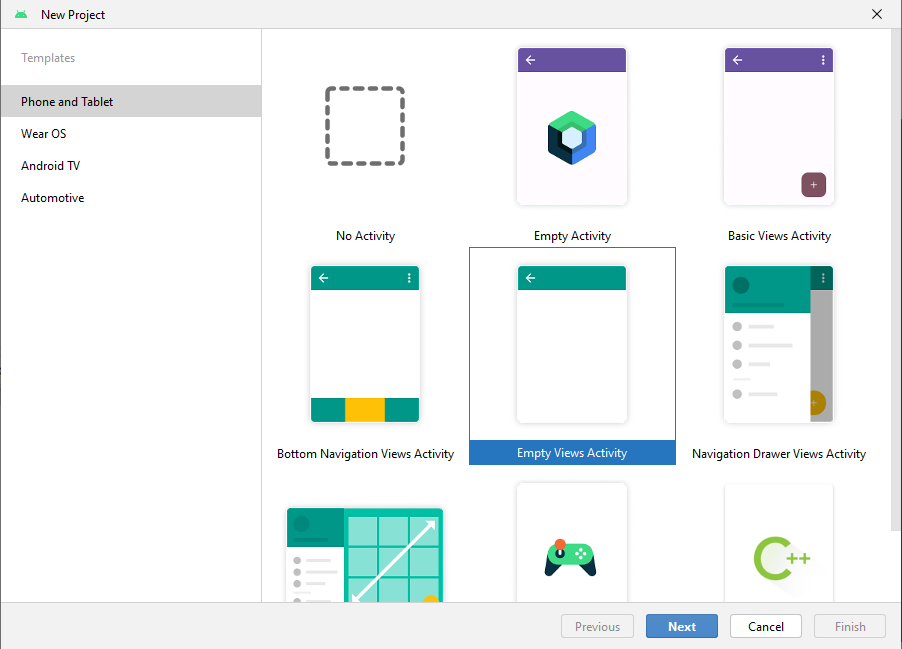
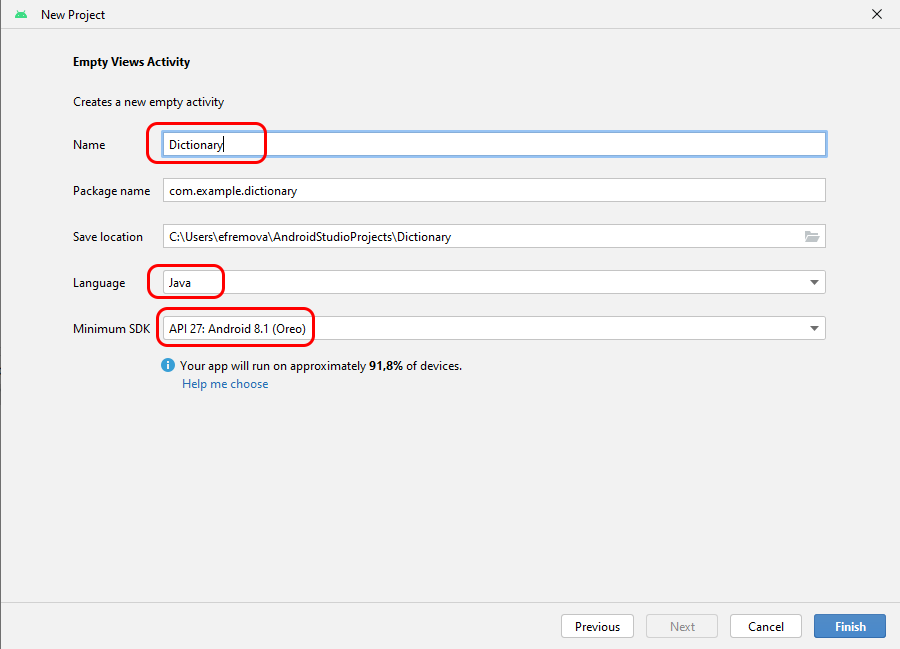
**Пример реализации визуальных списков при помощи RecyclerView**

1 Начало работы, общие действия

Создаем новый проект из шаблона Empty Views Activity:

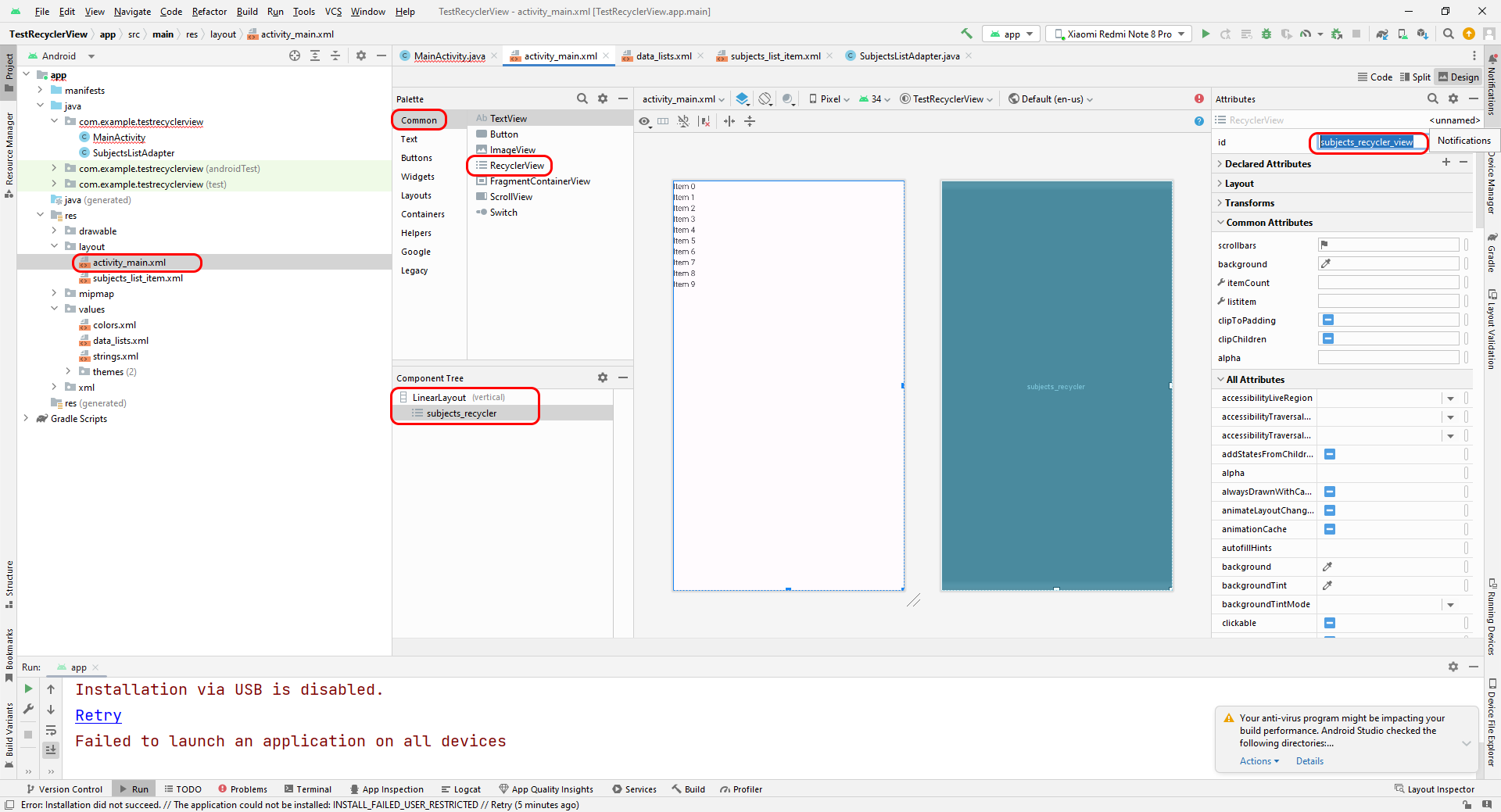


Обратите внимание, что лучше сразу задать нормальное имя проекта, если это начало вашего индивидуального проекта, имя изменить потом будет нельзя! Язык обязательно Java, а SDK не ставьте самый последний, возьмите 8 или 9, иначе будут проблемы при запуске и тестировании, поскольку наши планшеты не поддерживают последние SDK.

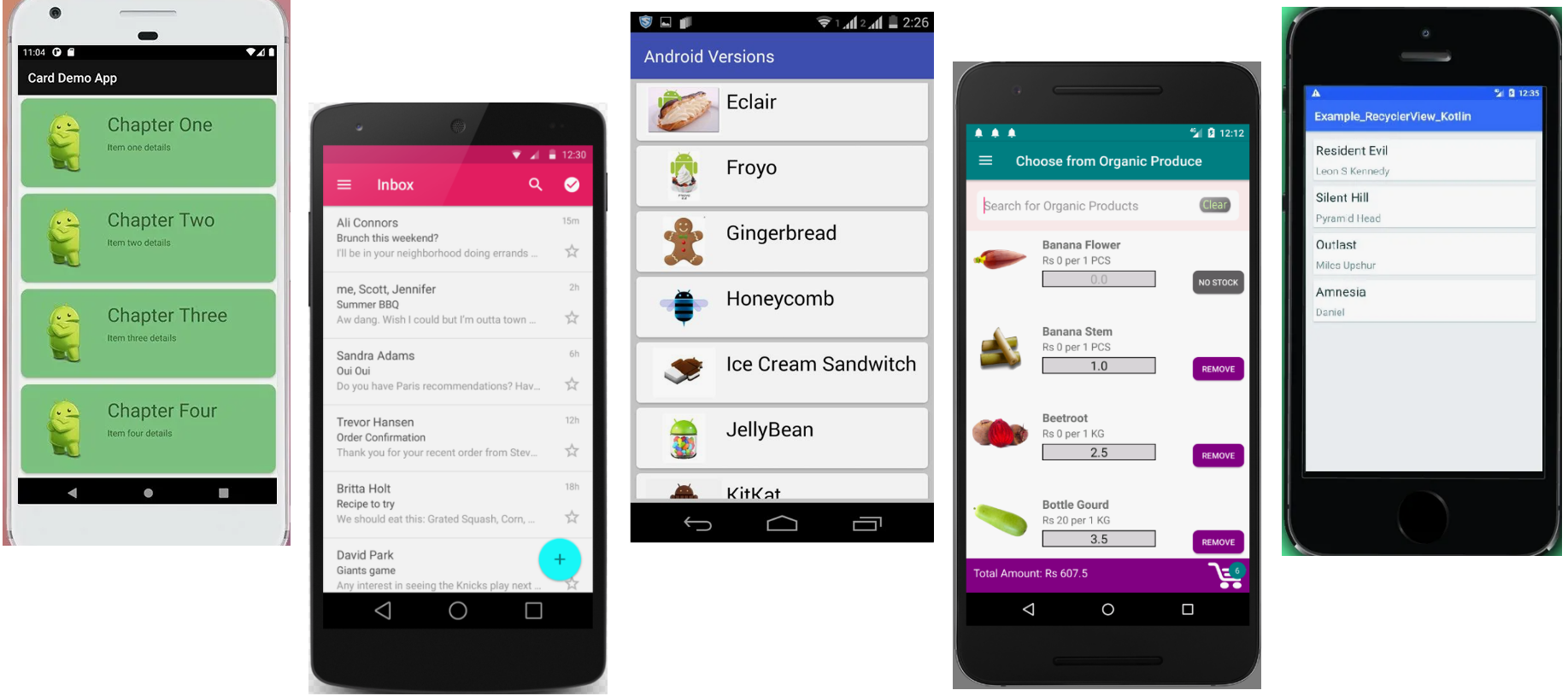


Далее пример будет разобран на главной активности, вы же можете разместить компонент RecyclerView на той активности или во фрагменте, где вам необходимо, логика реализации все равно останется.

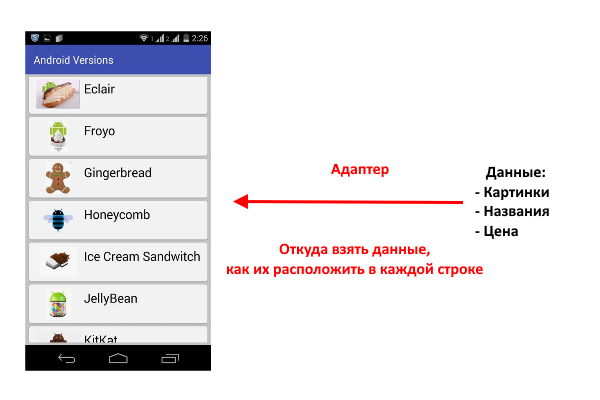
Размещаем компонент RecyclerView на главной активности в вертикальном LinearLayout (пример работы с LinearLayout в архиве ContactPage.zip). Обязательно в инспекторе нужно задать id для RecyclerView – «subjects\_recycler\_view».



В списке RecyclerView можно отобразить разнообразные данные – тест, картинки, ссылки, чекбоксы, поля ввода и др. Ниже на картинке примеры списков, которые можно реализовать при помощи RecyclerView. При этом можно встроить обработчик нажатия на элемент списка, обработчик ввода в тестовое поле и т.д. То есть это мощный компонент, при помощи которого можно реализовать любой список, начиная от простейшего текстового, заканчивая сложным списком с картинками и элементами управления.



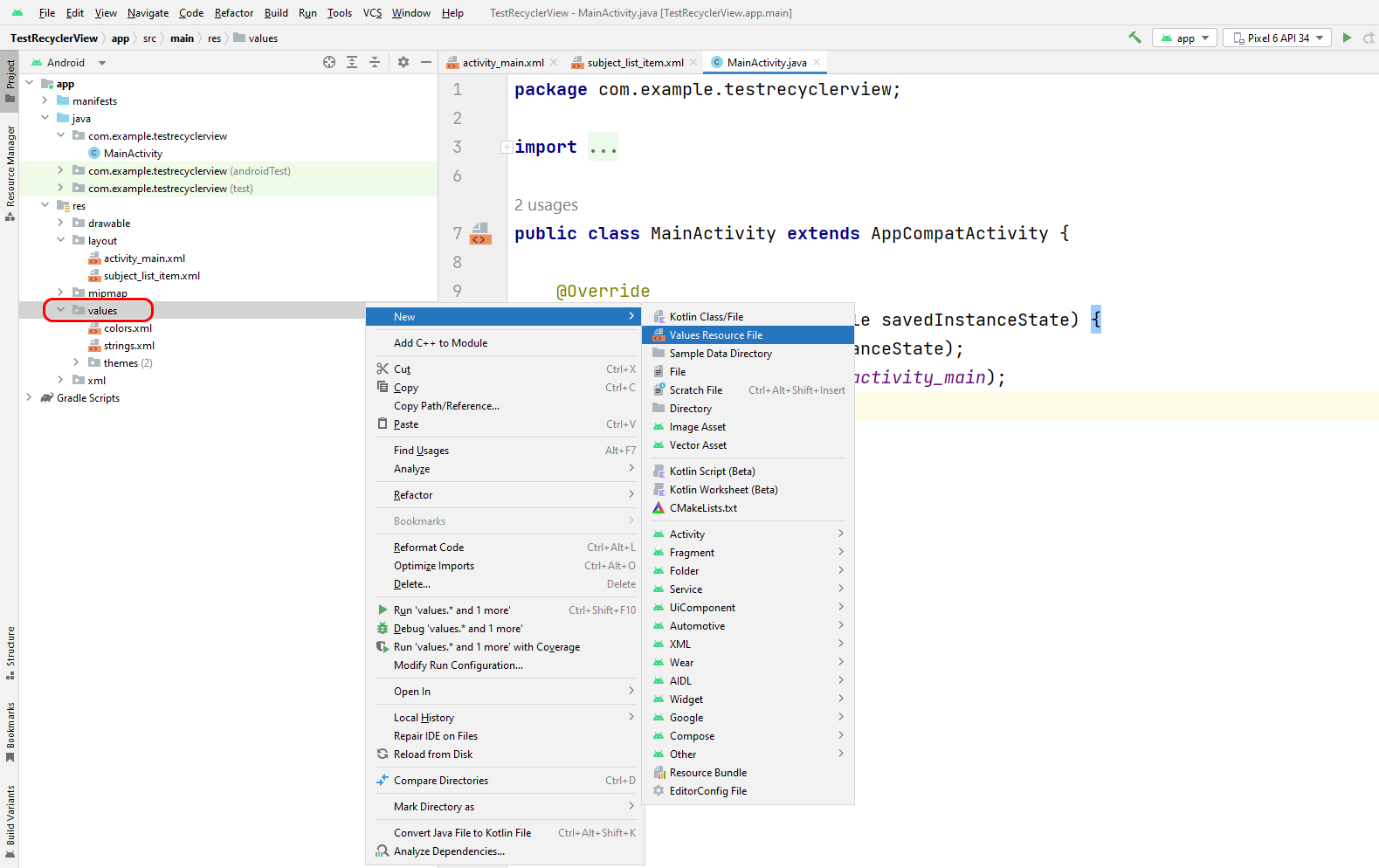
Особенность работы с RecyclerView заключается в том, что в таком списке визуальная часть списка и данные располагаются отдельно друг от друга и ничего не знают друг о друге. Ваша задача создать класс адаптера, в котором расписать, как будет выглядеть отдельная строка списка (все строки в списке будут выглядеть одинаково), где брать данные для заполнения списка, в какое поле какие данные нужно поместить. Именно за счет реализации этого адаптера каждый список может выглядеть абсолютно уникально.



