Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерный наук и технологий

Высшая школа интеллектуальных систем и суперкомпьютерных технологий.

**Отчёт по лабораторной работе №2**

**Дисциплина**: Проектирование мобильных приложений

**Тема**: Activity lifecycle. Alternative resources. Вариант 13

Выполнил студент гр. 3530901/80201: ­­­­­­­­­­­­­ ­ В.Д. Петров

(подпись)

Преподаватель: ­А.Н. Кузнецов

(подпись)

Санкт-Петербург

2020

**СОДЕРЖАНИЕ**

Введение…………………………………………………………………………….3

Задача №1. Activity…………………………………………………………………4

Задача №2. Alternative resources………………………………………………….11

Задача №3. Best-matching resource…………………………………………….....13

Задача №4 Сохранение состояния Activity………………………………………15

Заключение……………………………………………………………………..…24

Список использованных источников…………………………………………….25

**ВВЕДЕНИЕ**

Когда пользователь использует приложение, экземпляры Activity переходят из одного состояния в другое в их жизненном цикле. Класс Activity предоставляет определенное число обратных вызовов, которые позволяют экземпляру Activity узнать, что его состояние изменилось: он был создан, остановлен или возобновлен системой, либо же был уничтожен. С помощью методов жизненного цикла можно определить, как будет вести себя Activity, если пользователь из него выходит или заходит обратно. Например, можно остановить использование сети интернет, сохранить данные, которые не должны быть утеряны, либо же обезопасить приложение от крашей, когда пользователь выходит из него или когда поворачивает экран.

Приложения используют различные ресурсы, такие как: изображения, layout ресурсы, строки в пользовательском интерфейсе, анимации и другое. Для различных конфигураций устройств необходимы различные ресурсы, например для двух разных языков, английского и русского, будут необходимы строки интерфейса на соответствующих языках, чтобы пользователи могли понять, как пользоваться приложением. Во время исполнения приложения Android определяет наиболее подходящие ресурсы, основываясь на конфигурации устройства, на котором было запущено приложение. Алгоритм для определения наиболее подходящего ресурса называется best-match.

Целью данной работы является ознакомление с жизненным циклом Activity, а также изучение основных возможностей и особенностей альтернативных ресурсов.

**ЗАДАЧА №1. ACTIVITY**

Жизненный цикл Activity можно описать следующей схемой (Рис.1)

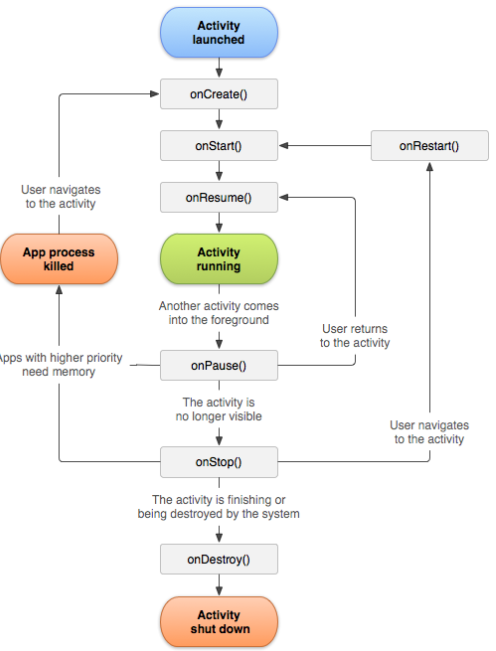


Рис. 1 Жизненный цикл Activity

Для навигации между состояниями жизненного цикла класс Activity предоставляет следующие обратные вызововы: onCreate(), onStart(), onResume(), onPause(), onStop(), onRestart() и onDestroy().

onCreate() – вызывается, когда система создает или перезапускает Activity. После создания Activity переходит в состояние Created. В данном методе нужно описать логику Activity, которая должна быть создана только однажды за всю его жизнь. Входным параметром является savedInstanceState, тип Bundle. При создании этот параметр равен Null, однако при перезапуске Activity после убийства процесса системой для освобождения оперативной памяти данный параметр содержит информацию о предыдущем сохраненном состоянии Activity. Когда данный метод завершается, Activity переходит в состояние Started

onStart() – вызывается, когда Activity находится в состоянии Started. Данный метод подготавливает Activity для выведения на экран устройства. В данном методе можно инициализировать код для поддержания пользовательского интерфейса. После завершения данного метода Activity переходит в состояние Resumed и выводится на экран.

