Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования

"Омский государственный технический университет" Кафедра " Автоматизированные системы обработки информации и управления"

# Лабораторная работа

## По дисциплине

"Современные инструментальные средства разработки программного обеспечения"

Выполнил: студент группы ПИН-201

Скрипченко Н. А.

Проверил: старший преподаватель

Кабанов А. А.

# 1. Какие технологии стоит использовать сейчас при разработке большого Android приложения?

## ChatGPT

Разработка большого Android-приложения может быть сложным и многогранным заданием, и выбор технологий зависит от конкретных требований вашего проекта. Однако ниже приведены некоторые ключевые технологии и инструменты, которые обычно используются при разработке больших Android-приложений:

- Язык программирования Kotlin: Kotlin является официальным языком программирования для разработки Android-приложений и предоставляет множество преимуществ, таких как безопасность типов, выразительность и расширяемость кода.
- Архитектурные паттерны: Использование архитектурных паттернов, таких как MVVM (Model-View-ViewModel) или Clean Architecture, помогает управлять сложностью приложения, разделять код на компоненты и обеспечивать легкость тестирования.
- Android Jetpack: Jetpack это набор библиотек и инструментов от Google, предназначенных для упрощения разработки Android-приложений. Включает в себя библиотеки для работы с жизненным циклом, навигацией, базами данных (Room), и другими аспектами разработки.
- 4. **Retrofit или OkHttp**: Эти библиотеки используются для работы с сетью. Retrofit предоставляет удобный способ взаимодействия с веб-службами, а OkHttp предоставляет мощный HTTP-клиент.

Рисунок 1 - Первый вопрос

- 5. **Dependency Injection (DI)**: DI-контейнеры, такие как Dagger или Hilt, помогают управлять зависимостями в приложении и делают код более тестируемым и поддерживаемым.
- 6. **Архитектурные библиотеки**: Используйте библиотеки, такие как LiveData и ViewModel из Jetpack, чтобы облегчить разработку пользовательского интерфейса и управление данными.
- 7. Разработка с учетом производительности: Для оптимизации производительности приложения используйте инструменты Android Profiler и применяйте лучшие практики, такие как асинхронное выполнение задач и кэширование данных.
- 8. **Тестирование**: Используйте фреймворки для тестирования, такие как JUnit и Espresso, для написания модульных, функциональных и UI-тестов.
- 9. **Инструменты сборки и CI/CD**: Используйте инструменты сборки, такие как Gradle, и настройте непрерывную интеграцию и доставку (CI/CD) для автоматической сборки, тестирования и развертывания приложения.
- Дизайн и UX: Уделите внимание дизайну материалов и пользовательскому опыту. Используйте библиотеки дизайна, такие как Material Design Components, для создания современного и интуитивно понятного интерфейса.
- 11. **Безопасность**: Обеспечьте безопасность приложения, следя за защитой данных и использованием правильных методов аутентификации и авторизации.

## Продолжение рисунка 1 - Первый вопрос

# 2. Какие есть архитектуры мобильных приложений?

## \$

#### ChatGPT

Существует несколько архитектурных подходов к разработке мобильных приложений, и выбор зависит от конкретных требований проекта и предпочтений разработчика. Ниже приведены некоторые из наиболее распространенных архитектурных паттернов для мобильных приложений:

#### 1. MVC (Model-View-Controller):

- Модель (Model): Содержит данные и бизнес-логику.
- Представление (View): Отвечает за отображение данных пользователю.
- **Контроллер (Controller)**: Управляет взаимодействием между моделью и представлением.

Этот паттерн удобен для небольших приложений, но может стать неэффективным и сложным для поддержки в крупных проектах.

#### 2. MVP (Model-View-Presenter):

- Модель (Model): Содержит данные и бизнес-логику.
- Представление (View): Отвечает за отображение данных пользователю.
- Презентер (Presenter): Управляет взаимодействием между моделью и представлением, а также принимает решения на основе пользовательского ввода.

MVP улучшает разделение ответственности по сравнению с MVC и облегчает тестирование.

