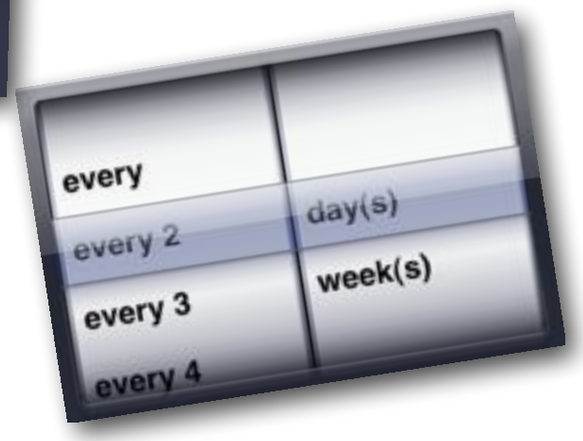
- Utilisation d'un composant UIPickerView
  - Equivalent des menus déroulants présentant un choix multiple
  - Composant dont la largeur occupe tout l'écran de l'iPhone
  - Version spécifique pour le choix de dates et heures



- Utilisation d'un composant UIPickerView
  - Le contrôleur est responsable du rendu à l'écran (apparence, animations, etc.), mais il ne sait rien des données à afficher, ni quoi faire lorsque l'utilisateur sélectionne une entrée
  - Les composants comme le Picker n'attendent pas qu'on leur dise ce qu'il faut afficher
  - UIPickerView possède deux propriétés qui pointent sur des objets
    - dataSource

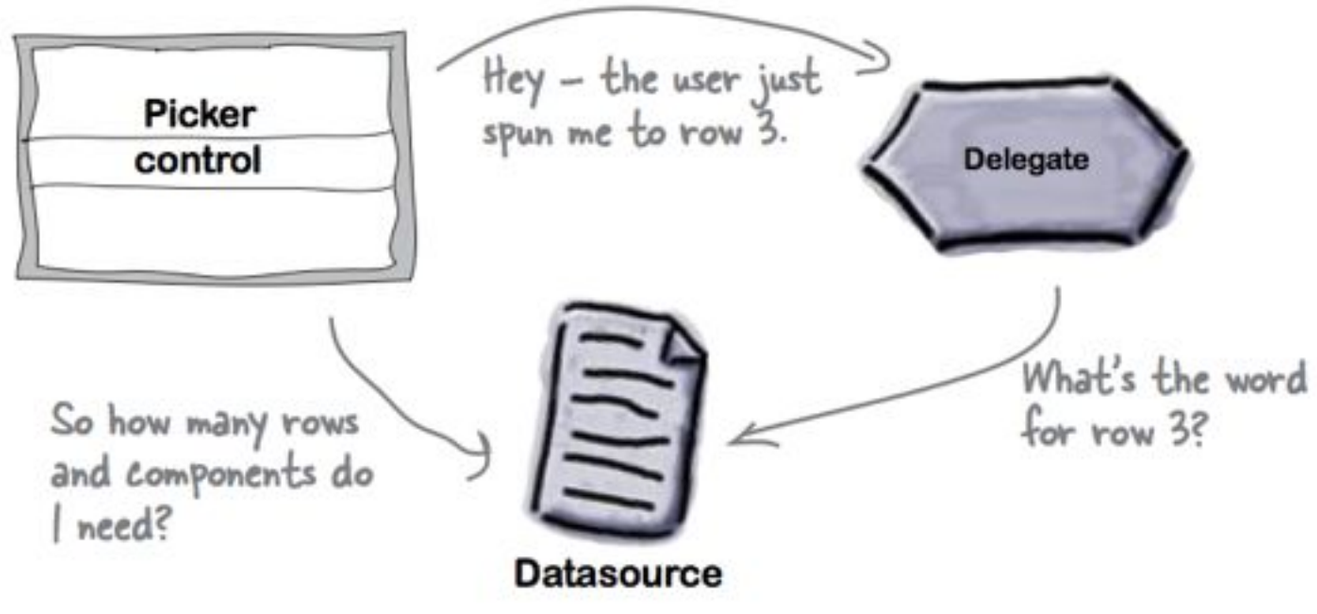
```
@property(nonatomic, assign) id<UIPickerViewDataSource> dataSource
```

      - Le contrôle demande à l'objet datasource ce dont il a besoin
      - La "datasource" doit fournir les informations dans un format que le contrôle attend. Elle fournit le nombre de composants (colonnes) et le nombre total de lignes

- Utilisation d'un composant UIPickerView
  - delegate

    ```
    @property(nonatomic, assign) id<UIPickerViewDelegate> delegate
    ```

    - Le délégué doit implémenter les méthodes qui retournent le rectangle dans lequel sont dessinées les lignes
    - Il fournit également le contenu de chaque ligne d'un composant sous la forme de chaîne de caractère ou de vue
  - Il répond aux sélections et désélections de l'utilisateur



