iOS Application Development

Lecture 4: Navigation and Workflow

Prof. Dr. Jan Borchers
Media Computing Group
RWTH Aachen University

WS '22/'23 • hci.rwth-aachen.de/ios
Recap

• Strings
• Functions
• Structs
  • Initializers, (mutating) functions
• Computed properties, property observers
• Structs vs. classes
• Collections (Array & Dictionary)
Optionals
Variables with nil

```swift
struct Book {
    var name: String
    var publicationYear: Int
}

let firstHarryPotter = Book(name: "Harry Potter and the Philosopher's Stone", publicationYear: 1997)
let secondHarryPotter = Book(name: "Harry Potter and the Chamber of Secrets", publicationYear: 1998)

let unannouncedBook = Book(name: "Rebels and Lions", publicationYear: 0)

• Zero isn't accurate, because that would mean the book is over 2,000 years old.

let unannouncedBook = Book(name: "Rebels and Lions", publicationYear: nil)
• Normal variable in Swift cannot be nil

```swift
var string = nil // error!!
```

• Optionals contain either an instance of the expected type or nothing at all (nil).

```swift
var string: String? = nil // this works
var string: String? = "something" // this works as well
```

```swift
struct Book {
    var name: String
    var publicationYear: Int?
}
```
Working with Optionals

• Optionals can be unwrapped using the **force-unwrap** operator `!`:

```swift
let unwrappedYear = publicationYear! // potential runtime error
```

• Before unwrapping an optional we need to make sure the value is not `nil`:

```swift
if publicationYear != nil {
    let actualYear = publicationYear!
    print(actualYear)
}
```

• Shorter version:

```swift
if let actualYear = publicationYear {
    print(actualYear)
} else {
}
```
Working with Optionals

• Can we use the same name for a constant using optional bindings?

```swift
let publicationYear : Int? = 2014
let actualYear : Int = 2022
if let actualYear = publicationYear {
    print(actualYear)
}
print(actualYear)
```
Working with Optionals

- It is possible to use the same variable names for optional bindings?

```swift
let publicationYear : Int? = 2014
let actualYear : Int = 2022
if let actualYear = publicationYear { // this works
    print(actualYear) // Output?
}
print(actualYear) // Output?
```
Working with Optionals

• It is possible to use the same variable names for optional bindings

```swift
let publicationYear : Int? = 2014
let actualYear : Int = 2022
if let actualYear = publicationYear { // this works
    print(actualYear) // Output: 2014
}
print(actualYear) // Output: 2022
```
Working with Optionals

• Unwrapping multiple optionals:

```swift
if let actualYear = publicationYear {
    if let bookEdition = edition {
        print(actualYear, bookEdition)
    }
}
```

• Optionals in functions:

```swift
func textFromURL(url: URL?) -> String?
{
    return nil
}
```

• Failable initializers:

```swift
init?()
{
    return nil
}
```
Optional Chaining

• Unwrapping nested optionals:

```swift
class Person {
    var age: Int
    var residence: Residence?
}
class Residence {
    var address: Address?
}
class Address {
    var buildingNumber: String
    var streetName: String
    var apartmentNumber: String?
}
```

```swift
if let theApartmentNumber = person.residence?.address?.apartmentNumber {
    print("He/she lives in apartment number \((theApartmentNumber)).")
}
```

• Shorter version:

```swift
if let theApartmentNumber = person.residence?.address?.apartmentNumber {
    print("He/she lives in apartment number \((theApartmentNumber)).")
}
```
Type Casting & Inspection
Type Casting

```swift
class Vehicle {}
class Car : Vehicle {}
class Motorcycle : Vehicle {}

func allVehicles() -> [Vehicle] {
    // returns all vehicles
}

let vehicles = allVehicles()
for vehicle in vehicles {
    if let car = vehicle as? Car {
        // ..
    } else if let motorcycle = vehicle as? Motorcycle {
        // ..
    }
}
```

- Force cast:

```swift
let cars = allVehiclesFrom
    (manufacturer: "Porsche") as! [Car]
```

- Use `as!` only when you are certain that the specific type is correct.

- If not your app will crash
The Any Type

• The Any type can represent an instance of any type: string, double, func, struct, class ...

