LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

mobile and distributed systems group

Praktikum iOS-Entwicklung

Sommersemester 2015

Prof. Dr. Linnhoff-Popien

Florian Dorfmeister, Mirco Schönfeld

Agenda

Event Handling – Interaktionen mit dem Display

- Touch-Gesten
- Touch-Events
- Selektoren
- Protokolle
- Responder Chain

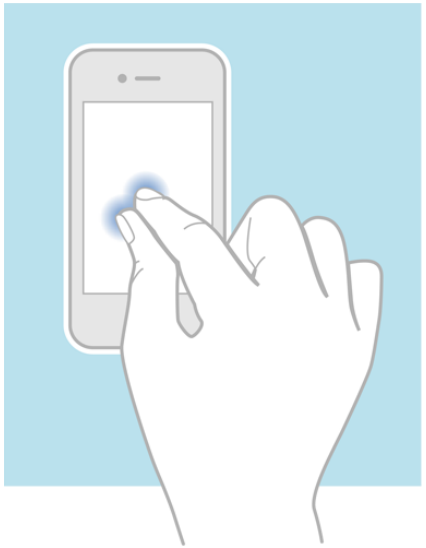
Event Handling – Interaktionen mit dem Display

Überblick

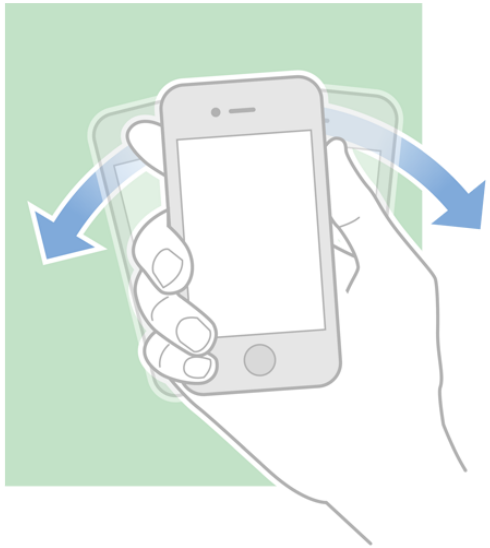
Der Nutzer kann durch Interaktion mit seinem Gerät verschiedene Events in iOS auslösen

Es gibt

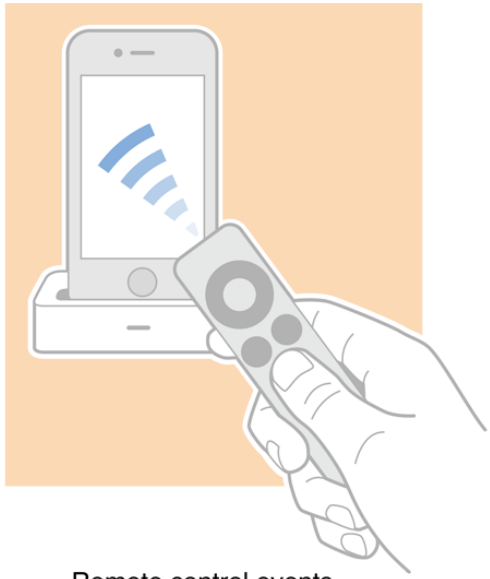
- (Multi-)Touch-Events
- Motion-Events
- Remote-Control-Events (zur Bedienung von Multimedia-Komponenten)



Multitouch events



Accelerometer events



Remote control events

Quelle: http://www.apple.com/

Überblick

Events sind Objekte, die an eine Anwendung gesendet werden, um sie über Interaktionen eines Nutzers zu informieren

Events, die durch den Nutzer ausgelöst werden, sind Instanzen der Klasse `UIEvent`

- enthalten Informationen, um auf das Event entsprechend zu reagieren
- haben einen Typ, z.B.
  - touch
  - shaking motion
  - remote control
- und einen Subtyp, z.B. für remote control
  - play
  - pause
  - stop
  - ...

TOUCH-GESTEN

Touch-Gesten: Überblick

Behandlung einzelner Berührungen der Finger mit dem Display (**Touch-Events)** ist in vielen Fällen viel zu aufwendig

Viele Interaktionen erfolgen nach dem gleichen Schema (Gesten)

- Tippen (Tap)
- Wischen (Swipe)
- Klammern (Pinch)
- Lange Drücken (LongPress)
- ...

→ Verwendung einer "Higher-Level-API" zur Behandlung von Gesten

Touch-Gesten: Überblick

Geste = Kombinationen von Touch-Events

Intuitive Gesten werden von iOS über "Gesture Recognition" direkt erkannt

- Tippen (`UITapGestureRecognizer`)
- Pinchen (`UIPinchGestureRecognizer`)
- Wischen (`UISwipeGestureRecognizer`)
- Ziehen/Verschieben (`UIPanGestureRecognizer`)
- ...

Für viele Gesten gibt es in Kombination mit typischen UI-Elementen eine direkte Integration in `UIKit`:

- Beispiel: `UIScrollView`
  - Pinch-Geste → Zoom
  - Wisch-Geste → Scroll

Touch-Gesten: Erkennen und behandeln

Gesture Recognizer (`UIGestureRecognizer`)

- Dienen der Abstraktion von der komplexen Event-Handling-Logik
- Stellen den bevorzugten Weg zur Behandlung von Touch-Events dar

Implementieren eigener Action-Target-Mechanismen:

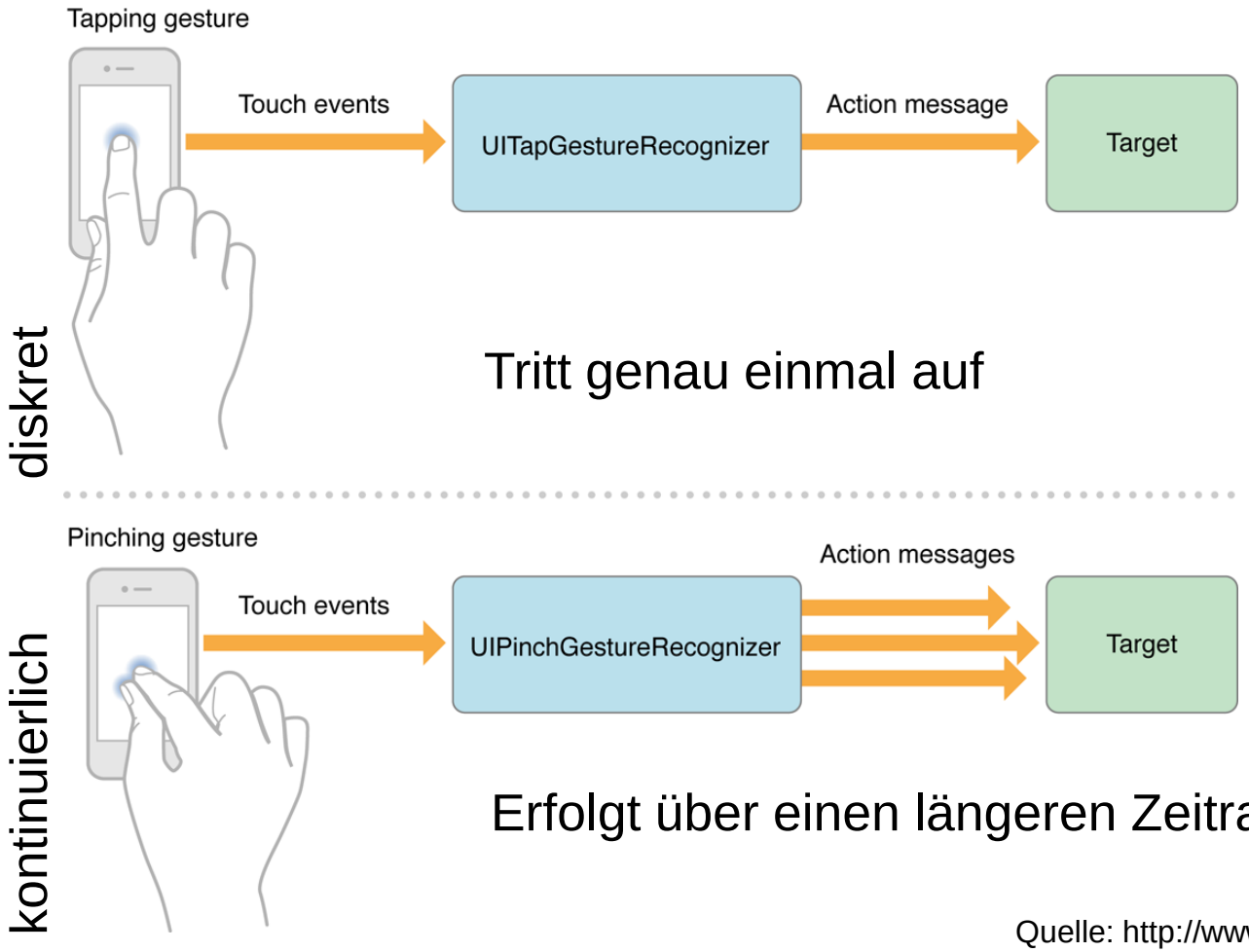
- Instanziieren eines Gesture Recognizers
- Hinzufügen des Gesture Recognizer zur eigenen View
  - Gesamte View reagiert auf die hinzugefügte Geste

Vorgehensweise:

- Verwendung und Anpassung von **Built-In** Gesture Recognizern, oder
- Implementierung eigener (**Custom**) Gesture Recognizer

