**Лекция 3 Мероприятия**

В этой лекции рассматриваются [**Activity**](https://developer.android.com/guide/components/activities/index.html) , которые являются базовым компонентом, используемым в приложениях Android. Activities предоставляют фреймворк для кода Java, который позволяет пользователю взаимодействовать с макетами, определенными в ресурсах.

В этой лекции используется код, найденный по адресу <https://github.com/info448/lecture03-activities> .

По данным Google:

Activity — это компонент приложения, предоставляющий экран, с которым пользователи могут взаимодействовать, чтобы что-то сделать.

Вы можете думать об Activity как об отдельном *экране* в вашем приложении, эквиваленте «окна» в системе GUI. Обратите внимание, что Activity не обязательно **должны** быть полноэкранными: они также могут быть плавающими модальными окнами, встроенными в другие Activity (например, половина экрана) и т. д. Но мы начнем с того, что будем думать о них как о полноэкранных. У нас может быть много Activity (экранов) в приложении, и они слабо связаны, поэтому мы можем легко перемещаться между ними.

Во многих отношениях Activity — это «механизм учета»: место для хранения *состояния* и *данных* , а также указания Android, что отображать на дисплее. В этом отношении он функционирует во многом как Контроллер (в смысле [Модель-Представление-Контроллер](https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller) )!

Также следует отметить из документации [8](https://info448.github.io/activities.html#fn8) :

Действие — это одно конкретное действие, которое может выполнить пользователь.

что подразумевает предложение по дизайну: Действия (экраны) разбивают ваше приложение на «задачи». Каждое Действие может представлять то, что пользователь делает в данный момент. Если пользователь делает что-то еще, это должно быть другое Действие (и, вероятно, другой экран).

**3.1 Создание деятельности**

Мы указываем Activity для приложения, создавая *подкласс* (расширяя) класс фреймворка [Activity](https://developer.android.com/reference/android/app/Activity.html). Мы используем **наследование** , чтобы создать специализированный тип Activity. Расширяя этот класс, мы наследуем все методы, необходимые для управления тем, как ОС Android взаимодействует с Activity — поведение, например, показ экрана, разрешение изменения Activities и закрытие Activity, когда она больше не используется.

Если вы посмотрите на Empty по умолчанию MainActivity, он на самом деле подклассы [AppCompatActivity](https://developer.android.com/reference/android/support/v7/app/AppCompatActivity.html), который сам по себе уже является специализированным подклассом , Activityкоторый предоставляет [ActionBar](http://developer.android.com/reference/android/support/v7/app/ActionBar.html)(панель инструментов в верхней части экрана с именем вашего приложения). Если вы измените класс на просто extend Activity, эта панель исчезнет.

Чтобы внести это изменение, вам нужно будет импортировать Activityкласс! Сочетание клавиш для импорта класса в Android Studio — alt+return, или вы можете сделать это вручную (найдите пакет в документации)! Я рекомендую вам изменить настройки IDE для автоматического импорта используемых вами классов.

Есть ряд других встроенных Activityподклассов, которые мы могли бы подклассифицировать вместо этого. Мы упомянем их, когда они станут актуальными. Многие из доступных классов были устарели в пользу **Fragments** , которые являются своего рода «под-активностями», вложенными в более крупные Activity. Фрагменты будут обсуждаться в более поздней лекции.

**3.2 Жизненный цикл деятельности**

Важно отметить: есть ли у этого Activity **конструктор** , который мы вызываем? *Нет!* Мы никогда не пишем код, который **создает экземпляр** нашего Activity (то есть: мы никогда не вызываем MainActivity()). В Android нет main()метода. Activity создаются и управляются операционной системой Android при запуске приложения.

Хотя мы никогда не вызываем конструктор или main()метод, у Activity есть четко определенный [жизненный цикл](https://developer.android.com/guide/components/activities/activity-lifecycle.html) , то есть ряд **событий** , которые происходят во время использования (например, когда Activity создается, когда она останавливается и т. д.).

