

```

        out.close()
        return file;
    } catch (IOException ex) {
        Log.e("Unable to write", ex);
        return null;
    }
}

@Override
protected void onPostExecute(File result) {
    // Это вызывается, когда мы завершаем работу
}

@Override
protected void onPreExecute() {
    // Вызывается перед началом работы
}

@Override
protected void onProgressUpdate(Void... values) {
    // Вряд ли это необходимо для данного примера
}
}
}

```

...а затем:

```
new FileOperation().execute("Некоторые параметры");
```

Пример создания и вызова AsyncTask см. здесь: <http://stackoverflow.com/questions/9671546/asynctask-android-example>. Также см. вопрос о том, как обрабатывать IOExceptions и другие ошибки: <http://stackoverflow.com/questions/3690980/asynctask-error-handling>.

## Глава 49. FileProvider

### 49.1. Общий доступ к файлу

В этом примере вы узнаете, как предоставить общий доступ к файлу другим приложениям. Мы будем использовать pdf-файл, хотя код работает и с любым другим форматом.

**Дорожная карта:**

*Укажите каталоги, где размещены файлы, к которым вы хотите предоставить общий доступ.*

Мы будем использовать FileProvider – класс, обеспечивающий безопасный обмен файлами между приложениями. Этот класс может предоставлять общий доступ к файлам только в заранее определенных каталогах, поэтому определим их.

1. Создайте новый XML-файл, который будет содержать пути, например res/xml/filepaths.xml.
2. Добавьте пути:

```
<paths xmlns:android="http://schemas.android.com/apk/res/android">
    <files-path name="pdf_folder" path="documents/" />
</paths>
```

Определите FileProvider и свяжите его с путями к файлам.

Это делается в манифесте:

```
<manifest>
    ...
    <application>
        ...
        <provider
            android:name="android.support.v4.context.FileProvider"
            android:authorities="com.mydomain.fileprovider"
            android:exported="false"
            android:grantUriPermissions="true">
            <meta-data
                android:name="android.support.FILE_PROVIDER_PATHS"
                android:resource="@xml/filepaths" />
        </provider>
        ...
    </application>
    ...
</manifest>
```

*Сгенерируйте URI для файла.*

Для совместного использования файла мы должны предоставить его идентификатор. Для этого используется URI (Uniform Resource Identifier).

```
// Предполагается, что загружаемый файл находится в подкаталоге documents/
// внутреннего хранилища
File documentsPath = new File(Context.getFilesDir(), "documents");
File file = new File(documentsPath, "sample.pdf");
// Разумеется, это можно сделать и в одной строке:
// File file = new File(Context.getFilesDir(), "documents/sample.pdf");

Uri uri = FileProvider.getUriForFile(getContext(), "com.mydomain.fileprovider", file);
```

Как видно из кода, сначала мы создаем новый класс File, предоставляющий файл. Для получения URI мы просим FileProvider получить его. Второй аргумент очень важен: он передает полномочия FileProvider. Он должен быть равен полномочиям FileProvider, определенным в манифесте.

*Совместное использование файла с другими приложениями.*

Для обмена файлом с другими приложениями используем ShareCompat:

```
Intent intent = ShareCompat.IntentBuilder.from(getContext())
    .setType("application/pdf")
    .setStream(uri)
    .setChooserTitle("Строка выбора")
    .createChooserIntent()
    .addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);

Context.startActivity(intent);
```

Chooser – это меню, из которого пользователь может выбрать, с каким приложением он хочет поделиться файлом. Флаг Intent.FLAG\_GRANT\_READ\_URI\_PERMISSION необходим для предоставления временного разрешения на чтение URI.

# Глава 50. Хранение файлов во внутреннем и внешнем хранилищах

Параметр	Подробности
name	Имя открываемого файла. Примечание: не может содержать разделителей путей
mode	Режим работы. Для работы по умолчанию используйте MODE_PRIVATE, а для добавления существующего файла – MODE_APPEND. Режимы MODE_WORLD_READABLE и MODE_WORLD_WRITEABLE были упразднены.
dir	Каталог файла, в котором будет создан новый файл
path	Путь, указывающий местоположение нового файла
type	Тип каталога файлов для извлечения. Может быть null или любой из следующих: DIRECTORY_MUSIC, DIRECTORY_PODCASTS, DIRECTORY_RINGTONES, DIRECTORY_ALARMS, DIRECTORY_NOTIFICATIONS, DIRECTORY_PICTURES или DIRECTORY_MOVIES

## 50.1. Android: внутренние и внешние хранилища – уточнение терминологии

Разработчики Android (в основном начинающие) часто путаются в терминологии, касающейся внутренних (Internal storage) и внешних хранилищ (External storage). На Stackoverflow можно найти множество вопросов по этому поводу. В основном это связано с тем, что терминология в Google и официальной документации по Android сильно отличается от терминологии обычного пользователя Android OS. Поэтому очевидно, что документирование этого поможет.

### Что мы думаем, или терминология пользователя (User's Terminology, UT)

#### Внутреннее хранилище (UT)

встроенная внутренняя память телефона

Пример: Объем встроенной памяти Nexus 6P составляет 32 ГБ.

#### Внешнее хранилище (UT)

съемная карта Secure Digital (SD) или накопитель micro SD

Пример: объем памяти на съемных картах SD таких производителей, как Samsung, Sandisk, Strontium, Transcend и др.

Но, согласно документации/руководству по Android – терминологии Google (Google's Terminology, GT)

#### Внутреннее хранилище (GT):

По умолчанию файлы, сохраненные во внутреннем хранилище, являются частными для вашего приложения, и другие приложения не могут получить к ним доступ (как и пользователь).

#### Внешнее хранилище (GT):

Это может быть съемный носитель (например, SD-карта) или внутренний (несъемный) накопитель.

#### Внешние хранилища (GT) можно разделить на два типа:

##### Первичное внешнее хранилище

Это то же самое, что и встроенная внутренняя память телефона (или) внутренний накопитель (терминология UT)

Пример: Объем встроенной памяти Nexus 6P составляет 32 ГБ.

##### Вторичное внешнее хранилище или съемный накопитель (GT)

Это то же самое, что и съемная карта памяти micro SD (или) внешний накопитель (терминология UT)

Пример: объем памяти на съемных картах SD таких производителей, как Samsung, Sandisk, Strontium, Transcend и др.

### Первичное внешнее хранилище

Доступ к этому типу накопителя можно получить на компьютере с ОС Windows, подключив телефон к ПК через USB-кабель и выбрав пункт *Камера (PTP)* в уведомлении о параметрах USB.

В двух словах:

**Внешнее хранилище (GT) = Внутреннее хранилище (UT) и внешнее хранилище (UT)**

**Съемный накопитель (GT) = Внешнее хранилище (UT)**

**Внутреннее хранилище (GT) не имеет термина в UT.**

Позвольте мне пояснить:

**Внутреннее хранилище (GT):** по умолчанию файлы, сохраненные во внутреннем хранилище, являются частными для вашего приложения, и другие приложения не могут получить к ним доступ. Пользователь приложения также не может получить к ним доступ с помощью файлового менеджера, даже после включения опции «показывать скрытые файлы» в файловом менеджере. Чтобы получить доступ к файлам во внутреннем хранилище (GT), необходимо выполнить рутировка телефона Android. Кроме того, при удалении приложения эти файлы удаляются.

Таким образом, внутренняя память, она же внутреннее хранилище (GT) – это **НЕ то**, что мы думаем, читая о характеристиках телефонов.

Как правило, внутренний накопитель (хранилище) (GT) располагается:

```
/data/data/your.application.package.appname/someDirectory/
```

### Внешнее хранилище (GT):

Каждое Android-совместимое устройство поддерживает общее «внешнее хранилище», которое можно использовать для сохранения файлов. Файлы, сохраненные во внешнем хранилище, доступны для чтения во всем мире и могут быть изменены пользователем при включении функции USB mass storage для передачи файлов на компьютер.

**Расположение внешнего хранилища (GT):** Это может быть *любое место* во внутренней памяти (UT) или на съемном накопителе (GT), то есть карте памяти micro SD. Это зависит от OEM-производителя телефона, а также от версии ОС Android.

Для чтения или записи файлов во внешнем хранилище (GT) ваше приложение должно получить системные разрешения `READ_EXTERNAL_STORAGE` или `WRITE_EXTERNAL_STORAGE`.

Например:

```
<manifest ...>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    ...
</manifest>
```

Если требуется и чтение, и запись файлов, то следует запросить только разрешение `WRITE_EXTERNAL_STORAGE`, поскольку оно неявно требует также и доступа на чтение.

Во внешнем хранилище (GT) можно также сохранять файлы, которые являются частными для приложений (app-private). Но:

Когда пользователь удаляет ваше приложение, этот каталог и всё его содержимое удаляются.

Когда нужно сохранять файлы, частные для приложений, во внешнем хранилище (GT)?

### Вторичное внешнее хранилище или съемный накопитель (GT)

Доступ к этому типу хранилища можно получить на ПК с ОС Windows, подключив телефон через USB-кабель и выбрав в уведомлении о параметрах USB пункт *Передача файлов*.

Если вы работаете с файлами, которые не предназначены для использования другими приложениями (например, графические текстуры или звуковые эффекты, используемые только вашим приложением), следует использовать частный каталог хранения на внешнем хранилище.

### **Методы хранения во внутреннем хранилище (GT):**

Оба эти метода присутствуют в классе Context:

```
File getDir (String name, int mode)
```

```
File getFilesDir ()
```

### **Методы хранения в первичном внешнем хранилище, то есть во внутреннем хранилище (UT):**

```
File getExternalStorageDirectory ()
```

```
File getExternalFilesDir (String type)
```

```
File getExternalStoragePublicDirectory (String type)
```

Вначале все использовали `Environment.getExternalStorageDirectory()`, которая указывала на корень первичного внешнего хранилища. В результате первичное внешнее хранилище (Primary External Storage) заполнялось случайным содержимым.

Позже были добавлены следующие два метода.

1. В класс Context добавлена функция `getExternalFilesDir()`, указывающая на **каталог приложения** на первичном внешнем хранилище. Этот каталог и его содержимое будут удалены при деинсталляции приложения.

2. `Environment.getExternalStoragePublicDirectory()` для централизованного хранения файлов известных типов, таких как фотографии и фильмы. Этот каталог и его содержимое **НЕ** будут удалены при деинсталляции приложения.

### **Методы хранения на съемных носителях (GT), то есть на картах micro SD**

До появления API уровня 19 официального способа хранения на SD-карте не существовало. Это делалось с помощью неофициальных библиотек или API. Официально один метод появился в классе Context на 19-м уровне API.

```
File[] getExternalFilesDirs (String type)
```

Он возвращает абсолютные пути к специфическим для приложения каталогам на всех общих/внешних устройствах хранения, где приложение может размещать принадлежащие ему постоянные файлы. Эти файлы являются внутренними для приложения и обычно не видны пользователю.

Это означает, что он вернет пути к **обоим** типам внешних накопителей (GT) – внутренней памяти и карте micro SD. Как правило, **вторым** будет путь к памяти карты micro SD (но не всегда). Поэтому необходимо проверить это, выполнив код с помощью данного метода.

