M2 BI&A - 2023/2024
ICOM - Université de Lyon 2

# Module LPS: Language de Programmation Spécialisés (*Swift*)

## Travaux Pratiques 3

Visualisation d'ensemble d'objets (`UITableView`, `UICollectionView`)
et liaison de code en *Objective-C*
(illustrations basées sur la version *Xcode* 14.0.1)

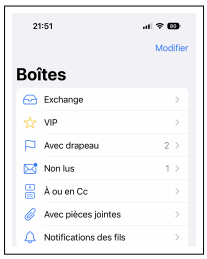
---

**Objectif:** Dans ce TP, nous allons exploiter un élément essentiel utile pour gérer l'affichage d'ensembles d'éléments graphiques à travers les objets `UITableView`. En particulier, nous verrons comment afficher des cellules avec des vues personnalisées. Enfin, nous verrons comment intégrer du code *Objective-C* dans un projet écrit en *Swift*.

---

### Concept du cours :

**Gestion des *vues des tableaux* (`UITableView`)** : sur *iOS*, l'affichage d'un tableau est un élément clé communs à de nombreuses interfaces d'applications. Cet affichage peut se présenter sous différentes formes comme sur l'image ci-contre avec un simple affichage *plain* qui comporte une série de rangées où peuvent apparaitre des icônes, labels, et entêtes et pied de page. D'autres classes comme les `UICollectionView` partagent les mêmes concepts de fonctionnement qui utilisent les concepts suivants.




**Concept de fonctionnement** : les vues de tableaux ou collections, utilisent un fonctionnement spécial conçu afin de fournir un affichage rapide même en cas de liste très longue ou incomplète. Il permet d'avoir un affichage réactif car l'interface n'est pas obligée d'attendre d'avoir récupéré tous les éléments avant de les afficher. Un autre avantage est de pouvoir faire patienter l'utilisateur en affichant les éléments actuellement disponibles. Le principe de fonctionnement est le suivant :

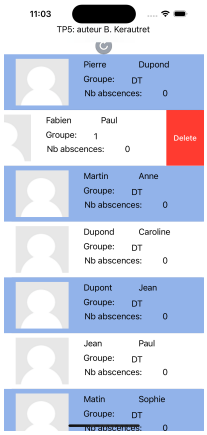
1. Un prototype de cellule (modèle graphique, représentant une rangée du tableau) doit être spécifié au contrôleur. Ce modèle peut être prédéfini par un modèle standard ou rajouté et personnalisé dans le *Storyboard*.
2. Le prototype est alors utilisé par le contrôleur afin de fabriquer les cellules et les intégrer dans la vue du tableau. Le contrôleur effectue aussi différentes optimisations pour le recyclage des cellules pour limiter l'allocation mémoire (coûteux).
3. Les méthodes permettant d'obtenir des informations sur les cellules à afficher sont appelées par le contrôleur (nombre d'éléments à afficher, contenu d'une cellule située à une certaine position).

Pour la suite, nous allons mettre en pratique ces concepts en réalisant une interface contenant un tableau dont les cellules auront été personnalisées et qui représenteront les étudiants d'une promotion (voir image (a) de la page suivante).

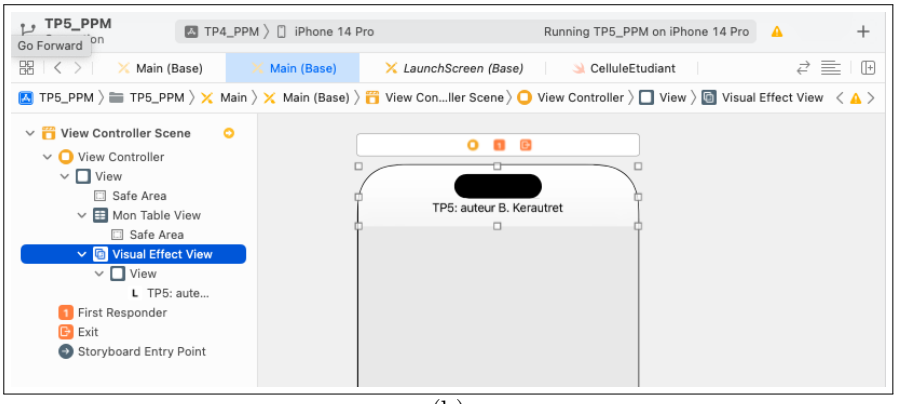
## **Partie I:** Préliminaire base de l'interface

Dans cette partie, il s'agit d'initier la base du projet en créant un nouveau projet de type **Application IOS** basée sur **Swift** utilisant un **storyboard**. Ensuite, nous créerons le modèle d'une cellule qui sera utilisée par le contrôleur de vue comme indiqué dans le concept du cours de la page précédente (point 1).

