Хранение данных

Хранение данных: Shared preferences; Internal Storage и External Storage.

[Хранение данных](#_2et92p0)

[Способы хранения данных](#_tyjcwt)

[Shared Preferences](#_3dy6vkm)

[Практика](#_1t3h5sf)

[Задача 1. Дать возможность сохранять и просматривать Shared Preferences.](#_4d34og8)

[Выбор между External storage и Internal storage](#_17dp8vu)

[Практика](#_3rdcrjn)

[Задача 2. Сохранение, загрузка, удаление текстового файла в Internal Storage.](#_26in1rg)

[Задача 3. Сохранение, загрузка, удаление текстового файла в External Storage.](#_lnxbz9)

[Домашнее задание](#_35nkun2)

[Дополнительные материалы](#_1ksv4uv)

[Используемая литература](#_2jxsxqh)

# 

# Хранение данных

## Способы хранения данных

Android предоставляет несколько различных возможностей для хранения данных приложения. В зависимости от задачи можно выбрать наиболее удобный способ. Существуют следующие виды хранения данных, предусмотренные для Android-устройств:

* Shared Preferences — приватное хранилище, где информация имеет вид «ключ-значение».
* Internal Storage — приватное хранилище, где данные находятся в памяти устройства.
* External Storage — хранилище с общим доступом во внешнем хранилище.
* SQLite — базы данных.
* Хранение данных на сервере и загрузка их через сеть.

## Shared Preferences

Shared Preferences используется в тех случаях, когда нужно хранить данные в виде «ключ-значение» и их объем невелик. Объект SharedPreferences представляет собой указатель на файл и методы чтения/записи из него.

Shared Preferences бывают двух видов:

1. Общие — можно создавать файлы с различными именами, доступ к ним будет из любого context-a в приложении. Используются, если может возникнуть необходимость пользоваться ими из любой части приложения.
2. Локальные — доступ к ним будет только из одной аctivity (название формируется из имени класса). Используются, когда доступ к ним может потребоваться только на текущей аctivity.