# Рисунок 2 - Второй вопрос

#### 3. MVVM (Model-View-ViewModel):

- Модель (Model): Содержит данные и бизнес-логику.
- Представление (View): Отображает данные и реагирует на действия пользователя
- **ViewModel**: Подготавливает данные для отображения и управляет состоянием представления.

MVVM позволяет легко обновлять пользовательский интерфейс при изменении данных и обеспечивает лучшую разделенную ответственность и тестирование.

#### 4. Clean Architecture:

- Сущности (Entities): Представляют основные бизнес-объекты и правила.
- ИспользуCases (Use Cases): Содержат бизнес-логику и определяют действия, которые можно выполнить над сущностями.
- Адаптеры (Adapters): Отвечают за взаимодействие с внешними системами, такими как базы данных или API.
- **Фреймворк (Framework)**: Содержит код для пользовательского интерфейса и фреймворков.

Clean Architecture обеспечивает высокую степень изоляции и гибкость, делая приложение легко тестируемым и изменяемым.

# Продолжение рисунка 2 - Второй вопрос

3. Какие отличия между архитектурами MVI, MVP, MVVM и какую из них лучше стоит взять при разработке большого Android приложения?

Архитектуры MVI (Model-View-Intent), MVP (Model-View-Presenter), и MVVM (Model-View-ViewModel) представляют собой разные подходы к построению архитектуры Android-приложений. Давайте рассмотрим основные отличия между ними и обсудим, какую из них выбрать при разработке большого Android-приложения:

#### 1. MVI (Model-View-Intent):

- Model: Содержит бизнес-логику и данные.
- View: Отображает данные и отправляет интенты (намерения) о действиях пользователей.
- Intent: Представляет собой события или намерения пользователей, которые приводят к изменению состояния Model.

#### Отличия

- MVI акцентирует внимание на явном представлении намерений пользователей и управлении ими.
- Этот паттерн подходит для приложений, где важно четкое отслеживание и обработка пользовательских событий.

#### 2. MVP (Model-View-Presenter):

- Model: Содержит бизнес-логику и данные.
- View: Отображает данные и обрабатывает пользовательский ввод.
- Presenter: Управляет бизнес-логикой ч связью между Model и View.

## Рисунок 3 - Третий вопрос

#### Отличия:

- MVP акцентирует внимание на четком разделении ответственности между View и Presenter.
- Этот паттерн подходит для приложений, где важно обеспечить тестирование и поддерживаемость кода.

#### 3. MVVM (Model-View-ViewModel):

- Model: Содержит бизнес-логику и данные.
- View: Отображает данные и реагирует на изменения ViewModel.
- ViewModel: Подготавливает данные для отображения и управляет состоянием View.

## Отличия:

- MVVM акцентирует внимание на управлении состоянием и обновлении View при изменении данных.
- Этот паттерн часто использует привязку данных (data binding) для автоматического обновления интерфейса при изменении ViewModel.

Выбор между этими архитектурами зависит от конкретных требований и предпочтений. Для большого Android-приложения лучший выбор будет зависеть от сложности проекта и опыта вашей команды разработчиков. Вот некоторые рекомендации:

## Продолжение рисунка 3 - Третий вопрос

- Если важно четкое управление намерениями пользователя и активное взаимодействие с ними, то MVI может быть хорошим выбором.
- Если вы ищете более традиционный подход с четким разделением между бизнес-логикой и пользовательским интерфейсом, то MVP подойдет.
- Если вам нужно легко обновлять интерфейс при изменении данных и вы хотите использовать привязку данных, то MVVM может быть предпочтительным.

Важно также помнить, что смешивание элементов разных архитектурных подходов (например, использование ViewModel с MVP) часто применяется в реальных проектах, чтобы достичь оптимального баланса между управляемостью кода и производительностью.

