ЛАБОРАТОРНА РОБОТА № 5
ООП. Структури та інтерфейси

Мета: засвоїти принципи проектування та оголошення структур та інтерфейсів; вивчити особливості реалізації структур та їх методів.

Хід роботи:

Завдання.

1. Оголосити структуру Worker, яка представляє інформацію про працівника і містить наступні поля:

- Name – прізвище та ініціали працівника;
- Year – рік початку роботи;
- Month – місяць початку роботи;
- WorkPlace – місце роботи.

Оголосити структуру Company, яка містить наступні поля:

- Name – назва компанії;
- Position – посада працівника;
- Salary – зарплата працівника.

2. Для кожної структури реалізувати конструктори:
3. Реалізувати set- та get- методи для кожного поля.
4. У структурі Worker створити методи:

- GetWorkerPosition(), який повертає посаду робочого.
- GetWorkExperience(), який обраховує і повертає стаж роботи на підприємстві у місяцях.
- GetTotalMoney(), що повертає загальну суму зароблених коштів за усі місяці роботи.

У структурі Company методи обрати самостійно.

5. Реалізувати ф-ї:

- ReadWorkersArray() – читає з клавіатури дані і повертає множину об’єктів

типу Worker (n штук);

- PrintWorker() – приймає тип Worker і виводить його на екран;
- PrintWorkers() – приймає множину типу Worker і виводить його на екран;
- GetWorkersInfo() – приймає множину типу Worker і повертає через вихідні параметри найбільшу та найменшу зарплату серед усіх працівників.

Структура програми:

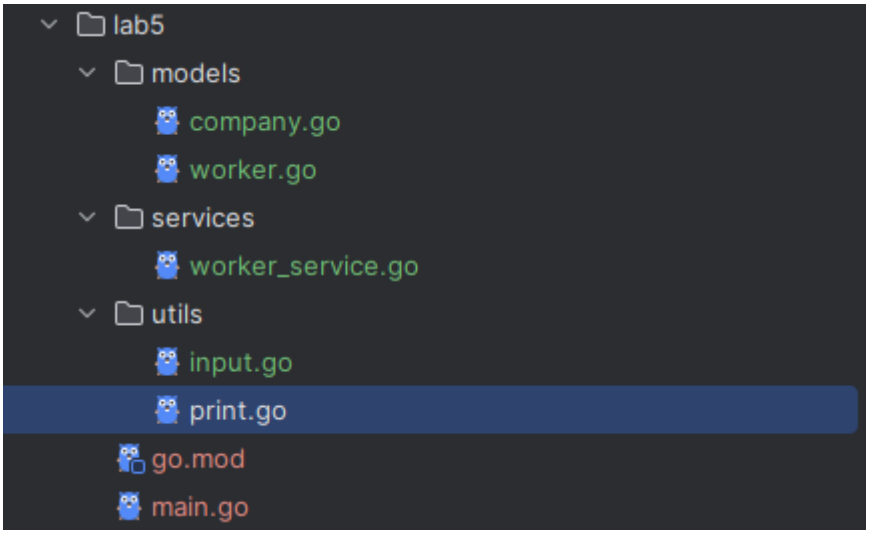

Рис. 1. Структура програми.

Листинг файлів company.go та worker.go:

```go
package models

type Company struct {
    Name     string
    Position string
    Salary   float64
}

func NewCompany(name string, pos string, sal float64) *Company {
    return &Company{name, pos, sal}
}

func (c *Company) SetName(n string)     { c.Name = n }
func (c *Company) SetPosition(p string) { c.Position = p }
func (c *Company) SetSalary(s float64)  { c.Salary = s }

func (c *Company) GetName() string     { return c.Name }
func (c *Company) GetPosition() string { return c.Position }
func (c *Company) GetSalary() float64  { return c.Salary }
```

```go
package models

import "time"

type Worker struct {
    Name       string
    StartYear  int
    StartMonth int
    WorkPlace  string
    Company    Company
}

func NewWorker(name string, year, month int, place string, company Company) *Worker {
```

```
    return &Worker{
       Name:       name,
       StartYear:  year,
       StartMonth: month,
       WorkPlace:  place,
       Company:    company,
    }
}

func (w *Worker) SetName(name string)        { w.Name = name }
func (w *Worker) SetStartYear(year int)      { w.StartYear = year }
func (w *Worker) SetStartMonth(month int)    { w.StartMonth = month }
func (w *Worker) SetWorkPlace(place string) { w.WorkPlace = place }
func (w *Worker) SetCompany(c Company)       { w.Company = c }

func (w *Worker) GetName() string      { return w.Name }
func (w *Worker) GetStartYear() int    { return w.StartYear }
func (w *Worker) GetStartMonth() int   { return w.StartMonth }
func (w *Worker) GetWorkPlace() string { return w.WorkPlace }
func (w *Worker) GetCompany() Company  { return w.Company }

func (w *Worker) GetPosition() string {
    return w.Company.Position
}

func (w *Worker) GetWorkExperience() int {
    now := time.Now()
    years := now.Year() - w.StartYear
    months := int(now.Month()) - w.StartMonth
    return years*12 + months
}

func (w *Worker) GetTotalEarnings() float64 {
    return float64(w.GetWorkExperience()) * w.Company.Salary
}
```