onResume() – вызывается, когда Activity находится в состоянии Resumed. В этом состоянии Activity взаимодействует с пользователем и остается в данном состоянии до тех пор, пока что-либо не уберет его из фокуса, например звонок, переход к другой Activity, выход из приложения или выключение экрана устройства. В данном методе нужно инициализировать любую функциональность, которая должна работать, пока Activity видимо для пользователя. Когда Activity теряет фокус, оно переходит в состояние Paused. Однако если пользователь вернется в Activity из состояния Paused, то будет снова вызван метод onResume().

onPause() – вызывается, когда система понимает, что Activity не находится на экране. В данном методе нужно остановить или отрегулировать операции, которые не должны выполняться в состоянии Paused, и вы ожидаете, что Activity скоро будет возобновлено. Также необходимо освободить системные ресурсы и все ресурсы, влияющие на время работы батареи. Однако в данном состоянии приложение может быть полностью видимым в режиме нескольких окон, поэтому желательно использовать метод onStop() для освобождения ресурсов связанных с пользовательских интерфейсом. Данный метод протекает очень быстро, поэтому нельзя в нем сохранять данные, передавать их с помощью интернета и т.д. Окончание метода onPause() не означает, что Activity вышло из состояния Paused, оно продолжает находиться в этом состоянии до тех пор, пока не возобновит свою работу, либо пока не станет полностью невидимым для пользователя.

onStop() – когда Activity становится полностью невидимым для пользователя, оно переходит в состояние Stopped. Это может случиться, например, когда новое Activity полностью появляется на экране. Система также может вызвать этот метод, когда Activity завершила свою работу и скоро будет уничтожено. В данном методе необходимо освободить все ресурсы, которые не нужны, пока пользователь не видит приложение. Также, если вы не можете найти более подходящей возможности, в данном методе можно сохранять данные в базы данных. Когда приложение находится в состоянии Stopped, оно продолжает находиться в оперативной памяти устройства, однако системе может понадобиться эта память, тогда она сохраняет состояние Activity, освобождает память и восстанавливает его вызовом метода onCreate(), когда пользователь снова начнет пользоваться им. Если оно возобновляет работу без освобождения памяти, то вызывается метод onResumed(), а если оно закончило работу, то вызывается метод onDestroy().

onRestart() – вызывается, когда Activity возобновляет работу после вызова метода onStop().

onDestroy() – вызывается, когда Activity уничтожена. Система вызывает данный метод в двух случаях: либо когда приложение завершило свою работу, либо когда необходимо временно уничтожить Activity из-за смены конфигурации, например из-за смены ориентации экрана. В данном методе нужно освободить все ресурсы, которые не были освобождены в предыдущих методах.

Исходный код для данного задания представлен по ссылке:

<https://github.com/ADsty/labs-android_labs/tree/master/lab2>

В данном задании необходимо продемонстрировать жизненный цикл приложения на любом нетривиальном примере. Для начала запустим наше приложение (Рис. 2)

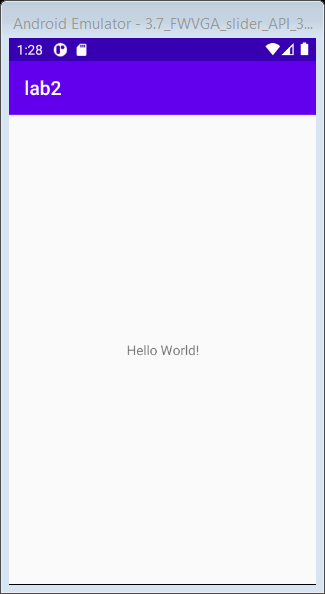


Рис. 2 Запуск приложения

При запуске были вызваны методы onCreate(), onStart() и onResume(). Затем отправим сообщение на телефон (Рис. 3)

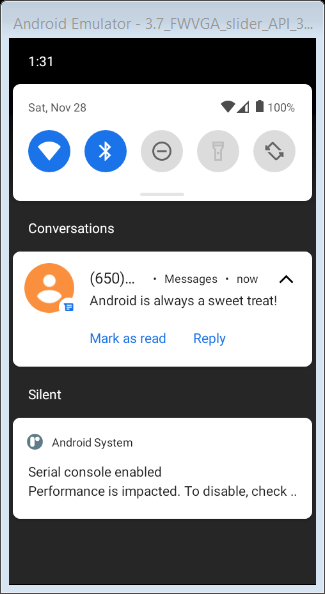


Рис. 3 Экран с сообщением

При переходе на экран уведомления часть Activity все еще остается на экране, поэтому никаких методов не вызывается. Перейдем в другое приложение для того, чтобы ответить на сообщение (Рис. 4)