## Продолжение рисунка 3 - Третий вопрос

# 4. Что сейчас лучше использовать Coroutines или RxJava?

## \$

## ChatGPT

Выбор между Coroutines и RxJava зависит от конкретных требований вашего проекта, опыта команды разработчиков и предпочтений. Оба этих инструмента предоставляют мощные средства для асинхронного программирования и управления потоками данных, но у них есть свои отличия:

#### **Kotlin Coroutines:**

- 1. **Легкость в использовании**: Coroutines встроены в язык Kotlin и предоставляют более естественный синтаксис для асинхронного кода. Они позволяют писать асинхронный код, похожий на синхронный.
- 2. **Меньше бойлерплейта**: Код на Coroutines часто бывает более компактным и менее подвержен ошибкам, чем эквивалентный код на RxJava.
- 3. **Корутины являются официальной рекомендацией Google**: Google активно поддерживает и рекомендует использование Coroutines для асинхронной разработки на платформе Android.

Рисунок 4 - Четвертый вопрос

#### RxJava:

- 1. **Мощные операторы**: RxJava предоставляет обширный набор операторов для манипуляции данными и управления потоками. Эти операторы могут быть очень полезными при сложных операциях с данными.
- 2. **Множество платформ**: RxJava не ограничивается только Android и Kotlin. Он доступен на разных платформах и в различных языках программирования, что делает его переносимым инструментом.
- 3. **Большой опыт и сообщество**: RxJava существует уже довольно долго и имеет большое сообщество пользователей. Есть много ресурсов и библиотек, основанных на RxJava.

# Продолжение рисунка 4 - Четвертый вопрос

Теперь касательно выбора:

- 1. Если вы разрабатываете новое приложение с использованием Kotlin, особенно если вам нужно выполнять асинхронные операции и работать с сетью или базой данных, то Coroutines являются хорошим и более естественным выбором.
- 2. Если вы уже используете RxJava в вашем проекте, и он хорошо себя зарекомендовал, и вы удовлетворены его производительностью и возможностями, то нет необходимости переходить на Coroutines.
- 3. В некоторых случаях можно даже комбинировать Coroutines и RxJava в одном проекте, используя их совместно, чтобы воспользоваться преимуществами обоих подходов.

Важно помнить, что выбор между Coroutines и RxJava зависит от конкретных потребностей вашего проекта и уровня комфорта и опыта вашей команды с тем или другим инструментом.

# Продолжение рисунка 4 - Четвертый вопрос

5. Какие библиотеки DI в android существуют и какую лучше сейчас использовать при разработке Android приложения?

Существует несколько библиотек для внедрения зависимостей (DI, Dependency Injection) в Android-приложения. Каждая из них имеет свои особенности и преимущества. Вот некоторые из наиболее популярных библиотек DI для Android и их краткое описание:

#### 1. Dagger 2 (Hilt):

- Dagger 2 это одна из самых популярных и мощных библиотек DI для
   Android. Она основана на аннотациях и генерации кода на этапе компиляции.
- **Hilt** это официальное решение от Google, которое упрощает интеграцию Dagger 2 с Android-приложениями.

#### 2. Koin:

• Koin - это легковесная библиотека DI для Kotlin, которая не требует генерации кода и является более простой в использовании по сравнению с Dagger 2.

#### 3. Kodein:

• **Kodein** - еще одна библиотека DI для Kotlin. Она также предоставляет простой и декларативный синтаксис для внедрения зависимостей.

#### 4. Toothpick:

• **Toothpick** - это библиотека DI, которая предлагает аннотации и рефлексию для внедрения зависимостей. Она проста в использовании и настройке.

## Рисунок 5 - Пятый вопрос

#### 5. Kode:

• **Kode** - это еще одна легковесная библиотека DI для Kotlin. Она уделяет внимание простоте и читаемости кода.

#### 6. Dagger Android:

• Dagger Android - это часть Dagger 2, специально созданная для интеграции с Android-приложениями. Она облегчает внедрение зависимостей в Android-компоненты, такие как Activity и Fragment.

#### 7. Kode-Injector:

• **Kode-Injector** - это еще одна Kotlin-ориентированная библиотека DI, которая упрощает внедрение зависимостей.

