Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Кафедра инженерной психологии и эргономики

Современные языки программирования

Отчет по лабораторной работе №1

«Использование языка программирования Kotlin»

Выполнил: Канунникова В. В.

Студент группы 310901

Преподаватель: Усенко Ф. В.

Минск 2024

**Цель работы**: выполнить разработку приложения с использованием языка программирования Kotlin.

**Вариант 9**: расширить взаимодействие классов. Организовать получение Работниками дать им возможность попросить увеличить зарплату. Результат просьбы увеличить зарплату должен зависеть от количества выполненных заказов.

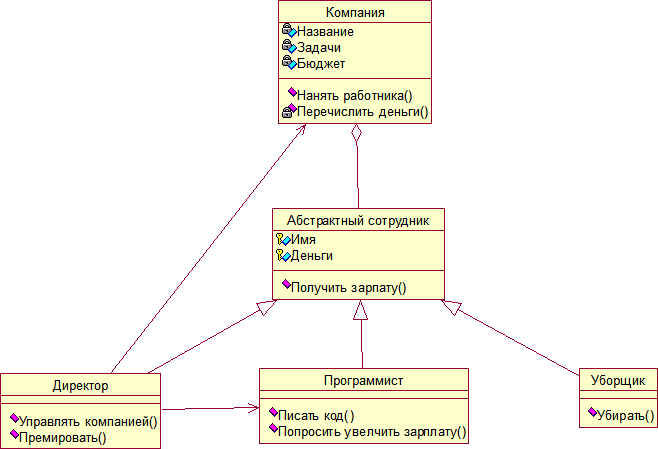


Рисунок 1 – Диаграмма классов

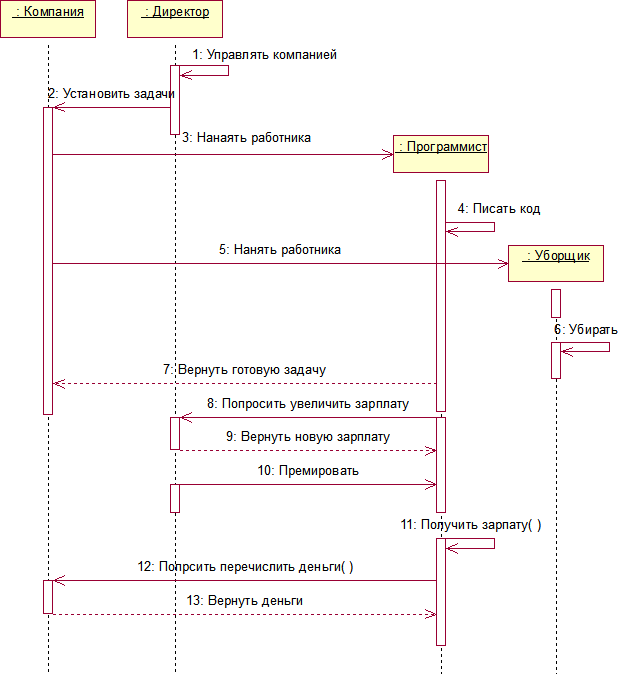


Рисунок 2 – Диаграмма последовательности

**Код программы:**

abstract class AbstractEmployee(

protected val name: String,

protected var money: Double,

private var completedOrders: Int = 0 // Количество выполненных заказов

) {

abstract fun receiveSalary() // Абстрактный метод для получения зарплаты

open fun requestRaise(): Boolean { // Метод для запроса повышения зарплаты

return false

}

protected fun completeOrder() { // Метод для увеличения количества выполненных заказов

completedOrders++

}

fun getCompletedOrders(): Int {

return completedOrders

}

fun getEmployeeName(): String {

return name

}

fun getEmployeeMoney(): Double {

return money

}

fun setEmployeeMoney(amount: Double) {

money = amount

}

}

class Director(

name: String,

money: Double

) : AbstractEmployee(name, money) {

override fun receiveSalary() { // Переопределение метода для получения зарплаты

println("${getEmployeeName()} получил зарплату: ${getEmployeeMoney()}")

}

fun manageCompany() { // Метод для управления компанией

println("${getEmployeeName()} управляет компанией")

}

fun reward() { // Метод для премирования сотрудников

println("${getEmployeeName()} премирует сотрудников")

}

}

class Programmer(

name: String,

money: Double

) : AbstractEmployee(name, money) {

override fun receiveSalary() { // Переопределение метода для получения зарплаты

println("${getEmployeeName()} получил зарплату: ${getEmployeeMoney()}")

