



Лекция №7
по курсу
«Мобильное программирование»
на тему «Жизненный цикл активностей»

Лектор: д.т.н., Оцоков Шамиль Алиевич,
email: otsokovShA@mpei.ru

Москва, 2021

Секундомер



Прошедшее время
в секундах.

Кнопка Start запускает
отсчет секунд.

Кнопка Stop останавливает
отсчет секунд.

Кнопка Reset обнуляет
счетчик.

Объекты Handler

В Java-программах, не предназначенных для Android, такие задачи выполняются с использованием фоновых потоков. В мире Android возникает проблема — только главный программный поток Android может обновлять пользовательский интерфейс, а любые попытки такого рода из других потоков порождают исключение `CalledFromWrongThreadException`.

Handler — класс Android, который может использоваться для планирования выполнения кода в некоторый момент в будущем. Также класс может использоваться для передачи кода, который должен выполняться в другом программном потоке.

В нашем примере Handler будет использоваться для планирования выполнения кода секундомера каждую секунду.

Чтобы использовать класс Handler, упакуйте код, который нужно запланировать, в объект Runnable, после чего используйте методы `post()` и `postDelayed()` класса Handler для определения того, когда должен выполняться код.

Объекты Handler

Метод post()

Метод `post()` передает код, который должен быть выполнен как можно скорее (обычно почти немедленно). Метод `post()` вызывается с одним параметром — объектом типа `Runnable`. Объект `Runnable` в мире `Android`, как и в традиционном языке `Java`, представляет выполняемое задание. Код, который требуется выполнить, помещается в метод `run()` объекта `Runnable`, а объект `Handler` позаботится о том, чтобы код был выполнен как можно быстрее.

Вызов метода выглядит так:

```
final Handler handler = new Handler();  
handler.post(Runnable);
```

Метод `postDelayed()`

Метод `postDelayed()` работает почти так же, как `post()`, но выполнение кода планируется на некоторый момент в будущем. Метод `postDelayed()` получает два параметра: `Runnable` и `long`. Объект `Runnable` содержит выполняемый код в методе `run()`, а `long` задает задержку выполнения кода в миллисекундах. Код выполняется при первой возможности после истечения задержки. Вызов метода выглядит так:

Объекты Handler

```
Handler handler = new Handler();
```

```
handler.postDelayed(Runnable runnable, long  
milliseconds);
```

```
new Runnable () {  
    public void run() {
```

код, который нужно
выполнить...

```
    }  
}
```

Объекты Handler

```
final Handler handler = new Handler();  
handler.postDelayed(Runnable, long);
```

Чтобы обновить секундомер, мы будем многократно планировать выполнение кода с использованием Handler; при этом каждый раз будет назначаться задержка продолжительностью 1000 миллисекунд. Каждое выполнение кода сопровождается увеличением переменной seconds и обновлением надписи.

Полный код метода runTimer():

```
private void runTimer() {  
final TextView timeView = (TextView)findViewById(R.id.time_view);  
final Handler handler = new Handler();  
handler.post(new Runnable() {
```

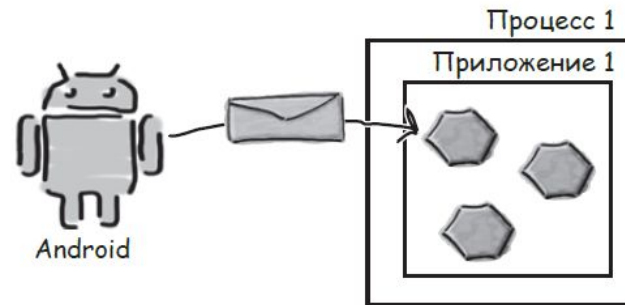
Вызов метода post() с передачей нового объекта Runnable. Метод post() обеспечивает выполнение без задержки, так что код в Runnable будет выполнен практически немедленно.

```
@Override  
public void run() {
```

Запуск приложения

```
int hours = seconds/3600;  
int minutes = (seconds%3600)/60;  
int secs = seconds%60;
```

1. Пользователь В файле AndroidManifest.xml приложения указано, какая активность должна использоваться как стартовая.
2. Android проверяет, существует ли процесс для этого приложения, и если не существует — создает новый процесс
3. Вызывается метод onCreate() активности
4. При завершении работы onCreate() м: устройстве



