Comparison Object Oriented Programming in Objective-C with GO

Thomas Martin Randl

Fakultät für Informatik

WS 2020/21

Abstract

This thesis clarifies the question of what distinguishes object-oriented languages and how **Object Oriented Programming** works in general. Afterwards two of the representatives of these programming languages **Objective-C** and **Go** are checked for the OOP criteria and compared with one another. Finally, a recommendation for action is made as to which of the languages should be used and when.

Keywords: Go, Objective-C, Object Oriented Programming

Table of Contents

List of Abbreviations

Go	Programming Language Go
OOP	Object Oriented Programming
OS	Operating System

1. Introduction

Object Oriented Programming (OOP) is a very popular and a widely used approach to develop all kinds of applications nowadays. To do so, there are a lot of programming languages from which the developer can choose.

The following chapters provide an overview of OOP in terms of its historical development and functionality. Afterward the two programming languages Objective-C and Go are compared based on their OOP capabilities.

2. History of OOP

The first approach to Object Oriented Programming was more than 40 years ago with the programming language Simula 67. However the first language to be really successful with this new approach in programming was Smalltalk. These languages fulfilled the basics needed for OOP. With a slow and steady development, OOP became a more and more used concept in new developed programming languages [KOP09].

Since the teaching of programming languages like Java took place in a wide range of university classes, OOP became more and more relevant to companies. This lies in the fact that graduates from all over the world tend to know how to use at least one OOP language. The fact that more and more companies made use of OOP languages in their software solutions pushed their popularity even further.

Nowadays OOP is the most spread concept between all programming languages. Besides their teaching all over the globe their popularity can be derived from the fact that problems from the real world can easily be transferred into the programming world. It is simple to imagine a button in a graphical interface to be an object with certain properties. Even non OOP languages nowadays tend to use object similar approaches as you can see with the programming language Go in Chapter 5. Object Oriented Programming in Go [KOP09]. Sometimes this works out well, but the OOP approach turns out to be the better way in most of these cases.

3. Object Oriented Programming

As we learned in chapter 2. History of OOP, OOP is a widely spread concept in modern programming languages. It takes proceedings from the real world into a programming context. This is because of the fact that you can declare every instance from the real world as an object. These objects on the other hand can be put in context and interact with each other. These traits of OOP are called classification and inheritance. With that in mind every object needs his own class with its own methods to interact with each other [KOP09]. In the following chapter we will learn how a class in OOP is structured and what base concepts OOP uses to set classes in context with each other

and how they are visible outside of the object. The extend to which a programming language uses these concepts defines how much of an OOP language it is.

3.1 Classification

Classification is a technique used to classify objects due to their skills. During the process objects are structured (chapter 3.3 Inheritance) and the skills are assigned to the objects class.

A class describes and implements a new type of object. Objects in OOP are derived from that class. Every class consists of several attributes and methods. An object can be created using the class as kind of a data type with parameters. The added parameters are assigned to the attributes in the constructor of the class. The created object interacts independently from other objects in the environment using the methods provided by the base class.

The methods of a class are declared in an interface. This can be used by other classes to interact with the class objects using these methods [KOP09].

3.2 Encapsulation

Encapsulation is used for hiding the object information from the environment. It is used to protect the attributes values of the object of manipulation. This is done by declaring the attribute visibility. If an attribute is declared as not visible from outside the object, it can only be accessed by using methods declared in the interface of the object class. Methods can also be declared in different visibility states depending on the programming language.

An encapsulated object can only be interacted with via the methods declared in the interface. Every attribute or method not mentioned in the interface is neither accessible nor visible from outside [KOP09].

3.3 Inheritance

As in chapter 3.1 Classification mentioned, Objects in OOP are structured by their skills and abilities. In OOP every Object is derived from a parent object. The resulting child object is able to use the functionality of the parent and its own. This property is called inheritance.

With the use of inheritance, it is possible to reuse the previous defined structures or to make use of an abstract object which can be specified in the child object. This helps the programmer to reduce the complexity and effort of the program.

Especially the use of an abstract class can be very useful. This is the case when a program makes use of different objects that contain several equal attributes. In this case the abstract class defines the base attributes and the child objects inherit them and implement their functionality for them [KOP09].