- Syntaxe Objective C - Protocoles
  - Déclaration d'une variable `id<UIPickerViewDelegate> delegate`
    - L'objet vers lequel pointe la variable d'instance delegate doit implémenter le protocole `UIPickerViewDelegate`
    - Un protocole est l'équivalent d'une interface en Java
      - déclare des méthodes qui peuvent être implémentées par n'importe quelle classe
      - certaines méthodes peuvent être optionnelles
      - Une classe qui implémente un protocole le précise dans la déclaration de l'interface
      - Une même classe peut implémenter plusieurs protocoles

```
@interface InstatwitViewController : UIViewController<UIPickerViewDataSource, UIPickerViewDelegate>
```

Syntaxe Objective C - Protocoles

Définition d'un protocole

```
@protocol MyProtocol

- (void)requiredMethod;

@optional
- (void)anOptionalMethod;
- (void)anotherOptionalMethod;

@required
- (void)anotherRequiredMethod;
@end
```

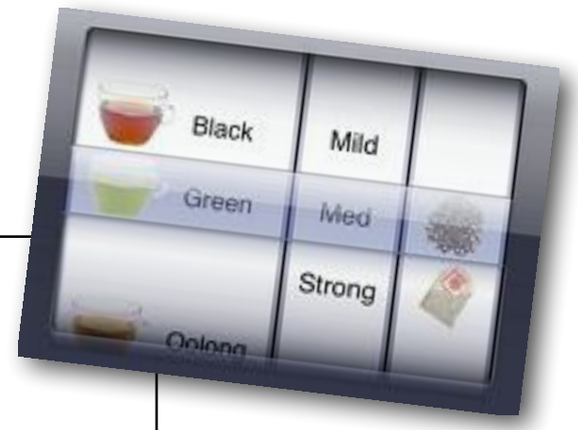
Possibilité de tester si un objet implémente un protocole

```
if ( ! [receiver conformsToProtocol:@protocol(MyXMLSupport)]  )
```

Utilisation d'un composant UIPickerView

Définitions des protocoles utilisés par UIPickerView

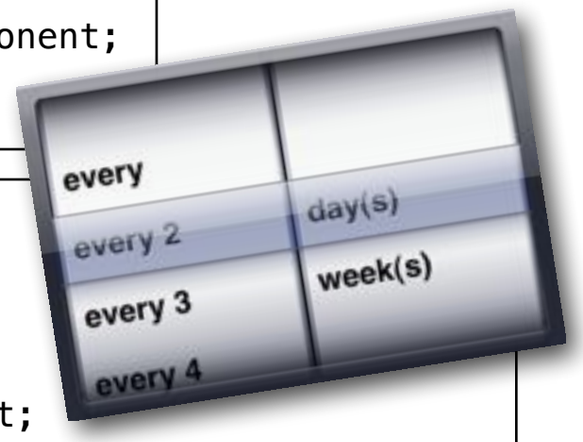
UIPickerView.h




```
@protocol UIPickerViewDataSource<NSObject>
@required

// returns the number of 'columns' to display.
- (NSInteger)numberOfComponentsInPickerView:(UIPickerView *)pickerView;

// returns the # of rows in each component..
- (NSInteger)pickerView:(UIPickerView *)pickerView numberOfRowsInComponent:(NSInteger)component;
@end
```




```
@protocol UIPickerViewDelegate<NSObject>
@optional

// returns width of column and height of row for each component.
- (CGFloat)pickerView:(UIPickerView *)pickerView widthForComponent:(NSInteger)component;
- (CGFloat)pickerView:(UIPickerView *)pickerView rowHeightForComponent:(NSInteger)component;

// these methods return either a plain UIString, or a view (e.g UILabel) to display the row for the component.
// for the view versions, we cache any hidden and thus unused views and pass them back for reuse.
// If you return back a different object, the old one will be released. the view will be centered in the row rect
- (NSString *)pickerView:(UIPickerView *)pickerView titleForRow:(NSInteger)row forComponent:(NSInteger)component;
- (UIView *)pickerView:(UIPickerView *)pickerView viewForRow:(NSInteger)row forComponent:(NSInteger)component
reusingView:(UIView *)view;

- (void)pickerView:(UIPickerView *)pickerView didSelectRow:(NSInteger)row inComponent:(NSInteger)component;

@end
```

- Utilisation d'un composant UIPickerView
  - Créer une application qui comporte un composant UIPickerView avec 2 colonnes utilisé pour produire rapidement un message à twitter
    - La première colonne permet de décrire l'activité
      - dors, mange, suis en cours, galère, cours, poireaute
    - La deuxième colonne permet d'ajouter l'état d'esprit
      - ;), :), :(, :O, 8), :o, :D, mdr, lol
  - L'appui sur le bouton tweet it doit provoquer la mise en forme d'une phrase qui sera affichée sur la console dans un premier temps
  - Dans un deuxième temps ajouter un champ de texte dans lequel saisir le message

