МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «Белгородский государственный национальный исследовательский университет»

Лабораторная работа №8
дисциплина «Современные методы, среды и языки программирования»
по теме «Создание многопоточных программ с синхронизированным доступом к ресурсам в Java»

Выполнил: студент группы 12002135 Макаров Д.С.
Проверил:

Белгород 2021

Лабораторная работа №8

«Создание многопоточных программ с синхронизированным доступом к ресурсам в Java»

Цель работы:Научиться создавать программный код решения типовых задач на языке Java с применением принципов многопоточного исполнения команд и управления доступом к разделяемым ресурсам.

Задание

Счета. Клиент может иметь несколько счетов в банке. Учитывать возможность блокировки/разблокировки счета. Реализовать поиск и сортировку счетов. Вычисление общей суммы по счетам. Вычисление суммы по всем счетам, имеющим положительный и отрицательный балансы отдельно.

Ход работы

Исходный код см. в приложении.

Приложение

Содержимое файла client.go

```
package client

import (
        "fmt"
        "lab8/pkg/account"
        "math/rand"
        "sort"
        "sync"

        "github.com/google/uuid"
)

type Accounts []account.Account

func (a Accounts) Len() int {
        return len(a)
}

func (a Accounts) Less(i, j int) bool {
        return a[i].Balance < a[j].Balance
}

func (a Accounts) Swap(i, j int) {
        a[i], a[j] = a[j], a[i]
}

type Client struct {
        Name     string
        accounts Accounts
        mutex    sync.Mutex
}

func (c *Client) OpenAccount() {
        c.mutex.Lock()
        c.accounts = append(c.accounts, account.CreateAccount())
        c.mutex.Unlock()
}

func (c *Client) CloseAccount() {
        c.mutex.Lock()
        l := c.accounts.Len()
        if l > 0 {
                index := rand.Intn(l)
                c.accounts = append(c.accounts[:index], c.accounts[index+1:]...)
        }
        c.mutex.Unlock()
}

func (c *Client) SortAccounts(reverseFlag bool) {
        c.mutex.Lock()
        if reverseFlag {
                sort.Sort(sort.Reverse(c.accounts))
        } else {
                sort.Sort(c.accounts)
        }
        c.mutex.Unlock()
}

func (c *Client) PrintSumAccount() {
        c.mutex.Lock()
        sum := int64(0)
        for _, item := range c.accounts {
                sum += item.Balance
        }
        fmt.Printf("Sum of accounts: %f rub\n", account.FormatBalance(sum))
        c.mutex.Unlock()
}

func (c *Client) PrintPosNegSumAccount() {
        c.mutex.Lock()
        posSum := int64(0)
        negSum := int64(0)
        for _, item := range c.accounts {
```

```
                if item.IsNegative() {
                        negSum += item.Balance
                } else {
                        posSum += item.Balance
                }
        }
        fmt.Printf("Sum of positive accounts: %f rub\n", account.FormatBalance(posSum))
        fmt.Printf("Sum of negative accounts: %f rub\n", account.FormatBalance(negSum))
        c.mutex.Unlock()
}

func CreateClient() Client {
        name, err := uuid.NewRandom()
        if err != nil {
                panic(err)
        }
        return Client{Name: name.String(), accounts: Accounts{}, mutex: sync.Mutex{}}
}
```

Содержимое файла account.go

```
package account

import (
        "math/rand"

        "github.com/google/uuid"
)

type Account struct {
        Name     string
        Balance  int64
        isLocked bool
}

func (a Account) IsNegative() bool {
        return a.Balance > 0
}

func FormatBalance(balance int64) float32 {
        return float32(balance) / 100
}

func (a *Account) SetLock(flag bool) {
        a.isLocked = flag
}

func CreateAccount() Account {
        name, err := uuid.NewRandom()
        if err != nil {
                panic(err)
        }
        newAccount := Account{Name: name.String(), Balance: rand.Int63(), isLocked: false}
        return newAccount
}
```

Содержимое файла main.go

```
package main

import (
        "lab8/pkg/client"
        "math/rand"
)

// Счета. Клиент может иметь несколько счетов в банке. Учитывать
// возможность блокировки/разблокировки счета. Реализовать поиск и
// сортировку счетов. Вычисление общей суммы по счетам. Вычисление суммы
// по всем счетам, имеющим положительный и отрицательный балансы
// отдельно.

func main() {
        c := client.CreateClient()
        for {
                action := rand.Intn(7)
                switch action {
                case 1:
```

```
                        go c.OpenAccount()
                case 2:
                        go c.SortAccounts(false)
                case 3:
                        go c.SortAccounts(true)
                case 4:
                        go c.PrintPosNegSumAccount()
                case 5:
                        go c.PrintSumAccount()
                case 6:
                        go c.CloseAccount()
                }
        }
}
```