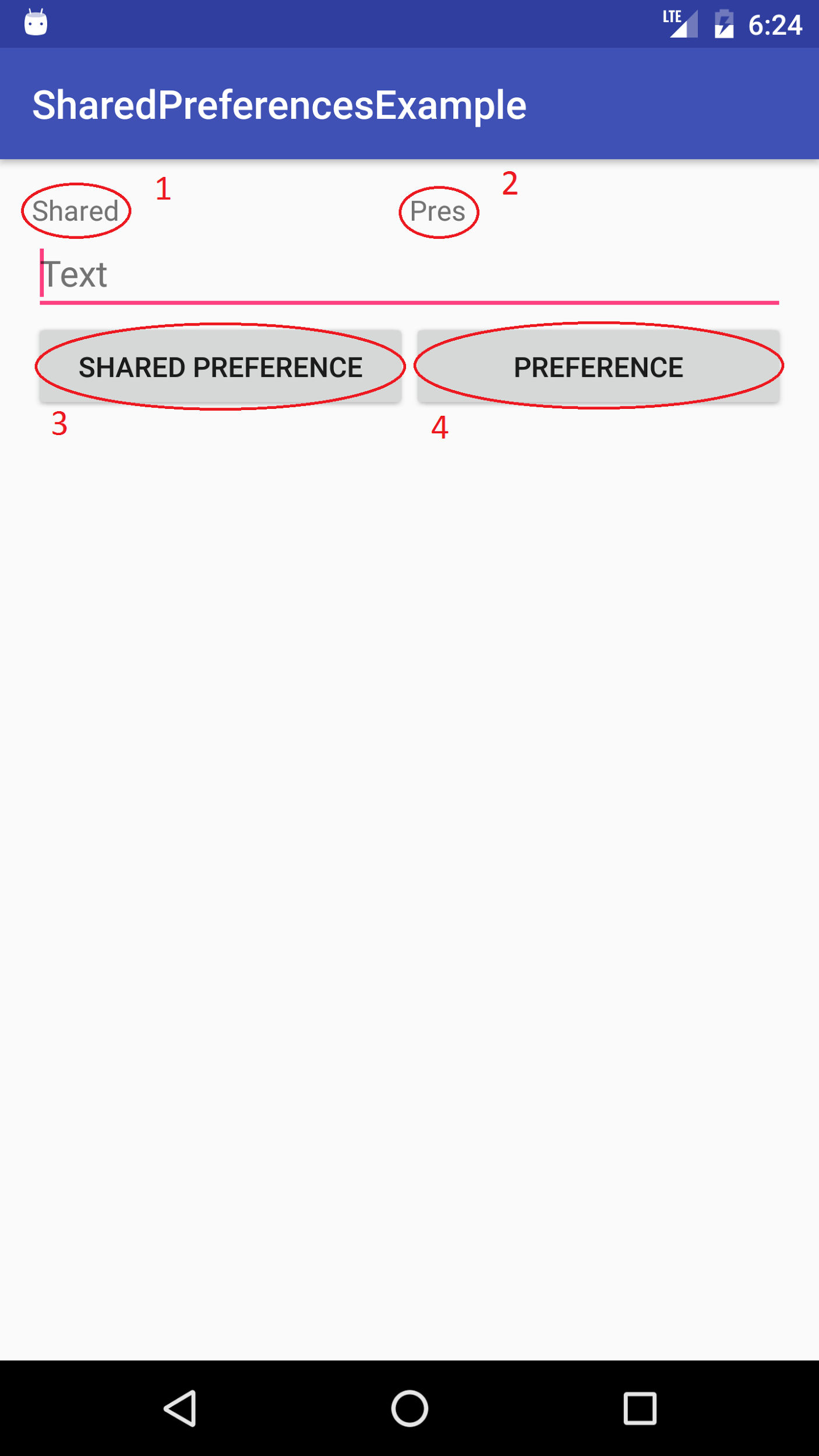
# Практика

## Задача 1. Дать возможность сохранять и просматривать Shared Preferences.

Проект с этой задачей называется SharedPreferencesExample, его можно найти в папке урока.

Хотим получить следующее:

1. TextView, куда загружаем ключ из общего context-а приложения.
2. TextView, куда загружаем ключ из context-а текущей activity.
3. Button, сохраняющий в context приложения.
4. Button, сохраняющий context текущей activity.
5. EditText, значения из которого сохраняем.



Создадим макет для такого представления (activity\_main.xml):

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:tools="http://schemas.android.com/tools"  android:id="@+id/activity\_main"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:paddingBottom="@dimen/activity\_vertical\_margin"  android:paddingLeft="@dimen/activity\_horizontal\_margin"  android:paddingRight="@dimen/activity\_horizontal\_margin"  android:paddingTop="@dimen/activity\_vertical\_margin"  android:orientation="vertical"  tools:context="com.geekbrains.sharedpreferencesexample.MainActivity">  <LinearLayout  android:orientation="horizontal"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content">  <TextView  android:text="@string/shared\_pref"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/textViewSharedPref"  android:layout\_weight="1" />  <TextView  android:text="@string/pref"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/textViewPref"  android:layout\_weight="1" />  </LinearLayout>  <EditText  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:inputType="textPersonName"  android:ems="10"  android:hint="@string/text"  android:id="@+id/editText" />  <LinearLayout  android:orientation="horizontal"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content">  <Button  android:text="@string/shared\_pref"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/buttonSharedPref"  android:layout\_weight="1" />  <Button  android:text="@string/pref"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/buttonPref"  android:layout\_weight="1" />  </LinearLayout>  </LinearLayout> |

Для сохранения в context приложения делаем следующее:

Пишем метод, который сохраняет данные из EditText в SharedPreferences, и в нем пишем:

|  |
| --- |
| 1. SharedPreferences sharedPref = getSharedPreferences("TestPreferences", 2. Context.MODE\_PRIVATE); 3. EditText editText = (EditText) findViewById(R.id.editText); 4. if((editText.getText().toString() != null) && 5. !(editText.getText().toString().isEmpty())) { 6. SharedPreferences.Editor editor = sharedPref.edit(); 7. editor.putString(getString(R.string.shared\_prefs\_key\_text), 8. editText.getText().toString()); 9. editor.commit(); 10. } 11. editText.setText(""); |

Получаем доступ к файлу настроек (стр. 2-3), используем функцию getSharedPreferences (String name, int mode), где:

* name — имя файла настроек, у нас это TestPreferences;
* mode — режим, в котором мы подключаемся к файлу. Существует 3 основных режима: MODE\_PRIVATE — доступ к файлу будет только у нашего приложения. MODE\_WORLD\_READABLE и MODE\_WORLD\_WRITABLE — данные будут доступны другим приложениям, которым известен идентификатор файла, соответственно для чтения и записи.

Получаем доступ к EditText и, если в него что-то введено, начинаем редактирование (стр.4-8).

Используя метод [putString](https://developer.android.com/reference/android/content/SharedPreferences.Editor.html#putString(java.lang.String,%20java.lang.String)) (String key, String value)(стр.9-10), сохраняем значение из EditText в Shared Preferences. Значение ключа в данном случае храним в файле локализации.

Вызов функции commit() сохраняет произведенное выше редактирование в файл.

В конце очищаем поле ввода — исключительно для удобства.

