Cours n°8 du 1er novembre 2012

3MCIn1 2012-2013

Au programme...

Suite du cours n°7 sur les Views

- **Autorotation**
- **Protocols**
  - delegation
  - data source
- **Gestion des événements**
  - UIGestureRecognizers

Autorotation

Gérer la rotation de l'appareil dans une application

Pour permettre la rotation de son app'

Implémenter cette méthode dans le Controller

```
- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)toInterfaceOrientation
{
    return YES;
}
```

supporte toutes les directions

```
{
    return UIInterfaceOrientationIsPortrait(toInterfaceOrientation);
}
```

uniquement mode portrait (haut et bas)

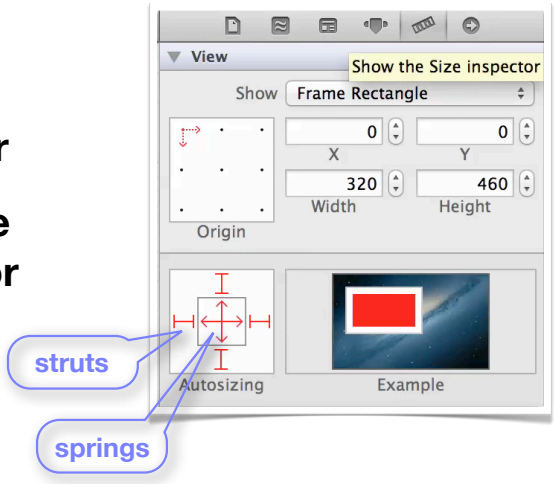
```
{
    return (toInterfaceOrientation != UIInterfaceOrientationPortraitUpsideDown);
}
```

tout sauf le mode portrait de bas en haut

Struts & springs

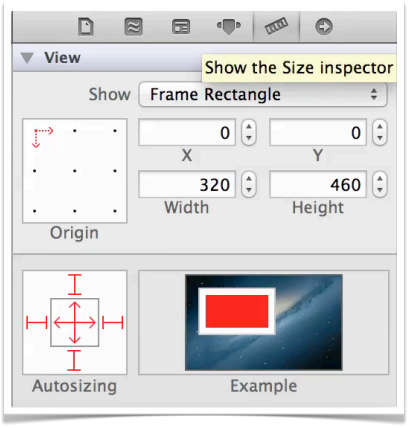
Pour décider du comportement de sa View par rapport à la superView

- **Lorsque l'appareil pivote, les frames des Views vont s'ajuster**
- **Les règles d'ajustement vont se fixer dans Xode > Size inspector via les struts & springs**

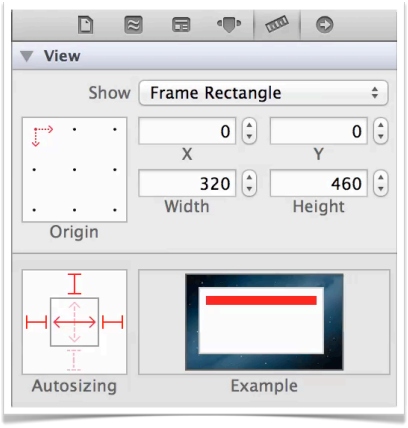


Struts & springs

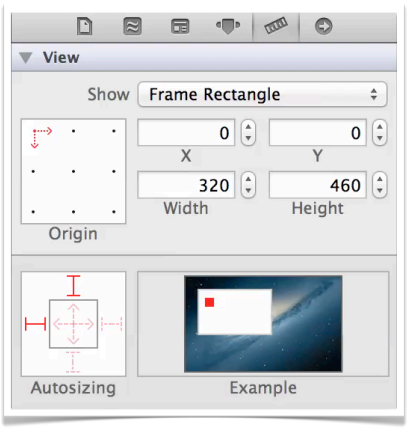
Trois exemples...



