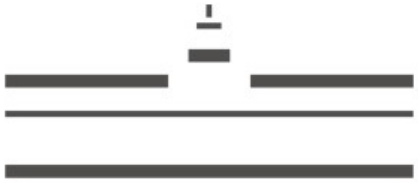


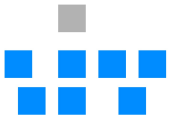
Westfälische
Wilhelms-Universität
Münster

Programmierung für mobile Endgeräte

Cocoa Touch (II)

wissen.leben
WWU Münster

Name: Patrick Förster



ZENTRUM FÜR
INFORMATIONS
VERARBEITUNG

Delegates

- Sobald man per Template ein neues Projekt anlegt, wird immer eine Klasse mit dem Postfix „Delegate“ erstellt, deren Instanz sozusagen als Vermittler zwischen dem iOS-Betreibssystem und der Applikation fungiert
- Eine Instanz dieser Klasse wird beim Starten der App erstellt
- Das Delegate wird bspw. informiert, wenn die Applikation durch den Nutzer per „Home“-Button in den Hintergrund geschoben oder wieder aktiviert wurde
- Das Delegate-Prinzip findet sich in vielen Klassen des Cocoa-Frameworks wieder
- Delegate-Instanzen gehorchen immer einem bestimmten Protokoll
- Die meisten UI-Objekte verfügen über ein Delegate, das informiert wird, sobald bestimmte Ereignisse auf dem Objekt ausgeführt oder angestoßen wurden
- Mit der Integration der pList wäre dieser Teil der Applikation fertig, allerdings mag es stören, dass das Absender der Suche noch explizit über den eigens erstellten Such-Button, statt mit dem „Return“ auf der Tastaur ausgelöst wird
- Die Klasse UITextField verfügt über ein Delegate, auf Basis des UITextFieldDelegate Protokolls, das genau dieses Ereignis abdeckt

Delegates (II)

- Ganz im Sinne des MVC-Patterns sollte ein Delegate Teil der Controller-Logik sein
- Da sich der Suchschlitz auf der Startseite befindet, wird der zugehörige Controller als Delegate für diesen eingetragen
- Dazu im Storyboard nach Rechtsklick auf das Textfeld das Delegate-Outlet mit dem ViewController verbinden
- der Vollständigkeit halber sollte man die Deklaration des Controllers noch um das entsprechende Delegate erweitern
- Die zum „Return“ Ereignisse gehörende Methode ist textFieldShouldReturn
- Innerhalb der Methode muss nun der Übergang eigenhändisch eingeleitet werden:

```
- (BOOL)textFieldShouldReturn:(UITextField *)textField {
    [textField resignFirstResponder];

    PGResultViewController* ctrl = [self.storyboard instantiateViewControllerWithIdentifier:@"ResultController"];
    ctrl.searchQuery = _textField.text;
    [self.navigationController pushViewController:ctrl animated:YES];

    return YES;
}
```

Delegates (III)

- Zuerst wird mit resignFirstResponder die Tastatur wieder geschlossen
- Dann wird der Ziel-Controller ermittelt
  - Damit der Controller über das Storyboard identifiziert werden kann, muss zuvor im Storyboard über den Identity-Inspector ein „Identifier“ gesetzt werden
- Dann wird analog zur prepareForSegue Methode die Suchanfrage an den Ziel-Controller übermittelt
- Abschließend wird nun die „Push“-Anweisung ausgeführt, um den Ziel-Controller auf den Stack zu schieben und somit den Übergang einzuleiten
- Als zusätzliche Erweiterung kann man noch die Anzahl der möglichen Buchstabeneingaben beschränken:
- Bei jeder Eingabe auf der Tastatur wird die shouldChangeCharactersInRange Methode des Delegates aufgerufen; ideal um die Eingabe zu überwachen:

