В прошлом уроке мы научились использовать RecyclerView для отображения списков. Но, согласитесь, выглядит наш список не очень здорово, не так ли?

Как минимум, ему не хватает разделителей.

ItemDecoration

В Android SDK существует класс RecyclerView.ItemDecoration. Он позволяет добавить к ячейкам дополнительные "визуальные эффекты", если можно так выразиться.

Как было сказано выше, мы хотим добавить разделители к списку. Какие у нас есть варианты?

Первое, что приходит в голову — добавить в лэйаут ячейки тонкий View темного цвета (в самый низ). Но такой способ не эффективен, к тому же, может получиться так, что он будет выглядеть ненативно на каких-то определенных версиях Android.

Правильный вариант — использовать механизм ItemDecoration. Существует специальный класс RecyclerView.DividerItemDecoration, созданный как раз для этих целей. Реализовать его невероятно просто!

В первую очередь, нужно вынести LayoutManager в отдельный объект:

LinearLayoutManager layoutManager = new LinearLayoutManager(this);

recyclerView.setLayoutManager(layoutManager);

Зачем? Как вы помните, RecyclerView отображает не только списки. Он создан для отображения коллекций данных, и он может отображать данные, к примеру, сеткой. За отображение данных отвечает LayoutManager.

В нашем случае нужен именно список, а не сетка, поэтому мы используем LinearLayoutManager. LinearLayoutManager умеет отображать не только вертикальные, но и горизонтальные списки. По-умолчанию, он отображает списки вертикально, но так же можно создать его следующим способом:

LinearLayoutManager layoutManager = new LinearLayoutManager(this, LinearLayoutManager.HORIZONTAL, false);

И тогда список будет скроллиться горизонтально. А если передать в качестве последнего параметра true, то он будет отображен в обратном направлении.

Так вот, DividerItemDecoration работает со списками и горизонтальной, и вертикальной ориентации, и ему нужно знать, какую ориентацию мы используем. Поэтому при создании передадим ему текущую ориентацию LayoutManager:

DividerItemDecoration dividerItemDecoration = new DividerItemDecoration(this, layoutManager.getOrientation());

Первый параметр — Context, второй — ориентация.

И просто добавим его в RecyclerView:

recyclerView.addItemDecoration(dividerItemDecoration);

Итого, получаем вот такой onCreate():

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

appManager = new AppManager(this);

List<AppInfo> installedApps = appManager.getInstalledApps();

AppsAdapter appsAdapter = new AppsAdapter();

RecyclerView recyclerView = findViewById(R.id.apps\_rv);

LinearLayoutManager layoutManager = new LinearLayoutManager(this);

recyclerView.setLayoutManager(layoutManager);

DividerItemDecoration dividerItemDecoration = new DividerItemDecoration(this, layoutManager.getOrientation());

recyclerView.addItemDecoration(dividerItemDecoration);

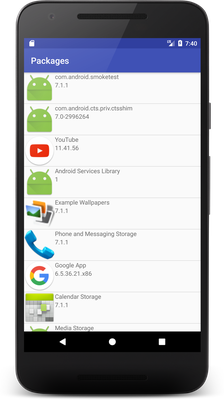
recyclerView.setAdapter(appsAdapter);

appsAdapter.setApps(installedApps);

appsAdapter.notifyDataSetChanged();

}

Вуаля — данные в нашем списке стало проще читать!

[](https://skillberg.com/media/uploads/2017/11/17/975118c279b04989b2c8aec0e9821d43.png)DividerItemDecoration в RecyclerView

ItemDecoration используется не только для этого, и у RecyclerView может быть несколько различных ItemDecoration — есть, например, возможность отслеживать свайпы и drag'n'drop при помощи ItemTouchHelper. Его мы рассмотрим в следующих уроках.

Отступы.

Несмотря на то, что мы добавили разделители, список все ещё выглядит не очень хорошо, правда? Как минимум, стоит добавить отступы к элементам внутри ячейки. В Android есть два типа отступов — **padding** и **margin**.