Метод сохранения в context текущей аctivity отличается только строкой инициализации SharedPreferences, поэтому остальное выносим в отдельный метод saveToSharedPreferences():

|  |
| --- |
| SharedPreferences pref = getPreferences(Context.MODE\_PRIVATE);  saveToSharedPreferences(pref); |

Как видно из реализации, метод getPreferences() принимает только один параметр — режим подключения, такой же как и у метода getSharedPreferences().

Для проверки результата загрузим в наши TextView оба сохраняемые значения. Сначала загрузка из общего context-а приложения:

|  |
| --- |
| private void loadSharedPrefs() {  SharedPreferences sharedPref = getSharedPreferences("TestPreferences",  Context.MODE\_PRIVATE);  String loadedPrefs =  sharedPref.getString(getString(R.string.shared\_prefs\_key\_text),  getString(R.string.pref));  TextView textViewPrefs = (TextView) findViewById(R.id.textViewSharedPref);  textViewPrefs.setText(loadedPrefs);  } |

Получаем доступ к файлу настроек также, как было описано ранее. Затем используем метод [getString](https://developer.android.com/reference/android/content/SharedPreferences.html#getString(java.lang.String,%20java.lang.String)) (String key, String defValue), он загружает значение по ключу — key. Если оно не обнаружено, используется значение defValue.

Записываем в TextView полученное значение.

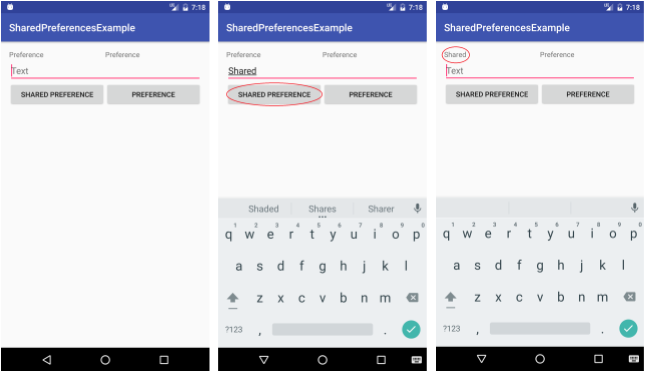
Затем загружаем из context-а текущей activity:

|  |
| --- |
| private void loadPrefs() {  SharedPreferences sharedPref = getPreferences(Context.MODE\_PRIVATE);  String loadedPrefs =  sharedPref.getString(getString(R.string.shared\_prefs\_key\_text),  getString(R.string.pref));  TextView textViewPrefs = (TextView) findViewById(R.id.textViewPref);  textViewPrefs.setText(loadedPrefs);  } |

Это очень похоже на то, как мы это делали для Shared Preferences общего context-а.

Теперь, чтобы наши TextView всегда показывали реальные значения, добавим метод loadSharedPrefs() в onCreate() и метод loadPrefs().

При чистой установке (после удаления или если приложение не было установлено) все SharedPreferences пусты. Следовательно в TextView будет записано значение, по умолчанию указанное в sharedPref.getString(). Именно так и покажем наши результаты:



## Screen Shot 2017-11-01 at 21.42.28.png

Не забывайте вешать на ваши кнопки слушателей событий (onClickListener) и, при нажатии на кнопку, получать значения из SharedPreferences, и вызывать методы для их сохранения в SharedPreferences.

# Выбор между External storage и Internal storage

Все устройства Android имеют две области хранения файлов: внутреннюю память и внешние хранилища. Эти области появились в первые годы существования Android, когда на большинстве устройств имелись встроенная память (внутреннее хранилище) и карты памяти (например, micro SD, внешнее хранилище). Некоторые устройства делят встроенную память на внутренний и внешний разделы, так что даже без съемных носителей в системе две области хранения файлов. И API-интерфейс работает одинаково, независимо от типа внешнего хранилища.

Внутренняя память:

* Всегда доступна.
* Сохраненные здесь файлы по умолчанию доступны только вашему приложению.
* При удалении пользователем вашего приложения система Android удаляет из внутренней памяти все файлы этого приложения.

Внутренняя память лучше всего подходит для ситуаций, когда вы хотите быть уверены, что ни пользователь, ни другие приложения не смогут получить доступ к вашим файлам.

Внешнее хранилище:

* Доступно не всегда, потому что пользователь может в любое время подключать и отключать такие хранилища, например, USB-накопители.
* Такие хранилища доступны для чтения везде, поэтому вы не контролируете чтение сохраненных в них данных.
* При удалении пользователем вашего приложения система Android удаляет из внешних хранилищ файлы этого приложения, только если они сохраняются в директории из getExternalFilesDir().

Внешнее хранилище лучше всего подходит для файлов без ограничений доступа и для файлов, которые вы хотите сделать доступными другим приложениям или пользователю через компьютер.

