Vergleich zwischen Go und Lua in der OOP

Semi Hasani

Agenda

Gegenüberstellung von OOP in Go und Lua

- Klassen und Objekte
- Kapselung
- Vererbung

Projektvorstellung

- Live Demo
- Code

Klassen & Objekte in Go

- Go verwendet keine Klassen sondern „structs“
- Um dem Struct Methoden hinzuzufügen wird jede Methode mit einem „receiver“ definiert
- In Go werden Konstruktoren oft nicht benötigt, da Literale eine einfache und direkte Initialisierung ermöglichen

```
type TodoItem struct {
    Description string
    Completed   bool
}

func (t *TodoItem) Toggle() {
    t.Completed = !t.Completed
}

item := TodoItem{
    Description: "Einkaufen gehen",
    Completed:   false,
}
```

Klassen & Objekte in Lua

- Keine echten Klassen, sondern Tabellen als Basis
- Konstruktoren werden benötigt, da Tabellen keine automatische Initialisierung haben
- Funktionen werden der Tabelle mit „Tabellenname: Funktionsname“ angeheftet

```lua
TodoItem = {}

function TodoItem:new(description, date)
    local obj = {
        description = description,
        date = date,
        completed = false
    }
    setmetatable(obj, self)
    self.__index = self
    return obj
end

function TodoItem:toggle()
    self.completed = not self.completed
end
```

Klassen & Objekte

Vergleich

- Go
  - Strikt und Typensicher
  - Statisch / Structs enthalten feste Attribute

  –> Geringere Fehleranfälligkeit

- Lua
  - Flexibler aber nicht Typensicher
  - Dynamisch / Attribute und Methoden können zur Laufzeit Tabellen hinzugefügt werden

  -> Höhere Fehleranfälligkeit

Kapselung in Go

- Go hat keine speziellen Schlüsselworte wie
  - private
  - protected
  - public

- Felder oder Methoden in einem Struct sind:
  - Kleingeschrieben : private
  - Großgeschrieben : public

```go
type Person struct {
    Name string // Öffentlich: kann von außerhalb des Pakets
    age  int    // Privat: nur innerhalb des Pakets sichtbar
}

// Öffentliche Methode
func (p *Person) GetAge() int {
    return p.age
}

// Private Methode
func (p *Person) setAge(age int) {
    p.age = age
}
```

Kapselung in Lua

- Lua verwendet das Schlüsselwort local, um Attribute privat zu machen
- Attribute ohne local sind global und von überall aus zugreifbar
- Der Zugriff auf private Attribute erfolgt über Getter- und Setter-Methoden

```lua
TodoItem = {}

function TodoItem:new(description, date)
    local obj = {}
    local completed = false -- Das ist privat

    -- Methode, um den Status zu ändern
    function obj:toggle()
        completed = not completed
    end

    -- Methode, um den Status zu lesen
    function obj:isCompleted()
        return completed
    end

    setmetatable(obj, self)
    self.__index = self
    return obj
end
```

Vererbung in Go

- In Go gibt es keine traditionelle Verbung, es gibt Kompositionen
- Kleinere Structs werden in größere eingesetzt
- Employee kann Methoden von Person aufrufen aber nicht umgekehrt

```go
// Basisstruktur
type Person struct {
    Name string
}

// Methode von Person
func (p Person) Greet() string {
    return "Hallo, " + p.Name
}

// Erweiterte Struktur (Komposition)
type Employee struct {
    Person
    Position string
}

func main() {
    e := Employee{Person: Person{Name: "Max"}, Position: "Engineer"}
    fmt.Println(e.Greet())       // Hallo, Max
    fmt.Println(e.Position)      // Engineer
}
```

Vererbung in Lua

- In Lua gibt es Verbung und Kompositionen
- Methoden der Basisklasse können immer Felder der Kindklasse zugreifen und umgekehrt
- „self" wird verwendet, um auf das aktuelle Objekt zuzugreifen
- Methoden können von der Kindklasse überschrieben werden

```lua
local TodoList = {}
TodoList.__index = TodoList

function TodoList:new()
    local instance = { items = {} }
    setmetatable(instance, self) -- Setzt TodoList als Metatable für instance
    return instance
end

function TodoList:addItem(item)
    table.insert(self.items, item)
    print(item .. " hinzugefügt.")
end

function TodoList:listItems()
    for i, item in ipairs(self.items) do
        print(i .. ": " .. item)
    end
end

-- Instanz erstellen und Methoden aufrufen
local instance = TodoList:new() -- instance "erbt" von TodoList
instance:addItem("Einkaufen") -- Ruft die Methode addItem von TodoList auf
instance:addItem("Hausaufgaben")
instance:listItems() -- Ruft die Methode listItems von TodoList auf
```

Vererbung

Vergleich

Go

- Verwendet Komposition, keine klassische Vererbung
- Statisch typisiert, sicher
- Klare Trennung zwischen Strukturen – keine Two-Way-Verbindung

Lua

- Unterstützt Vererbung über Metatables und __index
- Dynamisch und flexibel
- Basisklassen können durch self auf Felder der Kindklasse zugreifen (Two-Way-Verbindung)

Projekt - vorstellung

Go vs. Lua - OOP

Go

- Ideal für größere, robuste und skalierbare Projekte
- Statisch und typesafe: Fehler werden bereits zur Kompilierzeit erkannt
- Weniger flexibel: Attribute und Methoden müssen vorab definiert werden

Lua

- Gut geeignet für kleinere Projekte oder Prototypen
- Einfach und schnell: Anwendungen können ohne großen Overhead entwickelt werden
- Weniger sicher: Keine Typprüfung, höhere Fehleranfälligkeit bei komplexeren Anwendungen