Пример с фрагментом кода (был создан новый проект Android с пустой активностью, внутри следующий код):

Метод `protected void onCreate(Bundle savedInstanceState)` файла `MainActivity.java`:

```
File internal_m1 = getDir("custom", 0);
File internal_m2 = getFilesDir();

File external_m1 = Environment.getExternalStorageDirectory();

File external_m2 = getExternalFilesDir(null);
File external_m2_Args = getExternalFilesDir(Environment.DIRECTORY_PICTURES);
```

```

File external_m3 = Environment.getExternalStoragePublicDirectory(Environment.
DIRECTORY_PICTURES);

File[] external_AND_removable_storage_m1 = getExternalFilesDirs(null);
File[] external_AND_removable_storage_m1_Args =
getExternalFilesDirs(Environment.DIRECTORY_PICTURES);

```

Результат выполнения приведенного выше кода:

```

internal_m1: /data/data/your.application.package.appname/app_custom
internal_m2: /data/data/your.application.package.appname/files
external_m1: /storage/emulated/0
external_m2: /storage/emulated/0/Android/data/your.application.package.appname/files
external_m2_Args: /storage/emulated/0/Android/data/your.application.package.
appname/files/Pictures
external_m3: /storage/emulated/0/Pictures
external_AND_removable_storage_m1 (first path):
/storage/emulated/0/Android/data/your.application.package.appname/files
external_AND_removable_storage_m1 (second path):
/storage/sdcard1/Android/data/your.application.package.appname/files
external_AND_removable_storage_m1_Args (first path):
/storage/emulated/0/Android/data/your.application.package.appname/files/Pictures
external_AND_removable_storage_m1_Args (second path):
/storage/sdcard1/Android/data/your.application.package.appname/files/Pictures

```

**Примечание:** телефон был подключен к ПК с ОС Windows, включены обе опции разработчика, отладка по USB, а затем запущен данный код. Если не подключать телефон, а запустить код в эмуляторе Android, результат может отличаться.

Данные в телефоне могут храниться в следующих местах:

Накопитель micro SD: /storage/sdcard1

Внутреннее хранилище (UT): /storage/sdcard0

Обратите внимание, что /sdcard и /storage/emulated/0 также указывают на внутреннее хранилище (UT). Но это симлинки на /storage/sdcard0.

Подробнее о различных путях хранения в Android см.: <http://stackoverflow.com/a/38813578/2818583>.

Ваши файлы могут храниться не в тех же путях хранения, поскольку расположение/пути хранения могут отличаться на других мобильных телефонах в зависимости от производителя, изготовителя и различных версий ОС Android.

## 50.2. Использование внешнего хранилища

Внешнее хранилище (External Storage) – тип хранилища, который мы можем использовать для сохранения файлов на устройстве пользователя. Он имеет ряд ключевых отличий от “внутреннего” хранилища, а именно:

- Оно не всегда доступно. В случае съемного носителя (SD-карта) пользователь может просто извлечь его.
- Оно не является частным. Пользователь (и другие приложения) имеют доступ к этим файлам.

- Если пользователь удалит приложение, то файлы, сохраненные в каталоге, полученный с помощью функции `getExternalFilesDir()`, будут удалены.

Чтобы использовать внешнее хранилище, необходимо сначала получить соответствующие разрешения. Для этого необходимо использовать:

- `android.permission.WRITE_EXTERNAL_STORAGE` для чтения и записи
- `android.permission.READ_EXTERNAL_STORAGE` только для чтения

Чтобы предоставить эти права, необходимо определить их в файле `AndroidManifest.xml` следующим образом:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

**Примечание:** поскольку эти разрешения относятся к категории опасных (см. <https://developer.android.com/guide/topics/security/permissions.html#normal-dangerous>), если вы используете API уровня 23 или выше, вам придется запрашивать их во время выполнения программы.

Перед попыткой записи или чтения с внешнего хранилища всегда следует убедиться в доступности носителя:

```
String state = Environment.getExternalStorageState();
if (state.equals(Environment.MEDIA_MOUNTED)) {
    // Доступно для чтения и записи
}
if (state.equals(Environment.MEDIA_MOUNTED) ||
    state.equals(Environment.MEDIA_MOUNTED_READ_ONLY)) {
    // Доступно, по крайней мере, для чтения
}
```

При записи файлов во внешнее хранилище необходимо решить, каким должен быть файл – общедоступным (Public) или частным (Private).

Хотя оба этих типа файлов по-прежнему доступны пользователю и другим приложениям на устройстве, между ними есть ключевое различие.

Общедоступные файлы должны оставаться на устройстве, когда пользователь удаляет приложение. Примером файла, который должен быть сохранен как общедоступный, могут быть фотографии, сделанные с помощью приложения.

Частные файлы должны удаляться при удалении приложения. Эти типы файлов специфичны для приложения и не могут быть полезны пользователю или другим приложениям. Например, временные файлы, загружаемые/используемые вашим приложением.

Вот как получить доступ к каталогу `Documents` как хранилищу обоих типов файлов:

#### Общедоступные (Public):

```
// Получить доступ к каталогу приложения в каталоге Public documents устройства
File docs = new File(Environment.getExternalStoragePublicDirectory(
    Environment.DIRECTORY_DOCUMENTS), "YourAppDirectory");
// Создать каталог, если он еще не существует
myDocs.mkdirs();
```

#### Частные (Private):

```
// Доступ к каталогу личных документов вашего приложения
File file = new File(context.getExternalFilesDir(Environment.DIRECTORY_DOCUMENTS),
    "YourAppDirectory");
// Создать каталог, если он еще не существует
myDocs.mkdirs();
```

### 50.3. Использование внутреннего хранилища

По умолчанию все файлы, сохраненные во внутреннем хранилище, являются частными для вашего приложения. Они не могут быть доступны ни другим приложениям, ни пользователю в обычных условиях. Эти файлы удаляются, когда пользователь удаляет приложение.

#### Запись текста в файл:

```
String fileName="helloworld";
String textToWrite = "Hello, World!";
FileOutputStream fileOutputStream;

try {
    fileOutputStream = openFileOutput(fileName, Context.MODE_PRIVATE);
    fileOutputStream.write(textToWrite.getBytes());
    fileOutputStream.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

#### Добавление текста в существующий файл

Используйте `Context.MODE_APPEND` для параметра mode команды `openFileOutput`:

```
fileOutputStream = openFileOutput(fileName, Context.MODE_APPEND);
```

### 50.4. Получение каталога устройств

Сначала добавьте разрешение на чтение/получение каталога устройства:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

#### Создание класса модели:

```
//создайте один класс модели каталога
//для хранения названия и типа каталога в списке

public class DirectoryModel {
    String dirName;
    int dirType;// задается 1 или 0, где 0 - каталог, а 1 - файл.

    public int getDirType() {
        return dirType;
    }

    public void setDirType(int dirType) {
        this.dirType = dirType;
    }

    public String getDirName() {
        return dirName;
    }

    public void setDirName(String dirName) {
        this.dirName = dirName;
    }
}
```

Создание списка с использованием модели каталога для добавления данных каталога:

```
//определить список для отображения каталога
List<DirectoryModel> rootDir = new ArrayList<>();
```

Получение каталога:

```
//для получения каталога устройств
private void getDirectory(String currDir) { // передаем корневой каталог устройства
    File f = new File(currDir);
    File[] files = f.listFiles();
    if (files != null) {
        if (files.length > 0) {
            rootDir.clear();
            for (File inFile : files) {
                if (inFile.isDirectory()){//возвращается true, если это каталог
                    // является каталогом
                    DirectoryModel dir = new DirectoryModel();
                    dir.setDirName(inFile.toString().replace("/storage/emulated/0", ""));
                    dir.setDirType(0); // устанавливаем 0 для каталога
                    rootDir.add(dir);
                } else if (inFile.isFile()){// возвращаем true, если это файл
                    // является файлом
                    DirectoryModel dir = new DirectoryModel();
                    dir.setDirName(inFile.toString().replace("/storage/emulated/0", ""));
                    dir.setDirType(1); // устанавливаем 1 для файла
                    rootDir.add(dir);
                }
            }
        }
    }
    printDirectoryList();
}
}
```

Вывод списка каталогов в журнал:

```
private void printDirectoryList() {
    for (int i = 0; i < rootDir.size(); i++) {
        Log.e(TAG, "printDirectoryLogs: " + rootDir.get(i).toString());
    }
}
```

Использование:

```
//для получения каталога – вызов функции с корневым каталогом
String rootPath = Environment.getExternalStorageDirectory().toString(); // return
==> /storage/emulated/0/
getDirectory(rootPath );
```

Для получения внутренних файлов/папок из конкретного каталога используйте тот же метод, только измените аргумент, передайте в аргументе текущий выбранный путь и обработайте ответ.

Чтобы получить расширение файла:

```
private String getExtension(String filename) {
```

```

String filenameArray[] = filename.split("\\\\.");
String extension = filenameArray[filenameArray.length - 1];
Log.d(TAG, "getExtension: " + extension);

return extension;
}

```

## 50.5. Сохранение базы данных на SD-карте (резервное копирование базы данных)

```

public static Boolean ExportDB(String DATABASE_NAME, String packageName,
String folderName){
//DATABASE_NAME (имя базы данных), включая ".db" в конце, например, "myApp.db"
String DBName = DATABASE_NAME.substring(0, DATABASE_NAME.length() - 3);
File data = Environment.getDataDirectory();
FileChannel source=null;
FileChannel destination=null;
String currentDBPath = "/data/" + packageName + "/databases/" + DATABASE_NAME;
// получение пути приложения БД

File sd = Environment.getExternalStorageDirectory(); // получение пути
к SD-карте телефона
String backupPath = sd.getAbsolutePath() + folderName; // если необходимо
установить резервное копирование в конкретное имя папки
/* Будьте внимательны, имя папки должно начинаться так: "/myFolder",
не забудьте "/" в начале имени папки
вы можете определить имя папки следующим образом : "/myOuterFolder/
MyInnerFolder" и так далее ...
*/
File dir = new File(backupPath);
if(!dir.exists())// если по данному пути не было папки, то она создается.
{
    dir.mkdirs();
}

DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd_HH-mm-ss");
Date date = new Date();
/* использовать дату, включающую имя файла, для их упорядочивания
и предотвращения создания файла с тем же именем*.
File currentDB = new File(data, currentDBPath);
File backupDB = new File(backupPath, DBName + "(" + dateFormat.format(date) + ").db");
try {
    if (currentDB.exists() && !backupDB.exists()) {
        source = new FileInputStream(currentDB).getChannel();
        destination = new FileOutputStream(backupDB).getChannel();
        destination.transferFrom(source, 0, source.size());
        source.close();
        destination.close();
        return true;
    }
    return false;
} catch(IOException e) {
    e.printStackTrace();
    return false;
}
}

```

Вызовите этот метод таким образом:

```
ExportDB("myDB.db", "com.example.exam", "/myFolder");
```

# Глава 51. Zip-архивация в Android

```

import android.util.Log;
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.zip.ZipEntry;
import java.util.zip.ZipOutputStream;

public class Compress {
    private static final int BUFFER = 2048;

    private String[] _files;
    private String _zipFile;

    public Compress(String[] files, String zipFile) {
        _files = files;
        _zipFile = zipFile;
    }

    public void zip() {
        try {
            BufferedInputStream origin = null;
            FileOutputStream dest = new FileOutputStream(_zipFile);

            ZipOutputStream out = new ZipOutputStream(new
                BufferedOutputStream(dest));

            byte data[] = new byte[BUFFER];

            for(int i=0; i < _files.length; i++) {
                Log.v("Сжатие", "Добавление: " + _files[i]);
                FileInputStream fi = new FileInputStream(_files[i]);
                origin = new BufferedInputStream(fi, BUFFER);
                ZipEntry entry = new ZipEntry(_files[i].substring(_files[i].
                    lastIndexOf("/") + 1));
                out.putNextEntry(entry);
                int count;
                while ((count = origin.read(data, 0, BUFFER)) != -1) {
                    out.write(data, 0, count);
                }
                origin.close();
            }

            out.close();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}

```

# Глава 52. Распаковка Zip-файлов в Android

```

private boolean unpackZip(String path, String zipname){
    InputStream is;
    ZipInputStream zis;
    try {
        String filename;
        is = new FileInputStream(path + zipname);
        zis = new ZipInputStream(new BufferedInputStream(is));
        ZipEntry ze;
        byte[] buffer = new byte[1024];
        int count;

        while ((ze = zis.getNextEntry()) != null){
            filename = ze.getName();

            // Необходимо создать каталоги, если они не существуют,
            // или будет сгенерировано исключение...
            if (ze.isDirectory()) {
                File fmd = new File(path + filename);
                fmd.mkdirs();
                continue;
            }

