CG Core Graphic

- Text zeichnen
  - UILabel benutzen wann immer möglich
  - sonst 1. einen Font besorgen:

```
UIFont *myFont = [UIFont systemFontOfSize:12.0];
// oder
UIFont *theFont = [UIFont
    fontWithName:@"Helvetica" size:36.0];
// oder
NSArray *availableFonts = [UIFont familyNames];
// und einen raussuchen
```

CG Core Graphic

- dann 2.

```
NSString *text = @"toDraw";
[text drawAtPoint:(CGPoint)p withFont:theFont];
// NSString instance method ?!
```

- wie groß wird der Text bei zeichnen?

```
CGSize textSize = [text sizeWithFont:myFont];
// NSString instance method ?!
```

- UI Methoden schon in NSString (ist doch wohl UI-unabhängig) implementiert?

CG Core Graphic

- NEIN: in Objective-C gibt es sog. Categories, die Klassen nachträglich (nur um Methoden) erweitern können
- also nicht in `NSString`, sondern in UIKit implementiert:

```
// so in UIStringDrawing.h:
@interface NSString (UIStringDrawing)
- (CGsize) sizeWithFont(UIFont *)font
- (CGSize) drawATPoint(CGPoint) point; ...
@end
```

- Auch üblich: Kategorie in einem .h/m-File namens `Klasse+Kategorie`
- stackoverflow answered **May 12 at 13:25 by** Barry Wark:

> - *Categories are not named in the sense that your code will refer to them by name like a Protocol. The category name in this case is Xcode's (poor) translation of the file name `NSObject+SBJSON.h` to a category name* [`@interface NSObject (NSObject_SBJSON);`] *for some reason Xcode or Objective-C's compiler doesn't want the + in the category name.*
>
> *The only time the category name is used is to match interface and implementations.*

CG Core Graphic

- Bilder zeichnen
  - UIImageView benutzen wann immer möglich
    - (id)initWithImage:(UIImage *)image
    - (id)initWithImage:(UIImage *)image
      highlightedImage:(UIImage *)highlightedImage
- sonst
  - ein UIImage erzeugen

```
UIImage *image = [UIImage imageNamed:@"foo.jpg"];
// im Resources-Folder von xcode oder
UIImage *image = [[UIImage alloc]
    initWithContentsOfFile:(NSString *)fullPath];
UIImage *image = [[UIImage alloc]
                  initWithData:(NSData *)imageData];
```

CG Core Graphic

- oder einen Kontext füllen und ein UIImage rausholen

```
UIGraphicsBeginImageContext(CGSize);
// push new context
// draw with CGContext functions ...
UIImage *image =
          UIGraphicsGetImageFromCurrentContext();
UIGraphicsEndImageContext(); // pop it
```

CG Core Graphic

- dann

```
[image drawAtPoint:(CGPoint)p];
// p is upper left corner of the image
[image drawInRect:(CGRect)r];
// scales the image to fit in r
[image drawAsPatternInRect:(CGRect)patRect;
// tiles the image into patRect
```

- aus einem UIImage kann man auch jpg/png extrahieren:

```
NSData *jpgData = UIImageJPEGRepresentation
                  (image, (CGFloat)quality);
NSData *pngData = UIImagePNGRepresentation
                  ((UIImage *)image);
```

UIKit

- Views und ViewController für diverse Standardaufgaben:
  - oft (zumindest teilweise)mit dem IB erzeugbar
  - Wiederverwendung von Code UND Look&Feel
- `UIImage`'s kann der `UIImageView` anzeigen:
  - DEMO: mit IB oder programmatisch - die Anatomie von **View based** vs. **Window based Application**