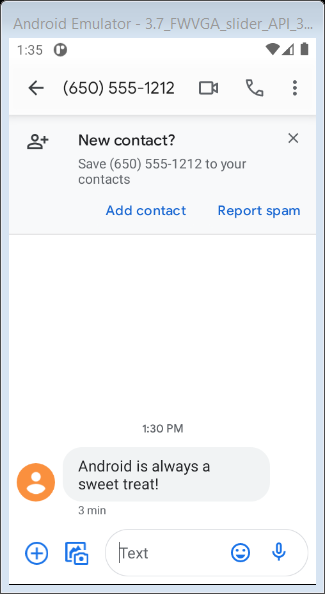


Рис. 4 Экран мессенджера

В данном случае были вызваны методы onPause() и onStop(), т.к Activity потеряло фокус и стало невидимым для пользователя. Вернемся в наше приложение (Рис. 5)

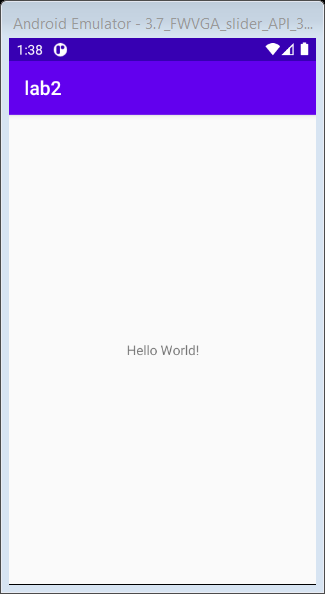


Рис. 5 Экран приложения после возвращения в него

Когда мы возобновили работу приложения после вызова метода onStop() были вызваны методы onRestart(), onStart() и onResume(). Теперь выйдем из приложения (Рис. 6)

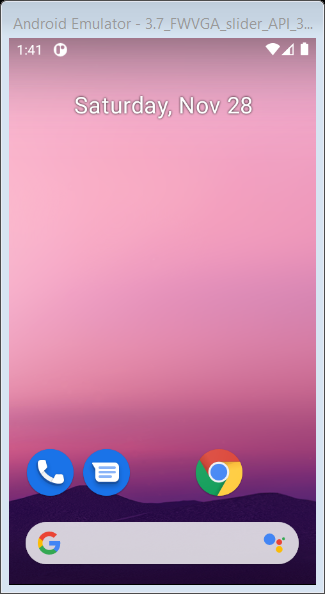


Рис.6 Экран после выхода из приложения

После завершения работы приложения были вызваны методы onPause(), onStop() и onDestroy.

**ЗАДАЧА №2. ALTERNATIVE RESOURCES**

Альтернативные ресурсы предоставляют возможность приложению корректно работать на различных конфигурациях устройств, таких как: разные разрешения экрана, разные языки системы, разные формы экрана, экраны с поддержкой HDR и многое другое. Все ресурсы приложения необходимо помещать в специально отведенные подкаталоги в каталоге res, например:

1. animator – подкаталог для анимации с возможностью изменения свойств объекта (неважно View или не View)
2. anim – подкаталог для анимации объекта класса View, анимации из каталога animator также могут быть сохранены тут, однако прошлый каталог предпочтительней для отделения этих типов анимации
3. color – подкаталог для XML файлов, которые определяют список состояний цвета для объектов в приложении (например для кнопки, чтобы при нажатии она меняла свой цвет)
4. drawable – подкаталог для растровых изображений (.png, .jpg, .gif)
5. mipmap – подкаталог для различных разрешений иконки приложения
6. layout – подкаталог для layout ресурсов приложения
7. menu – подкаталог для XML файлов меню приложения (Опции и др.)
8. raw – подкаталог для различных файлов, которые надо сохранить в необработанном виде.
9. values – подкаталог для XML файлов, которые содержат простые строки, числа или цвета
10. xml – подкаталог для различных XML файлов, например файлов конфигурации
11. font – подкаталог для шрифтов с расширениями .ttf, .otf, .ttc или XML файлов, которые включают <font-family> элемент.

