Лабораторная работа № 1
Создание макетов для Android-приложения.

1. Теоретический материал.

1.1. Введение в язык Kotlin

Переменные в kotlin объявляются с помощью ключевых слов **val** и **var**. Общий паттерн объявления переменной следующий (в [] заключены опциональные значения):

val/var имя_переменной [: тип_переменной] = значение_переменной

Тип следует за именем поля и отделяется от него двоеточием (:). Тип в большинстве случаев можно опустить.

Переменные объявленные с помощью **var** можно переназначить.

Переменные объявленные с помощью **val** нельзя переназначить. Но они не являются неизменяемыми, им всеравно могут изменть значения.

В большинстве случаев предпочтение следует отдавать ссылкам val.

Kotlin — это статически типизированный язык. Проверка типов выполняется во время компиляции. В Kotlin существует механизм вывода типов.

Особенности системы типов:

- Нет примитивных типов
- У всего есть тип
- Тип Unit
- Есть функциональные типы
- Null-безопасность
- Есть обобщенные типы

В Kotlin нет примитивных типов, которые бы загромождали его систему типов, по крайней мере, на уровне программиста. Отсутствие элементарных типов упрощает программирование, так как не требуется использовать специальные функции, и поддерживаются коллекции числовых и логических значений, не требующие преобразовывать элементарные значения в экземпляры классов и обратно. Внутреннее представление простых типов в Kotlin не связано с системой типов этого языка.

Система типов Kotlin поддерживает функциональные типы. С помощью функциональных типов функции могут получать другие функции в качестве параметров или возвращать их в виде значений. Такие функции называются - функциями высшего порядка.

Возможность передавать функции в качестве аргументов в другие функции является основой функциональных языков. Если последним аргументом функции является другая функция (функция высшего порядка), то можно вынести лямбда-выражение, переданное в качестве параметра, за скобки, которые обычно ограничивают фактический список параметров.

Kotlin поддерживает функциональные литералы: лямбда-выражения. В Koltin лямбда-выражения всегда окружены фигурными скобками. В этих скобках список аргументов находится слева от стрелки ->, а выражение, представляющее собой значение выполнения, лямбда-выражения, располагается справа.

Kotlin поддерживает объектно-ориентированную парадигму программирования. Представлением объекта является класс. Класс фактически представляет определение объекта. А объект является конкретным воплощением

класса.

Для создания объекта необходимо вызвать конструктор класса. По умолчанию компилятор создает конструктор, который не принимает параметров и который мы можем использовать. Но также мы можем определять свои собственные конструкторы. Классы в Kotlin могут иметь один первичный конструктор (primary constructor) и один или несколько вторичных конструкторов (secondary constructor).

Каждый класс может хранить некоторые данные или состояние в виде свойств. Свойства представляют переменные, определенные на уровне класса с ключевыми словами val и var.

Свойство в Kotlin напоминает сочетание поля Java и его метода чтения (если свойство доступно только для чтения и определено с помощью ключевого слова val) или его методов чтения и записи (если оно определено с помощью ключевого слова var).

Если свойство определено с помощью val, то значение такого свойства можно установить только один раз, то есть оно immutable.

Если свойство определено с помощью var, то значение этого свойства можно многократно изменять.

Объявлять методы доступа к свойствам не требуется. Они генерируются автоматически во время компиляции

Свойство должно быть инициализировано, то есть обязательно должно иметь начальное значение.

Код выглядит так, будто он обращается к полю name напрямую, на самом деле используется сгенерированный метод чтения. Он имеет то же имя, что и поле, и его вызов не должен сопровождаться круглыми скобками.

Класс также может содержать функции.

Функции определяют поведение объектов данного класса.

Функции класса определяется также как и обычные функции.

В функциях, которые определены внутри класса, доступны свойства этого класса.

Базовый класс (класс-родитель, родительский класс, суперкласс), который определяет базовую функциональность.

Производный класс (класс-наследник, подкласс), который наследует функциональность базового класса и может расширять или модифицировать ее.

