**Лекция 8 Уведомления и настройки**

В этой лекции рассматриваются некоторые дополнительные компоненты пользовательского интерфейса, общие для приложений Android: [**уведомления**](https://developer.android.com/guide/topics/ui/notifiers/notifications.html) , [**меню настроек**](https://developer.android.com/guide/topics/ui/settings.html) и [**диалоги**](https://developer.android.com/guide/topics/ui/dialogs.html) . Цель этой лекции — дать *представление* , а не полностью охватить эти концепции; дополнительные параметры и примеры см. в официальной документации Android.

В этой лекции используется код, найденный по адресу <https://github.com/info448/lecture08-notifications-settings> .

**8.1 Диалоги**

Ранее мы предоставляли пользователям обратную связь с помощью простых всплывающих окон, таких как [Toasts](https://developer.android.com/guide/topics/ui/notifiers/toasts.html) или [Snackbars](https://developer.android.com/training/snackbar/showing.html) . Однако иногда вам может понадобиться показать более сложное «всплывающее» представление — возможно, требующее дополнительного взаимодействия.

Диалог [28](https://info448.github.io/notifications-settings.html#fn28) — это «всплывающее» [модальное окно](https://en.wikipedia.org/wiki/Modal_window) (представление, которое не заполняет экран), которое либо просит пользователя принять решение, либо предоставляет дополнительную информацию. По сути, диалоги похожи на функцию [***и***](https://developer.android.com/guide/topics/ui/dialogs.html) ее варианты, используемые в JavaScript.window.alert()

Существует базовый Dialogкласс, но почти всегда вместо него мы используем предопределенный подкласс (похожий на тот, как мы использовали AppCompatActivity). [29](https://info448.github.io/notifications-settings.html#fn29) — самая распространенная версия: простое сообщение с кнопками, с помощью которых можно ответить (подтвердить, отменить и т. д.).[AlertDialog](https://developer.android.com/reference/android/support/v7/app/AlertDialog.html)

На самом деле мы не создаем экземпляр AlertDialogнапрямую (на самом деле, его конструкторы protectedнастолько недоступны для нас). Вместо этого мы используем вспомогательный класс- *фабрику,* называемый an [AlertDialog.Builder](https://developer.android.com/reference/android/support/v7/app/AlertDialog.Builder.html). Существует ряд шагов для использования конструктора для создания Dialog:

1. Создайте новый конструктор для этого конкретного диалога. Конструктор принимает в качестве Contextпод, чтобы создать диалог. Обратите внимание, что после инициализации конструктора вы можете создавать и пересоздавать тот же диалог одним вызовом метода — вот преимущества использования фабрики.
2. Вызовите методы «setter» в конструкторе, чтобы указать заголовок, сообщение и т. д. для диалогового окна, которое появится. Это может быть жестко закодированный текст или ссылка на ресурс XML String (как строка, обращенная к пользователю, последний вариант больше подходит для опубликованных приложений). Каждый метод setter вернет ссылку на конструктор, что упрощает их цепочку.
3. Используйте соответствующие методы сеттера для указания обратных вызовов (через DialogInterface.OnClickListener) для отдельных кнопок. Обратите внимание, что «положительная» кнопка обычно имеет текст "OK", но это можно настроить.
4. Наконец, фактически создайте экземпляр AlertDialogс помощью builder.create()метода, используя **show()**метод для отображения диалогового окна на экране!

*//java*

AlertDialog.Builder builder = **new** AlertDialog.Builder(**this**);

builder.setTitle("Alert!")

 .setMessage("Danger Will Robinson!");

builder.setPositiveButton("I see it!", **new** DialogInterface.OnClickListener() {

 **public** void onClick(DialogInterface dialog, int id) {

 *// User clicked OK button*

 }

});

AlertDialog dialog = builder.create();

dialog.show();

*//kotlin*

**val** builder = AlertDialog.Builder(**this**)

builder.apply {

 setTitle("Alert!")

 setMessage("Danger Will Robinson!")

 setPositiveButton("I see it!") { dialog, id ->

 Log.v(TAG, "You clicked okay! Good times :)")

 }

}

Важной частью обучения разработке приложений Android является умение читать API для обнаружения эффективных вариантов. Например, можете ли вы прочитать [API AlertDialog.Builder](https://developer.android.com/reference/android/support/v7/app/AlertDialog.Builder.html) и определить, как добавить кнопку «отмена» к оповещению?