Когда происходит каждое из этих событий, Android выполняет **метод обратного вызова** , точно так же, как вы указали onClick()метод для реагирования на нажатие кнопки. Мы можем *переопределить* эти обратные вызовы жизненного цикла, чтобы выполнять специальные действия (читай: запускать наш собственный код), когда происходят эти события.

Каков жизненный цикл?

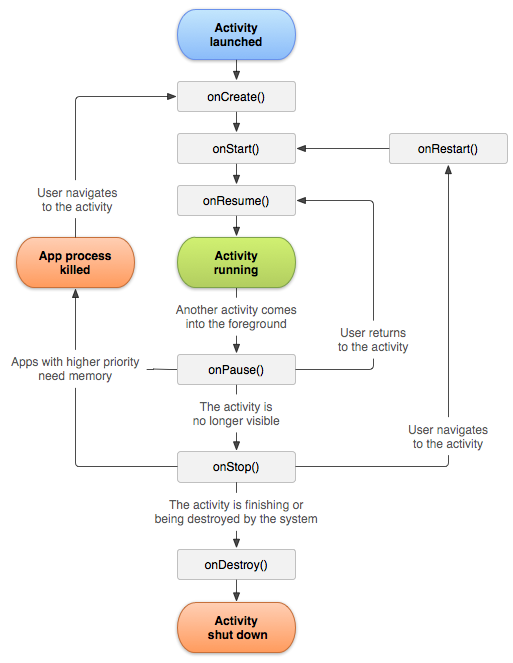


Диаграмма состояний жизненного цикла от Google [9.](https://info448.github.io/activities.html#fn9) См. также [альтернативную, упрощенную диаграмму](http://developer.android.com/images/training/basics/basic-lifecycle.png) .

В жизненном цикле активности происходит 7 «событий», которые обозначаются функцией *обратного вызова* , которую они выполняют:

* **onCreate()**: вызывается, когда Activity **впервые** создается/создается. Здесь вы инициализируете UI (например, указываете, какой макет использовать), а в противном случае выполняете ту работу, которая может быть в конструкторе.
* **onStart()**: вызывается непосредственно перед тем, как действие становится **видимым** для пользователя.

Разница между onStart()и onCreate()заключается в том, что его onStart()можно вызывать более одного раза (например, если вы покидаете Activity, тем самым скрывая его, и возвращаетесь позже, чтобы снова сделать его видимым).

* **onResume()**: вызывается непосредственно перед началом **взаимодействия с пользователем** , указывая, что Activity готово к использованию! Это немного похоже на то, когда Activity «имеет фокус».

While onStart()вызывается, когда Activity становится видимым, onResume()is вызывается, когда он готов к взаимодействию. Activity может быть видимым, но не интерактивным, например, если перед ним находится модальное всплывающее окно (частично скрывающее его). «onFocus» было бы лучшим названием для этого обратного вызова.

* **onPause()**: вызывается, когда система собирается запустить другую Activity (так что эта собирается потерять фокус). Это «зеркало» onResume(). *При паузе активность остается видимой!*

Этот обратный вызов обычно используется для *быстрого и временного* сохранения несохраненных изменений (например, сохранения черновика электронного письма в памяти) или остановки анимации или воспроизведения видео. Activity может быть закрыта (и, следовательно, находится на пути к выходу), а может просто терять фокус.

* **onStop()**: вызывается, когда Activity больше не видно. (например, другая Activity взяла верх, но также возможно, потому что текущая Activity была уничтожена). Этот обратный вызов является зеркалом onStart().

Этот обратный вызов — это то место, где вы должны сохранять любую информацию о состоянии (например, сохранение документа пользователя или состояния игры). Он предназначен для выполнения более сложной работы по «сохранению», чем onPause().