Unterscheidung von Gesten

Es existieren **diskrete** und **kontinuierliche** Gesten

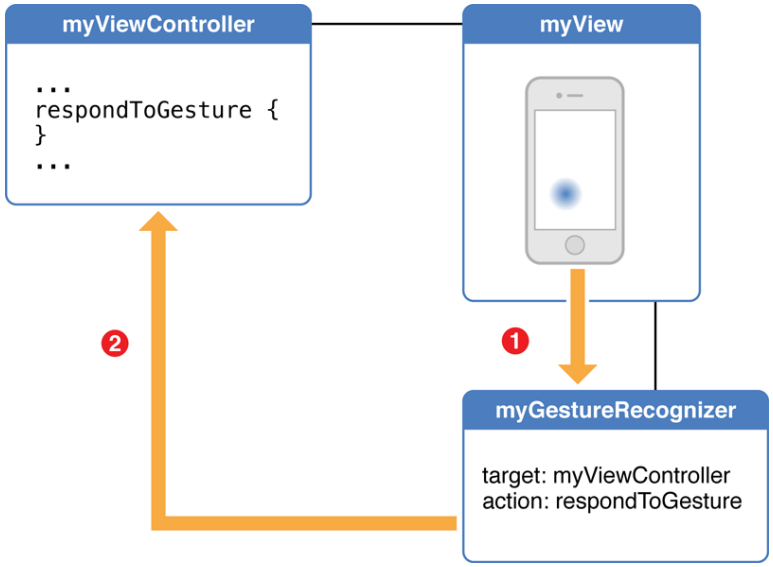


Quelle: http://www.apple.com/

Behandeln von Events mit Gesture Recognizern

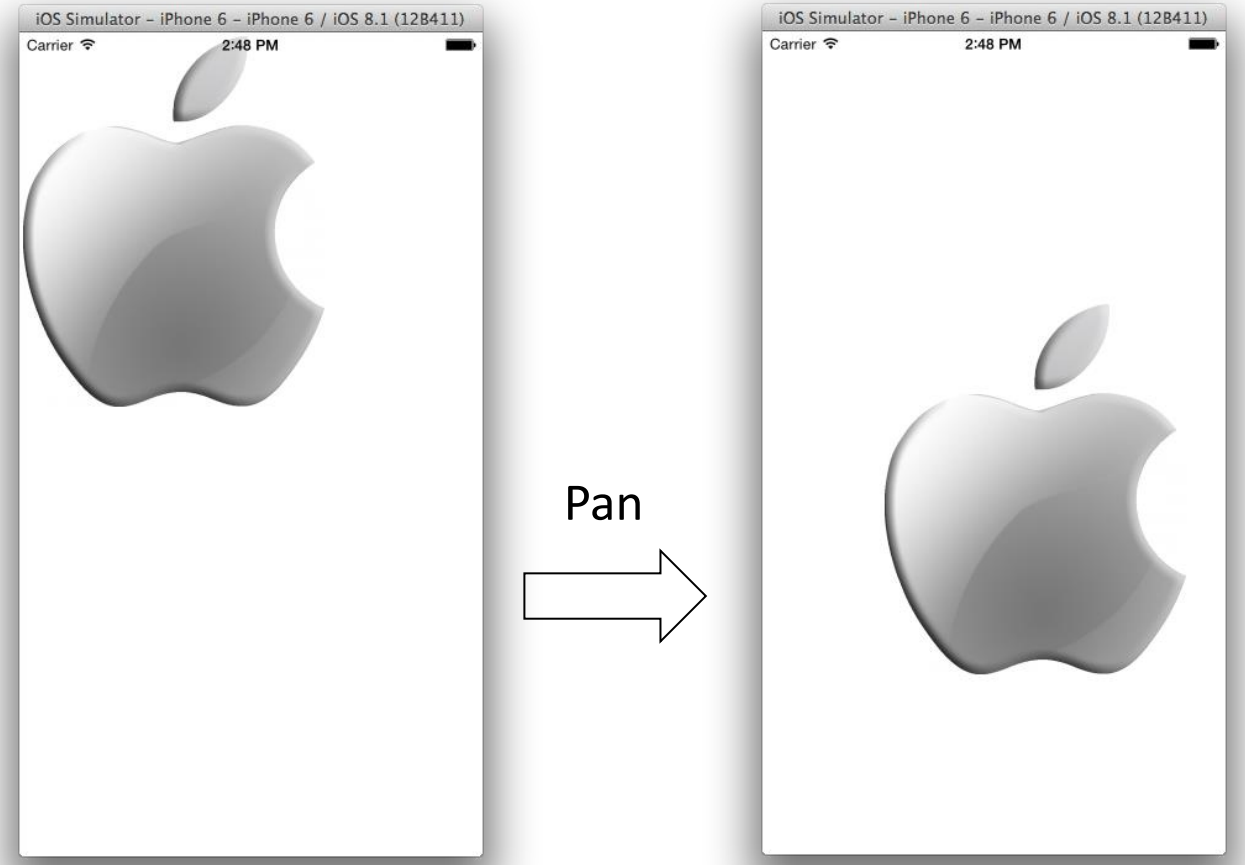
Hinzufügen eines Built-In Gesture Recognizers

- Erzeugen und konfigurieren einer Instanz von `UIGestureRecognizer`
  - Zuweisen eines Targets und einer Action
  - Zuweisen gestenspezifischer Attribute (optional; z.B. `numberOfTapsRequired`)
- Registrieren des Gesture Recognizers mit einer View
- Implementieren der Action-Methode



Quelle: http://www.apple.com/

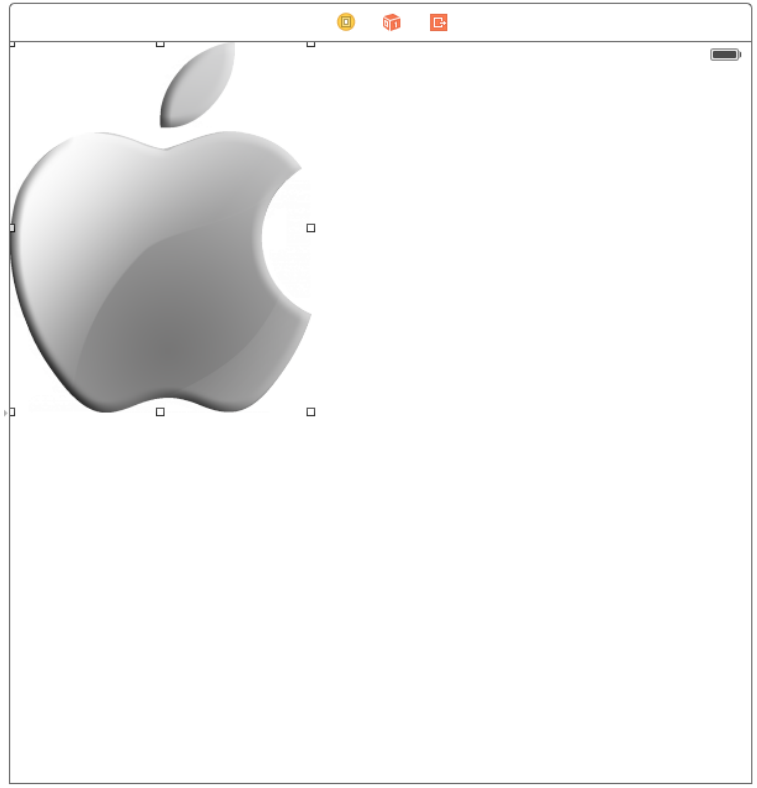
Beispiel: ImageTranslationRecognition



Beispiel: ImageTranslationRecognition

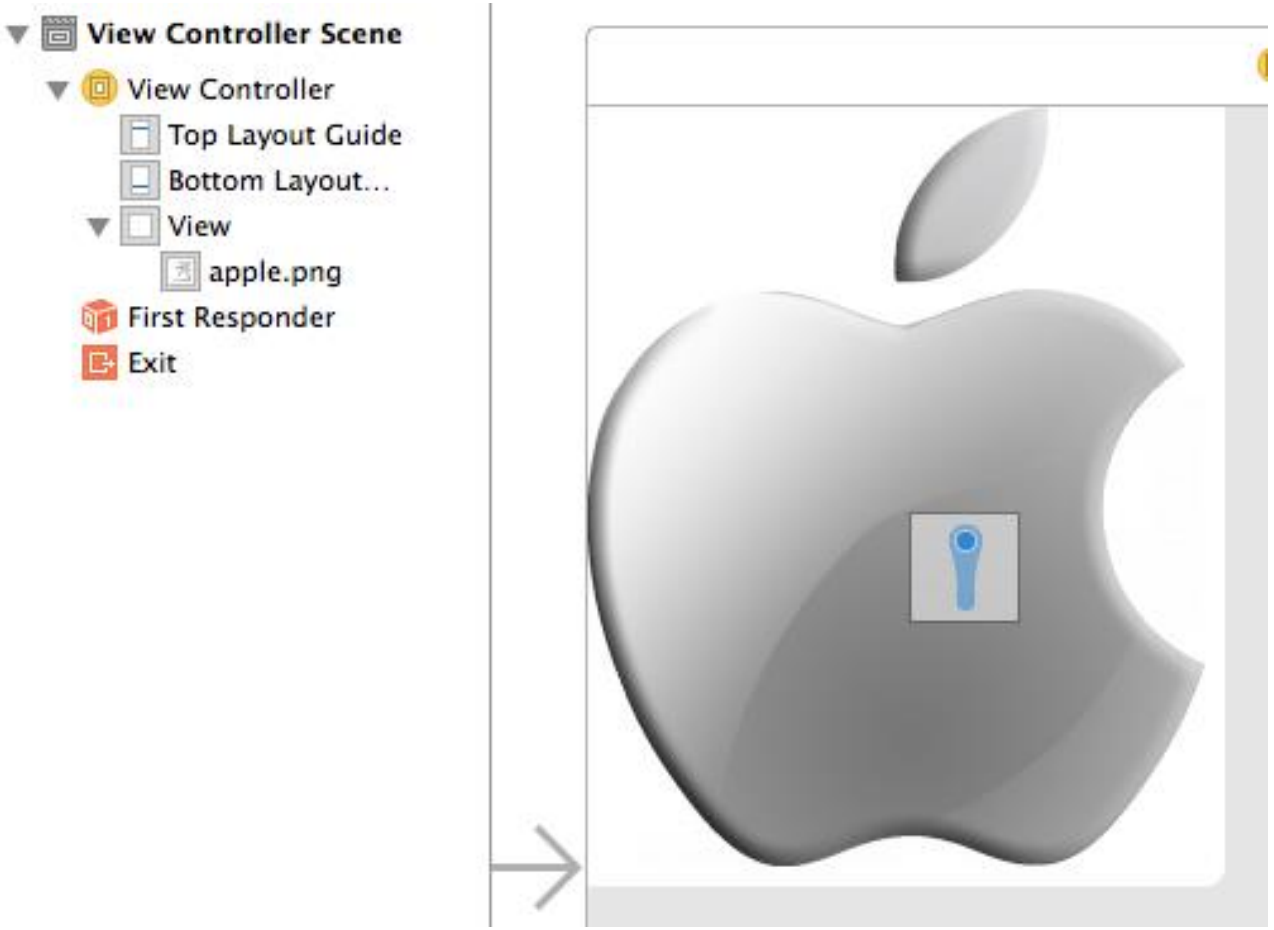
Vorgehensweise

- Erzeugen eines neuen Projekts "ImageTranslationRecognition" (Template: Single View Application)
- Im Storyboard:
  - Hinzufügen einer Image View vom Typ `UIImageView`
- Im Project Navigator:
  - Hinzufügen eines Image (Drag & Drop vom Finder)
- Unter „Editor“:
  - Size to fit content: Anpassen der Größe der Image View an das Image



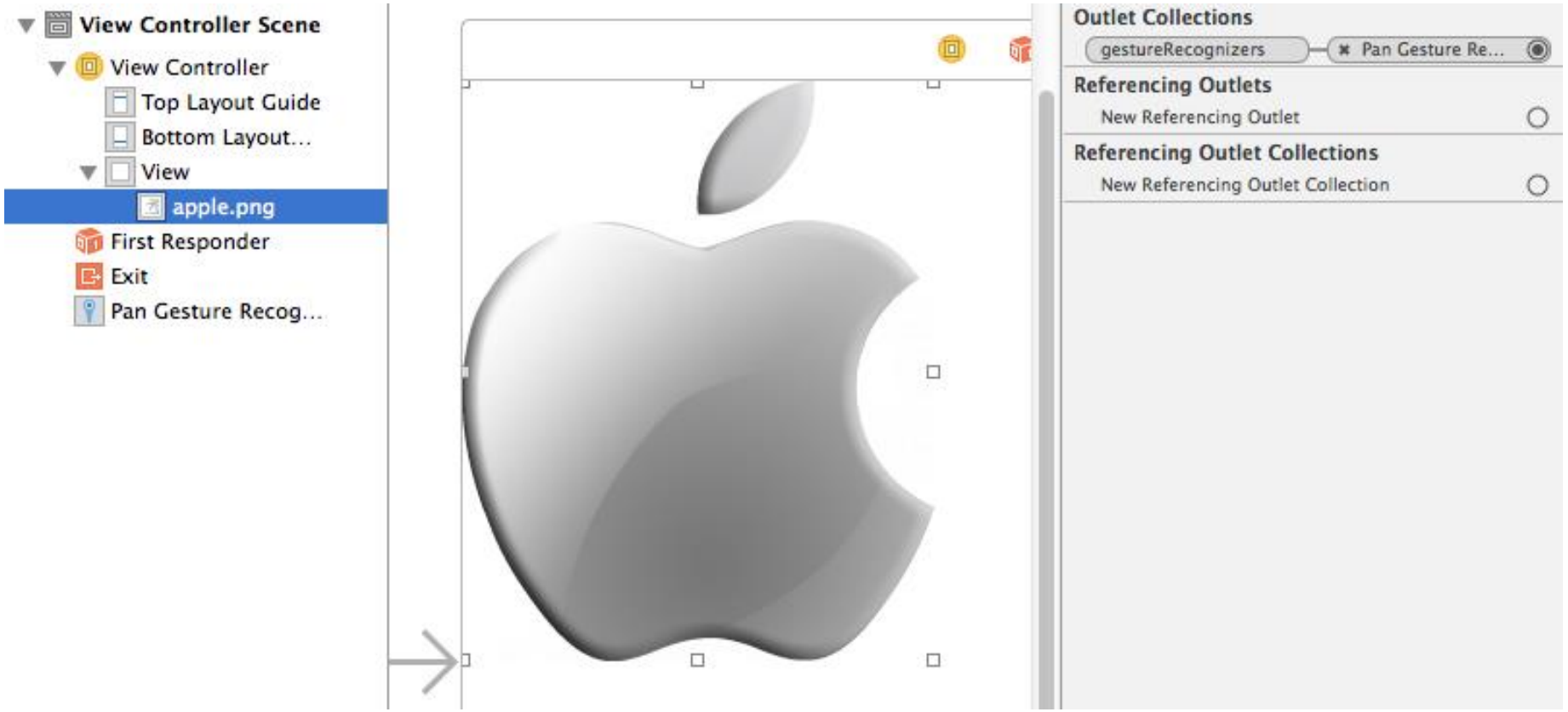
Beispiel: ImageTranslationRecognition

- Von der Objekt Library:
  - Hinzufügen eines Pan Gesture Recognizers vom Typ `UIPanGestureRecognizer` zur Image View (Drag & Drop auf die View)



