Во многих приложениях используются меню. В Android существует три типа меню:

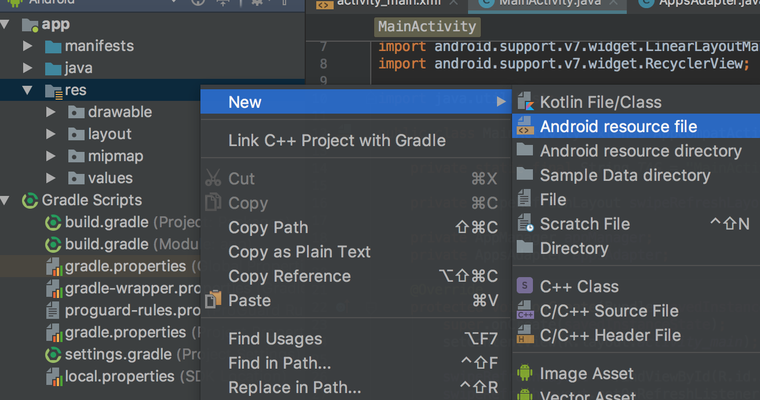
* **Options Menu** — в этом меню располагаются действия, которые затрагивают приложение в целом (поиск, настройки и т.д.).
* **Context Menu** — контекстное меню. Обычно отображается, когда пользователь совершает длительное нажатие на элемент. В нем располагаются действия, связанные с конкретным элементом (с тем, на который совершено нажатие).
* **Popup Menu** — всплывающее меню. В нем располагаются действия, относящиеся к конкретному View.

В этом уроке мы рассмотрим первый тип меню. Раньше **Options Menu** отображалось в нижней части экрана, когда пользователь нажимал на соответствующую аппаратную кнопку на устройстве. Начиная с Android 3.0 **Options Menu** перенесли в верхнюю часть приложения — в ActionBar (сейчас — Toolbar).

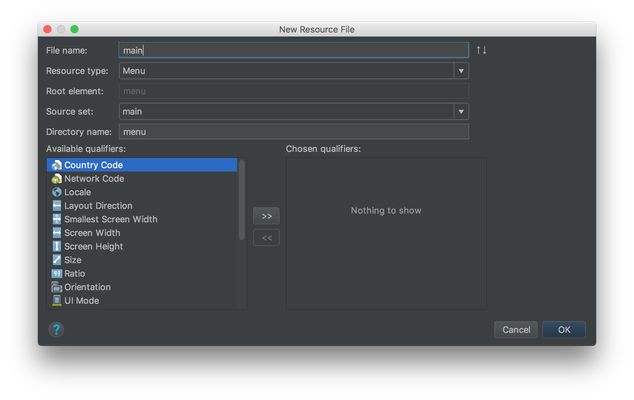
Создание Options Menu

Как и в случае с лэйаутами, в Android принято создавать меню декларативным способом (то есть в XML). Давайте создадим наше первое меню :)

Нажмите правой кнопкой на директорию res, затем выберите **New** -> **Android resource file**:

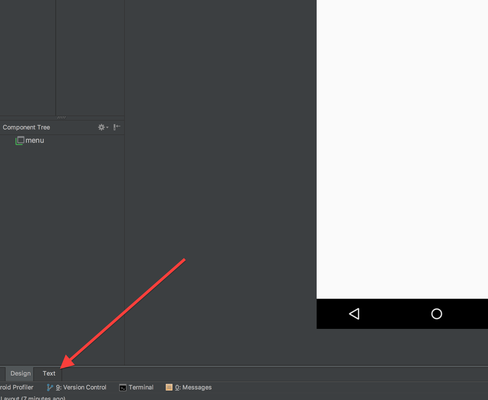
[](https://skillberg.com/media/uploads/2017/11/17/d7a2ff976dc1422c8610d1ad322645eb.png)Создание файла с ресурсами

В появившемся окне выберите **Menu** в пункте **Resource type**, а в поле **File name** введите "main":

[](https://skillberg.com/media/uploads/2017/11/17/79cfb98a8b6141788ca80cb61b968278.png)Создание файла меню

Нажмите на кнопку **ОК**. Откроется файл с визуальным редактором меню, однако, для лучшего понимания, мы будем работать с текстовым вариантом.

