



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

«МИРЭА – Российский технологический университет»

**РТУ МИРЭА**

---

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных  
технологий

## **Отчет по практической работе №5**

по дисциплине «Разработка мобильных приложений»

**Выполнил:**

Студент группы ИКБО-20-23

Комисарик М.А.

**Проверил:**

Старший преподаватель кафедры  
МОСИТ

Шешуков Л.С.

Москва 2025 г.

1 ТЕОРЕТИЧЕСКОЕ ВВЕДЕНИЕ .....	3
1.1 ScrollView.....	3
1.2 ListView .....	4
1.3 Spinner.....	13
1.4 Создание адаптера.....	16
1.5 RecyclerView .....	22
2 ПРАКТИЧЕСКОЕ ЗАДАНИЕ .....	28
2.1 ListView .....	28
2.1.1 Создание проекта .....	28
2.1.2 Разметка.....	28
2.1.3 Реализация логики списка .....	31
2.1.4 Тестирование .....	33
2.2 RecyclerView .....	37
2.2.1 Разметка.....	37
2.2.2 Реализация адаптера .....	39
2.2.3 Использование в Activity .....	41
2.2.4 Тестирование .....	41
2.3 ScrollView.....	43
2.4 Spinner.....	45
ЗАКЛЮЧЕНИЕ .....	46

# 1 ТЕОРЕТИЧЕСКОЕ ВВЕДЕНИЕ

## 1.1 ScrollView

ScrollView в Android — это контейнер, который позволяет прокручивать его содержимое по вертикали, что особенно полезно, когда содержимое занимает больше места, чем доступно на экране устройства. Это делает ScrollView идеальным выбором для макетов, где необходимо отобразить большое количество информации или элементов управления, не уместяющихся на одном экране.

Для использования ScrollView необходимо добавить данный контейнер в разметку пользовательского интерфейса и разместить в нем необходимые элементы (Рисунок 1).

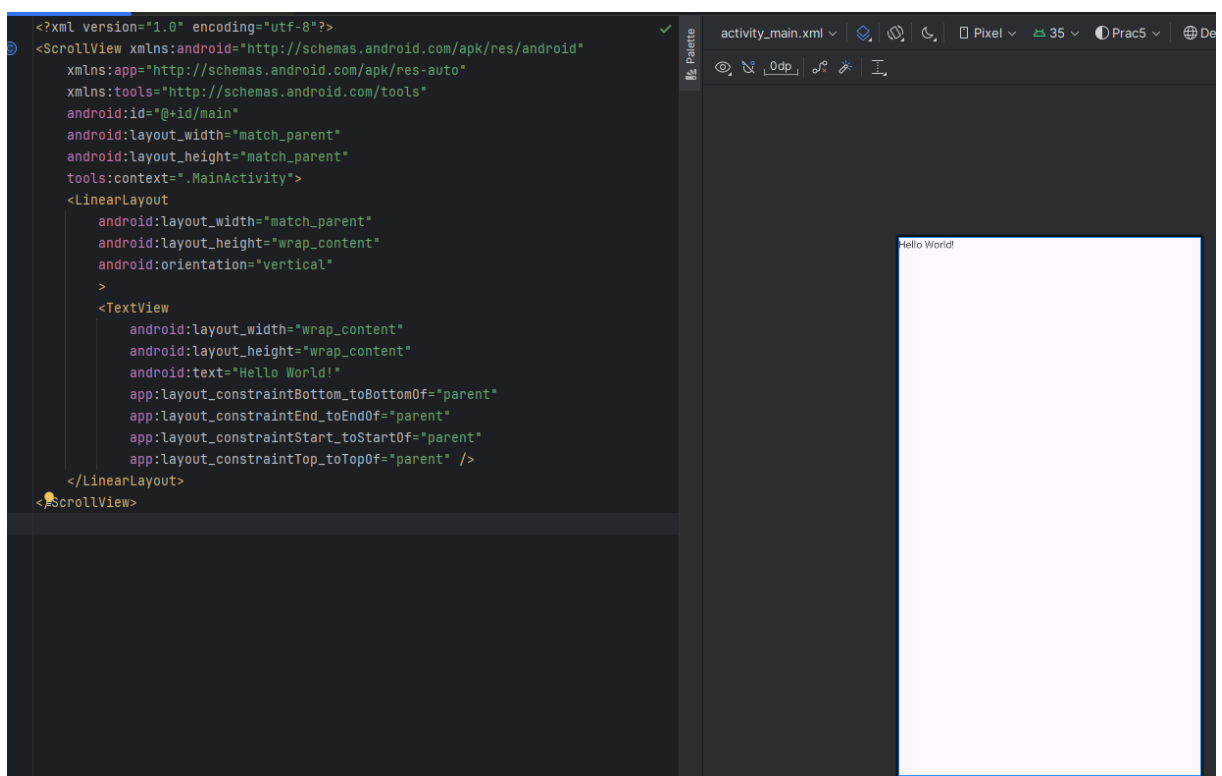


Рисунок 1 – Реализация ScrollView в коде

Так же важно отметить, что для отображения больших списков данных лучше использовать RecyclerView (мощный и гибкий компонент в Android, который используется для отображения больших наборов данных в виде

прокручиваемого списка или сетки) или `ListView` (один из классических компонентов в `Android`, который используется для отображения прокручиваемого списка элементов), так как они оптимизированы для эффективного отображения больших коллекций, управляя повторным использованием и отрисовкой только видимых элементов.

## 1.2 `ListView`

`Android` представляет широкую палитру элементов, которые представляют списки. Все они являются наследниками класса `android.widget.AdapterView` (базовый класс в `Android`, который представляет собой абстракцию для `View`-групп, отображающих данные через адаптер). Это такие виджеты как `ListView`, `GridView` (компонент пользовательского интерфейса, который позволяет отображать элементы в виде сетки (таблицы) с фиксированным количеством столбцов), `Spinner` (выпадающий список, который позволяет пользователю выбрать один элемент из набора вариантов). Они могут выступать контейнерами для других элементов управления.

При работе со списками происходит взаимодействие с тремя компонентами:

- визуальный элемент или виджет, который на экране представляет список (`ListView`, `GridView`) и который отображает данные;
- источник данных – массив, объект `ArrayList`, база данных и т.д., в котором находятся сами отображаемые данные;
- адаптер – специальный компонент, который связывает источник данных с виджетом списка.

Одним из самых простых и распространенных элементов списка является виджет `ListView`.

`ListView` в `Android` — это компонент пользовательского интерфейса, который используется для отображения элементов в виде вертикального списка

(Рисунок 2). Каждый элемент списка может быть одинаковым или различаться в зависимости от адаптера, который используется для связи данных с ListView. ListView является одним из базовых и часто используемых элементов управления для отображения коллекций данных, таких как массивы или списки.

```
<ListView  
    android:id="@+id/my_list_view"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content" />
```

**Рисунок 2 – Разметка ListView**

Теперь необходимо наполнить список данными. Сделать это можно, используя класс Adapter. Это важнейший компонент, который действует как мост между пользовательским интерфейсом компонентов, таких как ListView, RecyclerView, или Spinner, и данными для этих компонентов, обеспечивая доступ к данным и создавая представления для каждого элемента данных в компоненте. Adapter отвечает за преобразование каждого элемента данных во View (представления), которые затем могут быть вставлены в пользовательский интерфейс.

Типы Adapter в Android:

- ArrayAdapter: используется, когда данные представляют собой список или массив. Очень удобен для простых случаев, когда нужно отобразить массив строк или объектов в ListView или Spinner;
- CursorAdapter: подходит для отображения данных, полученных из базы данных. Используется в сочетании с Cursor, представляющим результат запроса к базе данных;
- RecyclerView.Adapter: специализированный адаптер для RecyclerView. Отличается от других типов адаптеров тем, что требует реализации ViewHolder паттерна, который улучшает производительность за счет уменьшения количества вызовов findViewById() при прокрутке списка.

Создадим список с именами и выведем его на экран (Рисунок 3).

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    EdgeToEdge.enable(this);
    setContentView(R.layout.activity_main);
    ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
        Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
        v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
        return insets;
    });
    // находим представление списка
    ListView usersListView = (ListView) findViewById(R.id.
        my_list_view);
    // создаем адаптер
    ArrayAdapter<String> usersAdapter = new ArrayAdapter<String>(context: this,
        android.R.layout.simple_list_item_1, userNames);
    // устанавливаем адаптер для списка
    usersListView.setAdapter(usersAdapter);
}
```

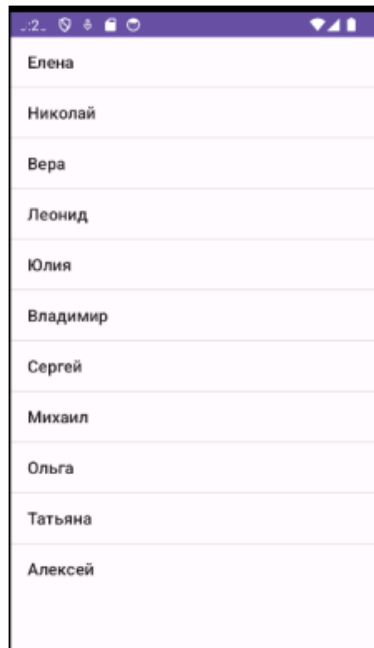
**Рисунок 3 – Использование ListView для отображения списка имён**

Здесь вначале получаем по id элемент ListView и затем создаем для него адаптер. Для создания адаптера использовался следующий конструктор `ArrayAdapter(this, android.R.layout.simple_list_item_1, userNames)`, где:

- `this`: текущий объект `activity`;
- `android.R.layout.simple_list_item_1`: файл разметки списка, который фреймворк представляет по умолчанию. Он находится в папке Android SDK по пути `platforms/[android-номер_версии]/data/res/layout`. Если нас не удовлетворяет стандартная разметка списка, мы можем создать свою и потом в коде изменить `id` на `id` нужной нам разметки;
- `userNames`: массив данных. Здесь необязательно указывать именно массив, это может быть список `ArrayList`.

В конце необходимо установить для `ListView` адаптер с помощью метода `setAdapter()`, который связывает элемент `ListView` с определенным адаптером.

В конечном итоге получится отображение имён в виде вертикального списка (Рисунок 4).

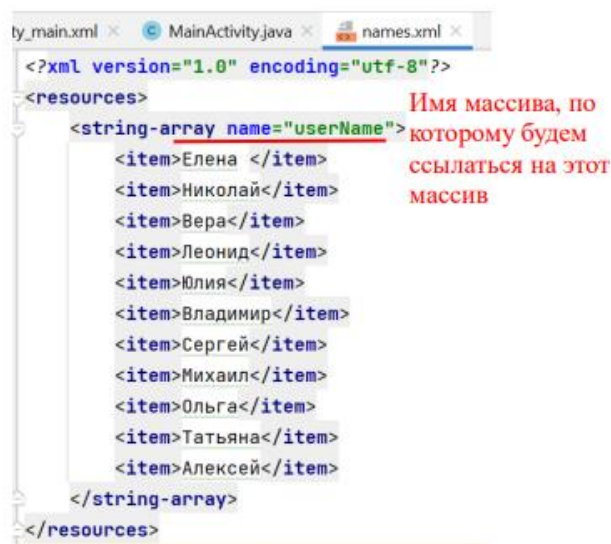


**Рисунок 4 – Отображение вертикального списка при помощи ListView**

В рассмотренной примере массив строк был определен программно в коде java. Однако подобный массив строк гораздо удобнее было бы хранить в файле xml в виде ресурса.

Ресурсы массивов строк представляют элемент типа string-array. Они могут находиться в каталоге res/values в xml-файле с произвольным именем.

Создадим новый ресурс и заполним его элементами (Рисунок 5).



**Рисунок 5 – Ресурсный файл имён**

Массив строк задается с помощью элемента `<string-array>`, атрибут name которого может иметь произвольное значение, по которому затем будут

ссылаться на этот массив. Все элементы массива представляют набор значений `<item>`.

После этого необходимо связать ресурс в коде java. Для получения ресурса в коде Java применяется выражение `R.array.<название_ресурса>` (Рисунок 6).

```
public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        // находим представление списка  
        ListView usersListView = (ListView) findViewById(R.id.my_list_view);  
        //Получаем ресурс с именами  
        String[] userNames = getResources().getStringArray(R.array.userName);  
        // создаем адаптер  
        ArrayAdapter<String> usersAdapter = new ArrayAdapter<String>( context: this,  
            android.R.layout.simple_list_item_1, userNames);  
        // устанавливаем адаптер для списка  
        usersListView.setAdapter(usersAdapter);  
    }  
}
```

Рисунок 6 – Привязка ресурса в Java

Получим тот же самый результат. Но нам необязательно добавлять список строк в `ListView` программно. У этого элемента есть атрибут `entries`, который в качестве значения может принимать ресурс `string-array` (Рисунок 7). В этом случае код `MainActivity` мы можем сократить до стандартного метода `OnCreate()`. Результат будет тот же.

```
<ListView  
    android:id="@+id/my_list_view"  
    android:entries="@array/userName"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"
```

Рисунок 7 – Атрибут `entries` в `ListView`

Кроме простого вывода списка элементов можно также выбирать элементы списка и обрабатывать данный выбор. Для этого необходимо связать список `ListView` с источником данных и закрепить за ним слушатель нажатия на



элемент списка внутри метода onCreate(). Чтобы было нагляднее добавим в разметку элемент TextView для вывода выбранного элемента (Рисунок 8).

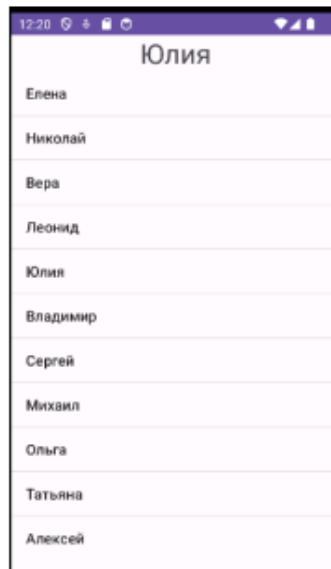
```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // находим представление списка
    ListView usersListView = (ListView) findViewById(R.id.my_list_view);
    //Получаем ресурс с именами
    String[] userNames = getResources().getStringArray(R.array.userName);
    // создаем адаптер
    ArrayAdapter<String> usersAdapter = new ArrayAdapter<String>(context: this,
        android.R.layout.simple_list_item_1, userNames);
    // устанавливаем адаптер для списка
    usersListView.setAdapter(usersAdapter);
    //Получаем элемент TextView, куда будет выводиться элемент
    TextView selectedName = findViewById(R.id.selectedName);
    //Добавляем для списка слушатель
    usersListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
            // по позиции получаем выбранный элемент
            String selectedItem = userNames[position];
            // установка текста элемента TextView
            selectedName.setText(selectedItem);
        }
    });
}
```

Рисунок 8 – Обработка выбора элемента списка имён

Для обработки выбора элемента списка устанавливается слушатель OnItemClickListener. Этот слушатель имеет один метод onItemClick, через параметры которого мы можем получить выделенный элемент и сопутствующие данные. Так, он принимает следующие параметры:

- parent: нажатый элемент AdapterView (в роли которого в данном случае выступает наш элемент ListView);
- view: нажатый виджет внутри AdapterView;
- position: индекс нажатого виджета внутри AdapterView;
- id: идентификатор строки нажатого элемента.

Используя эти параметры, мы можем разными способами получить выделенный элемент (Рисунок 9).



**Рисунок 9 – Отображение выбора элемента из списка имён**

Иногда требуется выбрать не один элемент, как по умолчанию, а несколько. Для этого, во-первых, в разметке списка надо установить атрибут `android:choiceMode="multipleChoice"` (Рисунок 10).

```
<ListView
    android:id="@+id/my_list_view"
    android:choiceMode="multipleChoice"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
```

**Рисунок 10 – Определение значения `android:choiceMode`**

После этого в коде `MainActivity` определяется обработка выбора элементов списка (Рисунок 11).

```
ArrayAdapter<String> usersAdapter = new ArrayAdapter<String>( context: this,
    android.R.layout.simple_list_item_multiple_choice, userNames);
usersListView.setAdapter(usersAdapter);

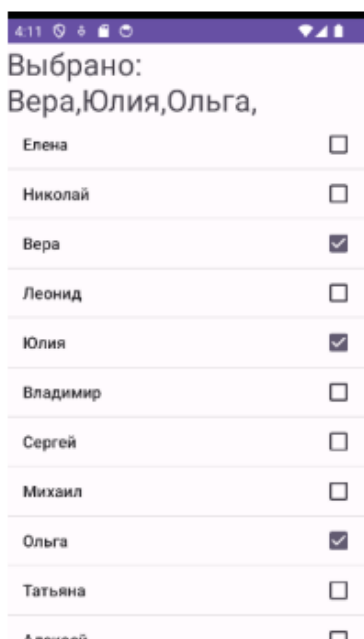
usersListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        SparseBooleanArray selected=usersListView.getCheckedItemPositions();

        String selectedItems="";
        for(int i=0;i < userNames.length;i++)
        {
            if(selected.get(i))
                selectedItems+=userNames[i]+",";
        }
        // установка текста элемента TextView
        selectedName.setText("Выбрано: " + selectedItems);
    }
});
```

**Рисунок 11 – Реализация логики обработки выбора элементов списка**

Ресурс `android.R.layout.simple_list_item_multiple_choice` представляет стандартную разметку, предоставляемую фреймворком, для создания списка с множественным выбором.

А при выборе элементов мы получаем все выбранные позиции в объект `SparseBooleanArray`, затем пробегаемся по всему массиву, и если позиция элемента в массиве есть в `SparseBooleanArray`, то есть она отмечена, то добавляем отмеченный элемент в строку (Рисунок 12).



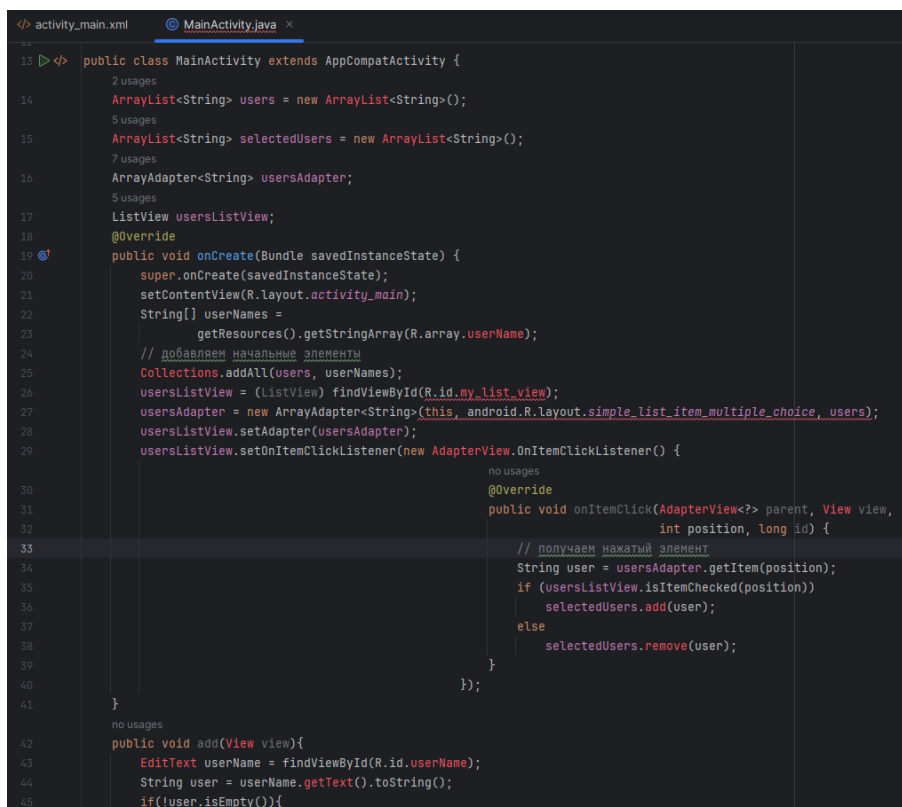
**Рисунок 12 – Отображение выбора элемента в списке имён**

После привязки `ListView` к источнику данных через адаптер можно работать с данными – добавлять, удалять, изменять только через адаптер. `ListView` служит только для отображения данных.

Для управления данными используются методы адаптера или напрямую источника данных. Однако после применения таких методов изменения коснутся только массива, выступающего источником данных. Чтобы синхронизировать изменения с элементом `ListView`, надо вызвать у адаптера метод `notifyDataSetChanged()`.

Создадим интерфейс, в котором определим для вывода списка `ListView` с возможностью множественного выбора элементов, для добавления и удаления

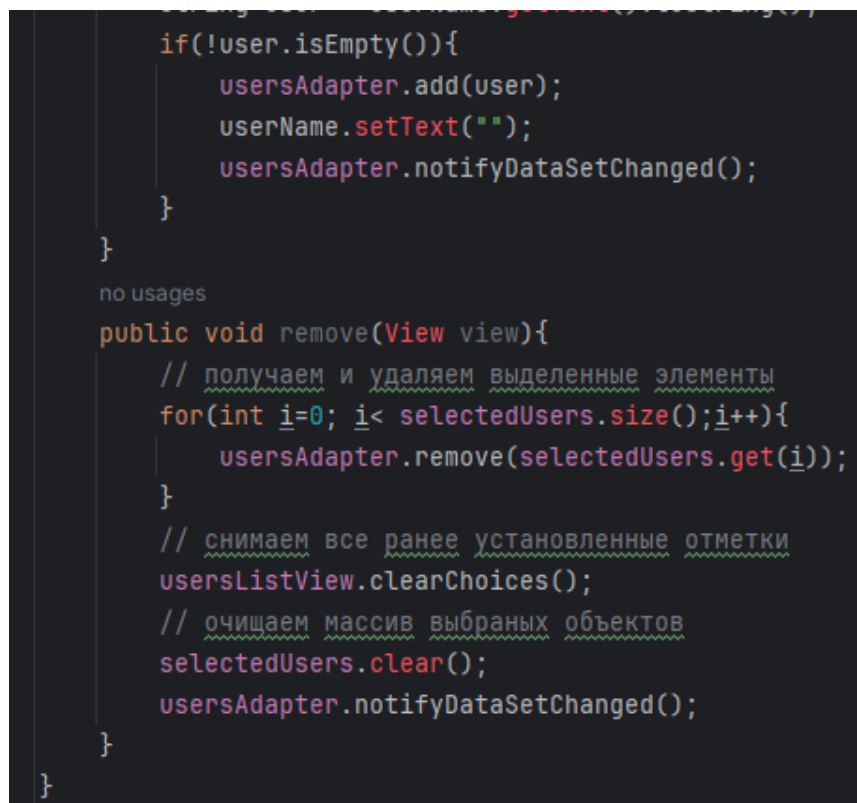
определены элементов две кнопки, а для ввода нового объекта в список поле EditText. После этого изменим класс MainActivity (Рисунки 13-14).



```

13  public class MainActivity extends AppCompatActivity {
14      ArrayList<String> users = new ArrayList<String>();
15      ArrayList<String> selectedUsers = new ArrayList<String>();
16      ArrayAdapter<String> usersAdapter;
17      ListView usersListView;
18      @Override
19      public void onCreate(Bundle savedInstanceState) {
20          super.onCreate(savedInstanceState);
21          setContentView(R.layout.activity_main);
22          String[] userNames =
23              getResources().getStringArray(R.array.userName);
24          // добавляем начальные элементы
25          Collections.addAll(users, userNames);
26          usersListView = (ListView) findViewById(R.id.my_list_view);
27          usersAdapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_multiple_choice, users);
28          usersListView.setAdapter(usersAdapter);
29          usersListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
30              @Override
31              public void onItemClick(AdapterView<?> parent, View view,
32                  int position, long id) {
33                  // получаем нажатый элемент
34                  String user = usersAdapter.getItem(position);
35                  if (usersListView.isItemChecked(position))
36                      selectedUsers.add(user);
37                  else
38                      selectedUsers.remove(user);
39              }
40          });
41      }
42      public void add(View view){
43          EditText userName = findViewById(R.id.userName);
44          String user = userName.getText().toString();
45          if(!user.isEmpty()){
  
```

Рисунок 13 – Изменения функционала итогового приложения, часть 1



```

46          if(!user.isEmpty()){
47              usersAdapter.add(user);
48              userName.setText("");
49              usersAdapter.notifyDataSetChanged();
50          }
51      }
52      no usages
53      public void remove(View view){
54          // получаем и удаляем выделенные элементы
55          for(int i=0; i< selectedUsers.size();i++){
56              usersAdapter.remove(selectedUsers.get(i));
57          }
58          // снимаем все ранее установленные отметки
59          usersListView.clearChoices();
60          // очищаем массив выбранных объектов
61          selectedUsers.clear();
62          usersAdapter.notifyDataSetChanged();
63      }
64  }
  
```

Рисунок 14 – Изменения функционала итогового приложения, часть 2

С добавлением все относительно просто: получаем введенную строку и добавляем в список с помощью метода `usersAdapter.add()`.

Чтобы обновить `ListView` после добавления вызывается метод `adapter.notifyDataSetChanged()`.

А для удаления создается дополнительный список `selectedUsers`, который будет содержать выделенные элементы. Для получения выделенных элементов и добавления их в список используется слушатель `AdapterView.OnItemClickListener`, метод `onItemClick()` которого вызывается при установке или снятии отметки с элемента, то есть при любом нажатии на элемент (Рисунок 15).

По нажатию на кнопку удаления пробегаемся по списку выделенных элементов и вызываем для каждого из них метод `usersAdapter.remove()`.

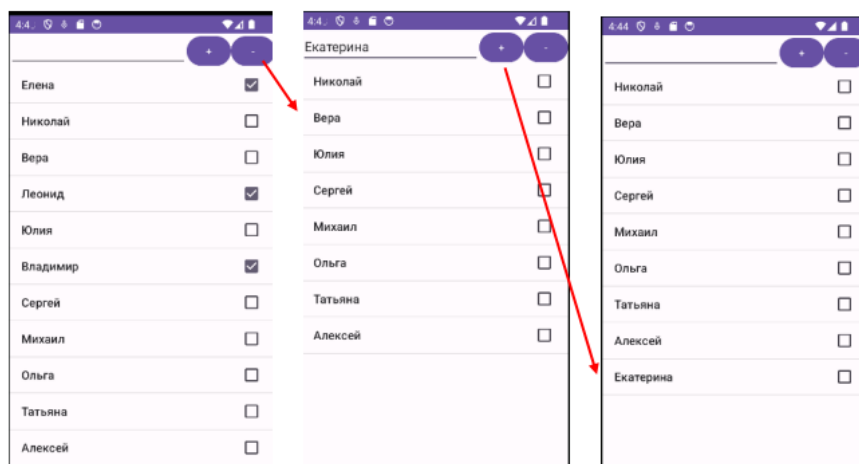


Рисунок 15 – Отображение изменённого функционала приложения

## 1.3 Spinner

Spinner представляет собой выпадающий список. В файле разметки объявляется как элемент `Spinner` (Рисунок 16).

```
<Spinner
    android:id="@+id/spinner"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Рисунок 16 – Разметка Spinner

В качестве источника данных, как и для ListView, для Spinner может служить простой список или массив, созданный программно, либо ресурс string-array. Взаимодействие с источником данных также будет идти через адаптер (Рисунок 17).

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    //Находим элемент Spinner
    Spinner spinnerName = findViewById(R.id.spinner);
    //Получаем ресурс с именами
    String[] userNames = getResources().getStringArray(R.array.userName);

    //Создаем адаптер ArrayAdapter с ресурса строк и стандартной разметки элемента spinner
    ArrayAdapter<String> adapter = new ArrayAdapter
        ( context: this, android.R.layout.simple_spinner_item, userNames);
    // Определяем разметку для использования при выборе элемента
    adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
    // Применяем адаптер к элементу spinner
    spinnerName.setAdapter(adapter);
}
```

Рисунок 17 – Реализация взаимодействия источника данных с адаптером

При такой реализации на экране будет отображаться список как на рисунке

18

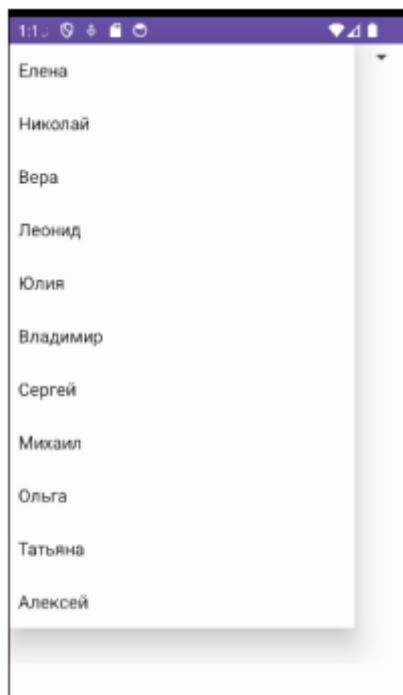


Рисунок 18 – Отображаемый список в итоговом приложении

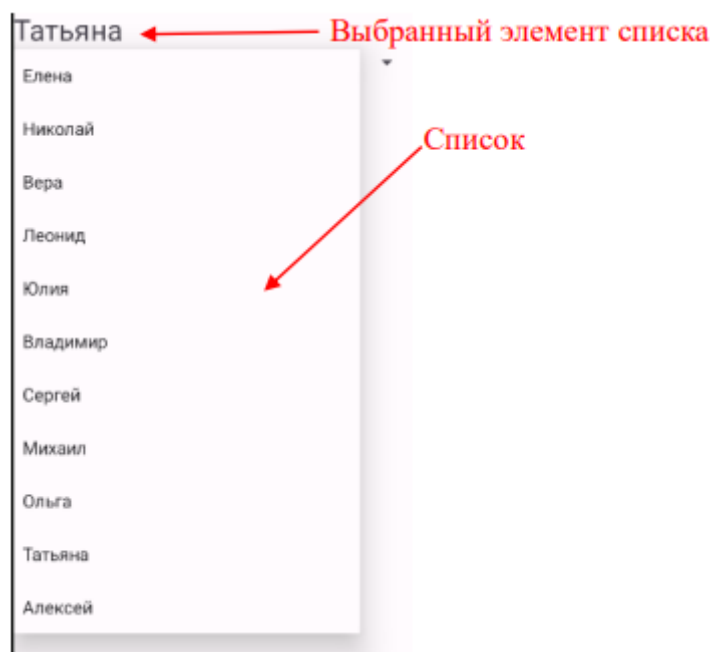
Используемый при создании ArrayAdapter ресурс android.R.layout.simple\_spinner\_item предоставляется платформой и является стандартной разметкой для создания выпадающего списка.

С помощью метода `adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item)` устанавливаются дополнительные визуальные возможности списка.

А передаваемый в метод ресурс `android.R.layout.simple_spinner_dropdown_item` используется для визуализации выпадающего списка и также предоставляется платформой.

Однако можно не только получать список, но и используя слушатель `OnItemSelectedListener`, в частности его метод `onItemSelected()`, можно обрабатывать выбор элемента из списка. Вначале добавим в разметку интерфейса текстовое поле, которое будет выводить выбранный элемент.

Для этого необходимо добавить элемент в разметки, чтобы выводить выбранный элемент и в коде java определить для элемента `Spinner` слушатель `OnItemSelectedListener` (Рисунок 19).



**Рисунок 19 – Отображение элемента, выбранного из списка имён**

Метод `onItemSelected` слушателя `OnItemSelectedListener` получает четыре параметра:

- `parent`: объект `Spinner`, в котором произошло событие выбора элемента;
- `view`: объект `View` внутри `Spinner`, который представляет выбранный элемент;
- `position`: индекс выбранного элемента в адаптере;
- `id`: идентификатор строки того элемента, который был выбран.

Получив позицию выбранного элемента, мы можем найти его в списке.

Для установки слушателя `OnItemSelectedListener` в классе `Spinner` применяется метод `setOnItemSelectedListener`.

## 1.4 Создание адаптера

Традиционные списки `ListView`, использующие стандартные адаптеры `ArrayAdapter`, прекрасно работают с массивами строк. Однако чаще сталкиваются с более сложными по структуре списками, где один элемент представляет не одну строку, а несколько строк, картинок и других компонентов.

Для создания сложного списка необходимо переопределить один из используемых адаптеров. Поскольку, как правило, используется `ArrayAdapter`, то его переопределим.

Перед этим необходимо определить модель, данные, которые будут отображаться в списке. Для этого нужно создать новый класс, который хранит два строковых поля – имя и фамилия человека (Рисунок 20).



```

1  package com.example.prac5;
2
3  public class Name {
4      private String name;
5      private String surname;
6
7      public Name(String name, String surname) {
8          this.name = name;
9          this.surname = surname;
10     }
11
12     public String getName() {
13         return name;
14     }
15
16     public void setName(String name) {
17         this.name = name;
18     }
19
20     public String getSurname() {
21         return surname;
22     }
23
24     public void setSurname(String surname) {
25         this.surname = surname;
26     }
27 }

```

**Рисунок 20 – Создание класса, хранящего имя и фамилию человека**

После этого необходимо создать новый файл разметки `list_item.xml`, который будет представлять разметку одного элемента в списке. Каждый элемент будет иметь два компонента `TextView` для отображения имени и фамилии.

Теперь можно создавать новый адаптер: создаем класс и называем его `NameAdapter` (Рисунок 21).

```

1  package com.example.prac5;
2
3  public class NameAdapter extends ArrayAdapter<Name> {
4      private LayoutInflater inflater;
5      private int layout;
6      private List<Name> names;
7      public NameAdapter(Context context, int resource, List<Name> name)
8      {
9          super(context, resource, name);
10         this.names=name;
11         this.layout=resource;
12         this.inflater=LayoutInflater.from(context);
13     }
14     public View getView(int position, View convertView, ViewGroup
15         parent) {
16         View view=inflater.inflate(this.layout, parent, false);
17         TextView nameView = view.findViewById(R.id.name);
18         TextView surnameView = view.findViewById(R.id.surname);
19         Name name = names.get(position);
20         nameView.setText(name.getName());
21         surnameView.setText(name.getSurname());
22         return view;
23     }
24 }

```

**Рисунок 21 – Описание логики класса NameAdapter**

Все взаимодействие со списком здесь будет идти через класс NameAdapter.

В конструктор базового класса три параметра:

- контекст, в котором используется класс. В его роли как правило выступает класс Activity;
- ресурс разметки интерфейса, который будет использоваться для создания одного элемента в ListView;
- набор объектов, которые будут выводиться в ListView;

В конструкторе NameAdapter мы получаем ресурс разметки и набор объектов и сохраняем их в отдельные переменные. Кроме того, для создания

объекта View по полученному ресурсу разметки потребуется объект LayoutInflater, который также сохраняется в переменную.

В методе getView() устанавливается отображение элемента списка. Данный метод принимает три параметра:

- position: передает позицию элемента внутри адаптера, для которого создается представление;
- convertView: старое представление элемента, которое при наличии используется ListView в целях оптимизации;
- parent: родительский компонент для представления элемента;

В данном случае с помощью объекта LayoutInflater создаем объект View для каждого отдельного элемента в списке.

Из созданного объекта View получаем элементы TextView по id – те элементы, которые определены в файле list\_item.xml. Далее используя параметр position, получаем объект Name, для которого создается разметка. Затем полученные элементы TextView наполняем из полученного по позиции объекта Name. И в конце созданный для отображения объекта Name элемент View возвращается из метода return view.

В файле activity\_main.xml определим ListView, в который будут загружаться данные. А в файле MainActivity соединим NameAdapter с ListView (Рисунок 22).

```

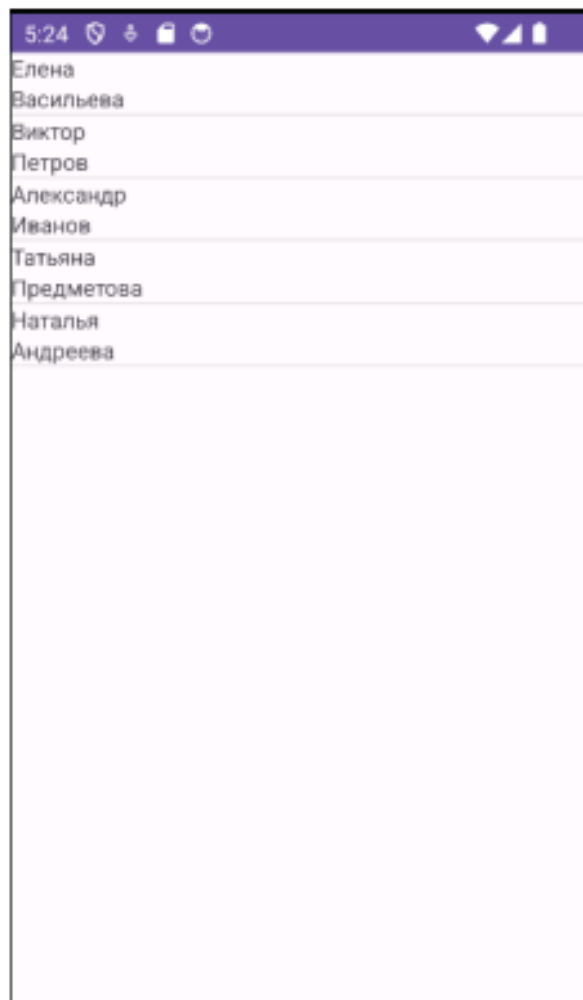
18 public class MainActivity extends AppCompatActivity {
19     ArrayList<Name> names = new ArrayList<Name>();
20     ListView nameList;
21     @Override
22     protected void onCreate(Bundle savedInstanceState) {
23         super.onCreate(savedInstanceState);
24         setContentView(R.layout.activity_main);
25         // начальная инициализация списка
26         setInitialData();
27         // получаем элемент ListView
28         nameList = findViewById(R.id.nameList);
29         // создаем адаптер
30         NameAdapter stateAdapter = new NameAdapter( context: this, R.layout.list_item, names);
31         // устанавливаем адаптер
32         nameList.setAdapter((ListAdapter) stateAdapter);
33         // слушатель выбора в списке
34         AdapterView.OnItemClickListener itemListener = new
35             AdapterView.OnItemClickListener() {
36             @Override
37             public void onItemClick(AdapterView<?> parent, View v, int
38                 position, long id) {
39                 // получаем выбранный пункт
40                 Name selectedName =
41                     (Name)parent.getItemAtPosition(position);
42                 Toast.makeText(getApplicationContext(), text: "Было выбрано имя " + selectedName.getName(),
43                     Toast.LENGTH_SHORT).show();
44             }
45         };
46         nameList.setOnItemClickListener(itemListener);
47     }
48     1 usage
49     private void setInitialData(){
50         names.add(new Name ( name: "Елена", surname: "Васильева"));
51         names.add(new Name ( name: "Виктор", surname: "Петров"));
52         names.add(new Name ( name: "Александр", surname: "Иванов"));
53         names.add(new Name ( name: "Татьяна", surname: "Предметова"));
54         names.add(new Name ( name: "Наталья", surname: "Андреева"));
55     }

```

**Рисунок 22 – Соединение NameAdapter с ListView**

В качестве источника данных здесь выступает класс ArrayList, который получает данные в методе setInitialData. Каждому добавляемому объекту Name в списке передается имя и фамилия.

При создании адаптера ему передается ранее созданный ресурс разметки list\_item.xml и список name (Рисунок 23).



**Рисунок 23 – Отображение списка имён**

Однако этот адаптер имеет один очень большой минус – при прокрутке в `ListView`, если в списке очень много объектов, то для каждого элемента, когда он попадет в зону видимости, будет повторно вызываться метод `getView`, в котором будет заново создаваться новый объект `View`. Соответственно будет увеличиваться потребление памяти и снижаться производительность.

Поэтому дальше оптимизируем код `NameAdapter`, изменив его метод `getView` (Рисунок 24).

```

public View getView(int position, View convertView, ViewGroup parent)
{
    LayoutInflater inflater = (LayoutInflater)getApplicationContext().getSystemService (Context.LAYOUT_INFLATER_SERVICE);
    ViewHolder viewHolder;
    //Проверяем, создавалась ли разметка для этого элемента ранее
    if(convertView==null){
        convertView = inflater.inflate(this.layout, parent, false);
        //создаем объект ViewHolder, который сохраняем в тег в
        convertView:
        viewHolder = new ViewHolder(convertView);
        convertView.setTag(viewHolder);
    }
    else{
        //получаем ViewHolder из тега
        viewHolder = (ViewHolder) convertView.getTag();
    }
    Name name = names.get(position);
    //во ViewHolder устанавливаются значения из объекта
    viewHolder.nameView.setText(name.getName());
    viewHolder.surnameView.setText(name.getSurname());
    return convertView;
}
3 usages
private class ViewHolder {
    2 usages
    final TextView nameView, surnameView;
    1 usage
    ViewHolder(View view){
        nameView = view.findViewById(R.id.name);
        surnameView = view.findViewById(R.id.surname);
    }
}

```

**Рисунок 24 – Оптимизация метод getView класса NameAdapter**

Для хранения ссылок на используемые элементы TextView определен внутренний приватный класс ViewHolder, который в конструкторе получает объект View, содержащий TextView.

В методе getView, если convertView равен null (то есть если ранее для объекта не создана разметка) создаем объект ViewHolder, который сохраняем в тег в convertView. Если же разметка для объекта в ListView уже ранее была создана, то обратно получаем ViewHolder из тега. Затем также для TextView во ViewHolder устанавливаются значения из объекта Name. И теперь ListView особенно при больших списках будет работать плавнее и производительнее.

## 1.5 RecyclerView

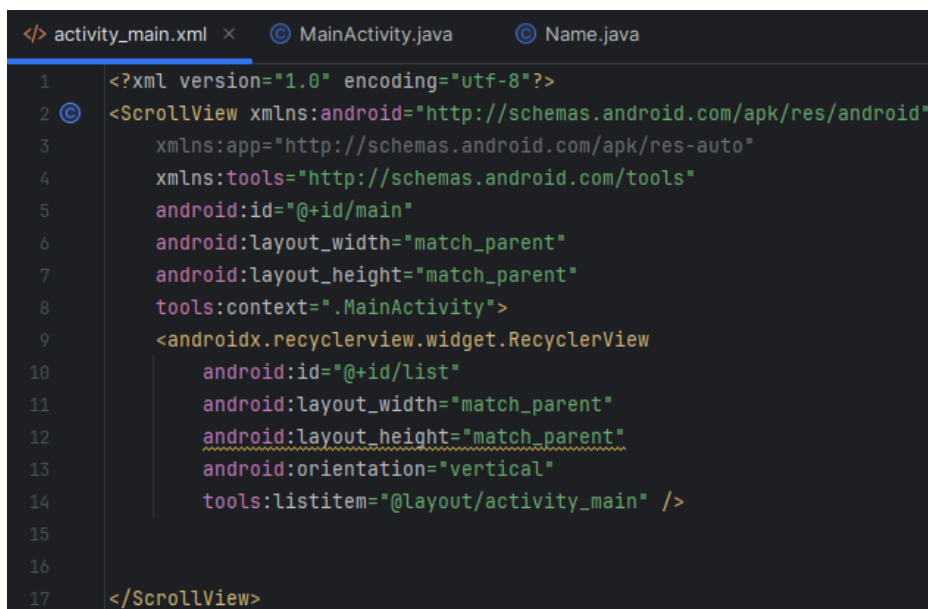
Использование ListView в Android-разработке долгое время было стандартным способом для отображения списка элементов. Однако с появлением

RecyclerView, многие разработчики перешли к его использованию из-за ряда преимуществ и улучшений в производительности и гибкости.

RecyclerView — это более продвинутый и гибкий компонент для отображения списков данных в Android, представленный в Android Support Library для обеспечения улучшенной производительности и большей гибкости по сравнению с ListView. Он предназначен для отображения больших наборов данных, при этом оптимизируя использование памяти путем переиспользования элементов списка. RecyclerView также предоставляет более легкий способ для отображения данных в списках и сетках с возможностью настройки анимаций и раскладок.

Для начала определяем то, как будет выглядеть разметка отдельного элемента списка. Для этого создадим его layout в отдельном xml файле: list\_item.xml.

Затем создаем элемент RecyclerView в пользовательском интерфейсе в файле activity\_main.xml (Рисунок 25).



```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/list"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        tools:listitem="@layout/activity_main" />
</ScrollView>
```

**Рисунок 25 – Описание элемента RecyclerView в файле activity\_main.xml**

Прежде чем наполнить RecyclerView данными, необходимо создать отдельный класс для обработки. Как и в случае с ListView, для вывода сложных объектов в RecyclerView необходимо определить свой адаптер (Рисунок 26).

```

public class SimpleAdapter extends
    RecyclerView.Adapter<SimpleAdapter.ViewHolder> {
    3 usages
    private List<String> items;
    no usages
    public SimpleAdapter(List<String> items) {
        this.items = items;
    }
    @NonNull
    @Override
    public SimpleAdapter.ViewHolder onCreateViewHolder(@NonNull
        ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(parent.getContext()).inflate
            (R.layout.list_item, parent, false);
        return new ViewHolder(view);
    }
    @Override
    public void onBindViewHolder(@NonNull SimpleAdapter.ViewHolder
        holder, int position) {
        String item = items.get(position);
        holder.textView.setText(item);
    }
    @Override
    public int getItemCount() {
        return items.size();
    }
    4 usages 3 related problems
    static class ViewHolder extends RecyclerView.ViewHolder {
        2 usages
        TextView textView;
        1 usage 3 related problems
        ViewHolder(View view) {
            super(view);
            textView = view.findViewById(R.id.item_text);
        }
    }
}

```

**Рисунок 26 – Описание логики класса SimpleAdapter**

Адаптер, который используется в RecyclerView, должен наследоваться от абстрактного класса RecyclerView.Adapter. Этот класс определяет три метода:

- onCreateViewHolder: возвращает объект ViewHolder, который будет хранить данные по одному объекту;
- onBindViewHolder: выполняет привязку объекта ViewHolder к объекту элемента по определенной позиции в списке;
- getItemCount: возвращает количество объектов в списке.



Для хранения данных в классе адаптера определен статический класс ViewHolder, который использует определенные в list\_item.xml элементы управления.

Далее наполняем RecyclerView элементами через код (Рисунок 27).

```
public class MainActivity extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        //Находим элемент RecyclerView  
        RecyclerView recyclerView = findViewById(R.id.list);  
        //Устанавливает макет отображения - горизонтально  
        recyclerView.setLayoutManager(new LinearLayoutManager(context, this));  
        // Пример списка строк  
        List<String> items = Arrays.asList("Элемент 1", "Элемент 2",  
            "Элемент 3");  
        //Создаем адаптер  
        SimpleAdapter adapter = new SimpleAdapter(items);  
        //Устанавливаем для списка адаптер  
        recyclerView.setAdapter(adapter);  
    }  
}
```

Рисунок 27 – Наполнение RecyclerView через код

Также продемонстрируем итоговый вариант приложения при подобном наполнении RecyclerView (Рисунок 28).

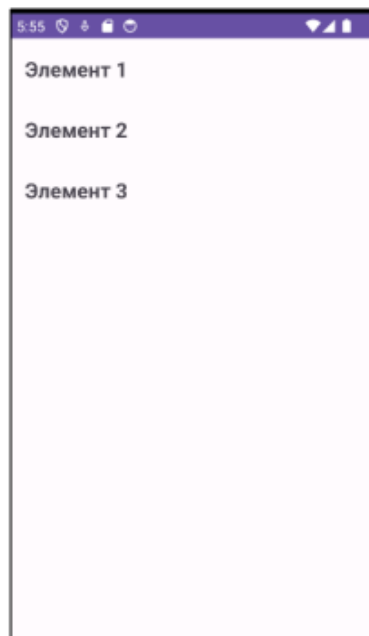


Рисунок 28 – Итоговый вид приложения

Для RecyclerView следует устанавливать атрибут layoutManager, который указывает на тип менеджера компоновки. Это можно сделать в разметки или

программно. Менеджер компоновки представляет объект, который представлен классом `LayoutManager`. По умолчанию библиотека `RecyclerView` предоставляет три реализации данного менеджера:

- `LinearLayoutManager`: упорядочивает элементы в виде списка с одной колонкой;
- `GridLayoutManager`: упорядочивает элементы в виде грида со столбцами и строками. Грид может упорядочивать элементы по горизонтали (горизонтальный грид) или по вертикали (вертикальный грид);
- `StaggeredGridLayoutManager`: аналогичен `GridLayoutManager`, однако не требует установки для каждого элемента в строке имели одну и ту же высоту (для вертикального грида) и одну и ту же ширину (для горизонтального грида);

Основные особенности `RecyclerView`:

- лучшая производительность: `RecyclerView` использует концепцию `ViewHolder` для минимизации вызовов метода `findViewById()`, что улучшает производительность прокрутки;
- больше контроля над раскладкой элементов: `RecyclerView` поддерживает `LayoutManager`, который определяет расположение элементов в списке или сетке, а также другие кастомные раскладки;
- встроенные анимации: `RecyclerView` предлагает встроенные анимации для операций добавления, удаления и перемещения элементов;
- гибкость: благодаря отдельному компоненту `Adapter`, `RecyclerView` легко адаптировать под различные типы данных и конфигурации отображения.

`RecyclerView` довольно гибок и при этом `ListView` представляет из себя более простую, но менее функциональную версию `RecyclerView` (Рисунок 29).

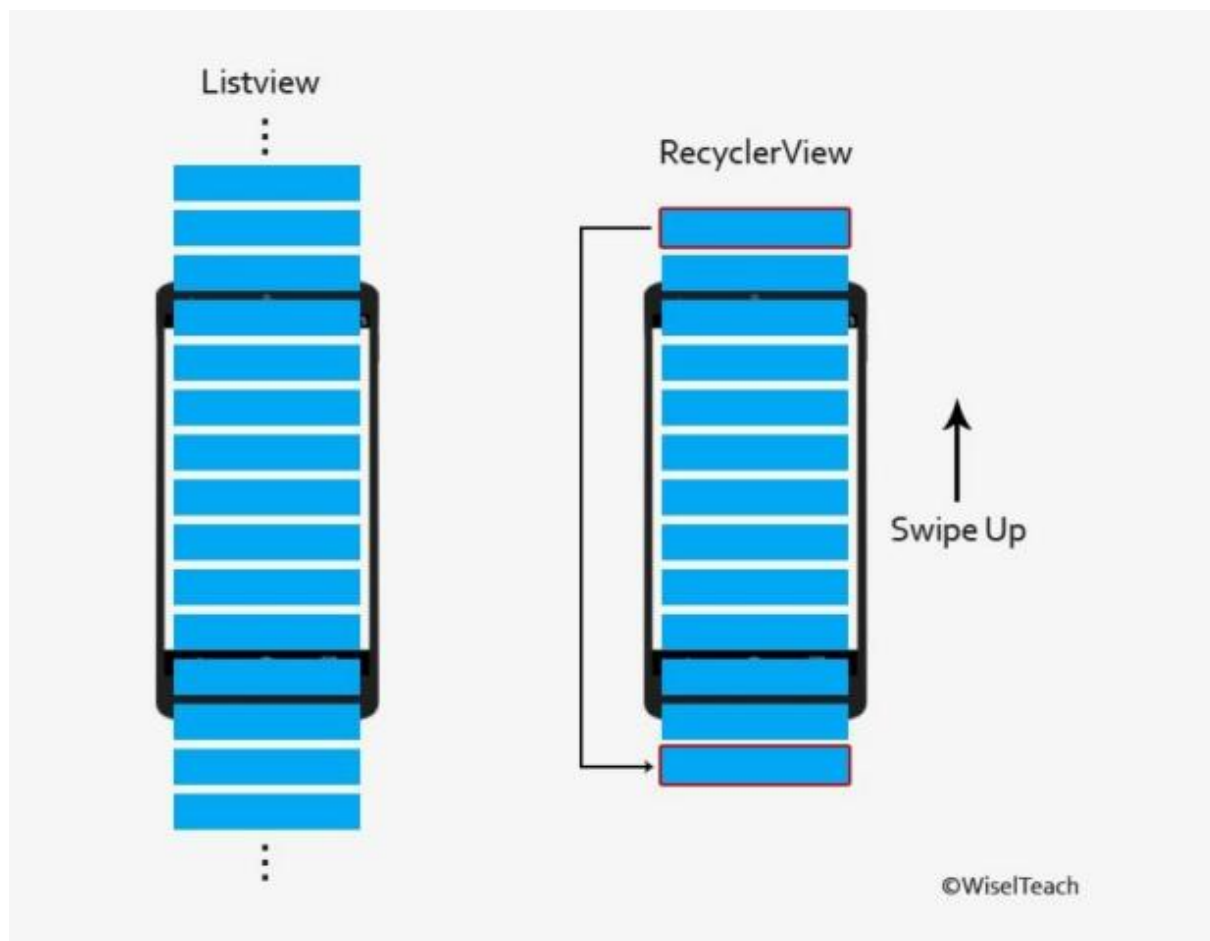


Рисунок 29 – Отображение отличий между `ListView` и `RecyclerView`

## 2 ПРАКТИЧЕСКОЕ ЗАДАНИЕ

### 2.1 ListView

#### 2.1.1 Создание проекта

Создадим проект и назовем его Pract5 (Рисунок 30).

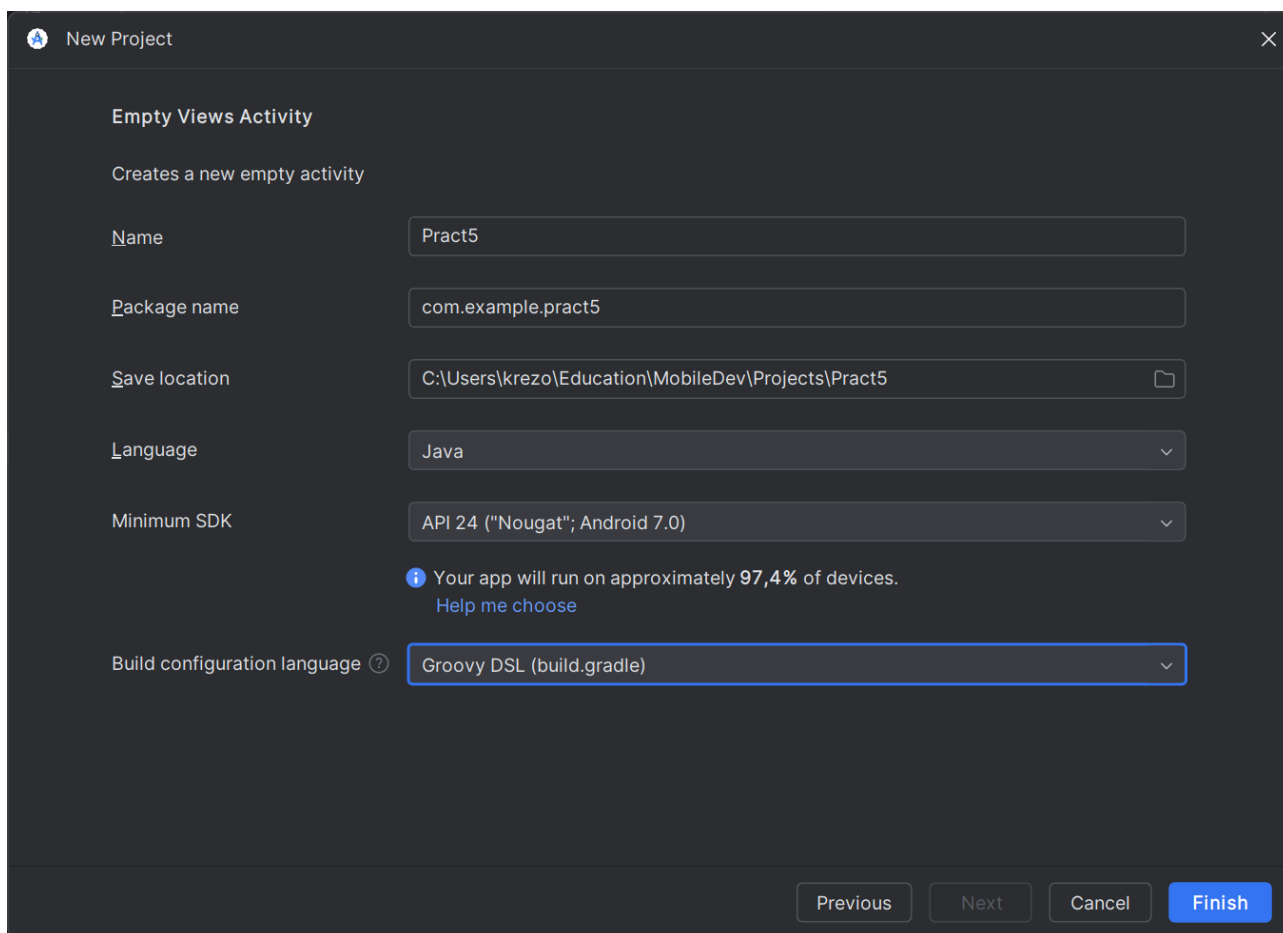



Рисунок 30 – Создание проекта

#### 2.1.2 Разметка

В файле ресурсов strings.xml добавим необходимые списки строк (Рисунок 31).



```
<string-array name="main_products">
    <item>Яблоки</item>
    <item>Апельсины</item>
    <item>Мандарины</item>
</string-array>

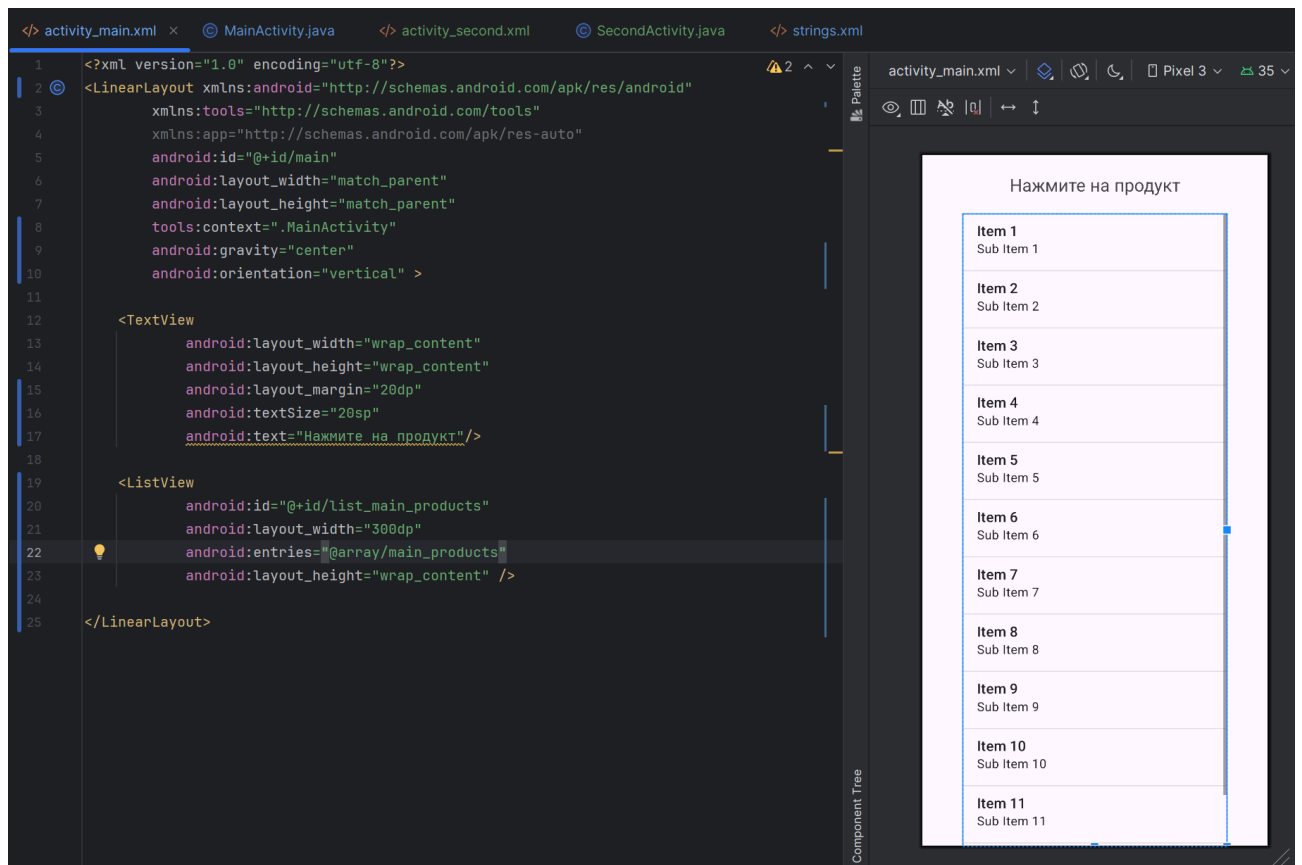
<string-array name="second_apples">
    <item>Голден</item>
    <item>Гала</item>
    <item>Черный принц</item>
    <item>Карамелька</item>
    <item>Спартан</item>
    <item>Фуджи</item>
</string-array>

<string-array name="second_oranges">
    <item>Гамлин</item>
    <item>Верна</item>
    <item>Салустiana</item>
    <item>Навел</item>
    <item>Вашингтон Навел</item>
    <item>Навел Лате</item>
</string-array>

<string-array name="second_tangerines">
    <item>Клементин</item>
    <item>Муркотт</item>
    <item>Надоркотт</item>
    <item>Уншиу</item>
    <item>Шива-микан</item>
</string-array>
```

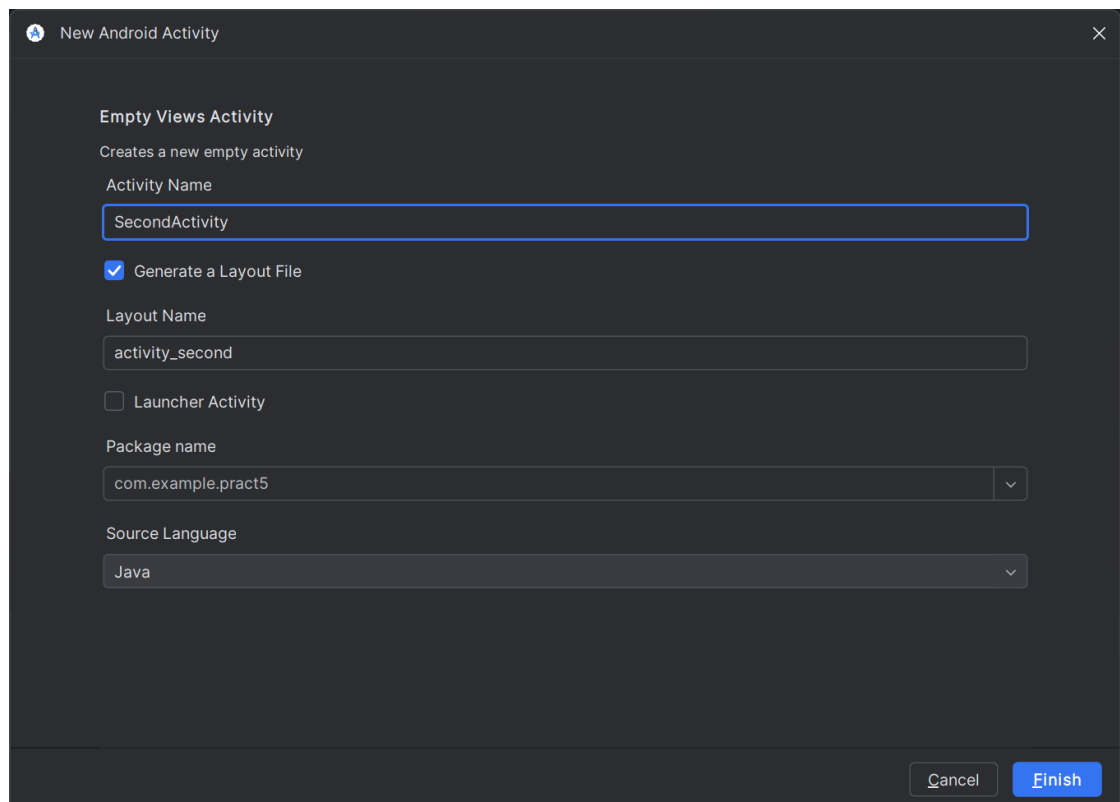
Рисунок 31 – Строковые ресурсы списков в файле strings.xml

В файле разметки activity\_main.xml добавим список ListView и зададим ему содержимое из ресурса "main\_products" (Рисунок 32).



**Рисунок 32 – Файл разметки activity\_main.xml**

Создадим класс Activity и назовем его SecondActivity (Рисунок 33).



**Рисунок 33 – Создание SecondActivity**

В файле разметки activity\_second.xml добавим список для различных сортов фруктов, а также кнопку и поле для ввода текста для добавления нового элемента списка. Этот список будет заполняться программно (Рисунок 34).

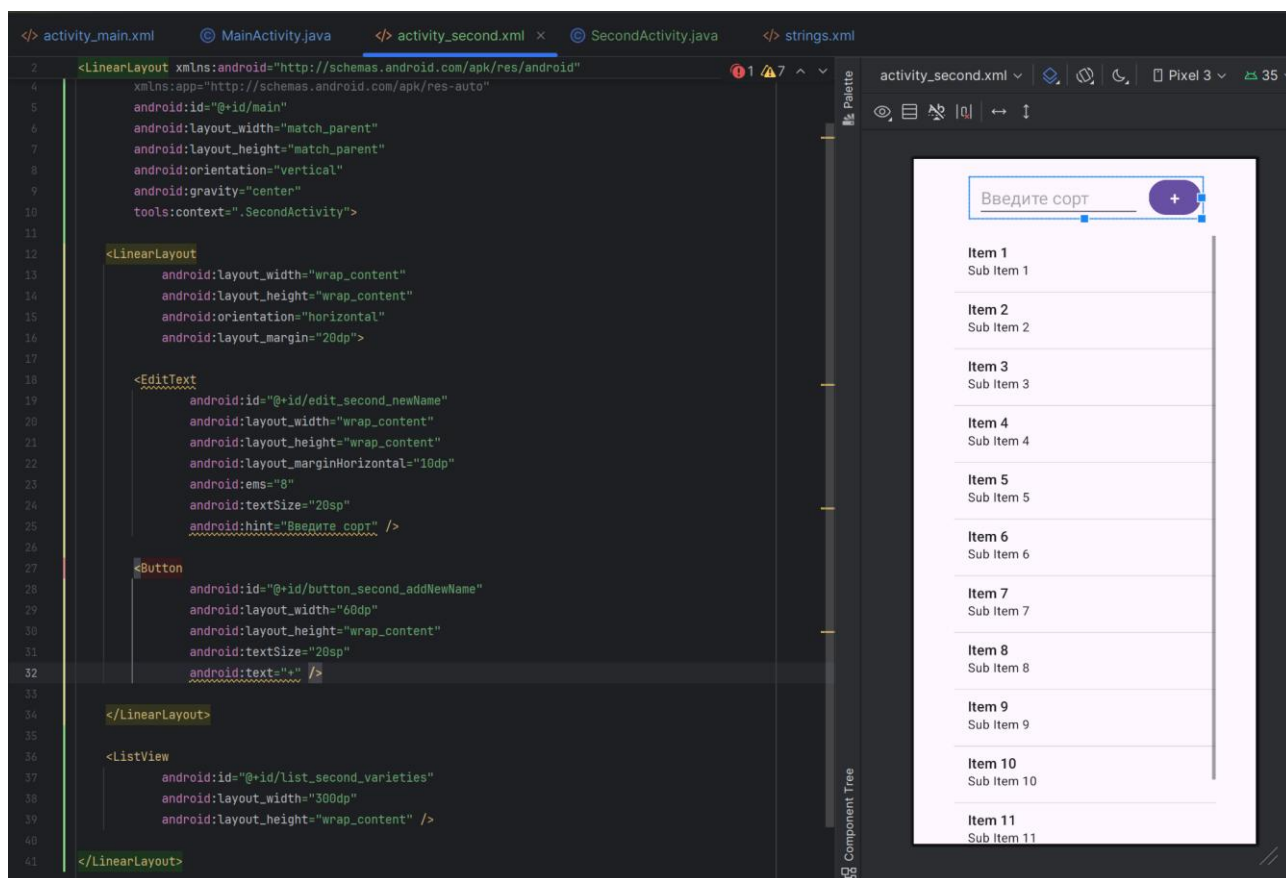


Рисунок 34 – Файл разметки activity\_second.xml

### 2.1.3 Реализация логики списка

В методе onCreate() класса MainActivity сделаем переход к SecondActivity с передачей индекса нажатого фрукта. Для этого установим обработчик нажатия на элемент списка (Рисунок 35).

```

@Override
protected void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    EdgeToEdge.enable( $this$enableEdgeToEdge: this);
    setContentView(R.layout.activity_main);
    ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) ->
    {
        Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
        v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
        return insets;
    });

    ListView productsView = findViewById(R.id.list_main_products);

    productsView.setOnItemClickListener(new AdapterView.OnItemClickListener()
    {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position, long id)
        {
            Intent intent = new Intent(view.getContext(), SecondActivity.class);
            intent.putExtra( name: "ARGUMENT_LIST_NUMBER", position);
            startActivity(intent);
        }
    });
}

```

**Рисунок 35 – Метод onCreate() класса MainActivity**

В методе onCreate() класса SecondActivity получим переданный из MainActivity индекс элемента и в соответствии с этим индексом выберем какой список отображать, после чего отобразим его с помощью ListView и ArrayAdapter (Рисунок 36).

```

Bundle args = getIntent().getExtras();
if (args == null) return;

int listNumber = args.getInt( key: "ARGUMENT_LIST_NUMBER");
String[] varieties;
switch (listNumber)
{
    case 0 :
        varieties = getResources().getStringArray(R.array.second_apples);
        break;
    case 1 :
        varieties = getResources().getStringArray(R.array.second_oranges);
        break;
    case 2 :
        varieties = getResources().getStringArray(R.array.second_tangerines);
        break;
    default:
        return;
}

ArrayList<String> varietiesList = new ArrayList<>(Arrays.asList(varieties));

ArrayAdapter<String> adapter = new ArrayAdapter<>( context: this, android.R.layout.simple_list_item_1, varietiesList);
ListView varietiesView = findViewById(R.id.list_second_varieties);
varietiesView.setAdapter(adapter);

```

**Рисунок 36 – Обработка переданных данных в методе onCreate() класса SecondActivity**



Далее в этом же методе реализуем функционал добавления и удаления элементов списка. Добавление элемента в список происходит по нажатию на кнопку "+", после чего в список добавляется новый элемент, который берется из текстового поля ввода. Для удаления элемента из списка использован обработчик долгого нажатия на элемент списка (Рисунок 37).

```
EditText editNewName = findViewById(R.id.edit_second_newName);
Button buttonAddNewName = findViewById(R.id.button_second_addNewName);
buttonAddNewName.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        String newName = editNewName.getText().toString();
        if (newName.isEmpty()) return;

        adapter.add(newName);
        adapter.notifyDataSetChanged();
    }
});