# Практика

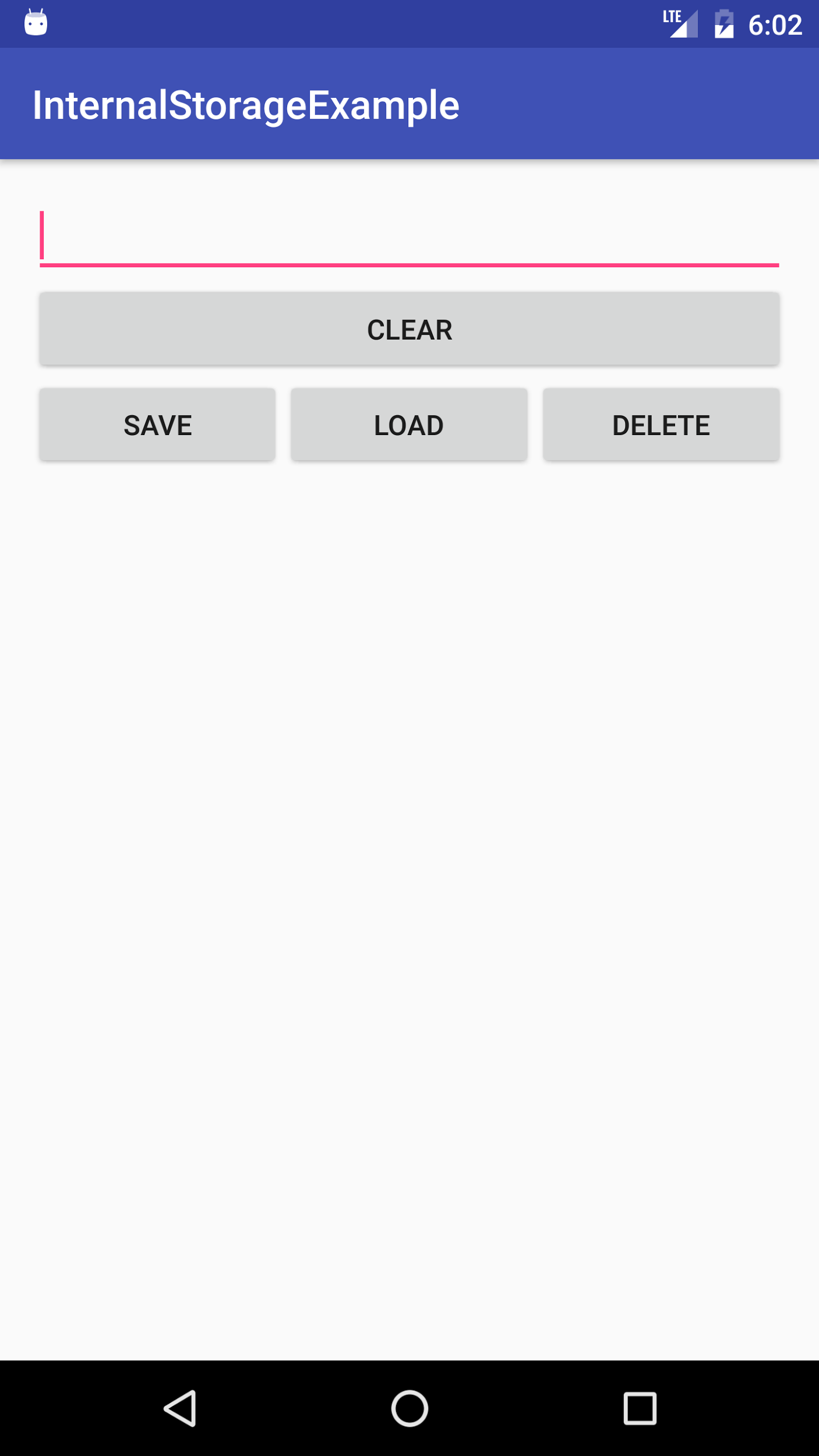
## Задача 2. Сохранение, загрузка, удаление текстового файла в Internal Storage.

Проект с этой задачей называется InternalStorageExample, его можно найти в папке урока.

В данном примере используется работа с файлами, в реальных приложениях всю работу с ними необходимо делать в отдельном потоке. Из-за того, что чтение из файла и запись в него занимают неопределенное количество времени, если это делать в основном потоке приложения, то это может остановить всю работу приложения, пока операция с файлом не будет завершена. Так делать опасно. Пользователю не понравится сидеть и смотреть на экран зависшего приложения. Поэтому все операции с файлами выполняются в отдельном потоке.

Из-за того, что работа с потоками еще не изучалась, все сделано в основном потоке приложения (наиболее простым способом).

Простейший интерфейс для текущей задачи выглядит следующим образом:



* Кнопка CLEAR очищает поле ввода.
* SAVE — сохранение в файл. В данном случае он будет иметь имя «savedText.txt».
* LOAD — загрузка этого файла в поле ввода.
* DELETE — удаление файла.