2 Создание списка с данными

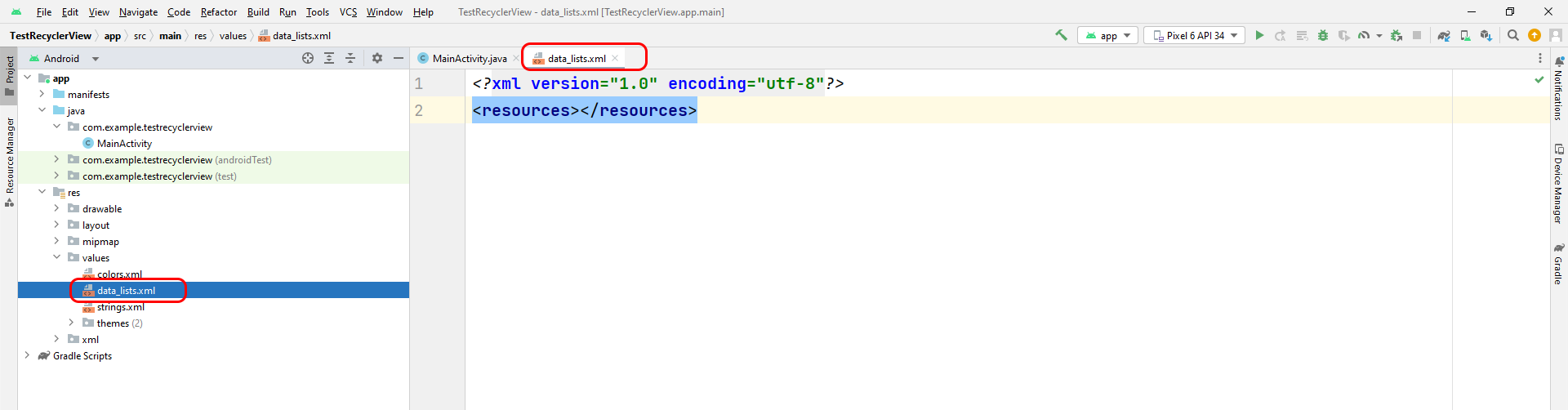
Для примера давайте попробуем создать список школьных предметов.

Для начала создадим сами данные. Для этого в папке res находим папку values и кликаем по ней правой клавишей мыши. В появившемся меню выбираем New -> Values Resource File



В появившемся окне вводим имя файла – data\_lists (данные списков), в котором будем хранить текстовые данные списков, которые понадобятся в нашем проекте.

Далее кликаем на кнопку OK и у нас создается пустой файл, в котором во второй строке стоят теги «resources», внутри которых мы и будем писать данные списка. Если непонятна теговая структура, то почитайте про файлы xml в разделе «2.5. Интерфейс Android-приложения».

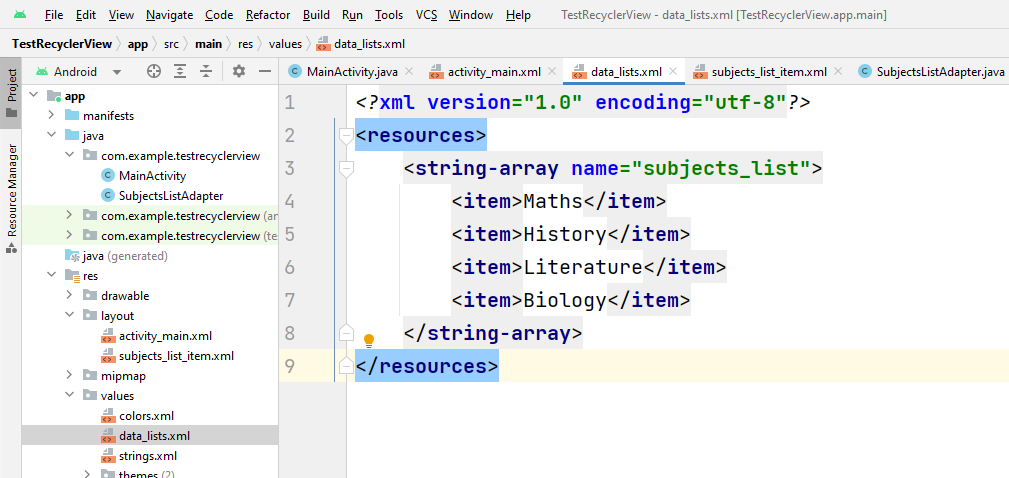


Добавим список школьных предметов следующим образом:

*<?***xml version="1.0" encoding="utf-8"***?>*<**resources**>  
 <**string-array name="subjects\_list"**>  
 <**item**>Maths</**item**>  
 <**item**>History</**item**>  
 <**item**>Literature</**item**>  
 <**item**>Biology</**item**>  
 </**string-array**>  
</**resources**>

Опять же добавление списка делается по определенным правилам – обязательно тег <**string-array/**>, внутри которого через теги <**item**/> происходит добавление каждого элемента списка.

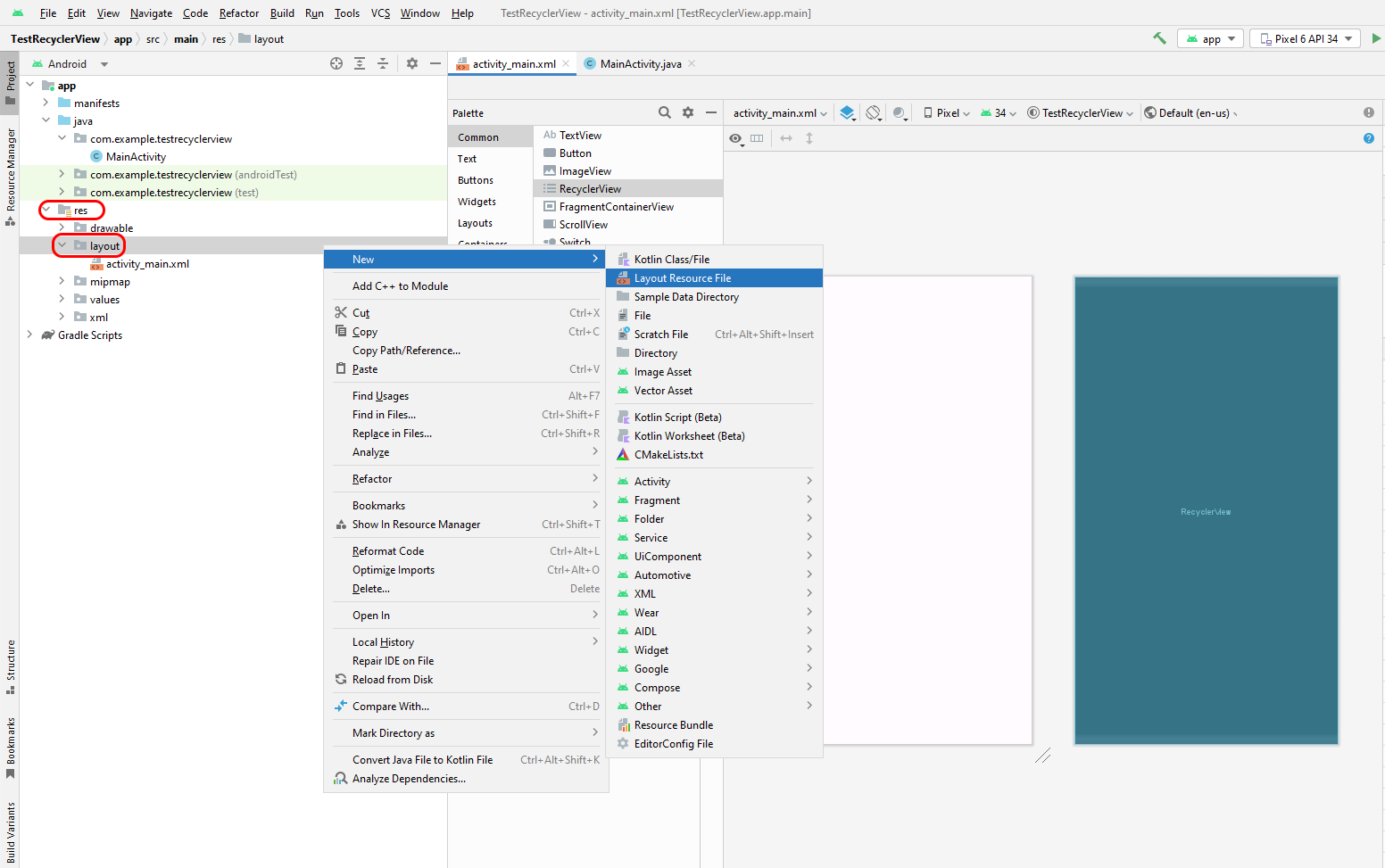
Так выглядит файл в Android Studio:



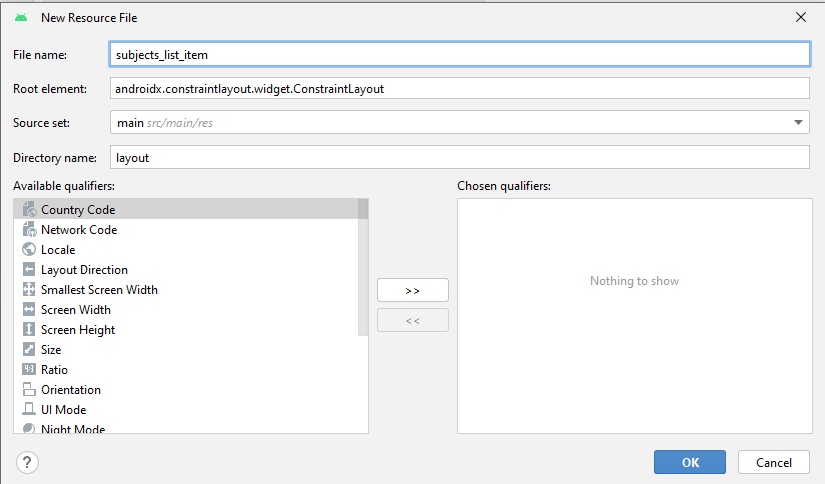
Наши данные готовы, пока это только текстовый список школьных предметов.

3 Создание разметки строки для визуального списка

Теперь создадим класс разметки для элемента строки списка, то есть как будет выглядеть каждая строка в нашем списке. Для этого в папке res находим папку layout и кликаем по ней правой клавишей мыши. В появившемся меню выбираем New -> Layout Resource File



В появившемся окне настроек файла введите имя, которое будет отражать название списка, для которого вы создаете разметку строки. Если в проекте у вас будет несколько списков, вам элементарно будет легче ориентироваться в этих файлах. Мы создаем разметку строки для списка школьных предметов, поэтому назовем файл subjects\_list\_item (элемент списка предметов). Обратите внимание, что для файлов разметки не используются заглавные буквы, а только строчные и слова разделяются символом подчеркивания – это правило, к которому нужно привыкнуть.



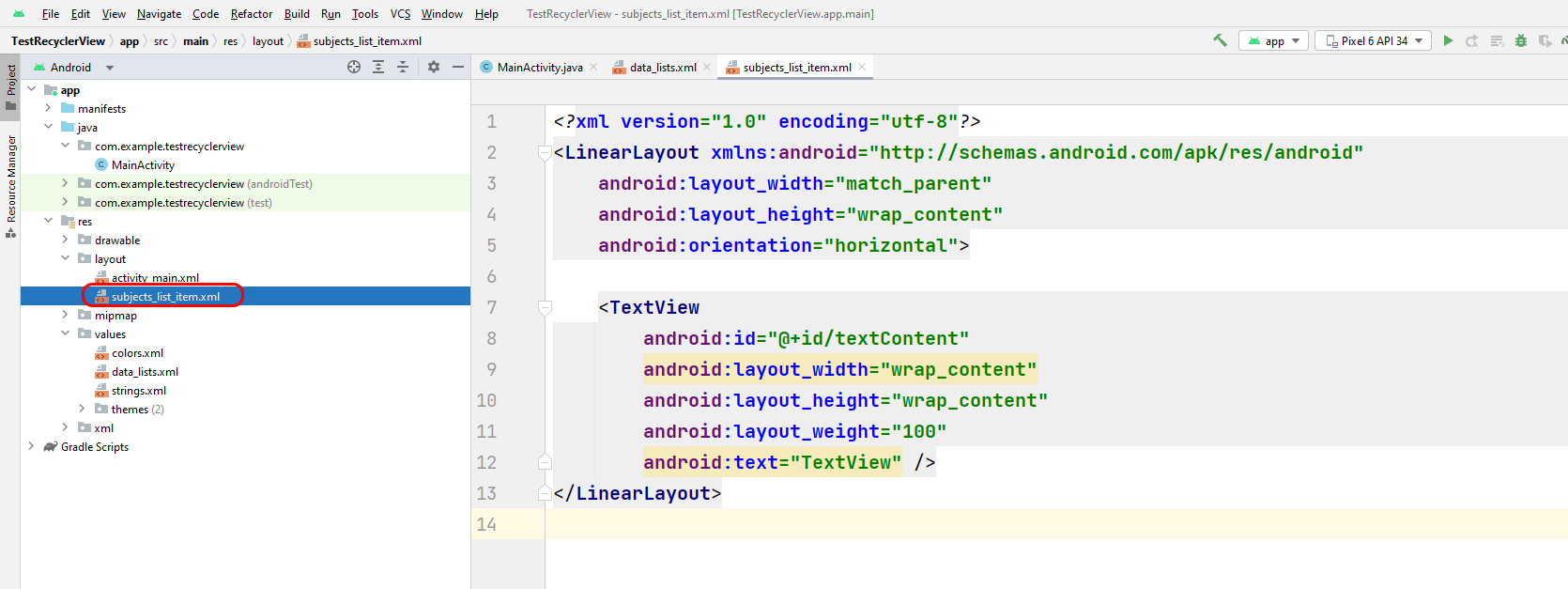
Больше ничего не нужно менять, просто нажимаем кнопку Ok.

Создается файл разметки, в котором нам нужно теперь создать разметку для одной строки списка! Это важно понимать, мы не создаем здесь весь список, а только одну строку!