Padding

**Padding** — это отступ **внутри** View, то есть отступ от контента до границ View.

Когда мы увеличиваем **padding**, View в общем случае увеличивается в размерах или уменьшает контент внутри себя. В XML лэйаутах за него отвечают следующие атрибуты:

* android:padding — устанавливает padding сразу со всех сторон.
* android:paddingTop — устанавливает padding сверху.
* android:paddingBottom — устанавливает padding снизу.
* android:paddingStart — устанавливает padding в начале View (слева на LTR локалях, справа — на RTL).
* android:paddingEnd — устанавливает padding в конце View (справа на LTR локалях, слева — на RTL).

Margin

**Margin** — это отступ **снаружи** View, то есть отступ от границ View до границ контейнера или соседнего View.

Когда мы увеличиваем margin, View в общем случае не меняется в размерах и не уменьшает контент, но при этом увеличивается отступ от соседних View и/или контейнера.

В XML лэйаутах за него отвечают следующие атрибуты:

* android:layout\_margin — устанавливает margin сразу со всех сторон.
* android:layout\_marginTop — устанавливает margin сверху.
* android: layout\_marginBottom — устанавливает margin снизу.
* android: layout\_marginStart — устанавливает margin в начале View (слева на LTR локалях, справа — на RTL).
* android: layout\_marginEnd — устанавливает margin в конце View (справа на LTR локалях, слева — на RTL).

Давайте посмотрим на наше приложение.

Нам определенно нужно увеличить отступы со всех краев ячейки, чтобы контент не прижимался к краям. Тут нам нужен **padding**.

Измените файл лэйаута ячейки, чтобы он выглядел следующим образом:

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:orientation="horizontal"

android:paddingBottom="8dp"

android:paddingEnd="8dp"

android:paddingStart="16dp"

android:paddingTop="8dp">

<ImageView

android:id="@+id/icon\_iv"

android:layout\_width="72dp"

android:layout\_height="72dp"

android:adjustViewBounds="true"

android:scaleType="fitXY" />

<LinearLayout

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:orientation="vertical">

<TextView

android:id="@+id/name\_tv"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content" />

<TextView

android:id="@+id/version\_tv"

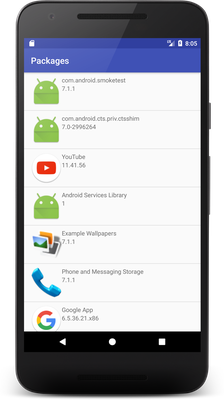
android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content" />

</LinearLayout>

</LinearLayout>

Мы добавили внутренние отступы со всех сторон, причем слева чуть больше. Уже стало лучше, не так ли?

[](https://skillberg.com/media/uploads/2017/11/17/0ea3d198b4ac4b7986bed69fcba7407a.png)Padding в ячейках RecyclerView

Однако, текст все равно слишком сильно прижимается к иконке и друг к другу. Нужно добавить отступ для названия приложения слева, и отступ для версии слева и сверху. Поскольку текст у нас находится в отдельном LinearLayout, достаточно просто добавить левый и правый отступ к нему, и верхний отступ к TextView с версией:

<LinearLayout

android:layout\_height="wrap\_content"

android:layout\_marginEnd="8dp"

android:layout\_marginStart="8dp"

android:layout\_width="wrap\_content"

android:orientation="vertical">

<TextView

android:id="@+id/name\_tv"

android:layout\_height="wrap\_content"

android:layout\_width="wrap\_content" />

<TextView

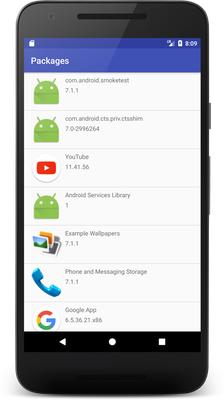
android:id="@+id/version\_tv"

android:layout\_height="wrap\_content"

android:layout\_marginTop="4dp"

android:layout\_width="wrap\_content" />

</LinearLayout>

[](https://skillberg.com/media/uploads/2017/11/17/0f8dde19e5844956b4d6f6e2d4b55f58.png)Отступы у TextView с версией