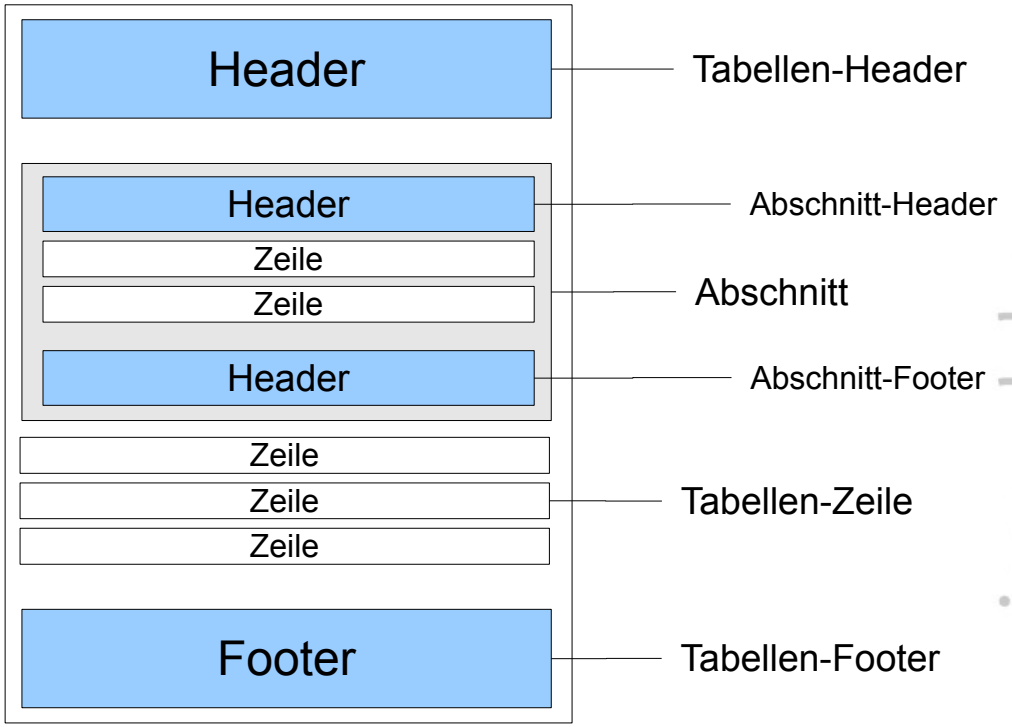
```
- (BOOL)textField:(UITextField *)textField shouldChangeCharactersInRange:(NSRange)range replacementString:
(NSString *)string {
    return textField.text.length < 3 || [string isEqualToString:@""];
}
```

Die Kennzeichen App

- In der Kennzeichen-App sollte in einer gesonderten Ansicht eine Übersicht über alle Kennzeichenkürzel ausgegeben werden
- Eine solche Darstellung ist prädestiniert zur tabellarischen Auflistung
- Noch einmal zur Wiederholung:
  - Um eine neue Ansicht in die App zu integrieren benötigt es nach dem MVC-Architekturmuster einen Controller, der die Interaktion mit der Ansicht verarbeitet und ggf. die nötigen Modeldaten liefert
  - Die Modeldaten liegen für die App bereits als plist-Datei vor
  - Ein ViewController fehlt allerdings noch
  - Zum Glück bietet das Cocoa-Touch Framework mit dem TableViewController bereits einen fertigen Baustein zur Verwaltung einer Ansicht im gewünschten Design

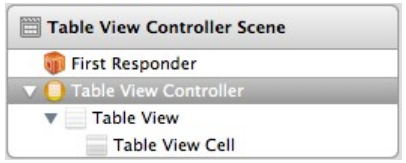
TableViewController

- Mit dem TableViewController können vertikal scrollbare Tabellen verwaltet werden
- Die Bezeichnung „Tabelle“ ist wenig irreführend, da der View eigentlich aus einer Auflistung von Zellen besteht
- Die Zellen selbst können allerdings individuell gestaltet werden
- Des weiteren können mehrere Zellen zu Abschnitten zusammengefasst werden
- Die Tabelle sowie die Abschnitte können zudem durch einen Header bzw. Footer erweitert werden
- Außerdem lässt sich eine Index-Navigation hinzufügen



TableViewController (II)

- Als erstes fügt man den TableViewController dem Storyboard hinzu
- Wenn man im Document-Outline den neuen TableView-Controller aufklappt, sieht man dass nicht wie üblich ein normales View-Outlet, sonden das zum Controller passende TableView-Outlet miterzeugt worden ist



