Beuth Hochschule für Technik Berlin
Fachbereich VI

Prof. Dr. Dragan Macos

Klausur zur Vorlesung iOS-Programmierung
Diese Probeklausur beinhaltet lediglich einige Beispielsfragen und keine exakten Klausurfragen.

Hinweise:

- Die Klausurdauer beträgt 90 Minuten.
- Zur Beantwortung der Fragen finden Sie genügend Platz in der Klausur.
  Bitte reißen Sie die Klausur nicht auseinander und verwenden Sie kein eigenes Papier.
- Die notwendige Bedingung für die Klausur ist die bestandene Laborübung.
- Die Klausur ist bestanden, wenn mindestens 50% der Punkte erreicht wurden.
- Durch den Antritt zur Prüfung erklären Sie sich für prüfungsfähig.
- Tragen Sie bitte zuerst Ihre persönlichen Daten ein.
- Wenn es sich bei dieser Klausur um Ihren letzten Versuch handelt, schreiben Sie bitte die entsprechende Anmerkung auf das Titelblatt.

Viel Erfolg!

Persönliche Daten:

Nachname	Vorname	Matrikelnr.	Semester	Datum

Bewertung:

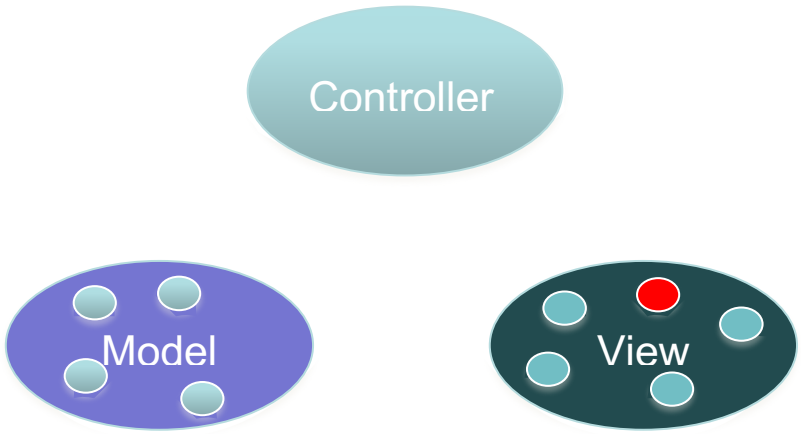
	Punkte		Summe
Klausur			
Insgesamt			
		Note:	

Beuth Hochschule für Technik Berlin
Fachbereich VI

Prof. Dr. Dragan Macos

Softwarearchitektur, iOS-Applikationen
Beispiel

1. Skizieren Sie auf dem folgenden Bild, welche Software-Komponenten in welcher Richtung miteinander kommunizieren dürfen.



/20

Fehlersuche, Beispiel

2. Finden Sie die Fehler im Swift-Code und erklären Sie diese.

```
14  class myClass {
15      var a = 20
16
17      init (aWert: Int) {
18          a = 10
19      }
20  }
21
22  let myInstance = myClass(aWert: 10)
23  let anotherMyInstance = myClass()
24  let myStr = "Highlander"
25  myStr += " and another Highlander"
26  let i = 1
27
28  if i {
29      print("Alles ok")
30  }
```

/20

Swift-Syntax/Semantik, Beispiele

3. Tupel-Dekomposition: Ergänzen Sie den unteren Swift-Code ohne Hinzufügen neuer Codezeilen, so dass die Programm-Ausgaben den in den Kommentaren angegebenen Ausgaben (gekennzeichnet mit "****Ausgabe") gleich sind.

```
let http404Error = (404, "Not Found")
let (                              ) = http404Error
println("The status code is \(statusCode)")
//***Ausgabe:"The status code is 404"
println("The status message is \(statusMessage)")
//***Ausgabe:"The status message is Not Found"
```

4.

a.) Schreiben Sie den Aufruf der Funktion

```
1   func greet(name: String, day: String) -> String {
2       return "Hello \(name), today is \(day)."
3   }
```

mit Parametern: „Bob" und „Tuesday" auf.

b.) Schreiben Sie den Aufruf der deklarierten Funktion mit den Parametern „Bill" und „Ted" auf.

```
func sayHello(to person: String, and anotherPerson: String) -> String {
    return "Hello \(person) and \(anotherPerson)!"
}
```

c.) Schreiben Sie den Aufruf der Funktion

```
1  func someFunction(parameterWithDefault: Int = 12) {
2      // function body goes here
3      // if no arguments are passed to the function call,
4      // value of parameterWithDefault is 12
5  }
```

mit dem Parameter 6 auf.

