Лекция. Этапы жизненного цикла активности

Прежде чем начать

В этом уроке вы узнаете о фундаментальной части Android - жизненном цикле активности.

В течение своей жизни активность переходит через различные состояния, а иногда и возвращается в них. Этот переход из одного состояния в другое называется **жизненным циклом активности**.

В Android активность - это точка входа для взаимодействия с пользователем.

В прошлом одна активность отображала один экран в приложении. При современных передовых методах одна активность может отображать несколько экранов, меняя их местами по мере необходимости.

Жизненный цикл активности длится от создания активности до ее уничтожения, когда система возвращает ресурсы активности. По мере того как пользователь входит и выходит из активности, каждая активность переходит из одного состояния в другое в жизненном цикле активности.

Как разработчик Android, вы должны понимать жизненный цикл активности. Если ваши действия не будут правильно реагировать на изменения состояния жизненного цикла, ваше приложение может создавать странные ошибки, путать поведение пользователей или использовать слишком много системных ресурсов Android. Понимание жизненного цикла Android и правильная реакция на изменения состояния жизненного цикла - важная часть разработки Android.

Предварительные условия

- Знание того, что такое активность и как ее создать в своем приложении
- Знание того, что делает метод `onCreate()` активности и какие операции выполняются в этом методе.

Что вы узнаете

- Как выводить информацию о регистрации в Logcat
- Основы жизненного цикла активности и обратные вызовы, которые вызываются при переходе активности из одного состояния в другое
- Как переопределять методы обратного вызова жизненного цикла для выполнения операций в разные моменты жизненного цикла активности

Что вы будете создавать

- Измените стартовое приложение под названием `Dessert Clicker` , чтобы добавить информацию для ведения журнала, которая отображается в Logcat.
- Переопределите методы обратного вызова жизненного цикла и запишите в журнал изменения состояния активности.
- Запустите приложение и обратите внимание на информацию в журнале, которая появляется при запуске, остановке и возобновлении активности.
- Реализуйте функцию rememberSaveable для сохранения данных приложения, которые могут быть потеряны при изменении конфигурации устройства.

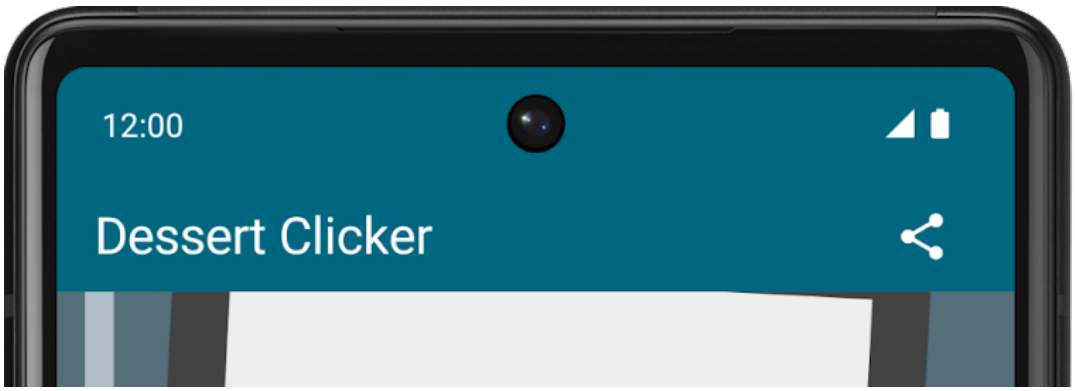
Обзор приложения

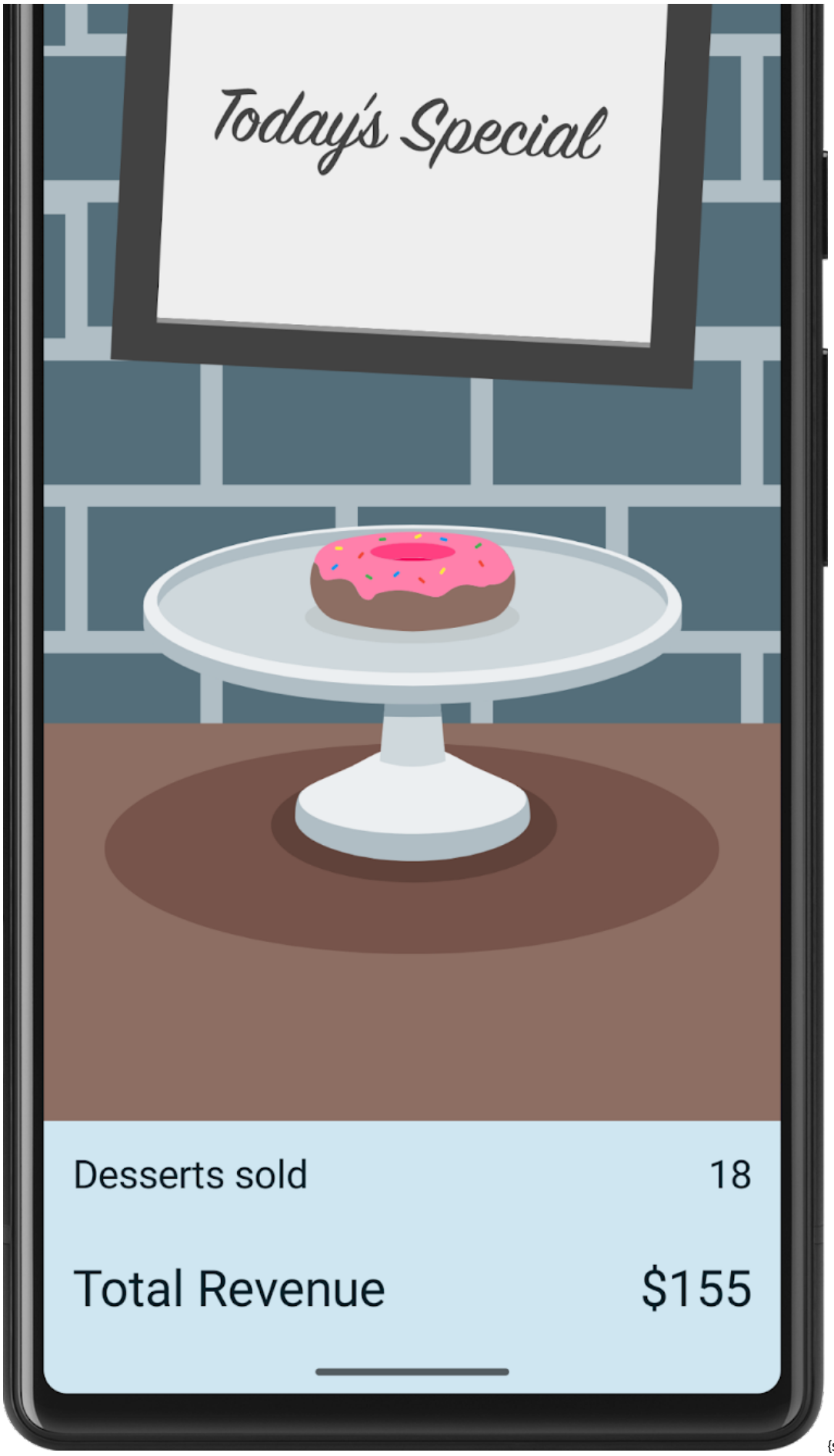
- в папку repositories
- скачайте папку dessert-clicker.7z
- распакуйте
- перейдите в папку проекта
- переключитесь на ветку `starter`

В этом кодовом модуле вы работаете со стартовым приложением под названием `Dessert Clicker` . В `Dessert Clicker` каждый раз, когда пользователь нажимает на десерт на экране, приложение «покупает» десерт для пользователя.

Приложение обновляет значения в макете для:

Количество десертов, которые были «куплены» Общая выручка за «купленные» десерты



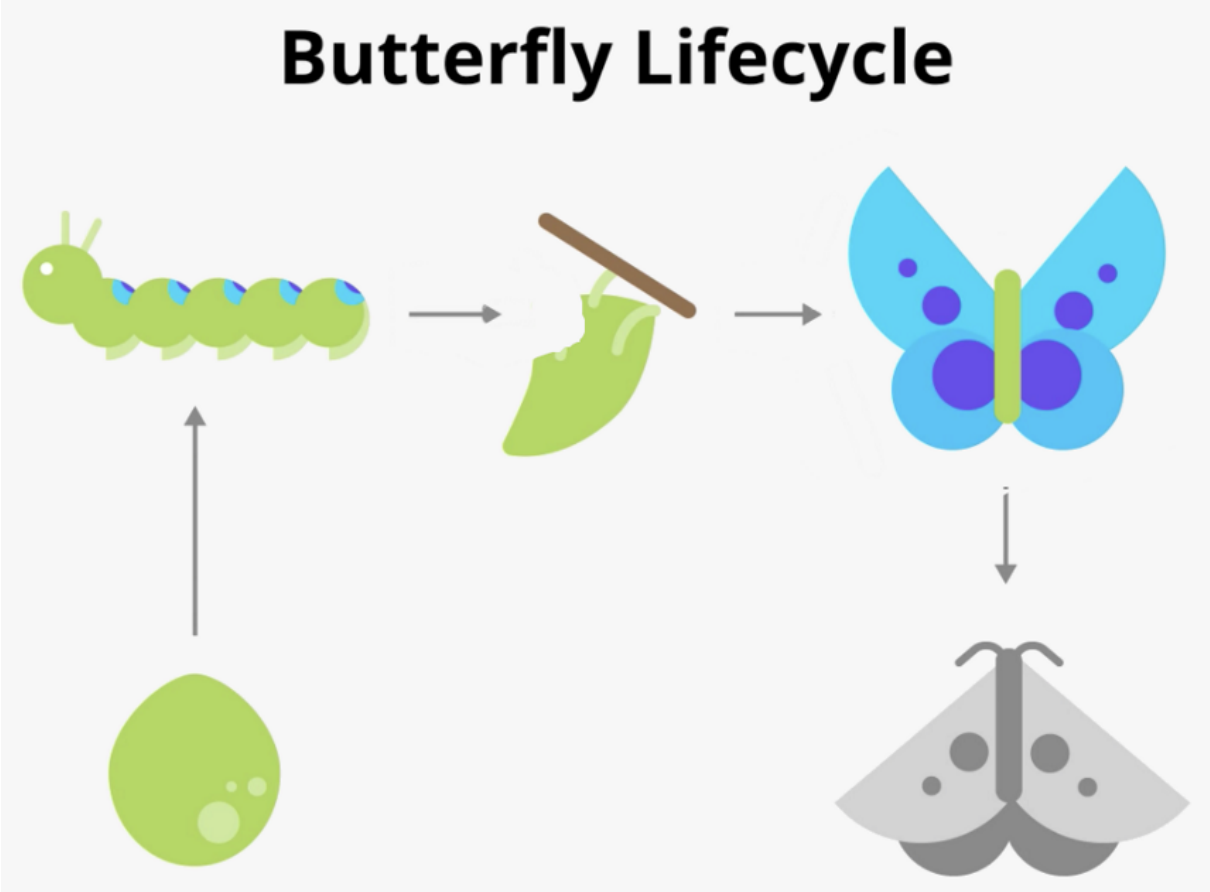


{style="width:500px;"}

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

Изучите методы жизненного цикла и добавьте базовое протоколирование

У каждого действия есть так называемый жизненный цикл. Этот термин является аллюзией на жизненные циклы растений и животных, например, жизненный цикл бабочки - различные состояния бабочки показывают ее рост от яйца до гусеницы, куколки, бабочки и смерти.



{style=«width:500px»}

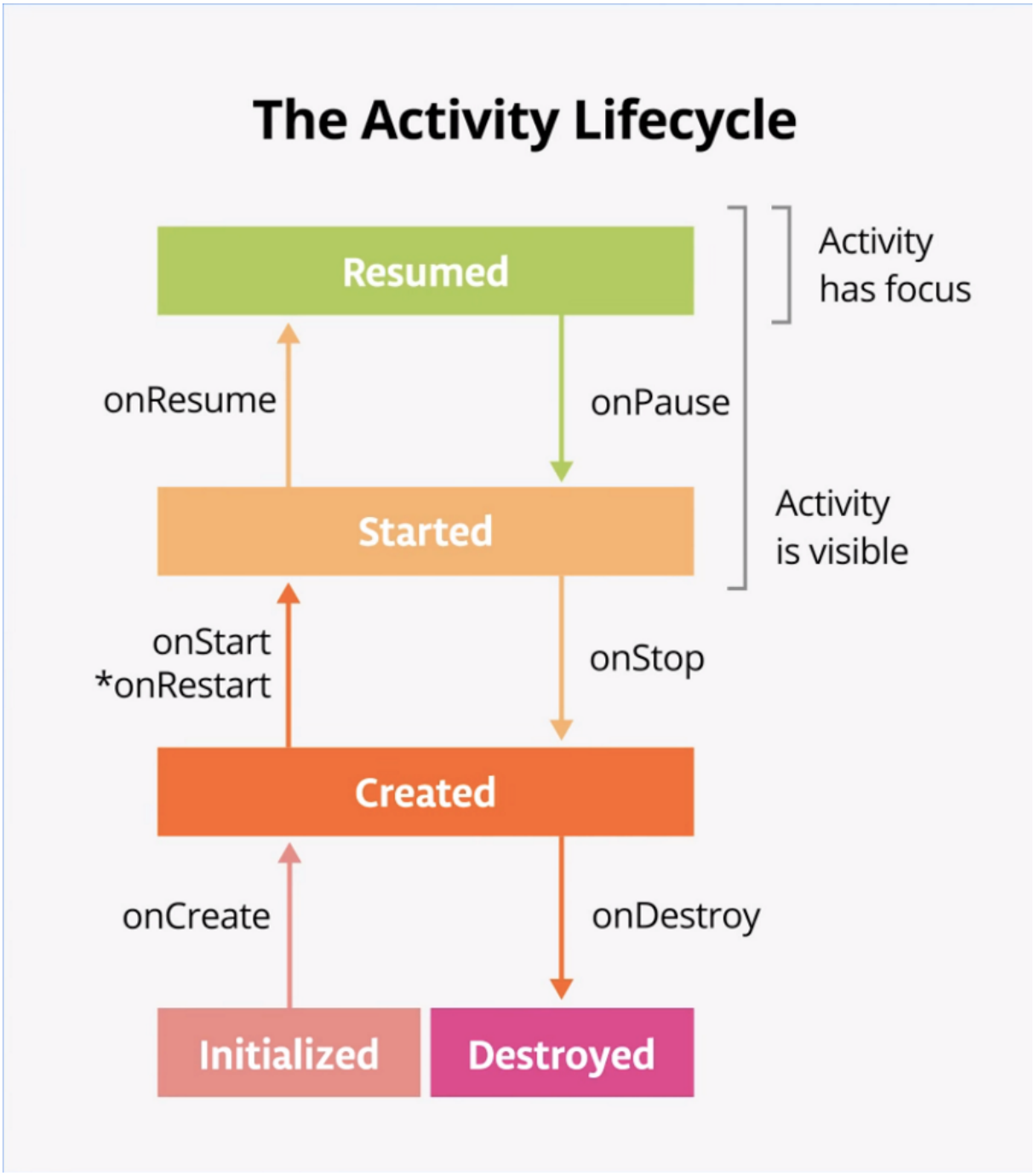
Аналогично, жизненный цикл активности состоит из различных состояний, через которые может пройти активность, начиная с момента ее инициализации и заканчивая ее уничтожением, когда операционная система (ОС) возвращает ей память. Как правило, точкой входа в программу является метод `main()`. Однако деятельность в Android начинается с метода `onCreate()`; этот метод эквивалентен стадии яйца в приведенном выше примере. Вы уже много раз использовали действия на протяжении этого курса, и вы можете узнать метод onCreate(). По мере того как пользователь запускает ваше приложение, переходит от одного вида деятельности к другому, перемещается внутри и вне приложения, деятельность меняет состояние.

На следующей диаграмме показаны все состояния жизненного цикла активности. Как видно из названий, эти состояния отражают состояние активности. Обратите внимание, что, в отличие от жизненного цикла бабочки, активность может переходить из одного состояния в другое на протяжении всего жизненного цикла, а не двигаться только в одном направлении.

> Примечание: В приложении для Android может быть несколько активностей. Однако рекомендуется иметь одну активность, и до сих пор именно ее вы реализовывали в этом курсе.

{style=«width:500px»}

Часто требуется изменить поведение или запустить код при изменении состояния жизненного цикла активности. Поэтому сам класс `Activity` и любые подклассы Activity, такие как `ComponentActivity`, реализуют набор методов обратного вызова жизненного цикла. Android вызывает эти обратные вызовы, когда активность переходит из одного состояния в другое, и вы можете переопределить эти методы в своих собственных активностях, чтобы выполнять задачи в ответ на изменения состояния жизненного цикла. На следующей диаграмме показаны состояния жизненного цикла вместе с доступными переопределяемыми обратными вызовами.



> Примечание: звездочка в методе `onRestart()` указывает на то, что этот метод не вызывается каждый раз, когда состояние переходит между Created и Started. Он вызывается только в том случае, если была вызвана функция `onStop()` и впоследствии активность была перезапущена.

Важно знать, когда Android вызывает переопределяемые обратные вызовы и что нужно делать в каждом методе обратного вызова, но обе эти диаграммы сложны и могут запутать. В этом уроке вместо того, чтобы просто прочитать, что означает каждое состояние и обратный вызов, вы проведете небольшую детективную работу и укрепите свое понимание жизненного цикла активности Android.

Шаг 1: Изучите метод `onCreate()` и добавьте логирование

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 протоколирования в Android. Ведение журнала позволяет записывать короткие сообщения в консоль во время работы приложения и использовать их, чтобы увидеть, когда срабатывают различные обратные вызовы.

