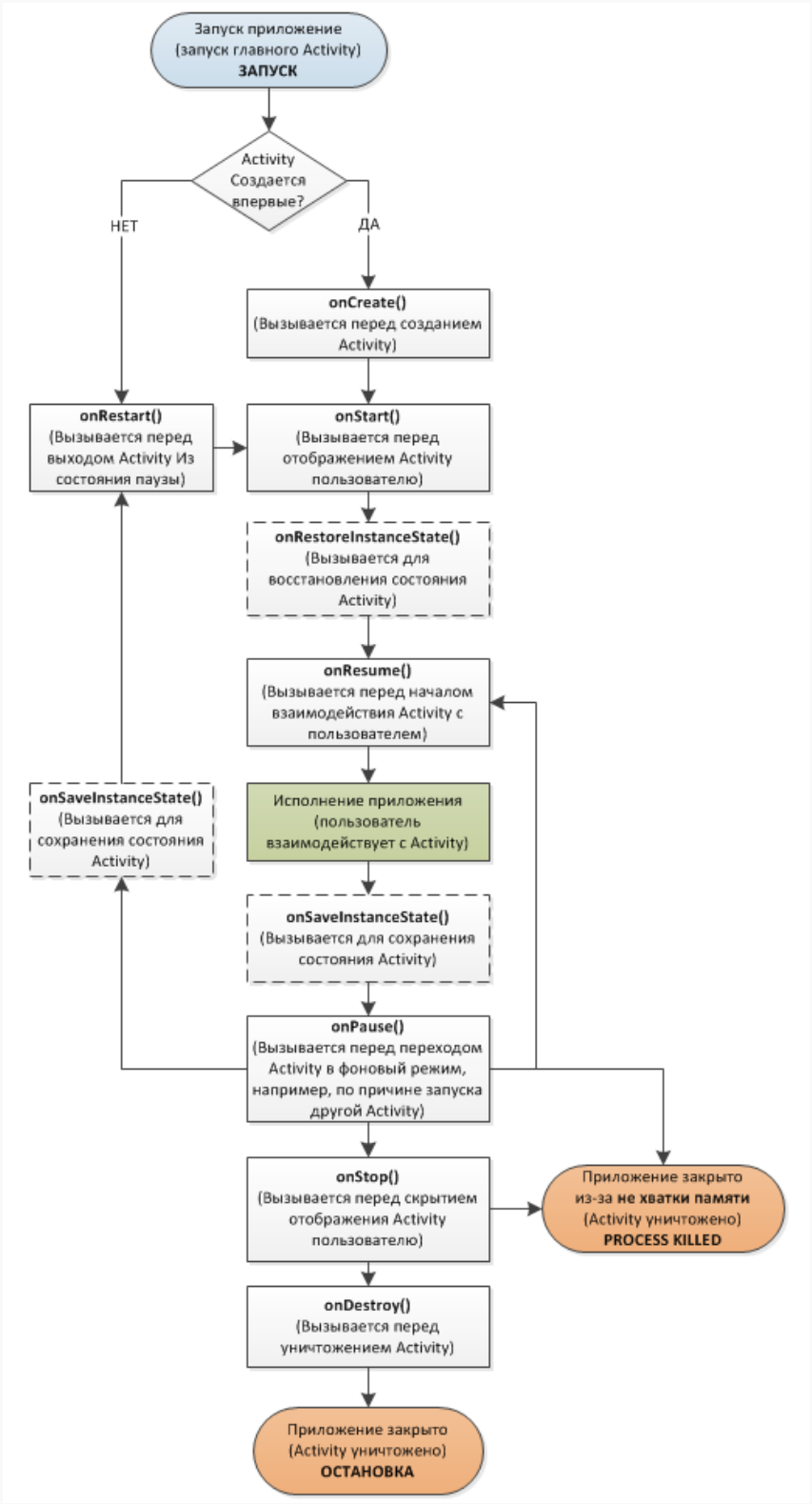
Жизненный цикл Android activity

Activity или действие можно рассматривать, как один экран (окно) вашего приложения для Google Android (андроид). Когда вы запускаете игру, появляется экран с меню - это первое Activity. Когда вы выбираете пункт меню, вы вызываете новое Activity.

Что в этом случае происходит?

На этот вопрос, мы постараемся ответить разобрав модель жизненного цикла приложения.

Модель жизненного цикла приложения можно изобразить в виде следующей схемы:



Обратите внимание, после метода onPause() и onStop() система может просто завершить ваш процесс (process killed) и методы onStop() и onDestroy() не будут вызваны. Это вполне обычная ситуация, поэтому с самого начала разработки вашего приложения для Google Android (андроид) вы должны учитывать возможность возникновения такого события.

Также необходимо понимать, когда Activity остановлено или приостановлено его состояние сохраняется т.к. Activity все еще хранится в памяти, и вся информация о его элементах и текущем состоянии ни куда не девается.

Но, когда система завершает процесс для того, чтобы освободить память (process killed), уже нет возможности возобновить состояние Activity. Как правило, пользователь об этом ничего не подозревает и ожидает увидеть Activity таким, каким он его запомнил. В данной ситуации, мы можем помочь сохранить состояние Activity, реализовав дополнительный метод onSaveInstanceState(Bundle).

Затем, при помощи любого из методов onCreate(Bundle) и onRestoreInstanceState(Bundle) мы можете извлечь сохраненное состояние Activity для восстановления.

Понимать то, как работает модель жизненного цикла приложения очень важно при построении приложений. Ниже, я приведу ряд рекомендаций по применению методов:

- onCreate() - используем для создания пользовательского интерфейса;
- onResume() - используем для запуска анимации или музыки, вызываем заставку;
- onPause() - используем для сохранения значений в базе данных т.к. далее, может произойти закрытие приложения системой (process killed);

В одной из книг Эда Бурнета, был предложен способ быстрого тестирования корректной работы кода по сохранению состояния приложения. Способ этот заключается в том, что изменение ориентации экрана (между портретным и ландшафтным режимом) приводит к тому, что текущая Activity проходит весь жизненный цикл. В процессе разработки приложений, мы будем использовать предложенный способ тестирования.

Основные методы жизненного цикла приложения

onCreate()

Вызывается при создании активности. Система может запускать и останавливать текущие окна в зависимости от происходящих событий. Android вызывает метод onCreate() после запуска или перезапуска Activity. Внутри этого метода настраивают статический интерфейс активности. Инициализирует статические данные активности, связывают данные со списками и т. д.. Связывает с необходимыми данными и ресурсами. Задает внешний вид через метод setContentView().

В этом методе загружайте пользовательский интерфейс, размещайте ссылки на свойства класса, связывайте данные с элементами управления, создавайте Сервисы и потоки. Метод onCreate() принимает объект Bundle, содержащий состояние пользовательского интерфейса, сохраненное в последнем вызове обработчика onSaveInstanceState. Для восстановления графического интерфейса в его предыдущем состоянии нужно задействовать эту переменную: внутри onCreate() или переопределяя метод onRestoreInstanceState().

Операции по инициализации, занимающие много времени, следует выполнять в фоновом процессе, а не с помощью метода onCreate(). В противном случае можно получить диалоговое окно ANR (Application Not Responding, приложение не отвечает).

В методе можно сделать проверку, запущено ли приложение впервые или восстановлено из памяти. Если значение переменной savedInstanceState будет null, приложение запускается первый раз:

```
// Приложение запущено впервые или восстановлено из памяти?
if ( savedInstanceState == null )   // приложение запущено впервые
{
   currentBillTotal = 0.0;    // инициализация суммы счета нулем
   // другой код
}
else // приложение восстановлено из памяти
{
     // инициализация суммы счета сохраненной в памяти суммой
     currentBillTotal = savedInstanceState.getDouble(BILL_TOTAL);
}
```

А значение переменной currentBillTotal можно сохранить в методе onSaveInstanceState():

```
@Override
protected void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);

        outState.putDouble(BILL_TOTAL, currentBillTotal);
} // end method onSaveInstanceState
```

onStart()

За onCreate() всегда следует вызов onStart(), но **перед onStart() не обязательно должен идти onCreate(),** так как onStart() **может вызываться и для возобновления работы приостановленного приложения** (приложение останавливается методом onStop()). При вызове onStart() окно еще не видно пользователю, но вскоре будет видно. Вызывается непосредственно перед тем, как активность становится видимой пользователю. Сопровождается вызовом метода onResume(), если активность получает передний план, или вызовом метода onStop(), если становится скрытой.

onResume()

Метод onResume() вызывается после onStart(), даже когда окно работает в приоритетном режиме и пользователь может его наблюдать. В этот момент пользователь взаимодеиствует с созданным вами окном. Приложение получает монопольные ресурсы. Запускает воспроизведение анимации, аудио и видео. Также может вызываться после onPause().

Имейте в виду, что система вызывает этот метод каждый раз, когда ваша активность идёт на переднем плане, в том числе, при первом создании. Таким образом, вы должны реализовать **onResume() для инициализации компонентов**, **которые вы освободили в OnPause()** и выполнять любые другие инициализации, которые должны происходить, когда активность вновь активна.

Пытайтесь размещать относительно быстрый и легковесный код, чтобы ваше приложение оставалось отзывчивым при скрытии с экрана или выходе на передний план.

Метод onResume может быть довольно легковесным. Вам не нужно перезагружать состояние пользовательского интерфейса внутри него, так как эти функции возложены на обработчики onCreate и onRestoreInstanceState.

Используйте метод **onResume для регистрации любых широковещательных приемников или других процессов**, которые должны приостанавливаться внутри обработчика onPause().

Например, после метода onPause(), в котором мы приостановили работу камеры (см. ниже) снова запускаем камеру:

```
@Override
public void onResume() {
    super.onResume();

    // Get the Camera instance as the activity achieves full user focus
    if (mCamera == null) {
        initializeCamera(); // Local method to handle camera init
    }
}
```

onPause()

Когда пользователь решает перейти к работе с новым окном, система вызовет для прерываемого окна метод onPause(). По сути происходит свертывание активности. Сохраняет незафиксированные данные. Деактивирует и выпускает монопольные ресурсы. Останавливает воспроизведение видео, аудио и анимацию. От onPause() можно перейти к вызову либо onResume(), либо onStop().