Лучше всего делать это на основе горизонтального LinearLayout. Давайте начнем с простого тестового списка, поэтому в разметку поместим только одно текстовое поле:

*<?***xml version="1.0" encoding="utf-8"***?>*<**LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:orientation="horizontal"**>  
  
 <**TextView  
 android:id="@+id/textContent"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_weight="100"  
 android:text="TextView"** />  
</**LinearLayout**>

Так это будет выглядеть в Android Studio:

****

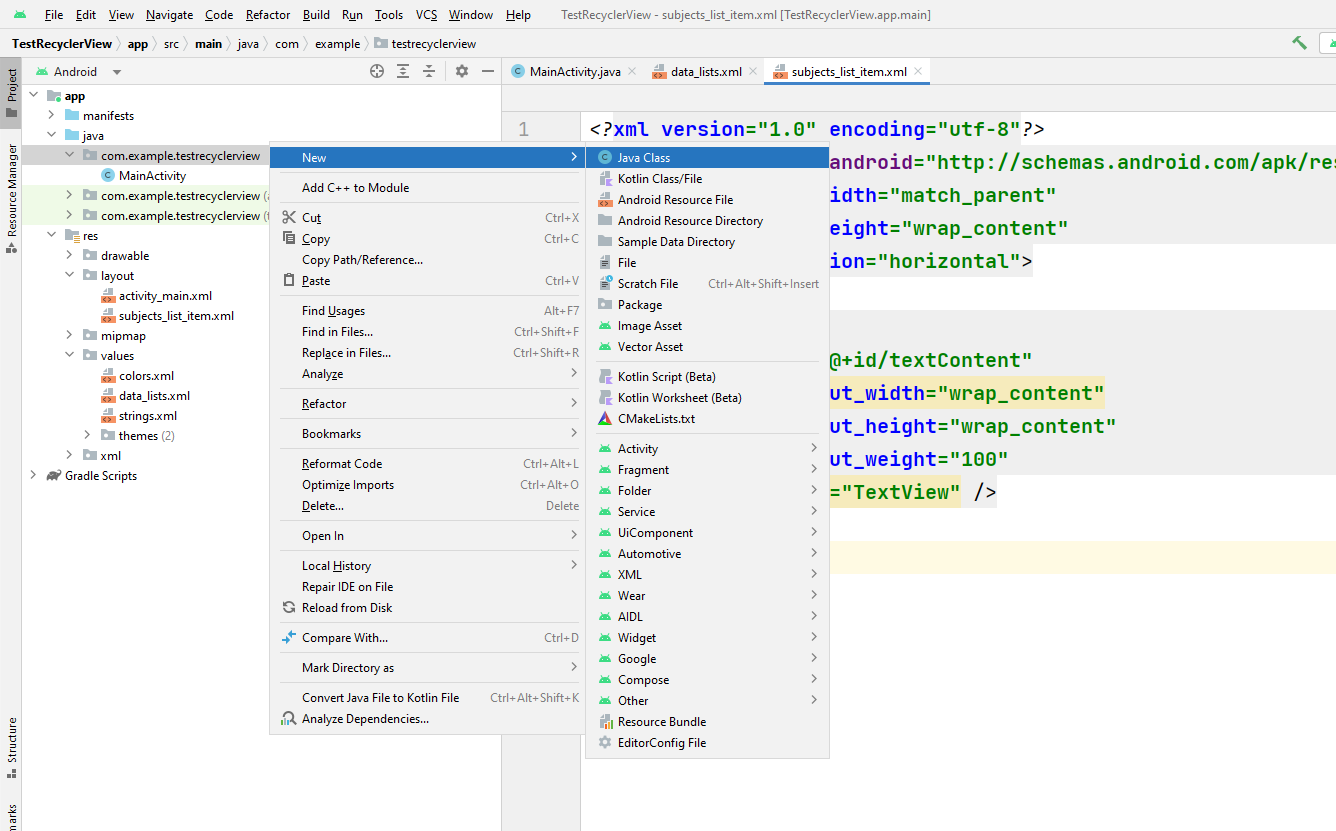
Грубо говоря логика работы списка RecyclerView будет такая:

* RecyclerView связывается с созданным адаптером
* адаптер сообщает RecyclerView, что для него есть данные в списке **subjects\_list** в файле ресурсов **data\_lists**, в количестве 4 строк.
* также адаптер сообщает RecyclerView, что каждая строка должна соответствовать разметке, созданной в файле subjects\_list\_item.xml
* RecyclerView создает 4 строки, каждая из которых выглядит одинаково как прописано в файле разметки subjects\_list\_item.xml
* далее адаптер начинает заполнять визуальный список данными. Берет первую строку, ищет в ней поле «textContent», размещает в него первый элемент списка **subjects\_list**. Затем переходит ко второй строке, ищет в ней поле «textContent», размещает в него второй элемент списка **subjects\_list**. И так далее…

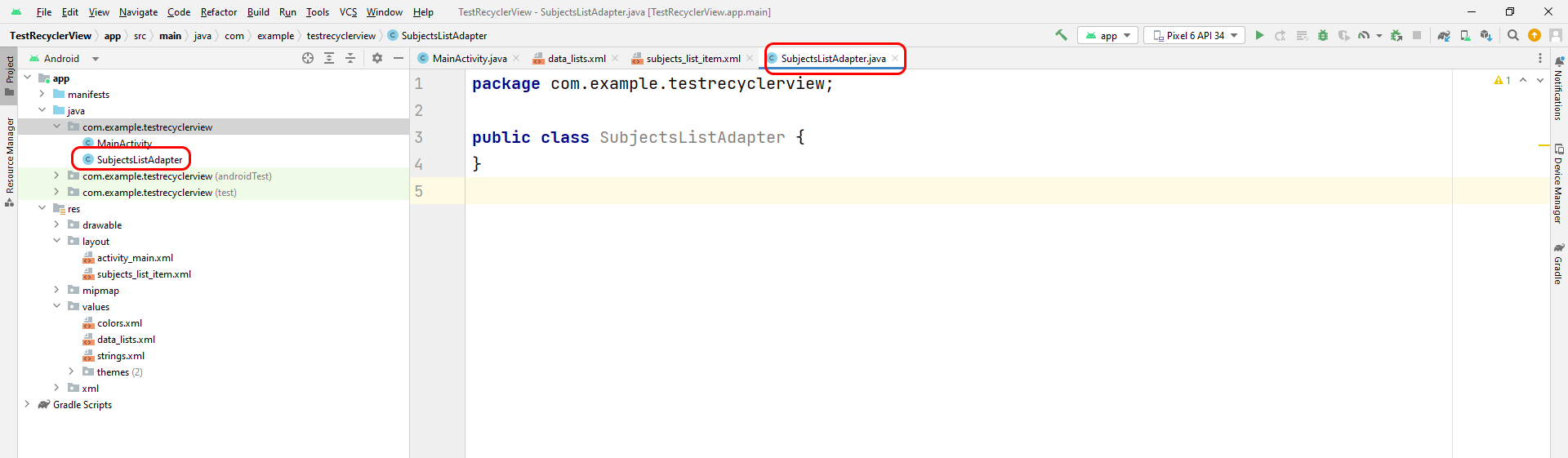
4 Создание класса адаптера

Теперь давайте создадим класс адаптера, в котором пропишем, что данные необходимо брать из списка **subjects\_list**, и помещать каждый элемент из этого списка в тестовое поле «textContent» из разметки строки.

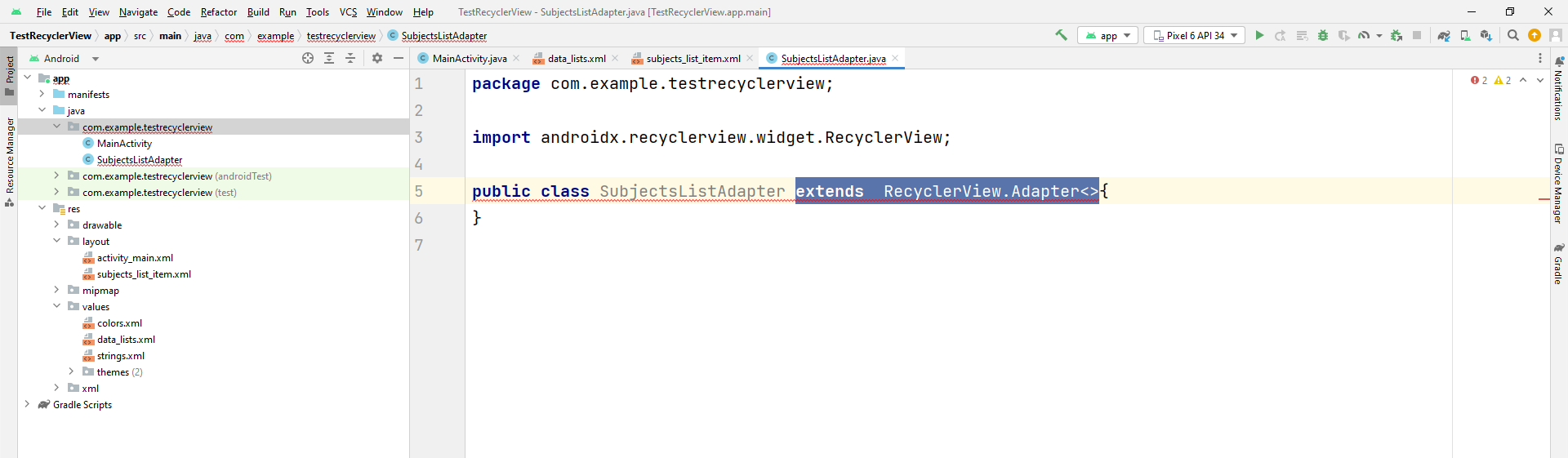
Создаем новый класс **SubjectsListAdapter**:

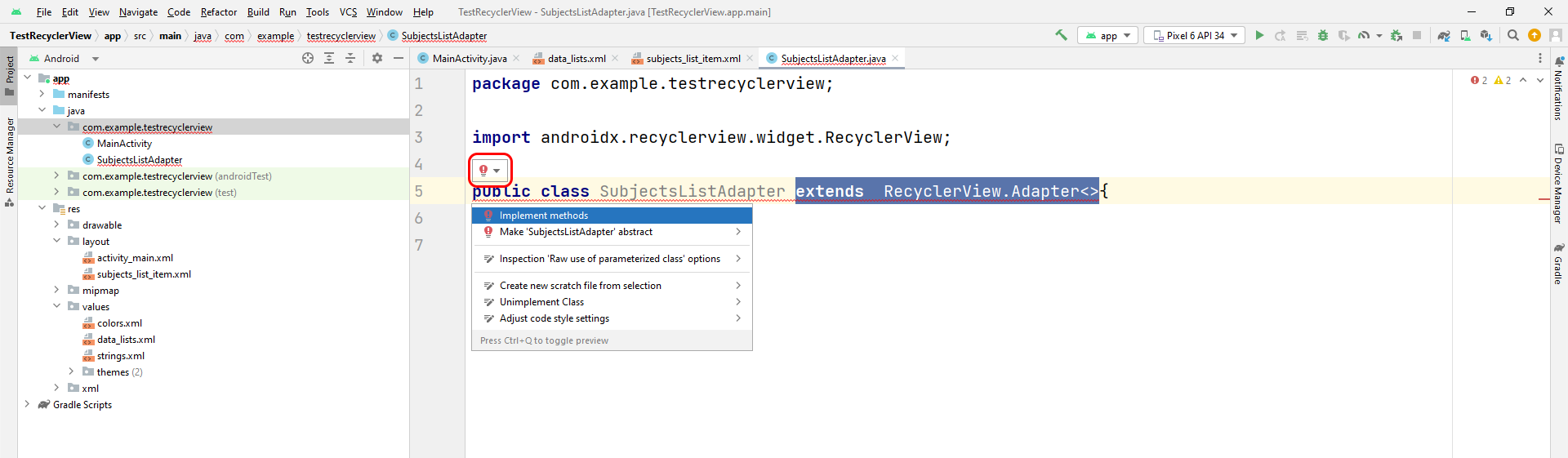
****

Файл класса:

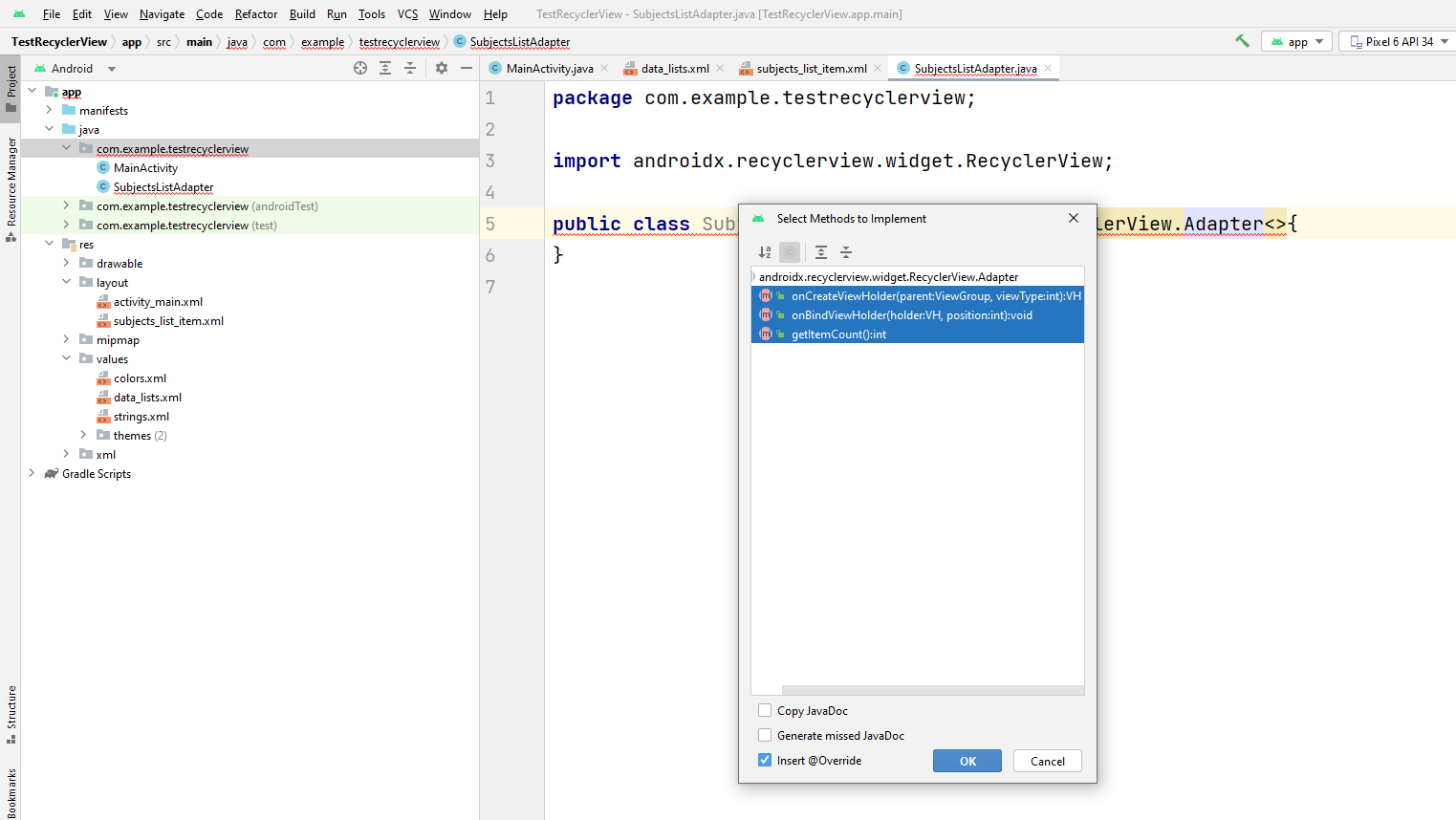
****

Модифицируем этот класс. Класс адаптера обязательно должен наследоваться от класса RecyclerView.Adapter, допишите это следующим образом:

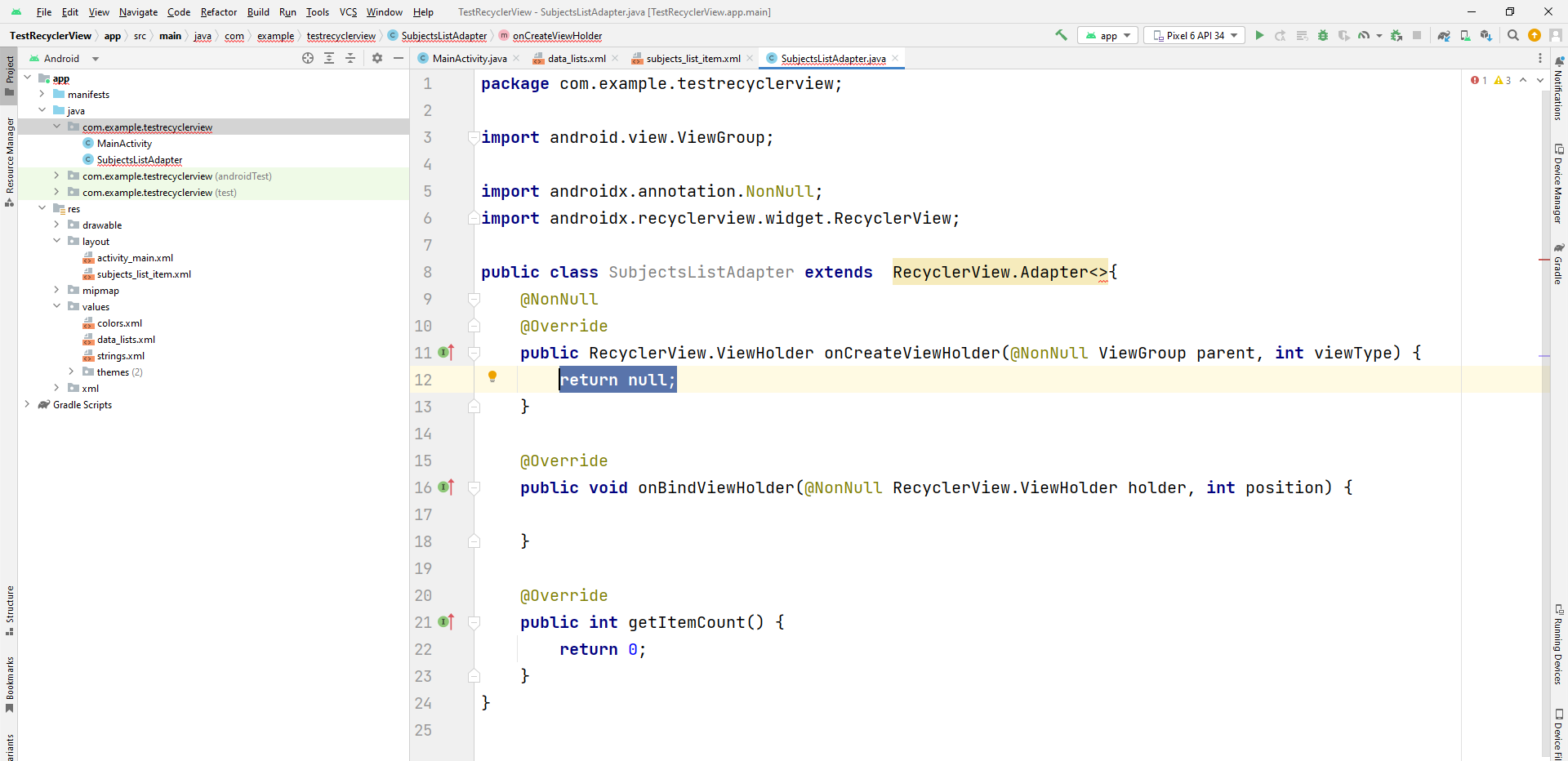


Теперь наведите курсор на красную лампочку, кликните по стрелочке рядом с ней и выберите пункт Implements methods. 

Если непонятен момент про переопределение методов, то почитайте раздел «2.6. Наследование и полиморфизм в Java». По факту есть абстрактный класс адаптера для RecyclerView, для работы которого обязательно нужно реализовать несколько методов, список которых и появится при выборе пункта «Implements methods»:

****

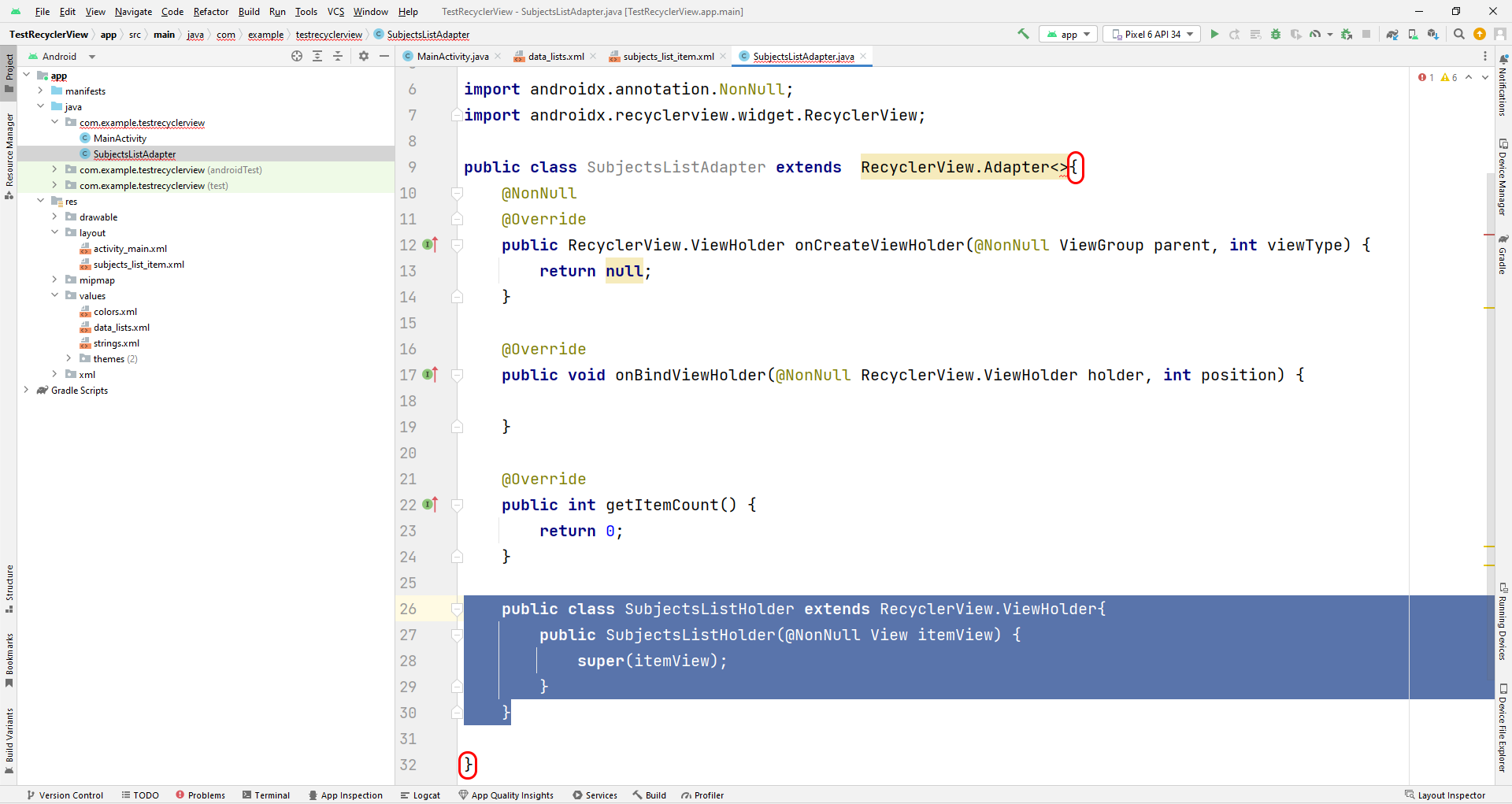
Кликаем по кнопке ОК в окне списка методов для переопределения. У нас в коде появятся шаблоны этих методов:



Помимо самого адаптера нам понадобится вспомогательный класс, который будет связываться с созданной нами ранее разметкой строки и разобьет эту разметку на отдельные элементы. То есть класс самого адаптера как бы обрабатывает весь список целиком, а нам нужен вспомогательный класс для обработки конкретной строки. Этот вспомогательный класс будет связан с классом адаптера **SubjectsListAdapter**, поэтому мы опишем его прямо в этом же файле.

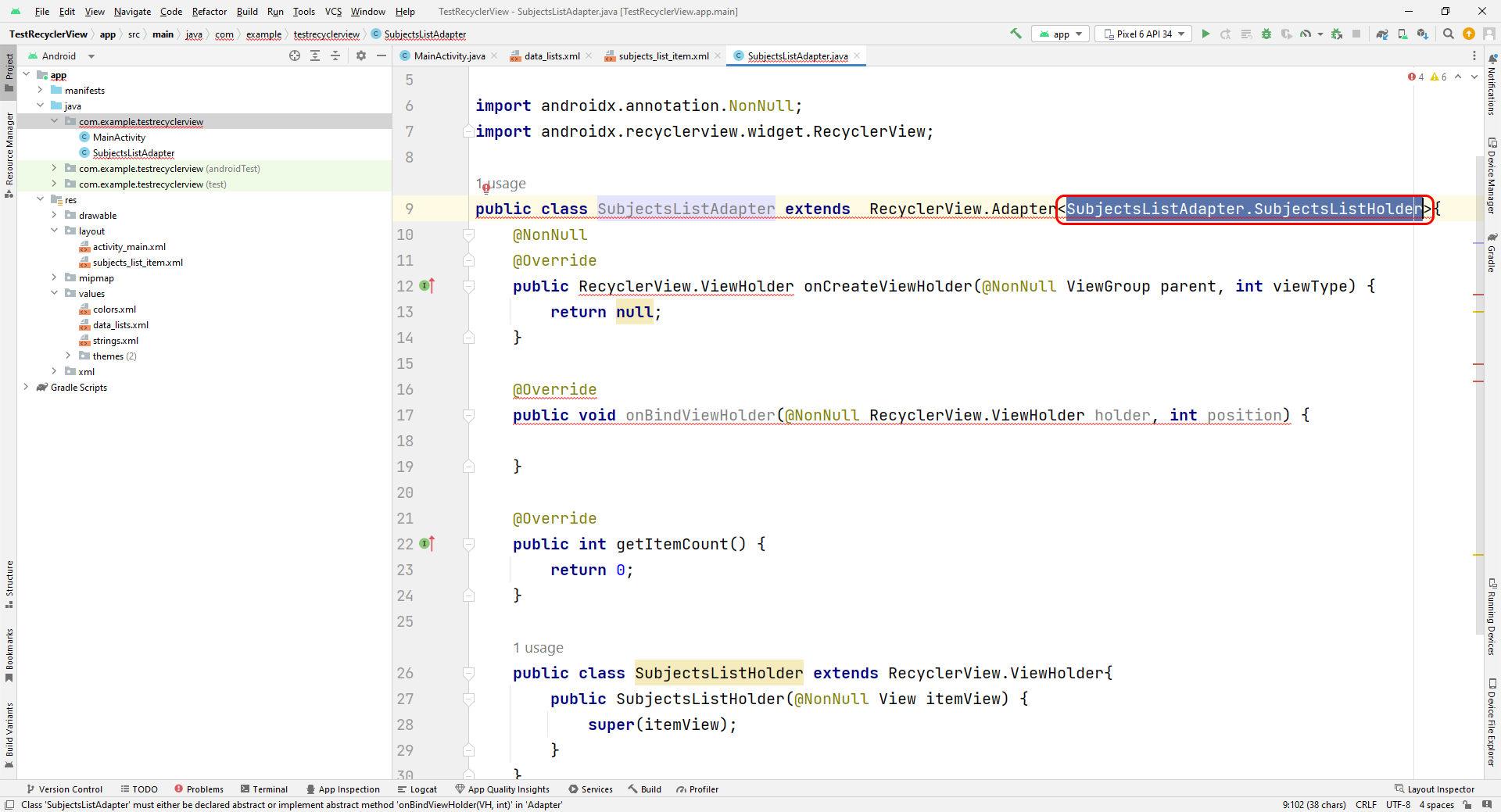
**public class** SubjectsListHolder **extends** RecyclerView.ViewHolder{  
 **public** SubjectsListHolder(@NonNull View itemView) {  
 **super**(itemView);  
 }  
}

Обратите внимание, что класс является внутренним для класса SubjectsListAdapter, то есть находится внутри него:

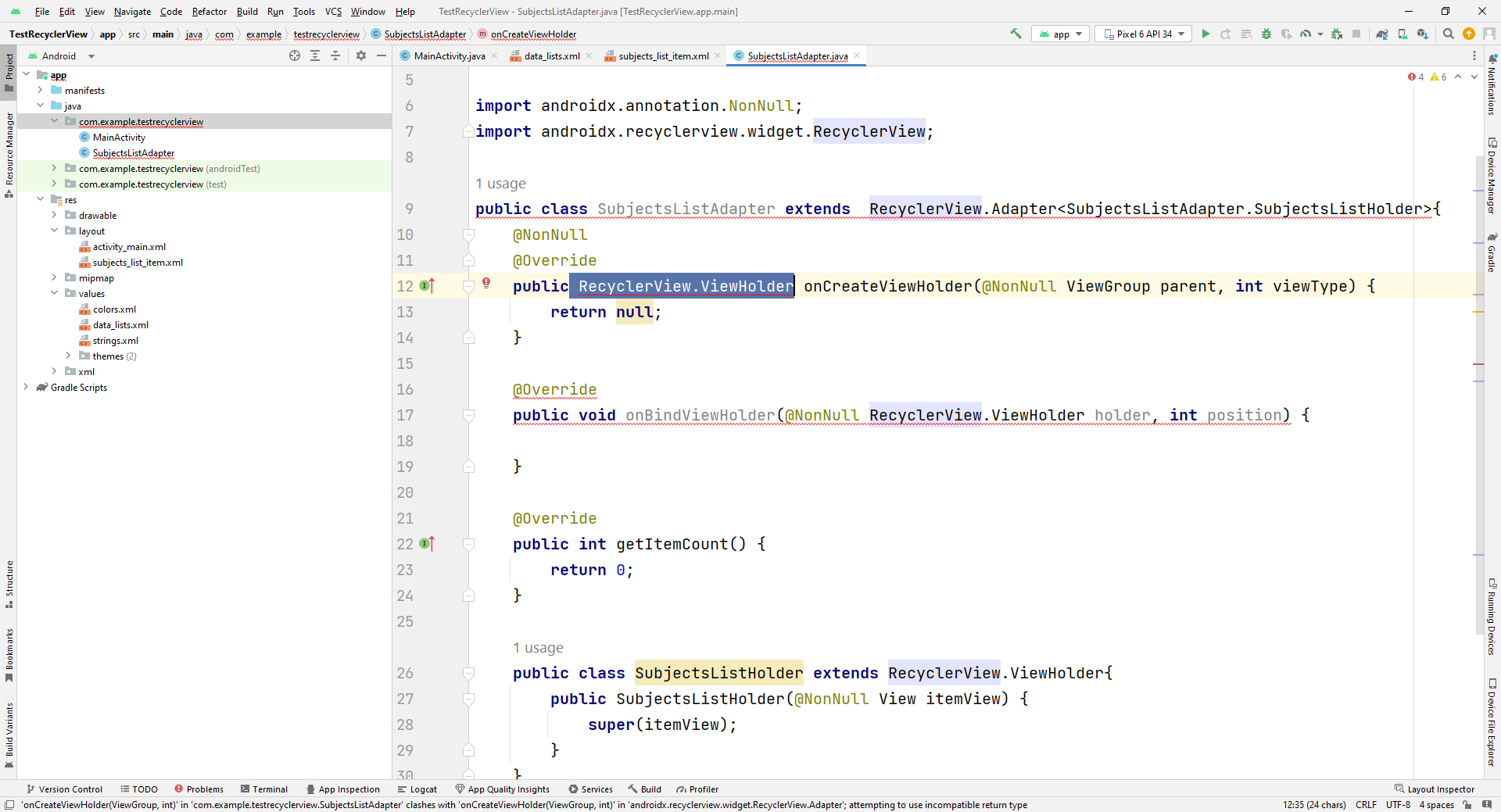


Класс SubjectsListHolder еще не дописан, по сути, это пока просто пустой шаблон, в котором ничего не происходит. Но пока оставим его и сначала доработаем класс адаптера, включив в него привязку к классу SubjectsListHolder, чтобы наш адаптер знал, что каждая строка в списке будет формироваться согласно SubjectsListHolder.