1.1 Construisez l'interface qui comportera un `UITableView`, un `UIVisualEffectView` (blur) et un `UILabel`. Pour vous aider à construire l'interface vous pouvez vous aider de l'image (b) de la figure suivante :




(a)



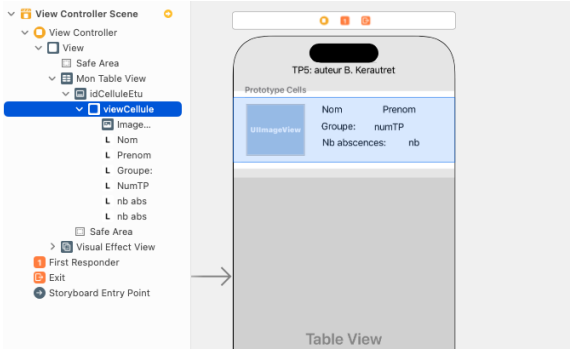

(b)

1.2 Comme nous cherchons à construire une interface qui va afficher des listes d'étudiants, il s'agit de construire une nouvelle classe `Etudiant` contenant les attributs suivants (de type `var`) :

- `nom`, `prénom`, `formation` et `groupeTP` : type `String`, initialisé lors de la construction (dans la méthode `init()`).
- `nomPhoto` : type `String`, initialisés à la définition avec la chaîne : `"photoEtudiant.png"`
- `numAbsence` : type `Int`, initialisé à 0 à la définition.

1.3 Rajoutez dans votre projet la photo `photoEtudiant.png` disponible sur moodle.

1.4 Dans le `TableViewController`, rajoutez une cellule (`UITableViewCell`), qui constituera un prototype que l'on adaptera pour obtenir la visualisation suivante (image (a)) :




(a)

## **Partie II:** Création de la classe des cellules et association au prototype

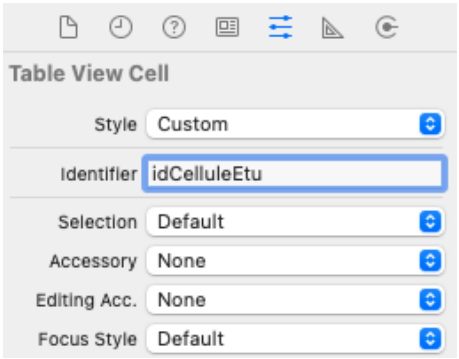
Maintenant que nous avons mis en place les éléments graphiques de l'interface, nous allons les associer dans le code de façon à pouvoir générer du contenu dynamique dans l'interface. Une première étape sera de créer une **classe associée au prototype de cellule** précédant de façon à ce que la contrôleur puisse générer des cellules en respectant votre prototype (point 2. de la description des concepts du cours).

2.1 Créez une nouvelle classe `CelluleEtudiant` en utilisant le modèle : *Cocoa Touch Class* et héritant de `UITableViewCell`. Associez chaque élément graphique du prototype de la cellule à un nouvel attribut dans la classe `CelluleEtudiant` (pour les 4 UILabels associés aux informations étudiants et pour l'`UIImageView`).
**Attention :** pour que l'association soit possible, vous devez absolument changer la classe par défaut `UITableViewCell` en `CelluleEtudiant`.

Toujours dans la classe `CelluleEtudiant`, il s'agit maintenant de rajouter une fonction de façon à ce que la cellule soit capable de se mettre à jour lorsqu'elle doit représenter un nouvel étudiant. En effet comme annoncé dans le point 2 du concept du cours, le contrôleur devra recycler des objets cellules pour les utiliser avec différents étudiants.

2.2 Dans la classe `CelluleEtudiant`, rajoutez donc `miseAjourCellule` prenant comme argument un objet de type `Etudiant` et qui mettra à jour tous les éléments graphiques de la cellule. Pour l'image de l'étudiant, vous pourrez utiliser l'attribut `image` de l'`UIImageView` et utiliser le constructeur d'une image qui utilise simplement le nom de l'image.

2.3 Comme vous pouvez le voir, un avertissement apparaît pour vous alerter que le prototype de votre cellule doit avoir un identifiant. Dans le *Storyboard*, en sélectionnant votre le prototype de votre cellule, saisissez un identifiant (par exemple : *idCelluleEtu* dans l'onglet *attribut*).