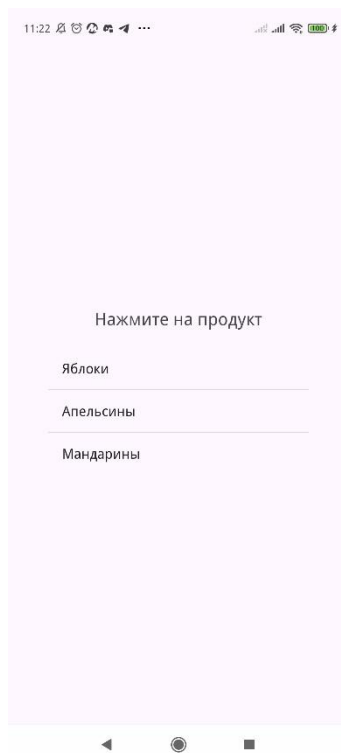
varietiesView.setOnItemLongClickListener(new AdapterView.OnItemLongClickListener()
{
    no usages
    @Override
    public boolean onItemLongClick(AdapterView<?> parent, View view, int position, long id)
    {
        varietiesList.remove(position);
        adapter.notifyDataSetChanged();
        return true;
    }
});
```

Рисунок 37 – Реализация функционала списка в методе onCreate() класса SecondActivity

### 2.1.4 Тестирование

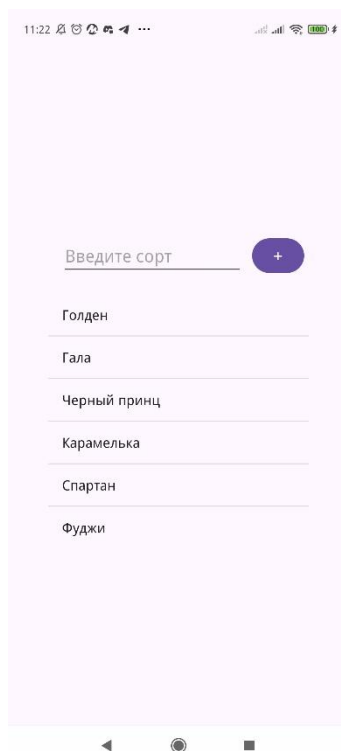
Протестируем работу списка фруктов и их сортов.

На рисунке 38 представлена начальная страница приложения.



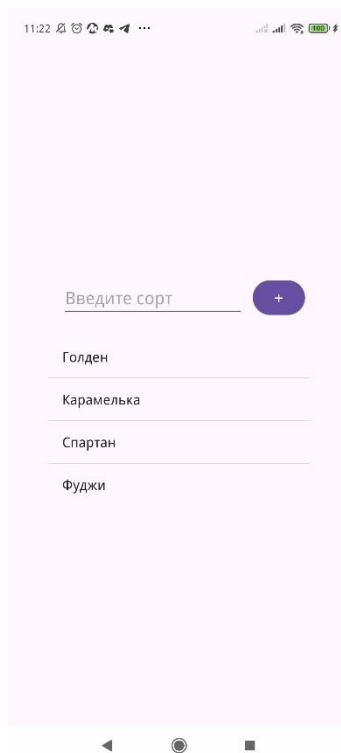
**Рисунок 38 – Начальная страница приложения**

При нажатии на «Яблоки» происходит переход на страницу со списком яблок (Рисунок 39).



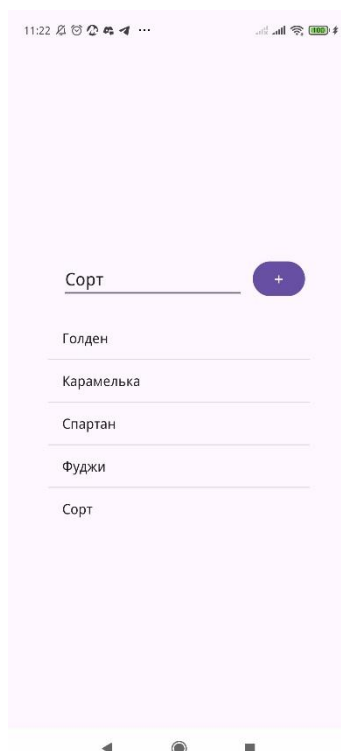
**Рисунок 39 – Сорта яблок до изменения**

При долгом зажатии происходит удаление элемента (Рисунок 40).



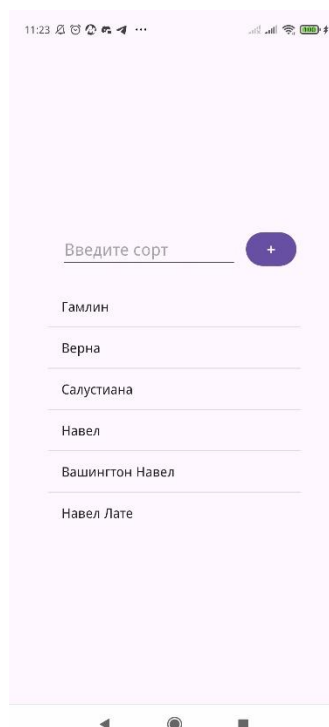
**Рисунок 40 – Сорта яблок после удаления элементов**

При вводе названия нового элемента и нажатии на кнопку происходит добавление нового элемента в список (Рисунок 41).



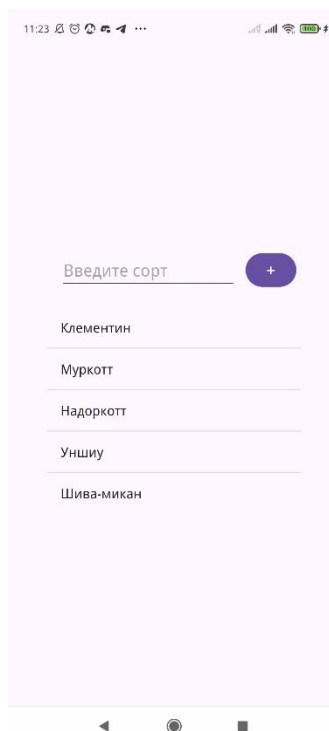
**Рисунок 41 – Сорта яблок после добавления элемента**

При нажатии на «Апельсины» происходит переход на страницу апельсинов (Рисунок 42)



**Рисунок 42 – Сорта апельсинов**

При нажатии на «Мандарины» происходит переход на страницу мандаринов (Рисунок 43)



**Рисунок 43 – Сорта мандаринов**

## 2.2 RecyclerView

### 2.2.1 Разметка

Создадим новый файл разметки элемента списка `item_third.xml` и определим в нем текстовое поле и картинку (Рисунок 44).

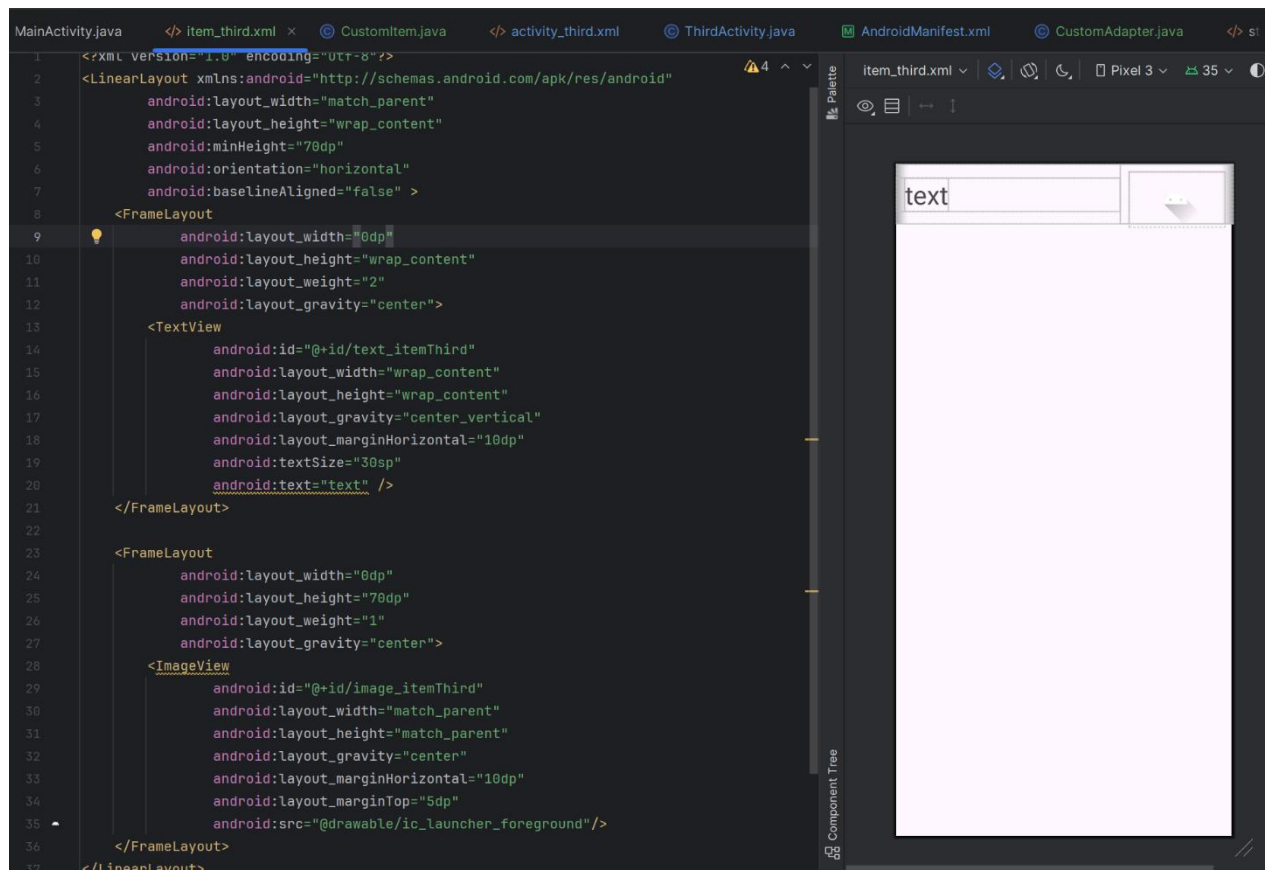
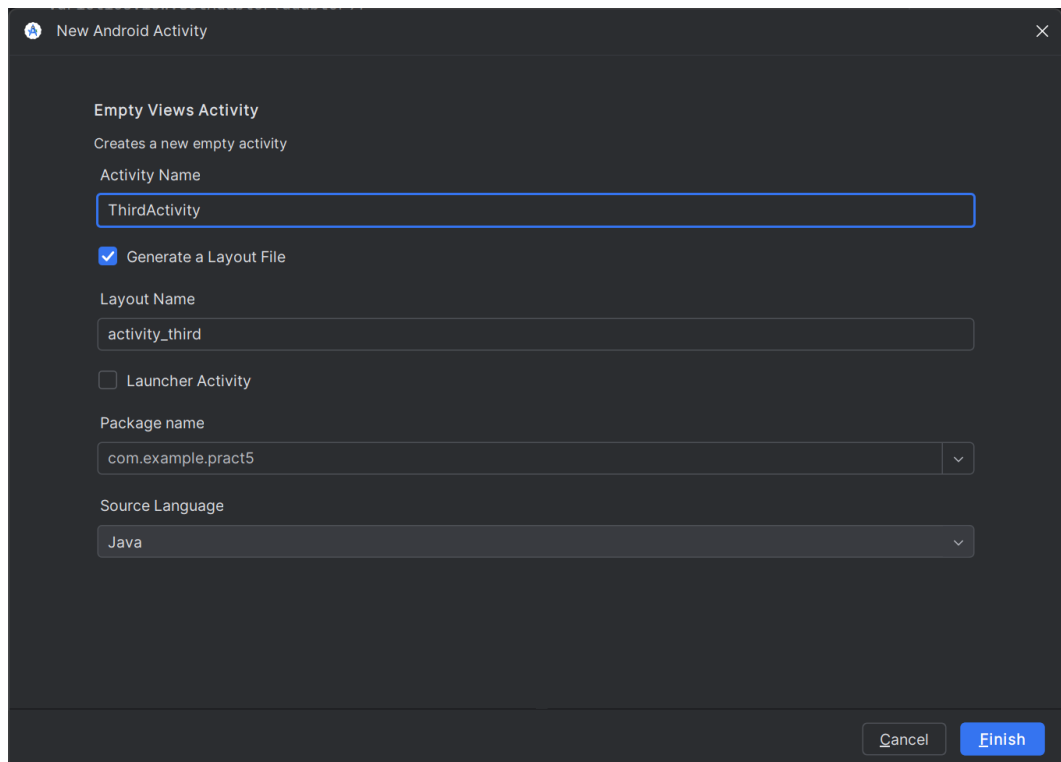


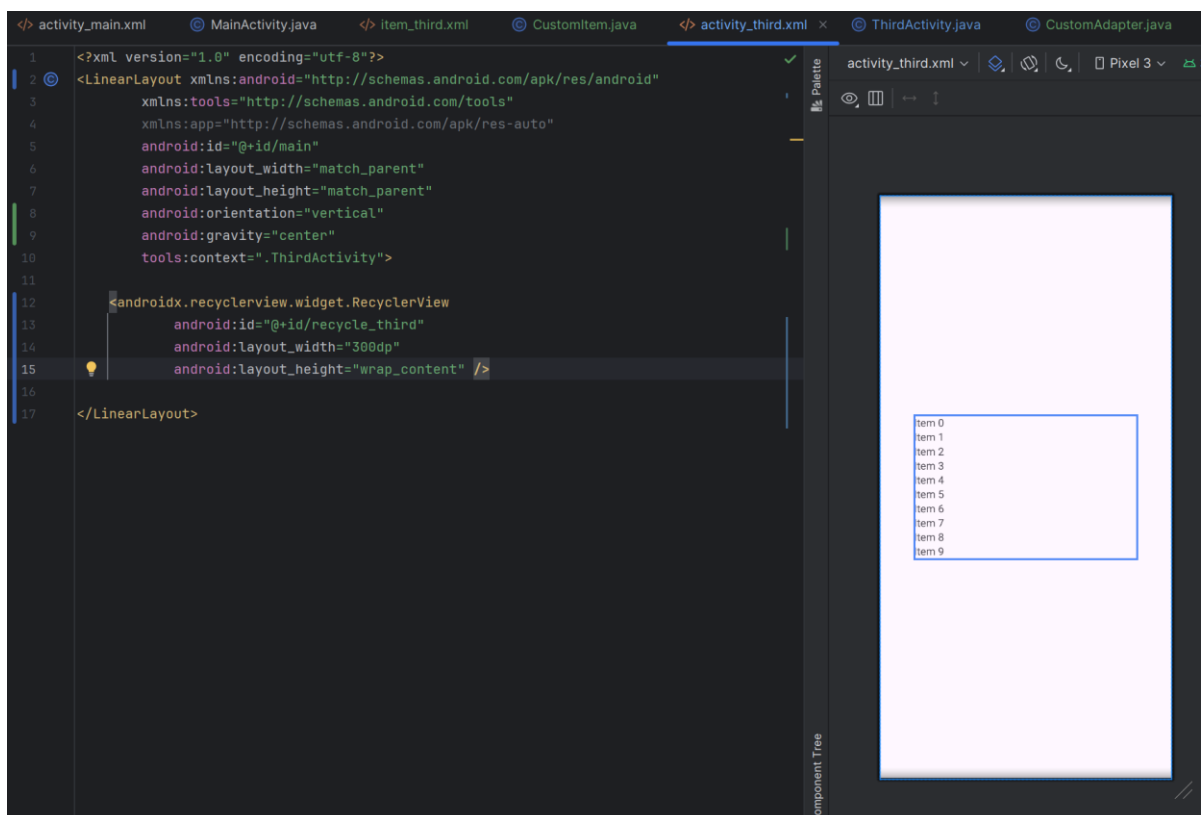
Рисунок 44 – Файл разметки `item_third.xml`

Создадим новую Activity и назовем ее `ThirdActivity` (Рисунок 45).



**Рисунок 45 – Создание ThirdActivity**

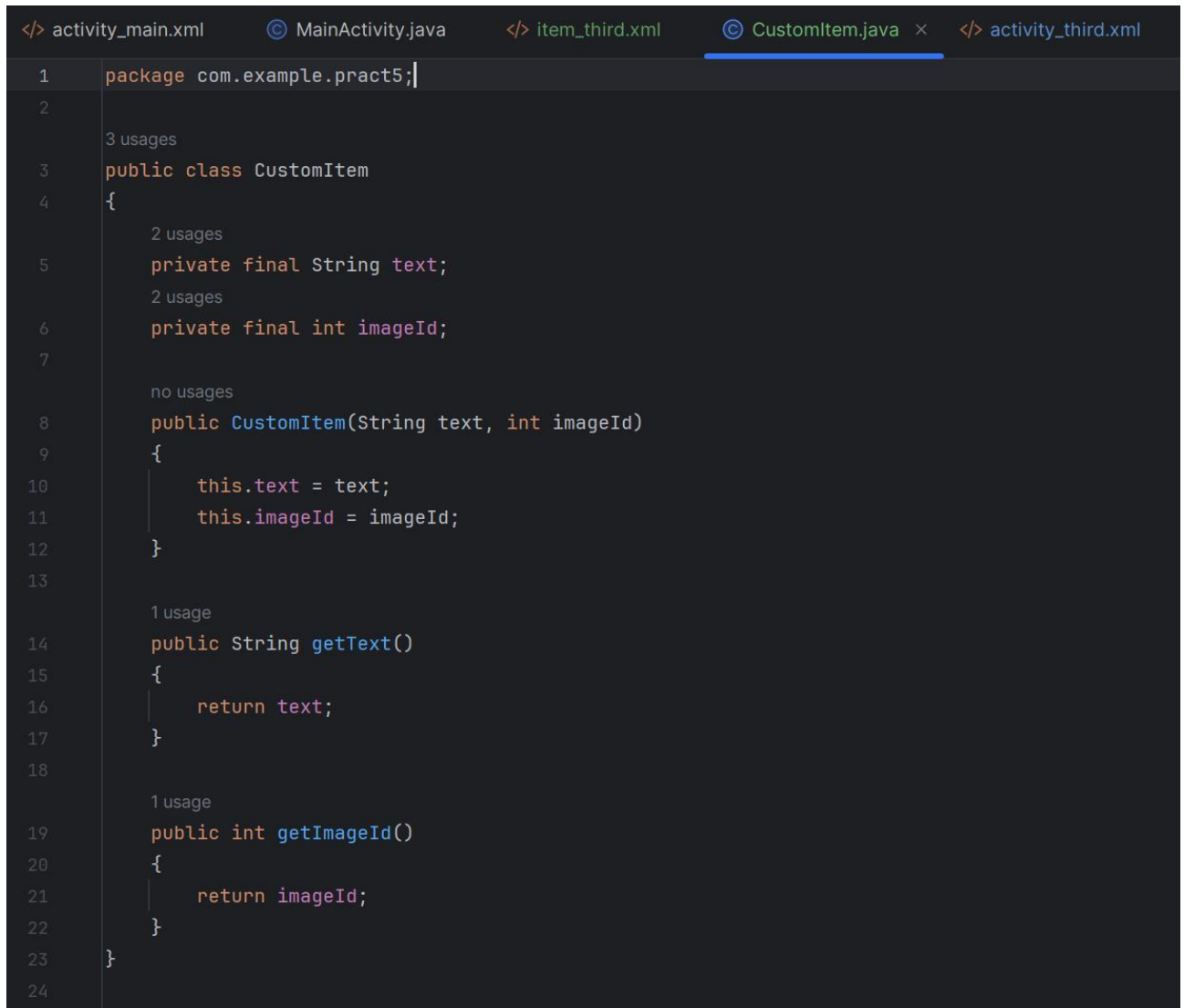
В файле разметки activity\_third.xml добавим компонент RecyclerView (Рисунок 46).



**Рисунок 46 – Файл разметки activity\_third.xml**

## 2.2.2 Реализация адаптера

Сначала создадим класс CustomItem элемента списка, содержимое которого будет отображаться в RecyclerView (Рисунок 47).



