Advanced Programming 1

Assignment 1 – GoLang Basics (from packages to methods)

These exercises cover Lecture 1 & 2 topics. See https://go.dev/tour/basics/1 & videos of lectures.

Deadline: 16:00 23.12.2024

Defense will take place on your 2-hour classes on 3rd (23.12 - 27.12) week.

Submission: submit an archive named as Assignment1_NameSurname.zip file to moodle.

Exercise 1 [25%]

Build a package to manage a library's book collection using structures, maps, and methods. Package name must start with github.com/NameSurname/Assignment1/Library

Define a struct Book with the following fields:

- ID (string)
- Title (string)
- Author (string)
- IsBorrowed (bool)

Create a Library struct that maintains a collection of books using a map[string]Book where the key is ID.

Implement the following methods for Library:

- AddBook(book Book) to add a new book.
- BorrowBook(id string) to borrow a book. Update IsBorrowed to true.
- ReturnBook(id string) to return a borrowed book.
- ListBooks() to print all books in the library.

Use loops to iterate over the books in the map.

Usage example when main.go is started:

User input:

1. Add – will add a book to a map
2. Borrow – will change IsBorrowed state from false to true only.
3. Return – will change IsBorrowed state from true to false only.
4. List – will show all available books (not borrowed)
5. Exit – finish program execution.

Exercise 2 [25%]

Implement an interface for calculating the area and perimeter of different shapes. Create a new package github.com/NameSurname/Assignment1/Shapes

Define an interface Shape with two methods:

- Area() float64
- Perimeter() float64

Create structs like Rectangle, Circle, Square, Triangle with relevant fields:

struct Rectangle:

- Length float64
- Width float64

struct Circle

- Radius float64

struct Square

- Length float64

struct Triangle

- SideA float64
- SideB float64
- SideC float64

Implement the Shape interface for all structs.

Write a function PrintShapeDetails(s Shape) that takes a Shape interface and prints its area and perimeter.

Use a slice of Shape to store multiple shapes and iterate using a loop.

Example:

```
shapes := []Shape{
    Rectangle{Length: 10, Width: 5},
    Circle{Radius: 7},
}

for _, shape := range shapes {
    PrintShapeDetails(shape)
}
```

Exercise 3 [25%]

Implement an employee management system using interfaces, maps, and methods.

Define an interface Employee with a method GetDetails() string.

```
type Employee interface {
    GetDetails() string
}
```

Create two structs FullTimeEmployee and PartTimeEmployee:

type FullTimeEmployee struct:

- ID uint64
- Name string
- Salary uint32

PartTimeEmployee:

- ID uint64
- Name string
- HourlyRate uint64 // amount in Tenge e.g. 5000 tg per hour -> 5000
- HoursWorked float32 // total hours worked. E.g. 20,5 hours -> 20.5

Implement the Employee interface for both structs:

- GetDetails() should return a formatted string with employee details.

Create a Company struct that maintains employees in a map[string]Employee.

Implement methods for Company:

- AddEmployee(emp Employee)
- ListEmployees()

Use a loop to iterate over the map and display employee details.

Usage example:

```
c := NewCompany()
c.AddEmployee(FullTimeEmployee{ID: "1", Name: "Alice", Salary: 5000})
c.AddEmployee(PartTimeEmployee{ID: "2", Name: "Bob", HourlyRate: 20, HoursWorked: 100})

c.ListEmployees()
```

Exercise 4 [25%]

Simulate a bank account system using structs, methods, and interfaces.

Define a struct BankAccount with the following fields:

- AccountNumber string
- HolderName string
- Balance float64

Implement the following methods for BankAccount:

- Deposit(amount float64) // to add money.
- Withdraw(amount float64) // to deduct money if sufficient balance exists.
- GetBalance() // to print the current balance.

Write a function Transaction(account *BankAccount, transactions []float64) that processes a slice of transaction amounts.

- Positive amounts are deposits.
- Negative amounts are withdrawals.

Write a runner function that will be using loop to work with this system.

Usage example:

User input here:

1. Deposit – will trigger Deposit method
2. Withdraw – will trigger Withdraw method
3. Get balance – will trigger GetBalance method
4. Exit – finish program execution.