Praktikum

Mobile Computing

Winter 2015/2016

iOS Tutorial

1 Einrichten der Entwicklungsumgebung

Für die Entwicklung von iOS-Apps wird XCode benötigt. Laden Sie dies über den AppStore herunter.

2 Einstieg in die Programmierung mit XCode

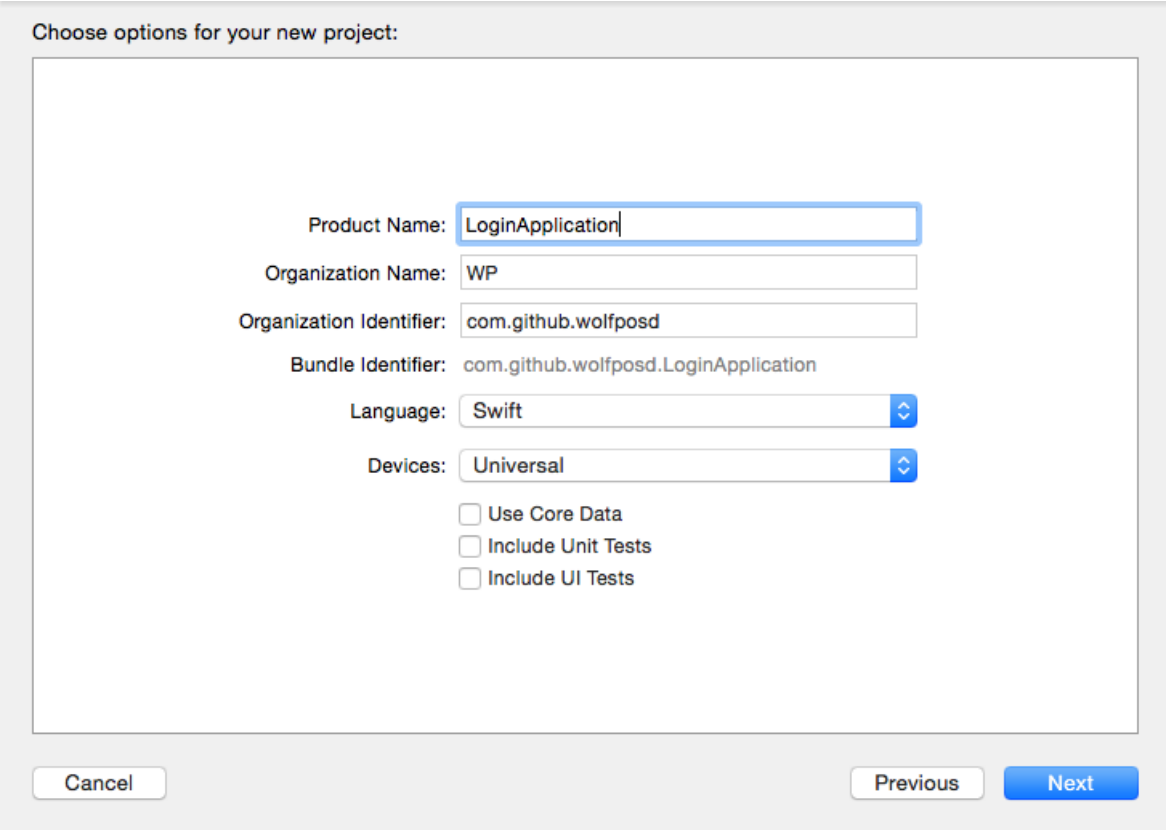
In diesem Teil des Tutorials lernen Sie ein iOS-Projekt unter XCode zu erstellen sowie eine erste, einfache Login-Anwendung zu entwickeln bzw. auszubauen. Eine Anwendung unter iOS besitzt immer eine graphische Benutzungsoberfläche. Sie wird geladen, sobald die Anwendung gestartet wird. Benutzungsoberflächen werden durch die spezielle Klasse „View" realisiert. Eine solche View repräsentiert genau eine Bildschirmseite, z.B. eine Login-Seite. Jede View besitzt einen eigenen Lebenszyklus mit unterschiedlichen Zuständen. iOS kontrolliert diesen Lebenszyklus und ruft bei einer Zustandsänderung eine festgelegte Methode der View auf, z.B. „`viewWillAppear(…)`" oder „`viewDidAppear(…)`". Beim Starten einer Anwendung wird also zunächst die viewWillAppear()-Methode der View aufgerufen. Dies ist der Einstiegspunkt für die eigene Programmierung.

2.1.1 iOS-Projekt einrichten

Starten Sie XCode. Erzeugen Sie ein neues Projekt über „File → New → New Project" in der Menüleiste. Wählen Sie iOS → Application → Single View Application

Wählen Sie für das Projekt folgende Einstellungen:

Configure your new project:

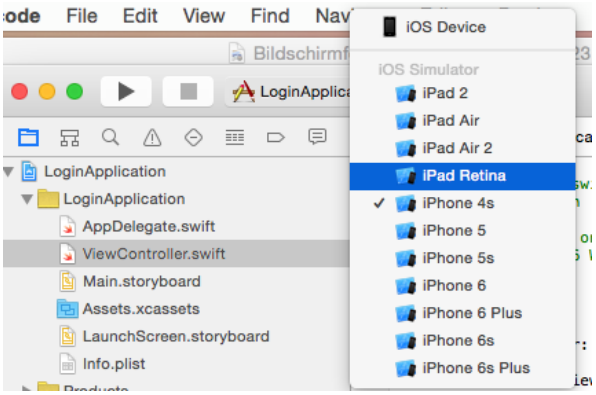


Product Name: <beliebig, wird auf dem Telefon als Anwendungsname verwendet> z.B. *LoginAnwendung*

Organization Name:	<beliebig>
Organization Identifier:	<beliebig> z.B vsis.informatik.uni-hamburg.de
Language:	Swift
Devices:	<beliebig, verwenden sie iPhone, wenn Sie keine iPads unterstützen wollen, bei Universal fällt größerer Aufwand an da Views individuell angepasst werden müssen>

Als Ordner für Ihr Projekt geben Sie bitte Ihr Home-Verzeichnis („~/“) an, z.B. „~/Xcodeprojects/“.

2.2 Erste iOS Anwendung erstellen



Starten Sie das eben erzeugte Project über den “Play”-Button oben links. Über die Schaltfläche daneben, können Sie das simulierte Gerät auswählen, z.B.: “iPhone 6s”.

Der Simulator sollte nun starten und eine weiße View anzeigen.

Sollten Sie keine Geräte zur Auswahl haben, muss ggfs. der Simulator noch installiert werden. Dazu drücken Sie `CMD` und `Komma` (oder Menüreiter: XCode → Preferences), und wählen im “Downloads”-Tab einen Simulator zum herunterladen an, z.B. “iOS 9.0 Simulator”.