Ресурсы, сохраненные в данных каталогах, будут ресурсами по умолчанию в данном приложении. Однако для разных конфигураций устройств потребуются альтернативные ресурсы. Чтобы задать ресурсы для определенной конфигурации необходимо в каталоге res создать подкаталог со следующим названием: <resources\_name>-<qualifier>, где resources\_name это названия подкаталогов, описанных выше, а qualifier это название, которое определяет нужную конфигурацию устройства, можно указывать несколько названий через тире, для этого необходимо будет указывать их в порядке приоритета. После создания этого подкаталога нужно будет поместить альтернативные ресурсы для заданной конфигурации туда, обязательным условием будет полное совпадение имен ресурсов с их аналогами в подкаталоге ресурсов по умолчанию. При добавлении альтернативного ресурса необходимо будет, чтобы в приложении присутствовал и ресурс по умолчанию, иначе могут возникнуть runtime exceptions. Нельзя указывать несколько квалификаторов одного типа к одному ресурсу, например нельзя создать подкаталог drawable-es-fr для испанского и французского языков, нужно создать два отдельных подкаталога drawable-es и drawable-fr. Квалификаторы для альтернативных ресурсов написаны ниже, в порядке приоритета:

1. MCC и MCN – код страны и код сети, полученные от SIM карты устройства
2. Язык и регион устройства
3. Направление layout ресурсов – справа налево или слева направо.
4. Наименьшая возможная ширина экрана
5. Наименьшая доступная ширина экрана
6. Наименьшая доступная длина экрана
7. Размер экрана
8. Соотношение сторон экрана
9. Форма экрана – круглый он или нет.
10. Наличие широкой цветовой гаммы
11. Наличие HDR
12. Ориентация экрана
13. Режим пользовательского интерфейса
14. “Ночной” режим
15. Плотность пикселей экрана
16. Тип тачскрина
17. Доступность экранной клавиатуры
18. Метод ввода текста – qwerty, 12key или же отсутствие ввода текста
19. Доступность кнопок навигации
20. Метод навигации – устройство имеет d-pad, не имеет никаких кнопок навигации кроме тачскрина и др.
21. Поддерживаемый API level

В данной задаче необходимо продемонстрировать пользу квалификатора доступности кнопок навигации. Данный квалификатор имеет два значения – navexposed когда кнопки доступны пользователю и navhidden когда кнопки не доступны. Данный квалификатор может использоваться для создания layout ресурсов с удобной навигацией по приложению в том случае, если на устройстве не доступны кнопки навигации (например, с помощью добавления в layout ресурс своих кнопок для возврата в прошлое состояние или начальный экран).

**ЗАДАЧА №3. BEST-MATCHING RESOURCE**

Для определения наиболее подходящего ресурса для данного устройства система Android использует алгоритм best-match:

1. Отсекаются все подкаталоги, противоречащие конфигурации устройства
2. Выбирается квалификатор с наивысшим приоритетом в списке (приоритет квалификаторов описан в предыдущем задании)
3. Использует ли какой-либо подкаталог этот квалификатор? Если нет, то алгоритм возвращается к пункту 2 и выбирает следующий квалификатор по списку. Если да, то алгоритм переходит к пункту 4
4. Отсекаются все подкаталоги, которые не содержат данный квалификатор
5. Алгоритм переходит к пункту 2 пока не останется один подкаталог

В данном задании необходимо показать на данном примере работу алгоритма best-match. Конфигурация устройства:

1. LOCALE\_LANG: fr
2. LOCALE\_REGION: rUS
3. SCREEN\_SIZE: small
4. SCREEN\_ASPECT: long
5. ROUND\_SCREEN: round
6. ORIENTATION: land
7. UI\_MODE: vrheadset
8. NIGHT\_MODE: notnight
9. PIXEL\_DENSITY: mdpi
10. TOUCH: finger
11. PRIMARY\_INPUT: 12key
12. NAV\_KEYS: wheel
13. PLATFORM\_VER: v25

Конфигурация ресурсов следующая:

1. (default)
2. fr-small-port-finger-nokeys-trackball-v25
3. notlong-port-night-finger-v27
4. long-vrheadset-12key
5. vrheadset-night-notouch-v27
6. fr
7. en-notlong-round-dpad
8. large-wheel-v26
9. fr-rFR-xlarge-notround-vrheadset-night-qwerty-dpad
10. rUS-notlong-round-car
11. notlong-round-port-car-nodpi-finger

Согласно алгоритму, на первом шаге необходимо убрать все подкаталоги, противоречащие конфигурации устройства. Подкаталог 7 противоречит конфигурации, т.к язык системы французский, а у данного ресурса указан английский. Подкаталог 9 также противоречит конфигурации, т.к регион устройства Россия, а в подкаталоге указана Франция. Подкаталог 8 не подходит: размер экрана large, хотя у устройства размер экрана small. Подкаталоги 3, 10 и 11 также не подойдут, т.к соотношение сторон у устройства long, а в данных подкаталогах – notlong. Подкаталог 2 противоречит ориентации устройства – port вместо land. Подкаталог 5 противоречит “дневному” режиму устройства – у него указан “ночной” режим. Таким образом после первого шага остаются следующие подкаталоги:

1. (default)
2. long-vrheadset-12key
3. fr

На втором шаге выбирается квалификатор с наивысшим приоритетом, в данном случае код страны и код сети интернет устройства, и поскольку ни один из каталогов не использует данный квалификатор, то выбирается следующий по списку квалификатор – язык и регион устройства. В данном случае подкаталог 3 использует данный квалификатор – у него указан язык устройства французский, поэтому переходим к следующему шагу и отсекаем все подкаталоги без этого квалификатора: первый и второй подкаталоги. Поскольку остался лишь один ресурс, то алгоритм заканчивает свою работу. Наиболее подходящим оказался подкаталог fr.

**ЗАДАЧА №4. СОХРАНЕНИЕ СОСТОЯНИЯ ACTIVITY**

В данной задаче необходимо найти ошибки в приложении и исправить их.

Исходный код для данного задания можно найти по ссылке:

<https://github.com/ADsty/labs-android_labs/tree/master/lab2_continuewatch>

Для начала запустим приложение (Рис. 7)

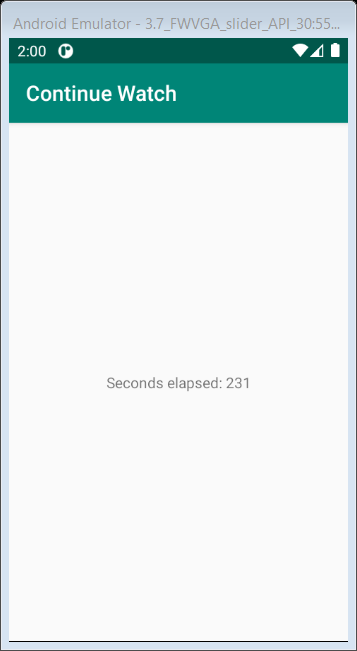
****

Рис. 7 Экран запущенного приложения

При повороте экрана счетчик сбрасывается (Рис. 8)

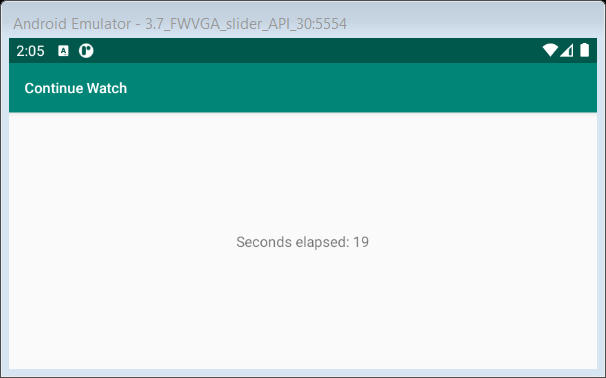


Рис. 8 Экран приложения при повороте

При сворачивании приложения счетчик продолжает свою работу (Рис. 9-11)

