

## Часть 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.

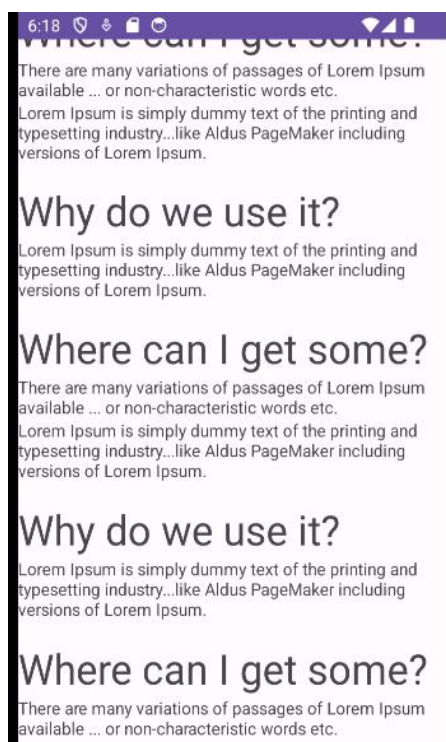
Для использования ScrollView необходимо добавить данный контейнер в разметку пользовательского интерфейса и разместить в нем необходимые элементы.

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        <!-- Другие дочерние элементы -->

    </LinearLayout>
</ScrollView>
```



Так же важно отметить, что для отображения больших списков данных лучше использовать RecyclerView или ListView, так как они оптимизированы для эффективного отображения больших коллекций, управляя повторным использованием и отрисовкой только видимых элементов.

## Часть 2. ListView

Android представляет широкую палитру элементов, которые представляют списки. Все они являются наследниками класса **android.widget.AdapterView**. Это такие виджеты как ListView, GridView, Spinner. Они могут выступать контейнерами для других элементов управления.

При работе со списками мы имеем дело с тремя компонентами.

1. Во-первых, это визуальный элемент или виджет, который на экране представляет список (ListView, GridView) и который отображает данные.
2. Во-вторых, это источник данных – массив, объект ArrayList, база данных и т.д., в котором находятся сами отображаемые данные.
3. И, в-третьих, это 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. 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. Ниже приведена разметка данного элемента пользовательского интерфейса.

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

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:

1. **`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`.
2. **`CursorAdapter`**: Подходит для отображения данных, полученных из базы данных. Используется в сочетании с `Cursor`, представляющим результат запроса к базе данных.
3. **`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 просто список с именами и выведем его на экран.

```
public class MainActivity extends AppCompatActivity {

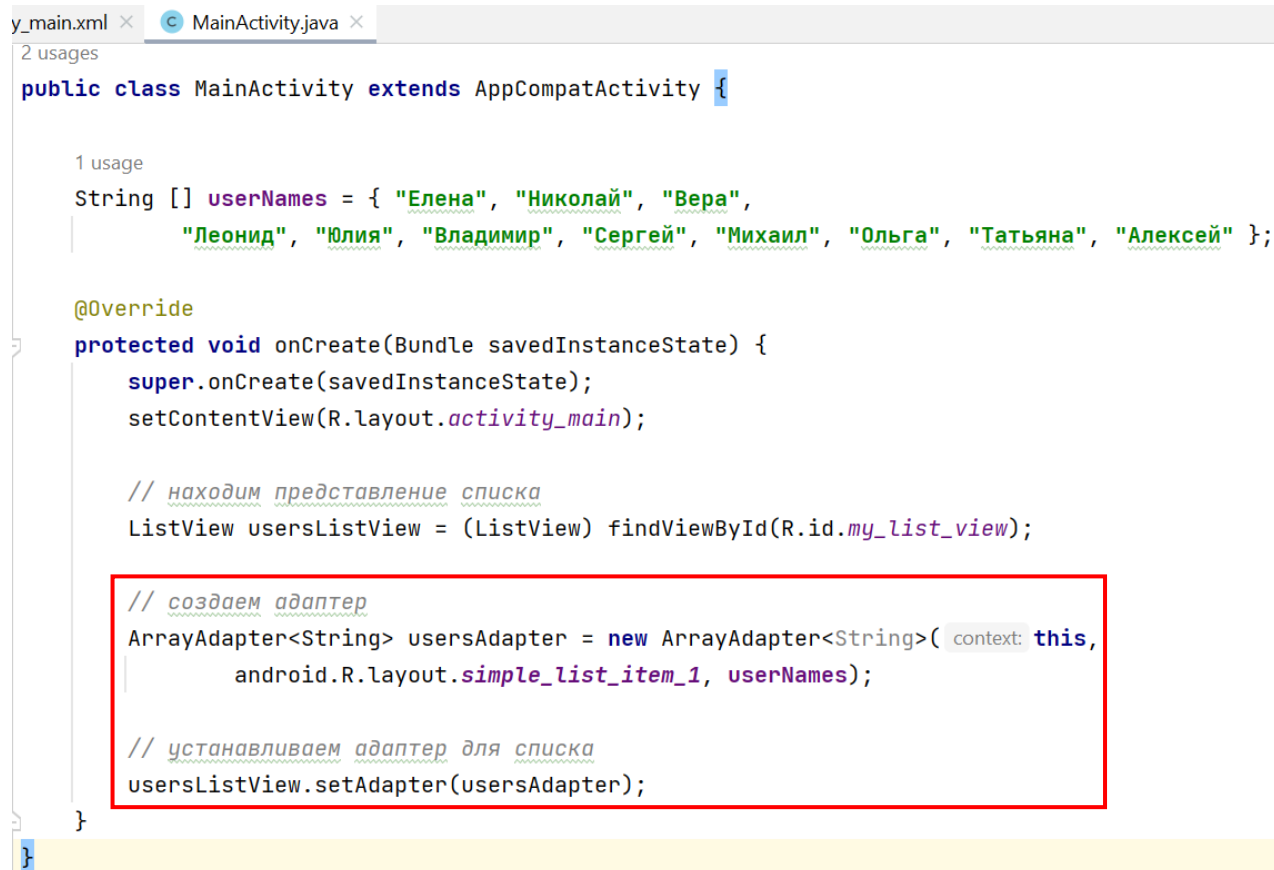
    String[] userNames = { "Елена", "Николай", "Вера", "Леонид", "Юлия",
        "Владимир", "Сергей", "Михаил", "Ольга", "Татьяна", "Алексей" };

    @Override
    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);

        // создаем адаптер
        ArrayAdapter<String> usersAdapter = new ArrayAdapter<String>(this,
        android.R.layout.simple_list_item_1, userNames);
```

```
// устанавливаем адаптер для списка
usersListView.setAdapter(usersAdapter);
}
}
```



The screenshot shows an IDE with two tabs: 'y\_main.xml' and 'MainActivity.java'. The 'MainActivity.java' tab is active, showing the following code:

```
2 usages
public class MainActivity extends AppCompatActivity {

    1 usage
    String [] userNames = { "Елена", "Николай", "Вера",
        "Леонид", "Юлия", "Владимир", "Сергей", "Михаил", "Ольга", "Татьяна", "Алексей" };

    @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);

        // 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);
    }
}
```

A red rectangular box highlights the following code block:

```
// 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);
```

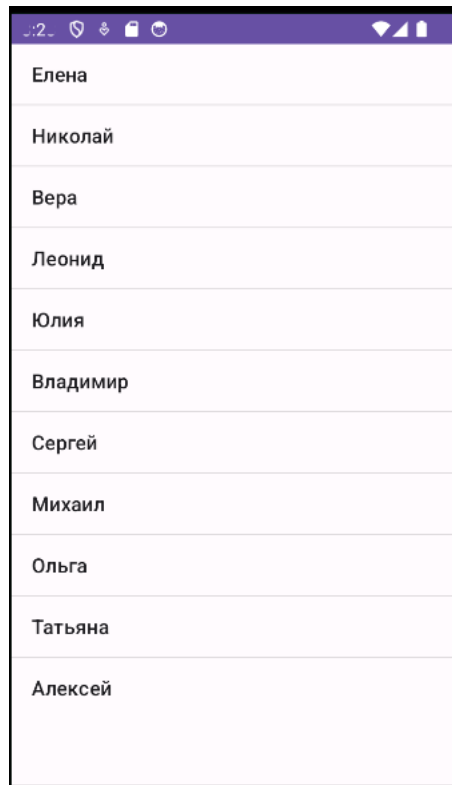
Здесь вначале получаем по id элемент ListView и затем создаем для него адаптер.

Для создания адаптера использовался следующий конструктор `ArrayAdapter<String>(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<T>`.

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

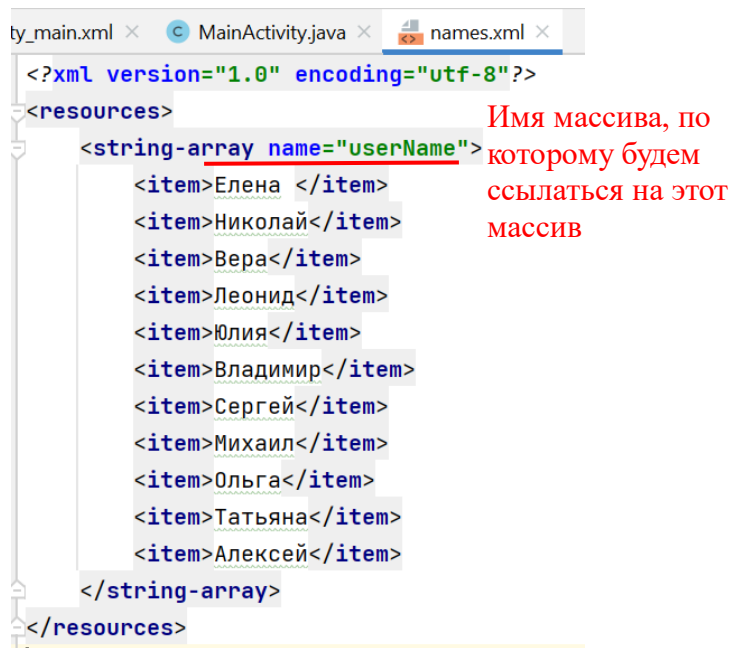
В итоге мы получим следующее отображение:



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.



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**.

```
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);
    }
}
```

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`. 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"
```

Кроме того, что можно просто выводить список элементов, можно также выбирать элементы списка и обрабатывать данный выбор. Для этого необходимо связать список `ListView` с источником данных и закрепим за ним слушатель нажатия на элемент списка внутри метода `OnCreate()`. Чтобы было нагляднее добавим в разметку элемент `TextView` для вывода выбранного элемента. Слушатель будет иметь следующий код:

```
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);
    }
});
```

```

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);
    //Добовляем 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);
        }
    });
}

```

Слушатель для обработки  
выбора элемента списка

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.





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

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

После этого определяется в коде MainActivity обработка выбора элементов списка.

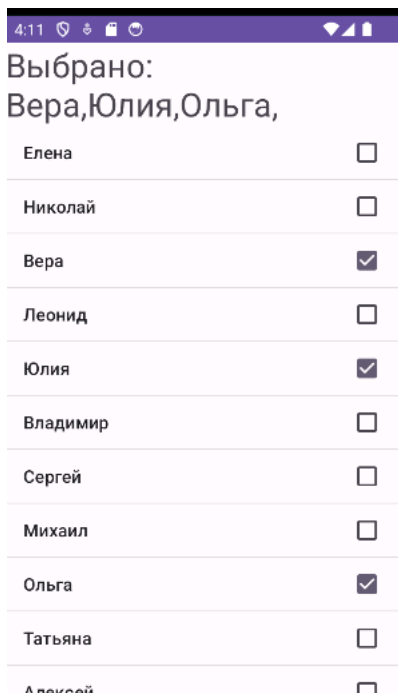
```
ArrayAdapter<String> usersAdapter = new ArrayAdapter<String>( context: this,
    android.R.layout.simple_list_item_multiple_choice, userNames);
usersListView.setAdapter(usersAdapter);

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);
    }
});
```

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.



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**.

```
public class MainActivity extends AppCompatActivity {
    ArrayList<String> users = new ArrayList<String>();
    ArrayList<String> selectedUsers = new ArrayList<String>();
    ArrayAdapter<String> usersAdapter;
    ListView usersListView;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        String[] userNames =
getResources().getStringArray(R.array.userName);
        // добавляем начальные элементы
        Collections.addAll(users, userNames);

        usersListView = (ListView) findViewById(R.id.my_list_view);
        usersAdapter = new ArrayAdapter<String>(this,
            android.R.layout.simple_list_item_multiple_choice,
users);
        usersListView.setAdapter(usersAdapter);

        usersListView.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view,
int position, long id) {
                // получаем нажатый элемент
                String user = usersAdapter.getItem(position);
                if (usersListView.isItemChecked(position))
                    selectedUsers.add(user);
                else
                    selectedUsers.remove(user);
            }
        });
    }
    public void add(View view) {
```

```

        EditText userName = findViewById(R.id.userName);
        String user = userName.getText().toString();
        if(!user.isEmpty()){
            usersAdapter.add(user);
            userName.setText("");
            usersAdapter.notifyDataSetChanged();
        }
    }

    public void remove(View view){
        // получаем и удаляем выделенные элементы
        for(int i=0; i< selectedUsers.size();i++){
            usersAdapter.remove(selectedUsers.get(i));
        }
        // снимаем все ранее установленные отметки
        usersListView.clearChoices();
        // очищаем массив выбранных объектов
        selectedUsers.clear();

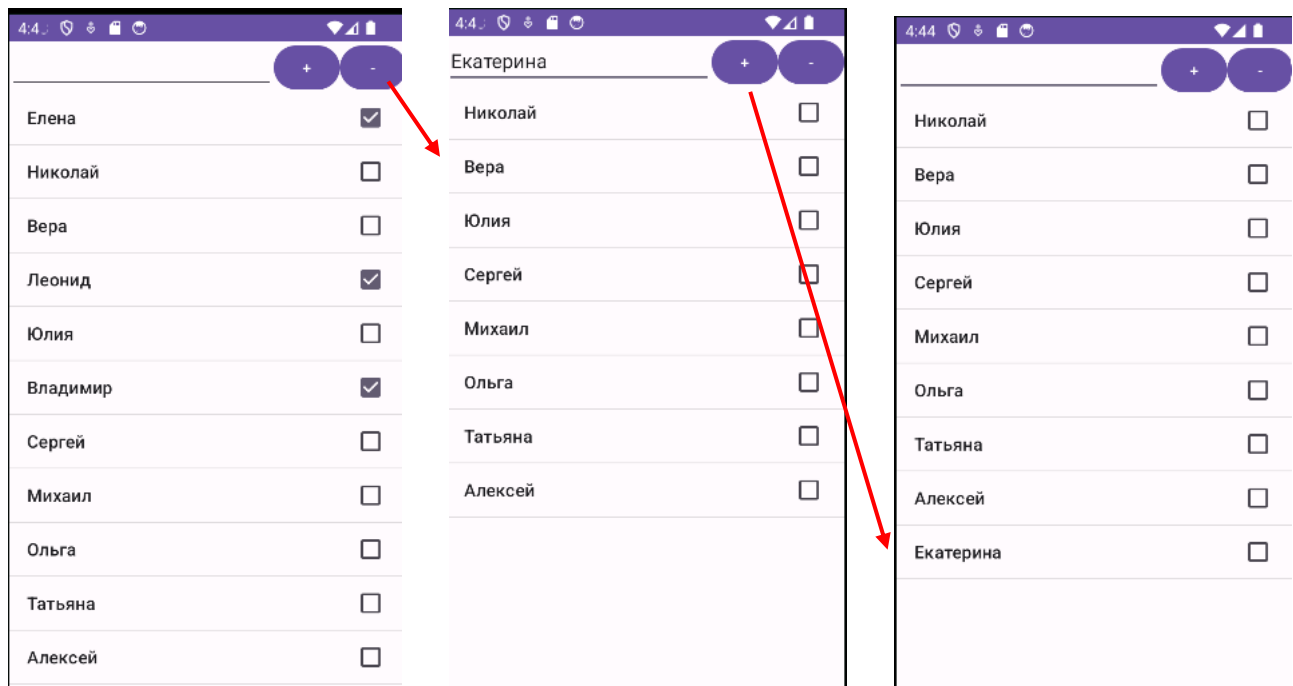
        usersAdapter.notifyDataSetChanged();
    }
}

```

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 снятия отметки с элемента, то есть при любом нажатии на элемент.

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



### Часть 3. Spinner

**Spinner** представляет собой выпадающий список. В файле разметки объявляется как элемент **Spinner**:

```
<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" />
```

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.

```

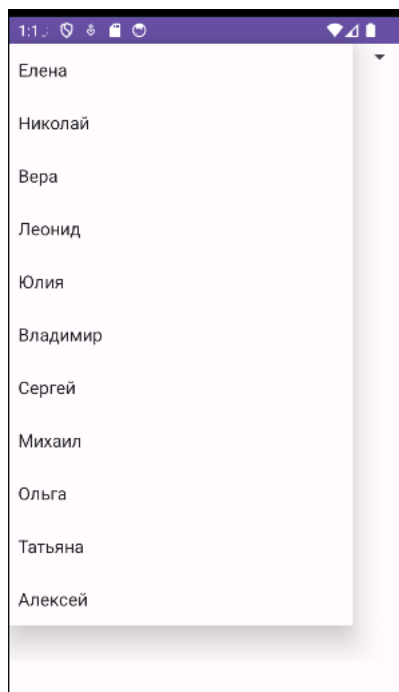
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    //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);
}

```

Получаем вот такое отображение списка:

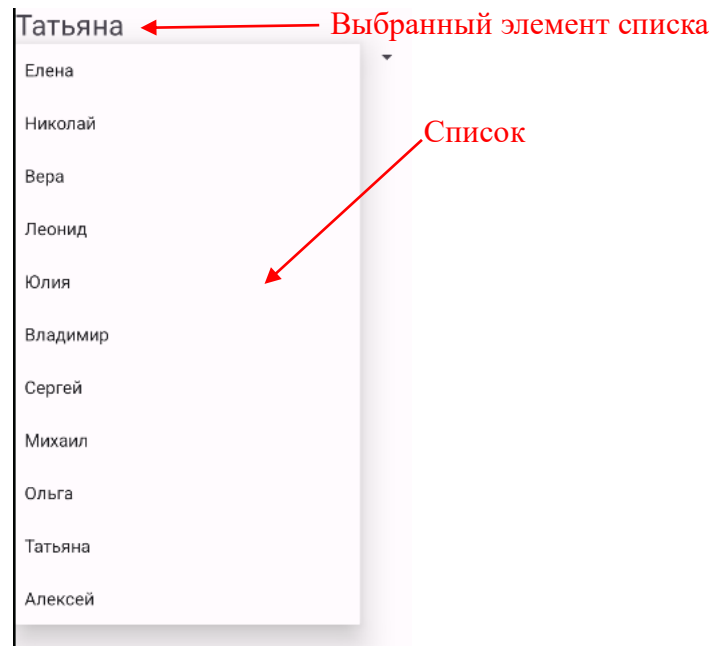


Используемый при создании ArrayAdapter ресурс **android.R.layout.simple\_spinner\_item** предоставляется платформой и является стандартной разметкой для создания выпадающего списка.

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 разметку, чтобы выводить выбранный элемент и в коде java определить для элемента Spinner слушатель **OnItemSelectedListener**.



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

- **parent:** объект **Spinner**, в котором произошло событие выбора элемента
- **view:** объект **View** внутри **Spinnera**, 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**.

#### Часть 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.

```
public class Name {  
    private String name;  
    private String surname;  
  
    public Name(String name, String surname) {  
        this.name = name;  
        this.surname = surname;  
    }  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
    public String getSurname() {  
        return surname;  
    }  
    public void setSurname(String surname) {  
        this.surname = surname;  
    }  
}
```

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**.



```

public class NameAdapter extends ArrayAdapter<Name> {
    private LayoutInflater inflater;
    private int layout;
    private List<Name> names;

    public NameAdapter(Context context, int resource, List<Name> name)
    {
        super(context, resource, name);
        this.names=name;
        this.layout=resource;
        this.inflater=LayoutInflater.from(context);
    }

    public View getView(int position, View convertView, ViewGroup
parent) {

        View view=inflater.inflate(this.layout, parent, false);

        TextView nameView = view.findViewById(R.id.name);
        TextView surnameView = view.findViewById(R.id.surname);

        Name name = names.get(position);

        nameView.setText(name.getName());
        surnameView.setText(name.getSurname());

        return view;
    }
}

```

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:

```
public class MainActivity extends AppCompatActivity {
    ArrayList<Name> names = new ArrayList<Name>();
    ListView nameList;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // начальная инициализация списка
        setInitialData();
        // получаем элемент ListView
        nameList = findViewById(R.id.nameList);
        // создаем адаптер
```

```

        NameAdapter stateAdapter = new NameAdapter(this,
R.layout.list_item, names);
        // устанавливаем адаптер
        nameList.setAdapter(stateAdapter);
        // слушатель выбора в списке
        AdapterView.OnItemClickListener itemListener = new
AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View v, int
position, long id) {
                // получаем выбранный пункт
                Name selectedName =
(Name)parent.getItemAtPosition(position);
                Toast.makeText(getApplicationContext(), "Было выбрано
имя " + selectedName.getName(),
                    Toast.LENGTH_SHORT).show();
            }
        };
        nameList.setOnItemClickListener(itemListener);
    }
    private void setInitialData(){

        names.add(new Name ("Елена", "Васильева"));
        names.add(new Name ("Виктор", "Петров"));
        names.add(new Name ("Александр", "Иванов"));
        names.add(new Name ("Татьяна", "Предметова"));
        names.add(new Name ("Наталья", "Андреева"));

    }
}