Для этого нажмите на вкладку **Text** в нижней части окна:

[](https://skillberg.com/media/uploads/2017/11/17/e2bbab67d95e47d3999df19e41296736.png)Переход к редактированию меню в режиме текста

Вы увидите следующий код:

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

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

</menu>

Создадим первый элемент меню. Каждый отдельный пункт создается с использованием тега item. Этот тег может принимать множество атрибутов:

* android:id — идентификатор элемента меню. То же самое, что и id View.
* android:title — заголовок элемента.
* android:titleCondensed — сокращённый текст. Может использоваться в случае, если обычный заголовок слишком длинный.
* android:icon — иконка элемента меню (ссылка на drawable-ресурс).
* android:onClick — метод, который должен быть вызван, когда происходит нажатие на пункт меню.
* android:showAsAction — задает тип отображения элемента меню. Может принимать следующие значения:
  + ifRoom — показывает элемент в app bar (верхней панели приложения), если для него есть места. Если места нет, то показывает его в overflow-меню (в выпадающем списке).
  + withText — отображает текст (заданный с использованием android:title). Может использоваться как флаг вместе с остальными флагами.
  + never — никогда не показывать элемент в app bar, всегда показывать его в overflow-меню.
  + always — всегда показывать элемент в app bar. Лучше никогда не использовать, за исключением ситуаций, когда нужно обязательно показать элемент. Лучше всего использовать ifRoom вместо этого варианта.
  + collapseActionView — Action View, ассоциированный с этим элементом меню, может сворачиваться.
* android:actionLayout — ссылка на лэйаут, который должен быть использован в качестве Action View.
* android:actionViewClass — имя класса View, который должен быть использован в качестве Action View.
* android:actionProviderClass — имя класса ActionProvider, который должен использоваться с этим элементом.
* android:alphabeticShortcut — алфавитный шорткат для пункта меню (один символ).
* android:alphabeticModifiers — модификатор, который должен быть использован с android:alphabeticShortcut. Может принимать следующие значения (можно комбинировать):
  + META
  + CTRL
  + ALT
  + SHIFT
  + SYM
  + FUNCTION
* android:numericShortcut — цифра, которую можно использовать в качестве шортката.
* android:numericModifiers — модификатор, который должен быть использован с android:numericShortcut. Принимает те же значения, что и android:alphabeticModifiers.
* android:checkable — является ли элемент выбираемым (то есть, показывать ли чекбокс рядом с ним). Принимает true или false.
* android:visible — показывать ли элемент. Принимает true или false.
* android:enabled — включен ли элемент. Принимает true или false.
* android:menuCategory — категория элемента. Возможные значения:
  + container — для элементов, являющихся частью контейнера.
  + system — для элементов, предоставляемых системой.
  + secondary — для нечасто используемых элементов.
  + alternative — для элементов, которые являются альтернативными действиями над отображаемыми данными.
* android:orderInCategory — порядок отображения элемента в категории.

Немало, да? :)

На самом деле, нам потребуются лишь несколько атрибутов. Создайте новый элемент, чтобы меню выглядело вот так:

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

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

xmlns:app="http://schemas.android.com/apk/res-auto">

<item

android:id="@+id/install\_item"

android:title="Установить"

app:showAsAction="never" />

</menu>

Теперь в коде MainActivity добавьте новый метод onCreateOptionsMenu():

@Override

public boolean onCreateOptionsMenu(Menu menu) {

return super.onCreateOptionsMenu(menu);

}

В этом методе мы должны создать меню и вернуть true. В случае, если мы создали меню, и хотим, чтобы система показала его, возвращаем true, если меню показывать не надо — false.

Точно так же, как и в случае с View, XML-файл меню — это всего лишь описание меню, а нам нужно создать объект из этого описания. Для этого есть специальный класс MenuInflater (по аналогии с LayoutInflater). Получить его инстанс можно вызовом метода getMenuInflater():

MenuInflater menuInflater = getMenuInflater();

Система передает нам объект класса Menu в качестве единственного параметра метода onCreateOptionsMenu(). Именно это меню мы должны создать, используя MenuInflater и XML-файл меню. Почти по аналогии с LayoutInflater:

menuInflater.inflate(R.menu.main, menu);

Первым параметром мы передаем индентификатор ресурса меню, а вторым — объект меню.

