## 15. ИНТЕНТЫ ПРИ РАБОТЕ С КАМЕРОЙ

В данном разделе будет реализовано взаимодействие приложения BookDepository с камерой. Располагая снимком обложки книги, записи становятся более интересными, при этом ими можно поделиться со всеми желающими.

Для создания снимков потребуется пара новых инструментов, которые используются в сочетании с уже знакомыми неявными интентами. Неявный интент используется при запуске приложения для работы с камерой и получения от него нового снимка.

***Место для хранения фотографий***

Прежде всего следует обеспечить место, в котором будет отображаться фотография. Для этого понадобятся два новых объекта View: ImageView для отображения фотографии на экране и Button для создания снимка (рисунок 15.1).

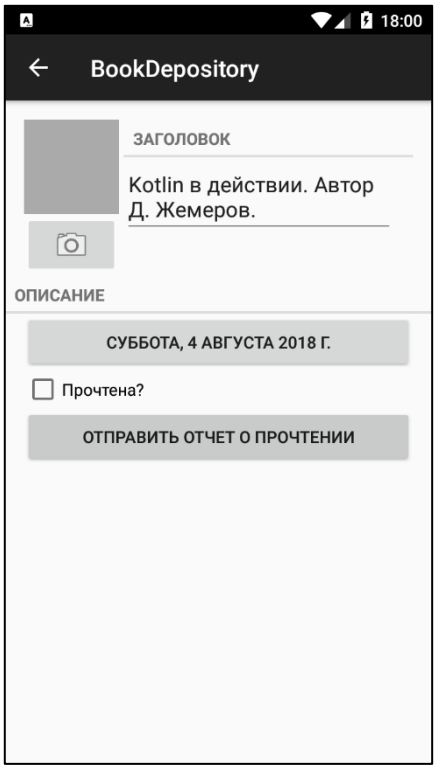


Рисунок 15.1 – Новые элементы интерфейса детализации

*Включение файлов макетов*

Новый макет будет включать большой раздел, который будет выглядеть одинаково как в книжной, так и в альбомной версии fragment_book.xml. Конечно, можно просто продублировать этот раздел res/layout/fragment_book.xml и res/layout-land/fragment_book.xml. Более эфективное решение —воспользоваться механизмом включения.

*Включение* (include) позволяет встроить один файл макета в другой. В данном случае встраивается раздел, содержащий общие элементы. Работа начинается с создания файла макета для части представления, изображенной на рисунке 15.2.

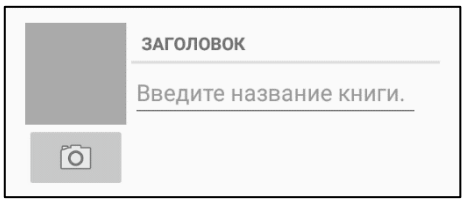


Рисунок 15.2 – Камера и текст

Присвоить файлу макета имя view_camera_and_title.xml. Начать с построения левой стороны (листинг 15.1).

Листинг 15.1 – Определение виджетов ImageView и ImageButton (view_camera_and_title.xml)

```
<?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:layout_marginLeft="16dp"
    android:layout_marginRight="16dp"
    android:layout_marginTop="16dp">
    <LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:layout_marginRight="4dp">
    <ImageView
        android:id="@+id/book_photo"
        android:layout_width="80dp"
        android:layout_height="80dp"
        android:scaleType="centerInside"
        android:background="@android:color/darker_gray"
        android:cropToPadding="true"/>
    <ImageButton
        android:id="@+id/book_camera"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:src="@android:drawable/ic_menu_camera"/>
    </LinearLayout>
</LinearLayout>
```

Затем создать правую сторону (листинг 15.2).

