8. ВЫВОД СПИСКОВ И LISTFRAGMENT

На данный момент уровень модели BookDepository состоит только из одного единственного экземпляра Book. В этом разделе приложение BookDepository будет обновлено так, чтобы оно поддерживало списки. В списке для каждой книги будет отображаться краткое описание и дата, а также признак её прочтения (рисунок 8.1).

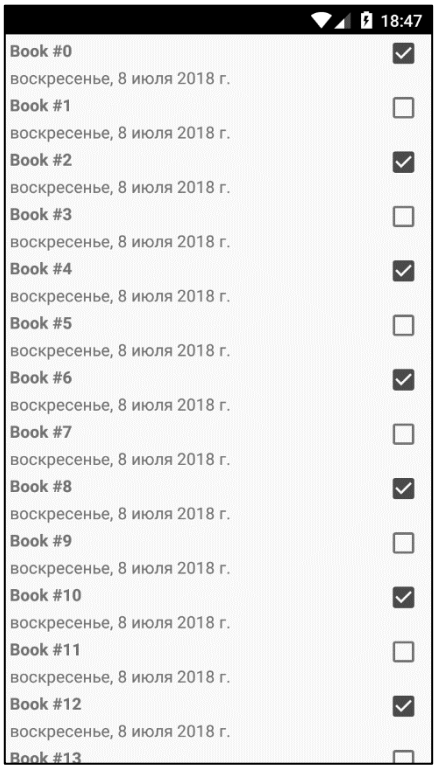

Рисунок 8.1 – Список книг

На рисунке 8.2 показана общая структура приложения BookDepository для этого раздела.

На уровне модели появляется новый объект BookLab, который представляет собой централизованное хранилище для объектов Book.

Для отображения списка на уровне контроллера BookDepository появляется новая активность и новый фрагмент: BookListActivity и BookListFragment.

Обновление уровня модели BookDepository

Прежде всего необходимо преобразовать уровень модели BookDepository из одного объекта Book в список List объектов Book.

Для хранения массива-списка книг будет использоваться *синглетный* (singleton) класс. Такие классы допускают создание только одного экземпляра.

