UIScrollView

- Zooming
  - eine weitere Delegate-Methode informiert über das Ende des Zooming:

```
- (void)scrollViewDidEndZooming:(UIScrollView *)sender
                       withView:(UIView *)zoomView
// returned from delegate method above
 atScale:(CGFloat)scale;
```

Wenn man einen (Custom-View) danach mit `scale` skaliert neu zeichnet, muss man danach die affine Transformation auf die Identität zurücksetzen:

```
zoomView.transform = CGAffineTransformIdentity;
```

UIScrollView

- Programmatisch scrollen

```
- (void)scrollRectToVisible:(CGRect)aRect animated:(BOOL)animated;
```

- Programmatisch zoomen

```
@property float zoomScale;
- (void)setZoomScale:(float)scale animated:(BOOL)animated;
- (void)zoomToRect:(CGRect)zoomRect animated:(BOOL)animated;
```

UINavigationController

- ein Controller von Contollern:
  - Idee: Stack of Controllers - Aktionen decken neue Controller (ihren view) auf, mit Back-Button wird zum vorherigen zurückgekehrt
  - ein Root ViewController, weitere dynamisch gestapelt

- UIViewController-Property

```
@property(nonatomic, readonly, retain) UINavigationController
                                              *navigationController

// nil if not in a navigation stack
```

- so kann eine Action in einem (obersten) ViewController einfach einen weiteren aufstapeln:

```
[self.navigationController pushViewController:vcToPush animated:YES];
```

UINavigationController

- wann verschwindet der oberste Controller (eigentlich sein view)?
  - meist (und automatisch) bei Klick auf Back-Button
  - manchmal programmatisch: z.B.

```
- (IBAction)deleteCurrentRecord:(UIButton *)sender {
// delete the record we are displaying
// we just deleted the record we are displaying!
// so it does not make sense to be on screen anymore, so pop
[self.navigationController popViewControllerAnimated:YES];
}
```

UINavigationController

- ViewController sollen (in beliebigen Kontexten) wiederverwendbar sein, sie sollen also ihren Vorgänger (im Stack) nicht kennen!
- Wie kann dennoch Information zurückfließen? Delegation! z.B.

```
- (IBAction)someAction:(UIButton *)sender {
// this action is going to cause another MVC's Controller to get pushed
PusheeViewController *pushee = [[PusheeViewController alloc] init];
pushee.someProperty = self.someValueThePusherKnows;
pushee.someOtherProperty = self.someOtherValueThePusherKnows;
pushee.delegate = self; // user protocol !
// self is the pusher and it implements the right (new) protocol
[self.navigationController pushViewController:pushee animated:YES];
// now we're done (we'll be pushed off screen) until we get popped back
// in the meantime, perhaps we will be informed of something as
// pushee's delegate
}
```

UITabBarController

- noch ein Controller of Controllers, verwaltet ein Anzahl von Controllern, die gleichzeitig geladen sind und über Tabs aktiviert werden

UITabBarController

```
- (BOOL)application:(UIApplication *)
  didFinishLaunchingWithOptions:(NSDictionary *) {
   UIViewController *vc1 = ...; // any ViewController
   UIViewController *vc2 = ...; // e.g. UINavigationControllers
   UIViewController *vc3 = ...;
   UITabBarController *tbc = [[UITabBarController alloc] init];
   tbc.viewControllers =
        [NSArray arrayWithObjects: vc1, vc2, v3, nil];
   [vc1 release]; [vc2 release]; [vc3 release];
   [window addSubview:tbc.view];
   [window makeKeyAndVisible];
   return YES;
}
```

UITabBarController

- die (automatisch erzeugten) Tabs tragen sind mit der (UIViewController) Property
  `@property NSString* title;` der Subcontroller beschriftet (und enthalten kein Bild) oder
- werden erzeugt aus der (UIViewController) Property
  `@property UITabBarItem *tabBarItem`

```
// call from initWithNibName:bundle: and awakeFromNib
// NOT in viewDidLoad: Tab wir angezeigt, bevor der View geladen wird.
- (void)setup {   // erst Klick auf Tab lädt den View
    UITabBarItem *item = [[UITabBarItem alloc]
       initWithTitle:@"Timer"
                image:[UIImage imageNamed:@"timer.png"]
                  tag:0]; // identifying tag, can ignore
    self.tabBarItem = item;
    [item release];
}
```