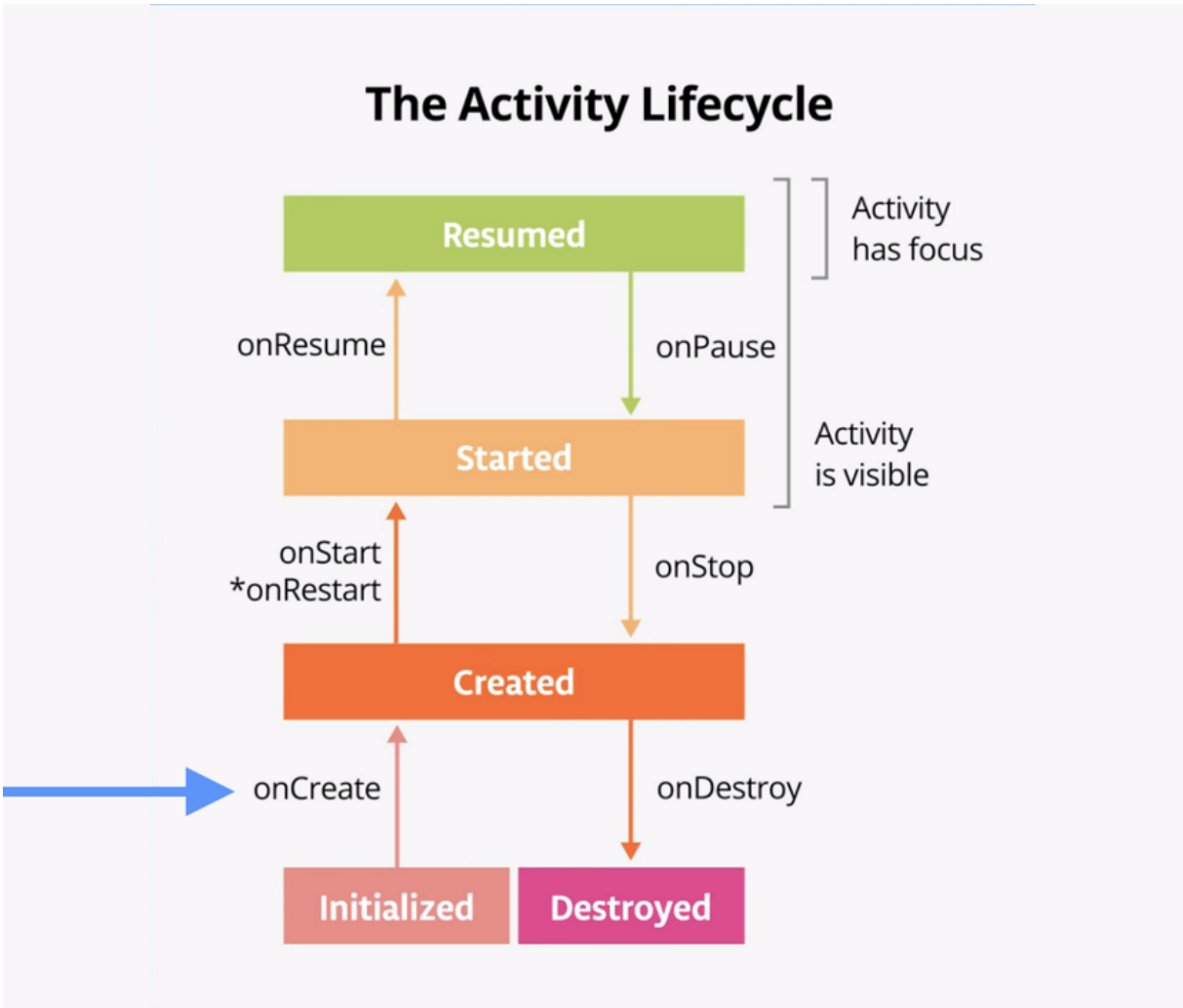
Запустите приложение `Dessert Clicker` и несколько раз нажмите на изображение десерта. Обратите внимание, как изменяется значение проданных десертов и общая сумма в долларах.

Откройте файл `MainActivity.kt` и изучите метод `onCreate()` для этой активности:

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

```
}
```

На диаграмме жизненного цикла активности вы можете узнать метод `onCreate()`, потому что вы уже использовали этот обратный вызов. Это единственный метод, который должна реализовывать каждая активность. Метод `onCreate()` - это место, где вы должны выполнять любые одноразовые инициализации для вашей активности. Например, в onCreate() вы вызываете `setContent()`, который задает макет пользовательского интерфейса активности.



Метод жизненного цикла onCreate() вызывается один раз, сразу после инициализации активности - когда ОС создает новый объект Activity в памяти. После выполнения onCreate() активность считается созданной.

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

Добавьте следующую константу на верхний уровень файла MainActivity.kt, над объявлением класса MainActivity. Хорошая традиция - объявлять константу `TAG` в файле, так как ее значение не будет меняться.

Чтобы обозначить ее как константу времени компиляции, используйте `const` при объявлении переменной. Константа времени компиляции - это значение, которое известно во время компиляции.

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

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

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

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

```

```

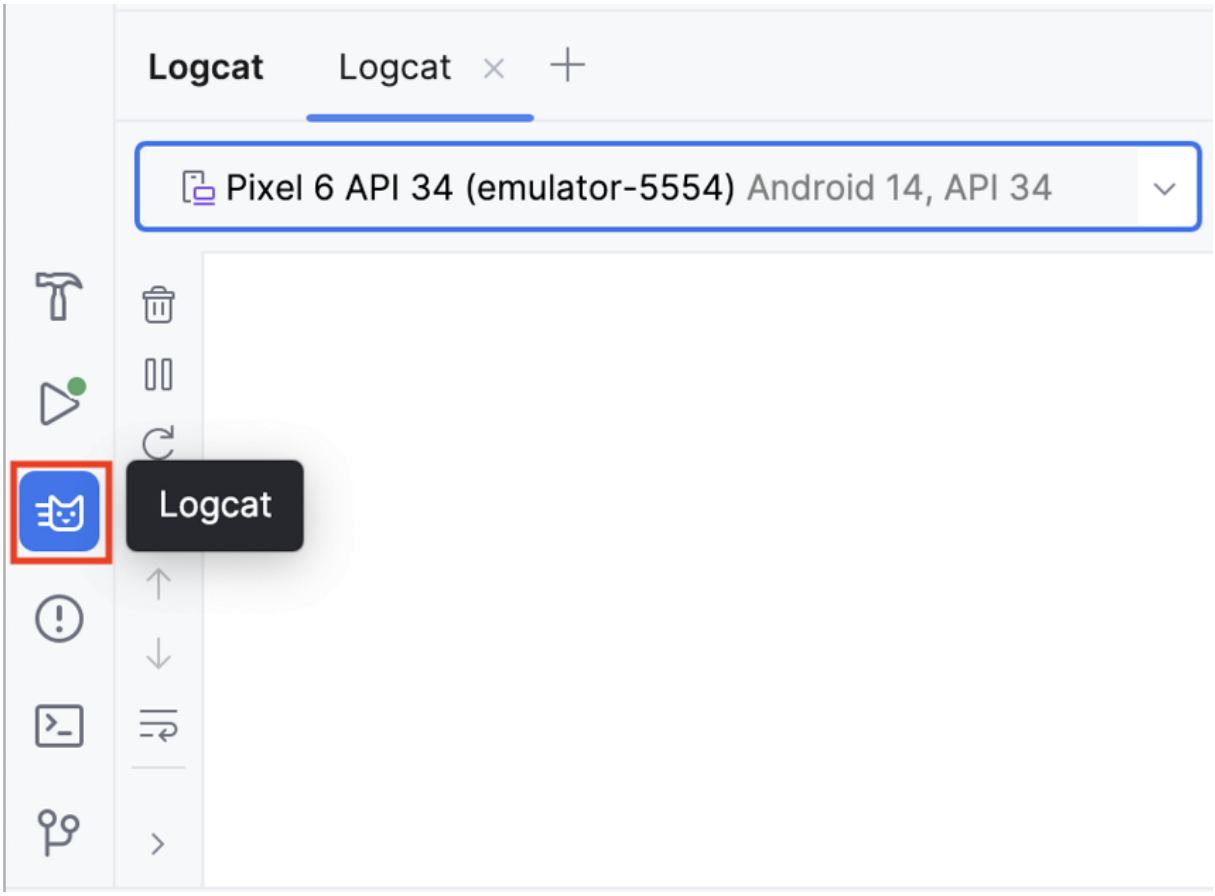
```
import android.util.Log
```

Класс Log записывает сообщения в `Logcat`. `Logcat` - это консоль для записи сообщений в журнал. Здесь появляются сообщения Android о вашем приложении, включая сообщения, которые вы явно отправляете в журнал с помощью метода `Log.d()` или других методов класса Log.

