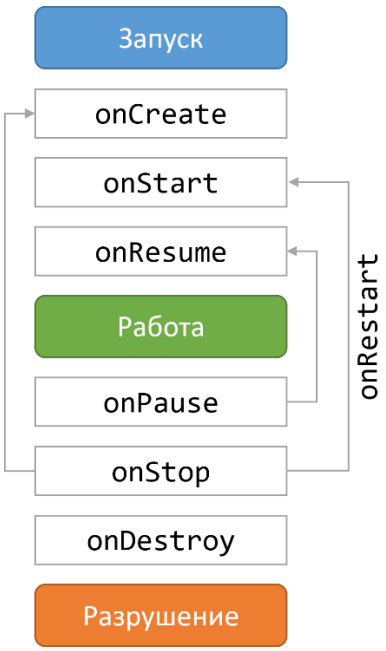
Жизненный цикл активности. Контейнер GridLayout

Жизненный цикл активности

Каждая активность, которая появляется на экране, в обязательном порядке проходит несколько этапов от момента создания до появления на экране, и несколько этапов после ухода с экрана и до момента разрушения. В некоторых случаях активность, которая уже ушла с экрана, может вернуться, тогда появляются дополнительные промежуточные этапы.

Можно выстроить этапы жизненного цикла активности следующим образом:



Для разработчика приложения есть возможность на каждом этапе выполнить какие-либо действия, для этого нужно в классе активности переопределить метод с соответствующим названием. Делать это не обязательно: если метод не переопределён, то система всё равно выполнит что ей требуется, просто программа не воспользуется шансом сделать какие-то дополнительные действия, которые ей могут быть полезны.

Когда случается каждое из этих событий?

- `onCreate` – при начале создания активности: в этом методе можно подключить разметку с пользовательским интерфейсом, а также провести начальную инициализацию нужных переменных
- `onStart` – непосредственно перед тем как уже созданный пользовательский интерфейс появится на экране
- `onResume` – непосредственно перед тем как пользовательский интерфейс станет доступным для взаимодействия с пользователем
- `onPause` – после того как активность потеряла фокус, например, пользователь открыл диалог, или переключился на другое приложение в многооконном режиме; активность всё ещё видима, но уже не доступна для непосредственного взаимодействия с пользователем
- `onStop` – после ухода активности с экрана, нужно остановить все действия, которые не требуется выполнять если пользователь не видит активность

- `onDestroy` – перед уничтожением активности

Изначально в программе уже переопределен метод `onCreate`, он выглядит примерно так:

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
}
```

Первой строчкой переопределённого метода всегда идёт вызов соответствующего метода из родительского класса. Если этого не сделать, работоспособность программы может быть нарушена, потому что не будут выполнены действия по умолчанию, относящиеся к этому этапу. Если забыть про вызов базового метода, Android Studio напомнит про это.

Вторая строка содержит метод `setContentView`, он подключает интерфейс, определённый в соответствующем XML-файле. Технически эта строка является не обязательной, если её удалить – активность останется пустой. Это может быть полезно если требуется создать полностью свой уникальный и нестандартный интерфейс – например, для графической игры или приложения со скинами. Но для большинства обычных программ эта строка полезна.

Другие переопределяемые методы должны вызывать лишь соответствующий метод родительского класса, больше условий нет:

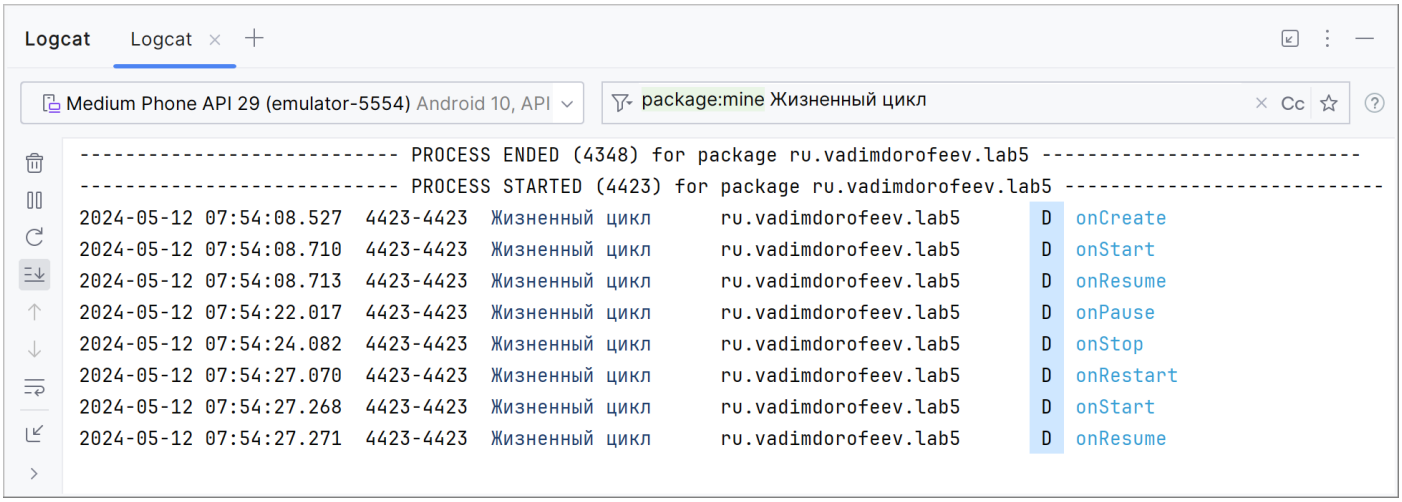
```
override fun onResume() {
    super.onResume()
    // Действия
}
```

Если требуется выполнять какие-то парные действия, например, при отображении программы включать музыку, а при сворачивании выключить, то лучше всего такие действия выполнять в парных событиях: `onCreate` – `onDestroy`, `onStart` – `onStop`, `onResume` – `onPause`. В противном случае есть вероятность что действие будет запущено два раза, а остановлено один, или наоборот.

Для лучшего понимания, когда вызывается какая-то из перечисленных функций, лучше всего создать новый проект, переопределить в нём каждую из перечисленных функций, и в ней вызвать функцию записи в лог, например, так:

```
override fun onResume() {
    super.onResume()
    Log.d("Жизненный цикл", "onResume")
}
```

Затем запустить проект в эмуляторе, открыть окно Logcat, и выполнять разные действия: сворачивать и разворачивать программу, закрывать её, запускать заново и т.д. В окне Logcat будут отображаться события, которые при этом происходят:



Контейнер GridLayout

`GridLayout` это контейнер, в котором дочерние элементы размещаются в прямоугольной сетке. Сетка состоит из набора условных линий, которые делят область контейнера на ячейки.

Строки и столбцы в сетке нумеруются с нуля. Общее количество столбцов задаётся с помощью свойств `rowCount` и `columnCount`:

```
<GridLayout
    android:rowCount="4"
    android:columnCount="3">
```

Внутри контейнера элементы начинают размещаться с верхней (нулевой) строки с левого (нулевого) столбца, и заполняют строку, а затем переходят на следующую строку, и так до тех пор пока не заполнят весь контейнер (или не закончатся дочерние элементы):

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9
3	10	11	12

Но при желании можно любому элементу указать в какой именно строке и столбце он должен находиться:

```
<Button
    android:layout_row="1"
    android:layout_column="2"/>
```

Если у следующих элементов конкретные строка и столбец не указаны, то они продолжат размещаться уже такого элемента.

Как правило, каждый дочерний элемент занимает одну ячейку занимают одну или несколько смежных ячеек, однако элемент можно растянуть на несколько столбцов или строк с помощью свойств `layout_columnSpan` или `layout_rowSpan`. Например, чтобы сделать вот такую панель для ввода, у последнего элемента с надписью «Вход» нужно установить свойство `layout_columnSpan="2"`:

Пространство между дочерними элементами можно указать либо по отдельности для каждого дочернего элемента (с помощью свойств `leftMargin`, `topMargin`, `rightMargin` и `bottomMargin`), или используя свойство `useDefaultMargins` у контейнера `GridLayout`. Это свойство устанавливает отступы, которые считаются рекомендованными в конкретной версии Android.

Если ширина или высота элементов неодинаковая, то столбцы и строки тоже могут оказаться разной ширины или высоты. Чтобы сделать их одинаковыми (или сохранить определённую пропорцию) нужно задать вес элемента с помощью свойств `layout_columnWeight` и `layout_rowWeight`. В этом случае веса всех элементов в каждом столбце и строке будут просуммированы, и их размеры будут распределены пропорционально.

Внутри ячейки элемент можно выровнять по одной или нескольким сторонам используя свойство `layout_gravity`. Ему можно указать строки `start` или `end` для выравнивания по горизонтали `top` или `bottom` для выравнивания по вертикали, или же `center`, `center_horizontal` или `center_vertical` для выравнивания по центру. Можно указать сразу несколько значений, например, `end|bottom` для размещения в правом нижнем углу ячейки.

Задание

Создайте приложение с цветными плитками.

Сетка, в которой будут находиться плитки, должна быть сделана с помощью контейнера `GridLayout`, и занимать всё свободное пространство в активности. Каждая цветная плитка представляет собой элемент `TextView`, растянутый на всю ширину и высоту ячейки. Цвет фона элемента задаётся с помощью метода `setBackoundColor`.

Плитки должны менять свои цвета как при нажатии на любую плитку, так и при появлении приложения на экране (например, при запуске программы, или если переключиться на другое приложение и тут же вернуться обратно).

Придумайте свой уникальный алгоритм генерации цветов для перекрашивания элементов: цвета всех элементов должны быть каким-то образом связаны друг с другом. В примере у всех элементов одинаковый базовый цвет, но разный альфа-канал, от 25 с шагом 25: