Этапы жизненного цикла активити

В течение своего существования деятельность переходит в различные состояния, а иногда и возвращается в них. Этот переход состояний известен как жизненный цикл активности.

В Android активити — это точка входа для взаимодействия с пользователем.

Раньше одно действие отображало один экран в приложении. Согласно современным рекомендациям, одно действие может отображать несколько экранов, меняя их местами по мере необходимости.

Жизненный цикл действия простирается от создания действия до его уничтожения, когда система восстанавливает ресурсы этого действия. Когда пользователь входит в действие и выходит из него, каждое действие переходит из одного состояния в жизненный цикл действия.

Как разработчик Android, вам необходимо понимать жизненный цикл активности. Если ваши действия неправильно реагируют на изменения состояния жизненного цикла, ваше приложение может генерировать странные ошибки, сбивать с толку поведение ваших пользователей или использовать слишком много системных ресурсов Android. Понимание жизненного цикла Android и правильное реагирование на изменения состояния жизненного цикла — важная часть разработки Android.

Что вы узнаете

- Как распечатать информацию журнала в Logcat
- Основы жизненного `Activity` цикла и обратные вызовы, которые вызываются, когда активность перемещается между состояниями.
- Как переопределить методы обратного вызова жизненного цикла для выполнения операций в разные моменты жизненного цикла действия.

Что ты построишь

- Измените начальное приложение под названием Dessert Clicker, чтобы добавить информацию журнала, отображаемую в Logcat.
- Переопределять методы обратного вызова жизненного цикла и регистрировать изменения состояния активности.
- Запустите приложение и запишите информацию журнала, которая появляется при запуске, остановке и возобновлении действия.
- Реализуйте `rememberSaveable`, чтобы сохранить данные приложения, которые могут быть потеряны в случае изменения конфигурации устройства.

2. Обзор приложения

В этой лаборатории кода вы работаете с начальным приложением Dessert Clicker. В Dessert Clicker каждый раз, когда пользователь касается десерта на экране, приложение «покупает» десерт для пользователя. Приложение обновляет значения в макете для:

- Количество десертов, которые «куплены»
- Общий доход от «купленных» десертов

Это приложение содержит несколько ошибок, связанных с жизненным циклом Android. Например, при определенных обстоятельствах приложение сбрасывает значения десерта на 0. Понимание жизненного цикла Android поможет вам понять, почему возникают эти проблемы и как их исправить.

Загрузите стартовый код

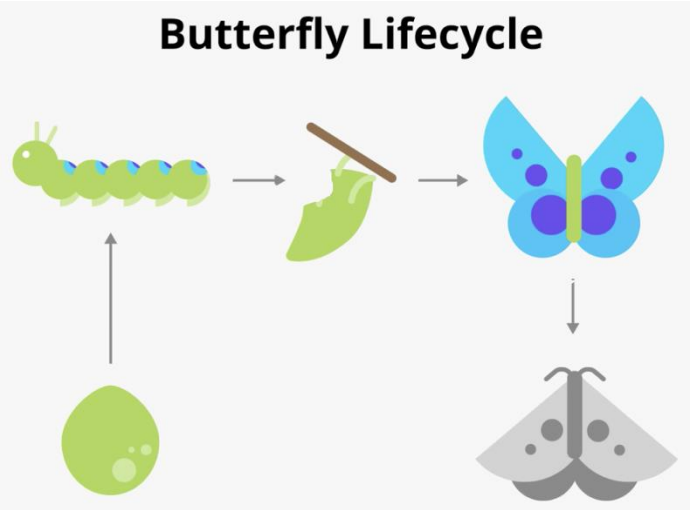
В Android Studio откройте `basic-android-kotlin-compose-training-dessert-clicker`папку.

URL стартового кода: https://github.com/google-developer-training/basic-android-kotlin-compose-training-dessert-clicker

Название ветки branch :`starter`

3. Изучите методы жизненного цикла и добавьте базовое ведение журнала.

Каждая деятельность имеет так называемый ***жизненный цикл*** . Этот термин является намеком на жизненные циклы растений и животных, например, на жизненный цикл бабочки: различные состояния бабочки показывают ее рост от яйца до гусеницы, куколки, бабочки и смерти.



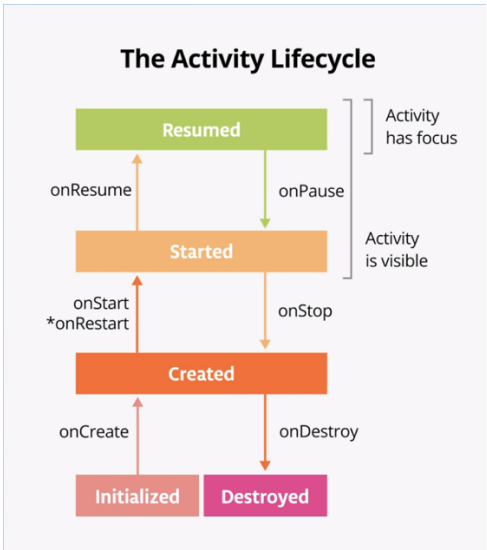
Аналогичным образом, жизненный цикл действия состоит из различных состояний, через которые может пройти действие, начиная с момента первой инициализации действия и заканчивая его уничтожением, после чего операционная система (ОС) освобождает свою память. Обычно точкой входа в программу является метод `main()`. Однако деятельность Android начинается с `onCreate()`метода; этот метод будет эквивалентом стадии яйца в приведенном выше примере. Вы уже много раз использовали упражнения в этом курсе и, возможно, узнали этот `onCreate()` метод. Когда пользователь запускает ваше приложение, перемещается между действиями, перемещается внутри и за пределами вашего приложения, действие меняет состояние.

На следующей диаграмме показаны все состояния жизненного цикла активности. Как видно из их названий, эти состояния представляют статус деятельности. Обратите внимание, что, в отличие от жизненного цикла «бабочки», действие может перемещаться между состояниями на протяжении всего жизненного цикла, а не двигаться только в одном направлении.

Примечание. Приложение Android может выполнять несколько действий. Однако рекомендуется иметь одно занятие, и до сих пор именно это вы и реализовывали в этом курсе.

Часто требуется изменить какое-либо поведение или запустить некоторый код при изменении состояния жизненного цикла действия. Таким образом, сам класс `Activity` и любые его подклассы, такие как `ComponentActivity`, реализуют набор методов обратного вызова жизненного цикла. Android вызывает эти обратные вызовы, когда действие переходит из одного состояния в

другое, и вы можете переопределить эти методы в своих собственных действиях для выполнения задач в ответ на эти изменения состояния жизненного цикла. На следующей диаграмме показаны состояния жизненного цикла, а также доступные переопределяемые обратные вызовы.



