В этом уроке мы добавим возможность удаления приложения, рассмотрим работу с ItemTouchHelper, Root, эмулятором Genymotion и выполнением задач в фоне.

ItemTouchHelper

Несколько уроков назад мы работали с ItemDecoration — добавляли разделители в RecyclerView. Сейчас же мы рассмотрим иное применение ItemDecoration.

ItemTouchHelper — класс, наследующийся от ItemDecoration и позволяющий обрабатывать такие события, как **Swipe to Dismiss** (смахивание влево или вправо) и **Drag and Drop**. Мы рассмотрим первый вариант: когда пользователь тащит элемент списка (то есть приложение) в сторону, мы удаляем это приложение. Это очень популярный паттерн, который можно увидеть, к примеру, в любом почтовом клиенте.

Использовать его очень легко: во-первых, нужно реализовать интерфейс ItemTouchHelper.Callback, во-вторых — создать новый инстанс ItemTouchHelper, в-третьих — прицепить его к RecyclerView. Давайте же начнём :)

В самом конце класса MainActivity реализуйте интерфейс ItemTouchHelper.Callback:

private final ItemTouchHelper.Callback itemTouchHelperCallback = new ItemTouchHelper.Callback() {

@Override

public int getMovementFlags(RecyclerView recyclerView, RecyclerView.ViewHolder viewHolder) {

return 0;

}

@Override

public boolean onMove(RecyclerView recyclerView, RecyclerView.ViewHolder viewHolder, RecyclerView.ViewHolder target) {

return false;

}

@Override

public void onSwiped(RecyclerView.ViewHolder viewHolder, int direction) {

}

};

Рассмотрим методы этого интерфейса:

* getMovementFlags() — этот метод вызывается, когда пользователь начинает тянуть ячейку. В нём мы должны вернуть значение, согласно которому система будет позволять или запрещать выполнить какое-то действие. Например, мы можем разрешить свайпы только в одну сторону и запретить перемещение ячеек. Обратите внимание на второй аргумент: благодаря тому, что нам передаётся ViewHolder, с которым производится действие, мы можем разрешить или запретить действия для определённой ячейки, основываясь на данных, которые отображает данная ячейка. Например, мы можем запретить удалять системные приложения.
* onMove() — вызывается, когда пользователь переместил ячейку. Вторым аргуметром передаётся ViewHolder ячейки, которую переместили, а третьим — ViewHolder ячейки, на которую произошло перетаскивание.
* onSwiped() — вызывается, когда пользователь завершил жест "свайп". Первым аргументом передаётся ViewHolder ячейки, которую свайпнули, а вторым — направление свайпа.

На данный момент нам нужно подправить только метод getMovementFlags().

Мы не будем разрешать перемещать ячейку, а разрешим только свайп слева направо. Сконвертировать наши пожелания в конкретный флаг поможет метод ItemTouchHelper.Callback.makeMovementFlags().

Первым аргументом мы сообщаем, разрешаем ли перемещения ячейки, а вторым — разрешаем ли свайпы. Итого, метод будет выглядеть вот так:

@Override

public int getMovementFlags(RecyclerView recyclerView, RecyclerView.ViewHolder viewHolder) {

return makeMovementFlags(ItemTouchHelper.ACTION\_STATE\_IDLE, ItemTouchHelper.END);

}

* ItemTouchHelper.ACTION\_STATE\_IDLE сообщает, что действие запрещено, так как мы не позволяем перемещать ячейку.
* ItemTouchHelper.END говорит о том, что мы разрешаем перемещение ячейки "в конец" — слева направо на LTR локалях и справа налево на RTL локалях.

Теперь идём в метод onCreate() и в конце, **перед** строчкой

reloadApps();

Вставляем следующий код:

ItemTouchHelper itemTouchHelper = new ItemTouchHelper(itemTouchHelperCallback);

recyclerView.addItemDecoration(itemTouchHelper);

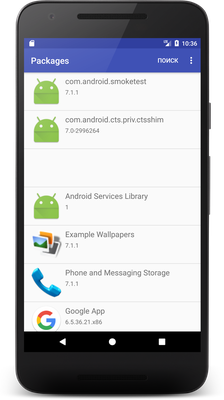
itemTouchHelper.attachToRecyclerView(recyclerView);

В первой строке мы создаём ItemTouchHelper, передавая ему только что созданный нами Callback.

Во второй строчке добавляем ItemTouchHelper в качестве ItemDecoration (так же, как делали с разделителями).

В третьей строчке мы "прикрепляем" ItemTouchHelper к RecyclerView. Вот и всё!