При наследовании производный класс должен вызывать первичный конструктор (а если такого нет, то конструктор по умолчанию) базового класса.

Вызвать конструктор базового класса в производном классе можно двумя способами:

- после двоеточия сразу указать вызов конструктора базового класса
- вызвать конструктор базового класса - определить в производном классе вторичный конструктор и в нем вызвать конструктор базового класса с помощью ключевого слова super

В производном классе для переопределения свойства перед ним указывается аннотация override.

Если свойство определяется через первичный конструктор, то также перед его определением ставится аннотация open

При переопределении в производном классе к этим функциям применяется аннотация override

1.2. Представления и макеты

Графический интерфейс пользователя формируется с помощью объектов: **View** (представление) класс **android.view.View** и **ViewGroup** (макет) класс **android.view.ViewGroup**. Класс **ViewGroup** является дочерним классом для **View**.

Класс **View** - основа для подклассов, которые называются виджетами и представляют собой элементы пользовательского интерфейса: текстовые поля, кнопки и т.д.

Объект **ViewGroup** - это контейнер, содержащий и упорядочивающий дочерние объекты **View**. Контейнерами являются такие элементы, как **RelativeLayout**, **LinearLayout**, **GridLayout**, **ConstraintLayout** и т.д.

Разметка интерфейса пользователя - это процесс размещения элементов интерфейса (виджетов) в конкретном окне приложения (для конкретной деятельности).

Способы разметки:

- использовать разметку состоящую из xml-тегов хранящуюся в отдельном файле xml;

- создавать графические элементы в теле программы используя соответствующие классы.

Рассмотрим некоторые атрибуты, используемые для настройки виджетов.

Размеры визуальных компонент задаются параметрами **layout_width** и **layout_height**, которые могут принимать значения:

– **match_parent** - полностью заполнит родительский элемент (контейнер);

– **wrap_content -** определяет размер в зависимости от содержимого самого элемента + отступ;

– абсолютный (точный) размер.

Важными элементами Android-приложений являются Адаптеры (Adapter), которые позволяют отображать динамически изменяющиеся данные. Класс AdapterView используется для отображения динамического контента на форме. Adapter действует как посредник между источником данных и AdapterView. Adapter извлекает данные из источника (как массив или база данных) и преобразует каждую запись в представление, которое можно добавить в AdapterView. При этом контент будет отображаться в виде прокручивающегося списка.

Наиболее часто используемыми адаптерами являются ArrayAdapter, SimpleAdapter и SimpleCursorAdapter.

ArrayAdapter используется для привязки к массиву объектов. По умолчанию ArrayAdapter использует метод toString() для каждого элемента в массиве, чтобы заполнить текстовыми данными TextView.

SimpleAdapter позволяет привязать ListView к списку ArrayList, содержащему объекты типа Map (ассоциативные массивы, содержащие пары «ключ-значение»). Для каждого такого объекта при отображении используется один элемент из ListView.

SimpleCursorAdapter используется для источника данных Cursor, который обычно является результатом запроса к СУБД или Контент-Провайдеру.

Чтобы применить Адаптер, необходимо разместить в макете один из компонент для отображения динамических данных:

2. Ход выполнения проекта.

2.1 Создание проекта

Создать пустой проект. Выбрать «Empty Views Activity», как приведено на рисунке № 1.

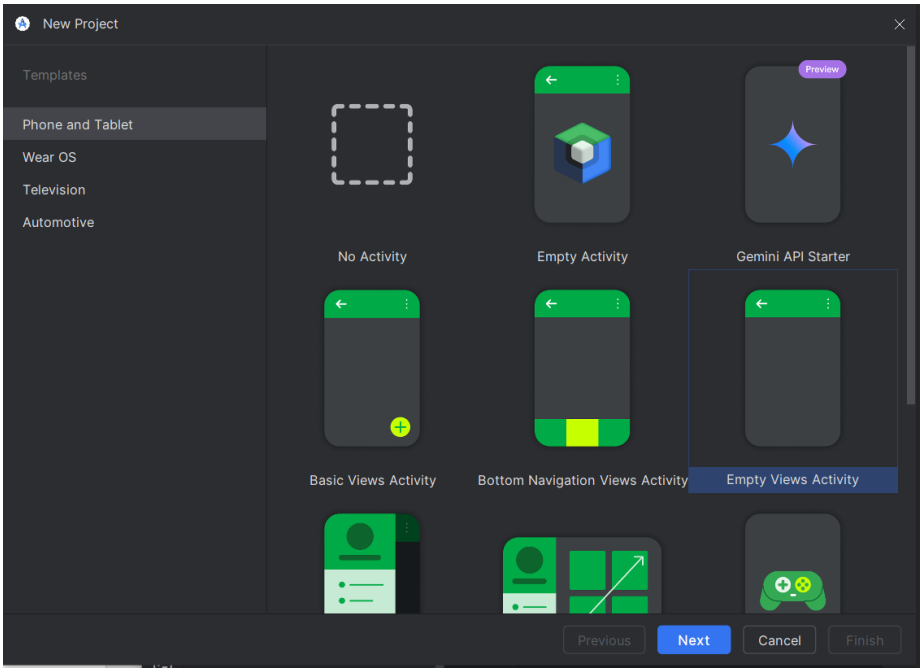

Рисунок № 1

В появившемся окне для создаваемого проекта необходимо указать названиепроекта, названия пакета, путь для сохранения, язык программирования Kotlin и минимальную версию SDK, которую будет поддерживать Ваше приложение. Пример окна приведен на рисунке № 2.

Нажмите кнопку «Finish». Будет создан шаблон проекта и нужно дождаться пока Gradle завершит импорт и настройку проекта (самим ничего запускать не нужно, Gradle будет запущен автоматически после создания). Настройка потребует некоторого вермени и доступа в сеть Интернет (при первом запуске). Прогресс работы Gradle отображается в нижнем правом углу «Android Studio».

После завершения настройки можно работать с проектом.

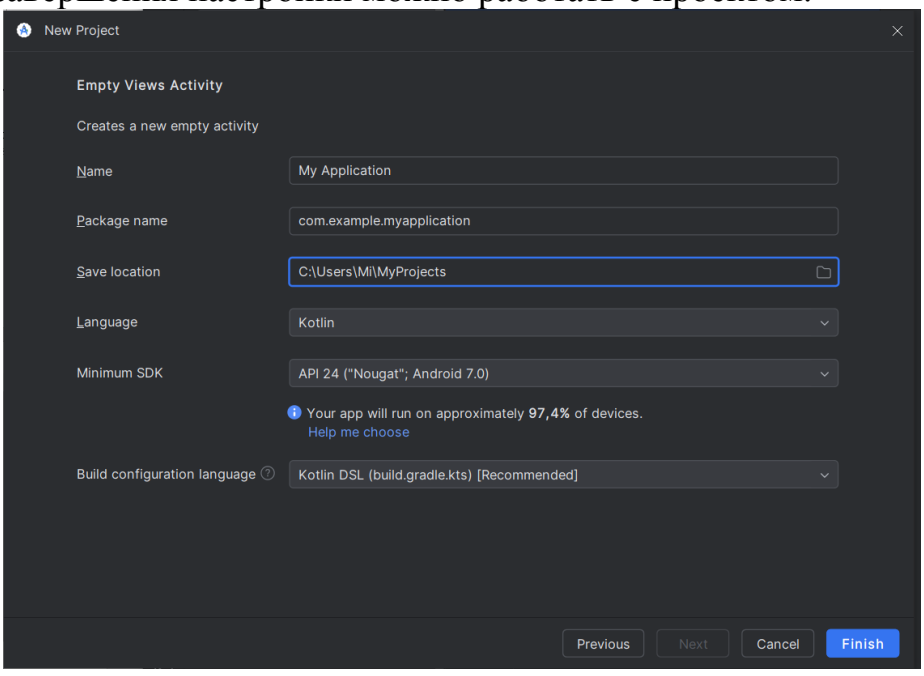

Рисунок № 2

2.2 Создадим класс активности.

В созданном проекте в разделе «kotlin+java» автоматически создан файл «MainActivity.kt» с кодом по умолчанию. Пример кода приведен в листинге № 1 (созданный код будет отличаться в разных версиях IDE).

Листинг № 1

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContentView(R.layout.activity_main)
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) {
v, insets ->
            val systemBars = insets.getInsets(WindowInsetsCompat.Type.sys-
temBars())
            v.setPadding(systemBars.left, systemBars.top, systemBars.right,
systemBars.bottom)
            insets
        }
    }
}
```

Создадим новую Activity. Для создания произведем щелчёк мышью по пакету и выберем создать новый класс в контекстном меню как приведено на рисунках №№ 3-4. Необходимо задать имя создаваемого класса. После создания в проекте появится файл с именем совпадающем с именем класса который был создан ранее и с расширением .kt. Его необходимо открыть.

Далее необходимо произвести наследование созданного класса от класса Activity и реализовать метод onCreate(), как приведено в листинге № 2.

Листинг № 2

```
package com.example.myapplication
import android.app.Activity
class HelloActivity: Activity() {override fun
onCreate(savedInstanceState: Bundle?) {
super.onCreate(savedInstanceState)
 }}
```

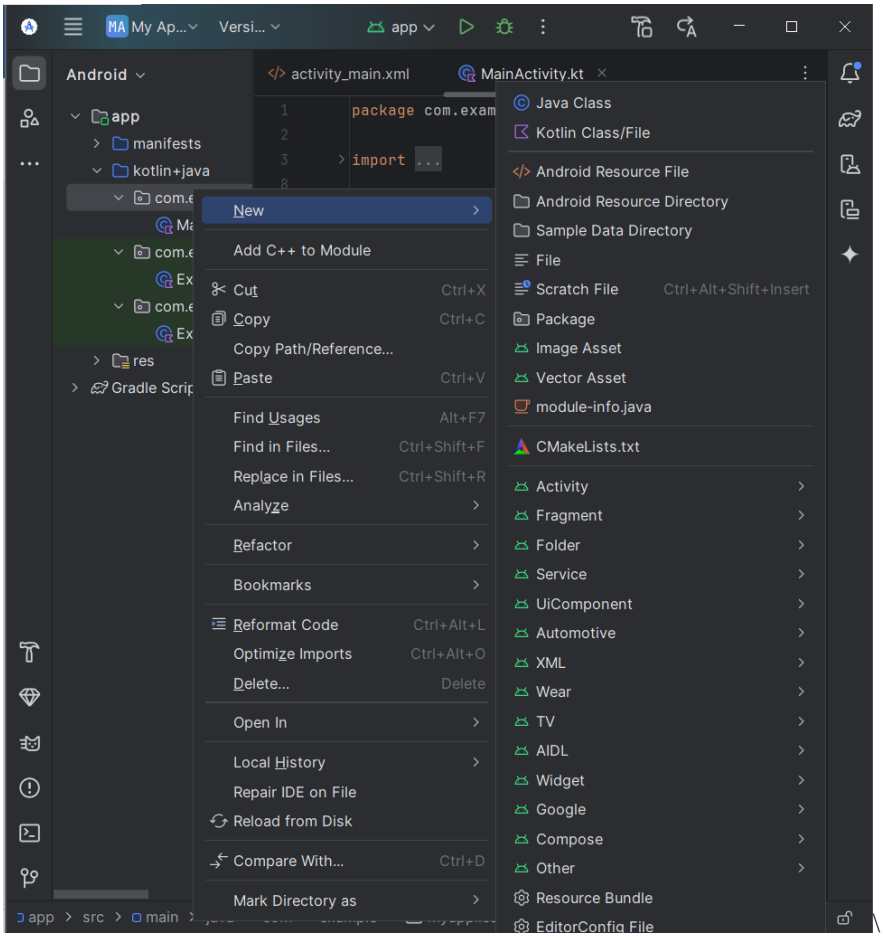

Рисунок 3

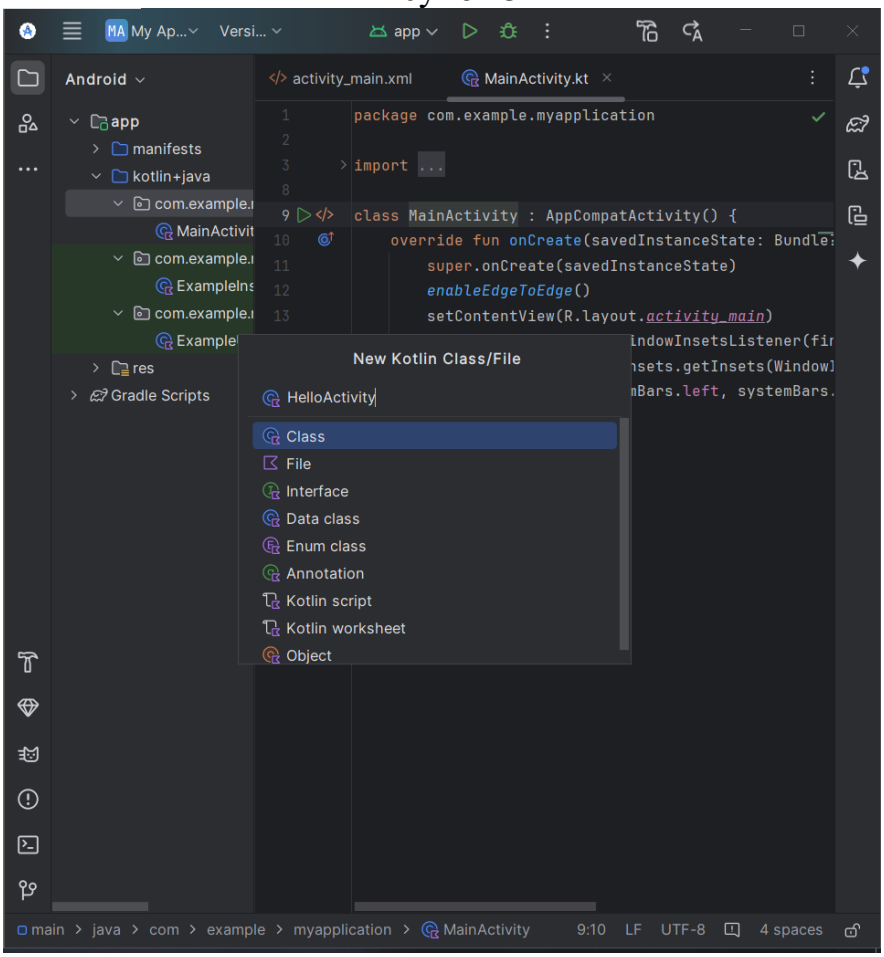

Рисунок 4

2.3. Регистрация activity в качестве стартовой

В приложении одна из активностей должна быть главной. Она запускается в качестве первого экрана приложения.

Пропишем ранее созданную HelloActivity в качестве главной для приложения вместо созданной по умолчанию. Для этого в файле AndroidManifest.xml создаем/редактируем xml-код в теге <activity> как продемонстрированно в листинге № 2.

В дочернем теге action тега intent-filter прописываем MAIN.

Для того, чтобы на экране приложений в устройстве появилась иконка запуска нашего приложения нужно также прописать в дочернем теге category тега intent-filter категорию LAUNCHER. Для activuty с такой категорией также необходимо указывать значение exported = true.

Листинг № 2

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/an-
droid"
    xmlns:tools="http://schemas.android.com/tools">
    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extrac-
tion_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.MyApplication"
        tools:targetApi="31">
        <activity
            android:name=".HelloActivity"
            android:exported="true">