Листинг файлів input.go та print.go:

```
package utils

import (
    "bufio"
    "fmt"
    "os"
    "strconv"
    "strings"

    "lab5/models"
)

func ReadString(prompt string) string {
    reader := bufio.NewReader(os.Stdin)
    for {
       fmt.Print(prompt)
       text, _ := reader.ReadString('\n')
       text = strings.TrimSpace(text)
       if text != "" {
          return text
       }
       fmt.Println("Input cannot be empty.")
    }
}

func ReadInt(prompt string) int {
    for {
       fmt.Print(prompt)
       var raw string
       _, err := fmt.Scan(&raw)
       if err != nil {
          return 0
       }
       if val, err := strconv.Atoi(raw); err == nil {
          return val
       }
       fmt.Println("Invalid integer. Try again.")
```

```
    }
}

func ReadFloat(prompt string) float64 {
    for {
        fmt.Print(prompt)
        var raw string
        _, err := fmt.Scan(&raw)
        if err != nil {
            return 0
        }
        if val, err := strconv.ParseFloat(raw, 64); err == nil {
            return val
        }
        fmt.Println("Invalid number. Try again.")
    }
}

func ReadWorkers(n int) []*models.Worker {
    workers := make([]*models.Worker, 0, n)

    for i := 0; i < n; i++ {
        fmt.Printf("\n--- Worker %d ---\n", i+1)

        name := ReadString("Name: ")
        year := ReadInt("Start year: ")
        month := ReadInt("Start month: ")
        place := ReadString("Workplace: ")

        companyName := ReadString("Company name: ")
        position := ReadString("Position: ")
        salary := ReadFloat("Salary: ")

        company := models.NewCompany(companyName, position, salary)
        worker := models.NewWorker(name, year, month, place, *company)

        workers = append(workers, worker)
    }

    return workers
}
```

```
package utils

import (
    "fmt"

    "lab5/models"
)

func PrintWorker(w *models.Worker) {
    fmt.Println("------------ Worker ------------")
    fmt.Println("Name:", w.Name)
    fmt.Printf("Start: %d-%02d\n", w.StartYear, w.StartMonth)
    fmt.Println("Workplace:", w.WorkPlace)
    fmt.Println("Position:", w.Company.Position)
    fmt.Printf("Salary: %.2f\n", w.Company.Salary)
}

func PrintWorkers(workers []*models.Worker) {
    for _, w := range workers {
        PrintWorker(w)
    }
}
```

Листинг файлу worker_service.go:

```
package services

import (
    "lab5/models"
    "lab5/utils"
)
```

```go
type WorkerService interface {
    ReadWorkers() []*models.Worker
    PrintWorker(*models.Worker)
    PrintWorkers([]*models.Worker)
    GetSalaryRange([]*models.Worker) (min float64, max float64)
}

type WorkerServiceImpl struct{}

func NewWorkerService() WorkerService {
    return &WorkerServiceImpl{}
}

func (s *WorkerServiceImpl) ReadWorkers() []*models.Worker {
    n := utils.ReadInt("Enter number of workers: ")
    return utils.ReadWorkers(n)
}

func (s *WorkerServiceImpl) PrintWorker(w *models.Worker) {
    utils.PrintWorker(w)
}

func (s *WorkerServiceImpl) PrintWorkers(workers []*models.Worker) {
    for _, w := range workers {
       s.PrintWorker(w)
    }
}

func (s *WorkerServiceImpl) GetSalaryRange(workers []*models.Worker) (min float64, max float64) {
    if len(workers) == 0 {
       return 0, 0
    }

    min = workers[0].Company.Salary
    max = workers[0].Company.Salary

    for _, w := range workers {
       sal := w.Company.Salary
       if sal < min {
          min = sal
       }
       if sal > max {
          max = sal
       }
    }

    return min, max
}
```

