Universität Heidelberg

Sommersemester 2014

Softwareentwicklung für iOS mit Objective-C und Xcode

Beispiellösungen der Übungsaufgaben

Nils Fischer

Aktualisiert am 1. Juni 2014
Begleitende Dokumente auf der Vorlesungsseite:
http://ios-dev-kurs.github.io

Inhaltsverzeichnis

1 Fibonacci

```
1  int a=1; // erstes Folgenglied
2  int b=2; // zweites Folgenglied
3  int di=0; // Abstand zur letzten geraden Zahl
4  while ((a+b)<1000) { // die Schleife wird ausgeführt, bis die nächste
          Fibonaccizahl zu groß wird
5     int c = a+b;
6     NSLog(@"%i",c);
7     a = b;
8     b = c;
9     di++;
10    if (c%2==0) { // Ausdruck ergibt YES für gerade Zahlen
11       NSLog(@"Zahl gerade, Abstand zur vorigen geraden Fibonaccizahl: %i", di)
             ;
12       di = 0;
13    }
14 }
```

2 Primzahlen

```
1  // prime numbers
2  for (int i=1; i<1000; i++) { // loop through all numbers from 1 to 1000
3     BOOL isPrime = YES;
4     for (int j=2; j<i-1; j++) { // check the division rest with every number
            from 2 up to (excluding) the current number and break when a divisor
            was found
5        if (i%j==0) {
```

```
6           isPrime = NO;
7           break;
8       }
9    }
10   if (isPrime) { // if isPrime is still YES here, no divisor was found
11      NSLog(@"Prime number found: %i", i);
12   }
13 }
```

3 Scientists

1. In Xcode könnt ihr eine neue Klasse mit ⌘+N erstellen. Nennt die Klasse `Scientist` und wählt die Superklasse `Person`. Die Main- und Header-Datei erscheinen nach dem Speichern im Project Navigator.

2. Im Interface in der Header-Datei wird die neue Methode zunächst definiert:

```
#import "Person.h"

@interface Scientist : Person

- (void)sayPrimeNumbersUpTo:(int)number;

@end
```

   Anschließend können wir sie in der Main-Datei implementieren:

```
#import "Scientist.h"

@implementation Scientist

- (void)sayPrimeNumbersUpTo:(int)number {
   // Algorithmus aus Aufgabe 'Primzahlen'
}

@end
```

   In der nun schon häufiger verwendeten `application:didFinishLaunchingWithOptions:` Methode in der *AppDelegate.m*-Datei, wollen wir die neue Klasse ausprobieren:

```
#import "AppDelegate.h"

#import "Scientist.h"

@implementation AppDelegate

- (BOOL)application:(UIApplication *)application
      didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {

   Scientist *aScientist = [[Scientist alloc] init];
   [aScientist sayPrimeNumbersUpTo:100];

   return YES;
}

```

```
15 @end
```

Führen wir die App aus, werden alle Primzahlen bis 100 in der Konsole ausgegeben.

3. Mit Objekten des Typs `Scientist` kann genauso verfahren werden wie mit Objekten des Typs `Person`, da sie voneinander abstammen und daher alle Attribute und Methoden erben. Schreibt also in der *AppDelegate.m*-Datei:

```
1  #import "AppDelegate.h"
2
3  #import "Scientist.h"
4
5  @implementation AppDelegate
6
7  - (BOOL)application:(UIApplication *)application
       didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
8
9     Scientist *aScientist = [[Scientist alloc] init];
10    aScientist.name = @"Alice";
11    [aScientist sayHello];
12
13    return YES;
14 }
15
16 @end
```

Es wird wieder ausgegeben:

```
1 Hello World! My name is Alice.
```

4. Die `Scientist`-Klasse besitzt bereits die Methode `sayHello`, da sie in ihrer Superklasse `Person` definiert und implementiert wird. Um sie zu überschreiben, müssen wir sie nur in der Main-Datei erneut implementieren:

```
1  #import "Scientist.h"
2
3  @implementation Scientist
4
5  - (void)sayPrimeNumbersUpTo:(int)number {
6     // Primzahlen-Algorithmus
7  }
8
9  - (void)sayHello {
10    NSLog(@"Hello World! My name is %@. I know all Prime Numbers.", self.
          name);
11 }
12
13 @end
```

Nun wird statt der Methodenimplementierung in der Superklasse 'Person' diese Neuimplementierung verwendet und der zusätzliche Text wird ausgegeben, wenn die Methode `sayHello` von einem Objekt der Klasse 'Scientist' aufgerufen wird:

```
1 Hello World! My name is Alice. I am a Scientist.
```