}

fun writeCode() {

println("${getEmployeeName()} пишет код")

completeOrder() // Увеличение количества выполненных заказов

}

override fun requestRaise(): Boolean { // Переопределение метода для запроса повышения зарплаты

if (getCompletedOrders() >= 3) {

setEmployeeMoney(getEmployeeMoney() \* 1.05)

println("${getEmployeeName()} получил повышение зарплаты до ${getEmployeeMoney()}")

return true

}

println("${getEmployeeName()} не выполнил достаточно заданий для повышения зарплаты")

return false

}

}

class Cleaner(

name: String,

money: Double

) : AbstractEmployee(name, money) {

override fun receiveSalary() { // Переопределение метода для получения зарплаты

println("${getEmployeeName()} получил зарплату: ${getEmployeeMoney()}")

}

fun clean() {

println("${getEmployeeName()} занимается уборкой")

completeOrder() // Увеличение количества выполненной работы

}

override fun requestRaise(): Boolean { // Переопределение метода для запроса повышения зарплаты

if (getCompletedOrders() >= 6) {

setEmployeeMoney(getEmployeeMoney() \* 1.03)

println("${getEmployeeName()} получил повышение зарплаты до ${getEmployeeMoney()}")

return true

}

println("${getEmployeeName()} не выполнил достаточно заданий для повышения зарплаты")

return false

}

}

class Company(

private val name: String,

private val tasks: List<String>, // Список задач компании

private var budget: Double

) {

fun hireEmployee(employee: AbstractEmployee) { // Метод для найма сотрудника

println("Нанят новый сотрудник: ${employee.getEmployeeName()}")

}

fun transferMoney(employee: AbstractEmployee, amount: Double) { // Метод для перевода денег сотруднику

if (budget >= amount) {

budget -= amount

employee.setEmployeeMoney(employee.getEmployeeMoney() + amount)

println("Переведено $amount сотруднику ${employee.getEmployeeName()}. Остаток бюджета: $budget")

} else {

println("Недостаточно средств для перевода денег сотруднику ${employee.getEmployeeName()}")

}

}

fun evaluateRaiseRequest(employee: AbstractEmployee): Boolean { // Метод для оценки запроса повышения зарплаты

val result = employee.requestRaise()

if (result) {

println("${employee.getEmployeeName()} получил повышение зарплаты!")

} else {

println("${employee.getEmployeeName()} не получил повышение зарплаты.")

}

return result

}

}

fun main() {

val company = Company("Company Crazy", listOf("Task1", "Task2"), 100000.0)

val director = Director("Eva", 10000.0)

val programmer = Programmer("Dan", 5000.0)

val cleaner = Cleaner("Alex", 2000.0)

company.hireEmployee(director)

company.hireEmployee(programmer)

company.hireEmployee(cleaner)

director.manageCompany()

repeat(4) { programmer.writeCode() } // Выполняем заказы программиста

repeat(7) { cleaner.clean() } // Выполняем заказы уборщика

company.evaluateRaiseRequest(director) // Запрос повышения для директора

company.evaluateRaiseRequest(programmer) // Запрос повышения для программиста

company.evaluateRaiseRequest(cleaner) // Запрос повышения для уборщика

programmer.receiveSalary()

cleaner.receiveSalary()

director.receiveSalary()

company.transferMoney(director, director.getEmployeeMoney())

company.transferMoney(programmer, programmer.getEmployeeMoney())

company.transferMoney(cleaner, cleaner.getEmployeeMoney())

}

Результат работы программы представлен на рисунке 3.

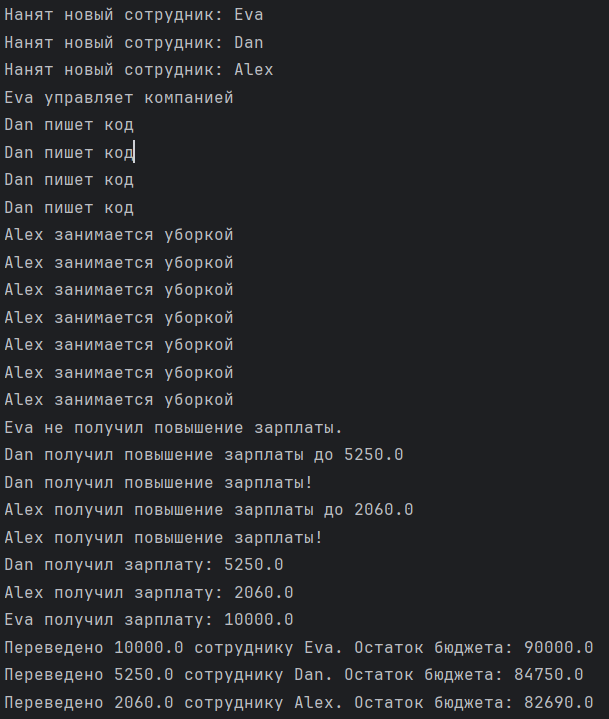


Рисунок 3 – Результат работы программы

**Ответы на вопросы к лабораторной работе:**

* 1. Что такое *JVM*?