UITabBarController

- es gibt einige vordefinierte TabBarItems:

```
UITabBarItem *item = [[UITabBarItem alloc]
    initWithSystemItem: UITabBarSystemItemSearch tag:0];
```

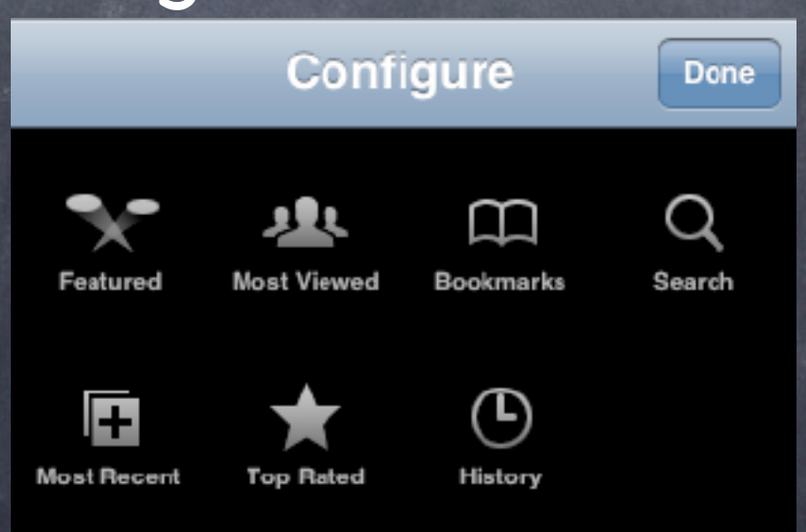
```
typedef enum {
    UITabBarSystemItemMore,
    UITabBarSystemItemFavorites,
    UITabBarSystemItemFeatured,
    UITabBarSystemItemTopRated,
    UITabBarSystemItemRecents,
    UITabBarSystemItemContacts,
    UITabBarSystemItemHistory,
    UITabBarSystemItemBookmarks,
    UITabBarSystemItemSearch,
    UITabBarSystemItemDownloads,
    UITabBarSystemItemMostRecent,
    UITabBarSystemItemMostViewed,
} UITabBarSystemItem;
```

UITabBarController

- am `tabBarItem` kann man auch ein Badge anheften:

```
- (void)somethingHappenedToCauseUsToNeedToShowABadgeValue {
    self.tabBarItem.badgeValue = @“2”;
}
```

- bei mehr als 4 Controllern wird automatisch als 5. ein `UITabBarSystemItemMore` eingefügt, bei Klick auf diesen, wird (ebenfalls automatisch) ein View angezeigt der die Auswahl neuer 4 erlaubt:



UITableView

- ein View um Daten in einer Liste (einspaltige Tabelle) anzuzeigen, Ableitung von `UIScrollView` (up/down)
- oft mit NavigationControllern verknüpft, so dass über baumartige Strukturen navigiert wird
- Nutzercode in Form zweier! Delegates einbringbar:
  - `delegate` (wie üblich UI-will/should/did- Aktionen)
  - `dataSource` (wo kommen die Daten her)

