Лекция №4 Понятие Activity и ЖЦ Activity

4.1 Понятие активности (Activity)

Каждая *активность (Activity)* – это экран (по аналогии с формой), который приложение может показывать пользователям. Чем сложнее создаваемое приложение, тем больше экранов (*Activity*) потребуется. При создании приложения потребуется, как минимум, начальный (главный) экран. При необходимости этот интерфейс дополняется второстепенными активностями, предназначенными для ввода информации, вывода и предоставления дополнительных возможностей. Запуск новой активности или возврат из нее приводит к «перемещению» между экранами.

Для создания новой активности необхоимо наследование от класса *Activity* или *AppCompatActivity*. Внутри реализации класса надо определить пользовательский интерфейс и реализовать требуемый функционал. Базовый каркас для новой активности выглядит следующим образом:

```
public class MainActivity extends AppCompatActivity  {

/** Вызывается при создании Активности */
@Override
protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState); }
}
```

Базовый класс *Activity* (*AppCompatActivity*) представляет собой пустой экран, который не особенно полезен, поэтому первое, что нужно сделать, это создать пользовательский интерфейс с помощью представлений (*View*) и разметки (*Layout*).

Для использования активности в приложении ее необходимо зарегестрировать в файле манифеста путем добавления элемента <*activity*> внутри узла <*application*>, в противном случае ее невозможно будет использовать:

```
<activity
    android:name=".MainActivity"
    android:label="@string/app_name"
    android:theme="@style/AppTheme.NoActionBar">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

В тег <*activity*> можно добавить элементы <*intent-filter*> для указания *намерений* (*Intent*), которые активность будет отслеживать.

4.2. Жизненный цикл Activity

Приложения Android не могут контролировать свой жизненный цикл. ОС сама управляет всеми процессами и, как следствие, активностями внутри них. При этом состояние активности помогает ОС определить приоритет родительской

активности. А приоритет приложения влияет на то, с какой вероятностью его работа (и работа дочерних активностей) будет прервана системой.

4.2.1. Стек Activity

Состояние каждой активности определяется ее позицией в стеке активностей, запущенных в данный момент. При запуске новой *Activity* представляемый ею экран помещается на вершину стека. Если пользователь нажимает кнопку *Назад* или эта активность закрывается каким-то другим образом, на вершину стека перемещается (и становится активной) нижележащая активность (рис. 4.1).

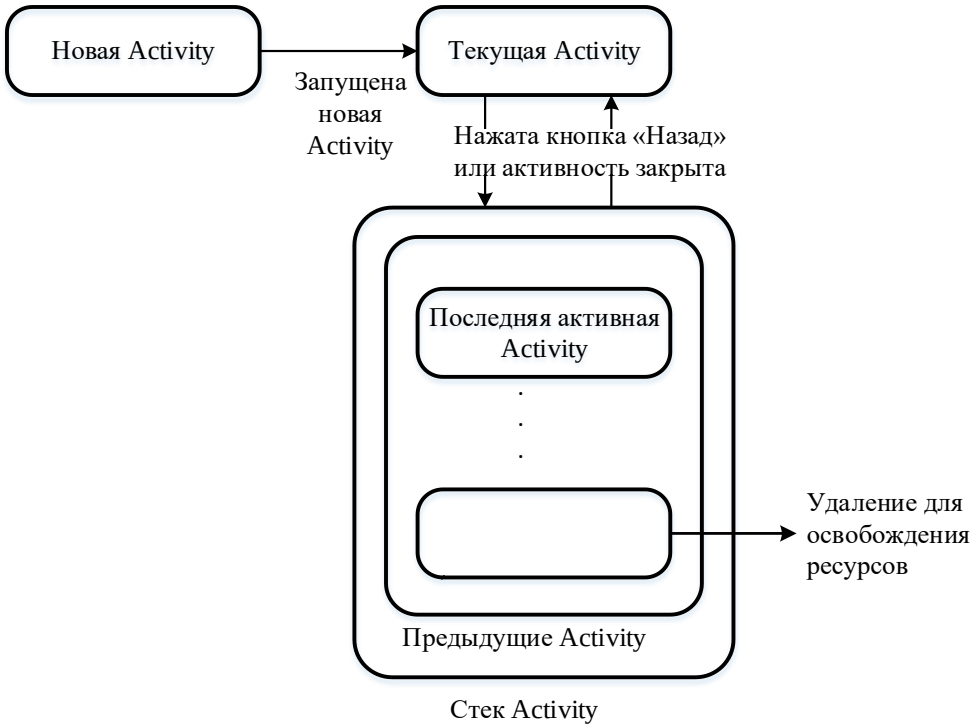


Рис.4.1. Стек активностей

На приоритет приложения влияет его самая приоритетная активность. Когда диспетчер памяти ОС решает, какую программу закрыть для освобождения ресурсов, он учитывает информацию в стеке для определения приоритета приложения.

4.2.2. Состояния Activity

Activity могут находиться в одном из четырех возможных состояний.

Active (активное). *Activity* находится на переднем плане (на вершине стека) и имеет возможность взаимодействовать с пользователем. Android будет пытаться сохранить ее работоспособность любой ценой, при необходимости прерывая работу других *Activity*, находящихся на более низких позициях в стеке. При выходе на передний план другой *Activity* работа данной Активности будет *приостановлена* или *остановлена*.

