Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа интеллектуальных систем и суперкомпьютерных технологий

Отчёт по лабораторной работе № 2

Дисциплина: Проектирование мобильных приложений

Тема: Activity Lifecycle. Alternative resources.

Выполнил студент гр. 3530901/90201 _____З.А. Фрид
(подпись)

Принял преподаватель _____ А.Н. Кузнецов
(подпись)

“____” _____ 2021 г.

Санкт-Петербург

2021

Оглавление

Ссылка на проекты

https://github.com/zina-frid/AndroidLabs/tree/main/projects/lab2

Цели

- Познакомиться с жизненным циклом Activity
- Изучить основные возможности и свойства alternative resources

Задачи

Задача 1. Activity

Продемонстрируйте жизненный цикл Activity на любом нетривиальном примере.

Задача 2. Alternative Resources

Привести пример использования альтернативного ресурса (тип ресурса согласно варианту 23 это UI mode).

Задача 3. Best-matching resource

Для заданного набора альтернативных ресурсов, предоставляемых приложением, и заданной конфигурации устройства (оба параметра согласно варианту 23) объясните, какой ресурс будет выбран в конечном итоге. Объяснение должно включать пошаговое исполнение алгоритма best matching с описанием того, какие ресурсы на каком шаге отбрасываются из рассмотрения и почему.

Вариант 23:
Конфигурация устройства: LOCALE_LANG: fr LOCALE_REGION: rUS SCREEN_SIZE: small SCREEN_ASPECT: long ROUND_SCREEN: notround ORIENTATION: land UI_MODE: television NIGHT_MODE: notnight PIXEL_DENSITY: ldpi TOUCH: notouch

```
PRIMARY_INPUT: qwerty
NAV_KEYS: trackball
PLATFORM_VER: v27

Конфигурация ресурсов:
(default)
port-xxhdpi-finger-qwerty-dpad
small-land-notnight
night-nodpi
land
en-desk-notouch-qwerty
long-nodpi-nokeys-trackball
rUS-notouch-v26
rFR-notnight-notouch-v26
en-rFR-notlong-vrheadset-12key-v26
```

Задача 4. Сохранение состояние Activity.

Студент написал приложение: continuewatch. Это приложение по заданию должно считать, сколько секунд пользователь провел в этом приложении.

Задача 4.1. Поиск ошибок.

Найдите и опишите все ошибки в этом приложении, которые можете найти.

Задача 4.2. Сохранение состояние Activity (1).

Исправьте неправильный подсчет времени в приложении ContinueWatch с использованием onSaveInstanceState/onRestoreInstanceState.

Задача 4.3. Сохранение состояние Activity (2).

Исправьте неправильный подсчет времени в приложении ContinueWatch с использованием Activity Lifecycle callbacks и Shared Preferences.

Задача 4.4. Сравнение решений.

Продемонстрируйте, что приложения из 4.2 и 4.3 имеют разное поведение. Объясните поведение в каждом случае.

Жизненный цикл Activity

По мере того, как пользователь взаимодействует с приложением (выходит из него, возвращается, закрывает, открывает и т.д.), экземпляры Activity в приложении переходят из одних состояний их жизненного цикла в другие.

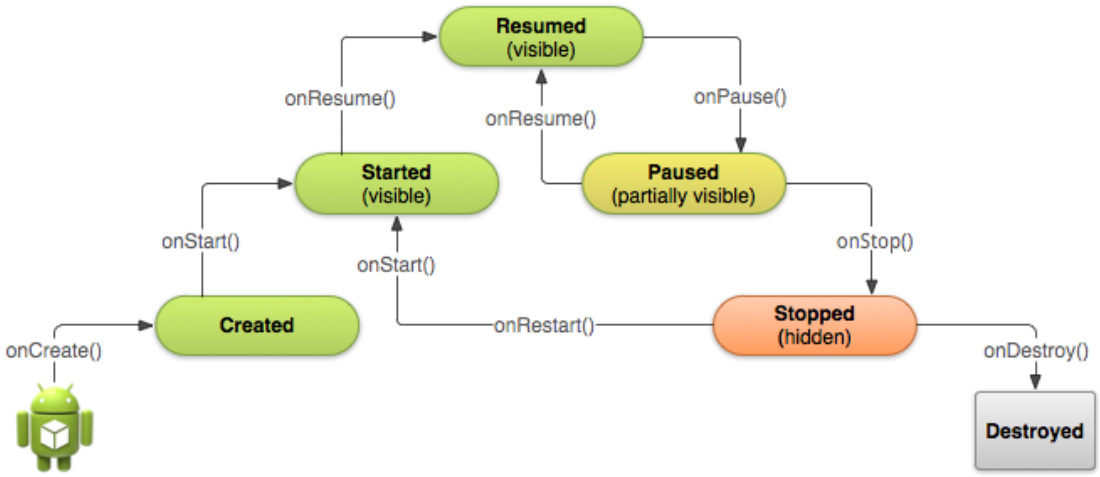

Рис. 1 Жизненный Цикл Activity (states)

Созданное при работе приложения Activity может быть в одном из трех состояний:

- *Resumed* - Activity видно на экране, оно находится в фокусе, пользователь может с ним взаимодействовать.
- *Paused* - Activity не в фокусе, пользователь не может с ним взаимодействовать, но его видно (оно перекрыто другим Activity, которое занимает не весь экран или полупрозрачно).
- *Stopped* - Activity не видно (полностью перекрывается другим Activity), соответственно оно не в фокусе и пользователь не может с ним взаимодействовать.

Класс Activity предоставляет шесть функций обратного вызова (callbacks), которые позволяют Activity знать, что состояние изменилось.

