Лекция 2. Жизненный цикл Activity

Author: Ruslan V. Akhpashev

email: ruslan.akhpashev@gmail.com

github: https://github.com/fzybot

Введение

При запуске приложения "обычно" открывается окно приложения, которые вы видете на экране телефона - это так называемое **Activity**. Практически все Activity взаимодействуют с пользователем. В Android SDK имеется class Activity, создающий окно приложения, в которое вы можете поместить ваш пользовательский интерфейс (**User Interface**) с помощью метода setContentView(View).

MainActivity.java

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // savedInstanceState необходимо для сохранения "текущего" состояния.
        setContentView(R.layout.activity_main); // activity_main - наше окно.

    }
}
```

Каждое **Activity** имеет свой жизненный цикл (**Life Cycle**), т. е. фазы работы окна. Во время использования приложения, пользователь часто переходит из одного окна (**Activity**) на другое, уводит приложение в фоновый режим, закрывает его, возвращается обратно. Каждое из этих действий ведет за собой выполнение одного из методов жизненного цикла. В силах разработчика управлять поведением каждого **Activity**, когда пользователь открывает или возобновляет работу с окном.

> ***Например**:* разрабатывая видео-плеер, хотелось бы нажать видео на паузу и не воспроизводить в фоне, когда пользователь переходит в другое приложение. При этом, когда пользователь снова открывает его, видео желательно возобновить на том же месте и не загружать заново и т.д.

Таким образом, весьма удобно и эффективно управлять работой приложения, без лишних "внезапных сбоев".

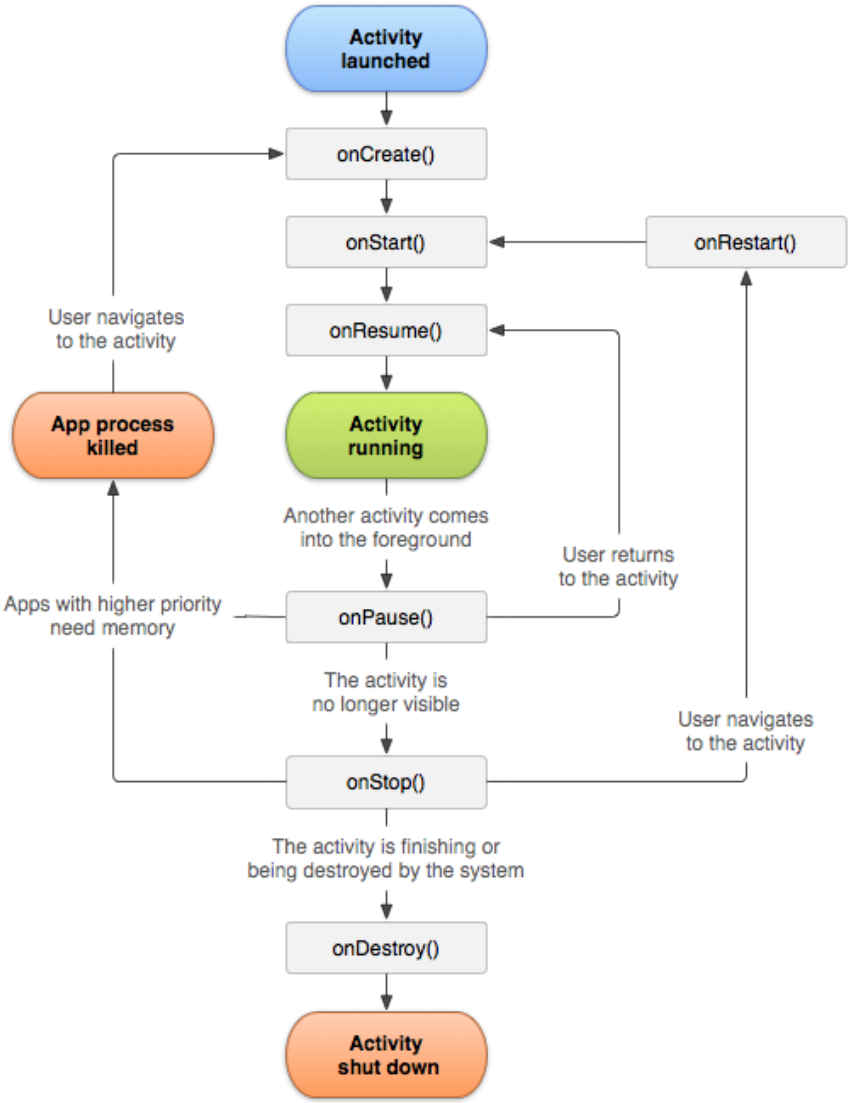
Например, исключить:

- Сбой приложения во время поступления телефонного звонка;
- Неэффективное потребление ресурсов мобильного устройства (процессорная мощность, оперативная память и т.д.);
- Потерю текущих "достижений" пользователя при возвращении в приложение спустя некоторое время;
- Сбой при повороте экрана между режимами альбомной ориентации и партретной.

Вот...

Жизненный цикл Activity (Activity-Lifecycle)

Для перехода между "стадиями" жизненного цикла Activity, класс Activity предоставляет 6 базовых методов (вызовов): onCreate(), onStart(), onResume(), onPause(), onStop(), onDestroy().



Изображение выше показывает упрощенную схему жизненного цикла **Activity**.

Методы (Lifecycle callbacks)

onCreate()

Самый важный из методов. Срабатывает, когда система впервые создает ваше окно программы. В данном **Activity** важно прописать базовую логику работы вашего окна, которая должна выполнится **ОДИН** раз за весь жизненный цикл **Activity**.

> ***Например***: в методе **onCreate()** можно определить (найти) все объекты, находящиеся в окне. В данном примере, мы определяем **TextView** по **id**.
>
> *MainActivity.java*