3.4 Polymorphism

A synonym for polymorphism is diversity. Taking this into the context of programming languages, polymorphism is the approach to accept and return values of more than one data type.

OOP uses this functionality with its inheritance (chapter 3.3 Inheritance). This allows the use of different objects as parameters and/or return values.

3.5 Persistence

Persistence stands for the lifetime an object exists in the program after it is created. There are some different approaches depending on the used programming language. While in C++ the user is responsible for deleting the created objects after their use expired, languages like Java use a so called "garbage collector".

Having this in mind, languages with a garbage collector are for no use in safety related software, because a fast reaction to a problem cannot be guaranteed if the garbage collector interrupts the program at the exact moment of an emergency. An automatic memory management on the other hand is much less bug prone due to the lower complexity [KOP09].

4. Object Oriented Programming in Objective-C

The programming language Objecitve-C was developed to extend the functionalities of the popular C programming language with OOP. This took place in the early 80s after the rising popularity of the UNIX Operating System (OS), which was almost completely developed in C. The developer of Objective-C Brad J. Cox wanted to create a whole new programming language by using C and combining it with the Smalltalk-80 language. After the Apple company acquired the rights to Objective-C in 1996 it introduced possibilities for development on their Mac OS X.

In 2007 the Apple company introduced the first IPhone. In the same year Apple reworked the Objective-C programming language and released Objective-C 2.0. The popularity of this programming language come from the fact that applications for the IPhone are developed with Objective-C [OC209].

Today Objective-C is the primary programming language for OS X and iOS applications. Due to the combination of the C syntax, types and flow control statements with the possibility to define classes and methods, it suits optimal to develop these applications because of the fact that they use OOP mostly all of the time.

To develop these applications apple provides his own integrated development environment with the name Xcode. It is also necessary to use a Mac computer to use this application, because it is only available in the Mac App Store [POC20]. As for the fact that Objective-C is a pure OOP language, it provides all the features discussed in chapter 3. Object Oriented Programming. The following chapters explain how they work and what may be special about them.

4.1 Classification with Objective-C

The usage of classes in Objective-C is a very common task. This is because of the fact, that it is used to develop applications and every part of an application either is an object or communicates with one. Apple provides a large library of predefined classes with the names Cocoa (Mac OS X) and Cocoa Touch (IOs). These classes can either be used directly or personalised for the application purpose.

Since everything in Objective-C can be achieved with classes they are structured very similar to Java or C++ classes. As can be read in chapter 4.3 Inheritance with Objective-C all classes are inherited from the root class NSObject. Every Objective-C class has an @interface and @implementation section. While the @interface usually is located in the header file .h the @implementation is implemented in the source file with the file ending .m.

The @interface section of the class defines its name, parent class (e.g. NSObject), attributes and methods. It is used to define the contents of the class and for other class instances to see what methods can be called on the objects of the class. The following example shows how this can be implemented.

Example implementation of an interface in Objective-C

```
# The head of interface MyClass with parent Class NSObject
@interface MyClass: NSObject
{
    # The attributes of MyClass
    int attribute1;
    char attribute2;
}
# Methods of MyClass
- (void) setAttributes: (int) i: (char) c;
@end
```

The implementation of the defined methods can be done as follows.

Example implementation of the defined methods in Objective-C

```
@implementation MyClass
# Implementation of the setAttributes Method defined in @interface
- (void) setAttributes: (int) i: (char) c
{
    attribute1 = i;
    attribute2 = c;
}
@end
```

To create an instance of a class in Objective-C you have to allocate memory and initialize it first. Afterwards you can interact with the object. If your class does not provide an own init function the init function calls the NSObjects init function as default. The alloc function is implemented in NSObject as well [OC209].

Example implementation of the initialisation and usage of a object in Objective-C

```
# Allocate memory and initialize the object myInstance as an instance from MyClass
# If no attributes are needed you can also write [MyClass new]
MyClass *myInstance = [[MyClass alloc] init];
# Call the method setAttributes on the created object
[myInstance setAttributes: 3: 'c'];
```

As for the fact that an Objective-C compiler is able to compile C and C++ code it is possible to combine C, C++ and Objective-C Code in one file and compile it.

