# UI-ФРАГМЕНТЫ

В данном разделе начинается разработка приложения BookDepository. Оно предназначено для учета прочтенных книг. В приложении BookDepository пользователь создает запись о книге, которую планирует прочитать, с заголовком, датой и фотографией обложки. Также можно отправить отзыв о книге по электронной почте, опубликовать его в Twitter, Facebook или другом приложении. Сообщая о прочтенных книгах, пользователь мотивирует других больше читать.

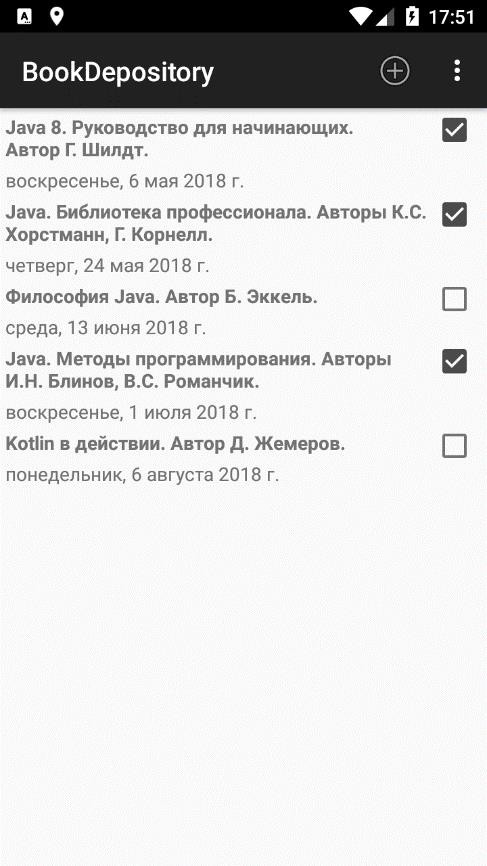
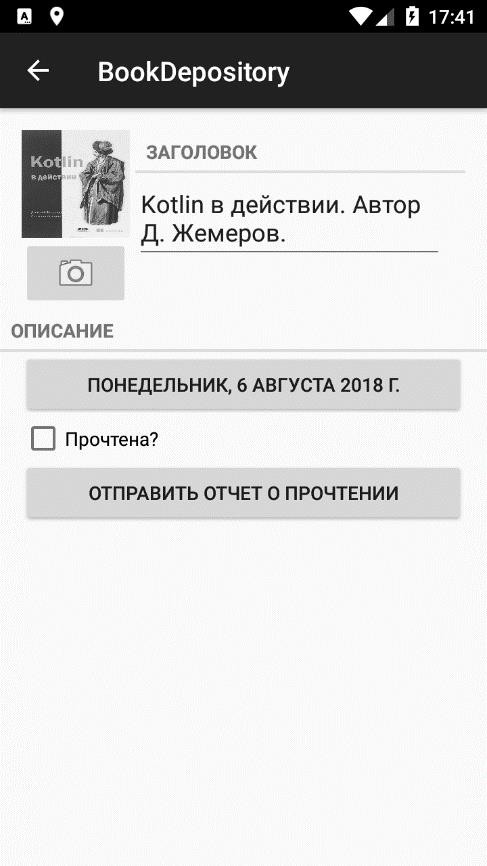


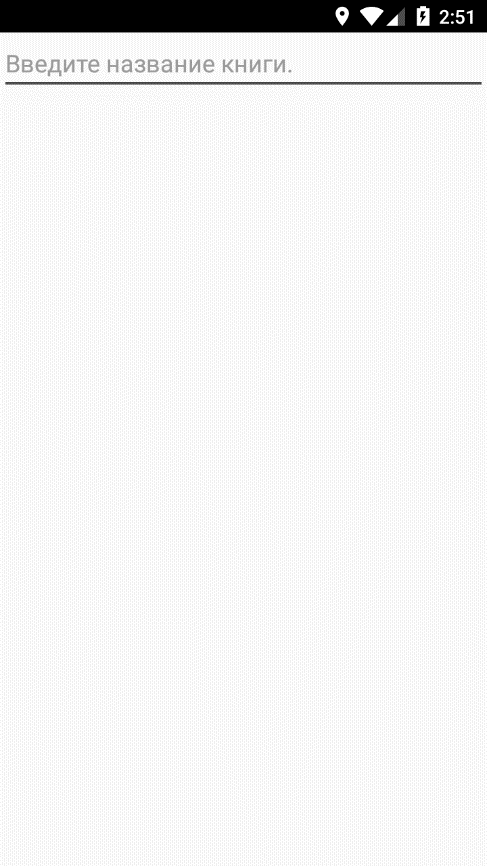
Рисунок 6.1 – Приложение BookDepository

BookDepository — сложное приложение, в нем используется интерфейс типа «список/детализация»: на главном экране выводится список желаемых к прочтению книг. Пользователь может добавить новую или выбрать существующую книгу для просмотра и редактирования информации.

*Начало работы над BookDepository*

На первом этапе будет реализовано представление детализации BookDepository. На рисунке 6.2 показано, как будет выглядеть приложение BookDepository в конце этого раздела.

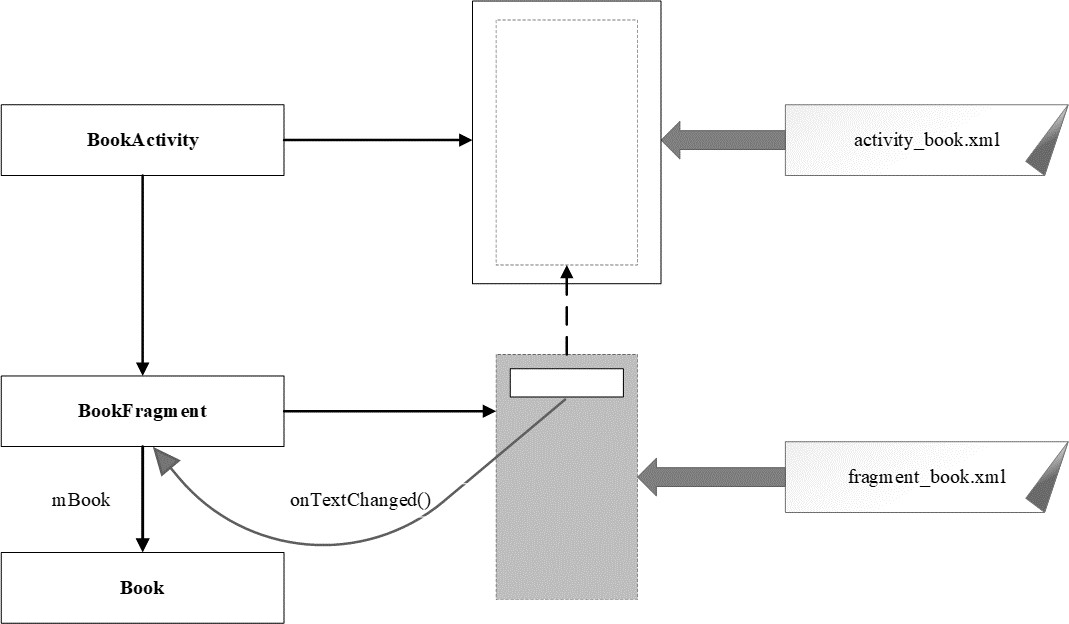
Рисунок 6.2 – Приложение BookDepository в конце раздела 6



Экраном, показанным на рисунке 6.2, будет управлять UI-фрагмент с именем BookFragment. *Хостом* (host) экземпляра BookFragment является активность с именем BookActivity.

Хост предоставляет позицию в иерархии представлений, в которой фрагмент может разместить свое представление (рисунок 6.3). Фрагмент не может вывести представление на экран сам по себе. Его представление отображается только при размещении в иерархии активности.

Рисунок 6.3 – BookActivity как хост BookFragment



На рисунке 6.4 изображена общая структура BookDepository. Класс BookFragment создает пользовательский интерфейс и управляет им, а также обеспечивает взаимодействие с объектами модели. В проект будут добавлены три класса, изображенных на рисунке 6.4: Book, BookFragment и BookActivity.



Рисунок 6.4 – Диаграмма объектов BookDepository

Экземпляр Book представляет одну книгу. В этом разделе описание книги будет состоять только из заголовка и идентификатора. Заголовок содержит текст (например, «Война и мир. Том 1.» или «Горе от ума.»), а идентификатор однозначно идентифицирует экземпляр Book.