Примечание. Звездочка на `onRestart()` методе указывает, что этот метод не вызывается каждый раз при переходе состояний между **Created** и **Started** . Он вызывается только в том случае, если был вызван `onStop()`, и впоследствии действие перезапускается.

Важно знать, когда Android вызывает переопределяемые обратные вызовы и что делать в каждом методе обратного вызова.

Шаг 1. Изучите метод onCreate() и добавьте ведение журнала.

Чтобы понять, что происходит с жизненным циклом Android, полезно знать, когда вызываются различные методы жизненного цикла. Эта информация поможет вам определить, где в приложении Dessert Clicker что-то идет не так.

Простой способ определить эту информацию — использовать функцию ведения журнала Android. Ведение журнала позволяет записывать короткие сообщения на консоль во время работы приложения и использовать ее, чтобы видеть, когда запускаются различные обратные вызовы.

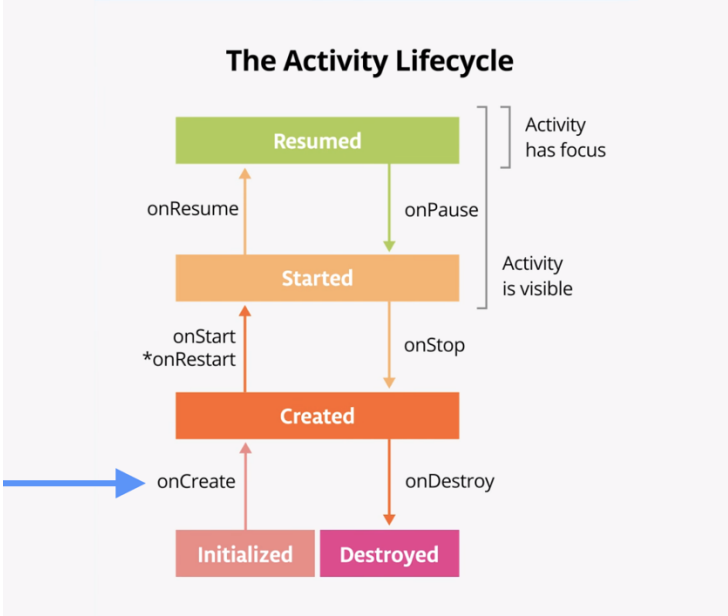
1. апустите приложение Dessert Clicker и несколько раз тапните по изображению десерта. Обратите внимание, как меняется стоимость проданных десертов и общая сумма в долларах.

2. ткройте `MainActivity.kt` и изучите метод `onCreate()` для этого действия:

```
override fun onCreate(savedInstanceState: Bundle?) {
    // ...
}
```

На диаграмме жизненного цикла активности вы можете узнать этот метод `onCreate()`, поскольку вы уже использовали этот обратный вызов раньше. Это единственный метод, который должен реализовывать каждое действие. В этом методе `onCreate()` вам следует выполнить любую

одноразовую инициализацию вашей деятельности. Например, в `onCreate()` вы вызываете `setContent()`, который определяет макет пользовательского интерфейса действия.



Метод `onCreate()` жизненного цикла вызывается один раз, сразу после *инициализации* активити — когда ОС создает новый объект `Activity` в памяти. После выполнения `onCreate()` активити считается *созданной* .

Примечание. Когда вы переопределяете метод `onCreate()`, вы должны вызвать реализацию суперкласса, чтобы завершить создание действия, поэтому внутри него вы должны немедленно вызвать `super.onCreate()`. То же самое справедливо и для других методов обратного вызова жизненного цикла.

обавьте следующую константу на верхний уровень `MainActivity.kt`, над объявлением класса `class MainActivity`.

Хорошим соглашением является объявление константы `TAG` в вашем файле, поскольку ее значение не изменится.

Чтобы пометить ее как константу времени компиляции, используйте при объявлении переменной `const`. Константа времени компиляции — это значение, известное во время компиляции.

```
private const val TAG = "MainActivity"
```

методе `onCreate()` сразу после вызова `super.onCreate()` добавьте следующую строку:

```
Log.d(TAG, "onCreate Called")
```

ри необходимости импортируйте класс `Log` (нажмите `Alt+Enter`, или `Option+Enter` на Mac и выберите «**Import**»). Если вы включили автоматический импорт, это должно произойти автоматически.

```
import android.util.Log
```

Класс `Log` записывает сообщения в **Logcat**.

Logcat — это консоль для регистрации сообщений. Здесь отображаются сообщения от Android о вашем приложении, включая сообщения, которые вы явно отправляете в журнал с помощью метода `Log.d()` или других `Log` методов класса.

В инструкции есть три важных аспекта `Log`:

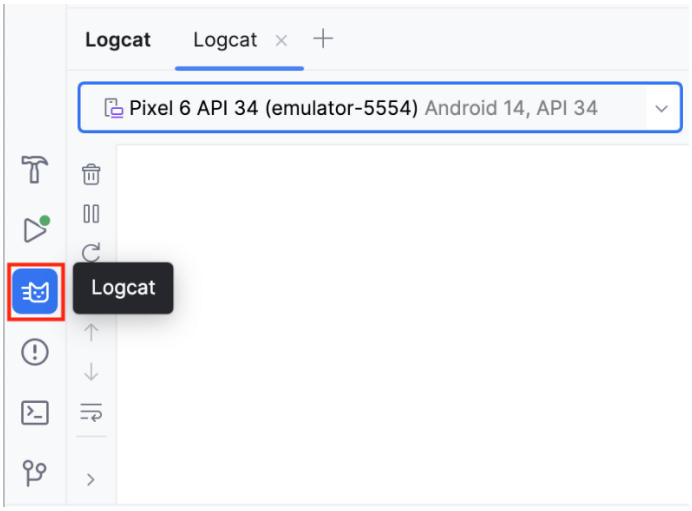
- **Приоритет сообщения журнала**, то есть насколько важно сообщение.
    - Метод `Log.v()` регистрирует подробные сообщения.
    - Метод `Log.d()` - отладочное сообщение.