Есть три важных аспекта инструкции Log:

Приоритет сообщения журнала, то есть насколько важно это сообщение. В данном случае Log.v() записывает в журнал подробные сообщения. Метод Log.d() записывает отладочное сообщение. Другие методы класса Log включают Log.i() для информационных сообщений, Log.w() для предупреждений и Log.e() для сообщений об ошибках. Тег журнала (первый параметр), в данном случае «MainActivity». Тег - это строка, которая позволяет легче находить сообщения журнала в Logcat. Обычно тег - это имя класса. Собственно сообщение журнала, называемое msg (второй параметр), представляет собой короткую строку, в данном случае «onCreate Called».

- Скомпилируйте и запустите приложение `Dessert Clicker` . Вы не увидите никаких различий в поведении приложения при нажатии на десерт. В Android Studio в нижней части экрана перейдите на вкладку Logcat.

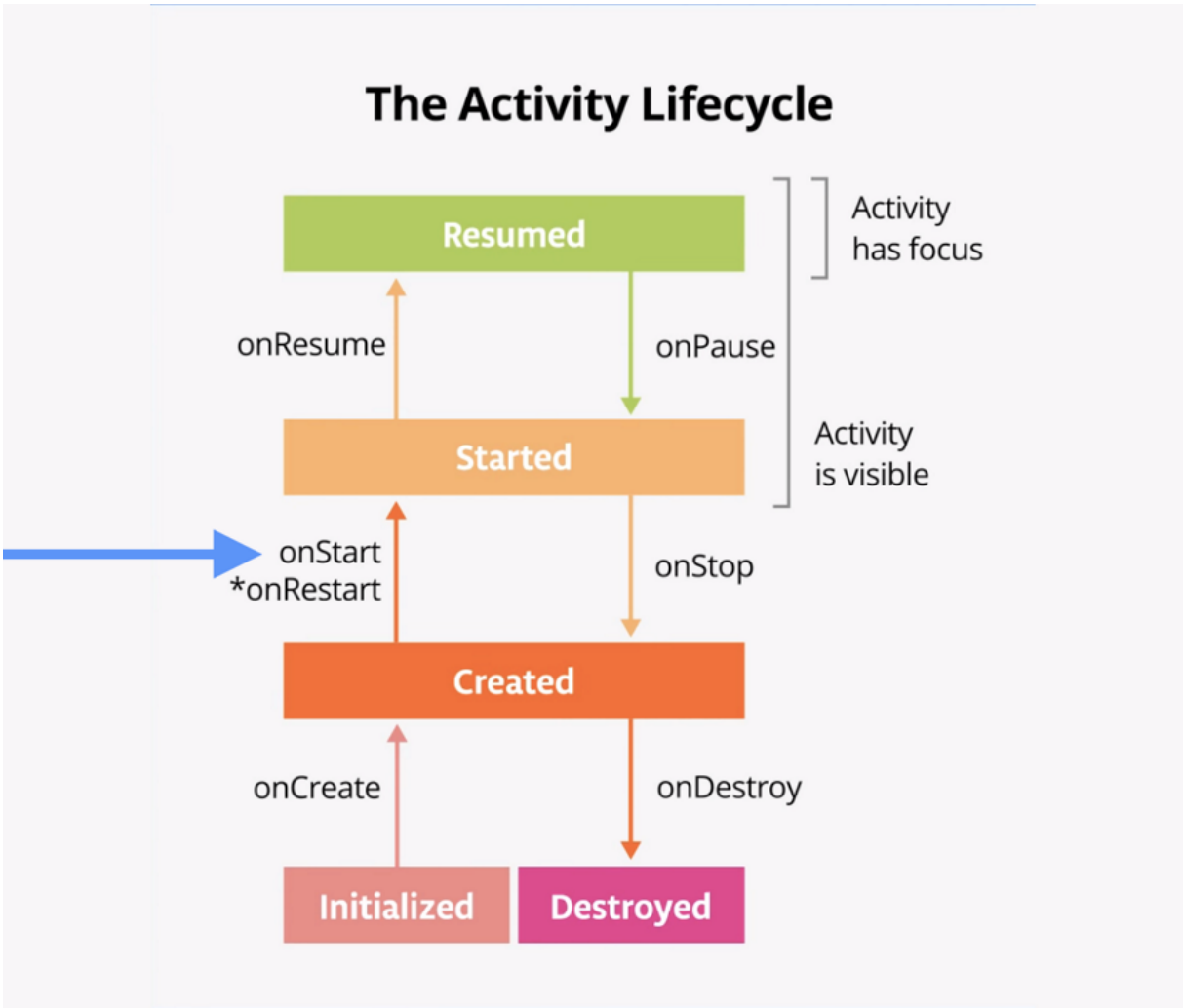


В окне `Logcat` введите в поле поиска tag:MainActivity.

`Logcat` может содержать множество сообщений, большинство из которых вам не пригодятся. Вы можете фильтровать записи Logcat различными способами, но поиск - самый простой. Поскольку в коде вы использовали MainActivity в качестве тега журнала, вы можете использовать этот тег для фильтрации журнала. Сообщение в журнале включает дату и время, тег журнала, имя пакета (com.example.dessertclicker) и собственно сообщение. Поскольку это сообщение появляется в журнале, вы знаете, что функция onCreate() была выполнена.

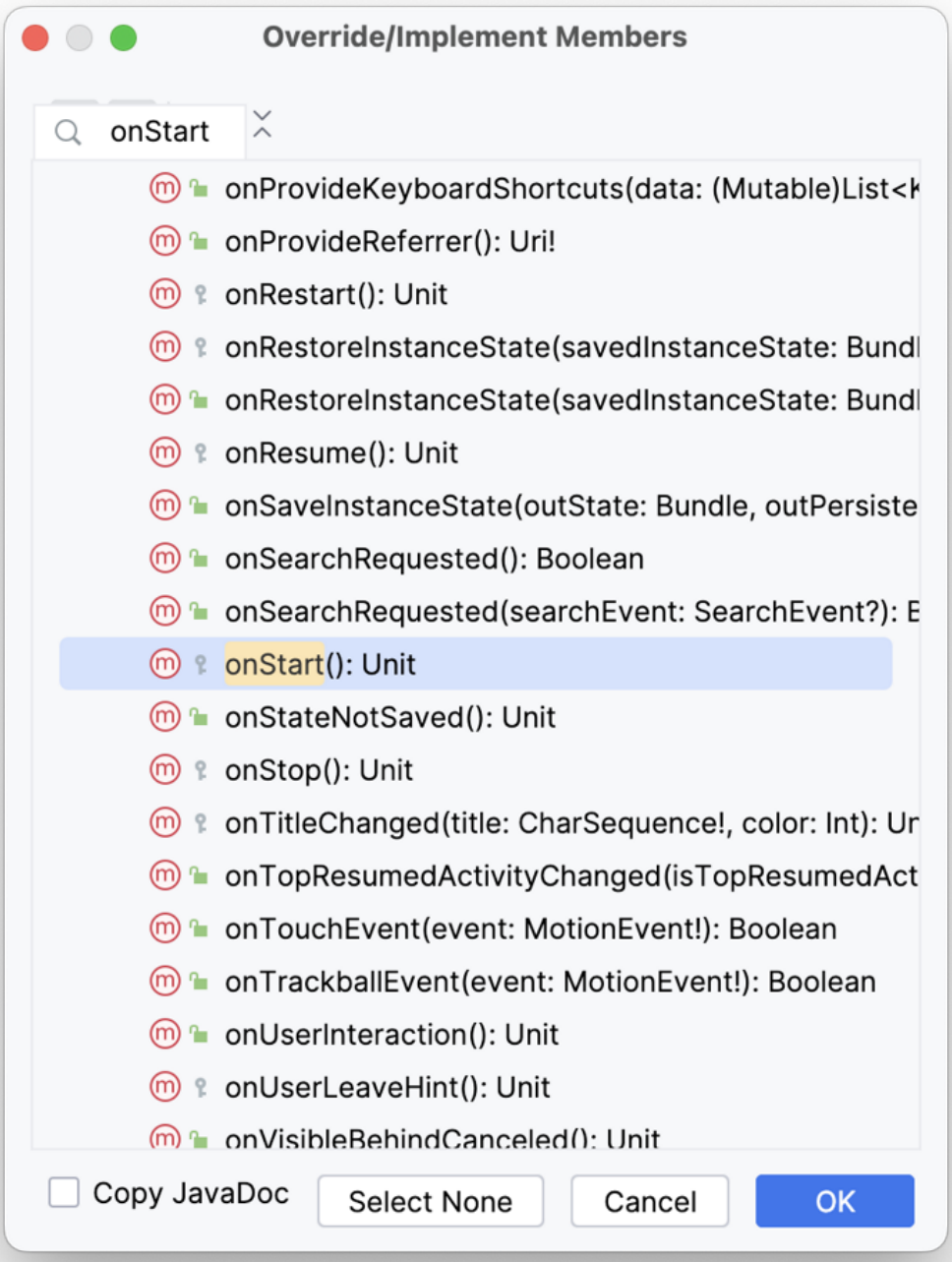
Шаг 2: Реализация метода onStart()

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



Обратите внимание, что onStart() сопряжен с соответствующим методом onStop() жизненного цикла. Если пользователь запустит ваше приложение, а затем вернется на главный экран устройства, активность будет остановлена и больше не будет видна на экране.