Da wir der Implementierung der Superklasse im Prinzip nur etwas hinzufügen wollen, ist es eleganter, stattdessen erst diese aufzurufen und dann den zusätzlichen Code auszuführen:

```
- (void)sayHello {
    [super sayHello]; // Die Implementierung Superklasse wird aufgerufen
    NSLog(@"I know all Prime Numbers.");
}
```

Ausgabe:

```
Hello World! My name is Alice.
I know all Prime Numbers.
```

4 Emails

Email.h

```
@interface Email : NSObject

@property (strong, nonatomic) NSMutableArray *allRecipients;

- (void)sendTo:(NSArray *)recipients;

@end
```

Email.m

```
#import "Email.h"
#import "Person.h"

@implementation Email

- (void)sendTo:(NSArray *)recipients {
    for (Person *person in recipients) {
        // treat email as spam and skip if it has already been sent to this
            person before
        if ([self.allRecipients containsObject:person]) continue;
        // add person to list of all recipients
        if (!self.allRecipients) self.allRecipients = [[NSMutableArray alloc]
            init];
        [self.allRecipients addObject:person];
        // deliver email
        [person receiveEmail:self];
    }
}

@end
```

Person.h

```objc
@class Email; // Forward Declaration

@interface Person : NSObject

@property (strong, nonatomic) NSString *name;

- (void)makeFriendsWith:(Person *)person;

- (void)sendEmail;
- (void)receiveEmail:(Email *)email;

@end
```

Person.m

```objc
#import "Person.h"
#import "Email.h"

@interface Person () // private interface

@property (strong, nonatomic) NSMutableArray *friends;

@end

@implementation Person

- (void)makeFriendsWith:(Person *)person {
    // make sure to skip connection if already existent or redundant
    if (!person||person==self||[self.friends containsObject:person]) return;
    // create friends array if not existent yet and add person
    if (!self.friends) self.friends = [[NSMutableArray alloc] init];
    [self.friends addObject:person];
    // trigger reverse connection
    [person makeFriendsWith:self];
    NSLog(@"%@ <-> %@", self.name, person.name);
}

- (void)sendEmail {
    Email *newEmail = [[Email alloc] init];
    [newEmail sendTo:self.friends];
}

- (void)receiveEmail:(Email *)email {
    NSLog(@"%@ received an Email.", self.name);
    [email sendTo:self.friends];
}

@end
```

AppDelegate.m

```objc
#import "AppDelegate.h"
#import "Person.h"

@implementation AppDelegate

```

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions
    :(NSDictionary *)launchOptions {

    Person *me = [[Person alloc] init];
    me.name = @"Nils";

    NSMutableArray *persons = [[NSMutableArray alloc] init];

    NSArray *names = @[@"Alice", @"Bob", @"Cindy", @"Bruce", @"Chris", @"Bill "
        , @"Susan"];
    for (NSString *name in names) {
        Person *newPerson = [[Person alloc] init];
        newPerson.name = name;

        [persons addObject:newPerson];
        [me makeFriendsWith:newPerson];
    }

    // befriend persons with same initial letter
    for (Person *person in persons) {
        for (Person *other in persons) {
            if ([person.name characterAtIndex:0]==[other.name characterAtIndex
                :0]) {
                [person makeFriendsWith:other];
            }
        }
    }

    // start simulation
    [me sendEmail];

    return YES;
}

@end
```

5 Scientists 2

Die Methode `sayHello` muss nicht mehr implementiert werden, denn eine Überschreibung der Superklassenimplementierung ist nicht mehr notwendig. Diese ruft nämlich nur die Methode `helloString` auf und gibt den Rückgabewert in der Konsole aus. Stattdessen müssen wir also `helloString` überschreiben, damit der Zusatz *I know all Prime Numbers.* ausgegeben wird:

```
- (NSString *)helloString {
    NSString *greeting = [super helloString];
    greeting = [greeting stringByAppendingString:@" I am a Scientist."];
    return greeting;
}
```

Oder in Kurzform:

```
- (NSString *)helloString {
```

```
2     return [[super helloString] stringByAppendingString:@" I am a Scientist."
            ];;
3 }
```

6 Simple UI

6.1 Counter App

1. Erstellen wir zunächst das User Interface. Wählt eine der Storyboard-Dateien im Project Navigator, entsprechend des Geräts, für das ihr das UI konfigurieren möchtet.
2. Zieht zwei Labels und drei Buttons aus der Object Library unten im Inspektor in die erste und einzige Scene und konfiguriert sie mit dem Inspektor *(s. S. 8, Abb. 1)*.

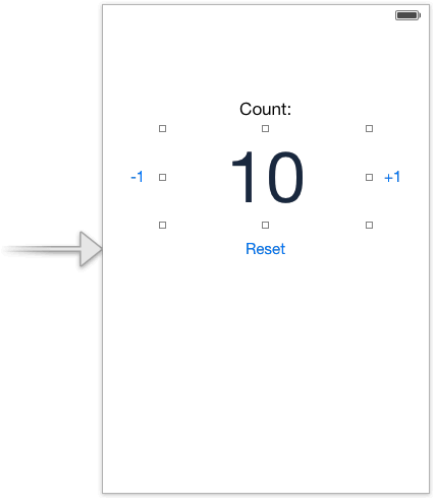

Abbildung 1: Einige Labels und Buttons sind für diese einfache App ausreichend

3. Wechselt in den Assistant-Editor, sodass die Datei *ViewController.m* rechts angezeigt wird. Wählt diese wenn nötig in der Jump bar des Assistant-Editors unter Automatic aus.
4. Definiert eine Property im privaten Interface, die zur Laufzeit der App die Referenz zum Label hält. Zusätzlich kann hier direkt die Property `int count` definiert werden, da diese ebenfalls nicht öffentlich sein muss. Definiert außerdem die benötigten Methoden:

```
1 @interface ViewController ()
2
3 @property (nonatomic) int count;
4
5 @property (strong, nonatomic) IBOutlet UILabel *countLabel;
6
7 - (void)updateLabel;
8
```

```
- (IBAction)increaseButtonPressed:(id)sender;
- (IBAction)decreaseButtonPressed:(id)sender;
- (IBAction)resetButtonPressed:(id)sender;

@end
```

5. Zieht mit gedrückter ctrl-Taste eine Verbindung zwischen Label und Property, sowie zwischen Buttons und Methoden.
6. Implementiert die Methoden in der Main-Datei:

```
- (void)updateLabel {
    self.countLabel.text = [NSString stringWithFormat:@"%i", self.count];
}

- (IBAction)increaseButtonPressed:(id)sender {
    self.count++;
    [self updateLabel];
}
- (IBAction)decreaseButtonPressed:(id)sender {
    self.count--;
    [self updateLabel];
}
- (void)resetButtonPressed:(id)sender {
    self.count = 0;
    [self updateLabel];
}
```

7. Build & Run !

6.2 BMI App

Analog zur Counter App erstellen wir ein neues Projekt und konfigurieren das User Interface im Storyboard *(s. S. 10, Abb. 2)*.

ViewController.m

```
#import "ViewController.h"

@interface ViewController ()

@property (strong, nonatomic) IBOutlet UITextField *gewichtTextfield;
@property (strong, nonatomic) IBOutlet UITextField *groesseTextfield;
@property (strong, nonatomic) IBOutlet UILabel *bmiLabel;

- (IBAction)berechneButtonPressed:(id)sender;

@end

@implementation ViewController

- (IBAction)berechneButtonPressed:(id)sender {
    float m = [self.gewichtTextfield.text floatValue];
    float l = [self.groesseTextfield.text floatValue];
    float bmi = m/l/l;
```

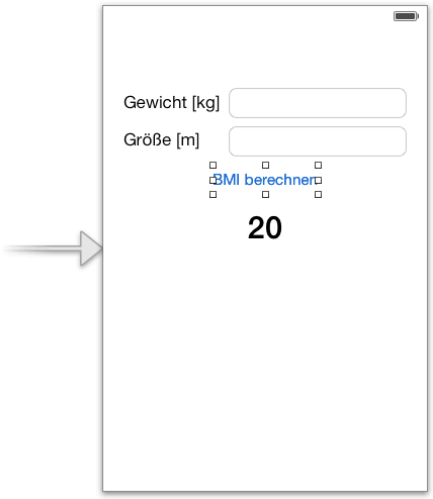

Abbildung 2: Achtet bei der UI Gestaltung darauf, dass die Interfaceelemente nicht von der erscheinenden Tastatur verdeckt werden

```
19      self.bmiLabel.text = [NSString stringWithFormat:@"%.1f", bmi];
20  }
21
22  @end
```

6.3 RGB App

Abbildung 3: Die RGB App zeigt Eingabefelder für die Farbkomponenten der Hintergrundfarbe

ViewController.m

