POLITÉCNICA

ETSIT UPM

dit UPM

# Desarrollo de Apps para iOS
# Animaciones

IWEB 2015-2016
Santiago Pavón

ver: 2015.11.19

# Tipos de Animaciones

- Animar cambios en las propiedades de una view:
  - Cambia el valor de frame, transform, alpha, …
- Animar cambios en la jerarquía de views (Transiciones):
  - Cambian las subviews.
- Animar las transiciones entre View Controllers.
  - Al mostrar un nuevo VC, o un Alert, …
- Core Animation.
  - De bajo nivel y muy potente.
- Dynamic Animation:
  - Basado en propiedades físicas: gravedad, colisiones, rozamiento, densidad, bordes, …

# `UIView`: Animaciones y Transiciones

# Animaciones

- UIView soporta animaciones al cambiar el valor de algunas propiedades:
  - **frame**, **bounds**, **center**, **transform**, **alpha**, **backgroundColor**, **contentStretch**

- La animación se define usando un método de clase de UIView y closures.
  - El método de clase tiene parámetros para ajustar la animación:
    - retrasos, duración, curva de velocidad, ...
  - El bloque contiene el código que cambia el valor de las propiedades de la UIView.
  - Puede existir una closure ***Completion*** que se ejecuta al terminar la animación.

- Aunque la animación no haya terminado, el cambio en los valores de las propiedades es instantáneo.
  - Nota: si se modifican las restricciones de autolayout durante una animación, es necesario llamar al método **layoutIfNeeded** en el bloque de la animación para calcular inmediatamente los nuevos tamaños y posiciones.

```
class func animateWithDuration(_ duration: NSTimeInterval,
                    animations animations: () -> Void)

class func animateWithDuration(_ duration: NSTimeInterval,
                    animations animations: () -> Void,
                    completion completion: ((Bool) -> Void)?)

class func animateWithDuration(_ duration: NSTimeInterval,
                         delay delay: NSTimeInterval,
                       options options: UIViewAnimationOptions,
                    animations animations: () -> Void,
                    completion completion: ((Bool) -> Void)?)

class func animateWithDuration(_ duration: NSTimeInterval,
                         delay delay: NSTimeInterval,
          usingSpringWithDamping dampingRatio: CGFloat,
           initialSpringVelocity velocity: CGFloat,
                       options options: UIViewAnimationOptions,
                    animations animations: () -> Void,
                    completion completion: ((Bool) -> Void)?)
```

# Ejemplo

```
@IBOutlet weak var label: UILabel!


var pos: CGFloat = 100

@IBAction func animate() {

    pos = 300 - pos

    let p = CGPointMake(pos,pos)

    UIView.animateWithDuration(1,
            delay: 0,
            usingSpringWithDamping: 0.6,
            initialSpringVelocity: 10,
            options: UIViewAnimationOptions.BeginFromCurrentState,
            animations: {self.label.center = p},
            completion: nil)
}
```

# Transiciones

- También se pueden animar los cambios en la jerarquía de views, y los cambios de visibilidad.

```
class func transitionFromView(_ fromView: UIView,
                        toView toView: UIView,
                      duration duration: NSTimeInterval,
                       options options: UIViewAnimationOptions,
                    completion completion: ((Bool) -> Void)?)


class func transitionWithView(_ view: UIView,
                      duration duration: NSTimeInterval,
                       options options: UIViewAnimationOptions,
                    animations animations: () -> Void,
                    completion completion: ((Bool) -> Void)?)
```