В этом методе необходимо остановить анимацию и другие действия, которые загружают процессор. Зафиксировать несохранённые данные, например, черновик письма. Освободить системные ресурсы, например, обработку данных от GPS.

Пытайтесь размещать относительно быстрый и легковесный код, чтобы ваше приложение оставалось отзывчивым при скрытии с экрана или выходе на передний план.

В большинстве реализаций класса Activity переопределяется как минимум метод onPause(). Внутри него фиксируются несохраненные изменения, потому как после его выполнения работа активности может прерваться без предупреждения. Исходя из архитектуры своего приложения, вы также можете приостановить выполнение потоков, процессов или широковещательных приемников, пока активность не появится на переднем плане.

Например, при работе с камерой метод используется следующим образом:

```
@Override
public void onPause() {
    super.onPause();

    // Release the Camera because we don't need it when paused
    // and other activities might need to use it.
    if (mCamera != null) {
        mCamera.release()
        mCamera = null;
    }
}
```

В тоже время вы не должны использовать OnPause() для хранения пользовательских изменений (таких, как персональные данные, введенные в форму) для постоянного хранения. Искключение допускается, когда вы уверены, что пользователи ожидают изменения, которые будут автоматически сохранены (например, при составлении электронной почты). Тем не менее, вы должны избегать выполнения интенсивной работы в OnPause(), таких как запись в базе данных, так как это может замедлить переход к следующей активности (вместо него вы должны выполнять тяжелую нагрузку во время операции отключения OnStop()).

Когда активность приостановлена, то все компоненты сохраняются в памяти и при возобновления нет необходимости повторно инициализировать их.

onStop()

Метод onStop() вызывается, когда окно становится невидимым для пользователя. Это может произойти при ее уничтожении, или если была запущена другая активность (существующая или новая), перекрывшая окно текущей активности. Всегда сопровождает любой вызов метода onRestart(), если активность возвращается, чтобы взаимодействовать с пользователем, или метода onDestroy(), если эта активность уничтожается.

Когда ваша деятельность останавливается, объекты активности хранятся в памяти и восстанавливаются, когда активность возобновляется. Вам не нужно повторно инициализировать компоненты, которые были созданы ранее. Кроме того, система отслеживает текущее состояние для каждого представления, поэтому, если пользователь введёт текст в текстовое поле EditText, то его содержание сохраняется и вам не нужно сохранять и восстанавливать его.

Примечание: Даже если система закрыла вашу активность, когда она была остановлена, она по-прежнему сохраняет состояние объектов, таких как текст в EditText в специальном объекте Bundle (в виде ключ-значение) и восстанавливает их, если пользователь переходит обратно к тому же экземпляру активности.

В этом методе можно сделать сложные операции по сохранению данных: для приостановки сложной анимации, потоков, отслеживания показаний датчиков, запросов к GPS, таймеров, Сервисов или других процессов, которые нужны исключительно для обновления пользовательского интерфейса. Нет смысла потреблять ресурсы (такты центрального процессора или сетевой трафик) для обновления интерфейса, в то время как он не виден на экране. Примените методы onStart() или onRestart() для возобновления или повторного запуска этих процессов, когда Активность опять станет видимой.

onRestart()

Если окно возвращается в приоритетный режим после вызова onStop(), то в этом случае вызывается метод onRestart(). Т.е. вызывается после того, как активность была остановлена и снова была запущена пользователем. Всегда сопровождается вызовом метода onStart().

onRestart предшествует вызовам метода onStart (кроме самого первого). Используйте его для специальных действий, которые должны выполняться только при повторном запуске Активности в рамках «полноценного» состояния.

onDestroy()

Метод вызывается по окончании работы активности, при вызове метода finish() или в случае, когда система уничтожает этот экземпляр активности для освобождения ресурсов. Эти два сценария уничтожения можно определить вызовом метода isFinishing(). Вызывается перед уничтожением активности. Это последний запрос, который получает активность от системы. Если определенное окно находится в верхней позиции в стеке, но невидимо пользователю и система решает завершить это окно, вызывается метод onDestroy(). В этом случае метод удаляет все статические данные активности. Отдаёт все используемые ресурсы.

Так как все необходимые операции по освобождению ресурсов вы сделали в методе onStop(), то в этом методе вы можете подстраховаться и проверить ещё раз все неосвобождённые ресурсы.

На практике вам чаще всего придется сталкиваться с методами onCreate(), onResume() и onPause(). Метод onCreate() будет вызываться при создании пользовательского интерфейса для работы с окном. Данный метод позволит вам связывать данные с виджетами и подключать обработчики событий к компонентам пользовательского интерфейса. При помощи onPause() вы сможете сохранить важную информацию в базе данных вашего приложения. Это последний безопасный метод, который будет вызываться

перед тем, как система завершит работу приложения. Метод onDestroy() не обязательно будет вызываться, поэтому не полагайтесь на этот метод при реализации критическом логики