- Gestion du clavier
  - Apparition automatique dès qu'un champ de texte a le "focus"
  - La fenêtre de l'application redirige les événements vers le "firstResponder"
    - lorsqu'un champ texte devient "first responder" il demande au système de faire apparaitre le clavier
    - pour que le clavier disparaisse il faut que le composant texte renonce à son statut de "first responder" par un appel à la méthode resignFirstResponder

- Gestion du clavier
  - Difficultés
    - Quand on a plusieurs champs de texte il faut savoir qui est le “first responder”

- Gestion du clavier
  - Solutions
    - Méthode `endEditing` de la classe UIView qui recherche dans la vue et dans ses sous-vues un champ de texte qui est le “first responder” et qui lui demande de relacher son statut de “first responder”
      - Le contrôleur de la vue possède une référence vers sa vue (self.view)
    - Les événements utilisés pour faire disparaître le clavier sont :
      - un appui sur le fond (en dehors des composants) : installer un "tap gesture recognizer" sur le fond)
      - un appui sur le bouton done du clavier (évènement DidEndOnExit)

- Ajout de l'envoi d'un tweet
  - Ajouter la fonctionnalité d'envoi d'un tweet prérempli
    - Ajouter le framework Social
    - Passer la main au contrôleur permettant la saisie et l'envoi d'un message
      - Utiliser SLComposeViewController
    - Pour effectuer une transition de la vue principale vers celle de l'envoi du tweet il faut que le controleur de vue actif "présente" le controleur de vue qui va prendre la suite
      - `presentViewController:animated:completion:`
    - Lorsque l'opération envoi d'un tweet est finie (annulation ou envoi) afficher un message (UIAlertView) pour indiquer si l'utilisateur a annulé l'envoi ou non

- Blocks
  - Le contrôleur de vue Twitter possède une méthode qui permet de fournir un bloc de code qui sera exécuté lorsque la composition sera terminée
    - ce contrôleur ne possède pas d'objet delegate et s'appuie sur un ajout du langage - les blocks - pour prévenir le contrôleur
  - la méthode setCompletionHandler possède un argument de type block
    - Ce block est un morceau de code qui sera exécuté plus tard
    - Lors de son exécution, ce block recevra un argument afin de savoir si l'opération a été annulée ou si le tweet a été envoyé

Blocks

- Le concept de block revient à stocker une référence vers un morceau de code dans une variable
- Ce code a accès à tous les éléments du contexte dans lequel il est déclaré (variables locales à la méthode à l'intérieur de laquelle il se trouve, variables d'instances, ...)
- Il peut être stocké dans une variable, passé en argument à une méthode, ...
- Les blocks sont l'équivalent des clotures (closures) ou des expressions lambda en Java (JDK 8), en C# (=>), et dans de nombreux autres langages (dont Swift)

Blocks - syntaxe

- Déclaration d'un block
  - `float (^operation)(float,float);`
    - variable dont le nom est operation
    - contient un block qui attend 2 arguments de type float
    - retourne un float
- Création d'un block

```
operation = ^(float a, float b){
        return a-b;
    };
```

- Utilisation

```
float res = operation(_sliderA.value,_sliderB.value);
```

Blocks - utilisation

- Les blocks fournissent un mécanisme de callback puissant car ils ont accès à tout le contexte disponible au moment de leur création
  - cela évite d'avoir à rendre accessible certaines données depuis plusieurs méthodes d'un même objet en passant par des variables d'instances ou en transmettant un objet de contexte
- Les frameworks Cocoa migrent vers une utilisation importante des blocks (solution exclusive sur de nombreux nouveaux frameworks)
- Usage
  - Completion handlers, Notification Handlers, Error Handlers
  - Enumeration (opération sur tous les éléments d'une collection, filtrage, recherche,tri)
  - Animation des vues et transitions
  - Programmation concurrente

Blocks - exemples

```
[UIView animateWithDuration:0.2 animations:^{
                                    view.alpha = 0.0;
                                    }
                                completion:^(BOOL finished){
                                    [view removeFromSuperview];
                                    }
                                ];
```

```
NSArray *array = @[@"A", @"B", @"C", @"A", @"B", @"Z", @"G", @"are", @"Q"];
NSSet *filterSet = [NSSet setWithObjects: @"A", @"Z", @"Q", nil];

BOOL (^test)(id obj, NSUInteger idx, BOOL *stop);

test = ^(id obj, NSUInteger idx, BOOL *stop) {
    if (idx < 5) {
        if ([filterSet containsObject: obj]) {
            return YES;
        }
    }
    return NO;
};

NSIndexSet *indexes = [array indexesOfObjectsPassingTest:test];
```