На данном этапе для простоты будет использоваться один экземпляр Book. В класс BookFragment включается поле (mBook) для хранения этой отдельной книги.

Представление BookActivity состоит из элемента FrameLayout, определяющего место, в котором будет отображаться представление BookFragment.

Представление BookFragment будет состоять из элементов LinearLayout и EditText. BookFragment определяет поле для виджета EditText (mTitleField) и назначает для него слушателя, обновляющего уровень модели при изменении текста.

*Создание нового проекта*

Создать новое приложение Android (FileNew Project…). Ввести имя приложения **BookDepository** и имя пакета ***ru.rsue.android*** (*android* – фамилия студента на латинице).

Щёлкнуть на кнопке Next и задать минимальный уровень SDK: API 19: Android 4.4 – Также проследить за тем, чтобы для типов приложений были установлены только флажки Phone и Tablet.

Снова щёлкнуть на кнопке Next, чтобы выбрать разновидность добавляемой активности. Выбрать пустую активность (**Blank** или **Empty Activity**) и продолжать работу с мастером.

На последнем шаге мастера ввести имя активности **BookActivity** и щёлкнуть на кнопке Finish.

*Библиотека поддержки*

Фрагменты появились в API уровня 11 вместе с первыми планшетами на базе Android и потребностью в гибкости пользовательского интерфейса. *Библиотека поддержки* (support library) включает полную реализацию фраг- ментов вплоть до API уровня 4. В нее включены два ключевых класса: Fragment (android.support.v4.app.Fragment) и FragmentActivity (android.support.v4.app.FragmentActivity).

Чтобы использовать библиотеку поддержки, необходимо включить ее в состав зависимостей проекта. Для этого надо перейти к структуре проекта (FileProject Structure...). Выбрать в левой части модуль app, а в нем перейти на вкладку **Dependencies**. На этой вкладке перечислены зависимости модуля app. Нажать кнопку «+» и выбрать зависимость Library, чтобы создать новую зависимость. Выбрать в списке библиотеку **support-v4** и щёлкнуть на кнопке OK.

Открыть файл **build.gradle** из модуля app. Проект содержит два файла build.gradle: один для проекта в целом, другой для модуля app. Добавленная библиотека появляется в списке.

*Cоздание уровня модели BookDepository*

В окне инструментов Project найти и открыть файл BookActivity.java.

Выбрать FragmentActivity суперклассом BookActivity (листинг 6.1). Листинг 6.1 – Настройка шаблонного кода (BookActivity.java)

public class BookActivity extends **~~AppCompatActivity~~ FragmentActivity {**

@Override

protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); setContentView(R.layout.activity\_book);

}

}

В окне инструментов Project щёлкнуть правой кнопкой мыши на пакете ru.rsue.android и выбрать команду NewJava Class. Ввести имя класса **Book** и щёлкнуть на кнопке OK.

Добавьте в Book.java следующий код.

Листинг 6.2 – Добавление кода в класс Book (Book.java)

public class Book { **private UUID mId; private String mTitle; public Book() {**

**mId = UUID.randomUUID(); //Генерирование уникального идентификатора**

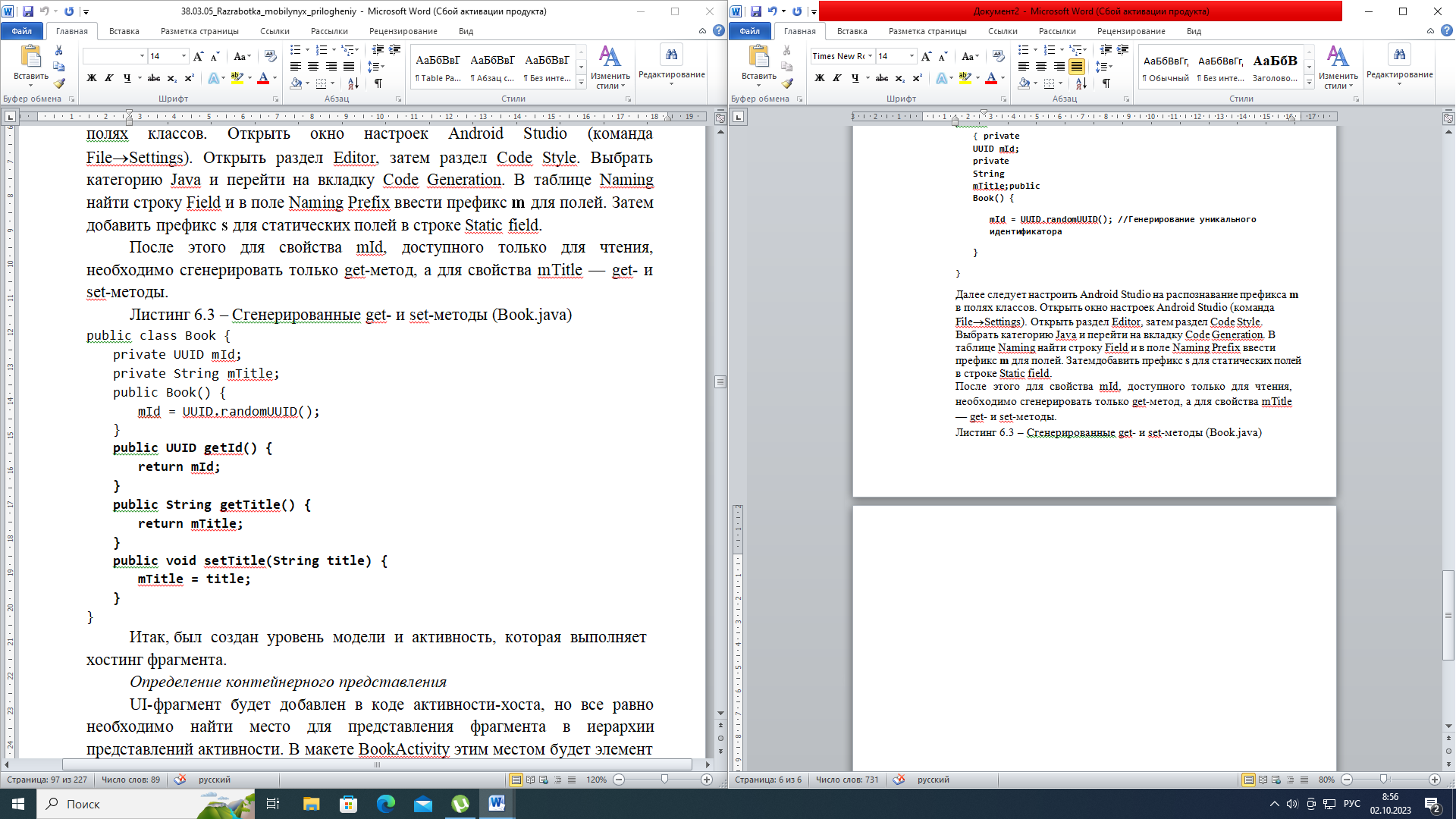
**}**

}

Далее следует настроить Android Studio на распознавание префикса **m** в полях классов. Открыть окно настроек Android Studio (команда FileSettings). Открыть раздел Editor, затем раздел Code Style. Выбрать категорию Java и перейти на вкладку Code Generation. В таблице Naming найти строку Field и в поле Naming Prefix ввести префикс **m** для полей. Затем добавить префикс **s** для статических полей в строке Static field.

После этого для свойства mId, доступного только для чтения, необходимо сгенерировать только get-метод, а для свойства mTitle — get- и set-методы.

Листинг 6.3 – Сгенерированные get- и set-методы (Book.java)



Итак, был создан уровень модели и активность, которая выполняет хостинг фрагмента.

*Определение контейнерного представления*

UI-фрагмент будет добавлен в коде активности-хоста, но все равно необходимо найти место для представления фрагмента в иерархии представлений активности. В макете BookActivity этим местом будет элемент FrameLayout, изображенный на рисунке 6.5.

Элемент FrameLayout станет *контейнерным представлением* для BookFragment. Контейнерное представление абсолютно универсально; оно не привязывается к классу BookFragment. Один макет будет использовать для хостинга разных фрагментов.

Найти макет BookActivity в файле res/layout/activity\_book.xml. Открыть файл и заменить макет по умолчанию элементом FrameLayout.