            FileOutputStream fout = new FileOutputStream(path + filename);

            // чтение и запись zip
            while ((count = zis.read(buffer)) != -1){
                fout.write(buffer, 0, count);
            }

            fout.close();
            zis.closeEntry();
        }
        zis.close();
    } catch(IOException e){
        e.printStackTrace();
        return false;
    }
    return true;
}

```

# Глава 53. Камера и галерея

## 53.1. Фотографирование

Добавьте разрешение на доступ к камере в файл AndroidManifest:

```

<uses-permission android:name="android.permission.CAMERA"></uses-permission>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

```

Xml-файл:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <SurfaceView android:id="@+id/surfaceView" android:layout_height="0dip"
        android:layout_width="0dip"></SurfaceView>
    <ImageView android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:id="@+id/imageView"></ImageView>
</LinearLayout>
```

Активность:

```
import java.io.IOException;

import android.app.Activity;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.hardware.Camera;
import android.hardware.Camera.Parameters;
import android.os.Bundle;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.widget.ImageView;

public class TakePicture extends Activity implements SurfaceHolder.Callback
{
    // переменная для хранения ссылки на Image View в файле main.xml
    private ImageView iv_image;
    // переменная для хранения ссылки на Surface View в файле main.xml
    private SurfaceView sv;

    // bitmap для отображения захваченного изображения
    private Bitmap bmp;

    // Переменные камеры
    // держатель поверхности – surface holder
    private SurfaceHolder sHolder;
    // переменная для управления камерой
    private Camera mCamera;
    // параметры камеры
    private Parameters parameters;

    /** Вызывается при первом создании активности. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        //получить представление изображения Image View в файле main.xml
        iv_image = (ImageView) findViewById(R.id.imageView);

        //получение вида поверхности Surface View в файле main.xml
        sv = (SurfaceView) findViewById(R.id.surfaceView);
```

```

//Получение поверхности
sHolder = sv.getHolder();
//добавить методы интерфейса обратного вызова, определенные ниже, в качестве
//обратных вызовов Surface View
sHolder.addCallback(this);
//указывает Android, что на этой поверхности данные будут постоянно
//заменяться
sHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
}

@Override
public void surfaceChanged(SurfaceHolder arg0, int arg1, int arg2, int arg3)
{
    //получение параметров камеры
    parameters = mCamera.getParameters();

    //установка параметров камеры
    mCamera.setParameters(parameters);
    mCamera.startPreview();

    //устанавливает, какой код должен быть выполнен после получения снимка
    Camera.PictureCallback mCall = new Camera.PictureCallback()
    {
        @Override
        public void onPictureTaken(byte[] data, Camera camera)
        {
            //декодировать данные, полученные камерой, в Bitmap
            bmp = BitmapFactory.decodeByteArray(data, 0, data.length);
            String filename=Environment.getExternalStorageDirectory()
                + File.separator + "testimage.jpg";
            FileOutputStream out = null;
            try {
                out = new FileOutputStream(filename);
                bmp.compress(Bitmap.CompressFormat.PNG, 100, out); // bmp - это
                //экземпляр вашего растрового изображения

                // PNG - формат без потерь, коэффициент сжатия (100) игнорируется
            } catch (Exception e) {
                e.printStackTrace();
            } finally {
                try {
                    if (out != null) {
                        out.close();
                    }
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            //установка изображения iv_image
            iv_image.setImageBitmap(bmp);
        }
    };
    mCamera.takePicture(null, null, mCall);
}

```

```

@Override
public void surfaceCreated(SurfaceHolder holder)
{
    // Поверхность создана, возьмите камеру и сообщите ей, где
    // для отрисовки предварительного просмотра.
    mCamera = Camera.open();
    try {
        mCamera.setPreviewDisplay(holder);
    } catch (IOException exception) {
        mCamera.release();
        mCamera = null;
    }
}

@Override
public void surfaceDestroyed(SurfaceHolder holder)
{
    // остановить предварительный просмотр
    mCamera.stopPreview();
    // отпустить камеру
    mCamera.release();
    // отвязать камеру от данного объекта
    mCamera = null;
}
}

```

## 53.2. Получение полноразмерной фотографии

Чтобы сделать фотографию, сначала нужно объявить необходимые разрешения в файле `AndroidManifest.xml`.

Нам нужны два разрешения:

- `Camera` – чтобы открыть приложение камеры. Если атрибут `required` имеет значение `true`, то при отсутствии аппаратной камеры установить это приложение не удастся.
- `WRITE_EXTERNAL_STORAGE` – это разрешение необходимо для создания нового файла, в который будет сохранен отснятый снимок.

### `AndroidManifest.xml`

```

<uses-feature android:name="android.hardware.camera"
    android:required="true" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

```

Основная идея получения полноразмерной фотографии с камеры заключается в том, что перед тем, как открыть приложение камеры и сделать снимок, необходимо создать новый файл для фотографии:

```

private void dispatchTakePictureIntent() {
    Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    // Убедитесь, что существует активность камеры для обработки намерения
    if (takePictureIntent.resolveActivity(getPackageManager()) != null) {
        // Создать файл, в который будет помещена фотография
        File photoFile = null;
        try {
            photoFile = createImageFile();
        } catch (IOException ex) {

```

```

        Log.e("DEBUG_TAG", "createFile", ex);
    }
    // Продолжать только в том случае, если файл был успешно создан
    if (photoFile != null) {
        takePictureIntent.putExtra(MediaStore.EXTRA_OUTPUT, Uri.
fromFile(photoFile));
        startActivityForResult(takePictureIntent, REQUEST_IMAGE_CAPTURE);
    }
}

private File createImageFile() throws IOException {
    // Создание имени файла изображения
    String timeStamp = new SimpleDateFormat("yyyyMMdd_HHmmss", Locale.getDefault()).
format(new Date());
    String imageFileName = "JPEG_" + timeStamp + "_";
    File storageDir = getAlbumDir();
    File image = File.createTempFile(
        imageFileName,           /* префикс */
        ".jpg",                 /* суффикс */
        storageDir              /* каталог */
    );
    // Сохранить файл: путь для использования с намерениями ACTION_VIEW
    mCurrentPhotoPath = image.getAbsolutePath();
    return image;
}

private File getAlbumDir() {
    File storageDir = null;
    if (Environment.MEDIA_MOUNTED.equals(Environment.getExternalStorageState())) {
        storageDir = new File(Environment.getExternalStorageDirectory()
            + "/dcim/"
            + "MyRecipes");
        if (!storageDir.mkdirs()) {
            if (!storageDir.exists()) {
                Log.d("CameraSample", "не удалось создать каталог");
                return null;
            }
        } else {
            Log.v(getString(R.string.app_name), "Внешнее хранилище не смонтировано
на READ/WRITE.");
        }
        return storageDir;
    }
    private void setPic() {
        /* Не хватает памяти для открытия более чем пары фотографий с камеры */
        /* Таким образом, осуществляется предварительное масштабирование целевого
растрового изображения, в которое декодируется файл */
        /* Получение размера ImageView */
    }
}

```

```
int targetW = recipeImage.getWidth();
int targetH = recipeImage.getHeight();

/* Получение размера изображения */
BitmapFactory.Options bmOptions = new BitmapFactory.Options();
bmOptions.inJustDecodeBounds = true;
BitmapFactory.decodeFile(mCurrentPhotoPath, bmOptions);
int photoW = bmOptions.outWidth;
int photoH = bmOptions.outHeight;
/* Определите, что нужно сократить меньше */
int scaleFactor = 2;
if ((targetW > 0) && (targetH > 0)) {
    scaleFactor = Math.max(photoW / targetW, photoH / targetH);
}

/* Установка параметров растрового изображения для масштабирования цели
декодирования изображения */
bmOptions.inJustDecodeBounds = false;
bmOptions.inSampleSize = scaleFactor;
bmOptions.inPurgeable = true;

Matrix matrix = new Matrix();
matrix.postRotate(getRotation());

/* Декодирование JPEG-файла в растровое изображение */
Bitmap bitmap = BitmapFactory.decodeFile(mCurrentPhotoPath, bmOptions);
bitmap = Bitmap.createBitmap(bitmap, 0, 0, bitmap.getWidth(), bitmap.
getHeight(), matrix, false);

/* Привязка растрового изображения к ImageView */
recipeImage.setImageBitmap(bitmap);
}

private float getRotation() {
try {
    ExifInterface ei = new ExifInterface(mCurrentPhotoPath);
    int orientation = ei.getAttributeInt(ExifInterface.TAG_ORIENTATION,
ExifInterface.ORIENTATION_NORMAL);

    switch (orientation) {
        case ExifInterface.ORIENTATION_ROTATE_90:
            return 90f;
        case ExifInterface.ORIENTATION_ROTATE_180:
            return 180f;
        case ExifInterface.ORIENTATION_ROTATE_270:
            return 270f;
        default:
            return 0f;
    }
} catch (Exception e) {
    Log.e("Add Recipe", "getRotation", e);
    return 0f;
}
}

private void galleryAddPic() {
Intent mediaScanIntent = new Intent(Intent.ACTION_MEDIA_SCANNER_SCAN_FILE);
File f = new File(mCurrentPhotoPath);
Uri contentUri = Uri.fromFile(f);
```

```

        mediaScanIntent.setData(contentUri);
        sendBroadcast(mediaScanIntent);
    }

    private void handleBigCameraPhoto() {

        if (mCurrentPhotoPath != null) {
            setPic();
            galleryAddPic();
        }
    }

@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == REQUEST_IMAGE_CAPTURE && resultCode == Activity.RESULT_OK) {
        handleBigCameraPhoto();
    }
}

```

### 53.3. Декодирование растрового изображения, правильно повернутого из URI

Рассмотрим декодирование растрового изображения, правильно повернутого из URI, полученного с помощью намерения:

```

private static final String TAG = "IntentBitmapFetch";
private static final String COLON_SEPARATOR = ":";
private static final String IMAGE = "image";

@Nullable
public Bitmap getBitmap(@NonNull Uri bitmapUri, int maxDimen) {
    InputStream is = context.getContentResolver().openInputStream(bitmapUri);
    Bitmap bitmap = BitmapFactory.decodeStream(is, null, getBitmapOptions(bitmapUri,
        maxDimen));

    int imgRotation = getImageRotationDegrees(bitmapUri);

    int endRotation = (imgRotation < 0) ? -imgRotation : imgRotation;
    endRotation %= 360;
    endRotation = 90 * (endRotation / 90);
    if (endRotation > 0 && bitmap != null) {
        Matrix m = new Matrix();
        m.setRotate(endRotation);
        Bitmap tmp = Bitmap.createBitmap(bitmap, 0, 0, bitmap.getWidth(), bitmap.
            getHeight(), m,
        true);
        if (tmp != null) {
            bitmap.recycle();
            bitmap = tmp;
        }
    }
}

private void setPic() {
    return bitmap;
}

private BitmapFactory.Options getBitmapOptions(Uri uri, int imageMaxDimen){
    BitmapFactory.Options options = new BitmapFactory.Options();
}

```

```

if (imageMaxDimen > 0) {
    options.inJustDecodeBounds = true;
    decodeImage(null, uri, options);
    options.inSampleSize = calculateScaleFactor(options, imageMaxDimen);
    options.inJustDecodeBounds = false;
    options.inPreferredConfig = Bitmap.Config.RGB_565;
    addInBitmapOptions(options);
}

private int calculateScaleFactor(@NonNull BitmapFactory.Options bitmapOptionsMeasureOnly, int imageMaxDimen) {
    int inSampleSize = 1;
    if (bitmapOptionsMeasureOnly.outHeight > imageMaxDimen || bitmapOptionsMeasureOnly.outWidth > imageMaxDimen) {
        final int halfHeight = bitmapOptionsMeasureOnly.outHeight / 2;
        final int halfWidth = bitmapOptionsMeasureOnly.outWidth / 2;
        while ((halfHeight / inSampleSize) > imageMaxDimen && (halfWidth / inSampleSize) > imageMaxDimen) {
            inSampleSize *= 2;
        }
    }
    return inSampleSize;
}

public int getImageRotationDegrees(@NonNull Uri imgUri) {
    int photoRotation = ExifInterface.ORIENTATION_UNDEFINED;

    try {
        boolean hasRotation = false;
        //Если изображение поступает из галереи и не находится в папке DCIM (схема: content://)
        String[] projection = {MediaStore.Images.ImageColumns.ORIENTATION};
        Cursor cursor = context.getContentResolver().query(imgUri, projection, null, null, null);
        if (cursor != null) {
            if (cursor.getColumnCount() > 0 && cursor.moveToFirst()) {
                photoRotation = cursor.getInt(cursor.getColumnIndex(projection[0]));
                hasRotation = photoRotation != 0;
                Log.d("Ориентация курсора: " + photoRotation);
            }
            cursor.close();
        }
        //Если изображение поступает с камеры (схема: файл://) или находится в папке DCIM (схема: контент://)
        if (!hasRotation) {
            ExifInterface exif = new ExifInterface(getAbsolutePath(imgUri));
            int exifRotation = exif.getAttributeInt(ExifInterface.TAG_ORIENTATION, ExifInterface.ORIENTATION_NORMAL);
            switch (exifRotation) {
                case ExifInterface.ORIENTATION_ROTATE_90: {
                    photoRotation = 90;
                    break;
                }
                case ExifInterface.ORIENTATION_ROTATE_180: {
                    photoRotation = 180;
                    break;
                }
            }
        }
    } catch (Exception e) {
        Log.e("Ошибка при получении ориентации изображения", e.getMessage());
    }
}

```

```

        case ExifInterface.ORIENTATION_ROTATE_270: {
            photoRotation = 270;
            break;
        }
    }
    Log.d(TAG, "Exif ориентация: " + photoRotation);
}
} catch (IOException e) {
    Log.e(TAG, "Ошибка определения поворота для изображения " + imgUri, e);
}
return photoRotation;
}

@TargetApi(Build.VERSION_CODES.KITKAT)
private String getAbsolutePath(Uri uri) {
    //Код фрагмента отредактирован из: http://stackoverflow.com/a/20559418/2235133
    String filePath = uri.getPath();
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.KITKAT && DocumentsContract.isDocumentUri(context, uri)) {
        // Вернется "image:x"
        String[] wholeID = TextUtils.split(DocumentsContract.getDocumentId(uri),
                COLON_SEPARATOR);
        // Разделение по двоеточию, использование второго элемента массива
        String type = wholeID[0];
        if(IMAGE.equalsIgnoreCase(type)){//Если изображение не имеет типа, значит, оно
            получено из удаленного местоположения, например, из Google Photos
            String id = wholeID[1];
            String[] column = {MediaStore.Images.Media.DATA};
            // где id равно
            String sel = MediaStore.Images.Media._ID + "=?";
            Cursor cursor = context.getContentResolver().query(MediaStore.Images.
                    Media.EXTERNAL_CONTENT_URI, column, sel, new String[]{id}, null);
            if (cursor != null) {
                int columnIndex = cursor.getColumnIndex(column[0]);
                if (cursor.moveToFirst()) {
                    filePath = cursor.getString(columnIndex);
                }
                cursor.close();
            }
            Log.d(TAG, "Получен абсолютный путь для uri" + uri);
        }
    }
    return filePath;
}

```

### 53.4. Установка разрешения камеры

Программная установка высокого разрешения:

```

Camera mCamera = Camera.open();
Camera.Parameters params = mCamera.getParameters();

// Проверьте, какие разрешения поддерживает ваша камера
List<Size> sizes = params.getSupportedPictureSizes();

// Перебираем все доступные разрешения и выбираем одно.
// Выбранное разрешение будет храниться в mSize.

```

```

Size mSize;
for (Size size : sizes) {
    Log.i(TAG, "Доступное разрешение: "+size.width+" "+size.height);
    mSize = size;
}
}

Log.i(TAG, "Выбранное разрешение: "+mSize.width+" "+mSize.height);
params.setPictureSize(mSize.width, mSize.height);
mCamera.setParameters(params);

```

## 53.5. Как запустить камеру или галерею и сохранить результат работы камеры

Прежде всего вам нужны Uri и временные папки, а также коды запросов:

```

public final int REQUEST_SELECT_PICTURE = 0x01;
public final int REQUEST_CODE_TAKE_PICTURE = 0x2;
public static String TEMP_PHOTO_FILE_NAME = "photo_";
Uri mImageCaptureUri;
File mFileTemp;

```

Затем инициализируйте mFileTemp:

```

public void initTempFile(){
    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)) {
        mFileTemp = new File(Environment.getExternalStorageDirectory() + File.separator
            + getResources().getString(R.string.app_foldername) + File.separator
            + getResources().getString(R.string.pictures_folder)
            , TEMP_PHOTO_FILE_NAME
            + System.currentTimeMillis() + ".jpg");
        mFileTemp.getParentFile().mkdirs();
    } else {
        mFileTemp = new File(getFilesDir() + File.separator
            + getResources().getString(R.string.app_foldername)
            + File.separator + getResources().getString(R.string.pictures_folder)
            , TEMP_PHOTO_FILE_NAME + System.currentTimeMillis() + ".jpg");
        mFileTemp.getParentFile().mkdirs();
    }
}

```

Открытие намерений Camera и Gallery:

```

public void openCamera(){
    Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    try {
        mImageCaptureUri = null;
        String state = Environment.getExternalStorageState();
        if (Environment.MEDIA_MOUNTED.equals(state)) {
            mImageCaptureUri = Uri.fromFile(mFileTemp);
        } else {
            mImageCaptureUri = InternalStorageContentProvider.CONTENT_URI;
        }
    }
}

```

```

        intent.putExtra(MediaStore.EXTRA_OUTPUT, mImageCaptureUri);
        intent.putExtra("return-data", true);
        startActivityForResult(intent, REQUEST_CODE_TAKE_PICTURE);
    } catch (Exception e) {
        Log.d("ошибка", "невозможно сделать фотографию", e);
    }
}

public void openGallery(){
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.JELLY_BEAN
        && ActivityCompat.checkSelfPermission(this, Manifest.permission.READ_EXTERNAL_STORAGE)
        != PackageManager.PERMISSION_GRANTED) {
        requestPermission(Manifest.permission.READ_EXTERNAL_STORAGE,
            getString(R.string.permission_read_storage_rationale), REQUEST_STORAGE_READ_ACCESS_PERMISSION);
    } else {
        Intent intent = new Intent();
        intent.setType("image/*");
        intent.setAction(Intent.ACTION_GET_CONTENT);
        intent.addCategory(Intent.CATEGORY_OPENABLE);
        startActivityForResult(Intent.createChooser(intent, getString(R.string.select_image)), REQUEST_SELECT_PICTURE);
    }
}

```

Затем в методе onActivityResult:

```

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode != RESULT_OK) {
        return;
    }
    Bitmap bitmap;
    switch (requestCode) {
        case REQUEST_SELECT_PICTURE:
            try {
                Uri uri = data.getData();
                try {
                    bitmap = MediaStore.Images.Media.getBitmap(getContentResolver(), uri);
                    Bitmap bitmapScaled = Bitmap.createScaledBitmap(bitmap, 800, 800, true);
                    Drawable drawable = new BitmapDrawable(bitmapScaled);
                    mImage.setImageDrawable(drawable);
                    mImage.setVisibility(View.VISIBLE);
                } catch (IOException e) {
                    Log.v("результат действия", "здесь ошибка : " + e.getContent());
                }
            } catch (Exception e) {
                Log.v("результат действия", "здесь ошибка : " + e.getContent());
            }
            break;
        case REQUEST_CODE_TAKE_PICTURE:
            try{
                Bitmap bitmappicture = MediaStore.Images.Media.getBitmap(getContentResolver() , mImageCaptureUri);

```

```

        mImage.setImageBitmap(bitmap);
        mImage.setVisibility(View.VISIBLE);
    }catch (IOException e){
        Log.v("ошибка камеры", e.getMessage());
    }
    break;
}
super.onActivityResult(requestCode, resultCode, data);
}

```

Эти разрешения необходимо указать в файле `AndroidManifest.xml`:

```

<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.CAMERA" />

```

Также нужно обрабатывать разрешения во время выполнения (<https://developer.android.com/training/permissions/requesting.html>), такие как чтение/запись внешнего хранилища и т. д. ... В этом примере проверяется разрешение `READ_EXTERNAL_STORAGE` в методе `openGallery`:

Метод `requestPermission`:

```

protected void requestPermission(final String permission, String rationale, final
int requestCode) {
    if (ActivityCompat.shouldShowRequestPermissionRationale(this, permission)) {
        showDialog(getString(R.string.permission_title_rationale), rationale,
                new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        ActivityCompat.requestPermissions(BasePermissionActivity.this,
                                new String[]{permission}, requestCode);
                    }
                }, getString(android.R.string.ok), null, getString(android.R.string.
cancel));
    } else {
        ActivityCompat.requestPermissions(this, new String[]{permission},
                requestCode);
    }
}

```

Затем переопределим метод `onRequestPermissionsResult`:

```

@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[]
permissions, @NonNull int[] grantResults) {
    switch (requestCode) {
        case REQUEST_STORAGE_READ_ACCESS_PERMISSION:
            if (grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                handleGallery();
            }
            break;
        default:
            super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    }
}

```

Метод `showAlertDialog`:

```

protected void showAlertDialog(@Nullable String title, @Nullable String message,
                               @Nullable DialogInterface.OnClickListener
onPositiveButtonClickListener,

```

```

    @NonNull String positiveText,
    @Nullable DialogInterface.OnClickListener
onNegativeButtonClickListener,
    @NonNull String negativeText) {
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setTitle(title);
builder.setMessage(message);
builder.setPositiveButton(positiveText, onPositiveButtonClickListener);
builder.setNegativeButton(negativeText, onNegativeButtonClickListener);
mAlertDialog = builder.show();
}

```

## Глава 54. API Camera2

Параметр	Подробности
CameraCaptureSession	Сконфигурированный сеанс захвата для устройства CameraDevice, используемый для захвата изображений с камеры или повторной обработки изображений, захваченных с камеры в том же сеансе ранее
CameraDevice	Представление одной камеры, подключенной к устройству Android
CameraCharacteristics	Свойства, описывающие устройство CameraDevice. Эти свойства фиксированы для данного устройства CameraDevice и могут быть запрошены через интерфейс CameraManager с помощью функции getCameraCharacteristics(String)
CameraManager	Менеджер системных служб для обнаружения, определения характеристик и подключения к устройствам CameraDevice. Получить экземпляр этого класса можно, вызвав Context.getSystemService()
CaptureRequest	Неизменяемый пакет настроек и выходов, необходимых для захвата одного изображения с устройства камеры. Содержит конфигурацию аппаратных средств захвата (матрица, объектив, вспышка), конвейер обработки, алгоритмы управления и выходные буферы. Также содержит список целевых поверхностей (target Surfaces), на которые необходимо отправить данные изображения для данного захвата. Может быть создан с помощью экземпляра CaptureRequest.Builder, полученного вызовом createCaptureRequest(int)
CaptureResult	Подмножество результатов захвата одного изображения с датчика изображения. Содержит подмножество конечной конфигурации аппаратных средств захвата (матрица, объектив, вспышка), конвейера обработки, алгоритмов управления и выходных буферов. Создается устройством CameraDevice после обработки запроса CaptureRequest

### 54.1. Предварительный просмотр основной камеры в TextureView

Вы должны добавить в манифест следующее разрешение:

```
<uses-permission android:name="android.permission.CAMERA"/>
```

Создадим активность (Camera2Activity.java), которая заполнит TextureView предварительным изображением камеры устройства.