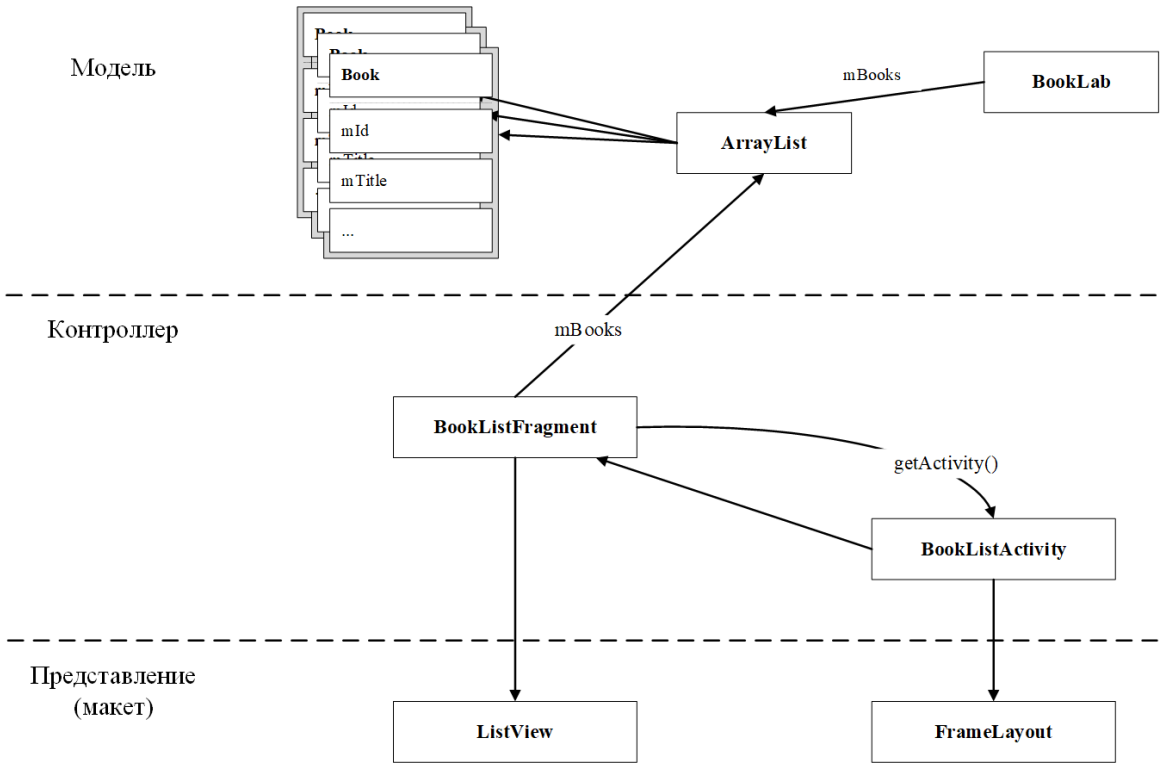

Рисунок 8.2 – Приложение BookDepository со списком

Экземпляр синглетного класса существует до тех пор, пока приложение остается в памяти, так что при хранении списка в синглетном объекте данные остаются доступными, что бы ни происходило с активностями, фрагментами и их жизненными циклами. Синглетные классы будут уничтожаться, когда Android удаляет приложение из памяти. Синглет BookLab не подходит для долгосрочного хранения данных, но позволяет приложению назначить одного владельца данных и предоставляет возможность простой передачи этой информации между классами-контроллерами.

Чтобы создать синглетный класс, следует создать класс с закрытым конструктором и методом `get()`. Если экземпляр уже существует, то `get()` просто возвращает его. Если экземпляр еще не существует, то `get()` вызывает конструктор для его создания.

Щёлкнуть правой кнопкой мыши на пакете ru.rsue.android и выбрать команду New→Java Class. Ввести имя класса BookLab и щёлкнуть на кнопке Finish.

В файле BookLab.java реализовать BookLab как синглетный класс с закрытым конструктором и методом `get()`.

Листинг 8.1 – Синглетный класс (BookLab.java)

```
public class BookLab {
    private static BookLab sBookLab;
    public static BookLab get(Context context) {
        if (sBookLab == null) {
            sBookLab = new BookLab(context);
```

```
        }
        return sBookLab;
    }
    private BookLab(Context context) {
    }
}
```

В реализации BookLab есть несколько моментов, заслуживающих внимания. Во-первых, префикс **s** у переменной `sBookLab`. Это условное обозначение используется Android, чтобы показать, что переменная `sBookLab` является статической.

Следует обратить внимание на закрытый конструктор `BookLab`. Другие классы не смогут создать экземпляр `BookLab` в обход метода `get()`.

Для начала предоставить `BookLab` несколько объектов Book для хранения. В конструкторе `BookLab` создайть пустой список List объектов Book. Добавить два метода: `getBooks()` возвращает List, а `getBook(UUID)` возвращает объект Book с заданным идентификатором (листинг 8.2).

Листинг 8.2 – Создание списка List объектов Book (BookLab.java)

```
public class BookLab {
    private static BookLab sBookLab;
    private List<Book> mBooks;
    public static BookLab get(Context context) {
        ...
    }
    private BookLab(Context context) {
        mBooks = new ArrayList<>();
    }
    public List<Book> getBooks() {
        return mBooks;
    }
    public Book getBook(UUID id) {
        for (Book book : mBooks) {
            if (book.getId().equals(id)) {
                return book;
            }
        }
        return null;
    }
}
```

List<E> — интерфейс поддержки упорядоченного списка объектов заданного типа. Он определяет методы получения, добавления и удаления элементов. Одна из распространенных реализаций List — ArrayList — использует для хранения элементов списка обычный массив Java.

Со временем List будет содержать объекты Book, созданные пользователем, которые будут сохраняться и загружаться повторно. А пока заполнить массив 100 однообразных объектов Book (листинг 8.3).

Листинг 8.3 – Генерирование тестовых объектов (BookLab.java)

```
private BookLab(Context Context) {
      mBooks = new ArrayList<>();
      for (int i = 0; i < 100; i++) {
           Book book = new Book();
           book.setTitle("Book #" + i);
           book.setReaded(i % 2 == 0); // Для каждого второго объекта
           mBooks.add(book);
      }
}
```

Теперь в приложении имеется полностью загруженный уровень модели и 100 книг для вывода на экран.

Абстрактная активность для хостинга фрагмента

Далее будет создадан класс BookListActivity, предназначенный для выполнения функций хоста для BookListFragment. Но сначала будет создано представление для BookListActivity.

Для BookListActivity можно просто воспользоваться макетом, определенным в файле activity_book.xml (листинг 8.4). Этот макет определяет виджет FrameLayout как контейнерное представление для фрагмента, который затем указывается в коде активности.

