В прошлом уроке мы с вами научились удалять приложения, используя root. Удаление работает, однако, есть одна недоработка: после удаления приложение остаётся в списке.

Нам нужно как-то узнать о том, что процесс удаления завершён, и, если он завершён успешно — обновить список, а если нет — показать пользователю уведомление.

Взаимодействуем с UI из AsyncTask

Для того, чтобы получить результат выполнения AsyncTask для начала создадим listener. Его можно объявить как внутренний интерфейс внутри UninstallAsyncTask. В итоге он будет выглядеть следующим образом:

public class UninstallAsyncTask extends AsyncTask<String, Void, Boolean> {

@Override

protected void onPreExecute() {

super.onPreExecute();

}

@Override

protected Boolean doInBackground(String... params) {

String packageName = params[0];

boolean result = RootHelper.uninstall(packageName);

return result;

}

@Override

protected void onPostExecute(Boolean result) {

super.onPostExecute(result);

}

public interface UninstallListener {

void onUninstalled();

void onFailed();

}

}

* onUninstalled() — этот метод мы будем вызывать в случае, если удаление прошло успешно
* onFailed() — этот метод мы вызовем, если удаление не удалось

Помните, мы обсуждали в прошлом уроке проблемы с утечками памяти? Поскольку мы будем создавать инстанс UninstallListener внутри Activity, нужно так же задуматься о возможных утечках, если MainActivityбудет уничтожена раньше, чем закончится выполнение AsyncTask.

Поэтому мы будем использовать так называемую слабую ссылку. Если мы просто объявим поле:

private UninstallListener uninstallListener;

То это поле будет сильной ссылкой. Это значит, что в случае завершения Activity раньше, чем закончится выполнение AsyncTask, то **Garbage Collector** не сможет удалить Activity из памяти, потому что на неё всё ещё есть сильная ссылка. В случае, если мы будем использовать слабую ссылку такого не произойдёт — Garbage Collector просто удалит Activity из памяти, и ссылка обнулится. Давайте попробуем на практике — для начала создадим поле в классе UninstallAsyncTask:

private final WeakReference<UninstallListener> uninstallListenerWeakReference;

Теперь создадим конструктор и инициализируем ссылку:

public UninstallAsyncTask(UninstallListener uninstallListener) {

super();

this.uninstallListenerWeakReference = new WeakReference<>(uninstallListener);

}

Отлично, ссылка создана. Теперь нужно её использовать. Использовать её мы будем в методе onPostExecute():

@Override

protected void onPostExecute(Boolean result) {

super.onPostExecute(result);

// Получаем сильную ссылку

UninstallListener uninstallListener = uninstallListenerWeakReference.get();

// Проверяем на null

if (uninstallListener != null) {

// Вызываем соответствующий метод

if (result) {

uninstallListener.onUninstalled();

} else {

uninstallListener.onFailed();

}

}

}

Поскольку слабая ссылка — это всего лишь "обёртка", использовать её напрямую мы не можем.

Сначала нужно получить сильную ссылку. После этого проверяем ссылку на null, ведь на самом деле объекта уже может и не быть в памяти (как раз если Activity была удалена), и тогда мы получим NullPointerException. Ну и после этого просто вызываем нужный нам метод.

Осталось лишь добавить UninstallListener в MainActivity. В конце Activity добавьте в код:

private final UninstallAsyncTask.UninstallListener uninstallListener = new UninstallAsyncTask.UninstallListener() {

@Override

public void onUninstalled() {

Toast.makeText(MainActivity.this, "Удалено!", Toast.LENGTH\_LONG).show();

}

@Override

public void onFailed() {

Toast.makeText(MainActivity.this, "Не удалось удалить!", Toast.LENGTH\_LONG).show();

}

};

Теперь добавим uninstallListener в конструктор AsyncTask в методе uninstallWithRoot():

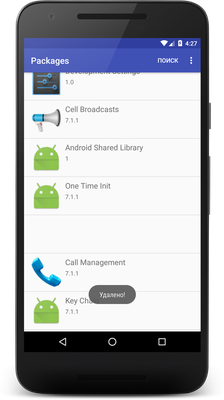
private void uninstallWithRoot(AppInfo appInfo) {

UninstallAsyncTask uninstallAsyncTask = new UninstallAsyncTask(uninstallListener);

uninstallAsyncTask.execute(appInfo.getPackageName());

}

Теперь запустите приложение и попробуйте удалить любое несистемное приложение:

[](https://skillberg.com/media/uploads/2017/11/17/66b734e52a7f49738b244731ee5d61b2.png)Удаление несистемных приложений

Работает!

Теперь давайте сделаем так, чтобы список пакетов обновлялся в каждом из вариантов (даже в случае неудачи — нам ведь нужно вернуть свайпнутое приложение обратно):

private final UninstallAsyncTask.UninstallListener uninstallListener = new UninstallAsyncTask.UninstallListener() {

@Override

public void onUninstalled() {

reloadApps();

}

@Override

public void onFailed() {

reloadApps();

}

};

Запустите приложение и попробуйте удалить любое несистемное приложение — оно исчезнет из списка. Супер.

Удаление системных приложений

Попробуйте теперь удалить совершенно любое системное приложение. Не беспокойтесь, вы ничего не сломаете — потому что у вас это не получится :)

Почему же так? Давайте разберемся, как Android хранит приложения.

Помимо того, что он хранит информацию об установленных приложениях во внутренней базе данных, он так же хранит и APK-файл. Системные и обычные приложения хранятся по-разному:

* /data/app — место, где хранятся APK обычных приложений. Файловая система доступна как для чтения, так и для записи.
* /system/app — место, где хранятся APK системных приложений. Файловая система смонтирована только для чтения.