Paused (приостановленное). В этом состоянии *Activity* может быть видна на экране, но не может взаимодействовать с пользователем: в этот момент она

приостановлена. Это случается, когда на переднем плане находятся полупрозрачные или плавающие (например, диалоговые) окна. Работа приостановленной активности может быть прекращена, если ОС необходимо выделить ресурсы активности переднего плана. Если активность полностью исчезает с экрана, она *останавливается*.

Stopped (остановленное). *Activity* невидима, она находится в памяти, сохраняя информацию о своем состоянии. Такая активность становится кандидатом на преждевременное закрытие, если системе потребуется память. При остановке активности разработчику важно сохранить данные и текущее состояние пользовательского интерфейса (состояние полей ввода, позицию курсора и т. д.). Если активность завершает свою работу или закрывается, она становится *неактивной*.

Inactive (неактивное). Когда работа *Activity* завершена, она находится в неактивном состоянии. Такие активности удаляются из стека и должны быть перезапущены, чтобы их можно было использовать.

Изменение состояния приложения – недетерминированный процесс и управляется исключительно менеджером памяти Android. При необходимости Android вначале закрывает приложения, содержащие *неактивные Activity*, затем *остановленные* и в крайнем случае *приостановленные*.

Для обеспечения полноценного интерфейса приложения изменения его состояния должны быть незаметными для пользователя. При остановке или приостановке работы активности необходимо обеспечить сохранение ее состояния, которое можно восстановить при выходе активности на передний план. Для этого в классе *Activity* имеются обработчики событий, переопределение которых позволяет разработчику отслеживать изменение состояний активностей.

4.2.3. Отслеживание изменения состояний Activity

Обработчики событий класса *Activity* позволяют отслеживать изменение состояний соответствующего объекта активности во время всего жизненного цикла (рис. 4.2):

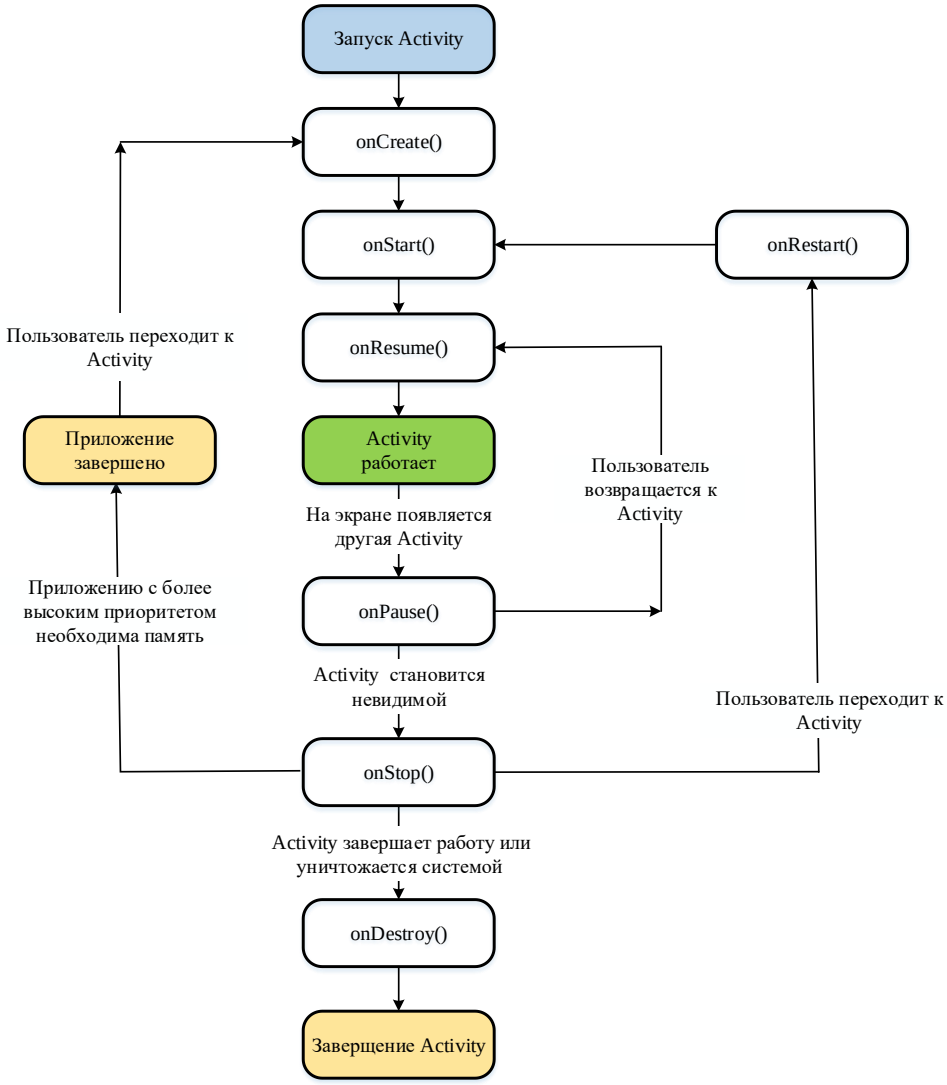


Рис. 4.2. Обработчики событий *Activity*

```
protected void onCreate(Bundle saveInstanceState);
protected void onStart();
protected void onRestoreInstanceState
                      (Bundle saveInstanceState);
protected void onRestart();
protected void onResume();
protected void onPause();
protected void onSaveInstanceState(Bundle saveInstanceState);
protected void onStop();
protected void onDestroy();
```

Рассмотрим методы обратного вызова *Activity*.

