.A.

- il va **s'identifier** (en vérifiant dans la table Participant son login et son mot de passe)
- il va **récupérer ses cours**….donc juste **SES** cours individuels. *On pourra par la suite récupérer les cours individuels à venir (c'est à dire que la date du cours soit supérieure ou égale à la date du jour)*

Ce que vous devez faire :

- vous avez déjà une classe Cours Individuels dans votre projet
  - créer une **surcharge de constructeur** comme ci-contre :

```
/*nouveau constructeur pour l'importation de la BDDR
* en effet, pour le moniteur, on récupère juste un nom */
public CoursIndividuels(String i, Date d, Integer h, String unM) {
    super(i, d, h);
    leMoniteur = new Participant(unM);
}
```

- Créer dans Modele la méthode utile permettant de **supprimer les cours dans la base DB40** (Suppression de toutes les instances de la classe Cours (donc les cours individuels et collectifs)

```
public void SupprimerLesCours() {
//on parcours les 2 listes de cours, et on supprime chaque instance dans la BD4O
avec database.delete
    open();
    ObjectSet<Cours> resultCC = dataBase.queryByExample(CoursCollectifs.class);
    ObjectSet<Cours> resultCI = dataBase.queryByExample(CoursIndividuels.class);

    while (resultCC.hasNext()) {
        dataBase.delete(resultCC.next());
    }
    while (resultCI.hasNext()) {
        dataBase.delete(resultCI.next());
    }
    dataBase.close();
}
```

- On a déjà dans Modele la méthode utile permettant de **sauver un Cours dans la base DB40**

4. ACTIVITY IDENTIFICATION

Cette activity va suivre la stratégie d'identification expliquée ci-dessus :

Au démarrage de cette activity :

- Récupérer TOUS les différents éléments graphiques par `findViewById`. Les mettre en `setVisibility(View.GONE);`
- **Si pas** d'enregistrement de la classe (cours) dans db4O (listeCours est vide) donc pas de données dans la DB4o donc il faut les récupérer en s'identifiant
  - Mettre **l'ImageButton identification à visible**.
  - Sur le clic de l'imageBouton Identification, mettre le login, le mot de passe et le bouton testConnexion **visibles**
  - Puis, le moniteur rentre son id et mp et vérification sur le serveur via TestConnexion.php (ici ne sert qu'à vérifier si la connexion est effectuée avec le serveur donc pas vraiment besoin de l'id et du mot de passe mais on va s'en servir juste après pour l'import)
  - Modifier TestConnexion.php **fourni** pour retourner juste **OK ou NOK si la connexion au serveur de base de données est établie.**
  - dans la classe **Connexion, sur** la méthode `protected void onPostExecute (Boolean result)`, décommenter la ligne suivante :

```
// ((IdentificationActivity)mActivity.get()).methodetraitementresultat(sb.toString());
```

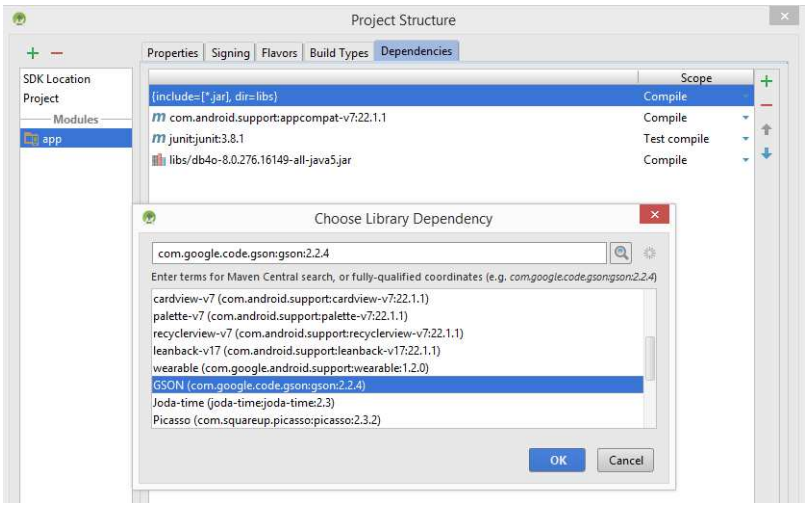
Il faut créer la méthode `methodetraitementresultat` dans l'activity `IdentificationActivity` pour pouvoir afficher le résultat de la connexion (utiliser la fonction *alertMessage* déjà présente).

- S'il existe au moins un enregistrement de la classe Cours dans DB4o (listeCours est non vide) alors **pas besoin de s'identifier** (car il a déjà importé des données).
  - On appelera l'activity MainActivity.
  - On pourra alors afficher les différents cours (pour l'instant tous les cours).Dans un deuxième temps, on affichera juste les cours du moniteur (à savoir **SES** cours **individuels**)

IMPORT ET EXPORT DE DONNÉES

Pour les activity ImportActivity et ExportActivity, nous allons utiliser GSON qui est une bibliothèque permettant de gérer les parsing JSON sous Android d'une manière robuste et facile.