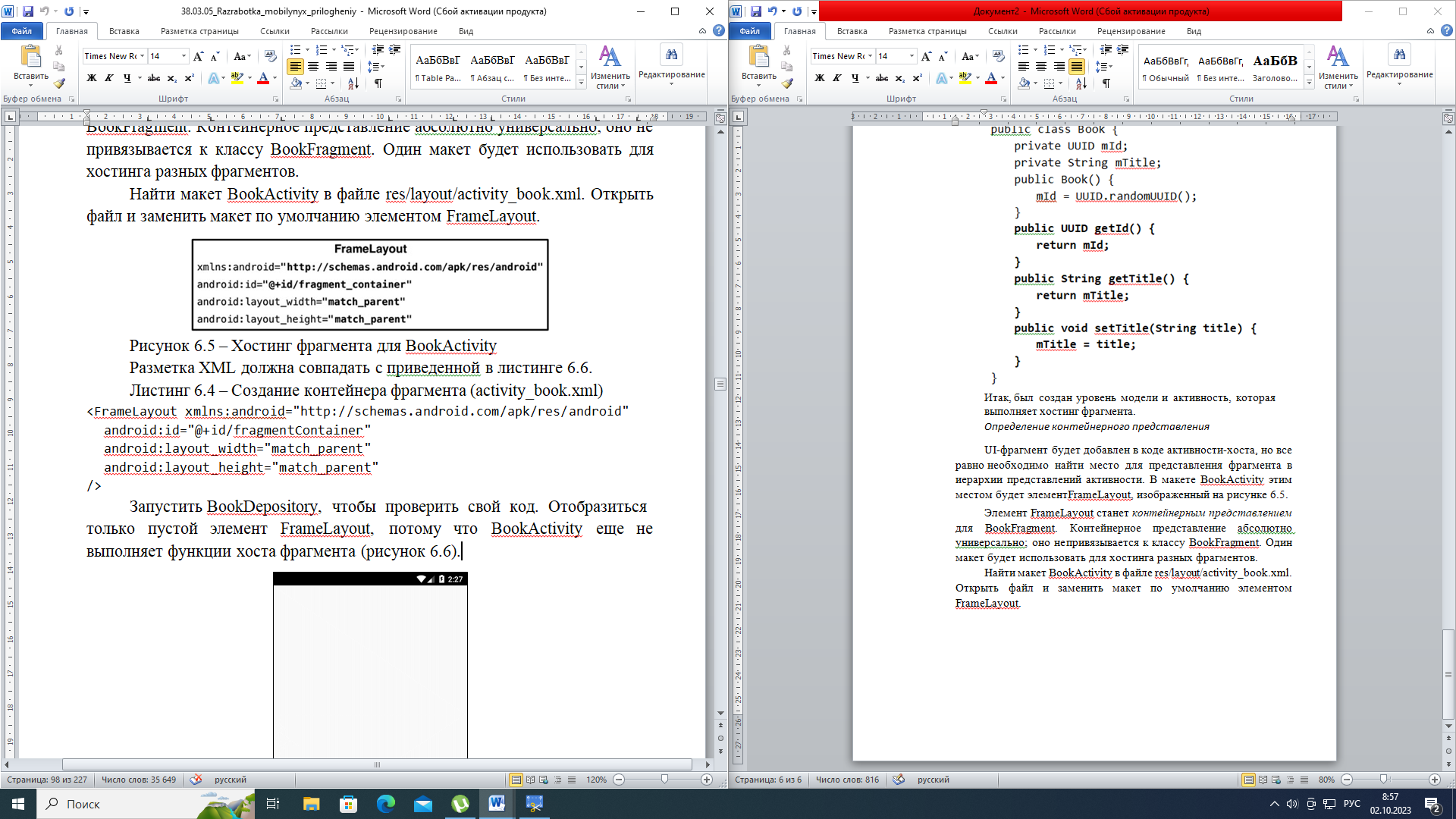


Рисунок 6.5 – Хостинг фрагмента для BookActivity

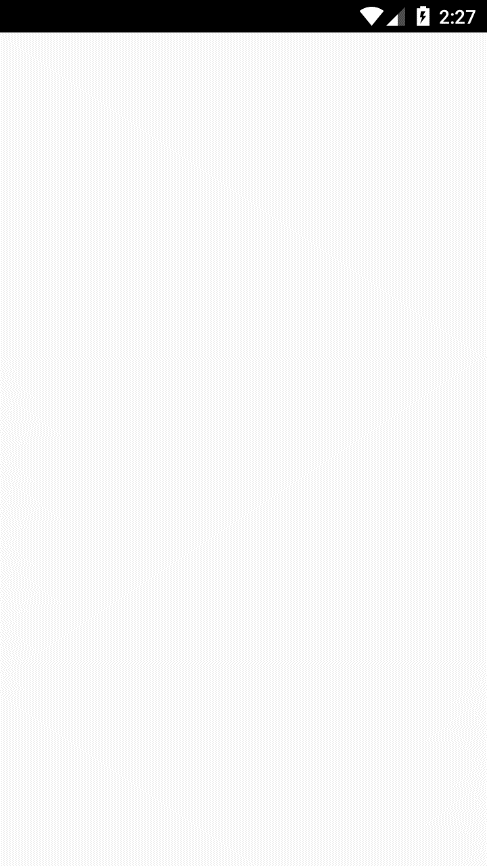
Разметка XML должна совпадать с приведенной в листинге 6.6. Листинг 6.4 – Создание контейнера фрагмента (activity\_book.xml)

<FrameLayout xmlns:android="<http://schemas.android.com/apk/res/android>" android:id="@+id/fragmentContainer" android:layout\_width="match\_parent" android:layout\_height="match\_parent"

/>

Запустить BookDepository, чтобы проверить свой код. Отобразиться только пустой элемент FrameLayout, потому что BookActivity еще не выполняет функции хоста фрагмента (рисунок 6.6).

Рисунок 6.6 – Пустой элемент FrameLayout



## Создание UI-фрагмента

Представление BookFragment будет отображать информацию, содержащуюся в экземпляре Book. На рисунке 6.7 изображен макет представления BookFragment. Он состоит из вертикального элемента LinearLayout, содержащего EditText — виджет с областью для ввода и редактирования текста.