      информационные сообщения,

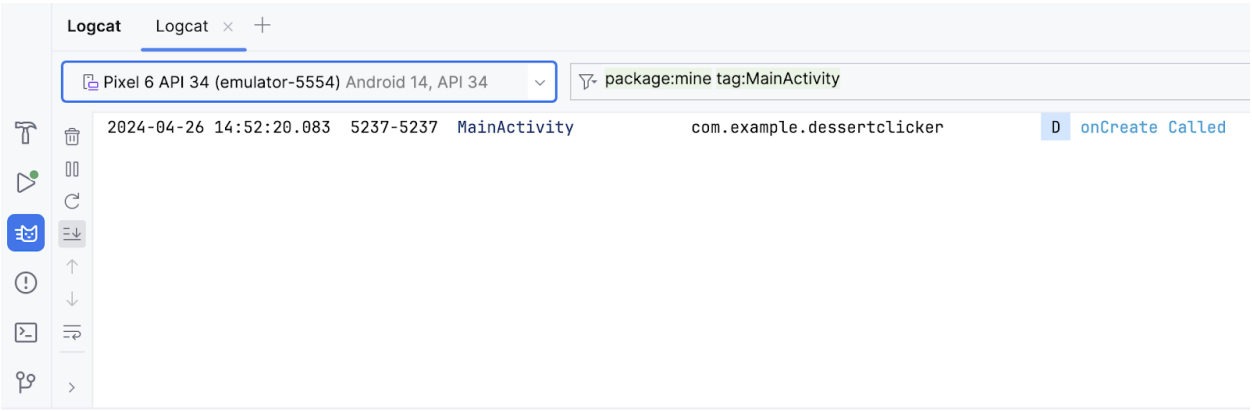
      предупреждения

      сообщения об ошибках.
- Журнал `tag` (первый параметр), в данном случае `"MainActivity"`. Тег представляет собой строку, которая позволяет вам легче находить сообщения журнала в Logcat. Тег обычно представляет собой имя класса.
- Фактическое сообщение журнала, называемое `msg` (второй параметр), представляет собой короткую строку, которая в данном примере равна `"onCreate Called"`.

компилируйте и запустите приложение Dessert Clicker. Вы не увидите никаких различий в поведении приложения, когда нажимаете на десерт. В Android Studio в нижней части экрана щелкните вкладку **Logcat** .



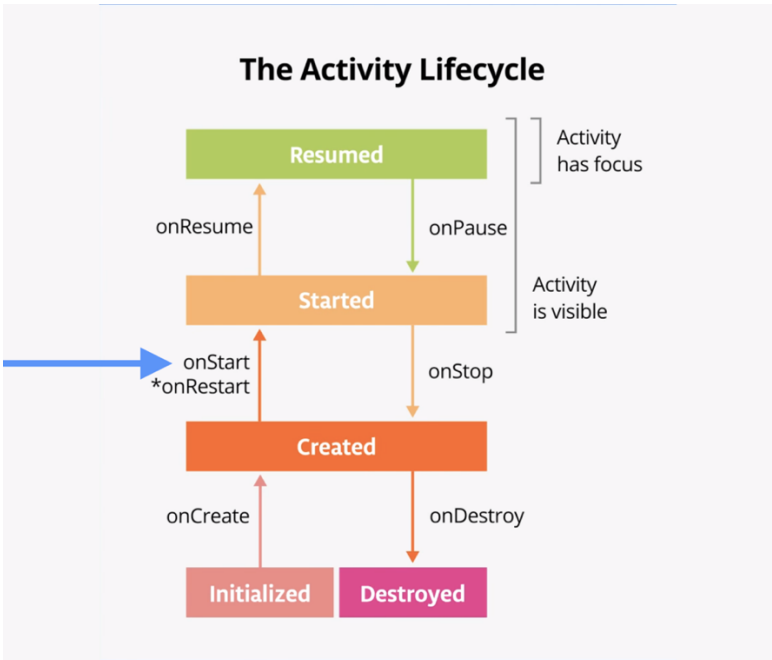
окне **Logcat** введите `tag:MainActivity` в поле поиска.



Logcat может содержать множество сообщений, большинство из которых вам не пригодятся. Вы можете фильтровать записи Logcat разными способами, но поиск — самый простой. Поскольку вы использовали тег журнала `MainActivity` в своем коде, вы можете использовать этот тег для фильтрации журнала. Ваше сообщение журнала включает дату и время, тег журнала, имя пакета (`com.example.dessertclicker`) и фактическое сообщение. Поскольку это сообщение появляется в журнале, вы знаете, что `onCreate()` было выполнено.

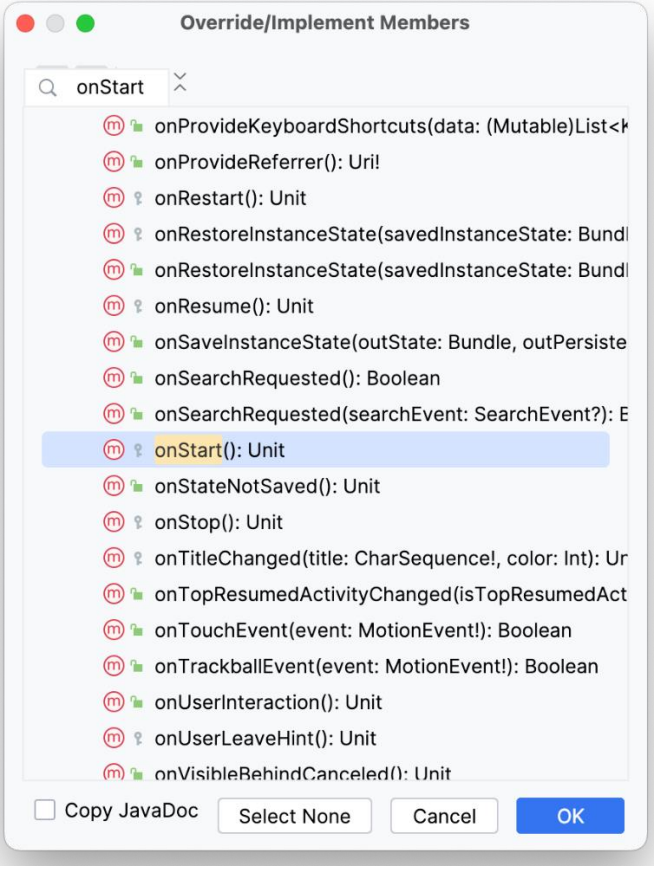
Шаг 2. Реализуйте onStart() метод

Метод `onStart()` жизненного цикла вызывается сразу после `onCreate()`. После `onStart()` ваша активность видна на экране. В отличие от `onCreate()`, который вызывается только один раз для инициализации вашей активности, `onStart()` может вызываться системой много раз в течение жизненного цикла вашей активности.



Обратите внимание, что `onStart()` связано с соответствующим методом `onStop()`жизненного цикла. Если пользователь запускает ваше приложение, а затем возвращается на главный экран устройства, действие прекращается и больше не отображается на экране.