Ну и выделим заголовок, увеличив размер текста:

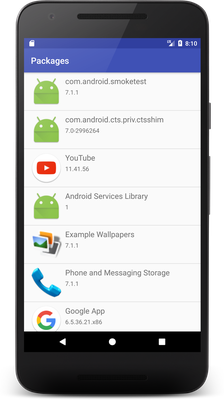
<TextView

android:id="@+id/name\_tv"

android:layout\_height="wrap\_content"

android:layout\_width="wrap\_content"

android:textSize="18sp" />

[](https://skillberg.com/media/uploads/2017/11/17/c64c38de1caf4083946948a170322260.png)Размер текста в TextView

Куда лучше, чем было, не так ли?

SwipeRefreshLayout

То, что мы умеем отображать данные — это здорово, но что, если пользователь захочет их обновить? Единственный на данный момент способ сделать это — перезапустить приложение, что, согласитесь, не лучший вариант.

Во-первых, нужно дать пользователю возможность вручную обновить данные.

Во-вторых, нужно отслеживать установку и удаление приложений.

О втором пункте мы поговорим в одном из следующих уроков, а вот первый подробно разберем сейчас.

Общепринятый паттерн обновления данных в списках — Pull-to-Refresh, то есть, пользователю нужно потянуть список вниз и отпустить его. Для реализации Pull-to-Refresh в Android долгое время нужно было использовать сторонние библиотеки, пока Google не добавил SwipeRefreshLayout в Support Library.

Чтобы добавить SwipeRefreshLayout, просто замените FrameLayout на него в лэйауте MainActivity:

<?xml version="1.0" encoding="utf-8"?>

<android.support.v4.widget.SwipeRefreshLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"

android:id="@+id/swipe\_refresh\_layout"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context="com.test.packages.MainActivity">

<android.support.v7.widget.RecyclerView

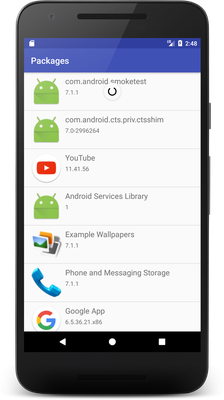
android:id="@+id/apps\_rv"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent" />

</android.support.v4.widget.SwipeRefreshLayout>

Теперь запустите приложение, потяните список вниз и отпустите:

[](https://skillberg.com/media/uploads/2017/11/17/2e522b1b9df043dcb619e37e529f05dc.png)SwipeRefreshLayout и RecyclerView

Как видите, визуально процесс обновления начался. Однако, обновления данных, само собой, не происходит. Чтобы обновление данных заработало, нужно выполнить ещё несколько простых шагов.

В Java-коде MainActivity добавьте новый объект класса SwipeRefreshLayout.OnRefreshListener:

private final SwipeRefreshLayout.OnRefreshListener onRefreshListener = new SwipeRefreshLayout.OnRefreshListener() {

@Override

public void onRefresh() {

}

};

Метод onRefresh() вызывается тогда, когда пользователь отпустил список после перетягивания вниз — то есть именно в тот момент, когда нужно начать обновление данных. В этом методе мы должны получить список приложений и обновить список. После получения списка установленных приложений мы просто передадим его в адаптер и вызовем notifyDatasetChanged(). Для удобства давайте создадим отдельный метод с именем reloadApps():

private void reloadApps() {

List<AppInfo> installedApps = appManager.getInstalledApps();

appsAdapter.setApps(installedApps);

appsAdapter.notifyDataSetChanged();

}

Чтобы он заработал, нужно сделать appsAdapter членом класса MainActivity. Теперь код MainActivity будет выглядеть вот так:

public class MainActivity extends AppCompatActivity {

private static final String TAG = "MainActivity";

private AppManager appManager;

private AppsAdapter appsAdapter;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

appManager = new AppManager(this);

appsAdapter = new AppsAdapter();

RecyclerView recyclerView = findViewById(R.id.apps\_rv);