Запустите приложение и попробуйте свайпнуть слева направо. Одна ячейка уедет вправо и исчезнет, оставив после себя пустое пространство:

[](https://skillberg.com/media/uploads/2017/11/17/094ac203e7ee4b27917f264e961fe864.png)Свайп с использованием ItemTouchHelper

Получилось!

Само собой, сейчас ничего не удаляется. Нужно добавить ещё немного кода, чтобы удаление заработало. Как вы помните, в методе onSwiped() мы получаем ViewHolder той ячейки, которую свайпнули.

Из ViewHolder нам нужно как-то получить информацию о приложении, потому что просто от ViewHolder толку не очень много :)

Помните, как мы использовали тэги у View для хранения информации о файле, который выбрал пользователь? Поступим точно так же.

Откройте AppsAdapter и в конце метода onBindViewHolder() добавьте строчку:

holder.itemView.setTag(appInfo);

Теперь в методе onSwiped() нашего коллбэка в MainActivity нужно получить этот тэг:

AppInfo appInfo = (AppInfo) viewHolder.itemView.getTag();

Теперь перейдём к реализации удаления приложений.

Удаление приложений

Без использования Root не получится удалить приложение без участия пользователя. Механизм удаления похож на механизм установки: при помощи неявного Intent мы говорим системе, что хотим, чтобы она удалила определенное приложение. Система показывает пользователю диалог, и пользователь либо подтверждает удаление, либо нет.

Создайте новый метод в MainActivity:

private void startAppUninstallation(AppInfo appInfo) {

}

В нём нам нужно создать неявный Intent с Action ACTION\_UNINSTALL\_PACKAGE:

Intent intent = new Intent(Intent.ACTION\_UNINSTALL\_PACKAGE);

Теперь нужно передать имя пакета приложения, которое мы хотим удалить:

intent.setData(Uri.parse("package:" + appInfo.getPackageName()));

Ну и запустить Activity с этим Intent:

startActivity(intent);

В итоге метод выглядит следующим образом:

private void startAppUninstallation(AppInfo appInfo) {

Intent intent = new Intent(Intent.ACTION\_UNINSTALL\_PACKAGE);

intent.setData(Uri.parse("package:" + appInfo.getPackageName()));

startActivity(intent);

}

Ну и остаётся лишь добавить вызов этого метода в метод onSwiped():

@Override

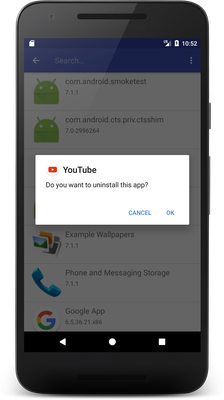
public void onSwiped(RecyclerView.ViewHolder viewHolder, int direction) {

AppInfo appInfo = (AppInfo) viewHolder.itemView.getTag();

startAppUninstallation(appInfo);

}

Запустите приложение и свайпните любое приложение в списке:

[](https://skillberg.com/media/uploads/2017/11/17/cfd030493dbb45ed834bbff8e47bde01.png)Удаление приложений с использованием ItemTouchHelper

Само собой, удалить системное приложение не получится (на то оно и системное) — получится удалить только сторонние приложения. А ещё мы должны каждый раз спрашивать пользователя, точно ли он хочет удалить приложение. Это не очень здорово. Как же быть?

Root

Во всех UNIX и UNIX-like системах, в том числе и в Linux, существует понятие суперпользователя (superuser). Обычно имя этого пользователя — root. У рута есть неограниченные возможности по изменению чего-либо в системе — он может создавать, изменять и удалять любые файлы и производить с системой любые действия, недоступные обычному пользователю.

Android — не исключение, однако у обычного пользователя по умолчанию нет возможности получить привилегии рута. Это сделано как для безопасности пользователя (чтобы сторонние приложения не могли наделать чего-то плохого с системой), так и для того, чтобы пользователь ничего не сломал.

Тем не менее, с самого появления Android некоторые пользователи всё-таки умудряются получить root, благодаря постоянно обнаруживающимся уязвимостям (в основном — в ядре Linux). Поэтому существует целый класс системных утилит, позволяющих производить какие-либо действия с системой, не предусмотренные в Android SDK.

Например, это могут быть приложения, позволяющие удалить ненужные системные приложения, или файловые менеджеры, в которых можно совершать манипуляции с местами в файловой системе, недоступными для обычных приложений.

К сожалению, несмотря на то, что в эмуляторе Android есть рут, приложениям он недоступен. Поэтому мы поставим альтернативный эмулятор — Genymotion.