> открыв `MainActivity.kt` курсор внутри класса `MainActivity`, выберите **«Код» > «Переопределить методы...»** или нажмите `Control+O`. Появится диалоговое окно с длинным списком всех методов, которые вы можете переопределить в этом классе.



ачните ввод `onStart` для поиска правильного метода. Чтобы перейти к следующему соответствующему элементу, используйте стрелку вниз. Выберите `onStart()` из списка и нажмите **«OK»** , чтобы вставить шаблонный код переопределения. Код выглядит следующим образом:

```
override fun onStart() {
   super.onStart()
}
```

нутри метода `onStart()`добавьте сообщение журнала:

```
override fun onStart() {
   super.onStart()
   Log.d(TAG, "onStart Called")
}
```

к

ведите `tag:MainActivity` в поле поиска, чтобы отфильтровать журнал. Обратите внимание, что оба метода `onCreate()`и `onStart()` были вызваны один за другим, и что ваши действия видны на экране.

ажмите кнопку **«Домой»** на устройстве, а затем используйте экран «Недавние», чтобы вернуться к действию. Обратите внимание, что действие возобновляется с того места, где оно было остановлено, со всеми теми же значениями, и `onStart()` регистрируется в Logcat во второй раз. Обратите также внимание, что метод `onCreate()`больше не вызывается.

```
2024-04-26 14:54:48.721  5386-5386  MainActivity    com.example.dessertclicker    D  onCreate Called
2024-04-26 14:54:48.756  5386-5386  MainActivity    com.example.dessertclicker    D  onStart Called
```

и

з

```
2024-04-26 14:55:41.674  5386-5386  MainActivity      com.example.dessertclicker       D  onStart Called
```

Примечание. Экспериментируя с устройством и наблюдая за обратными вызовами жизненного цикла, вы можете заметить необычное поведение при повороте устройства. Вы узнаете об этом поведении позже в этой лаборатории кода.

Шаг 3. Добавьте дополнительные операторы журнала

На этом этапе вы реализуете ведение журнала для всех остальных методов жизненного цикла.

1. ереопределите остальные методы жизненного цикла `MainActivity` и добавьте операторы журнала для каждого из них, как показано в следующем коде:

```
override fun onResume() {
   super.onResume()
   Log.d(TAG, "onResume Called")
}

override fun onRestart() {
   super.onRestart()
   Log.d(TAG, "onRestart Called")
}

override fun onPause() {
   super.onPause()
   Log.d(TAG, "onPause Called")
}

override fun onStop() {
   super.onStop()
   Log.d(TAG, "onStop Called")
}

override fun onDestroy() {
   super.onDestroy()
   Log.d(TAG, "onDestroy Called")
}
```

2. компилируйте и снова запустите Dessert Clicker и проверьте Logcat.

Обратите внимание, что на этот раз, помимо `onCreate()`и `onStart()`, есть сообщение журнала для `onResume()`обратного вызова жизненного цикла.

```
2024-04-26 14:56:48.684  5484-5484  MainActivity      com.example.dessertclicker     D  onCreate Called
2024-04-26 14:56:48.709  5484-5484  MainActivity     com.example.dessertclicker     D  onStart Called
2024-04-26 14:56:48.713  5484-5484  MainActivity     com.example.dessertclicker     D  onResume Called
```

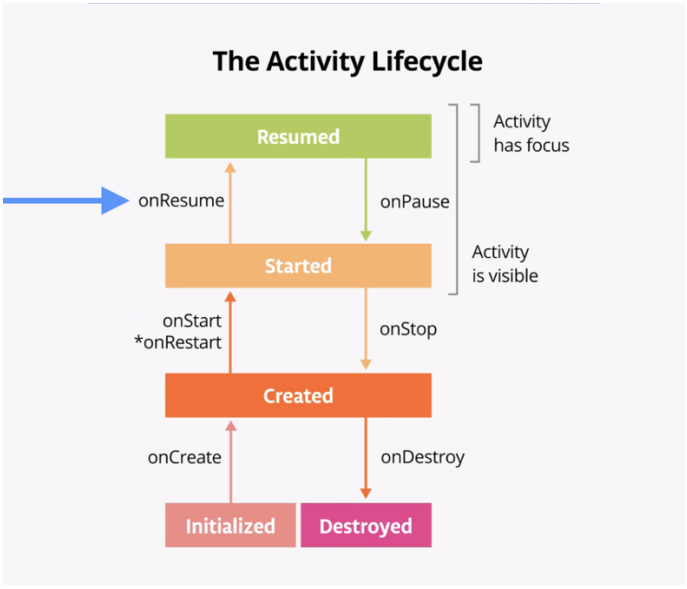
Когда действие начинается с самого начала, вы видите все три обратных вызова жизненного цикла, вызываемые по порядку:

- когда система создает приложение.

делает приложение видимым на экране, но пользователь пока не может с ним взаимодействовать.

выводит приложение на передний план, и теперь пользователь может с ним взаимодействовать.

Несмотря на название, метод `onResume()` вызывается при запуске, даже если возобновлять нечего.



4. Изучите варианты использования жизненного цикла

Теперь, когда вы настроили приложение Dessert Clicker для ведения журналов, вы готовы начать использовать приложение и изучить, как запускаются обратные вызовы жизненного цикла.

Вариант использования 1. Открытие и закрытие активности.

Вы начинаете с самого простого варианта использования: первый запуск приложения, а затем его закрытие.

компилируйте и запустите приложение Dessert Clicker, если оно еще не запущено. Как вы видели, обратные `onCreate()` вызовы `onStart()` и `onResume()` вызываются при первом запуске активности.

```
2024-04-26 14:56:48.684  5484-5484  MainActivity    com.example.dessertclicker    D  onCreate Called
2024-04-26 14:56:48.709  5484-5484  MainActivity    com.example.dessertclicker    D  onStart Called
2024-04-26 14:56:48.713  5484-5484  MainActivity    com.example.dessertclicker    D  onResume Called
```

остучите по кексу несколько раз.

ажмите кнопку **«Назад»** на устройстве.

Обратите внимание, что в Logcat `onPause()` и `onStop()` вызываются именно в таком порядке.

```
2024-04-26 14:58:19.984  5484-5484  MainActivity   com.example.dessertclicker      D  onPause Called
2024-04-26 14:58:20.491  5484-5484  MainActivity    com.example.dessertclicker     D  onStop Called
2024-04-26 14:58:20.517  5484-5484  MainActivity    com.example.dessertclicker     D  onDestroy
Called
```

В этом случае нажатие кнопки **«Назад»** приводит к удалению действия (и приложения) с экрана и перемещению в конец стека действий.