onCreate(Bundle saveInstanceState). Это первый метод, с которого начинается выполнение активности (рис. 4.3). В нем производится первоначальная настройка активности. В частности, создаются объекты визуального интерфейса. В качестве параметров метод получает объект *Bundle*, который содержит прежнее состояние активности, если оно было сохранено (если создается заново, то *null*). Этот метод необходимо обязательно реализовать, поскольку система вызывает его при создании *Activity*. Здесь нужно инициализировать ключевые компоненты и вызвать *setContentView*() для определения макета пользовательского интерфейса.

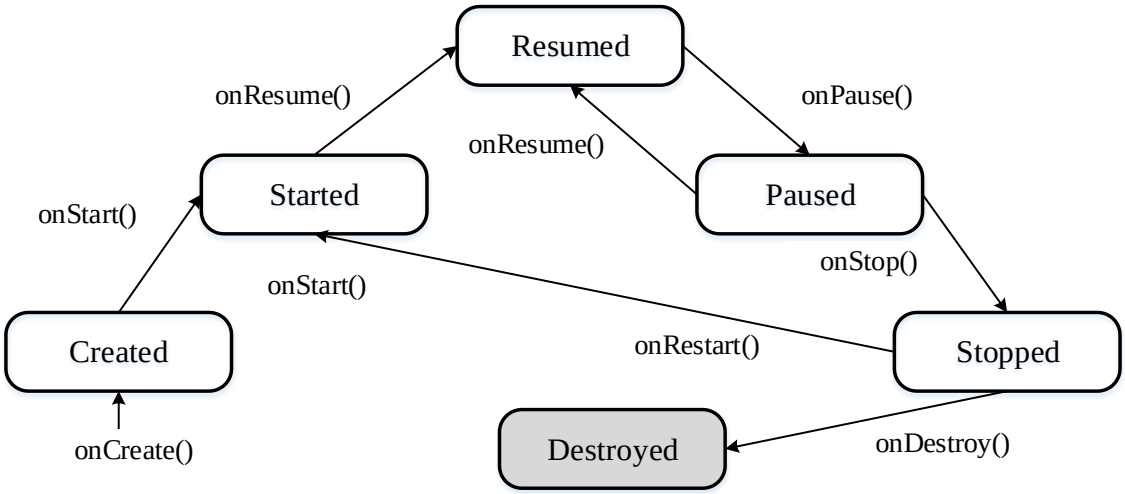


Рис.4.3. Состояния *Activity* и переходы между состояниями

***onStart* ().** В методе *onStart*() осуществляется подготовка к выводу активности на экран устройства. Как правило, этот метод не требует переопределения и всю работу производит встроенный код. После завершения работы метода активность отображается на экране, вызывается метод *onResume*, а активность переходит в состояние *Resumed*.

***onRestoreInstanceState*().** После завершения метода *onStart*() вызывается метод *onRestoreInstanceState*(), который призван восстанавливать сохраненное состояние из объекта *Bundle*. Этот метод вызывается только тогда, когда *Bundle* не равен *null* и содержит ранее сохраненное состояние. Так, при первом запуске приложения метод *onRestoreInstanceState*() не будет вызываться.

***onResume* ().** Вызывается непосредственно перед тем, как активность начинает взаимодействие с пользователем. На этом этапе активность находится в самом верху стека активностей и в нее поступают данные пользователя.

***onPause* ().** Если пользователь решит перейти к другой активности, то система вызывает метод *onPause*. В этом методе можно освобождать используемые ресурсы, приостанавливать процессы, останавливать работу камеры (если она используется) и т. д., чтобы они меньше сказывались на производительности системы. Но на работу данного метода отводится очень мало времени, поэтому не стоит здесь сохранять какие-то данные, особенно если при этом требуется обращение к сети, например, отправка данных или обращение к базе данных.

После выполнения этого метода активность становится невидимой, не отображается на экране, но она все еще активна. Если пользователь решит вернуться к ней, то система снова вызовет метод *onResume*() и активность опять появится на экране (см. рис. 4.3).

Другой вариант работы может возникнуть, если система видит, что для работы активных приложений необходимо больше памяти. В этом случае ОС может сама завершить работу активности, которая невидима и находится в фоне.

***onSaveInstanceState*(*Bundle saveInstanceState*).** Метод *onSaveInstanceState*() вызывается после метода *onPause*(), но до вызова *onStop*(). В *onSaveInstanceState* производится сохранение состояния приложения в передаваемом в качестве параметра объекте *Bundle*.

Если ОС уничтожает активность в целях восстановления памяти, объект уничтожается, в результате чего системе не удается просто восстановить состояние

активности. Вместо этого ОС необходимо повторно создать активность. Но пользователю неизвестно, что система уже ее уничтожила и создала повторно, поэтому он, скорее всего, ожидает, что *Activity* осталась прежней. В этой ситуации можно обеспечить сохранение важной информации о состоянии активности путем реализации дополнительного метода обратного вызова, который позволяет сохранить информацию (см. рис. 4.3).

***onStop* ().** Вызывается в случае, когда активность больше не отображается для пользователя. Пользователь может нажать на кнопку *Back* (*Назад*). В этом случае у *Activity* вызывается метод *onStop*().

Система вызывает этот метод в качестве первого признака выхода пользователя из *Activity* (это не всегда означает, что она будет уничтожена).

***onDestroy*().** Завершается работа активности вызовом метода *onDestroy*(), который возникает, если система решит убить активность либо при вызове метода *finish*().

При изменении ориентации экрана ОС завершает активность и затем создает ее заново, вызывая метод *onCreate*(). В большинстве случаев не следует ее явно завершать. Android выполнит такое управление за вас. Вызов методов завершения может отрицательно сказаться на ожидаемом поведении приложения.

4.2.4. Сохранение состояния Activity

Из описания методов обратного вызова активности очевидно, что есть два способа возврата к отображению для пользователя в неизмененном состоянии: уничтожение с последующим повторным созданием, когда активность должна восстановить свое ранее сохраненное состояние, или остановка активности и ее последующее восстановление в неизмененном состоянии (рис. 4.4).

Нет никаких гарантий, что метод *onSaveInstanceState*() будет вызван до того, как активность будет уничтожена. Если система вызывает метод *onSaveInstanceState*(), она делает это до вызова метода *onStop*(). Однако даже если не реализовать метод *onSaveInstanceState*(), часть состояния *Activity* восстанавливается реализацией по умолчанию. В частности, реализация по умолчанию вызывает соответствующий метод *onSaveInstanceState*() для каждого объекта *View* в макете, благодаря чему каждое представление может предоставлять ту информацию о себе, которую следует сохранить.

Поскольку вызов метода *onSaveInstanceState*() не гарантируется, следует использовать его только для записи переходного состояния активности (никогда не надо использовать его для хранения постоянных данных). Лучше пользоваться для этого методом *onPause*().

Для того чтобы проверить возможность приложения восстанавливать свое состояние, надо повернуть устройство для изменения ориентации экрана. При изменении ориентации экрана ОС уничтожает и повторно создает *Activity*, чтобы применить альтернативные ресурсы, которые могут быть доступны для новой конфигурации экрана.

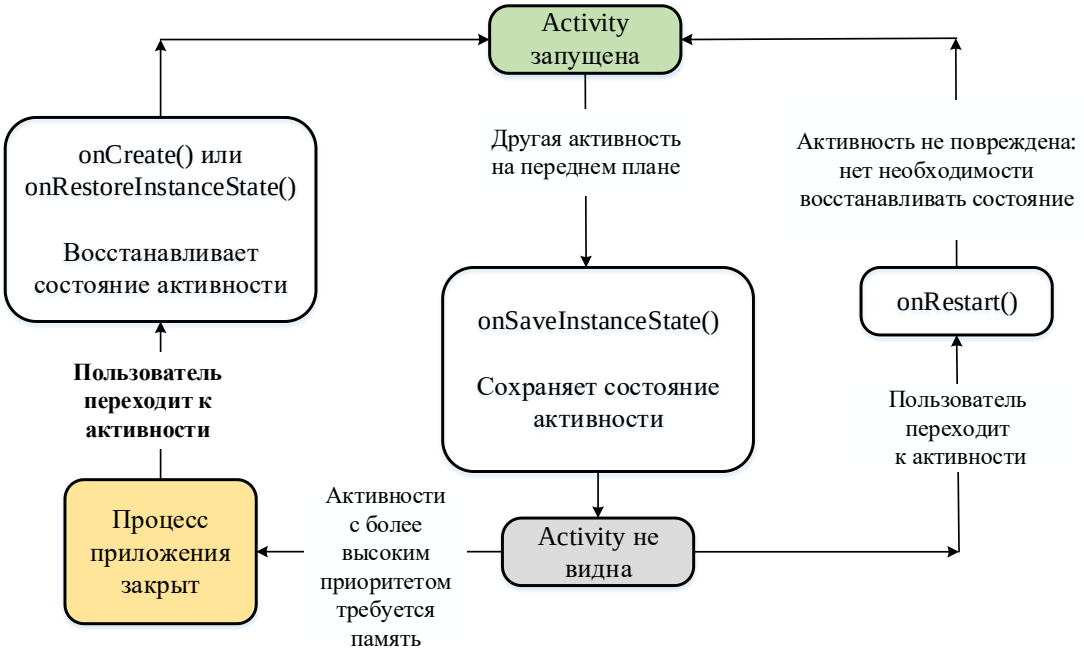


Рис. 4.4. Сохранение состояния *Activity*

Для понимания жизненного цикла активности следует переопределить методы *onPause*, *onStart*, *onRestart* для класса и внести изменения в метод *onCreate*:

```
public class MainActivity extends AppCompatActivity  {

    // Вызывается при создании Активности
    @Override
    public void onCreate(Bundle savedInstanceState)     {
        super.onCreate(savedInstanceState);
    }
// Инициализирует Активность

// Вызывается после завершения метода onCreate
// Используется для восстановления состояния UI
        @Override
        public void onRestoreInstanceState(Bundle savedInstanceState)          {
            super.onRestoreInstanceState(savedInstanceState);
 // Восстанавливаем состояние UI из объекта savedInstanceState
// Данный объект также был передан методу onCreate
        }
// Вызывается перед тем, как Активность снова становится видимой
        @Override
        public void onRestart(){
        super.onRestart();
// Восстанавливаем состояние UI с учетом того,
// что данная Активность уже была видимой
        }
// Вызывается когда Активность стала видимой
        @Override public void onStart(){
            super.onStart();
// Проделываем необходимые действия для
// Активности, видимой на экране
        }
// Должен вызываться в начале видимого состояния.