4.2 Encapsualtion with Objective-C

In Objective-C one can implement attributes in the following three states of protection:

- **private** (Accessible only from within the class)
- **protected** (Accessible only from within the class and its subclasses)
- **public** (Accessible in the whole program)

Therefore Objective-C supports encapsulation. The way it is achieved however is more laborious than in similar programming languages like Java.

Attributes which are declared in the interface are visible per default. To change the accessibility one has to define getter and setter methods. This can be done by declaring properties. Objective-C defines the getters and setters automatically when there is a @property tag in front of the attribute. This property tag can be supplied with attributes (e.g. readonly) as well. The following code snippet shows how the protected and public accesibilities can be achieved. To declare a private attribute it has to be implemented only in the @implementation section outside of the methods [POC20].

Encpsulation private, protected and public in Objective-C

```
1 @interface MyClass: NSObject
2
3 # Protected is achieved with the nonatomic property attribute
4 @property (nonatomic) int attribute;
5
6 # Public is achieved by declaring public accessible getters and setters
7 - (void) setAttribute;
8 @end
```

The problem with the nonatomic attribute of the @property tag is the fact, that it cannot be guaranteed that it is accessible at this exact moment due to other threads accessing it [POC20].

To call a method of the class inside of another method of the same class you can use the keyword self. This refers to the class and the compiler is able to asign the correct method. An example call would be [self mySecondMethod] [OC209].

4.3 Inheritance with Objective-C

Objective-C supports multilevel inheritance. Therefore a class can inherit all methods and attributes of a parent class. On top of all classes in Objective-C is the NSObject class as the root object. All classes below inherit its methods. It is also possible to overwrite functions from a parent class. This has the effect, that the new implementation of this function will be executed when called on an object from this class. The instance attributes defined in the parent class can also be accessed in the child class [OC209].

Example inheritance in Objective-C

```
1 @interface MyClass: MyParentClass
```

Objective-C does not provide a mechanism for multiple inheritance. Therefore a class can only inherit one parent class.

4.4 Polymorphism with Objective-C

Objective-C supports polymorphism. The compiler is able to determine which method is called from which instance. This is due to the fact that classes encapsulate their attributes and methods and the compiler therefore knows the related object for the method even if the method name is used by other classes as well [OC209].

4.5 Persistence with Objective-C

As it is common in OOP languages, Objective-C has a garbage collector. Its purpose is to delete unused objects like it is stated in chapter 3.5 Persistence. Regardless of this fact, it is proper style to delete instances of classes after they are not needed anymore. This has two main reasons.

First of all the Objective-C garbage collector is not available on all apple platforms like e.g. the IPhone. Additional Objective-C is a derivative of C. As it is common knowledge C and C++ are languages where it is essential to maintain the device storage by yourself. If it is not done properly it results in unnecessary high storage workload and can even lead to a crash of the OS. Therefore the maintenance of the programs instances should be taken care of very carefully in Objective-C.

To release the memory used by an instance, you just need to add a command like seen in the following code snippet after the instance is not needed anymore.

Example release of an instance in Objective-C

```
1 # This releases the instance and frees the memory
2 [ myInstance release ]
```

The release function like alloc and init does not need to be implemented by the class. This function is also inherited of NSObject [OC209].

5. Object Oriented Programming in Go

The programming language Go was introduced by Google in 2009. It has been developed since by a team at google and a lot of other contributors from the open source community. The BSD style license it is released with allowed the community the further development to this day.

Its initial cause was to create a language that is more accessible and save than C/C++ in terms of syntax, compile time and functionality. The focus was to develop an easier solution for scalable network services and cloud computing. Whilst Go differs in many ways from the C programming language, its roots with this language are preserved in the fact, that it still uses C like pointers. But they do not support pointer arithmetic which is because of the fact, that Go puts his focus on fast compiling [GOL20].

Go does a balancing act between velocity and accessibility. Its purpose is to deliver a solution which is faster than competing languages like python [WSP20] and more accessible than the really fast languages C and C++. Sadly Go does not fit well with GUI development or the development of embedded systems [COP20].