В Android Studio, когда открыт файл MainActivity.kt и курсор находится в классе MainActivity, выберите Code > Override Methods... или нажмите Control+O. Появится диалог с длинным списком всех методов, которые вы можете переопределить в этом классе.



{style="width:500px"}

- Начните вводить данные с помощью кнопки Начать, чтобы найти нужный метод. Чтобы прокрутить список до следующего подходящего элемента, используйте стрелку вниз. Выберите onStart() из списка и нажмите OK, чтобы вставить код переопределения шаблона. Код выглядит так, как показано в следующем примере:

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

- Внутри метода onStart() добавьте сообщение в журнал:

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

- Скомпилируйте и запустите приложение `Dessert Clicker` и откройте панель Logcat.
- Введите `tag:MainActivity` в поле поиска, чтобы отфильтровать журнал. Обратите внимание, что методы onCreate() и onStart() были вызваны один за другим, и что ваша активность видна на экране.
- Нажмите кнопку `Home` на устройстве, а затем используйте экран `Recents`, чтобы вернуться к деятельности. Обратите внимание, что активность возобновляется с того места, где она остановилась, со всеми теми же значениями, и что onStart() регистрируется во второй раз в Logcat. Обратите внимание также на то, 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
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.

Шаг 3: Добавьте дополнительные сообщения журнала

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

- Переопределите оставшиеся методы жизненного цикла в MainActivity и добавьте сообщения о журнале 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")
}
```

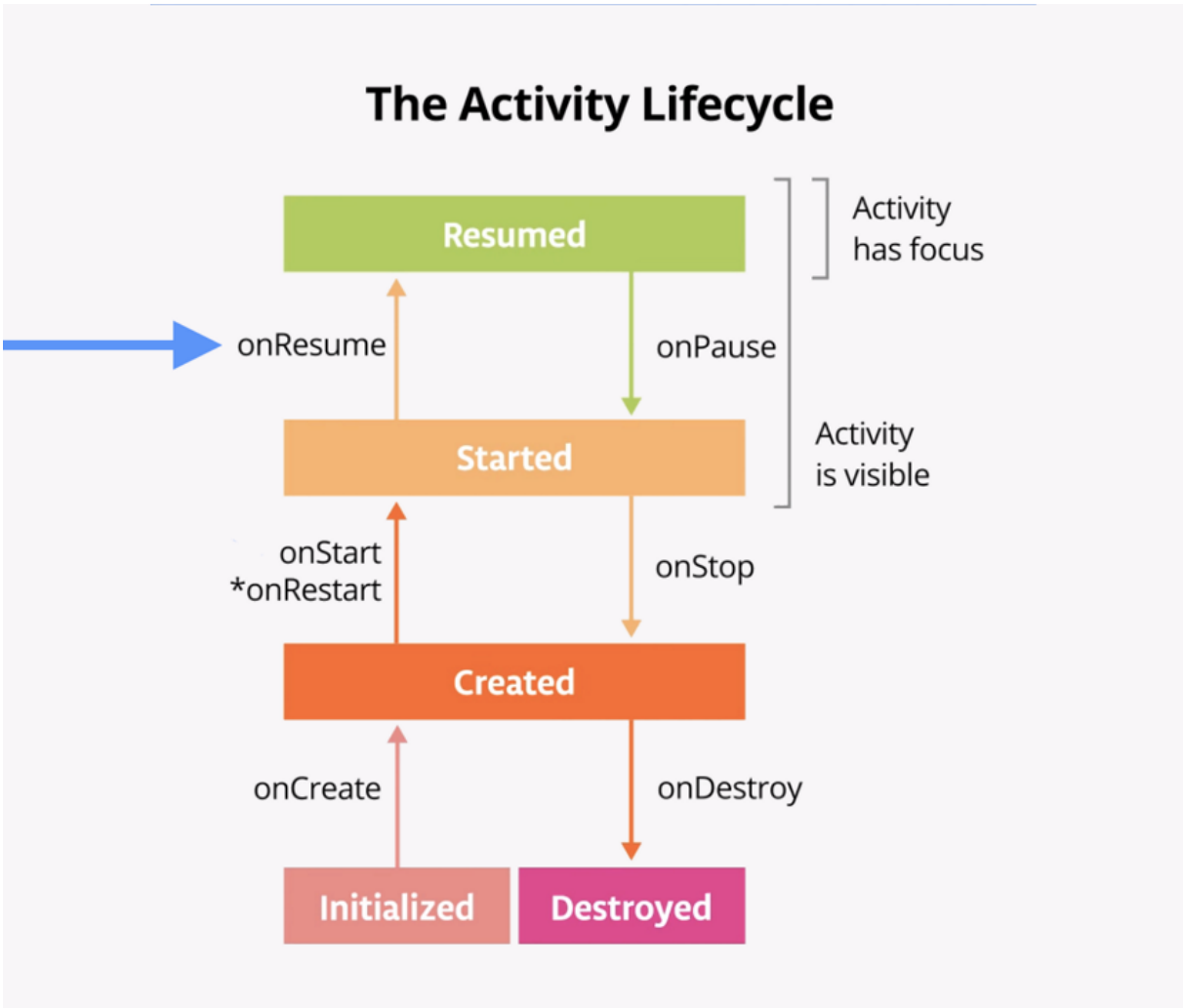
- Скомпилируйте и снова запустите `Dessert Clicker` и просмотрите Logcat.

Обратите внимание, что на этот раз, помимо onCreate() и onStart(), в журнале появилось сообщение об обратном вызове жизненного цикла onResume().

```
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 деятельность начинается с самого начала, вы видите, как все три этих обратных вызова жизненного цикла вызываются по порядку:

- onCreate(), когда система создает приложение.
- onStart() делает приложение видимым на экране, но пользователь еще не может с ним взаимодействовать.
- onResume() выводит приложение на передний план, и пользователь теперь может с ним взаимодействовать. Несмотря на название, метод onResume() вызывается при запуске, даже если возобновлять нечего.



{style=«width:500px»}

Изучите сценарии использования жизненного цикла

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

Пример использования 1: открытие и закрытие активности Начните с самого простого варианта использования, который заключается в первом запуске приложения и последующем его закрытии.

Скомпилируйте и запустите приложение `Dessert Clicker` , если оно еще не запущено. Как вы видели, обратные вызовы onCreate(), onStart() и onResume() вызываются, когда активность запускается в первый раз.

```
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
```

Несколько раз коснитесь кекса. Нажмите кнопку «Назад» на устройстве. Заметьте в Logcat, что 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 использование кнопки «Назад» приводит к тому, что активность (и приложение) удаляется с экрана и перемещается в конец стека активностей.

ОС Android может закрыть вашу активность, если ваш код вручную вызовет метод finish() активности или если пользователь принудительно завершит работу приложения. Например, пользователь может принудительно выйти из приложения или закрыть его на экране Recents. ОС также может самостоятельно завершить вашу активность, если приложение долгое время не показывалось на экране. Android делает это для экономии заряда батареи и возврата ресурсов, которые использовало приложение, чтобы они были доступны другим приложениям. Это лишь несколько примеров

того, почему система Android уничтожает вашу активность. Есть и другие случаи, когда система Android уничтожает вашу активность без предупреждения.

> Примечание: функции onCreate() и onDestroy(), которые будут рассмотрены в этом кодебаге, вызываются только один раз в течение жизни одного экземпляра активности: onCreate() для инициализации приложения в первый раз, а onDestroy() для обнуления, закрытия или уничтожения объектов, которые активность могла использовать, чтобы они не продолжали использовать ресурсы, например память.

Пример использования 2: Переход от активности и обратно к ней

Теперь, когда вы запустили приложение и закрыли его, вы увидели большинство состояний жизненного цикла, когда активность создается в первый раз. Вы также видели большинство состояний жизненного цикла, через которые проходит активность, когда ее закрывают. Но пользователи взаимодействуют со своими устройствами Android, переключаясь между приложениями, возвращаясь домой, запуская новые приложения и прерываясь на другие действия, такие как телефонные звонки.