- Ein Rechtsklick auf den View und man sieht, dass dieser zwei Delegates bereitstellt, die bereits auf den Controller verweisen
- Das dataSource-Delegate stellt die Verbindung zwischen dem View und den Daten her und erfüllt das Protokoll UITableViewDataSource
- Damit in die neue Ansicht gewechselt werden kann, muss der zugehörige Controller in die Navigation des NavigationController eingebunden werden
- Dazu legt man in der Startansicht einen neuen Button an und verbindet diesen durch einen Übergang mit dem neuen TableViewController
- Außerdem muss noch eine neue TableViewController-Klasse definiert und zugewiesen werden

UITableViewDataSource-Delegate

- In der Ansicht sollen die Kennzeichen alphabetisch sortiert und gruppiert nach ihrem ersten Buchstaben dargestellt werden
- D.h. die Ansicht hat insgesamt 26 verschiedene Abschnitt
- Diese Information wird durch die Delegate-Methode numberOfSectionsInTableView an den View weitergegeben:

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return 26;
}
```

- Die nächste Angabe, die gemacht werden muss, ist die Anzahl Datensätze innerhalb eines Abschnittes
- Die zugehörige Delegate-Methode heißt: tableView: numberOfRowsInSection:
- Die nötige Angabe sollte man wenn möglich nicht hartkodiert anbieten, sondern aus den gegebenen Datensätzen ableiten
- Da die Methode allerdings relativ häufig – bei jedem Neuaufbau der Ansicht – aufgerufen wird, lohnt es sich die Daten zwischezuspeichern

Die PGKfzLookup-Hilfsklasse

- Die Hilfsklasse PGKfzLookup führt die Aufteilung der Datensätze bei ihrer Instanziierung durch und speichert das Ergebnis:

```
_dictByLeadingCharacter = [NSMutableDictionary dictionaryWithCapacity:[_dictonary count]];

for (int i = 0; i < 26; i++) {
    [_dictByLeadingCharacter setObject:[NSMutableArray array] forKey: [NSString stringWithFormat:@"%c", 65 + i]];
}

for (NSString* key in [self keys]) {
    NSString* lead = [key substringWithRange:NSMakeRange(0, 1)];
    NSMutableArray* array = [_dictByLeadingCharacter objectForKey:lead];

    [array addObject: key];
}
```

- Die Tabelle dictByLeadingCharacter enthält für jeden Buchstaben des Alphabets eine Liste der Kennzeichen, die mit diesem Buchstaben beginnt
- Über die Methode entriesForLead liefert eine Instanz alle Kennzeichen zu einem Buchstaben:

```
-(NSArray*) entriesForLead: (NSString*) string {return [_dictByLeadingCharacter objectForKey:string];}
```

UITableViewDataSource-Delegate (II)

- Nachdem die Hilfsklasse per import im TableViewController bekannt gemacht hat, kann auch tableView: numberOfRowsInSection:implementiert werden

```
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section {
    return [[[PGKfzLookup sharedInstance] entriesForLead:[NSString stringWithFormat:@"%c", 65 + section]] count];
}
```

- Startet man nun die App und wechselt in die Übersicht, wird die App abstürzen
- Der Grund: Es wurde nun zwar definiert wie viele Datensätze in einem bestimmten Abschnitt angezeigt werden sollen, allerdings nicht welche
- Zur Darstellung wird ein Datensatz in eine Layout-Zelle gepackt
- Da mehrere Datensätze durchaus gleich dargestellt werden können, muss nicht zwangsläufig für jeden Datensatz eine Zelle erstellt werden
- Es reicht genau so viele Zelle bereitzustellen, um den Bereich, der dem TableView zugeordnet wurde, zu füllen
- Wird die Ansicht z.B. gescrollt, können die Zellen der verschwindenden Datensätze für die neuen wiederverwertet werden

UITableViewDataSource-Delegate (III)

- Durch die tableView:cellForRowAtIndexPath Methode werden die Zellen erstellt und mit den nötigen Daten gefüllt:

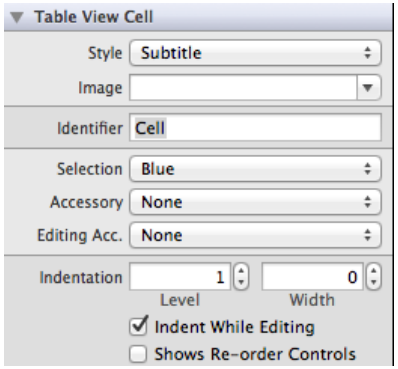
```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    static NSString *cellIdentifier = @"Cell";
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:cellIdentifier];
    if (!cell)  cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault reuseIdentifier:cellIdentifier];
    NSArray* array = [[PGKfzLookup ...] entriesForLead:[NSString stringWithFormat:@"%c", 65 + indexPath.section]];
    cell.textLabel.text = [array objectAtIndex:indexPath.row];
    cell.detailTextLabel.text = [[PGKfzLookup ...] lookup:cell.textLabel.text];

