



LUDWIG-
MAXIMILIANS-
UNIVERSITÄT
MÜNCHEN



Praktikum iOS-Entwicklung

Wintersemester 2016 / 2017

Prof. Dr. Linnhoff-Popien

Mirco Schönfeld, Carsten Hahn





LUDWIG-
MAXIMILIANS-
UNIVERSITÄT
MÜNCHEN



TOUCH-EVENTS

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?

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 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**)

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 **solte** 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

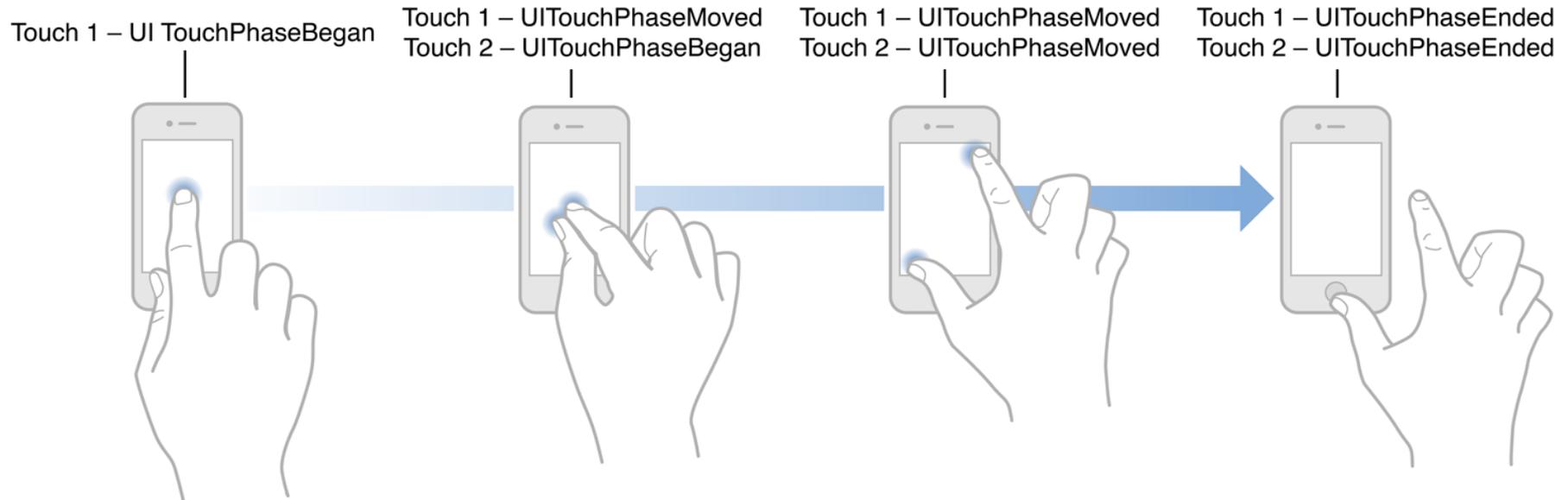
Für jeden Berührungspunkt (Finger) wird ein `UITouch`-Objekt mit folgenden Properties erzeugt

- `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
- Seit iOS 9 `force` und `maximumPossibleForce` für 3D Touch

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

Beispiel



Achtung:

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

- 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!

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)

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 **UITouch**-Objekte **der entsprechenden Phase** enthält
- **UIEvent**:
 - Ein **UIEvent**-Objekt, das alle **UITouch**-Objekte **der entsprechenden Multitouch-Sequenz** enthält

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`

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...)