Активность, которую мы будем использовать, представляет собой типичную AppCompatActivity:

```
public class Camera2Activity extends AppCompatActivity {
```

Атрибуты (для понимания некоторых из них может потребоваться прочтение всего приложения)

```
MAX_PREVIEW_SIZE, гарантируемый API Camera2, равен 1920x1080
```

```
private static final int MAX_PREVIEW_WIDTH = 1920;
```

```
private static final int MAX_PREVIEW_HEIGHT = 1080;
```

TextureView.SurfaceTextureListener обрабатывает несколько событий жизненного цикла TextureView. В данном случае мы слушаем эти события. Когда SurfaceTexture готова, мы инициализируем камеру. При изменении размеров мы соответствующим образом настраиваем предварительный просмотр:

```
private final TextureView.SurfaceTextureListener mSurfaceTextureListener {
```

```
    = new TextureView.SurfaceTextureListener() {
```

```
@Override
```

```
public void onSurfaceTextureAvailable(SurfaceTexture texture, int width, int height) {
```

```
    openCamera(width, height);
```

```
}
```

```
@Override
```

```
public void onSurfaceTextureSizeChanged(SurfaceTexture texture, int width, int height) {
```

```
    configureTransform(width, height);
```

```
}
```

```
@Override
```

```
public boolean onSurfaceTextureDestroyed(SurfaceTexture texture) {
```

```
    return true;
```

```
}
```

```
@Override
```

```
public void onSurfaceTextureUpdated(SurfaceTexture texture) {
```

```
}
```

```
};
```

CameraDevice представляет собой камеру одного физического устройства. В этом атрибуте мы сохраняем идентификатор текущего устройства CameraDevice:

```
private String mCameraId;
```

Это представление (TextureView), которое мы будем использовать для "рисования" предварительного просмотра камеры:

```
private TextureView mTextureView;
```

CameraCaptureSession для предварительного просмотра камеры:

```
private CameraCaptureSession mCaptureSession;
```

Ссылка на открытое устройство CameraDevice:

```
private CameraDevice mCameraDevice;
```

Размер предварительного просмотра камеры:

```
private Size mPreviewSize;
```

CameraDevice.StateCallback вызывается при изменении состояния CameraDevice:

```
private final CameraDevice.StateCallback mStateCallback = new CameraDevice.StateCallback() {
```

```
    @Override
```

```
    public void onOpened(@NonNull CameraDevice cameraDevice) {
```

```
        // Этот метод вызывается при открытии камеры. Здесь мы запускаем  
        // предварительный просмотр камеры.
```

```
        mCameraOpenCloseLock.release();
```

```
        mCameraDevice = cameraDevice;
```

```
        createCameraPreviewSession();
```

```
}
```

```
    @Override
```

```
    public void onDisconnected(@NonNull CameraDevice cameraDevice) {
```

```
        mCameraOpenCloseLock.release();
```

```
        cameraDevice.close();
```

```
        mCameraDevice = null;
```

```
}
```

```
    @Override
```

```
    public void onError(@NonNull CameraDevice cameraDevice, int error) {
```

```
        mCameraOpenCloseLock.release();
```

```
        cameraDevice.close();
```

```
        mCameraDevice = null;
```

```
        finish();
```

```
}
```

```
};
```

Дополнительный поток для выполнения задач, которые не должны блокировать работу пользовательского интерфейса:

```
private HandlerThread mBackgroundThread;
```

Обработчик для выполнения задач в фоновом режиме:

```
private Handler mBackgroundHandler;
```

ImageReader, обрабатывающий захват неподвижных изображений:

```
private ImageReader mImageReader;
```

CaptureRequest.Builder для предварительного просмотра камеры:

```
private CaptureRequest.Builder mPreviewRequestBuilder;
```

CaptureRequest, генерируемый mPreviewRequestBuilder:

```
private CaptureRequest mPreviewRequest;
```

Семафор для предотвращения завершения работы приложения до закрытия камеры:

```
private Semaphore mCameraOpenCloseLock = new Semaphore(1);
```

Постоянный идентификатор запроса на разрешение:

```
private static final int REQUEST_CAMERA_PERMISSION = 1;
```

## Методы жизненного цикла Android

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_camera2);

    mTextureView = (TextureView) findViewById(R.id.texture);
}

@Override
public void onResume() {
    super.onResume();
    startBackgroundThread();

    // При выключении и повторном включении экрана текстура SurfaceTexture уже
    // доступна, и "onSurfaceTextureAvailable" вызываться не будет. В этом случае мы
    // можем открыть камеру и начать предварительный просмотр отсюда
    // (в противном случае мы будем ждать, пока поверхность не будет готова
    // в SurfaceTextureListener).
    if (mTextureView.isAvailable()) {
        openCamera(mTextureView.getWidth(), mTextureView.getHeight());
    } else {
        mTextureView.setSurfaceTextureListener(mSurfaceTextureListener);
    }
}

@Override
public void onPause() {
    closeCamera();
    stopBackgroundThread();
    super.onPause();
}
```

## Связанные методы Camera2

Методы, использующие API Camera2:

```
private void openCamera(int width, int height) {
    if (ContextCompat.checkSelfPermission(this, Manifest.permission.CAMERA))
        != PackageManager.PERMISSION_GRANTED) {
        requestCameraPermission();
        return;
    }
    setUpCameraOutputs(width, height);
    configureTransform(width, height);
    CameraManager manager = (CameraManager) getSystemService(Context.CAMERA_SERVICE);
    try {
        if (!mCameraOpenCloseLock.tryAcquire(2500, TimeUnit.MILLISECONDS)) {
            throw new RuntimeException("Time out waiting to lock camera opening.");
    }
```

```

        }
        manager.openCamera(mCameraId, mStateCallback, mBackgroundHandler);
    } catch (CameraAccessException e) {
        e.printStackTrace();
    } catch (InterruptedException e) {
        throw new RuntimeException("Interrupted while trying to lock camera
opening.", e);
    }
}
}

```

Закрытие текущей камеры:

```

private void closeCamera() {
    try {
        mCameraOpenCloseLock.acquire();
        if (null != mCaptureSession) {
            mCaptureSession.close();
            mCaptureSession = null;
        }
        if (null != mCameraDevice) {
            mCameraDevice.close();
            mCameraDevice = null;
        }
        if (null != mImageReader) {
            mImageReader.close();
            mImageReader = null;
        }
    } catch (InterruptedException e) {
        throw new RuntimeException("Interrupted while trying to lock camera
closing.", e);
    } finally {
        mCameraOpenCloseLock.release();
    }
}

```

Установка переменных-членов, связанных с камерой:

```

private void setUpCameraOutputs(int width, int height) {
    CameraManager manager = (CameraManager) getSystemService(Context.CAMERA_SERVICE);
    try {
        for (String cameraId : manager.getCameraIdList()) {
            CameraCharacteristics characteristics
                = manager.getCameraCharacteristics(cameraId);

            // В данном примере мы не используем фронтальную камеру.
            Integer facing = characteristics.get(CameraCharacteristics.LENS_FACING);
            if (facing != null && facing == CameraCharacteristics.LENS_FACING_FRONT) {
                continue;
            }

            StreamConfigurationMap map = characteristics.get(
                CameraCharacteristics.SCALER_STREAM_CONFIGURATION_MAP);
            if (map == null) {
                continue;
            }

            // Для захвата неподвижных изображений используется наибольший доступный
            // размер.
            Size largest = Collections.max(

```

```

        Arrays.asList(map.getOutputSizes(ImageFormat.JPEG)),
        new CompareSizesByArea());
    mImageReader = ImageReader.newInstance(largest.getWidth(), largest.
    getHeight(), ImageFormat.JPEG, /*maxImages*/2);
    mImageReader.setOnImageAvailableListener(
        null, mBackgroundHandler);

    Point displaySize = new Point();
    getWindowManager().getDefaultDisplay().getSize(displaySize);
    int rotatedPreviewWidth = width;
    int rotatedPreviewHeight = height;
    int maxPreviewWidth = displaySize.x;
    int maxPreviewHeight = displaySize.y;

    if (maxPreviewWidth > MAX_PREVIEW_WIDTH) {
        maxPreviewWidth = MAX_PREVIEW_WIDTH;
    }

    if (maxPreviewHeight > MAX_PREVIEW_HEIGHT) {
        maxPreviewHeight = MAX_PREVIEW_HEIGHT;
    }

    // Опасность! Попытка использовать слишком большой размер предварительного
    // просмотра может превысить возможности пропускной способности
    // шины камеры, что приводит к великолепному качеству предварительного
    // просмотра, но приводит к хранению большого объема захваченного "мусора".
    mPreviewSize = chooseOptimalSize(map.getOutputSizes(SurfaceTexture.
    class), rotatedPreviewWidth, rotatedPreviewHeight, maxPreviewWidth,
    maxPreviewHeight, largest);
    mCameraId = cameraId;
    return;
}

} catch (CameraAccessException e) {
    e.printStackTrace();
} catch (NullPointerException e) {
    // Ошибка NPE происходит, когда Camera2 API используется, но не поддерживается
    // на устройстве, на котором выполняется этот код.
    Toast.makeText(Camera2Activity.this, "Camera2 API не поддерживается на данном
    устройстве", Toast.LENGTH_LONG).show();
}
}
}

```

Создание нового сеанса CameraCaptureSession для предварительного просмотра камеры:

```

private void createCameraPreviewSession() {
    try {
        SurfaceTexture texture = mTextureView.getSurfaceTexture();
        assert texture != null;

        // Настраиваем размер буфера по умолчанию на нужный нам размер
        // предварительного просмотра камеры.
        texture.setDefaultBufferSize(mPreviewSize.getWidth(), mPreviewSize.
        getHeight());

        // Это выходная поверхность, которая нам нужна для запуска предварительного
        // просмотра.
        Surface surface = new Surface(texture);
    }
}
}

```

```

// Настраиваем CaptureRequest.Builder с выходной поверхностью.
mPreviewRequestBuilder = mCameraDevice.createCaptureRequest(CameraDevice.
TEMPLATE_PREVIEW);
mPreviewRequestBuilder.addTarget(surface);

// Создаем сессию CameraCaptureSession для предварительного просмотра камеры.
mCameraDevice.createCaptureSession(Arrays.asList(surface, mImageReader.
getSurface()),
new CameraCaptureSession.StateCallback() {

    @Override
    public void onConfigured(@NonNull CameraCaptureSession
cameraCaptureSession) {
        // Камера уже закрыта
        if (null == mCameraDevice) {
            return;
        }

        // Когда сессия готова, мы начинаем отображать предварительный
        // просмотр.
        mCaptureSession = cameraCaptureSession;
        try {
            // Автофокусировка должна быть непрерывной для предварительного
            // просмотра камеры.
            mPreviewRequestBuilder.set(CaptureRequest.CONTROL_AF_MODE,
                CaptureRequest.CONTROL_AF_MODE_CONTINUOUS_PICTURE);

            // Наконец, мы начинаем отображать предварительный просмотр камеры.
            mPreviewRequest = mPreviewRequestBuilder.build();
            mCaptureSession.setRepeatingRequest(mPreviewRequest, null,
                mBackgroundHandler);
        } catch (CameraAccessException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void onConfigureFailed(
        @NonNull CameraCaptureSession cameraCaptureSession) {
        showToast("Failed");
    }
}, null
);
} catch (CameraAccessException e) {
    e.printStackTrace();
}
}
}

```

Методы, связанные с разрешениями:

```

private void requestCameraPermission() {
    if (ActivityCompat.shouldShowRequestPermissionRationale(this, Manifest.
    permission.CAMERA)) {
        new AlertDialog.Builder(Camera2Activity.this)
            .setMessage("Разрешение на запрос строки R")
            .setPositiveButton(android.R.string.ok, new DialogInterface.
    OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    ActivityCompat.requestPermissions(Camera2Activity.this,

```

```

        new String[]{Manifest.permission.CAMERA}, REQUEST_CAMERA_
    PERMISSION);
}
}
.setNegativeButton(android.R.string.cancel,
    new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            finish();
        }
})
.create();

} else {
    ActivityCompat.requestPermissions(this, new String[]{Manifest.permission.
    CAMERA}, REQUEST_CAMERA_PERMISSION);
}
}

@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {
    if (requestCode == REQUEST_CAMERA_PERMISSION) {
        if (grantResults.length != 1 || grantResults[0] != PackageManager.PERMISSION_GRANTED) {
            Toast.makeText(Camera2Activity.this, "ERROR: Разрешения на камеру не
            предоставлены", Toast.LENGTH_LONG).show();
        }
    } else {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    }
}
}

```

## Фоновый поток и методы обработчика

```

private void startBackgroundThread() {
    mBackgroundThread = new HandlerThread("CameraBackground");
    mBackgroundThread.start();
    mBackgroundHandler = new Handler(mBackgroundThread.getLooper());
}

private void stopBackgroundThread() {
    mBackgroundThread.quitSafely();
    try {
        mBackgroundThread.join();
        mBackgroundThread = null;
        mBackgroundHandler = null;
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}

```

## Служебные методы

При выборе размеров, поддерживаемых камерой, выбирается наименьший из них, который не меньше соответствующего размера вида текстуры, не больше соответствующего максимального размера, и соотношение сторон которого совпадает с указанным значением. Если такого размера не существует, то выбирается наибольший, который не меньше со-

ответствующего максимального размера, и соотношение сторон которого соответствует заданному значению.

