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.** interface Abiturient ← abstract class Student ← class Student Of Faculty.

**Задание 2.** Создатьсуперкласс Учащийся и подклассы Школьник и Студент. Создать массив объектов суперкласса и заполнить этот массив объектами. Показать отдельно студентов и школьников.

**Задание 3.** В задании 3 ЛР No4, где возможно, заменить объявления суперклассов объявлениями абстрактных классов или интерфейсов.

**Выполнение:**

**Код программы:**

**Task\_1.kt**

fun main(args: Array<String>) {

val student = StudentOfFaculty(3, 10)

println("Years: " + student.GetYears())

println("Experience: " + student.GetExperience())

}

interface IAbiturient {

fun GetYears(): Int

}

abstract class Student(private val years: Int) : IAbiturient {

override fun GetYears(): Int {

return years

}

}

class StudentOfFaculty(years: Int, private val experience: Int) : Student(years) {

fun GetExperience(): Int {

return experience

}

}

**Task\_2.kt**

private fun Task(obj1: Students, obj2: Students) {

if (obj1.GetKnoweledge() > obj2.GetKnoweledge()) {

println("Student1 has taken the automatic offset " + "on the subject from a friend Student2")

} else if (obj1.GetKnoweledge() < obj2.GetKnoweledge()) {

println("Student2 has taken the automatic offset " + "on the subject from a friend Student1")

} else {

println("They both have gone to the army")

}

}

private fun money(obj1: Schoolboy, obj2: Schoolboy) {

if (obj1.GetPower() > obj2.GetPower()) {

println("Schoolboy1 has taken the money of Schoolboy2")

} else if (obj1.GetPower() < obj2.GetPower()) {

println("Schoolboy2 has taken the money of Schoolboy1")

} else {

println("They both have gone to the prison")

}

}

fun main(args: Array<String>) {

val learners = ArrayList<Learner>()

val student1 = Students(50, "Kirill", "D")

val student2 = Students(70, "Daniil", "K")

val schoolboy1 = Schoolboy(400, "Stas", "School31")

val schoolboy2 = Schoolboy(300, "Vasua", "School54")

money(schoolboy1, schoolboy2)

Task(student1, student2)

println(schoolboy1.GetDocument())

println(student2.GetDocument())

learners.add(student1)

learners.add(student2)

learners.add(schoolboy1)

learners.add(schoolboy2)

for (item in learners) {

if (item.javaClass == Student::class.java) {

println(item.name + " - Student")

}

if (item.javaClass == Schoolboy::class.java) {

println(item.name + " - Schoolboy")

}

}

}

abstract class Learner {

var name: String? = null

var years = 0

private var passport: String? = null

open fun GetDocument(): String? {

return passport

}

}

class Students(var knoweledge: Int, name: String, recordbook: String) : Learner() {

var university: String? = null

private val recordbook: String

override fun GetDocument(): String? {

return recordbook

}

fun GetUniversity(): String? {

return university

}

fun GetKnoweledge(): Int {

return knoweledge

}

init {

this.name = name

this.recordbook = recordbook

}

}

class Schoolboy(var power: Int, name: String, journal: String) : Learner() {

var school: String? = null

private val \_journal: String

override fun GetDocument(): String? {

return \_journal

}

fun GetSchool(): String? {

return school

}

fun GetPower(): Int {

return power

}

init {

this.name = name

\_journal = journal

}

}

**Task\_3.kt**

fun main(args: Array<String>) {

val client1 = Payments.Client()

val client2 = 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)

admin.ShowInfo()

client2.ShowInfo()

}

class Sum {

var sum = 0

var setSum: Int

get() = sum

set(value) {

sum = value

}

}

object Payments {

var Clients = ArrayList<Client>()

abstract class User {

open fun ShowInfo() {}

}

class Client : User() {

private val account: Account

private val card: CCard

private val \_code: Int = (100..999).random()

override fun ShowInfo() {

println("$\_code root")

}

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

{

if (sum != null) {

card.Pay(sum)

}

}

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)

}

}

}

class Administrator : Payments.User() {

override fun ShowInfo() {

println("Admin root")

}

fun BlockClientCard(obj: Payments.Client) {

if (obj.GetCount() < 0) {

obj.CloseCard()

} else {

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

}

}

}

class CCard(\_account: Account) {

private 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.")

}

}

init // any Card has Account

{

Account = \_account

}

}

class Account(\_count: Int) {

var number = 0

private set

var count = 0

private 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 = count + sum

}

}

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

{

if (!validation) {

println("Account is not valid")

return

} else {

count = count - sum

}

}

init {

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

count = \_count // open on our private Sum

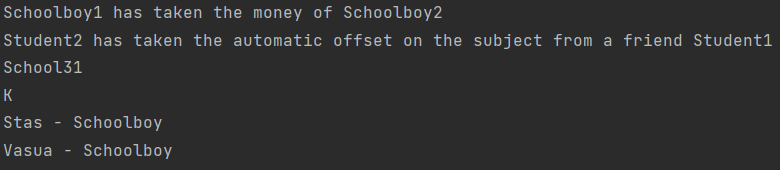
validation = true // default - Account is valid

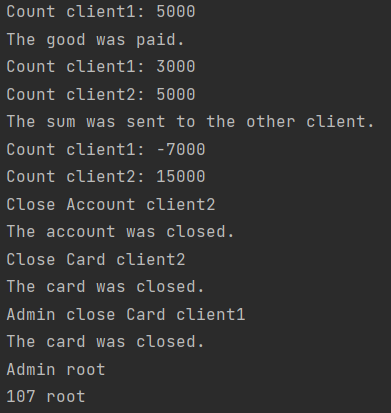
}

}

**Результаты:**

****

****



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