Comparison of object-oriented programming in PHP and Go

Table of Contents

1. Introduction to semester thesis

In the context of this work, the aspects of object-oriented programming for programming languages PHP and Go are compared. For the sake of comprehensibility, the necessary terms are clarified at the beginning of the work. To better understand the differences and similarities between these languages, a small code project was created for this purpose. It is a banking system that is used by bank employees to perform common banking activities such as withdrawing or depositing money or carrying out transactions. Only individual code snippets are used during the comparison. The entire code project can be accessed at the following link.

2. Explanation of terms

This section introduces the languages to be compared, followed by a description of the main features of object-oriented programming.

2.1. Go

Go is a fairly new programming language developed in 2007 by Google employees Robert

Griesemer, Rob Pike and Ken Thompson and released in 2009. The language developed is intended to be expressive, efficient in compilation and execution, and effective in writing reliable and robust programs. Go, like many other languages, is based on the C programming language and is therefore considered a tool for professional programmers that achieves maximum impact with minimal resources. The special feature of Go is the diversity of different tools, since it has adopted many good ideas from other languages while avoiding the features that led to complexity and unreliable code. What is new about the language is the possibilities for concurrency and the approach to data abstraction and object-oriented programming. Because of the possibility of automatic storage (garbage collection), Go is particularly well suited for building infrastructures such as networked servers and tools and systems for programmers. Go programs tend to run faster than programs written in dynamic languages and suffer far fewer crashes due to unexpected type errors.In summary, Go is a general-purpose language that can be found in a wide variety of areas, such as graphics, mobile applications or machine learning. Go is an open source project, i.e. the source code for the compiler, the libraries and the tools is freely available to everyone. Go runs on Unix-like systems - Linux, FreeBSD, OpenBSD, Mac OS X - as well as on Plan9 and Microsoft Windows. Programs written in one of these environments generally work without modification on the others [Donovan].

2.2. PHP

PHP (recursive acronym of PHP: Hypertext Preprocessor) is a widely used open-source general-purpose scripting language that is specifically suited for web programming and can be embedded in HTML. PHP differs from client-side languages such as Javascript in that the code is executed on the server, where it generates HTML output that is then sent to the client. So the client only gets the result of the execution of the script, with no way of knowing what the actual code looks like. The best thing about using PHP is that it is easy for beginners, but it also offers an enormous range of features for professional programmers. Even though PHP development is focused on server-side programming, it is possible to do much more with PHP. Here are three main areas in which PHP scripts are used:

- Server-side programming: This is the traditional and also main field of PHP. With the help of the PHP parser, a web server and a web browser, all web applications can be developed.
- Command line programming: It is also possible to write PHP scripts that can work without a server or browser. Only the PHP parser is needed and different programs can be developed and executed.
- Writing desktop applications: PHP is probably not the very best language to write desktop applications with graphical interface, but with a lot of experience advanced PHP features can be used to write such programs. This way you have the possibility to write cross-platform applications.

PHP can be used on all major operating systems, including Linux, many Unix flavors (including HP-UX, Solaris, and OpenBSD), Microsoft Windows, macOS, RISC OS, and many others. PHP also supports most of the web servers in use today. This includes Apache, Microsoft Internet Information Server, Personal Web Server, Netscape and iPlanet servers, and many others. For the majority of servers PHP provides its own module, for the others that support the CGI standard PHP can work as a CGI processor [PHP].

2.3. Object-oriented programming

Object-oriented programming (OOP for short) is a programming paradigm based on the concept of object orientation. The basic idea is to align the architecture of a software with the basic structures of that domain of reality which concerns the given application. To reduce the complexity of representing reality in a software, object-oriented programming follows four principles: Abstraction, Inheritance, Encapsulation and Polymorphism [OOP].

2.3.1. Abstraction

Abstraction is one of the key concepts of object-oriented programming. The main goal is to manage the complexity of the program by hiding unnecessary details from the user. This allows the user to implement more complex logic on the abstraction provided without understanding or even thinking about all the hidden complexity [OOP].

2.3.2. Inheritance

In object-oriented programming, inheritance is understood as a procedure in which one class takes over the properties of another class. The inherited properties include attributes, methods as well as events. With the inheritance a new class or subclass is developed as extension of one or several already existing classes (superclasses). The goal of inheritance is thus to reuse the parent class properties. A subclass inherits in principle the specification of its superclass. The subclass takes over thereby all obligations and warranties of the superclass. With the procedure of the inheritance it is possible to specify and define the attributes, methods and events common to several classes at central place. Already defined characteristics of a class should not have to be defined again. This applies in particular if classes are already very similar [OOP].

2.3.3. Encapsulation

In object-oriented programming, the data explicitly belong to an object, so that direct access as to the data structures in structured programming is no longer permitted. Objects have thus the exclusive right to change their data or also read access to them. If a user (for example, another object) wants to read or change this data, he must contact the object via a clearly defined interface and request a change to the data. The biggest advantage is that the object itself can ensure that the consistency of the data is maintained. This makes it easier to ensure the correctness of the program. In addition, the effort for changes is reduced, because the internal data and functions of an object can be changed without having to adjust other objects as well [OOP].

2.3.4. Polymorphism

In the field of object orientation, polymorphism refers to the fact that different objects can exhibit different behavior when the same operation is called. A distinction is made here between dynamic and static polymorphism. Object-oriented systems, which support dynamic polymorphism, are able to assign objects of different type to a variable. The type of the variable itself describes only an interface. The variable can then be assigned all objects whose class implements this interface. Which method is called when an operation is invoked depends on the class membership of the object assigned to the variable. We speak of static polymorphism (overloading) when the invocation of an operation is mapped to a method based on the concrete type of variables or constants. Unlike dynamic polymorphism, the contents of the variables do not play a role in deciding which concrete method to call [OOP].

3. Comparison between PHP and Go on object-oriented programming

This section shows how the programming languages PHP and Go implement the aspects of object-oriented programming presented in Section 2.3 Code examples are given for PHP and Go for all aspects. The examples are simplified code snippets from the code project for a banking system.

3.1. Abstraction in PHP and Go

Abstraction is realized in PHP through abstract classes and methods. An abstract class has at least one abstract method. Abstract methods provide only the declaration of the signature of the method, but the implementation is left to the subclasses. In the implementation, the subclasses have some freedom such as adding additional arguments or changing the visibility of the method to a higher level. An abstract class or method is defined with the abstract keyword [PHPDocs]. The following code snippets show an example from the code project of an abstract method *showInfos()* of the abstract class *bank* account which is implemented differently by the two subclasses *savingsAccount* and *checkingAccount*.

```php
abstract class BankAccount
{
    /** @return string */
    public abstract function showInfos() : string;
}
```

```php
class CheckingAccount extends BankAccount
{
    /** @return string */
    public function showInfos(): string
    {
        return $this->getAccountNumber()."   |   ".
            "CheckingAccount" ."   |   ".
            $this->getBalance()."   |   ".
            $this->getInterestRate()."   |   ".
            $this->getAccountMaintenanceCharge()."\n";
    }
}
```

```
class SavingsAccount extends BankAccount
{
    /** @return string */
    public function showInfos(): string
    {
        return $this->getAccountNumber()."   |   ".
            "SavingsAccount" ."   |   ".
            $this->getBalance()."   |   ".
            $this->getInterestRate()."\n";
    }
}
```

Go, on the other hand, does not support abstraction through the usual constructs like other object-oriented programming languages. Abstraction is realized in Go through interfaces [GoDocs]. Here is a similar example as in the code project:

```
type iBankAccount interface {
    showInfos()
}
type bankAccount struct {
    accountNumber string
    balance float32
}
type checkingAccount struct {
    bankAccout
    interestRate float32
    accountMaintenanceCharge float32
}
type savingsAccount struct {
    bankAccount
    interestRate float32
}
func (c *checkingAccount) showInfos() {
    fmt.Printf("%s | %s | %s | %s | %s ",
    c.accountNumber, c.balance, c.interestRate, c.accountMaintenanceCharge)
}
func (s *savingsAccount) showInfos() {
    fmt.Printf("%s | %s | %s | %s ",
    c.accountNumber, c.balance, c.interestRate)
}
```