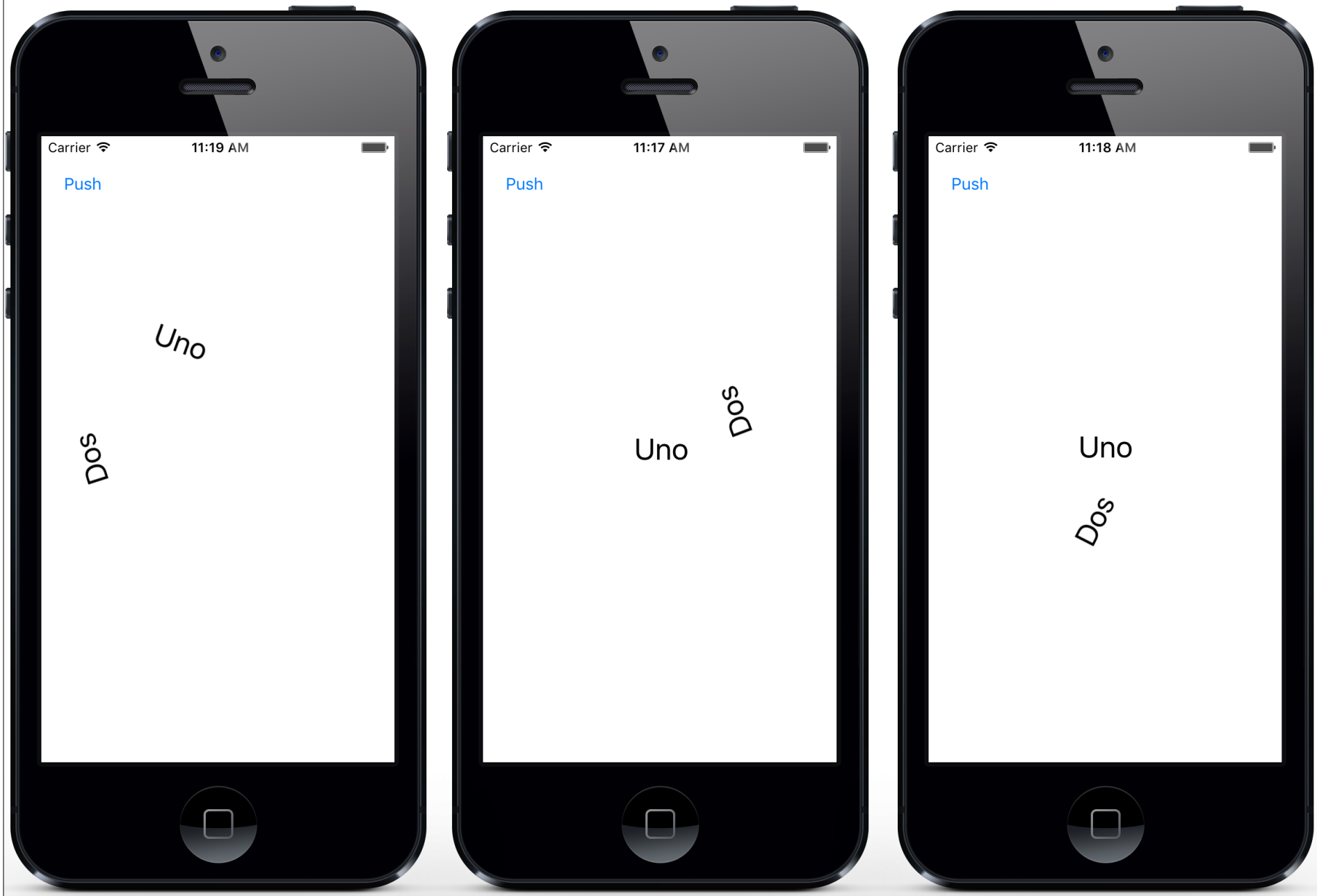
# UIViewAnimationOptions

- `.LayoutSubviews`
  Lay out subviews at commit time so that they are animated along with their parent.
- `.AllowUserInteraction`
  Allow the user to interact with views while they are being animated.
- `.BeginFromCurrentState`
  Start the animation from the current setting associated with an already in-flight animation.
- `.Repeat`
  Repeat the animation indefinitely.
- `.Autoreverse`
  Run the animation backwards and forwards.
- `.OverrideInheritedDuration`
  Force the animation to use the original duration value specified when the animation was submitted.
- `.OverrideInheritedCurve`
  Force the animation to use the original curve value specified when the animation was submitted.
- `.AllowAnimatedContent`
  Animate the views by changing the property values dynamically and redrawing the view.
- `.ShowHideTransitionViews`
  This key causes views to be hidden or shown (instead of removed or added) when performing a transition.
- `.CurveEaseInOut`
  An ease-in ease-out curve causes the animation to begin slowly, accelerate and then slow again.
- `.CurveEaseIn`
  An ease-in curve causes the animation to begin slowly, and then speed up as it progresses.
- `.CurveEaseOut`
  An ease-out curve causes the animation to begin quickly, and then slow as it completes.
- `.CurveLinear`
  A linear animation curve causes an animation to occur evenly over its duration.
- `.TransitionNone`
  No transition is specified.
- `.TransitionFlipFromLeft`
  A transition that flips a view around its vertical axis from left to right.
- `.TransitionFlipFromRight`
  A transition that flips a view around its vertical axis from right to left.
- `.TransitionCurlUp`
  A transition that curls a view up from the bottom.
- `.TransitionCurlDown`
  A transition that curls a view down from the top.
- `.TransitionCrossDissolve`
  A transition that dissolves from one view to the next.
- `.TransitionFlipFromTop`
  A transition that flips a view around its horizontal axis from top to bottom.
- `.TransitionFlipFromBottom`
  A transition that flips a view around its horizontal axis from bottom to top.