Хотя AlertDialogэто наиболее распространенный Dialog, Android поддерживает и другие подклассы. Например, [DatePickerDialog](https://developer.android.com/reference/android/app/DatePickerDialog.html)и [TimePickerDialog](https://developer.android.com/reference/android/app/TimePickerDialog.html)предоставляет предопределенные пользовательские интерфейсы для выбора даты или времени соответственно. Подробности об их использовании см. [в руководстве Pickers .](https://developer.android.com/guide/topics/ui/controls/pickers.html)

**ДиалогФрагменты**

Описанный выше процесс создаст и отобразит диалоговое окно, но у этого диалогового окна есть несколько проблем с тем, как оно взаимодействует с остальной частью фреймворка Android, а именно с жизненным циклом Activity, в которое оно встроено.

Например, если устройство меняет конфигурацию (например, поворачивается из портретной в альбомную), то Activity уничтожается и создается заново (его onCreate()метод будет вызван снова). Но если это происходит во время отображения Dialog, то android.view.WindowLeakedпроизойдет ошибка, и Dialog будет потерян!

Чтобы избежать этих проблем, нам нужно иметь способ дать этому Диалогу его собственный жизненный цикл, который может взаимодействовать с жизненным циклом Действия… что-то вроде того, чтобы сделать его *модульной* частью Действия… верно, нам нужно сделать его Фрагментом! В частности, мы будем использовать подкласс Фрагмента, называемый [DialogFragment](http://developer.android.com/guide/topics/ui/dialogs.html#DialogFragment), который является Фрагментом, который отображается как модальный диалог, плавающий над Действием (дополнительная работа не требуется).

Как и в предыдущих примерах Fragment, нам нужно будет создать собственный подкласс DialogFragment. Часто проще всего сделать это *вложенным классом* , если Dialog не будет выполнять много работы (например, показывать простое подтверждение).

Вместо того, чтобы указывать макет фрагмента через onCreateView(), мы можем вместо этого переопределить onCreateDialog()обратный вызов, чтобы указать Dialogобъект, который предоставит иерархию представления для фрагмента. Этот диалог можно создать с AlertDialog.Builderклассом, как и раньше!

*//java*

**public** static **class** HelloDialogFragment **extends** DialogFragment {

 **public** static HelloDialogFragment newInstance() {

 Bundle args = **new** Bundle();

 HelloDialogFragment fragment = **new** HelloDialogFragment();

 fragment.setArguments(args);

 **return** fragment;

 }

 **public** Dialog onCreateDialog(Bundle savedInstanceState) {

 AlertDialog.Builder builder = **new** AlertDialog.Builder(getActivity());

 *//...*

 AlertDialog dialog = builder.create();

 **return** dialog;

 }

}

*//kotlin*

**class** HelloDialogFragment : DialogFragment() {

 **companion** **object** {

 **fun** newInstance(): HelloDialogFragment {

 **val** args = Bundle()

 **val** fragment = HelloDialogFragment()

 fragment.arguments = args

 **return** fragment

 }

 }

 **override** **fun** onCreateDialog(savedInstanceState: Bundle?): Dialog {

 **return** activity?.let { *//confirm Activity isn't null*

 **val** builder = AlertDialog.Builder(it)

 builder.apply {

 *//...*

 }

 builder.create()

 } ?: **throw** IllegalStateException("Activity cannot be null")

 }

}

Наконец, мы можем фактически показать это DialogFragment, создав его экземпляр (не забудьте использовать newInstance()фабричный метод!), а затем вызвав show()метод для него, чтобы он отображался как Dialog. show()Метод принимает FragmentManagerиспользуемый для управления этой транзакцией. Используя DialogFragment, можно изменить конфигурацию устройства (повернуть телефон), и Dialog сохранится.

Вот еще один изящный трюк: a DialogFragment— это просто a Fragment. Это значит, что мы можем использовать его *везде, где* обычно используем Fragments… включая встраивание их в макеты! Например, если вы создали DetailsFragmentподкласс DialogFragmentвместо Fragment, его можно будет использовать точно так же, как и раньше. Это все еще Fragment, просто с дополнительными функциями — одна из которых — show()метод, который будет показывать его как Dialog!