Maintenant le message d'avertissement a du disparaître et il reste à configurer le contrôleur de façon à ce qu'il soit capable de générer les cellules et de les afficher.

## **Partie III:** Génération des cellules dans le contrôleur

Comme évoqué dans l'introduction de ce TP, le principe pour afficher des cellules repose sur un contrôleur qui est chargé de gérer les cellules. Plus précisément, le contrôleur devra être conforme aux deux protocoles suivants :

- Protocole `UITableViewDataSource` : mentionne des méthodes utiles pour gérer un flux de données.
- Protocole `UITableViewDelegate` : préconise les méthodes pour la visualisation de cellules (par exemple les méthodes pour sélectionner, effacer, etc).

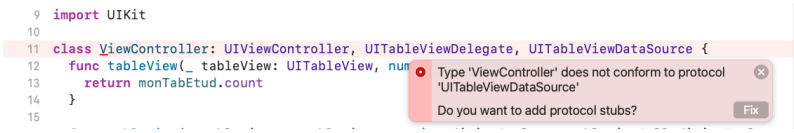
3.1 Déclarez la classe de votre contrôleur `ViewControler` comme étant conforme aux deux protocoles `UITableViewDelegate` et `UITableViewDataSource`. Une erreur apparaît mais elle sera corrigée dans les questions suivantes.

3.2 Rajoutez un nouvel attribut (`monTabEtudiant`) défini comme un tableau d'objet `Etudiant` et créez une fonction `creationTableauEtudiant` qui ajoutera au moins cinq étudiants dans le tableau. Appelez cette fonction dans la fonction `viewDidLoad` du `ViewController`.

3.3 Comme le contrôleur va avoir besoin d'accéder à une instance d'un `UITableView` pour afficher ses données, il faut que vous créiez une référence du `UITableView` du *Storyboard* dans le code du `ViewControler`. En utilisant l'assistant ou manuellement, rajoutez cette référence dans une nouvelle variable `monTableView`.

3.4 Afin d'effectuer un décalage pour l'affichage des cellules, modifier l'attribut `contentInset` de l'objet précédant `monTableView`.

3.5 Pour l'instant le contrôleur n'est toujours pas conforme au protocole `UITableViewDataSource`. Pour le rendre conforme, il faut implémenter obligatoirement deux méthodes particulières. Consulter la documentation pour retrouver le nom et l'utilité des deux méthodes (méthodes déclarées avec *Required*). Si votre version d'*Xcode* vous le propose, vous pouvez acceptez la suggestion qui doit correspondre aux deux fonctions nécessaires.




3.6 Implémentez les deux méthodes requises de la question précédente.

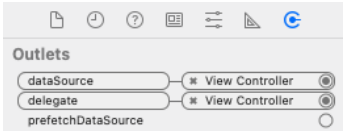
**Indication :** pour implémenter la méthode qui renvoie la nouvelle cellule, il faudra récupérer et renvoyer la cellule que le contrôleur créé directement :

```
1       let aCell = monTableView.dequeueReusableCell(withIdentifier: ) as! CelluleEtudiant
```

Vous compléterez cet appel de fonction en utilisant le code de la question 2.3.

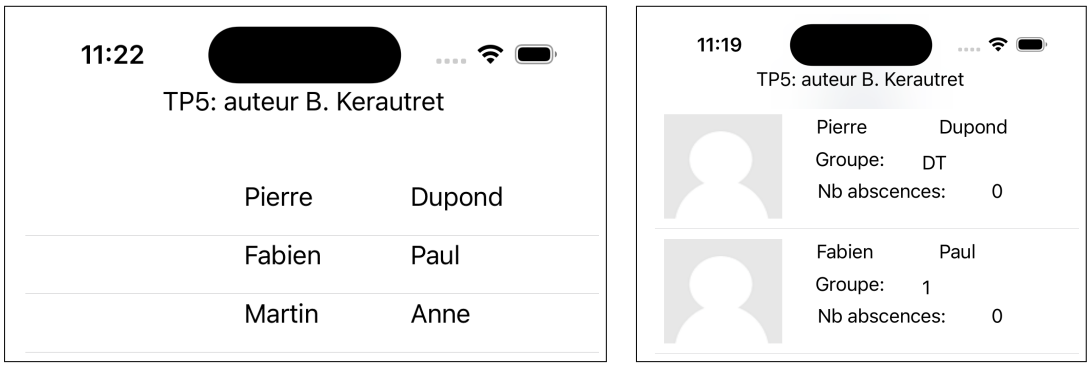
3.7 Comme vous avez du le remarquer le programme n'affiche toujours pas la liste des étudiants. Cela est normal puisqu'il reste encore à déclarer la classe `ViewController` comme étant la classe déléguée du `UITableView`. Il existe deux possibilités pour effectuer cette association :

— Dans le *storyboard*, en sélectionnant l'objet `UITableView`, dans l'onglet *Connection Inspector*, section *Outlet*, il est possible de raccorder l'élément *delegate* et *dataSource* au contrôleur principal.