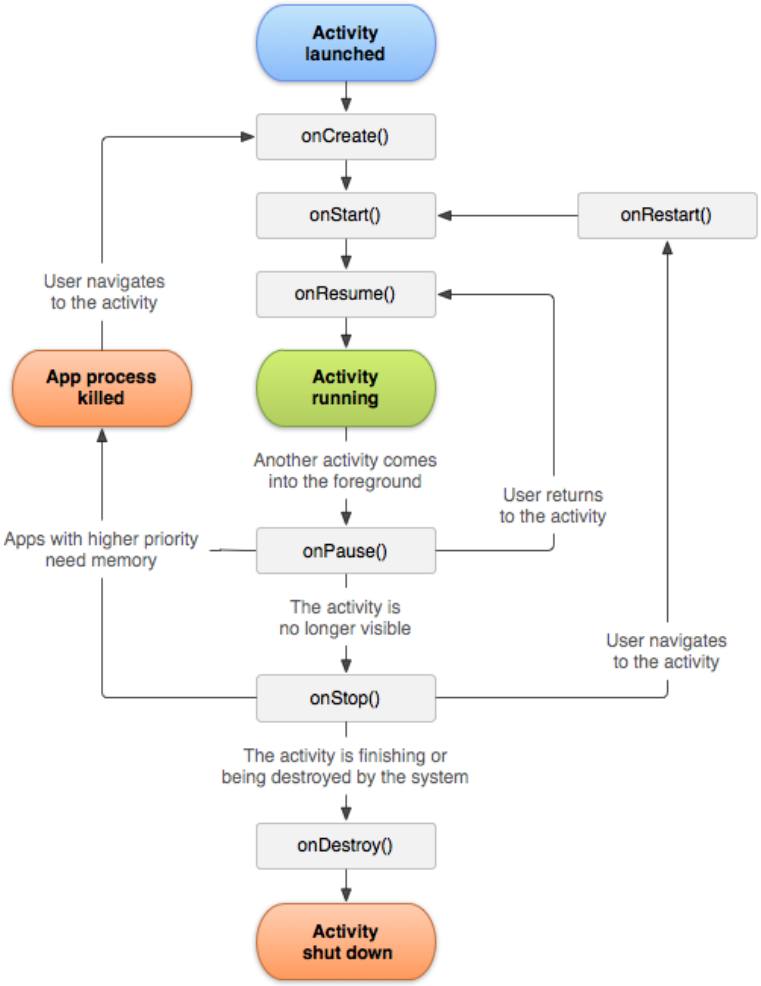

Рис. 2 Жизненный Цикл Activity (callbacks)

onCreate() – метод, с которого начинается выполнение Activity. Переводит Activity в состояние Created. Выполняется только один раз, поэтому внутри него следует выполнять такие действия как: первоначальная настройка Activity, восстановление предыдущего состояния через объект Bundle, инициализация данных для приложения и т. д. После выполнения переводит Activity в состояние Started.

onStart() – метод, вызываемый перед тем, как Activity будет видно пользователю. Когда Activity входит в состояние Started система вызывает эту функцию обратного вызова. После выполнения данного вызова Activity переходит в состояние Resumed и вызывает onResume().

onResume() – вызывается перед тем, как Activity будет доступно для активности пользователя (взаимодействие). Вызов при переходе в состояние Resumed.

onPause() – метод обратного вызова при переходе Activity в состояние Paused.

В этом методе принято освобождать ресурсы устройства. Не следует выполнять очень долгие операции, потому что в этот состоянии Activity всё ещё видима для пользователя, и если будет происходить загрузка данных на сервер, то приложение будет закрываться очень долго. Подобные действия лучше всего совершать в следующем методе.

onStop() – метод при переходе Activity в состояние Stopped. В нем оно полностью невидимо для пользователя. В данном методе следует освобождать ресурсы, которые не нужны пользователю, когда он не взаимодействует с Activity. Далее можно или вернуться в Activity или завершить его работу.

onDestroy() – метод, который вызывается перед тем, как Activity будет уничтожено.

Также есть дополнительный вызов *onRestart()* – вызывается после *onStop(),* когда текущий Activity должен быть снова отображен пользователю. После него следуют *onStart()* и *onResume().*

Важно помнить, что эти методы не вызывают смену состояния. Наоборот, смена состояния Activity является триггером, который вызывает эти методы.

Задача 1. Activity

Ссылка на проект:

https://github.com/zina-frid/AndroidLabs/tree/main/projects/lab2/lifecycle

В рамках данного задания было создано самое простое приложение с одним текстовым блоком, переопределены методы состояний приложения для вывода сообщений в журнал логов.

