Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерного проектирования

Кафедра инженерной психологии и эргономики

Дисциплина: Программирование мобильных информационных систем

Отчёт
по лабораторной работе №4
на тему
Объектно-ориентированное программирование (ООП)

Выполнил:
ст. гр. 214302
Самойлов Р.И.

Проверил:
Усенко Ф.В.

Минск 2024

1. Задание: **Реализация интерфейсов с множественным наследованием:** Напишите интерфейсы для различных типов транспорта (Drivable, Flyable, Sailable). Создайте классы, которые реализуют несколько интерфейсов (например, AmphibiousCar, FlyingBoat) и добавляют дополнительные методы для переключения режимов и управления.

Листинг кода:

```
package org.example.classes

import org.example.interfaces.Flyable
import org.example.interfaces.Sailable

class FlyingBoat : Flyable, Sailable {
    private var mode: String = "Sailing" // Режим по умолчанию — плавание

    fun switchModeToFly() {
        mode = "Flying"
        println("Переключено на режим полета.")
    }

    fun switchModeToSail() {
        mode = "Sailing"
        println("Переключено на режим плавания.")
    }

    override fun fly() {
        if (mode == "Flying") {
            println("Летающая лодка летит по воздуху.")
        } else {
            println("Невозможно лететь в режиме плавания! Сначала переключитесь
на режим полета.")
        }
    }

    override fun sail() {
        if (mode == "Sailing") {
            println("Летающая лодка плывет по воде.")
        } else {
            println("Невозможно плыть в режиме полета! Сначала переключитесь на
режим плавания.")
        }
    }
}
package org.example.classes

import org.example.interfaces.Drivable
import org.example.interfaces.Sailable

class AmphibiousCar : Drivable, Sailable {
    private var mode: String = "Driving" // Режим по умолчанию — езда
```

```
    fun switchModeToDrive() {
        mode = "Driving"
        println("Переключено на режим езды.")
    }

    fun switchModeToSail() {
        mode = "Sailing"
        println("Переключено на режим плавания.")
    }

    override fun drive() {
        if (mode == "Driving") {
            println("Амфибийная машина едет по дороге.")
        } else {
            println("Невозможно ехать в режиме плавания! Сначала переключитесь
на режим езды.")
        }
    }

    override fun sail() {
        if (mode == "Sailing") {
            println("Амфибийная машина плывет по воде.")
        } else {
            println("Невозможно плыть в режиме езды! Сначала переключитесь на
режим плавания.")
        }
    }
}

  package org.example.interfaces

interface Flyable {
    fun fly()
}
```

Контрольные вопросы:

Что такое структура, класс, интерфейс

Структура – **data class** (классы данных), которые часто используются для представления структур. Это типы, предназначенные для хранения данных с минимальными функциями.

Класс в Kotlin — это шаблон для создания объектов, который может содержать свойства (переменные) и методы (функции). Классы могут быть использованы для создания объектов и описания их поведения.

Классы могут иметь конструкторы, свойства и методы.

Они поддерживают наследование (с использованием open), полиморфизм, инкапсуляцию и другие принципы ООП.

Интерфейс в Kotlin — это абстрактный тип, который может содержать **методы без реализации** (или с реализацией, начиная с Kotlin 1.2) и **свойства**.

Интерфейсы используются для описания поведения, которое могут реализовать различные классы.

Отличие интерфейса от абстрактного класса

1. Методы:

Абстрактный класс может содержать как абстрактные методы (без реализации), так и методы с реализацией.

Интерфейс может содержать только абстрактные методы, если не указано иное (начиная с Kotlin 1.2, интерфейсы могут содержать методы с реализацией по умолчанию).

2. Наследование:

Абстрактный класс может быть наследован только от одного класса (единственный класс может быть базовым).

Интерфейс может быть реализован множеством классов, а также класс может реализовывать несколько интерфейсов, что позволяет реализовывать множественное наследование через интерфейсы.

3. Конструкторы:

Абстрактный класс может иметь конструкторы (как первичный, так и вторичный).

Интерфейс не может иметь конструкторов. Однако класс, реализующий интерфейс, может иметь свои конструкторы.

4. Состояние:

Абстрактный класс может содержать состояние (переменные).

Интерфейс не может хранить состояние, то есть он не может иметь полей, только свойства (с реализацией или без).

5. Использование:

Абстрактный класс обычно используется для представления общей функциональности для группы связанных классов.

Интерфейс используется для задания поведения, которое может быть реализовано разными, не связанными классами.

Компаньон-объект что это и отличие от статик в java

Компаньон-объект — это специальный объект внутри класса, который ведет себя как статический метод или поле в Java. Компаньон-объекты позволяют обращаться к методам и свойствам класса без создания экземпляра.

Создание единственного экземпляра с помощью компаньон-объекта:

Компаньон-объект часто используется для реализации паттерна Синглтон. В этом случае в классе создается компаньон-объект, который

инициализирует единственный экземпляр класса. Это гарантирует, что объект будет создан только один раз и доступен во всей программе.

Когда нужно создать только один экземпляр класса (например, для доступа к базе данных или конфигурации приложения), можно использовать компаньон-объект для контроля этого процесса.

Принципы ооп

Инкапсуляция

Инкапсуляция заключается в скрытии внутреннего состояния объекта и предоставлении доступа к этому состоянию только через методы. Это позволяет защищать данные от непреднамеренных изменений и поддерживать контроль над поведением объекта.

В Kotlin инкапсуляция реализуется с помощью модификаторов доступа (private, protected, internal, public), а также геттеров и сеттеров.

Наследование

Наследование позволяет создавать новый класс на основе уже существующего, повторно используя его функциональность и расширяя или изменяя её.

В Kotlin для наследования классов нужно использовать ключевое слово open для базового класса, чтобы он мог быть унаследован. Класс-потомок использует : для указания базового класса и может переопределять методы с помощью ключевого слова override.

Полиморфизм

Полиморфизм позволяет объектам одного типа быть использованными как объекты другого типа, если они относятся к одному и тому же классу или интерфейсу. Это позволяет создавать более гибкие и универсальные программы.

Полиморфизм в Kotlin реализуется через переопределение методов и использование абстрактных классов и интерфейсов.

Абстракция

Абстракция позволяет скрыть сложность системы и работать только с необходимыми деталями. В Kotlin абстракция реализуется через абстрактные классы и интерфейсы, которые могут содержать абстрактные методы (без реализации) и методы с реализацией.

Абстрактные классы не могут быть инстанцированы напрямую, но могут содержать частично реализованные методы. Интерфейсы только задают контракт для реализации в классах.

Теневые поля и свойства

Теневые свойства возникают, когда подкласс переопределяет свойство родительского класса или интерфейса, скрывая его.

Пример: свойство sound в классе Dog скрывает свойство sound из класса Animal.

Теневые поля скрывают поля родительского класса. Когда свойство переопределяется, поле родителя становится недоступным, и используется новое поле подкласса.

Пример: name в классе Dog скрывает поле name в классе Animal.

Использование field позволяет получить доступ к скрытому полю в методах getter и setter, даже если оно переопределено в подклассе.

Отличие объекта от экземпляра

В Kotlin ключевое слово object создаёт **синглтон** — это уникальный объект, существующий в единственном экземпляре. Объект, созданный через object, можно использовать для хранения состояния или методов, которые нужны на уровне класса, а не на уровне его отдельных экземпляров.

Объект (синглтон) создаётся один раз и используется для хранения общего состояния или функциональности.

Экземпляр — это объект, созданный на основе класса с помощью ключевого слова class и оператора new. Каждый раз при создании экземпляра класса создаётся новый объект, который имеет своё собственное состояние и поведение (если оно задано).

Экземпляр — это отдельный объект на основе класса, с уникальным состоянием, и таких экземпляров может быть несколько.

Конструкторы первичные, вторичные

Первичный - Это конструктор, объявляемый в заголовке класса и инициализирующий его свойства.

class Car(val brand: String, val model: String, var year: Int)

Вторичный конструктор используется для создания альтернативных способов инициализации объекта. Он объявляется с помощью ключевого слова constructor.

Вторичный конструктор полезен, когда нужно задать значения по умолчанию или поддерживать различные способы создания объекта.

Можно ли несколько конструкторов создать?

Да, с помощью вторичных

Как переопределить toString()

В Kotlin метод toString() можно переопределить в классе для предоставления настраиваемого строкового представления объекта. По умолчанию, метод toString() возвращает строку, содержащую имя класса и хеш-код объекта, но его можно переопределить для более информативного вывода.

Переопределение метода toString:

Для переопределения метода toString(), нужно использовать ключевое слово override и предоставить свою реализацию, которая будет возвращать строку, отражающую состояние объекта.

```
class Person(val name: String, val age: Int) {
    // Переопределение метода toString
    override fun toString(): String {
        return "Person(name='$name', age=$age)"
    }
}

fun main() {
    val person = Person("John", 30)
    println(person)  // Выводит: Person(name='John', age=30)
}
```

Множественное наследование

В Kotlin множественное наследование возможно только через интерфейсы, так как классы могут наследовать только один базовый класс (Kotlin поддерживает одиночное наследование).

Класс может реализовывать несколько интерфейсов, что позволяет комбинировать их поведение. Каждый интерфейс может содержать абстрактные методы и методы с реализацией.