            <intent-filter>
                <action android:name=
                                "android.intent.action.MAIN" />
                <category android:name=
                            "android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

2.4. Запуск на эмуляторе

Запустим созданное приложение на эмуляторе или на подключенном устройстве. При запуске приложения с помощью нажатия на кнопку «Run» или комбинацией клавиш «Shift+F10» запуститься эмулятор по умолчанию и там будет выведено пустое окно, так как еще не добавлен ни один визуальный элемент.

2.5. Создание файла для макета

Создадим файл макета. Кликом правой клавищи мыши на каталоге **layout** в разделе **res** вызываем контекстное меню и создаем файл activity_helloact.xml который будет содержать макет. При создании в качестве

базового элемента указать LinearLayout.

Процесс создания приведен на рисунках №№5-6.

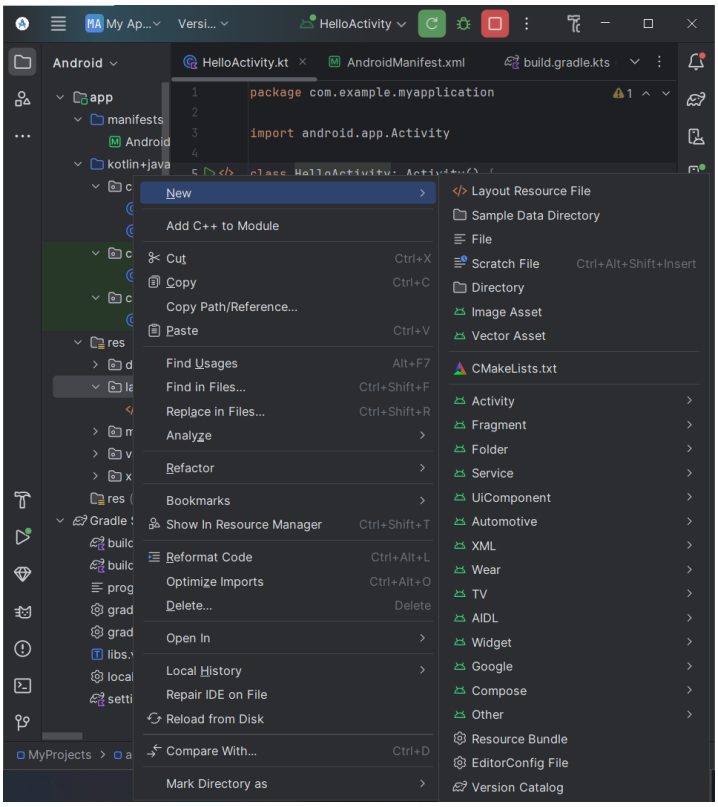

Рисунок № 5

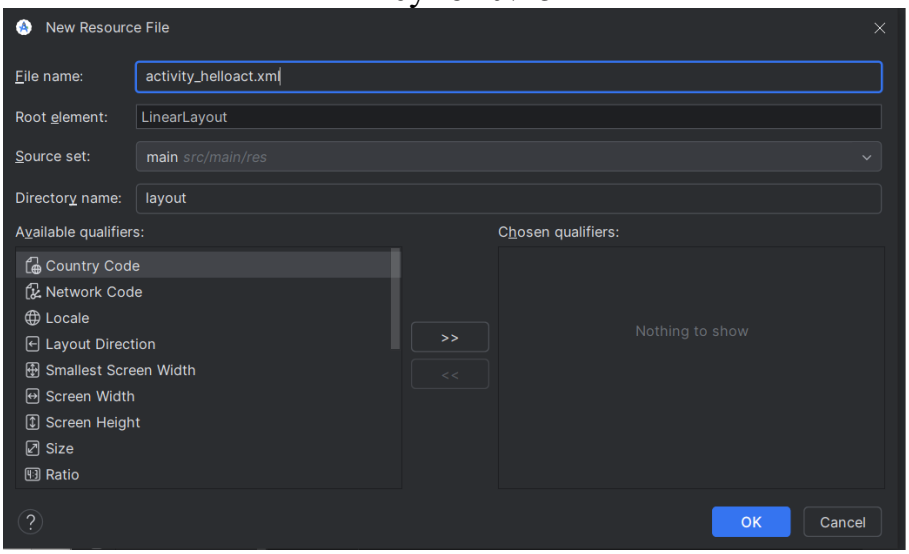

Рисунок № 6

2.6. Создание визуальных элементов

Используем два визуальных элемента: текст и кнопку.

Создадим текст с помощью тега TextView и произведен настройку отображения с помощью атрибутов указанного тега. Код создания текста приведен в листинге № 3.

Листинг № 3