* **onRestart()**: вызывается, когда Activity возвращается из состояния «остановки». Это событие позволяет вам запускать отдельный код, когда приложение «перезапускается», а не создается в первый раз. Это наименее часто используемый обратный вызов жизненного цикла.
* **onDestroy()**: вызывается, когда Activity собирается закрыться. Это может произойти, если пользователь завершил приложение ***или*** (и это важно!) ОС пытается сэкономить память и поэтому самостоятельно завершает Activity.

Обратный onDestroy()вызов может выполнять окончательную очистку приложения, но лучше иметь такую ​​функциональность в onPause()или onStop(), поскольку они выполняются более надежно.

Действия *также* уничтожаются (и создаются заново) при изменении конфигурации устройства, например, если вы поворачиваете телефон!

Приложения Android работают на устройствах со значительными аппаратными ограничениями с точки зрения как памяти, так и времени работы батареи. Таким образом, ОС Android очень агрессивно относится к тому, чтобы не оставлять приложения запущенными «в фоновом режиме». Если она определяет, что приложение больше не нужно (например, потому что оно было скрыто некоторое время), это приложение будет уничтожено (закрыто). Обратите внимание, что это уничтожение непредсказуемо, поскольку «необходимость» открытия приложения зависит от правил распределения ресурсов ОС.

Таким образом, на практике следует реализовывать действия так, как будто они могут быть уничтожены в любой момент — нельзя рассчитывать на то, что они продолжат выполняться, если они не видны.

Обратите внимание, что приложениям может не потребоваться использовать все эти обратные вызовы! Например, если нет разницы между началом с нуля и возобновлением с остановки, то вам не нужен onRestart()(since onStart()идет посередине). Аналогично, onStart()может не понадобиться, если вы просто используете onCreate()and onResume(). Но эти жизненные циклы обеспечивают большую детализацию и возможность избегать дублирования кода.

**Переопределение методов обратного вызова**

При создании нового Empty обратный MainActivityвызов onCreate()уже переопределен, поскольку именно там указан макет.

Обратите внимание, что этот обратный вызов принимает Bundleв качестве параметра. A [Bundle](https://developer.android.com/reference/android/os/Bundle.html)— это объект, который хранит пары **ключ-значение** , как super-simple HashMap(или Object в JavaScript, или dictionary в Python). Bundles могут содержать только базовые типы (числа, строки) и поэтому используются для временного «объединения» *небольших* объемов информации. Подробности см. выше.

Также обратите внимание, что мы вызываем super.onCreate(). ***Всегда вызывайте цепочку наследования!*** . Это позволяет поведению на уровне системы продолжаться без каких-либо проблем.

Мы также можем добавить другие обратные вызовы: например, onStart()(см. [документацию](https://developer.android.com/guide/components/activities/activity-lifecycle.html#lc) для примеров). Опять же, функция автозаполнения IDE позволяет вам просто ввести имя обратного вызова и получить полную сигнатуру метода бесплатно!

Мы можем быстро добавить в события обратные вызовы и Log.v()вызовы для подтверждения их выполнения. Затем вы можете использовать телефон, чтобы увидеть, как они происходят:

* onCreate()и вызываются при создании экземпляра приложения onStart().onResume()
* Вы можете просмотреть onPause()активность, перетащив вниз панель уведомлений из верхней части экрана.
* Вы можете просмотреть onStop()активность, вернувшись на главный экран (нажмите на кружок внизу).
* Вы можете выполнить onDestroy()действие, изменив конфигурацию и повернув телефон: нажмите кнопку «повернуть» на панели инструментов эмулятора.

**Сохранение и восстановление состояния активности**

Как упоминалось выше, onCreate()метод Activity принимает в Bundleкачестве параметра. Это Bundleиспользуется для хранения информации о текущем состоянии Activity, так что если Activity будет уничтожено и создано заново (например, при повороте телефона), его можно будет [восстановить в том же состоянии](https://developer.android.com/guide/components/activities/activity-lifecycle.html#saras) , и пользователь не потеряет никаких данных.