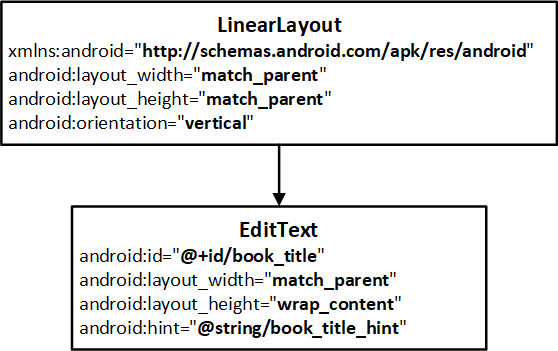
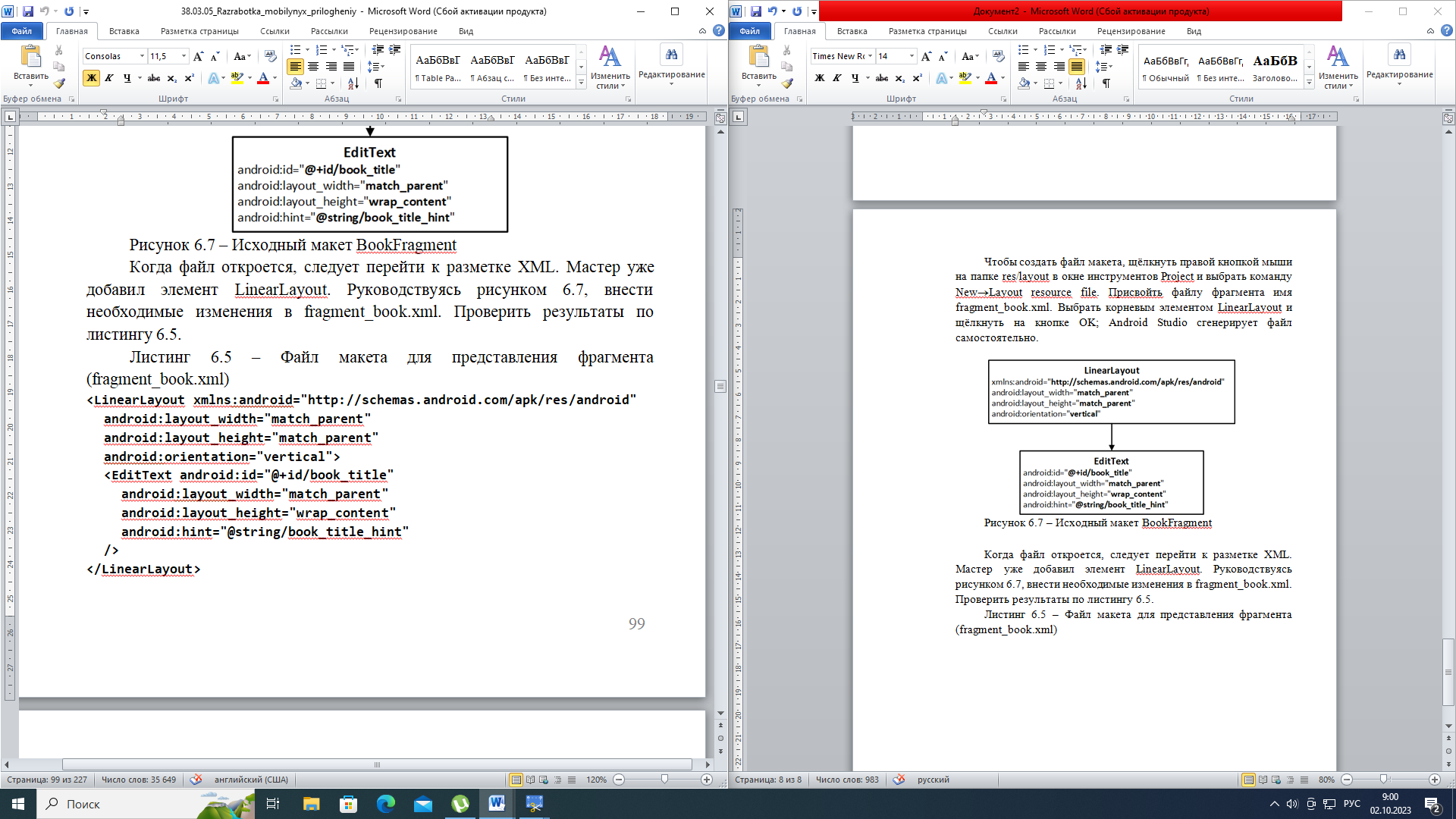
Чтобы создать файл макета, щёлкнуть правой кнопкой мыши на папке res/layout в окне инструментов Project и выбрать команду NewLayout resource file. Присвойть файлу фрагмента имя fragment\_book.xml. Выбрать корневым элементом LinearLayout и щёлкнуть на кнопке OK; Android Studio сгенерирует файл самостоятельно.

Рисунок 6.7 – Исходный макет BookFragment

Когда файл откроется, следует перейти к разметке XML. Мастер уже добавил элемент LinearLayout. Руководствуясь рисунком 6.7, внести необходимые изменения в fragment\_book.xml. Проверить результаты по листингу 6.5.

Листинг 6.5 – Файл макета для представления фрагмента (fragment\_book.xml)



Открыть файл res/values/strings.xml, добавьте строковый ресурс book\_title\_hint.

Листинг 6.6 – Добавление и удаление строк (res/values/strings.xml)

<?xml version="1.0" encoding="utf-8"?>

<resources>

<string name="app\_name">BookDepository</string>

**~~<string name="hello\_world">Hello world!</string>~~**

<string name="menu\_settings">Settings</string>

<string name="title\_activity\_book">BookActivity</string>

**<string name="book\_title\_hint">Введите название книги.</string>**

</resources>

Сохранить файлы. Вернуться к fragment\_book.xml, чтобы увидеть предварительное изображение представления фрагмента.

***Создание класса BookFragment***

Щёлкнуть правой кнопкой мыши на пакете ru.rsue.android и выбрать команду NewJava Class. Ввести имя класса BookFragment и щёлкнуть на кнопке OK, чтобы сгенерировать класс.

Теперь этот класс нужно преобразовать во фрагмент. Изменить класс BookFragment так, чтобы он субклассировал **Fragment**.

Листинг 6.7 – Субклассирование класса Fragment (BookFragment.java)

public class BookFragment **extends Fragment** {

}

При субклассировании Fragment Android Studio находит два класса с именем Fragment: Fragment (android.app) и Fragment (android.support.v4.app). Версия Fragment из android.app реализует версию фрагментов, встроенную в ОС Android. В приложении будет использована версия библиотеки поддержки, поэтому при появлении диалогового окна выбрать версию класса Fragment из android.support.v4.app (рисунок 6.8).

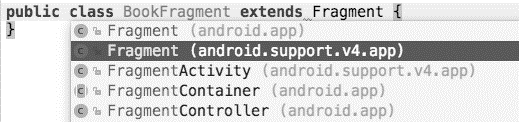
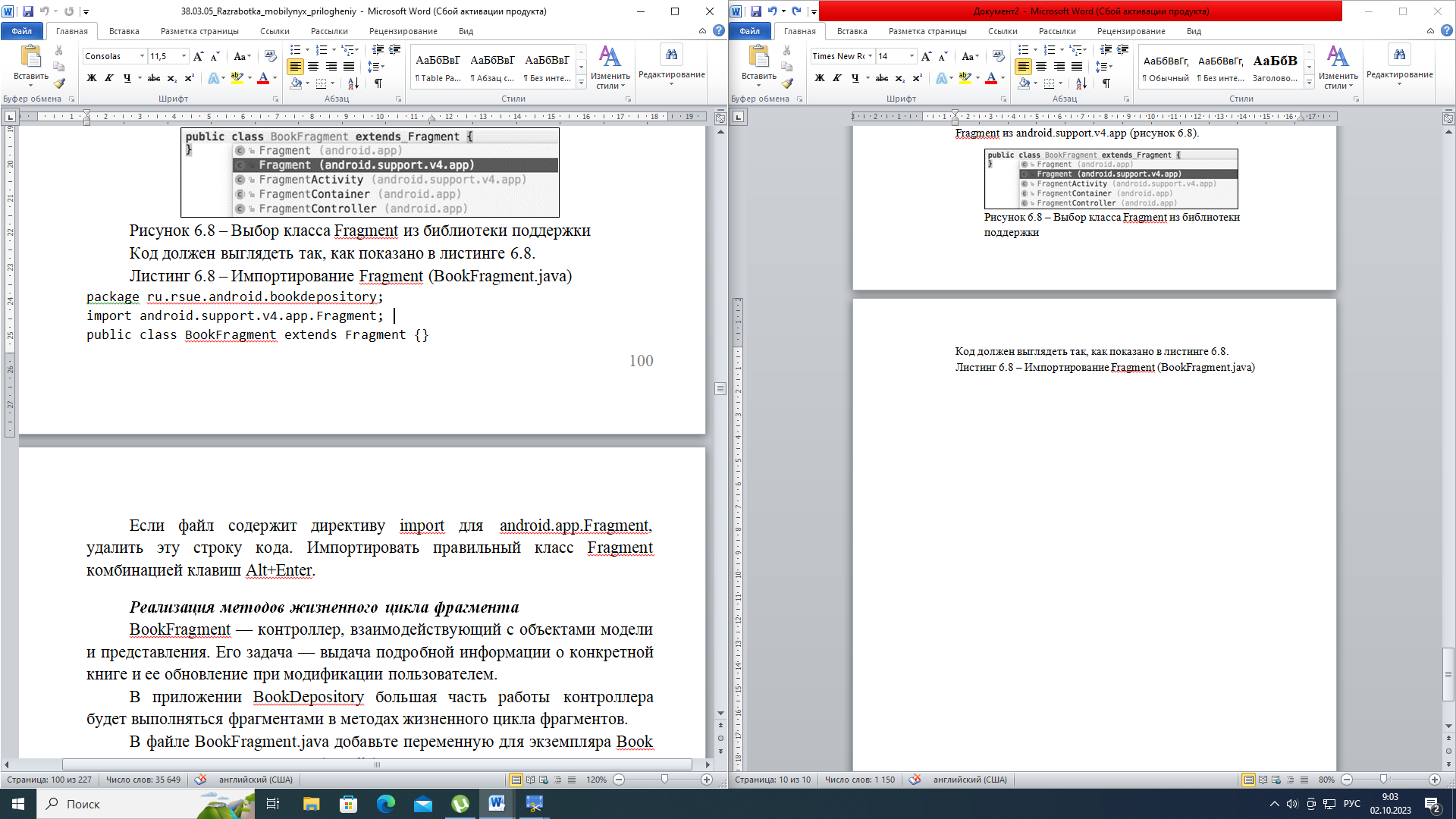


Рисунок 6.8 – Выбор класса Fragment из библиотеки поддержки

Код должен выглядеть так, как показано в листинге 6.8.