Etirement dans toutes les directions: la View «colle» aux dimensions de la superView



Etirement dans la largeur: la View «colle» au haut de la superView mais ne s'étire pas en hauteur

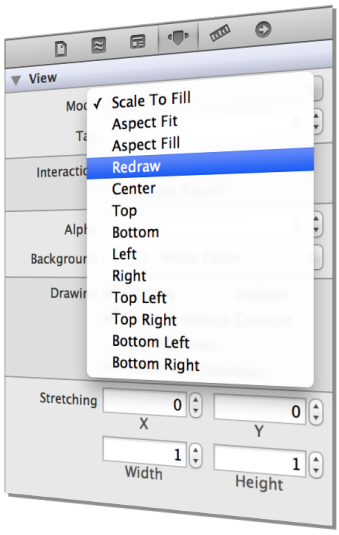


Aucun étirement: la View conserve ses dimensions originales et «colle» au coin supérieur gauche de la superView

Redessiner l'interface

Lorsque les bounds changent (par exemple lors d'une rotation)

- **Par défaut, l'interface n'est pas redessinée dans ces circonstances**
  - l'affichage est étiré pour correspondre à ses nouvelles dimensions (scale to fill)
- **On peut forcer un redraw dans Xcode > Attributes inspector**

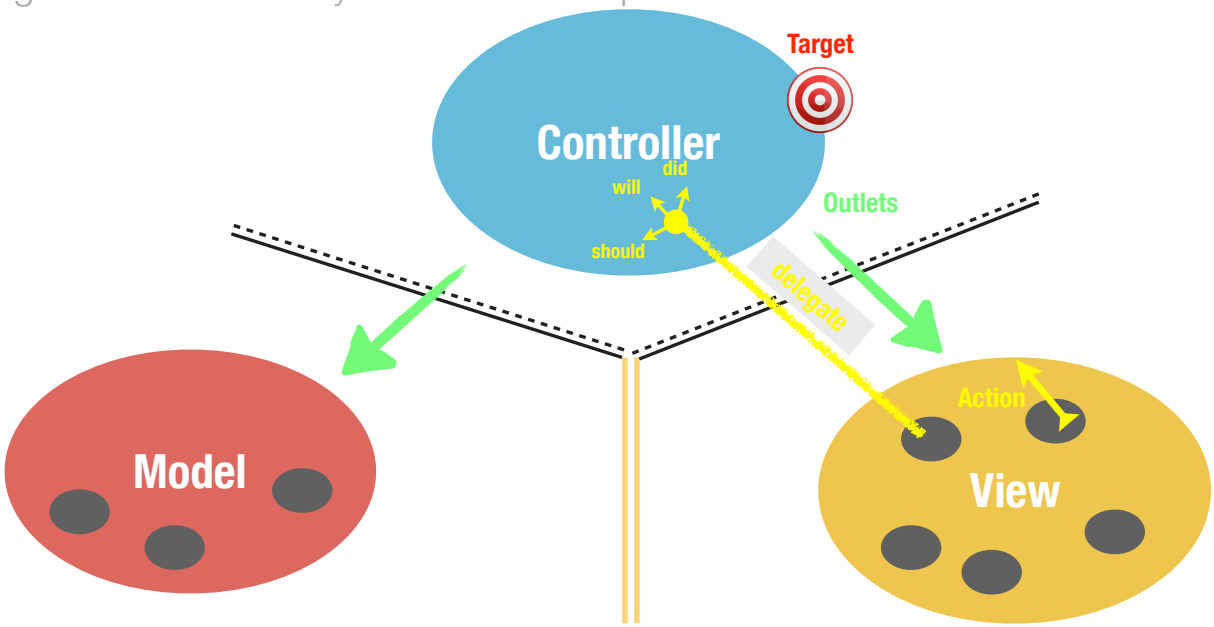


Protocols

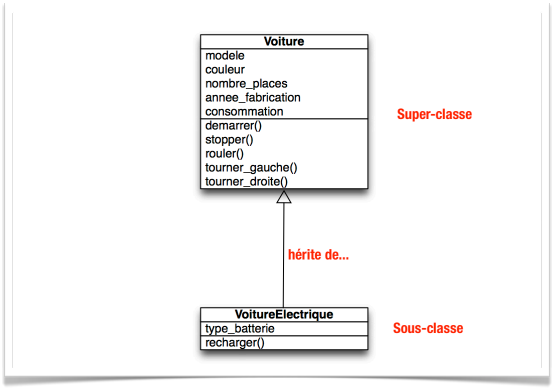
Délégation, data source

Rappel: MVC

Délégation: un autre moyen de communiquer entre la View et le Controller



Comment un objet peut-il utiliser des méthodes d'une autre classe que la sienne?



Rappel: l'héritage

L’héritage

Comme un moyen de partager des méthodes

- **est un moyen assez contraignant...**
- **La classe de l’objet doit être connue (pas de possibilité d’utiliser le type id)**
- **L’objet de la sous-classe doit être capable d’implémenter l’ensemble des méthodes de la super-classe**

Pour cette raison, on utilise les protocoles

@protocol

Syntaxiquement

cela ressemble à la déclaration d'une classe

- **classe**

```
@interface FaceView : UIView

@end
```

- **protocole**

```
@protocol monProtocole

@end
```

- **et se déclare dans le fichier .h de la classe qui veut faire utiliser ce protocole par d'autres ou dans un fichier .h autonome créé uniquement pour ce protocole**
- **différence avec @interface: l'implémentation se passe ailleurs (dans une autre classe ou un autre objet)**

@protocol: un exemple

Distinguer les éléments obligatoires et optionnels

ceux qui implémentent **monProtocole** doivent également implémenter **Autre** et **NSObject**

```
@protocol monProtocole <Autre, NSObject>

- (void)faisQuelqueChose;          // implémentation obligatoire

@optional

- (int)chercheQuelqueChose;
- (void)faisQuelqueChoseDOptionnelAvecUnArgument:(NSString *)monArgument;   /* implémentation optionnelle */

@required