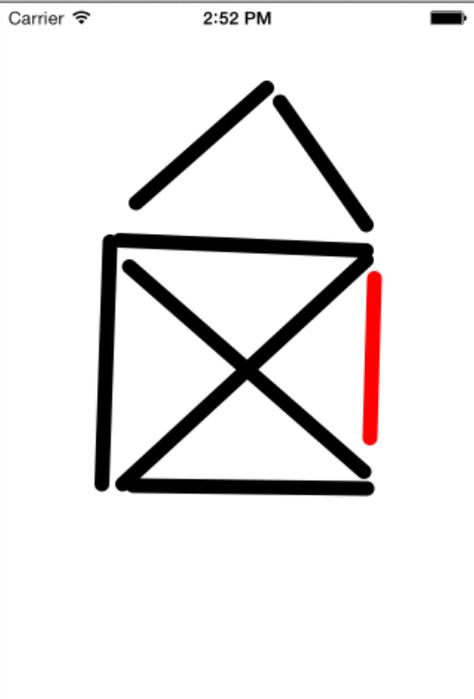
Viele weitere wichtige Details im *"Event Handling Guide for iOS"*

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**)

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

```
55  
56 - (void)touchesBegan:(NSSet *)touches withEvent:(UIEve  
57 iOS-Simulator - iPhone Retina (3.5-inch) / iO...  
58  
59 Carrier 2:52 PM  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95 // Erhalten von bearbeiteten touches aus dem Diction  
96 for (UITouch *touch in touches) {  
97     NSValue *key = [NSValue valueWithNonretainedOb
```



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)

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
```

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
```

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;
}
```

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);
    }
}
```

Implementierung der `clearAll`-Methode von `DrawLinesView`

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

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;
    }
}
```

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 valueForKeyWithNonretainedObject:touch];
        // Dereferenzieren der entsprechenden Linie
        Line *line = self.pendingLines[key];
        // Update des Endpunkts
        line.end = [touch locationInView:self];
    }
    [self setNeedsDisplay];
}
```

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];
}
```

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];
}
```