UITableView

- zwei Styles möglich

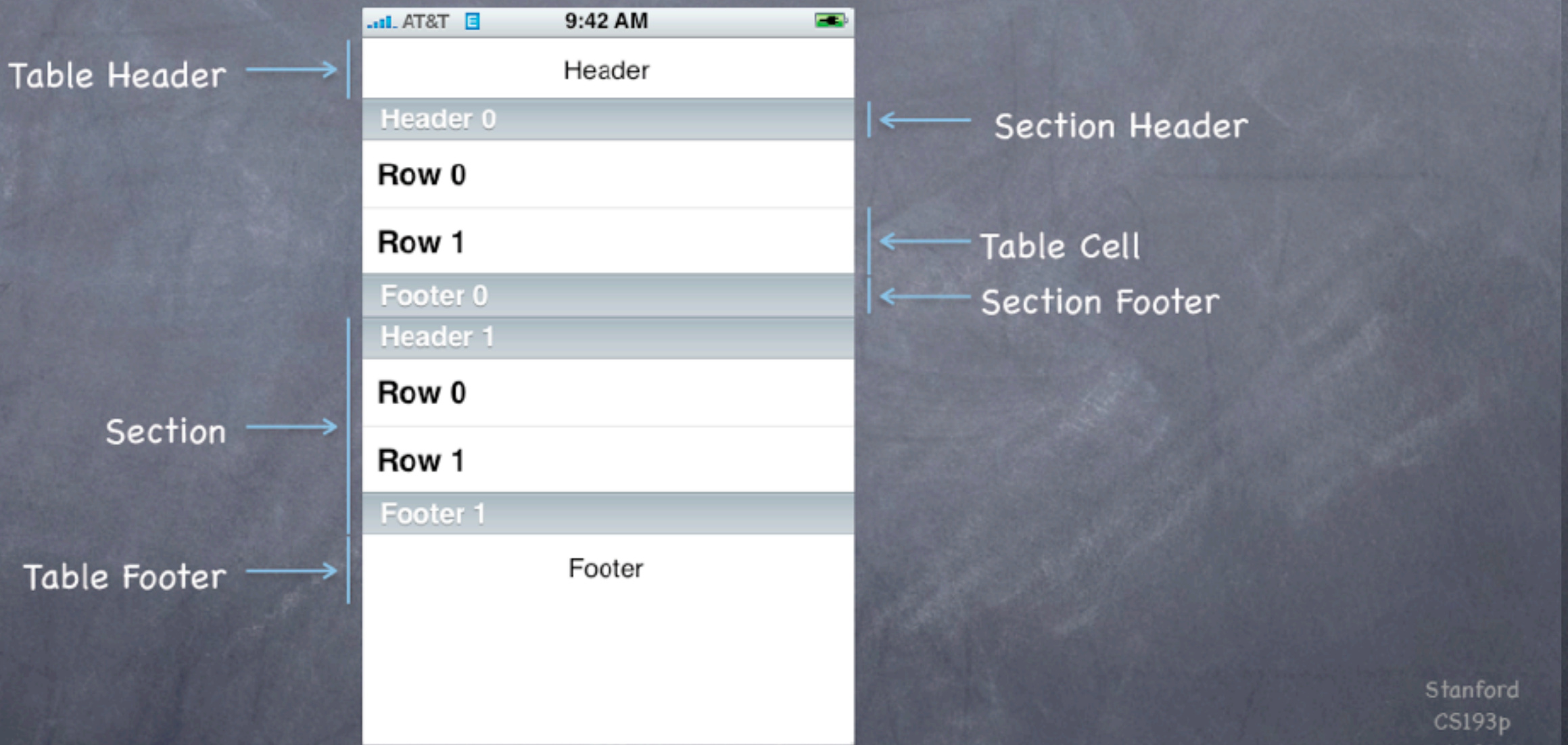
```
UITableViewStylePlain
UITableViewStyleGrouped
```

- Festlegung im DI (nicht änderbar):

```
- (id)initWithFrame:(CGRect)aRect
              style:(UITableViewStyle)style;
```

UITableView

Plain Style



Stanford
CS193p
Fall 2010

UITableView

Grouped Style

Stanford
CS193p
Fall 2010

UITableView

- Wie kommen Daten in die Tabelle?
  - Delegated: der übergeordnete Controller ist zumeist dataSource für den UITableView.

```
@property (assign) id <UITableViewDataSource> dataSource;
```

- es werden immer nur soviel Daten angezeigt (& angefordert!) wie der Platz es zulässt

UITableView

- initial muss der TableView die Gesamtanzahl der Daten(-sätze) kennen, um seine (UIScrollView contentSize) zu bestimmen: erste Anfrage an die dataSource "how many rows?"
  - Wie viele Sektionen?