```swift
var items: [Any] = [5,"Tom", 6.7, Car()]
if let firstItem = items[0] as? Int {
    print(firstItem+4) //9
}
```

• The AnyObject type can represent any class within Swift, but not a struct
Guard
func singHappyBirthday() {
    if birthdayIsToday {
        if invitedGuests > 0 {
            if cakeCandlesLit {
                print("Happy Birthday to you!")
            } else {
                print("The cake's candles haven't been lit.")
            }
        } else {
            print("It's just a family party.")
        }
    } else {
        print("No one has a birthday today.")
    }
}

guard condition else {
    //false: execute some code & return
    return
}
//true: execute some code

func singHappyBirthday() {
    guard birthdayIsToday else {
        print("No one has a birthday today.")
        return
    }
    guard invitedGuests > 0 else {
        print("It's just a family party.")
        return
    }
    guard cakeCandlesLit else {
        print("The cake's candles haven’t been lit.")
        return
    }
    print("Happy Birthday to you!")
}
Guard

• If statements only allow access to the constant within the brackets.

```swift
if let eggs = goose.eggs {
    print("The goose laid \(eggs.count) eggs.")
}
//`eggs` is not accessible here hereafter
```

• Guard statements allow access to the constant throughout the rest of the function

```swift
guard let eggs = goose.eggs else { return }
//`eggs` is accessible hereafter
print("The goose laid \(eggs.count) eggs.")
```

• Unwrapping multiple optionals:

```swift
func processBook(title: String?, price: Double?, pages: Int?) {
    guard let theTitle = title, let thePrice = price, let thePages = pages else {
        return
    }
    print("\((theTitle) costs $\((thePrice) and has \((thePages) pages.')")
}
```
Enumeration
Enumerations

• Define an enumeration:

```
enum CompassPoint {
    case north
    case east
    case south
    case west
}
```

• Using enumerations:

```
var compassHeading = CompassPoint.west
// The compiler assigns ‘compassHeading’ as a ‘CompassPoint’

var anotherHeading: CompassPoint = .west
anotherHeading = .north
```

```
enum CompassPoint {
    case north, east, south, west
}

let compassHeading: CompassPoint = .west

switch compassHeading {
    case .north:
        print("I am heading north")
    case .east:
        print("I am heading east")
    case .south:
        print("I am heading south")
    case .west:
        print("I am heading west")
}
```
Enumerations

• Type safety benefits:

```swift
struct Movie {
    var name: String
    var releaseYear: Int
    var genre: String
}

let movie = Movie(name: "Finding Dory",
                   releaseYear: 2016,
                   genre: "Animated")

let movie = Movie(name: "Finding Dory",
                   releaseYear: 2016,
                   genre: "Tom")

enum Genre {
    case animated, action, romance,
           documentary, biography, thriller
}

struct Movie {
    var name: String
    var releaseYear: Int
    var genre: Genre
}

let movie = Movie(name: "Finding Dory",
                   releaseYear: 2016,
                   genre: .animated)
```
Segue and Navigation Controller
Segues
Navigation Controller

- Settings
  - Airplane Mode
  - Wi-Fi
  - Bluetooth
  - Cellular
  - Personal Hotspot
  - Carrier
  - Notifications
  - Control Center
  - Do Not Disturb
  - General
  - Display & Brightness

- General
  - About
  - Software Update
  - AirDrop
  - Handoff
  - CarPlay
  - Accessibility
  - iPhone Storage
  - Background App Refresh
  - Restrictions

- Accessibility
  - VoiceOver
  - Zoom
  - Magnifier
  - Display Accommodations
  - Speech
  - Larger Text
  - Bold Text
  - Button Shapes
  - Increase Contrast
  - Reduce Motion
  - On/Off Labels
Tab Bar Controller
Summary

- Optionals, Guard, Enumerations
- Type casting and inspection
- Segues
- Navigation Controller

Tomorrow:
  - More UIViewController, Auto Layout, and Swift Protocols and Extensions