Например, Bundleможет хранить информацию о состоянии для элементов View, например, какой текст пользователь ввел в EditText. Таким образом, когда пользователь поворачивает свой телефон, он не потеряет введенные данные формы! Если View был присвоен атрибут android:id, то этот идентификатор используется для *автоматического* сохранения состояния этого View, без дополнительных усилий с вашей стороны. Поэтому вы всегда должны указывать идентификаторы входных View!

Вы также можете добавить собственную пользовательскую информацию в , Bundleпереопределив обратный вызов Activity onSaveInstanceState()(используйте тот, AppCompactActivityкоторый принимает только один параметр). Он принимает в качестве параметра , Bundleкоторый создается с сохраненными данными: вы можете добавить больше информации в этот Bundle, используя соответствующий put()метод (похожий на метод, используемый для Maps, но чувствительный к типу):

*//java*

*//declare map key as a constant*

**private** static final String MSG\_KEY = "message\_key";

@Override

**protected** void onSaveInstanceState(Bundle outState) {

*//put value "Hello World" in bundle with specified key*

outState.putString(MSG\_KEY, "Hello World");

**super**.onSaveInstanceState(outState);

}

*//kotlin*

*//declare map key as a constant*

**private** **val** MSG\_KEY = "message\_key";

**override** **fun** onSaveInstanceState(outState: Bundle) {

*//put value "Hello World" in bundle with specified key*

outState.putString(MSG\_KEY, "Hello World");

**super**.onSaveInstanceState(outState);

}

* Обратите внимание, что ключи Bundle всегда следует объявлять как *КОНСТАНТЫ,* чтобы облегчить чтение/изменение и выявить опечатки.
* Обязательно всегда вызывайте, super.onSaveInstanceState()чтобы суперкласс мог выполнить свою работу по сохранению состояния иерархии View! Фактически, причина, по которой View «автоматически» сохраняют свое состояние, заключается в том, что этот метод вызывает их собственный onSaveInstanceState()обратный вызов.

Вы можете получить доступ к этому сохраненному Bundle из метода Activity, onCreate()когда Activity воссоздается. Обратите внимание, что если Activity создается впервые *,* то Bundle будет null— проверка на значение null, таким образом, является хорошим способом проверить, воссоздается ли Activity или нет:

*//java*

**protected** void onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

**if**(savedInstanceState != **null**){ *//Activity has been recreated*

String msg = savedInstanceState.getString(MSG\_KEY);

}

**else** { *//Activity created for first time*

}

}

*//kotlin*

**override** **fun** onCreate(savedInstanceState: Bundle?) {

**super**.onCreate(savedInstanceState);

**if**(savedInstanceState != **null**){ *//Activity has been recreated*

**val** msg:**String** = savedInstanceState.getString(MSG\_KEY);

}

**else** { *//Activity created for first time*

}

}

Помните, что a Bundleможет содержать только *небольшое* количество примитивных данных: всего несколько чисел или строк. Для более сложных данных вам нужно будет использовать альтернативные решения для хранения данных, которые будут обсуждаться в последующих лекциях.

**3.3 Контекст**

Если вы посмотрите документацию по [Activity](https://developer.android.com/reference/android/app/Activity.html)классу, то заметите, что он сам является подклассом чего-то под названием [10.](https://info448.github.io/activities.html#fn10) Это **абстрактный класс** , который действует как ссылка на информацию о текущей рабочей среде : он представляет данные об окружающей среде (например, «Какая ОС запущена? Подключена ли клавиатура?»).[**Context**](http://developer.android.com/reference/android/content/Context.html)Context

* Вы можете *думать* о Контексте как о представлении «Приложения», хотя это нечто большее ( [Application](https://developer.android.com/reference/android/app/Application.html)на самом деле это подкласс Context!)

