# Annexe 4 – Améliorations à l’annexe 2

## classes DecimalFormat, Spinner, ArrayAdapter, Hashtable

À ce moment-ci, tout le monde devrait avoir terminé l’annexe 2 ( Transfert Intérac ). À la fin de cette annexe, on avait parlé de 3 améliorations notables qu’on pourrait lui apporter. Regardons ces améliorations de plus près.

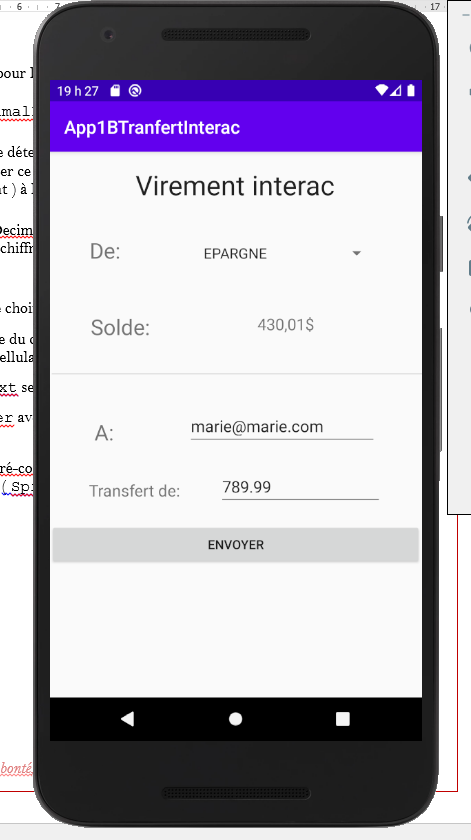
( *25 min* )

## 1.Utilisation d’un format pour les nombres décimaux / les montants monétaires

🡪solution : classe DecimalFormat.

🡪Cette classe permet de déterminer un modèle pour des nombres décimaux. Ensuite, on peut appliquer ce modèle à **l’affichage de ces nombres** ( sans les modifier intrinsèquement ) à l’aide de la méthode format.

* Aller voir la classe DecimalFormat dans l’API et afficher les montants de notre app avec deux chiffres après la virgule.

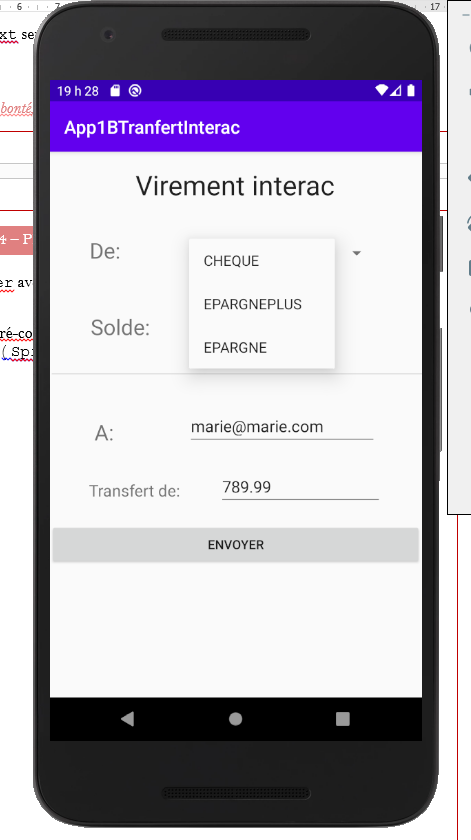


*( 30 min )*

## 2. Utilisation d’une liste de choix

Faire entrer le nom/type du compte et risquer que l’usager ne fasse une faute de frappe ( surtout sur un cellulaire ) n’était pas la meilleure idée.

Remplaçons le EditText servant à entrer le nom du champ par un Spinner. ( convertView dans le fichier .xml )



Pour remplir le Spinner avec nos 3 types de comptes, on a besoin d’un adaptateur.