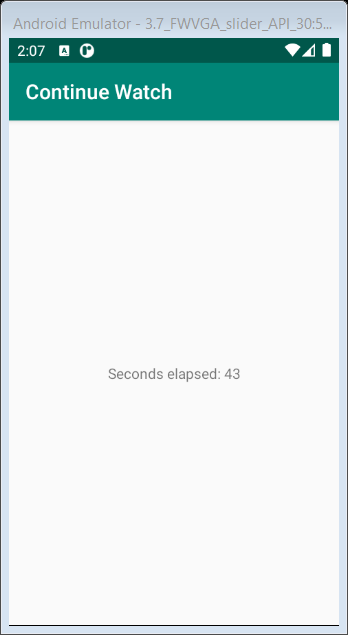


Рис. 9 Экран работающего приложения

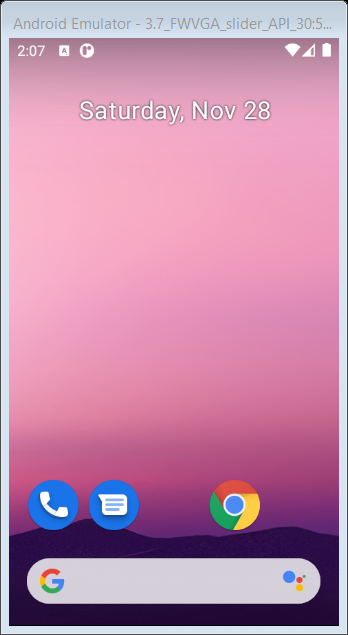


Рис. 10 Экран после сворачивания приложения

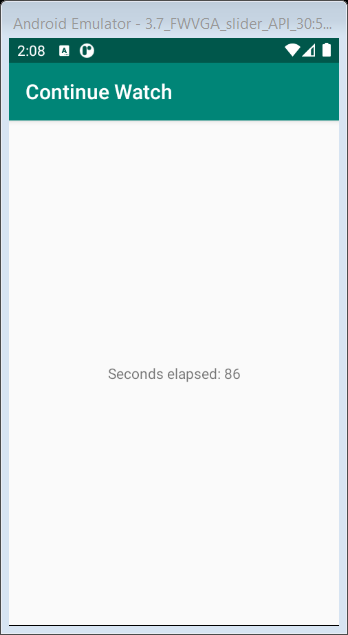


Рис. 11 Экран после возобновления работы приложения

Для решения данной задачи был использован экземпляр класса SharedPreferences, который сохранял данные о количестве секунд, проведенных в приложении, а также была использована дополнительная переменная типа Boolean. В методе onCreate (Листинг 1) получаем экземпляр класса SharedPreferences. В этом же методе устанавливается содержимое Activity, а именно TextView, в котором на данный момент написано “Seconds elapsed: 0”, данное изменение было внесено для того, чтобы в начальный небольшой промежуток времени у пользователя не мелькала надпись “Hello World”, которая не имеет никакого отношения к счетчику.

Листинг 1 – метод onCreate

override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_main*)  
 sharedPreferences = getSharedPreferences("seconds", Context.*MODE\_PRIVATE*)  
}

В методе onResume (Листинг 2) устанавливаем значение флага stopped равное false, после чего пытаемся получить сохраненное в SharedPreferences значение типа int, если нашли его – присваиваем его переменной secondsElapsed. После всех этих операций начинаем работу потока методом start.

Листинг 2 – метод onResume

override fun onResume() {  
 super.onResume()  
 stopped = false  
 if (sharedPreferences.contains("savedSeconds"))  
 secondsElapsed = sharedPreferences.getInt("savedSeconds", 0)  
 backgroundThread.start()  
}

В методе onPause (Листинг 3) присваиваем значение флага stopped равное true, после этого никакие новые обновления UI вызваны не будут, после работы метода Thread.sleep поток завершит свое выполнение. С помощью метода join дожидаемся окончания работы потока, после чего проверяем, что поток не активен (флаг isActive равен false). Таким образом мы гарантируем, что при повороте экрана, выходе из приложения и других ситуациях не будет одновременно работать более одного потока, связанного со счетчиком. В конце метода onPause сохраняем текущее значение переменной secondsElapsed в SharedPreferences.

Листинг 3 – метод onPause

override fun onPause() {  
 super.onPause()  
 stopped = true  
 backgroundThread.join()  
 if(!backgroundThread.*isAlive*){  
 *println*("Counter stopped")  
 }  
 sharedPreferences.edit().putInt("savedSeconds", secondsElapsed).apply()  
}

Внутри потока (Листинг 4) было изменено следующее: введено условие выхода, если переменная stopped будет равна true, а также проверка переменной stopped внутри цикла, чтобы в ситуации, где мы завершили работу activity раньше завершения метода Thread.sleep поток не смог обратиться к уже не существующему UI.

Листинг 4 – исходный код потока backgroundThread

var backgroundThread = Thread **{** while (!stopped) {  
 Thread.sleep(1000)  
 if(!stopped) {  
 textSecondsElapsed.post **{** textSecondsElapsed.*text* = "Seconds elapsed: " + secondsElapsed++  
 **}** }  
 }  
**}**

Таким образом были исправлены все найденные в приложении ошибки. При повороте экрана сохраняется значение секунд (Рис. 12-13).