2.2.1 View bearbeiten

In iOS werden alle Views im sog. Storyboard verwaltet (erkennbar am Namen „XYZ.storyboard“).

Hinweis: Es besteht auch die Möglichkeit Views programmatisch oder über Einzeldateien (.xib, ausgesprochen nib) anzulegen. Die bevorzugte Variante ist heutzutage allerdings das Storyboard.

Fügen Sie der View ein *Text Field* und einen *Button* hinzu. Da iOS eine Aufteilung zwischen Design und Funktionalität erlaubt, werden grafische Elemente nicht in ObjC/Swift definiert, sondern im Storyboard.

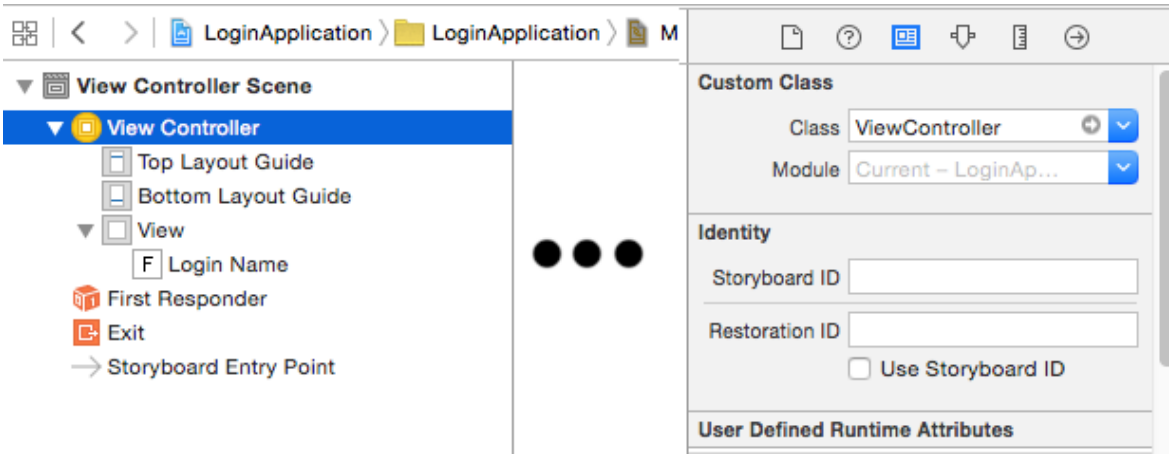
Ändern sie den Text des Buttons zu „Login“ und den „Placeholder Text“ zu Login Name.

Starten sie – ohne den Simulator zu schließen – die App erneut über den Play-Button.

2.2.2 ButtonAction

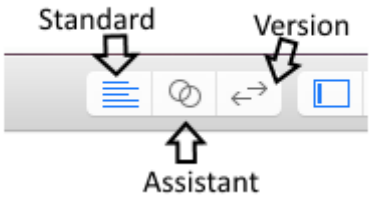
Beim Drücken des Buttons, soll eine *Toast Notification* angezeigt werden. Hierfür muss in der dazugehörigen Klasse die Logik implementiert werden.

In iOS wird das MVC (Model-View-Controller) Muster verwendet um grafische Oberflächen zu Erstellen. Das heißt, dass zu jeder View (im Storyboard) eine ViewController-Klasse gehört.



Wenn sie im Storyboard in der „View Controller Scene" (links) einen Controller auswählen, erscheint im Rechten Bereich (im Identity Inspector) die zugehörige Klasse.

Wechseln Sie nun die Ansicht des Editors von "Standard editor" auf "Assistant editor".



Der Editor sollte nun zweigeteilt erscheinen. Auf der Linken Seite ist weiterhin das Storyboard, während auf der rechten Seite nun die ViewController.swift-Klasse erscheint.

Um eine Aktion im Storyboard mit einem Code-Schnipsel zu verknüpfen muss eine `@IBAction` angelegt werden. Dazu **ziehen** Sie mit **gedrückter rechter Maustaste** den Button in das Code-Fenster. Im erscheinenden Dialog wählen Sie "Action" einen Namen, z.B. "loginButtonTouchUpInside" und bestätigen mit "Connect". Nun wird eine leere Methode (`@IBAction func login..`) erstellt die aufgerufen wird, wenn auf den Button gedrückt wird.

2.2.3 Textfeld verknüpfen

Um eine Code-Referenz auf das Textfeld zu Erstellen, ziehen Sie wieder mit grdrückter rechter Maustaste das Textfeld in das Code-Fenster. Wählen Sie diesmal "Outlet" als Connection, geben Sie einen passenden namen (z.B.: loginTextField) und bestätigen sie mit Connect.

Nach diesen Schritten sollte ihr Code in etwa so aussehen:

```
class ViewController: UIViewController {

    @IBOutlet weak var loginTextField: UITextField!

    override func viewDidLoad() {
        super.viewDidLoad()
```

```
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }

    @IBAction func loginButtonTouchUpInside(sender: AnyObject) {
       // Methode vom Login-Button
    }
}
```

Für gewöhnlich werden Outlets oben in der Klasse angelegt (ähnlich wie Felder in Java) und Actions weiter unten im Quelltext.

2.2.4 Zugriff auf Felder und Variablen/Konstanten

In Swift (und ObjC 2.0) können auf Klassenfelder per Punktnotation zugegriffen werden. Dies geschieht nicht über das Keyword "`this`" sondern über "`self`".

Um den Text aus dem Textfeld zu lesen sollten Sie diesen zuerst in einer Variablen (oder Konstanten) speichern.

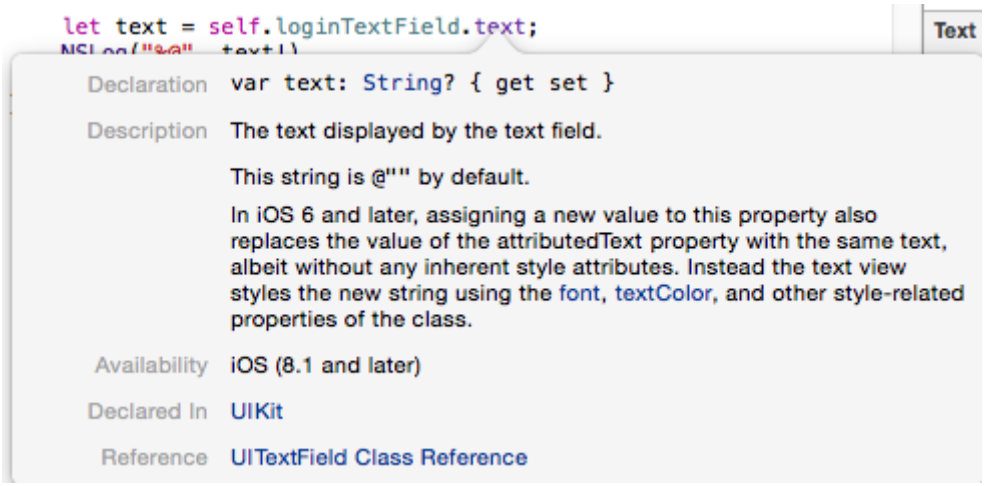
Anders als in Java, wo man bei Variablen den Typ immer angeben muss, können in Swift Variablen dynamisch angelegt werden. Dies geschieht mittles "`var`". Konstanten benutzen das Keyword "`let`". Möchte man den Typ dennoch gerne spezifizieren, kann dies durch einen Doppelpunkt gefolgt vom Typ geschehen:

```
var str: String = "myString"
// eine String Variable

var text2 = self.loginTextField.text
// irgendeine Variable, zur Zeit vom Typ String

let text = self.loginTextField.text
// eine String Konstante
```

2.2.5 Optionals

Eins der Charakteristischen Sprachfeatures von Swift sind die Optionals. Als Entwickler haben wir die Möglichkeit anderen Entwicklern, die vielleicht Teile unseres Programmecodes wiederverwenden, darauf hinzuweisen das einige Werte nicht vorhanden sein müssen (Null-Objekte, in Swift/ObjC „`nil`").

Optionals erkennt man am nachgestellten Fragezeichen. Dies können wir zum Beispiel in der Dokumentation der "`text`"-Methode unseres Loginfeldes sehen. Dazu Klicken Sie während Sie die ALT-Taste gedrückt halten, auf die "`text`"-Methode.



Um eine optionale Variable zu "entpacken", also den konkreten Wert auszulesen, wird ein Ausrufungszeichen nachgestellt.

```
var optionalText :String? // eine Optionale String Variable
var text = optionalText!  // entpackter String
```

Um nun gleichzeitig eine Optionale Variable zu entpacken, den Wert auf Existenz zu prüfen und weiterzuverwenden wird folgendes Konstrukt verwendet:

```
if let text = self.loginTextField.text
{
    // ab hier kann die Konstante text verwendet werden
}
else
{
    // text is nil
}
```

2.2.6 System.out.printn in Swift

Ein häufiges Mittel beim Testen von Anwendungen ist die Ausgabe von Statusinformationen auf die *Konsole* mittels der Anweisung `System.out.println(„...“)`.

In Swift und Objective-C wird hierzu die Funktion NSLog verwendet. Die Signatur sieht folgendermaßen aus:

```
public func NSLog(format: String, _ args: CVarArgType...)
```

Die NSLog Funktion nimmt einen „Formatierungsstring“ und eine beliebige Anzahl an Parametern entgegen (vergleichbar mit System.out.format(…) in Java oder sprintf(…) in C/C++).

Ein konkreter Aufruf von NSLog sieht etwa so aus:

```
NSLog("Hier kann text stehen")
NSLog("Text: %@", text) // let text = "text"
NSLog("%@,%@,%@,%@, %d", "1", "2", "3", "4", 5)
```

Als Formatierungszeichen werden hier `%@` für Strings und `%d` für Zahlen verwendet. Eine Vollständige Liste von Formatierungszeichen findet sich hier:

https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/Strings/Articles/formatSpecifiers.html

Natürlich kann man auch wie in anderen Entwicklungsumgebungen Breakpoints setzten um Apps zu debuggen. Dazu klicken Sie einfach auf die Code-Zeilenanzeige am linken Editorrand um einen Breakpoint zu setzen. Standardmäßig werden in XCode alle Apps im Debugmodus ausgeführt.

3 Versionsverwaltung mit Subversion

Subversion (SVN) ist eine Open-Source-Software zur Versionsverwaltung von Dateien und Verzeichnissen. Die Versionierung erfolgt in einem zentralen Repository, so dass Änderungen an den Inhalten verteilt auf den Computern der Bearbeiter ausgeführt werden können.

3.1 Projekte zur Versionsverwaltung hinzufügen

Da SVN nichtmehr die bevorzugte Versionsverwaltung in OSX ist, ist der Support in XCode leider auch etwas beschränkt. Deshalb müssen Sie einige Befehle im Terminal ausführen.

Öffnen sie das Terminal in Applications -> System Applications -> Terminal. Alternativ starten Sie es mit Spotlight: (CMD + Leertaste) und „Terminal" eintippen.

Folgende Befehle laden das leere SVN herunter in einen Ordner namens „svnfolders"

```
mkdir svnfolders
cd svnfolders
svn checkout svn+ssh://LOGINNAME@rzrepository.informatik.uni-hamburg.de/informatik/viks/vsis/repositories/lehre/mc2016/mcXX/
cd mcXX
```

Schließen Sie XCode. Bewegen Sie ihr aktuelles Projekt im Finder in den soeben erstellten Ordner ~/svnfolders/mcXX/.

Wieder im Terminal:

```
cd mcXX
svn status
```

Der Output von „svn status" sollte etwa so aussehen:

```
MacVM:mc00 wolf$ svn status
?       LoginApplication
```

Um nun die Dateien zum hinzufügen zu markieren, schreiben Sie im Terminal:

```
svn add LoginApplication
```

Starten sie ihr Projekt erneut in XCode über die .project Datei ihres Projektes. Im Projekt-Explorer (links) sollten nun alle neuen Dateien mit einem „**A**" versehen sein. Wählen Sie Source-Control → Commit aus um ihr Projekt einzuchecken.

Verwenden Sie **IMMER** eine sinnvolle aussagekräftige Commit-Message!

Hinweis: Folgende Dateien stellen lokale Einstellungen dar und sollten nicht eingecheckt oder committet werden:

- User Data
- Workspace → User Data

3.2 Projekte aus der Versionsverwaltung herunterladen

Um das soeben zur Verfügung gestellte Projekt nun auf einem anderen Rechner weiterzubearbeiten führen Sie einen sog. *Checkout* aus.

In den Einstellungen (CMD + KOMMA, oder XCode→Preferences) erstellen Sie unter dem Accounts-Tab einen neues Repository vom Typ SVN mit der Adresse:

```
svn+ssh://LOGINNAME@rzrepository.informatik.uni-
hamburg.de/informatik/viks/vsis/repositories/lehre/mc2016/mcXX/
```

In der Menüleiste: Source Control → Checkout → Subversion

Folgen Sie dem Dialog und Laden das Repository auf ihre Festplatte.

3.3 Daten aktualisieren

Haben Sie lokal Änderungen durchgeführt und möchten diese in das zentrale Repository hochladen, wählen Sie im Menüband *Source Control → Commit.* Dateien die verändert wurden sind mit einem „M" (für modified) gekennzeichent.

Um Ihr lokales Projekt mit den Daten aus dem zentralen SVN-Repository zu aktualisieren, wählen Sie im Menüband *Source Control → Update.*

Wichtig!

Jede logische (zusammenhängende und getestete) Änderung sollte umgehend „committed" werden, damit sie den anderen Team-Mitgliedern zur Verfügung steht. Ebenso wichtig sind häufige Updates, um frühzeitig Änderungen der anderen Mitglieder berücksichtigen zu können.

Wenn zwei Mitglieder dieselbe Datei bearbeiten, kann nur eine Partei die Änderungen direkt hochladen. Ein Versuch der anderen Partei die Änderungen im zentralen Repository zu überschreiben, scheitert (siehe SVN Console). Hier muss vor dem Commit zunächst lokal aktualisiert werden. Sofern die jeweiligen Änderungen unterschiedliche Bereiche einer Datei betreffen (z.B. unterschiedliche Methoden), werden die unterschiedlichen Versionen automatisch während der Aktualisierung zusammengeführt (engl. „merge"). Betrafen die Änderungen denselben Bereich einer Datei, entsteht ein „Konflikt", der manuell behoben werden muss. Eine entsprechende Kennzeichnung der Konfliktstelle findet sich in der Datei.

Mittels „Source Control → Mark selected Files as resolved" kann nach der Konfliktbehandlung das Problem als gelöst markiert und die Datei anschließend per Commit hochgeladen werden.

4 Ein bisschen Swift

In diesem Kapitel werden einige Besonderheiten, Features und die Syntax von Swift kurz vorgestellt. Viele Sachen lassen sich einfach im Playground ausprobieren. Der Playgrund ist ein REPL-Environment (Run-Eval-Print-Loop) in dem man direkt „ohne" Compilieren Swift-Code schreiben und auswerten lassen kann. Starten Sie zum ausprobieren einiger Beispiele einen neuen Playground (File → New → Playground)

4.1 Kontrollstrukturen, Schleifen, etc

Wie in jeder höheren Programmiersprache gibt es auch in Swift alle gängigen Mechanismen wie if-then-else, oder for-Schleifen.

```
if ("12" == "12") { // Strings können mit == verglichen werden
    // Klammern sind optional, tragen aber zur Lesbarkeit bei
    "12 ist gleich 12?"
}
else {
    "verschieden"
}
```

For-Schleifen:

```
for (var i = 0; i < 10; i++)
{
    i
}
```

For-Each Schleife:

```
var array = [1,2,3,4]

for number in array
{
    number
}
```

While und Do-While Schleife:

```
var number = 12
while( number > 0){
    number--
}

var number = 12
repeat {
    number--
} while (number > 0)
```

Switch:

```
var text = "hallo"
switch(text)
{
```

```
    case "hallo": text+text
    case "peter" : "wer ist peter?"
    default: "default antwort"
}
```

4.2 Klassen, Methoden und Felder

Wie in Java bestehen in Swift/ObjC Klassen aus den selben Bausteinen. Es gibt Felder (Instanzvariablen), Methoden und Konstruktoren. Genauso können Klassen auch von anderen Klassen erben.

Eine Klassendefinition für eine User-Klasse könnte etwa so aussehen:

```
public class User {

    let vorname:String // final Feld

    // Konstruktor mit einem Param.
    init(vorname:String) {
         self.vorname = vorname
    }
    // Methode ohne Params, gibt String zurück
    private func whatsMyName() -> String {
        return self.vorname
    }

    internal func countLength() -> Int {
        return self.vorname.utf16.count
    }

    func squareMe(x:Int) -> Int {
        return x*x
    }
}


var peter = User(vorname:"Peter") // neuen User anlegen
peter.vorname                     // gibt den Vornamen
peter.whatsMyName()               // gibt den Vornamen
peter.countLength()               // gibt die Länge
peter.squareMe(12)                // gibt 144 aus
```

Um Methoden oder Klassen bestimmte Zugriffsrechte zu geben gibt es folgende Sichtbarkeits-Modifier:

public	Jeder hat Zugriff
internal	Alle aus dem Selben Modul haben Zugriff
private	Nur die Klasse hat Zugriff

Die Standard Sichtbarkeit ist „internal" und muss nicht explizit angegeben werden.

5 Datenbankzugriff für die Server-Anwendung (optional)

Der Zugriff auf eine Datenbank kann über vielerlei Wege erfolgen, z.B. mittels existierender Programme wie dem *Squirrel SQL-Client*, aber auch mittels selbst erstellter Anwendungen. Beide Wege sollen im Folgenden beschritten werden.

5.1 Squirrel SQL-Client

Auf den VSIS-Poolrechnern ist der Squirrel SQL-Client inkl. des benötigten MySQL-Treibers bereits installiert. Zum Starten nutzen Sie das entsprechende Icon auf dem Desktop oder rufen Sie die Datei „squirrel_sql.bat" im Verzeichnis „D:\Programme\Squirrel-sql-3.7" auf. Möchten Sie von ihrem eigenen Notebook auf die Datenbank zugreifen, finden Sie im Anhang eine Anleitung zur Einrichtung.

5.1.1 Testen der Verbindung

Für die Verbindung zur Datenbank benötigen Sie einen *Alias*. Dabei handelt es sich hier um eine Instanz einer Treiberkonfiguration, welche die Daten für eine Verbindung zu einer bestimmten Datenbank auf einem bestimmten Server enthält. Ein neues Alias lässt sich in der Alias-Sicht anlegen. Aus dem Menü *Windows* wählen Sie *View Aliases* und klicken im linken Menü auf das kleine Pluszeichen. In dem sich öffnenden Fenster *Add Alias* Geben Sie folgende Daten an:

Name:	MobiCom
Driver:	MySQL Driver
URL:	jdbc:mysql://vsisls4.informatik.uni-hamburg.de/mc**XX**?useUnicode=true&characterEncoding=UTF8
	User Name: mc**XX** (wie Gruppenkennung)
	Password: Das Gruppenpasswort (Betreuer fragen)

Mit dem Button *Test* und anschließend *Connect* finden Sie heraus, ob die Verbindung erfolgreich war. Abschließend klicken Sie auf *OK*, um das Alias zu speichern.

5.1.2 Neue Tabelle anlegen

Die Datenbank kann nun verwendet werden. Klicken Sie mit der rechten Maustaste auf das neue Alias und wählen Sie *Connect*.