```

private static Size chooseOptimalSize(Size[] choices, int textureViewWidth,
int textureViewHeight, int maxWidth, int maxHeight, Size aspectRatio) {

    // Собираем поддерживаемые разрешения, которые, по крайней мере, не меньше,
    // чем размер поверхности предварительного просмотра (preview Surface)
    List<Size> bigEnough = new ArrayList<>();
    // Собираем поддерживаемые разрешения, которые меньше, чем поверхность
    // предварительного просмотра
    List<Size> notBigEnough = new ArrayList<>();
    int w = aspectRatio.getWidth();
    int h = aspectRatio.getHeight();
    for (Size option : choices) {
        if (option.getWidth() <= maxWidth && option.getHeight() <= maxHeight &
            option.getHeight() == option.getWidth() * h / w) {
            if (option.getWidth() >= textureViewWidth && option.getHeight() >=
                textureViewHeight) {
                bigEnough.add(option);
            } else {
                notBigEnough.add(option);
            }
        }
    }
}

// Выбираем наименьшее из достаточно больших. Если нет ни одного достаточно
// большого, выбираем наибольшее из тех, что недостаточно велики.
if (bigEnough.size() > 0) {
    return Collections.min(bigEnough, new CompareSizesByArea());
} else if (notBigEnough.size() > 0) {
    return Collections.max(notBigEnough, new CompareSizesByArea());
} else {
    Log.e("Camera2", "Не удалось найти подходящий размер превью");
    return choices[0];
}
}

```

Этот метод выполняет необходимое преобразование матрицы в `mTextureView`:

```

private void configureTransform(int viewWidth, int viewHeight) {
    if (null == mTextureView || null == mPreviewSize) {
        return;
    }
    int rotation = getWindowManager().getDefaultDisplay().getRotation();
    Matrix matrix = new Matrix();
    RectF viewRect = new RectF(0, 0, viewWidth, viewHeight);
    RectF bufferRect = new RectF(0, 0, mPreviewSize.getHeight(), mPreviewSize.
        getWidth());
    float centerX = viewRect.centerX();
    float centerY = viewRect.centerY();
    if (Surface.ROTATION_90 == rotation || Surface.ROTATION_270 == rotation) {
        bufferRect.offset(centerX - bufferRect.centerX(), centerY - bufferRect.
            centerY());
    }
    matrix.setRectToRect(viewRect, bufferRect, Matrix.ScaleToFit.FILL);
    float scale = Math.max(
        (float) viewHeight / mPreviewSize.getHeight(),
        (float) viewWidth / mPreviewSize.getWidth());
}

```

```

        matrix.postScale(scale, scale, centerX, centerY);
        matrix.postRotate(90 * (rotation - 2), centerX, centerY);
    } else if (Surface.ROTATION_180 == rotation) {

        matrix.postRotate(180, centerX, centerY);
    }
    mTextureView.setTransform(matrix);
}

```

В этом методе два размера сравниваются по их площадям:

```

static class CompareSizesByArea implements Comparator<Size> {

    @Override
    public int compare(Size lhs, Size rhs) {
        // Осуществляем приведение, чтобы при умножении не было переполнения.
        return Long.signum((long) lhs.getWidth() * lhs.getHeight() - (long) rhs.
            getWidth() * rhs.getHeight());
    }
}

```

## Глава 55. API Fingerprint

### 55.1. Как использовать Android Fingerprint API для сохранения паролей пользователей

Данный пример класса-помощника взаимодействует с менеджером отпечатков пальцев и выполняет шифрование и дешифрование пароля. Обратите внимание, что для шифрования в данном примере используется метод AES – это не единственный способ шифрования. В данном примере данные шифруются и дешифруются следующим образом.

#### Шифрование:

- Пользователь сообщает помощнику желаемый нешифрованный пароль.
- Пользователю необходимо предоставить отпечаток пальца.
- После аутентификации помощник получает ключ из KeyStore и шифрует пароль с помощью шифра Cipher.

Пароль и «соль» IV (она создается заново при каждом шифровании и не используется повторно) сохраняются в общих настройках для последующего использования в процессе дешифрования.

#### Расшифровка:

- Пользователь запрашивает расшифровку пароля.
- Пользователю необходимо предоставить отпечаток пальца.
- Помощник строит шифр, используя «соль» IV, и после аутентификации пользователя хранилище ключей получает ключ из хранилища KeyStore и расшифровывает пароль.

```

public class FingerPrintAuthHelper {

    private static final String FINGER_PRINT_HELPER = "FingerPrintAuthHelper";
    private static final String ENCRYPTED_PASS_SHARED_PREF_KEY = "ENCRYPTED_PASS_
    SHARED_PREF_KEY";
    private static final String LAST_USED_IV_SHARED_PREF_KEY = "LAST_USED_IV_SHARED_
    PREF_KEY";
    private static final String MY_APP_ALIAS = "MY_APP_ALIAS";
}

```



```
private KeyguardManager keyguardManager;
private FingerprintManager fingerprintManager;

private final Context context;
private KeyStore keyStore;
private KeyGenerator KeyGenerator;

private String lastError;

public interface Callback {
    void onSuccess(String savedPass);

    void onFailure(String message);

    void onHelp(int helpCode, String helpString);
}

public FingerPrintAuthHelper(Context context) {
    this.context = context;
}

public String getLastErrorMessage() {
    return lastError;
}

@TargetApi(Build.VERSION_CODES.M)
public boolean init() {
    if (Build.VERSION.SDK_INT < Build.VERSION_CODES.M) {
        setError("Данная версия Android не поддерживает аутентификацию
        по отпечаткам пальцев");
        return false;
    }

    keyguardManager = (KeyguardManager) context.getSystemService(KEYGUARD_
SERVICE);
    fingerprintManager = (FingerprintManager) context.
    getSystemService(FINGERPRINT_SERVICE);

    if (!keyguardManager.isKeyguardSecure()) {
        setError("Пользователь не включил блокировку экрана");
        return false;
    }

    if (!hasPermission()) {
        setError("Пользователь не дал разрешения на использование отпечатка
        пальца");
        return false;
    }

    if (!fingerprintManager.hasEnrolledFingerprints()) {
        setError("Пользователь не зарегистрировал отпечатки пальцев");
        return false;
    }

    if (!initKeyStore()) {
        return false;
    }
    return true;
}
```

```
}

@NoArgsConstructor
@RequiresApi(api = Build.VERSION_CODES.M)
private Cipher createCipher(int mode) throws NoSuchPaddingException,
NoSuchAlgorithmException, UnrecoverableKeyException, KeyStoreException,
InvalidKeyException, InvalidAlgorithmParameterException {
    Cipher cipher = Cipher.getInstance(KeyProperties.KEY_ALGORITHM_AES + "/" +
        KeyProperties.BLOCK_MODE_CBC + "/" +
        KeyProperties.ENCRYPTION_PADDING_PKCS7);

    Key key = keyStore.getKey(MY_APP_ALIAS, null);
    if (key == null) {
        return null;
    }
    if(mode == Cipher.ENCRYPT_MODE) {
        cipher.init(mode, key);
        byte[] iv = cipher.getIV();
        saveIv(iv);
    } else {
        byte[] lastIv = getLastIv();
        cipher.init(mode, key, new IvParameterSpec(lastIv));
    }
    return cipher;
}

@NonNull
@RequiresApi(api = Build.VERSION_CODES.M)
private KeyGenParameterSpec createKeyGenParameterSpec() {
    return new KeyGenParameterSpec.Builder(MY_APP_ALIAS, KeyProperties.PURPOSE_
ENCRYPT | KeyProperties.PURPOSE_DECRYPT)
        .setBlockModes(KeyProperties.BLOCK_MODE_CBC)
        .setUserAuthenticationRequired(true)
        .setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_PKCS7)
        .build();
}

@RequiresApi(api = Build.VERSION_CODES.M)
private boolean initKeyStore() {
    try {
        keyStore = KeyStore.getInstance("AndroidKeyStore");
        keyGenerator = KeyGenerator.getInstance(KeyProperties.KEY_ALGORITHM_AES,
            "AndroidKeyStore");
        keyStore.load(null);
        if (getLastIv() == null) {
            KeyGenParameterSpec keyGeneratorSpec = createKeyGenParameterSpec();
            keyGenerator.init(keyGeneratorSpec);
            keyGenerator.generateKey();
        }
    } catch (Throwable t) {
        setError("Не удалось инициализировать keyStore и keyGenerator: " +
t.getMessage());
        return false;
    }
    return true;
}

@RequiresApi(api = Build.VERSION_CODES.M)
private void authenticate(CancellationSignal cancellationSignal,
```

```
FingerPrintAuthenticationListener authListener, int mode) {
    try {
        if (hasPermission()) {
            Cipher cipher = createCipher(mode);
            FingerprintManager.CryptoObject crypto = new FingerprintManager.
            CryptoObject(cipher);
            fingerprintManager.authenticate(crypto, cancellationSignal, 0,
                authListener, null);
        } else {
            authListener.getCallback().onFailure("Пользователь не предоставил
                разрешение на использование отпечатка пальца");
        }
    } catch (Throwable t) {
        authListener.getCallback().onFailure("Произошла ошибка: " +
            t.getMessage());
    }
}

private String getSavedEncryptedPassword() {
    SharedPreferences sharedPreferences = getSharedPreferences();
    if (sharedPreferences != null) {
        return sharedPreferences.getString(ENCRYPTED_PASS_SHARED_PREF_KEY, null);
    }
    return null;
}

private void saveEncryptedPassword(String encryptedPassword) {
    SharedPreferences.Editor edit = getSharedPreferences().edit();
    edit.putString(ENCRYPTED_PASS_SHARED_PREF_KEY, encryptedPassword);
    edit.commit();
}

private byte[] getLastIv() {
    SharedPreferences sharedPreferences = getSharedPreferences();
    if (sharedPreferences != null) {
        String ivString = sharedPreferences.getString(LAST_USED_IV_SHARED_PREF_
        KEY, null);

        if (ivString != null) {
            return decodeBytes(ivString);
        }
    }
    return null;
}

private void saveIv(byte[] iv) {
    SharedPreferences.Editor edit = getSharedPreferences().edit();
    String string = encodeBytes(iv);
    edit.putString(LAST_USED_IV_SHARED_PREF_KEY, string);
    edit.commit();
}

private SharedPreferences getSharedPreferences() {
    return context.getSharedPreferences(FINGER_PRINT_HELPER, 0);
}

@RequiresApi(api = Build.VERSION_CODES.M)
private boolean hasPermission() {
    return ActivityCompat.checkSelfPermission(context, Manifest.permission.USE_
```