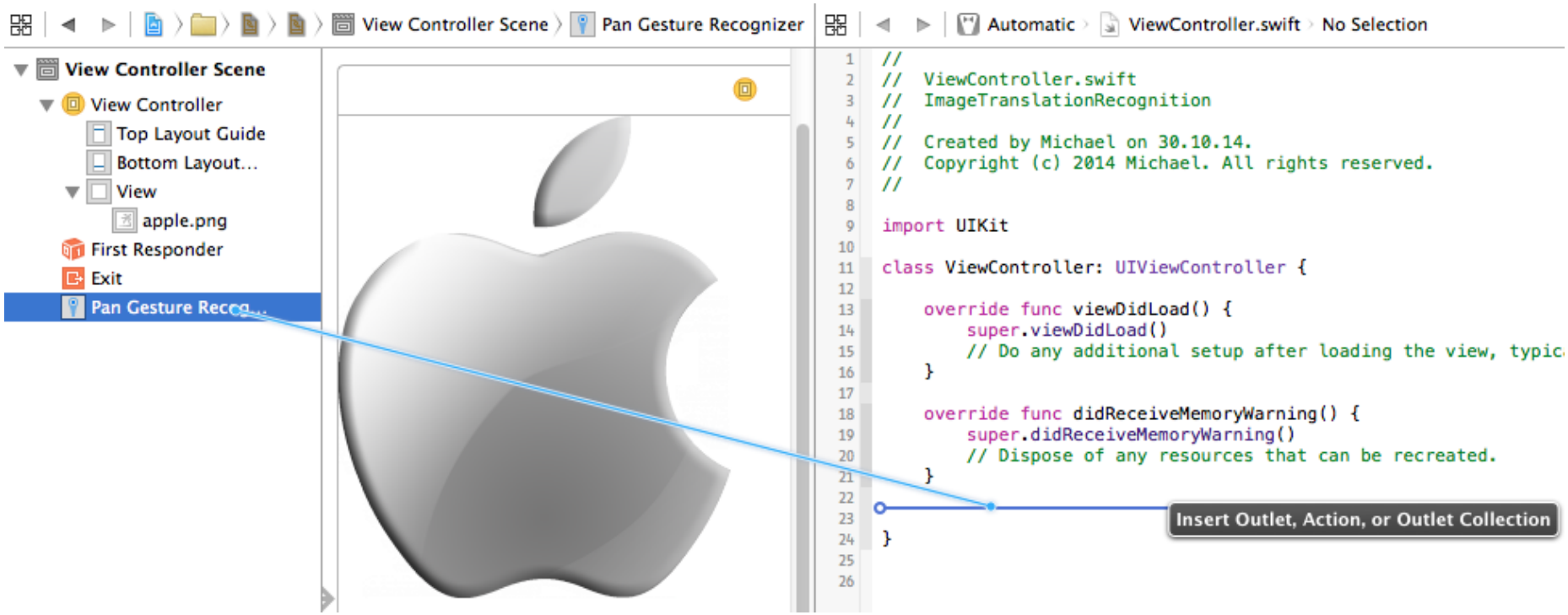
Beispiel: ImageTranslationRecognition

- Pan Gesture Recognizer ist jetzt als Outlet bei der Image View installiert



Beispiel: ImageTranslationRecognition

- Erzeugen und Verbinden einer Action-Methode



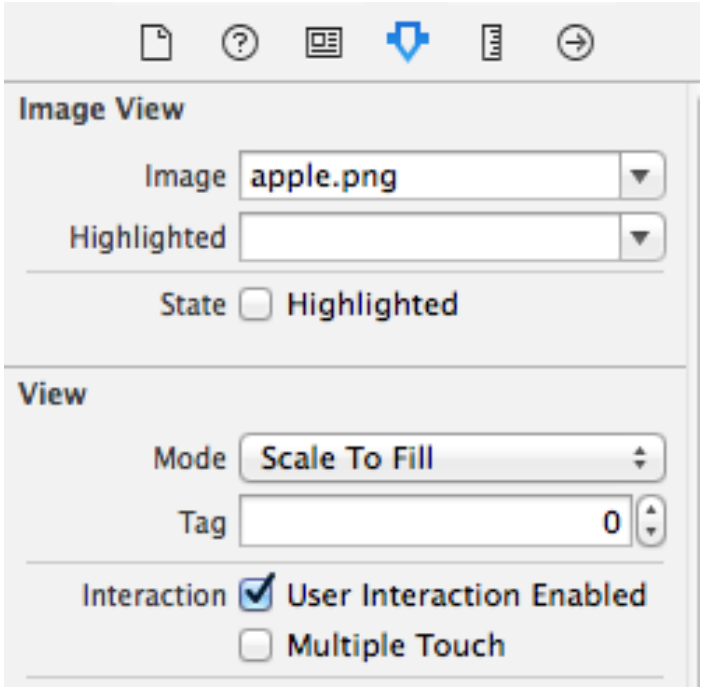
Beispiel: ImageTranslationRecognition

- Implementierung der Action-Methode

```
//  ViewController.swift

import UIKit

class ViewController: UIViewController {

    @IBAction func handlePan(sender: UIPanGestureRecognizer) {
        // Die Translation liefert hier den absoluten Abstand
        // zu dem Punkt, an dem der Finger auf den Bildschirm traf
        // bezogen auf das Koordinatensystem der übergebenen View
        let translation:CGPoint = sender.translationInView(self.view)

        // Anpassen der Position des Images
        sender.view!.center = CGPoint(x:sender.view!.center.x + translation.x,
            y:sender.view!.center.y + translation.y)

        // Zurücksetzen der Translation, um nur relative Positionsveränderungen
        // zur Position im vorhergehenden Aufruf der Action-Methode zu erhalten
        sender.setTranslation(CGPointZero, inView: self.view)
    }
}
```

Beispiel: ImageTranslationRecognition

Im Attributes Inspector: Erlauben von Nutzerinteraktionen mit der Image View



Beispiel: ImageTranslationRecognition

Allgemeine Anmerkungen

- Der Aufruf der Action-Methode erfolgt jedes Mal, wenn die Geste in der entsprechenden View erkannt wird

- Das Referenzieren eines Gesture Recognizer außerhalb der Action-Methode ist über eine Outlet-Connection möglich

Anmerkungen zum `UIPinchGestureRecognizer`

- Wenn man den Skalierungsfaktor nicht selbst verändert, wird er absolut berechnet
  - s Skalierungsfaktor
  - d_{Start} Abstand der Finger bei erster Berührung
  - d_{Now} Gegenwärtiger Abstand der Finger

→ $s = d_{Now} / d_{Start}$

Programmatisches Erzeugen eines UIGestureRecognizers

Initialisierung mit `initWithTarget:action:`

- `target`:
  - Target-Objekt, das in einer Action-Methode auf eine Geste reagiert
  - I.d.R. der zugehörige View Controller
- `action`:
  - Ein **Selektor**
  - Spezifiziert die Action-Methode, die beim Auftreten der Geste ausgeführt werden soll

Registrieren des Gesture Recognizers mit `addGestureRecognizer:` des entsprechenden `UIView`-Objekts

Programmatische Erzeugung erfolgt i.d.R. in `viewDidLoad`

Programmatisches Erzeugen eines UIGestureRecognizers

Beispiel:

- die Action-Methode wird als **Selektor** spezifiziert

```
-(void)viewDidLoad {
    [super viewDidLoad];

    // Erzeugen und Initialisieren einer Tipgeste
    UITapGestureRecognizer *tapRecognizer =
        [[UITapGestureRecognizer alloc] initWithTarget:self
        action:@selector(respondToTapGesture:)];

    // Nur einzelnes Tippen wird als Geste erkannt
    tapRecognizer.numberOfTapsRequired = 1;

    // Hinzufügen des Gesture Recognizers zur eigenen View
    [self.view addGestureRecognizer:tapRecognizer];

    // Weitere Initialisierungsschritte
}
```

Selektor: Was ist das?

Ein Selektor

- identifiziert eine Methode
- ist der Name, der verwendet wird, um eine Methode zu referenzieren, die auf einem Objekt aufgerufen werden soll, bzw.
- ist ein **eindeutiger Bezeichner**, der den Namen einer Methode beim Übersetzen des Quellcodes ersetzt
- entspricht einem dynamischen Zeiger, der – unabhängig von der dazugehörigen Klasse – automatisch die Implementierung einer passenden Methode referenziert

Selektor referenzieren in Objective-C

Selektoren besitzen den Typ `SEL`

Es gibt zwei Wege, Selektoren zu referenzieren

- Zur Übersetzungszeit
  - Verwenden der Direktive `@selector`
  - `SEL mySel = @selector(methodName)`
- Zur Laufzeit (wenn Name der Methode zur Übersetzungszeit noch nicht bekannt)
  - Verwenden der Funktion `NSSelectorFromString`
  - Der String-Parameter entspricht dem Namen der aufzurufenden Methode
  - `SEL mySel = NSSelectorFromString(@"methodName");`

Selektoren verwenden in Objective-C

Aufruf einer Methode über einen Selektor mit den Methoden (aus `NSObject`)

- `performSelector:`
- `performSelector:withObject:`
- `performSelector:withObject:withObject:`
- `Bei mehr/anderen Parametern: NSInvocation`

Beispiel:

```
SEL mySelector = @selector(accelerate);

[myBoat performSelector:mySelector];
[myCar performSelector:mySelector];
```

Selektoren verwenden in Objective-C

Objective-C Verwendet Message Passing!

- Methodenaufrufe über Selektoren sind unsicher!
- Man kann vor dem Aufruf eines Selektors überprüfen, ob das Callee-Objekt eine entsprechende Methode besitzt!

```
CustomClass *myObject = [CustomClass new];
SEL mySel = NSSelectorFromString(@"methodName");

if ([myObject respondsToSelector:@selector(mySel)]) {
    [myObject performSelector:mySel];
}
else {
    // Fehler behandeln...
}
```

Selektoren in Swift

In Swift werden Selektoren durch die Struktur `Selector` repräsentiert

- Erzeugung erfolgt mit Hilfe von String Literalen

```
let mySel: Selector = "methodName:"
```

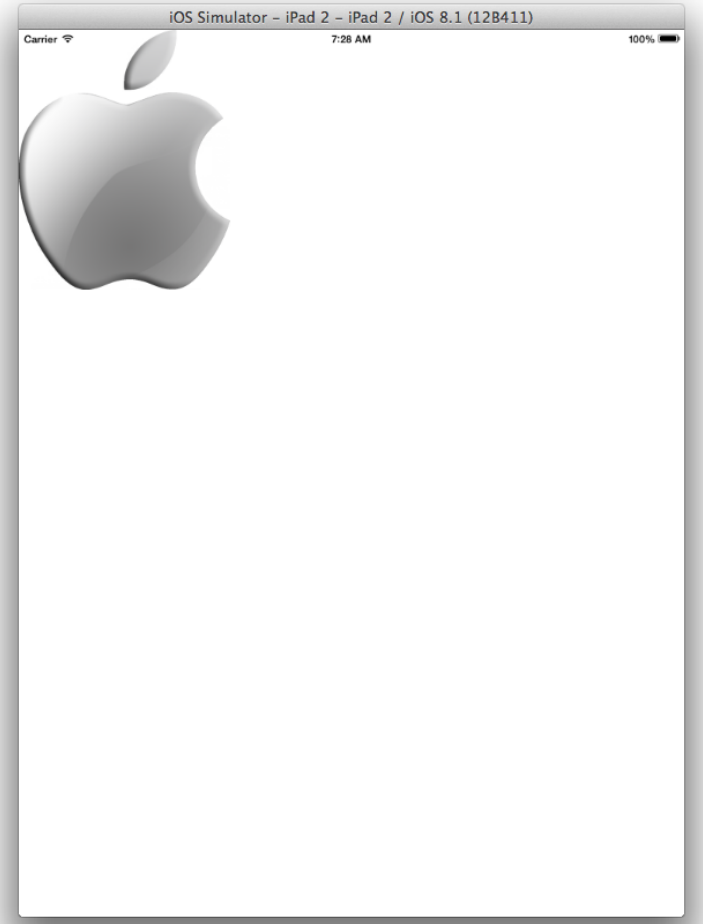
- String Literale lassen sich automatisch in Selektoren konvertieren
  → Literale können direkt als Argument übergeben werden, wenn ein Selektor erwartet wird