Рис. 12 Количество секунд до поворота экрана

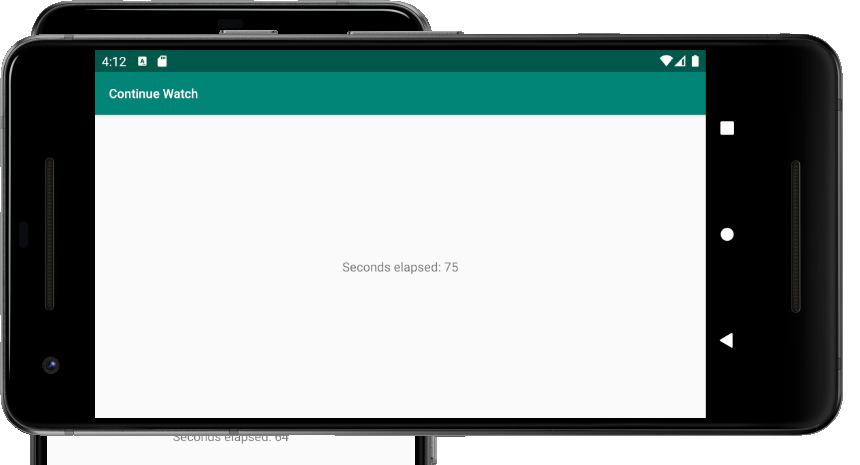


Рис. 13 Количество секунд после поворота

Как можно видеть, количество секунд более не обнуляется. Количество потоков, работающих с UI, не превышает 1. После поворота экрана работающий ранее поток переходит в состояние Dead и пропадает из профайлера, а на его месте появляется новый, который с небольшой задержкой начинает прерванную работу (Рис.14).

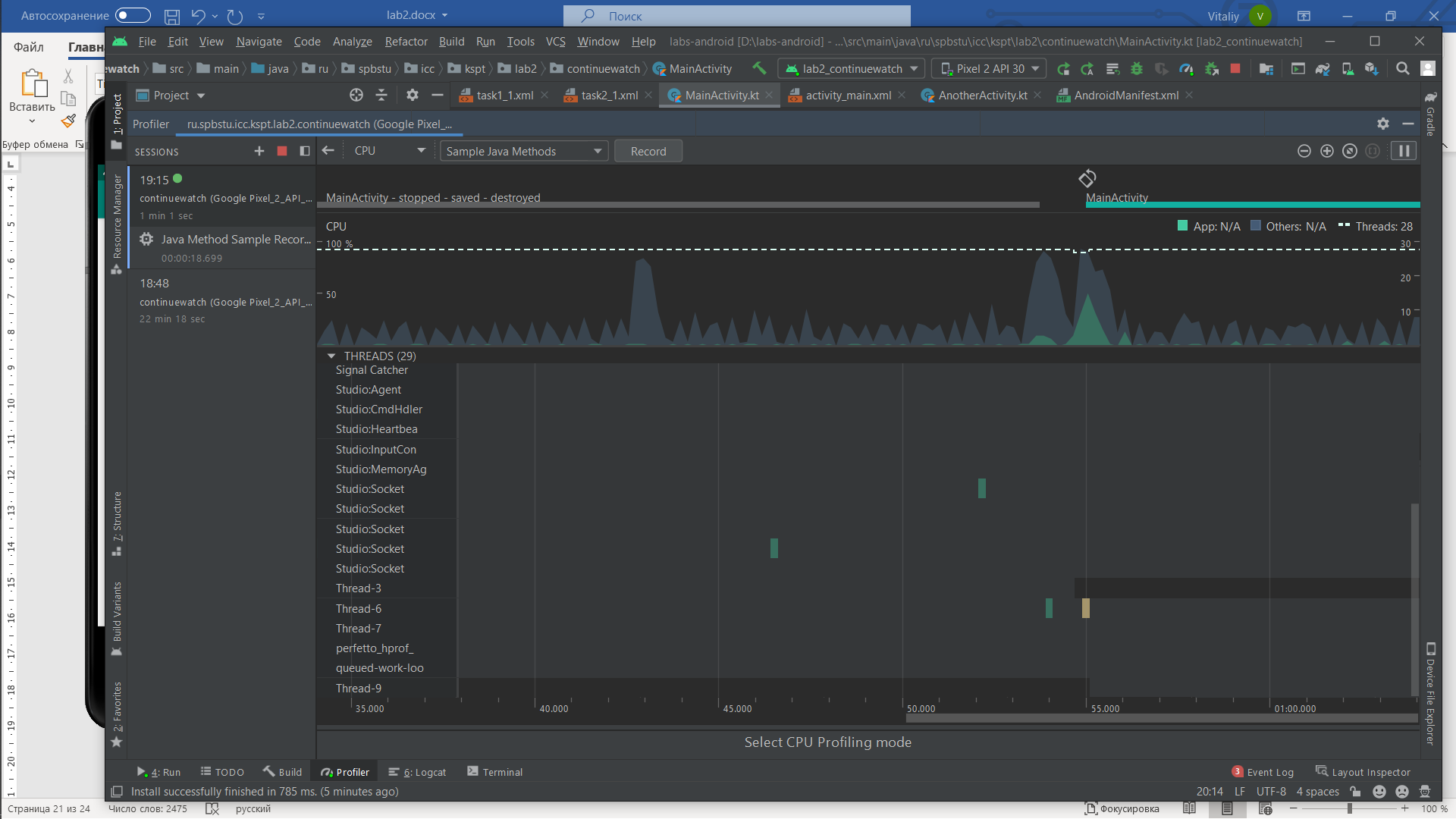


Рис. 14 Пример завершения работы старого потока и начала работы нового

В данном примере Thread 3 завершает свое исполнение – после того как метка дойдет до черной зоны, поток пропадет из профайлера. В то же самое время появляется новый поток Thread 9, который спустя небольшой промежуток после смерти Thread 3 начинает работу – это новый поток для работы счетчика, который и будет обновлять UI до своей остановки.