- (NSArray *)chercheDeMultiplesElements:(int)nombreElements;   // implémentation obligatoire

@property (nonatomic, strong) NSString *proprieteDeMonProtocole;

@end
```

Utilisation d'un protocole par une classe

Une classe va annoncer dans son @interface si elle implémente un protocole

```
#import <monProtocole.h>

@interface HappinessViewController : UIViewController <monProtocole>

@end
```

HappinessViewController doit alors implémenter la totalité des méthodes non-optionnelles

On peut alors créer des variables de type id qui requièrent un protocole...

```
id <monProtocole> monObjet = [[HappinessViewController] alloc] init];
```

C'est possible car HappinessViewController a annoncé qu'elle implémente monProtocole

... ou des arguments de méthode qui requièrent un protocole...

```
- (void) donneMoiUnObjetQuiRequiertMonProtocole:(id <monProtocole>)unObjetQuiRequiertMonProtocole;
```

... ou des propriétés qui requièrent un protocole...

```
@property (nonatomic, weak) id <monProtocole> maProprieteQuiRequierMonProtocole;
```


delegate & dataSource

sont les deux protocoles les plus régulièrement utilisés

- **ils sont déclarés comme une propriété le plus souvent weak dans la classe qui va déléguer la gestion d'un comportement ou la source de ses données**

```
@property (nonatomic, weak) IBOutlet id <FaceViewDataSource> dataSource;
```

Dans le fichier UIScrollView.h

```
   @protocol UIScrollViewDelegate
@optional
- (UIView *)viewForZoomingInScrollView:(UIScrollView *)sender;
- (void)scrollViewDidEndDragging:(UIScrollView *)sender willDecelerate:(BOOL)decelerate;
...
@end

@interface UIScrollView : UIView
@property(nonatomic,assign) id <UIScrollViewDelegate> delegate;
@end
```

Dans le fichier MyViewController.h

```
@interface MyViewController : UIViewController <UIScrollViewDelegate>
@property (nonatomic, weak) IBOutlet UIScrollView *scrollView;
@end

@implementation MyViewController
- (void)setScrollView:(UIScrollView *)scrollView
{
    _scrollView = scrollView;
    self.scrollView.delegate = self;
}
- (UIView *)viewForZoomingInScrollView:(UIScrollView *)sender { return ... };
@end
```


Gestion des événements

UIGestureRecognizer

Après le dessin dans une View...