Реализуя действия кнопки CLEAR, мы просто очистим текст в нашем TextView:

|  |
| --- |
| private View.OnClickListener clearTextListener = new View.OnClickListener() {  @Override  public void onClick(View view) {  editText.setText("");  }  }; |

SAVE выглядит следующим образом:

|  |
| --- |
| private View.OnClickListener saveTextListener = new View.OnClickListener() {  @Override  public void onClick(View view) {  FileOutputStream outputStream;  try {  if (!editText.getText().toString().isEmpty()) {  outputStream = openFileOutput(FILENAME, Context.MODE\_PRIVATE);  outputStream.write(editText.getText().toString().getBytes());  outputStream.flush();  outputStream.close();  }  } catch (Exception e) {  e.printStackTrace();  }  }  }; |

Класс FileOutputStream содержит функционал, используемый для записи в файл. Если в EditText был введен текст, то первым делом происходит инициализация файла openFileOutput(String name, int mode), где name — имя файла, mode — режим, его значения были описаны в примере 1. Этот метод производит открытие файла, а если такого файла нет, то создает его.

Далее write() записывает значение EditText в файл,; flush() — проверяет, закончена ли запись, и, если нет, ждет, пока она не закончится; close() — закрывает файл, если этого не сделать при следующем обращении к нему, может возникнуть Exception.

LOAD выглядит немного сложнее:

|  |
| --- |
| private View.OnClickListener loadTextListener = new View.OnClickListener() {  @Override  public void onClick(View view) {  FileInputStream inputStream;  try {  StringBuilder text = new StringBuilder();  // getFilesDir() возвращает путь к папке нашего приложения  File file = new File(getApplicationContext().getFilesDir(),FILENAME);  if(!file.exists()) {  showToast(getText(R.string.toast\_file\_not\_exist));  return;  }  BufferedReader br = new BufferedReader(new FileReader(file));  String line;  // есть текст в br считываем его построчно  while ((line = br.readLine()) != null) {  // добавляем считанное в StringBuilder  text.append(line);  // перенос на новую строку не считывается поэтому добавляем его  text.append('\n');  }  // последний добавленный перенос лишний, убираем его  text.deleteCharAt(text.length() - 1);  br.close();  editText.setText(text);  }  catch (Exception e) {  e.printStackTrace();  }  }  }; |

Для получения доступа к файлу мы используем конструктор [File](https://developer.android.com/reference/java/io/File.html#File(java.io.File,%20java.lang.String)) (File parent, String child), где parent — путь, где расположен файл, child — имя файла. Для получения пути мы используем метод getFilesDir(), он возвращает File, соответствующий внутренней директории приложения. Также можно воспользоваться методом getCacheDir(), но, как следует из названия, храните там только временные файлы. Если системе понадобится память, она может удалить эти файлы, поэтому ею следует пользоваться с осторожностью.

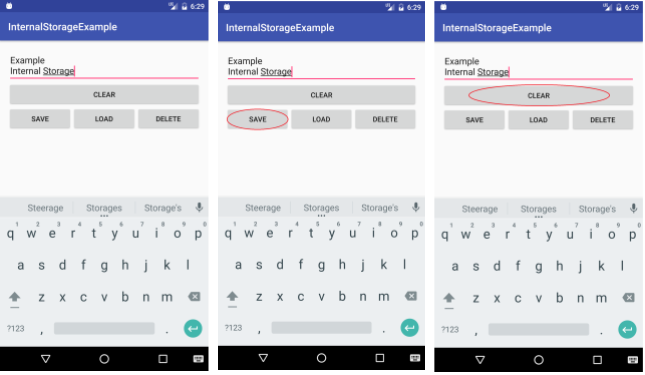
Если файл не существует, то показываем Toast с соответствующим сообщением и завершаем выполнение метода.

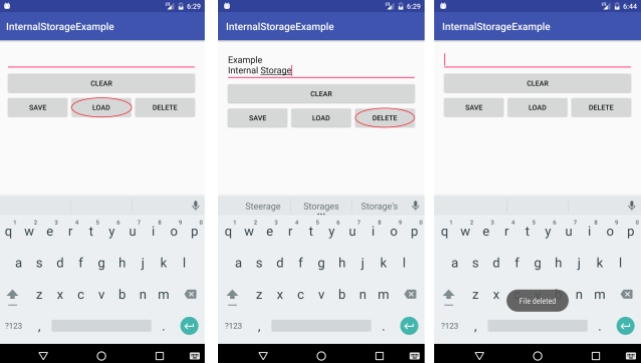
Иначе читаем из файла текст и записываем его в EditText.