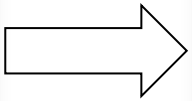
```
button.addTarget(
    self, action: "buttonPressed:", forControlEvents: .TouchUpInside)
```

- **Achtung:** Methoden zum Aufruf von Selektoren wie `performSelector:` oder `respondsToSelector:` sind inhärent unsicher
  → Swift bietet diese Methoden nicht an (stattdessen Verwendung von Optionals)

Beispiel zur Verwendung von Selektoren: TapRecognition

Tippen auf den Bildschirm bewirkt das Verschieben eines Images an die angetippte Stelle

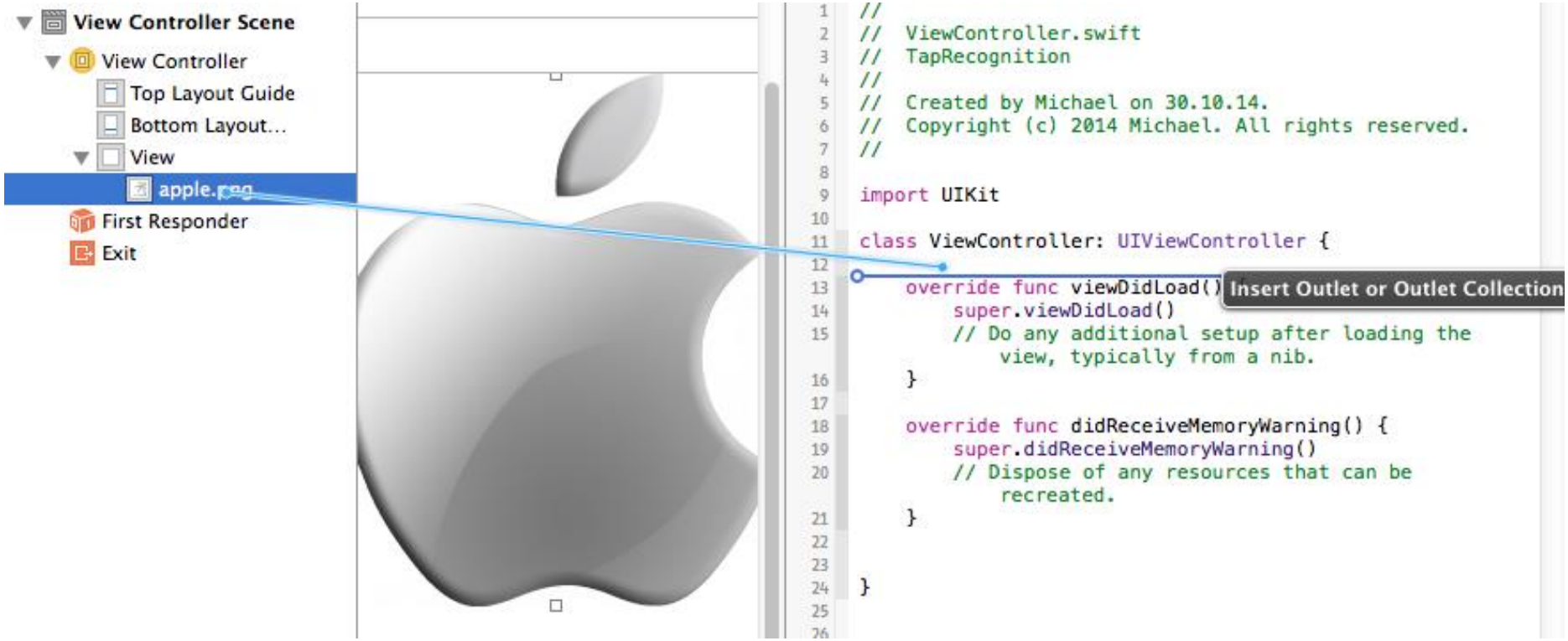




Beispiel: TapRecognition

Vorgehensweise

- Analog zum Beispiel ImageTranslationRecognition
- Erzeugen einer Outlet-Connection zur Image View



Beispiel: TapRecognition

Programmatisches Hinzufügen eines `UITapGestureRecognizers` (geht natürlich auch über Interface-Builder...)

```
//  ViewController.swift

import UIKit

class ViewController: UIViewController {

    @IBOutlet weak var imageView: UIImageView!

    override func viewDidLoad() {
        // Initialisieren eines UITapGestureRecognizers
        let tgr = UITapGestureRecognizer(target: self, action:handleTap)

        // Registrieren mit der eigenen View
        self.view.addGestureRecognizer(tgr)
    }
}
```

Beispiel: TapRecognition

Implementierung der Action-Methode

- Achtung: Nur zwei Signaturen erlaubt:
  - `func handleTap() -> Void`
  - `func handleTap(tgr:UITapGestureRecognizer) -> Void`

```
func handleTap(tgr:UITapGestureRecognizer) -> Void {
    // Position wo das Event auftrat
    let location = tgr.locationInView(self.view)

    // Verschieben des Images an die neue Position
    self.imageView.center = location;

}
```

Behandlung von Touch-Gesten im Gesture Recognizer

Eine Instanz von `UIGestureRecognizer` gehört **zu genau einer** `UIView`-Instanz

- Hinzufügen desselben Gesture Recognizers zu verschiedenen `UIView`-Instanzen nicht möglich!
- Ein wiederholtes Hinzufügen überschreibt das vorhergehende!
- Beispiel:
  - Nur `view2` erhält die Nachricht `tapTapTap`

```
// MyController.m
[...]

UITapGestureRecognizer *tapGesture =
   [[UITapGestureRecognizer alloc] initWithTarget:self
      action:@selector(tapTapTap:)];

[self.view1 addGestureRecognizer:tapGesture];
[self.view2 addGestureRecognizer:tapGesture];

[...]
```

Behandlung von Touch-Gesten im Gesture Recognizer

`UIGestureRecognizer` unterbricht standardmäßig Behandlung von Touch-Events, wenn eine Geste erkannt wurde

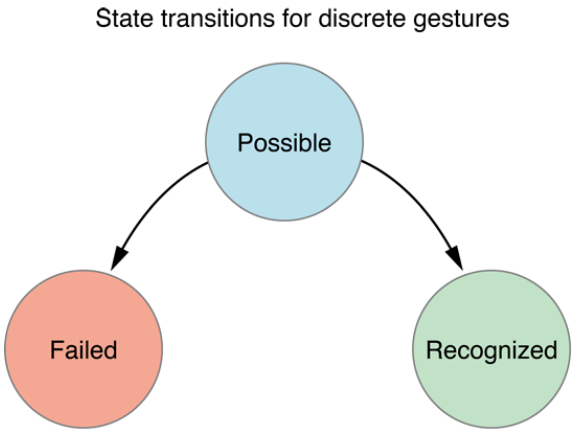
Problem:

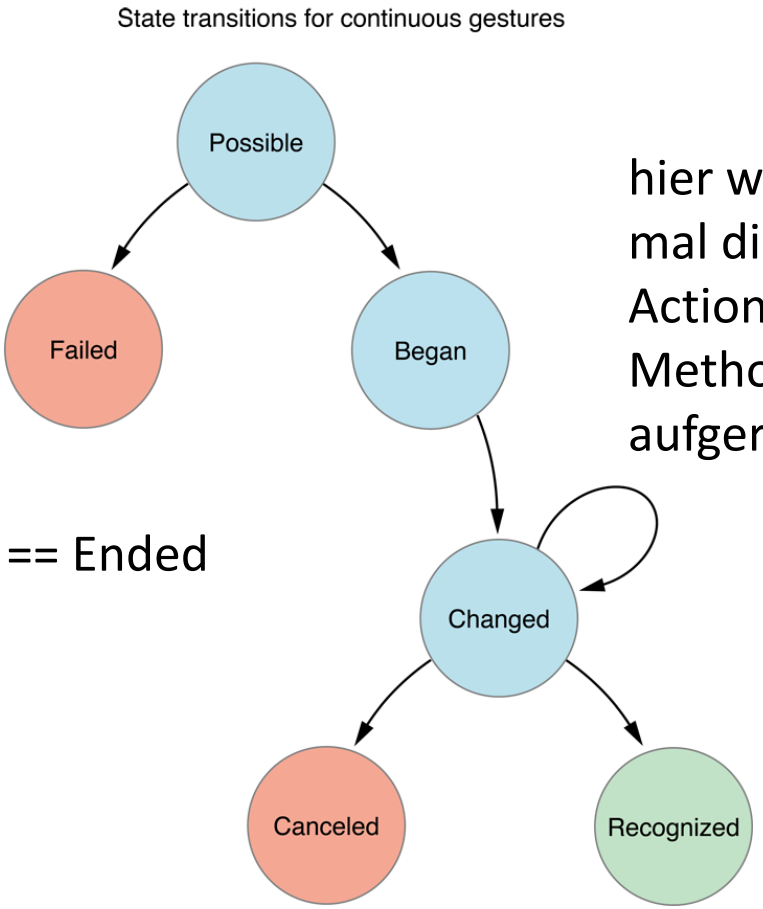
- `UIView` mit einer Gestenerkennung erhält möglicherweise nicht alle Touch-Events.

Gesture Recognizer: Zustandsmodell

Gesture Recognizer operieren in einer Zustandsmaschine

- mehr dazu im *"Event Handling Guide for iOS"*





hier wird jedes mal die Action-Methode aufgerufen

Achtung: Recognized == Ended

Quelle: http://www.apple.com/

Gesture Recognizer: Zustandsmodell

Testen des gegenwärtigen Zustands über die Konstanten

- `UIGestureRecognizerStatePossible` (Default State)
  - Es wurden eventuell Touch-Events registriert, aber noch keine Geste erkannt
- `UIGestureRecognizerStateBegan`
  - Es wurde ein Touch-Event empfangen und eine **kontinuierliche** Geste erkannt
    → Löst eine Action-Nachricht aus
- `UIGestureRecognizerStateChanged`
  - Es wurden Touch-Events erkannt, die eine Veränderung in der **kontinuierlichen** Geste signalisieren
    → Löst eine Action-Nachricht aus
- `UIGestureRecognizerStateEnded`
  - Es wurden Touch-Events erkannt, die das Ende einer (**kontinuierlichen**) Geste signalisieren
    → Löst eine Action-Nachricht aus
    → Reset auf `UIGestureRecognizerStatePossible`

Gesture Recognizer: Zustandsmodell

Testen des gegenwärtigen Zustands über die Konstanten

- `UIGestureRecognizerStateFailed`
  - Es wurde eine Sequenz von Touch-Events empfangen, die aber nicht als Teil der gegenwärtigen Geste erkannt werden konnten
    → Keine Action-Nachricht
    → Reset auf `UIGestureRecognizerStatePossible`

- `UIGestureRecognizerStateRecognized`
  - Es wurden eine Sequenz von Touch-Events als Geste erkannt
    → Löst eine Action-Nachricht aus
    → Reset auf `UIGestureRecognizerStatePossible`

Hinweise:

- Die Konstanten sind als `enum` realisiert.
- In Swift erfolgt der Zugriff jedoch über die Member Values (z.B. `UIGestureRecognizerState.Began`)
- In Swift existiert keine Konstante `.Recognized`

Gesture Recognizer: Erkennen paralleler Gesten

Beispiel: Drag & Drop:

- Realisierung normalerweise mit LongPress und Pan auf ein UI-Objekt.
- LongPress zum Starten des Drag-Vorgangs, Pan zum Bewegen
- Die gesamte Pan-Geste **liegt vollständig innerhalb** der LongPress-Geste!