# Dynamic Animation

# Dynamic Animation

- Aminar elementos basándose en la Física.
  - Gravedad, rozamiento, colisiones, …
- Pasos para programarlo:
  - Crear un objeto **UIDynamicAnimator**.
    - Se encarga de la animación siguiendo los **behaviors** e **items** añadidos .
  - Crear varios objetos **UIDynamicBehavior**.
    - Uno para la gravedad, otro para las colisiones, …
    - Estos objetos se añaden al objeto **UIDynamicAnimator** creado inicialmente.
  - Crear varios objetos **UIDynamicItem**.
    - Son cada uno de los elementos que se animarán.
    - Estos objetos se añaden a los objetos **UIDynamicBehavior** que interese.



```
@IBOutlet weak var unoLabel: UILabel!
@IBOutlet weak var dosLabel: UILabel!


var animator: UIDynamicAnimator!


var gravity: UIGravityBehavior!
var collision: UICollisionBehavior!
var dynamicItem: UIDynamicItemBehavior!
var push: UIPushBehavior!
var attachment: UIAttachmentBehavior!
var snap: UISnapBehavior!
```

```
override func viewDidLoad() {
    super.viewDidLoad()

    animator = UIDynamicAnimator(referenceView: view)

    gravity = UIGravityBehavior(items: [unoLabel, dosLabel])
    gravity.setAngle(0.4, magnitude: 1)

    collision = UICollisionBehavior(items: [unoLabel, dosLabel])
    collision.translatesReferenceBoundsIntoBoundary = true

    dynamicItem = UIDynamicItemBehavior(items: [unoLabel, dosLabel])
    dynamicItem.elasticity = 1
    dynamicItem.addAngularVelocity(1, forItem: unoLabel)

    push = UIPushBehavior(items: [unoLabel, dosLabel], mode: .Instantaneous)
    push.setAngle(CGFloat(M_PI), magnitude: 0.5)

    attachment = UIAttachmentBehavior(item: unoLabel, attachedToItem: dosLabel)
    attachment.damping = 0
    attachment.frequency = 1

    snap = UISnapBehavior(item: unoLabel, snapToPoint: view.center)
    snap.damping = 0

    animator.addBehavior(gravity)
    animator.addBehavior(collision)
    animator.addBehavior(dynamicItem)
    animator.addBehavior(push)
    animator.addBehavior(attachment)
    animator.addBehavior(snap)
}
```

```
@IBAction func pushPressed(sender: UIButton) {
    push.active = true
}
```

# UIDynamicItem

- Los elementos que se pueden animar deben ser conformes con el protocolo UIDynamicItem.
  - Requiere las propiedades **bounds**, **center** y **transform**
  - y opcionalmente **collisionBoundsType** y **collisionBoundingPath**.
  - La animación cambia la posición y gira los elementos, pero no cambia su tamaño.
- Las UIViews son conformes con él.
- Si se cambia la posición o se transforma un elemento programaticamente, hay que indicarlo llamando a:

```
func updateItemUsingCurrentState(item: UIDynamicItem)
```

# UIDynamicBehaviour

- Un behavior configura el comportamiento de los items añadidos a él.
- Es la clase padre de los comportamientos:
  - **`UIAttachmentBehavior`**, **`UICollisionBehavior`**, **`UIGravityBehavior`**, **`UIDynamicItemBehavior`**, **`UIPushBehavior`** y **`UISnapBehavior`**.
- Y puede usarse también para contener varios comportamientos hijos agrupados.
- Tienen la propiedad

```
var action: (() -> Void)?
```

  - que es un closure que se llama cada vez que el comportamiento actúa sobre sus items.

# UIDynamicAnimatorDelegate

- UIDynamicAnimator tiene un delegado al que se informa:
  - cada vez que la animación se detiene porque se ha llegado a un estado estable en el que nada tiene que moverse,
  - y cuando la animación va a empezar otra vez porque hay items que pueden/deben moverse.

```
func dynamicAnimatorDidPause(UIDynamicAnimator)

func dynamicAnimatorWillResume(UIDynamicAnimator)
```