In terms of OOP even Google is not sure whether or not Go is an OOP language. The total absence of some features, which are discussed later in this chapter, could lead to the conclusion that Go simply is no OOP language. But at least it can be argued, that Go allows an OOP like style of programming [IGO18]. The degree to which Go differs from classic programming languages like C++ or Java will

be discussed in the following chapters.

5.1 Classification with Go

Go does not provide a classic syntax for creating a class. Go does not even provide classes in general. To achieve a classification Go uses structs similar to structs in the C programming language. The following code snippet shows how a class is implemented in Go.

Example for a class in Go

```
# This is the type containing the attributes of the class ClassA
type ClassA struct {
    color color.Color
    name  string
}

# This is a method of class ClassA
func (c ClassA) SayName() {
    fmt.Println(name);
}
```

This states out that a struct is a user defined type that can hold a list of attributes. In combination with functions using the struct as a base, as shown in the snippet, Go is able to offer similar functionality than other languages by using classes [COP20].

5.2 Encapsulation with Go

Classic OOP languages use keywords like "protected", "private" and "public" to encapsulate attributes and methods of their classes. Go does offer a different approach.

Go encapsulates on package level by differentiating between lower or upper case on the first letter of the method, type or interface name.

Encapsulating possibilities in Go

```
1 # This is a public struct due to the capital letter
2 type ClassA struct {}
3
4 #This is a private struct due to the lower case letter
5 type classB struct { x, y float64 }
6
7 #This is a private method due ot the lower case letter
8 func (c *ClassA) string() string {
9     return fmt.Println("I can only be called inside my package")
10 }
11
12 # This is a public method due to the capital letter
13 func (c *ClassA) Draw() {
14     fmt.Println("I can be called from outside my package")
15 }
```

As is seen in the above code snippet one can make some methods of a struct private and others public. Even structs can be declared private. This allows the base concept of encapsulation even if it is not as convenient like in other OOP languages as mentioned before. Calling the encapsulating functions public or private is not completely correct, since it is more of a control mechanism to allow the export of a function, type or interface [IGO18].

If you declare a method with a parameter in front of the method name you can conveniently call it like for example "myStruct.myFunction()". The Go compiler compiles it like a normal parameter after the function name anyway. So this is just syntactic sugar which gives you the illusion to write and call a real OOP method [IGO18].

5.3 Inheritance with Go

The Go programming language does not implement inheritance in any way. This being said, Go has is own way to map relationships between types. Therefore, it uses composition instead of inheritance.

Languages like Java and C++ require the programmer to know the relationships of his objects at first implementation. Additonaly, multiple inheritance can get very complex very fast. Therefore the developers of Go made the choice to implement a lightweight alternative, which allows the implementation of relationships between types [FAQ20].

Inheritance alternative approach in Go

```
# Class A
type ClassA struct { x, y float64 }

# Class B extends ClassA
type ClassB struct {
    c     ClassA  # ClassB embedding a classA instance extends ClassA with the
    num   int     # attributes of ClassB
}
```

That being said, we do not speak so much of ClassA inherits from ClassB, as ClassA extends ClassB by embedding a ClassB instance inside the struct of ClassA. Go also supports overriding functions. Overloading a function like in Java, however is not possible with Go [COP20].

5.4 Polymorphism with Go

Since Go does not support inheritance, but follows the principle of composition there is a different approach to polymorphism than usual. To achieve the treatment of different objects uniformly it implements interfaces implicitly [IGO18]. This leads to a very intuitive approach as can be seen in the following code snippet.

Inheritance alternative approach in Go

```
# Interface for the ploymorphism methods
type getAttributes interface {
    getArea() float64
}

type square struct {
    length float64
}

type rectangle struct {
    width  float64
    height float64
}
# The implementation of getArea() for square
func (s square) getArea() float64 {
    return s.length * 2
}
# The implementation of getArea() for rectangle
func (r rectangle) getArea() float64 {
    return r.width * r.height
}
# The function to call to get the output for all objects with a getArea() method
func calcArea(a getAttributes) {
    fmt.Println("The area of the attribute is:", a.getArea())
}

func main() {
    s := square{length: 4}
    r := rectangle{width: 2, height: 3}
    calcArea(s) # The area of the attribute is: 8
    calcArea(r) # The area of the attribute is: 6
}
```

In this example code there are two structs that implement the same function getArea(). Both have different approaches since rectangle and square areas are calculated differently. The interface getAttributes is basically a named collection for all method signatures defined by the interface. This interface allows the call of methods contained by it and therefore we can call the calcArea function on square and rectangle and the respective methods are called. This means that Go delivers a very intuitive way of polymorphism.