DELETE

|  |
| --- |
| private View.OnClickListener deleteFileListener = new View.OnClickListener() {  @Override  public void onClick(View view) {  File file = new File(getApplicationContext().getFilesDir(),FILENAME);  if (file.exists()) {  file.delete();  showToast(getText(R.string.toast\_file\_deleted));  clearTextListener.onClick(view);  }  else {  showToast(getText(R.string.toast\_file\_not\_exist));  }  }  }; |

Получив доступ к файлу через конструктор File(), мы, чтобы избежать ошибок, проверяем, существует ли он. Если файл существует, метод delete() удаляет файл. Показываем Toast, что файл удален, и очищаем EditText. Иначе показываем Toast, что файл не существует.



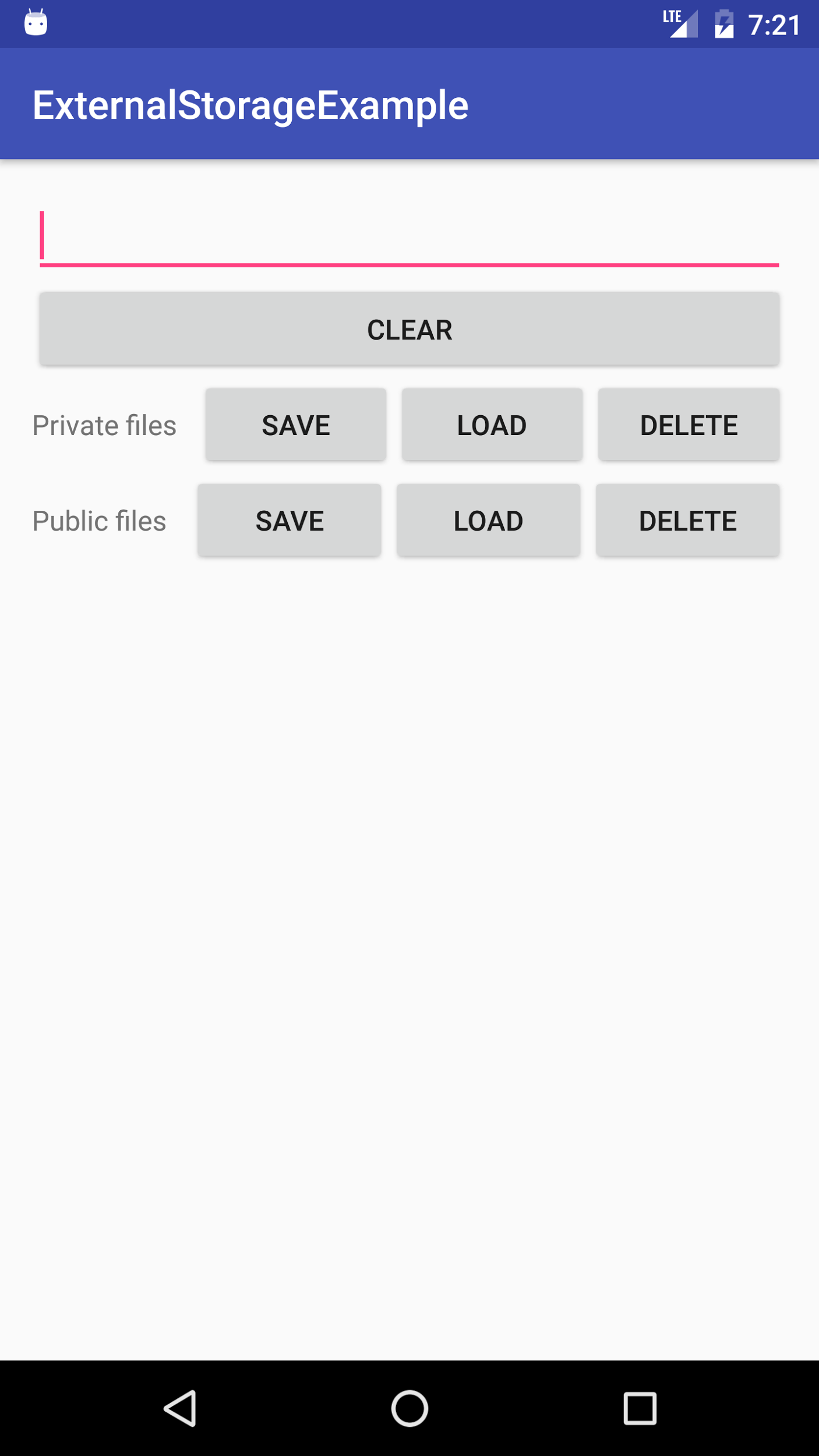


## Задача 3. Сохранение, загрузка, удаление текстового файла в External Storage.

Проект с этой задачей называется ExternalStorageExample, его можно найти в папке урока.