ОС Android может закрыть ваше действие, если ваш код вручную вызывает метод действия finish() или если пользователь принудительно закрывает приложение. Например, пользователь может принудительно закрыть или закрыть приложение на экране «Последние». ОС также может самостоятельно отключить вашу активити, если ваше приложение долгое время не отображалось на экране. Android делает это, чтобы продлить срок службы батареи и освободить ресурсы, которые использовало приложение, чтобы они были доступны другим приложениям. Это всего лишь несколько примеров того, почему система Android уничтожает вашу активити. Существуют и другие случаи, когда система Android уничтожает вашу активити без предупреждения.

Примечание: onCreate() и onDestroy(), которые будут рассмотрены позже в этой кодовой лаборатории, вызываются только один раз за время существования одного экземпляра действия: onCreate() для инициализации приложения в первый раз, а также onDestroy() для аннулирования, закрытия или уничтожения объектов, которые действие могло использовать, поэтому что они не продолжают использовать ресурсы, такие как память.

Вариант использования 2: переход от действия и обратно к нему

Теперь, когда вы запустили приложение и закрыли его, вы увидели большинство состояний жизненного цикла, когда действие создается впервые. Вы также видели большинство состояний жизненного цикла, через которые проходит действие при его закрытии. Но когда пользователи взаимодействуют со своими устройствами Android, они переключаются между приложениями, возвращаются домой, запускают новые приложения и справляются с прерываниями, вызванными другими действиями, такими как телефонные звонки.

Ваша активити не закрывается полностью каждый раз, когда пользователь уходит от нее:

- Когда ваша активити больше не отображается на экране, это называется переводом активити в фоновый *режим* . Противоположностью этому является ситуация, когда действие находится на *переднем плане* или на экране.
- Когда пользователь возвращается в ваше приложение, то же действие перезапускается и снова становится видимым. Эта часть жизненного цикла называется ***видимым*** жизненным циклом приложения .

Когда ваше приложение работает в фоновом режиме, оно, как правило, не должно работать активно, чтобы сохранить системные ресурсы и время автономной работы. Вы используете жизненный цикл Activity и его обратные вызовы, чтобы знать, когда ваше приложение переходит в фоновый режим, чтобы вы могли приостановить любые текущие операции. Затем вы перезапускаете операции, когда ваше приложение выходит на передний план.

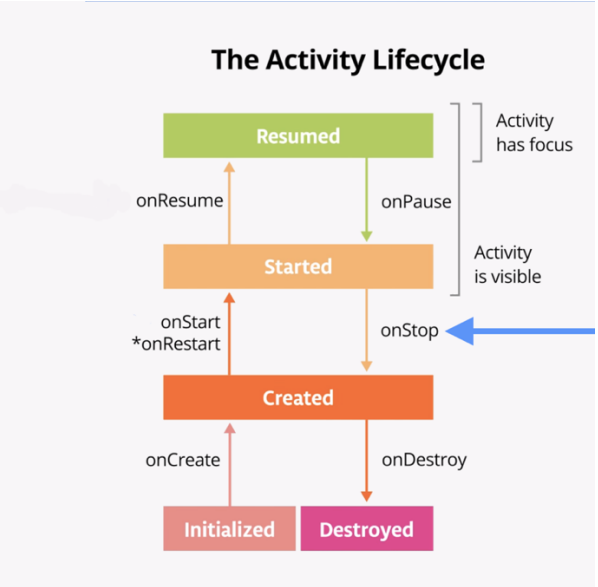
На этом этапе вы рассматриваете жизненный цикл активности, когда приложение переходит в фоновый режим и снова возвращается на передний план.

апустив приложение Dessert Clicker, нажмите на кекс несколько раз.

ажмите кнопку **«Домой»** на своем устройстве и наблюдайте за Logcat в Android Studio. Возврат на главный экран переводит ваше приложение в фоновый режим, а не полностью закрывает его. Обратите внимание, что вызываются методы onPause()и

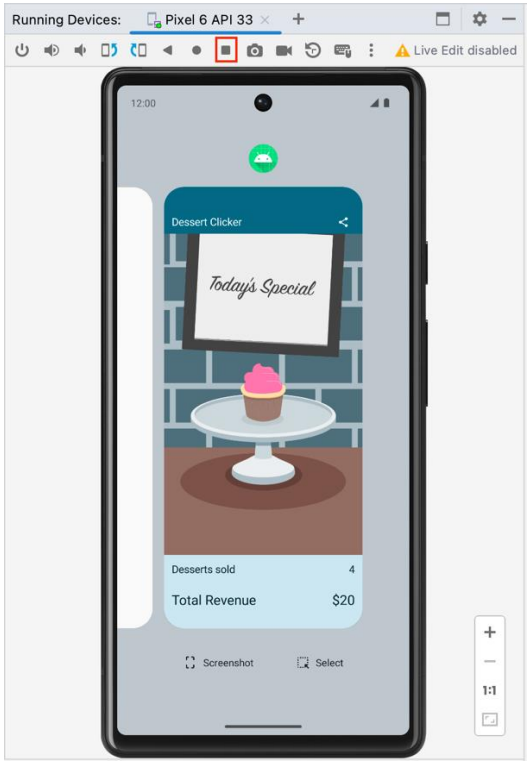
```
2024-04-26 15:00:04.905  5590-5590  MainActivity            com.example.dessertclicker  D  onPause Called
2024-04-26 15:00:05.430  5590-5590  MainActivity            com.example.dessertclicker  D  onStop Called
```

При вызове `onPause()` приложение больше не имеет фокуса. После `onStop()` приложение больше не отображается на экране. Хотя действие остановлено, объект `Activity` все еще находится в памяти в фоновом режиме. ОС Android не уничтожила активность. Пользователь может вернуться в приложение, поэтому Android сохраняет ресурсы вашей активности.



спользуйте экран «Недавние», чтобы вернуться в приложение. В эмуляторе доступ к экрану «Недавние» можно получить с помощью квадратной системной кнопки, показанной на изображении ниже.

Обратите внимание, что в Logcat действие возобновляется с помощью `onRestart()`, а `onStart()` затем возобновляется с `onResume()`.