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
```

Anmerkung zur Klasse `NSArray`:

- `NSArray`-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
- `NSArray`-Instanzen dienen u.a. dazu, solche Datentypen z.B. Instanzen der Klassen `NSString` oder `NSNumber` hinzuzufügen (Elemente solcher Collections müssen Objekte sein!)
- `NSArray`-Objekte sind stets unveränderlich (immutable)

Anmerkung zur Klasse `NSUserDefaults`:

- `+(NSUserDefaults *)valueWithNonretainedObject:(id)anObject`
 - Die Methode liefert eine Instanz vom Typ `NSUserDefaults`, 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

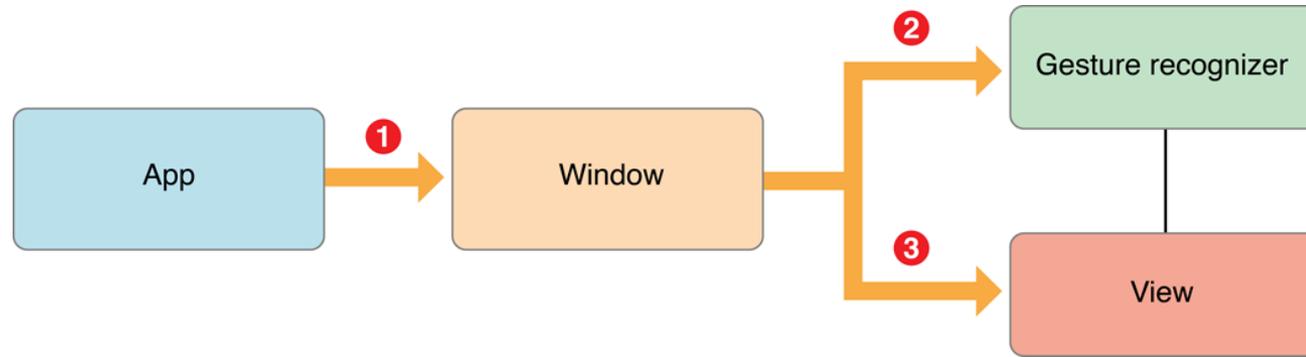
Wann werden von wem welche Touch-Events behandelt?

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?

Standardpfad der Auslieferung von Touch-Events



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

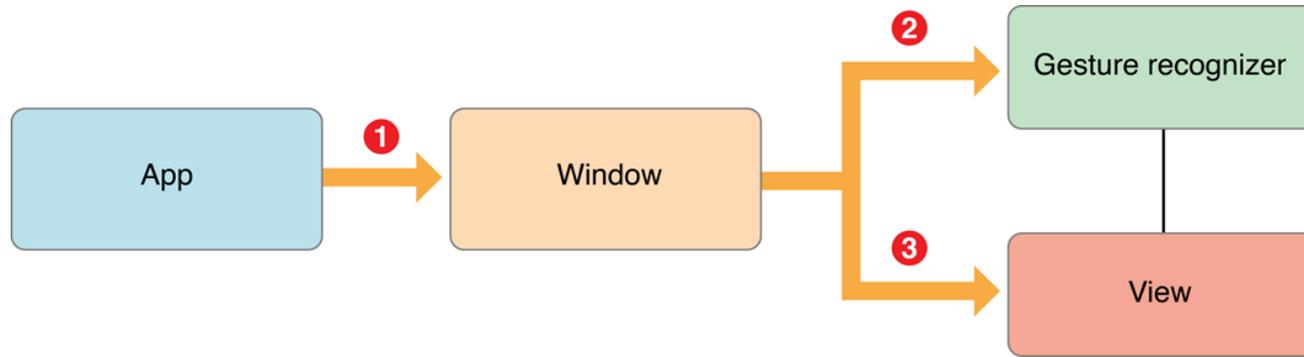
UITouch-Objekte werden zuerst an die registrierten Gesture Recognizer einer View (Superview) gesendet

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