* Используйте , setStyle(DialogFragment.STYLE\_NO\_TITLE, android.R.style.Theme\_Holo\_Light\_Dialog)чтобы сделать фрагмент более похожим на диалог.

Правда в том, что диалоги не очень часто используются в Android (по сравнению с другими системами пользовательского интерфейса). Приложения, скорее всего, просто динамически изменят отображаемый фрагмент или активность, а не будут прерывать поток пользователя, создавая всплывающее модальное окно. И 80% используемых диалогов *—* это AlertDialogs. Тем не менее, стоит ознакомиться с этим процессом и шаблонами, на которых он основан!

**8.2 Уведомления**

Мы можем сообщить пользователю, что происходит с приложением, выведя всплывающее уведомление или диалог, но часто мы хотим уведомить пользователя о чем-то за пределами обычного пользовательского интерфейса Activity (например, когда приложение не запущено или не мешает другим взаимодействиям). Для этого мы можем использовать [**Notifications**](https://developer.android.com/guide/topics/ui/notifiers/notifications.html)[30](https://info448.github.io/notifications-settings.html#fn30) . Это специализированные представления, которые отображаются в *области уведомлений (значки в верхней части экрана операционной системы) и в панели уведомлений* системы , к которым пользователь может получить доступ в любой момент — даже находясь вне приложения — проведя пальцем вниз по экрану.

Документация Android для компонентов пользовательского интерфейса в целом довольно подробная и полезная (в конце концов, Google хочет убедиться, что разработчики могут создавать эффективные приложения, тем самым делая платформу стоящей). И поскольку существует так много различных элементов пользовательского интерфейса, и они постоянно меняются, для того, чтобы заниматься реальной разработкой Android, вам нужно уметь читать, синтезировать и применять эту документацию. Таким образом, эта лекция покажет, как использовать эту документацию и применять ее для создания уведомлений. Мы пройдемся по документации, чтобы добавить функцию, которая при нажатии кнопки «уведомить» будет отображать уведомление, сообщающее, сколько раз мы нажали эту кнопку.

* Чтобы узнать больше, откройте документацию по уведомлениям по адресу <https://developer.android.com/training/notify-user/build-notification> .
* Просматривая документацию, мы видим ссылку на [Notifications Overview](https://developer.android.com/guide/topics/ui/notifiers/notifications) для начала. Эта страница также ссылается на [Notification Design Guide](https://material.io/design/platform-guidance/android-notifications.html#usage) , который является хорошим местом, чтобы разобраться, как разрабатывать *эффективные* уведомления.
* Лично я предпочитаю работать с примерами кода, изменяя их до тех пор, пока не получу что-то, что делает то, что мне нужно. Поэтому я предлагаю **медленно** прокручивать вниз , пока не найдете пример, который можно скопировать/вставить или хотя бы сослаться. Затем вы можете прокрутить обратно, чтобы узнать больше подробностей о том, как работает этот код. Оглавление также полезно для этого.

В конце концов вы найдете подраздел [«Создание простого уведомления»](https://developer.android.com/guide/topics/ui/notifiers/notifications.html#SimpleNotification) , который кажется отличным местом для начала!

Первая часть создания этого уведомления — использование NotificationCompat.Builder(используйте v4версию поддержки; v7теперь устарело). Мы видели этот тип класса Builder с AlertBuilder, и здесь применяется та же концепция: это класс, используемый для создания уведомления для нас. Мы вызываем сеттеры, чтобы указать свойства уведомления.

* Начиная с API 26 (Oreo), мы также обязаны указать [**канал уведомлений**](https://developer.android.com/guide/topics/ui/notifiers/notifications.html#ManageChannels) (называемый «категорией уведомлений» в пользовательском интерфейсе). Они используются, чтобы помочь пользователям управлять уведомлениями, группируя их вместе и позволяя пользователям контролировать настройки этой группы — например, будут ли они отображаться, какие звуки они воспроизводят и т. д.

Каналы уведомлений идентифицируются по идентификатору строки, который уникален в пределах пакета, поэтому мы обычно определяем его как константу. На данный момент мы можем создать один канал для всего нашего приложения. На самом деле мы настроим канал позже, но все уведомления должны быть связаны с одним.