Genymotion

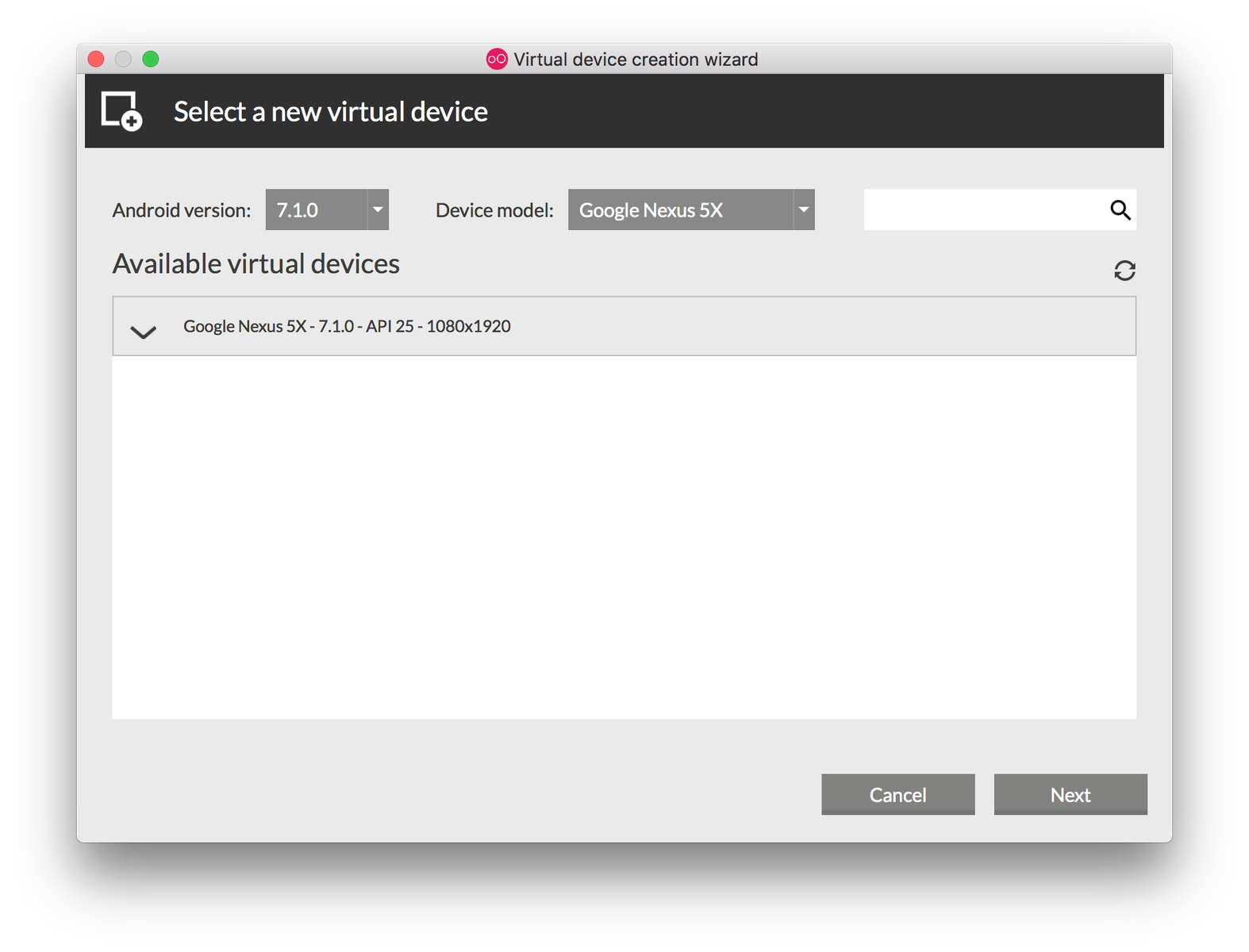
Когда-то давно эмулятор Android был очень-очень медленным (если очень-очень мягко выражаться). Медленным он был из-за того, что эмулировал архитектуру ARM, а на компьютерах используется в основном x86. Эмуляция другой архитектуры — это очень ресурсоёмкая задача. Благо, Google решил эту проблему, выпустив HAXM и официальные сборки Android под x86. HAXM позволяет использовать аппаратную виртуализацию, и эмулятор работает очень быстро.

Однако, это произошло не сразу, и неудивительно, что появились продукты, избавляющие разработчиков от ужасов тормозящего эмулятора. Один из таких продуктов — Genymotion. По сути, Genymotion — это VirtualBox с кастомизированным внешним видом и специальной сборкой Android.