Листинг 1.1 Код файла MainActivity.kt

```
package com.zinafrid.lifecycle

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        Log.d("MainActivity", "onCreate()")
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }

    override fun onStart() {
        Log.d("MainActivity", "onStart()")
        super.onStart()
    }

    override fun onResume() {
        Log.d("MainActivity", "onResume()")
        super.onResume()
    }

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

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

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

Листинг 1.2 Код файла activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

1) SMS и входящий звонок

После запуска приложения (Рис. 3) в журнале логов, как и ожидалось, появились сообщения о том, что вызваны методы onCreate, onStart и onResume (Приложение было запущено, выведено на экран и стало доступно для взаимодействия).

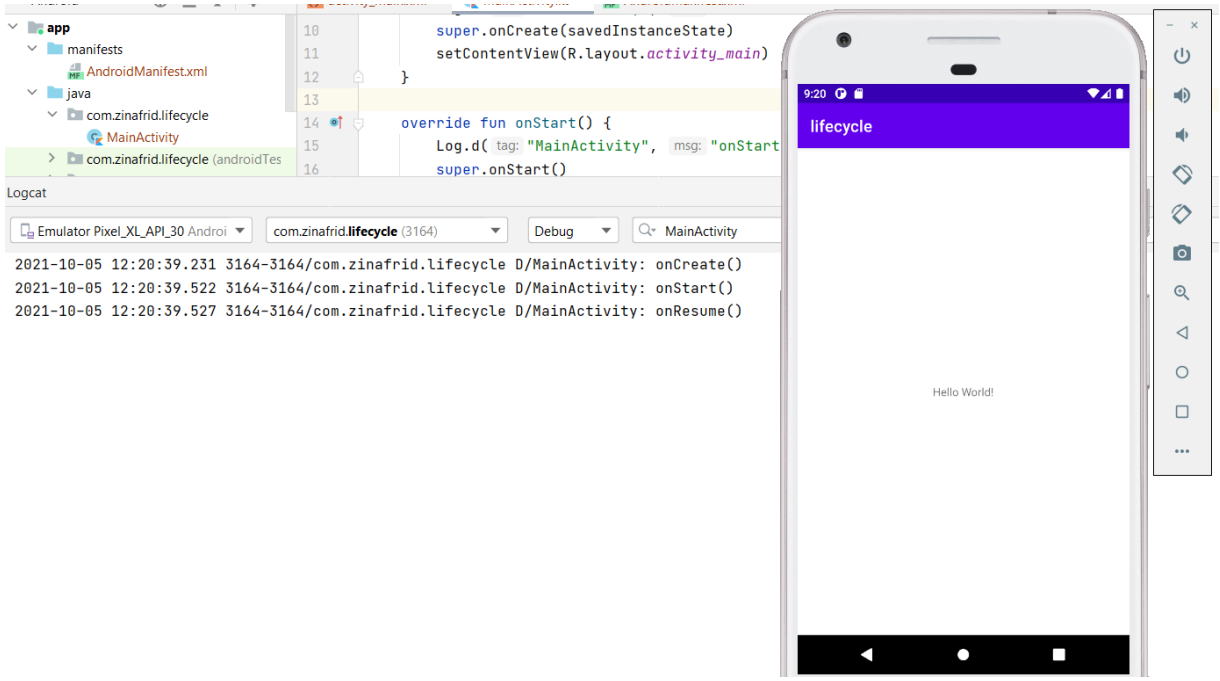

Рис. 3 Запуск приложения

Отправим на наше устройство SMS, которое всплывает на экране в виде баннера (Рис. 4). При получении уведомления Activity никак на это не отреагировало.

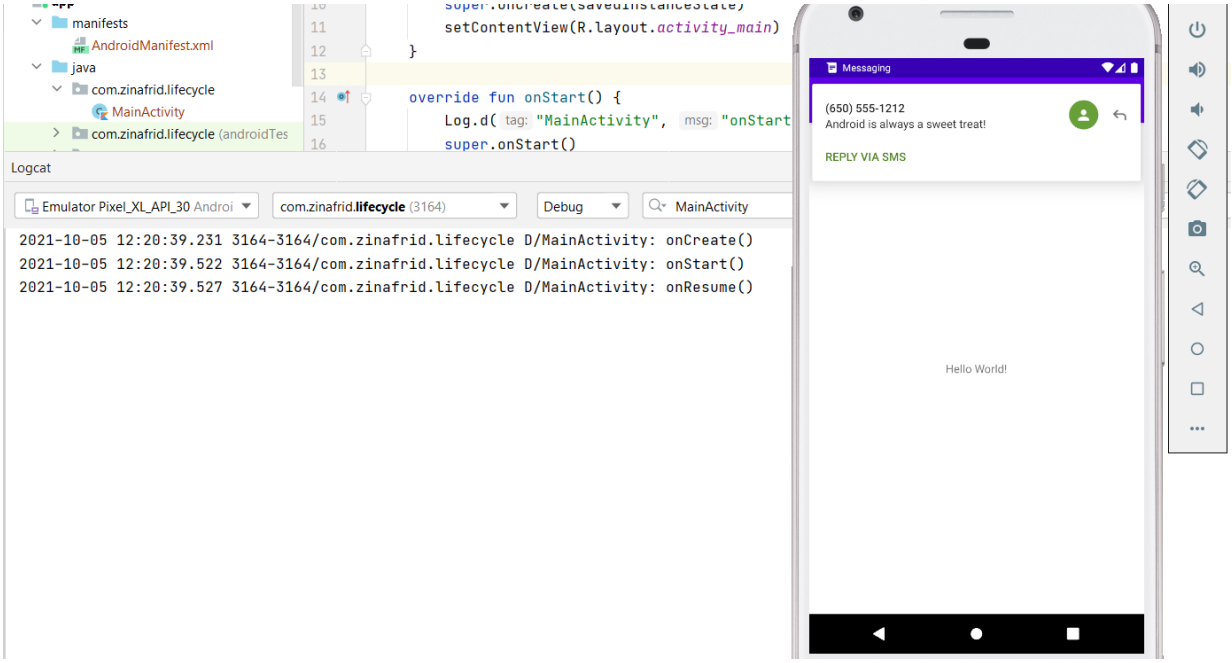

Рис. 4 Баннер с SMS

Нажмем на баннер, чтобы открыть сообщения и перейти ко всему диалогу (Рис. 5). На рисунке можно увидеть, что были вызваны методы onPause и onStop, приложение переходит в состояние Stopped. Это связано с тем, что новое приложение заняло весь экран, а значит наше больше не отображается, но всё еще запущено (метод onDestroy не вызывался).

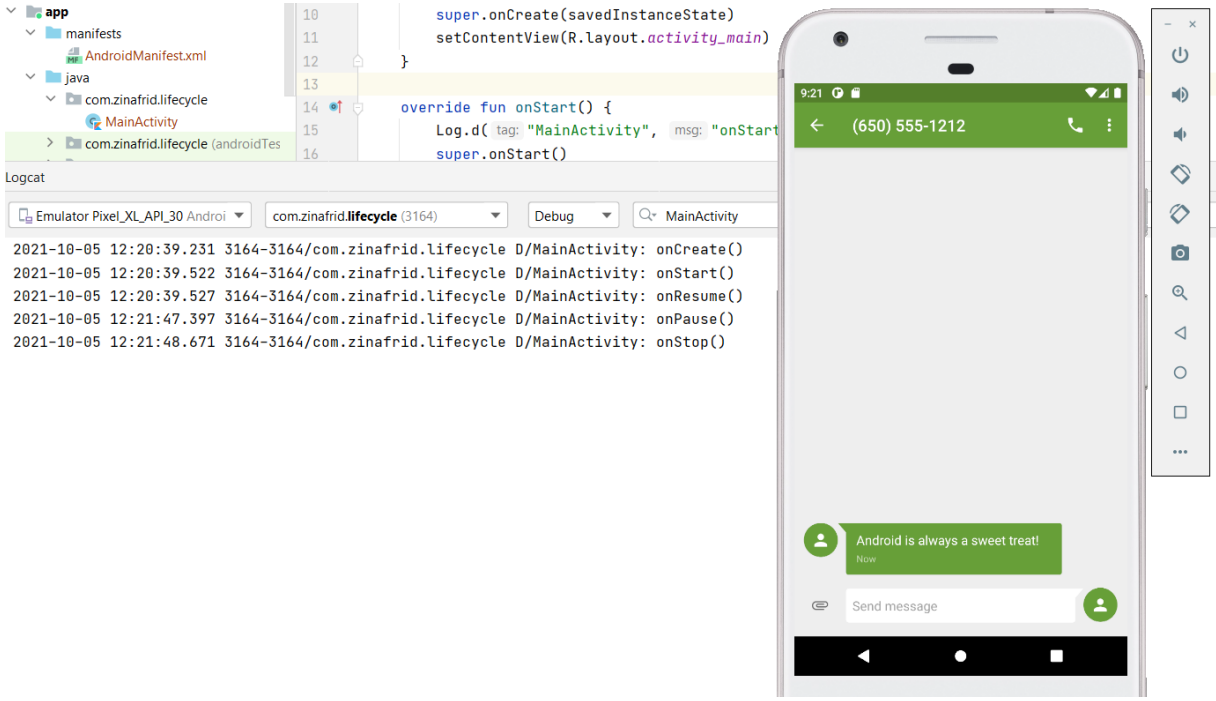

Рис. 5 Открытый диалог в сообщениях

После этого вернем окно нашего приложения (Рис. 6) и, как видно из журнала логов, вызваны методы onStart и onResume, то есть мы проходим через состояния Started и попадаем обратно в Resumed, и пользователь снова может взаимодействовать с нашим приложением.

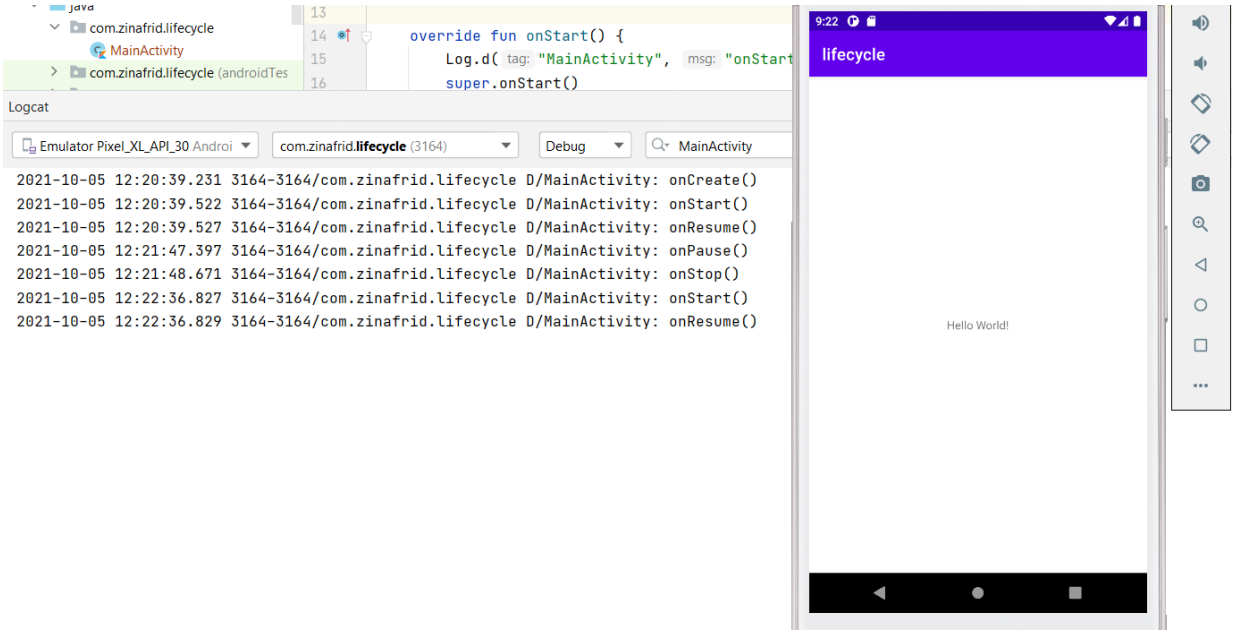

Рис. 6 Окно нашего приложения

Теперь попробуем позвонить (Рис. 7). Из логов видно, что были вызваны методы onPause и onStop, то есть, как и в случае с SMS, окно звонка заняло весь экран, а значит наше приложение не отображается, но все ещё запущено.

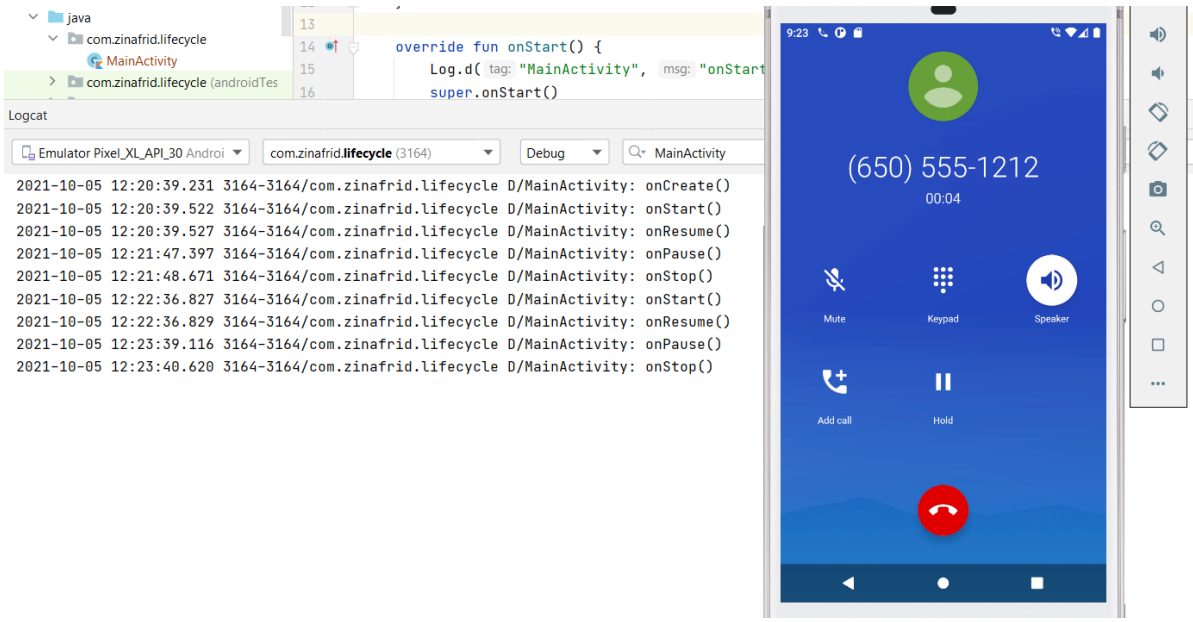

Рис. 7 Входящий звонок

2) Запуск многооконного режима

Во время работы приложения был запущен многооконный режим (Рис. 8). В журнале можно наблюдать последовательность сообщений, говорящих о том, что при переходе в новый режим процесс был уничтожен, а потом заново запущен, после чего перешел в режим ожидания, а возможность взаимодействовать была передана другому приложению. После выбора второго окна, наше приложение снова готово к взаимодействию с пользователем.