```
1  #import "ViewController.h"
```

```
2
3  @interface ViewController ()
4
5  @property (strong, nonatomic) IBOutlet UITextField *rTextfield;
6  @property (strong, nonatomic) IBOutlet UITextField *gTextfield;
7  @property (strong, nonatomic) IBOutlet UITextField *bTextfield;
8
9  @property (strong, nonatomic) NSTimer *randomTimer;
10
11 - (IBAction)farbeAnzeigenButtonPressed:(id)sender;
12 - (IBAction)randomButtonPressed:(id)sender;
13 - (IBAction)switchValueChanged:(UISwitch *)sender;
14
15 @end
16
17 @implementation ViewController
18
19 - (IBAction)farbeAnzeigenButtonPressed:(id)sender {
20    float r = [self.rTextfield.text floatValue];
21    float g = [self.gTextfield.text floatValue];
22    float b = [self.bTextfield.text floatValue];
23    self.view.backgroundColor = [UIColor colorWithRed:r/255. green:g/255. blue:
            b/255. alpha:1.];
24 }
25
26 - (IBAction)randomButtonPressed:(id)sender {
27    self.rTextfield.text = [NSString stringWithFormat:@"%i", arc4random_uniform
            (256)];
28    self.gTextfield.text = [NSString stringWithFormat:@"%i", arc4random_uniform
            (256)];
29    self.bTextfield.text = [NSString stringWithFormat:@"%i", arc4random_uniform
            (256)];
30    [self farbeAnzeigenButtonPressed:sender];
31 }
32
33 - (IBAction)switchValueChanged:(UISwitch *)sender {
34    if (sender.isOn) {
35       self.randomTimer = [NSTimer scheduledTimerWithTimeInterval:0.1 target:
               self selector:@selector(randomButtonPressed:) userInfo:nil repeats
               :YES];
36    } else {
37       [self.randomTimer invalidate];
38       self.randomTimer = nil;
39    }
40 }
```

8 View Hierarchie

Als Beispiel ist hier die RGB App implementiert:

AppDelegate.m

```
1  #import "AppDelegate.h"
2
3  @interface AppDelegate ()
```

```
4
5  @property (strong, nonatomic) UITextField *rTextfield;
6  @property (strong, nonatomic) UITextField *gTextfield;
7  @property (strong, nonatomic) UITextField *bTextfield;
8
9  @property (strong, nonatomic) NSTimer *randomTimer;
10
11 - (void)farbeAnzeigenButtonPressed:(id)sender;
12 - (void)randomButtonPressed:(id)sender;
13 - (void)switchValueChanged:(UISwitch *)sender;
14
15 @end
16
17 @implementation AppDelegate
18
19 - (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions
       :(NSDictionary *)launchOptions {
20
21     // create and display the window
22
23     self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds
           ]];
24     self.window.backgroundColor = [UIColor whiteColor];
25     [self.window makeKeyAndVisible];
26
27     // create textfields and buttons and add to view hirachy
28
29     self.rTextfield = [[UITextField alloc] initWithFrame:CGRectMake(20, 50, 66,
            44)];
30     [self.window addSubview:self.rTextfield];
31     self.rTextfield.placeholder = @"Red";
32
33     self.gTextfield = [[UITextField alloc] initWithFrame:CGRectOffset(self.
           rTextfield.frame, self.rTextfield.frame.size.width+20, 0)];
34     [self.window addSubview:self.gTextfield];
35     self.gTextfield.placeholder = @"Green";
36
37     self.bTextfield = [[UITextField alloc] initWithFrame:CGRectOffset(self.
           gTextfield.frame, self.rTextfield.frame.size.width+20, 0)];
38     [self.window addSubview:self.bTextfield];
39     self.bTextfield.placeholder = @"Blue";
40
41     UIButton *farbeAnzeigenButton = [[UIButton alloc] initWithFrame:CGRectMake
           (20, self.rTextfield.frame.origin.y+self.rTextfield.frame.size.height
           +20, 280, 44)];
42     [self.window addSubview:farbeAnzeigenButton];
43     [farbeAnzeigenButton setTitle:@"Farbe anzeigen" forState:
           UIControlStateNormal];
44     [farbeAnzeigenButton setTitleColor:[UIColor blackColor] forState:
           UIControlStateNormal];
45     [farbeAnzeigenButton addTarget:self action:@selector(
           farbeAnzeigenButtonPressed:) forControlEvents:
           UIControlEventTouchUpInside]; // the equivalent to an IBAction
46
47     UIButton *randomButton = [[UIButton alloc] initWithFrame:CGRectOffset(
           farbeAnzeigenButton.frame, 0, farbeAnzeigenButton.frame.size.height
           +20)];
48     [self.window addSubview:randomButton];
```

```
49      [randomButton setTitle:@"zufällig" forState:UIControlStateNormal];
50      [randomButton setTitleColor:[UIColor blackColor] forState:
              UIControlStateNormal];
51      [randomButton addTarget:self action:@selector(randomButtonPressed:)
              forControlEvents:UIControlEventTouchUpInside];
52
53      UISwitch *rndSwitch = [[UISwitch alloc] initWithFrame:CGRectOffset(
              randomButton.frame, 0, randomButton.frame.size.height+20)];
54      [self.window addSubview:rndSwitch];
55      [rndSwitch addTarget:self action:@selector(switchValueChanged:)
              forControlEvents:UIControlEventValueChanged];
56
57      return YES;
58  }
59
60  - (void)farbeAnzeigenButtonPressed:(id)sender {
61      float r = [self.rTextfield.text floatValue];
62      float g = [self.gTextfield.text floatValue];
63      float b = [self.bTextfield.text floatValue];
64      self.window.backgroundColor = [UIColor colorWithRed:r/255. green:g/255.
              blue:b/255. alpha:1.];
65  }
66
67  - (void)randomButtonPressed:(id)sender {
68      self.rTextfield.text = [NSString stringWithFormat:@"%i", arc4random_uniform
              (256)];
69      self.gTextfield.text = [NSString stringWithFormat:@"%i", arc4random_uniform
              (256)];
70      self.bTextfield.text = [NSString stringWithFormat:@"%i", arc4random_uniform
              (256)];
71      [self farbeAnzeigenButtonPressed:sender];
72  }
73
74  - (void)switchValueChanged:(UISwitch *)sender {
75      if (sender.isOn) {
76          self.randomTimer = [NSTimer scheduledTimerWithTimeInterval:0.1 target:
                  self selector:@selector(randomButtonPressed:) userInfo:nil repeats
                  :YES];
77      } else {
78          [self.randomTimer invalidate];
79          self.randomTimer = nil;
80      }
81  }
```

9 Auto Layout

Die möglichen Constraints sind in Gleichungsform oder Visual Format Syntax angegeben. `contentView` bezeichnet die View an erster Stelle der Hierarchie.

a)
- `segmentedControl.centerX = contentView.centerX`
- `H:|-0-[progressView]-0-|`
- `H:|-0-[mainView]-0-|`

- `V:|-[segmentedControl]-[progressView]-[mainView]-0-|`

b) Die beiden Buttons `playButton` und `pauseButton` sind Subviews einer View `buttonCenterView` mit transparenter Hintergrundfarbe.

- `H:|-0-[playButton]-20-[pauseButton]-0-|`
- `playButton.centerY = pauseButton.centerY = buttonCenterView.centerY`

Die `buttonCenterView` besitzt damit eine Intrinsic Content Size, die von der Größe ihrer enthaltenen Buttons bestimmt wird.

- `buttonCenterView.centerX = contentView.centerX`
- `H:|-[volumeLabel]-[volumeSlider]-|`
- `H:|-0-[mainView]-0-|`
- `V:|-[buttonCenterView]-[volumeSlider]-[mainView]-0-|`
- `volumeLabel.centerY = volumeSlider.centerY`

c) Die Platzhalter Intrinsic Content Size von 300x300 und 100x100 kann im Attributes Inspector eingestellt werden. Zur Laufzeit wird diese verschwinden, da die Views keinen Inhalt besitzen. Es können stattdessen auch explizite `height` und `width` Constraints gesetzt werden, sodass die Views zur Laufzeit diese Größe erhalten. Im Allgemeinen sollte die Größe einer View jedoch von ihrer Intrinsic Content Size bestimmt werden.

- `largeView.centerX = contentView.centerX`
- `largeView.centerY = contentView.centerY @500`
- `smallView.centerX = contentView.centerX`
- `V:|-(>=20)-[largeView]-(>=20)-[smallView]-20-|`
- `largeView.verticalCompressionResistance = 501`

d) Wir verwenden vier zusätzliche Views als Platzhalter und setzen ihr Attribut `BOOL hidden` im Attributes Inspector auf `YES`. Die drei zu positionierenden Views erhalten eine Platzhalter Intrinsic Content Width von 50 oder eine explizite `width` Constraint.

- `V:|-20-[view0]-20-|`
- `V:|-20-[view1]-20-|`
- `V:|-20-[view2]-20-|`
- `placeholderView0.width = placeholderView1.width = placeholderView2.width = placeholderView3.width`
- `placeholderView0.centerY = placeholderView1.centerY = placeholderView2.centerY = placeholderView3.centerY = contentView.centerY //nur notwendig, damit das Layout eindeutig ist`
- `H:|-0-[placeholderView0]-0-[view0]-0-[placeholderView1]-0-[view1]-0-[placeholderView2]-0-[view2]-0-[placeholderView3]-0-|`