— L'autre possibilité est d'affecter dans le code les attributs `delegate` et `dataSource` de l'objet `monTableView` à la classe qui la contient (en utilisant `self`).

Après avoir effectué ces liaisons, vous devez obtenir une visualisation comparable à l'image (a) suivante :




(a) (b)

Comme vous l'avez remarqué les cellules présentent des labels et des photos ne sont pas mises à jour et dont les tailles ne correspondent pas à la taille modélisée dans le *Storyboard* du prototype de la cellule.

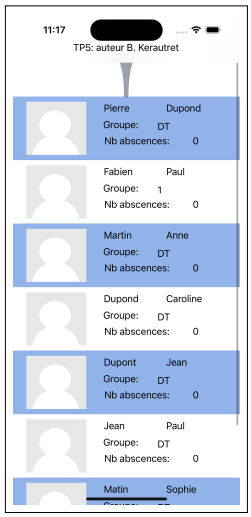
3.8 Réglez le problème de la taille de la cellule en explorant dans le *Storyboard* les propriétés de l'objet représentant le conteneur du tableau.

3.9 Modifiez la fonction `tableView:cellForRowAtIndexPath:` de façon à ce que les labels soient mis à jour avec les informations des étudiants et de façon à personnaliser les cellules en alternant la couleur de fond (bleue clair et gris) comme sur la figure de la page 1.

3.10 On souhaite aussi maintenant pouvoir supprimer une cellule. Il suffit pour cela d'implémenter la méthode `tableView:commitEditingStyle:` du protocole `UITableViewDataSource`. Rajoutez cette méthode et mettez à jour le tableau `monTableView` puis rechargez l'affichage du tableau (en utilisant la fonction `reloadData` sur l'objet représentant votre `UITableView`).

## **Partie IV:** Utilisation de code *Objective-C* existant

Toujours dans la contexte de l'application basée sur le tableau, nous allons montrer comment utiliser du code d'objective-C à l'intérieur d'une application écrite en *Swift*. Il s'agira de rajouter du code permettant d'obtenir l'icône de rechargement d'un tableau tel que celui illustré ci dessous :




4.1 Récupérez les deux fichiers écrits en *Objective-C* disponibles sur *moodle*. En faisant un *drag and drop*, rajoutez ces fichiers dans votre projet. Au moment de l'importation *Xcode* doit vous

demander si vous souhaitez qu'il rajoute un fichier entête pour faire le pont avec *Objective-C* (un *Objective-C bridging header*) : **répondez Oui**.

4.2 Dans le fichier `TP4_PPM-Bridging-Header.h`, rajoutez les lignes suivantes :

```
#ifndef ef ODRefreshControl_Bridging_Header_h
#define ODRefreshControl_Bridging_Header_h

#import "ODRefreshControl.m"

#endif
```

4.3 Dans votre classe `ViewController`, rajoutez une variable de type `ODRefreshControl` et **l'initialiser dans la fonction** `viewDidLoad()` du controleur principal (explorez les différents constructeurs de l'objet `ODRefreshControl`).

4.4 Vous devriez voir le nouvel élément apparaître lors du déplacement des cellules. Modifier éventuellement la géométrie du nouvel élément (en modifiant son attribut frame.origin.y) de façon à ce qu'il apparaisse comme sur l'image précédente.

4.5 Enfin, associez une action au nouvel élément. Pour cela, il faut utiliser la méthode du nouvel élément (toujours dans la méthode `viewDidLoad()` : `addTarget`. Pour préciser une action en paramètre vous devez mettre le mot clé `#selector` (exemple : `#selector(VotreMethodeCible)`) et dans la définition, vous devrez rajouter `@objc` qui permet de rendre accessible votre fonction pour du code écrit en *Objective-C*.

4.6 Enfin vous pouvez arrêter l'animation de chargement avec la méthode `endRefreshing()`.