    return cell;
}
```

- In den ersten drei Zeilen der Methode wird überprüft, ob der View eine freie Zelle zur Verfügung hat
- In dies nicht der Fall, muss eine neue erzeugt werden:
- Danach wird die Zelle mit Daten gefüllt: Über die PGKfzLookup-Klasse wird mit Hilfe des gelieferten indexPathes die zum Buchstaben gehörige Liste aller Kennzeichen gesucht und der entsprechende Eintrag herausgesucht

UITableViewDataSource-Delegate (IV)

- Startet man die App und wechselt in die Übersicht, werden alle Kennzeichen in einer Liste angezeigt, allerdings fehlen die zugehörigen Landkreise/Städte
- In der tableView:cellForRowAtIndexPath Methode wurde ein Bezeichner genutzt um die Art der Zelle zu identifizieren, die zur Darstellung genutzt werden soll
- Im Storyboard wurde nach dem Hinzufügen des TableViewController eine Default-Zelle erstellt
- Im Attribute-Inspector kann man zwischen vier verschiedenen Stilen wählen
- Damit die Auswahl sowie alle im Inspector definierten Eigenschaften allerdings auch Anwendung finden, muss der dort vergebene Bezeichner mit dem in der tableView:cellForRowAtIndexPath Methode benutzen übereinstimmen



UITableViewCell

- Damit ist die Übersicht prinzipiell fertig, aber flexibler ist man, wenn man sich selbst um das Design der Zellen kümmert
- Bisher wurden lediglich Controller-Klassen selbst definiert, aber natürlich lassen sich auch eigene Definition für die View-Objekte erstellen
- Für die Zellen wird eine neue Klasse erstellt, die von UITableViewCell erbt und über den Identity-Inspector der Default-Zelle zugeordnet
- Der zuvor gesetzte Stil wird wieder auf „Custom“ zurückgesetzt
- Nun kann die Zelle gestaltet werden
  - Zur erst einen View hinzufügen, auf dem dann weitere Elemente platziert werden können
  - Dann analog zum Ergebnis-View das „leere Plakete“ Bild und ein Label für das Kennzeichen hinzufügen
  - Hinzukommend ein weiteres Label für den Landkreis/ die Stadt
- Zur besseren Übersicht können über den Identity-Inspector Bezeichner für die UI-Objekte verteilt werden, die im Document-Outline angezeigt werden

UITableViewCell (II)

- Wie gehabt müssen die Labels jetzt noch mit IBOutlets verbunden werden, die innerhalb der neuen Zellenklasse angelegt werden und durch Properties nach Außen zugreifbar gemacht werden
- In der tableView: cellForRowAtIndexPath Methode können nun Instanzen des neuen Zellentyps erstellt und die Labels entsprechend gefüllt werden:

```
static NSString *cellIdentifier = @"Cell";
PGTableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:cellIdentifier];
if (!cell) {
    cell = [[PGTableViewCell alloc] initWithStyle:UITableViewCellStyleDefault reuseIdentifier:cellIdentifier];
}