Таким образом, метод будет выглядеть вот так:

@Override

public boolean onCreateOptionsMenu(Menu menu) {

super.onCreateOptionsMenu(menu);

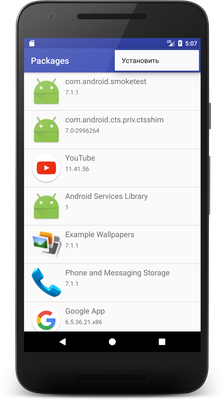
MenuInflater menuInflater = getMenuInflater();

menuInflater.inflate(R.menu.main, menu);

return true;

}

Запустите приложение, нажмите на три вертикальные точки в правом верхнем углу, и увидите overflow-меню с созданным нами пунктом:

[](https://skillberg.com/media/uploads/2017/11/17/5117e64522574c55b95d70a801e6239a.png)Меню в Android

Отлично!

Обработка выбора пункта меню

Мало просто отобразить меню — мы ведь должны как-то среагировать на выбор пункта. За это отвечает метод onOptionsItemSelected().

Система будет вызывать этот метод каждый раз, когда пользователь нажмёт на какой-либо элемент меню.

В качестве параметра передается объект класса MenuItem, содержащий информацию об элементе меню, на который нажал пользователь (по сути, отражающий item из XML-файла.

Мы можем среагировать на событие, и вернуть false, если мы хотим, чтобы система продолжила обработку, или true, чтобы сообщить системе, что мы обработали нажатие и передавать событие дальше не требуется.

Именно для того, чтобы понять, на какой пункт меню нажали, мы задавали атрибут android:id. Идентификатор элемента меню можно получить, используя метод getItemId() объекта MenuItem.

Общепринятой практикой является использование оператора switch, в нашем случае — следующим образом:

@Override

public boolean onOptionsItemSelected(MenuItem item) {

switch (item.getItemId()) {

case R.id.install\_item:

return true;

default:

return super.onOptionsItemSelected(item);

}

}

Таким образом мы просто сообщаем системе, что обработали событие выбора элемента меню, и больше ничего не делаем.

Toast — всплывающие сообщения.

Бывают ситуации, когда нужно показать пользователю короткое информационное сообщение. Для этих целей существует класс Toast. Нам он как раз подойдет — когда пользователь нажмёт на пункт меню, мы покажем ему короткое сообщение.

Создайте в MainActivity новый метод showToast():

private void showToast() {

}

Toast создаётся статическим методом makeText(), принимающим следующие параметры:

* Контекст.
* Текст, который нужно отобразить (CharSequence или ID ресурса строки).
* Длительность. Можно использовать одну из двух констант (задать другую длительность не получится):
  + LENGTH\_LONG — показывать в течение длительного времени (около 3.5 секунд, может незначительно отличаться в разных версиях ОС).
  + LENGTH\_SHORT — показывать в течение короткого времени (около 2 секунд).

Давайте создадим Toast (в методе showToast(), который мы недавно создали):

Toast toast = Toast.makeText(this, "Hello", Toast.LENGTH\_LONG);

А теперь покажем созданный Toast, используя метод show():

toast.show();

Теперь добавим метод showToast() в onOptionsItemSelected():

@Override

public boolean onOptionsItemSelected(MenuItem item) {

switch (item.getItemId()) {

case R.id.install\_item:

showToast();

return true;

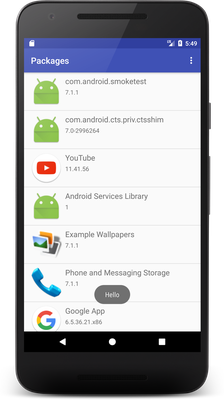
default:

return super.onOptionsItemSelected(item);

}

}

И посмотрим, что получилось:

[](https://skillberg.com/media/uploads/2017/11/17/9846065c3d2c45918ca46814dbab785a.png)Показываем Toast при выборе элемента меню

Ура, сработало!