* + Обратите внимание, что это необходимо сделать, если вы ориентируетесь на API 26 или более позднюю версию; в противном случае сигнатуры методов немного изменятся.
* Просматривая эти настройки: у меня нет ресурса для рисования, который можно использовать для иконки, поэтому я не хочу включать спецификацию иконки. Немного посмотрев вверх, вы заметите, что требуется значок уведомления [,](https://developer.android.com/guide/topics/ui/notifiers/notifications.html#Required) поэтому нам нужно будет его создать.

Мы можем создать новый Image Asset для значка уведомления ( File > New > Image Asset), как мы делали ранее с иконками запуска и иконками Material. Укажите «тип» как Notification, дайте ему подходящее имя и выберите клипарт по вашему выбору.

* Мы задаем «заголовок» и содержание уведомления.
* «Приоритет» — это то, насколько важно уведомление (например, должно ли оно прерывать пользователя или нет). Он используется для API 25 и ниже; API 26 (Oreo) вместо этого встраивает это в «важность» канала уведомлений. Но мы устанавливаем его здесь для обратной совместимости.

Следующий шаг — фактическое создание [Notification Channel](https://developer.android.com/training/notify-user/channels) . Хотя мы дали ему имя, нам нужно фактически создать объект для управления настройками этого канала.

* Обратите внимание, что код для запуска каналов доступен только для API 26 (Oreo). Поскольку их нет в библиотеке поддержки, нам нужно использовать оператор, ifчтобы убедиться, что мы не пытаемся запустить код на старых устройствах! Мы делаем это, сравнивая с Build.VERSION.SDK\_INT:
* **if** (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {
* *// Create the NotificationChannel*

}

* Канал создается как новый NotificationChannelобъект и получает идентификатор (чтобы он знал свое имя), отображаемую метку и *важность* .

Для важности IMPORTANCE\_HIGHзаставит его всплывать и издавать звук. Смотрите [документацию](https://developer.android.com/training/notify-user/channels#importance) для других опций. Также ознакомьтесь [с руководством по дизайну](https://material.io/design/platform-guidance/android-notifications.html#settings) для лучших практик по указанию приоритета уведомления.

* + Для более ранних версий Android вы устанавливаете priority. Для того чтобы уведомление визуально навязчиво отображалось пользователю, его [приоритет](https://developer.android.com/guide/topics/ui/notifiers/notifications.html#Priority) должен быть достаточно высоким (он должен быть NotificationCompat.PRIORITY\_HIGHили выше) **, а также** оно должно использовать либо звук *, либо* вибрацию (оно должно быть *действительно важным* , чтобы получить всплывающее уведомление).

Вы можете заставить уведомление вибрировать, используя [setVibrate()](http://developer.android.com/reference/android/support/v4/app/NotificationCompat.Builder.html#setVibrate(long[]))метод, передав ему массив времени (в миллисекундах), в которое следует включить и выключить вибрацию. Шаблон: [delay, vibrate, sleep, vibrate, sleep, ...] Вы также можете назначить звук по умолчанию с помощью (например)builder.setSound(Settings.System.DEFAULT\_NOTIFICATION\_URI);

* Мы используем NotificationManager( *не* библиотеку поддержки!) для «регистрации» канала, чтобы можно было добавлять в него уведомления.
* **val** notificationManager: NotificationManager = getSystemService(Context.NOTIFICATION\_SERVICE) **as** NotificationManager

notificationManager.createNotificationChannel(channel)

Пропустив вперед, мы можем фактически *показать* уведомление, которое мы создали на данный момент. Мы используем NotificationManager(аналогично FragmentManager, SmsManager, и т. д.) для получения *службы уведомлений* («приложения», которое обрабатывает все уведомления для ОС). Мы говорим этому менеджеру фактически выдать построенный Notificationобъект.

* Мы также передаем notify()методу IDномер для ссылки на конкретное уведомление. Это позволит нам ссылаться на это уведомление и обновлять его позже. Мы можем **обновить** это уведомление позже, просто выпустив уведомление с тем же **ID**номером, и оно «заменит» предыдущее!
* Например, мы можем сделать так, чтобы наш текст основывался на некоторой переменной экземпляра, а уведомление отслеживало количество кликов!