Листинг 8.4 – Файл activity_book.xml уже содержит универсальную разметку

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

Поскольку в файле activity_book.xml не указан конкретный фрагмент, он может использоваться для любой активности, выполняющей функции хоста для одного фрагмента. Переименуем его в activity_fragment.xml, чтобы отразить этот факт.

В окне инструментов Project щёлкнуть правой кнопкой мыши на файле res/layout/activity_book.xml. (Щелкнуть нужно на activity_book.xml, а не на fragment_book.xml.)

Выбрать в контекстном меню команду Refactor→Rename.... Ввести имя activity_fragment.xml. При переименовании ресурса ссылки на него обновляются автоматически.

Среда Android Studio должна автоматически обновить ссылки на новый файл activity_fragment.xml. Если будет получено сообщение об ошибке в BookActivity.java, то придется вручную обновить ссылку BookActivity, как показано в листинге 8.5.

Листинг 8.5 – Обновление файла макета для BookActivity (BookActivity.java)

```
public class BookActivity extends FragmentActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       ~~setContentView(R.layout.activity_book);~~
       setContentView(R.layout.activity_fragment);
       FragmentManager fm = getSupportFragmentManager();
       Fragment fragment =
                fm.findFragmentById(R.id.fragment_container);
       if (fragment == null) {
            ...
       }
    }
}
```

Абстрактный класс Activity

Для создания класса BookListActivity можно повторно использовать код BookActivity. Код, написанный для BookActivity (листинг 8.5): прост и практически универсален. Собственно в нем есть всего одно неуниверсальное место: создание экземпляра BookFragment перед его добавлением в FragmentManager.

Листинг 8.6 – Класс BookActivity почти универсален (BookActivity.java)

```
public class BookActivity extends FragmentActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_fragment);
       FragmentManager fm = getSupportFragmentManager();
       Fragment fragment =
                fm.findFragmentById(R.id.fragment_container);
       if (fragment == null) {
            fragment = new BookFragment();
            fm.beginTransaction()
            .add(R.id.fragment_container, fragment)
            .commit();
       }
```

```
    }
}
```

Почти в каждой активности, которая будет создаваться в этом приложении, будет присутствовать такой же код. Чтобы избежать повторений, следует выделить его в абстрактный класс.

Создать новый класс с именем `SingleFragmentActivity` в пакете BookDepository. Сделать его субклассом `FragmentActivity` и объявить абстрактным классом.

Листинг 8.7 – Создание абстрактной активности (SingleFragmentActivity.java)

```
public abstract class SingleFragmentActivity extends FragmentActivity {
}
```

Теперь включить следующий фрагмент в SingleFragmentActivity.java. Не считая выделенных частей, он идентичен старому коду BookActivity.

Листинг 8.8 – Добавление обобщенного суперкласса (SingleFragmentActivity.java)

```
public abstract class SingleFragmentActivity extends FragmentActivity {
    protected abstract Fragment createFragment();
    @Override
    public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_fragment);
       FragmentManager fm = getSupportFragmentManager();
       Fragment fragment =
                fm.findFragmentById(R.id.fragment_container);
       if (fragment == null) {
            fragment = createFragment();
            fm.beginTransaction()
            .add(R.id.fragment_container, fragment)
            .commit();
       }
    }
}
```

В этом коде представление активности заполняется по данным activity_fragment.xml. Затем осуществляется поиск фрагмента в FragmentManager этого контейнера, создавая и добавляя его, если он не существует.

Код в листинге 8.8 отличается от кода BookActivity только абстрактным методом createFragment(), который используется для создания экземпляра фрагмента.

Субклассы SingleFragmentActivity реализуют этот метод так, чтобы он возвращал экземпляр фрагмента, хостом которого является активность.

Использование абстрактного класса

Открыть класс с именем BookActivity, назначить его суперклассом SingleFragmentActivity, удалить реализацию `onCreate(Bundle)` и реализовать метод `createFragment()` так, как показано в листинге 8.9.

Листинг 8.9 – Переработка BookActivity (BookActivity.java)

```
public class BookActivity extends FragmentActivity
        SingleFragmentActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_fragment);
        FragmentManager fm = getSupportFragmentManager();
        Fragment fragment =
                fm.findFragmentById(R.id.fragment_container);
        if (fragment == null) {
            fragment = new BookFragment();
            fm.beginTransaction()
            .add(R.id.fragment_container, fragment)
            .commit();
        }
    }
    @Override
    protected Fragment createFragment() {
        return new BookFragment();
    }
}
```

Создание контроллеров для вывода списков

Далее будут создаданы два новых класса-контроллера: BookListActivity и BookListFragment.