1. AJOUT DE LA BIBLIOTHÈQUE GSON



2. IMPORT DES DONNÉES

Création d'une nouvelle activity ImportActivity **:**

- créer un bouton "lancer importation" qui sur le clic du bouton va lancer l'import des données (url http://***192.168.0.22***/LansleVillage/**import**.php)
- si ok et si des données instances de la classe Cours existent dans la base DB40
  - les effacer en premier *(méthode SupprimerLesCours*() déjà *créée dans Modele)*
  - puis recréer les objets Cours à partir des enregistrements récupérés de l'import. *(méthode Save déjà créée dans Modele)*

Création de import.php à l'image de TestConnexion.php

Nouvelle méthode dans ImportActivity pour récupérer le JSON retourné par le thread de l'importation

Ajouter à l'activity ImportActivity, la méthode recupTuplesBDD (String s), méthode qui sera appelée à partir de la méthode onPostExecute de la classe Connexion

Code de cette méthode :

```
public void recupTuplesBDD(String s) {
    /* a la fin du thread (onPostExecute), on appelle cette methode*/
    CoursIndividuels vCoursI;
    maBDO = new Modele();
    maBDO.SupprimerLesCours();
    Toast.makeText(getApplicationContext(),"suppression des cours EXISTANTS dans
DB4O",Toast.LENGTH_LONG).show();

    JsonElement json = new JsonParser().parse(s.toString());
    JsonArray varray = json.getAsJsonArray();
    Gson gson = new GsonBuilder().setDateFormat("yyyy-mm-dd").create();

    for(JsonElement obj : varray )
    {
        //ATTENTION : les champs de la BDD doivent avoir le MEME NOM nom que les
proprietes de la classe
        //sinon CORRESPONDANCE IMPOSSIBLE
        vCoursI = gson.fromJson(obj.getAsJsonObject(),CoursIndividuels.class);

Toast.makeText(getApplicationContext(),vCoursI.toString(),Toast.LENGTH_LONG).show();

        maBDO.save(vCoursI);

    }
    Toast.makeText(getApplicationContext(),"FIN de
l'IMPORT",Toast.LENGTH_LONG).show();
    finish();
}
```

Cette dernière affiche un message de bonne ou mauvaise fin et rend accessible client et export si ces derniers sont inaccessibles en passant le résultat à la main activity ce via le `setResult de l'activity` et le `onActivityResult de la main activity.`

Comment savoir si l'import est terminé ?

Nous allons faire en sorte que l'ImportActivity "prévienne" l'activité appelante en lui renvoyant un code retour (***resultCode***)
Comme l'activité appelante peut appeler plusieurs activités différentes, il faut les différencier en les appelant avec une sorte d'identifiant nommé (***requestCode***).

Nouveauté pour appeler une Activity, le `startActivityForResult`

La communication avec la main activity se fera via un `startActivityForResult idem pour import et export`

Dans l'activity main

Permet de connaitre l'activity appelée pour le retour
static final int CODE_RETOUR_ACTIVITY_IMPORT = 1;
static final int CODE_RETOUR_ACTIVITY_EXPORT = 2;
etc

```
// On créé un objet Bundle, c'est ce qui va nous permetre d'envoyer
                  // des données à l'autre Activity
                  Bundle objetbundle = new Bundle();

                  objetbundle.putString("xxxx", "xxxxx");
                  // On créé l'Intent qui va nous permettre d'afficher l'autre
                  // Activity
                  Intent intent = new Intent(this, xxxx.class);
                  // On affecte à l'Intent le Bundle que l'on a créé
                  intent.putExtras(objetbunble);
                  // On démarre l'autre Activity
                  startActivityForResult(intent, CODE_RETOUR_ACTIVITY1);
```

puis contrôle du retour de l'autre activity

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
            // on regarde quelle Activity a répondu (request code) et ce qu'elle nous
envoie le resultCode

            switch (requestCode) {
            case CODE_RETOUR_ACTIVITY_IMPORT:
                  // on créé une AlertDialog pour indiquer le retour de l'activty

                  // On regarde qu'elle est la réponse envoyée par l'activity Import
                  // resultCode et en fonction de la réponse on ????.
                  switch (resultCode) {
                  case ???:
                        return;

                  case ????:
                        return;
                  }

            case CODE_RETOUR_ACTIVITY_EXPORT:
                  // On regarde qu'elle est la réponse envoyée par l'activity Import
                  // resultCode et en fonction de la réponse on ???.
                  switch (resultCode) {
                  case ???:
                        return;

                  case ???:
                        return;
                  }
            }
      }
```

Dans l'activity appelée (ici ImportActivity)

```
setResult(???); //mettre un code resultCode pour savoir si cela c'est bien passé
finish();
```

CONNEXION AVEC LE SERVEUR