**8.2.1 Действия при нажатии**

Нам удалось показать уведомления, но мы также хотим, чтобы они что-то *делали* , например, когда вы нажимаете на него — указывая [Действие уведомления](https://developer.android.com/guide/topics/ui/notifiers/notifications#Actions) . По крайней мере, нажатие на уведомление должно открывать соответствующее приложение. И поскольку Intentsэто сообщения для открытия действий, имеет смысл, что нажатие на уведомление отправит Intent. Более того, помните, что уведомления существуют за пределами приложения, которое их создает; поэтому нам нужно использовать Намерения для отправки сообщений (мы не можем просто указать функцию обратного вызова).

Но помимо того, чтобы просто сделать Intentдля отправки, мы на самом деле собираемся создать то, что называется [PendingIntent](http://developer.android.com/reference/android/app/PendingIntent.html). Подробности не *очень* читабельны, но это в основном обертка вокруг , Intentкоторую мы передаем **другому** классу. Затем, когда этот другой компонент хочет отправить Intent, он может «развернуть» его из PendingIntentи запустить команду, как если бы это были мы (с точки зрения «источника», разрешений и т. д.). Intent — это тот, который мы отправили, но «ожидает» доставки/активации другой службой. Поэтому, когда мы выбираем уведомление, служба Notification Drawer может открыть этот PendingIntentпакет, вытащить , который Intentмы хотим отправить, а затем отправить его. И этот Intent, который отправляет Notification Drawer, — разбудить нашу Activity, и он отправляется с *нашими* разрешениями (а не разрешениями Drawer), как если бы мы отправили его сами!

* Это как если бы вы попросили у профессора рекомендательное письмо и дали бы ему конверт с маркой (что вы всегда должны делать!). Когда они получат посылку, они могут отправить свое письмо, используя *ваш* конверт и марку!
* Мы создаем PendingIntentс помощью PendingIntent.getActivity()метода — он принимает контекст, идентификационный код (похожий на коды запросов, которые мы используем при отправке намерений для результатов; это позволяет нам узнать, откуда PendingIntentпришла команда), объект, Intentкоторый нужно запустить, и флаг PendingIntent.FLAG\_CURRENT\_UPDATE, чтобы при повторном вызове PendingIntent он обновлял существующее, а не заменял его новым ожидающим действием (например, если мы обновляем действие нашего уведомления).
* Обратите внимание, что вы можете создать «искусственный backstack» (например, если Notification должен переместить пользователя в середину рабочего процесса вашего приложения), используя класс [TaskStackBuilder](https://developer.android.com/training/notify-user/navigation#build_a_pendingintent_with_a_back_stack). Этот конструктор позволит вам указать «историю» намерений и использовать ее для создания правильного PendingIntent.
* Обязательно расскажите NotificationCompat.Builderо *цели контента, которую* мы только что определили!

Теперь мы можем нажать на уведомление и заставить его что-то сделать! Обратите внимание, что, как всегда, есть ряд других деталей/деталей/функций, которые мы можем указать, но я оставляю вам возможность поискать их в документации.

Поскольку курс фокусируется на разработке, эта лекция ссылается, но **не** обсуждает руководящие принципы дизайна пользовательского интерфейса. Например: какой текст следует поместить в уведомление? *Когда* следует использовать уведомление? В [документации по дизайну](https://material.io/design/platform-guidance/android-notifications.html) Android есть множество рекомендаций по этим вопросам . Основные руководящие принципы HCI и мобильного дизайна также применимы здесь (например, делать действия очевидными, давать обратную связь, избегать необратимых действий и т. д.).

**8.3 Настройки**

Последняя тема этой лекции — поддержка предоставления пользователю возможности решать, должно ли нажатие кнопки создавать уведомления или нет. Например, иногда пользователь может просто захотеть увидеть тосты! Самый чистый способ поддержки такого рода пользовательских предпочтений — создать некоторые [настройки](https://developer.android.com/guide/topics/ui/settings.html) с помощью Preferences.

**SharedPreferences**

[**Shared Preferences**](https://developer.android.com/guide/topics/data/data-storage.html#pref)[31](https://info448.github.io/notifications-settings.html#fn31) — это один из способов сохранения**данных** в приложении*между запусками приложения* , то есть данные хранятся на диске, а не только в памяти, поэтому не будут потеряны, если приложение будет уничтожено. SharedPreferences хранит*пары ключ-значение* примитивов (строки, целые числа и т. д.), похожие на те, что мы вставляли вBundles. Эти данные будут храниться между сеансами приложения: если я сохраню некоторые данные в Preferences и закрою приложение, они будут там, когда я вернусь.