Standardpfad der Auslieferung von Touch-Events



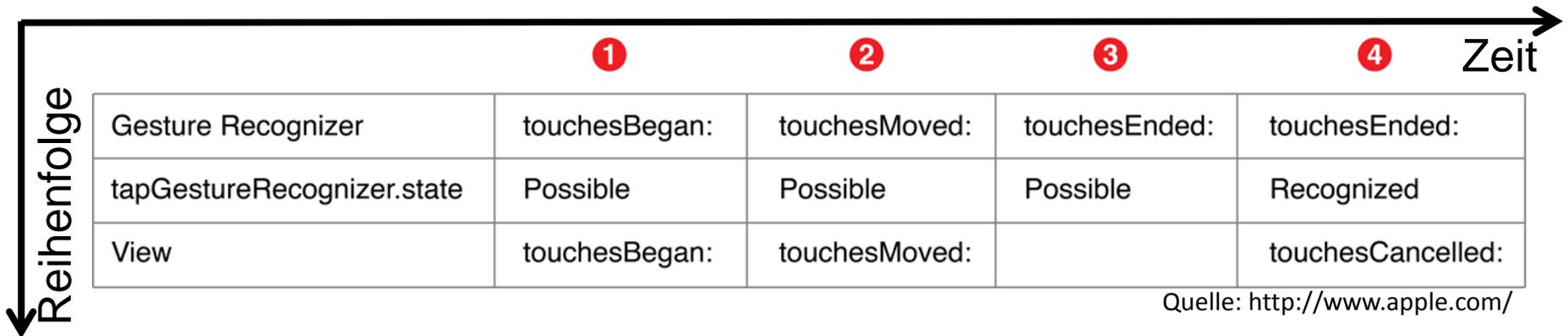
Quelle: <http://www.apple.com/>

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

Beispiel:

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



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

delaysTouchesBegan

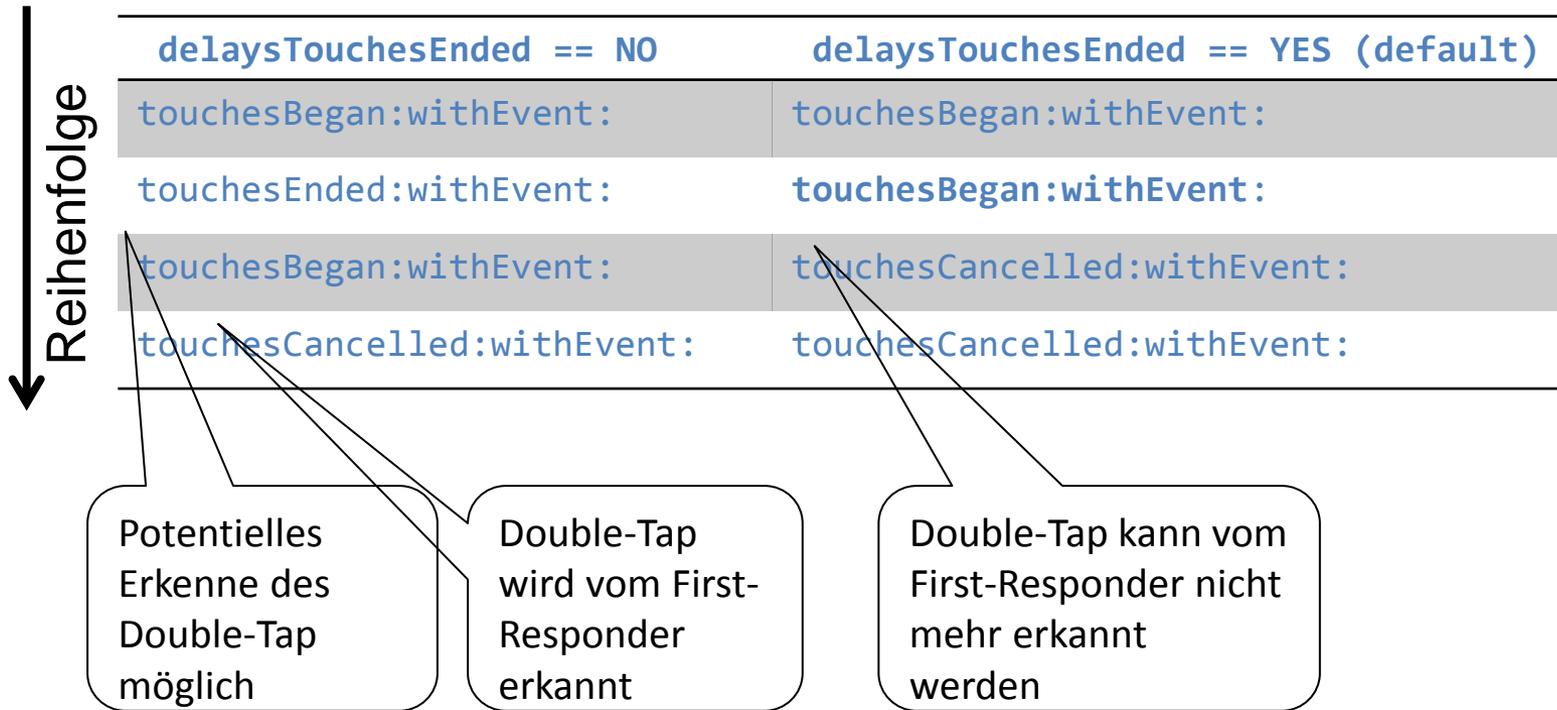
- Falls **YES**: Verhindert die Auslieferung von **UITouch**-Objekten in der Begin-Phase. Stellt sicher, dass keinerlei UITouch-Objekt vor Erkennung einer Geste an die View gesendet wird (default: **NO**)
- Kann zu schlechter Antwortzeit der UI führen!

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**)

Beispiel zu `delaysTouchesEnded`:

- Erkennen eines Double-Taps im First-Responder bei gleichzeitig installiertem `UITapGestureRecognizer` (`numberOfTapsRequired == 2`)
- Reihenfolge eingehender Nachrichten bei der First-Responder-View



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

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)



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;`

