Федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский политехнический университет Петра Великого»

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

**Отчёт по лабораторной работе №2**

**Дисциплина:** Проектирование мобильных приложений

**Тема:** «Activity Lifecycle. Alternative resources»

**Вариант №14**

Выполнил студент гр. 3530901/70202

Д.В. Павлов

*(подпись)*

Преподаватель

А.Н.Кузнецов

*(подпись)*

Санкт-Петербург

2019

1. **Цели работы**

• Ознакомиться с жизненным циклом *Activity*;

• Изучить основные возможности и свойства *alternative resources*.

1. **Выполнение работы**
   1. **Activity**

В данном пункте лабораторной работы необходимо продемонстрировать жизненный цикл *activity* на любом нетривиальном примере.

Далее представлено, какие методы были вызваны в различных случаях как для устройства, использующего *Android* версии 9.0, так и для устройства на *Android* 4.1.

Вызов функций прослеживался через *logcat*, как имеющих метку *STATE*.

1. Старт приложения

Android 4.1:

onCreate

onStart

onResume

Android 9.0:

onCreate

onStart

onResume

Вызов *lifecycle* методов для обоих устройствах соответствует ожиданиям.

1. Включение/выключение экрана

Android 4.1:

onPause

onStop

onStart

onResume

Android 9.0:

onPause

onStop

onStart

onResume

Ничего также не вызывает вопросов.

1. Переход к списку запущенных приложений

Android 4.1:

По моему мнению, введение вызовов *lifecycle* методов является правильным решением, поскольку в данной случае приложение не будет потреблять ресурсы, когда пользователь им не пользуется.

Android 9.0:

onPause

onStop

1. Переход на домашний экран из приложения

Android 4.1:

onPause

onStop

Android 9.0:

onPause

onStop

Вызов *lifecycle* методов для обоих версий *Android* одинаково.

1. Открытие запущенного ранее приложения через список запущенных приложений

Android 4.1:

onStart

onResume

Android 9.0:

onStart

onResume

Вызов *lifecycle* методов для обоих версий *Android* в данном случаеодинаково.

1. Поворот экрана

Android 4.1:

onPause

onStop

onDestroy

onCreate

onStart

onResume

Android 9.0:

-

Неожиданным было поведение приложения на устройстве, использующем *Android* 9.0 – при повороте экрана не было вызвано ни одного *lifecycle* метода, хотя, в соответствии с информацией, представленной в официальной документации, система временно уничтожает *activity* при повороте экрана, т.к. имеет место изменение конфигурации.

1. Приход сообщения / вызова.

Android 4.1:

-

Android 9.0:

-

Появление уведомления / вызова сверху экрана никак не влияет на поведение приложения.

1. Отклонение входящего вызова

Android 4.1:

-

Android 9.0:

-

Отклонение входящего вызова никак не влияет на поведение приложения.

1. Принятие входящего вызова и возврат к приложению после его окончания

Android 4.1:

onPause

onStop

onStart

onResume

Android 9.0:

onPause

onStop

onStart

onResume

При принятии входящего сначала вызывается метод onPause, происходит зависание (длительность зависит от быстродействия устройства), после чего на экране появляется режим звонка и мы уходим на onStop. После завершения общения друг за другом происходят onStart и onResume.

1. Ответ на СМС в появившейся строке уведомления

Android 4.1:

-

Android 9.0:

-

Никак не влияет на поведение приложения.

1. Закрытие приложения через стек запущенных приложений (после перехода к списку запущенных приложений соответственно)

Android 4.1:

onPause

onStop

onDestroy

Android 9.0:

onPause

onStop

onDestroy

Сюрпризов не произошло.

1. Закрытие приложения кнопкой «назад»

Android 4.1:

onPause

onStop

onDestroy

Android 9.0:

onPause

onStop

onDestroy

Также все очевидно.

1. Открытие *Notification bar*

Android 4.1:

-

Android 9.0:

-

В более ранних версиях в данном случае вызывался метод onPause, но на личном опыте проверить не удалось, так как попытки запуска на устройствах API15 и ниже не увенчались успехом.

1. Открыть окно СМС через уведомление

Android 4.1:

onPause

onStop

Android 9.0:

onPause

onStop

Довольно очевидно, так как переходим к другому приложению.

