03.05.2019

Lisa Obermaier, Simon Thum

Mobile Application Development

Prof. Dr. Gudrun Socher

Introduction to Swift

1. Motivations for Swift

Swift is a rather new programming language and hat its first official release on September 9, 2014 with Swift 1.0. It combines features from Objective-C and features used by modern programming languages. It also bridges the gap between compiled languages, which are faster in execution, and interpreted languages, which are easier to learn. [[1]](#footnote-1)

1. Technical background
   1. Swift at a glance and beyond its basics

Swift is an imperative and functional programming language. It follows a block-structure, is object- and protocol-oriented. It contains static typing as well as type inference and provides dot-notations and UTF-8 encoding.

With a closer look, there come even more useful constructs with this programming language. Such as Generics, Extensions, Closures and Optionals. Swift uses ARC for Memory Management – Automatic Reference Counting. It also has proper Error Handling and Assertions, which are useful for Debugging. In the following, we will take a closer look into these constructs.

* 1. Generics

Whenever the code doesn’t need to be very specific and precise, it can be useful to write Generics instead of specified types.[[2]](#footnote-2)

*[Example from Slides]*

In our example we want to swap strings in the first function and doubles in the second function. It is cleary visible that we are rewriting the same code just to chance String into Doubles. Instead we can use the generic <T> to suggest every variable of type T has to be of the same type, but in general can be any possible type.

* 1. Extensions
  2. Closures

Closures are self-contained blocks of functionality and blocks can be subroutines, functions, procedures or methods. So a closure is a block, whose code refers to variables outside of the closure block. Most commonly closures are used, when you don’t know when you want to perform a certain block.[[3]](#footnote-3)

*[Example from Slides]*

* 1. Optionals

Optionals are used, when a value could not be assigned. An optional integer would be displayed as „Int?“ while the question mark indicated the opional type.[[4]](#footnote-4) This construct gives the possibility to avoid unintentional calculations or programming around possible missing values. Instead a missing value of our „Int?“-Type would be nil.[[5]](#footnote-5)

*[Example from Slides]*

In our example we convert our number to a possible int as you can see in the second line. The variable has the type of „Int?“ and is printed correctly, because we were able to convert the string *possibleNumber* into the int *convertedNumber*.

* 1. Memory Management: ARC
  2. Error Handling

Swift uses the keywods *try, throw(s)* and *catch* for error handling. A function, that could throw an error needs *throws* in the signature and every following call of this function needs a try-catch. If the try does not work, the error is properly handled as implemented. [[6]](#footnote-6)

*[Example from slides]*

In our example we are trying to call the function *makeASandwich()*, which fails, if there are not enough clean dishes and we call the error *washDishes()*, or if we are missing some ingredients, which calls the error *buyGroceries()* with the certain missing ingredient.

* 1. Debugging: Assertions

1. iOS App Development with Swift
2. Examples
3. Summary

1. 2018 Swift 4 for absolute beginners, page 83 [↑](#footnote-ref-1)
2. 2016 Book Practical Swift, page 101f. [↑](#footnote-ref-2)
3. 2018 Book Learn Computer Science With Swift, page 216 [↑](#footnote-ref-3)
4. 2018 Book Swift 4 for absolute Beginners, page 33 [↑](#footnote-ref-4)
5. 2018 Learn Computer Science, page 201ff. [↑](#footnote-ref-5)
6. 2016 Practical Swift, page 7f. [↑](#footnote-ref-6)