Примечание . `onRestart()`вызывается системой только в том случае, если действие уже было создано, и в конечном итоге переходит в состояние **«Создано»** при `onStop()`вызове, но возвращается обратно в состояние **«Запущено** » вместо перехода в состояние **«Уничтожено»** . Метод `onRestart()`— это место для размещения кода, который вы хотите вызывать только в том случае, если ваша деятельность **не** запускается в первый раз.

```
2024-04-26 15:00:39.371  5590-5590  MainActivity    com.example.dessertclicker    D  onRestart Called
2024-04-26 15:00:39.372  5590-5590  MainActivity    com.example.dessertclicker    D  onStart Called
2024-04-26 15:00:39.374  5590-5590  MainActivity    com.example.dessertclicker    D  onResume Called
```

Когда активити возвращается на передний план, метод `onCreate()` больше не вызывается. Объект активити не был уничтожен, поэтому его не нужно создавать заново. Вместо `onCreate()`, вызывается метод `onRestart()`. Обратите внимание, что на этот раз, когда действие возвращается на передний план, количество **проданных десертов** сохраняется.

> апустите хотя бы одно приложение, кроме Dessert Clicker, чтобы на экране «Последние» на устройстве отображалось несколько приложений.
>
> ткройте экран «Недавние» и откройте еще одно недавнее действие. Затем вернитесь к недавним приложениям и верните Dessert Clicker на передний план.

Обратите внимание, что вы видите здесь те же обратные вызовы в Logcat, что и при нажатии кнопки **«Домой»** . Вызываются `onPause()`и `onStop()`, когда приложение переходит в фоновый режим, а затем вызываются `onRestart()`, `onStart()`и `onResume()`, когда оно возвращается.

Примечание. Они `onStart()` и `onStop()`вызываются несколько раз, когда пользователь переходит к действию и обратно.

Эти методы вызываются, когда приложение останавливается и переходит в фоновый режим или когда приложение перезапускается и возвращается на передний план. Если в таких случаях

вам нужно поработать в приложении, переопределите соответствующий метод обратного вызова жизненного цикла.

Вариант использования 3: Частично скрыть действие

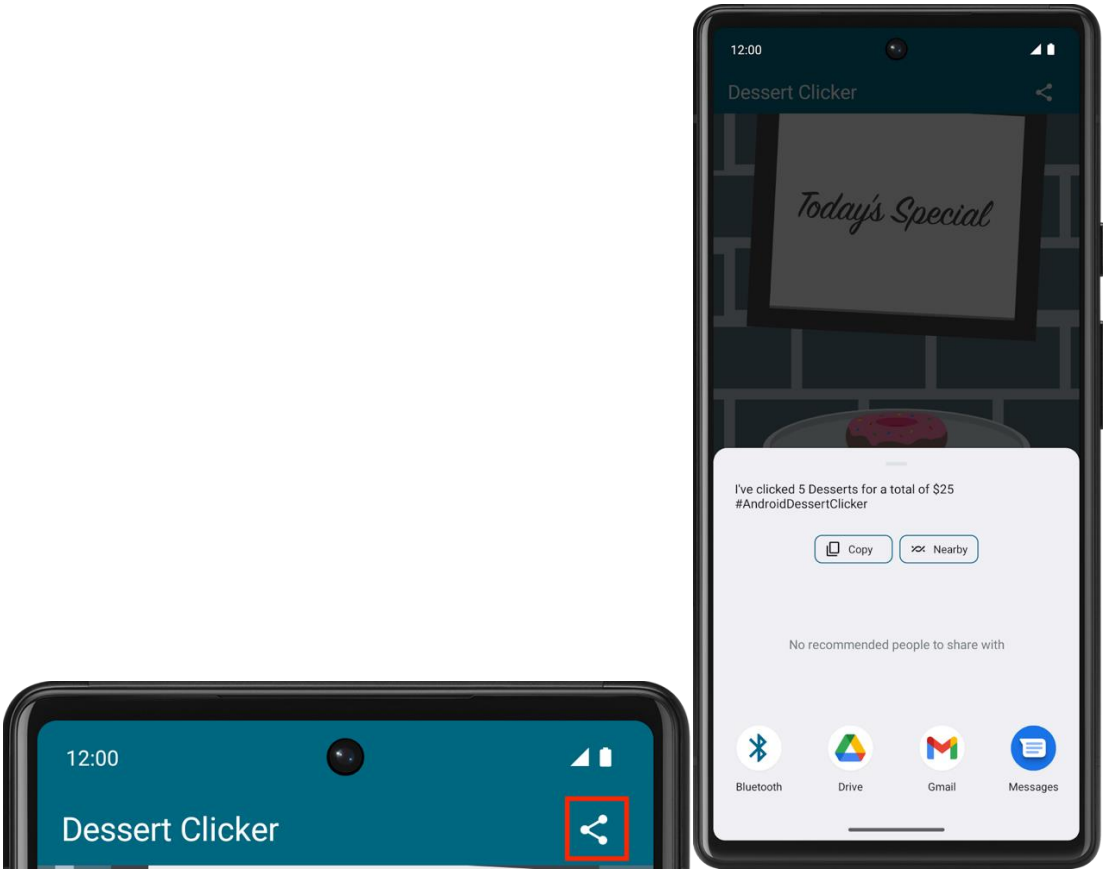
Вы узнали, что когда приложение запускается и вызывается `onStart()`, оно становится видимым на экране. При вызове `onResume()`приложение получает фокус пользователя, то есть пользователь может взаимодействовать с приложением. Часть жизненного цикла, в которой приложение полностью отображается на экране и ориентировано на пользователя, называется **временем жизни переднего плана** .

Когда приложение переходит в фоновый режим, фокус теряется после `onPause()`, и приложение больше не отображается после `onStop()`.

Разница между фокусом и видимостью важна. Действие может быть *частично* видно на экране, но не быть в центре внимания пользователя. На этом этапе вы рассмотрите один случай, когда действие частично видно, но не фокусируется на пользователе.

> апустив приложение Dessert Clicker, нажмите кнопку **«Поделиться»** в правом верхнем углу экрана.

Действия по совместному использованию отображаются в нижней половине экрана, но они по-прежнему видны в верхней половине.



зучите Logcat и обратите внимание, что был вызван только `onPause()`.

```
2024-04-26 15:01:49.535  5590-5590  MainActivity            com.example.dessertclicker  D  onPause Called
```

В этом случае использования `onStop()`не вызывается, поскольку активити все еще частично видимо. Но активити не имеет пользовательского фокуса, и пользователь не может с ним взаимодействовать — действие «поделиться», находящееся на переднем плане, имеет фокус пользователя.

Почему эта разница важна? Прерывание `onPause()`обычно длится недолго, прежде чем вернуться к своему занятию или перейти к другому действию или приложению. Обычно вы хотите постоянно обновлять пользовательский интерфейс, чтобы остальная часть вашего приложения не зависала.

