|  |  |  |
| --- | --- | --- |
|  |  |  |
| 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 группыИКБО-20-23 | Комисарик М.А. |
| **Проверил:**  Старший преподаватель кафедры МОСИТ | Шешуков Л.С. |

Москва 2025 г.СОДЕРЖАНИЕ

[**РТУ МИРЭА** 1](#_Toc193903946)

[1 ТЕОРЕТИЧЕСКОЕ ВВЕДЕНИЕ 3](#_Toc193903947)

[2 ПРАКТИЧЕСКОЕ ЗАДАНИЕ 4](#_Toc193903948)

[3 ЗАКЛЮЧЕНИЕ 5](#_Toc193903949)

# ТЕОРЕТИЧЕСКОЕ ВВЕДЕНИЕ

## 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 умещающихся на одном экране.

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

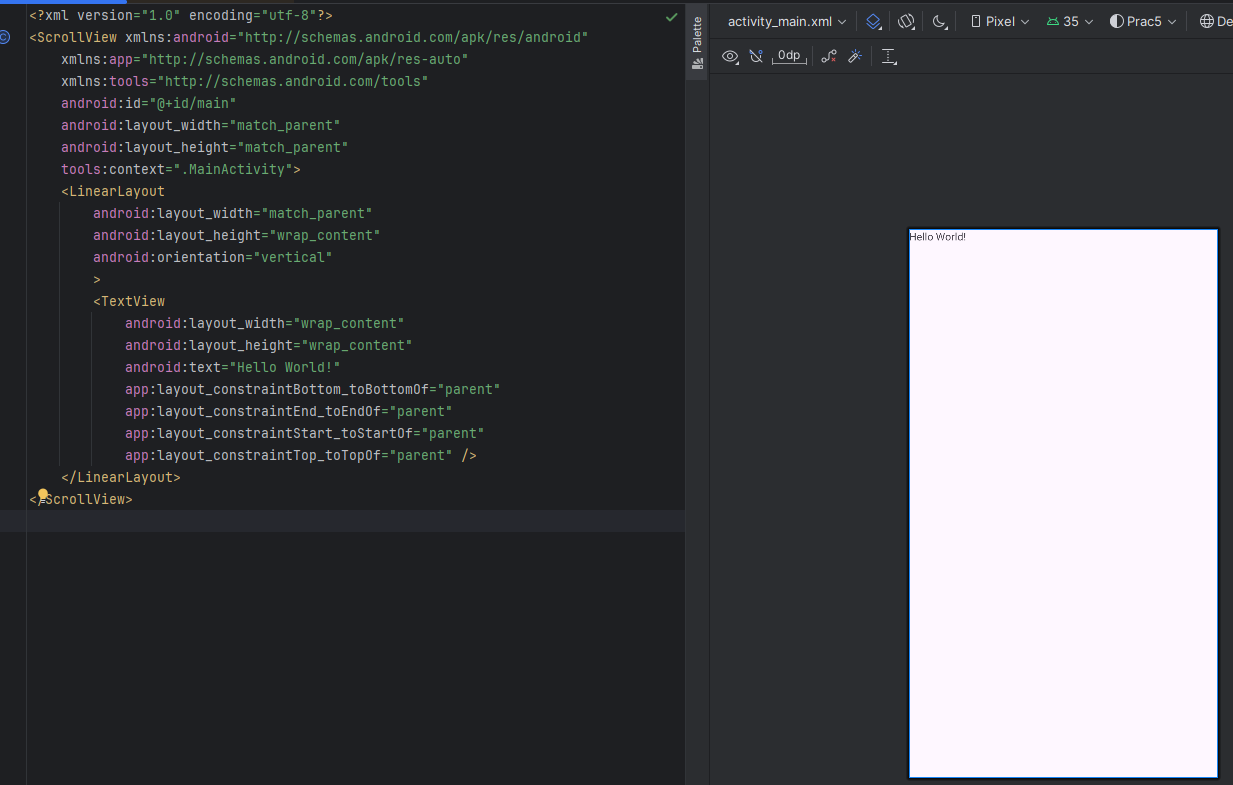


Рисунок 1 – Реализация ScrollView в коде

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.

## ListView

Android представляет широкую палитру элементов, которые представляют списки. Все они является 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.

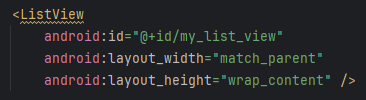


Рисунок 2 – Разметка ListView

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).

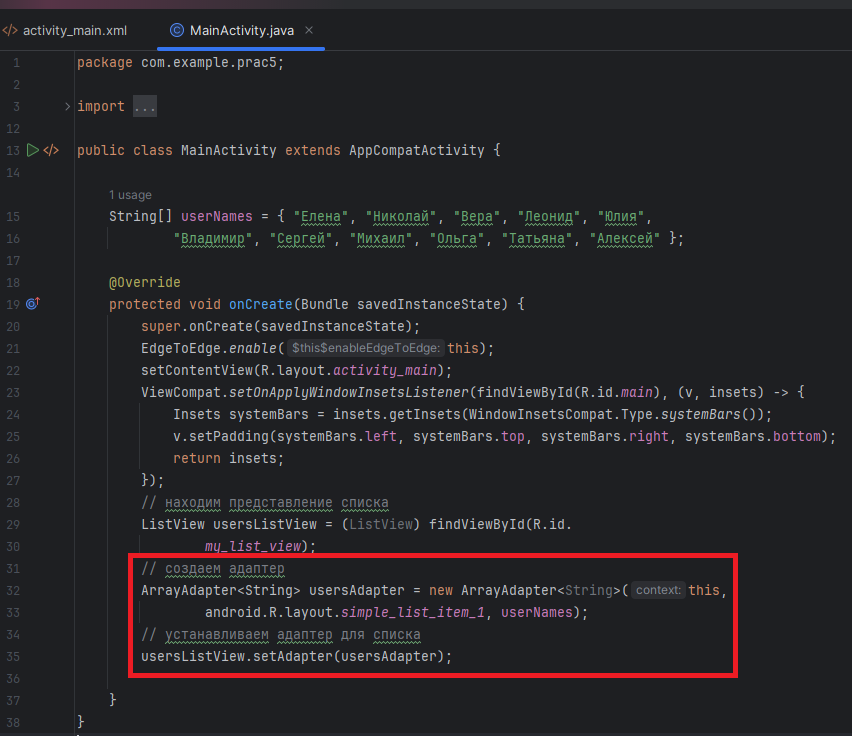


Рисунок 3 – Использование ListView для отображения списка имён

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

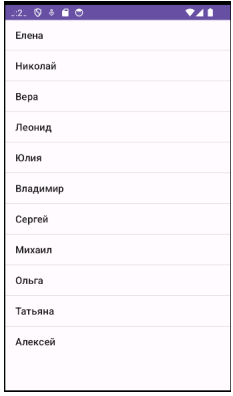


Рисунок 4 – Отображение вертикального списка при помощи ListView

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 находится в каталоге res/values в xml-файле с произвольным именем.

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

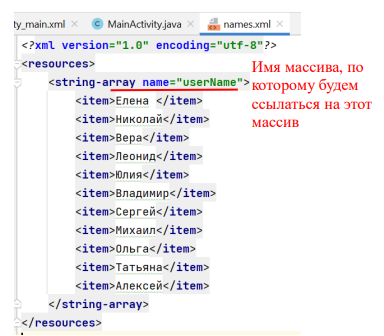


Рисунок 5 – Ресурсный файл имён

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).

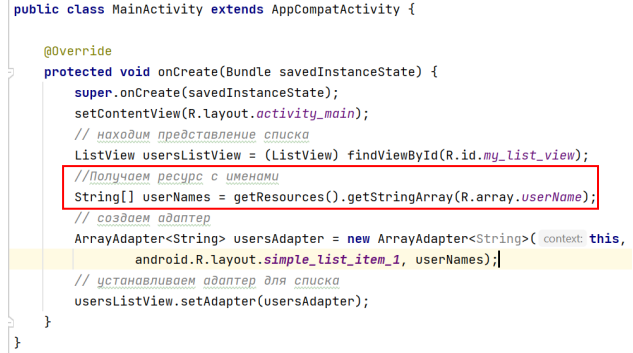


Рисунок 6 – Привязка ресурса в Java

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.



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).

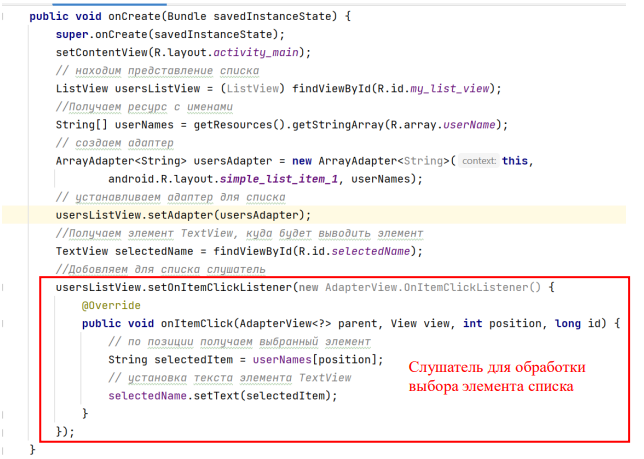


Рисунок 8 – Обработка выбора элемента списка имён

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).

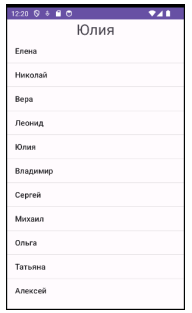


Рисунок 9 – Отображение выбора элемента из списка имён

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).



Рисунок 10 – Определение значения android:choiceMode

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

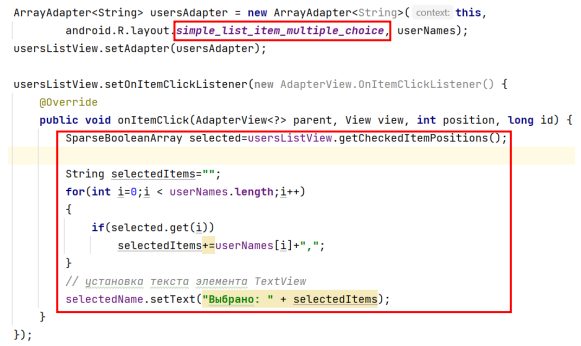


Рисунок 11 – Реализация логики обработки выбора элементов списка

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).