3.2. Inheritance in PHP and Go

To inherit from a class, PHP has the keyword *extends*. This gives the subclass access to all the properties (attributes and methods) of the superclass and can even override them. If you want to keep or add the functionality of the superclass you can use the command *parent::methodName()* to include the implementation of the superclass [PHPDocs]. The following three code snippets show the inheritance example from the code project. The inherited property is shown here as the constructor with both, its own attributes and the inherited attributes of the superclass.

```php
abstract class BankAccount
{
    /** @var string */
    private string $accountNumber;
    /** @var Customer */
    private Customer $customer;
    /** @var float */
    private float $balance;
    /**
     * @param string $accountNumber
     * @param Customer $customer
     * @param float $balance
     */
    public function __construct(string $accountNumber, Customer $customer, float
$balance = 0.0)
    {
        $this->accountNumber = $accountNumber;
        $this->customer = $customer;
        $this->balance = $balance;
    }
}
```

```php
class CheckingAccount extends BankAccount
{
private float $interestRate;
    private float $accountMaintenanceCharge;
    /**
     * @param string $accountNumber
     * @param Customer $customer
     * @param float $balance
     * @param float $interestRate
     * @param float $accountMaintenanceCharge
     */
    public function __construct(string $accountNumber, Customer $customer, float
$balance, float $interestRate = 0.05, float $accountMaintenanceCharge = 7.0)
    {
        parent::__construct($accountNumber, $customer, $balance);
        $this->interestRate = $interestRate;
        $this->accountMaintenanceCharge = $accountMaintenanceCharge;
    }
}
```

```
class SavingsAccount extends BankAccount
{
/** @var float */
private float $interestRate;
    /**
     * @param string $accountNumber
     * @param Customer $customer
     * @param float $balance
     * @param float $interestRate
     */
    public function __construct(string $accountNumber, Customer $customer, float
$balance, float $interestRate = 0.8)
    {
        parent::__construct($accountNumber, $customer, $balance);
        $this->interestRate = $interestRate;
    }
}
```

Since Go does not support the typical inheritance via classes, some kind of inheritance is realized through embedding structures. Structs cannot be extended directly, but use a concept called composition, where the struct is used to form other objects. With composition, both the attributes and the methods of the base struct can be embedded and called in the child struct [GoDocs]. The following example shows the realization of inheritance in Go using the example of the banking system.

```
type bankAccount struct {
    accountNumber string
    balance       float32
}
func (b *bankAccount) debit(amount float32) {
    b.balance = b.balance - amount
}
type savingsAccount struct {
    bankAccount
    interestRate float32
}
func main() {
    savingsAcc := savingsAccount{bankAccount{"1", 10}, 0.05}
    fmt.Println("Bevor: ", savingsAcc.balance) //Bevor: 10
    savingsAcc.debit(2)
    fmt.Println("After: ", savingsAcc.balance) //After: 8
}
```

3.3. Encapsulation in PHP and Go

The access to a property, method or constant in PHP can be defined by prefixing the declaration with one of the keywords *public*, *protected* or *private*. Publicly declared elements can be accessed from anywhere. Protected restricts access to parent classes and derived classes (as well as the class that defines the element). Private limits visibility only to the class that defines the element [PHPDocs]. For demonstration purposes, the code project was modified so that the three attributes have different visibilities. The *accountNumber* can be called from anywhere, the *customer* only in the *Models* namespace and the *balance* only in the *BankAccount* class.

```
abstract class BankAccount
{
    /** @var string */
    public string $accountNumber;
    /** @var Customer */
    protected Customer $customer;
    /** @var float */
    private float $balance;
}
```

Encapsulation in Go happens only at the package level. In contrast to PHP, Go has no *public*, *private* or *protected* keywords. Visibility is controlled by using formats with and without uppercase letters. Uppercase identifiers are exported and can be used outside of package, while lowercase identifiers are not exported and can only be accessed only within the same package. Visibility control can be applied to the following identifiers: structure, struct method, structure field, function, variable [GoDocs]. In the following example, it is not possible to access the bank account attribute *accountNumber* in the *main* package due to encapsulation. Since the *Balance* attribute is capitalized, it can also be accessed from external packages such as *main*.

```
package bank
type BankAccount struct {
    accountNumber string
    Balance       float32
}
```

```
package main
import "fmt"
func main() {
    b := BankAccount{"1", 10}
    b.accountNumber // Not allowed
    b.Balance //allowed
}
```