Контекст *используется* для выполнения действий «уровня приложения»: в основном для работы с ресурсами (доступа к ним и их загрузки), но также для связи между Activity. По сути, он позволяет нам ссылаться на состояние, в котором мы работаем: «контекст» для нашего кода (например, «где это происходит?»). Это своего рода *отражение* или метапрограммирование, в некотором роде. Например, getResources()метод, обсуждаемый в последней главе, является методом класса Context, потому что нам нужно иметь какой-то способ указать, *какой* набор ресурсов загружать!

Существует несколько различных видов контекстов, на которые мы могли бы сослаться:

* Контекст приложения (например, Applicationобъект) ссылается на состояние всего приложения. По сути, это объект Java, который построен из <application>элемента в Manifest (и поэтому содержит этот уровень информации).
* Контекст Activity (например, Activityобъект), который ссылается на состояние этой Activity. Опять же, это примерно соответствует объектам Java, созданным из <activity>элементов из Manifest.

Каждый из этих Contextобъектов существует в течение жизни соответствующего компонента: то есть, ActivityКонтекст доступен, пока существует Activity (исчезает после onDestroy()), тогда как ApplicationКонтексты существуют, пока существует приложение. Мы почти всегда будем использовать Activityконтекст, так как это безопаснее и с меньшей вероятностью вызовет утечки памяти.

Внутри Activityобъекта (например, в функции обратного вызова жизненного цикла) можно ссылаться на текущий Activityс помощью this. И поскольку Activityэто Context, можно также использовать thisдля ссылки на текущий контекст Activity. Вы часто будете видеть Contextметоды, например, getResources()называемые недекорированными методами (без явного this).

Вам нужно будет ссылаться на Context всякий раз, когда вы захотите сделать что-то за пределами Activity, с которой вы работаете: будь то доступ к ресурсам, отображение [Toast](https://developer.android.com/guide/topics/ui/notifiers/toasts.html) (первый параметр — Toast.makeText()) Contextили открытие другой Activity.

**3.4 Множественные действия**

Весь смысл взаимодействия с жизненным циклом Activity заключается в том, чтобы справиться с тем фактом, что приложения Android могут иметь несколько действий и взаимодействовать с несколькими другими приложениями. В этом разделе мы кратко обсудим, как включить несколько действий в приложение (чтобы почувствовать, как жизненный цикл может повлиять на них). Обратите внимание, что работа с несколькими действиями будет более подробно рассмотрена в следующей лекции.

Мы можем легко создать New Activity через Android Studio, используя File > New > Activity. Мы также могли бы просто добавить новый файл .javaили .ktфайл с классом Activity в нем, но использование Android Studio также предоставит onCreate()заглушку метода, а также ресурс макета.

* Для практики создайте новую **пустую** активность под названием SecondActivity. Вам следует отредактировать ресурс макета этой активности так, чтобы <TextView>отображалось соответствующее сообщение.

Важно, что для каждого созданного вами Activity в файл **Manifest** добавляется запись AndroidManifest.xml. Этот файл действует как *«таблица содержания»* для нашего приложения, предоставляя информацию о том, как выглядит ваше приложение (то есть какие Activity у него есть), чтобы ОС могла открывать соответствующие Activities по мере необходимости. (Если вы создаете файл Activity .javaвручную, вам также нужно будет вручную добавить эту запись).

Действия перечислены как <activity>элементы, вложенные в <application>элемент. Если вы просмотрите файл, вы сможете увидеть элемент, представляющий первый MainActivity; дочерние элементы этой записи будут рассмотрены позже.

* Мы можем добавлять android:labelатрибуты к этим <activity>элементам, чтобы дать действиям более удобные отображаемые имена (например, в ActionBar).

**Намерения**

В Android мы не начинаем новые Activity, создавая их экземпляры (помните, *мы никогда не создаем экземпляры Activities* !). Вместо этого мы отправляем операционной системе сообщение с просьбой, чтобы Activity выполнила определенное действие (т. е. запустилась и отобразилась на экране). Эти сообщения называются [**Intents**](https://developer.android.com/guide/components/intents-filters.html) и используются для связи между компонентами приложения, такими как Activities. Система Intent позволяет Activities взаимодействовать, даже если у них нет ссылок друг на друга (мы не можем просто вызвать метод для этой другой Activity).