5.5 Persistence with Go

As mentioned in chapter 3.5 Persistence OOP languages often use a garbage collector, which is responsible for deleting unused objects. Go also makes use of such a garbage collection. This results in the fact that the programmer does not need to care about deleting objects. On the other hand is

Go no fit for programming a system with high safety requirements [IGO18].

6. Differences between Go and Objective-C regarding OOP

After evaluating the two programming languages Go and Objective-C regarding their OOP characteristics in the chapters 4. Object Oriented Programming in Objective-C and 5. Object Oriented Programming in Go we will now compare them and state out where it makes sense to use them.

Objective-C does provide a straightforward approach on OOP while also delivering a simple syntax based on the C programming language. It comes with a lot of libraries to develop applications for Apple computers and smartphones. This support from Apple helps the developer in a big way. While Objective-C enables most of the OOP features in an easy way, it has two minor points to criticize.

While multilevel inheritance in Objective-C is intuitive by just adding the parent class to the interface declaration, it would be nice if Objective-C had also a solution for multiple inheritance. The second point to criticize relates to the persistence solution in Objective-C. The garbage collection is provided in just some of the target OS, while on others there is no garbage collection. The question whether a garbage collector makes sense cannot be answered here, but if it is possible to use one it should be available for all targets. If only to avoid programming errors.

Nevertheless Objective-C is an intuitive programming language which delivers easy solutions for most of the OOP programming. Additionally it has to be mentioned that Apple provides a good documentation for the language.

The Go programming language is a prospering language developed by Google and the Go community. It handles the difficulties of languages like C++ regarding memory management and compilation time. To do so, it deliberately does not use a normal OOP approach. Since it is no OOP language one might say the comparison with an OOP language like Objective-C is easy, but it as stated in the chapter 5. Object Oriented Programming in Go Go does provide a possibility to program similar to OOP in all of the criteria considered. As a conclusion, it can be said that Go is no OOP language, but it can be used as one.

As for the question which language is the better one regarding OOP there is no general answer. If the programmer wants to have the OOP functionalities in the language I would recommend the usage of Objective-C. But as for the fact that Objective-C can only be used in the context of Apple products it does not pay off to learn this language if the programmer does not program them often. Go on the other hand can be used in a wide area of use cases since it is not limited to one specific field. But Go also has its limitations. It does not fit well in programming of user interfaces. Objective-C has its main focus on this.

That all being said the programmer has to decide which of the two languages he wants to use for his specific use case. But both of them provide him with solid solutions for OOP programming.

References

COC20 Website, developer.apple.com, called 2020-12-09, https://developer.apple.com/library/archive/documentation/General/Conceptual/CocoaEncyclopedia/Initialization/Initialization.html

COP20 Johannes Weigend, Concepts of Programming Languages, called 2020-12-03, https://github.com/jweigend/concepts-of-programming-languages

FAQ20 Website, golang.org, called 2020-12-03, https://golang.org/doc/faq#Is_Go_an_object-oriented_language

GOL20 Website, golang.org, called 2020-12-03, https://golang.org/

KOP09 Arnd Poetzsch-Heffter, Konzepte objektorientierter Programmierung, Rev. 2, 2009, Springer

IGO18 Website, medium.com, called on 2019-12-03, https://medium.com/gophersland/gopher-vs-object-oriented-golang-4fa62b88c701

OC209 Stephen G. Kochan, Objective-C 2.0, 2009, Addison-Wesley

POC20 Website, developer.apple.com, called 2020-12-09, https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html

POL20 Website, duden.de, called 2020-12-02, https://www.duden.de/rechtschreibung/Polymorphismus

WSP20 Lucas Lukac, getstream.io, called 2020-12-03, https://getstream.io/blog/switched-python-go/