Swift vs. OZ

Jesus Covarrubias | Tyler Gillette

Swift Introduction

Swift is a phenomenal way to compose any software. It's an intelligent and interactive programming language that consolidates the best in present day language thinking with astuteness from the more extensive Apple engineering culture and the commitments from its open-source network. The compiler is streamlined for execution and the language is advanced for improvement, without bargaining on either.

New programmers and software engineers will find Swift to be friendly to use. It's a industrial quality programming language that is as expressive and pleasant as a scripting language. Composing Swift code in a playground gives you a chance to try different things with code and see the outcomes promptly, without the overhead of building and running an application.

Where many programming languages leave off is where Swift picks up. It allows for memory to be managed automatically, error handling allows controlled recovery from unexpected failures, optionals ensure that nil values are handled explicitly, integers are checked for overflow, and variables are always initialized before use.

Swift has been years in the making, and it continues to evolve with new features and capabilities.

Object-oriented programming in Swift

Object oriented programming languages use classes. A class is defined as an entity that specifies an object in an incremental way, by defining the classes that the object inherits from (its direct ancestors) and defining how the class is different from the direct ancestors. Most modern languages support classes as a linguistic abstraction. To make the concepts precise we will add a simple yet powerful class construct.

In Oz, classes are defined as an incremental definition of an Abstract Data Type (ADT) as referenced in the book, *"Concepts, Techniques, and Models of Computer Programming"* by Peter Van Roy and Seif Haridi, that defines the ADT as a modification of other ADTs. There is a rich set of concepts for defining classes. We classify these concepts into two sets, according as they permit the class to define an ADT completely or incrementally.

Oz is able to have its ADT's be defined as a Complete ADT or an Incremental ADT. The complete ADT lets Oz use the advantages of dynamic typing languages which allows for powerful forms of polymorphism that are very hard to implement in statically typed languages. With the Incremental ADT it let's connect classes to other classes, similar to inheritance.

An example of how Oz implements classes is as follows:

```
class DogCatcherNet
  attr customerReviewStars:
       weightInPounds:
  meth init(stars weight)
    customerReviewStars := stars
    weightInPounds := weight
  end
end
```

Unlike other programming languages, Swift doesn't require you to create separate interface and implementation files for custom structures and classes. In Swift, you define a structure or class in a single file, and the external interface to that class or structure is automatically made available for other code to use.

An example of how Swift implements static classes is as follows:

```
class DogCatcherNet {
  let customerReviewStars: CustomerReviewStars
```

```swift
  let weightInPounds: Double
  // ☆ Add Optional called dog of type Dog here

  init(stars: CustomerReviewStars, weight: Double) {
    customerReviewStars = stars
    weightInPounds = weight
  }
}
```

Here is an example of how to use a class declared in Swift which is static by default.

```swift
let net = DogCatcherNet(stars: .two, weight: 2.6)
debugPrint("Printing a net: \(net)")
debugPrint("Printing a date: \(Date())")
print()
```

Although Swift is originally statically typed it can also be used to create dynamic classes just like Oz does in there definition of complete ADT's in *"Concepts, Techniques, and Models of Computer Programming"* by Peter Van Roy and Seif Haridi, Chapter 7.

Here is an example of how Swift can become dynamic when creating classes similar to how Oz uses inheritance and forwarding with the class defined as complete Abstract Data Types.

```swift
extension DogCatcherNet: CustomDebugStringConvertible {
  var debugDescription: String {
    return "DogCatcherNet(Review Stars: \(customerReviewStars),"
      + " Weight: \(weightInPounds))"
  }
}
```

Oz can be utilized to program classes that are static and dynamic and although Swift is not as flexible in creating classes in Object Oriented Programming, if needed, Swift has the ability to create a dynamically typed class that is available for other class declarations to use.

Functional programming in Swift

Swift supports some the functional paradigm out of the box. The goal of functional paradigm is to evaluate functions from a mathematical standpoint and to keep the states the same. It manages this for example by using keyword let instead of var. This allows the variable to be assigned once rather than changing the state/value of it. A nice perk of supporting functional programming is that it's less likely to have bugs and easier to read. One thing to note is functions in swift are first-class and in OZ they are not. There are some aspects of OZ that are first class, such as the messages in objects and in attributes.

Swift can use very little syntax to create a variable and a function that looks through a list of numbers and checks if they are even and then return the list of only even numbers. We can do the same thing in OZ, but when done in OZ, we are using much more syntax and also creating loops and case statements. This gives Swift the advantage by allowing the code to do the same thing, but being shorter and much easier to read.

Here is an example of how to use a let in Swift which lets you define a static variable.

```
let even = [1, 2, 3, 4, 5].filter { $0 % 2 == 0 }
```

Here is an example of how to define a variable in OZ, although a variable can be updated.

```
declare
fun {Evens X}
   case X of X|Xr then
      if (X mod 2) == 0 then X|{Evens Xr}
      else {Evens Xr}
      end
   else nil
   end
end
Even = {Evens [1 2 3 4 5]}
```

In conclusion, Swift is a great language that is very versatile and easy to use. Swift is an intelligent and interactive programming language that combines many useful features of both object-oriented and functional programing languages. These features are comparable to OZ's and in most cases are better defined in swift with much stronger supporting documentation. One good example that was given is the comparison of shorthand in Swift vs. OZ. Swift is a popular language, and it's easy to see why. Swift is an open source language that is always being improved and updated, unlike OZ, which gives Swift an upper hand.