Не забывайте вызывать метод show()! Это наиболее частая ошибка при использовании Toast.

Action View. Реализуем поиск.

Поиск — функция, крайне необходимая для многих приложений. В нашем приложении поиск пригодится для фильтрации списка — когда нужно быстро найти определенное приложение, установленное в системе.

Поскольку единственное, что от нас требуется — это отфильтровать список, нам вполне подойдёт один из вариантов Action View в меню — SearchView. Измените файл меню, чтобы он выглядел следующим образом:

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

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

xmlns:app="http://schemas.android.com/apk/res-auto">

<item

android:id="@+id/search\_item"

android:title="Поиск"

app:actionViewClass="android.support.v7.widget.SearchView"

app:showAsAction="ifRoom|collapseActionView" />

<item

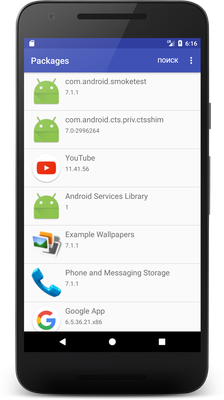
android:id="@+id/install\_item"

android:title="Установить"

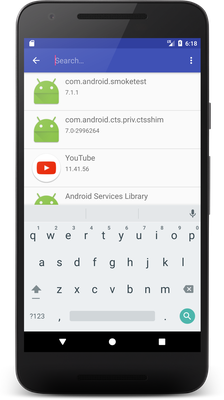
app:showAsAction="never" />

</menu>

Запустив приложение, вы увидите следующее:

[](https://skillberg.com/media/uploads/2017/11/17/6189d5d656ae4a99b6131822ff933f4e.png)ActionView в меню

А нажав на этот пункт — поле для ввода текста:

[](https://skillberg.com/media/uploads/2017/11/17/18e019b9ccf242829f153688bb1abbf9.png)Поле для поиска в ActionView

В него уже можно вводить текст, однако фильтроваться, само собой, ничего не будет.

В методе onCreateOptionsMenu() найдём соответствующий пункт меню, а в нём — и сам SearchView:

@Override

public boolean onCreateOptionsMenu(Menu menu) {

super.onCreateOptionsMenu(menu);

MenuInflater menuInflater = getMenuInflater();

menuInflater.inflate(R.menu.main, menu);

MenuItem searchItem = menu.findItem(R.id.search\_item);

SearchView searchView = (SearchView) searchItem.getActionView();

return true;

}

Нам надо отслеживать ввод текста в поле для поиска. Для этого зададим SearchView специальный Listener — SearchView.OnQueryTextListener:

searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {

@Override

public boolean onQueryTextSubmit(String query) {

return false;

}

@Override

public boolean onQueryTextChange(String newText) {

return false;

}

});

Метод onQueryTextSubmit() вызывается, когда пользователь нажал на кнопку "поиск" на клавиатуре, после ввода поискового запроса.

Нам нужен метод onQueryTextChange(). Он вызывается каждый раз, когда текст в поле поиска изменился. Если мы обработали событие изменения текста, возвращаем true, а если не обработали, и хотим, чтобы система продолжила обработку — возвращаем false.

Выведем введённый текст в лог:

@Override

public boolean onQueryTextChange(String newText) {

Log.i(TAG, "Text: " + newText);

return true;

}

Запустите приложение, чтобы убедиться, что все работает.

А теперь нужно отфильтровать данные. Лучше всего сделать это в адаптере.

Стратегия такова: сохраняем оригинальный список приложений, заводим ещё один список, и в него добавляем отфильтрованные приложения.

В адаптере для отображения данных используем отфильтрованный список.

Создаем новое поле filteredApps:

private List<AppInfo> filteredApps = new ArrayList<>();

Кроме того, создадим дополнительное поле, в котором будем хранить текущую строку, с помощью которой будем фильтровать список приложений. По умолчанию эта строка будет пустой:

private String query = "";

Создадим метод filterApps():

private void filterApps() {

filteredApps.clear();

if (query.isEmpty()) {

filteredApps.addAll(apps);

} else {

for (AppInfo app : apps) {

if (app.getName().toLowerCase().contains(query)) {

filteredApps.add(app);

}

}

}

}

В нём мы фильтруем список приложений.

