МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» (ДГТУ)

Факультет
Информатика и вычислительная техника

Кафедра
Программное обеспечение вычислительной техники и автоматизированных систем

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ: КЛАССЫ И ИЕРАРХИИ

Практикум

по выполнению лабораторной работы №2
по дисциплине «Языки программирования высокого уровня»

Технология разработки сложных программных систем
(указывается направленность (профиль) образовательной программы)

09.04.04 Программная инженерия
(указывается код и наименование направления подготовки)

Ростов-на-Дону
2024 г.

Составители: канд. техн. наук, зав. каф. В.В. Долгов

УДК 004.432

Базовые типы данных, управляющие конструкции и функции: метод. указания. – Ростов н/Д: Издательский центр ДГТУ, 2024.

В методическом указании рассматриваются вопросы создания консольных приложений на языке программирования Kotlin, консольный ввод/вывод, работа с базовыми типами данных и основными управляющими конструкциями языка, а также описание и вызов функций. Дано понятие, назначение и общие вопросы использования исключений как способа обработки ошибочных ситуаций, возникающих при работе программ. Приведены задания к лабораторной работе, помогающие закрепить на практике полученные знания, и контрольные вопросы для самопроверки.
Предназначено для обучающихся по направлению 09.04.04 «Программная инженерия», профиль «Технология разработки сложных программных систем».

Ответственный за выпуск:
зав. кафедрой «Программное обеспечение вычислительной техники и автоматизированных систем» Долгов В.В.

1 Теоретическая часть

В объектно-ориентированном программировании (ООП) важнейшую роль играет способность моделировать сложные системы посредством создания иерархий классов. Язык программирования Kotlin, сочетающий в себе мощь и лаконичность, предлагает ряд инструментов для эффективного использования принципов ООП, включая наследование и полиморфизм. В данной работе мы рассмотрим основы проектирования иерархии классов в Kotlin, уделяя особое внимание методам, свойствам, наследованию и полиморфизму.

Класс в Kotlin объявляется с помощью ключевого слова class. Он может содержать свойства и методы (характеристики и поведение), которые определяют состояние и поведение объектов данного класса.

```
class Person(val name: String, var age: Int) {
    fun greet() = println("Hello, my name is $name.")
}
```

В приведенном выше примере мы создали класс Person с конструктором, который принимает имя и возраст. Метод greet() позволяет объекту вывести приветствие.

Наследование позволяет создавать новый класс на основе существующего, унаследовав его свойства и методы. В Kotlin класс по умолчанию является final (не может быть унаследован). Чтобы разрешить наследование, необходимо пометить класс ключевым словом «open».

```
open class Vehicle(val make: String, val model: String) {
    open fun drive() = println("Driving $make $model.")
}
```

Здесь Vehicle – открытый класс с открытым методом drive(). Теперь мы можем создать подкласс, наследующий Vehicle:

```
class Car(make: String, model: String, val doors: Int) : Vehicle(make,
model) {
    override fun drive() = println("Driving a car: $make $model with
$doors doors.")
}
```

В этом примере класс Car наследует Vehicle и переопределяет метод drive(). Ключевое слово «override» указывает на переопределение метода базового класса.

Полиморфизм позволяет объектам разных классов, связанных посредством наследования, иметь единообразный интерфейс при разном внутреннем поведении

(одинаковые методы при различных алгоритмах их реализации). Например, как в примере ниже:

```
fun main() {
    val vehicle: Vehicle = Car("Toyota", "Corolla", 4)
    vehicle.drive() // Выведет: Driving a car: Toyota Corolla with 4
doors.
}
```

Здесь переменная vehicle имеет тип Vehicle, но хранит объект Car. При вызове drive() будет использована версия из класса Car благодаря полиморфизму.

У абстрактных классов не могут быть созданы их объекты, и они служат для определения общего интерфейса для подклассов. Абстрактный же метод не имеет реализации в базовом классе и должен быть переопределен в подклассе.

```
abstract class Shape {
    abstract fun area(): Double
}

class Circle(val radius: Double) : Shape() {
    override fun area() = Math.PI * radius * radius
}
```

В данном примере Shape – абстрактный класс с абстрактным методом area(). Circle реализует этот метод, предоставляя собственную реализацию вычисления площади.

При переопределении метода можно вызвать реализацию базового класса с помощью ключевого слова «super».

```
open class Printer {
    open fun print() = println("Printing document.")
}

class ColorPrinter : Printer() {
    override fun print() {
        super.print()
        println("Printing in color.")
    }
}
```

Здесь ColorPrinter переопределяет метод print(), но также вызывает реализацию базового класса.

Рассмотрим в качестве примера иерархии классов для геометрических фигур. Абстрактный базовый класс Shape (Фигура) может быть описан как:

```
abstract class Shape(val name: String) {
    abstract fun area(): Double
    open fun describe() = println("This is a $name.")
}
 а его подклассы, реализующие функционал конкретных фигур как:
class Rectangle(val width: Double, val height: Double) :
Shape("Rectangle") {
    override fun area() = width * height
}

class Triangle(val base: Double, val height: Double) :
Shape("Triangle") {
    override fun area() = (base * height) / 2
}

class Circle(radius: Double) : Shape("Circle"), Drawable {
    val radius = radius
    override fun area() = Math.PI * radius * radius
    override fun draw() = println("Drawing a circle with radius
$radius.")
}
```

Несмотря на то, что это различные классы, благодаря совместимости типов и полиморфизму мы можем использовать описанные классы совместно как приведено в примере ниже, где список shapes содержит объекты разных классов.

```
fun main() {
    val shapes: List<Shape> = listOf(
        Rectangle(10.0, 5.0),
        Triangle(6.0, 4.0),
        Circle(3.0)
    )

    for (shape in shapes) {
        shape.describe()
        println("Area: ${shape.area()}")
        if (shape is Drawable) {
            (shape as Drawable).draw()
        }
        println()
    }
}
```

Свойства в Kotlin также могут быть унаследованы и переопределены. В примере ниже класс Manager переопределяет свойства baseSalary и bonus.

```
open class Employee {
    open val baseSalary: Double = 30000.0
    open val bonus: Double get() = baseSalary * 0.1
```

```
}

class Manager : Employee() {
    override val baseSalary = 50000.0
    override val bonus: Double get() = baseSalary * 0.2
}
```

2 Задание к лабораторной работе

Создать в одной из сред программирования (на выбор предлагается IntelliJ IDEA Community Edition или Visual Studio Code [3]) проект консольного приложения на языке Kotlin. Используя функции ввода/вывода информации, реализовать программу в соответствии с вариантом задания (табл. 2). В процессе выполнения работы запрещено использовать стандартные функции или библиотеки языка программирования Kotlin либо среды выполнения JVM, выполняющие значимую часть функционала задания.

Код программы должен содержать обработку ошибочных ситуаций, которые могут возникнуть в ходе выполнения программы. В случае возникновения ошибок, организовать информативный вывод данных о возникшей ошибке на экран с предложением продолжить выполнение, проигнорировав ошибку, или завершить выполнение программы.

При защите работы студент должен уметь: создавать консольные проекты, устанавливать точки останова для отладки программы, выполнять программу пошагово в режиме отладки, просматривать значения переменных при отладке.

Таблица 2. Варианты заданий

№ вари-анта	Задание к лабораторной работе
1	Разработайте иерархию классов для моделирования банковской системы. Базовый класс Account должен содержать общие свойства и методы для всех типов счетов: - Свойства: accountNumber (номер счета), balance (баланс), owner (владелец). - Методы: deposit(amount: Double) (пополнение), withdraw(amount: Double) (снятие), displayBalance(). От Account наследуются следующие классы: - SavingsAccount (сберегательный счет): имеет дополнительное свойство interestRate (ставка процента) и метод addInterest(). - CheckingAccount (текущий счет): имеет свойство transactionLimit (лимит транзакций) и переопределяет методы для учета лимита.

	- BusinessAccount (бизнес-счет): имеет свойство overdraftLimit (лимит овердрафта) и методы для управления овердрафтом. Реализуйте класс Bank, который управляет коллекцией счетов, предоставляет методы для открытия новых счетов, поиска счета по номеру, выполнения транзакций. Создайте консольное приложение для взаимодействия с пользователем: открытие счетов, выполнение операций, отображение информации.
2	Создайте иерархию классов для управления элементами библиотеки. Базовый класс LibraryItem: - Свойства: id, title, publicationYear. - Методы: displayInfo(). Наследники базового класса: - Book: свойства author, genre, numberOfPages. - Magazine: свойства volume, issueNumber. - DVD: свойства director, duration. Дополнительно, реализуйте класс User с свойствами userId, name, borrowedItems. Класс Library управляет коллекциями LibraryItem и User, предоставляет методы для добавления/удаления элементов, регистрации пользователей, выдачи и возврата элементов. Создайте консольное приложение, позволяющее: - Добавлять и удалять книги, журналы, DVD. - Регистрировать пользователей. - Выдавать и принимать элементы у пользователей. - Отображать информацию об элементах и пользователях.
3	Разработайте иерархию классов для моделирования структуры университета. Базовый класс Person: - Свойства: name, surname, birthDate. - Методы: displayInfo(). Наследники: - Student: свойства studentId, coursesEnrolled. - Staff: свойства employeeId, department. От Staff наследуются: - Teacher: свойства subjectsTaught, методы assignGrade(). - Administrator: свойства responsibilities. Класс Course с свойствами courseId, courseName, teacher, studentsEnrolled. Класс University управляет коллекциями Person и Course, предоставляет методы для зачисления студентов, назначения преподавателей, формирования расписания. Консольное приложение должно позволять:

	- Добавлять студентов и сотрудников. - Создавать курсы и назначать преподавателей. - Записывать студентов на курсы. - Отображать расписание и информацию о пользователях.
4	Создайте иерархию классов для моделирования зоопарка. Базовый класс Animal: - Свойства: species, age, name. - Методы: eat(), sleep(), makeSound(). Наследники: - Mammal: дополнительные методы или свойства для млекопитающих. - Bird: свойства wingSpan, метод fly(). - Reptile: свойства isVenomous. Конкретные классы животных (например, Lion, Eagle, Snake) наследуются от соответствующих подклассов и переопределяют методы. Класс Zoo управляет коллекцией Animal, предоставляет методы для добавления, удаления животных, кормления, проведения шоу. Консольное приложение должно позволять: - Добавлять новых животных. - Проводить ежедневные процедуры (кормление, уборка). - Отображать информацию о животных.
5	Разработайте иерархию классов для транспортных средств. Базовый класс Vehicle: - Свойства: make, model, year. - Методы: start(), stop(), displayInfo(). Наследники: - Car: свойства numberOfDoors, fuelType. - Bicycle: свойства type (горный, шоссейный), hasBell. - Bus: свойства capacity, routeNumber. Класс FleetManager управляет парком транспортных средств, предоставляет методы для добавления, обслуживания, назначения на маршруты. Консольное приложение должно позволять: - Регистрировать транспортные средства. - Планировать обслуживание. - Назначать транспорт на маршруты. - Отображать состояние парка.
6	Создайте иерархию классов для моделирования сети. Базовый класс NetworkDevice: - Свойства: ipAddress, macAddress.