    ```
    - (NSInteger)numberOfSectionsInTableView:(UITableView *)sender;
    ```

    optional (default == 1)
  - Wie viele Zeilen in Sektion Nr. section?

    ```
    - (NSInteger) tableView: (UITableView *) sender
      numberOfRowsInSection:(NSInteger)section;
    ```

    REQUIRED !

UITableView

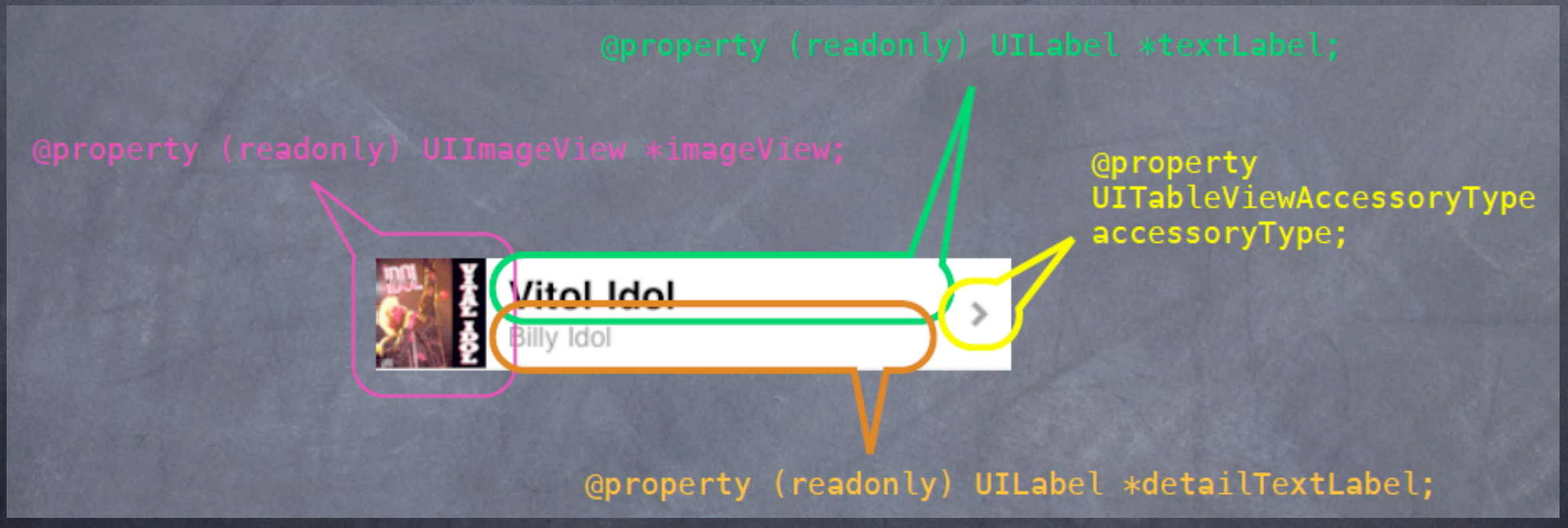
- später werden die Daten angefragt, sobald sie anzuzeigen sind

```
- (UITableViewCell *)tableView:(UITableView *)sender
        cellForRowAtIndexPath:(NSIndexPath *)indexPath;
```

- Mit `NSIndexPath` wird Sektion und Zeile in einem Argument übergeben:
  - `indexPath.section` Sektion
  - `indexPath.row` Zeile

UITableViewCell

- was ist eine `UITableViewCell`
  - ein UIView, der eine Zeile eines TableViews darstellt
  - der Inhalt wird durch Properties bestimmt:

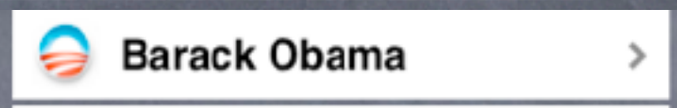


UITableViewCell

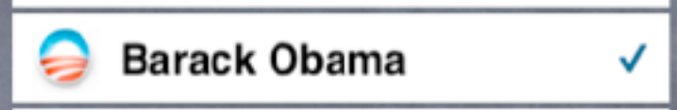
- accessoryType ?

```
typedef enum {
   UITableViewCellAccessoryNone,
   UITableViewCellAccessoryDisclosureIndicator,
   UITableViewCellAccessoryDetailDisclosureButton,
   UITableViewCellAccessoryCheckmark
} UITableViewCellAccessoryType;
```

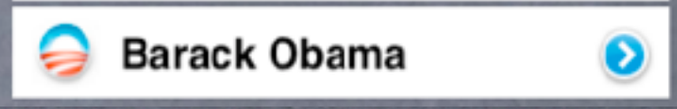
UITableViewCellAccessoryDisclosureIndicator



UITableViewCellAccessoryCheckmark



UITableViewCellAccessoryDetailDisclosureButton




```
- (UITableViewAccessoryType)tableView:(UITableView *)sender
accessoryTypeForRowWithIndexPath:(NSIndexPath *)indexPath {
    if (....) {
        return UITableViewCellAccessoryDetailDisclosureButton
    }
}

- (void)tableView:(UITableView *)sender
accessoryButtonTappedForRowWithIndexPath:(NSIndexPath *)indexPath {
    // only get here if a blue button is tapped
}
```

UITableViewCell

- Erzeugung von UITableViewCell's auf Anfrage (in `tableView:cellForRowAtIndexPath:`) mit dem DI

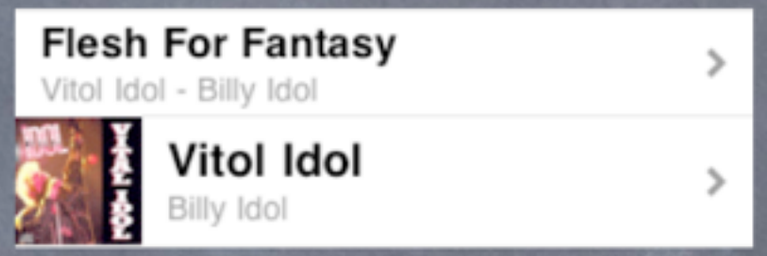
```
- initWithStyle:(UITableViewCellStyle)style
reuseIdentifier:(NSString *)reuseId;
```

- style:

```
typedef enum {
   UITableViewCellStyleDefault,
   UITableViewCellStyleValue1,
   UITableViewCellStyleValue2,
   UITableViewCellStyleSubtitle
} UITableViewCellStyle;
```

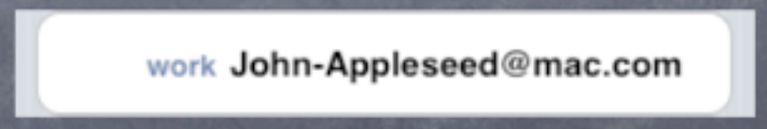
UITableViewCellStyleDefault

UITableViewCellStyleSubtitle



UITableViewCellStyleValue1

UITableViewCellStyleValue2



UITableViewCell

- Beispiel: Daten sind NSStrings in einem NSArray myArray
- eine lange Liste von Strings

numberOfSectionsInTableView: muss nicht implementiert werden (1)

- Anzahl der Zeilen ist count den Arrays

```
- (NSInteger)tableView:(UITableView *)sender
 numberOfRowsInSection:(NSInteger)section {
    return myArray.count; // section == 0
}
```

UITableViewCell

- Bereitstellung der Daten

```
- (UITableViewCell *)tableView:(UITableView *)sender
        cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    UITableViewCell *cell = ...; // more to come on this
    cell.textLabel.text = [myArray objectAtIndex:indexPath.row];
    // section always 1
    cell.imageView.image = [myImages objectAtIndex:indexPath.row];
    cell.detailTextLabel.text = [mySubtitles objectAtIndex:indexPath.row];
    cell.accessoryType = <figure out what kind of accessory type we want>;
    return cell;
}
```