Щёлкнуть правой кнопкой мыши на пакете ru.rsue.android, выбрать команду New→Java Class и присвоить классу имя BookListActivity.

Изменить новый класс BookListActivity так, чтобы он тоже субклассировал SingleFragmentActivity и реализовал метод createFragment().

Листинг 8.10 – Реализация BookListActivity (BookListActivity.java)

```
public class BookListActivity extends SingleFragmentActivity {
    @Override
    protected Fragment createFragment() {
        return new BookListFragment();
    }
}
```

Если класс BookListActivity содержит другие методы, такие, как: onCreate — удалить их. Пусть класс SingleFragmentActivity выполняет свою работу, а реализация BookListActivity будет по возможности простой.

Для создания класса BookListFragment снова щёлкнуть правой кнопкой мыши на пакете ru.rsue.android, выбрать команду New→Java Class и присвоить классу имя BookListFragment.

Листинг 8.11 – Реализация BookListFragment (BookListActivity.java)

```
public class BookListFragment extends Fragment {
   // Пока пусто
}
```

Пока BookListFragment остается пустой оболочкой фрагмента. Он будет использован позднее.

Объявление BookListActivity

Теперь, когда класс BookListActivity создан, его следует объявить в манифесте. Кроме того, список книг должен выводиться на первом экране, который виден пользователю после запуска BookDepository; следовательно, активность BookListActivity должна быть активностью лаунчера.

Включить в манифест объявление BookListActivity и переместить фильтр интентов из объявления BookActivity в объявление BookListActivity, как показано в листинге 8.12.

Листинг 8.12 – Объявление BookListActivity активностью лаунчера (AndroidManifest.xml)

```
...
   <application
   ...
   android:label="@string/app_name"
   android:theme="@style/AppTheme" >
      <activity android:name=".BookListActivity">
         <intent-filter>
         <action android:name="android.intent.action.MAIN" />
         <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
      <activity android:name=".BookActivity"
         android:label="@string/app_name">
         <intent-filter>
         <action android:name="android.intent.action.MAIN" />
         <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
   </application>
</manifest>
```

BookListActivity теперь является активностью лаунчера. Запустить BookDepository; на экране появляется виджет FrameLayout из BookListActivity, содержащий пустой фрагмент BookListFragment (рисунок 8.3).

RecyclerView

Итак, в BookListFragment должен отображаться список. Для этого следует воспользоваться классом RecyclerView.

Класс RecyclerView является субклассом ViewGroup. Он выводит список дочерних объектов View, по одному для каждого элемента. В зависимости от сложности отображаемых данных дочерние объекты View могут быть сложными или очень простыми.

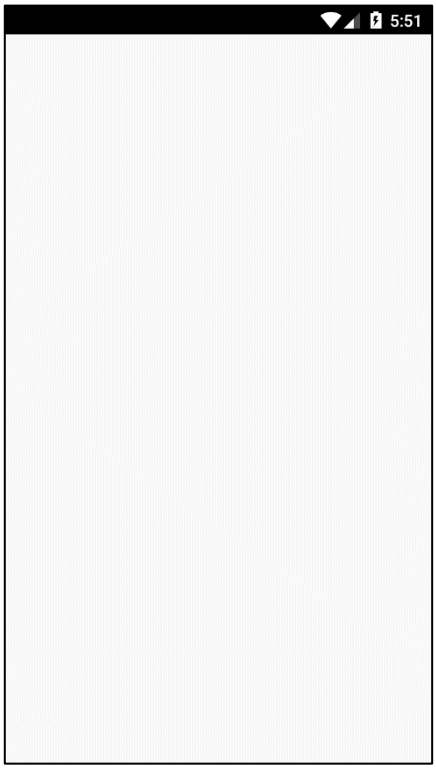

Рисунок 8.3 – Пустой экран BookListActivity

Первая реализация передачи данных для отображения будет очень простой: в элементе списка будет отображаться только краткое описание объекта Book, а объект View представляет собой простой виджет TextView (рисунок 8.4).

На рисунке 8.4 изображены 12 виджетов TextView. Позднее пользователь сможет провести по экрану BookDepository, чтобы прокрутить 100 виджетов TextView для просмотра всех объектов Book.

Единственная обязанность RecyclerView — повторное использование виджетов TextView и их позиционирование на экране. Чтобы обеспечить их

исходное размещение, он работает с двумя классами: субклассом Adapter и субклассом ViewHolder.