```
// CheckmarkGestureRecognizer.h
```

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

```
#import <UIKit/UITapGestureRecognizerSubclass.h>
```

```
@interface CheckmarkGestureRecognizer : UITapGestureRecognizer
```

```
-(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
```

```
// 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];
}
```

```
-(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;
        }
    }
}
```

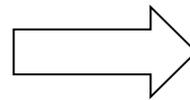
```
-(void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event {
    [super touchesEnded:touches withEvent:event];
    if ((self.state == UIGestureRecognizerStatePossible) && self.strokeUp) {
        CGPoint nowPoint = [touches.anyObject locationInView:self.view];
        // Liegt der Endpunkt rechts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

```
-(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

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"

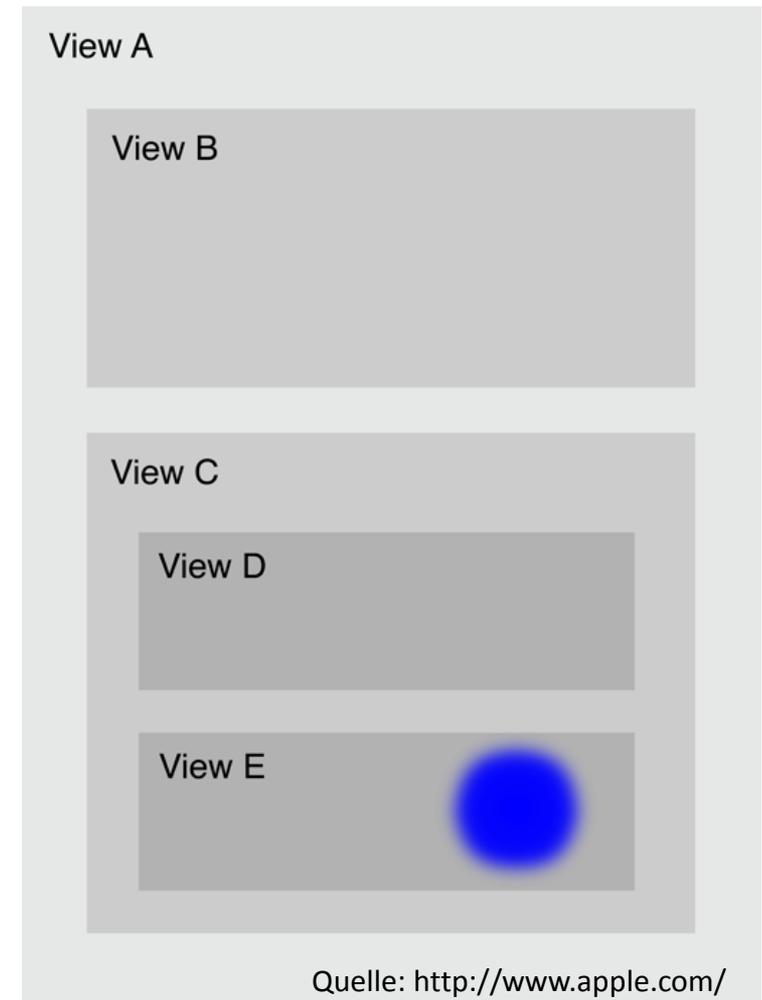