В данном примере используется работа с файлами в главном потоке, что является потоконебезопасно, — так делать в настоящих приложениях нельзя. В реальных приложениях всю работу с файлами необходимо совершать в отдельном потоке. Из-за того, что работа с потоками еще не изучалась, все сделано в основном потоке приложения (наиболее простым способом).

Простейший интерфейс для текущей задачи выглядит очень похоже на интерфейс из задачи 2:



Кнопка CLEAR очищает поле ввода.

Далее идут 2 горизонтальные группы кнопок со схожей функциональностью, но находящимися в разных файлах:

* Private files — относятся к папке приложения на внешнем носителе.
* Public files — относятся к общей папке на внешнем носителе.
* SAVE — сохранение в файл, в данном случае он будет иметь имя «savedText.txt».
* LOAD — загрузка этого файла в поле ввода.
* DELETE — удаление файла.

Первым делом нужно добавить разрешение на доступ к External Storage в манифест (AndroidManifest.xml), добавляем строку с тегом uses-permission:

|  |
| --- |
| <manifest xmlns:android="http://schemas.android.com/apk/res/android"  package="com.geekbrains.externalstorageexample">  <application  ...  </application>  <uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" />  </manifest> |

* android.permission.WRITE\_EXTERNAL\_STORAGE — означает, что мы можем писать и читать из External Storage.

Сохранение будет в 3-х функциях. Сначала в общей папке — в нашем случае это Environment.DIRECTORY\_DOCUMENTS :

|  |
| --- |
| private View.OnClickListener savePublicListener = new View.OnClickListener() {  @Override  public void onClick(View view) {  File file = new File(getApplicationContext().getExternalFilesDir(Environment.DIRECTORY\_DOCUMENTS), FILENAME);  onSaveText(file);  }  }; |

Получаем доступ к файлу, используя конструктор File(). Для определения директории используем метод getExternalFilesDir(), передавая ему Environment.DIRECTORY\_DOCUMENTS. Получаем путь к папке общей документации на устройстве.

Функция onSaveText(), в которой помещен код, одинаковый для обоих методов:

|  |
| --- |
| public void onSaveText(File file) {  if (!isExternalStorageWritable()) {  showToast(getText(R.string.toast\_external\_storage\_not\_found));  return;  }  try {  if (!editText.getText().toString().isEmpty()) {  FileOutputStream outputStream = new FileOutputStream(file, false);  outputStream.write(editText.getText().toString().getBytes());  outputStream.flush();  outputStream.close();  }  } catch (Exception e) {  showToast(e.getMessage());  }  } |

Проверяем доступность External Storage для записи. Если недоступно, сообщаем об этом в Toast и прерываем метод. Если External Storage доступен и в EditText введен текст, сохраняем текст в файл.

И, наконец, функция, сохраняющая все в папке приложения:

|  |
| --- |
| private View.OnClickListener savePrivateListener = new View.OnClickListener() {  @Override  public void onClick(View view) {  File dir = new  File(getApplicationContext().getExternalFilesDir(null).getPath());  if (!dir.exists()) {  dir.mkdirs();  }  File file = new File(dir, FILENAME);  onSaveText(file);  }  }; |

Функция getExternalFilesDir() с параметром null возвращает путь к папке приложения.

Сохранение в папку приложения не гарантирует ее существования, поэтому нам приходится создать ее, если она не существует. Для этого используется функция mkdirs().

В остальном все так же, как и при сохранении в общей папке.

Загрузка файла в обоих случаях очень мало отличается друг от друга:

|  |
| --- |
| private View.OnClickListener loadPrivateListener = new View.OnClickListener() {  @Override  public void onClick(View view) {  File file = new File(getApplicationContext().getExternalFilesDir(null),  FILENAME);  onLoadText(file);  }  };  private View.OnClickListener loadPublicListener = new View.OnClickListener() {  @Override  public void onClick(View view) {  File file = new File(getApplicationContext().getExternalFilesDir(Environment.DIRECTORY\_DOCUMENTS), FILENAME);  onLoadText(file);  }  }; |

Получение доступа к файлам осуществляется так же, как и при сохранении.

Все, что в них одинаково, как видно из кода выше, находится в методе onLoadText():