Problem:

- Standardmäßig wird das parallele Erkennen mehrerer Gesten unterdrückt
- Hier muss aber `Pan` während eines `LongPress` erkannt werden
- LongPress gibt Touch-Events nicht weiter
  → Pan kann nicht erkannt werden!

Lösung:

- Implementierung des `UIGestureRecognizerDelegate` Protokolls

Protokoll: Was ist das?

Die Klasse eines Objekts unterscheidet sich häufig von dessen **Rolle** in einem System.

Beispiel

- Die Klasse eines Objekts ist `NSMutableArray` aber seine Rolle in der Anwendung ist eine Warteschleife (für Druckaufträge).

Spezifikation von Rollen erfolgt in Objective-C / Swift über Protokolle (ähnlich zu Interfaces in Java).

Protokolle

- stellen eine Alternative zur Vererbung dar
- können von beliebigen Klassen implementiert werden
- ermöglichen es Objekten schwach assoziierter Klassen miteinander zu kommunizieren

Protokoll: Definition in Objective C

Protokolle spezifizieren die zu implementierenden Methoden

Methoden können als **verpflichtend** oder **optional** spezifiziert werden

Beispiel: `UIGestureRecognizerDelegate`-Protokoll

```
// Dieses Protokoll ist konform zum NSObject Protokoll
@protocol UIGestureRecognizerDelegate <NSObject>

// die folgenden Methoden MÜSSEN implementiert werden
@required
[...]

// die folgenden Methoden KÖNNEN implementiert werden
@optional
-(BOOL)gestureRecognizer:(UIGestureRecognizer *)gr
    shouldRecognizeSimultaneouslyWithGestureRecognizer:
    (UIGestureRecognizer *)o;

[...]

@end
```

Protokoll: Definition in Swift

Protokolle können ebenfalls als **verpflichtend** oder **optional** spezifiziert werden

- Dazu muss man es mit dem Attribut `@objc` markieren

```
@objc protocol SomeDelegate {
    // die folgenden Methoden MÜSSEN implementiert werden
    func doSomeCalculatiom(count: Int) -> Int

    // die folgenden Methoden KÖNNEN implementiert werden
    optional func doSomeOtherStuff() -> Void
[...]

@end
```

In Swift existieren viele weitere Möglichkeiten zur Definition von Protokollen und zur Restriktion ihrer Anwendung

- Siehe: https://developer.apple.com/library/ios/documentation/swift/conceptual/Swift_Programming_Language/Protocols.html

Gesture Recognizer: Erkennen paralleler Gesten

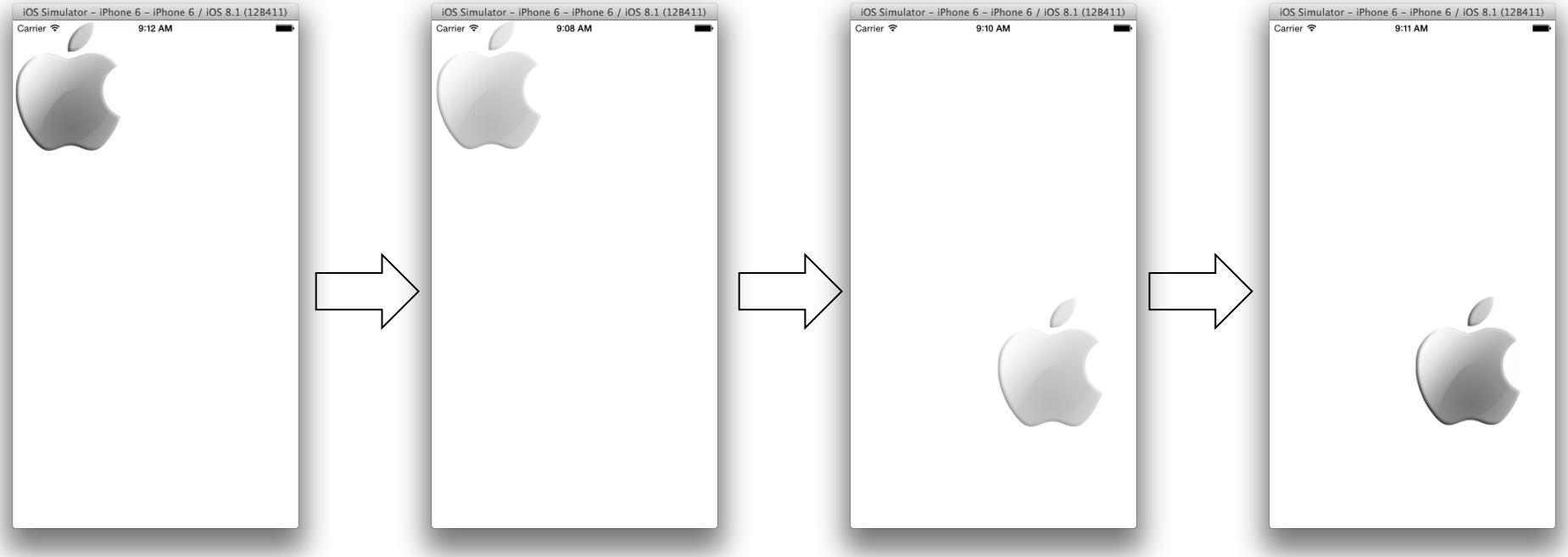
Problem:

- Parallele Erkennung von LongPress und Pan

Lösung:

- Implementierung des UIGestureRecognizerDelegate Protokolls

Beispiel: DragAndDrop



Long Press Pan Drop

Beispiel: DragAndDrop

Vorgehensweise

- Analog zum Beispiel "ImageTranslationRecognition"
- Kennzeichnung des View Controller als Delegate

```
//  ViewController.swift

import UIKit

class ViewController: UIViewController, UIGestureRecognizerDelegate {
    [...]
}
```

Beispiel: DragAndDrop

Erzeugen notwendiger Outlets und Properties

```
//  ViewController.swift

import UIKit

class ViewController: UIViewController, UIGestureRecognizerDelegate {

    @IBOutlet weak var imageView: UIImageView!
    var panGR:UIPanGestureRecognizer!
    var longPressGR:UILongPressGestureRecognizer!

    [...]
}
```

Initialisierung der Gesture Recognizer

```
//  ViewController.swift

import UIKit

class ViewController: UIViewController, UIGestureRecognizerDelegate {
    [...]

    override func viewDidLoad() {
        // Initialisieren/Registrieren eines UILongPressGestureRecognizer
        self.longPressGR = UILongPressGestureRecognizer(
            target:self, action:"handleLongPress:")
        self.imageView.addGestureRecognizer(self.longPressGR)

        // Initialisieren/Registrieren eines UIPanGestureRecognizer
        self.panGR = UIPanGestureRecognizer(target: self, action: "handlePan:")
        self.imageView.addGestureRecognizer(self.panGR)

        // Setzen des UILongPressGestureRecognizer als Delegate
        self.longPressGR.delegate = self
    }
}
```

Beispiel: DragAndDrop

Gibt die Protokoll-Methode `true` zurück, können die beiden übergebenen `UIGestureRecognizer` parallel zueinander Gesten erkennen.

```
//  ViewController.swift

import UIKit

class ViewController: UIViewController, UIGestureRecognizerDelegate {
    [...]

    func gestureRecognizer(
            UIGestureRecognizer,
            shouldRecognizeSimultaneouslyWithGestureRecognizer:
            UIGestureRecognizer) -> Bool {
        // Erlaube parallele Verarbeitung von Pan während LongPress
        // Achtung: Dies erlaubt auch den LongPress während Pan!
        return true
    }

}
```

Implementierung des LongPress-Handlers

```
//  ViewController.swift

import UIKit

class ViewController: UIViewController, UIGestureRecognizerDelegate {
    [...]
    func handleLongPress(gestureRecognizer:UILongPressGestureRecognizer) {
        // Visualisiere, dass jetzt ein Drag & Drop passiert
        if gestureRecognizer.state == UIGestureRecognizerState.Began
        || gestureRecognizer.state == UIGestureRecognizerState.Changed {
            self.imageView.alpha = 0.5
        }
        else {
            // Visualisiere, dass Drag & Drop zu Ende ist
            self.imageView.alpha = 1.0
        }
    }
}
```

Implementierung des Pan-Handlers

```
//  ViewController.swift

import UIKit

class ViewController: UIViewController, UIGestureRecognizerDelegate {

    [...]

    func handlePan(gestureRecognizer:UIPanGestureRecognizer) {
        // Führe Pan nur aus, wenn gerade ein LongPress aktiv ist
        if self.longPressGR.state == UIGestureRecognizerState.Began
        || self.longPressGR.state == UIGestureRecognizerState.Changed {
            // Verschiebe Image an die Position, wo das Event auftrat
            self.imageView.center =
                gestureRecognizer.locationInView(self.view)
        }
    }
}
```

Touch-Events

Überblick

Gesture Recognizer können nicht immer zur Behandlung von Touch-Events verwendet werden

- Oft ergibt sich kein Vorteil, wenn man das Erkennen eines Touches von der assoziierten Aktion entkoppelt

- Stehen die Inhalte einer View in direktem Zusammenhang mit den erkannten Touches, bietet sich auch deren direkte Behandlung im entsprechenden `UIView`-Objekt an

- Beispiel: Zeichenprogramm
  - Linie soll sich verändern, während sich der Finger bewegt

Wie und wo kommt man an diese Events?

Touches, Touch-Events und Multitouch-Sequenzen

Ein **Touch** entspricht dem Berühren oder Bewegen eines Fingers über den Bildschirm

Eine Geste besteht aus einem oder mehreren Touches

Ein Touch wird als Instanz der Klasse `UITouch` repräsentiert

Beispiel Pinch:

- Besteht aus zwei Touches
- Zwei Finger auf dem Bildschirm, die sich aufeinander zu bzw. voneinander weg bewegen

Touches, Touch-Events und Multitouch-Sequenzen

Touches werden über **Events** kommuniziert

- Ein Event beinhaltet **alle** `UITouch`-Objekte, die innerhalb einer **Multitouch**-Sequenz auftreten

Eine **Multitouch-Sequenz**

- startet mit dem Berühren des ersten Fingers und
- endet mit dem Entfernen des letzten Fingers vom Bildschirm

Wenn sich ein Finger bewegt, werden `UITouch`-Objekte gekapselt in einem `UIEvent`-Objekt gesendet (Typ `UIEventTypeTouches`)

Touches, Touch-Events und Multitouch-Sequenzen

Ein `UITouch`-Objekt

- verfolgt genau einen Finger (= Tracking)
- dauert (bzw. existiert) exakt so lange, wie die Multitouch-Sequenz

Während der Multitouch-Sequenz kümmert sich iOS um das Update der Attribute des `UITouch`-Objekts

- Man muss `UITouch`-Objekte nicht „zwischenspeichern“
- Man **sollte** das auch nie probieren!
  - Werden vom OS verwaltet (ARC, etc.)