Какую библиотеку DI выбрать, зависит от ваших предпочтений и потребностей вашего проекта. Dagger 2 с Hilt является мощным и распространенным решением, особенно если вы разрабатываете большое Android-приложение. Однако, если вам нужна более простая и быстрая настройка, Koin или Kode могут быть хорошими альтернативами.

Также важно учитывать, какая библиотека лучше интегрируется с вашим текущим стеком технологий и какую поддержку и сообщество имеет выбранная библиотека, чтобы обеспечить долгосрочную поддержку вашего проекта.

# Продолжение рисунка 5 - Пятый вопрос

# 6. Какой язык программирования Kotlin или Java сейчас стоит использовать при разработке Android приложения и почему?

## **\$**

#### ChatGPT

На момент моего последнего обновления в январе 2022 года Kotlin является предпочтительным языком программирования для разработки Android-приложений, и Google официально поддерживает Kotlin в качестве первого класса для Android-разработки. Вот несколько причин, почему Kotlin стоит использовать при разработке Android-приложений:

- Официальная поддержка Google: Google объявил Kotlin официальным языком разработки для Android в 2017 году. Это означает, что Google активно развивает инструменты и библиотеки, которые улучшают работу с Kotlin на платформе Android.
- 2. Совместимость с Java: Kotlin полностью совместим с Java, что означает, что вы можете легко интегрировать код на Kotlin в существующие Java-приложения и наоборот. Вы можете постепенно переходить к Kotlin, не переписывая приложение с нуля.
- 3. Улучшенная безопасность типов: Kotlin предоставляет более строгую систему типов, которая позволяет выявлять ошибки на этапе компиляции, что уменьшает вероятность ошибок времени выполнения.
- 4. Улучшенное синтаксическое сахарное: Kotlin предлагает чистый и более выразительный синтаксис, что делает код более читаемым и поддерживаемым по сравнению с Java.

# Рисунок 6 - Шестой вопрос

- 5. Поддержка асинхронного программирования: Kotlin предоставляет корутины (coroutines), которые делают асинхронное программирование более простым и читаемым, особенно при работе с сетью и многозадачными операциями.
- 6. **Меньше бойлерплейта**: Kotlin позволяет уменьшить количество бойлерплейта в коде, что улучшает производительность разработки.
- Активное сообщество и библиотеки: Kotlin имеет широкое и активное сообщество разработчиков, что означает наличие большого количества библиотек и ресурсов, которые облегчают разработку Android-приложений.

Важно отметить, что, хотя Kotlin является предпочтительным языком для Android-разработки, Java по-прежнему поддерживается и может использоваться при необходимости. Однако, если у вас нет особых ограничений или причин использовать Java, рекомендуется рассмотреть Kotlin как более современный и продвинутый вариант для разработки Android-приложений.

# Продолжение рисунка 6 - Шестой вопрос

7. Какую минимальную версию Android стоит использовать в разработке?

Извините за недоразумение, но я не имею доступа к актуальной статистике использования версий Android, так как моя база данных обновляется только до января 2022 года, и я не могу предоставить текущий график использования версий Android.

Определение минимальной версии Android для вашего приложения зависит от нескольких факторов:

- Целевая аудитория: Определите, какие версии Android используют ваши целевые пользователи. Обычно минимальный уровень API (minSdkVersion) устанавливается так, чтобы охватить большинство вашей целевой аудитории.
- Функциональность: Если ваше приложение требует функциональности, доступной только в более новых версиях Android, вы можете установить более высокий minSdkVersion. Тем не менее, имейте в виду, что это может исключить более старые устройства.
- 3. Разработчики библиотек: Учтите, какие версии библиотек и фреймворков вы собираетесь использовать в вашем приложении. Убедитесь, что они совместимы с выбранной версией Android.
- 4. **Обновление**: В зависимости от роста аудитории и изменения распространения версий Android, вы можете пересмотреть и обновить minSdkVersion, чтобы охватить большую часть пользователей **1**

## Рисунок 7 - Седьмой вопрос

Обычно разработчики стараются поддерживать Android 5.0 (API Level 21) и выше как минимальный уровень API, чтобы охватить большинство современных устройств. Однако конкретный выбор зависит от целей вашего проекта и вашей целевой аудитории.