Ваша активность не закрывается полностью каждый раз, когда пользователь переходит от нее:

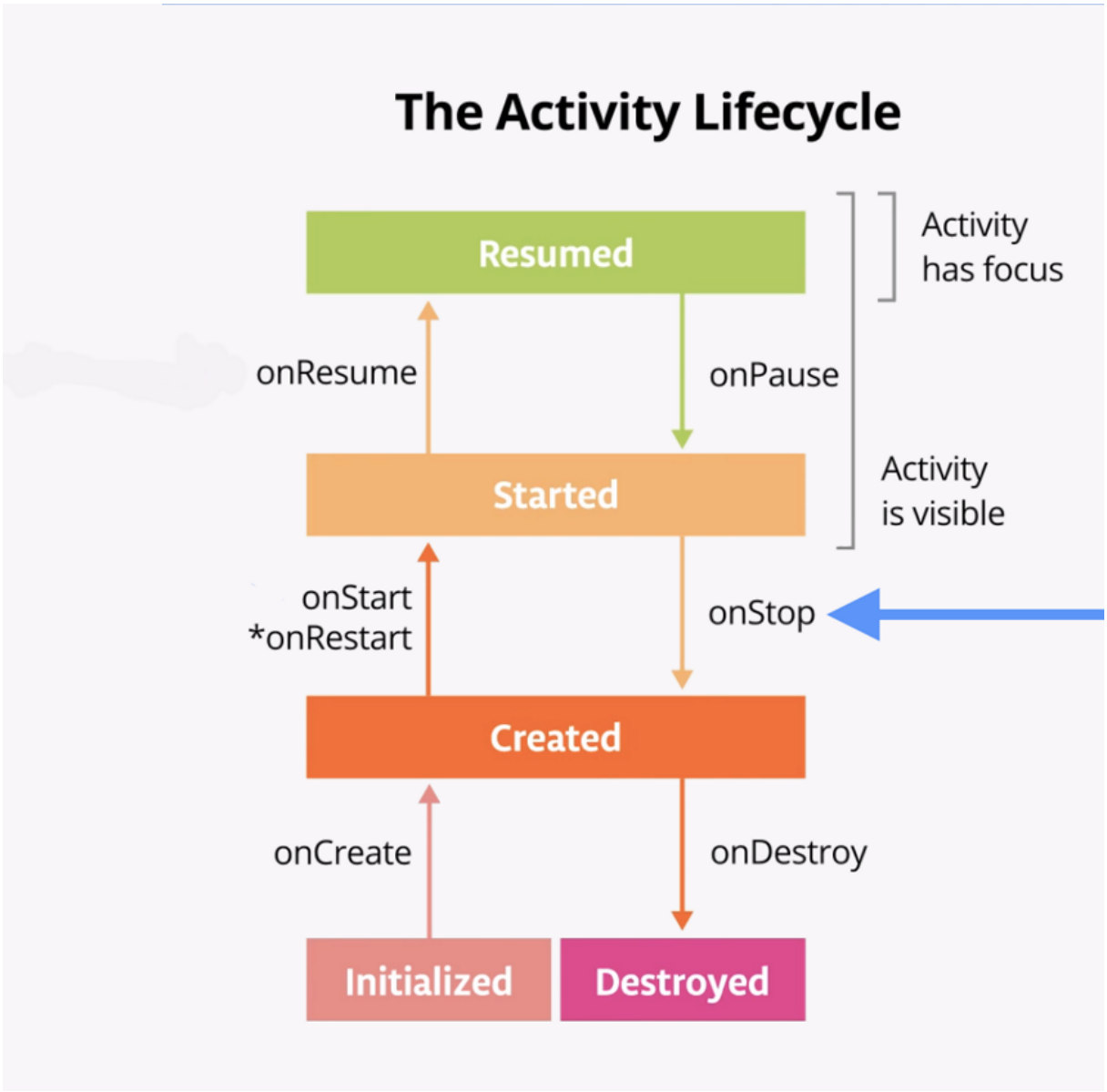
Когда ваша деятельность больше не видна на экране, это состояние называется переводом деятельности в фоновый режим. Противоположный случай - когда активность находится на переднем плане, или на экране. Когда пользователь возвращается в ваше приложение, та же самая активность перезапускается и снова становится видимой. Эта часть жизненного цикла называется видимым жизненным циклом приложения. Когда ваше приложение находится в фоновом режиме, оно, как правило, не должно активно работать, 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, и приостановить все текущие операции. Затем вы перезапускаете эти операции, когда ваше приложение переходит на передний план.

В этом шаге мы рассмотрим жизненный цикл активности, когда приложение переходит в фоновый режим и снова возвращается на передний план.

- Запустив приложение `Dessert Clicker` , несколько раз нажмите на кекс.
- Нажмите кнопку Home на своем устройстве и посмотрите на Logcat в Android Studio. Возврат на главный экран переводит приложение в фоновый режим, а не закрывает его совсем. Обратите внимание, что вызываются методы onPause() и onStop().

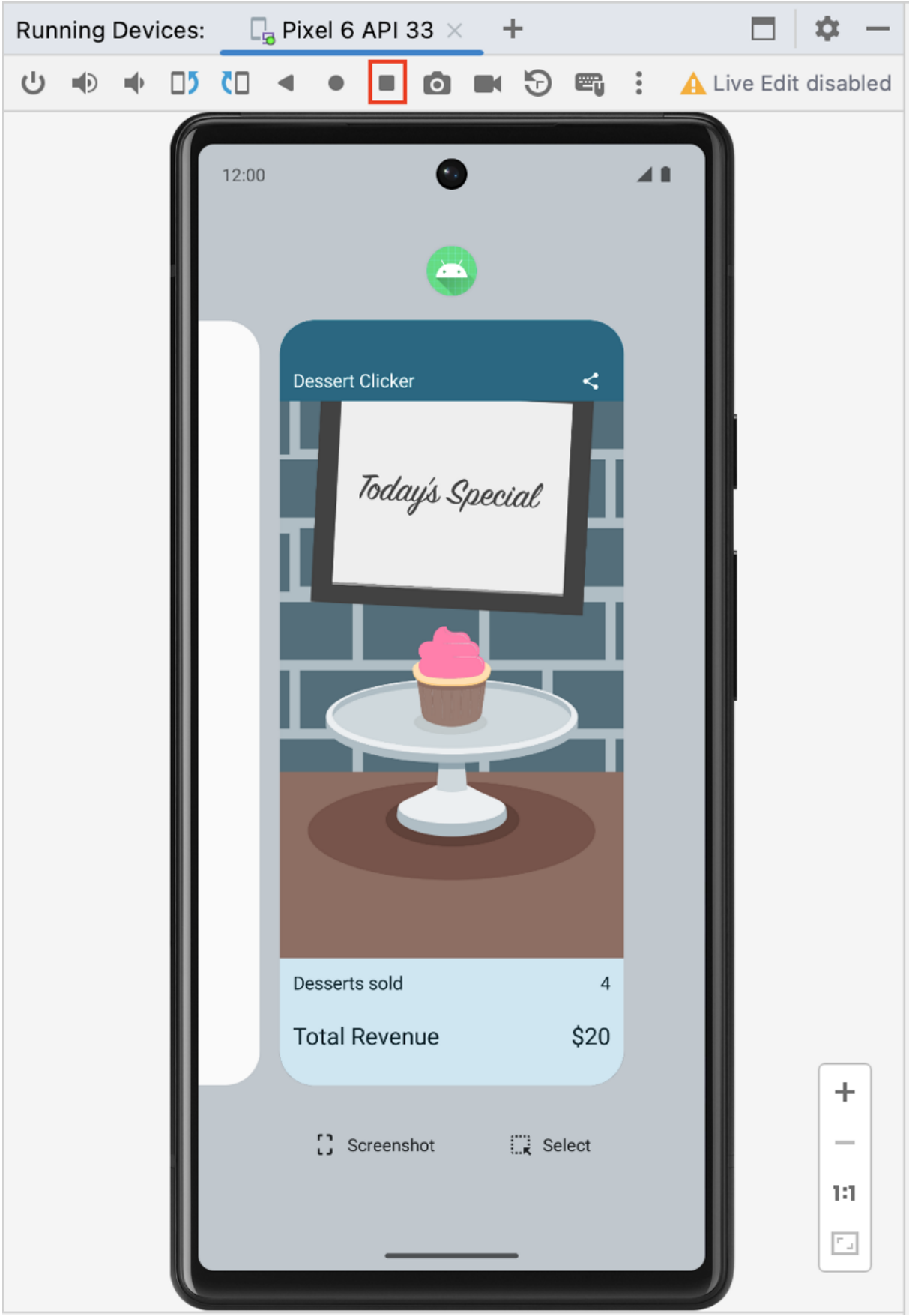
```
2024-04-26 15:00:04.905  5590-5590  MainActivity            com.example.dessertclicker          D  onPause Called
2024-04-26 15:00:05.430  5590-5590  MainActivity            com.example.dessertclicker          D  onStop Called
```

Когда вызывается onPause(), приложение больше не имеет фокуса. После onStop() приложение больше не видно на экране. Хотя активность остановлена, объект Activity все еще находится в памяти в фоновом режиме. ОС Android не уничтожила активность. Пользователь может вернуться к приложению, поэтому Android сохраняет ресурсы активности.



Чтобы вернуться к приложению, используйте экран `Recents`. На эмуляторе экран `Recents` можно открыть с помощью квадратной системной кнопки, показанной на изображении ниже.

Заметьте в Logcat, что активность перезапускается с помощью onRestart() и onStart(), а затем возобновляется с помощью onResume().