- Falls nötig: Kopien von Werten erzeugen oder schwache Referenz

Stadium und Ort von Touches verfolgen

Für jeden Berührungspunkt (Finger) wird ein `UITouch`-Objekt mit folgenden Properties erzeugt

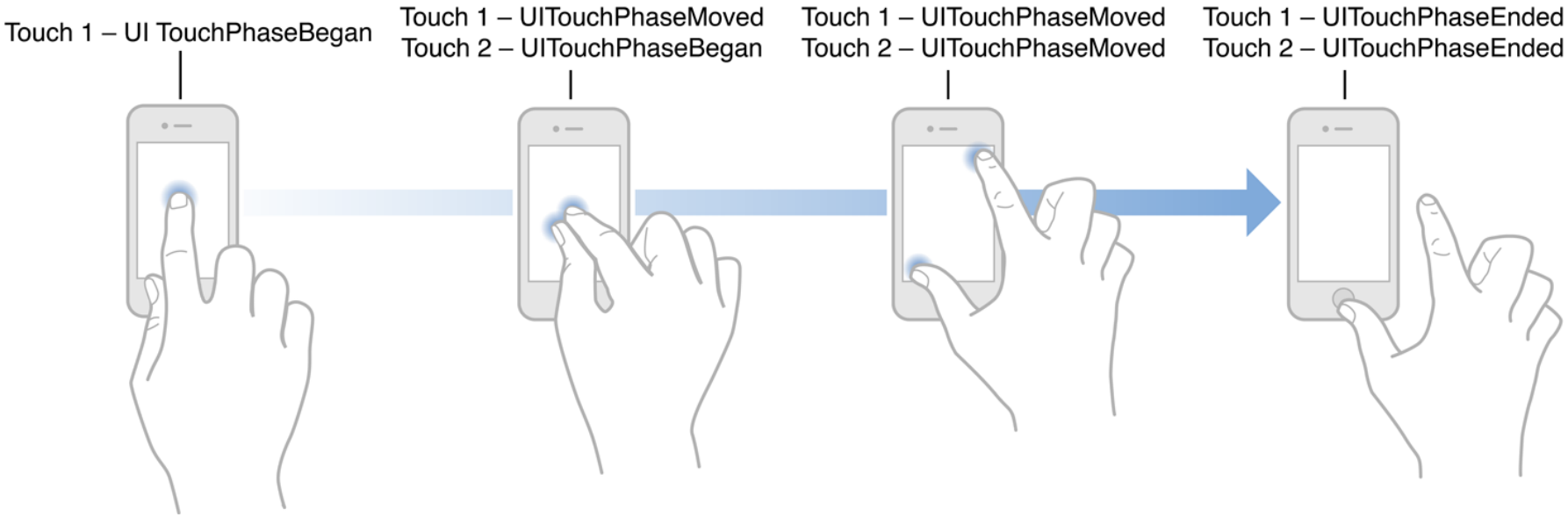
- `phase`: Began, Moved, Ended bzw. Cancelled
- `locationInView`: Position in einer View
- `previousLocationInView`: Vorhergehende Position in der View
- `timeStamp`: Zeitpunkt des Auftretens bzw. der letzten Veränderung

Location steht auf drei Arten zur Verfügung:

- Als Referenz auf das `UIWindow`-Objekt, in dem der Touch auftrat
- Als Referenz auf das `UIView`-Objekt innerhalb dieses `UIWindow`-Objekts
- Als Position innerhalb dieses/eines `UIView`-Objekts

Phasen eines UITouch-Objekts

Beispiel



Quelle: http://www.apple.com/

Achtung:

- iOS kümmert sich selbst um die Interpretation einer Berührung des Bildschirms und der Transformation in ein `UITouch`-Objekt
- Kein eigener Code an dieser Stelle notwendig!

UIResponder-Schnittstelle

Klassen, deren Instanzen über Events informiert werden wollen, müssen die `UIResponder`-Schnittstelle implementieren

- `UIResponder` ist die Elternklasse von
  - `UIView`, `UIViewController`, `UIControl`, `UIApplication` und `UIWindow`

- `UIResponder` bietet Methoden zur Behandlung von Touches an
  - `touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event`
    (Finger "trifft" auf den Bildschirm)
  - `touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event`
    (Finger bewegt sich auf dem Bildschirm)
  - `touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event`
    (Finger verlässt den Bildschirm)
  - `touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event`
    Wenn eine Multitouch-Sequenz durch ein System-Event abgebrochen wird (z.B. eingehender Anruf)

Verwenden der UIResponder-Schnittstelle

Jede der Methoden entspricht einer der **Touch-Phasen**

- Began, Moved, Ended bzw. Cancelled

Treten für eine dieser Phasen **neue bzw. veränderte** `UITouch`-Objekte auf, wird die entsprechende `touch`-Methode aufgerufen

`touch`-Methoden Parameter:

- `NSSet`:
  - Ein `NSSet`-Objekt, das alle **neuen bzw. veränderten** `UITouch`-Objekte der entsprechenden Phase enthält
- `UIEvent`:
  - Ein `UIEvent`-Objekt, das **alle** `UITouch`-Objekte der entsprechenden Multitouch-Sequenz enthält

Verwenden der UIResponder-Schnittstelle

Beispiel:

- Ein `UITouch`-Objekt `obj` wechselt von der Began- in die Moved-Phase

→ Aufruf von `touchesMoved:withEvent`:

  - `NSSet touches` beinhaltet nun dieses `UITouch`-Objekt `obj` und alle anderen `UITouch`-Objekte, die sich ebenfalls in der Moved-Phase befinden
  - `UIEvent event` beinhaltet **alle** `UITouch`-Objekte der Multitouch-Sequenz

- Es gilt also:
  - `event.allTouches.count >= touches.count`

Verwenden der UIResponder-Schnittstelle

Ein paar weitere Hinweise:

- Entfernt sich der Finger vom Bildschirm, wird das zugehörige `UITouch`-Objekt ein letztes Mal aktualisiert.
- `touchesEnded:withEvent:` `NSSet` enthält das `UITouch`-Objekt, dessen dazugehöriger Finger den Bildschirm nun **nicht mehr** berührt.
- Nach Ausführung von `touchesEnded:withEvent:` werden die übergebenen `UITouch`-Objekte gelöscht.
- Starten mehrere Berührungen gleichzeitig auf demselben `UIView`-Objekt, enthält das `NSSet` beim Aufruf von `touchesBegan:withEvent:` mehrere Elemente. (Das Zeitfenster für Gleichzeitigkeit ist sehr klein…)

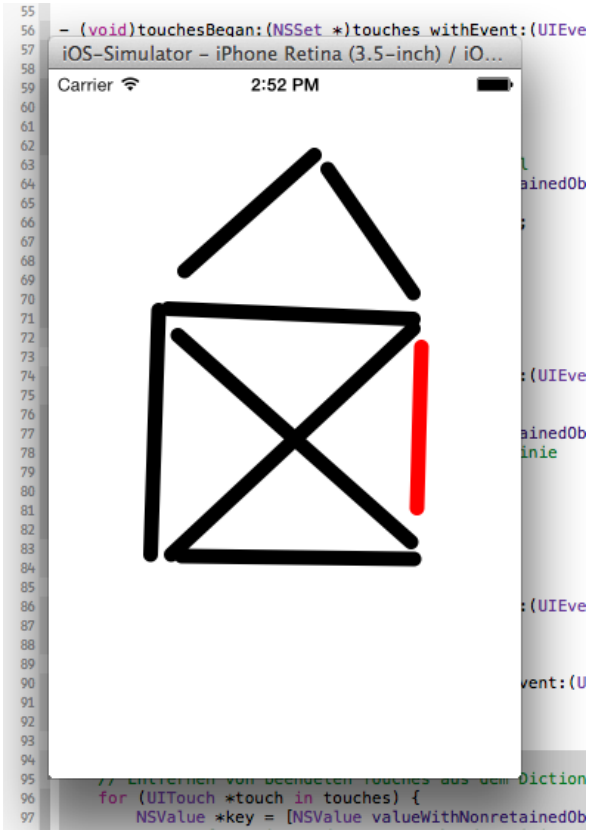
Viele weitere wichtige Details im *"Event Handling Guide for iOS"*

Voraussetzungen zur Behandlung von Touch-Events durch eine View

- Die entsprechende Klasse muss von (einer Unterklasse von) `UIResponder` erben
- Die entsprechende Klasse muss die `UIResponder`-Schnittstelle (`touch`-Methoden) implementieren
- Die Property `userInteractionEnabled` muss auf `YES` bzw. `true` gesetzt sein
- Die mit dem `UIResponder`-Objekt assoziierte View (View selbst bzw. das assoziierte `UIViewController`-Objekt) muss sichtbar sein (nicht `transparent`; nicht `hidden`)

Beispiel: DrawLines

Einfaches Zeichenprogramm, das das parallele Zeichnen von Linien erlaubt (Beispiel basiert auf *"The Big Nerd Ranch Guide"*)



Beispiel: DrawLines

Vorgehensweise

- Erzeugen eines neuen Projekts "DrawLines" (Template: Single View Application)
- Erzeugen einer Klasse `Line` zur Speicherung von Linien (erbt von `NSObject`)
- Erzeugen einer Klasse `DrawLinesView` zur Anzeige der Zeichnung (erbt von `UIView`)

Achtung:

- Dieses Beispiel dient nur zur Verdeutlichung des Touch-Konzepts (keine saubere Trennung zwischen Modell und View)

Beispiel: DrawLines

Implementierung der Klasse `Line`

```
// Line.h

#import <Foundation/Foundation.h>

@interface Line : NSObject
@property (nonatomic) CGPoint begin;
@property (nonatomic) CGPoint end;
@end
```

```
// Line.m

#import "Line.h"

@implementation Line

@end
```

Beispiel: DrawLines

Hinzufügen benötigter Datenstrukturen zur Klasse `DrawLinesView`

```
// DrawLinesView.h

#import <UIKit/UIKit.h>

@interface DrawLinesView : UIView

// zur Verwaltung "unfertiger" Linien
@property NSMutableDictionary *pendingLines;

// zur Speicherung gezeichneter Linien
@property NSMutableArray *finishedLines;

// zum Löschen aller Linien
-(void)clearAll;
@end
```

Beispiel: DrawLines

Initialisierung der Datenstrukturen und Konfiguration der View

```
// DrawLinesView.m

-(instancetype)initWithFrame:(CGRect)frame {
    self = [super initWithFrame:frame];
    if (self) {
        self.pendingLines = [NSMutableDictionary new];
        self.finishedLines = [NSMutableArray new];
        self.backgroundColor = [UIColor whiteColor];
        // Standard ist NO (d.h. alle Touches außer dem Ersten würden
        // ignoriert
        self.multipleTouchEnabled = YES;
    }
    return self;
}
```

Beispiel: DrawLines

Implementierung der `drawRect`-Methode von `DrawLinesView`

```
// DrawLinesView.m
-(void)drawRect:(CGRect)rect {
    CGContextRef ctx = UIGraphicsGetCurrentContext();
    CGContextSetLineWidth(ctx, 10.0);
    CGContextSetLineCap(ctx, kCGLineCapRound);
    // fertige Linien werden schwarz
    [[UIColor blackColor] set];
    for(Line *line in self.finishedLines) {
        CGContextMoveToPoint(ctx, line.begin.x, line.begin.y);
        CGContextAddLineToPoint(ctx, line.end.x, line.end.y);
        CGContextStrokePath(ctx);
    }
    // "unfertige" Linien werden rot
    [[UIColor redColor] set];
    for(NSValue *v in self.pendingLines) {
        Line *line = [self.pendingLines objectForKey:v];
        CGContextMoveToPoint(ctx, line.begin.x, line.begin.y);
        CGContextAddLineToPoint(ctx, line.end.x, line.end.y);
        CGContextStrokePath(ctx);
    }
}
```

Beispiel: DrawLines

Implementierung der `clearAll`-Methode von `DrawLinesView`

```
// DrawLinesView.m

- (void)clearAll {
    [self.pendingLines removeAllObjects];
    [self.finishedLines removeAllObjects];
    [self setNeedsDisplay];
}
```

Beispiel: DrawLines

Implementierung von `touchesBegan:withEvent`

```
// DrawLinesView.m
-(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
    for (UITouch *touch in touches) {
        // Double Tap soll alles löschen
        if(touch.tapCount > 1) {
            [self clearAll];
            return;
        }
        // Verwende (schwache) Referenz auf UITouch-Objekt als Schlüssel
        NSValue *key = [NSValue valueWithNonretainedObject:touch];
        // Erzeugen einer Linie
        CGPoint p = [touch locationInView:self];
        Line *newLine = [Line new];
        newLine.begin = p;
        newLine.end = p;
        self.pendingLines[key] = newLine;
    }
}
```

Beispiel: DrawLines

Implementierung von `touchesMoved:withEvent`

```
// DrawLinesView.m

-(void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event {
    // Update der ausstehenden Linien
    for (UITouch *touch in touches) {
        // Verwende (schwache) Referenz auf UITouch-Objekt als Schlüssel
        NSValue *key = [NSValue valueWithNonretainedObject:touch];
        // Dereferenzieren der entsprechenden Linie
        Line *line = self.pendingLines[key];
        // Update des Endpunkts
        line.end = [touch locationInView:self];
    }
    [self setNeedsDisplay];
}
```

Beispiel: DrawLines

Eine Linie soll fertiggestellt werden, wenn

- der Finger sich vom Display bewegt (`touchesEnded:withEvent:`)
- ein System-Event die Anwendung unterbricht (`touchesCancelled:withEvent:`)

Hier werden zur Vereinfachung in beiden Fällen die Linie fertigstellen

```
// DrawLinesView.m

-(void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event {
    [self endTouches:touches];
}

-(void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event {
    [self endTouches:touches];
}
```

Beispiel: DrawLines

Implementierung von endTouches:touches

```
// DrawLinesView.m

-(void)endTouches:(NSSet*)touches {
    // Entfernen von beendeten Touches aus dem Dictionary
    for (UITouch *touch in touches) {
        // Verwende (schwache) Referenz auf UITouch-Objekt als Schlüssel
        NSValue *key = [NSValue valueWithNonretainedObject:touch];
        // Dereferenzieren der entsprechenden Linie
        Line *line = self.pendingLines[key];
        // falls ein Double Tap vorliegt, ist line == nil!
        if(line) {
            [self.finishedLines addObject:line];
            [self.pendingLines removeObjectForKey:key];
        }
    }
    [self setNeedsDisplay];
}
```

Beispiel: DrawLines

Initialisierung der View im View Controller

```
// DrawLinesView.m

#import "DrawLinesViewController.h"
#import "DrawLinesView.h"

@implementation DrawLinesViewController

-(void)loadView {
    self.view = [[DrawLinesView alloc] initWithFrame:CGRectZero];
}

@end
```

Beispiel: DrawLines

Anmerkung zur Klasse `NSValue`:

- `NSValue`-Objekte dienen als einfache Container für einzelne C bzw. Objective-C Datentypen.