Листинг 15.2 – Определение виджетов TextView и EditText (view_camera_and_title.xml)

```
<LinearLayout
    ...
    <LinearLayout
        ...
    </LinearLayout>
    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:layout_weight="1">
        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/book_title_label"
            style="?android:listSeparatorTextViewStyle"/>
        <EditText
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:id="@+id/book_title"
            android:layout_marginRight="16dp"
            android:hint="@string/book_title_hint"/>
    </LinearLayout>
</LinearLayout>
```

Перейти в режим графического конструктора и убедиться в том, что файл макета выглядит так, как показано на рисунке 15.2.

Для включения этого макета в другие файлы макетов используются теги include. В теге include атрибут layout не использует обычный префикс android.

Начать следует с изменения главного файла макета (рисунок 15.3).

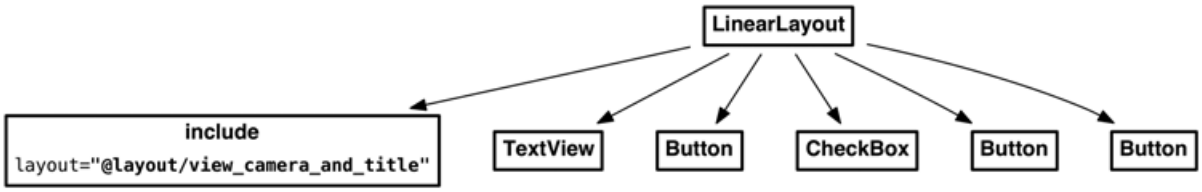


Рисунок 15.3 – Включение макета камеры для книжной ориентации (res/layout/fragment_book.xml)

Затем проделать то же для альбомного макета (рисунок 15.4).

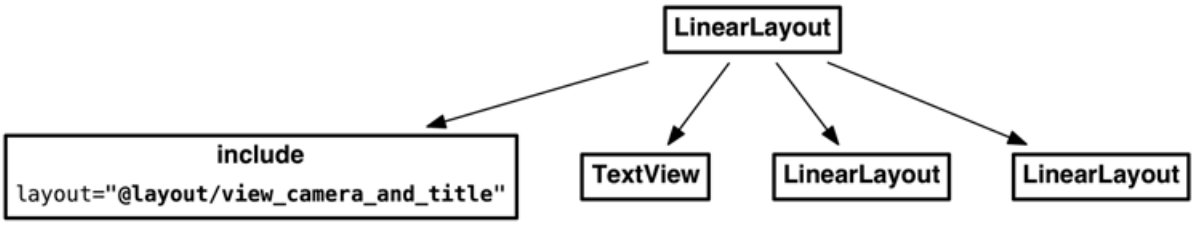


Рисунок 15.4 – Включение макета камеры для альбомной ориентации (res/layout-land/fragment_book.xml)

Запустить приложение BookDepository; новый пользовательский интерфейс должен выглядеть так, как показано на рисунке 15.1.

***Внешнее хранилище***

Фотографиям недостаточно одного лишь места на экране. Полноразмерная фотография слишком велика для хранения в базе данных SQLite. Ей необходимо место для хранения в файловой системе устройства.

Обычно такие данные размещаются в закрытом (приватном) хранилище. Такие методы, как: `Context.getFileStreamPath(String)` и `Context.getFilesDir()`, позволяют хранить в папке по соседству с папкой databases, в которой размещается база данных SQLite, и обычные файлы. В таблице 15.1 перечислены методы для работы с внешними файлами и каталогами в Context.

Таблица 15.1 – Методы для работы с внешними файлами и каталогами в Context

| Метод | Назначение |
|---|---|
| File getFilesDir() | Возвращает дескриптор каталога для закрытых файлов приложения |
| FileInputStream openFileInput(String name) | Открывает существующий файл для ввода (относительно каталога файлов) |
| FileOutputStream openFileOutput(String name, int mode) | Открывает существующий файл для вывода, возможно, с созданием (относительно каталога файлов) |
| File getDir(String name, int mode) | Получает (и, возможно, создает) подкаталог в каталоге файлов |
| String[] fileList() | Получает список имен файлов в главном каталоге файлов (например, для использования с `openFileInput(String)`) |
| File getCacheDir() | Возвращает дескриптор каталога, используемого |

| | |
|---|---|
| | для хранения кэш-файлов. Будьте внимательны, поддерживайте порядок в этом каталоге и старайтесь использовать как можно меньше пространства |

Если сохраняются файлы, которые впоследствии будут использованы другим приложением, или возвращаются файлы от другого приложения (как, например, сохраненные фотографии), хранение должно осуществляться во внешнем хранилище.