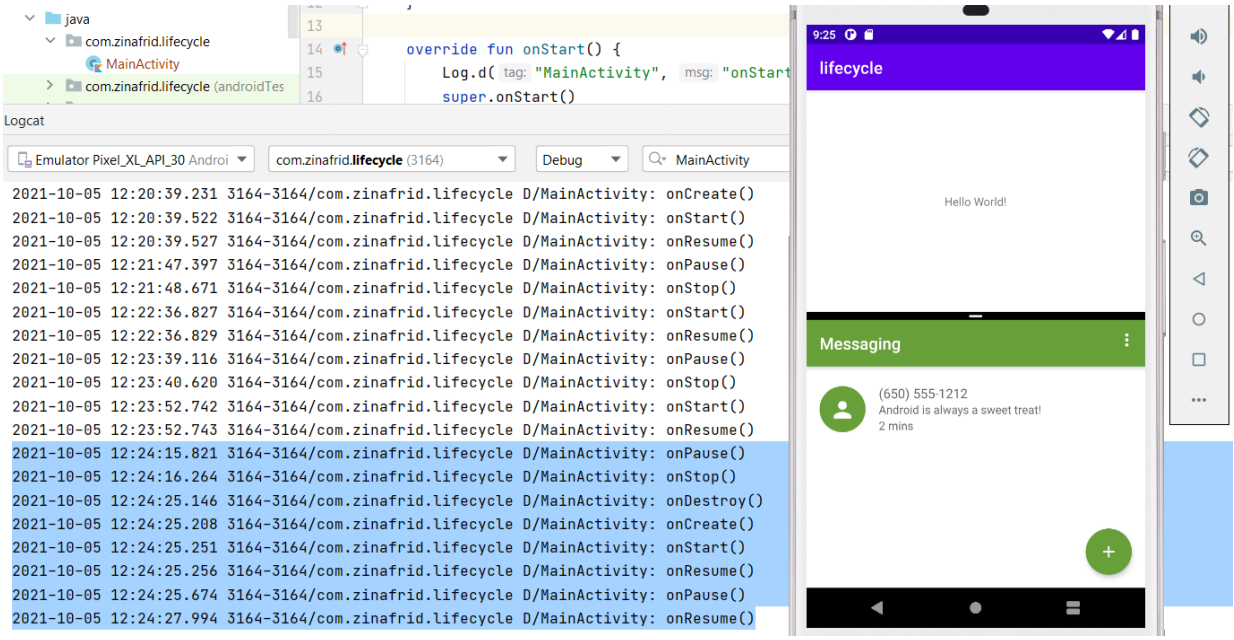

Рис. 8 Запуск многооконного режима

Далее многооконный режим был отключен, а в журнале появилась еще одна последовательность сообщений, говорящая о том, что приложение снова было полностью закрыто и заново запущено (Рис. 9).

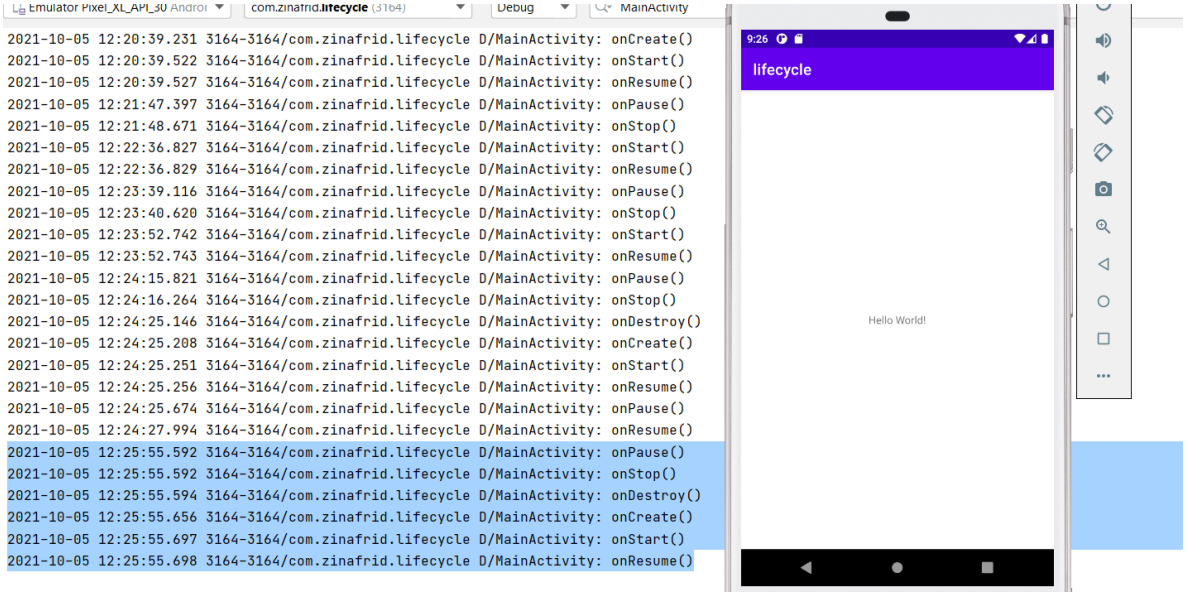

Рис. 9 Окно приложения

И наконец, зайдем в меню задач и удалим наше приложение оттуда. В списке логов мы видим, что при выполнении этих действий вызываются три метода onPause, onStop и onDestroy. Приложение было полностью закрыто.

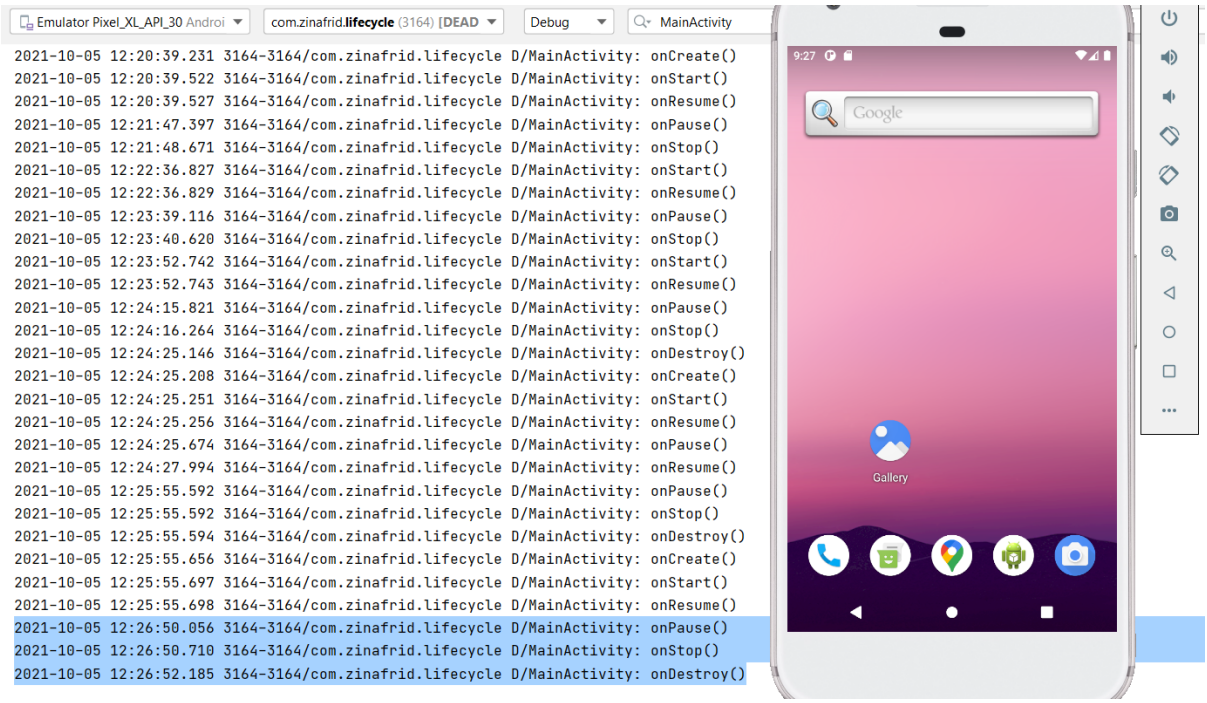

Рис. 10 Главный экран после закрытия приложения

Альтернативные ресурсы

На данный момент в мире существует огромное количество различных конфигураций устройств (телефоны, наручные часы, компьютеры, плееры, планшеты и т.д.), также устройства могут различаться ещё по большому числу параметров, например, язык или размер экрана. При написании android-приложения возникает вопрос, каким образом добиться универсальности. Для решения этой проблемы были созданы альтернативные ресурсы. Они позволяют выбрать те ресурсы, которые созданы для какой-либо характеристики устройства.

Задача 2. Alternative Resources

Тип ресурса: UI mode

Допустим, имеется приложение, в котором можно смотреть фильмы и слушать книги или подкасты.

Рассмотрим какой-нибудь день какого-нибудь человека. Например, человек встаёт с утра, идет завтракать и собирается на работу, а в это время он хочет слушать какой-нибудь подкаст, но, чтобы не ходить с телефоном по всему дому, человек надевает свои часы. В часах данное приложение будет использовать такое значение квалификатора, как “watch”, которое используется, когда устройство имеет дисплей и носится на руке. Потом этот человек садится в машину и едет на работу, но, увы, попадает в пробку, поэтому решает дослушать утренний подкаст или посмотреть видео уже с помощью возможностей своего автомобиля. Приложение в автомобиле будет использовать уже другое значение квалификатора, а именно “car”. Вечером человек уставший приходит с работы и хочет как-то расслабиться, посмотрев фильм. Он может включить телевизор, на котором приложение использует значение квалификатора “television”, или же погрузиться в атмосферу

полностью и включить свои очки виртуальной реальности, на которых приложение использует значение квалификатора “vrheadset”.

Таким образом, с помощью альтернативного ресурса UI mode приложение может запускаться на различных устройствах. Он может пригодиться, например, когда заряд на основном устройстве закончился, когда хочется большего комфорта, то есть смотреть с экрана телевизора, а не с маленького экрана телефона, или когда наоборот, возможности использовать другое устройство просто нет.