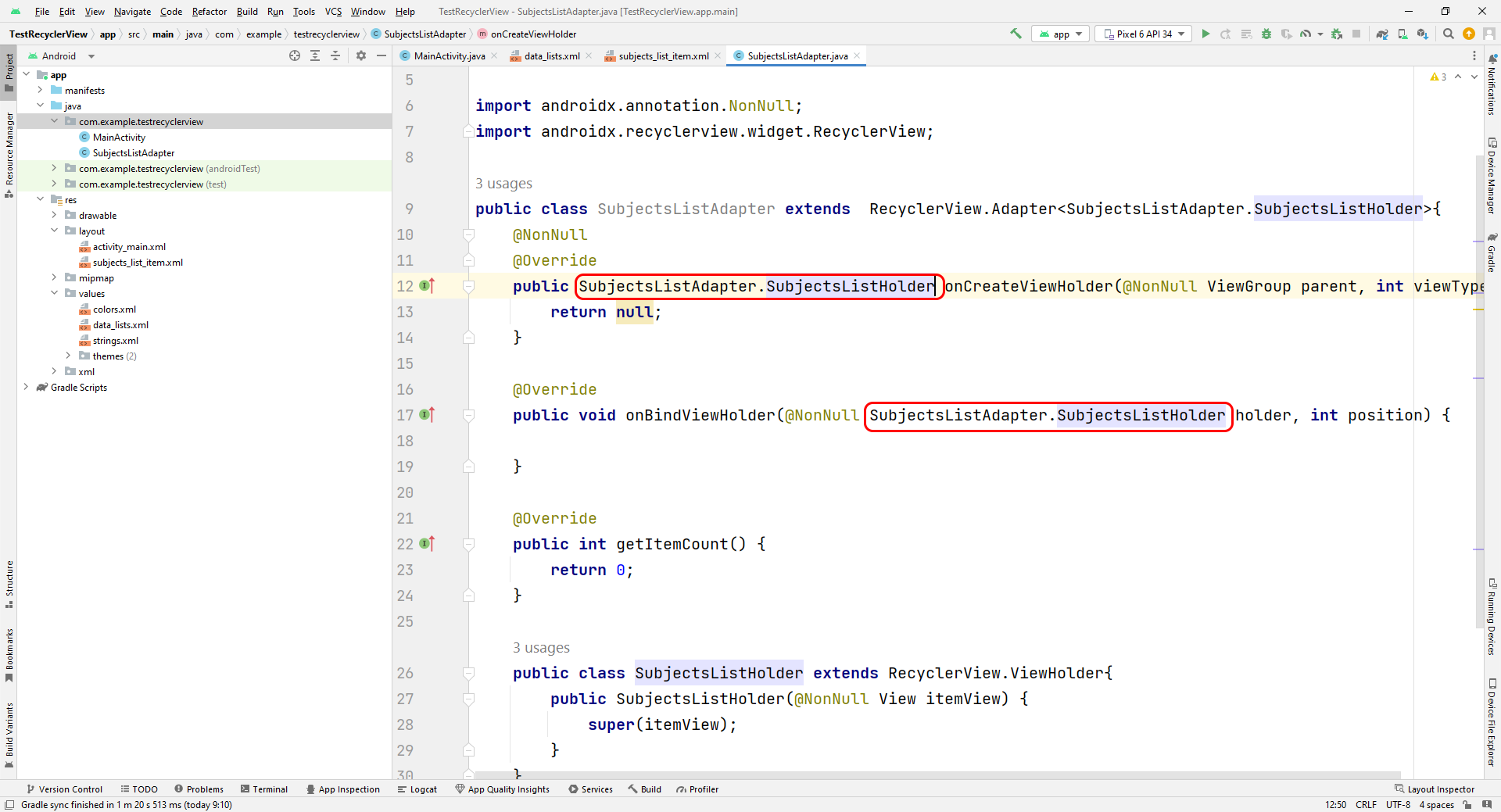
Для этого в классе адаптера в угловых скобках необходимо указать, что адаптер будет использовать класс SubjectsListHolder. Обратите внимание, что поскольку класс является внутренним, то мы сначала указываем класс самого адаптера, а через точку класс холдера:



Теперь аккуратно исправим все ошибки. По умолчанию класс адаптера ссылается на стандартный класс RecyclerView.ViewHolder:



Но у нас есть свой класс, отвечающий за формирование строк списка – SubjectsListHolder, поэтому нужно везде заменить ссылки на стандартный Holder на ссылки на наш класс:

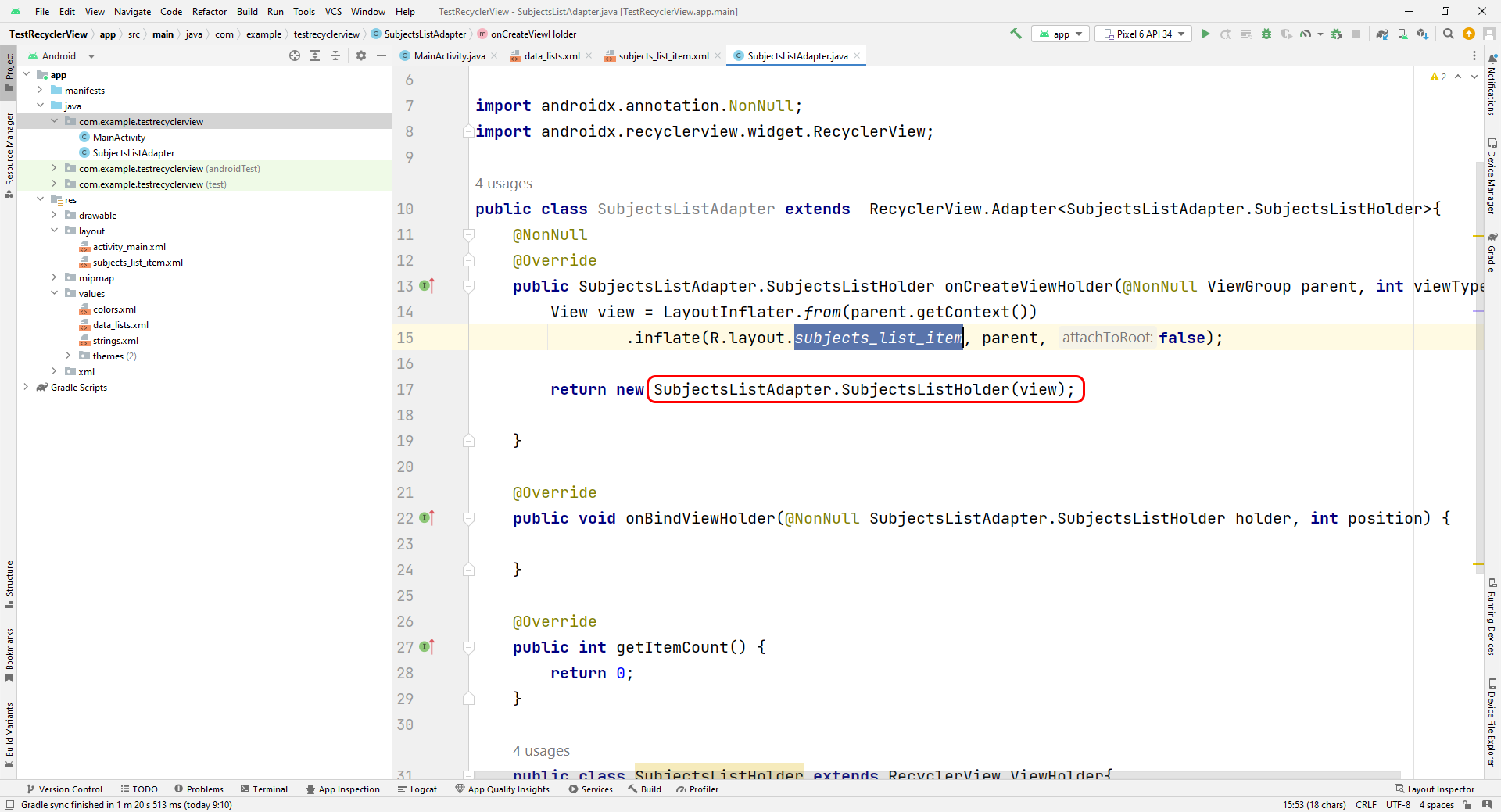


На этом этапе все ошибки должны пропасть. Но работать пока еще ничего не будет.

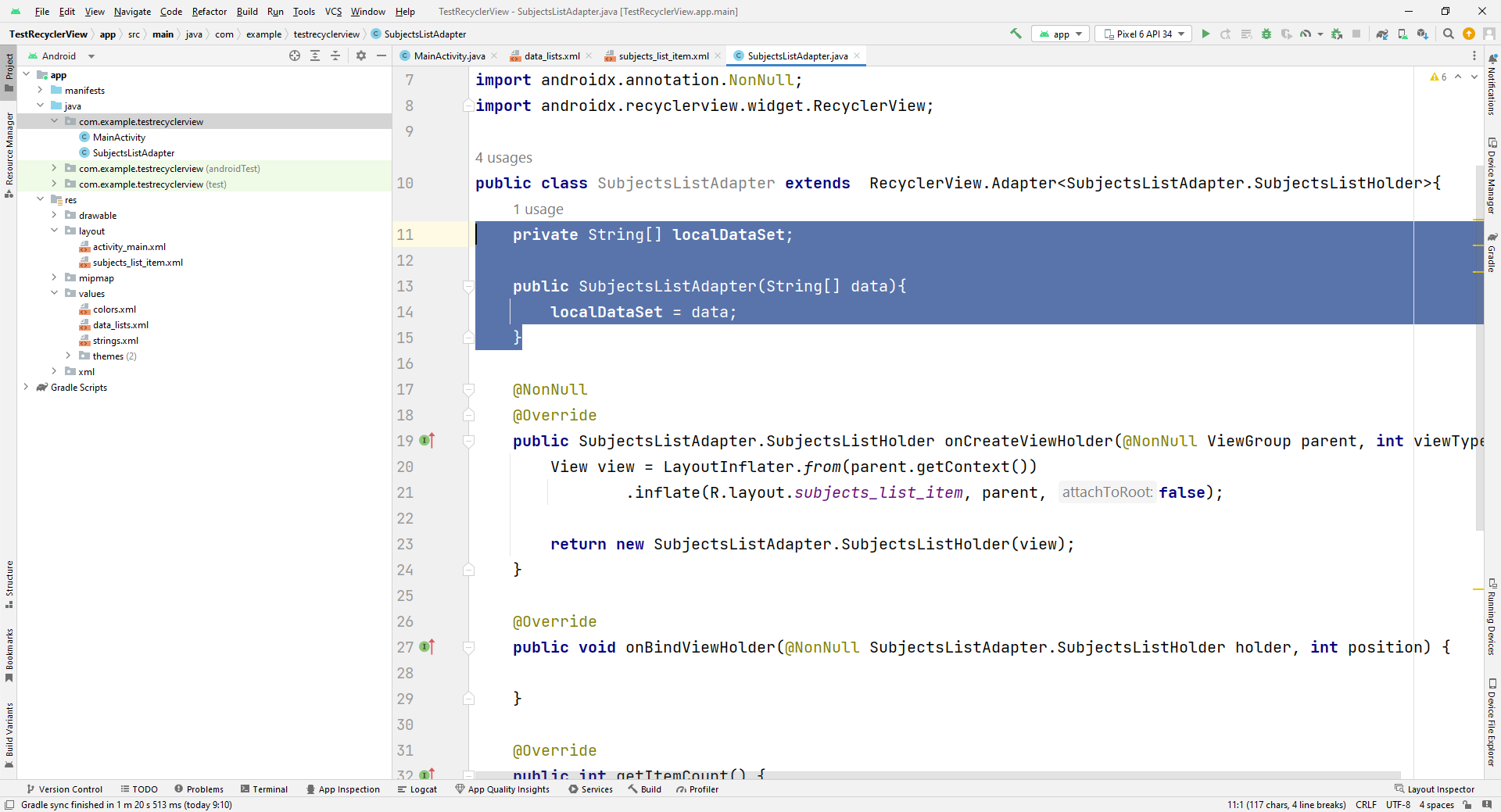
Следующий шаг – в методе адаптера onCreateViewHolder нужно создать вьюшку (визуальное представление) нашей строки согласно классу SubjectsListHolder:

View view = LayoutInflater.*from*(parent.getContext())  
 .inflate(R.layout.*subjects\_list\_item*, parent, **false**);  
  
**return new** SubjectsListAdapter.SubjectsListHolder(view);

В первой строке мы создаем вьюшку строки согласно разметке subjects\_list\_item, а во второй строке возвращаем ее, поскольку такова сигнатура метода onCreateViewHolder:



Теперь давайте добавим в класс адаптера привязку к данным. Мы создаем заранее определенный список школьных предметов, поэтому нам нужен простой массив строковых переменных. Сразу добавим конструктор класса адаптера, где входным параметром будем передавать этот список. Получается в классе активности или фрагмента мы можем эти данные взять из ресурсов, а затем передать их в класс адаптера списка. Это стандартный подход, поскольку доступ к ресурсам приложения мы можем получить только в классе активности или фрагмента, об этом тоже нужно просто знать и помнить.



Теперь доработаем метод getItemCount(), который должен возвращать общее количество строк списка. В нашем случае количество строк равно количеству элементов массива **localDataSet:**



Теперь вернемся к классу SubjectsListHolder и опишем в нем разбиение разметки на отдельные элементы.

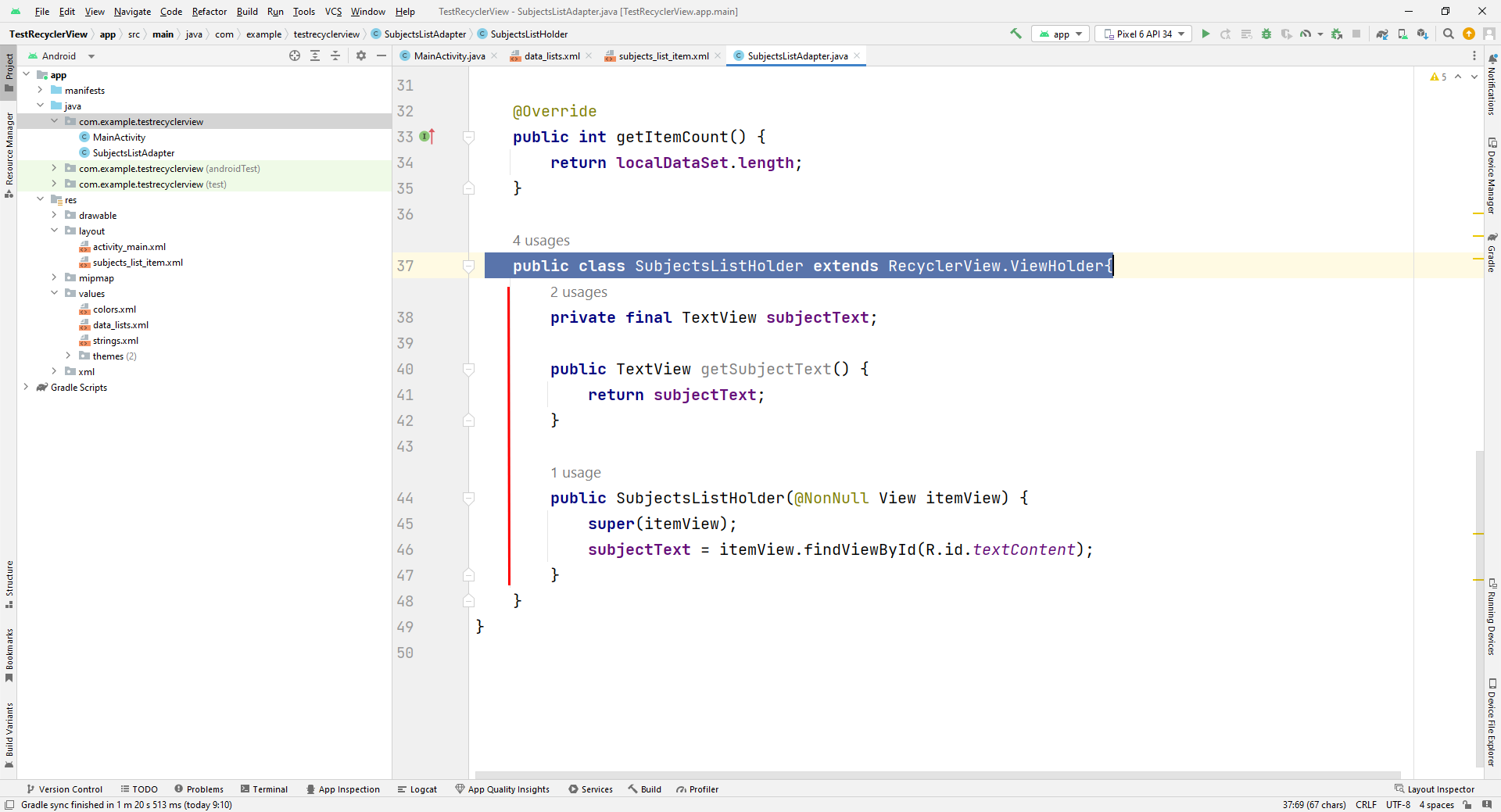
Поскольку в разметке у нас только текстовое поле, то нам понадобится переменная для ссылки на него (строка 37 на скрине ниже). Также необходим метод для получения этого поля, метод

**public** TextView getSubjectText() {  
 **return subjectText**;  
}

И необходимо доработать конструктор этого класса, где как раз и происходит связывание отдельных элементов разметки с переменными.