- Erlaubt sind
  - Skalare Typen (z.B. int, float, und char)
  - Pointer
  - Strukturen
  - Objektreferenzen

- `NSValue`-Instanzen dienen u.a. dazu, solche Datentypen z.B. Instanzen der Klassen `NSArray` oder `NSSet` hinzuzufügen (Elemente solcher Collections müssen Objekte sein!)

- `NSValue`-Objekte sind stets unveränderlich (immutable)

Beispiel: DrawLines

Anmerkung zur Klasse `NSValue`:

- `+(NSValue *)valueWithNonretainedObject:(id)anObject`
  - Die Methode liefert eine Instanz vom Typ `NSValue`, die das Objekt `anObject` beinhaltet.
  - Großer Vorteil: Es wird dabei eine **schwache Referenz** auf `anObject` angelegt!
  - `anObject` kann somit als Schlüssel innerhalb des Dictionarys verwendet werden, ohne dass die Collection das Objekt besitzt

Touch Events vs. Gesture Recognizer

Wann werden von wem welche Touch-Events behandelt?

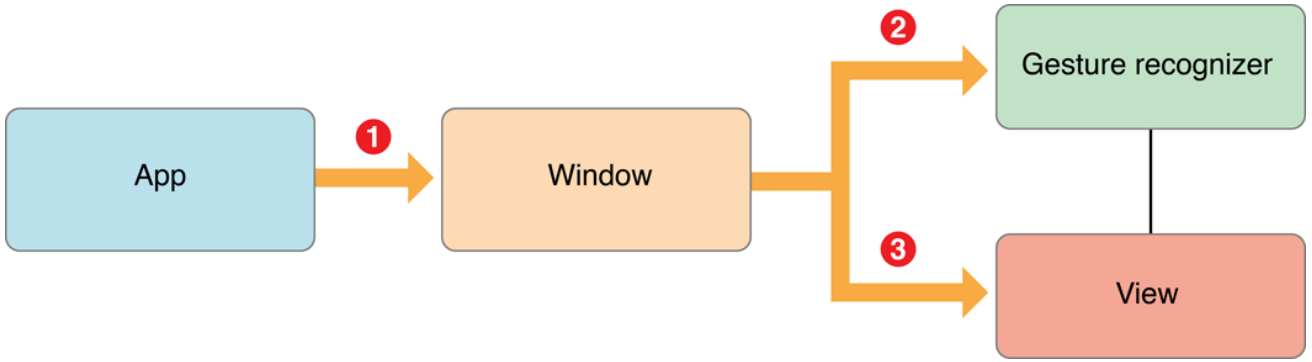
Man kann Touch-Events über Gesture Recognizer oder direkt über den entsprechenden Responder behandeln

Was wenn beide Mechanismen Anwendung finden?

Wer wird zuerst über Touches informiert?

Auslieferung von Touch-Events

Standardpfad der Auslieferung von Touch-Events



Quelle: http://www.apple.com/

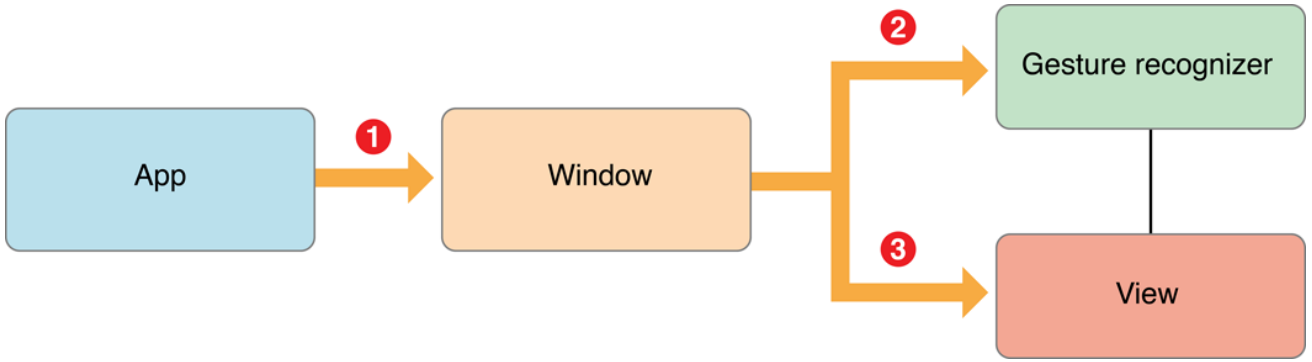
`UITouch`-Objekte werden zuerst an die registrierten Gesture Recognizer einer View (Superview) gesendet

Erst dann gelangen sie zur View selbst!

Die Auslieferung von `UITouch`-Objekten an die View wird dazu von der `UIWindow`-Instanz verzögert
→ Recognizer können Touch zuerst auswerten

Auslieferung von Touch-Events

Standardpfad der Auslieferung von Touch-Events



Quelle: http://www.apple.com/

Wird eine Geste erkannt, werden keine (weiteren) `UITouch`-Objekte an die View gesendet

Etwaige vorhergehende Touch-Events, die die View schon erhalten hat, werden als ungültig gekennzeichnet

Standard-Auslieferung von Touch-Events

Beispiel:

- Diskrete Geste, die sich durch einen Zwei-Finger-Tap definiert

Zeit

Reihenfolge

	❶	❷	❸	❹
Gesture Recognizer	touchesBegan:	touchesMoved:	touchesEnded:	touchesEnded:
tapGestureRecognizer.state	Possible	Possible	Possible	Recognized
View	touchesBegan:	touchesMoved:		touchesCancelled:

Quelle: http://www.apple.com/

1. `UIWindow`-Instanz sendet zwei `UITouch`-Objekte an Gesture Recognizer (GR). GR kann noch keine Geste erkennen → Weiterleitung der `UITouch`-Objekte an die View
2. `UIWindow`-Instanz sendet zwei `UITouch`-Objekte an GR. GR kann immer noch keine Geste erkennen → Weiterleitung der `UITouch`-Objekte an die View
3. `UIWindow`-Instanz sendet ein `UITouch`-Objekt an GR. GR hat noch nicht genug Info, um Geste komplett zu erkennen, **ABER** `UIWindow`-Instanz leitet `UITouch`-Objekte **NICHT** an die View weiter
4. `UIWindow`-Instanz sendet zweites `UITouch`-Objekt an GR. GR erkennt die Geste → View erklärt vorhergehende Touches als ungültig.
   (Wenn Gestenerkennung hier fehlschlägt, werden die beiden letzten `UITouch`-Objekt an die View weitergereicht)

Einflussnahme auf die Auslieferung von Touch-Events

Einflussnahme auf die Abhandlung von Touch-Events erfolgt über Anpassung entsprechender Properties in den registrierten Gesture Recognizern

`delaysTouchesBegan`

- Falls `YES`: Verhindert die Auslieferung von `UITouch`-Objekten in der Begin-Phase. Stellt sicher, dass keinerlei UITouch-Objekt vor Erkennung einer Geste an die View gesendet wird (default: `NO`)
- Kann zu schlechter Antwortzeit der UI führen!

`delaysTouchesEnded`

- Falls `YES`: Stellt sicher, dass die View keine Actions ausführt, die ein GR später u.U. noch abbrechen muss.
- Falls `NO`: `UITouch`-Objekten in der Ended-Phase werden direkt an die View weitergereicht und parallel mit dem Gesture Recognizer verarbeitet (default: `YES`)

Einflussnahme auf die Auslieferung von Touch-Events

Beispiel zu `delaysTouchesEnded`:

- Erkennen eines Double-Taps im First-Responder bei gleichzeitig installiertem `UITapGestureRecognizer` (`numberOfTapsRequired` == 2)

- Reihenfolge eingehender Nachrichten bei der First-Responder-View

Reihenfolge

`delaysTouchesEnded == NO`	`delaysTouchesEnded == YES (default)`
`touchesBegan:withEvent:`	`touchesBegan:withEvent:`
`touchesEnded:withEvent:`	**`touchesBegan:withEvent:`**
`touchesBegan:withEvent:`	`touchesCancelled:withEvent:`
`touchesCancelled:withEvent:`	`touchesCancelled:withEvent:`

Potentielles Erkenne des Double-Tap möglich

Double-Tap wird vom First-Responder erkannt

Double-Tap kann vom First-Responder nicht mehr erkannt werden

Ignorieren von `UITouch`-Objekten mit `ignoreTouch:forEvent:`

- Z.B. falls Gesture Recognizer einen Touch analysiert, der nicht Teil seiner Geste ist

→ Aufruf (auf sich selbst) bewirkt direkte Weitergabe des `UITouch`-Objekts an die View

Beispiel: Erzeugen eines eigenen Gesture Recognizers

Der Checkmark Gesture Recognizer

- Beispiel basiert auf adaptiertem Quellcode von: http://www.apple.com/
- Das Zeichnen eines Checkmarks (Häkchen) soll als Geste erkannt werden
- Implementierung als diskrete Geste (Zustände: Possible, Failed, Ended == Recognized)

Beispiel: Erzeugen eines eigenen Gesture Recognizers

Vorgehensweise

- Erzeugen einer Klasse, die von `UIGestureRecognizer` erbt
- Importieren des Headers `UIGestureRecognizerSubclass.h`
- Überschreiben der Methoden
  - `-(void)reset;`
  - `-(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event;`
  - `-(void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event;`
  - `-(void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event;`
  - `-(void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event;`

Beispiel: Erzeugen eines eigenen Gesture Recognizers


```
// CheckmarkGestureRecognizer.h

#import <UIKit/UIKit.h>
#import <UIKit/UIGestureRecognizerSubclass.h>

@interface CheckmarkGestureRecognizer : UIGestureRecognizer
-(void)reset;
-(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event;
-(void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event;
-(void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event;
-(void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event;
@end
```

Beispiel: Erzeugen eines eigenen Gesture Recognizers


```
// CheckmarkGestureRecognizer.m

#import "CheckmarkGestureRecognizer.h"

@interface CheckmarkGestureRecognizer()
@property (nonatomic) BOOL strokeUp;
@property (nonatomic) CGPoint midPoint;
@end

@implementation CheckmarkGestureRecognizer

-(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
    [super touchesBegan:touches withEvent:event];
    if ([touches count] != 1) {
        self.state = UIGestureRecognizerStateFailed;
        return;
    }
    // Setze den Mittelpunkt auf den Startpunkt
    self.midPoint = [touches.anyObject locationInView:self.view];
}
```