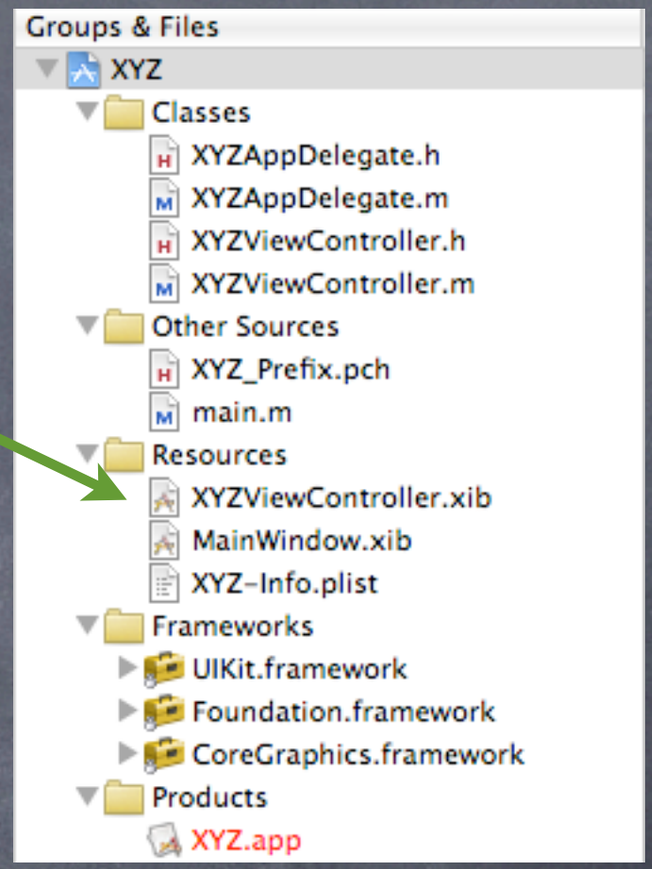
- **View based Application: XYZ**
  - `XYZViewController` leer, wird mit IB gefüllt
  - `XYZAppDelegate`

```
@class XYZViewController;


@interface XYZAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    XYZViewController *viewController;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet XYZViewController *viewController;

@end
```



aber wo?

UIKit

- View Based AppDelegate:

```
@implementation XYZAppDelegate

@synthesize window;
@synthesize viewController;

#pragma mark -
#pragma mark Application lifecycle

- (BOOL)          application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    // Override point for customization after application launch.
    // Add the view controller's view to the window and display.
    [self.window addSubview:viewController.view]; // muss wohl schon existieren
    [self.window makeKeyAndVisible];
    return YES;
}
- (void)applicationWillResignActive:(UIApplication *)application {/* leer */ }
- (void)applicationDidEnterBackground:(UIApplication *)application {/* leer */ }
- (void)applicationWillEnterForeground:(UIApplication *)application {/* leer */ }
- (void)applicationDidBecomeActive:(UIApplication *)application {/* leer */ }
- (void)applicationWillTerminate:(UIApplication *)application {/* leer */ }

#pragma mark -
#pragma mark Memory management

- (void)applicationDidReceiveMemoryWarning:(UIApplication *)application {/* leer */ }
- (void)dealloc { [viewController release]; [window release]; [super dealloc]; }


@end
```

UIKit

- aber wo wird der (Root-)ViewController erzeugt? alles magisch

  http://stackoverflow.com/questions/5772786/iphone-view-based-application-template-how-is-the-view-controller-xib-loaded

  - in `XYZ-info.plist`: `Main nib file base name : MainWindow`
  - in `MainWindow.nib`: Name des ViewControllers hinterlegt.
  - beim Laden dieses Nib-Files wird die Klasse dynamisch geladen

    `... [NSClassFromString(@“XYZViewController“) alloc ] init];`
  - aber der DI jedes ViewControllers ist:

```
- (id)initWithNibName:(NSString *)nibNameOrNil
                bundle:(NSBundle *)nibBundleOrNil;
```

`init` (CI) ruft `initWithNibName:nil bundle:nil`

XYZViewController.nib (xib)

BundleName aus XYZ-info.plist

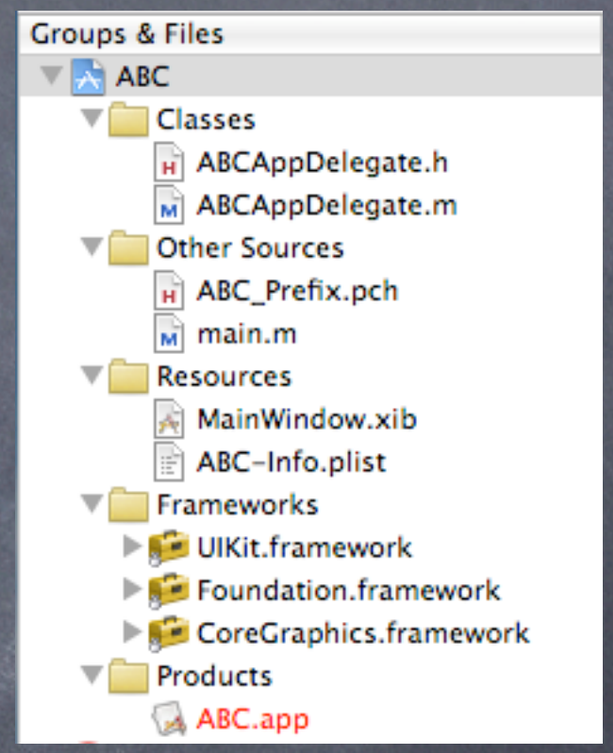
UIKit

- **Window based Application: ABC**
  - kein ABCViewController
  - ABCAppDelegate

