|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ **Информатика и системы управления**

КАФЕДРА **Компьютерные системы и сети (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.04.01 Информатика и вычислительная техника**

МАГИСТЕРСКАЯ ПРОГРАММА **09.04.01/07 Интеллектуальные системы анализа,**

**обработки и интерпретации больших данных**

**Отчет**

**по лабораторной работе № 6**

**Вариант № 6**

**Название:** Внутренние классы и интерфейсы

**Дисциплина:** Языки программирования для работы с большими данными

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-22М |  |  | Д.А. Залимханов |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | П.В. Степанов |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2024

**Цель:** изучить и освоить работу внутренних классов и интерфейсов в Kotlin.

**Задание 1:** cоздать класс Shop (магазин) с внутренним классом, с помощью объектов которого можно хранить информацию об отделах, товарах и услуг.

Код класса Shop:

class Shop(private val name: String) {

private val departments = mutableListOf<Department>()

fun addDepartment(department: Department) {

departments.add(department)

}

fun listDepartments() {

println("Shop: $name")

departments.forEach { department ->

println("Department: ${department.name}")

println(" Products:")

department.products.forEach { product ->

println(" - $product")

}

println(" Services:")

department.services.forEach { service ->

println(" - $service")

}

}

}

inner class Department(val name: String) {

val products = mutableListOf<String>()

val services = mutableListOf<String>()

fun addProduct(product: String) {

products.add(product)

}

fun addService(service: String) {

services.add(service)

}

}

}

Код функции Main:

fun main() {

val myShop = Shop("My Super Shop")

val electronics = myShop.Department("Electronics")

electronics.addProduct("Smartphone")

electronics.addProduct("Laptop")

electronics.addService("Warranty extension")

val groceries = myShop.Department("Groceries")

groceries.addProduct("Bread")

groceries.addProduct("Milk")

myShop.addDepartment(electronics)

myShop.addDepartment(groceries)

myShop.listDepartments()

}

Результат работы программы представлен на рисунке 1.

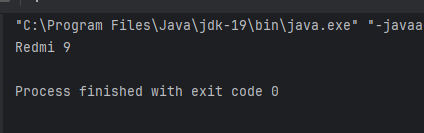


Рисунок 1 – Результат работы программы

**Задание 2:** создать класс Cправочная Cлужба Oбщественного Tранспорта с внутренним классом, с помощью объектов которого можно хранить информацию о времени, линиях маршрутов и стоимости проезда.

Код класса TransportDirectory:

class TransportDirectory {

private val routes = mutableListOf<Route>()

fun addRoute(route: Route) {

routes.add(route)

}

fun showInfo() {

routes.forEach { route ->

println("Маршрут: ${route.name}")

println(" Время: с ${route.startTime} до ${route.endTime}")

println(" Линии маршрутов: ${route.roads.joinToString()}")

println(" Стоимость проезда: ${route.cost} руб.")

}

}

inner class Route(val name: String, val startTime: String, val endTime: String, val cost: Int) {

val roads = mutableListOf<String>()

fun addRoad(road: String) {

roads.add(road)

}

}

}

Код функции Main:

fun main() {

val transportDirectory = TransportDirectory()

val route1 = transportDirectory.Route("Маршрут 1", "08:00", "22:00", 50)

route1.addRoad("Линия А")

route1.addRoad("Линия Б")

val route2 = transportDirectory.Route("Маршрут 2", "09:00", "23:00", 45)

route2.addRoad("Линия В")

route2.addRoad("Линия Г")

transportDirectory.addRoute(route1)

transportDirectory.addRoute(route2)

transportDirectory.showInfo()

}

Результат работы программы показан на рисунке 2.

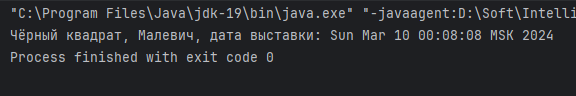


Рисунок 2 – Результат работы программы

**Задание 3:** реализовать абстрактные классы или интерфейсы, а также наследование и полиморфизм для следующих классов:

interface Корабль <- abstract class Военный Корабль <- class Авианосец.

Код интерфейса Ship:

interface Ship {

fun sail()

}

Код абстрактного класса MilitaryShip:

abstract class MilitaryShip : Ship {

abstract fun fireWeapon()

}

Код класса AircraftCarrier:

class AircraftCarrier : MilitaryShip() {

override fun sail() {

println("The aircraft carrier is sailing.")

}

override fun fireWeapon() {

println("The aircraft carrier fires its weapons.")

}

fun launchAircraft() {

println("Launching aircraft.")

}

}

Код функцииа Main:

fun main() {

val aircraftCarrier = AircraftCarrier()

aircraftCarrier.sail()

aircraftCarrier.fireWeapon()

aircraftCarrier.launchAircraft()

}

Результат работы программы представлен на рисунке 3.

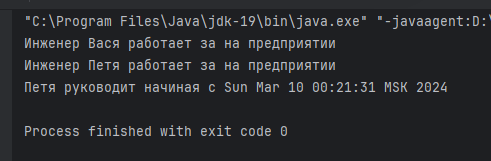
****

Рисунок 3 – Результат работы программы

**Задание 4:** реализовать абстрактные классы или интерфейсы, а также наследование и полиморфизм для следующих классов:

interface Врач <- class Хирург <- class Нейрохирург.

Код интерфейса Doctor:

interface Doctor {

fun diagnose()

}

Код абстрактного класса Surgeon:

abstract class Surgeon : Doctor {

override fun diagnose() {

println("Diagnosing as a surgeon.")

}

open fun performSurgery() {

println("Performing surgery.")

}

}

Код класса Neurosurgeon:

class Neurosurgeon : Surgeon() {

override fun performSurgery() {

println("Performing neurosurgery.")

}

fun performBrainSurgery() {

println("Performing brain surgery specifically.")

}

}

Код функции Main:

fun main() {

val neurosurgeon = Neurosurgeon()

neurosurgeon.diagnose()

neurosurgeon.performSurgery()

neurosurgeon.performBrainSurgery()

}

Результат работы программы показан на рисунке 4.

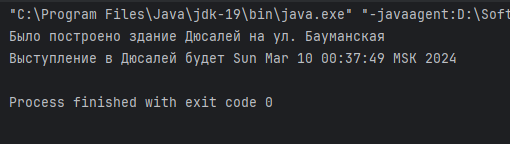


Рисунок 4 – Результат работы программы

**Вывод:** была освоена работа внутренних классов и интерфейсов в Kotlin.