Beispiel: Erzeugen eines eigenen Gesture Recognizers


```
-(void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event {
    [super touchesMoved:touches withEvent:event];
    if (self.state == UIGestureRecognizerStateFailed) return;
    CGPoint nowPoint = [touches.anyObject locationInView:self.view];
    CGPoint prevPoint = [touches.anyObject previousLocationInView:self.view];
    // Sind wir noch auf dem Weg nach rechts unten?
    if (!self.strokeUp) {
        // Setze den Mittelpunkt, falls noch auf dem Weg nach rechts unten
        if (nowPoint.x >= prevPoint.x && nowPoint.y >= prevPoint.y) {
            self.midPoint = nowPoint;
        }
        // Ab jetzt interessiert nur noch das Ende der Geste
        else if (nowPoint.x >= prevPoint.x && nowPoint.y < prevPoint.y) {
            self.strokeUp = YES;
        }
        else {
            self.state = UIGestureRecognizerStateFailed;
        }
    }
}
```

Beispiel: Erzeugen eines eigenen Gesture Recognizers


```
-(void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event {
    [super touchesEnded:touches withEvent:event];
    if ((self.state == UIGestureRecognizerStatePossible) && self.strokeUp) {
        CGPoint nowPoint = [touches.anyObject locationInView:self.view];
        // Liegt der Endpunkt rechts oberhalb des Mittelpunkts?
        if(nowPoint.x >= self.midPoint.x && nowPoint.y < self.midPoint.y) {
            self.state = UIGestureRecognizerStateRecognized;
        }
        else {
            self.state = UIGestureRecognizerStateFailed;
        }
    }
}
```

Beispiel: Erzeugen eines eigenen Gesture Recognizers


```
-(void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event {
    [super touchesCancelled:touches withEvent:event];
    self.midPoint = CGPointZero;
    self.strokeUp = NO;
    self.state = UIGestureRecognizerStateFailed;
}

-(void)reset {
    [super reset];
    self.midPoint = CGPointZero;
    self.strokeUp = NO;
}
@end
```

Beispiel: Erzeugen eines eigenen Gesture Recognizers

Ergebnis:

Responder-Chain

Responder-Chain

Die Verarbeitung von Touch-Events erfolgt in der Responder-Chain

- Touches können an unterschiedlichen Stellen auf dem Screen erfolgen
- Bei der Verwendung mehrerer (Sub-)Views muss entschieden werden, welche dieser Views (bzw. welcher View Controller) ein entsprechendes Event behandeln soll
- UIKit erzeugt ein `UIEvent`-Objekt, wenn ein durch den Nutzer initiiertes Ereignis (z.B. Touch) erfolgt
- Das Event wird in die Event-Queue der gerade aktiven Anwendung gestellt
- Touch-Events werden als `UIEvent`-Objekt "verpackt"

Responder-Chain

Events nehmen einen spezifischen Pfad, bis sie zu dem Objekt gelangen, das das Event behandeln kann

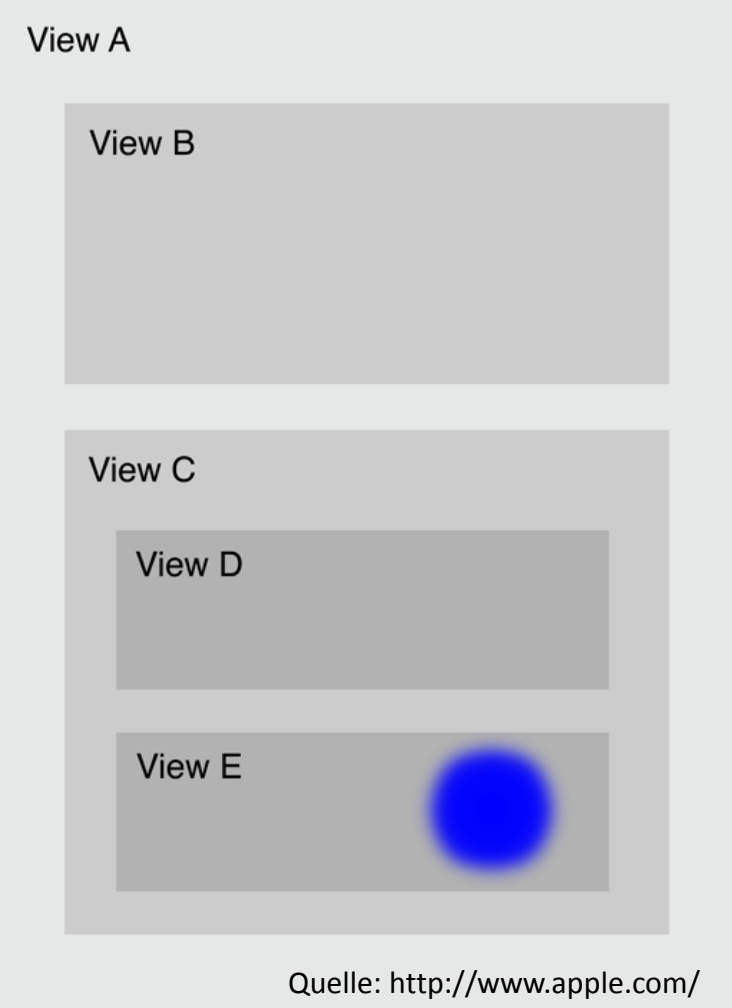
Ablauf

- Das Singelton `UIApplication`-Objekt nimmt das nächste Event aus der Event-Queue
- Das Event wir (i.d.R.) an das `UIWindow`-Objekt der Anwendung gesendet
- Der nächste Empfänger hängt vom Typ des Events ab
  - Touch-Events: Es wird versucht, das Event an diejenige View weiterzuleiten, in der das Event auftrat (**Hit-Test-View**)
  - Motion und Remote Control Events: Das Event wird an den **First Responder** gesendet (siehe dazu: *"Event Handling Guide for iOS"*)
- Der Pfad dient dazu, ein Objekt zu finden, das das Event behandeln und darauf reagieren kann

Responder-Chain

Hit-Testing

- Rekursive Suche nach der kleinsten Subview, in der das Event auftrat
- Kann die entsprechende View das Event nicht verarbeiten, wird es über die **Responder-Chain** weitergereicht, bis ein Objekt gefunden wird, das das Event behandeln kann
- Hier: A → C → E (Hit-Test-View)



Quelle: http://www.apple.com/

Responder-Chain

Responder-Chain ist eine Kette miteinander verbundener `UIResponder`-Objekte

Die Kette

- startet mit dem First Responder (z.B. `UIView`, die durch Hit-Testing identifiziert wurde)
- endet beim `UIApplication`-Objekt

Alle Responder erben von `UIResponder` (nicht nur UI-spezifische!)

Responder-Chain

Wie wird man Responder?

- Überschreiben der Methode `canBecomeFirstResponder`, so dass sie `YES` zurückliefert

- Empfangen einer `becomeFirstResponder` Nachricht (u.U. muss man sich die Nachricht selber schicken)
  - Achtung: Das Senden sollte erst erfolgen, wenn der gesamte Objektgraph der Anwendung existiert
  - Typischerweise wird die Nachricht in der Methode `viewDidAppear:` gesendet (nicht in `viewWillAppear`!)

Responder-Chain

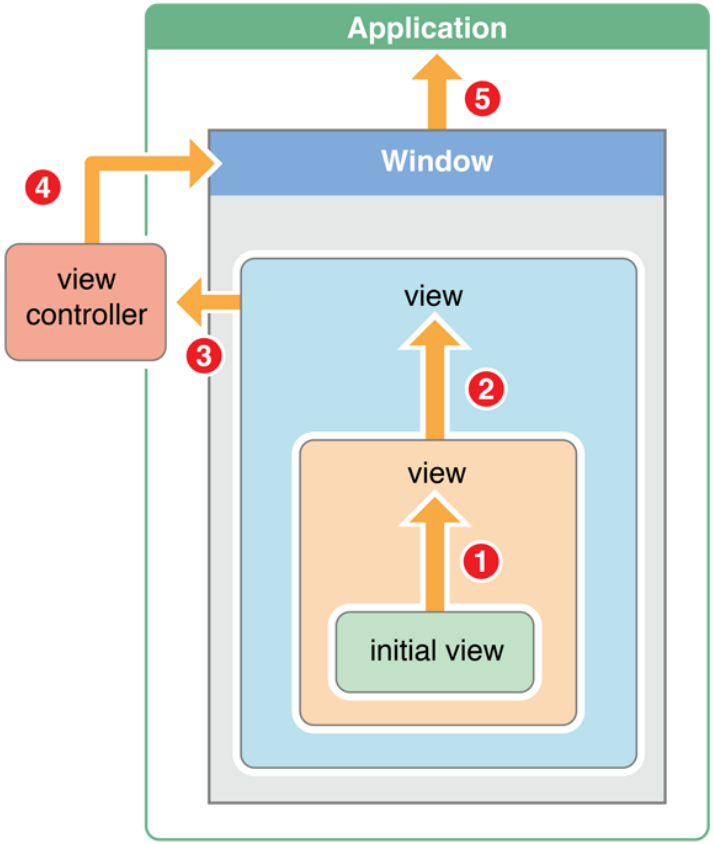
Wie entsteht die Verkettung?

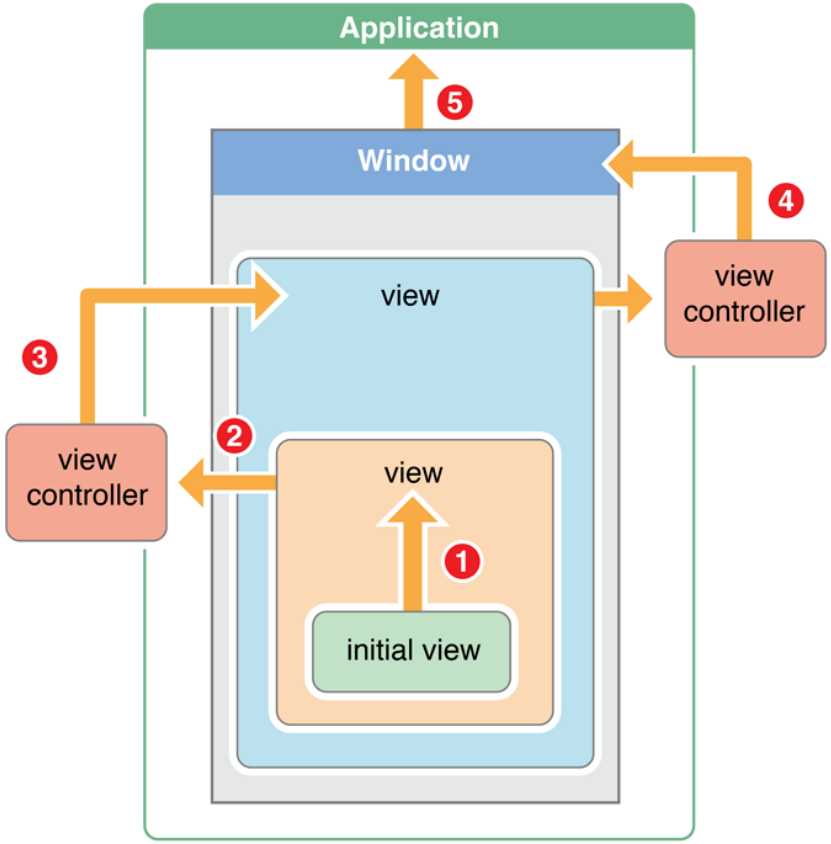
- `UIView`-Objekt gibt sein assoziiertes `UIViewController`-Objekt zurück
  - Wenn `nil` → Rückgabe der Superview
- `UIViewController`-Objekt gibt die Superview seines assoziierten View-Objekts zurück
- `UIWindow`-Objekt gibt das `UIApplication`-Objekt zurück
- `UIApplication`-Objekt → `nil`

Für eine abweichende Behandlung von Events muss man sich selbst um die Verkettung der Responder kümmern

- Dazu: Überschreiben der Methode `nextResponder`
- Achtung: Immer erst Aufruf der Implementierung der Elternklassen

Responder-Chain: Beispiele





Quelle: http://www.apple.com/

Responder-Chain: Beispiele

Eine View will Touches behandeln

- Wenn es sich um einen Double-Tap handelt, soll sich aber der assoziierte View Controller, die Superview, oder ein anderes nachfolgendes Objekt in der Responder-Chain darum kümmern

```
// MyCustomView.m

[...]
-(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
    for (UITouch *touch in touches) {
        if[touch.tapCount == 2] {
            [self.nextResponder touchesBegan:touches withEvent:event];
            return;
        }
    }
    // sonst weitere Behandlung hier...
}
[...]
```