Задача 3. Best-matching resource

Выбрать ресурс по алгоритму для best-matching resource.

Вариант 23:	
Конфигурация устройства: LOCALE_LANG: fr LOCALE_REGION: rUS SCREEN_SIZE: small SCREEN_ASPECT: long ROUND_SCREEN: notround ORIENTATION: land UI_MODE: television NIGHT_MODE: notnight PIXEL_DENSITY: ldpi TOUCH: notouch PRIMARY_INPUT: qwerty NAV_KEYS: trackball PLATFORM_VER: v27	Конфигурация ресурсов: (default) port-xxhdpi-finger-qwerty-dpad small-land-notnight night-nodpi land en-desk-notouch-qwerty long-nodpi-nokeys-trackball rUS-notouch-v26 rFR-notnight-notouch-v26 en-rFR-notlong-vrheadset-12key-v26

1) Для начала выбросим все конфигурации ресурсов противоречащие конфигурации устройства.

(default) ~~port-xxhdpi-finger-qwerty-dpad~~ (ORIENTATION: land) small-land-notnight

~~night-nodpi~~ (NIGHT_MODE: notnight)
land
~~en-desk-notouch-qwerty~~ (LOCALE_LANG: fr)
~~long-nodpi-nokeys-trackball~~ (PRIMARY_INPUT: qwerty)
~~rUS-notouch-v26~~ (PLATFORM_VER: v27)
~~rFR-notnight-notouch-v26~~ (LOCALE_REGION: rUS)
~~en-rFR-notlong-vrheadset-12key-v26~~ (LOCALE_LANG: fr)

2) Далее идём по таблицы квалификаторов альтернативных ресурсов: “MCC and MNC”.

(default)
small-land-notnight
land

Совпадений нет, поэтому исключать нечего. Далее “Language and region” (для целей задания принято считать, что язык и регион не связаны и такой ресурс создать можно), но совпадений также нет. Продолжаем спускаться по списку вниз.

Следующее совпадение только со “Screen size”.

~~(default)~~
small-land-notnight
~~land~~

Таким образом, оставшийся каталог small-land-notnight.

Задача 4. Сохранение состояние Activity.

Задача 4.1. Поиск ошибок.

Необходимо протестировать приложение continuewatch и описать все найденные ошибки. Для поиска ошибок приложение было запущено на эмуляторе.

В ходе тестирования были найдены следующие ошибки:

1. Секундомер продолжает считать, когда приложение не активно (не отображается на экране).
2. Счетчик сбрасывается при кратковременных закрытиях приложения (при повороте экрана или переходе в многооконный режим и выходе из него).
3. При запуске приложения отображается текст "Hello world", который потом заменяется на “Second elapsed”. А при повороте экрана, аналогично, только текст “TextView”.
4. При повороте экрана таймер съезжает.

Исправления:

Во-первых, стоит разобраться с тем, когда счетчик увеличивается, а когда нет. Для этого добавим переменную, типа Boolean, которая будет сигнализировать о том, находится ли приложение на экране или нет. А в методах обратного вызова onStop() и onResume() будем её изменять.

Во-вторых, в файле MainActivity.kt IDE обратило наше внимание на строку:

```
textSecondsElapsed.setText("Seconds elapsed: " + secondsElapsed++)
```

Потому что не следует объединять отображаемый текст с setText.

Сделаем так, чтобы текст “Seconds elapsed: ” отображался в отдельном TextView, а сам таймер в отдельном, то есть изменим main_activity.xml. Аналогичные изменения внесем и в layout для горизонтального положения. Попутно исправляется и третья найденная ошибка. А в файле MainActivity.kt заменим вышеупомянутую строку на следующую (то есть будет выводить лишь изменяющееся значение счетчика):

```
textSecondsElapsed.text = "${secondsElapsed++}"
```

В-третьих, TextView в layout для горизонтального положения ни к чему не привязан. Без связей он будет находиться по центру только во время

разработки, при запуске он съедет на (0,0). Поэтому добавим связи. По сути теперь, после внесенных изменений, дефолтный main_activity будет одинаково подходить и для вертикального, и для горизонтального положения, однако я не стала удалять последний, так как для целей задания это не важно.

Задача 4.2. Сохранение состояние Activity (1).

Ссылка на проект:
https://github.com/zina-frid/AndroidLabs/tree/main/projects/lab2/continuewatch

Используем методы onSaveInstanceState/onRestoreInstanceState для исправления неправильного подсчета времени.

Листинг 4.1 Методы onSaveInstanceState/onRestoreInstanceState