JVM (Java Virtual Machine) – это виртуальная машина для запуска программ написанных на языках Java, Kotlin и др. Она управляет системной памятью и обеспечивает переносимую среду выполнения для приложения

1. Пакеты в *Kotlin*?

Используются для организации кода в логическую единицу и предотвращения конфликтов имен. В начале файла с помощью слова package.

1. Классы в *Kotlin*.

Объявляются с исп ключевого слова class. Класс может иметь первичный и вторичный конструкторы, свойства и методы.

1. Как объявить переменную в *Kotlin*? Различие между *var* и *val*.

Объявляются переменные с помощью var и val. Var исп для объявления изменяемых переменных. Val исп для объявления неизменяемых переменных.

1. Сколько конструкторов доступно в *Kotlin*?

Класс может иметь один первичный и один или несколько вторичных конструкторов.

1. Для чего служат абстрактные классы?

Абстрактный класс используется для определения общих характеристик, которые должны быть реализованы в подклассах.

1. Какой порядок вызова конструкторов при наследовании?

Сначала вызывается конструктор базового класса, затем конструктор производного класса.

1. Как переопределить метод родительского класса?

Передается с исп ключевого слова override.

1. Что такое оператор «*Elvis*» в *Kotlin* и как он используется?

Оператор «Elvis» (?:) исп для предоставления значения по умолчанию, если выражение слева от оператора равно null.

1. Как объявить и использовать лямбда-выражения в *Kotlin*?

Лямбда-выражения объявляются с использованием синтаксиса {параметр -> тело}. Предназначен для определения функциональных выражений.

1. Что такое модификаторы доступа (*access modifiers*) в *Kotlin*?

Они определяют видимость класса, методов и свойств. Public – доступен везде (по умолчанию), private – доступен только внутри файла или класса, protected – доступен в классе и его подклассах, internal – доступен внутри модуля.

1. Как работает оператор *when* в *Kotlin* и какие его особенности?

Он позволяет сравнить значение переменной с набором возможных значений и выполнить данный блок кода. Может быть исп как выражение, возвращающее значение, или как оператор, выполняющий блоки кода. Может объединять несколько условий в одну ветку с помощью запятой. Поддерживает проверку значений на принадлежность диапазону или коллекции с помощью операторов in и !in. When не требует break для выхода из блока кода, т.к. выполнение автоматически прекращается. Может сравнить значение с результатом произвольных выражений.

1. Как создать и использовать классы и объекты в *Kotlin*?

Классы создаются с исп ключевого слова class, может содержать свойства (переменные) и методы. Объекты с исп ключевого слова var или val и вызова конструктора класса. Класс может иметь один первичный конструктор и один или несколько вторичных конструкторов. Вторичные конструкторы объявляются с исп клбчевого слова constructur. Инициализаторы исп для выполнения кода при. Создании объекта. Они объявляются с исп ключ слова init.

1. Что такое *inline*-функции и зачем их применять?

Inline-функция исп для оптимизации кода, уменьшая накладные расходы на вызов функций.

1. Как обрабатывать исключения с помощью блока *try-catch* в *Kotlin*?

В блок try помещается код, который может вызвать исключение. Catch перехватывает и обрабатывает исключение если оно возникает.

1. Как работает оператор in для проверки наличия элемента в коллекции?

Оператор in исп для проверки наличия элемента в коллекции. Возвращает true, если элемент присутствует в коллекции и false если нет.

1. Как объявить и использовать расширения для стандартных классов в *Kotlin*?

Расширение позволяет добавлять новые функции и свойства к сущ кассам без необходимости наследования или исп паттернов. С помощью расширений (extensions).

1. Что такое функциональные типы данных (*function types*) и как их применять?

Функциональные типы данных представляют собой функции и могут быть использованы как параметры или возвращаемые значения.

**Вывод**: была изучена работа языка *Kotlin* и выполнена разработка приложения.