**public** SubjectsListHolder(@NonNull View itemView) {  
 **super**(itemView);  
 **subjectText** = itemView.findViewById(R.id.*textContent*);  
}

В Android Studio это будет выглядеть вот так:



Получается класс адаптера ничего не знает про разметку, он просто будет использовать метод getSubjectText(), чтобы обратиться к нужному полу для размещения данных. А вот класс SubjectsListHolder ничего не знает про данные, его задача просто выискивать в разметке нужный компонент и возвращать его для заполнения.

Осталось доработать метод onBindViewHolder, добавив в него код:

holder.getSubjectText().setText(**localDataSet**[position]);



Этот метод onBindViewHolder вызывается для каждой строки, чтобы заполнить ее данными. На вход подается параметр SubjectsListAdapter.SubjectsListHolder holder, и **int** position, то есть позиция заполняемой строки. Мы вызываем для переменной holder метод обращения к текстовому полю и помещаем в это поле элемент под номером position из массива данных localDataSet.

5 Связывание визуального списка и адаптера

Теперь в классе активности нужно связать созданный нами класс адаптера и визуальный список RecyclerView.

RecyclerView recyclerView = findViewById(R.id.*subjects\_recycler\_view*);  
String[] data = getResources().getStringArray(R.array.*subjects\_list*);  
SubjectsListAdapter subjectsListAdapter = **new** SubjectsListAdapter(data);  
recyclerView.setLayoutManager(**new** LinearLayoutManager(**this**));  
recyclerView.setAdapter(subjectsListAdapter);

В первой строке связываемся по id с компонентом RecyclerView. Во второй строке считываем в массив data данные из списка. Как уже упоминалось, метод getResources() можно вызвать только из активности или фрагмента.

В третьей строке создаем класс адаптера, в четвертой задаем компоновку списка – список у нас линейный. И в пятой строке указываем, что recyclerView будет использовать наш адаптер.

Теперь запустим приложение и на экране увидим список предметов:



Конечно, это самый простой список, но его легко можно трансформировать на более сложные.

Например, добавим картинку и описание предмета. То есть у нас уже появляется структура:

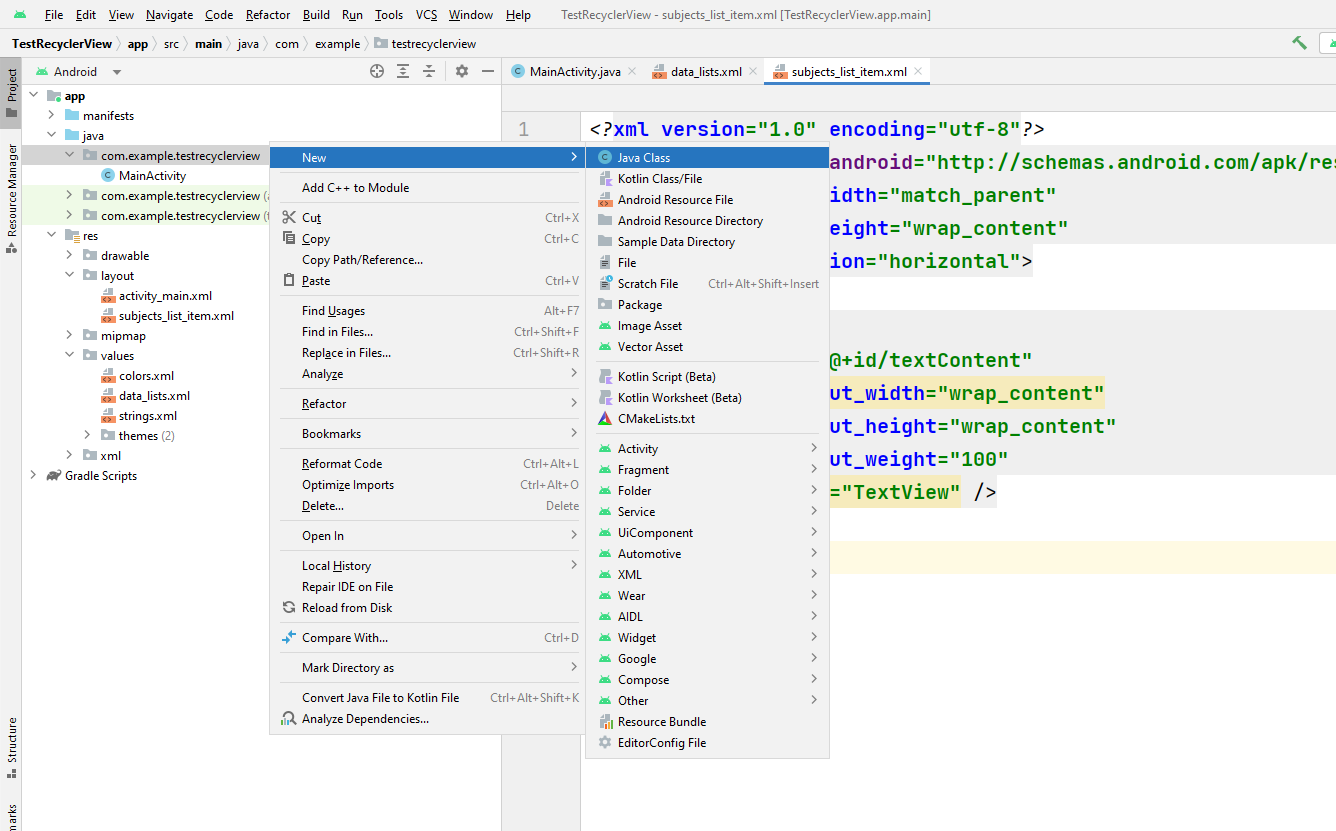
- название предмета

- изображение

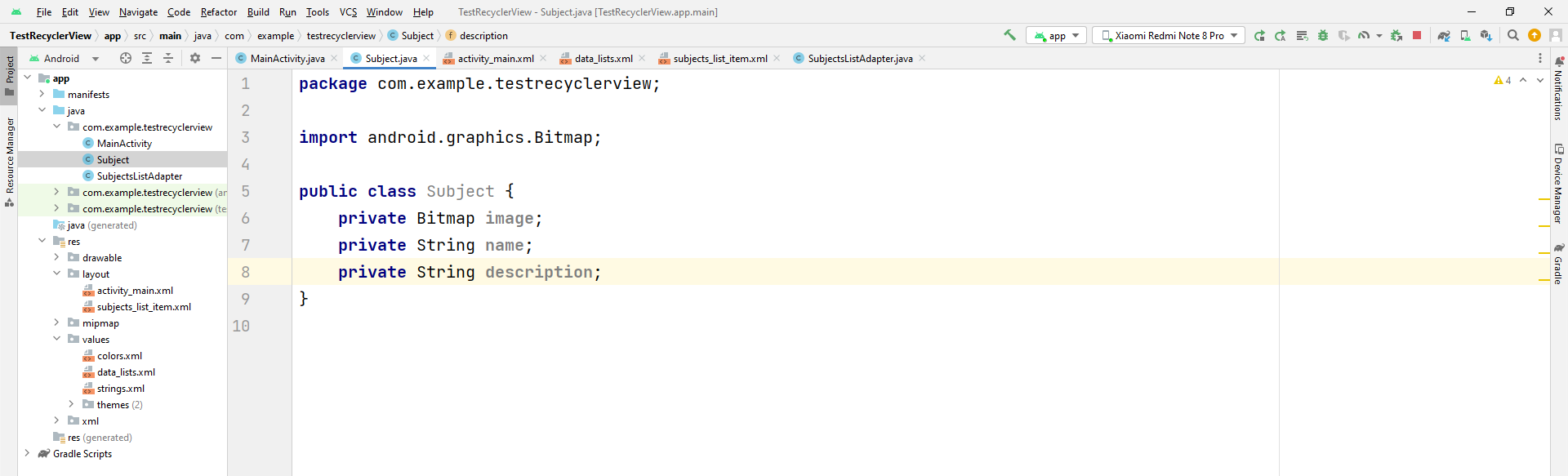
- описание

Чтобы сохранить эти все данные, логично написать отдельный класс, который будет хранить информацию по отдельному предмету, а затем формировать массив не просто строковых переменных, а массив экземпляров класса.

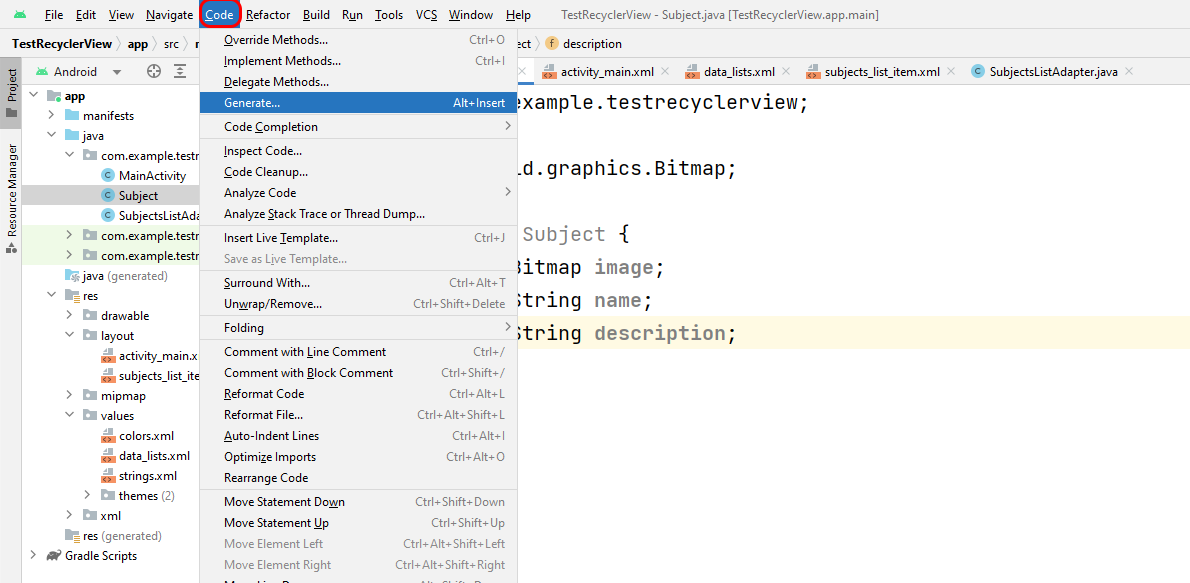
Создадим класс Subject. Класс создаем пять при помощи контекстного меню:

****

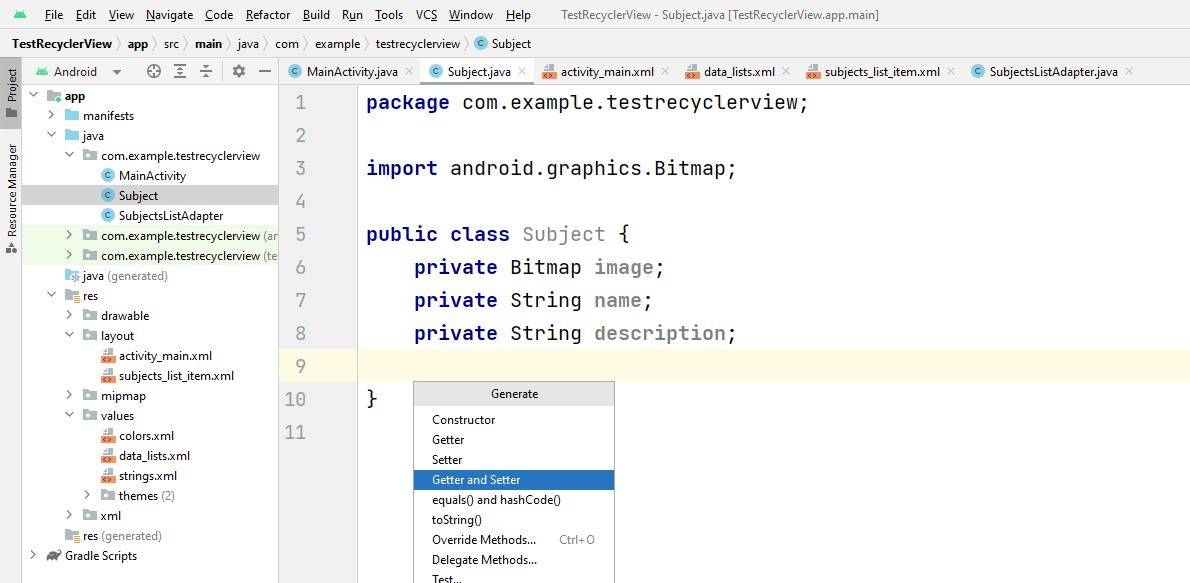
Добавим в класс необходимые поля. Хранить картинки можно с типом Bitmap. Обратите внимание, что все члены класса у нас закрытые:



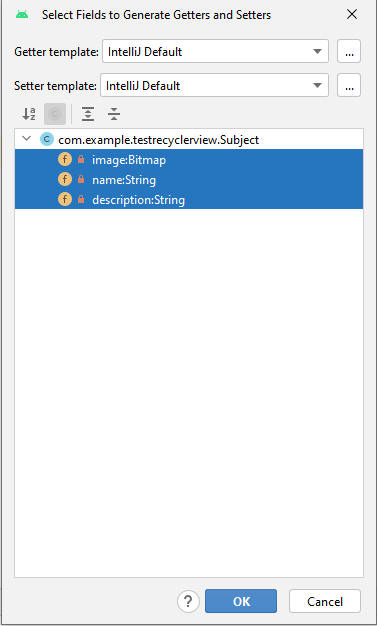
Для доступа к полям создадим геттеры и сеттеры. Быстро сгенерировать эти методы можно при помощи пункта меню Genereate:



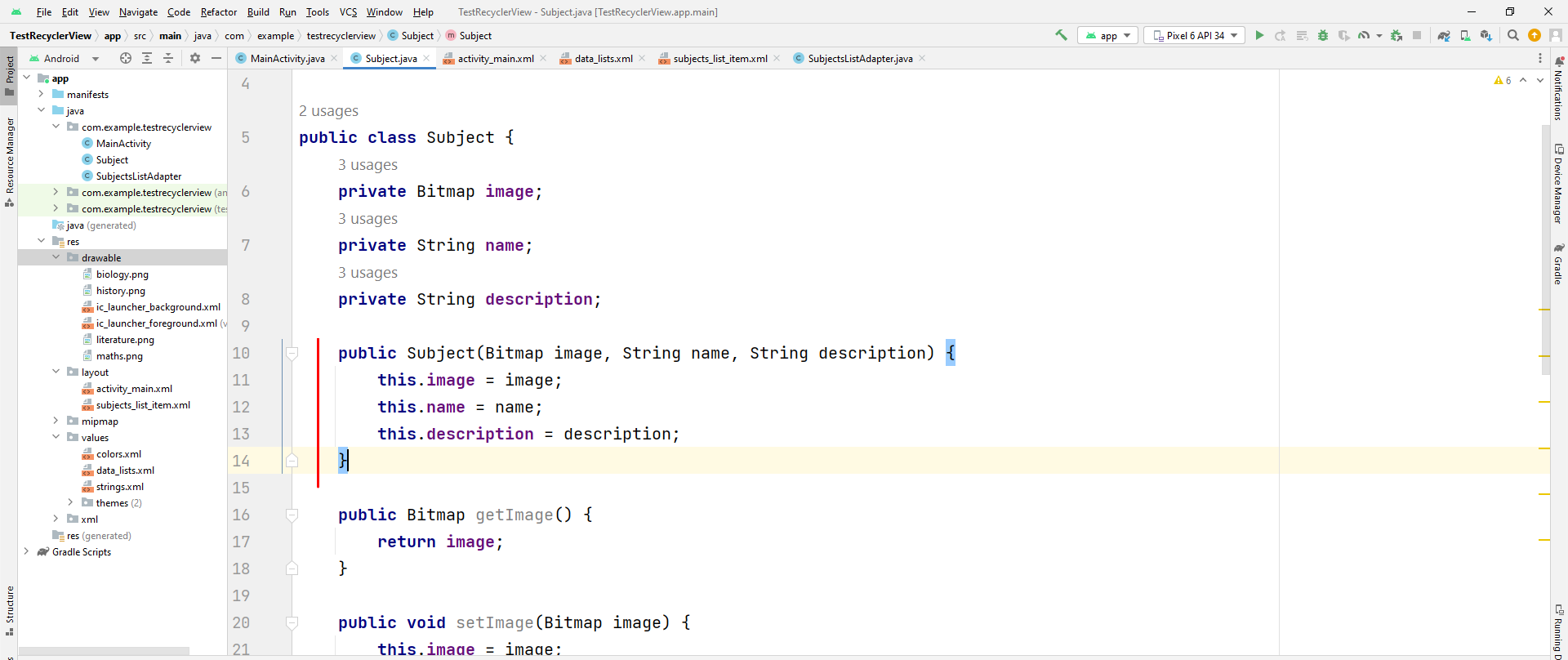
В появившемся меню выбираем соответствующий пункт:



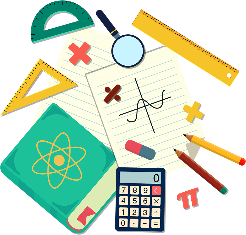
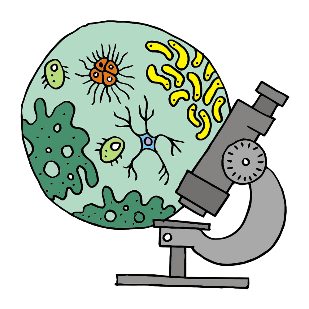
А затем в окне выбираем все поля и нажимаем кнопку Ok:



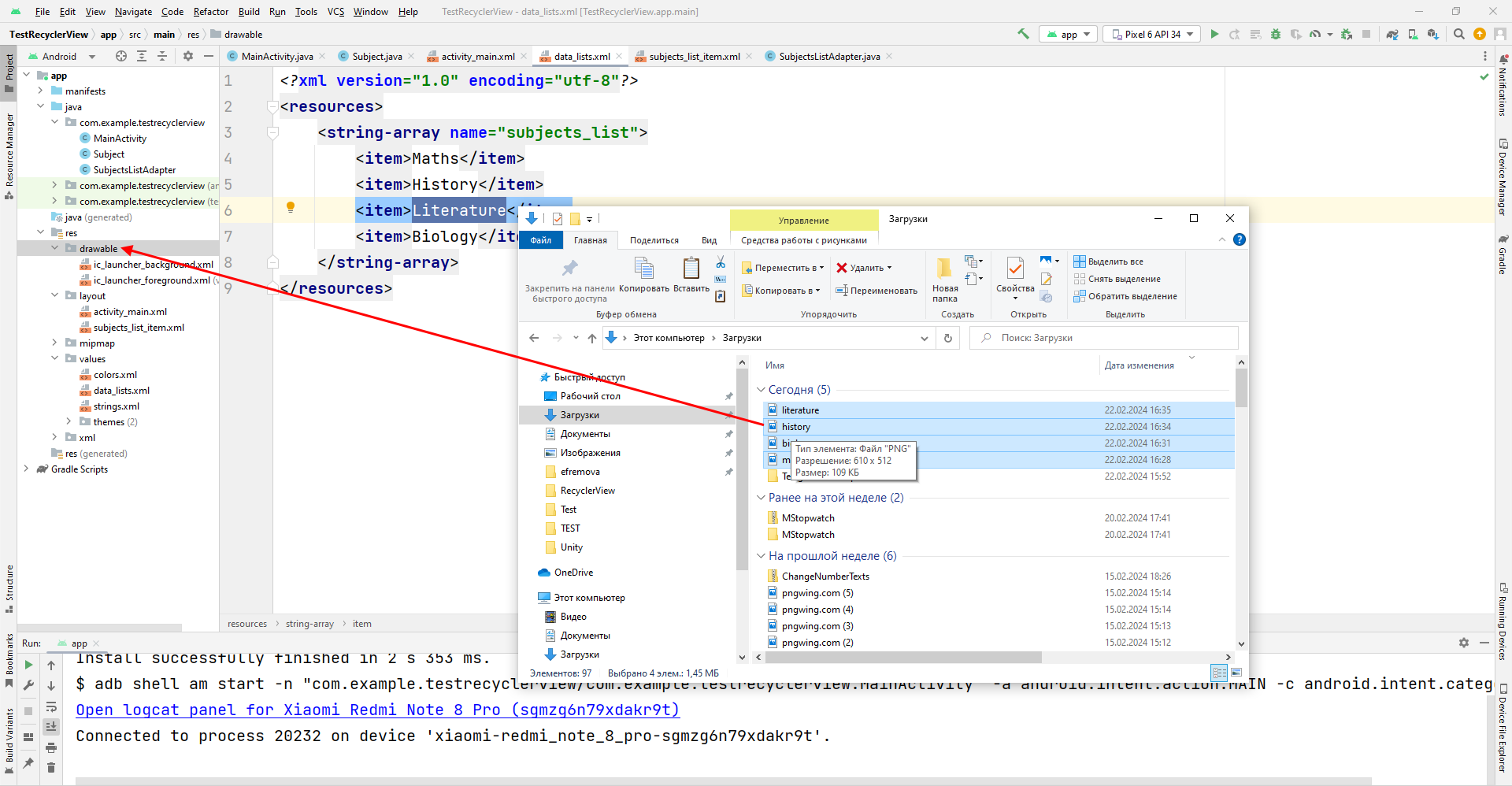
Аналогичным способом можно добавить и конструктор, со всеми полями в качестве входных параметров:



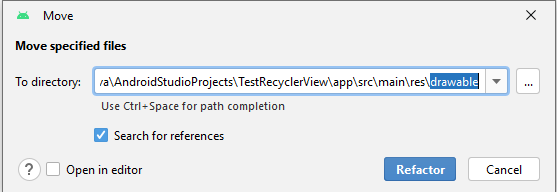
Добавим картинки для предметов, которые можно найти в интернете (ищите без авторских прав) или нарисовать самим. Например, вот такие:



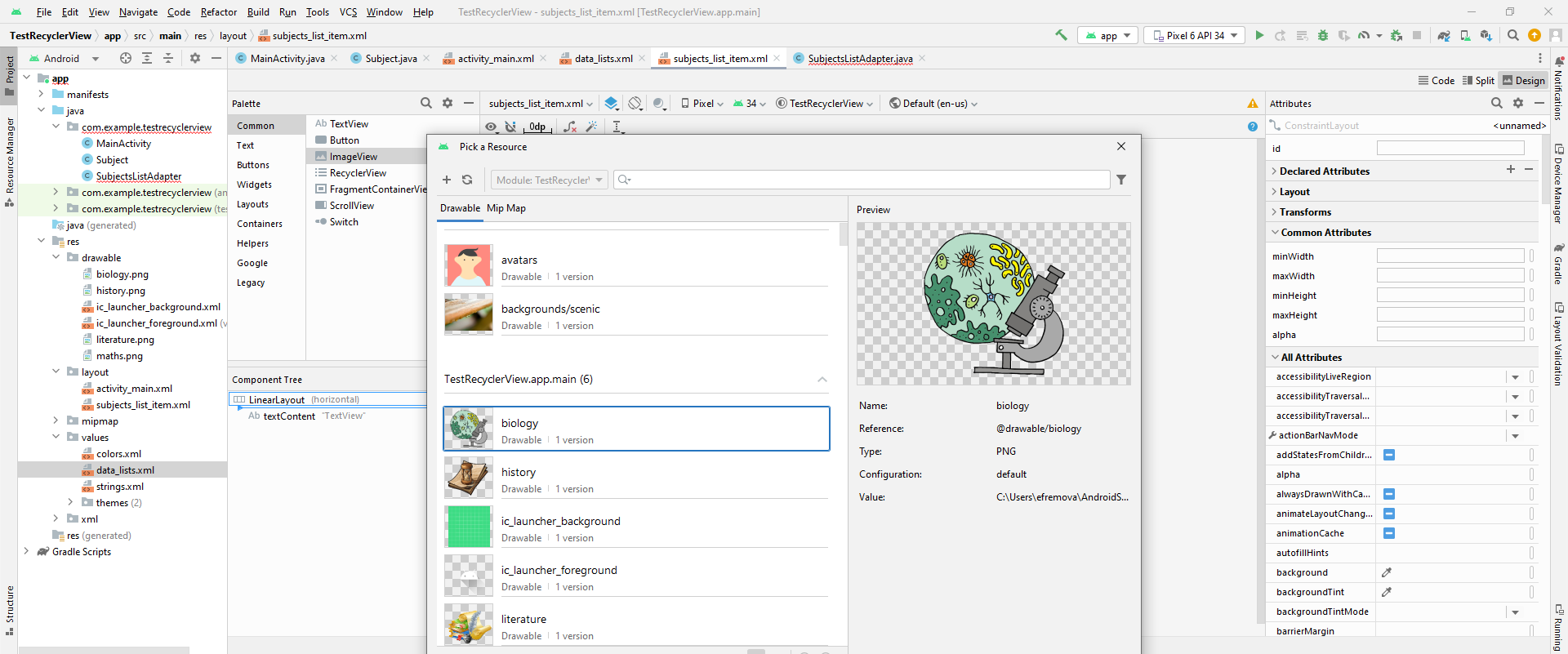
Чтобы поместить их в проект просто сохраните их как отдельные файлы .png и мышкой перетащите в папку drawable:



В появившемся окне нажмите кнопку Refactor:



Теперь давайте сразу исправим разметку для строки в файле subjects\_list\_item.xml. Добавим на первое место компонент ImageView, на втором останется уже созданное текстовое поле, и на треье место добавим еще одно текстовое поле. При добавлении компонента ImageView потребуется указать изображение по умолчанию. Можно выбрать любое, например, картинку биологии.



Код разметки будет выглядеть вот так:

*<?***xml version="1.0" encoding="utf-8"***?>*<**LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:weightSum="100"  
 android:orientation="horizontal"**>  
  
 <**ImageView  
 android:id="@+id/imageView"  
 android:layout\_width="0dp"  
 android:layout\_height="match\_parent"  
 android:layout\_weight="20"  
 app:srcCompat="@drawable/biology"** />  
  
 <**TextView  
 android:id="@+id/textContent"  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content"  
 android:layout\_weight="20"  
 android:text="Name"** />  
  
 <**TextView  
 android:id="@+id/textDescription"  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content"  
 android:layout\_weight="60"  
 android:text="Description"** />  
</**LinearLayout**>

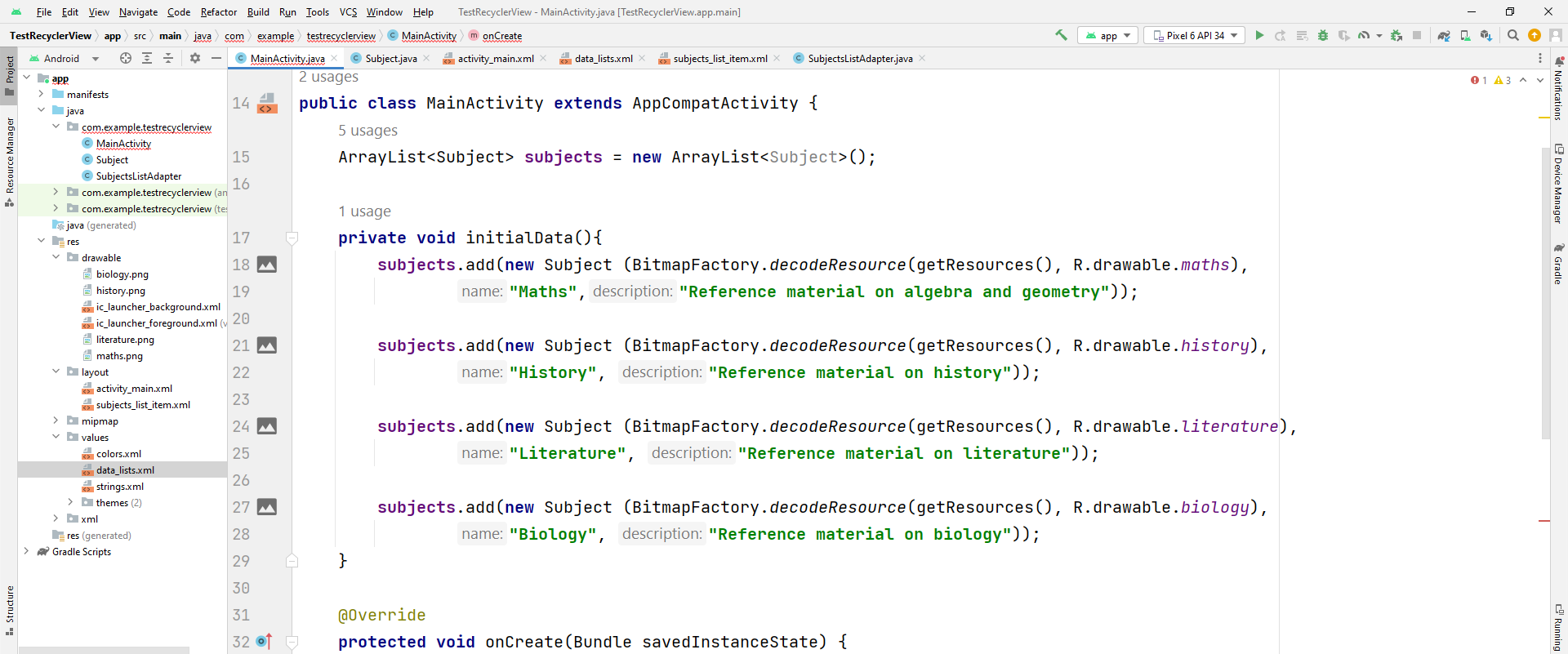
Теперь в классе активности сформируем массив данных. Для этого заведем переменную

ArrayList<Subject> **subjects** = **new** ArrayList<Subject>();

Переменную сразу сделаем глобальной, чтобы потом можно было обращаться к этому списку данных. И список сделаем не массивом, а через класс ArrayList, который позволяет добавлять и удалять элементы. Подробнее про него читаем в разделе «4.1. Массивы. Класс ArrayList». Для постоянного списка предметов это, наверное, не нужно, но в качестве иллюстрации формирования различных списков данных будет полезно.