3.4. Polymorphism in PHP and Go

Polymorphism in PHP can be implemented either by using interfaces or abstract classes. Interfaces are constructs that are similar to classes, with only one main difference. Interfaces define the method header, but not the functionality of the method. The implementation of the method is then done in those classes that implement this interface. These classes are obliged to implement all methods of the interface [PHPDocs]. The following two code snippets show the example of the *Banking* interface and its implementation in the *Bank* class.

```php
interface Banking
{
    /**
    * @param BankAccount $from
    * @param BankAccount $to
    * @param float $amount
    * @return bool
    */
    function doTransaction(BankAccount $from, BankAccount $to, float $amount) : bool;
}
```

```php
class Bank implements Banking
{
    /**
     * @param BankAccount $from
     * @param BankAccount $to
     * @param float $amount
     * @return bool
     * @throws TransactionFailedException|InvalidAmountException
     */
    public function doTransaction(BankAccount $from, BankAccount $to, float $amount) :
bool{
        if ($amount > 0 && $from->getBalance() >= $amount) {
            $from->debit($amount);
            $to->deposit($amount);
            return true;
        } else {
            throw new TransactionFailedException("Make sure that the amount is greater
than 0 and that there is enough money to be debited!");
        }
    }
}
```

Polymorphism in Go is implemented using interfaces. A type implements an interface if it provides definitions for all methods declared in the interface [GoDocs]. An example of Polymorphism in Go is shown in section 3.1.

4. Special features of object-oriented programming in PHP

Since PHP fully supports object orientation, functions have evolved over time that are not used or even offered in every object-oriented language, especially not in Go. This chapter introduces a couple of those features.

4.1. Traits

One of the main goals of object orientation is code reuse. This goal is achieved through inheritance by moving common functionality and properties to the parent class. However, this makes the code very tightly linked and can lead to excessive use of inheritance. To solve this problem PHP has traits. Traits allow the free reuse of different methods in many classes that do not have to be in the same class hierarchy. A trait is similar to a class, but cannot be instantiated. To use a trait, you use the keyword *use Traitname* and can thus access all the methods of the trait. [PHPOop]

4.2. Autoloading

In programming, and especially in object-oriented programming, you try to group similar classes into packages. This has the consequence that one must load the classes which are in another package, first in order to use them. In PHP you can import a class quite simply with *require_once /package/classname.php*. However, if the number of classes increases, which is quite common in larger software, the *require_once* option is suboptimal. For this, PHP offers the possibility to import whole packages with one command through *autoloader*. There are two options available: - implement your own autoloader that loads a specific package (with *spl_autoload_register()*) - use dependency manager *Composer* (see Composer) which takes care of the automatic loading of the classes [PHPOop]

4.3. Exception Handling

PHP has an exception model similar to that of other programming languages. An exception can be thrown and caught in PHP. To enable catching of potential exceptions, the respective code should be enclosed by a try block. Each try block must have at least one associated catch or finally block. PHP offers a large set of predefined exceptions, but also the possibility to write your own exceptions. A custom exception class can be created by deriving from the built-in exception class and defined by custom logic [PHPOop]. Here is an example:

```php
class NoAccountException extends Exception
{
    public function __construct($message = "", $code = 0, Throwable $previous = null)
    {
        parent::__construct($message, $code, $previous);
    }
    public function getErrorMessage(): string
    {
        return "\nAn account with this account number does not exists!\n";
    }
}
```

5. Conclusion

This thesis describes similarities and differences between PHP and Go in terms of object-oriented programming. Although Go was not designed for object orientation, it offers many possibilities to make the program object-oriented. PHP, on the other hand, was designed from the beginning to be object-oriented and therefore offers more features that can facilitate such programming. Despite many differences, both languages have their weaknesses and strengths and can be used for a variety of different applications. In my opinion, PHP will continue to evolve and remain in use, but due to the fact that PHP is already an embedded language, I don’t think any revolutionary features will be developed. Go, on the other hand, is a fairly young programming language with a lot of flexibility, simplicity and potential for the future of programming.

References

- [Donovan] Donovan, Alan A. A., Brian W. Kernighan. The Go programming language. New York: Addison-Wesley. 2016.
- [PHP] Manual. PHP: Was kann PHP?. https://www.PHP.net/manual/de/intro-whatcando.PHP, 06.01.2022
- [OOP] Bernhard Lahres, Gregor Raýman, Stefan Strich. Objektorientierte Programmierung. Das umfassende Handbuch. Rheinwerk Computing. 2021
- [PHPDocs] PHP Documentation. https://www.php.net/docs.php. 08.01.2022
- [GoDocs] Go Documentation. https://go.dev/doc/. 08.01.2022
- [PHPOop] PHP OOP. https://www.phptutorial.net/php-oop/. 09.01.2022