МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №4

Выполнил

Василюк П. О.,

студент группы ПО-5

Проверил

Крощенко А.А.,

ст. преп. Кафедры ИИТ,

«\_\_» \_\_\_\_\_\_\_\_ 2021 г.

Брест, 2021

**Цель работы:** приобрести практические навыки в области объектно-ориентированного проектирования.

**Вариант 2**

**Задание 1.** Создать класс Payment (покупка) с внутренним классом, с помощью объектов которого можно сформировать покупку из нескольких товаров.

**Задание 2.** Создать класс Абзац, используя класс Строка.

**Задание 3.** Система Платежи. Клиент имеет Счет в банке и Кредитную Карту (КК). Клиент может оплатить Заказ, сделать платеж на другой Счет, заблокировать КК и аннулировать Счет. Администратор может заблокировать КК за превышение кредита.

**Выполнение:**

**Код программы:**

**Task\_1-2.kt**

fun main(args: Array<String>) {

var payment:Payment = Payment()

var frstItm:Payment.Items = Payment.Items("Milk",10)

var scndItm:Payment.Items = Payment.Items("Salt",5)

var thrdItm:Payment.Items = Payment.Items("Egg",2)

payment.addItem(frstItm)

payment.addItem(scndItm)

payment.addItem(thrdItm)

println("BILL { ${frstItm.toString()}, ${scndItm.toString()}, ${thrdItm.toString()} }")

println("Summ: " + payment.sum())

// --------------------------------------------------------------------- //

val str = Strings("Hello world!")

val pgph = Paragraph(2, str)

pgph.printPgph()

}

**Payment.kt**

class Payment {

private var bill = arrayListOf<Items>()

constructor() {

bill = arrayListOf<Items>()

}

fun addItem(item: Items): ArrayList<Items> {

bill.add(item)

return bill

}

fun sum(): Int = bill.sumOf { it.getPr };

class Items {

lateinit var itm: String

var pr: Int = 0

constructor(itm:String, pr: Int){

this.itm = itm

this.pr = pr

}

override fun toString(): kotlin.String {

return "{ " +

"Item='" + itm + '\'' +

", Price=" + pr +

" }";

}

var getItm: String

get() = itm

set(value) {

itm = value

}

var getPr: Int

get() = pr

set(value) {

pr = value

}

}

}

**Paragraph.kt**

class Paragraph (private var num: Int, private val str: Strings){

fun printPgph() {

println("Paragraph $num, string - ")

str.printStr()

}

}

**Strings.kt**

class Strings (private var str: String){

private var len: Int = str.length

fun printStr() {

println("\nValue: $str, length: $len")

}

}

**Main.kt**

import kotlin.collections.ArrayList

import kotlin.random.Random.Default.nextInt

fun main(args: Array<String>) {

val client1: Payments.Client = Payments.Client()

val client2: Payments.Client = Payments.Client()

val good1 = Sum()

good1.setSum = 2000

val admin: Administrator = Administrator()

println("Count client1: " + client1.GetCount())

client1.Pay(good1)

println("Count client1: " + client1.GetCount())

println("Count client2: " + client2.GetCount())

client1.PayTo(client2.GetAccount(), 10000)

println("Count client1: " + client1.GetCount())

println("Count client2: " + client2.GetCount())

println("Close Account client2")

client2.CloseAccount()

println("Close Card client2")

client2.CloseCard()

println("Admin close Card client1")

admin.BlockClientCard(client1)

}

class Sum {

var sum = 0

var setSum: Int

get() = sum

set(value) {

sum = value

}

}

open class Payments {

open class User { // some functional

}

class Client : User() {

private val account: Account

private val card: CCard

fun Pay(good: Sum?) // using Card

{

if (good != null) {

card.Pay(good)

}

}

fun PayTo(other: Account?, sum: Int) // using Card

{

if (other != null) {

card.PayTo(other, sum)

}

}

fun CloseCard() // using Card

{

card.Close()

}

fun CloseAccount() // using Account

{

account.CloseAccount()

}

fun GetCount(): Int {

return card.Count()

}

fun GetAccount(): Account {

return account

}

init {

account = Account(5000)

card = CCard(account)

Clients.add(this)

}

}

companion object {

var Clients = ArrayList<Client>()

}

}

class Administrator : Payments.User() {

fun BlockClientCard(obj: Payments.Client) {

if (obj.GetCount() < 0) {

obj.CloseCard()

} else {

println("Card is not blocked. The count is correct.")

}

}

}

// any Card has Account

class CCard (var Account: Account) {

private var Closed = false

fun Close() {

Closed = true

println("The card was closed.")

}

fun Count(): Int // return Count from Account

{

return if (Closed) {

println("Card is locked")

0

} else {

Account.count

}

}

fun Pay(obj: Sum) // taking Good and change our Count

{

if (Closed) {

println("Card is locked")

return

} else {

Account.TakeSum(obj.sum)

println("The good was paid.")

}

}

fun PayTo(other: Account, sum: Int) {

if (Closed) {

println("Card is locked")

return

} else {

Account.TakeSum(sum)

other.AddSum(sum)

println("The sum was sent to the other client.")

}

}

}

class Account(\_count: Int) {

var number = 0

private set

var count = 0

var validation = false

private set

fun CloseAccount() {

validation = false

println("The account was closed.")

}

fun AddSum(sum: Int) // add some sum to Count

{

if (!validation) {

println("Account is not valid")

return

} else {

count += sum

}

}

fun TakeSum(sum: Int) // take some sum from Count

{

if (!validation) {

println("Account is not valid")

return

} else {

count -= sum

}

}

init {

number = (1000..9999).random() // the number is random value

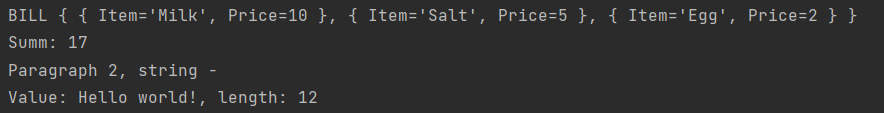
count = \_count // open on our private Sum

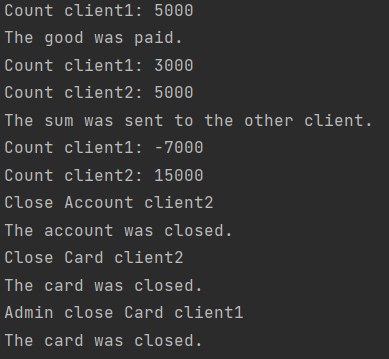
validation = true // default - Account is valid

}

}

Результаты:

****

****

**Вывод:** приобрели практические навыки в области объектно-ориентированного проектирования.