Класс RecyclerView находится в одной из многочисленных библиотек поддержки Google. Первым шагом в использовании RecyclerView станет добавление библиотеки RecyclerView к зависимостям приложения.

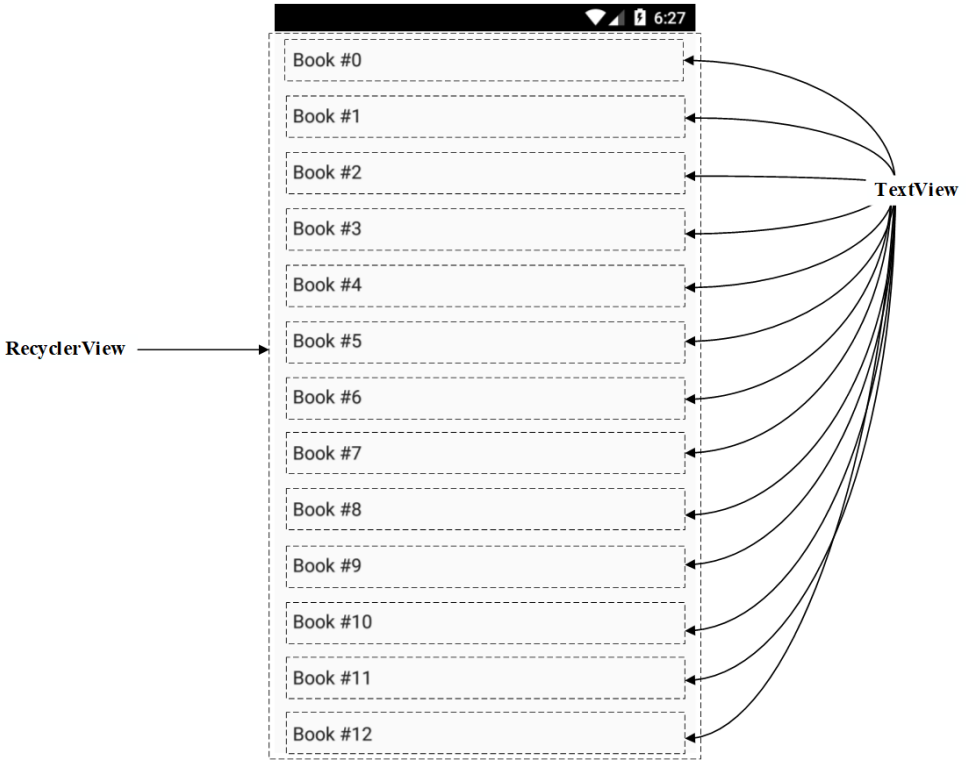

Рисунок 8.4 – RecyclerView с дочерними виджетами TextView

Открыть окно структуры проекта командой File→Project Structure.... Выбрать модуль app слева, перейти на вкладку Dependencies. Щёлкнуть на кнопке + и выбрать Library dependency, чтобы добавить зависимость.

Найти и выделить библиотеку recyclerview-v7; щёлкнуть на кнопке OK, чтобы добавить библиотеку как зависимость (рисунок 8.5).