```

        FINGERPRINT) == PackageManager.PERMISSION_GRANTED;
    }

@RequiresApi(api = Build.VERSION_CODES.M)
public void savePassword(@NonNull String password, CancellationSignal cancellationSignal, Callback callback) {
    authenticate(cancellationSignal, new
        FingerPrintEncryptPasswordListener(callback, password), Cipher.ENCRYPT_MODE);
}

@RequiresApi(api = Build.VERSION_CODES.M)
public void getPassword(CancellationSignal cancellationSignal, Callback callback) {
    authenticate(cancellationSignal, new
        FingerPrintDecryptPasswordListener(callback), Cipher.DECRYPT_MODE);
}

@RequiresApi(api = Build.VERSION_CODES.M)
public boolean encryptPassword(Cipher cipher, String password) {
    try {
        // Зашифровать текст
        if(password.isEmpty()) {
            setError("Пароль пуст");
            return false;
        }

        if (cipher == null) {
            setError("Не удалось создать шифр");
            return false;
        }

        ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
        CipherOutputStream cipherOutputStream = new
            CipherOutputStream(outputStream, cipher);
        byte[] bytes = password.getBytes(Charset.defaultCharset());
        cipherOutputStream.write(bytes);
        cipherOutputStream.flush();
        cipherOutputStream.close();
        saveEncryptedPassword(encodeBytes(outputStream.toByteArray()));

    } catch (Throwable t) {
        setError("Encryption failed " + t.getMessage());
        return false;
    }
}

return true;
}

private byte[] decodeBytes(String s) {
    final int len = s.length();

    // "111" не является допустимой шестнадцатеричной кодировкой.
    if( len%2 != 0 )
        throw new IllegalArgumentException("hexBinary должен быть четной длины:
            "+s);

    byte[] out = new byte[len/2];

```



```

    int l = hexToBin(s.charAt(i+1));
    if( h== -1 || l== -1 )
        throw new IllegalArgumentException("содержит недопустимый символ для
hexBinary: "+s);
    out[i/2] = (byte)(h*16+l);
}
return out;
}

private static int hexToBin( char ch ) {
    if( '0'<=ch && ch<='9' ) return ch-'0';
    if( 'A'<=ch && ch<='F' ) return ch-'A'+10;
    if( 'a'<=ch && ch<='f' ) return ch-'a'+10;
    return -1;
}

private static final char[] hexCode = "0123456789ABCDEF".toCharArray();

public String encodeBytes(byte[] data) {
    StringBuilder r = new StringBuilder(data.length*2);
    for ( byte b : data) {
        r.append(hexCode[(b >> 4) & 0xF]);
        r.append(hexCode[(b & 0xF)]);
    }
    return r.toString();
}

@NotNull
private String decipher(Cipher cipher) throws IOException,
IllegalBlockSizeException, BadPaddingException {
    String retVal = null;
    String savedEncryptedPassword = getSavedEncryptedPassword();
    if (savedEncryptedPassword != null) {
        byte[] decodedPassword = decodeBytes(savedEncryptedPassword);
        CipherInputStream cipherInputStream = new CipherInputStream(new
ByteArrayInputStream(decodedPassword), cipher);

        ArrayList<Byte> values = new ArrayList<>();
        int nextByte;
        while ((nextByte = cipherInputStream.read()) != -1) {
            values.add((byte) nextByte);
        }
        cipherInputStream.close();

        byte[] bytes = new byte[values.size()];
        for (int i = 0; i < values.size(); i++) {
            bytes[i] = values.get(i).byteValue();
        }
        retVal = new String(bytes, Charset.defaultCharset());
    }
    return retVal;
}

private void setError(String error) {
    lastError = error;
    Log.w(FINGER_PRINT_HELPER, lastError);
}

```

```
@RequiresApi(Build.VERSION_CODES.M)
protected class FingerPrintAuthenticationListener extends FingerprintManager.AuthenticationCallback {

    protected final Callback callback;

    public FingerPrintAuthenticationListener(@NonNull Callback callback) {
        this.callback = callback;
    }

    @Override
    public void onAuthenticationError(int errorCode, CharSequence errString) {
        callback.onFailure("Ошибка аутентификации [" + errorCode + "] " + errString);
    }

    /**
     * Вызывается, если при аутентификации возникла восстанавливаемая ошибка.
     * Справочная строка предоставляется для того, чтобы дать пользователю указания,
     * что именно пошло не так, например "Сенсор грязный, пожалуйста, почистите его".
     * @param helpCode – Целое число, идентифицирующее сообщение об ошибке
     * @param helpString – Человекочитаемая строка, которая может быть показана
     * в пользовательском интерфейсе.
     */
    public void onAuthenticationHelp(int helpCode, CharSequence helpString) {
        callback.onHelp(helpCode, helpString.toString());
    }

    /**
     * Вызывается, когда отпечаток пальца распознан.
     * @param result – Объект, содержащий данные, связанные с аутентификацией
     */
    public void onAuthenticationSucceeded(FingerprintManager.AuthenticationResult result) {
    }

    /**
     * Вызывается, когда отпечаток пальца действителен, но не распознан.
     */
    public void onAuthenticationFailed() {
        callback.onFailure("Аутентификация не удалась");
    }

    public @NonNull
    Callback getCallback() {
        return callback;
    }
}

@RequiresApi(api = Build.VERSION_CODES.M)
private class FingerPrintEncryptPasswordListener extends
FingerPrintAuthenticationListener {

    private final String password;

    public FingerPrintEncryptPasswordListener(Callback callback, String password) {
        super(callback);
        this.password = password;
    }
}
```



```
public void onAuthenticationSucceeded(FingerprintManager.AuthenticationResult result) {
    Cipher cipher = result.getCryptoObject().getCipher();
    try {
        if (encryptPassword(cipher, password)) {
            callback.onSuccess("Зашифровано");
        } else {
            callback.onFailure("Шифрование не удалось");
        }
    } catch (Exception e) {
        callback.onFailure("Encryption failed " + e.getMessage());
    }
}

@RequiresApi(Build.VERSION_CODES.M)
protected class FingerPrintDecryptPasswordListener extends FingerPrintAuthenticationListener {

    public FingerPrintDecryptPasswordListener(@NonNull Callback callback) {
        super(callback);
    }

    public void onAuthenticationSucceeded(FingerprintManager.AuthenticationResult result) {
        Cipher cipher = result.getCryptoObject().getCipher();
        try {
            String savedPass = decipher(cipher);
            if (savedPass != null) {
                callback.onSuccess(savedPass);
            } else {
                callback.onFailure("Неудачная расшифровка");
            }
        } catch (Exception e) {
            callback.onFailure("Расшифровка не удалась " + e.getMessage());
        }
    }
}
```

Приведенная ниже активность является очень простым примером получения сохраненного пользователем пароля и взаимодействия с помощником:

```
public class MainActivity extends AppCompatActivity {  
    private TextView passwordTextView;  
    private FingerPrintAuthHelper fingerPrintAuthHelper;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        passwordTextView = (TextView) findViewById(R.id.password);  
        errorTextView = (TextView) findViewById(R.id.error);  
  
        View setPasswordButton = findViewById(R.id.set_password_button);  
        setPasswordButton.setOnClickListener(new View.OnClickListener() {  
            @Override  
            public void onClick(View v) {
```

```

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
            fingerPrintAuthHelper.savePassword(passwordTextView.getText().
                toString(), new CancellationSignal(), getAuthListener(false));
        }
    });
};

View getPasswordButton = findViewById(R.id.get_password_button);
getPasswordButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
            fingerPrintAuthHelper.getPassword(new CancellationSignal(),
                getAuthListener(true));
        }
    }
});

// Запуск помощника по работе с отпечатками пальцев. В случае неудачи покажите
// пользователю ошибку
private void startFingerPrintAuthHelper() {
    fingerPrintAuthHelper = new FingerPrintAuthHelper(this);
    if (!fingerPrintAuthHelper.init()) {
        errorTextView.setText(fingerPrintAuthHelper.getLastErrorMessage());
    }
}

@NonNull
private FingerPrintAuthHelper.Callback getAuthListener(final boolean isGetPass) {
    return new FingerPrintAuthHelper.Callback() {
        @Override
        public void onSuccess(String result) {
            if (isGetPass) {
                errorTextView.setText("Успех!!! Pass = " + result);
            } else {
                errorTextView.setText("Зашифрованный пасс = " + result);
            }
        }

        @Override
        protected void onActivityResult(int requestCode, int resultCode, Intent data) {
            if (requestCode == RESULT_CODE) {
                @Override
                public void onFailure(String message) {
                    errorTextView.setText("Failed - " + message);
                }

                @Override
                public void onHelp(int helpCode, String helpString) {
                    errorTextView.setText("Нужна помощь - " + helpString);
                }
            }
        }
    };
}
}

```

## 55.2. Добавление сканера отпечатков пальцев в приложение

Чтобы использовать эту возможность в своем приложении, сначала добавьте разрешение USE\_FINGERPRINT в манифест:

```
<uses-permission
    android:name="android.permission.USE_FINGERPRINT" />
```

Ниже приводится порядок действий:

Сначала необходимо создать симметричный ключ в хранилище ключей Android с помощью KeyGenerator, который может быть использован только после аутентификации пользователя с помощью отпечатка пальца и передачи KeyGenParameterSpec.

```
KeyPairGenerator.getInstance(KeyProperties.KEY_ALGORITHM_EC, "AndroidKeyStore");
keyPairGenerator.initialize(
    new KeyGenParameterSpec.Builder(KEY_NAME, KeyProperties.PURPOSE_SIGN)
        .setDigests(KeyProperties.DIGEST_SHA256)
        .setAlgorithmParameterSpec(new ECGenParameterSpec("secp256r1"))
        .setUserAuthenticationRequired(true)
        .build());
keyPairGenerator.generateKeyPair();
```

Установив значение KeyGenParameterSpec.Builder.setUserAuthenticationRequired в true, можно разрешить использование ключа только после аутентификации пользователя, в том числе при аутентификации по отпечатку пальца.

```
KeyStore keyStore = KeyStore.getInstance("AndroidKeyStore");
keyStore.load(null);
PublicKey publicKey = keyStore.getCertificate(MainActivity.KEY_NAME).getPublicKey();

KeyStore keyStore = KeyStore.getInstance("AndroidKeyStore");
keyStore.load(null);
PrivateKey key = (PrivateKey) keyStore.getKey(KEY_NAME, null);
```

Затем запустите прослушивание отпечатка пальца на дактилоскопическом датчике, вызвав FingerprintManager.authenticate с шифром, инициализированным созданным симметричным ключом. В качестве альтернативы можно вернуться к проверенному паролю на стороне сервера в качестве аутентификатора.

Создайте и инициализируйте FingerprintManager из fingerprintManger.class getContext().getSystemService(FingerprintManager.class). Для аутентификации используйте FingerprintManger API и создайте подкласс, используя FingerprintManager.AuthenticationCallback, и переопределите методы:

```
onAuthenticationError
onAuthenticationHelp
onAuthenticationSucceeded
onAuthenticationFailed
```

## Начало

Чтобы начать прослушивание события fingerPrint, вызовите метод authenticate с криптографией:

```
fingerprintManager
    .authenticate(cryptoObject, mCancellationSignal, 0, this, null);
```

## Отмена

Для прекращения прослушивания вызова сканера:

```
android.os.CancellationSignal;
```

После проверки отпечатка пальца (или пароля) осуществляется обратный вызов FingerprintManager.AuthenticationCallback#onAuthenticationSucceeded().

```
@Override
```

```
public void onAuthenticationSucceeded(AuthenticationResult result) { }
```

# Глава 56. Bluetooth и Bluetooth Low Energy API

## 56.1. Разрешения

Добавьте это разрешение в файл манифеста для использования функций Bluetooth в вашем приложении:

```
<uses-permission android:name="android.permission.BLUETOOTH" />
```

Если необходимо инициировать обнаружение устройства или манипулировать настройками Bluetooth, то также необходимо добавить это разрешение:

```
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
```

Для работы с Android API уровня 23 и выше потребуется доступ к местоположению:

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<!-- ИЛИ -->
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

## 56.2. Проверка включения Bluetooth

```
private static final int REQUEST_ENABLE_BT = 1; // Уникальный код запроса
BluetoothAdapter mBluetoothAdapter;

// ...
if (!mBluetoothAdapter.isEnabled()) {
    Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
    startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
}
// ...