Приблизительное время решения задачи составило около часа, так как производились попытки запуска на API15 и ниже, которые ни к чему не привели. Задание выполнялось в паре.

**2.2. Alternative Resources**

Помимо *default* ресурсов, в *Android* есть также и альтернативные ресурсы. Почти каждое приложение должно предоставлять такие ресурсы для поддержки устройств различной конфигурации – с разными расширениями экрана, на различных языках и др. Во время работы приложения, *Android* определяет конфигурацию устройства, на котором приложение запущено и загружает подходящий ресурс.

Задачей данного пункта является демонстрация работы альтернативного ресурса *primary non-touch navigation method* на каком-либо примере.

Был создан новый альтернативный ресурс *– drawable-dpad* (Рис.2.2.1, *dpad*– одно из возможных значений для *primary non-touch navigation* method квалификатора). Определяет устройство, у которого есть dpad.

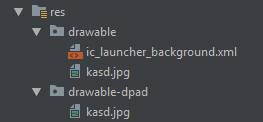


Рис.2.2.1.

В обе директории были добавлены 2 разных изображения, но под одним название.

Проект был запущен на обычном смартфоне и на самостоятельном созданном устройстве с dpad. Как и ожидалось, если у устройства имеет тачпад, изображение берется из *default* директории, когда нет – используются альтернативные ресурсы.

Данный альтернативный ресурс, к примеру, используется c геймпадами для android-устройств. Так-как там присутствует много разных кнопок, для визуального закрепления горячих клавиш должен использоваться другой ресурс, потому что людям без такого прибора данная информация не будет актуальна.

Время решения задачи составило около часа, так как сначала бралась конфигурация wheel. Оказалось, что эмулятор её не поддерживает. После замены wheel на dpad задание было сделано за 5 минут.

**2.3.** **Best-matching resource**

В данном случае необходимо для заданного набора альтернативных ресурсов, предоставляемых приложением, и заданной конфигурации устройства, объяснить, какой ресурс будет выбран в конечном итоге.

Процесс выбора ресурсов следующий:

1. Убираются из рассмотрения опции, противоречащие конфигурации устройства;
2. Берется квалификатор высшего порядка (в соответствии с таблицей конфигурационных квалификаторов, представленной в документации) и происходит проверка, удовлетворяет ли одна из директорий ресурсов данному квалификатору. Если ни одна директория такому квалификатору не удовлетворяет, проверяется следующий квалификатор, в противном случае из рассмотрения исключаются директории, не содержащие данного квалификатора.

Пользуясь данным алгоритмом, можно сделать вывод о том, что ресурс, который будет выбран для заданной квалификации устройства - *mdpi*, поскольку другие ресурсы противоречат по одному или нескольким квалификаторам. Рассуждения представлены на Рис.2.3.1.

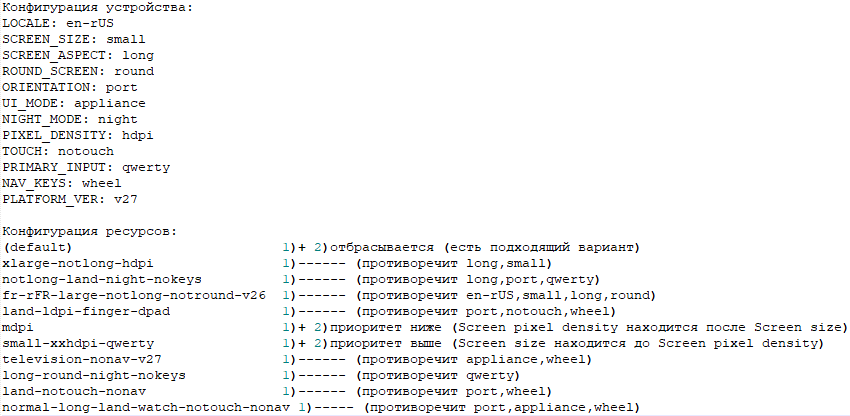


Рис.2.3.1.

Время решения задачи составило около 10 минут.