/20

5. Welche Ausgabe auf dem Bildschirm macht folgendes Programm

```
var pesonalienJulia = ("Julia", 18, "Moltkestr.", 45)
var pesonalienFranzi = ("Franni", 16, "Marschnerstr.",  120)
var personen = [pesonalienJulia, pesonalienFranzi]
pesonalienFranzi.2 = "Kurfürstendamm"
pesonalienFranzi.3 = 10
for personalie in personen
{
    print("Name: \(personalie.0), Alter: \(personalie.1)")
    print("Straße: \(personalie.2), Nummer: \(personalie.3)")
}
```

/10

6. Welche Ausgabe auf dem Bildschirm macht folgendes Programm

```
let yetAnotherPoint = (1, -1)
switch yetAnotherPoint {
case let (x, y) where x == y:
       print("(\(x), \(y)) is on the
        line x == y")
case let (x, y) where x == -y:
       print("(\(x), \(y)) is on the
        line x == -y")
case let (x, y):
       print("(\(x), \(y)) is just
        some arbitrary point")
}
```

/20

7. Funktionsaufruf

```
func swapTwoInts(a: inout Int, _ b: inout Int){
    let temporaryA = a
    a = b
    b = temporaryA
}

var someInt = 3
var anotherInt = 107
```

Wie sieht der Aufruf der definierten Funktion mit den o. a. Variablen als Parametern aus?

/20

---

8. Funktionsaufruf

```
func join(string s1: String, toString s2: String, withJoiner
        joiner: String)
    -> String {
        return s1 + joiner + s2
}
```

Wie sieht der Aufruf der Funktion mit Parametern: "Hallo", "Ihr Kinder", "," aus?

/20

---

9. Funktionsaufruf

```
func join(s1: String, s2: String, joiner: String) -> String {
    return s1 + joiner + s2
}
```

/20

---

10. Welche Typen haben die drei Funktionen:

```
func addTwoInts(a: Int, _ b: Int) -> Int {
    return a + b
}
func multiplyTwoInts(a: Int, _ b: Int) -> Int {
    return a * b
}
```

```
func printHelloWorld() {
    println("hello, world")
}
```

/20

11. Definieren Sie eine Konstante „onePlusTwoPlusTen“, die den Typ *ArithmeticExpression* hat und den arithmetischen Ausdruck “1+2+10“ darstellt. Sie können dabei beliebig viele Hilfsvariablen bzw. Konstanten (wie z. B. die Konstante *one*) definieren.

```
enum ArithmeticExpression {
    case number(Int)
    indirect case addition(ArithmeticExpression,
          ArithmeticExpression)
    indirect case multiplication(ArithmeticExpression,
          ArithmeticExpression)
}
```

```
let one = ArithmeticExpression.number(1)
```

/20

12. Erweitern Sie das Programm, so dass er folgende Ausgabe erzeugt:
-4...
-3...
-2...
-1...
zero!

```
func chooseStepFunction(backwards: Bool) -> (Int) -> Int {
    func stepForward(input: Int) -> Int { return input + 1 }
    func stepBackward(input: Int) -> Int { return input - 1
        }
    return backwards ? stepBackward : stepForward
}
var currentValue = -4
let moveNearerToZero = chooseStepFunction(currentValue > 0)
// moveNearerToZero now refers to the nested stepForward()
        function
while currentValue != 0 {
    println("\(currentValue)... ")
    currentValue = moveNearerToZero(currentValue)
}
println("zero!")
// -4...
// -3...
// -2...
// -1...
// zero!
```

/20

13. Schreiben Sie in die grauen Felder die Rückgabewerte der Funktionsaufrufe

```
func makeIncrementor(forIncrement amount: Int) -> ()
        -> Int {
    var runningTotal = 0
    func incrementor() -> Int {
        runningTotal += amount
        return runningTotal
    }
    return incrementor
}
```

```
let incrementByTen = makeIncrementor(forIncrement: 10)
incrementByTen()
```

Rückgabewert:

```
incrementByTen()
```

Rückgabewert:

```
incrementByTen()
```

Rückgabewert:

```
let alsoIncrementByTen = incrementByTen
alsoIncrementByTen()
```

Rückgabewert:

```
let incrementBySeven = makeIncrementor(forIncrement: 7)
incrementBySeven()
```

Rückgabewert:

```
incrementByTen()
```

Rückgabewert:

/20

14. Schreiben Sie in die grauen Felder die Bildschirm-Ausgaben des Swift-Programms.

```
let tenEighty = VideoMode()
tenEighty.resolution = hd
tenEighty.interlaced = true
tenEighty.name = "1080i"
tenEighty.frameRate = 25.0
let alsoTenEighty = tenEighty
alsoTenEighty.frameRate = 30.0
println("The frameRate property of tenEighty is now \
        (tenEighty.frameRate)")
```

```
class VideoMode {
    var resolution = Resolution()
    var interlaced = false
    var frameRate = 0.0
    var name: String?
}
```

Ausgabe:

/20

---

15. Programmieren: Erweitern Sie folgendes Programm, so dass er folgende Ausgabe erzeugt: „six times three is 18“

```
struct TimesTable {
    let multiplier: Int
    subscript(index: Int) -> Int {

    }
}
let threeTimesTable = TimesTable(multiplier: 3)

println("six times three is \
        (threeTimesTable[6])")
// prints "six times three is 18"
```

/20

---

16. Schreiben Sie in die grauen Kästchen die Bildschirmausgaben des unteren Swift-Codes. Resolution ist ein **Strukturtyp**!

```
let hd = Resolution(width: 1920, height: 1080)
var cinema = hd
cinema.width = 2048
println("cinema is now \(cinema.width) pixels wide")
```

Ausgabe:

```
println("hd is still \(hd.width) pixels wide")
```

Ausgabe:

/20

17. Swift-Sprachkonstrukte analysieren

a. Für die gekennzeichneten Sprachkonstrukte von Swift sollen folgende Angaben gemacht werden.

Zu 1:

Wie heißt eine auf diese Art und Weise definierte Variable innerhalb einer Struktur?
ANTWORT:_____

Zu 2:

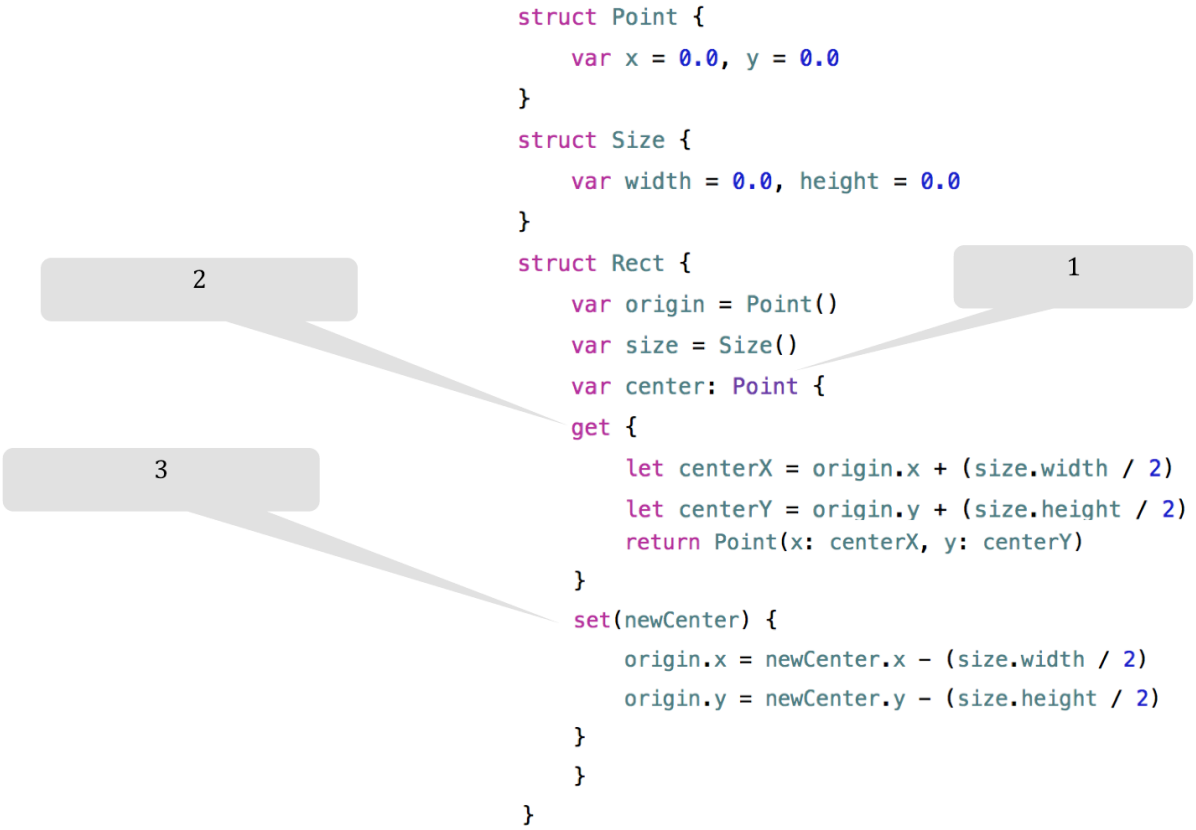
Wie heißt das Konstrukt?
ANTWORT:_____

Wann wird es aufgerufen?
ANTWORT:_____

Zu 3:

Wie heißt das Konstrukt?
ANTWORT:_____

Wann wird es aufgerufen?
ANTWORT:_____



b. Falls im u. a. Code einige der o. a. Konstrukte aufgerufen werden, markieren Sie diese mit den entsprechenden o. a. Nummern.

```
var square = Rect(origin: Point(x: 0.0, y: 0.0),
    size: Size(width: 10.0, height: 10.0))
let initialSquareCenter = square.center
square.center = Point(x: 15.0, y: 15.0)
```

18. Erklären Sie im u. a. Code die Zuweisung self.map.delegate = self.

```
class ViewController: UIViewController, MKMapViewDelegate{

    @IBOutlet weak var map: MKMapView!
    var locationManager = CLLocationManager()

    override func viewDidLoad() {
        super.viewDidLoad()
        locationManager.requestAlwaysAuthorization()
        self.map.delegate = self
        map.showsUserLocation = true
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

    func mapView(mapView: MKMapView!, didUpdateUserLocation userLocation: MKUserLocation!) {
        let curentLocation = userLocation.location.coordinate
        self.map.region  = MKCoordinateRegionMakeWithDistance(curentLocation, 1000, 1000)

    }
}
```

/20

19. Welche zwei Arten von Initialisierern für Swift-Klassen gibt es?

/20

---

20. Welche zwei Phasen der Initialisierung einer Swift-Klasse gibt es?

/20

---

21. Beantworten Sie folgende Fragen:

c. Wen muss ein designierter Initialisierer aufrufen?

d. Welche Initialisierer darf ein einfacher Initialisierer aufrufen?

e. Was soll am Ende einer Kette von einfachen Initialisierern stehen?

/20

---

22. Welche Initialisierer hat die Klasse *RecipeIngredient*?
Anmerkung: Es sollen die Namen aller Initialisierer aufgelistet werden.

```swift
class Food {
    var name: String
    init(name: String) {
        self.name = name
    }
    convenience init() {
        self.init(name: "[Unnamed]")
    }
}
```

```swift
class RecipeIngredient: Food {
    var quantity: Int
    init(name: String, quantity: Int) {
        self.quantity = quantity
        super.init(name: name)
    }
    override convenience init(name: String) {
        self.init(name: name, quantity: 1)
    }
}
```

/20

23. Welche Werte haben die Properties der Klasse ReciepeIngredient nach den Aufrufen von angegebenen Initialisierern?

```swift
class Food {
    var name: String
    init(name: String) {
        self.name = name
    }
    convenience init() {
        self.init(name: "[Unnamed]")
    }
}
```

```swift
class RecipeIngredient: Food {
    var quantity: Int
    init(name: String, quantity: Int) {
        self.quantity = quantity
        super.init(name: name)
    }
    override convenience init(name: String) {
        self.init(name: name, quantity: 1)
    }
}
```

```swift
let oneMysteryItem = RecipeIngredient()
let oneBacon = RecipeIngredient(name: "Bacon")
let sixEggs = RecipeIngredient(name: "Eggs", quantity: 6)
```

name: quantity:
name: quantity:
name: quantity:

/20

24. Beantworten Sie folgende Fragen zum unteren Beispielprogramm?

```
let myConstF = { (a: Int, b:Int)->Int in a + b }
let myConstI = { (a: Int, b:Int)->Int in a + b }(1, 2)
```

a.) Welchen Typ hat die Konstante myConstF? Antwort:

b.) Welchen Typ hat die Konstante myConstI? Antwort:

c.) Was gibt das untere Programm aus (Display/Konsole)?
Antwort:

```
let myConstF = { (a: Int, b:Int)->Int in a + b }
let myConstI = { (a: Int, b:Int)->Int in a + b }(1, 2)

print("myConstI= \(myConstI)")
print("myConstF(10,20)= \(myConstF(10,20))")
```

/20

25. Kombination verschiedener Typen in Swift. Beispiel: Dictionary-Struktur mit Funktionen als Values. Die Funktionen können wiederrum sowohl als vordefinierte Funktionen als auch als Closures im Dictionary gespeichert werden.