> Примечание: `onRestart()` вызывается системой только в том случае, если активность уже была создана и в конечном итоге переходит в состояние Created, когда вызывается onStop(), но возвращается обратно в состояние Started вместо того, чтобы перейти в состояние Destroyed. Метод onRestart() - это место для размещения кода, который нужно вызывать только в том случае, если активность запускается не 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
```

Когда активность возвращается на передний план, метод onCreate() больше не вызывается. Объект активности не был уничтожен, поэтому его не нужно создавать заново. Вместо метода onCreate() вызывается метод onRestart(). Обратите внимание, что на этот раз, когда активность возвращается на передний план, номер проданных Десертов сохраняется.

- Запустите хотя бы одно приложение, кроме `Dessert Clicker` , чтобы на устройстве было несколько приложений в экране Recents. Вызовите экран Recents и откройте еще одну недавнюю активность. Затем вернитесь к недавним приложениям и снова выведите `Dessert Clicker` на передний план.

Обратите внимание, что в Logcat вы видите те же обратные вызовы, что и при нажатии кнопки Home. onPause() и onStop() вызываются, когда приложение уходит в фон, а затем onRestart(), onStart() и onResume() вызываются, когда оно возвращается.

> Примечание: onStart() и onStop() вызываются несколько раз по мере того, как пользователь переходит к деятельности и выходит из нее.

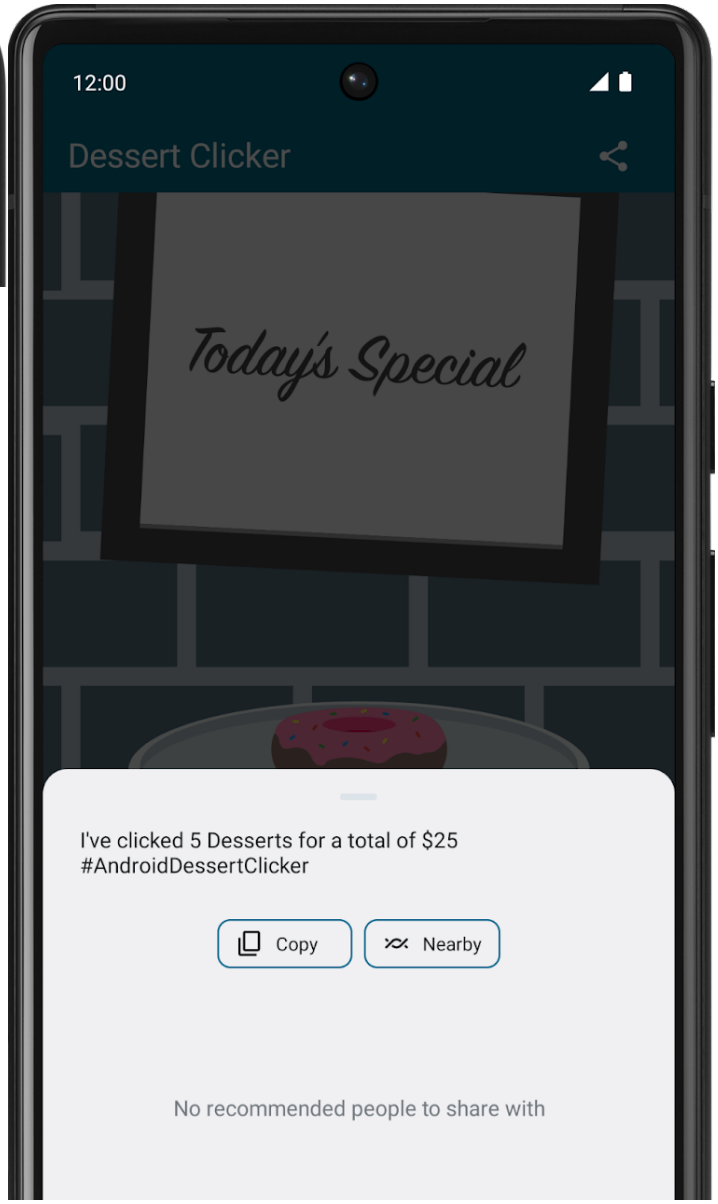
Эти методы вызываются, когда приложение останавливается и переходит в фоновый режим или когда приложение перезапускается и возвращается на передний план. Если вам нужно выполнить какую-то работу в вашем приложении во время этих случаев, переопределите соответствующий метод обратного вызова жизненного цикла.

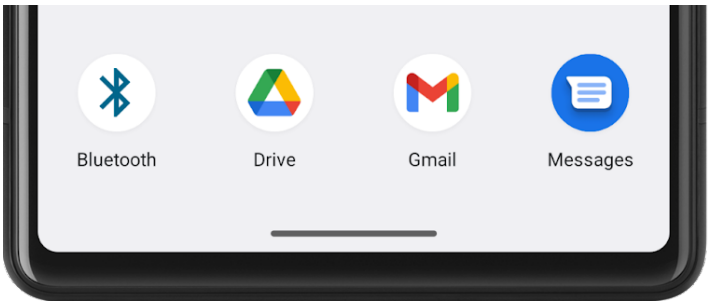
> Случай использования 3: Частичное скрытие активности Вы узнали, что когда приложение запускается и вызывается onStart(), оно становится видимым на экране. Когда вызывается onResume(), приложение получает фокус пользователя, то есть пользователь может взаимодействовать с приложением. Часть жизненного цикла, в которой приложение полностью находится на экране и имеет фокус пользователя, называется временем переднего плана.

Когда приложение переходит в фоновый режим, фокус теряется после `onPause()`, а после onStop() приложение перестает быть видимым.

Разница между фокусом и видимостью очень важна. Активность может быть частично видна на экране, но не иметь фокуса пользователя. В этом шаге вы рассмотрите один случай, когда активность частично видна, но не имеет фокуса пользователя.

- Запустив приложение `Dessert Clicker` , нажмите кнопку Share в правом верхнем углу экрана. Активность обмена появляется в нижней половине экрана, но активность по-прежнему видна в верхней половине.





- Изучите Logcat и обратите внимание, что была вызвана только функция onPause().

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

В этом случае функция onStop() не вызывается, потому что активность все еще частично видна. Но активность не имеет фокуса пользователя, и пользователь не может взаимодействовать с ней - фокус пользователя имеет активность «share», находящаяся на переднем плане.

Почему эта разница важна? Прерывание с помощью только `onPause()` обычно длится короткое время, после чего пользователь возвращается к своей деятельности или переходит к другой деятельности или приложению. Как правило, вы хотите продолжать обновлять пользовательский интерфейс, чтобы остальное приложение не казалось застывшим.

Какой бы код ни выполнялся в `onPause()`, он блокирует отображение других объектов, поэтому код в onPause() должен быть легким. Например, если поступает телефонный звонок, код в onPause() может отложить уведомление о входящем звонке.

- Щелкните за пределами диалогового окна общего доступа, чтобы вернуться в приложение, и обратите внимание, что вызывается onResume(). И onResume(), и onPause() имеют отношение к фокусу. Метод onResume() вызывается, когда активность получает фокус, а onPause() - когда активность теряет фокус.

Изучение изменений конфигурации

Есть еще один случай в управлении жизненным циклом активности, который важно понять: как изменения конфигурации влияют на жизненный цикл ваших активностей.

Изменение конфигурации происходит, когда состояние устройства меняется настолько радикально, что самый простой способ для системы устранить изменения - полностью выключить и перестроить деятельность. Например, если пользователь меняет язык устройства, может потребоваться изменить весь макет, чтобы учесть различные направления текста и длину строк. Если пользователь подключает устройство к док-станции или добавляет физическую клавиатуру, раскладке приложения может потребоваться использовать преимущества другого размера или расположения дисплея. А если изменится ориентация устройства - если его повернут из книжной в альбомную ориентацию или обратно, - макет может потребоваться изменить, чтобы он соответствовал новой ориентации. Давайте посмотрим, как приложение ведет себя в этом случае.

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

Изменение конфигурации вызывает вызов `onDestroy()`

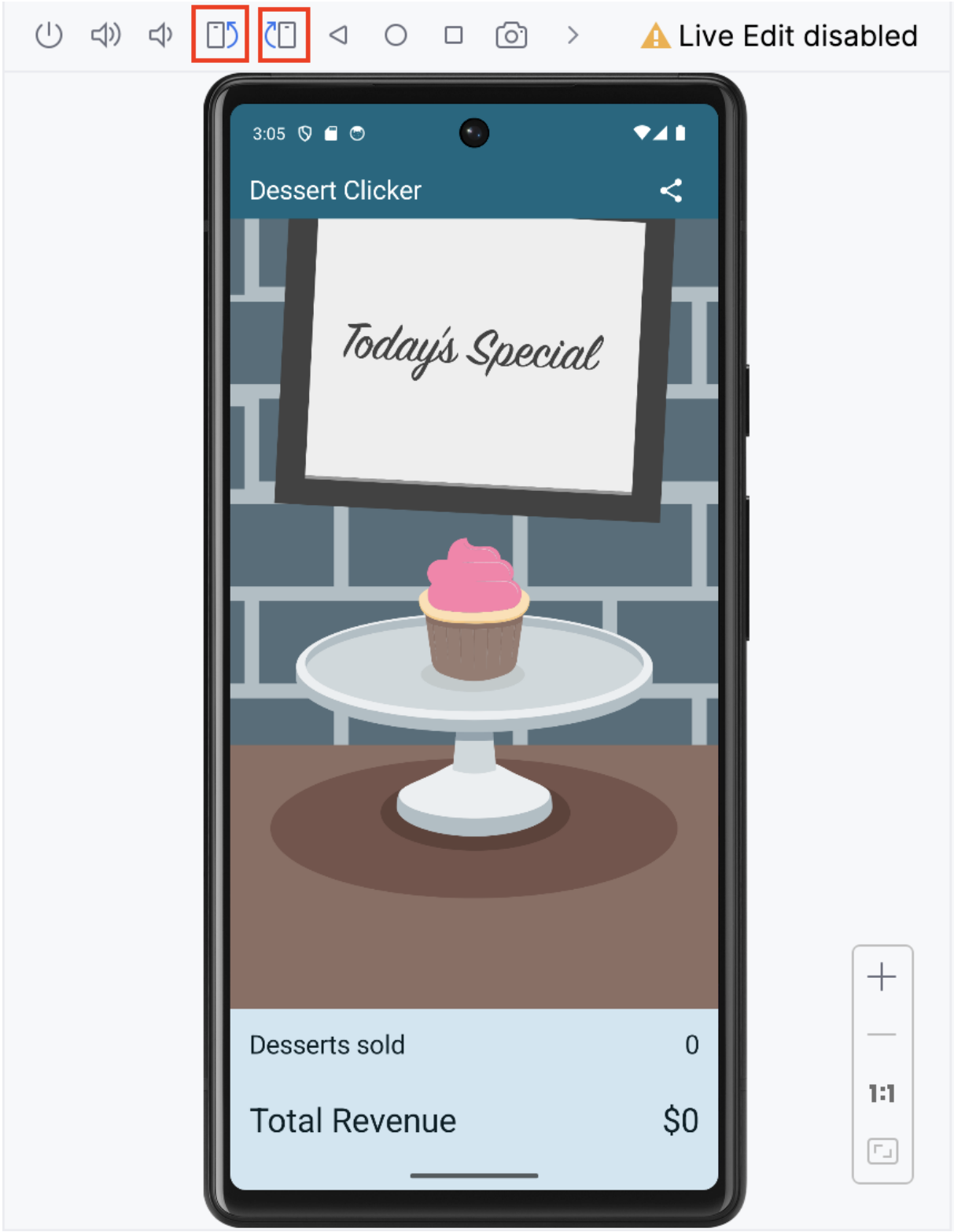
Поворот экрана - это один из типов изменения конфигурации, которое приводит к выключению и перезапуску активности. Чтобы смоделировать это изменение конфигурации и изучить его последствия, выполните следующие действия:

- Скомпилируйте и запустите свое приложение. Убедитесь, что блокировка поворота экрана в эмуляторе отключена.
- Поверните устройство или эмулятор в ландшафтный режим. Вы можете повернуть эмулятор влево или вправо с помощью кнопок поворота.
- Изучите Logcat и поймите, что при завершении работы активность вызывает onPause(), onStop() и onDestroy(), именно в таком порядке.