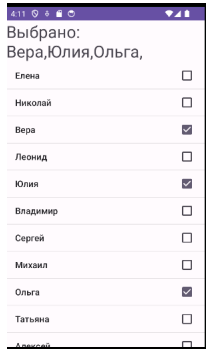


Рисунок 12 – Отображение выбора элемента в списке имён

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 Рисунок 13-Рисунок 14).

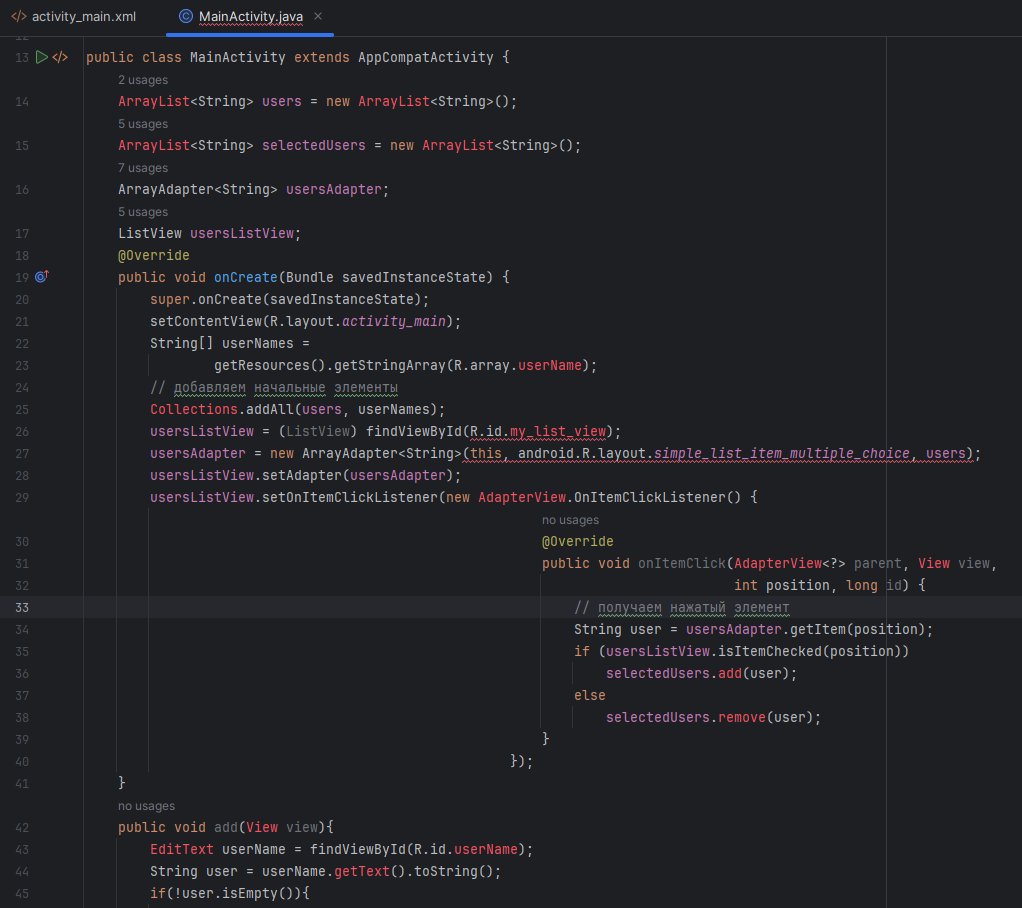


Рисунок 13 – Изменения функционала итогового приложения, часть 1

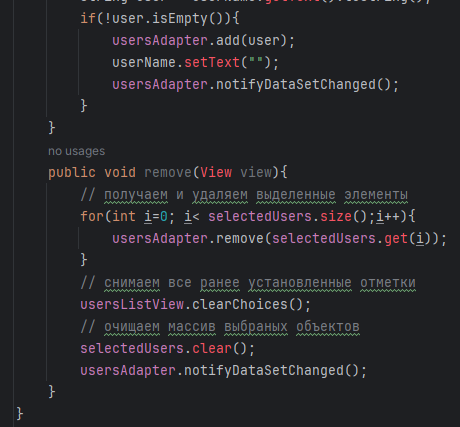


Рисунок 14 – Изменения функционала итогового приложения, часть 2

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 снятия 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().

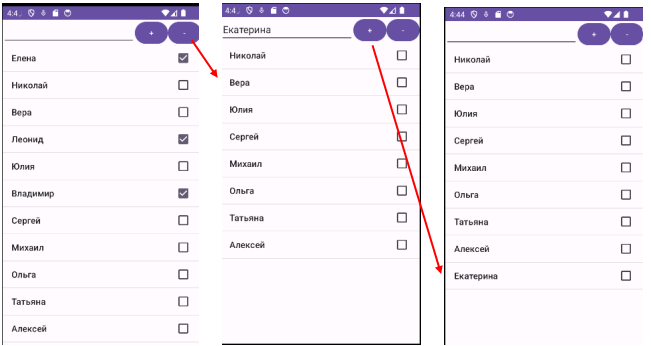


Рисунок 15 – Отображение изменённого функционала приложения

## Spinner

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

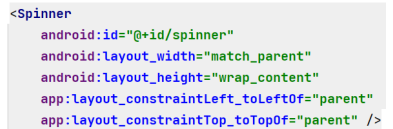


Рисунок 16 – Разметка Spinner

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).



Рисунок 17 – Реализация взаимодействия источника данных с адаптером

При такой реализации на экране будет отображаться список как на рисунке Рисунок 18

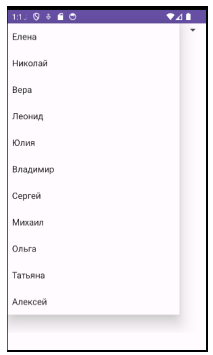


Рисунок 18 – Отображаемый список в итоговом приложении

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).

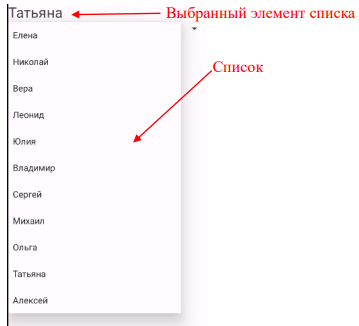


Рисунок 19 – Отображение элемента, выбранного из списка имён

Метод onItemSelected слушателя OnItemSelectedListener получает четыре параметра:

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

## 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).

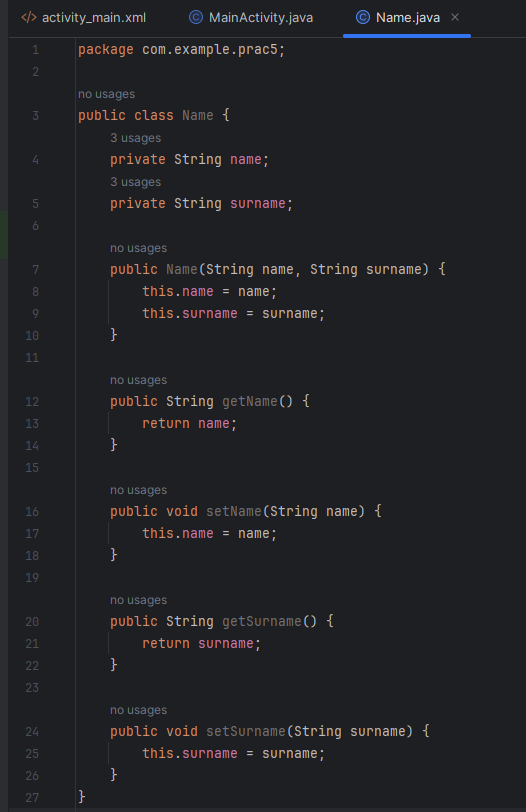


Рисунок 20 – Создание класса, хранящего имя и фамилию человека

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).

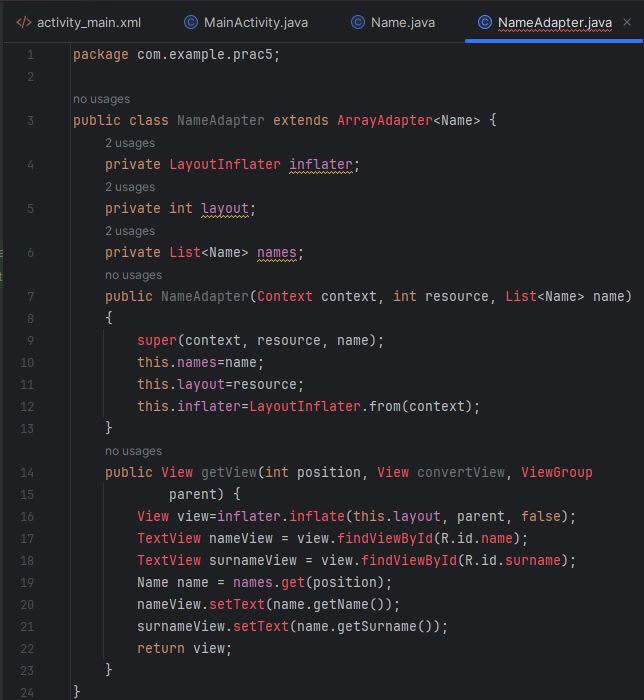


Рисунок 21 – Описание логики класса NameAdapter

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).