Legen Sie nun mittels eines SQL-Statements (CREATE TABLE) eine neue Tabelle mit dem Namen „benutzer" an. Die Tabelle soll über folgende Attribute verfügen:

```
Kennung, Vorname, Nachname, Passwort und E-Mail-Adresse
```

Verwenden Sie dabei geeignete Datentypen für die Felder und vergeben Sie einen Primärschlüssel.

5.1.3 Insert into

Fügen Sie mittels eines SQL-Statements (INSERT INTO) die Mitglieder Ihres Teams als Benutzer in die soeben erstellte Tabelle ein.

5.1.4 Select

Lassen Sie sich mit einer SQL-Abfrage (SELECT) alle Tupel der Tabelle „benutzer" ausgeben. Haben Sie dies erfolgreich gemeistert, soll in einem nächsten Schritt der Zugriff auf die Datenbank von einem eigenen Java-Client aus erfolgen.

5.2 Java-Client mit JDBC unter Android Studio

Folgende Schritte beziehen sich auf die Nutzung von Android Studio. Eclipse oder andere Java-Entwicklungsumgebungen sind allerdings genauso gut geeignet. XCode selbst bietet keine Java-Unterstützung mehr.

Für das automatische einbinden von Bibliotheken bietet sich in Eclipse das Erstellen eines Maven-Java Projektes an. Bitte fragen Sie bei Bedarf einen Betreuer.

5.2.1 Neues Modul

Starten Sie Android Studio und erzeugen Sie unter *File->New->New Module* ein neues „Java-Library" Projekt. Wählen Sie „MobiCom-Tutorial-Datenbank" als Projektnamen. Wählen Sie als Klassennamen „JDBCTest" und erstellen Sie in der neu erzeugten Klasse eine Main-Methode.

5.2.2 MySQL Connector

Um auf die MySQL-Datenbank zugreifen zu können, benötigen Sie eine entsprechende Java-Bibliothek, die Ihnen den Zugriff erleichtert. Auf der Praktikumsseite (unter „Literatur") finden Sie einen Link zur entsprechenden Bibliothek (*MySQL Connector/J*):

https://vsis-www.informatik.uni-hamburg.de/vsis/teaching/coursekvv/363

Nach dem Entpacken des Archivs, speichern Sie die Datei *mysql-connector-java-<version>-bin.jar* in ihrem soeben neu angelegten Modul. Ziehen sie dazu die .jar-Datei in den „libs"-Ordner des Moduls (Wechseln Sie ggf. in Android Studio die „Project"-Perspektive am oberen linken Fensterrand). Über „Rechtsklick auf JAR → Add as Library → Aktueller Modulname, fügen Sie die Library zum „Buildpath" hinzu.

Alternativ können Sie die benötigte Bibliothek automatisch einbinden lassen. Dazu fügen Sie es als Dependency zum Build-Skript hinzu. Öffnen Sie die *build.gradle* ihres Datenbank-Modules und passen Sie das Skript an, damit es etwa wie folgt aussieht:

```
apply plugin: 'java'

dependencies {
    compile fileTree(include: ['*.jar'], dir: 'libs')
    compile 'mysql:mysql-connector-java:5.1.38' //hinzufügen
}
```

Nach dem Speichern der Datei wird eine Warnmeldung (Gradle files have changed since last project sync) erscheinen, welche mit „Sync Now" am rechten Bildschirmrand bestätigt werden muss. Sollte diese Meldung nicht erschienen sein muss der Prozess manuell über „Tools → Android → Sync Project with Gradle Files" gestartet werden. Nun werden alle benötigten Bibliotheken automatisch von Android Studio heruntergeladen und hinzugefügt.

5.2.3 DB Verbindung

Zunächst muss eine Verbindung mit der Datenbank hergestellt werden. Dies geschieht durch das Laden des Datenbanktreibers und der Angabe der Serverdaten (s.u.). Importieren Sie dazu in der Klasse „JDBCTest" das Paket `java.sql.*` und fügen Sie den folgenden Code in die Main-Methode ein. Sorgen Sie zudem für eine angemessene Fehlerbehandlung.

```
Class.forName("com.mysql.jdbc.Driver");
String url= "jdbc:mysql://vsisls4.informatik.uni-
                hamburg.de/mcXX?useUnicode=true&characterEncoding=UTF8";
```

```
String username ="mcXX";
String password ="XXX";
Connection con = DriverManager.getConnection(url, username, password);
```

5.2.4 Testen der Datenbank

Testen Sie die Datenbankverbindung durch eine einfache Abfrage. Lassen Sie sich dazu aus der Datenbank alle Tupel aus der Tabelle „benutzer" auf der Konsole ausgeben. Sie können dafür zum Beispiel wie folgt vorgehen:

```
String sql = "SELECT * FROM benutzer";
Statement st = con.createStatement();
ResultSet rs = st.executeQuery(sql);
while (rs.next()) {
      String vorname = rs.getString("Vorname");
      String nachname = rs.getString("Nachname");
      System.out.println(vorname + " " + nachname);
}
```

5.2.5 DB schließen

Nicht vergessen(!): Ist der Zugriff auf die Datenbank abgeschlossen, wird die Verbindung zur Datenbank beendet.

```
con.close();
```

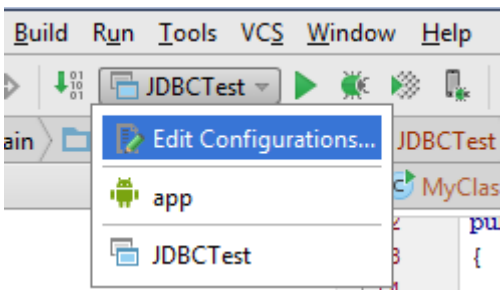
5.2.6 Run Configuration

Um dieses Modul in Android Studio zu starten muss eine neue „Run Configuration" angelegt werden.

Dazu klicken Sie auf „Run → Edit Configurations" in der Menüleiste. Fügen Sie eine neue Konfiguration vom Typ „Application" (nicht „Android Application") hinzu und füllen Sie die Felder passend aus:

Name:	JDBCTest
Main Class:	de.vsis.JDBCTest
Use classpath of module:	MobiCom-Tutorial-Datenbank

Wählen Sie anschließend *JDBCTest* aus den „Run Configurations" aus und drücken Sie den Play-Button.