Память внешнего хранилища делится на два вида: первичная область и все остальное. На всех устройствах Android присутствует по крайней мере одна область для внешнего хранения: *первичная* (primary) область, которая находится в папке, возвращаемой `Environment.getExternalStorageDirectory()`. Это может быть и SD-карта, но в наши дни она чаще интегрируется в самом устройстве. На некоторых устройствах имеется дополнительная внешняя память; она относится к категории «все остальное».

Класс Context тоже предоставляет методы для получения доступа к внешнему хранилищу (таблица 15.2).

Таблица 15.2 – Основные методы для работы с файлами и каталогами в Context

| **Метод** | **Назначение** |
|---|---|
| File getExternalCacheDir() | Возвращает дескриптор папки кэша в первичной внешней области. |
| File[] getExternalCacheDirs() | Возвращает папки кэша для нескольких разделов внешнего хранилища. |
| File getExternalFilesDir(String) | Возвращает дескриптор папки в первичном внешнем хранилище, предназначенной для хранения обычных файлов. При передаче типа String можно обратиться к папке, предназначенной для конкретного типа содержимого. Константы типов определяются в Environment с префиксом DIRECTORY_. Например, изображения хранятся в Environment.DIRECTORY_PICTURES. |
| File[] getExternalFilesDirs(String) | То же, что `getExternalFilesDir(String)`, но возвращает все возможные папки для заданного типа. |

| Метод | Назначение |
|---|---|
| File[] getExternalMediaDirs() | Возвращает дескрипторы для всех внешних папок, предоставляемых Android для хранения изображений, видео и музыки. От вызова `getExternalFilesDir(Environment.DIRECTORY_PICTURES)` этот метод отличается тем, что папка автоматически обрабатывается медиасканером, соответственно файлы становятся доступными для приложений, которые воспроизводят музыку, отображают графику или видеоролики. Соответственно все, что размещается в папке, возвращаемой `getExternalMediaDirs()`, автоматически появляется в этих приложениях. |

Эти методы предоставляют простые средства для обращения к первичной области, а также *относительно* простые средства для обращения ко всему остальному. Все эти методы сохраняют файлы в общедоступных местах, так что будьте осторожны.

*Выбор места для хранения фотографии*

Пора выделить фотографиям место, где они будут существовать. Сначала добавить в Book метод для получения имени файла.

Листинг 15.1 – Добавление свойства для получения имени файла (Book.java)

```
...
    public String getPhotoFilename() {
        return "IMG_" + getId().toString() + ".jpg";
    }
}
```

Метод `Book.getPhotoFilename()` не знает, в какой папке будет храниться фотография. Однако имя файла будет уникальным, поскольку оно строится на основании идентификатора Book.

Затем следует найти место, в котором будут располагаться фотографии. Класс BookLab отвечает за все, что относится к долгосрочному хранению данных в BookDepository, поэтому он становится наиболее естественным кандидатом. Добавить в BookLab метод `getPhotoFile(Book)`, который будет возвращать эту информацию.

Листинг 15.2 – Определение местонахождения файла фотографии (BookLab.java)

```
public class BookLab {
    ...
    public Book getBook(UUID id) {
        ...
    }
    public File getPhotoFile(Book book) {
        File externalFilesDir = mContext
            .getExternalFilesDir(Environment.DIRECTORY_PICTURES);
        if (externalFilesDir == null) {
            return null;
        }
        return new File(externalFilesDir, book.getPhotoFilename());
    }
...
```

Этот код не создает никакие файлы в файловой системе. Он только возвращает объекты File, представляющие нужные места. При этом он проверяет наличие внешнего хранилища для сохранения данных. Если внешнее хранилище недоступно, `getExternalFilesDir(String)` возвращает null — как и весь метод.