Листинг 6.8 – Импортирование Fragment (BookFragment.java)



Если файл содержит директиву import для android.app.Fragment, удалить эту строку кода. Импортировать правильный класс Fragment комбинацией клавиш Alt+Enter.

## Реализация методов жизненного цикла фрагмента

BookFragment — контроллер, взаимодействующий с объектами модели и представления. Его задача — выдача подробной информации о конкретной книге и ее обновление при модификации пользователем.

В приложении BookDepository большая часть работы контроллера будет выполняться фрагментами в методах жизненного цикла фрагментов.

В файле BookFragment.java добавьте переменную для экземпляра Book и реализацию Fragment.onCreate(Bundle).

Листинг 6.9 – Переопределение Fragment.onCreate(Bundle) (BookFragment.java)

public class BookFragment extends Fragment {

**private Book mBook; @Override**

**public void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState);**

**mBook = new Book();**

**}**

}

В файле BookFragment.java добавить реализацию onCreateView(…),

которая заполняет разметку fragment\_book.xml.

Листинг 6.10 – Переопределение onCreateView(…) (BookFragment.java)

public class BookFragment extends Fragment { private Book mBook;

...

**@Override**

**public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {**

**View v = inflater.inflate(R.layout.fragment\_book, container, false);**

**return v;**

**}**

}

В методе onCreateView(…) явно заполняется представление фрагмента, вызывая LayoutInflater.inflate(…) с передачей идентификатора ресурса макета. Второй параметр определяет родителя представления, что обычно необходимо для правильной настройки виджета. Третий параметр указывает, нужно ли включать заполненное представление в родителя. В данном случае передается false, потому что представление будет добавлено в коде активности.

***Подключение виджетов во фрагменте***

В методе onCreateView(…) также настраивается реакция виджета EditText на ввод пользователя. После того как представление будет заполнено, метод получает ссылку на EditText и добавляет слушателя.

Листинг 6.11 – Настройка виджета EditText (BookFragment.java)

public class BookFragment extends Fragment { private Book mBook;

**private EditText mTitleField;**

...

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {

View v = inflater.inflate(R.layout.fragment\_book, parent, false); **mTitleField = (EditText)v.findViewById(R.id.book\_title); mTitleField.addTextChangedListener(new TextWatcher() {**

**@Override**

**public void beforeTextChanged(**

**CharSequence s, int start, int count, int after) {**

**// Здесь намеренно оставлено пустое место**

**}**

**@Override**

**public void onTextChanged(**

**CharSequence s, int start, int before, int count) { mBook.setTitle(s.toString());**

**}**

**@Override**

**public void afterTextChanged(Editable c) {**

**// И здесь тоже**

**}**

**});**

return v;

}

}

Получение ссылок в Fragment.onCreateView(…) происходит практически так же, как в Activity.onCreate(…). Единственное различие заключается в том, что для представления фрагмента вызывается метод View.findViewById(int). Метод Activity.findViewById(int), который был использован ранее, является вспомогательным методом, вызывающим View.findViewById(int) в своей внутренней реализации. У класса Fragment аналогичного вспомогательного метода нет, поэтому следует вызывать основной метод.

Назначение слушателей во фрагменте работает точно так же, как в активности. В листинге 6.11 создается анонимный класс, который реализует интерфейс слушателя TextWatcher. Этот интерфейс содержит три метода, но в данный момент интерес представляет только один: onTextChanged(…).

В методе onTextChanged(…) вызывается toString() для объекта CharSequence, представляющего ввод пользователя. Этот метод возвращает строку, которая затем используется для задания заголовка Book.

Код BookFragment готов.

## Добавление UI-фрагмента в FragmentManager

Когда в Honeycomb появился класс Fragment, в класс Activity были внесены изменения: в него был добавлен компонент, называемый FragmentManager. Он отвечает за управление фрагментами и добавление их представлений в иерархию представлений активности (рисунок 6.9).

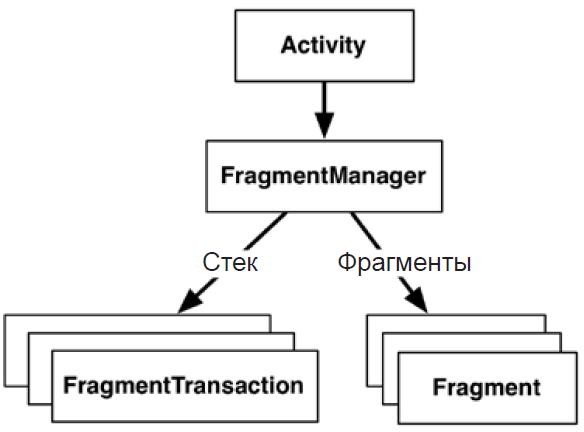
FragmentManager управляет двумя структурами: списком фрагментов и стеком транзакций фрагментов.

Рисунок 6.9 – FragmentManager

Чтобы добавить фрагмент в активность в коде, следует обратиться с вызовом к объекту FragmentManager активности. Прежде всего необходимо получить сам объект FragmentManager. В BookActivity.java включите следующий код в onCreate(…).

Листинг 6.12 – Получение объекта FragmentManager (BookActivity.java)

public class BookActivity extends FragmentActivity { @Override

public void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); setContentView(R.layout.activity\_book); **FragmentManager fm = getSupportFragmentManager();**

}

}

Метод getSupportFragmentManager() вызывается, потому что в приложении используется библиотека поддержки и класс FragmentActivity. Если бы библиотека поддержки не использовалась, то вместо FragmentActivity можно было бы субклассировать Activity и вызвать getFragmentManager().

## Транзакции фрагментов

После получения объекта FragmentManager добавить следующий код, который передает ему фрагмент для управления.

Листинг 6.13 – Добавление BookFragment (BookActivity.java)

public class BookActivity extends FragmentActivity { @Override

public void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); setContentView(R.layout.activity\_book); FragmentManager fm = getSupportFragmentManager();

**Fragment fragment = fm.findFragmentById(R.id.fragmentContainer); if (fragment == null) {**

**fragment = new BookFragment(); fm.beginTransaction()**

**.add(R.id.fragmentContainer, fragment)**

**.commit();**

**}**

}

}

Следует обратить внимание на метод add(…) и окружающий его код. Этот код создает и закрепляет *транзакцию фрагмента*.

Транзакции фрагментов используются для добавления, удаления, присоединения, отсоединения и замены фрагментов в списке фрагментов. Они лежат в основе механизма использования фрагментов для формирования и модификации экранов во время выполнения. FragmentManager ведет стек транзакций, по которому можно перемещаться.

Метод FragmentManager.beginTransaction() создает и возвращает экземпляр FragmentTransaction. Класс FragmentTransaction использует *динамичный интерфейс*: методы, настраивающие FragmentTransaction, возвращают FragmentTransaction вместо void, что позволяет объединять их вызовы в цепочку. Таким образом, выделенный код в листинге 6.13 означает: «Создать новую транзакцию фрагмента, включить в нее одну операцию add, а затем закрепить».

Теперь BookActivity является хостом для BookFragment. Чтобы убедиться в этом, запустить приложение BookDepository. На экране отображается представление из файла fragment\_book.xml (рисунок 6.2).

# МАКЕТЫ И ВИДЖЕТЫ

В данном разделе будут более подробно рассмотрены макеты и виджеты, а также в BookDepository будет включено хранение даты и статуса.

*Обновление Book*

Открыть файл Book.java и добавить два новых поля. Поле типа Date представляет дату, к которой следует прочитать книгу, а поле типа boolean — признак того, что книга была прочитана.

Листинг 7.1 – Добавление полей в класс Book (Book.java)

public class Book { private UUID mId; private String mTitle; **private Date mDate; private boolean mReaded;**

public Book() {

mId = UUID.randomUUID();

**mDate = new Date();**

}

...

}

Возможно, Android Studio найдет два класса с именем Date.