// На самом деле Android вызывает данный обработчик только
// для Активностей, восстановленных из неактивного состояния
        @Override public void onResume(){
            super.onResume();
// Восстанавливаем приостановленные обновления UI,
// потоки и процессы, «замороженные, когда
```

```
// Активность была в неактивном состоянии
        }

// Вызывается перед выходом из активного состояния,
// позволяя сохранить состояние в объекте savedInstanceState
        @Override
        public void onSaveInstanceState(Bundle savedInstanceState) {
// Объект savedInstanceState будет в последующем
// передан методам onCreate и onRestoreInstanceState
     super.onSaveInstanceState(savedInstanceState);
        }
// Вызывается перед выходом из активного состояния
            @Override
            public void onPause(){
// «Замораживаем» обновления UI, потоки или
// «трудоемкие» процессы, ненужные, когда Активность
// не на переднем плане
             super.onPause();
            }
// Вызывается перед выходом из видимого состояния
            @Override
            public void onStop(){
// «Замораживаем» обновления UI, потоки или
// «трудоемкие» процессы, ненужные, когда Активность
// не на переднем плане.
// Сохраняем все данные и изменения в UI, так как
// процесс может быть в любой момент убит системой
       super.onStop();
            }
// Вызывается перед уничтожением активности
            @Override
            public void onDestroy(){
// Освобождаем все ресурсы, включая работающие потоки,
// соединения с БД и т. д.
                super.onDestroy();
            }
        }
```

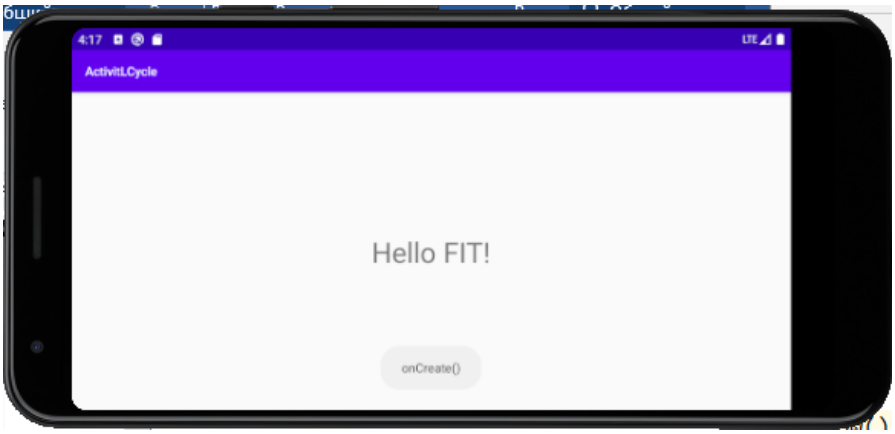
Теперь необходимо добавить вывод *Toast*:

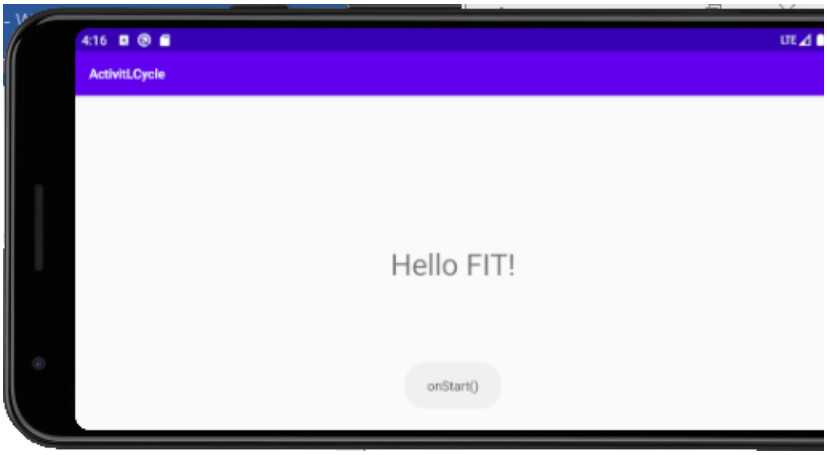
```
@Override
    public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState); setContentView(R.layout.activity_main);
        Toast.makeText(this, "onCreate()", Toast.LENGTH_LONG).show();
    }
    @Override
    protected void onPause() {
        Toast.makeText(this, "onPause()", Toast.LENGTH_LONG).show();
super.onPause();
    }
    @Override
    protected void onRestart() { super.onRestart();
        Toast.makeText(this, "onRestart()", Toast.LENGTH_LONG).show();
    }
    @Override
    protected void onStart() { super.onStart();
        Toast.makeText(this, "onStart()", Toast.LENGTH_LONG).show();
    }
}
```

Можно запустить приложение, посмотреть результат и изменить ориентацию экрана.