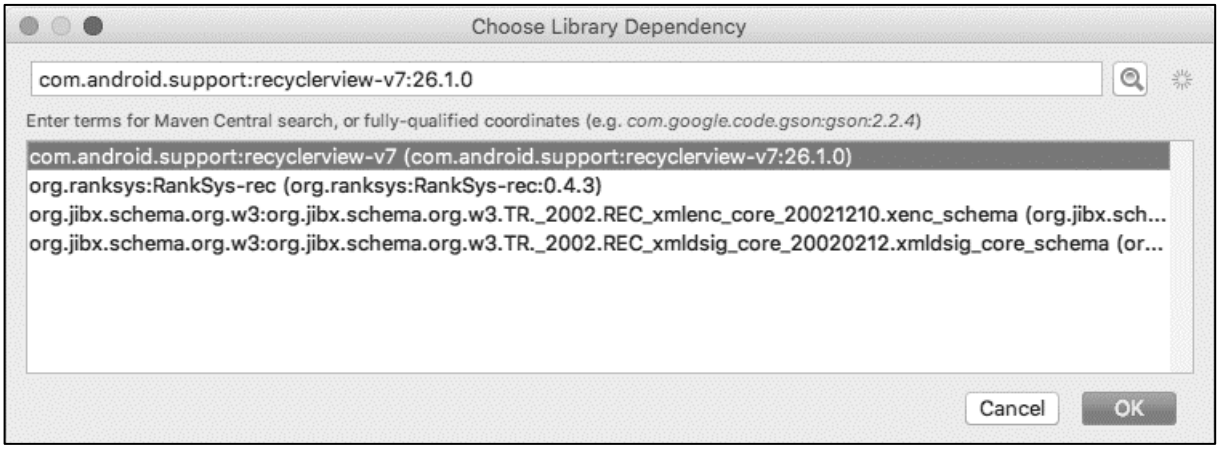

Рисунок 8.5 – Добавление зависимости для RecyclerView

Виджет RecyclerView будет находиться в файле макета BookListFragment. Сначала необходимо создать файл макета: щёлкнуть правой кнопкой мыши на каталоге res/layout и выбрать команду New→Layout resource file. Ввести имя fragment_book_list и щёлкнуть на кнопке OK, чтобы создать файл.

Открыть только что созданный файл fragment_book_list, заменить его корневое представление на RecyclerView и присвойть ему идентификатор.

Листинг 8.13 – Включение RecyclerView в файл макета (fragment_book_list.xml)

```
<android.support.v7.widget.RecyclerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/book_recycler_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

Представление BookListFragment готово; теперь его нужно связать с фрагментом. Изменить класс BookListFragment так, чтобы он использовал этот файл макета и находил RecyclerView в файле макета, как показано в листинге 8.14.

Листинг 8.14 – Подготовка представления для BookListFragment (BookListFragment.java)

```
public class BookListFragment extends Fragment {
 private RecyclerView mBookRecyclerView;
 @Override
 public View onCreateView(LayoutInflater inflater, ViewGroup container,
             Bundle savedInstanceState) {
   View view = inflater.inflate(R.layout.fragment_book_list, container,
       false);
   mBookRecyclerView = (RecyclerView) view
       .findViewById(R.id.book_recycler_view);
   mBookRecyclerView.setLayoutManager(new LinearLayoutManager
       (getActivity()));
   return view;
 }
}
```

Сразу же после создания виджета RecyclerView ему назначается другой объект LayoutManager. Это необходимо для работы виджета RecyclerView. Если ему не будет предоставлен объект LayoutManager, то возникнет ошибка.

Запустить приложение. Снова отобразится пустой экран, но сейчас перед пользователем пустой виджет RecyclerView. Объекты Book остаются невидимыми до тех пор, пока не будут определены реализации Adapter и ViewHolder.

Реализация адаптера и ViewHolder

На первом этапе следует реализовать определение ViewHolder как внутреннего класса BookListFragment.

Листинг 8.15 – Простая реализация ViewHolder (BookListFragment.java)

```
public class BookListFragment extends Fragment {
    ...
    private class BookHolder extends RecyclerView.ViewHolder {
        public TextView mTitleTextView;
        public BookHolder(View itemView) {
            super(itemView);
            mTitleTextView = (TextView) itemView;
        }
    }
}
```

В своем текущем виде ViewHolder хранит ссылку на одно представление: виджет TextView для заголовка. Ожидается, что itemView относится к типу TextView, в противном случае при выполнении кода произойдет ошибка.

После определения ViewHolder создать адаптер.

Листинг 8.16 – Начало работы над адаптером (BookListFragment.java)

```
public class BookListFragment extends Fragment {
    ...
    private class BookAdapter extends RecyclerView.Adapter<BookHolder> {
        private List<Book> mBooks;
        public BookAdapter(List<Book> books) {
            mBooks = books;
        }
    }
}
```

(Код в листинге 8.16 не компилируется. Вскоре этот недостаток будет исправлен.)

Класс RecyclerView взаимодействует с адаптером, когда потребуется создать объект ViewHolder или связать его с объектом Book. Сам виджет RecyclerView ничего не знает об объекте Book, но адаптер располагает полной информацией о Book.

Далее следует реализовать три метода BookAdapter.

Листинг 8.17 – Реализация методов BookAdapter (BookListFragment.java)

```
private class BookAdapter extends RecyclerView.Adapter<BookHolder> {
    ...
    @Override
    public BookHolder onCreateViewHolder(ViewGroup parent, int viewType)
```