```
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 ротации устройства

- Скомпилируйте и запустите свое приложение и откройте Logcat.
- Несколько раз нажмите на кекс и обратите внимание, что количество проданных десертов и общая выручка не равны нулю.
- Убедитесь, что блокировка поворота экрана в эмуляторе отключена.
- Поверните устройство или эмулятор в ландшафтный режим. Вы можете повернуть эмулятор влево или вправо с помощью кнопок поворота.



{style=«width:500px»}

- Изучите вывод 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, что когда устройство или эмулятор поворачивает экран, система вызывает все обратные вызовы жизненного цикла, чтобы завершить активность. Затем, когда активность создается заново, система вызывает все обратные вызовы жизненного цикла, чтобы запустить активность.

Когда устройство поворачивается, а активность закрывается и создается заново, активность снова запускается со значениями по умолчанию - изображение десерта, количество проданных десертов и общий доход обнуляются.

Чтобы узнать, почему эти значения сбрасываются и как их исправить, необходимо изучить жизненный цикл композитного объекта и то, как он умеет наблюдать и сохранять свое состояние.

Жизненный цикл композита

Изначально пользовательский интерфейс вашего приложения строится из запускаемых композитных функций в процессе, который называется **Composition**.

Когда состояние вашего приложения изменяется, планируется перекомпозиция. **Рекомпозиция** - это когда Compose повторно выполняет композитные функции, состояние которых могло измениться, и создает обновленный пользовательский интерфейс. Композиция обновляется, чтобы отразить эти изменения.

> Единственный способ создать или обновить композицию - это ее начальная композиция и последующие рекомпозиции.

> У составных функций есть свой собственный жизненный цикл, который не зависит от жизненного цикла Activity.

Ее жизненный цикл состоит из событий: `вход в Композицию`, `повторная композиция 0 или более раз`, а затем `выход из Композиции`.

Для того чтобы Compose мог отследить и вызвать рекомпозицию, ему необходимо знать, когда состояние изменилось. Чтобы указать Compose, что она должна отслеживать состояние объекта, объект должен быть типа `State` или `MutableState`. Тип State является неизменяемым и может быть только прочитан. Тип MutableState является изменяемым и допускает чтение и запись.

Вы уже видели и использовали `MutableState` в приложении `Lemonade` и приложении `Tip Time` на предыдущих работах.

Чтобы создать мутабельную переменную revenue, вы объявляете ее с помощью mutableStateOf. Начальное значение по умолчанию - 0.

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

> Хотя этого достаточно, чтобы Compose запускал перекомпозицию при изменении значения дохода, этого недостаточно, чтобы сохранить обновленное значение. При каждом повторном выполнении Compose значение выручки будет заново инициализировано до первоначального значения по умолчанию 0.

Чтобы указать Compose сохранять и повторно использовать его значение во время рекомпозиций, вам нужно объявить его с помощью API `remember`.

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

Если значение revenue меняется, Compose планирует все композитные функции, которые считывают это значение, для рекомпозиции.

Хотя Compose запоминает состояние дохода при перекомпоновке, он не сохраняет это состояние при изменении конфигурации. Чтобы Compose сохраняла состояние при изменении конфигурации, необходимо использовать **rememberSaveable**.

Для получения дополнительной практики и информации обратитесь к лекции по состояниям Intro to state in Compose.

Использование функции `rememberSaveable` для сохранения значений при изменении конфигурации Вы используете функцию `rememberSaveable` для сохранения значений, которые понадобятся вам, если Android OS уничтожит и воссоздаст активность.

Чтобы сохранить значения при перекомпоновке, необходимо использовать функцию `remember`. Используйте функцию `rememberSaveable` для сохранения значений во время перекомпоновки и изменения конфигурации.

> Примечание: Иногда Android выключает весь процесс приложения, который включает все связанные с ним действия. Android выполняет такое завершение работы, когда система находится в напряжении и ей грозит визуальное отставание, поэтому в этот момент не запускаются дополнительные обратные вызовы или код. Процесс вашего приложения просто тихо завершается в фоновом режиме. Но для пользователя это не выглядит так, будто приложение закрыто. Когда пользователь возвращается к приложению, которое было закрыто системой Android, Android перезапускает это приложение. Вы хотите убедиться, что пользователь не потеряет данные, когда это произойдет.

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

В MainActivity обновите группу из пяти переменных, которые в данный момент используют `remember`, на `rememberSaveable`.

```
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)
}
```

В MainActivity обновите группу из пяти переменных, которые в данный момент используют remember, на rememberSaveable.

Резюме

- **Жизненный цикл активности** Жизненный цикл активности - это набор состояний, через которые проходит активность. Жизненный цикл активности начинается, когда ОС Android впервые создает активность, и заканчивается, когда ОС уничтожает активность. По мере того как пользователь перемещается между активностями, внутри и вне вашего приложения, каждая активность переходит из одного состояния в другое в жизненном цикле активности.
- Каждое состояние жизненного цикла активности имеет соответствующий метод обратного вызова, который вы можете переопределить в своем классе Activity.
- Основной набор методов жизненного цикла: `onCreate()`, `onRestart()`, `onStart()`, `onResume()`, `onPause()`, `onStop()`, `onDestroy()`.
- Чтобы добавить поведение, которое происходит при переходе вашей активности в состояние жизненного цикла, переопределите метод обратного вызова состояния.
- Чтобы добавить скелетные переопределяемые методы в свои классы в Android Studio, выберите Code > - Override Methods... или нажмите Control+O.
- **Ведение журнала с помощью Log** API протоколирования Android, а именно класс Log, позволяет писать короткие сообщения, которые отображаются в Logcat в Android Studio. Чтобы написать отладочное сообщение, используйте `Log.d()`. 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 в альбомный режим или из альбомного в книжный. 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 активность с нуля, выполняя все обратные вызовы запуска жизненного цикла. Когда Android завершает работу приложения из-за изменения конфигурации, он перезапускает активность с помощью функции onCreate().
- Чтобы сохранить значение, которое должно пережить изменение конфигурации, объявите его переменные с помощью `rememberSaveable`.