* У меня нет веского обоснования для такого названия, кроме того, что Intents объявляет «намерение» ОС что-то сделать (например, начать Activity).
* Вы можете думать о намерениях как о письмах, которые вы отправляете по почте: они адресованы определенной цели (например, другому действию или, точнее, контексту *)* и содержат краткое сообщение о том, что нужно сделать.

An [Intent](http://developer.android.com/reference/android/content/Intent.html)— это объект, который мы *можем* создать: например, мы можем создать new Intentв обработчике событий, когда мы нажимаем кнопку на MainActivity. IntentКласс имеет несколько различных конструкторов, но тот, с которого мы начнем, выглядит так:

*//java*

Intent intent = **new** Intent(MainActivity.this, SecondActivity.class);

*//kotlin*

**val** intent = Intent(this@MainActivity, SecondActivity::**class**.java)

Первый параметр — это то, Contextс помощью чего будет доставлено это намерение (например, this). Обратите внимание, что мы используем полностью определенное this@MainActivity, чтобы указать, что мы не говорим об анонимном классе обработчика событий.

Вторым параметром этого конструктора является то, Classкуда мы хотим отправить Intent ( ::classотносится к classпеременной, определенной в SecondActivity, а .javaсвойство получает ссылку на класс Java, которую ожидает Android; это метапрограммирование!). По сути, это «адрес» на конверте для сообщения, которое мы отправляем.

После создания экземпляра Intentмы можем использовать это сообщение для запуска Activity, вызвав [startActivity()](http://developer.android.com/reference/android/app/Activity.html#startActivity(android.content.Intent))метод (унаследованный от Activity) и передав ему Intent:

startActivity(intent);

Этот метод «отправит» сообщение операционной системе, которая передаст Intent соответствующему Activity, указав этому Activity запуститься, как только оно получит сообщение.

При таком взаимодействии мы теперь можем нажать кнопку, чтобы запустить второе действие (и посмотреть, как это повлияет на наши обратные вызовы жизненного цикла).

* И мы можем использовать кнопку **«Назад»,** чтобы вернуться назад!

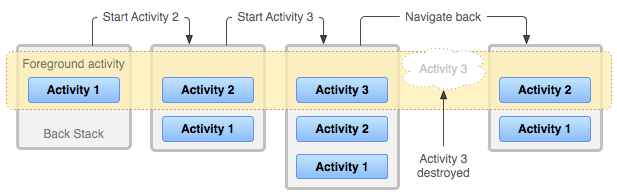
На самом деле есть несколько разных видов Intents(это **Явное Намерение** , потому что оно явно указывает, в какую Активность оно отправлено), и мы можем сделать с ними гораздо больше. Мы более подробно рассмотрим Намерения в следующей лекции; сейчас мы сосредоточимся в основном на Одиночных Активностях.

* Например, если вы посмотрите на Манифест, вы увидите, что у него MainActivityесть <intent-filter>дочерний элемент, который позволяет ему получать определенные виды Намерений, включая те, которые возникают при первом запуске приложения!

**Назад и задачи**

Мы показали, что у нас может быть много Activity (и, конечно, гораздо больше может существовать в нескольких приложениях), и мы можем перемещаться между ними, отправляя Intents и нажимая кнопку «Назад». Но как именно эта кнопка «Назад» может отслеживать, куда перейти?

Абстрактный тип данных, обычно связанный с функциональностью «назад» или «отменить», — это **стек** , и это именно то, что использует Android. Каждый раз, когда вы *начинаете* новую Activity, Android создает экземпляр этого объекта и помещает его на вершину стека. Затем, когда вы нажимаете кнопку «назад», эта activity «выталкивается» из стека, и вы попадаете в Activity, которая теперь находится на вершине.