	- Методы: sendData(), receiveData(), displayInfo(). Наследники: - Computer: свойства operatingSystem, методы installSoftware(). - Server: свойства services, методы startService(), stopService(). - Router: свойства routingTable, методы routePacket(). Класс Network управляет устройствами, предоставляет методы для добавления устройств, моделирования передачи данных, отображения топологии. Консольное приложение должно позволять: - Добавлять устройства в сеть. - Устанавливать связи между устройствами. - Отправлять данные от одного устройства к другому. - Отображать состояние сети.
7	Разработайте иерархию классов для интернет-магазина. Базовый класс Product: - Свойства: productId, name, price, description. - Методы: displayInfo(). Наследники: - Electronics: свойства brand, warrantyPeriod. - Clothing: свойства size, material, gender. - Book: свойства author, publisher, genre. Класс ShoppingCart с методами addProduct(), removeProduct(), calculateTotal(). Класс Order со свойствами orderId, cart, orderDate, методы placeOrder(). Консольное приложение должно позволять: - Просматривать каталог товаров. - Добавлять товары в корзину. - Оформлять заказ. - Отображать историю заказов.
8	Создайте иерархию классов для игры. Класс Player: - Свойства: name, symbol (например, 'X' или 'O'). - Методы: makeMove(). Наследники: - HumanPlayer: реализует ввод хода пользователя. - ComputerPlayer: реализует алгоритм выбора хода. Класс Board: - Свойства: grid (двумерный массив). - Методы: displayBoard(), updateBoard(), checkWin(), isFull().

	Класс Game управляет процессом игры, меняет ходы игроков, определяет победителя или ничью. Консольное приложение должно позволять: - Выбрать режим игры (человек против человека, человек против компьютера). - Играть партию с отображением состояния поля после каждого хода. - Сообщать о результате партии.
9	Создайте иерархию классов для управления музыкой. Базовый класс MusicItem: - Свойства: title, duration. - Методы: play(), displayInfo(). Наследники: - Song: свойства artist, album, genre. - Podcast: свойства host, episodeNumber, topic. - Audiobook: свойства author, narrator, chapters. Классы Artist и Album с соответствующими свойствами и связями с Song. Класс Playlist управляет коллекцией MusicItem, методы addItem(), removeItem(), playAll(). Консольное приложение должно позволять: - Добавлять музыкальные элементы в библиотеку. - Создавать и управлять плейлистами. - Воспроизводить музыку. - Искать по артистам, жанрам и т.д.
10	Разработайте иерархию классов для управления рейсами. Базовый класс Flight: - Свойства: flightNumber, origin, destination, departureTime, arrivalTime. - Методы: displayInfo(). Наследники: - PassengerFlight: свойства passengerCapacity, методы для управления списком пассажиров. - CargoFlight: свойства cargoCapacity, методы для управления грузами. Класс Aircraft: - Свойства: aircraftId, model, seatingCapacity, range. Класс CrewMember: - Свойства: crewId, name, role (пилот, стюардесса). - Методы: displayInfo(). Класс Airport управляет рейсами, самолетами и экипажем, предоставляет методы для планирования рейсов, назначения самолетов и экипажа, отслеживания статус рейсов.

	Консольное приложение должно позволять: - Просматривать расписание рейсов. - Добавлять новые рейсы, самолеты, членов экипажа. - Назначать самолеты и экипаж на рейсы. - Отображать статус и детали рейсов.

3 Материально-техническое обеспечение работы

Аудитория для проведения лабораторных занятий должна быть укомплектована специализированной мебелью и индивидуальными компьютерами следующей минимальной комплектации:

- Процессор: не менее двух исполнительных ядер, совместимый с системой команд x86 и x64, с поддержкой аппаратной виртуализации.
- Оперативная память: не менее 8 Гб.
- Монитор: не менее 24" (дюймов) по диагонали.
- Наличие локальной сети со скоростью обмена не менее 1 Гб/сек.
- Наличие доступа в сеть Интернет со скоростью не менее 1 Мбит/сек.
- Наличие клавиатуры и манипулятора «мышь».

На компьютерах должно быть установлено следующее программное обеспечение:

- Операционная система: любая современная операционная система не ранее 2022 года выпуска допускающая установку JDK версии не ниже 21 и среды разработки программного обеспечения, поддерживающей язык программирования Kotlin.
- Среды разработки: IntelliJ IDEA Community Edition или Visual Studio Code.
- Среда исполнения: JDK версии не ниже 21.