```
public class MainActivity extends AppCompatActivity {

    private String TAG = "Life Cycle";
    TextView testText;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Checking the instance state
        if(savedInstanceState == null){
            //it is the first time the fragment is being called
            Log.i(TAG, (String) "The activity is started first time");
        }else{
            //not the first time so we will check SavedInstanceState bundle
            Log.i(TAG, (String) "Here we need to load previous state");
        }
        setContentView(R.layout.activity_main);

        testText = (TextView) findViewById(R.id.HelloWorldView);
    }
}
```

Id текстового поля можно назначить в файле **.xml**. Вообще, файл .xml (язык разметки) реализует **UI (User interface)** нашего Activity, добавляет элементы (текстовые поля, фон, кнопки и т.д.), меняет стилистику элементов (цвет, шрифт, размер и т.д.), раотает на пользователя. Можно назвать это **Frontend**, а все что описывается в классе *MainActivity.java* - **Backend**.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        android:id="@+id/HelloWorldView" # ID for TextView
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

onStart()

Метод **onStart()** позволяет пользователю увидеть окно приложения. Метод включается в себя инициализацию кода для поддержки работы **UI**. Это удобный метод для отрисовки визуальных элементов окна, запуска анимации и т.д.

Метод довольно быстрый, не задерживается при запуске окна, после чего переходит в режим **onResume()**.

onResume()

В данном режиме окно программы непосредственно взаимодействует с пользователем. Этот режим работает постоянно, пока приложение находится в **"фокусе"** пользователя. Фокус может теряться, когда: *пользватель выходит из приложения, переходит на другое Activity, поступает звонок, при котором приложение автоматически уходит в фоновый режим и т.д*.

```
public class CameraComponent implements LifecycleObserver {

    ...

    @OnLifecycleEvent(Lifecycle.Event.ON_RESUME) // событие onResume()
    public void initializeCamera() {
        if (camera == null) {
            getCamera();
        }
    }
    ...
}
```

Код выше показывает пример включения камеры только в режиме **onResume()**. В случае многооконного режима, если вы хотите предоставить параллельную работу двух окон, лучше включать камеру в **onStart()** методе.

onPause()

Метод срабатывае, когда пользователь выходит из текущего **Activity**. На примере работы камеры, если пользователь не хочет оставлять камеру включенной при переходе в другое окно, нужно прописать ее отключение в методе **onPause()**.

> ***ВАЖНО***: не стоит в данном режиме сохранять пользовательские данные или текущее достижение в окне (текст, анимация и т.д.). лучше это сделать в **onStop()**.

```
public class JavaCameraComponent implements LifecycleObserver {

    ...

    @OnLifecycleEvent(Lifecycle.Event.ON_PAUSE)
    public void releaseCamera() {
        if (camera != null) {
            camera.release();
            camera = null;
        }
    }

    ...
}
```

onStop()

Система вызывает метод **onStop()**, когда пользователь больше не видит ваще **Activity**.

Идеальный момент для сохранения текущего состояния Activity (текст, анимация, текущий прогресс пользователя в приложении и т.д.). Необходимо сохранять, что бы при следующем запуске окна у пользователя не обновилась\обнулилась информация в окне.

```
@Override
protected void onStop() {
    // call the superclass method first
    super.onStop();

    // save the note's current draft, because the activity is stopping
    // and we want to be sure the current note progress isn't lost.
    ContentValues values = new ContentValues();
    values.put(NotePad.Notes.COLUMN_NAME_NOTE, getCurrentNoteText());
    values.put(NotePad.Notes.COLUMN_NAME_TITLE, getCurrentNoteTitle());
}
```

onDestroy()

Следует высвободить все ресурсы, которые не были высвобождены в методе **onStop()**.