***Использование интента камеры***

Следующий шаг — непосредственное создание снимка. Здесь все просто: необходимо снова воспользоваться неявным интентом.

Надо начать с сохранения местонахождения файла фотографии. (Эта информация будет использоваться еще в нескольких местах, поэтому сохранение избавит от лишней работы.)

Листинг 15.3 – Сохранение местонахождения файла фотографии (BookFragment.java)

```
...
private Book mBook;
private File mPhotoFile;
private EditText mTitleField;
...
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    UUID bookId = (UUID) getArguments().getSerializable(ARG_BOOK_ID);
    mBook = BookLab.get(getActivity()).getBook(bookId);
    mPhotoFile = BookLab.get(getActivity()).getPhotoFile(mBook);
}
...
```

На следующем шаге будет подключена кнопка, которая непосредственно создает снимок.

*Отправка интента*

Нужное действие `ACTION_CAPTURE_IMAGE` определяется в классе `MediaStore`. Этот класс определяет открытые интерфейсы, используемые в Android при работе с основными аудиовизуальными материалами — изображениями, видео и музыкой. К этой категории относится и интент, запускающий камеру.

По умолчанию `ACTION_CAPTURE_IMAGE` послушно запускает приложение камеры и делает снимок, но результат не является фотографией в полном разрешении. Вместо нее создается миниатюра с малым разрешением, которая упаковывается в объект Intent, возвращаемый в `onActivityResult(…)`.

Чтобы получить выходное изображение в высоком разрешении, необходимо сообщить, где должно храниться изображение в файловой системе. Эта задача решается передачей URI для места, в котором должен сохраняться файл, в `MediaStore.EXTRA_OUTPUT`.

Если целевая версия платформы 24 и выше, то для того чтобы предоставить доступ к определенному файлу или папке и сделать их доступными для других приложений, необходимо использовать класс `FileProvider`. Прежде всего необходимо объявить `FileProvider` путем включения тега `<provider>` в файл AndroidManifest.xml, указав уникальный параметр атрибута `android:authority`, чтобы избежать конфликтов, например, можно использовать `${applicationId}.provider` и другие широко используемые подходы.

Листинг 15.4 – Объявление FileProvider в манифесте (AndroidManifest.xml)

```
<application
...
    <provider
        android:name="android.support.v4.content.FileProvider"
        android:authorities="${applicationId}.provider"
        android:exported="false"
        android:grantUriPermissions="true">
        <meta-data
            android:name="android.support.FILE_PROVIDER_PATHS"
            android:resource="@xml/provider_paths"/>
    </provider>
</application>
```

Затем создать файл provider_paths.xml в папке res/xml. Возможно предварительно потребуется создать папку, если она не существует. Содержимое файла показано в листинге 15.5 – В нем формируется предоставление доступа к внешнему хранилищу в корневой папке (path = ".") с именем external_files.

Листинг 15.5 – Предоставление доступа к внешнему хранилищу (provider_paths.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<paths xmlns:android="http://schemas.android.com/apk/res/android">
    <external-path name="external_files" path="."/>
</paths>
```

На последнем эпапе следует определить используемую версию платформы и исползовать соответствующий подход для формирования URI, как показано ниже в листинге 15.6.

Листинг 15.6 – Формирование URI и отправка интента камеры (BookFragment.java)