4 Порядок выполнения и сдачи работы

Для выполнения лабораторной работы рекомендуется придерживаться следующего порядка выполнения:

1. Ознакомится с темой и целями лабораторной работы
2. Изучить теоретический материал
3. Подготовить рабочее окружение
4. Разработать программный код лабораторной работы в соответствие с заданием
5. Произвести проверку работоспособности, выполнить тестирование и отладку кода

6. Проанализировать полученные результаты и примененные в ходе решения подходы
7. Выполнить самооценку и рефлексию
8. Сдать лабораторную работу преподавателю и получить от него обратную связь

При сдаче студентом лабораторной работы основным отчетом выступает исходный код самостоятельно созданной в процессе выполнения работы программы.

Исходный код должен быть отформатирован согласно принятым для используемого языка программирования стандартам. Является желательным наличие в исходном коде комментариев, описывающих основные части программы и особенности их функционирования. В то же время студент должен быть готов объяснить работу программы в целом и каждую отдельную ее часть при полном отсутствии комментариев (например, они могут быть удалены преподавателем при сдаче работы).

Обязательным условием сдачи является умение студента самостоятельно восстановить любой участок исходного кода программы (но не более 20 строк подряд) после его удаления. Удаленный участок должен быть самостоятельно восстановлен обучающимся заново в присутствие преподавателя. При восстановлении удаленного участка кода запрещается использовать операции Undo (Отменить) текстовых редакторов или переписывание кода участка из других источников.

Исходный код сдаваемой программы должен быть представлен в электронном виде.

Сдача исходного кода ранее сдававшихся программ или программ, код которых выложен в сети Интернет, не допускается.

5 Контрольные вопросы к лабораторной работе

1. Объясните концепцию наследования в Kotlin. Как вы реализовали наследование в вашей иерархии классов? Приведите примеры из вашей лабораторной работы.
2. Какие модификаторы доступа существуют в Kotlin, и как они влияют на проектирование иерархии классов? Как вы использовали их в своей работе?
3. Что такое полиморфизм, и как он реализуется в Kotlin? Приведите пример из вашего проекта, где вы использовали полиморфизм для решения задачи.

4. Опишите процесс создания абстрактного класса в Kotlin. Зачем вы использовали абстрактные классы в вашей иерархии, и какие преимущества это дало?
5. Как в Kotlin определяется интерфейс, и в чем его отличие от абстрактного класса? Приведите пример использования интерфейса в вашей лабораторной работе.
6. Объясните, как вы использовали переопределение методов в вашей иерархии классов. Какие ключевые слова применяются для этого в Kotlin?
7. Как вы организовали взаимодействие между классами в вашей иерархии? Приведите пример использования композиции или агрегации в вашем проекте.
8. Как вы тестировали вашу реализацию иерархии классов на корректность работы механизмов наследования и полиморфизма? Какие инструменты или подходы вы использовали для тестирования?

6 Перечень использованных информационных ресурсов

1. Kotlin Docs | Kotlin Documentation [Электронный ресурс]. – URL: https://kotlinlang.org/docs/home.html
2. Kotlin | Изменяемые и неизменяемые коллекции [Электронный ресурс]. – URL: https://metanit.com/kotlin/tutorial/7.1.php
3. IntelliJ IDEA – the Leading Java and Kotlin IDE [Электронный ресурс]. – URL: https://www.jetbrains.com/idea/
4. Visual Studio Code - Code Editing. Redefined [Электронный ресурс]. – URL: https://code.visualstudio.com/

Редактор А.А. Литвинова

ЛР № 04779 от 18.05.01.	В набор	В печать
Объем 0,5 усл.п.л., уч.-изд.л.	Офсет.	Формат 60x84/16.
Бумага тип №3.	Заказ №	Тираж 75. Цена

Издательский центр ДГТУ

Адрес университета и полиграфического предприятия:

344010, г. Ростов-на-Дону, пл. Гагарина, 1.