```
    {
        LayoutInflater layoutInflater =
                  LayoutInflater.from(getActivity());
        View view = layoutInflater
            .inflate(android.R.layout.simple_list_item_1, parent, false);
        return new BookHolder(view);
    }
    @Override
    public void onBindViewHolder(BookHolder holder, int position) {
        Book book = mBooks.get(position);
        holder.mTitleTextView.setText(book.getTitle());
    }
    @Override
    public int getItemCount() {
        return mBooks.size();
    }
}
```

Итак, адаптер готов; остается связать его с RecyclerView. Следует реализовать метод updateUI, который настраивает пользовательский интерфейс BookListFragment. Пока он создает объект BookAdapter и назначает его RecyclerView.

Листинг 8.18 – Подготовка адаптера (BookListFragment.java)

```
public class BookListFragment extends Fragment {
    private RecyclerView mBookRecyclerView;
    private BookAdapter mAdapter;
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
                container, Bundle savedInstanceState) {
        ...
        mBookRecyclerView.setLayoutManager(new LinearLayoutManager
            (getActivity()));
        updateUI();
        return view;
    }
    private void updateUI() {
        BookLab bookLab = BookLab.get(getActivity());
        List<Book> books = bookLab.getBooks();
        mAdapter = new BookAdapter(books);
        mBookRecyclerView.setAdapter(mAdapter);
    }
    ...
}
```

Запустить приложение BookDepository и прокрутить новый список RecyclerView, который должен выглядеть примерно так, как показано на рисунке 8.6.

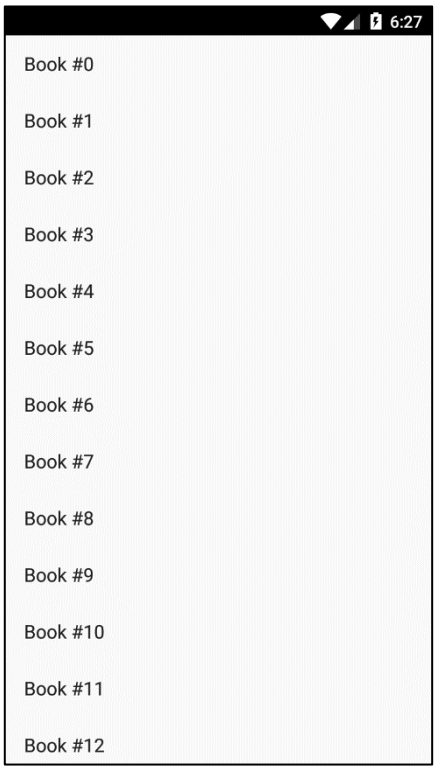

Рисунок 8.6 – Список объектов Book

Создание макета элемента списка

В приложении BookDepository макет элемента списка должен включать краткое описание книги, дату и признак прочтения (рисунок 8.7). Такой макет состоит из двух виджетов — TextView и CheckBox.

Рисунок 8.7 – Список с пользовательским макетом элементов

Новый макет элемента списка создается точно так же, как для представления активности или фрагмента. В окне инструментов Project щёлкнуть правой кнопкой мыши на каталоге res/layout и выбрать команду New→Layout resource file. В открывшемся диалоговом окне ввести имя файла list_item_book, выбрать корневой элемент RelativeLayout и щёлкнуть на кнопке OK.

Листинг 8.19 – Макет пользовательского элемента списка (list_item_book.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android=
                "http://schemas.android.com/apk/res/android"
                android:layout_width="match_parent"
                android:layout_height="wrap_content">
    <CheckBox
            android:id="@+id/list_item_book_readed_check_box"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentRight="true"
            android:padding="4dp"/>

    <TextView
            android:id="@+id/list_item_book_title_text_view"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_toLeftOf=
           "@id/list_item_book_readed_check_box"
            android:textStyle="bold"
            android:padding="4dp"
            tools:text="Заголовок книги"/>

    <TextView
            android:id="@+id/list_item_book_date_text_view"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_toLeftOf=
           "@id/list_item_book_readed_check_box"
            android:layout_below="@id/list_item_book_title_text_view"
            tools:text="Дата прочтения книги"/>
</RelativeLayout>
```

Использование нового представления элемента списка

Теперь внести изменения в класс BookAdapter, чтобы использовать новый файл макета list_item_book.

Листинг 8.20 – Заполнение пользовательского макета (BookListFragment.java)

```
private class BookAdapter extends RecyclerView.Adapter<BookHolder> {
   ...
   @Override
   public BookHolder onCreateViewHolder(ViewGroup parent, int viewType)
   {
      LayoutInflater layoutInflater =
                LayoutInflater.from(getActivity());
```

```
        View view = layoutInflater
            .inflate(android.R.layout.simple_list_item_1
                R.layout.list_item_book, parent, false);
        return new BookHolder(view);
    }
    ...
}
```

Наконец, пора наделить BookHolder новыми обязанностями. Внести изменения в класс BookHolder, чтобы он получал виджет TextView с кратким описанием, TextView с датой и CheckBox с признаком прочтения книги.