LinearLayoutManager layoutManager = new LinearLayoutManager(this);

recyclerView.setLayoutManager(layoutManager);

DividerItemDecoration dividerItemDecoration = new DividerItemDecoration(this, layoutManager.getOrientation());

recyclerView.addItemDecoration(dividerItemDecoration);

recyclerView.setAdapter(appsAdapter);

}

private void reloadApps() {

List<AppInfo> installedApps = appManager.getInstalledApps();

appsAdapter.setApps(installedApps);

appsAdapter.notifyDataSetChanged();

}

private final SwipeRefreshLayout.OnRefreshListener onRefreshListener = new SwipeRefreshLayout.OnRefreshListener() {

@Override

public void onRefresh() {

}

};

}

Метод reloadApps() мы будем вызывать в самом конце onCreate() и в методе onRefresh():

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

appManager = new AppManager(this);

appsAdapter = new AppsAdapter();

RecyclerView recyclerView = findViewById(R.id.apps\_rv);

LinearLayoutManager layoutManager = new LinearLayoutManager(this);

recyclerView.setLayoutManager(layoutManager);

DividerItemDecoration dividerItemDecoration = new DividerItemDecoration(this, layoutManager.getOrientation());

recyclerView.addItemDecoration(dividerItemDecoration);

recyclerView.setAdapter(appsAdapter);

reloadApps();

}

lang=java

private final SwipeRefreshLayout.OnRefreshListener onRefreshListener = new SwipeRefreshLayout.OnRefreshListener() {

@Override

public void onRefresh() {

reloadApps();

}

};

Так же нам пригодится найти сам SwipeRefreshLayout и задать созданный нами onRefreshListener:

public class MainActivity extends AppCompatActivity {

// ....

private SwipeRefreshLayout swipeRefreshLayout;

@Override

protected void onCreate(Bundle savedInstanceState) {

// ....

swipeRefreshLayout = findViewById(R.id.swipe\_refresh\_layout);

swipeRefreshLayout.setOnRefreshListener(onRefreshListener);

// ....

Запустив приложение, вы увидите, что список вроде как обновляется, но индикатор обновления не исчезает. Дело в том, что мы должны "выключать" его самостоятельно. В конце метода reloadApps() добавьте такую строчку:

swipeRefreshLayout.setRefreshing(false);

Супер, теперь обновление данных работает и индикатор исчезает после выполнения обновления! А итоговый код MainActivity выглядит так:

public class MainActivity extends AppCompatActivity {

private static final String TAG = "MainActivity";

private SwipeRefreshLayout swipeRefreshLayout;

private AppManager appManager;

private AppsAdapter appsAdapter;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

swipeRefreshLayout = findViewById(R.id.swipe\_refresh\_layout);

swipeRefreshLayout.setOnRefreshListener(onRefreshListener);

appManager = new AppManager(this);

appsAdapter = new AppsAdapter();

RecyclerView recyclerView = findViewById(R.id.apps\_rv);

LinearLayoutManager layoutManager = new LinearLayoutManager(this);

recyclerView.setLayoutManager(layoutManager);

DividerItemDecoration dividerItemDecoration = new DividerItemDecoration(this, layoutManager.getOrientation());

recyclerView.addItemDecoration(dividerItemDecoration);

recyclerView.setAdapter(appsAdapter);

reloadApps();

}

private void reloadApps() {

List<AppInfo> installedApps = appManager.getInstalledApps();

appsAdapter.setApps(installedApps);

appsAdapter.notifyDataSetChanged();

swipeRefreshLayout.setRefreshing(false);

}

private final SwipeRefreshLayout.OnRefreshListener onRefreshListener = new SwipeRefreshLayout.OnRefreshListener() {

@Override

public void onRefresh() {

reloadApps();

}

};

}

На этом всё, а в следующем уроке мы изучим работу с меню!

ПРИМЕРЫ КОДА

[ИСХОДНЫЙ КОД УРОКА](https://github.com/Skillberg/PackagesAndroid/tree/lesson-13)