Пример стека Activity от Google [11](https://info448.github.io/activities.html#fn11) .

Однако у вас могут быть разные «последовательности» действий, над которыми вы работаете: возможно, вы начинаете писать электронное письмо, а затем переходите к проверке ленты Twitter через другой набор действий. Android разбивает эти последовательности на группы, называемые [**задачами**](https://developer.android.com/guide/components/activities/tasks-and-back-stack.html) . *Задача* — это набор действий, организованных в стек, и в фоновом режиме вашего устройства может быть несколько задач.

Задачи обычно начинаются с «Домашнего экрана» Android, затем запуск приложения запускает новую задачу. Запуск новых действий из этого приложения добавит их в стек задачи. Если вы вернетесь *на* Домашний экран, текущая задача будет перемещена на задний план, поэтому кнопка «назад» не позволит вам перемещаться по этому стеку.

* Полезно рассматривать Задачи как различные вкладки в веб-браузере, где «стек возврата» представляет собой историю веб-страниц, посещенных в пределах этой вкладки.
* В качестве демонстрации попробуйте переключиться на другое (встроенное) приложение, а затем вернуться к примеру приложения. Как работает кнопка «Назад» в каждой ситуации?

Важное предостережение: Задачи отличаются друг от друга, поэтому вы можете иметь разные копии одной и той же Activity в нескольких стеках (например, активность Camera может быть частью задач приложений Facebook и Twitter, если вы увлекаетесь селфи). Однако это поведение можно изменить, см. [Управление задачами](https://developer.android.com/guide/components/activities/tasks-and-back-stack.html#ManagingTasks)

**Навигация вверх**

Мы можем сделать эту «обратную» навигацию немного более интуитивно понятной для пользователей, предоставив явную [навигацию вверх](https://developer.android.com/design/patterns/navigation.html) , а не просто заставляя пользователей возвращаться к Activity в том порядке, в котором они их просматривали (например, если вы пролистываете электронные письма и хотите вернуться к домашнему списку). Для этого нам нужно просто добавить немного конфигурации к нашим Activity:

* В коде Java мы хотим добавить больше функциональности ActionBar. *Подумайте* : в какой обратный вызов жизненного цикла следует поместить эту спецификацию?
* *//java*
* *//specify that the ActionBar should have an "home" button*

getSupportActionBar().setHomeButtonEnabled(**true**);

*//kotlin*

*//can access the supportActoinBar directly (though assert it is not null)*

supportActionBar!!.setHomeButtonEnabled(**true**)

* Затем в **Manifest** добавьте android:parentActivityNameатрибут к SecondActivity, со значением, равным полному имени класса (включая package **и** appname!) вашего MainActivity. Это позволит вам использовать визуальные элементы «back» (например, ActionBar) для возврата к «родительской» активности. Подробнее см. в [разделе Up Navigation .](https://developer.android.com/training/implementing-navigation/ancestral.html)
* **<activity** android:name=".SecondActivity"
* android:label="Second Activity"
* android:parentActivityName="edu.uw.activitydemo.MainActivity"**>**
* **<meta-data**
* android:name="android.support.PARENT\_ACTIVITY"
* android:value="edu.uw.activitydemo.MainActivity" **/>**

**</activity>**

Элемент <meta-data>обеспечивает обратную совместимость для API уровня 15 (поскольку android:parentActivityNameатрибут определен только для API уровня 16+).

1. <https://developer.android.com/reference/android/app/Activity.html>[↩](https://info448.github.io/activities.html#fnref8)
2. <http://developer.android.com/images/activity_lifecycle.png>[↩](https://info448.github.io/activities.html#fnref9)
3. <https://developer.android.com/reference/android/content/Context.html>[↩](https://info448.github.io/activities.html#fnref10)
4. <http://developer.android.com/images/fundamentals/diagram_backstack.png>[↩](https://info448.github.io/activities.html#fnref11)