Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования

Кафедра проектирования информационно-компьютерных систем

Отчёт

по лабораторной работе №4

на тему:

**ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ (ООП)**

Проверил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ф.В. Усенко

(подпись)

Выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ И.И. Козел

(подпись) гр. 214302

Минск, 2024

**ЦЕЛЬ РАБОТЫ**

Изучить принципы объектно-ориентированного программирования (ООП) в *Kotlin*, включая классы, объекты, наследование, полиморфизм, интерфейсы и абстрактные классы. Научиться создавать и использовать собственные классы, а также применять принципы ООП на практике.

**ЗАДАНИЕ**

Вариант 12. Реализация системы управления заказами с учетом поставок: Разработайте систему управления заказами для интернет-магазина, включая классы Order, Product, Shipment, и Supplier. Реализуйте методы для управления процессом заказа, отслеживания поставок и автоматического перерасчета стоимости в зависимости от изменений в заказе или поставке.

**РЕШЕНИЕ**

import java.util.\*

open class Supplier(val supplierName: String)

open class Product(val productName: String, val productPrice: Double, supplierName: String) : Supplier(supplierName)

class Order(val orderID: Int) {

var orderPrice: Double = 0.0

var orderTime: Int = 0

val orderProducts: MutableList<Product> = mutableListOf()

fun addProduct(product: Product) {

orderProducts.add(product)

orderPrice += product.productPrice

orderTime += 1

}

}

class Shipment(val shipmentID: Int) {

var shipmentPrice: Double = 0.0

var shipmentTime: Int = 0

val shipmentOrders: MutableList<Order> = mutableListOf()

fun addOrder(order: Order) {

shipmentOrders.add(order)

shipmentPrice += order.orderPrice

shipmentTime += shipmentOrders.count()

}

fun removeOrder(order: Order) {

if (shipmentOrders.remove(order)) {

shipmentPrice -= order.orderPrice

shipmentTime -= shipmentOrders.count()

}

}

}

fun createSampleProducts(): List<Product> {

return listOf(

Product("Носки", 5.0, "Дзержинская швея"),

Product("Майка", 10.0, "Гродшвей"),

Product("Шорты", 15.0, "Адидас"),

Product("Байка", 20.0, "Сампл"),

Product("Штаны", 25.0, "Коминтерн")

)

}

fun main() {

val scanner = Scanner(System.`in`)

val shipments = mutableListOf<Shipment>()

val products = createSampleProducts()

var orderCounter = 0

var shipmentCounter = 0

while (true) {

println("\nМеню:")

println("1. Создать поставку")

println("2. Удалить поставку")

println("3. Создать заказ")

println("4. Удалить заказ")

println("5. Показать все поставки")

println("0. Выход")

when (scanner.nextInt()) {

1 -> {

shipmentCounter++

shipments.add(Shipment(shipmentCounter))

println("Создана поставка №$shipmentCounter")

}

2 -> {

shipments.forEach { shipment ->

println("Партия №${shipment.shipmentID}, Стоимость: ${shipment.shipmentPrice}, Дней до привоза: ${shipment.shipmentTime}")}

println("Введите № поставки для удаления:")

val id = scanner.nextInt()

val shipment = shipments.find { it.shipmentID == id }

if (shipment != null) {

shipments.remove(shipment)

println("Удалена поставка №$id")

} else {

println("Поставка №$id не найдена.")

}

}

3 -> {

println("Выберите № поставки для добавления заказа:")

shipments.forEach { println("Поставка №${it.shipmentID}") }

val shipmentId = scanner.nextInt()

val shipment = shipments.find { it.shipmentID == shipmentId }

if (shipment != null) {

val order = Order(++orderCounter)

println("Добавьте товары в заказ:")

products.forEachIndexed { index, product ->

println("${index + 1}. Наименование: ${product.productName}, Цена: ${product.productPrice}, Производитель: ${product.supplierName}")

}

println("Введите № товаров (через запятую):")

val productIndices = scanner.next().split(",").map { it.trim().toInt() - 1 }

productIndices.forEach { index ->

if (index in products.indices) {

order.addProduct(products[index])

}

}

shipment.addOrder(order)

println("Создан заказ №${order.orderID} в поставке №${shipment.shipmentID}")

} else {

println("Поставка №$shipmentId не найдена.")