Скачать его можно на [официальном сайте](https://www.genymotion.com/download/). Для скачивания и использования придётся зарегистрироваться, но это быстро и бесплатно. Кроме того, нужно скачать и установить [VirtualBox](https://www.virtualbox.org/wiki/Downloads).

При первом запуске Genymotion предложит принять лицензию для персонального использования и создать первый виртуальный девайс. Соглашаемся и с тем, и с другим.

Выберите, к примеру, вот такой вариант:

[](https://skillberg.com/media/uploads/2017/11/17/74c1b8cce6914bf7a132bda8dc171190.png)Создание устройства в Genymotion

Далее просто подтверждайте все действия, и когда установка завершится — запустите свежесозданный эмулятор.

Libsuperuser

Чтобы упростить работу с операциями, выполняющимися от рута, мы будем использовать библиотеку libsuperuser от небезызвестного Chainfire — разработчика **SuperSU**.

Добавьте библиотеку в build.gradle:

implementation 'eu.chainfire:libsuperuser:1.0.0.201704021214'

Создайте новый класс RootHelper:

public class RootHelper {

}

В этом классе добавьте метод:

public static boolean uninstall(String packageName) {

return false;

}

В этом методе мы будем удалять приложение и возвращать статус — удалось ли его удалить.

Способ выполнения команд от имени суперпользователя отличается от того, что мы делали ранее: при использовании рута нам нужно выполнять команды в шелле, вместо выполнения "обычного" кода.

За установку и удаление приложений в Android отвечает утилита pm (package manager). например, если мы хотим удалить приложение с именем пакета com.some.package, нам нужно вызвать вот такую команду:

pm uninstall com.some.package

В случае успешного выполнения, команда напишет "Success" в stdout. Давайте же сделаем это!

Чтобы выполнить команду от рута, нужно использовать метод Shell.SU.run(), передав в качестве аргумента команду. Метод вернёт список строк, которые вернул процесс при выполнении, или null, если не получилось выполнить команду (например, если рут отсутствует). Давайте создадим ещё один метод, который поможет нам в выполнении команд:

@Nullable

private static String executeCommand(String command) {

List<String> stdout = Shell.SU.run(command);

if (stdout == null) {

return null;

}

StringBuilder stringBuilder = new StringBuilder();

for (String line : stdout) {

stringBuilder.append(line).append("\n");

}

return stringBuilder.toString();

}

Метод выполняет команду, проверяет, не null ли вернулся, и если не null, то конвертирует список строк в одну строку.

А теперь реализуем, собственно, удаление в методе uninstall():

public static boolean uninstall(String packageName) {

String output = executeCommand("pm uninstall " + packageName);

if (output != null && output.toLowerCase().contains("success")) {

return true;

} else {

return false;

}

}

Всё просто!

Нужно только учитывать, что таким образом мы опять же сможем удалять только несистемные приложения (зато не беспокоя каждый раз пользователя). Удаление системных приложений мы рассмотрим позже.

Однако, есть один нюанс: если мы просто попробуем использовать этот метод в MainActivity, то интерфейс подвиснет на время выполнения команды, и, вполне вероятно, что система покажет пользователю сообщение **ANR**(**Activity Not Responding**). Нехорошо! Давайте разберемся поподробнее, почему так происходит и что же с этим делать.

Потоки в Android

Как вы знаете, каждое приложение в практически любой операционной системе имеет [потоки](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D0%BE%D0%BA_%D0%B2%D1%8B%D0%BF%D0%BE%D0%BB%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F). В Android потоки разделяются на две категории — **UI** и **фоновые** (background).

В UI потоке происходит отрисовка интерфейса, и в нём нельзя производить длительные операции, иначе интерфейс начнёт лагать, либо же вообще подвиснет. Более того, система вообще не даст выполнить некоторые операции на UI потоке: например, если попробовать выполнить любой сетевой запрос, приложение просто упадёт. Доступ к интерфейсу можно получить только в этом потоке — если вы попробуете что-либо сделать с элементами интерфейса из фонового потока, приложение упадёт.

Поэтому код, который долго выполняется, всегда нужно выносить в фоновые потоки.

Самый простой способ сделать это — использовать AsyncTask.

AsyncTask

AsyncTask позволяет выполнять код в фоновом потоке, при этом он сообщает нам о событиях, которые происходят в процессе — о начале работы, о прогрессе и о завершении.