Создадим отдельный метод (удобно, когда код разделен на методы, отвечающие за определенные действия) для заполнения массива данных:

**private void** initialData(){  
 **subjects**.add(**new** Subject (BitmapFactory.*decodeResource*(getResources(), R.drawable.*maths*),  
 **"Maths"**,**"Reference material on algebra and geometry"**));  
  
 **subjects**.add(**new** Subject (BitmapFactory.*decodeResource*(getResources(), R.drawable.*history*),  
 **"History"**, **"Reference material on history"**));  
   
 **subjects**.add(**new** Subject (BitmapFactory.*decodeResource*(getResources(), R.drawable.*literature*),  
 **"Literature"**, **"Reference material on literature"**));  
   
 **subjects**.add(**new** Subject (BitmapFactory.*decodeResource*(getResources(), R.drawable.*biology*),  
 **"Biology"**, **"Reference material on biology"**));  
}



В методе мы четыре раза создаем новый объект **new** Subject через конструктор - на первом месте подаем картинку из ресурсов, на втором название предмета, на третьем описание:

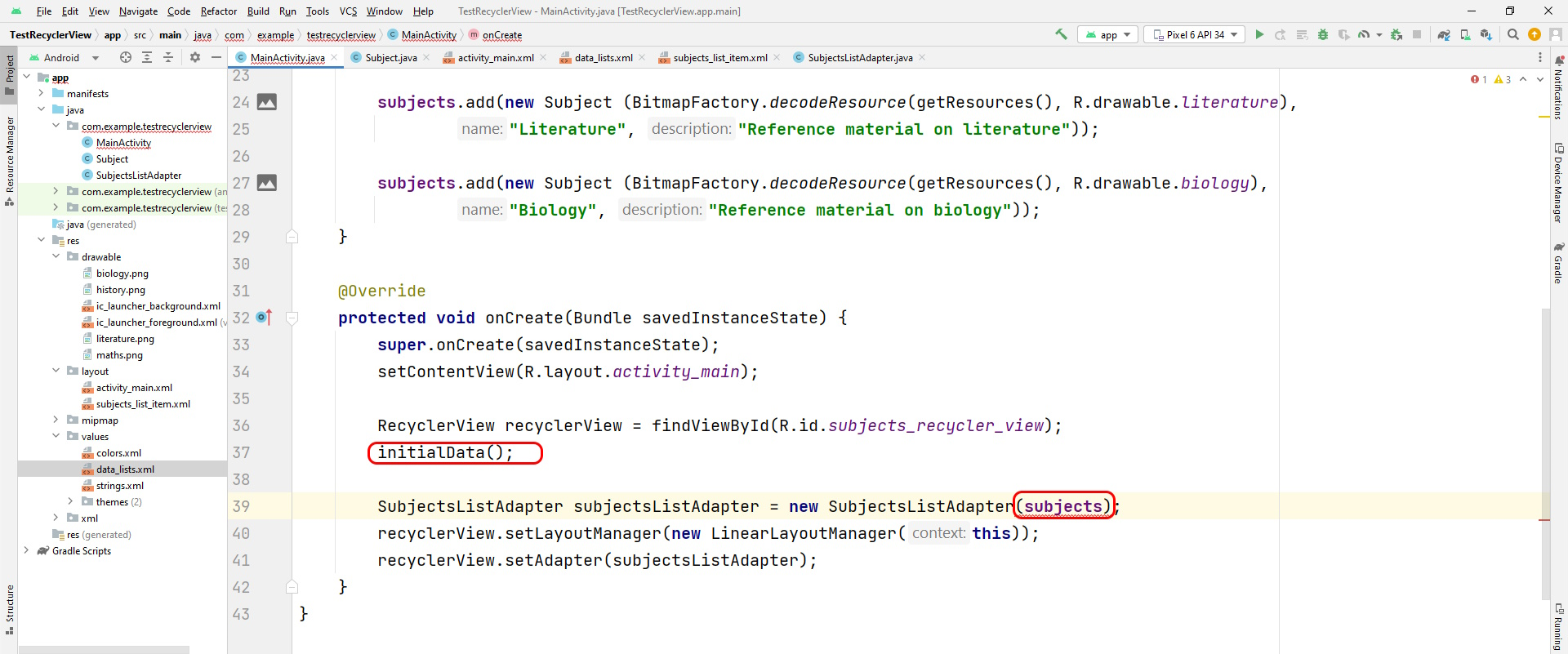
**new** Subject (BitmapFactory.*decodeResource*(getResources(), R.drawable.*maths*),  
 **"Maths"**,**"Reference material on algebra and geometry"**)

И затем сразу добавляем его в наш список с данными:

**subjects**.add(**new** Subject ( … ))

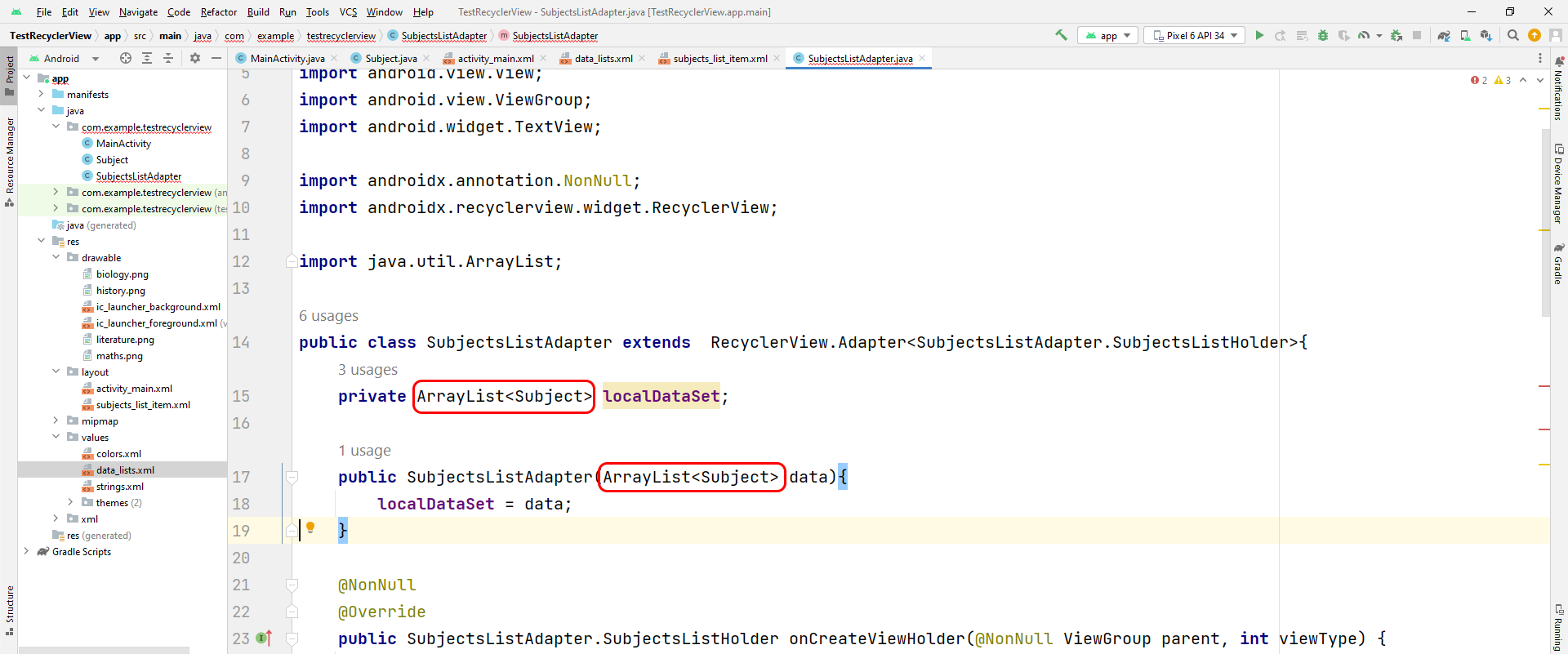
Здесь уже нам не понадобится файл со строковым списком data\_lists.xml.

Изменения в коде создания списка будут совсем небольшими. Вместо получения списка из ресурсов мы вызываем метод initialData() и в конструктор адаптера передаем ArrayList<Subject> **subjects:**



На данном этапе в строке 39 будет ошибка, поскольку мы изменили тип входного списка для адаптера, но она исправится сама, когда мы доработаем класс адаптера.

Теперь перейдем в класс адаптера и исправим в нем ошибки. Для начала необходимо изменить тип списка для хранения данных с простого массива на ArrayList:

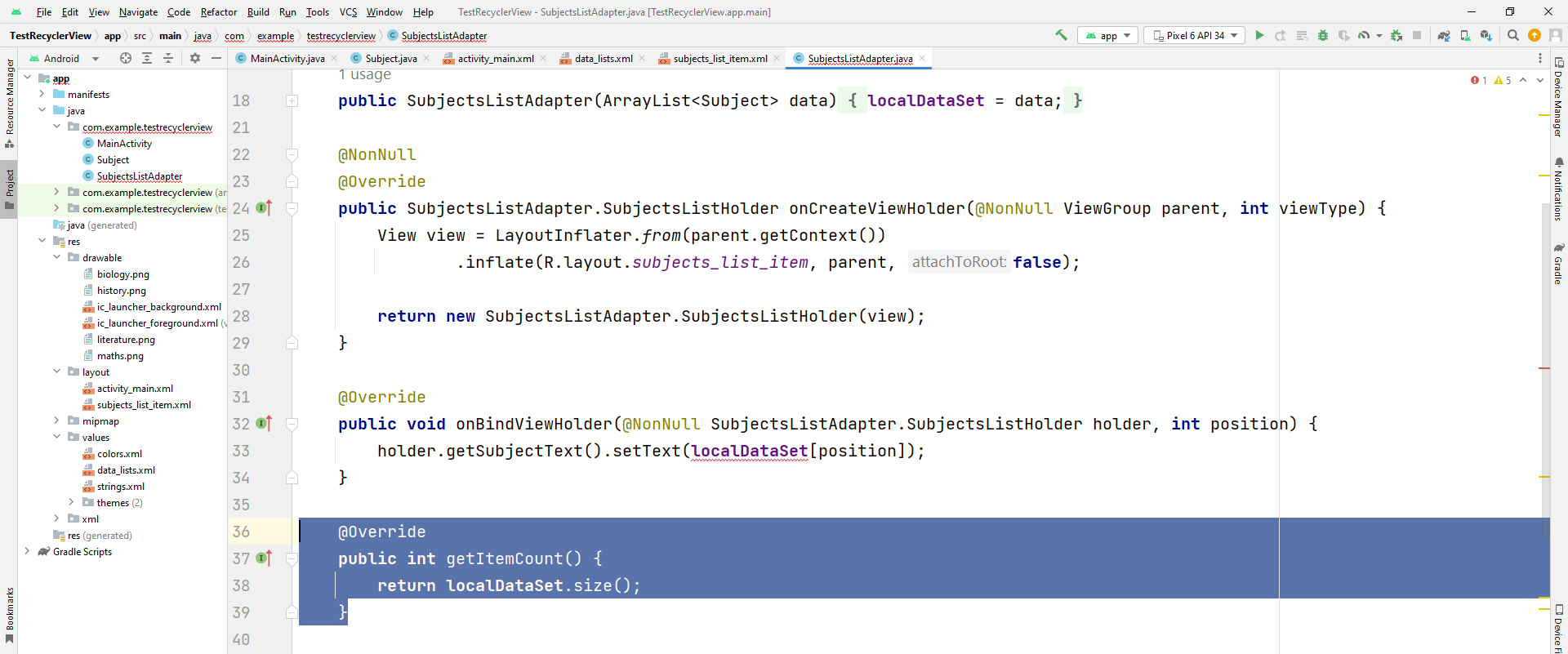


Оставим пока класс адаптера, там еще есть ошибки, но пока мы их не можем исправить, сначала нужно доработать класс SubjectsListHolder следующим образом:

**public class** SubjectsListHolder **extends** RecyclerView.ViewHolder{  
 **private final** TextView **subjectText**;  
 **private final** TextView **subjectDescription**;  
 **private final** ImageView **subjectImage**;  
  
 **public** TextView getSubjectText() {  
 **return subjectText**;  
 }  
 **public** TextView getSubjectDescription() {  
 **return subjectDescription**;  
 }  
 **public** ImageView getSubjectImage() {  
 **return subjectImage**;  
 }  
  
 **public** SubjectsListHolder(@NonNull View itemView) {  
 **super**(itemView);  
 **subjectText** = itemView.findViewById(R.id.*textContent*);  
 **subjectDescription** = itemView.findViewById(R.id.*textDescription*);  
 **subjectImage** = itemView.findViewById(R.id.*imageView*);  
 }  
}

У нас появились новые поля **subjectDescription** и **subjectImage**, соответственно методы для обращения к нимgetSubjectDescription() и getSubjectImage(). И в конструкторе класса мы связываем эти переменные с компонентами разметки строки.

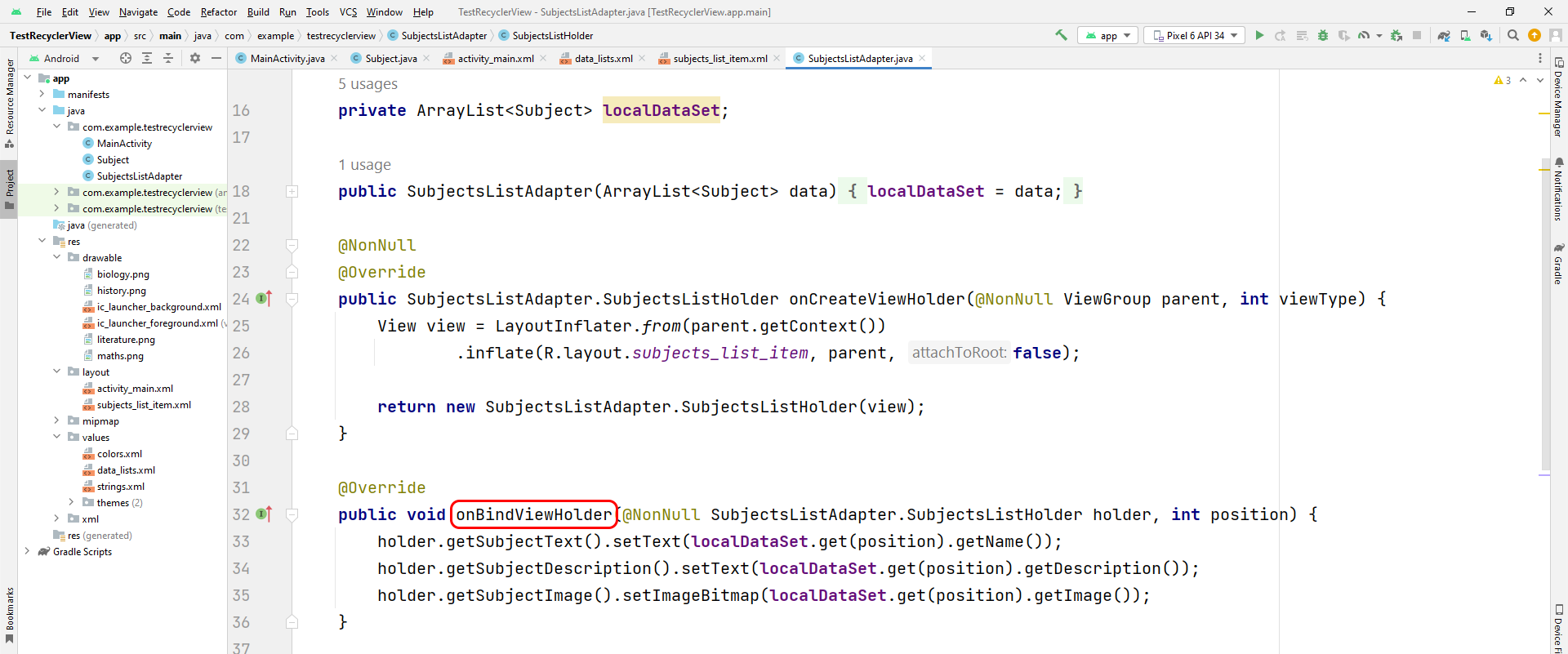
Возвращаемся к классу адаптера и исправляем ошибку получения размера – вместо стандартного массива мы используем теперь ArrayList, и размер у него определяется методом size().



Следующая ошибка в методе onBindViewHolder() при обращении к конкретному элементу опять же из-за использования ArrayList вместо стандартного массива. Обращение к текущему элементу у ArrayList реализовано через метод get().

Но, помимо этого, каждый элемент списка у нас является объектом класса Subject, и в каждой строке списка нам нужно отобразить картинку для предмета, его название и описание. Поэтому onBindViewHolder() будет выглядеть следующим образом:

@Override  
**public void** onBindViewHolder(@NonNull SubjectsListAdapter.SubjectsListHolder holder, **int** position) {  
 holder.getSubjectText().setText(**localDataSet**.get(position).getName());  
 holder.getSubjectDescription().setText(**localDataSet**.get(position).getDescription());  
 holder.getSubjectImage().setImageBitmap(**localDataSet**.get(position).getImage());  
}



Запущенное приложение будет выглядеть следующим образом:



Теперь в файле разметки для строки списка можно настроить более красивый вид для каждой строки, в коде класса SubjectsListAdapter добавить обработчики, реагирующие на одинарное или двойное нажатие по строке списка и т.д.