Поворот экрана

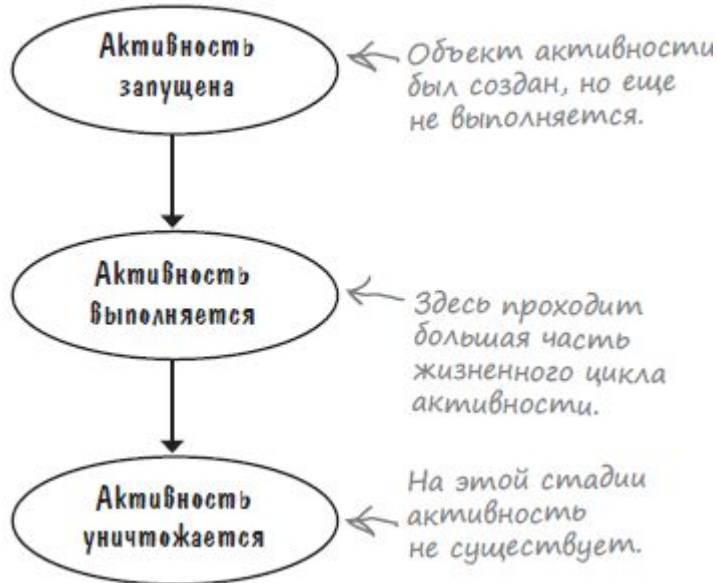
1. Пользователь запускает приложение и щелкает на кнопке Start, чтобы секундомер заработал.

2 Пользователь поворачивает устройство

3. Метод `onCreate()` выполняется заново, и вызывается метод `runTimer()`. Так как активность была создана заново, переменным `seconds` и `running` возвращаются значения по умолчанию (`running = false`)

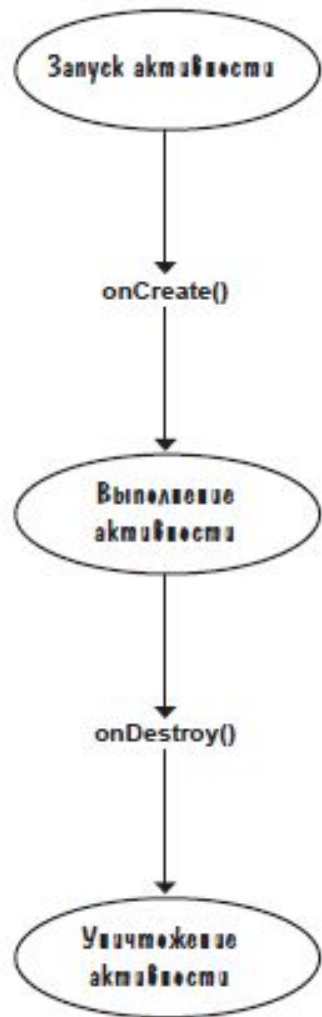
При изменении конфигурации устройства все компоненты приложения, отображающие пользовательский интерфейс, должны быть обновлены в соответствии с новой конфигурацией. Если повернуть устройство, Android замечает, что ориентация и размеры экрана изменились, и интерпретирует этот факт как изменение конфигурации устройства. Текущая активность уничтожается и создается заново, чтобы приложение могло выбрать ресурсы, соответствующие новой конфигурации

Поворот экрана



Активность выполняется, когда она находится на переднем плане на экране. Метод `onCreate()` вызывается при создании активности; именно здесь происходит основная настройка активности. Метод `onDestroy()` вызывается непосредственно перед уничтожением активности

Поворот экрана



- 1** **Запуск активности.**
Android создает объект активности и выполняет его конструктор.
- 2** **Метод `onCreate()` выполняется непосредственно после запуска активности.**
В методе `onCreate()` должен размещаться весь код инициализации, так как этот метод всегда вызывается после запуска активности и до начала ее выполнения.
- 3** **Во время выполнения активность находится на переднем плане, а пользователь может взаимодействовать с ней.**
В этой фазе проходит большая часть жизненного цикла активности.
- 4** **Метод `onDestroy()` вызывается непосредственно перед уничтожением активности.**
Метод `onDestroy()` позволяет выполнить любые необходимые завершающие действия — например, освободить ресурсы.
- 5** **После выполнения метода `onDestroy()` активность уничтожается.**
Активность перестает существовать.

Поворот экрана (два подхода)

1. Запретить Android перезапуск активности при изменениях в конфигурации.

2 Сохранить текущее состояние, чтобы активность могла воссоздать себя в том же состоянии

Недостаток первого подхода:

- Игнорируется встроенное поведение Android, а это может создать проблемы
- Возможно, самостоятельно обработать изменение конфигурации.

