Classes Structures

Stefan Kempinger

Best Practices

Best Practices

Whenever you define a new structure or class, you define a new Swift type.
Give types `UpperCamelCase` names (such as `SomeStructure` and `SomeClass` here) to match the capitalization of standard Swift types (such as `String`, `Int`, and `Bool`).
Give properties and methods `lowerCamelCase` names (such as `frameRate` and `incrementCount`) to differentiate them from type names.

Structures

Structures

```
struct GradeStruct {
   var grade: Int
   var subject: String?

   init(grade: Int) {
      self.grade = grade
   }
}

var aStruct = GradeStruct (grade: 3)
var bStruct = aStruct
bStruct.grade = 2

print(aStruct.grade)      // 3
print(bStruct.grade)      // 2
```

Werte müssen initialisiert oder Optional sein

Eine Zuweisung ist eine Kopie der Werte

Classes

Classes

```
class GradeClass {
   var grade: Int
   var subject: String?

   init(grade: Int) {
      self.grade = grade
   }
}

var aClass = GradeClass (grade: 3)
var bClass = aClass
bClass.grade = 2

print(aClass.grade)      // 2
print(bClass.grade)      // 2
```

Werte müssen initialisiert oder Optional sein

Eine Zuweisung ist eine Kopie der Referenz

Classes

```
class GradeClass {
   var grade: Int
   var subject: String?

   init(grade: Int) {
      self.grade = grade
   }
}
```

```
var aClass = GradeClass (grade: 3)
var bClass = aClass
bClass.grade = 2

print(aClass.grade)      // 2
print(bClass.grade)      // 2

if aClass === bClass
        print(“De san gleich !”)
```

Classes

```
class GC {
   var grade: Int
   var subject: String?

   init(grade: Int) {
      self.grade = grade
   }

   static func == (_ links:GC, _ rechts:GC) -> Bool {
      return links.grade == rechts.grade
   }
}
```

```
var aClass = GC (grade: 3)
var bClass = GC (grade: 3)

print(aClass.grade)      // 3
print(bClass.grade)      // 3

if aClass == bClass {
         print(“De san gleich !”)
}

If aClass !== bClass{
         print("Des is oba a aundane Referenz!")
}
```

Unterschiede

- Durch Vererbung kann eine Klasse Eigenschaften einer anderen vererbt bekommen.
  - Deshalb kann man auch den Typen von Klassen zur Laufzeit casten.
- Klassen können Destruktoren besitzen.
- Mit Reference counting werden “unbenötigte” Referenzen entfernt (problematisch!)

Beispiel

Die Classes “Schüler” und “Schulklasse” sind zu erstellen.

Ein Schüler hat einen Namen und eine Referenz auf seine Klasse.
Eine Klasse hat auch einen Namen, die Zahl der Schüler und die Schülernamen.

Wenn ein neuer Schüler angelegt wird, soll er automatisch zu seiner Schulklasse hinzugefügt werden.

Zusatz: Wenn ein Schüler ersetzt/gelöscht wird soll er auch aus seiner Klasse gelöscht werden.

Classes Structures

Stefan Kempinger

Links

https://www.tutorialspoint.com/compile_swift_online.php

Swift Classes: http://tpcg.io/4byDyO

Struct vs. Classes: http://tpcg.io/rb6RDL

https://www.weheartswift.com/swift-classes-part-1/
https://docs.swift.org/swift-book/LanguageGuide/ClassesAndStructures.html
https://docs.swift.org/swift-book/LanguageGuide/Initialization.html
https://docs.swift.org/swift-book/LanguageGuide/Deinitialization.html
https://docs.swift.org/swift-book/LanguageGuide/AutomaticReferenceCounting.html
https://www.tutorialspoint.com/swift/swift_classes.htm

Classes

Because classes are reference types, it is possible for multiple constants and variables to refer to the same single instance of a class behind the scenes. (The same is not true for structures and enumerations, because they are value types and are always copied when they are assigned to a constant or variable, or passed to a function.)

It can sometimes be useful to find out if two constants or variables refer to exactly the same instance of a class. To enable this, Swift provides two identity operators:

- Identical to (===)
- Not identical to (!==)

Warum ARC scheiße ist

Clearly the reference count must be updated using an atomic operation; otherwise it could be damaged if accessed by multiple threads on a multiple core system.

This generates lots of bus traffic between cores, hence can be very slow.

Garbage collected environment can therefore be a lot faster for multi threaded software running on many cores.