Листинг файлу main.go:

```go
package main

import (
    "fmt"
    "lab5/services"
)

func main() {
    var workerService services.WorkerService
    workerService = services.NewWorkerService()

    workers := workerService.ReadWorkers()

    fmt.Println("\n=== Workers List ===")
    workerService.PrintWorkers(workers)

    minSal, maxSal := workerService.GetSalaryRange(workers)
    fmt.Printf("\nMinimum salary: %.2f\n", minSal)
    fmt.Printf("Maximum salary: %.2f\n", maxSal)
}
```

Результат виконання програми:

```
Enter number of workers: 4

--- Worker 1 ---
Name: Input cannot be empty.
Name: Jan
Start year: 2024
Start month: 3
Workplace: Input cannot be empty.
Workplace: Origon
Company name: Z
Position: Manager
Salary: 1600

--- Worker 2 ---
Name: Input cannot be empty.
Name: 353
Start year: fdfdf
Invalid integer. Try again.
Start year: 2025
Start month: sf
Invalid integer. Try again.
Start month: 12
Workplace: Input cannot be empty.
Workplace: 33fff
Company name: s2525s
Position: 574
Salary: 17777

--- Worker 3 ---
Name: Input cannot be empty.
Name: 5asf
Start year: 20223
Start month: 24512
Workplace: Input cannot be empty.
Workplace: 12
Company name: sdasd
Position: fawvc
Salary: 2013

--- Worker 4 ---
Name: Input cannot be empty.
Name: 512asd
Start year: 2022
Start month: 12424
Workplace: Input cannot be empty.
Workplace: 12
Company name: dasdagw
Position: fawfa
Salary: 241245

=== Workers List ===
------------ Worker ------------
Name: Jan
Start: 2024-03
Workplace: Origon
Position: Manager
Salary: 1600.00
--------------------------------
------------ Worker ------------
Name: 353
Start: 2025-12
Workplace: 33fff
Position: 574
Salary: 17777.00
--------------------------------
------------ Worker ------------
Name: 5asf
Start: 20223-24512
Workplace: 12
Position: fawvc
Salary: 2013.00
--------------------------------
------------ Worker ------------
Name: 512asd
Start: 2022-12424
Workplace: 12
Position: fawfa
Salary: 241245.00
--------------------------------

Minimum salary: 1600.00
Maximum salary: 241245.00
```

Рис. 2-3. Результат виконання програми.

Посилання на репозиторій - https://github.com/JanRizhenko/GoLang-sigma-practice

Висновок: Під час вивчення теми «ООП. Структури та інтерфейси» було засвоєно основні принципи проєктування та організації даних у мові програмування Go. Структури виявилися потужним інструментом для логічного об’єднання пов’язаних даних та створення власних типів, що дозволяють зручно моделювати об’єкти реального світу. Вивчення методів структур показало, як можна реалізовувати поведінку об’єктів, зберігаючи при цьому контроль над станом і забезпечуючи інкапсуляцію. Особливу увагу було приділено використанню приймачів — як по значенню, так і по вказівнику — та наслідкам цього вибору для ефективності та консистентності коду. Ознайомлення з інтерфейсами дало розуміння того, як можна визначати поведінку, незалежну від конкретних реалізацій, що значно підвищує гнучкість програм і дозволяє застосовувати принципи поліморфізму. В процесі роботи було відпрацьовано практичні навички оголошення структур, реалізації методів та створення інтерфейсів, а також інтеграції всіх цих елементів у модульну та логічно зрозумілу архітектуру програми. Ця тема допомогла закріпити фундаментальні концепції об’єктно-орієнтованого підходу в контексті мови Go, навчила думати про структури і їх взаємодію як про єдину систему та сформувала базу для розробки масштабованих, читабельних і підтримуваних програмних рішень.