```
</> activity_main.xml  MainActivity.java  </> item_third.xml  CustomItem.java x  </> activity_third.xml
1  package com.example.pract5;|
2
3  3 usages
4  public class CustomItem
5  {
6      2 usages
7      private final String text;
8      2 usages
9      private final int imageId;
10
11      no usages
12      public CustomItem(String text, int imageId)
13      {
14          this.text = text;
15          this.imageId = imageId;
16      }
17
18      1 usage
19      public String getText()
20      {
21          return text;
22      }
23
24      1 usage
25      public int getImageId()
26      {
27          return imageId;
28      }
29  }
```

Рисунок 47 – Описание класса CustomItem

Создадим класс CustomAdapter, наследующий от класса RecyclerView.Adapter<CustomAdapter.ViewHolder>, где CustomAdapter.ViewHolder – класс, вложенный в CustomAdapter.

CustomAdapter.ViewHolder хранит в себе уникальные элементы View, которые требуется обновлять при повторном использовании элементов списка, вышедших за границы экрана (Рисунок 48).

```

4 usages
public static class ViewHolder extends RecyclerView.ViewHolder
{
    2 usages
    private final TextView textView;
    2 usages
    private final ImageView imageView;
    1 usage
    public ViewHolder(@NonNull View itemView)
    {
        super(itemView);
        textView = itemView.findViewById(R.id.text_itemThird);
        imageView = itemView.findViewById(R.id.image_itemThird);
    }
}

```

**Рисунок 48 – Описание класса CustomAdapter.ViewHolder**

Опишем необходимые методы из родительского класса RecyclerView.Adapter и добавим конструктор класса CustomAdapter (Рисунок 49).

```

public class CustomAdapter extends RecyclerView.Adapter<CustomAdapter.ViewHolder>
{
    3 usages
    private final List<CustomItem> items;

    no usages
    public CustomAdapter(List<CustomItem> items)
    {
        this.items = items;
    }

    @NonNull
    @Override
    public CustomAdapter.ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType)
    {
        View view = LayoutInflater.from(parent.getContext())
            .inflate(R.layout.item_third, parent, attachToRoot: false);
        return new ViewHolder(view);
    }

    @Override
    public void onBindViewHolder(@NonNull CustomAdapter.ViewHolder holder, int position)
    {
        CustomItem item = items.get(position);

        holder.textView.setText(item.getText());
        holder.imageView.setImageResource(item.getImageId());
    }

    @Override
    public int getItemCount()
    {
        return items.size();
    }
}

```

**Рисунок 49 – Описание класса CustomAdapter**



### 2.2.3 Использование в Activity

Воспользуемся новым Adapter классом в методе onCreate() класса ThirdActivity. Для этого требуется создать список с необходимыми элементами и передать его в конструктор при создании объекта класса CustomAdapter. После этого нужно найти соответствующий RecyclerView, установить в нем LayoutManager и CustomAdapter (Рисунок 50).

```
@Override
protected void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    EdgeToEdge.enable(this);
    setContentView(R.layout.activity_third);
    ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) ->
    {
        Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
        v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
        return insets;
    });

    RecyclerView recycler = findViewById(R.id.recycler_third);
    recycler.setLayoutManager(new LinearLayoutManager(context, this));

    ArrayList<CustomItem> items = new ArrayList<>(List.of(
        new CustomItem(text: "Яйцо", R.drawable.image_third_egg1),
        new CustomItem(text: "Другое Яйцо", R.drawable.image_third_egg2),
        new CustomItem(text: "Большое Яйцо", R.drawable.image_third_egg3),
        new CustomItem(text: "Яйцо", R.drawable.image_third_egg1),
        new CustomItem(text: "Другое Яйцо", R.drawable.image_third_egg2),
        new CustomItem(text: "Большое Яйцо", R.drawable.image_third_egg3),
        new CustomItem(text: "Яйцо", R.drawable.image_third_egg1),
        new CustomItem(text: "Другое Яйцо", R.drawable.image_third_egg2),
        new CustomItem(text: "Большое Яйцо", R.drawable.image_third_egg3)
    ));

    CustomAdapter adapter = new CustomAdapter(items);
    recycler.setAdapter(adapter);
}
```

Рисунок 50 – Реализация RecyclerView с использованием класса CustomAdapter

### 2.2.4 Тестирование

Протестируем работу списка RecyclerView (Рисунок 51).



**Рисунок 51 – Список RecyclerView**

Пролистнем список в конец, чтобы убедиться в его работе (Рисунок 52).



**Рисунок 52 – Конец списка RecyclerView**

Тестирование прошло успешно.

## 2.3 ScrollView

Сначала создадим новую Activity и назовем ее FourthActivity (Рисунок 53).

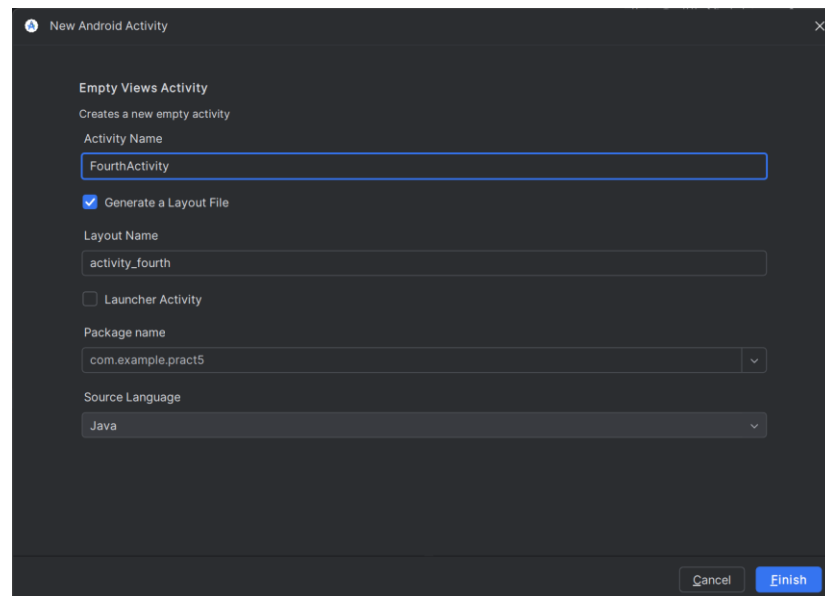


Рисунок 53 – Создание класса FourthActivity

В файле разметки activity\_fourth.xml расположим элемент ScrollView и добавим в него несколько текстовых полей. Под ScrollView расположим поле для ввода текста (Рисунок 54).

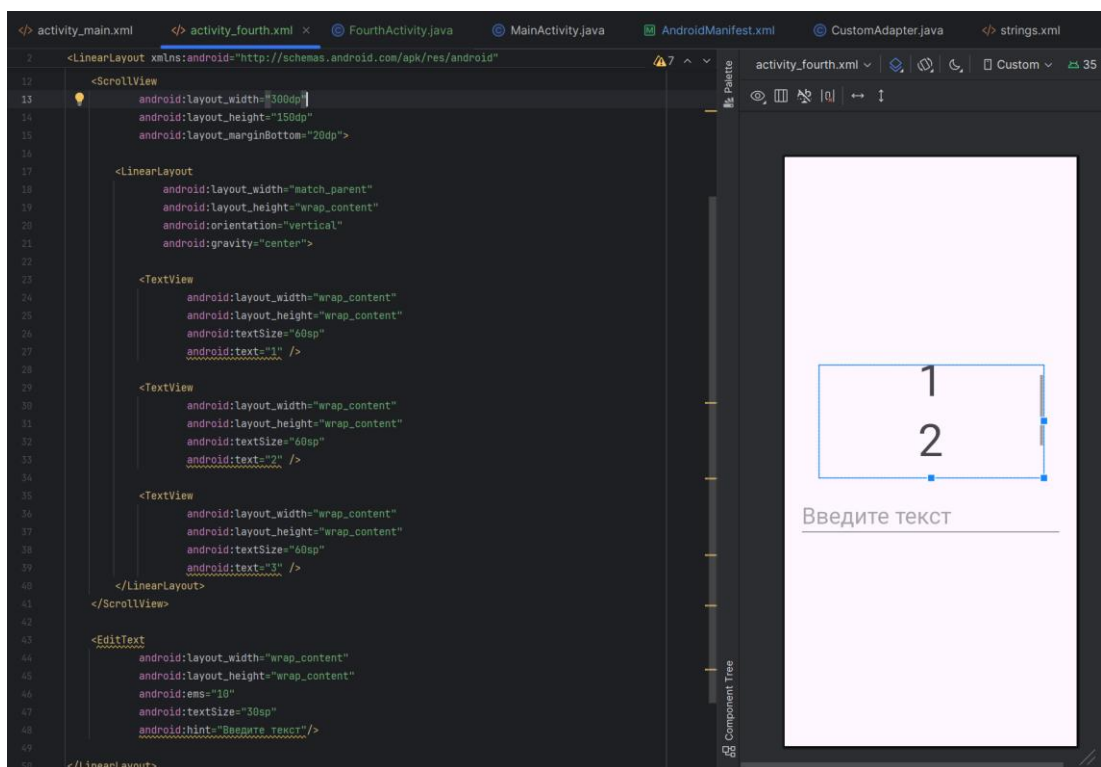
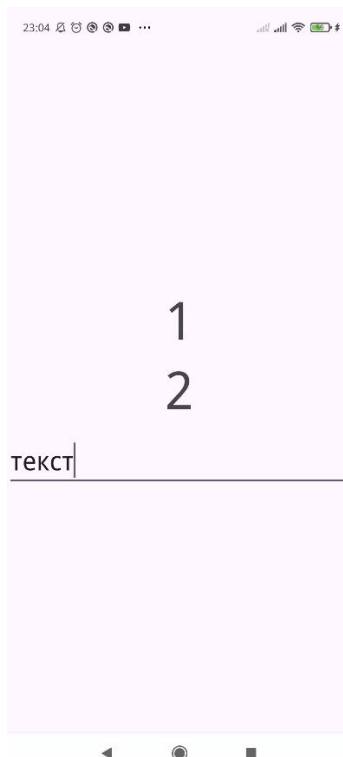


Рисунок 54 – Файл разметки activity\_fourth.xml

Протестируем работу ScrollView (Рисунок 55).



**Рисунок 55 – ScrollView до прокручивания**

Прокрутим ScrollView в конец (Рисунок 56).



**Рисунок 56 – ScrollView после прокручивания**

Все работает.

## 2.4 Spinner

В файл разметки activity\_fourth.xml добавим элемент Spinner и установим атрибут "entries=@array/main\_products", чтобы задать элементы выпадающего списка (Рисунок 57).



Рисунок 57 – Разметка элемента Spinner

После этого протестируем работу элемента (Рисунок 58).

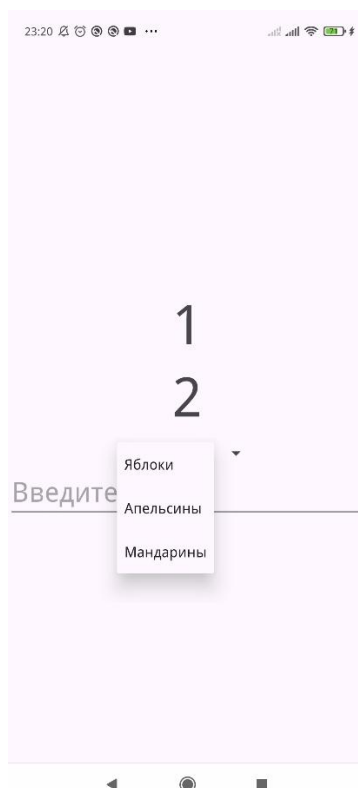


Рисунок 58 – Тестирование элемента Spinner

Тестирование прошло успешно.

## **ЗАКЛЮЧЕНИЕ**

В ходе проделанной работы было проведено ознакомление с ListView и RecyclerView, а также другими инструментами отображения списков. Полученные знания были закреплены путём выполнения практического задания.