Альтернативным решением является использование методом onSaveInstanceState и onResumeInstanceState. Первый метод вызывается после метода onPause и способен сохранять переменные стандартных типов в объект класса Bundle. Второй метод вызывается после метода onStart и его задачей будет вытащить эти переменные из объекта Bundle и присвоить их текущим переменным. Данная реализация будет отличаться только заменой SharedPreferences на эти методы, соответственно для проверки будет использоваться только сценарий с сохранением состояния счетчика при повороте – в остальных ситуациях приложение будет вести себя аналогично предыдущему решению.

В методе onSaveInstanceState (Листинг 5) в объект outstate класса Bundle мы кладем значение переменной secondsElapsed по ключу count.

Листинг 5 – метод onSaveInstanceState

override fun onSaveInstanceState(outState: Bundle) {  
 super.onSaveInstanceState(outState)  
 outState.putInt("count", secondsElapsed)  
}

В методе onRestoreInstanceState (Листинг 6) из объекта savedInstanceState класса Bundle мы получаем сохраненное по ключу count значение и присваиваем его текущей переменной secondsElapsed

Листинг 6 – метод onRestoreInstanceState

override fun onRestoreInstanceState(savedInstanceState: Bundle) {  
 super.onRestoreInstanceState(savedInstanceState)  
 secondsElapsed = savedInstanceState.getInt("count")  
}

Проверим, что при повороте экрана значение счетчика сохраняется (Рис. 15-16).



Рис. 15 Количество секунд до поворота экрана

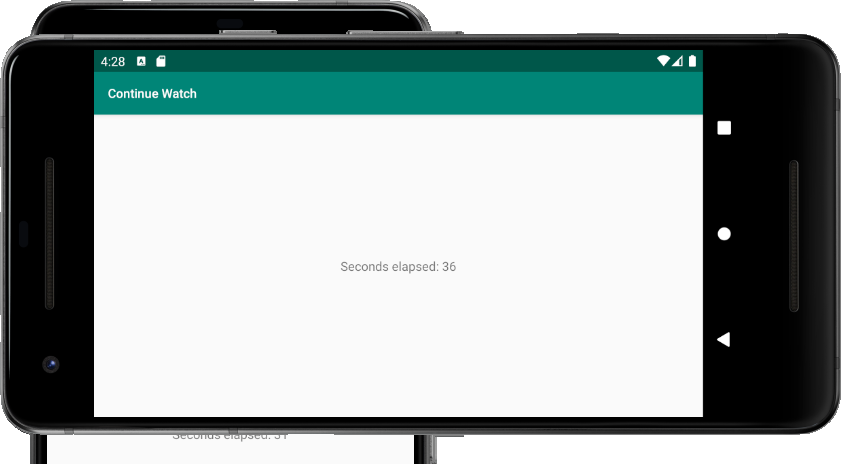


Рис. 16 Количество секунд после поворота

Таким образом, альтернативное решение так же исправляет все ошибки.

**ЗАКЛЮЧЕНИЕ**

В данной работе был изучен и продемонстрирован на примере жизненный цикл Activity, с помощью использования некоторых методов жизненного цикла удалось исправить ошибки в приложении из задания 4, были изучены возможности альтернативных ресурсов – они необходимы для корректной работы приложения на разных конфигурациях устройств. Также был изучен и продемонстрирован на примере алгоритм определения наилучшего ресурса для заданной конфигурации устройства best-match. Он отсекает все неподходящие подкаталоги ресурсов до тех пор, пока не останется лишь один нужный.

Исходный код для лабораторных работ находится на [github](https://github.com/ADsty/labs-android_labs) (<https://github.com/ADsty/labs-android_labs> )

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

<https://developer.android.com/reference/android/content/SharedPreferences>

[https://developer.android.com/guide/components/activities/activity-lifecycle](https://developer.android.com/guide/components/activities/activity-lifecycle#lc)

<https://developer.android.com/guide/topics/resources/providing-resources>

<https://classroom.udacity.com/courses/ud9012>