Comment gérer les événements qui s'y produisent?

- **On peut obtenir l'information brute sur les positions et les déplacements des doigts sur l'écran (en terme de suites de coordonnées de points) ou**
- **On utilise des UIGestureRecognizer, «patterns» de mouvements prédéterminés que le système va chercher à reconnaître**
  - **UIGestureRecognizer = super-classe abstraite**
  - **Sous-classes concrètes:**
    - **UITapGestureRecognizer, UIPinchGestureRecognizer, UIRotationGestureRecognizer, UISwipeGestureRecognizer, UIPanGestureRecognizer, UILongPressGestureRecognizer**

Deux étapes

Pour utiliser des UIGestureRecognizer

1. **Attacher un UIGestureRecognizer à une View qui va constamment observer celle-ci pour chercher à reconnaître un mouvement bien précis**
2. **Fournir l'implémentation d'une méthode qui va prendre en charge ce mouvement**

(1) est en général effectué par le Controller

(2) est souvent fourni par la View elle-même

Ajouter un UIGestureRecognizer à une View

à partir du Controller

une sous-classe de UIGestureRecognizer

```
- (void)setPannableView:(UIView *)pannableView
{
    _pannableView = pannableView;
    UIPanGestureRecognizer *pangr =
    [[UIPanGestureRecognizer alloc] initWithTarget:pannableView action:@selector(pan:)];
    [pannableView addGestureRecognizer:pangr];
}
```

le Target, c'est celui qui va s'occuper de prendre en charge la Gesture une fois qu'elle sera commencée.

Ici: la View elle-même

ici, c'est la méthode qui sera appelée par le Controller lorsqu la Gesture sera reconnue

Implémenter la cible d'un gesture recognizer

Chaque classe concrète fourni quelques méthodes pour nous aider

- **Par exemple, UIPanGestureRecognizer fournit 3 méthodes:**

```
- (CGPoint)translationInView:(UIView *)aView;
- (CGPoint)velocityInView:(UIView *)aView;
- (void)setTranslation:(CGPoint)translation inView:(UIView *)aView;
```

Implémenter la cible d'un gesture recognizer

Une propriété très importante de la classe UIGestureRecognizer

```
@property (readonly) UIGestureRecognizerState State;
```

- elle commence dans l'état Possible jusqu'à ce qu'elle soit reconnue;
- elle devient alors Recognized (mouvements discrets comme un Tap) ou Began (mouvements continus comme un Pan)
- dans ce dernier cas, elle devient alors Changed, puis Ended.

Implémenter la cible d'un gesture recognizer

Que deviendrait la méthode pan: ?

on agit uniquement quand l'état est **Changed** ou **Ended**

```
- (void)pan:(UIPanGestureRecognizer *)recognizer
{
    if (recognizer.state == UIGestureRecognizerStateChanged ||
        recognizer.state == UIGestureRecognizerStateEnded) {

        CGPoint translation = [recognizer translationInView:self];

        // exemple d'un graphique qui aurait une origine que l'on déplace avec un Pan
        self.origin = CGPointMake(self.origin.x + translation.x, self.origin.y +
                                  translation.y);

        [recognizer setTranslation:CGPointZero inView:self];
    }
}
```

On appelle la méthode qui fournit la translation et retourne les coordonnées du nouveau point

On remet la distance cumulative à zéro

Un autre exemple de gesture recognizer

Que l'on utilisera dans la démo: UIPinchGestureRecognizer

```
UIPinchGestureRecognizer

@property CGFloat scale;
@property (readonly) CGFloat velocity;
```