Также нам понадобится метод setQuery():

public void setQuery(String query) {

this.query = query;

filterApps();

}

В этом методе мы задаём текст для фильтрации, после чего фильтруем список.

Ну и замените обращения кapps на filteredApps в методах getItemCount() и onBindViewHolder(). В итоге, наш адаптер будет выглядеть вот так:

public class AppsAdapter extends RecyclerView.Adapter<AppsAdapter.ViewHolder> {

private List<AppInfo> apps = new ArrayList<>();

private List<AppInfo> filteredApps = new ArrayList<>();

private String query = "";

public void setApps(List<AppInfo> apps) {

this.apps = apps;

filterApps();

}

private void filterApps() {

filteredApps.clear();

if (query.isEmpty()) {

filteredApps.addAll(apps);

} else {

for (AppInfo app : apps) {

if (app.getName().toLowerCase().contains(query)) {

filteredApps.add(app);

}

}

}

}

public void setQuery(String query) {

this.query = query;

filterApps();

}

// В этом методе мы создаем новую ячейку

@Override

public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {

Context context = parent.getContext();

LayoutInflater layoutInflater = LayoutInflater.from(context);

View view = layoutInflater.inflate(R.layout.view\_item\_app, parent, false);

ViewHolder viewHolder = new ViewHolder(view);

return viewHolder;

}

// В этом методе мы привязываем данные к ячейке

@Override

public void onBindViewHolder(ViewHolder holder, int position) {

AppInfo appInfo = filteredApps.get(position);

holder.nameTv.setText(appInfo.getName());

holder.versionTv.setText(appInfo.getVersionName());

holder.iconIv.setImageDrawable(appInfo.getIcon());

}

// В этом методе мы возвращаем количество элементов списка

@Override

public int getItemCount() {

return filteredApps.size();

}

/\*\*

\* View holder

\* Хранит информацию о ячейке

\*/

static class ViewHolder extends RecyclerView.ViewHolder {

private final ImageView iconIv;

private final TextView nameTv;

private final TextView versionTv;

public ViewHolder(View itemView) {

super(itemView);

iconIv = itemView.findViewById(R.id.icon\_iv);

nameTv = itemView.findViewById(R.id.name\_tv);

versionTv = itemView.findViewById(R.id.version\_tv);

}

}

}

Осталось лишь связать ввод текста в поиск с фильтрацией данных. В методе onQueryTextChange() интерфейса OnQueryTextListener добавляем строки:

appsAdapter.setQuery(newText.toLowerCase().trim());

appsAdapter.notifyDataSetChanged();

Полный код onCreateOptionsMenu():

@Override

public boolean onCreateOptionsMenu(Menu menu) {

super.onCreateOptionsMenu(menu);

MenuInflater menuInflater = getMenuInflater();

menuInflater.inflate(R.menu.main, menu);

MenuItem searchItem = menu.findItem(R.id.search\_item);

SearchView searchView = (SearchView) searchItem.getActionView();

searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {

@Override

public boolean onQueryTextSubmit(String query) {

return false;

}

@Override

public boolean onQueryTextChange(String newText) {

appsAdapter.setQuery(newText.toLowerCase().trim());

appsAdapter.notifyDataSetChanged();

return true;

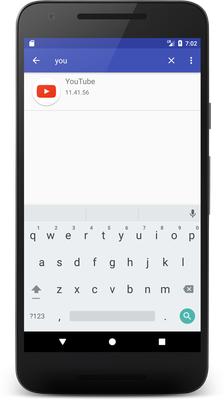
}

});

return true;

}

Запустите приложение и попробуйте ввести что-то в поиск:

[](https://skillberg.com/media/uploads/2017/11/17/2f2b0e7be9694976b9f5df44ad34271b.png)Поиск с использованием SearchView

Супер! Да, эта реализация не идеальна, но вполне сгодится, чтобы показать, как работает виджет SearchView, и как можно фильтровать списки.

ПРИМЕРЫ КОДА

[ИСХОДНЫЙ КОД УРОКА](https://github.com/Skillberg/PackagesAndroid/tree/lesson-14)