Листинг 8.21 – Поиск представлений в BookHolder (BookListFragment.java)

```
private class BookHolder extends RecyclerView.ViewHolder {
    public TextView mTitleTextView;
    private TextView mTitleTextView;
    private TextView mDateTextView;
    private CheckBox mReadedCheckBox;
    public BookHolder(View itemView) {
        super(itemView);
        mTitleTextView = (TextView) itemView;
        mTitleTextView = (TextView)
            itemView.findViewById(R.id.list_item_book_title_text_view);
        mDateTextView = (TextView)
            itemView.findViewById(R.id.list_item_book_date_text_view);
        mReadedCheckBox = (CheckBox)
            itemView.findViewById(R.id.list_item_book_readed_check_box)
    }
}
```

Добавить в BookHolder метод bindBook(Book), чтобы немного упростить код.

Листинг 8.22 – Связывание представлений в BookHolder (BookListFragment.java)

```
private class BookHolder extends RecyclerView.ViewHolder {
    private Book mBook;
    ...
    public void bindBook(Book book) {
        mBook = book;
        mTitleTextView.setText(mBook.getTitle());
        mDateTextView.setText(mBook.getDate().toString());
        mReadedCheckBox.setChecked(mBook.isReaded());
    }
}
```

Получив объект Book, объект BookHolder обновляет TextView с кратким описанием, TextView с датой и CheckBox с признаком прочтения книги в соответствии с содержимым Book.

Теперь у BookHolder есть все необходимое для выполнения его работы. BookAdapter достаточно использовать новый метод bindBook.

Листинг 8.23 – Связывание адаптера с BookHolder (BookListFragment.java)

```
private class BookAdapter extends RecyclerView.Adapter<BookHolder> {
    ...
    @Override
    public void onBindViewHolder(BookHolder holder, int position) {
        Book book = mBooks.get(position);
        holder.mTitleTextView.setText(book.getTitle());
        holder.bindBook(book);
    }
    ...
}
```

Запустить BookDepository и понаблюдать за новым макетом list_item_book в действии (рисунок 8.9).

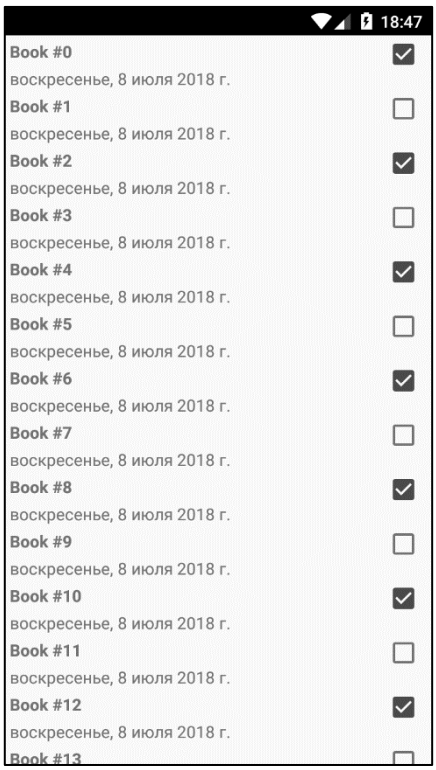

Рисунок 8.9 – А теперь с новыми макетами элементов!

Далее будет реализовано отображение простого уведомления Toast при касании объекта Book в списке. Так как каждое представление View связывается с некоторым ViewHolder, то необходимо сделать объект ViewHolder реализацией OnClickListener для своего View.

Внести изменения в класс BookHolder для обработки касаний в строках.

Листинг 8.24 – Обработка касаний в BookHolder (BookListFragment.java)

```
private class BookHolder extends RecyclerView.ViewHolder
        implements View.OnClickListener {
    ...
    public BookHolder(View itemView) {
        super(itemView);
        itemView.setOnClickListener(this);
        ...
    }
    ...
    @Override
    public void onClick(View v) {
        Toast.makeText(getActivity(),
            mBook.getTitle() + " clicked!", Toast.LENGTH_SHORT)
                .show();
    }
}
```

В листинге 8.24 объект BookHolder реализует интерфейс OnClickListener. Для itemView — представления View всей строки — BookHolder назначается получателем событий щелчка.

Запустить приложение BookDepository и коснуться строки в списке. На экране появляется уведомление Toast с сообщением о касании.