* AsyncTask имеет следующие методы:
* onPreExecute() — вызывается перед началом работы. В нём есть доступ к UI, и мы можем, например, показать диалог, сообщающий пользователю о том, что мы что-то делаем.
* doInBackground() — весь код внутри этого метода выполняется в фоновом потоке. Из этого метода нельзя получить доступ к UI.
* onPostExecute() — вызывается после завершения работы. Выполняется на UI-потоке, в этом методе мы можем отобразить результат выполнения задачи и, например, скрыть диалог, сообщающий о выполнении задачи.
* onProgressUpdate() — вызывается, когда изменяется прогресс выполнения задачи. Выполняется на UI-потоке.
* publishProgress() — мы можем использовать этот метод, чтобы сообщить об изменения прогресса.
* cancel() — отменяет выполнение AsyncTask.
* onCancelled() — выполняется после того, как выполнение было отменено. Имеет доступ к UI.

Сам класс AsyncTask является дженериком, то есть мы можем использовать собственные типы при работе с AsyncTask — использовать нужный нам тип параметров, прогресса и результата выполнения.

Давайте попробуем реализовать AsyncTask. Создайте новый класс UninstallAsyncTask:

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

@Override

protected void onPreExecute() {

super.onPreExecute();

}

@Override

protected Boolean doInBackground(String... params) {

return null;

}

@Override

protected void onPostExecute(Boolean result) {

super.onPostExecute(result);

}

}

Видите в первой строчке типы String, Void и Boolean?

* String — тип параметров, которые будут переданы в AsyncTask перед выполнением. Нам нужно передать имя пакета приложения, которое будет удалено, поэтому мы используем String. Он будет использован в doInBackground() — видите там аргумент String... params?
* Void — тип прогресса. Мы не будем сообщать о прогрессе, поэтому просто передаём Void.
* Boolean — тип результата. Мы будем возвращать true в случае успешного выполнения и false, если что-то пошло не так. Результат используется в методе onPostExecute() — параметр Boolean result.

Измените метод doInBackground(), чтобы он выглядел вот так:

@Override

protected Boolean doInBackground(String... params) {

String packageName = params[0];

boolean result = RootHelper.uninstall(packageName);

return result;

}

В первой строчке мы получаем имя пакета из первого параметра. AsyncTask спроектирован таким образом, чтобы в него можно было передать несколько параметров. Мы будем использовать только один, поэтому берём первый из списка.

Во второй строчке, собственно, удаляем приложение.

И в третьей — возвращаем результат.

Спустя некоторое (очень небольшое) время выполнится метод onPostExecute() и в качестве аргумента resultтуда придёт тот самый результат, который мы вернули в doInBackground().

Теперь нужно связать AsyncTask с MainActivity.

Кстати, **очень важное замечание**: в большинстве примеров в интернете зачастую AsyncTask реализован как внутренний класс в Activity или же вообще как анонимный класс. **Так делать нельзя ни в коем случае**! Дело в том, что в обоих случаях внутренний класс (то есть AsyncTask) сохраняет ссылку на внешний класс (то есть Activity). Как правило, код внутри AsyncTask выполняется достаточно долго. А теперь представьте, что Activity по какой-то причине была уничтожена до завершения AsyncTask. "Ну уничтожилась и уничтожилась, что ж страшного?" — спросите вы. А проблема-то в том, что Activityзанимает относительно много места в памяти, и сборщик мусора не может удалить её из памяти, потому что на неё всё ещё ссылается AsyncTask! "Но ведь нам нужно иметь ссылку на Activity, чтобы отобразить прогресс или результат", возразите вы. Да, конечно, нужно. И о том, как это сделать мы поговорим чуть позже в следующем уроке.

А пока создайте новый метод в MainActivity:

private void uninstallWithRoot(AppInfo appInfo) {

UninstallAsyncTask uninstallAsyncTask = new UninstallAsyncTask();

uninstallAsyncTask.execute(appInfo.getPackageName());

}

В первой строчке мы создаём AsyncTask.

Во второй — запускаем его, передав в качестве параметра имя пакета приложения. В принципе, этого достаточно, чтобы удаление работало, осталось лишь вызвать метод. Измените метод startAppUninstallation(), чтобы он выглядел так:

private void startAppUninstallation(AppInfo appInfo) {

uninstallWithRoot(appInfo);

}

Запустите приложение в эмуляторе Genymotion и убедитесь, что всё работает.

Отлично! В следующем уроке мы научим AsyncTask взаимодействовать с UI.

ССЫЛКИ

[GENYMOTION](https://www.genymotion.com/download/) [VIRTUALBOX](https://www.virtualbox.org/wiki/Downloads)

ПРИМЕРЫ КОДА

[ИСХОДНЫЙ КОД УРОКА](https://github.com/Skillberg/PackagesAndroid/tree/lesson-18)