Чтобы получить актуальную статистику использования версий Android, вы можете посетить официальный сайт Android Developers, где регулярно публикуются данные о распространении версий Android. Это поможет вам принять информированное решение о минимальной версии Android для вашего приложения.

# Продолжение рисунка 7 - Седьмой вопрос

8. Принципы SOLID: приведи примеры на языке Kotlin и их значение

Конечно, давайте рассмотрим принципы SOLID с примерами на языке программирования Kotlin и опишем их значения:

1. Принцип единственной ответственности (Single Responsibility Principle, SRP): Значение: Класс должен иметь только одну причину для изменения, т.е., он должен иметь только одну ответственность.

#### Пример:

```
kotlin

// Пример нарушения SRP:
class OrderManager {
   fun createOrder() { /* Создать заказ */ }
   fun shipOrder() { /* Отправить заказ */ }
   fun generateInvoice() { /* Сгенерировать счет */ }
}

// Лучше разделить на отдельные классы с одной ответственностью:
class OrderCreator { /* Создать заказ */ }
class OrderShipper { /* Отправить заказ */ }
class InvoiceGenerator { /* Сгенерировать счет */ }
```

## Рисунок 8 - Принцип единственной ответственности

2. Принцип открытости/закрытости (Open/Closed Principle, OCP):

**Значение**: Система должна быть открыта для расширения, но закрыта для изменения. Это означает, что новую функциональность можно добавлять без изменения существующего кода.

## Пример:

```
kotlin

// Пример нарушения ОСР:
class OrderProcessor {
    fun processOrder(order: Order) {
        if (order.paymentMethod == PaymentMethod.CREDIT_CARD) {
            // Обработать заказ с кредитной картой
        } else if (order.paymentMethod == PaymentMethod.PAYPAL) {
            // Обработать заказ с РауРа1
        }
    }
}

// Лучше использовать паттерн стратегии для расширения функциональности interface PaymentProcessor {
    fun processPayment(order: Order)
}
```

Рисунок 9 - Принцип открытости закрытости

3. Принцип подстановки Барбары Лисков (Liskov Substitution Principle, LSP): Значение: Объекты подклассов должны быть способны заменять объекты базовых классов без изменения правильности программы.

#### Пример

```
kotlin

// Пример нарушения LSP:
open class Bird {
    open fun fly() { /* Лететь */ }
}

class Ostrich : Bird() {
    override fun fly() {
        throw UnsupportedOperationException("Страус не умеет летать")
    }
}

// Замена объекта Bird на Ostrich нарушает контракт:
val bird: Bird = Ostrich()
bird.fly() // Будет выброшено исключение
```

Рисунок 10 - Принцип Барбары Лисков

4. Принцип разделения интерфейса (Interface Segregation Principle, ISP): Значение: Клиенты не должны зависеть от интерфейсов, которые они не используют. Интерфейсы должны быть специализированными и иметь только необходимые методы.

#### Пример:

```
kotlin

// Пример нарушения ISP:
interface Worker {
   fun work()
   fun eat()
}

// Клиенты, которые нуждаются только в работе, вынуждены реализовывать class Engineer: Worker {
   override fun work() { /* Работать */ }
   override fun eat() { /* Поесть */ }
}

// Лучше разделить интерфейсы на более мелкие:
interface Workable {
   fun work()
}
```

Рисунок 11 - Принцип разделения интерфейсов

5. Принцип инверсии зависимостей (Dependency Inversion Principle, DIP):

**Значение**: Модули верхнего уровня не должны зависеть от модулей нижнего уровня. Оба должны зависеть от абстракций.

#### Пример:

```
kotlin

// Пример нарушения DIP:
class LightBulb {
    fun turnOn() { /* Включить лампочку */ }
    fun turnOff() { /* Выключить лампочку */ }
}

class Switch {
    private val bulb = LightBulb()

    fun operate() {
        bulb.turnOn()
        // Switch зависит от конкретной реализации LightBulb
    }
}

// Лучше инвертировать зависимость:
interface Switchable {
    fun turnOn()
    fun turnOff()
}
```