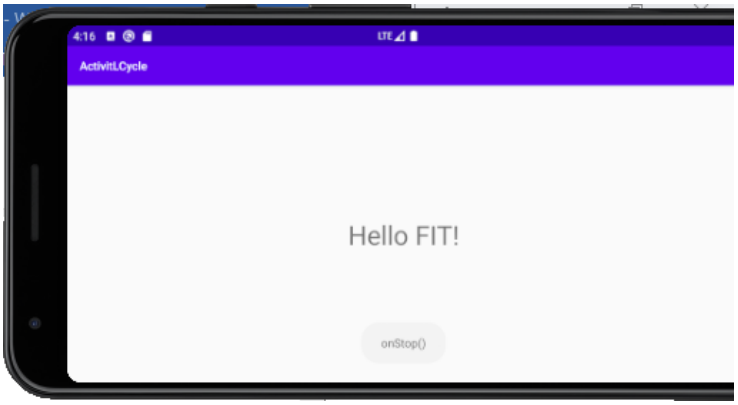
Запустите приложение и посмотрите результат. Измените ориентацию экрана.

Вы должны увидеть следующее:





Которые последовательно сменяют друг друга. Этот пример позволяет изучить ЖЦ активности.

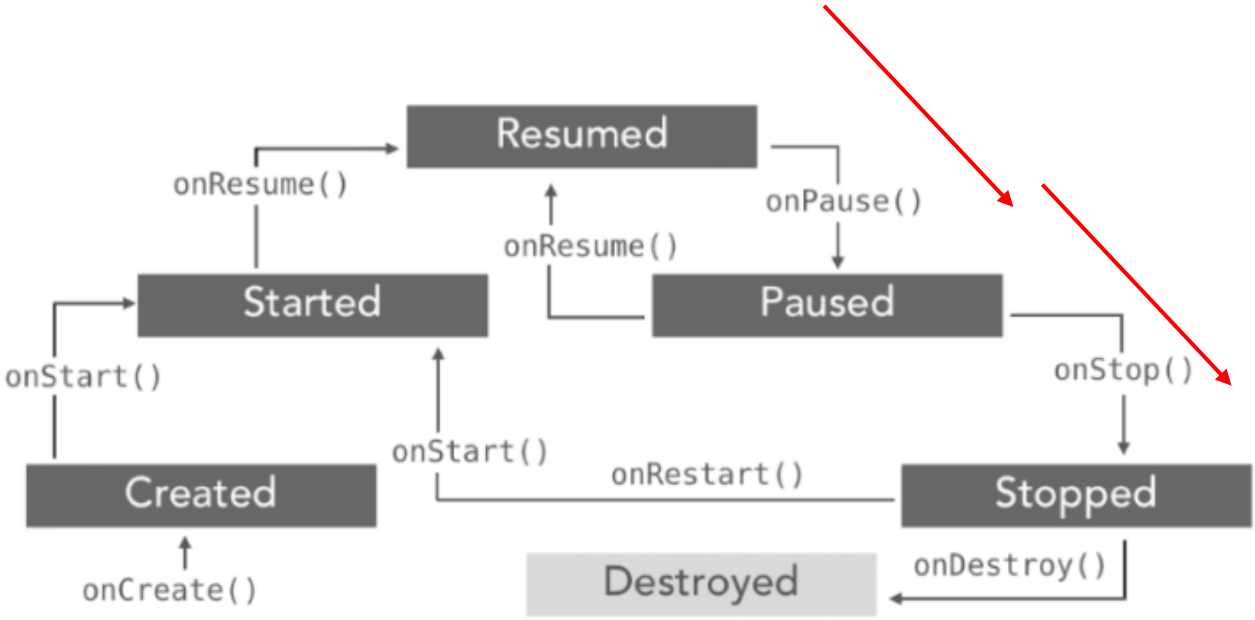


```
@Override
protected void onRestoreInstanceState(@NonNull Bundle savedInstanceState) {
  super.onRestoreInstanceState(savedInstanceState);
    Log.i(TAG,savedInstanceState.getString("saved"));
    Toast.makeText(this, "onRestoreInstanceState()", Toast.LENGTH_LONG).show();
    Log.i(TAG,"onRestoreInstanceState()");
}

@Override
protected void onSaveInstanceState(@NonNull Bundle outState) {
    super.onSaveInstanceState(outState);
    outState.putString("saved", "September 21");
    Toast.makeText(this, "onSaveInstanceState", Toast.LENGTH_LONG).show();
    Log.i(TAG,"onSaveInstanceState");

}
```

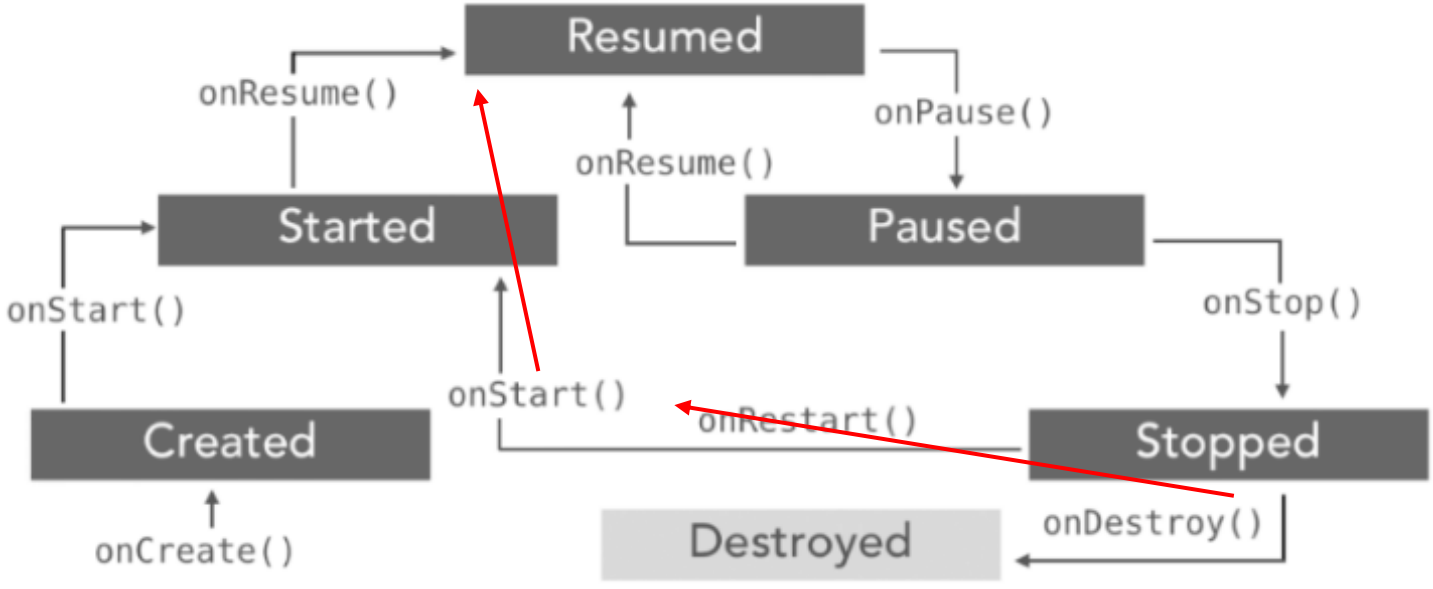
4.3 Переходы между состояниями Activity

Рассмотрим типичные сценарии

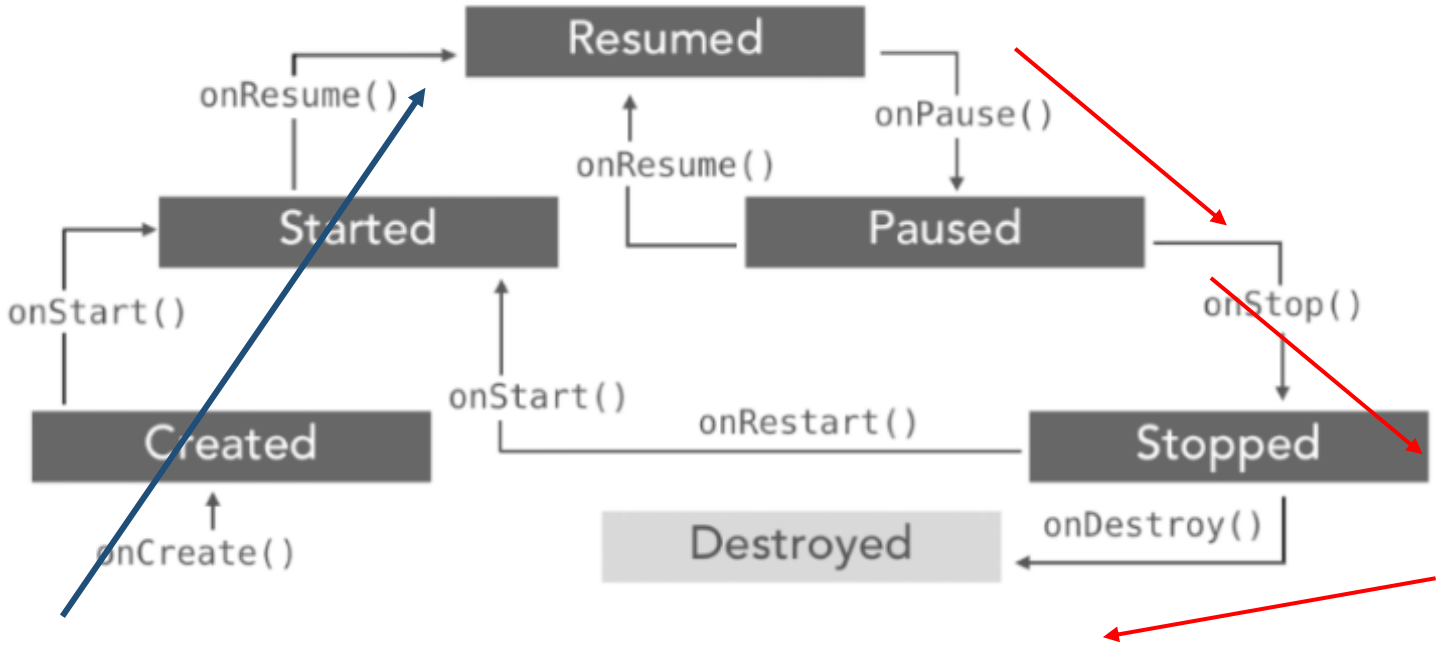