```
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Hello!"/>
```

Создадим кнопку с помощью тега Button. Код создания кнопки приведен в листинге № 4.

Листинг № 4

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom"
    android:text="Нажми!"/>
```

2.7. Подключение созданного макета к activity

Подключим созданный макет к activity. Для этого используем, метод setContentView. Использование вышеуказанного метода продемонтсрировано в листинге № 5.

Листинг № 5

```
package com.example.myapplication

import android.app.Activity
import android.os.Bundle
import android.widget.Button

class HelloActivity : Activity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_helloact)
    }}
```

В результате при запуске приложения теперь должно быть выведено окно приведенное на рисунке № 7.

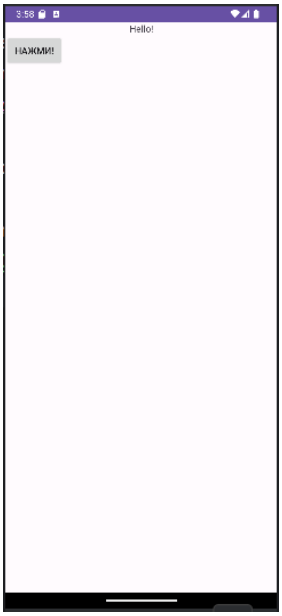

Рисунок № 7

2.8. Обработка нажатия.

Сделаем обработку нажатия на созданную кнопку.

С помощью метода findViewById найдем кнопку по id указанному в файле макета и получем объект который будет представлять ее в коде. Далее с помощью метода setOnClickListener установим функцию которая будет обрабатывать нажатие на кнопку (менять текст кнопки). Код реализующий вышеуказанные действия приведен в листинге № 6.

Листинг № 6.

```
package com.example.myapplication

import android.app.Activity
import android.os.Bundle
import android.widget.Button

class HelloActivity : Activity() {

    override fun onCreate(savedInstanceState: Bundle?)
{
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_helloact)

        val button1
                = findViewById<Button>(R.id.button1)

        button1.setOnClickListener {
            button1.text = "НАЖАТО!"
        } }}
```

2.9. Представление списков

Для отображения структурированных данных которые могут динамически добавляться и удалятся используются такие виджеты как ListView, GridView, Spinner. Все они является наследниками класса android.widget.AdapterView.

Добавим ListView используя одноименный тег. Код с использованием ListView приведен в листинге № 7.

Листинг № 7

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.an-
droid.com/apk/res/android"
        android:id="@+id/activity_main"
        android:orientation="vertical"
     android:layout_width="match_parent"
     android:layout_height="match_parent">

    <ListView
        android:id="@+id/textList"
        android:layout_gravity="center"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
    </ListView>

    <Button
        android:id="@+id/button1"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom"
        android:text="Добавить!"/>
</FrameLayout>
```

Изменим отображение элементов ListView. Создадим новый файл-макета item.xml.

Листинг № 8

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.an-
droid.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/itemContent"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
</LinearLayout>
```

2.12. Использование адаптеров

ListView отображает данные которые ему предоставляются. ListView отвечает только за визуализацию, а за хранение и доставку отвечают другие элементы.

Данные обычно хранятся в коллекциях, базах данных, файлах и т.д.

Связь между данными и ListView осуществляет с помощью адаптеров.

Определим источник данных в виде списка. Затем создадим адаптер и привяжем его у источнику данных, а также укажем ему использовать созданое нами отображение элементов и куда будет подставлен наш текст. Установим созданный адаптер в ListView. Код реализующий вышеуказанные действия приведен в листинге № 9.

Листинг № 9

```
val myStringArray = ArrayList<String>()
val textAdapter: ArrayAdapter<String> = ArrayAdapter(this, R.layout.item,
R.id.itemContent, myStringArray)
textList.adapter = textAdapter
```

Результат выполнения кода приведен на рисунке № 8.

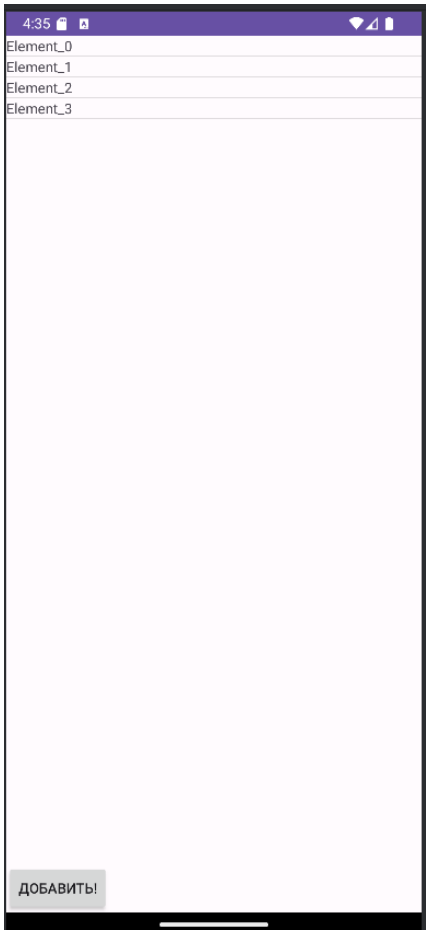

Рисунок № 8

3. Задание для самостоятельной реализации

Самостоятельно необходимо создать два макета и activity для них:

1) Форма регистрации содержащая как минимум следующие поля:

- Поле логина.
- Поле пароля.
- ФИО.
- Дата рождения.
- Выбор пола М/Ж.
- Аватар (картинка).
- Кнопка для выбора аватара (опционально, можно реализовать выбор рисунка для аватара кликом по нему).

2) Список пользователей - просто список и как минимум кнопка добавить в низу.

В ходе реализации самостоятельного задания не нужно делать переходы между activity, контекстные меню и прочие элементы (они будут реализованы в слудеющих лабораторных). Все обработчики кнопок должны выводить в лог сообщения, что они делают. Демонстрация функциональности будет с помощью редактирования файла AndroidManifest.xml. Сначало сделать основной HelloActivity, затем по очереди для формы регистрации и списка пользователей.

4. Вопросы для самопроверки

- Как задать переменную в Kotlin?

- Как создать класс в Kotlin?
- Как унаследоваться от класса в Kotlin?
- Чем отличаются модификаторы val и var в Kotlin?
- Какие примитивные типы есть в Kotlin?
- Какие базовые типы есть в Kotlin?
- Как задать тип переменной в Kotlin?
- Какой модификатор используется для переопределения функции в Kotlin?
- Что такое View?
- Какие View вы знаете?
- Что такое ViewGroup?
- Какие ViewGroup вы знаете?
- Какие способы создания виджетов для activity?
- В чем преимущество xml-макетов?
- С помощью чего настраиваются параметры визуальных элементов таких как позиционирование, цвет и т.д. в xml?
- Как связвать xml-макет и activity?
- В каком методе связывается xml-макет?
- Как найти объект из xml-макета для использования в коде программы?
- Что используется для поиска объектов из xml-макета?
- Как обрабатывать событие клика по кнопке?
- Как задать уникальный идентификатор для элемента в xml-макете?
- Для чего задавать уникальный идентификатор для элемента в xml-макете?
- Для его использовать строковые ресурсы?