@Override
protected void onActivityResult(final int requestCode, final int resultCode, final
Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    if (requestCode == REQUEST_ENABLE_BT) {
        if (resultCode == RESULT_OK) {
            // Bluetooth был включен
        } else if (resultCode == RESULT_CANCELED) {
            // Bluetooth не был включен
        }
    }
}
```

## 56.3. Поиск ближайших устройств Bluetooth Low Energy

API BluetoothLE впервые был представлен в API 18. Однако в API 21 изменился способ сканирования устройств. Поиск устройств должен начинаться с определения UUID службы (<https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt#services>), который будет сканироваться (либо официально принятые 16-битные UUID, либо проприетарные). В данном примере показано, как сделать независимый от API способ поиска Bluetooth Low Energy (BLE)-устройств:

### 1. Создание модели устройства Bluetooth:

```
public class BTDevice {
    String address;
    String name;

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

### 2. Определение интерфейса сканирования Bluetooth:

```
public interface ScanningAdapter {
    void startScanning(String name, String[] uuids);
    void stopScanning();
    List<BTDevice> getFoundDeviceList();
}
```

### 3. Создание класса фабрики сканирования:

```
public class BluetoothScanningFactory implements ScanningAdapter {

    private ScanningAdapter mScanningAdapter;

    public BluetoothScanningFactory() {
        if (isNewerAPI()) {
            mScanningAdapter = new LollipopBluetoothLEScanAdapter();
        } else {
            mScanningAdapter = new JellyBeanBluetoothLEScanAdapter();
        }
    }

    private boolean isNewerAPI() {
        return Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP;
    }

    @Override
    public void startScanning(String[] uuids) {
        mScanningAdapter.startScanning(uuids);
    }

    @Override
    public void stopScanning() {
```

```

        mScanningAdapter.stopScanning();
    }
    @Override
    public List<BTDevice> getFoundDeviceList() {
        return mScanningAdapter.getFoundDeviceList();
    }
}
}

```

#### 4. Создание фабричной реализации:

API 21+:

```

import android.annotation.TargetApi;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.le.BluetoothLeScanner;
import android.bluetooth.le.ScanFilter;
import android.bluetooth.le.ScanResult;
import android.bluetooth.le.ScanSettings;
import android.os.Build;
import android.os.ParcelUuid;
import bluetooth.model.BTDevice;
import java.util.ArrayList;
import java.util.List;

@TargetApi(Build.VERSION_CODES.LOLLIPOP)
public class LollipopBluetoothLEScanAdapter implements ScanningAdapter {
    BluetoothLeScanner bluetoothLeScanner;
    ScanCallback mCallback;
    List<BTDevice> mBluetoothDeviceList;
    public LollipopBluetoothLEScanAdapter() {
        bluetoothLeScanner = BluetoothAdapter.getDefaultAdapter().getBluetoothLeScanner();
        mCallback = new ScanCallback();
        mBluetoothDeviceList = new ArrayList<>();
    }

    @Override
    public void startScanning(String[] uids) {
        if (uids == null || uids.length == 0) {
            return;
        }
        List<ScanFilter> filterList = createScanFilterList(uids);
        ScanSettings scanSettings = createScanSettings();
        bluetoothLeScanner.startScan(filterList, scanSettings, mCallback);
    }

    private List<ScanFilter> createScanFilterList(String[] uids) {
        List<ScanFilter> filterList = new ArrayList<>();
        for (String uuid : uids) {
            ScanFilter filter = new ScanFilter.Builder()
                .setServiceUuid(ParcelUuid.fromString(uuid))
                .build();
            filterList.add(filter);
        }
        return filterList;
    }
}

```

```

private ScanSettings createScanSettings() {
    ScanSettings settings = new ScanSettings.Builder()
        .setScanMode(ScanSettings.SCAN_MODE_BALANCED)
        .build();
    return settings;
}

@Override
public void stopScanning() {
    bluetoothLeScanner.stopScan(mCallback);
}

@Override
public List<BTDevice> getFoundDeviceList() {
    return mBluetoothDeviceList;
}

public class ScanCallback extends android.bluetooth.le.ScanCallback {

    @Override
    public void onScanResult(int callbackType, ScanResult result) {
        super.onScanResult(callbackType, result);
        if (result == null) {
            return;
        }
        BTDevice device = new BTDevice(); device.setAddress(result.getDevice().getAddress());
        device.setName(new StringBuffer(result.getScanRecord().getDeviceName()).toString());
        if (device == null || device.getAddress() == null) {
            return;
        }
        if (isAlreadyAdded(device)) {
            return;
        }
        mBluetoothDeviceList.add(device);
    }

    private boolean isAlreadyAdded(BTDevice bluetoothDevice) {
        for (BTDevice device : mBluetoothDeviceList) {
            String alreadyAddedDeviceMACAddress = device.getAddress();
            String newDeviceMACAddress = bluetoothDevice.getAddress();
            if (alreadyAddedDeviceMACAddress.equals(newDeviceMACAddress)) {
                return true;
            }
        }
        return false;
    }
}
}

```

## 5. Получение списка найденных устройств путем вызова:

```
scanningFactory.startScanning({uuidlist});
```

ожидание в течение нескольких секунд...

```
List<BTDevice> bluetoothDeviceList = scanningFactory.getFoundDeviceList();
```

## 56.4. Сделать устройство обнаруживаемым

```

private static final int REQUEST_DISCOVERABLE_BT = 2; // Уникальный код запроса
private static final int DISCOVERABLE_DURATION = 120; // Время длительности
// обнаружения в секундах
// 0 - всегда можно обнаружить
// максимальное значение 3600
private final BroadcastReceiver receiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if (BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE.equals(action)) {
            Intent discoverableIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
            discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION,
                DISCOVERABLE_DURATION);
            startActivityForResult(discoverableIntent, REQUEST_DISCOVERABLE_BT);
        }
    }
}
@Override
protected void onActivityResult(final int requestCode, final int resultCode, final Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == REQUEST_DISCOVERABLE_BT) {
        if (resultCode == RESULT_OK) {
            // Устройство доступно для обнаружения
        } else if (resultCode == RESULT_CANCELED) {
            // Устройство не обнаруживается
        }
    }
}

```

## 56.5. Подключение к устройству Bluetooth

После получения доступа к Bluetooth-устройству с ним можно взаимодействовать. Такое взаимодействие осуществляется с помощью потоков ввода-вывода сокетов.

Перечислим основные шаги по установлению связи через Bluetooth.

### 1. Инициализация сокета:

```

private BluetoothSocket _socket;
//...
public InitializeSocket(BluetoothDevice device){
    try {
        _socket = device.createRfcommSocketToServiceRecord(<Your app UDID>);
    } catch (IOException e) {
        //Ошибка
    }
}

```

### 2. Подключение к сокету:

```

try {
    _socket.connect();
} catch (IOException connEx) {
    try {
        _socket.close();
    } catch (IOException closeException) {

```

```

    //Ошибка
}
}

if (_socket != null && _socket.isConnected()) {
    //Сокет подключен, теперь мы можем получить наши потоки ввода/вывода
}

```

### 3. Получение потоков ввода/вывода сокетов:

```

private InputStream _inStream;
private OutputStream _outStream;
//...
try {
    _inStream = _socket.getInputStream();
    _outStream = _socket.getOutputStream();
} catch (IOException e) {
    //Ошибка
}

```

**Входной поток** – используется как канал входящих данных (прием данных от подключенного устройства).

**Выходной поток** – используется как исходящий канал данных (отправка данных на подключенное устройство).

После завершения 3-го шага мы можем принимать и отправлять данные между двумя устройствами, используя ранее инициализированные потоки.

#### 1. Получение данных (чтение из входного потока сокета):

```

byte[] buffer = new byte[1024]; // буфер (наши данные)
int bytesCount; // количество прочитанных байт

```

```

while (true) {
    try {
        // чтение данных из входного потока
        bytesCount = _inStream.read(buffer);
        if (buffer != null && bytesCount > 0) {
            // Разбор полученных байтов
        }
    } catch (IOException e) {
        // Ошибка
    }
}

```

#### 2. Отправка данных (запись в выходной поток):

```

public void write(byte[] bytes) {
    try {
        _outStream.write(bytes);
    } catch (IOException e) {
        // Ошибка
    }
}

```

- Разумеется, функции подключения, чтения и записи должны выполняться в выделенном потоке.

- Должны находиться объекты Sockets и Stream.

## 56.6. Поиск ближайших Bluetooth-устройств

Сначала объявите BluetoothAdapter:

```
BluetoothAdapter mBluetoothAdapter;
```

Теперь создайте приемник BroadcastReceiver для ACTION\_FOUND:

```
private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        //Устройство найдено
        if (BluetoothDevice.ACTION_FOUND.equals(action)) {
            // Получение объекта BluetoothDevice из намерения
            BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
            // Добавить имя и адрес в массив адаптера для отображения в списке
            mArrayAdapter.add(device.getName() + "\n" + device.getAddress());
        }
    }
};
```

Зарегистрируйте приемник вещания:

```
IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
registerReceiver(mReceiver, filter);
```

Затем запустите обнаружение близлежащих Bluetooth-устройств, вызвав startDiscovery:

```
mBluetoothAdapter.startDiscovery();
```

Не забудьте снять с регистрации BroadcastReceiver внутри onDestroy:

```
unregisterReceiver(mReceiver);
```

## Глава 57. Разрешения во время выполнения

Начиная с Marshmallow, в Android введена модель разрешений во время выполнения (Runtime Permissions, см. <https://developer.android.com/training/permissions/requesting.html>). Разрешения делятся на две категории – обычные (Normal) и опасные (Dangerous) (об этом см. <https://developer.android.com/guide/topics/permissions/requesting.html#normal-dangerous>), причем опасные разрешения (см. <https://developer.android.com/guide/topics/permissions/requesting.html#perm-groups>) теперь предоставляются пользователем во время выполнения программы.

### 57.1. Множественные разрешения

```
public static final int MULTIPLE_PERMISSIONS = 10; // код по желанию
String[] permissions = new String[] {
    Manifest.permission.WRITE_EXTERNAL_STORAGE,
```

```

Manifest.permission.CAMERA,
Manifest.permission.ACCESS_COARSE_LOCATION,
Manifest.permission.ACCESS_FINE_LOCATION
};

@Override
void onStart() {
    if (checkPermissions()) {
        // разрешения выданы.
    } else {
        // показать диалог, информирующий об отсутствии определенных разрешений
    }
}

private boolean checkPermissions() {
    int result;
    List<String> listPermissionsNeeded = new ArrayList<>();
    for (String p:permissions) {
        result = ContextCompat.checkSelfPermission(getActivity(),p);
        if (result != PackageManager.PERMISSION_GRANTED) {
            listPermissionsNeeded.add(p);
        }
    }
    if (!listPermissionsNeeded.isEmpty()) {
        ActivityCompat.requestPermissions(this, listPermissionsNeeded.toArray(new
String[listPermissionsNeeded.size()]), MULTIPLE_PERMISSIONS);
        return false;
    }
    return true;
}
@Override
public void onRequestPermissionsResult(int requestCode, String permissions[], int[]
grantResults) {
    switch (requestCode) {
        case MULTIPLE_PERMISSIONS:
            if(grantResults.length > 0 && grantResults[0] == PackageManager.
PERMISSION_GRANTED){
                // разрешения выданы.
            } else {
                // разрешения не выданы.
            }
            return;
    }
}
}

```

## 57.2. Несколько разрешений на выполнение из одних и тех же групп разрешений

В манифесте мы имеем четыре опасных разрешения на выполнение из двух групп:

```

<!-- Требуется для чтения и записи в файл shredPref. -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>

<!-- Требуется для получения местоположения устройства. -->

```