Использовать комбинацию клавиш Alt+Enter, чтобы импортировать класс вручную. Когда будет предложено выбрать импортируемую версию класса Date, надо выбрать версию java.util.Date.

Инициализация переменной Date конструктором Date по умолчанию присваивает mDate текущую дату.

Затем сгенерируйте get- и set-методы для новых полей (щёлкнуть правой кнопкой мыши на файле и выбрать команду Generate…Getter and Setter).

Листинг 7.2 – Сгенерированные get- и set-методы (Book.java)

public class Book {

...

public void setTitle(String title) { mTitle = title;

}

**public Date getDate() { return mDate;**

**}**

**public void setDate(Date date) { mDate = date;**

**}**

**public boolean isReaded() {**

**return mReaded;**

**}**

**public void setReaded(boolean readed) { mReaded = readed;**

**}**

}

Далее будет выполнено обновление макета в fragment\_book.xml новыми виджетами и их связывание с виджетами в BookFragment.java.

*Обновление макета*

На рисунке 7.1 представлено, каким образом представление BookFragment будет выглядеть в конце раздела.

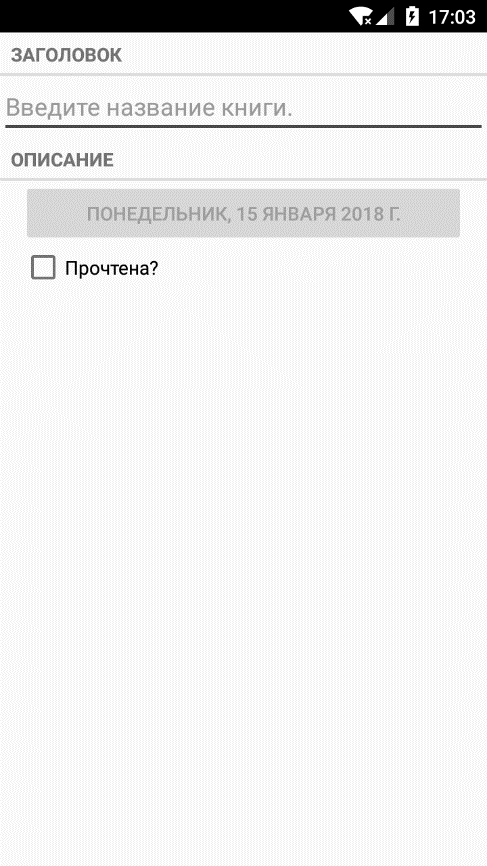


Рисунок 7.1 – Приложение BookDepository в конце раздела 7

Чтобы добиться этого экрана, в макет BookFragment будет добавлено четыре виджета: два виджета TextView, Button и CheckBox.

Открыть файл fragment\_book.xml и внести изменения, представленные в листинге 7.3. Возможно, возникнут ошибки отсутствия строковых ресурсов — вскоре они будут созданы.

Листинг 7.3 – Добавление новых виджетов (fragment\_book.xml)

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android=

"<http://schemas.android.com/apk/res/android>"

...

>

**<TextView**

**android:layout\_width="match\_parent"**

**android:layout\_height="wrap\_content" android:text="@string/book\_title\_label" style="?android:listSeparatorTextViewStyle"**

**/>**

<EditText android:id="@+id/book\_title" android:layout\_width="match\_parent" android:layout\_height="wrap\_content" android:layout\_marginLeft="16dp" android:layout\_marginRight="16dp" android:hint="@string/book\_title\_hint"

/>

**<TextView**

**android:layout\_width="match\_parent" android:layout\_height="wrap\_content" android:text="@string/book\_details\_label" style="?android:listSeparatorTextViewStyle"**

**/>**

**<Button android:id="@+id/book\_date" android:layout\_width="match\_parent" android:layout\_height="wrap\_content" android:layout\_marginLeft="16dp" android:layout\_marginRight="16dp"**

**/>**

**<CheckBox android:id="@+id/book\_readed" android:layout\_width="match\_parent" android:layout\_height="wrap\_content" android:layout\_marginLeft="16dp" android:layout\_marginRight="16dp" android:text="@string/book\_readed\_label"**

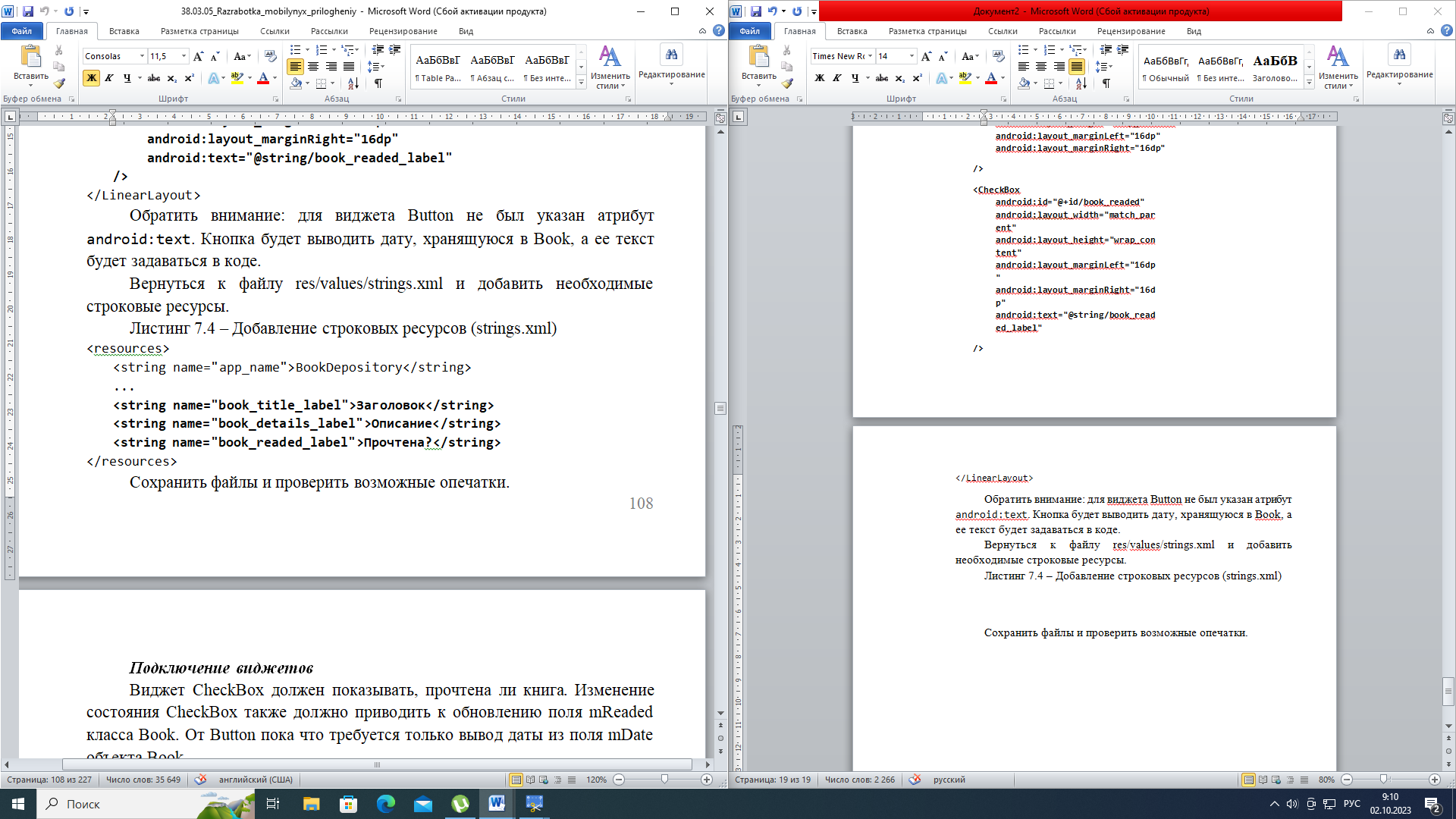
**/>**

</LinearLayout>

Обратить внимание: для виджета Button не был указан атрибут android:text. Кнопка будет выводить дату, хранящуюся в Book, а ее текст будет задаваться в коде.

Вернуться к файлу res/values/strings.xml и добавить необходимые строковые ресурсы.

Листинг 7.4 – Добавление строковых ресурсов (strings.xml)



Сохранить файлы и проверить возможные опечатки.