NSArray* array = [[PGKfzLookup ...] entriesForLead:[NSString stringWithFormat:@"%c", 65 + indexPath.section]];
cell.licenceLabel.text = [array objectAtIndex:indexPath.row];
cell.detailsLabel.text = [[PGKfzLookup ...] lookup:cell.licenceLabel.text];

[cell.licenceLabel setFont:[UIFont fontWithName:@"Euro Plate" size:25]];

return cell;
```

Index

- Als letztes wird der Tabellenübersicht eine Index-Leiste hinzugefügt, über die schneller zu den einzelnen Buchstaben-Abschnitten gesprungen werden kann
- Die dafür zu implementierenden Methoden gehören wieder zum UITableViewDataSource:
  - Liste aller Überschriften:

```
-(NSArray*) sectionIndexTitlesForTableView:(UITableView *)tableView {
    NSMutableArray* array = [NSMutableArray arrayWithCapacity:26];
    for (NSUInteger i = 0; i < 26; i++) [array addObject:[NSString stringWithFormat:@"%c", (65 + i)]];
    return array;
}
```

- Überschrift zu einem bestimmten Abschnitt:

```
-(NSString*) tableView:(UITableView *)tableView titleForHeaderInSection:(NSInteger)section {
        return [NSString stringWithFormat:@"%c", (65 + section)];
}
```

- Welcher Abschnitt gehört zu welcher Überschrift:

```
-(NSInteger) tableView:(UITableView *)tableView sectionForSectionIndexTitle:(NSString *)title atIndex:(NSInteger)index {
    return (NSInteger) [title characterAtIndex:0] - 65;
}
```

Details

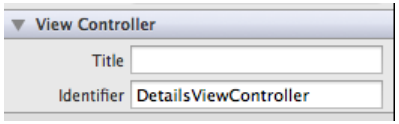
- Im Prinzip bietet die Übersicht nun alle nötigen Informationen
- Dennoch könnte man sich vorstellen zu den einzelnen Landkreisen/Städten noch gesonderte Detailinformation
- Diese sollen in einer separaten Detailansicht dargestellt werden, die durch einen Touch auf einen bestimmten Tabelleneintrag geladen wird

- Die nötigen Schritte wurden nun bereits des Öfteren durchgeführt
  - Einen ViewController über das Storyboard hinzufügen
  - Eine neue Controller-Klasse erstellen und zuweisen
  - Die Ansicht über das Storyboard designen
  - IBOutlets/IBActions hinzufügen und über das Storyboard verbinden
  - Den neuen View durch einen Übergang mit der Tabellenansicht verbinden

- Letzteres stellt einem vor ungeahnte Probleme: Leider wird die prepareForSegue Methode vor didSelectRowAtIndexPath aufgerufen, die ihrerseits aufgerufen wird, sobald der Nutzer einen Eintrag auswählt

Details (II)

- Der autogenerierte Rumpf der select-Methode suggeriert allerdings bereits, dass dort der geeignete Zeitpunkt ist, den Übergang händisch auszulösen
- Zuerst muss dafür der neue Controller über den Attribute-Inspector mit einem Bezeichner versehen werden



- Per programmatischem Zugriff auf das Storyboard kann nun über diesen Bezeichner eine neue Instanz des Controllers basierend auf dem im Storyboard definierten Layout erzeugt werden:

```
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath {
    PGDetailsViewController* ctrl = [self.storyboard instantiateViewControllerWithIdentifier:@"DetailsViewController"];
    NSArray* array = [[PGKfzLookup ...] entriesForLead:[NSString stringWithFormat:@"%c", 65 + indexPath.section]];
    ctrl.selection = [array objectAtIndex:indexPath.row];

    [self.navigationController pushViewController:ctrl animated:YES];
}
```

- Ohne Segue über das Storyboard wird so allerdings nicht automatisch ein neues Navigation-Objekt erzeugt, das die Überschrift anzeigt. Dies kann korrigiert werden indem man händisch einen neuen Item im View einfügt

MapKit

- In der Detailansicht könnte man sich nun austoben
- Für das Beispiel beschränken sich die angezeigten Informationen allerdings darauf, eine Karte mit der zum Kennzeichen gehörenden Stadt einzublenden
- Dazu wird das MapKit-Framework benutzt, das zuerst dem Projekt hinzugefügt werden muss
- Nach Auswahl des Projekts im Projekt-Navigator wählt man das entsprechende Target und dann Summary, scrollt dort nach unten bis man auf „Linked Frameworks and Liberies“ trifft und fügt hier über den „+“ Button das MapKit-Framework hinzu
- Nun kann im Storyboard mit dem MapView-Baustein die Karte in die Detailansicht eingebaut werden
- Für den Zugriff auf die Karte muss – wie gehabt – auch ein IBOutlet im Controller angelegt und verbunden werden