**2.4. Сохранение состояние Activity**

Задано написанное приложение *continuewatch,* которое должно считать, сколько секунд пользователь провел в этом приложении. При этом приложение должно считать секунды только тогда, когда оно отображается на экране, и значение счетчика должно сохраняться при перезапуске приложения. Необходимо найти ошибки в этом приложении и исправить их.

Было произведено логирование. Поведение счетчика никак не изменялось при вызове различных методов *lifecycle* (счет шел постоянно, за исключением onDestroy).

Для корректной работы приложения был модернизирован backgroundThread = Thread{}. Туда было добавлено условие *if (isWorking),* где *isWorking* – булевая переменная. Она принимает значение *true*, когда приложение отображено на экране, и *false*, когда оно скрыто.

Для исправления ошибок использовались методы *Activity::onSaveInstanceState/onRestoreInstanceState,* сохраняющие состояние приложения после вызова метода *onStop()* и восстанавливающие его перед вызовом метода *onStart()* соответственно.

Решение задачи:

package ru.spbstu.icc.kspt.lab2.continuewatch  
  
import android.annotation.SuppressLint  
import androidx.appcompat.app.AppCompatActivity  
import android.os.Bundle  
import android.util.Log  
import kotlinx.android.synthetic.main.activity\_main.\*  
  
const val *KEY\_SECONDS\_ELAPSED* = "key\_seconds\_elapsed"  
  
class MainActivity : AppCompatActivity() {  
 private var isWorking = true  
 private var secondsElapsed = 0  
  
 private val TAG = "STATE"  
  
 @SuppressLint("SetTextI18n")  
 private var backgroundThread = Thread **{** while (true) {  
 if (isWorking) textSecondsElapsed.post **{** textSecondsElapsed.*text* = "Seconds elapsed: " + secondsElapsed++  
 **}** Thread.sleep(1000)  
 }  
 **}** override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 log("onCreate")  
 setContentView(R.layout.*activity\_main*)  
 backgroundThread.start()  
 }  
  
 /\*override fun onStart() {  
 super.onStart()  
 log("onStart")  
 }  
  
 override fun onResume() {  
 super.onResume()  
 log("onResume")  
 }\*/  
  
 override fun onPause() {  
 super.onPause()  
 isWorking = false  
 log("onPause")  
 }  
  
 /\*override fun onStop() {  
 super.onStop()  
 log("onStop")  
 }  
  
 override fun onDestroy() {  
 super.onDestroy()  
 log("onDestroy")  
 }\*/  
  
 override fun onRestart() {  
 super.onRestart()  
 isWorking = true  
 log("onRestart")  
 }  
  
 private fun log(msg: String) {  
 Log.i(TAG, msg)  
 }  
  
 override fun onRestoreInstanceState(savedInstanceState: Bundle) {  
 super.onRestoreInstanceState(savedInstanceState)  
 secondsElapsed = savedInstanceState.getInt(*KEY\_SECONDS\_ELAPSED*)  
 log("onRestoreInstanceState")  
 }  
  
 override fun onSaveInstanceState(outState: Bundle) {  
 super.onSaveInstanceState(outState)  
 outState.putInt(*KEY\_SECONDS\_ELAPSED*, secondsElapsed)  
 log("onSaveInstanceState = $outState")  
 }  
}

Время решения задачи удалось значительно сократить благодаря сайту udacity.com. Вместе с ним оно составило около 40 минут.

1. **Выводы**

В ходе выполнения лабораторной работы было проведено ознакомление с *Activity lifecycle* методами. Полный жизненный цикл *Android activity*:

onCreate() – onStart() – onResume() – onPause() – onStop() – onDestroy()

Затем, были изучены основные возможности и свойства альтернативных ресурсов:

1)Было продемонстрировано использование одного из них *(primary non-touch navigation method, конфигурация dpad)*. Для этого даже понадобилось создавать новое устройство.

2)Также был отработан алгоритм определения альтернативных ресурсов при заданной конфигурации устройства.

В последнем задании было произведено ручное тестирование путем логирования приложения. Были изучены методы *onSaveInstanceState* и *onRestoreInstanceState*, позволяющие запоминать и восстанавливать состояние *Activity*. Они понадобились для исправления ошибок, допущенных студентом.