```
...
private ImageButton mPhotoButton;
private ImageView mPhotoView;

private static final int REQUEST_DATE = 0;
private static final int REQUEST_PHOTO = 1;
...
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                Bundle savedInstanceState) {
    ...
    mPhotoButton = (ImageButton) v.findViewById(R.id.book_camera);
    final Intent captureImage =
                new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    PackageManager packageManager = getActivity().getPackageManager();
    boolean canTakePhoto = mPhotoFile != null &&
            captureImage.resolveActivity(packageManager) != null;
    if (canTakePhoto) {
        Uri uri;
        if (Build.VERSION.SDK_INT<24)
            uri = Uri.fromFile(mPhotoFile);
        else
            uri = FileProvider.getUriForFile(getActivity(),
                    BuildConfig.APPLICATION_ID + ".provider", mPhotoFile);
        captureImage.putExtra(MediaStore.EXTRA_OUTPUT, uri);
        mPhotoButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
```

```
            startActivityForResult(captureImage, REQUEST_PHOTO);
        }
    });
    mPhotoView = (ImageView) v.findViewById(R.id.book_photo);
    return v;
}
```

Приведенный выше код такде формирует неявный интент для сохранения фотографии в месте, определяемом `mPhotoFile`. Добавляет код, который блокирует кнопку при отсутствии приложения камеры или недоступности места для сохранения фотографии. Также направляется запрос к `PackageManager` активности, реагирующие на неявный интент камеры, с целью проверки доступности приложения камеры.

Запустить BookDepository и нажать кнопку запуска приложения камеры (рисунок 15.5).

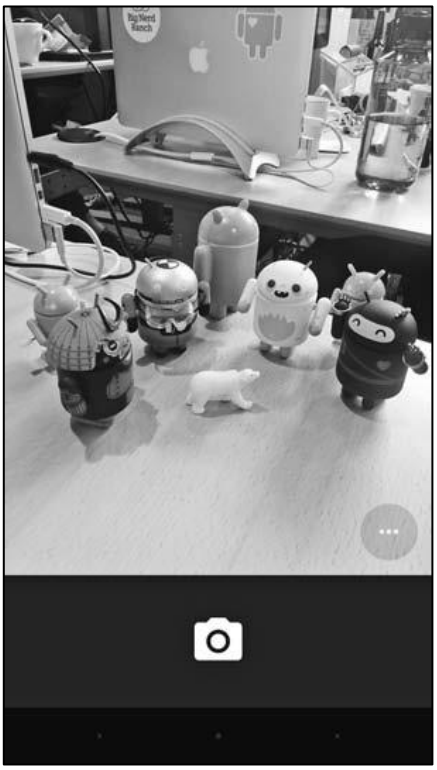

Рисунок 15.5 – Приложение BookDepository, работающее с камерой

***Масштабирование и отображение растровых изображений***

Приложение успешно делает снимки, которые сохраняются в файловой системе для дальнейшего использования. Следующий шаг — поиск файла с изображением, его загрузка и отображение для пользователя. Для этого необходимо загрузить данные изображения в объект Bitmap достаточного размера. Чтобы построить объект Bitmap на базе файла, достаточно воспользоваться классом BitmapFactory:

```
Bitmap bitmap = BitmapFactory.decodeFile(mPhotoFile.getPath());
```

Однако Bitmap — простой объект для хранения необработанных данных пикселов. Таким образом, даже если исходный файл был сжат, в объекте Bitmap никакого сжатия не будет. Поэтому 24-битовое изображение с камеры на 16 мегапикселов, которое может занимать всего 5 Мбайт в формате JPG, в объекте Bitmap разрастается до 48(!) Мбайт.

Найти обходное решение возможно, но это означает, что изображение придется масштабировать вручную. Для этого можно сначала просканировать файл и определить его размер, затем вычислить, насколько его нужно масштабировать, для того чтобы он поместился в заданную область, и, наконец, заново прочитать файл для создания уменьшенного объекта Bitmap.

Создать для этого метода новый класс с именем PictureUtils.java и добавить в него статический метод с именем `getScaledBitmap(String, int, int)`.

Листинг 15.7 – Создание метода getScaledBitmap(…) (PictureUtils.java)