## Подключение виджетов

Виджет CheckBox должен показывать, прочтена ли книга. Изменение состояния CheckBox также должно приводить к обновлению поля mReaded класса Book. От Button пока что требуется только вывод даты из поля mDate объекта Book.

В файле BookFragment.java добавить две переменные экземпляра.

Листинг 7.5 – Добавление переменных экземпляра для виджетов (BookFragment.java)

public class BookFragment extends Fragment { private Book mBook;

private EditText mTitleField; **private Button mDateButton; private CheckBox mReadedCheckBox;** @Override

public void onCreate(Bundle savedInstanceState) {

...

Затем в onCreateView(…) получите ссылку на новую кнопку, задайте в тексте кнопки дату прочтения книги и заблокируйте ее.

Листинг 7.6 – Назначение текста Button (BookFragment.java)

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup parent, Bundle savedInstanceState) {

View v = inflater.inflate(R.layout.fragment\_book, parent, false);

...

mTitleField.addTextChangedListener(new TextWatcher() {

...

});

**mDateButton = (Button)v.findViewById(R.id.book\_date); mDateButton.setText(mBook.getDate().toString()); mDateButton.setEnabled(false);**

return v;

}

Блокировка кнопки гарантирует, что она не будет реагировать на

нажатия. При этом изменяется оформление кнопки, чтобы сообщить пользователю о заблокированном состоянии. В дальнейшем блокировка кнопки будет снята при назначении слушателя.

Теперь можно заняться CheckBox: необходимо получить ссылку и назначить слушателя, который будет обновлять поле mReaded объекта Book.

Листинг 7.7 Назначение слушателя для изменений CheckBox (BookFragment.java)

...

mDateButton = (Button)v.findViewById(R.id.book\_date); mDateButton.setText(mBook.getDate().toString()); mDateButton.setEnabled(false);

**mReadedCheckBox = (CheckBox)v.findViewById(R.id.book\_readed); mReadedCheckBox.setOnCheckedChangeListener(new**

**OnCheckedChangeListener() { @Override**

**public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {**

**// Назначение флага прочтения книги**

**mBook.setReaded(isChecked);**

**}**

**});**

return v;

}

При создании OnCheckedChangeListener будет предложено выбрать один из двух вариантов импортирования. Следует выбрать версию android.widget.CompoundButton.

Запустить BookDepository. Переключить состояние CheckBox и убедиться в том, что кнопка заблокирована, и на ней отображается текущая дата.

## Поля и отступы

В файле fragment\_book.xml виджетам назначаются атрибуты margin и padding. Атрибуты margin являются параметрами макета; они определяют расстояние между виджетами. Так как виджет располагает информацией только о самом себе, за соблюдение полей отвечает родитель виджета.

Напротив, отступ не является параметром макета. Атрибут android:padding сообщает виджету, с каким превышением размера содержимого он должен прорисовывать себя. Можно допустить, необходимо заметно увеличить размеры кнопки даты без изменения размера текста. Добавить в Button следующий атрибут, сохранить макет и запустить приложение заново (рисунок 7.3).

Листинг 7.8 – Назначение отступов

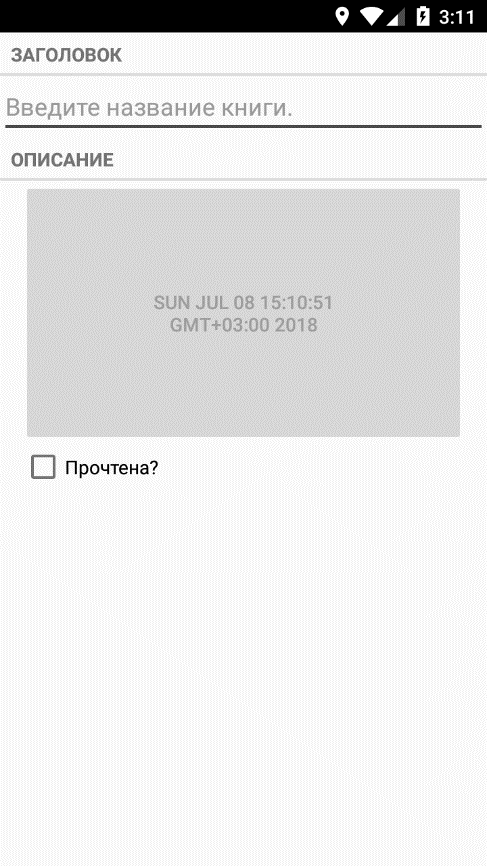
<Button android:id="@+id/book\_date"

...

android:layout\_marginLeft="16dp" android:layout\_marginRight="16dp" **android:padding="80dp"** />

Следует удалить этот атрибут, прежде чем продолжать работу.

Рисунок 7.3 – Для любителей больших кнопок



## Использование графического конструктора

До настоящего момента макеты создавались, вводя разметку XML. В этом разделе будет использован графический конструктор для построения альтернативного альбомного макета BookFragment.

Большинство встроенных классов макетов, таких, как: LinearLayout,

— автоматически изменяют размеры себя и своих потомков при поворотах. Однако в некоторых случаях изменение размеров по умолчанию недостаточно эффективно использует свободное пространство.

Запустить приложение BookDepository и повернуть устройство, чтобы увидеть макет BookFragment в альбомной ориентации (рисунок 7.4).

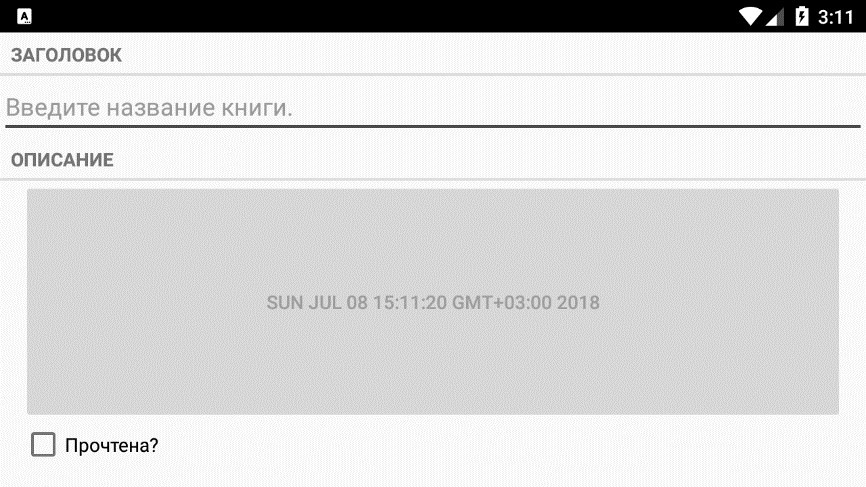


Рисунок 7.4 – BookFragment в альбомной ориентации

Кнопка даты становится слишком длинной; было бы лучше, если бы в альбомной ориентации кнопка и флажок располагались рядом друг с другом.

Чтобы внести эти изменения, переключить в графический конструктор. Открыть файл fragment\_book.xml и выбрать вкладку Design в нижней части панели.

Графический конструктор может сгенерировать альбомную версию файла макета. Найти кнопку с изображением листа бумаги в правом верхнем углу (рисунок 7.5). Щёлкнуть на этой кнопке и выбрать команду Create Landscape Variation.



Рисунок 7.5 – Создание альтернативного макета в графическом конструкторе

На экране появляется новый макет. При этом автоматически создается каталог res/layout-land, а существующий файл макета fragment\_book.xml копируется в новый каталог.