Какой бы код ни выполнялся, он `onPause()` блокирует отображение других элементов, поэтому сохраняйте облегченный код `onPause()`. Например, если поступает телефонный звонок, введенный код `onPause()` может задержать уведомление о входящем вызове.

> елкните за пределами диалогового окна общего доступа, чтобы вернуться в приложение, и обратите внимание на `onResume()` вызов.

И то `onResume()`и другое `onPause()`связано с фокусом. Метод `onResume()`вызывается, когда действие получает фокус, и `onPause()`вызывается, когда действие теряет фокус.

5. Изучите изменения конфигурации.

Есть еще один случай управления жизненным циклом действий, который важно понимать: как изменения конфигурации влияют на жизненный цикл ваших действий.

Изменение *конфигурации* происходит, когда состояние устройства меняется настолько радикально, что самый простой способ для системы разрешить это изменение — полностью завершить работу и перестроить активити. Например, если пользователь меняет язык устройства, возможно, потребуется изменить весь макет, чтобы учесть разные направления текста и длину строк. Если пользователь подключает устройство к док-станции или добавляет физическую клавиатуру, макету приложения может потребоваться использовать преимущества другого размера или макета дисплея. А если ориентация устройства изменится (если устройство повернуто из книжной ориентации в альбомную или наоборот), возможно, потребуется изменить макет, чтобы он соответствовал новой ориентации. Давайте посмотрим, как приложение ведет себя в этом сценарии.

Последний обратный вызов жизненного цикла, который нужно продемонстрировать `onDestroy()`, который вызывается после `onStop()`. Он вызывается непосредственно перед уничтожением активити. Это может произойти, когда код приложения вызывает вызов `finish()` или системе необходимо уничтожить и воссоздать действие из-за изменения конфигурации.

Изменение конфигурации приводит к вызову onDestroy()

Поворот экрана — это один из типов изменения конфигурации, который приводит к остановке и перезапуску активности. Чтобы смоделировать это изменение конфигурации и изучить его последствия, выполните следующие шаги:

> компилируйте и запустите ваше приложение.
>
> бедитесь, что блокировка поворота экрана в эмуляторе отключена.
>
> оверните устройство или эмулятор в альбомный режим. Вы можете вращать эмулятор влево или вправо с помощью кнопок вращения.

зучите Logcat и поймите, что когда действие завершается, оно вызывает `onPause()`,

```
2024-04-26 15:03:32.183  5716-5716  MainActivity            com.example.dessertclicker       D  onPause
Called
2024-04-26 15:03:32.185  5716-5716  MainActivity            com.example.dessertclicker       D  onStop
Called
2024-04-26 15:03:32.205  5716-5716  MainActivity            com.example.dessertclicker       D
onDestroy Called
```

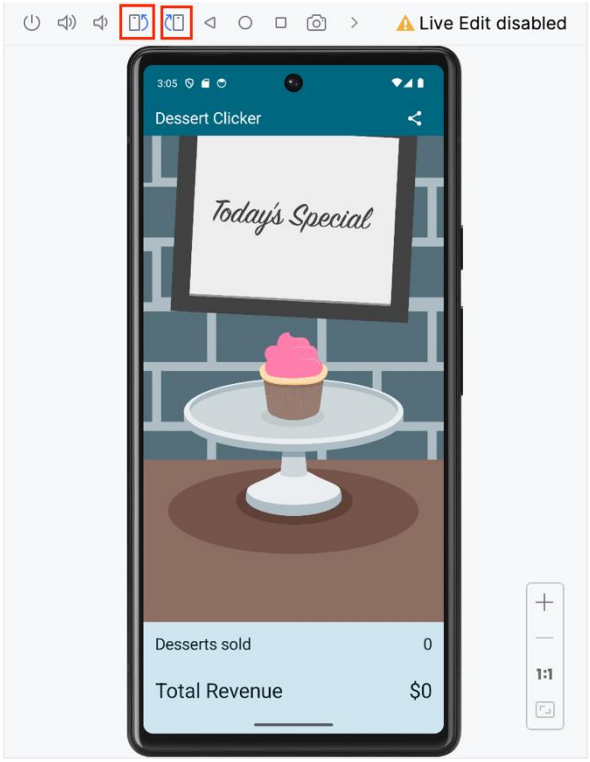
Потеря данных при повороте устройства

компилируйте и запустите приложение и откройте Logcat.

ажмите на кекс несколько раз и обратите внимание, что проданные десерты и общий доход не равны нулю.

бедитесь, что блокировка поворота экрана в эмуляторе отключена.

оверните устройство или эмулятор в альбомный режим. Вы можете вращать эмулятор влево или вправо с помощью кнопок вращения.



зучите выходные данные в Logcat. Отфильтруйте вывод на `MainActivity`.

```
2024-04-26 15:04:29.356  5809-5809  MainActivity            com.example.dessertclicker       D  onCreate
Called
2024-04-26 15:04:29.378  5809-5809  MainActivity            com.example.dessertclicker       D  onStart
Called
2024-04-26 15:04:29.382  5809-5809  MainActivity            com.example.dessertclicker       D
onResume Called
2024-04-26 15:06:52.168  5809-5809  MainActivity            com.example.dessertclicker       D  onPause
Called
```

```
2024-04-26 15:06:52.183  5809-5809  MainActivity            com.example.dessertclicker  D  onStop Called
2024-04-26 15:06:52.219  5809-5809  MainActivity            com.example.dessertclicker  D  onDestroy Called
2024-04-26 15:06:52.302  5809-5809  MainActivity            com.example.dessertclicker  D  onCreate Called
2024-04-26 15:06:52.308  5809-5809  MainActivity            com.example.dessertclicker  D  onStart Called
2024-04-26 15:06:52.312  5809-5809  MainActivity            com.example.dessertclicker  D  onResume Called
```

Обратите внимание: когда устройство или эмулятор поворачивает экран, система вызывает все обратные вызовы жизненного цикла, чтобы завершить действие. Затем, когда действие создается заново, система вызывает все обратные вызовы жизненного цикла, чтобы запустить действие.

Когда устройство поворачивается, а действие выключается и создается заново, действие возобновляется со значениями по умолчанию — изображением десерта, количеством проданных десертов и общим доходом, сбрасываемым обратно на ноль.

Чтобы узнать, почему эти значения сбрасываются и как их исправить, вам необходимо узнать о жизненном цикле составного объекта и о том, как он умеет наблюдать и сохранять свое состояние.