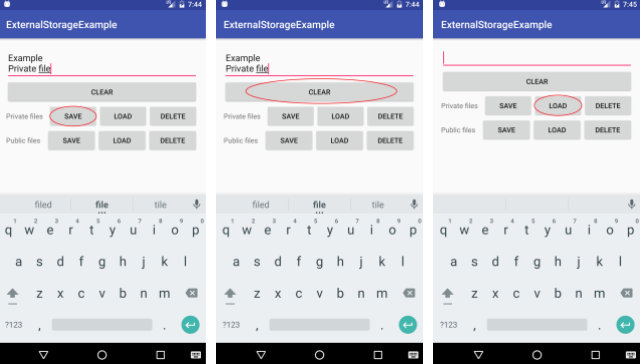
|  |
| --- |
| public void onLoadText(File file) {  if (!isExternalStorageReadable()) {  showToast(getText(R.string.toast\_external\_storage\_not\_found));  return;  }  try {  StringBuilder text = new StringBuilder();  if(!file.exists()) {  showToast(getText(R.string.toast\_file\_not\_exist));  return;  }  BufferedReader br = new BufferedReader(new FileReader(file));  String line;  // есть текст в br, считываем его построчно  while ((line = br.readLine()) != null) {  // добавляем считанное в StringBuilder  text.append(line);  // перенос на новую строку не считывается, поэтому добавляем его  text.append('\n');  }  // последний добавленный перенос лишний, убираем его  text.deleteCharAt(text.length() - 1);  br.close();  editText.setText(text);  }  catch (Exception e) {  showToast(e.getMessage());  }  } |

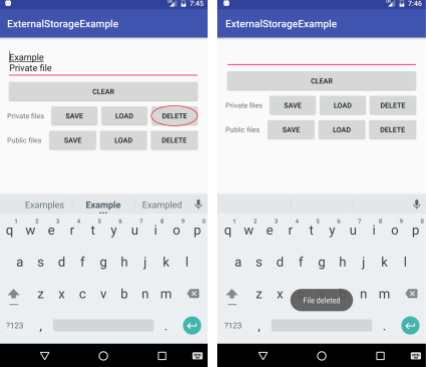
Сначала проверяем доступность External Storage для чтения, далее все так же, как при работе с Internal Storage в примере 2.

Удаление, как и загрузка, отличается от того, что было в примере 2 про Internal Storage, только путем до файла:

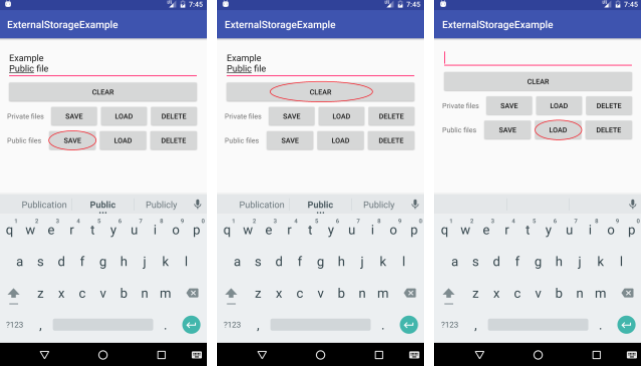
|  |
| --- |
| private View.OnClickListener deletePrivateFileListener = new View.OnClickListener() {  @Override  public void onClick(View view) {  File file = new  File(getApplicationContext().getExternalFilesDir(null), FILENAME);  onDeleteFile(file);  }  };  private View.OnClickListener deletePublicFileListener = new View.OnClickListener() {  @Override  public void onClick(View view) {  File file = new File(getApplicationContext().getExternalFilesDir(Environment.DIRECTORY\_DOCUMENTS), FILENAME);  onDeleteFile(file);  }  };  public void onDeleteFile(File file) {  if (file.exists()) {  file.delete();  showToast(getText(R.string.toast\_file\_deleted));  }  else {  showToast(getText(R.string.toast\_file\_not\_exist));  }  editText.setText("");  } |

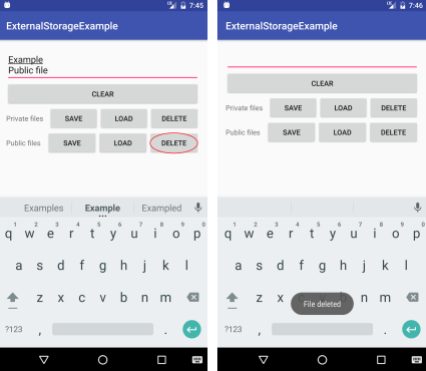
При работе с папкой приложения получаем:





При работе с общей папкой получаем:





# Домашнее задание

Добавить в проекте хранение заметок в файле на устройстве через сериализацию (через БД будем делать потом). При открытии приложения оно сперва должно быть пустым с надписью по центру "нет заметок". Затем, каждую новую заметку, а также изменение старой сохраняем через сериализацию всего списка заметок.

# Дополнительные материалы

1. <https://developer.android.com/guide/topics/data/data-storage.html>
2. <https://developer.android.com/training/basics/data-storage/shared-preferences.html?hl=ru>

# Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. [https://developer.android.com](https://developer.android.com/guide/topics/ui/themes.html)
2. Пол Дейтел, Харви Дейтел, Александр Уолд. Android для разработчиков. // 3-е изд.: Пер. с англ. — Е.: Матвеев, 2016. — 512 с.