```
override fun onSaveInstanceState(outState: Bundle) {
    outState.run {
        putInt(seconds, secondsElapsed)
    }
    super.onSaveInstanceState(outState)
}

override fun onRestoreInstanceState(savedInstanceState: Bundle) {
    super.onRestoreInstanceState(savedInstanceState)
    savedInstanceState.run {
        secondsElapsed = getInt(seconds)
    }
}
```

Например, при изменении ориентации значительно меняется конфигурация устройства, поэтому операционной системе проще всего разрушить и перезапустить Activity. Когда ОС разрушает наше приложение с определёнными целями, вызывается метод onSaveInstanceState. Таким образом, необходимо переопределить метод onSaveInstanceState() для сохранения секунд, поэтому записываем их количество в Bundle outstate. Bundle - ассоциативный массив с парой: ключ (String) – значение (Parcelable).

Если в методе onCreate() при запуске приложения аргумент типа Bundle не равен null, то вызывается метод onRestoreInstanceState, и происходит восстановление данных при помощи переданного Bundle. Если Bundle равен null, то создаётся новая сущность Activity.

Данный подход позволил исправить ошибки в реализации приложения, и теперь оно соответствует требованиям задания. Если приложение завершено, то при следующем заходе счётчик сбрасывался в 0.

Задача 4.3. Сохранение состояние Activity (2).

Ссылка на проект:

https://github.com/zina-frid/AndroidLabs/tree/main/projects/lab2/continuewatch2

Неправильный подсчет времени в приложении ContinueWatch можно исправить по-другому, с использованием Shared Preferences.

Листинг 4.2 Использование Shared Preferences