}

}

4 -> {

shipments.forEach { shipment ->

println("Партия №${shipment.shipmentID}, Стоимость: ${shipment.shipmentPrice}, Дней до привоза: ${shipment.shipmentTime}")}

println("Введите № поставки для удаления заказа:")

val shipmentId = scanner.nextInt()

val shipment = shipments.find { it.shipmentID == shipmentId }

if (shipment != null) {

println("Заказы в поставке №$shipmentId:")

shipment.shipmentOrders.forEach { println("№${it.orderID}, Стоимость: ${it.orderPrice}")

for ((index, product) in it.orderProducts.withIndex()){

println("${index + 1}. Наименование: ${product.productName}, Цена: ${product.productPrice}, Производитель: ${product.supplierName}")

}

println("\n")

}

println("Введите № заказа для удаления:")

val orderId = scanner.nextInt()

val order = shipment.shipmentOrders.find { it.orderID == orderId }

if (order != null) {

shipment.removeOrder(order)

println("Удален заказ №$orderId из поставки №$shipmentId")

} else {

println("Заказ №$orderId не найден.")

}

} else {

println("Поставка №$shipmentId не найдена.")

}

}

5 -> {

shipments.forEach { shipment ->

println("Партия №${shipment.shipmentID}, Стоимость: ${shipment.shipmentPrice}, Дней до привоза: ${shipment.shipmentTime}")

shipment.shipmentOrders.forEach { order ->

println("Заказ №${order.orderID}, Стоимость: ${order.orderPrice}, Дней до доставки: ${order.orderTime}")

for ((index, product) in order.orderProducts.withIndex()){

println("${index + 1}. Наименование: ${product.productName}, Цена: ${product.productPrice}, Производитель: ${product.supplierName}")

}

}

println("\n")

}

}

0 -> return

else -> println("Неверный выбор. Пожалуйста, попробуйте снова.")

}

}

}

Результат работы программы представлен на рисунке 1.

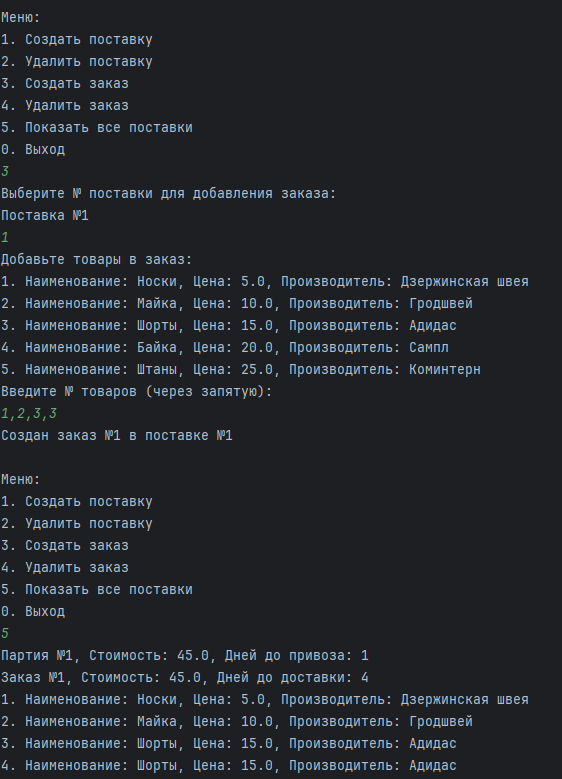


Рисунок 1 – Результат выполнения программы

**ВЫВОД**

В ходе лабораторной работы были изучены принципы объектно-ориентированного программирования (ООП) в *Kotlin*, включая классы, объекты, наследование, полиморфизм, интерфейсы и абстрактные классы. Изучено создание и использование собственных классов, а также принципов ООП на практике.