* Настройки хранятся в **XML-файле** в файловой системе. По сути, мы сохраняем списки пар ключ-значение как базовое XML-дерево в текстовом файле (похожем на valuesфайлы ресурсов, которые мы создали). Обратите внимание, что это *не ресурс* , а файл, который также структурирован как XML.
* SharedPreferences не очень хорош для сложных или обширных структурированных данных (поскольку он хранит только пары ключ-значение, и только примитивы). Используйте другие варианты для более сложного сохранения данных, например, помещение данных в базу данных или использование файловой системы — оба варианта обсуждаются в последующих лекциях.

Хоть они и *называются* «Предпочтениями», они не только для «пользовательских предпочтений»! Мы можем сохранять любые небольшие фрагменты примитивных данных в файле SharedPreferences.

Мы можем получить доступ к файлу SharedPreferences с помощью .getSharedPreferences(String, int)метода. Первый параметр String— это имя файла SharedPreference, к которому мы хотим получить доступ (у нас может быть несколько XML-файлов; просто используйте , getPreferences()чтобы использовать один по умолчанию). Второй параметр int— это флаг, указывающий, должны ли другие приложения иметь доступ к этому файлу. MODE\_PRIVATE( 0) — это значение по умолчанию, MODE\_WORLD\_READABLEа MODE\_WORLD\_WRITEABLE— другие параметры.

Мы можем редактировать этот XML-файл, вызывая .edit()объект , SharedPreferencesчтобы получить SharedPreferences.Editor, который является Bundleобъектом-аналогом, в который мы можем putпомещать значения.

* Нам нужно вызвать .commit()редактор, чтобы сохранить наши изменения в файловой системе!

Наконец, мы можем просто вызывать get()методы объекта SharedPreferences, чтобы извлечь из него данные! Второй параметр этих методов — это значение по умолчанию, если предпочтение еще не существует, что позволяет легко избежать nullошибок.

**Для практики** попробуйте сохранить количество уведомлений в функции Activity onStop()и извлечь его в onCreate(). Это позволит вам сохранить количество даже при уничтожении Activity.

**Настройки предпочтений**

Хотя файл SharedPreferences действует как общее хранилище данных, он называется Shared *Preferences* , поскольку чаще всего используется для «пользовательских настроек», например «Настроек» для приложения.

* Да, термин «предпочтения» может использоваться в значении «общих предпочтений» или «пользовательских предпочтений» (или и того, и другого, если это одно и то же!) в зависимости от контекста.

«Меню настроек пользователя» — это элемент, с которым сталкивается пользователь, поэтому мы захотим определить его как [ресурс XML](https://developer.android.com/guide/topics/ui/settings.html#DefiningPrefs) . Но мы не будем пытаться создать собственную компоновку и взаимодействие: вместо этого мы просто определим список из [32](https://info448.github.io/notifications-settings.html#fn32) элементов как ресурс![Preferences](https://developer.android.com/reference/android/preference/Preference.html)

* Мы можем создать новый ресурс с помощью мастера создания нового ресурса Android Studio. «Тип» для этого на самом деле просто XML(generic), хотя наш «корневой элемент» будет <PreferenceScreen>(спасибо интеллектуальным значениям по умолчанию!). По соглашению ресурс preferences называется preferences.xml.

Внутри <PreferenceScreen>мы добавляем больше элементов: один для представления каждого предпочтения, которое мы хотим, чтобы пользователь мог настроить (или каждую «строку» экрана настроек). Мы можем определить различные типы Preferenceобъектов, такие как <CheckBoxPreference>, <EditTextPreference>, <SwitchPreference>, или <ListPreference>(для диалога с переключателями). Есть также несколько других вариантов; см. [Preference](https://developer.android.com/reference/android/preference/Preference.html)базовый класс.