Жизненный цикл составного элемента

Пользовательский интерфейс вашего приложения изначально создается на основе выполнения компонуемых функций в процессе под названием Composition.

При изменении состояния вашего приложения запланирована рекомпозиция. Рекомпозиция — это когда Compose повторно выполняет компонуемые функции, состояние которых могло измениться, и создает обновленный пользовательский интерфейс. Состав обновляется с учетом этих изменений.

Единственный способ создать или обновить композицию — это ее первоначальная композиция и последующие рекомпозиции.

Составные функции имеют собственный жизненный цикл, не зависящий от жизненного цикла действия. Его жизненный цикл состоит из событий: вход в состав, перекомпоновка 0 или более раз, а затем выход из состава.

Чтобы Compose мог отслеживать и запускать рекомпозицию, ему необходимо знать, когда состояние изменилось. Чтобы указать Compose, что он должен отслеживать состояние объекта, объект должен иметь тип `State` или `MutableState`. Тип `State` неизменяем и доступен только для чтения. Тип `MutableState` является изменяемым и допускает чтение и запись.

Вы уже видели и использовали `MutableState` приложения Lemonade и Tip Time в предыдущих лабораториях кода.

Чтобы создать изменяемую переменную `revenue`, вы объявляете ее с помощью `mutableStateOf`. `0` это его начальное значение по умолчанию.

```
var revenue = mutableStateOf(0)
```

Хотя этого достаточно, чтобы Compose инициировал рекомпозицию при изменении значения дохода, недостаточно сохранить его обновленное значение. Каждый раз, когда составной объект повторно выполняется, он повторно инициализирует значение дохода до исходного значения по умолчанию, равного `0`.

Чтобы указать Compose сохранять и повторно использовать свое значение во время рекомпозиции, вам необходимо объявить его с помощью `remember`.

```
var revenue by remember { mutableStateOf(0) }
```

Если значение `revenue` изменяется, Compose планирует все составные функции, которые считывают это значение, для повторной композиции.

Хотя Compose запоминает состояние дохода во время рекомпозиции, он не сохраняет это состояние во время изменения конфигурации. Чтобы Compose сохранял состояние во время изменения конфигурации, вы должны использовать **`rememberSaveable`**.

Дополнительные сведения и практические сведения см. [во введении, чтобы указать в Compose](#) codelab.

Используйте **rememberSaveable** для сохранения значений при изменении конфигурации.

Вы используете функцию `rememberSaveable` для сохранения необходимых вам значений, если ОС Android уничтожает и воссоздает действие.

Чтобы сохранить значения во время рекомпозиции, вам нужно использовать `remember`. Используйте `rememberSaveable` для сохранения значений во время рекомпозиций и изменений конфигурации.

Примечание. Иногда Android завершает весь процесс приложения, включая все действия, связанные с приложением. Android выполняет такое отключение, когда система находится в состоянии стресса и существует опасность визуального зависания, поэтому на этом этапе не выполняются никакие дополнительные обратные вызовы или код. Процесс вашего приложения просто закрывается в фоновом режиме. Но для пользователя это не похоже на то, что приложение закрыто. Когда пользователь возвращается к приложению, которое система Android завершает, Android перезапускает это приложение. Вы хотите гарантировать, что в этом случае пользователь не понесет потери данных.

Сохранение значения с помощью `rememberSaveable` гарантирует, что оно будет доступно при восстановлении активности, если это необходимо.

обновите группу из пяти переменных, которые в настоящее время используются

```
var revenue by remember { mutableStateOf(0) }
...
var currentDessertImageId by remember {mutableStateOf(desserts[currentDessertIndex].imageId) }
var revenue by rememberSaveable { mutableStateOf(0) }
...
var currentDessertImageId by rememberSaveable { mutableStateOf(desserts[currentDessertIndex].imageId)
```

```
}
```

компилируйте и запустите ваше приложение.

ажмите на кекс несколько раз и обратите внимание, что проданные десерты и общий доход не равны нулю.

оверните устройство или эмулятор в альбомный режим.

братите внимание, что после уничтожения и воссоздания активности изображение десерта, проданные десерты и общий доход восстанавливаются до прежних значений.

7. Резюме

Жизненный цикл активности

- Жизненный *цикл действия* — это набор состояний, через которые проходит действие. Жизненный цикл активности начинается, когда ОС Android впервые создает активность, и заканчивается, когда ОС уничтожает активность.
- Когда пользователь перемещается между действиями внутри и снаружи вашего приложения, каждое действие перемещается между состояниями жизненного цикла действия.
- Каждое состояние жизненного цикла активности имеет соответствующий метод обратного вызова, который вы можете переопределить в своем `Activity`классе. Основной набор методов жизненного цикла: `onCreate()`, `onRestart()`, `onStart()`, `onResume()`, `onPause()`, `onStop()`, `onDestroy()`.
- Чтобы добавить поведение, которое происходит при переходе активности в состояние жизненного цикла, переопределите метод обратного вызова состояния.
- Чтобы добавить методы переопределения скелета в классы в Android Studio, выберите **«Код» > «Методы переопределения...»** или нажмите `Control+O`.

Ведение журнала с помощью журнала

- API ведения журнала Android и, в частности класс `Log`, позволяют вам писать короткие сообщения, которые отображаются в Logcat в Android Studio.
- Используйте `Log.d()` для написания отладочного сообщения. Этот метод принимает два аргумента: *тег* журнала, обычно имя класса, и *сообщение* журнала, короткую строку.
- Используйте окно **Logcat** в Android Studio для просмотра системных журналов, включая написанные вами сообщения.

Изменения конфигурации

- Изменение *конфигурации* происходит, когда состояние устройства меняется настолько радикально, что самый простой способ для системы разрешить это изменение — уничтожить и перестроить активность.
- Самый распространенный пример изменения конфигурации — когда пользователь поворачивает устройство из книжного режима в альбомный или из альбомного в портретный. Изменение конфигурации также может произойти, когда меняется язык устройства или пользователь подключает аппаратную клавиатуру.

- Когда происходит изменение конфигурации, Android вызывает все обратные вызовы завершения жизненного цикла активности. Затем Android перезапускает действие с нуля, выполняя все обратные вызовы запуска жизненного цикла.
- Когда Android закрывает приложение из-за изменения конфигурации, оно перезапускает действие с помощью `onCreate()`.
- Чтобы сохранить значение, которое должно пережить изменение конфигурации, объявите его переменные с помощью `rememberSaveable`.

Узнать больше

сорт

- Просмотр журналов с помощью Logcat
- Сохраняемый API

  сорт

  сорт
- Руководство разработчика занятий