В файле `AndroidManifest.xml`: `android:configChanges="`

`изменение_конфигурации`»

```
<activity
```

```
android:name="com.hfad.stopwatch.StopwatchActivity"
```

```
android:label="@string/app_name"
```

```
android:configChanges="orientation|screenSize" >
```

```
public void onConfigurationChanged(Configuration config) {  
}
```

Поворот экрана (два подхода)

Второй подход

Сохранение текущего состояния активности и ее последующее воссоздание в методе `onCreate()`.

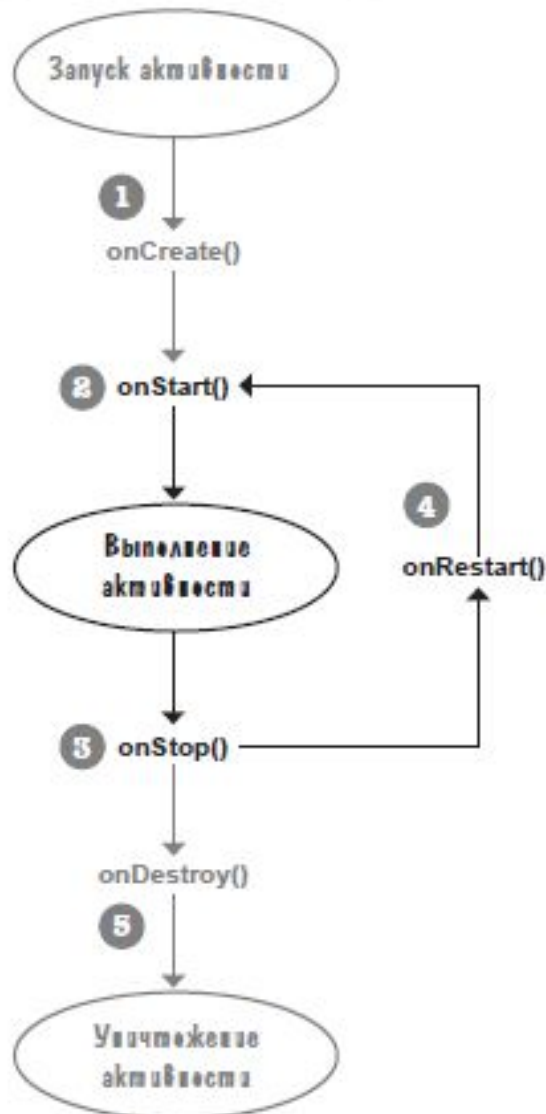
Чтобы сохранить текущее состояние активности, необходимо реализовать метод `onSaveInstanceState()`. Метод `onSaveInstanceState()` вызывается перед уничтожением активности; это означает, что вам представится возможность сохранить все значения, которые нужно сохранить, прежде чем они будут безвозвратно потеряны.

```
public void onSaveInstanceState(Bundle savedInstanceState)
{
}
```

Метод `onCreate()` получает параметр `Bundle`. Таким образом, если вы добавите значения переменных `running` и `seconds` в `Bundle`, метод `onCreate()` сможет восстановить их при повторном создании активности.

```
bundle.put*("name", value)
bundle.get*("name");
```

Жизненный цикл активностей



- 1** Активность запускается, выполняется ее метод `onCreate()`.
Выполняется код инициализации активности из метода `onCreate()`. В этой точке активность еще не видна, так как метод `onStart()` еще не вызывался.
- 2** Метод `onStart()` выполняется после метода `onCreate()`. Он вызывается, когда активность собирается стать видимой. После выполнения метода `onStart()` пользователь видит активность на экране.
- 3** Метод `onStop()` выполняется, когда активность перестает быть видимой пользователю. После выполнения метода `onStop()` активность не видна на экране.
- 4** Если активность снова становится видимой пользователю, вызывается метод `onRestart()`, за которым следует вызов `onStart()`.
Активность может проходить через этот цикл многократно, если она несколько раз скрывается и снова становится видимой.
- 5** Наконец, активность уничтожается. Метод `onStop()` обычно вызывается перед `onDestroy()`, но он может быть пропущен при серьезной нехватке памяти в системе.

Переопределение методов жизненного цикла

1. Реализовать метод `onStop()` активности, чтобы отсчет времени останавливался, если приложение стало невидимым

2. Реализовать метод `onStart()`, чтобы отсчет времени возобновлялся, когда приложение становится видимым.

В переопределениях методов жизненного цикла активности должен вызываться метод суперкласса. Если этого не сделать, то произойдет исключение

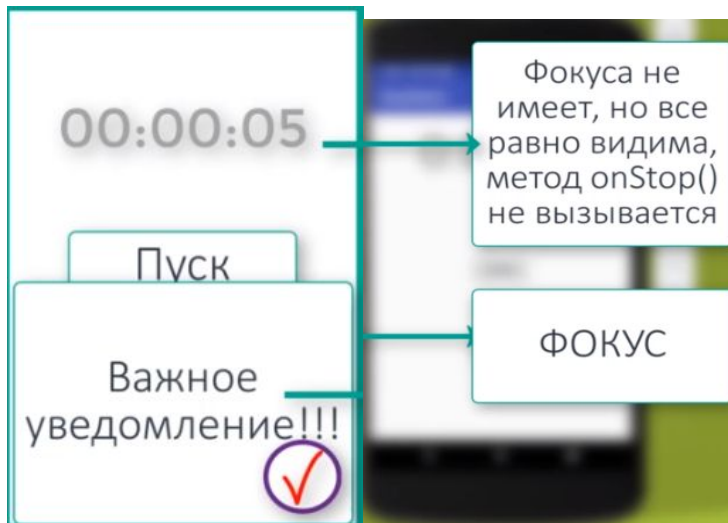
Изменить код активности так, чтобы если секундомер работал перед вызовом `onStop()`, он снова запускался после получения фокуса активностью

Метод on Pause

Частичная видимость активности

Активность находится в приостановленном состоянии, если она потеряла фокус, но остается видимой для пользователя. Такая активность продолжает существовать и сохраняет всю свою информацию состояния.

Метод `onPause()` вызывается тогда, когда ваша активность видима, но фокус принадлежит другой активности. Метод `onResume()` вызывается непосредственно перед тем, как ваша активность начинает взаимодействовать с пользователем.