```
import java.lang.ref.WeakReference;
import android.app.Activity;
import android.os.AsyncTask;
import android.widget.Toast;

public class Connexion extends AsyncTask<String, String, Boolean> {
      // Référence à l'activité qui appelle
    private WeakReference<Activity> mActivity = null;
    private String vclassactivity;
    private StringBuilder sb = new StringBuilder();
    public Connexion (Activity pActivity) {
      mActivity = new WeakReference<Activity>(pActivity);
      //permet de récupérer la class de l'appelant
      vclassactivity=pActivity.getClass().toString();
      }
    @Override
    protected void onPreExecute () {// Au lancement, on envoie un message à l'appelant
      if(mActivity.get() != null)
        Toast.makeText(mActivity.get(), "Thread on démarre",Toast.LENGTH_SHORT).show();
    }
    @Override
    protected void onPostExecute (Boolean result) {
      if (mActivity.get() != null) {
        if(result){
          Toast.makeText(mActivity.get(), "Fin ok", Toast.LENGTH_SHORT).show();
//pour exemple on appelle une méthode de l'appelant qui va gérer la fin ok du thread
            if (vclassactivity.contains("Identification"))
            {
                  ((Identification)mActivity.get()).methodetraitementresultat(param);
            }
      }
        else
          Toast.makeText(mActivity.get(), "Fin ko", Toast.LENGTH_SHORT).show();
      }
    }
    @Override
    protected Boolean doInBackground (String... params) {// Exécution en arrière plan
      // on simule ici une activité de 2 sec ne sert à rien
      try {
            Thread.sleep(2000);
      }catch(InterruptedException e) {
        return false;
      }
      String[] vstring={ "Thread","appel du doInBackground" };
      publishProgress(vstring);
        return true;
    }
    @Override
    protected void onProgressUpdate (String... param) {
      // utilisation de on progress pour afficher des message pendant le doInBackground
      // ici pour exemple on appelle une méthode de l'appelant qui peut par exemple
modifier son layout
      if (vclassactivity.contains("Identification"))
      {
            ((Identification)mActivity.get()).alertmsg(param[0],param[1]);
      }
    }
    @Override
    protected void onCancelled () {
      if(mActivity.get() != null)
        Toast.makeText(mActivity.get(), "Annulation", Toast.LENGTH_SHORT).show();
    }
}
```

CONNEXION AU SERVEUR AVEC HTTPURLCONNECTION ET PASSAGE DE PARAMÈTRES EN JSON

```
StringBuilder sb = new StringBuilder();
HttpURLConnection urlConnection = null;
try {
      URL url = new URL(vurl); //passée par paramètre
      urlConnection = (HttpURLConnection)url.openConnection();
      urlConnection.setRequestProperty("Content-Type", "application/json");
      urlConnection.setRequestProperty("Accept", "application/json");
      urlConnection.setRequestMethod("POST");
      urlConnection.setDoOutput(true);
      urlConnection.setConnectTimeout(5000);
      OutputStreamWriter out = new OutputStreamWriter(
                  urlConnection.getOutputStream());
// selon l'activity appelante on peut passer des paramètres en JSON exemple
if (vclassactivity.contains("Identification"))
{
      // Création objet jsonn clé valeur
      JSONObject jsonParam = new JSONObject();
      // Exemple Clé valeur utiles à notre application
      jsonParam.put("ID", vid);
      jsonParam.put("pass", vpass);
      out.write(jsonParam.toString());
      out.flush();
}
      out.close();
      // récupération du serveur
      int HttpResult = urlConnection.getResponseCode();
      if (HttpResult == HttpURLConnection.HTTP_OK) {
            BufferedReader br = new BufferedReader(new InputStreamReader(
                  urlConnection.getInputStream(), "utf-8"));
            String line = null;
            while ((line = br.readLine()) != null) {
                  sb.append(line);
            }
            br.close();
            String[] vstring0 = { "Reçu du serveur",sb.toString()};
            publishProgress(vstring0);
// on se servira du sb dans la méthode onPostExecute pour appel de la gestion de
fin de thread si ok
      } else {
            String[] vstring0 = { "Erreur",urlConnection.getResponseMessage()};
            publishProgress(vstring0);
      }
      // gestion des erreurs
      } catch (MalformedURLException e) {
            String[] vstring0 = { "Erreur", "Pbs url" };
            publishProgress(vstring0);
            return false;
      } catch (java.net.SocketTimeoutException e) {
            String[] vstring0 = { "Erreur", "temps trop long" };
            publishProgress(vstring0);
               return false;
      } catch (IOException e) {
            String[] vstring0 = { "Erreur", "Pbs IO" };
            publishProgress(vstring0);
            return false;
      } catch (JSONException e) {
            String[] vstring0 = { "Erreur", "Pbs json" };
            publishProgress(vstring0);
            return false;
      } finally {
```

```
            if (urlConnection != null){
                urlConnection.disconnect();
            }
        }
        return true;
```