```
#import <MapKit/MapKit.h>
...
IBOutlet MKMapView _mapView;
```

MapKit (II)

- Um die Karte auf den entsprechenden Ausschnitt zu schieben, wird die Google-API zur Abfrage von Langen- und Breitengrad genutzt
- Dazu wird per Http-Request eine Anfrage an Google gestellt, mit der Bitte die Rückgabe als CSV (comma separated values) zu liefern:

```
-(CLLocationCoordinate2D) getCoordinatesForSelection: (NSString*)selection {
    CLLocationCoordinate2D location;
    selection =  [selection stringByAddingPercentEscapesUsingEncoding:NSUTF8StringEncoding];
    NSURL* googleAPI = [NSURL URLWithString:
        [NSString stringWithFormat:@"http://maps.google.com/maps/geo?q=%@&output=csv", selection]];
    NSError* error = nil;
    NSString* googleResult = [NSString stringWithContentsOfURL:googleAPI encoding: NSASCIIStringEncoding error:&error];
    if(! googleResult ) ...
    NSArray *listItems = [googleResult componentsSeparatedByString:@","];
    if([listItems count] > 1 && [[listItems objectAtIndex:0] isEqualToString:@"200"]) {
        location.latitude  = [[listItems objectAtIndex:2] doubleValue];
        location.longitude = [[listItems objectAtIndex:3] doubleValue];
    }
    return location;
}
```

- Antwort liefert vier Werte: Status, Genauigkeit, Breiten- und. Längengrad

MapKit (III)

- Die Koordinaten werden nun zusammen mit der Größe des anzuzeigenden Ausschnittes an die Karte weitergereicht:

```
-(void) deduceMapCoordinatesForSelection: (NSString*) selection {
    MKCoordinateRegion region;
    region.center = [self getCoordinatesForSelection: selection];
    region.span.latitudeDelta = 0.05f;
    region.span.longitudeDelta = 0.05f;

    [_mapView setRegion:region];
}

- (void)viewDidLoad {
    ...
    [self deduceMapCoordinatesForSelection:resolved];
}
```

- Achtung: Für die Abfrage muss natürlich eine Internetverbindung bestehen!
- Apple hat Beispielcode bereits gestellt mit dem man überprüfen kann, ob zum aktuellen Zeitpunkt im Programm überhaupt eine Internetverbindung besteht: http://developer.apple.com/library/ios/#samplecode/Reachability/Introduction/Intro.html

Google-Maps

- Zusätzlich oder anstatt des Kartenausschnitts per MapKit kann man auch in die Google-Maps App wechseln, so denn diese auf dem Gerät installiert ist
- Im Beispiel fügt man einen neuen Button in der Detailansicht und ein neue IBAction im Controller hinzu und verbindet diese
- Über die Klasse UIApplication und die Methode openURL kann dann analog zum Http-Request eine Anfrage an das iOS-System gestellt werden:

```
NSString* resolved = [[PGKfzLookup sharedInstance] lookup: _selection];
NSString* url = [NSString stringWithFormat:@"http://maps.google.com/maps?q=%@", resolved];

[[UIApplication sharedApplication] openURL:[NSURL URLWithString:url]];
```

- Falls die Google-Map App installiert ist, führt eine URL der obigen Art direkt zu App, ansonsten über den Browser zu Web-Appikation Google-Maps
- Die Abfrage hier ist allerdings sehr grob gehalten und liefert unter Umständen mehrere und gar keine Treffer
- Auf die gleiche Art und Weise können noch weitere Apps wie etwas Mail, AppStore direkt aufgerufen werden