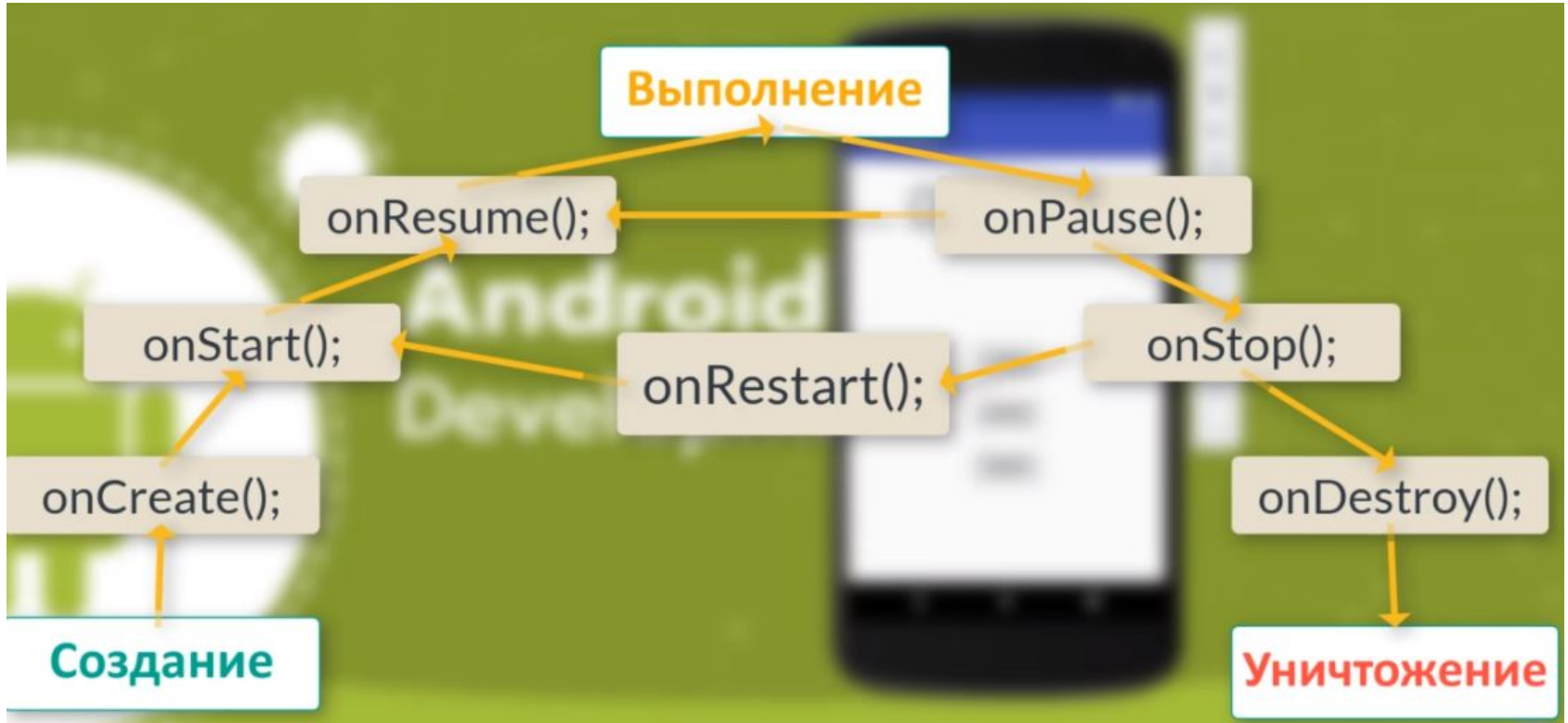
Метод on Pause

Частичная видимость активности

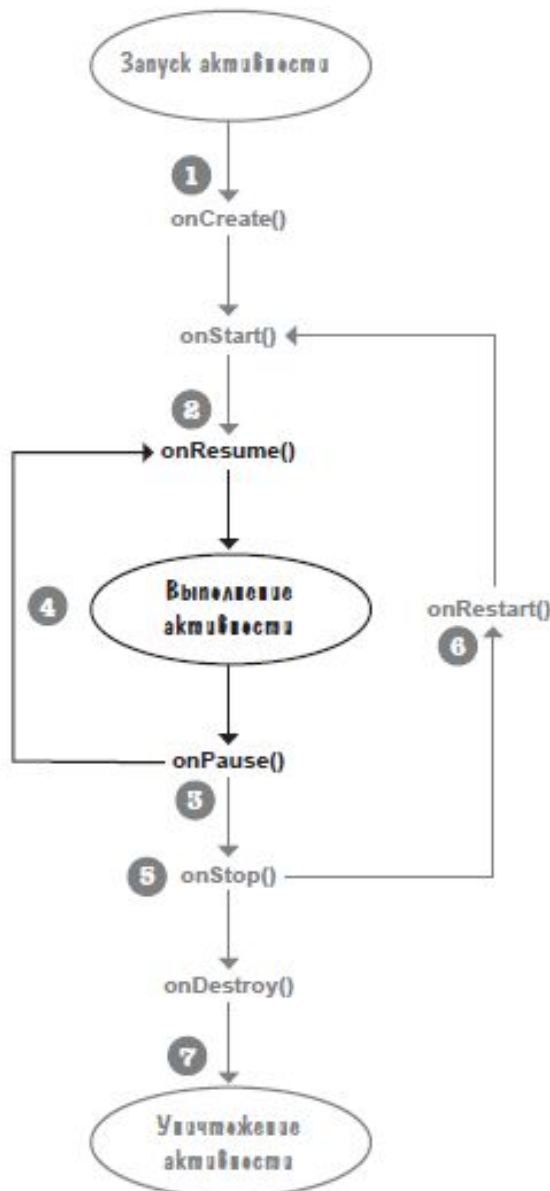
Активность находится в приостановленном состоянии, если она потеряла фокус, но остается видимой для пользователя. Такая активность продолжает существовать и сохраняет всю свою информацию состояния.

Метод `onPause()` вызывается тогда, когда ваша активность видима, но фокус принадлежит другой активности. Метод `onResume()` вызывается непосредственно перед тем, как ваша активность начинает взаимодействовать с пользователем.

Жизненный цикл активностей



Жизненный цикл активностей



1 Активность запускается, выполняются ее методы `onCreate()` и `onStart()`. В этой точке активность видна, но еще не обладает фокусом.

2 Метод `onResume()` вызывается после метода `onStart()`. Он выполняется тогда, когда активность собирается перейти на передний план. После выполнения метода `onResume()` активность обладает фокусом, а пользователь может взаимодействовать с ней.

3 Метод `onPause()` выполняется тогда, когда активность перестает находиться на переднем плане. После выполнения метода `onPause()` активность остается видимой, но не обладает фокусом.

4 Если активность снова возвращается на передний план, вызывается метод `onResume()`. Активность, которая многократно теряет и получает фокус, может проходить этот цикл несколько раз.

5 Если активность перестает быть видимой пользователю, вызывается метод `onStop()`. После выполнения метода `onStop()` активность не видна пользователю.

6 Если активность снова станет видимой, вызывается метод `onRestart()`, за которым следуют `onStart()` и `onResume()`. Активность может проходить через этот цикл многократно.

7 Наконец, активность уничтожается. При переходе от выполнения к уничтожению метод `onPause()` вызывается до того, как активность будет уничтожена. Также обычно при этом выполняется метод `onStop()`.