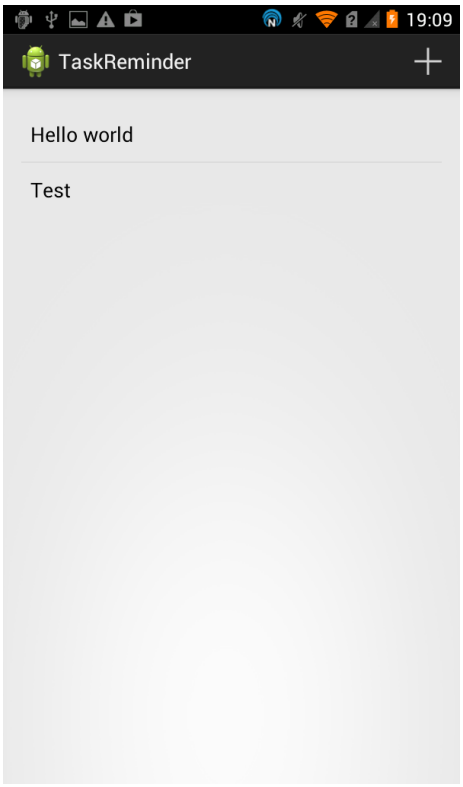
Mon application : Pense Bête

L'objectif de ce TP est de développer une application permettant de stocker une liste de tâches à effectuer. Pensez à télécharger le pack d'icônes mis à disposition par Google : https://developer.android.com/design/downloads/index.html#action-bar-icon-pack.

Explications



Afin d'arriver au résultat final, vous devez :

- Utiliser l'objet « ListView » permettant de gérer une vue en liste
- Une classe héritant de « BaseAdapter » permettant de gérer l'accès aux données (lecture, enregistrement, …)
- Gérer l'évènement appui long pour supprimer un élément
- Ajouter un bouton dans l'« ActionBar » pour insérer de nouvelles tâches

Layout : ListView

Permet d'afficher facilement une liste d'éléments.

Les données sont gérées au travers d'un Adapter. http://developer.android.com/guide/topics/ui/layout/listview.html

Adapter

Un « Adapter » est un moyen de faire le lien entre les données et l'affichage.

http://developer.android.com/reference/android/widget/Adapter.html

Généralement, Il faut créer son propre « Adapter » héritant de « BaseAdapter ». Cette méthode nécessite d'implémenter quelques méthodes afin d'avoir un fonctionnement total, telles que : « getCount() », « getItem() », « getItemId() » et « getView ».

En dehors de « getView », les méthodes se limitent à une seule ligne de code.

« getView() » est un peu plus complexe, car le principe de cette méthode est de peupler la vue correspondant à la ligne passée en paramètre. Il faut ainsi créer une nouvelle vue à partir d'une vue grâce au « LayoutInflater »[1] et la remplir grâce aux données stockées dans notre liste.

[1] http://developer.android.com/reference/android/view/LayoutInflater.html

```
public class TaskAdapter extends BaseAdapter {
	ArrayList<Task> array = new ArrayList<Task>();
	Context context = null;
	private static final String FILENAME = "data.json";

	public TaskAdapter(Context context) {
		this.context = context;
	}

	public void load() { […] }

	public void save() { […]}

	@Override
	public int getCount() {
		return array.size();
	}

	@Override
	public Object getItem(int arg0) {
		return array.get(arg0);
	}

	@Override
	public long getItemId(int arg0) {
		return arg0;
	}

	@Override
	public View getView(int arg0, View arg1, ViewGroup arg2) {
		LayoutInflater li  = LayoutInflater.from(context);
		View v = li.inflate(android.R.layout.simple_list_item_1, null);
		TextView tv = (TextView)v.findViewById(android.R.id.text1);

		Task task = (Task) getItem(arg0);
		tv.setText(task.getName());
		return v;
	}

	public void remove(int pos) {
		array.remove(pos);
		notifyDataSetChanged();
	}

	public void add(Task t) {
		array.add(t);
		notifyDataSetChanged();
	}
}
```

Il ne restera plus qu'à lier votre Adapter à la vue :

```
v = (ListView)findViewById(R.id.listView1);
adapter = new TaskAdapter(this);
adapter.load();
v.setAdapter(adapter);
```

Sérialiser des données

Dans ce projet, afin de réduire au maximum les problématiques liées au parsing des données, nous allons stocker nos données en utilisant le format JSON[2].

L'écriture et la lecture des données s'effectuent à l'intérieur de l'« Adapter ».

Ecrire dans un fichier

Le plus simple est d'utiliser un « FileOutputStream » permettant d'écrire une chaîne de caractères dans un fichier. Cette chaîne de caractères est issue de l'objet « JSONArray », en ayant au préalablement transformé (sérialisé) nos objets « Task » en « JSONObject » (Méthode de l'objet « Task » disponible ci-après).

```
	public void save() {
		try {

				FileOutputStream fos = context.openFileOutput(FILENAME,
Context.MODE_PRIVATE);
				JSONArray jarray = new JSONArray();
				for (Task t : array) {
					jarray.put(t.getJSONObject());
				}
				fos.write(jarray.toString().getBytes());
				fos.close();
			} catch (IOException e) {
				e.printStackTrace();
			}
	}
```

[2] http://en.wikipedia.org/wiki/JSON

```
public class Task {
	private String name;

	public Task(String name) {
		this.name = name;
	}

	String getName() {
		return name;
	}

	public JSONObject getJSONObject() {
	  JSONObject obj = new JSONObject();
	  try {
		obj.put("Name", name);
	  } catch (JSONException e) {
		e.printStackTrace();
	  }
	  return obj;
    }
}
```

Lire dans un fichier

Il ne s'agit ni de la méthode la plus optimisée, ni de la plus propre. Cependant elle permet de comprendre facilement le cheminement entre la lecture du fichier et la génération du tableau de taches.

Grâce à l'objet « BufferedReader » nous pouvons récupérer ligne par ligne le contenu du fichier et le stocker dans une « String ».

L'objet « JSONArray » peut être construit avec cette chaîne de caractères. Une fois construit, il suffit de l'itérer afin de créer de nouveaux objets « Task » et ainsi reconstruire la liste de tâches.

Vous noterez que la variable « array » appartient à notre « Adapter », et quel que soit l'appel de la méthode « notifyDataSetChanged() » permet d'indiquer à la vue que l'affichage doit être rafraîchi.

```
public void load() {
            try {
                  BufferedReader br = new BufferedReader(new
InputStreamReader(context.openFileInput(FILENAME)));
                  String res = "";
                  String line = null;
                  try {
                        while (( line = br.readLine()) != null) {
                              res += line;
                        }
                  } catch (IOException e) {
                        e.printStackTrace();
                  }
                  try {
                        JSONArray jarray = new JSONArray(res);
                        array = new ArrayList<Task>();
                        for (int i = 0; i < jarray.length(); i++) {
                              JSONObject o = jarray.getJSONObject(i);
                              Task t = new Task(o.getString("Name"));
                              array.add(t);
                        }
                        notifyDataSetChanged();
                  } catch (JSONException e) {
                        e.printStackTrace();
                  }
            } catch (FileNotFoundException e) {
                  e.printStackTrace();
            }
      }
```

ActionBar

Il s'agit d'une fonctionnalité apparue depuis la version 3.0 d'Android. Elle permet de se passer du bouton physique de menu. Le principe d'utilisation reste globalement le même qu'à l'époque des menus classiques.

Il suffit d'ajouter une entrée dans le fichier xml contenu dans le dossier « menu » correspondant à votre vue.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >

    <item android:id="@+id/action_add"
          android:icon="@drawable/ic_action_new"
          android:title="@string/action_new"
          android:showAsAction="ifRoom" />

    <item
        android:id="@+id/action_settings"
        android:orderInCategory="100"
        android:showAsAction="never"
        android:title="@string/action_settings"/>
</menu>
```

Puis de contrôler les événements dans la méthode « onOptionsItemSelected » de l'activité en appelant la méthode « add_task() » afin qu'elle permette l'ajout d'une tâche (Pensez à utiliser une pop-up : Objet « Dialog »).

```
@Override
      public boolean onOptionsItemSelected(MenuItem item) {
            switch (item.getItemId()) {
                  case R.id.action_add:
                        add_task();
                        return true;
            }
            return false;
      }
```

Dialog

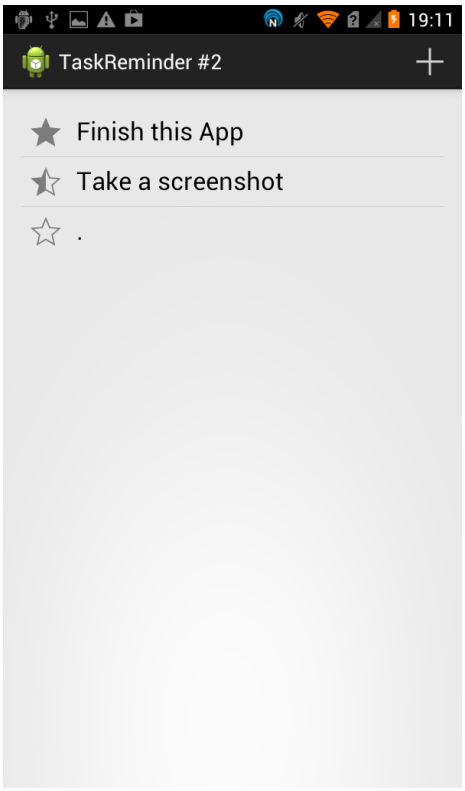
Les « Dialog » sont des petites fenêtres s'occupant de poser une question à l'utilisateur ou bien demander la saisie d'une information.

https://developer.android.com/guide/topics/ui/dialogs.html

Grâce à une classe dédiée à la construction de cet objet, il est possible de créer une fenêtre en trois étapes. Voici un exemple correspondant à notre besoin pour l'appui long :

```
v.setOnItemLongClickListener(new OnItemLongClickListener() {
	Integer position;
	@Override
	public boolean onItemLongClick(AdapterView<?> arg0, View arg1,
		int pos, long id) {
		// 1. Instantiate an AlertDialog.Builder with its constructor
		AlertDialog.Builder builder = new
AlertDialog.Builder(MainActivity.this);
		Task t = (Task)adapter.getItem(pos);
		position = pos;
		// 2. Chain together various setter methods to set the dialog
characteristics
		builder.setMessage("Do you confirm the suppression of the task
'" + t.getName()+ "' ?")
		.setTitle("Confirmation")
		.setPositiveButton(R.string.yes, new
DialogInterface.OnClickListener() {
			public void onClick(DialogInterface dialog, int id) {
				// User clicked OK button
				adapter.remove(position);
				adapter.save();
			}
		})
		.setNegativeButton(R.string.cancel, new
DialogInterface.OnClickListener() {
			public void onClick(DialogInterface dialog, int id) {
			}
		});
		// 3. Get the AlertDialog from create()
		AlertDialog dialog = builder.create();
		dialog.show();
		return false;
	}
});
```

Aller plus loin : Ajouter une notion de priorité



Dans ce projet, nous avons utilisé un « itemLayout » standard fourni par Android. Pour aller plus loin, il serait intéressant de créer son propre layout permettant l'ajout d'une icône comme la capture d'écran à gauche de paragraphe.

Il serait aussi nécessaire de modifier la fenêtre Dialog d'ajout afin d'insérer un « Spinner » permettant de sélectionner le niveau de priorité.

Résultat

Le projet « TaskReminder » et la version améliorée « TaskReminder2 » sont disponibles sur le dépôt suivant :

https://github.com/steven-martins/tp-android