* Эти элементы должны включать следующие атрибуты XML (среди прочих):
	+ android:keyключ, который следует использовать при сохранении настроек в файле SharedPreferences
	+ android:titleвидимое пользователем имя для настройки
	+ android:defaultvalueзначение по умолчанию для предпочтения (используйте trueили falseдля флажков).
	+ Дополнительные параметры можно найти в документации [Preference](https://developer.android.com/reference/android/preference/Preference.html#lattrs).
* Мы можем дополнительно разделить эти настройки, чтобы организовать их: мы можем поместить их в PreferenceCategoryтег (со своим собственным titleи key), чтобы сгруппировать их вместе.
* Наконец, мы можем указать, что наши настройки имеют несколько экранов, вложив PreferenceScreenэлементы. Это создает «подэкраны» (как подменю): когда мы нажимаем на элемент, он переносит нас на следующий экран.

Обратите внимание, что более понятный (но более трудоемкий) способ сделать это, если у вас *много* настроек, — использовать [preference-headers](http://developer.android.com/guide/topics/ui/settings.html#PreferenceHeaders), что позволяет создавать более качественные многопанельные макеты… но поскольку мы не создаем приложения с таким количеством настроек, этот процесс оставляем в качестве упражнения для читателя.

Как только все настройки будут определены в XML, нам просто нужно будет отобразить их в нашем приложении! Поскольку настройки будут отдельным экраном, нам нужно будет создать еще одну Activity (хотя ей не понадобится макет!). Чтобы фактически отобразить XML настроек, мы будем использовать PreferenceFragmentкласс (специализированный фрагмент для отображения списков Preferenceобъектов); обычно его легко создать как вложенный класс в Activity.

* Для фрагмента нам не нужно указывать onCreateView()метод, вместо этого мы просто загрузим этот Preferenceресурс в onCreate()метод, используя addPreferencesFromResource(R.xml.preferences). Это заставит PreferenceFragmentсоздать соответствующий макет!
* Для Activity нам просто нужно, чтобы он загружал этот Fragment через FragmentTransaction:
* getFragmentManager().beginTransaction()
* .replace(android.R.id.content, **new** SettingsFragment())

 .commit();

* + Activity даже не нужно загружать макет: просто укажите транзакцию! Но если мы хотим включить другие вещи (например, ActionBar), нам нужно будет структурировать Activity и его макет более подробно.
	+ Обратите внимание, что контейнер для нашего фрагмента — android.R.id.content. Это относится к «корневому элементу» текущего представления — в основном к тому, setContentView()во что обычно происходит инфляция.
* Также есть PreferenceActivityкласс, но официальная рекомендация: **не используйте его** . Многие из его методов устарели, и поскольку мы используем фрагменты через библиотеку поддержки, нам следует придерживаться процесса фрагментации.

Наконец, как нам взаимодействовать с этими настройками? Вот в чем фокус: preferencesресурс XML **автоматически** связывается с SharedPreferencesфайлом! И фактически, каждый раз, когда мы настраиваем настройку в PreferenceFragment, значения в этом файле также редактируются! По сути, PreferenceFragment *автоматически* редактирует SharedPreferences на основе взаимодействия с пользователем. Нам никогда не нужно писать в файл самим, просто считываем из него (и мы считываем из него точно так же, как считываем любой другой файл SharedPreferences, как описано выше).

XML preferenceсоответствует файлу «по умолчанию» SharedPreferences, к которому мы получим доступ через:

SharedPreferences sharedPref = PreferenceManager.getDefaultSharedPreferences(**this**);

* И затем у нас есть объект, из которого мы можем извлечь данные с помощью getString(), getBoolean()и т. д.

Это позволит нам проверить настройки, прежде чем показывать уведомление!

Это основы использования настроек. Более подробную информацию см. [в документации](https://developer.android.com/guide/topics/ui/settings.html) , а также [в руководстве по дизайну](https://developer.android.com/design/patterns/settings.html) для получения рекомендаций по организации настроек.

1. <https://developer.android.com/guide/topics/ui/dialogs.html>[↩](https://info448.github.io/notifications-settings.html#fnref28)
2. <https://developer.android.com/reference/android/support/v7/app/AlertDialog.html>[↩](https://info448.github.io/notifications-settings.html#fnref29)
3. <https://developer.android.com/guide/topics/ui/notifiers/notifications.html>[↩](https://info448.github.io/notifications-settings.html#fnref30)
4. <https://developer.android.com/guide/topics/data/data-storage.html#pref>[↩](https://info448.github.io/notifications-settings.html#fnref31)
5. <https://developer.android.com/reference/android/preference/Preference.html>[↩](https://info448.github.io/notifications-settings.html#fnref32)