UITableViewCell

- Bereitstellung der Daten

```
... // in tableView:(UITableView *)sender
            cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    UITableViewCell *cell =
        [[UITableViewCell alloc] initWithStyle:someStyle
                               reuseIdentifier: ???]
```

- Problem: hätte MyArray 10.000 Einträge, so würden im Verlauf der Zeit max. 10.000 UITableViewCells erzeugt werden, von denen aber immer nur „eine Handvoll" zu sehen ist :-(

UITableViewCell

- dafür ist der reuseIdentifier (UITableViewCell-Pool):

```
... // in tableView:(UITableView *)sender
            cellForRowAtIndexPath:(NSIndexPath *)indexPath {
UITableViewCell *cell =
        [sender dequeueReusableCellWithIdentifier:@“MyTV”];
if (!cell) { // no reusable cell in pool
   cell = [[[UITableViewCell alloc]
                    initWithStyle:someStyle
                  reuseIdentifier: @“MyTV”] autorelease ];
 ...
```

- `dequeueReusableCellWithIdentifier` wird von `UITableView` automatisch bereitgestellt

UITableViewCell

- anstelle des accessory Indicators kann auch ein ganzer View angezeigt werden

```
@property (retain) UIView *accessoryView;
```

- Man kann auch einen custom view in einer tableViewCell hinterlegen, der das gesamte Erscheinungsbild bestimmt

```
@property (readonly) UIView *contentView;
```

  - property abholen und custom view als subview einfügen

back to UITableView

- was kann die dataSource noch steuern?
  - den Inhalt von (section)-header und -footer

```
- (NSString *)tableView:(UITableView *)sender
    titleForHeaderInSection:(NSInteger)section;
- (NSString *)tableView:(UITableView *)sender
    titleForFooterInSection:(NSInteger)section;
```

  - die Editierbarkeit von Zellen (inserting, rearranging, deleting rows) [lassen wir hier aus :-)]

UITableViewDelegate

- bisher wurden die Methoden von UITableView's dataSource besprochen
- UITableView hat eine weiteren ‚protocol-driven' Delegate mit dem Namen delegate.
- der delegate steuert, wie der UITableView angezeigt wird, nicht was er anzeigt (that's the dataSource's job).
- über diesen delegate werden die Interaktionen mit dem TableView zugänglich, z.B. was ist zu tun, wenn eine Zelle ausgewählt wurde?
- sehr oft werden dataSource und delegate vom selben Objekt (dem Controller dessen (Sub-)view der UITableView ist) realisiert

UITableViewDelegate

- wichtige Methoden:
  - gleich wird eine Zelle angezeigt

```
- (void)tableView:(UITableView *)sender
   willDisplayCell:(UITableViewCell *)cell
   forRowAtIndexPath:(NSIndexPath *)indexPath;
```

letze Chance, die Zelle zu modifizieren (z.B. setzen der backgroundColor)

  - Zelle wurde ausgewählt

```
- (void)tableView:(UITableView *)sender
   didSelectRowAtIndexPath:(NSIndexPath *)indexPath {
   // z.B.
   SomeViewController *vcToPush = [[SomeViewController alloc] init];
   vcToPush.someProperty = <something dependent on my data at indexPath>;
   [self.navigationController pushViewController:vcToPush animated:YES];
   [vcToPush release];
}
```

UITableViewDelegate

- weitere Methoden:
  - die Höhe von Zellen einstellen

```
- (CGFloat)tableView:(UITableView *)sender
heightForRowAtIndexPath:(NSIndexPath *)indexPath;
```

  - die Höhe von Headers und Footers einstellen

```
- (CGFloat)tableView:(UITableView *)sender
heightForHeaderInSection:(NSInteger)section;
```

  - eigene custom header/footer views einstellen

```
- (UIView *)tableView:(UITableView *)sender
viewForHeaderInSection:(NSInteger)section;
```

  - Selektion von Zellen verbieten

```
- (NSIndexPath *)tableView:(UITableView *)sender
   willSelectRowAtIndexPath:(NSIndexPath *)indexPath {
   if (<we don't want row at indexPath to be selected>) return nil;
}
```

UITableViewController

- eine ‚Convenience class`
  - ein UIViewController der die dataSource und delegate Protokolle von UITableView implementiert
  - implementiert loadView:

    Setzt die view property auf einen UITableView der automatisch erzeugt wird
  - und weiteres ...

    selektiert und deselektiert Zellen zur richtigen Zeit, wenn ein navigation controller verwendet wird.

    lässt scroll indicators aufflackern, wenn der view erscheint

    ....

UIWebView

- Ein kompletter Internet Browser in einem UIView
  - kann nicht nur HTML darstellen, sondern auch PDF's und andere komplexe Dokumente
- Basiert auf dem WebKit
  - ein open source HTML rendering framework (gestartet von Apple).
- ‚Seiten laden leicht gemacht!'
- ein delegate kann User Interaktion steuern/ beobachten
- unterstützt JavaScript
  - aber beschränkt auf 5 Sekunden Laufzeit und 10Mb Speicher

UIWebView

- Anzeige von (Web-)Inhalten:

```
- (void)loadRequest:(NSURLRequest *)request;
- (void)loadHTMLString:(NSString *)string baseURL:(NSURL *)baseURL;
- (void)loadData:(NSData *)data
        MIMEType:(NSString *)MIMEtype
textEncodingName:(NSString *)encodingName
         baseURL:(NSURL *)baseURL;
```

UIWebView

- NSURLRequest

```
+ (NSURLRequest *)requestWithURL:(NSURL *)url;
+ (NSURLRequest *)requestWithURL:(NSURL *)url
    cachePolicy:(NSURLRequestCachePolicy)policy
    timeoutInterval:(NSTimeInterval)timeoutInterval;
```

- NSURL
  - ein „wohlgeformter" NSString, z.B. file:// ... oder http://...
  - dies ist der empfohlene Weg im iOS API, Dateinamen zu spezifizieren !

```
+ (NSURL *)urlWithString:(NSString *)urlString;
+ (NSURL *)fileURLWithPath:(NSString *)path
                isDirectory:(BOOL)isDirectory;
```

- NSURLRequestCachePolicy
  - ignoriere local cache; ignoriere caches im Internet; benutze expired caches; ... (NSURLRequestUseProtocolCachePolicy = 0)

UIWebView

- für "Browser UI" neben einem WebView

```
@property (readonly) BOOL loading;
@property (readonly) BOOL canGoBack;
@property (readonly) BOOL canGoForward;
- (void)reload;
- (void)stopLoading;
- (void)goBack;
- (void)goForward;
```

- Darstellung und Detektoren

```
@property BOOL scalesPageToFit; // default is NO!
@property UIDataDetectorTypes dataDetectorTypes;
// UIDataDetectorTypePhoneNumber/Link/Address/CalendarEvent
```

startet die entsprechenden Anwendungen

UIWebView

- UIWebViewDelegate Methoden

```
- (BOOL)webView:(UIWebView *)sender
shouldStartLoadWithRequest:(NSURLRequest *)request
            navigationType:(UIWebViewNavigationType)navigationType
/* UIWebViewNavigationType:
   UIWebViewNavigationTypeLinkClicked
   UIWebViewNavigationTypeFormSubmitted  */
```

```
- (void)webViewDidStartLoad:(UIWebView *)sender
- (void)webViewDidFinishLoad:(UIWebView *)sender
- (void)webView:(UIWebView *)sender
      didFailLoadWithError:(NSError *)error;
```

- **Important:** Before releasing an instance of UIWebView for which you have set a delegate, you must first set its delegate property to nil. This can be done, for example, in your dealloc method.

- **Important:** You should not embed UIWebView or UITableView objects in UIScrollView objects. If you do so, unexpected behavior can result because touch events for the two objects can be mixed up and wrongly handled.