Différents adaptateurs pré-conçus sont disponibles et servent à initialiser différentes composantes ( Spinner, ListView, AutoCompleteTextView … )

* Adaptateurs déjà codés :

 **ArrayAdapter ( pour les tableaux ou fichiers XML )**

 CursorAdapter ( pour les bases de données )

 SimpleAdapter ( pour des items composites )

* Créer notre propre adaptateur sous-classe de BaseAdapter

L’adaptateur ArrayAdapter

Habituellement 3 paramètres :

1) Contexte d’utilisation : habituellement l’instance de Activity ( this )

2) Identifiant de la vue à utiliser : déterminé par des constantes

android.R.layout.simple\_list\_item\_1

android.R.layout.simple\_list\_item\_checked

plusieurs autres…

3) le tableau ou une référence à la liste à afficher

* On associe ensuite l’adaptateur au Spinner à l’aide de la méthode setAdapter

**Gestion des événements du Spinner :**

* On doit inscrire un Spinner à l’interface-écouteur OnItemSelectedListener, comprenant deux méthodes.
* Explication des paramètres de la méthode onItemSelected :

Cliquez ou appuyez ici pour entrer du texte.

## ( 45 min )

## 3. Utiliser des objets pour représenter chacun des comptes

Nous avions simulé un solde fixe de 500$ pour chacun des comptes mais il serait plus adéquat que chaque compte puisse avoir son propre solde; de plus, ce sera l’occasion d’intégrer une classe-modèle à notre vue.

1. Créer une classe Compte qui contiendra 2 variables d’instance : le nom du compte ( Chèque, Epargne… ) et son solde. Codez un constructeur permettant d’initialiser les variables avec des paramètres ainsi que des méthodes d’accès et de mutation.
2. Dans notre activité, on doit donc créer 3 objets Compte chacun avec un nom et un solde distinct.
3. Où utilisera-t-on ces objets ?

Cliquez ou appuyez ici pour entrer du texte.

1. On doit donc trouver une structure Collection permettant de récupérer un Compte en ne connaissant que son nom.

X Vector

X ArrayList

Je vous propose une table de hachage Hashtable

## Hashtables

* Une table de hachage ( Hashtable ) est une structure simple afin d'entreposer des paires clé-élément
* Structure simple : à l'aide de la clé, on peut retrouver l'élément correspondant ( pas de notion d'index )
* Les clés doivent être uniques

|  |  |
| --- | --- |
| CLÉ ( n’importe quel type d’objet ( String ) | ÉLÉMENT ( n’importe quel type d’objet ) |
| « 3334433 » | new Etudiant ( « 3334433 », « Lise », new String[]{“c45”, “c90”},… ) |
|  |  |
|  |  |
|  |  |
| « 1111111 » | new Etudiant ( « 1111111 », « Herménégilde », new String[]{“c43”, “c44”},… ) |
|  |  |
|  |  |

// créer un objet Hashtable

Hashtable<String, Point> ht = new Hashtable<String, Point>();

//lier les clés et les valeurs en les ajoutant dans la table de hachag ht.put ("p1", new Point(3,4));

ht.put("p2", new Point(5,7));

ht.put("p3", new Point(12,24));

// vérifier si la Hashtable est vide

System.out.println ( ht.isEmpty()); // retourne false

// vérifier si la clé existe dans la table de hachage System.out.println(ht.containsKey("p2")); // retourne true, emploie equals

// détermine si la valeur existe dans la table de hachage System.out.println(ht.containsValue(new Point (12,12))); // retourne false

// retourne le nb de paires dans la table de hachage

System.out.println ( ht.size()); retourne 3

// affiche les valeurs associés aux clés - transtypage nécessaire s'il n'y a pas de génériques

System.out.println(ht.get("p3").getY()); // retourne 24

// méthode keySet retourne un Set de toutes les clés

Set<String> ensembleCles = ht.keySet();

for ( String str:ensembleCles)

System.out.println ( str ); // for amélioré, Set met en œuvre l’interface Iterable

// méthode values retourne une Collection des valeurs de la Hashtable

Collection <Point> ensembleValeurs = ht.values();

for ( Point p : ensembleValeurs)

System.out.println ( p.getX()); //retourne les valeurs des variables x

## 4. ( 15 min ) Utiliser une expression régulière pour s’assurer que l’adresse courriel est valide

* La méthode matches permet de vérifier une String vs une expression régulière
* On veut :

«  un certain nombre de lettres / chiffres » @ «  un certain nombre de lettres / chiffres et des points »

* Utiliser les multiplicateurs …voir la « cheat sheet » sur ericlabonte.com

## 5. ( 20 min ) Affichez les messages à l’aide de boîtes de messages plutôt que dans les champs texte

* Pour des boîtes de message simples, on peut utiliser la classe AlertDialog en collaboration avec un Builder ( qui est une application de ce patron de conception )

Voir l’aide suivante : <https://developer.android.com/develop/ui/views/components/dialogs#AlertDialog>

N’oubliez pas de faire show() sur l’objet AlertDialog créé à la fin !!