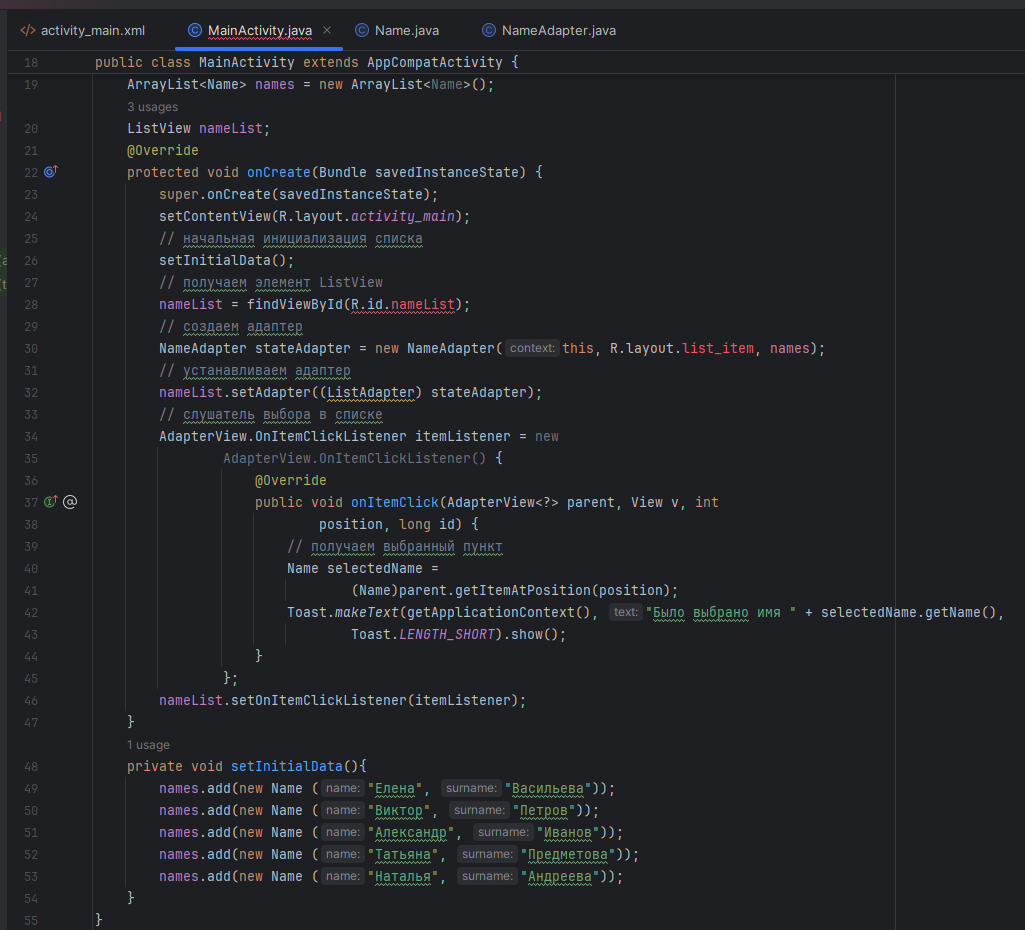


Рисунок 22 – Соединение NameAdapter с ListView

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).

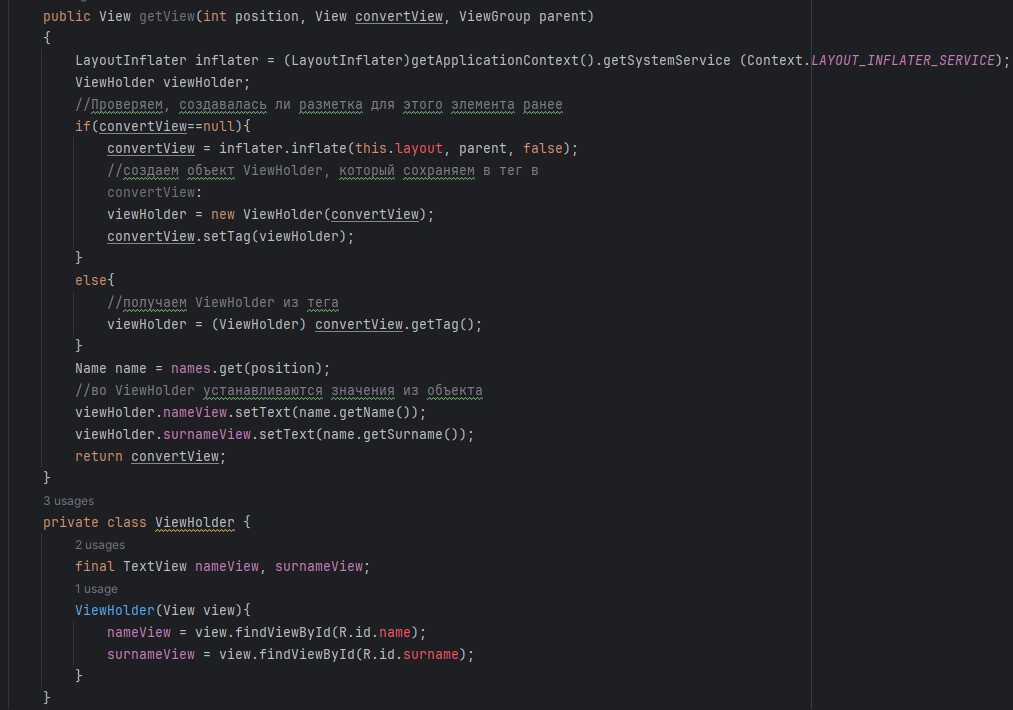


Рисунок 24 – Оптимизация метод getView класса NameAdapter

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.

## 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).

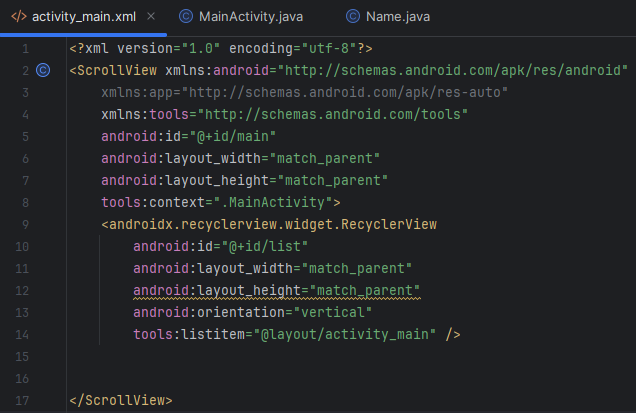


Рисунок 25 – Описание элемента RecyclerView в файле activity\_main.xml

Прежде чем наполнить RecycleView 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).



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).

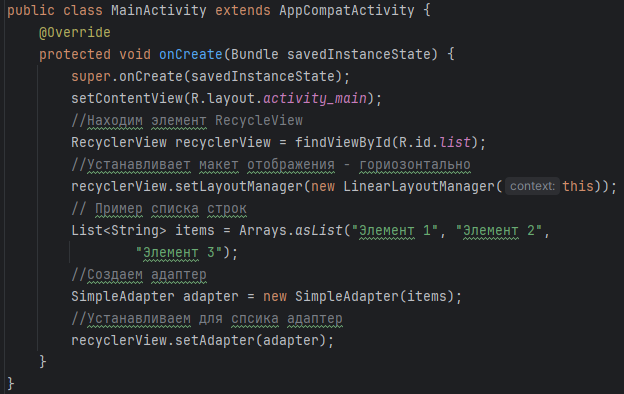


Рисунок 27 – Наполнение RecyclerView через код

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

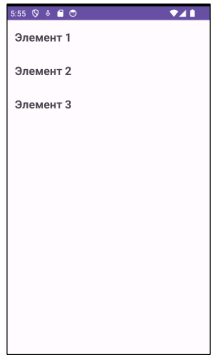


Рисунок 28 – Итоговый вид приложения

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 столлбцами 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 ширну (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).

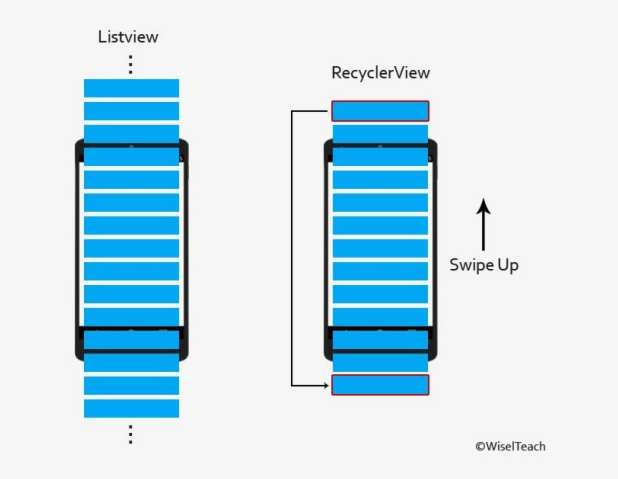


Рисунок 29 – Отображение отличий между ListView и RecyclerView

# ПРАКТИЧЕСКОЕ ЗАДАНИЕ

## ListView

### Создание проекта

Создадим проект и назовем его Pract5 (Рисунок 30).

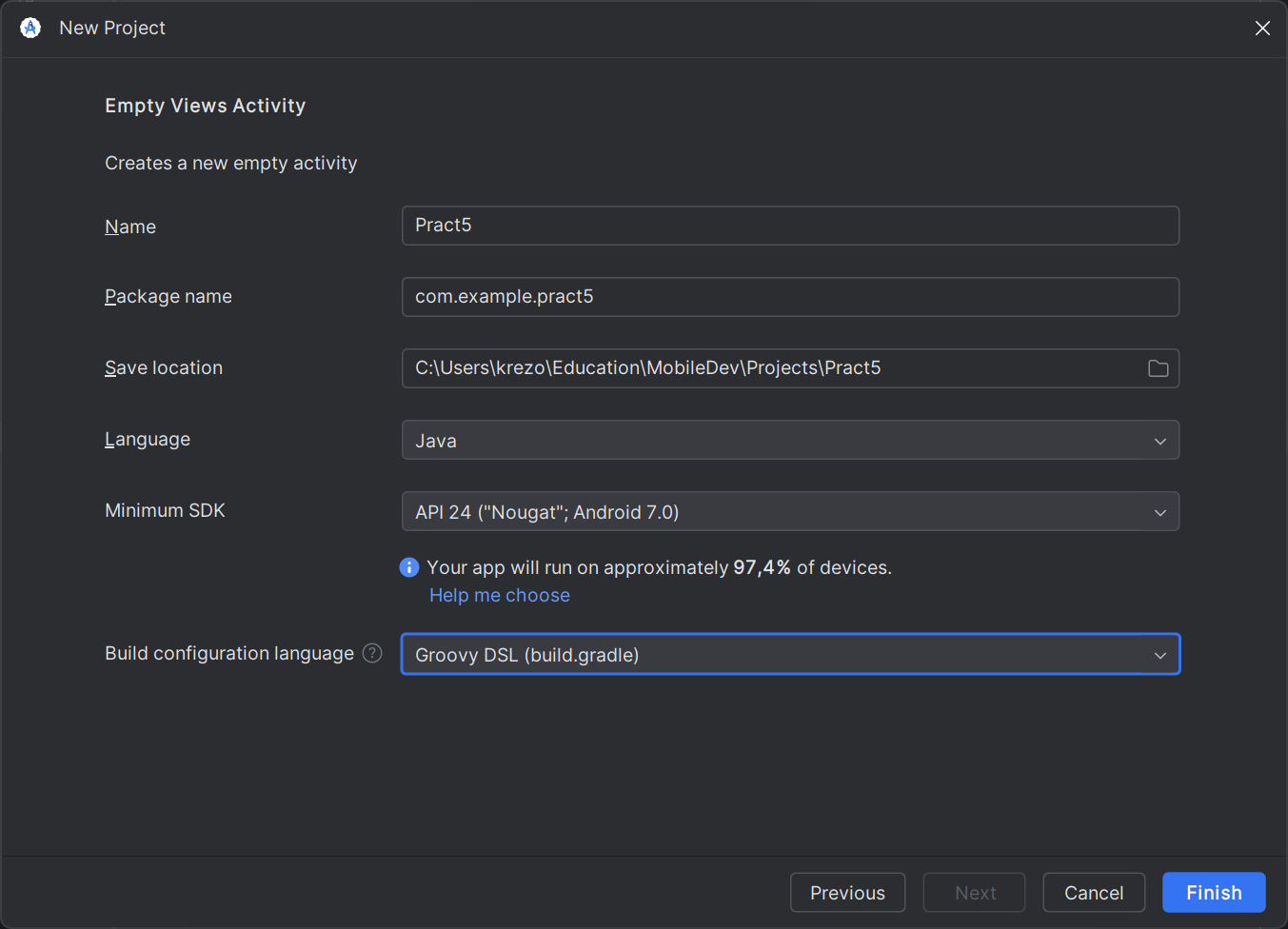


Рисунок 30 – Создание проекта

### Разметка

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

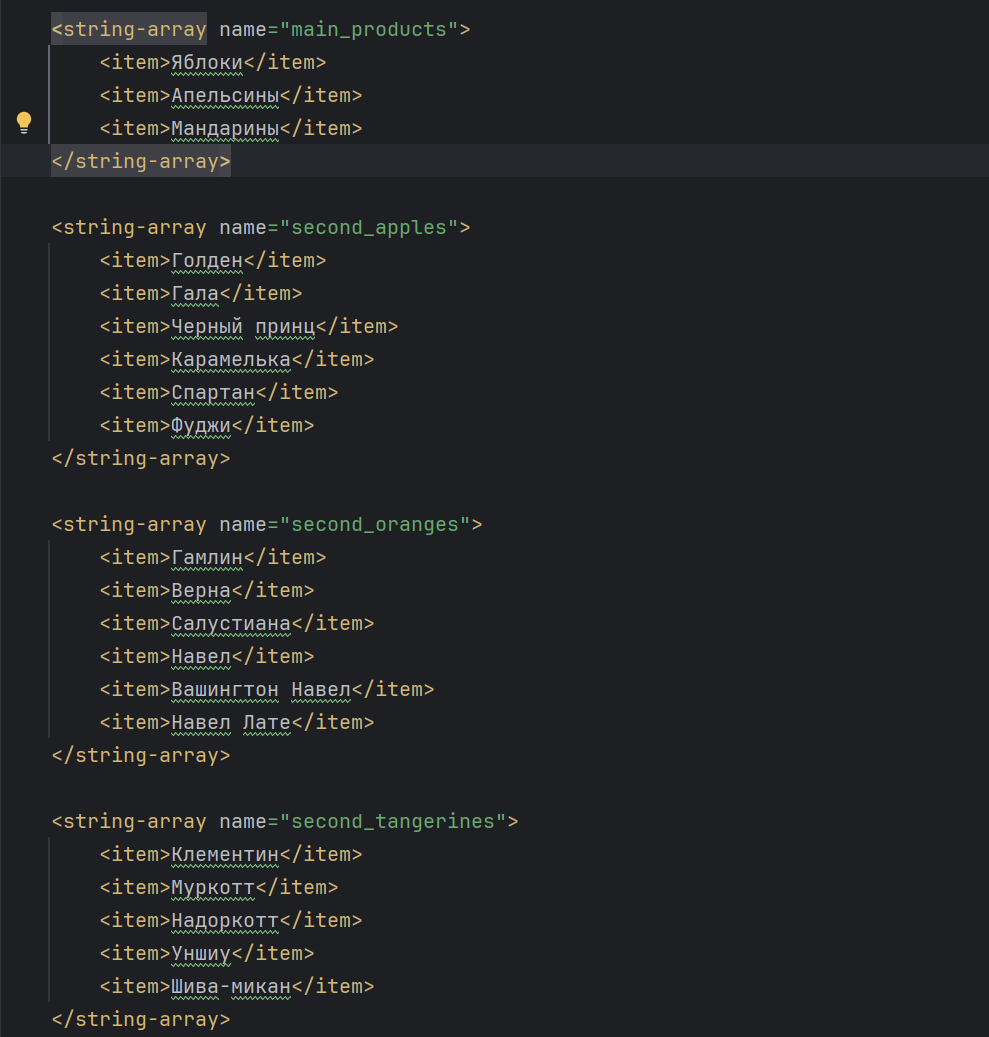


Рисунок 31 – Строковые ресурсы списков в файле strings.xml

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

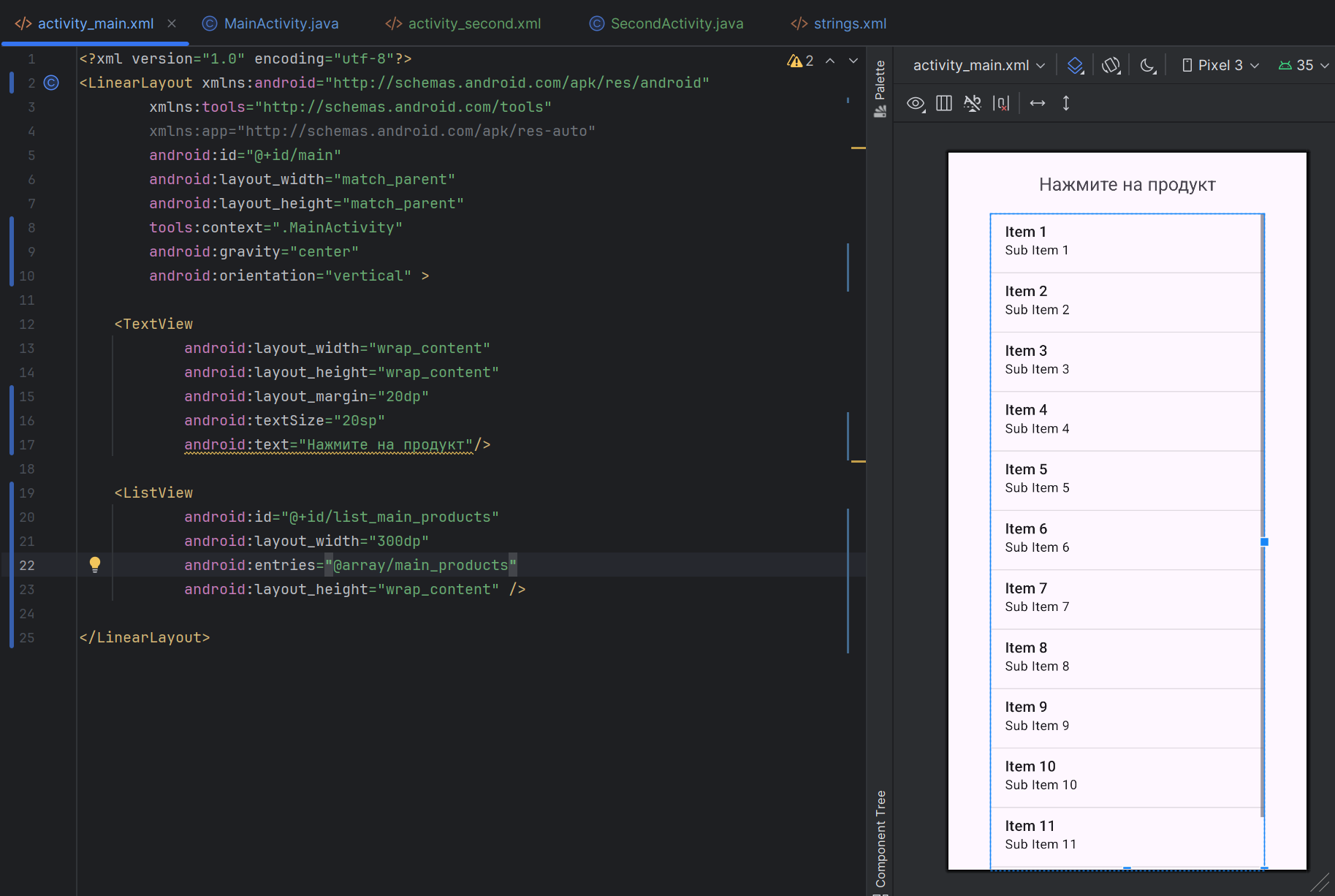


Рисунок 32 – Файл разметки activity\_main.xml

Создадим класс Activity и назовем его SecondActivity (Рисунок 31).

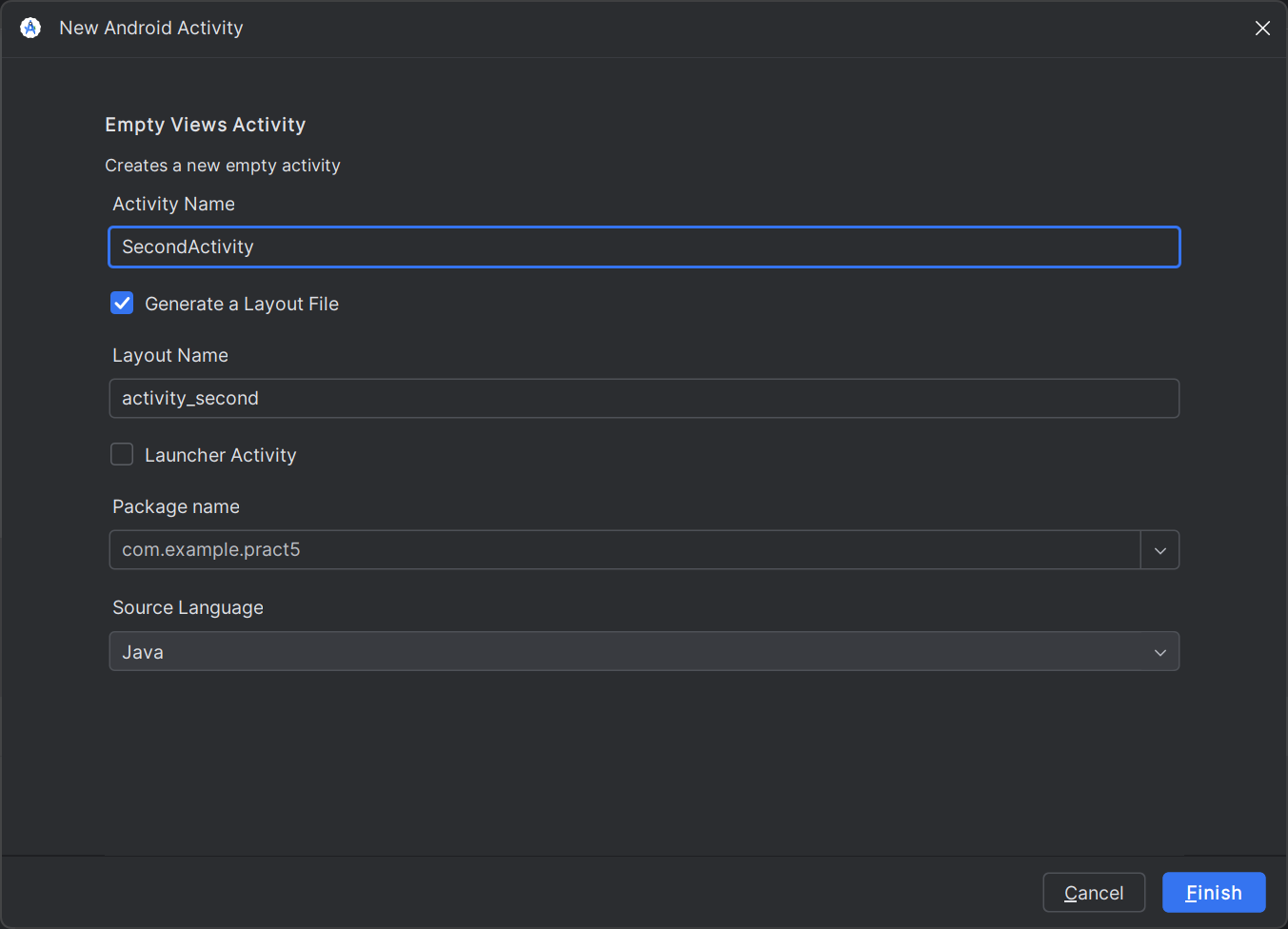


Рисунок 33 – Создание SecondActivity

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).

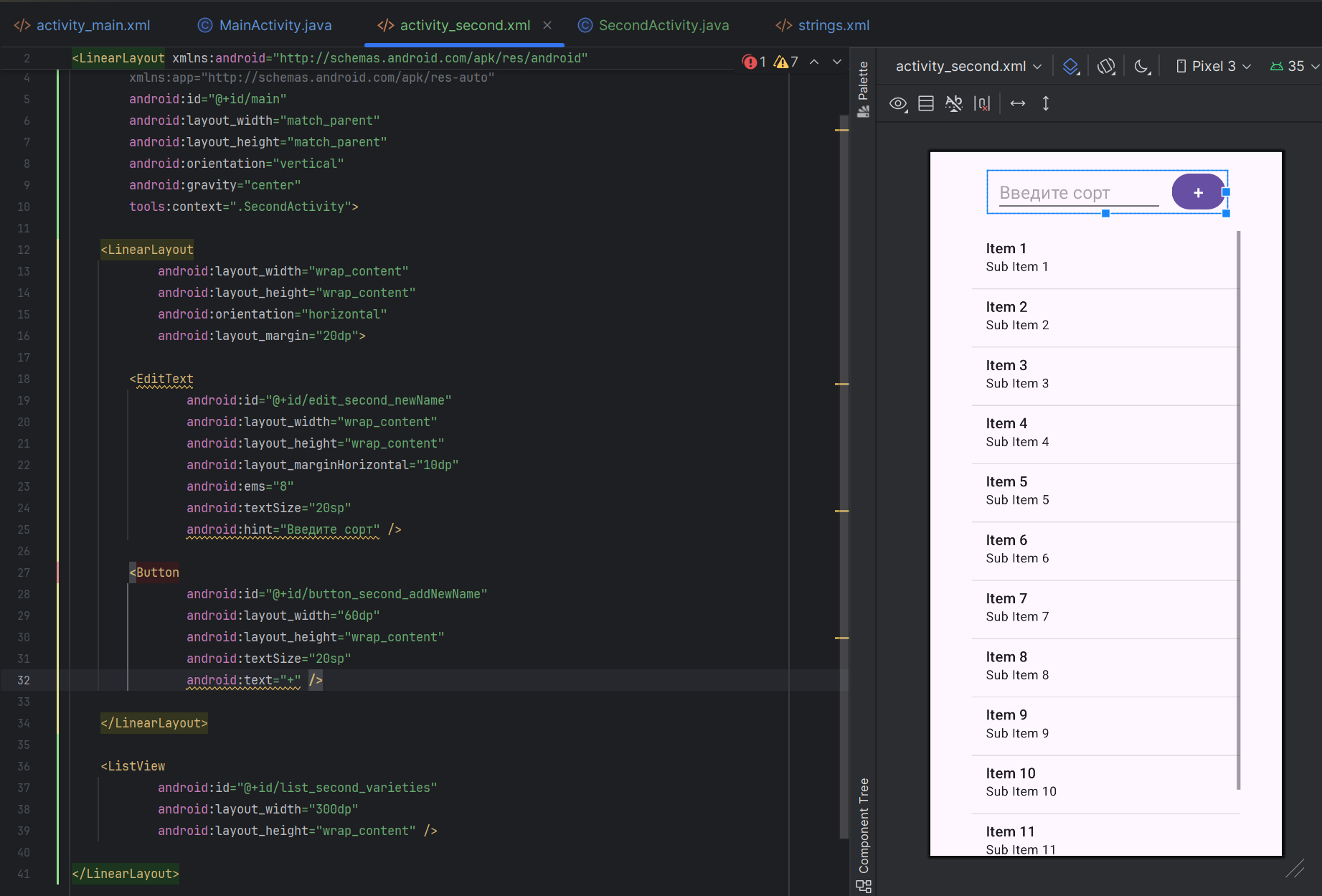


Рисунок 34 – Файл разметки activity\_second.xml

### 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).

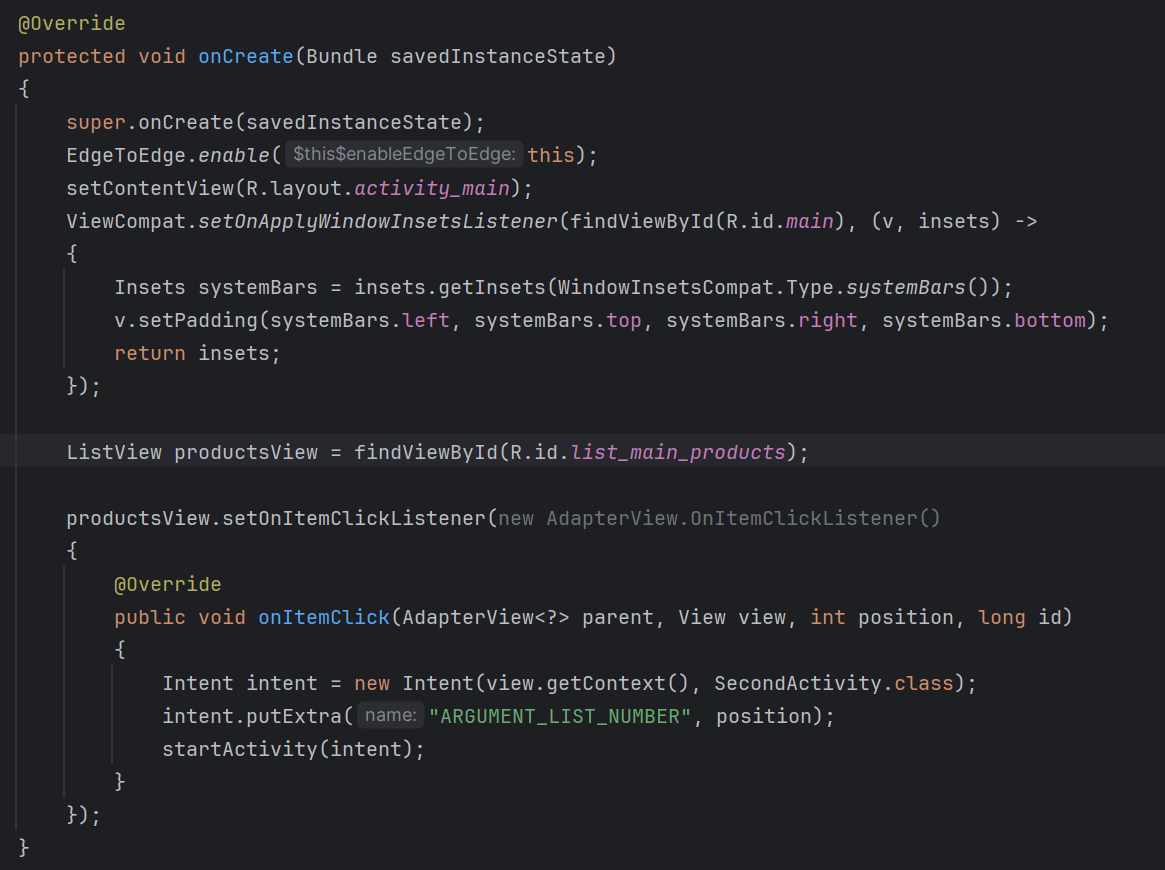


Рисунок 35 – Метод onCreate() класса MainActivity

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).



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).

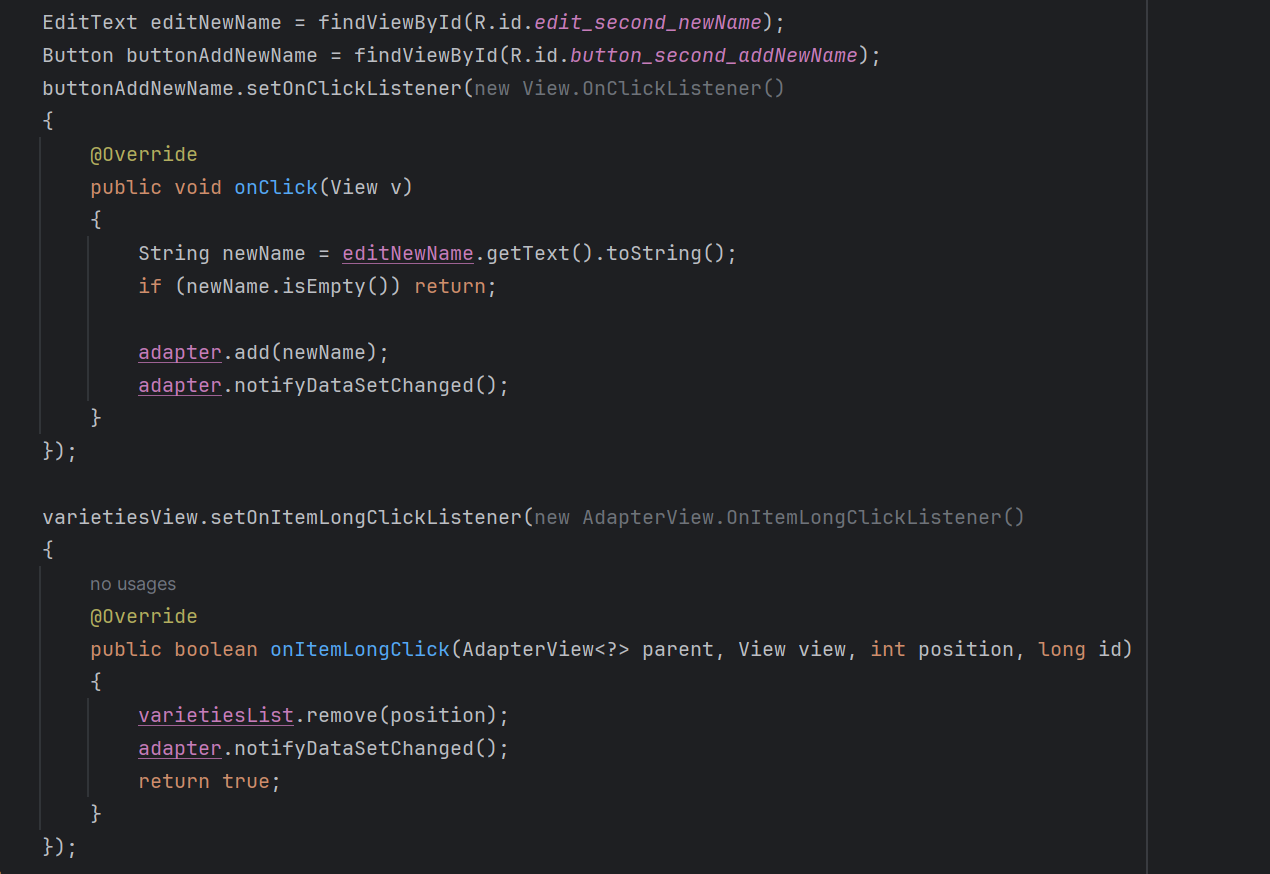


Рисунок 37 – Реализация функционала списка в методе onCreate() класса SecondActivity

### Тестирование

Протестируем работу списка фруктов и их сортов.

На рисунке Рисунок 38 представлена начальная страница приложения.

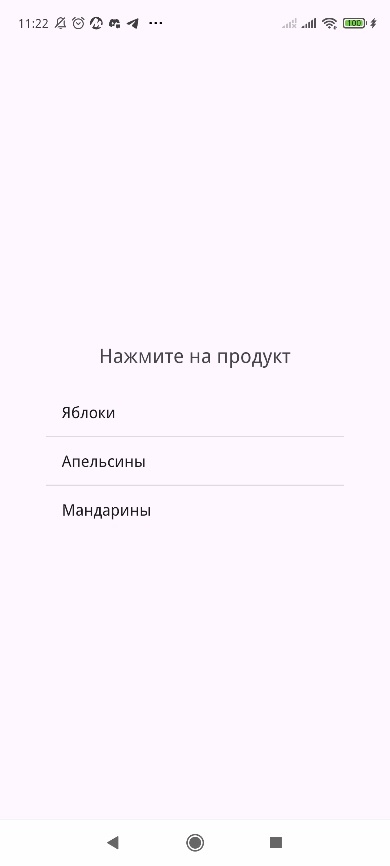


Рисунок 38 – Начальная страница приложения

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

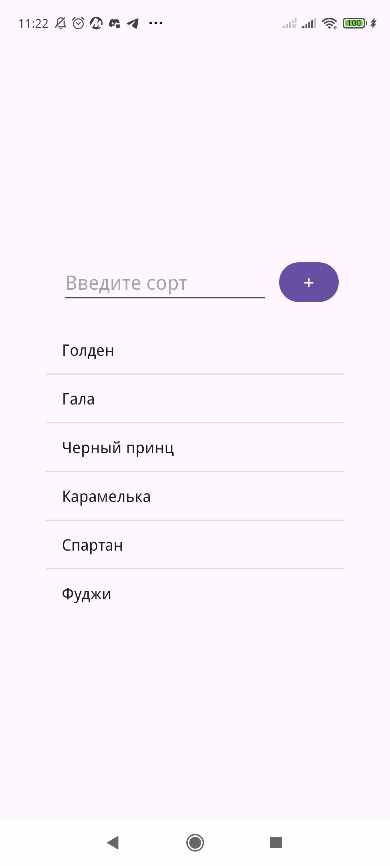


Рисунок 39 – Сорта яблок до изменения

При долгом зажатии происходит удаление элемента (Рисунок 40).

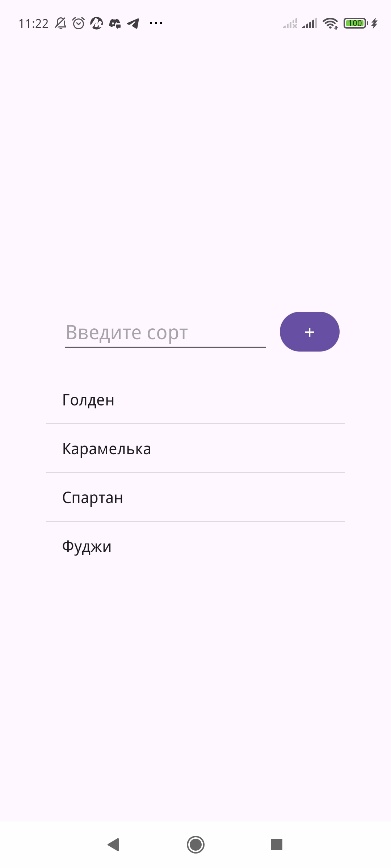


Рисунок 40 – Сорта яблок после удаления элементов

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

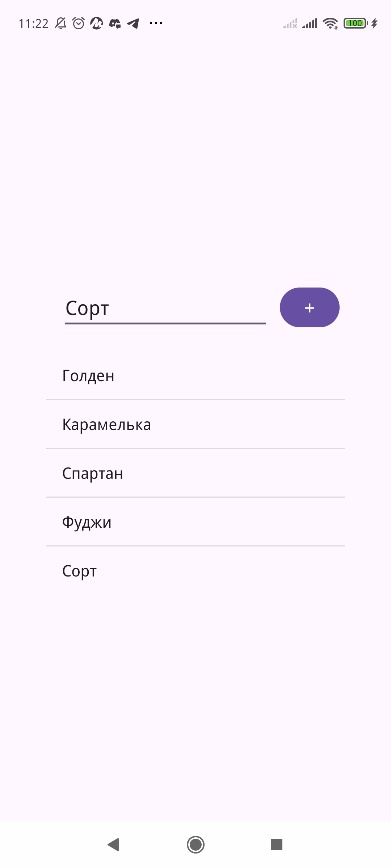


Рисунок 41 – Сорта яблок после добавления элемента

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

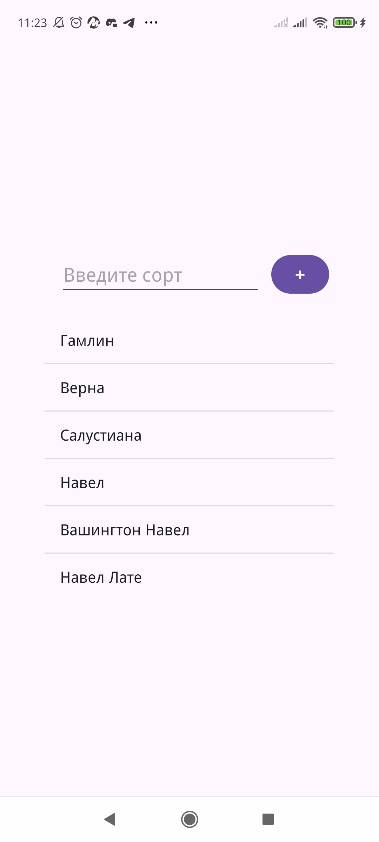


Рисунок 42 – Сорта апельсинов

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

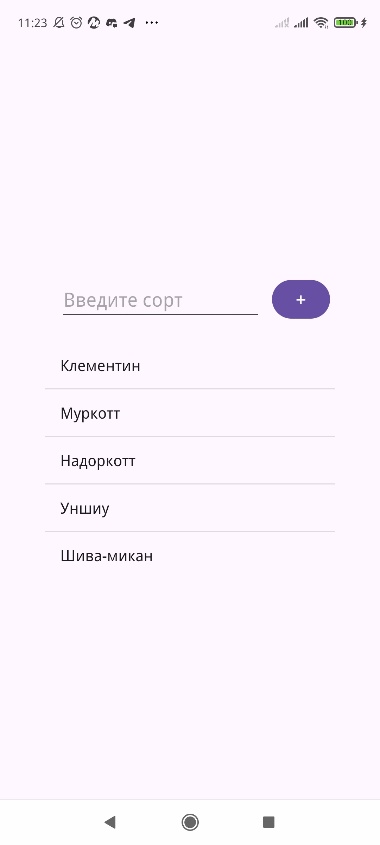


Рисунок 43 – Сорта мандаринов

## RecyclerView

### Разметка

Создадим новый файл разметки элемента списка item\_third.xml и определим в нем текстовое поле и картинку (Рисунок 44).