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

@interface ABCAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
}
@property (nonatomic, retain) IBOutlet UIWindow *window;

@end
```



Window Based AppDelegate:

```
@implementation ABCAppDelegate

@synthesize window;

#pragma mark -
#pragma mark Application lifecycle

- (BOOL)          application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    // Override point for customization after application launch.
    // Add the view controller's view to the window and display.
    // NO VIEWController: build your own!
    [self.window makeKeyAndVisible];
    return YES;
}
- (void)applicationWillResignActive:(UIApplication *)application {/* leer */ }
- (void)applicationDidEnterBackground:(UIApplication *)application {/* leer */ }
- (void)applicationWillEnterForeground:(UIApplication *)application {/* leer */ }
- (void)applicationDidBecomeActive:(UIApplication *)application {/* leer */ }
- (void)applicationWillTerminate:(UIApplication *)application {/* leer */ }

#pragma mark -
#pragma mark Memory management

- (void)applicationDidReceiveMemoryWarning:(UIApplication *)application {/* leer */ }
- (void)dealloc { [window release]; [super dealloc]; }


@end
```

Application Lifecycle

- was passiert nach

  `application: didFinishLaunchingWithOptions:`

  - Application geht in die "run loop" :

    goon:

    erzeuge einen neuen `autorelease` Pool

    warte auf Events (touch, timed event, I/O, etc.)

    Event wird zugestellt an `UIKit`-Objekte und oft an eigene Objekte (via `delegates`, etc.)

    wenn alle Behandlungen abgeschlossen sind, wird der Screen neu gezeichnet (entsprechende `drawRect:` Methoden werden gerufen)

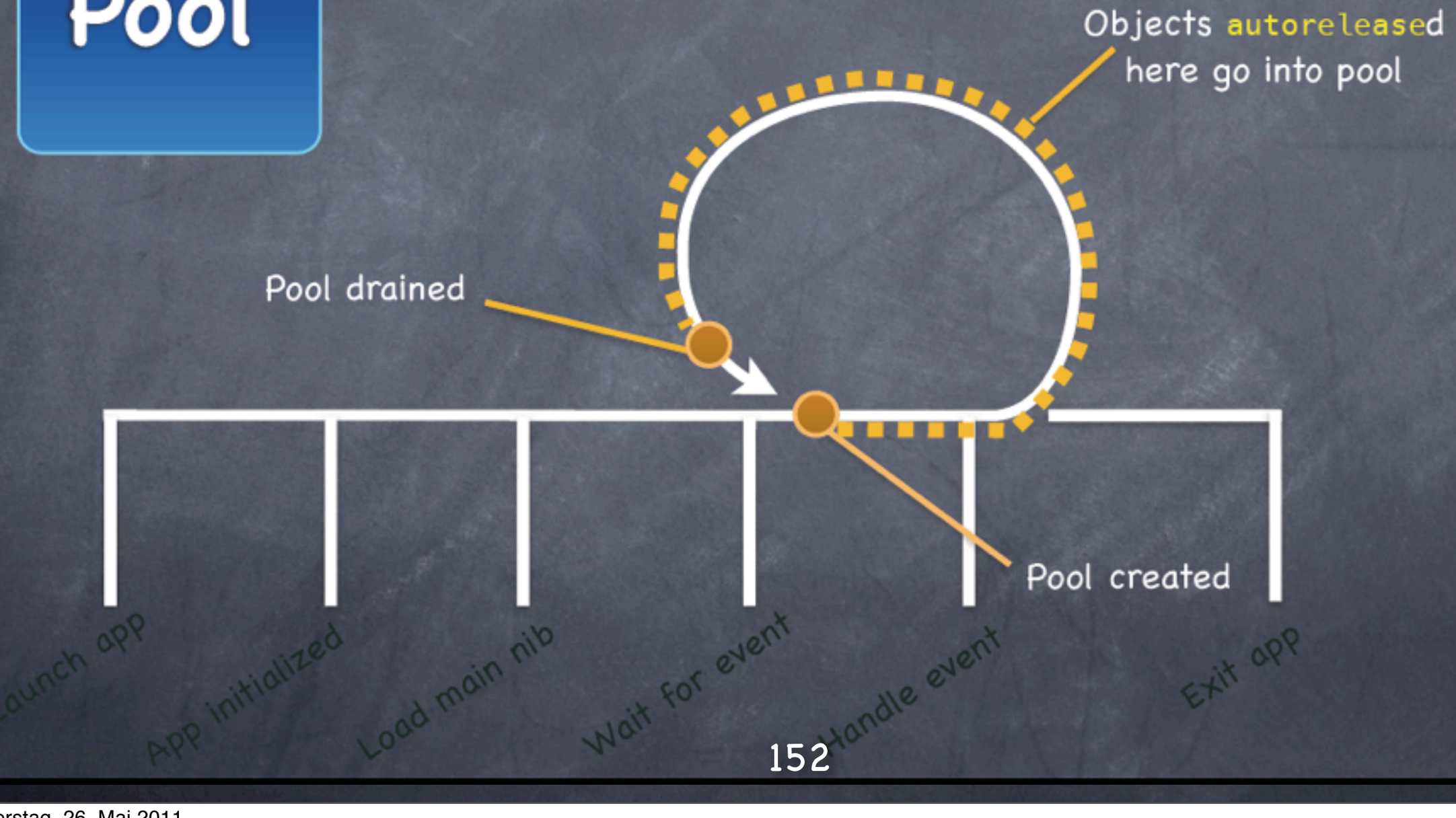
    Der `autorelease` Pool wird „abgelassen" (drain)

    goto goon

Application Lifecycle

Autorelease Pools

Pool



Stanford
CS193p
Fall 2010

ViewController Lifecycle

- ViewController entstehen und vergehen: Speicher muss bereinigt werden
- haben eine sehr wichtige Property (in UIViewController)
  - @property (retain) UIView *view;
- Zeiger auf den Top-level UIView im Controller's View (in MVC terms)
- eigene ViewController können am Lebenszyklus aktiv teilhaben, indem sie gewisse Methoden implementieren

ViewController Lifecycle

- es geht los mit `alloc` und `initWithNibName:bundle:` (siehe oben)
- wenn auf diese Weise kein xib-File gefunden wird, ruft der ViewController `- (void)loadView` an `self`
- loadView's Implementation MUSS die `view` Property setzen
- Nicht loadView implementieren UND .xib File bereitstelen (undefined behav.)

ViewController Lifecycle

- nach der Initialisierung (so oder so) wird gerufen
- `- (void)viewDidLoad;`
- ein guter Platz für weiteres Setup am ViewController - ABER: Die Geometrie des view's (bounds) ist noch unbekannt.
- Initialisierungen, die die Geometrie kennen müssen, kann man vornehmen in

  `- (void)viewWillAppear:(BOOL)animated;`

  bounds ist gesetzt (durch einen Superview)
- der view wird (zumeist) nur einmal geladen, aber kann des öfteren erscheinen/verschwinden, also hier nichts unterbringen, was besser in viewDidLoad gehört.

ViewController Lifecycle

- man wird informiert, wenn der View (gleich) verschwindet:

```
- (void)viewWillDisappear:(BOOL)animated { // ein Beispiel:
[super viewWillDisappear:animated];
// call this in all the viewWill/Did methods
// let's be nice to the user and remember the scroll position
[self rememberScrollPosition]; // we'll have to implement this
// do some other clean up now that we've been removed from the screen
[self saveDataToPermanentStore];
// but be careful not to do anything time-consuming here, or app will
// be sluggish maybe even kick off a thread to do what needs doing here
}
```

- "did" Versionen gibt's von beiden:

```
- (void)viewDidAppear:(BOOL)animated;
- (void)viewDidDisappear:(BOOL)animated;
```

ViewController Lifecycle

- schon bekannt:
  - (void)viewDidUnload;
  - gerufen in low-memory Situationen.
- Achtung: hier release an Outlets rufen!

ViewController Lifecycle

- Device Rotation
  - im ViewController festzulegen mittels

```
- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)anOrientation {
    return (anOrientation == UIInterfaceOrientationPortrait) ||
           (anOrientation == UIInterfaceOrientationPortraitUpsideDown);
}
```

  - Default ist: nur `UIInterfaceOrientationPortrait`.
  - es gibt auch noch: `UIInterfaceOrientationLandscapeLeft` und `UIInterfaceOrientationLandscapeRight`.
  - im Simulator mittels Menüpunkt: Hardware > Links/Rechts drehen

ViewController Lifecycle

- Device Rotation
  - wenn eine Drehung stattfindet, sollte der Controller darauf auch geeignet reagieren (sonst wird „zwangsweise“ skaliert) mittels:

```
- (void)willRotateToInterfaceOrientation:(UIInterfaceOrientation)
                                anOrientation duration:(NSTimeInterval)seconds;
- (void)didRotateFromInterfaceOrientation:(UIInterfaceOrientation)
                                anOrientation;
```

  - Aktuelle Orientierung ist in:

```
@property UIInterfaceOrientation interfaceOrientation;
```

  - „Teure“ Aktionen (z.B. eine Animation) in will beenden und in did fortsetzen.
  - Am einfachsten erreichbar, indem der view seine subviews geeignet (skalierbar) enthält, im IB gibt “struts and springs” die dies unterstützen
  - alternativ komplett neues Layout möglich (von will/did) einzustellen

UIImageView

siehe S. 142

- Ein `UIView`, der UIImages zeichnen kann
- Erzeugbar mit dem IB oder programmatisch:

```
UIImageView *imageView = [[UIImageView alloc]
initWithImage:(UIImage *)image];
```

- das UIImage kann man auswechseln, der `frame` bleibt wie initial!

```
@property (retain) UIImage *image;
```

UIImageView

- UIImageView kann auch (durch ein weiteres UIImage) hervorgehoben werden:

```
@property BOOL highlighted;
@property (retain) UIImage *highlightedImage;
UIImageView *imageView = [[UIImageView alloc] initWithImage:
                (UIImage *) highlightedImage:(UIImage *)];
```

- UIImageView kann eine Folge von UIImages animiert anzeigen

```
@property (retain) NSArray *animationImages; // of UIImage
@property (retain) NSArray *highlightedAnimationImages;
@property NSTimeInterval animationDuration;
@property NSInteger animationRepeatCount;
@property BOOL isAnimating;
- (void)startAnimating;
- (void)stopAnimating;
```

UIScrollView

- Wenn ein view „zu groß“ für das Gerät ist
- Panning & Zooming erwünscht? UIScrollView!

- Ein UIView, der „alles“ automatisch kann
- Viele wichtige UIView`s sind Ableitungen davon: UITextView, UITableView

UIScrollView

- stellt einen View mit (virtueller) Größe dar:

```
@property CGSize contentSize;
```

- oft nur ein subview, der den gesamten (verbleibenden) Platz einnimmt.
- Erzeugung
  - IB, aber http://stackoverflow.com/questions/1135163/how-do-i-use-uiscrollview-in-interface-builder:

    Strangely enough you can not do this from Interface Builder. You will have to do it from the view controller managing this scroll view.

  - oder in Code mit alloc/initWithFrame:
  - Beispiel: das gesamte Display als ScrollView

    in application:didFinishLaunchingWithOptions:

```
CGRect frame = [[UIScreen mainScreen] applicationFrame];
UIScrollView *scrollView = [[UIScrollView alloc] initWithFrame:frame];
[window addSubview:scrollView]; // noch leer !
```

UIScrollView

- der „zu große" UIView wird als subview eingetragen

```
UIImage *image = [UIImage imageNamed:@"bigimage.jpg"];
UIImageView *imageView = [[UIImageView alloc] initWithImage:image];
// frame.size = image.size
[scrollView addSubview:imageView];
```

- contentSize setzen:

```
scrollView.contentSize = imageView.frame.size; // for example
```

- weitere Properties:

```
@property BOOL bounces; // whether bounces at the edges
@property BOOL scrollEnabled; // whether scrolling is currently allowed
@property BOOL directionalLockEnabled;
// whether user's scrolling start locks allowed scroll
```

UIScrollView

```
struct CGAffineTransform {
    CGFloat a;
    CGFloat b;
    CGFloat c;
    CGFloat d;
    CGFloat tx;
    CGFloat ty;
};
typedef struct CGAffineTransform CGAffineTransform;
```

$$\begin{bmatrix} a & b & 0 \\ c & d & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

- Zooming
  - Alle UIView's haben eine property transform == affine Transformation (translate, scale, rotate).
    http://developer.apple.com/library/ios/#documentation/GraphicsImaging/Reference/CGAffineTransform/Reference/reference.html%23//apple_ref/doc/constant_group/CGAffineTransformIdentity
  - UIScrollView modifiziert diese beim Zooming.
- Funktioniert NUR, wenn minimum/maximum ZoomScale gesetzt wurde

```
scrollView.minimumZoomScale = 0.5; // 0.5 means half its normal size
scrollView.maximumZoomScale = 2.0; // 2.0 means twice its normal size
```

- UND wenn eine UIScrollViewDelegate-Methode den zu zoomenden view benennt

```
- (UIView *)viewForZoomingInScrollView:(UIScrollView *)sender;
// einfach wenn es nur einen gibt :-)
```