6 Entwicklung einfacher Client-Server-Anwendungen (optional)

Das *Client-Server-Modell* ist das Standardkonzept für die Verteilung von Aufgaben innerhalb eines Netzwerks. Ein *Server* ist dabei ein Programm, das eine bestimmte Software-Funktionalität bereitstellt. Im Rahmen des Client-Server-Konzepts können andere Programme, die *Clients*, diese Funktionalität nutzen. Dabei können sich Clients und Server durchaus auf unterschiedlichen Rechnern befinden – sie müssen es aber nicht unbedingt! Das Programmieren und Testen von einfachen Client-Server-Anwendungen kann daher auch relativ komfortabel auf einem einzigen Rechner erfolgen und nach Fertigstellung auf die Zielrechner verteilt werden. Die folgenden Abschnitte dieses Tutorials zeigen ein einfaches Beispiel für eine Client-Server-Beziehung zwischen zwei Java-Anwendungen mittels *Sockets*.

Viele moderne Webseiten und –Applikationen (Facebook, Twitter, etc.) verwenden darauf aufsetzend ein Programmierparadigma namens REST. Basierend auf dem Hypertext Transfer Protocol (HTTP/S) nimmt der Server über URL-Pfade und HTTP-Requests Anfragen entgegen und antwortet mit maschinenlesbaren kodierten Daten (meist JSON oder XML). Für die Entwicklung einer Client-Server-Anwendung mittels eines REST-Servers liegt ein gesondertes Tutorial vor. Dieses ist besonders empfehlenswert für Sie, wenn Sie bereits Erfahrung in der Socket-Programmierung besitzen und auch sonst ein erweitertes Grundverständnis von verteilten Anwendungen haben.

Vorteilhaft bei der Gestaltung einer eigenen socket-basierten Anwendung ist die Zwei-Wege-Kommunikation. Während bei HTTP/REST-Servern nur ein *„Pull"* der Daten möglich ist, können bei eigenen Anwendungen die Server auch proaktiv Daten an Clients senden (*„Push"*), z.B. um diese zeitnah über Neuigkeiten zu informieren.

6.1 Entwicklung der Server-Anwendung

6.1.1 Neues Modul

Erstellen Sie wie bereits im vorausgegangenen Kapitel ein neues Java-Library Modul in Android Studio. Wählen Sie einen geeigneten Modulnamen, z.B. „ServerProjekt".

Erstellen Sie in dem Projekt eine neue Klasse mit Main-Methode für den Start der Server-Anwendung.

6.1.2 ServerSocket erstellen

Erstellen Sie (z.B. in Ihrer Main-Methode) ein neues Exemplar der von Java bereitgestellten Klasse `ServerSocket`, um eingehende Kommunikationsverbindungen auf einem bestimmten Port ihres Rechners ihrer neuen Anwendung anzunehmen. Wählen Sie dazu einen beliebigen (freien) Port (z.B. 6666):

```
ServerSocket serverSocket = new ServerSocket(6666);
```

Nun muss der Server dazu gebracht werden, auf eingehende Verbindungen zu warten, was über den Aufruf der Methode `accept()` geschieht. Dabei wird die Verbindung zum Client, welcher den Server aufruft, als `Socket`-Objekt zurückgegeben. Dieser Socket stellt sozusagen eine Repräsentation der Verbindung zum (entfernten) Client dar und ist notwendig, um auf die vom Client übertragenen Daten zugreifen zu können und dem Client eine Antwort zu schicken:

```
Socket socket = serverSocket.accept();
```

Wenn eine eingehende Verbindung ankommt, läuft der Programmfluss weiter und die eingehenden Daten können verarbeitet werden. Da die Daten jedoch als *Streams* (d.h. als geordnete Folge von Bytes mit zumeist unbekannter Länge) transportiert werden, müssen diese zunächst durch geeignete *Reader* bzw. *Writer*-Klassen gelesen bzw. geschrieben werden. Im Anschluss können die eingehenden Daten verarbeitet und ggf. eine Antwort an den Client generiert werden. Diese wird dann wiederum als Stream zurückgesendet und nach Abschluss der gesamten Kommunikation werden alle *Reader* und *Writer* sowie der *Socket* geschlossen:

```
BufferedReader in = new BufferedReader(
                        new InputStreamReader(socket.getInputStream())
                    );

String request = in.readLine(); // Eine Zeile verarbeiten


// Beginn der anwendungsspezifischen Verarbeitung der Client-Anfrage
String response = "Hello " + request;
// Ende der Verarbeitung

PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
out.println(response);
out.close();
in.close();
socket.close();
```

Soll der Server irgendwann beendet werden, muss zudem der ServerSocket geschlossen werden:

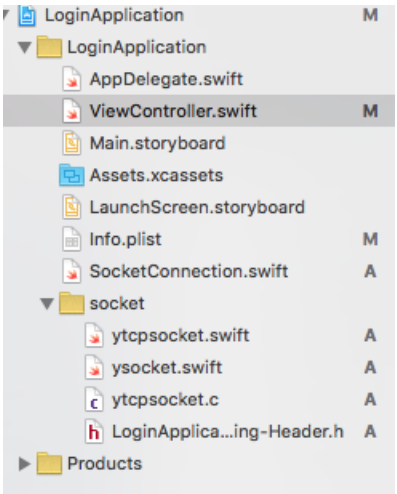
```
serverSocket.close();
```

Da bei der Kommunikation im Allgemeinen vielfältige Fehler auftreten können (z.B. falls der angegeben Port belegt ist), muss darüber hinaus für eine angemessene Fehlerbehandlung gesorgt werden (hier nicht gezeigt).

6.2 Entwicklung der Client-Anwendung

6.2.1 Bibliothek importieren

Importieren Sie die Socket-Bibliothek[1] in XCode. Dazu Entpacken Sie das auf der VSYS-Seite bereitgestellte Archiv und ziehen die drei Dateien in Ihr XCode Projekt. Sollte ein Dialog über das Erstellen von „Bridging-Headern" erscheinen, bestätigen Sie diesem mit Ja. Wählen Sie außerdem die Option „Copy Files to Project". Es bietet sich an für die neuen Dateien eine Gruppe anzulegen und diese dort zu platzieren.



6.2.2 Verbindung herstellen

Benutzen Sie die TCPClient Klasse um eine neue Socket-Verbindung herzustellen. Die TCP Klasse bieten zum Verbinden, Lesen und Schreiben entsprechende Methoden an:

[1] Socket Bibliothek: https://github.com/swiftsocket/SwiftSocket

```
// Verbindet den Client
let (success,msg) = tcpclient.connect(timeout: 10)

// Sendet einen String
let (success,msg) = tcpclient.send(str: "this is my data\n")
// Wichtig: Senden Sie unbedingt \n als Terminierungssymbol

// liest die nächsten 300.000 bytes
client.read(300000, timeout: 10)
```

Die Methoden `send` und `connect` geben als Rückgabewerte ein Tupel aus Boolean/Stirng zurück. Diese müssen benutzt werden um erfolgreiches Verbinden zum Server und Senden der Daten zu bestätigen.