Events nehmen einen spezifischen Pfad, bis sie zu dem Objekt gelangen, das das Event behandeln kann

Ablauf

- Das Singleton **UIApplication**-Objekt nimmt das nächste Event aus der Event-Queue
- Das Event wird (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

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)



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!)

Wie wird man Responder?

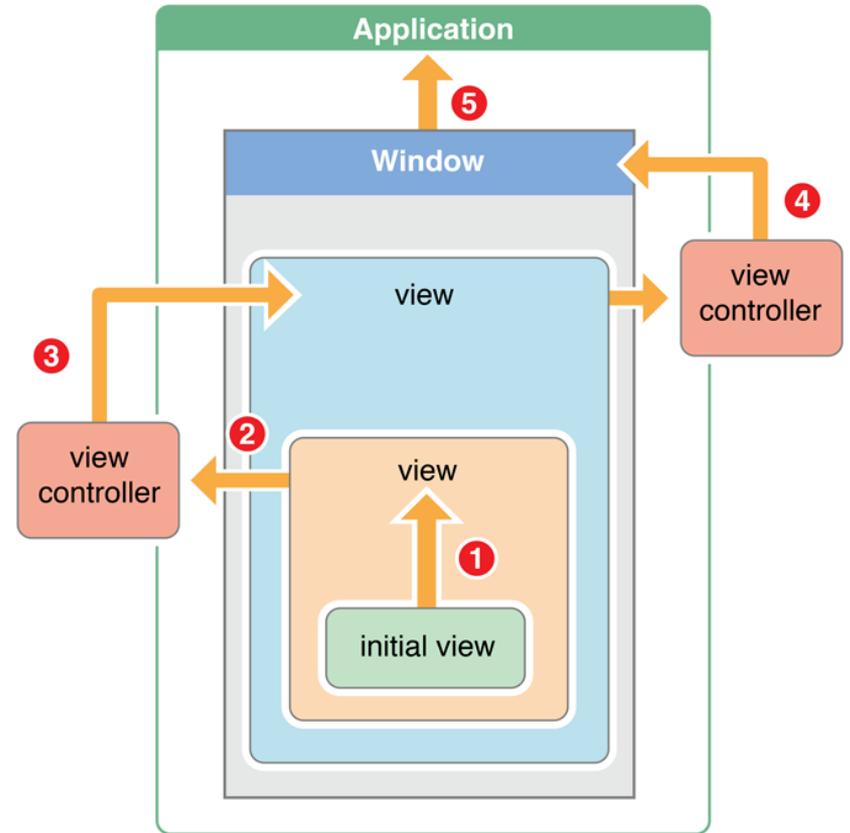
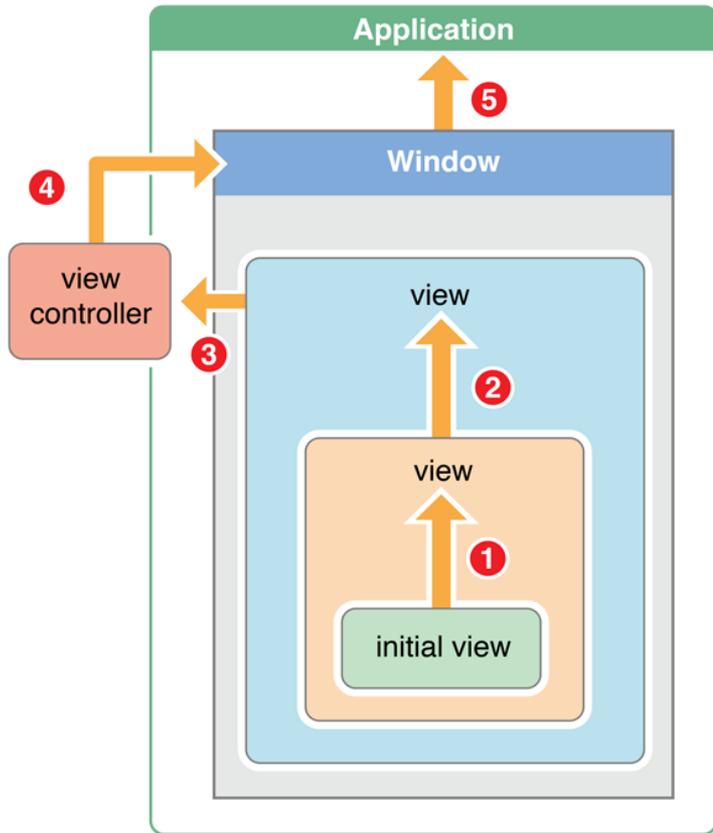
- Überschreiben der Methode `canBecomeFirstResponder`, so dass sie **YES** zurückliefert
- Empfangen einer `becomeFirstResponder` Nachricht (u.U. muss man sich die Nachricht selber schicken)
 - Achtung: Das Senden sollte erst erfolgen, wenn der gesamte Objektgraph der Anwendung existiert
 - Typischerweise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

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



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

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...  
}  
[...]
```