```
public class PictureUtils {
public static Bitmap getScaledBitmap(String path, int destWidth,
                int destHeight) {
   // Чтение размеров изображения на диске
   BitmapFactory.Options options = new BitmapFactory.Options();
   options.inJustDecodeBounds = true;
   BitmapFactory.decodeFile(path, options);
   float srcWidth = options.outWidth;
   float srcHeight = options.outHeight;

   // Вычисление степени масштабирования
   int inSampleSize = 1;
   if (srcHeight > destHeight || srcWidth > destWidth) {
      if (srcWidth > srcHeight) {
         inSampleSize = Math.round(srcHeight / destHeight);
      } else {
         inSampleSize = Math.round(srcWidth / destWidth);
      }
   }
   options = new BitmapFactory.Options();
   options.inSampleSize = inSampleSize;

   // Чтение данных и создание итогового изображения
   return BitmapFactory.decodeFile(path, options);
   }
}
```

Ключевой параметр `inSampleSize` определяет величину «образца» для каждого пиксела исходного изображения: образец с размером 1 содержит

один горизонтальный пиксел для каждого горизонтального пиксела исходного файла, а образец с размером 2 содержит один горизонтальный пиксел для каждых двух горизонтальных пикселов исходного файла. Таким образом, если значение `inSampleSize` равно 2, количество пикселов в изображении составляет четверть от количества пикселов оригинала.

Также при запуске фрагмента неизвестна величина `PhotoView`. До обработки макета никаких экранных размеров не существует. Первый проход этой обработки происходит после выполнения `onCreate(…)`, `onStart()` и `onResume()`, поэтому `PhotoView` и не знает своих размеров.

Для данной задачи есть два решения: либо подождать, пока будет сделан первый проход, либо воспользоваться консервативной оценкой. Второй способ менее эффективен, но более прямолинеен.

Написать еще один статический метод с именем `getScaledBitmap(String, Activity)` для масштабирования Bitmap под размер конкретной активности.

Листинг 15.8 – Метод масштабирования с консервативной оценкой (PictureUtils.java)

```
public class PictureUtils {
    public static Bitmap getScaledBitmap(String path, Activity activity)
    {
        Point size = new Point();
        activity.getWindowManager().getDefaultDisplay()
             .getSize(size);
        return getScaledBitmap(path, size.x, size.y);
    }
    ...
}
```

Метод проверяет размер экрана и уменьшает изображение до этого размера. Виджет `ImageView`, в который загружается изображение, всегда меньше размера экрана, так что эта оценка весьма консервативна.

Чтобы загрузить объект Bitmap в `ImageView`, добавить в BookFragment метод для обновления `mPhotoView`.

Листинг 15.9 – Обновление mPhotoView (BookFragment.java)

```
...
    private String getBookReport() {
        ...
    }
    private void updatePhotoView() {
        if (mPhotoFile == null || !mPhotoFile.exists()) {
            mPhotoView.setImageDrawable(null);
        } else {
```

```
            Bitmap bitmap = PictureUtils.getScaledBitmap(
            mPhotoFile.getPath(), getActivity());
            mPhotoView.setImageBitmap(bitmap);
        }
    }
}
```

Затем вызвать этот метод из `onCreateView(…)` и `onActivityResult(…)`.

Листинг 15.10 – Вызов updatePhotoView() (BookFragment.java)

```
mPhotoButton.setOnClickListener(new View.OnClickListener() {
    ...
});
mPhotoView = (ImageView) v.findViewById(R.id.book_photo);
updatePhotoView();
return v;
}