Обычные приложения должны быть удалены с использованием **Package Manager** (как мы и делали ранее). Системные же можно удалить, просто удалив APK из директории /system/app.

Давайте немного модифицируем наш код, чтобы получить возможность удалять системные приложения. В первую очередь нужно изменить класс AppInfo — добавить в него информацию о том, является ли приложение системным, а так же добавить путь до APK (нам ведь нужно его удалять).

Измените AppInfo, добавив соответствующие поля, геттеры и параметры конструктора:

public class AppInfo {

private final String packageName;

private final int versionCode;

private final String versionName;

private final String name;

private final Drawable icon;

private final boolean isSystem;

private final File apkFile;

public AppInfo(String packageName, int versionCode, String versionName, String name, Drawable icon, boolean isSystem, File apkFile) {

this.packageName = packageName;

this.versionCode = versionCode;

this.versionName = versionName;

this.name = name;

this.icon = icon;

this.isSystem = isSystem;

this.apkFile = apkFile;

}

public String getPackageName() {

return packageName;

}

public int getVersionCode() {

return versionCode;

}

public String getVersionName() {

return versionName;

}

public String getName() {

return name;

}

public Drawable getIcon() {

return icon;

}

public boolean isSystem() {

return isSystem;

}

public File getApkFile() {

return apkFile;

}

}

Метод getInstalledApps() класса AppManager также изменится:

public List<AppInfo> getInstalledApps() {

List<AppInfo> installedApps = new ArrayList<>();

List<PackageInfo> installedPackages = packageManager.getInstalledPackages(0);

for (PackageInfo installedPackage : installedPackages) {

AppInfo appInfo = new AppInfo(

installedPackage.packageName, // Имя пакета

installedPackage.versionCode, // Код версии

installedPackage.versionName, // Имя версии

installedPackage.applicationInfo.loadLabel(packageManager).toString(), // Имя приложения

installedPackage.applicationInfo.loadIcon(packageManager), // Иконка приложения

((installedPackage.applicationInfo.flags & ApplicationInfo.FLAG\_SYSTEM) == 1), // Системное ли приложение

new File(installedPackage.applicationInfo.sourceDir) // Путь до APK

);

installedApps.add(appInfo);

}

return installedApps;

}

Теперь добавим в класс RootHelper метод, который будет удалять системные приложения:

public static boolean uninstallSystem(File appApk) {

executeCommand("mount -o rw,remount /system");

executeCommand("rm " + appApk.getAbsolutePath());

executeCommand("mount -o ro,remount /system");

// Проверяем, удалился ли файл

String output = executeCommand("ls " + appApk.getAbsolutePath());

if (output != null && output.trim().isEmpty()) {

return true;

} else {

return false;

}

}

Помните, я упоминал, что раздел /system, в котором хранятся системные APK, смонтирован в режиме "только для чтения"?

Соответственно, для того, чтобы удалить оттуда файл, нужно перемонтировать его в режим "чтение и запись", то есть **rw**. Это мы и делаем в первой команде.

Во второй мы удаляем APK.

В третьей — перемонтируем раздел обратно в **read only**.

Ну и в четвёртой проверяем, удалился ли файл. Если файл не был найден, команда ls ничего не напишет в stdout, то есть в переменной output будет пустая строка.

Пришло время изменить UninstallAsyncTask. Сейчас он может работать только с именем пакета, используя которое он удаляет "обычные" приложения. Нам же нужно, чтобы он мог удалять любые приложения, при этом самостоятельно выбирая способ удаления. Для этого нужно изменить его параметры: Он должен принимать не String, а AppInfo. Измените строчку объявления класса следующим образом:

public class UninstallAsyncTask extends AsyncTask<AppInfo, Void, Boolean> {

Также придётся поменять тип параметра в doInBackground():

protected Boolean doInBackground(AppInfo... params) {

В самом же методе нам нужно посмотреть, является ли приложение системным, и, исходя из этого, удалить его подходящим способом. В итоге, метод будет выглядеть так:

@Override

protected Boolean doInBackground(AppInfo... params) {

AppInfo appInfo = params[0];

if (appInfo.isSystem()) {

return RootHelper.uninstallSystem(appInfo.getApkFile());

} else {

return RootHelper.uninstall(appInfo.getPackageName());

}

}

Осталось лишь изменить параметр в метод uninstallWithRoot() в MainActivity:

private void uninstallWithRoot(AppInfo appInfo) {

UninstallAsyncTask uninstallAsyncTask = new UninstallAsyncTask(uninstallListener);

uninstallAsyncTask.execute(appInfo);

}

Всё! Запустите приложение и попробуйте удалить любое системное приложение, чтобы проверить, всё ли работает. Единственный нюанс: приложения перестанут быть видимыми только после перезагрузки устройства, поэтому в списке они останутся.

**Важно**: используйте эмулятор для проверок. Если вы удалите что-то важное на эмуляторе — ничего страшного не произойдёт, просто пересоздадите эмулятор и всё. Починить же реальное устройство будет куда сложнее.

До встречи в следующем уроке!

ПРИМЕРЫ КОДА

[ИСХОДНЫЙ КОД УРОКА](https://github.com/Skillberg/PackagesAndroid/tree/lesson-19)

ДОМАШНЕЕ ЗАДАНИЕ

Реализуйте установку приложения с использованием Root, если он имеется на устройстве. Используйте для этого команду pm install путь\_до\_апк.