```

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`:



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** следующим образом:

```
public View getView(int position, View convertView, ViewGroup parent)
{
    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;
}
private class ViewHolder {
    final TextView nameView, surnameView;
    ViewHolder(View view){
        nameView = view.findViewById(R.id.name);
        surnameView = view.findViewById(R.id.surname);
    }
}

```

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.

## Часть 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**.

```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/list"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:listitem="@layout/list_item" />
```

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.

```
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;
import java.util.List;

public class SimpleAdapter extends
RecyclerView.Adapter<SimpleAdapter.ViewHolder> {
    private List<String> items;

    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();
    }

    static class ViewHolder extends RecyclerView.ViewHolder {
        TextView textView;
        ViewHolder(View view) {
            super(view);
            textView = view.findViewById(R.id.item_text);
        }
    }
}

```

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.
- **`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 с RecyclerView элементами через код.

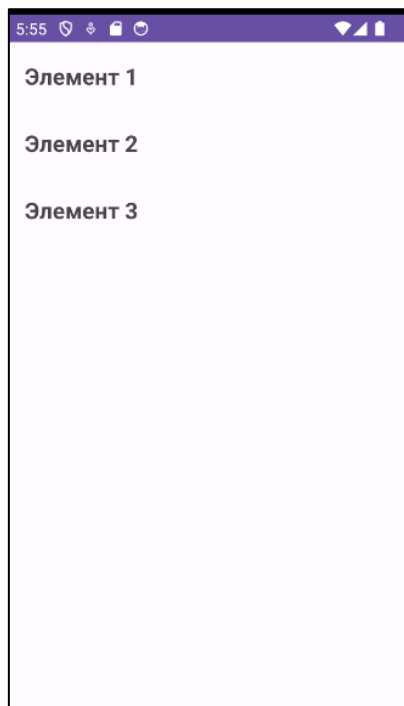
```
import androidx.appcompat.app.AppCompatActivity;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
import android.os.Bundle;
import java.util.Arrays;
import java.util.List;

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(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);
    }
}
```





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:

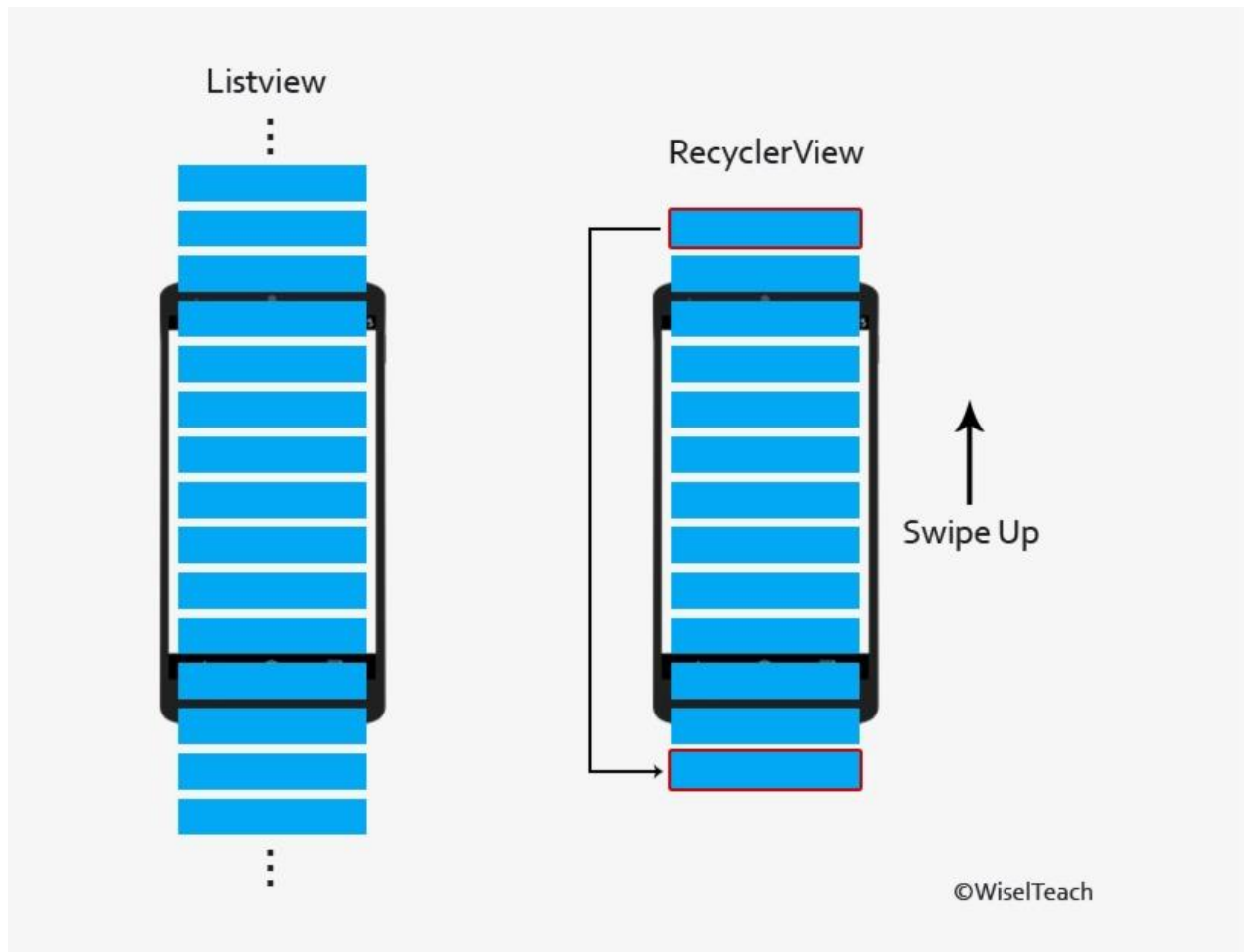
- **LinearLayoutManager**: упорядочивает элементы в виде списка с одной колонкой
- **GridLayoutManager**: упорядочивает элементы в виде грида со столбцами и строками. Грид может упорядочивать элементы по горизонтали (горизонтальный грид) или по вертикали (вертикальный грид)
- **StaggeredGridLayoutManager**: 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:

1. Лучшая производительность: RecyclerView использует концепцию ViewHolder для минимизации вызовов метода `findViewById()`, что улучшает производительность прокрутки.
2. Больше контроля над раскладкой элементов: RecyclerView поддерживает LayoutManager, который определяет расположение элементов в списке или сетке, а также другие кастомные раскладки.

3. Встроенные анимации: RecyclerView предлагает встроенные анимации для операций добавления, удаления и перемещения элементов.
4. Гибкость: Благодаря отдельному компоненту Adapter, RecyclerView легко адаптировать под различные типы данных и конфигурации отображения.

Ниже приведено отличие ListView от RecyclerView.



### Задание

1. Создать несколько Activity. На первом экране выводится список категорий продуктов (или своя категория). В зависимости от выбранного элемента списка, происходит переход на следующую Activity. На второй Activity выводится подробный список элементов выбранной категории. Например, выбираете категорию «Яблоки» и на второй Activity у вас перечень сортов яблок. Также должны быть реализованы возможности добавления и удаления элементов из списка второго Activity (хотя бы для одной выбираемой категории, в остальных можно просто вывести список).

2. Создать Activity с RecyclerView для отображения данных. Выводить несколько полей с разными элементами: ОБЯЗАТЕЛЬНО картинку и TextView.
3. Создать Activity со ScrollView с элементами TextView (минимальное количество элементов должно быть такое, чтобы возможно было проверить функцию скроллинга) и EditText (внизу страницы, чтобы можно было ввести текст).
4. Создать Activity с выпадающим списком (Spinner).

### **Источники**

- 1) <https://developer.alexanderklimov.ru/android/views/listview.php?ysclid=lteshr96nw599691729>
- 2) <https://www.geeksforgeeks.org/android-listview-in-java-with-example/>
- 3) <https://metanit.com/java/android/5.1.php?ysclid=lteshwrqs1580283742>
- 4) <https://habr.com/ru/articles/705064/>
- 5) <https://developer.android.com/reference/android/widget/ScrollView>