@Override
public void onActivityResult(int requestCode, int resultCode,
     Intent data) {
    if (resultCode != Activity.RESULT_OK) {
        return;
    }
    if (requestCode == REQUEST_DATE) {
        ...
    } else if (requestCode == REQUEST_PHOTO) {
        updatePhotoView();
    }
}
```

Запустить приложение снова. Изображение выводится в уменьшенном виде.

*Объявление функциональности*

Остается решить еще одну задачу: сообщить об использовании камеры потенциальным пользователям. Когда приложение использует некоторое оборудование (например, камеру или NFC) или любой другой аспект, который может отличаться от устройства к устройству, настоятельно рекомендуется сообщить о нем Android. Это позволит другим приложениям (например, магазину Google Play) заблокировать установку приложения, если в нем используются возможности, не поддерживаемые данным устройством.

Чтобы объявить, что в приложении используется камера, включите тег `<uses-feature>` в AndroidManifest.xml.

Листинг 15.11 – Добавление тега uses-feature (AndroidManifest.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
...
   <uses-feature android:name="android.hardware.camera"
       android:required="false"
   />
...
```

В этом примере в тег добавляется необязательный атрибут `android:required`. По умолчанию объявление об использовании некоторой возможности означает, что без нее приложение может работать некорректно. К BookDepository это не относится. Использованый метод `resolveActivity(…)` проверит наличие приложения камеры, после чего корректно заблокирует кнопку, если приложение не найдено.

Атрибут `android:required="false"` корректно обрабатывает эту ситуацию. При этом Android сообщается, что приложение может нормально работать без камеры, но некоторые части приложения окажутся недоступными.

**Самостоятельные задания.**

*Задание 1. Вывод увеличенного изображения*

Пользователь видит уменьшенное изображение, но вряд ли ему удастся рассмотреть его во всех подробностях. Создайте новый фрагмент `DialogFragment`, в котором отображается увеличенная версия фотографии обложки книги. Когда пользователь нажимает в какой-то точке миниатюры, на экране должен появиться DialogFragment с увеличенным изображением.

*Задание 2. Эффективная загрузка миниатюры*

В этом разделе был использован довольно грубый подход для оценки размера изображение, до которого оно должно быть уменьшено. Такое решение не идеально, но оно работает и быстро реализуется. В существующих API можно использовать `ViewTreeObserver` — объект, который можно получить от любого представления в иерархии Activity:

```
ViewTreeObserver observer = mImageView.getViewTreeObserver();
```

Для `ViewTreeObserver` можно зарегистрировать разнообразных слушателей, включая `OnGlobalLayoutListener`. Этот слушатель инициирует событие каждый раз, когда происходит проход обработки макета.

Изменить свой код так, чтобы он использовал размеры `mPhotoView`, когда они действительны, и ожидал прохода обработки макета перед первым вызовом `updatePhotoView()`.

*Задание 3. Добавление изображения из Галереи*

Пользователи чаше всего читают электронные книги и сфотографировать обложку не представляется возможным. Создать новую кнопку, которая будет создавать неявный интент для открытия приложения Галерея и загрузки выбранной фотографии (рисунок 15.6).

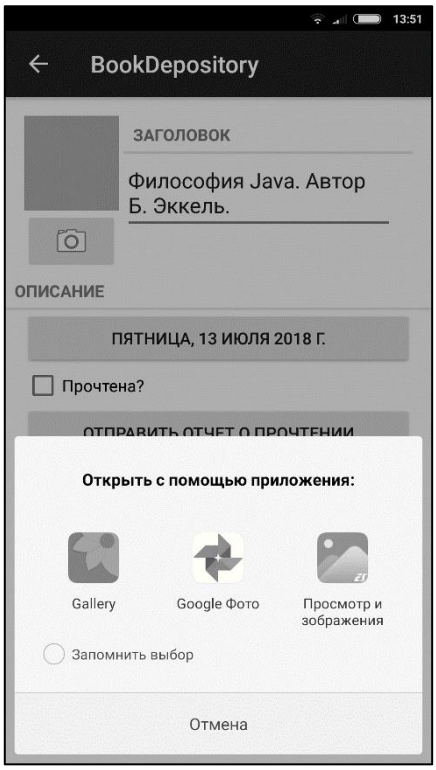


Рисунок 15.6 – Приложения, готовые открыть изображения для загрузки обложки