6.3 Verarbeitung der Anfragen von mehreren Clients

Wie Ihnen vielleicht aufgefallen ist, kann ihr Server bisher nur eine einzige Anfrage von einem einzelnen Client verarbeiten. Um auch mehrere Anfragen gleichzeitig entgegennehmen zu können, müssen Sie diese in jeweils eigenen leichtgewichtigen Prozessen verarbeiten. Hierfür stellt Java das Konzept der *Threads* bereit.

6.3.1 RequestHandler

Fügen Sie Ihrem Server-Projekt eine neue Klasse namens „RequestHandler" hinzu, welche von der von Java bereitgestellten Klasse `Thread` erbt. Von dieser Klasse soll nun für jeden eingehenden Aufruf eines Clients eine neue Instanz erstellt werden, welche sich um die Bearbeitung genau dieser Anfrage kümmert. Durch die Verwendung eines so entstehenden neuen Threads können Anfragen mehrerer Clients dann *nebenläufig* bearbeitet werden.

Ein Thread muss stets die Methode `public void run()` implementieren, welche den nebenläufig auszuführenden Code enthält und beim Start des Threads automatisch aufgerufen wird. In diesem Fall können Sie hier als Verarbeitungslogik die im Server-Teil des Tutorials dargestellte Stream-Verarbeitung durchführen. Damit alle hierfür benötigten Informationen zur Verfügung stehen, übergeben wir das vom Server beim Eintreffen einer neuen Anfrage erzeugte *Socket* an den Konstruktor des *RequestHandlers*:

```
public class RequestHandler extends Thread {

	private Socket socket = null;

	public RequestHandler(Socket socket) {
		super("RequestHandler");	// Name des Threads
		this.socket = socket;
	}

	public void run() {
		// Nebenläufig auszuführende Verarbeitungslogik
	}
}
```

6.3.2 Threads erstellen

Nun muss beim Eintreffen einer neuen Verbindung nur noch eine neue Instanz des `RequestHandlers` erzeugt und der Thread gestartet werden. Dies geschieht über die Methode `start()`, welche wir zum Beispiel aus einer (Endlos-)Schleife aufrufen können, welche permanent auf neue eingehende Verbindungen lauscht. Die Implementierung für die Server-Klasse unserer Server-Anwendung reduziert sich damit (ohne Beachtung der Fehlerbehandlungsroutinen) auf den folgenden Code:

```
ServerSocket serverSocket = new ServerSocket(6666);

boolean listening = true;
while (listening) {
       new RequestHandler(serverSocket.accept()).start();
}

serverSocket.close();
```

6.3.3 Testen

Testen Sie Ihre neue Server-Anwendung mit einem oder mehreren Clients. Sie können auch versuchen, die Anwendung auf mehrere Rechner zu verteilen.

6.4 Integration des Datenbank-Zugriffs

Erweitern Sie Ihre Client-Server-Anwendung nun so, dass der Server nach der Übermittlung einer korrekten Kennung mit Passwort eine Liste der Namen (Vorname und Nachname) der in der Datenbank eingetragenen Benutzer zurücksendet (vgl. Kapitel 4). Verwenden Sie bei der Datenbank-Abfrage für das Prüfen der Kennung ein `PreparedStatement`, um Ihre Anwendung vor böswilligen Angriffen zu schützen. Die Übertragung zwischen Client und Server können Sie als einfache Zeichenkette realisieren, wobei Kennung und Passwort jeweils einzeln in einer Zeile stehen und auch bei der zurückzugebenen Liste der Namen jeweils ein Name pro Zeile übertragen wird.

Machen Sie sich dabei auch Gedanken, wie auf Fehlerfälle (z.B. falsches Passwort, Verbindung zur Datenbank gestört) geeignet reagiert werden kann und berücksichtigen Sie dies bei Ihrer Implementierung.

7 iOS Anwendung zur Darstellung des Datenbankabrufs erweitern

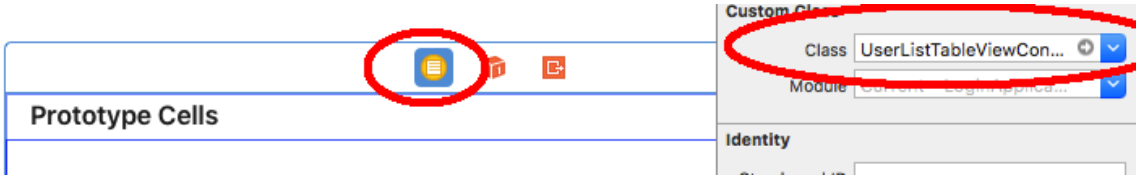
7.1 UI Ausbauen

Bauen Sie ihren Login-View aus, damit er außer einem Nutzernamen auch ein Passwort entgegennimmt.

Verändern Sie die Funktionalität des Loginbuttons so, dass die App sich beim Klick mit dem Server über User/Password authentifiziert. Der Server soll bei einer gültigen Authentifikation mit einer Liste der Nutzer antworten.

Erstellen Sie im Storyboard eine neue View vom Typ *„TableViewController"*. Erstellen Sie außerdem eine Neue Klasse *UserListTableViewController* und lassen sie von *UITableViewController* erben.

Verknüpfen Sie die View mit der Klasse im Storyboard:



Setzen Sie ebenfalls die Storyboard-ID der View auf *„UserListTableViewController"*

Um den neuen Controller anzuzeigen verwenden Sie folgenden Code:

```
let storyboard = UIStoryboard(name: "Main", bundle: nil)

let viewcontroller =
storyboard.instantiateViewControllerWithIdentifier("UserListTableViewCon
troller") as? UserListTableViewController

self.presentViewController(viewcontroller!,animated: true, completion:
{})
```

Sie sollten vor der „Präsentierung" des Controller die vom Server ankommenden Daten an diesen übergeben.

7.2 Threading

Um in iOS nicht den UI-Thread zu blockieren gibt es mehrere Möglichkeiten. Es ist möglich bestimmte Methoden in einem Background-Thread oder auf dem Main-Thread auszuführen.

```
performSelectorOnMainThread(Selector(meineMethode()),  withObject:
nil, waitUntilDone: false);
```

Es gibt auch eine C-Level Funktion die in Swift wesentlich einfacher und eleganter aussieht:

```
let myqueue = dispatch_get_global_queue(
                    DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);
```

```
dispatch_async(myqueue) {
    // im Hintergrund Thread

    dispatch_async(dispatch_get_main_queue()) {
            // im UI Thread
    }
}
```

Verwenden Sie obigen Code um die Server-Abfrage asynchron zu starten und präsentieren dann den Controller wieder aum dem UI-Thread. Für gewöhlich werden *UIActivityIndicators* benutzt um den Nutzer auf etwaige Verzögerungen hinzuweisen.