1) *Переключаемся на другое приложение (не видима)*



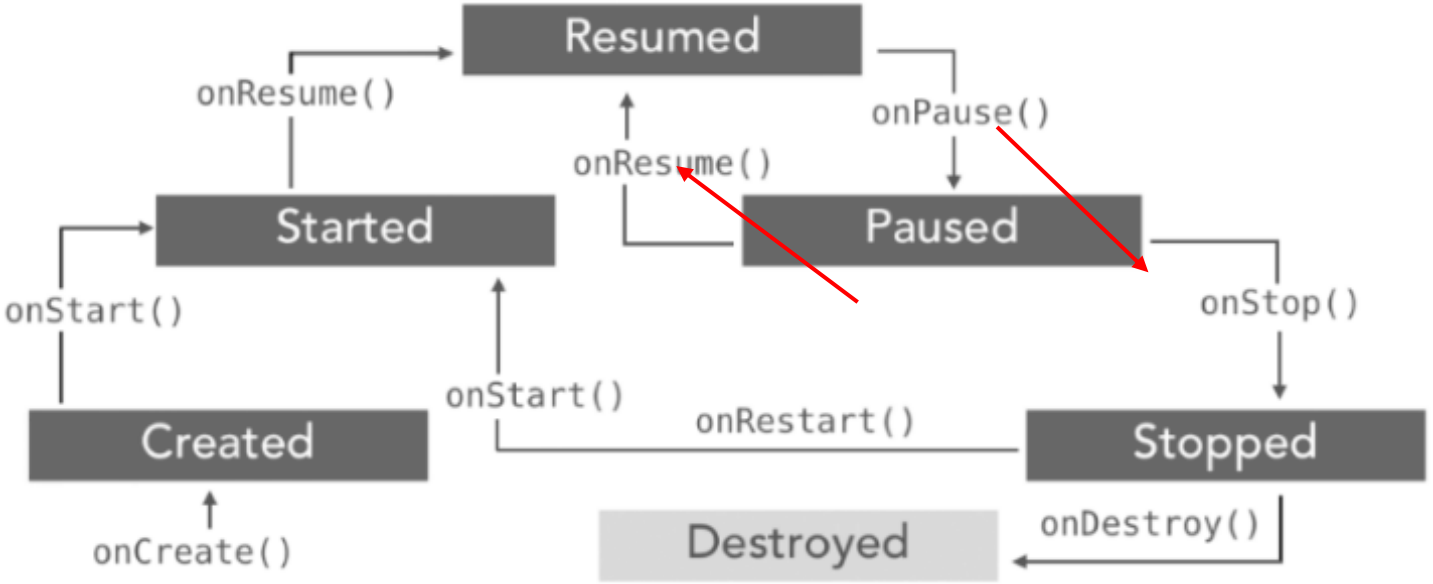
2) Вернуться к Activity



3) Нажимаем на кнопку Back



4) multi-window mode, диалоговое окно, сообщение и т.п.



<u>Итак:</u>

1. *Каждое приложение – отдельный процесс с определенным приоритетом*
2. *Activity слабо связаны друг с другом*
3. *Одна из Activity в приложении обозначается как «основная»*
4. *Каждая Activity может запустить другую Activity*
5. *При запуске новой Activity старая помещается в стек переходов назад – back stack (удаляется при окончании работы приложения)*
6. *Для уведомления об изменении состояния используются методы обратного вызова жизненного цикла Activity*

Взаимодействие Activity

Как правило, приложение состоит из нескольких activity, которые слабо связаны друг с другом. Обычно одна из activity в приложении обозначается как «основная», предлагаемая пользователю при первом запуске приложения. В свою очередь, каждая activity может запустить другую activity для выполнения различных действий. Каждый раз, когда запускается новая activity, предыдущая останавливается, однако система сохраняет ее в стеке («стек переходов назад»).

Согласование операций

Когда одна activity запускает другую, в жизненных циклах обеих из них происходит переход из одного состояния в другое. Первая activity приостанавливается и заврешается (однако она не будет остановлена, если она по-прежнему видима на фоне), а вторая activity создается. В случае, если эти операции обмениваются данным, сохраненными на диске или в другом месте, важно понимать, что первая activity не останавливается полностью до тех пор, пока не будет создана вторая activity. Наоборот, процесс запуска второй накладывается на процесс остановки первой activity.
Порядок обратных вызовов жизненного цикла четко определен, в частности, когда в одном и том же процессе находятся две activity , и одна из них запускает другую

Такая предсказуемая последовательность выполнения обратных вызовов жизненного цикла позволяет управлять переходом информации из одной activity в другую.