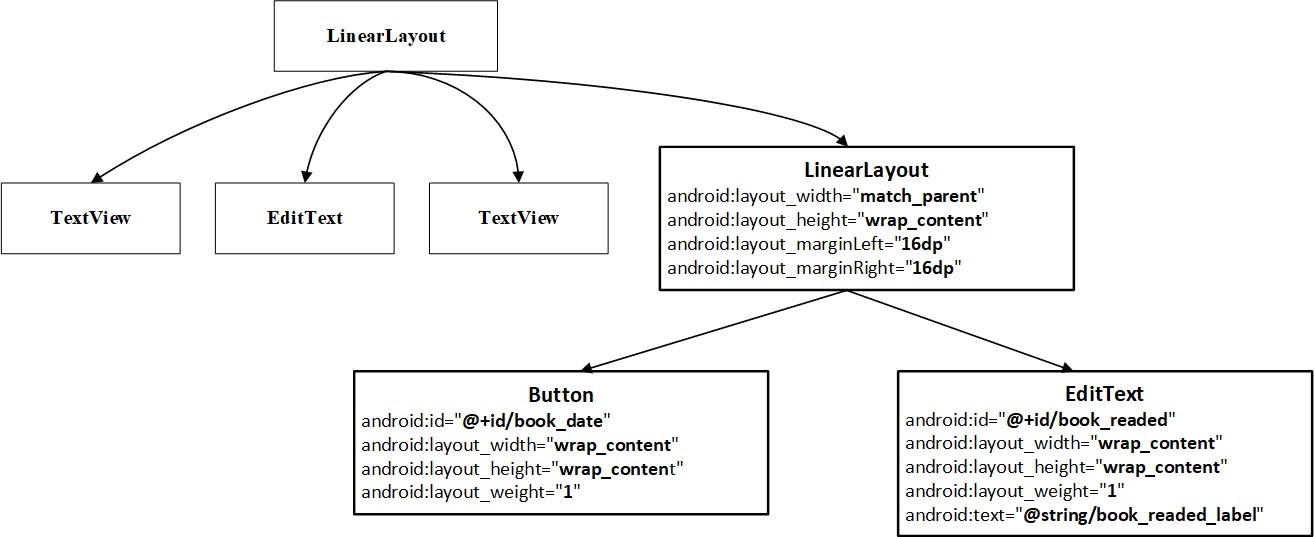
Следующие изменения необходимо внести в альбомный макет (рисунок 7.6).

Рисунок 7.6 – Альбомный макет для BookFragment

Изменения можно разделить на четыре группы:

* 1. Добавление виджета LinearLayout в макет.
  2. Редактирование атрибутов LinearLayout.
  3. Назначение виджетов Button и CheckBox потомками LinearLayout.
  4. Обновление параметров макета Button и CheckBox.

*Добавление нового виджета*

Чтобы добавить виджет, можно выделить его в палитре и перетащить на дерево компонентов. Щёлкнуть на категории Layouts в палитре, если она еще не раскрыта. Выбрать пункт LinearLayout (Horizontal) и перетащить его на дерево компонентов. Отпустить перетаскиваемый объект над кнопкой даты. Убедиться в том, что новый виджет LinearLayout является потомком корневого элемента LinearLayout, как показано на рисунке 7.7.

Виджеты также можно добавлять перетаскиванием из палитры в область предварительного просмотра. Однако виджеты Layout часто пусты или закрыты другими представлениями, поэтому может быть трудно понять, где следует разместить виджет в области предварительного просмотра для получения нужной иерархии. Перетаскивание на дерево компонентов существенно упрощает выполнение этой операции.

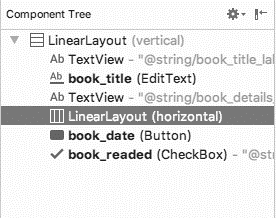


Рисунок 7.7 – Добавление LinearLayout в fragment\_book.xml

*Редактирование атрибутов в свойствах*

Выбрать новый виджет LinearLayout в дереве компонентов, чтобы отобразить его атрибуты на панели свойств. Найти атрибуты layout:width и layout:height.

Присвоить атрибуту layout:width значение match\_parent, а атрибуту layout:height — значение wrap\_content, как показано на рисунке 7.8 – Теперь виджет LinearLayout заполняет всю доступную ширину и занимает столько места по вертикали, сколько необходимо для отображения CheckBox и Button.

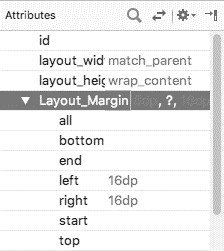
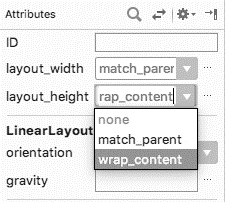
Необходимо, чтобы поля нового виджета LinearLayout совпадали с полями других виджетов. Открыть атрибут layout:margin, выделить поле рядом с Left и ввести значение 16dp. Сделать то же самое для правого (Right) поля (рисунок 7.9).

Рисунок 7.8 – Изменение

ширины и высоты LinearLayout

Рисунок 7.9 – Назначение полей

в режиме свойств



Сохранить файл макета и переключиться на разметку XML при помощи вкладки text в нижней части области предварительного просмотра. Только что добавленный элемент LinearLayout с атрибутами полей будет доступен для просмотра.

*Реорганизация виджетов в дереве компонентов*

Следующий шаг — назначение виджетов Button и CheckBox потомками нового виджета LinearLayout. Вернуться к графическому конструктору, выбрать виджет Button и перетащите его на LinearLayout.

В дереве компонентов отражается тот факт, что Button теперь является потомком нового виджета LinearLayout (рисунок 7.10). Проделайте то же самое с CheckBox.

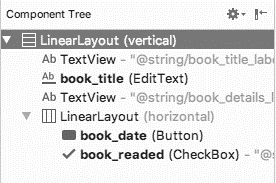


Рисунок 7.10 – Button и Checkbox становятся потомками нового виджета LinearLayout

Если потомки размещаются не в том порядке, их можно переупорядочить посредством перетаскивания. Также в дереве компонентов можно удалять виджеты из макета, но: удаление виджета приводит к удалению его потомков.

В области предварительного просмотра виджет CheckBox не виден — его скрывает Button. Виджет LinearLayout проверил ширину (match\_parent) своего первого потомка (Button) и выделил ему все пространство, ничего не оставив CheckBox (рисунок 7.11).



Рисунок 7.11 – Виджет Button, который был определен первым, скрывает CheckBox

Чтобы восстановить равноправие потомков LinearLayout, следует изменить параметры макета потомков.

*Обновление параметров макета потомков*

Сначала выделить кнопку даты в дереве компонентов. На панели свойств щёлкнуть на текущем значении layout:width и заменить его значением wrap\_content. Удалить оба значения 16dp полей кнопки. Теперь, когда кнопка находится внутри LinearLayout, поля ей не нужны. Наконец, найти поле layout:weight в разделе Layout Parameters и задать ему значение

1. Это поле соответствует атрибуту android:layout\_weight на рисунке 7.6.

Выбрать виджет CheckBox в дереве компонентов и внести те же изменения: атрибут layout\_width должен содержать wrap\_content, атрибуты полей должны быть пустыми, а атрибут Weight должен быть равен 1.

В области предварительного просмотра надо убедиться в том, что оба виджета теперь видны. Сохранить файл и вернуться к XML, чтобы подтвердить изменения. В листинге 7.9 приведена соответствующая разметка XML.

Листинг 7.9 – Разметка XML макета, созданного в графическом конструкторе (layout-land/fragment\_book.xml)

...

<TextView

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content" android:text="@string/book\_details\_label" style="?android:listSeparatorTextViewStyle"

/>

<LinearLayout android:layout\_width="match\_parent" android:layout\_height="wrap\_content" android:layout\_marginLeft="16dp" android:layout\_marginRight="16dp" >

<Button

android:id="@+id/book\_date" android:layout\_width="wrap\_content" android:layout\_height="wrap\_content" android:layout\_weight="1" />

<CheckBox

android:id="@+id/book\_readed" android:layout\_width="wrap\_content" android:layout\_height="wrap\_content" android:layout\_weight="1" android:text="@string/book\_readed\_label" />

</LinearLayout>

</LinearLayout>

Запустить BookDepository, повернуть устройство в альбомную ориентацию и убедиться в том, что макет для новой конфигурации устройства успешно оптимизирован.

**Самостоятельные задания.**

*Задание. Форматирование даты*

На текущий момент объект Date больше напоминает временную метку (timestamp), чем традиционную дату. При вызове toString() для Date возвращается именно временная метка, которая отображается на кнопке. Временные метки хорошо подходят для отчетов, но на кнопке было бы лучше выводить дату в формате, более привычном для человека (например,

«15 января 2018»). Для этого можно воспользоваться экземпляром класса android.text.format.DateFormat. Для выполнения этого задания надо ознакомиться с описанием этого класса в документации Android.

Использовать методы класса DateFormat для формирования строки в стандартном формате или же подготовить собственную форматную строку. Также можно создать форматную строку для вывода дня недели («Понедельник, 15 января 2018»). Например, так: mDateButton.setText(DateFormat.getDateInstance(...).format(mBook

.getmDate()));