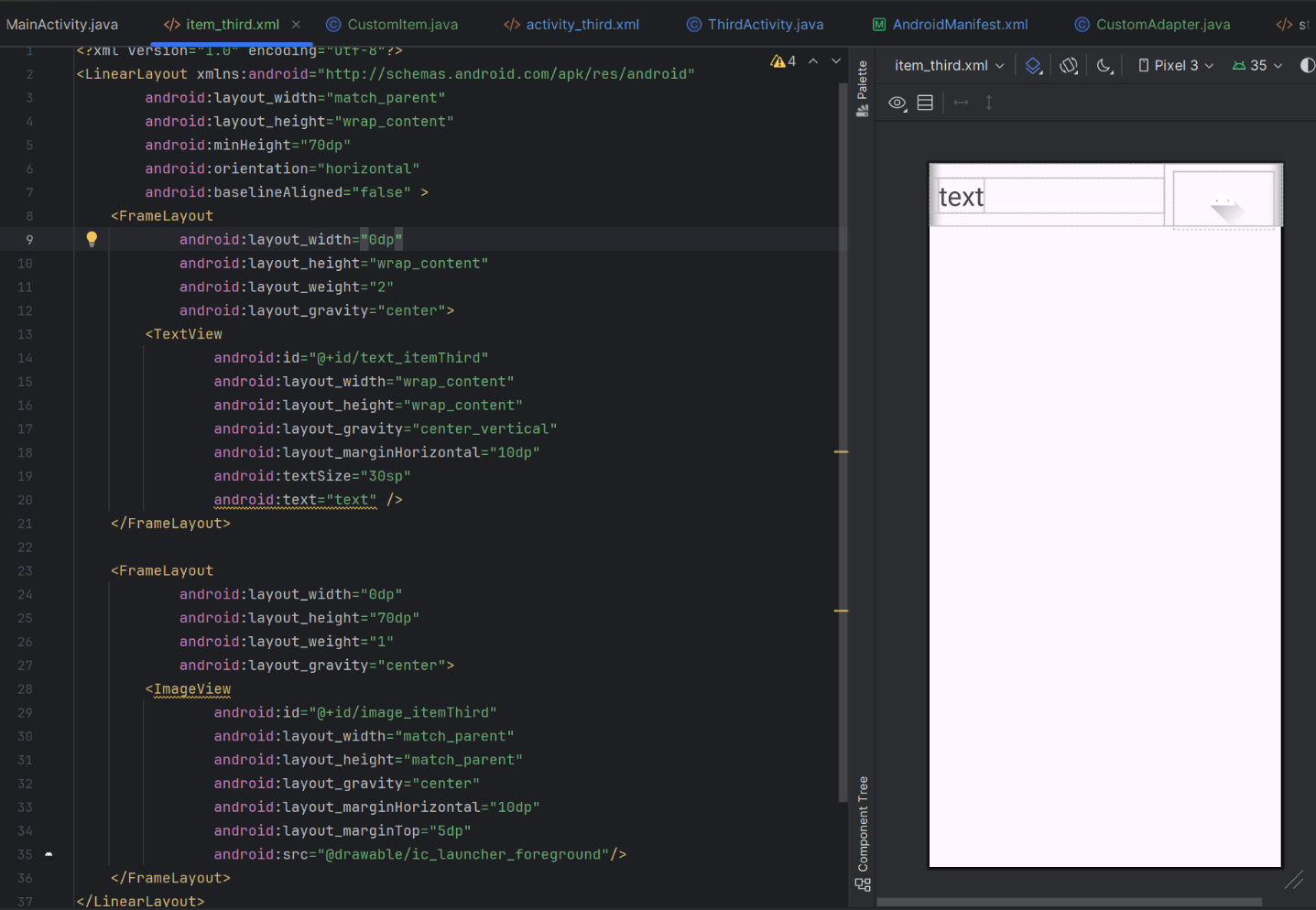


Рисунок 44 – Файл разметки item\_third.xml

Создадим новую Activity и назовем ее ThirdActivity (Рисунок 44).

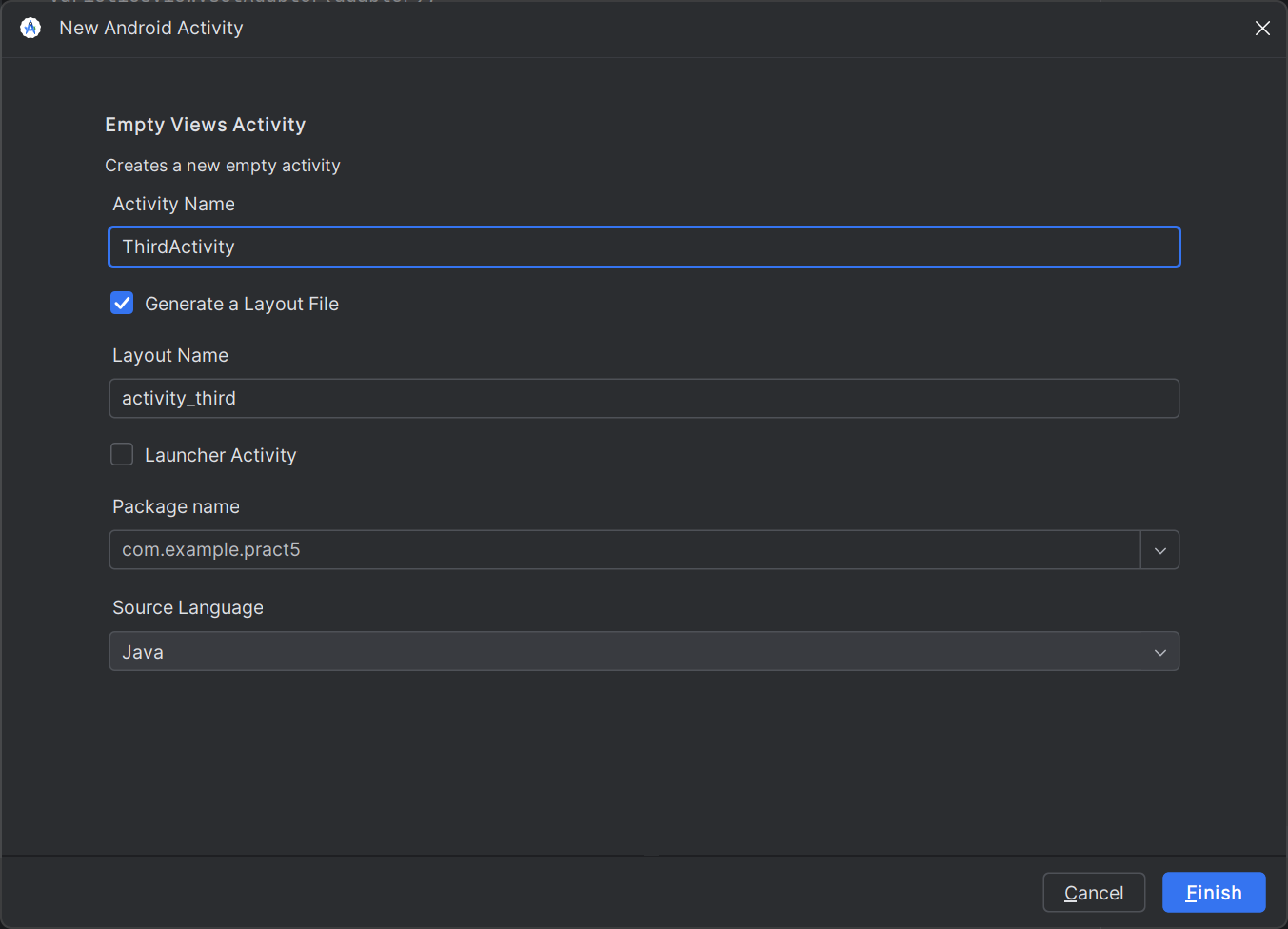


Рисунок 45 – Создание ThirdActivity

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

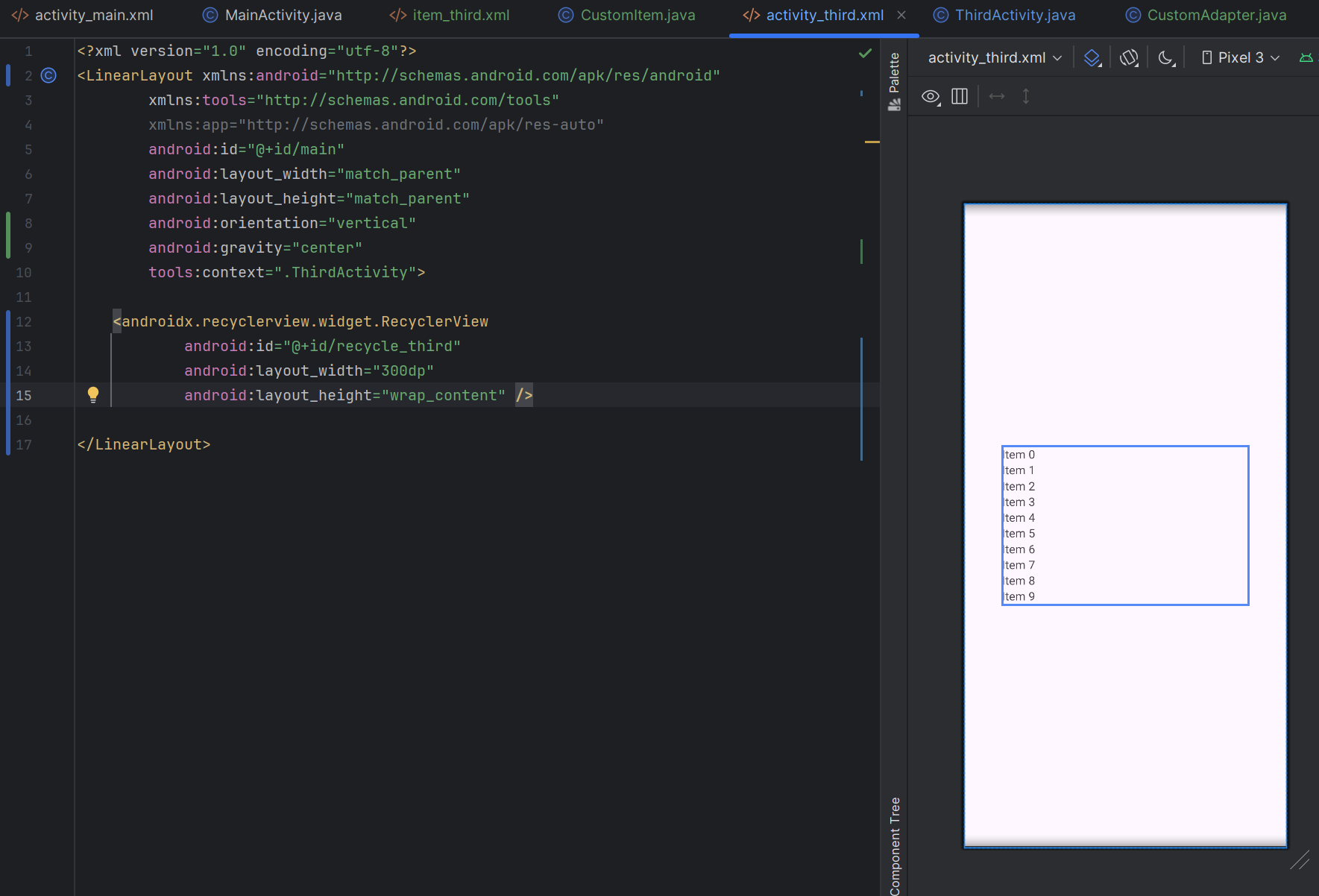


Рисунок 46 – Файл разметки activity\_third.xml

### Реализация адаптера

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

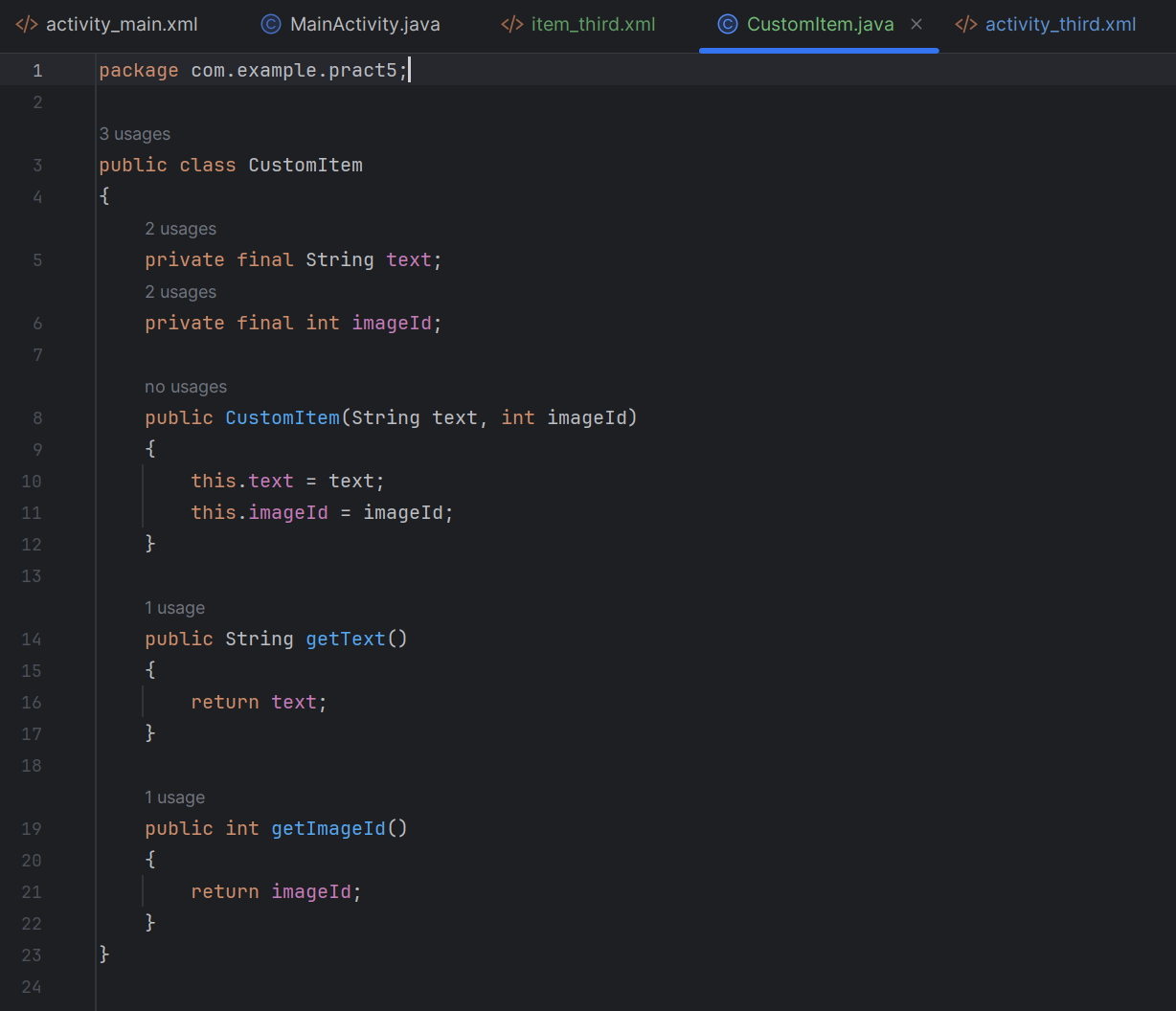


Рисунок 47 – Описание класса CustomItem

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).

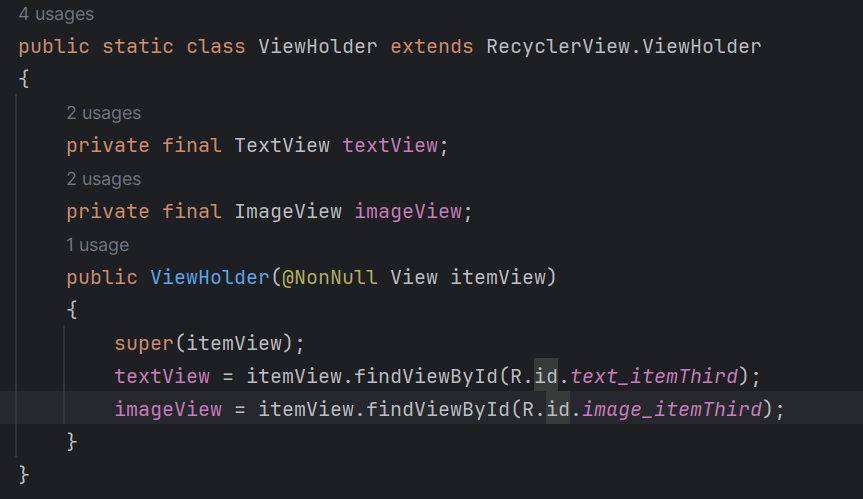


Рисунок 48 – Описание класса CustomAdapter.ViewHolder

В классе CustomAdapter требуется описать 3 родительских метода:

* onCreateViewHolder():
* onBindViewHolder();
* getItemCount().

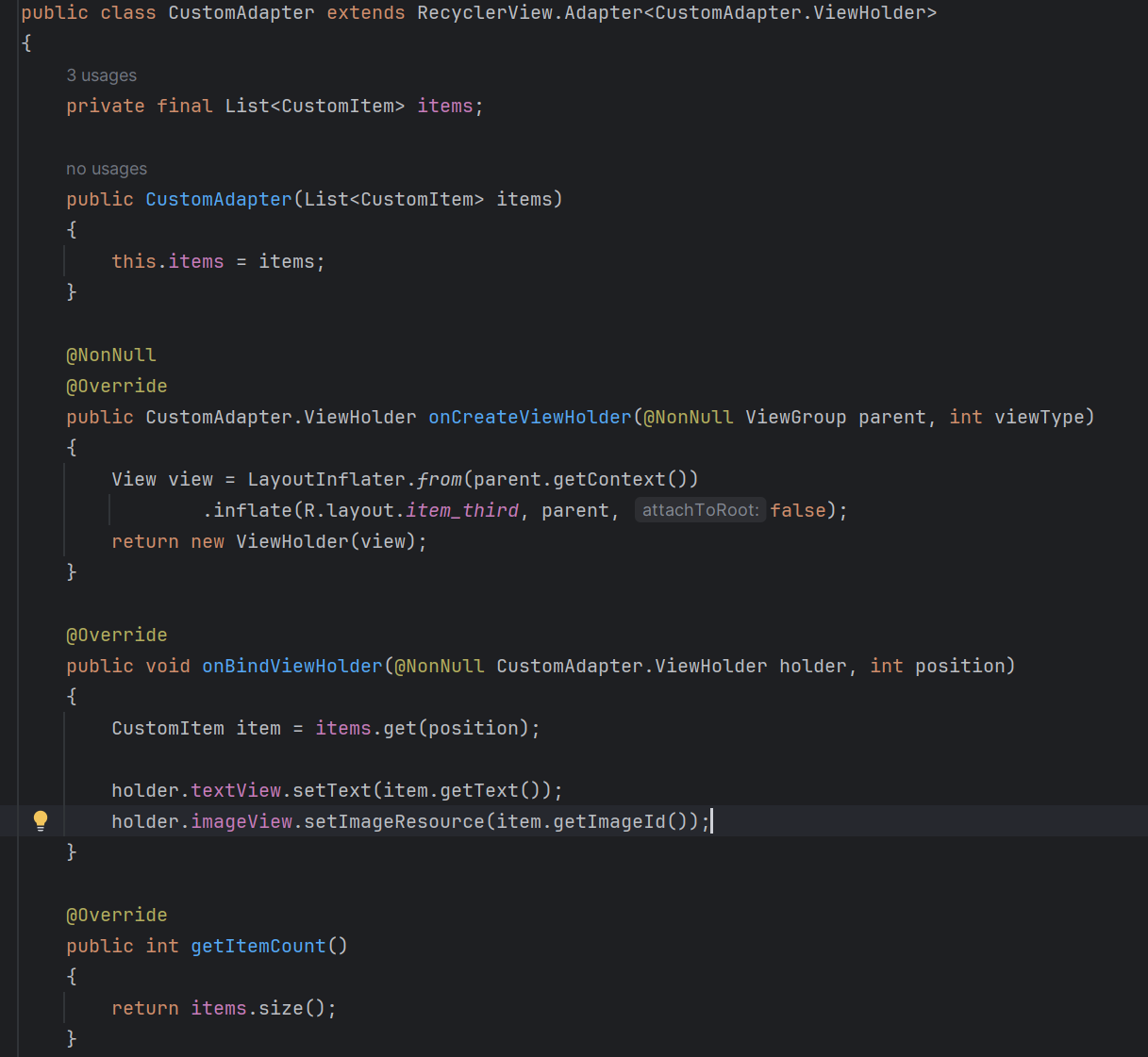


Рисунок 49 – Описание класса CustomAdapter

# ЗАКЛЮЧЕНИЕ

В ходе выполнения работы было изучено использование Activity Result API для передачи данных между активностями. Также было рассмотрено управление фрагментами, включая их создание, размещение и навигацию между ними с помощью FragmentManager.