Рисунок 12 - Принцип инверсии зависимостей

9. Какой процент населения в России носит часы? Приведи график разделения на мужчину, женщины и дети

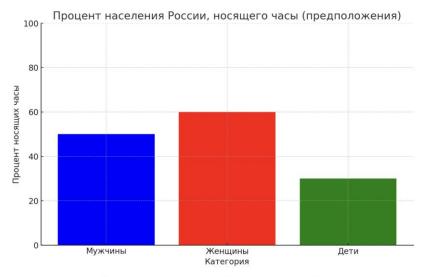
## ChatGPT

На текущий момент у меня нет доступа к интернету для поиска свежей статистики, и специфические данные о проценте населения России, носящего часы, разделённые по гендеру и возрасту, могут быть не легко доступны. Такая информация обычно требует опросов или исследований, проведённых соответствующими организациями.

Тем не менее, я могу помочь вам создать гипотетический график, основанный на предполагаемых данных. Давайте предположим следующее распределение:

- Мужчины взрослые: 50% носят часы.
- Женщины взрослые: 60% носят часы.
- Дети: 30% носят часы.

Рисунок 13 - Девятый вопрос



Вот гипотетический график, показывающий предполагаемый процент населения России, носящего часы, разделённый по категориям: мужчины, женщины и дети. Согласно этим данным:

Около 50% мужчин носят часы.

Около 60% женщин носят часы.

Около 30% детей носят часы.

# Продолжение рисунка 13 - Девятый вопрос

# 10. Почему тема "магазин часов" актуально в России, обоснуй ее?



#### ChatGPT

Тема "магазин часов" может быть актуальной в России по нескольким причинам, отражающим культурные, экономические и социальные аспекты страны:

## 1. Культурная ценность и статус:

Часы традиционно считаются не только утилитарным предметом для отслеживания времени, но и важным аксессуаром, символизирующим статус и вкус. В России, как и во многих других странах, качественные часы могут рассматриваться как знак успеха и престижа.

#### 2. Растущий средний класс:

С ростом среднего класса и увеличением располагаемого дохода у населения возрастает спрос на качественные товары и брендовые изделия, включая часы. Люди заинтересованы в покупке часов как для личного использования, так и в качестве подарков.

#### 3. Мода и личный стиль:

Модные тенденции и желание выразить свой индивидуальный стиль также способствуют популярности магазинов часов. Часы считаются важной частью образа, и потребители ищут разнообразие дизайнов и брендов, чтобы подчеркнуть свою уникальность.

Рисунок 14 - Десятый вопрос

## 4. Технологический прогресс:

С развитием технологий появляются умные часы, которые предлагают множество функций помимо отслеживания времени, таких как мониторинг здоровья, связь со смартфонами и даже возможность оплаты покупок. Интерес к новейшим технологиям может стимулировать спрос на эти устройства.

## 5. Туризм и путешествия:

Россия привлекает туристов со всего мира, и многие из них могут искать местные или международные бренды часов в качестве сувениров или знаков качества.

## 6. Традиции и подарки:

Часы часто выбирают в качестве подарка на важные события, такие как юбилеи, свадьбы, выпускные и профессиональные достижения. В России, где подарки играют важную роль в социальных и профессиональных отношениях, это может способствовать актуальности магазинов часов.

#### Заключение:

Хотя конкретные причины актуальности магазина часов могут варьироваться, общий интерес к часам в России может быть связан с сочетанием утилитарной потребности, желания выразить личный иль, интереса к новым технологиям и сохранения традиционных ценностей, связанных с подарками и статусом.

Продолжение рисунка 14 - Десятый вопрос

# Вывод

В ходе выполнения лабораторной работы были заданы 10 вопросов в ChatGPT связанных с темой выпускной квалификационной работой и Android разработкой.