```
private lateinit var sharedPref: SharedPreferences

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    textSecondsElapsed = findViewById(R.id.timer)
    sharedPref = getSharedPreferences(SECONDS, Context.MODE_PRIVATE)
    backgroundThread.start()
}

override fun onStop() {
    active = false
    val editor = sharedPref.edit()
    editor.putInt(SECONDS, secondsElapsed)
    editor.apply()
    super.onStop()
}

override fun onResume() {
    active = true
    secondsElapsed = sharedPref.getInt(SECONDS, 0)
    super.onResume()
}
```

Как сказано в документации, если у нас есть относительно небольшая коллекция пар "ключ-значение", которую мы хотим сохранить, то нам следует использовать SharedPreferences API.

В код программы был добавлен экземпляр класса SharedPreferences. Этот класс позволяет записывать необходимые данные во внутреннее хранилище приложения. Тем самым данные, которые терялись при определённых состояниях Activity, были сохранены, а приложение стало работать как надо.

С помощью метода getSharedPreferenses(String name, int mode) можно получить содержимое файла настроек “name”, возвращая объект SharedPreferences, через который можно извлекать и изменять его значения. Всем вызывающим с одним и таким же именем возвращается один экземпляр объекта SharedPreferences, что означает, что они будут видеть изменения, сделанные друг другом. Если каталога настроек еще не существует, он будет создан при вызове этого метода.

С помощью edit() и putInt() записываем количество секунд в файл настроек, а с помощью getInt() получаем их оттуда.

Данный подход позволил исправить ошибки в реализации приложения, и теперь оно соответствует требованиям задания. Если приложение завершено, то при следующем заходе счётчик сбрасывался в 0.

Задача 4.4. Сравнение решений.

Bundle и SharedPreferences – это два разных подхода.

Объекты класса Bundle в основном используются для сохранения состояния объекта в процессе жизненного цикла того или иного компонента android-приложения (например, Activity). Bundle содержит состояние Activity, если оно было сохранено, и, например, при перевороте экрана, когда Activity

разрушается, в новое передаётся наш Bundle, с помощью которого мы восстанавливаем значение таймера.

SharedPreferences используется для сохранения каких-либо пар ключ-значение для их долгосрочного хранения. Основное предназначение SharedPreferences – сохранение настроек приложения. То есть, по сути, это файл с настройками, поэтому мы можем сохранять значение таймера, а при возвращении в приложение одно будет взято оттуда и будет прежним.

Вывод

В ходе данной лабораторной работы был изучен жизненный цикл Activity. Были рассмотрены методы обратного вызова (callbacks), которые определяют поведение Activity при смене состояния, и получены навыки работы с ними.

Были изучены основные возможности и свойства альтернативных ресурсов. Для одного из таких ресурсов была приведена ситуация, в которой он может пригодиться. Также было выполнено упражнение по best-matching resource для понимая алгоритма выбора лучшего ресурса.

Была проведена работа по тестированию, нахождению и исправлению ошибок в приложении. При исправлении ошибок было приведено два способа сохранения состояния Activity. В первом случае использовался объект Bundle, который помогает восстановить значение. Во втором случае мы работали с SharedPreferences, то есть сохраняли настройки приложения в файл. Несмотря на то, что оба метода решили одну и ту же задачу корректно, они разные, имеют разные принципы работы, поэтому при написании приложения, необходимо выбирать конкретный метод под свои цели.

Также получены навыки работы с эмулятором.

Список источников

1. https://github.com/andrei-kuznetsov/android-lectures
2. https://developer.android.com/guide/components/activities/activity-lifecycle
3. https://developer.android.com/guide/topics/resources/providing-resources
4. https://developer.android.com/guide/topics/resources/providing-resources#BestMatch
5. https://developer.android.com/guide/components/activities/activity-lifecycle#save-simple,-lightweight-ui-state-using-onsaveinstancestate
6. https://developer.android.com/training/data-storage/shared-preferences