7.2.1 UITableViewController

Um die TableView mit Funktionalität zu bestücken, müssen seine Methoden implementiert werden.

Generell empfiehlt es sich bei TableViews ein Feld vom Typ Array anzulegen in denen die Daten gespeichert werden.

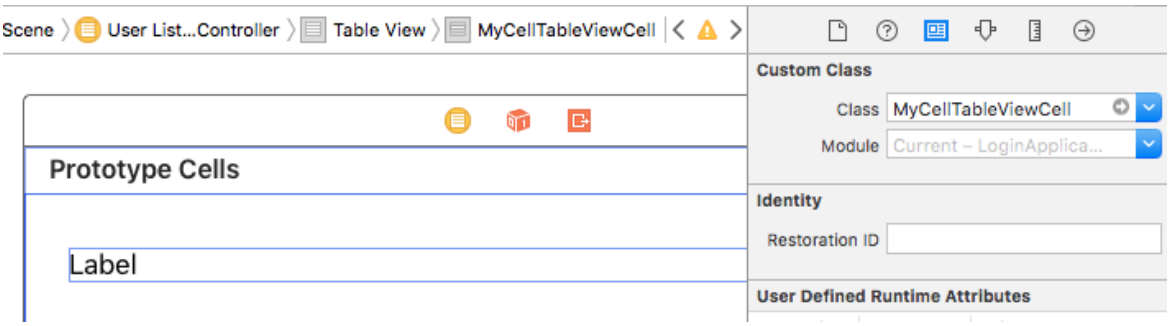
So kann z.B. für die Methode *tableView....numberOfRowsInSection*

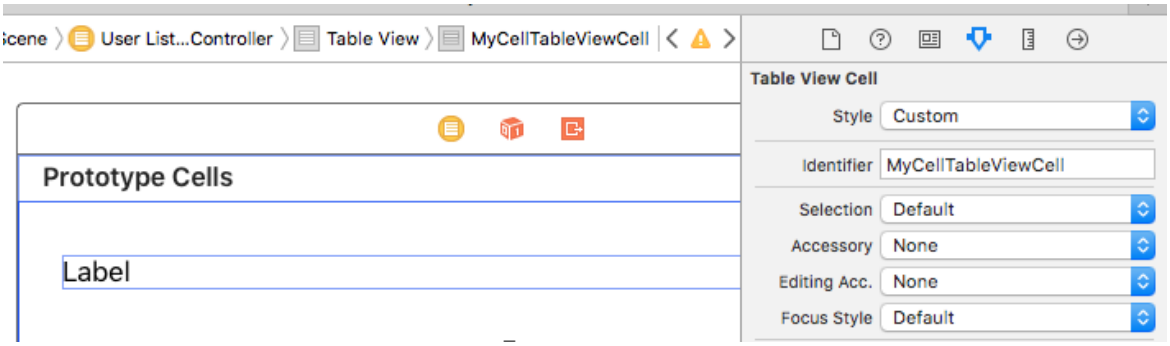
```
return self.array.count
```

zur Bestimmung der Elemente verwendet werden. Da zurzeit nur eine „Section" vorhanden ist, kann der *section*-Parameter ignoriert werden.

Implementieren sie zusätzlich noch die Methode tableView...numberOfSections indem Sie „1" zurückgeben.

Dekorieren Sie die Prototype-Cell im Storyboard ihres UITableViewControllers nach ihren Wünschen. Legen Sie zusätzlich eine Swift-Klasse (z.B.: *MyCellTableViewCell*) vom Typ UITableViewCell an und verknüpfen diese mit der Prototype-Cell im Storyboard.





Wenn Sie die Prototype-Cell im Storyboard auswählen und zur Assistant-Editor View umstellen öffnet sich ihre UITableViewCell-Klasse. Hier können Sie wie bisher mit gedrückter rechter Maustaste die Outlets der Zelle mit dem Code verknüpfen.

```
class MyCellTableViewCell: UITableViewCell {

    @IBOutlet weak var mylabel: UILabel!

    ...
}
```

In Ihrem TableViewController kann nun folgendermaßen eine Zelle mit Daten bestückt werden:

```
override func tableView(tableView: UITableView,
cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {

let cell =
tableView.dequeueReusableCellWithIdentifier("MyCellTableViewCell",
forIndexPath: indexPath) as! MyCellTableViewCell

    cell.mylabel.text =  "My Text"

    return cell
}
```

Hinweis: In TableViews und ähnlichen Views werden nicht explizit neue Zellen erstellt. Stattdessen werden „alte" Zellen, die nichtmehr im sichtbaren Bereich sind, wiederverwendet um die CPU/Speicherauslastung möglichst gering zu halten. Sollten keine Zellen im Speicher vorhanden sein, werden durch `dequeueReusable-CellWithIdentifier` automatisch neue Zellen angelegt.

7.2.2 Testen

Testen Sie nun Ihre Anwendung und überprüfen Sie ob der Login-Name und die Namen in der View angezeigt werden.

8 Anhang: Einrichten der Entwicklungsumgebung auf eigenen Geräten

8.1 Benötigte IDEs und Programme:

- XCode
- Simulator
- Command Line Tools
- Eclipse / Android Studio für Java
- Java SDK 8
- Squirrel

8.3 Einrichten des Squirrel SQL-Clients

Möchten Sie von Ihrem eigenen Rechner aus auch auf die Datenbank mittels des Squirrel SQL-Clients zugreifen, so können Sie sich das Programm kostenlos herunterladen:

http://www.squirrelsql.org/

Nach der Installation muss zunächst der Datenbank-Treiber konfiguriert werden. Da wir eine MySQL-Datenbank verwenden, benötigen Sie hierzu zunächst die MySQL-JDBC-Treiber (*MySQL Connector/J*), welche zum Beispiel über einen Link auf der Praktikumsseite unter „Literatur" heruntergeladen werden können oder gehen Sie auf

http://dev.mysql.com/downloads/connector/j/

… und wählen Sie „Platform Independent", um die Downloads zu sehen. Nach dem Download entpacken Sie das Archiv.

8.3.1

Starten Sie den Squirrel-SQL-Client. Anschließend klicken Sie im Menü *Windows* auf den Eintrag *View Drivers*. Links öffnet sich eine Liste von Datenbanktreibern. Darin befindet sich ein Eintrag *MySQL Driver*. Mit einem Doppelklick darauf öffnet sich das Konfigurationsfenster. Im Tab *Extra Class Path* muss nun über *Add* die JAR-Datei hinzugefügt werden. Mit einem Klick auf *OK* ist die Treiberkonfiguration abgeschlossen.