САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ПЕТРА ВЕЛИКОГО

Институт компьютерных наук и технологий
Высшая школа интеллектуальных систем и суперкомпьютерных технологий

Отчет
по Лабораторная Работа №2
Дисциплина
«Проектирование мобильных приложений»

выполнил: Кривицкий В.В.
группа: 3530901/90202
преподаватель: Кузнецов А.Н.

Санкт-Петербург
2021

Репозиторий

https://github.com/OGSegu/AndroidLabs

1. Цели

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

2. Ход работы

Activity. *Продемонстрируйте жизненный цикл Activity на любом нетривиальном примере*

Создадим Activity и переопределим все базовые методы, добавив к ним логирование, для удобства изучения. Листинг 1.

Листинг 1. Activity с переопределенными методами

```
public class MainActivity extends AppCompatActivity {
    private final static String TAG = "MainActivity";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.d(TAG, "onCreate");
    }
    @Override
    protected void onDestroy(){
        super.onDestroy();
        Log.d(TAG, "onDestroy");
    }
    @Override
    protected void onStop(){
        super.onStop();
        Log.d(TAG, "onStop");
    }
    @Override
    protected void onStart(){
        super.onStart();
        Log.d(TAG, "onStart");
    }
    @Override
    protected void onPause(){
        super.onPause();
        Log.d(TAG, "onPause");
    }
    @Override
    protected void onResume(){
        super.onResume();
        Log.d(TAG, "onResume");
    }
    @Override
    protected void onRestart(){
        super.onRestart();
        Log.d(TAG, "onRestart");
    }
}
```

Все базовые действия, по типу сворачивания приложения, открытие другого, завершение его и тд., не представляют особый интерес, так как происходящее достаточно очевидно. Предлагаю рассмотреть более занятный пример с принятием звонка, получением-открытием смс, блокировкой экрана устройства.

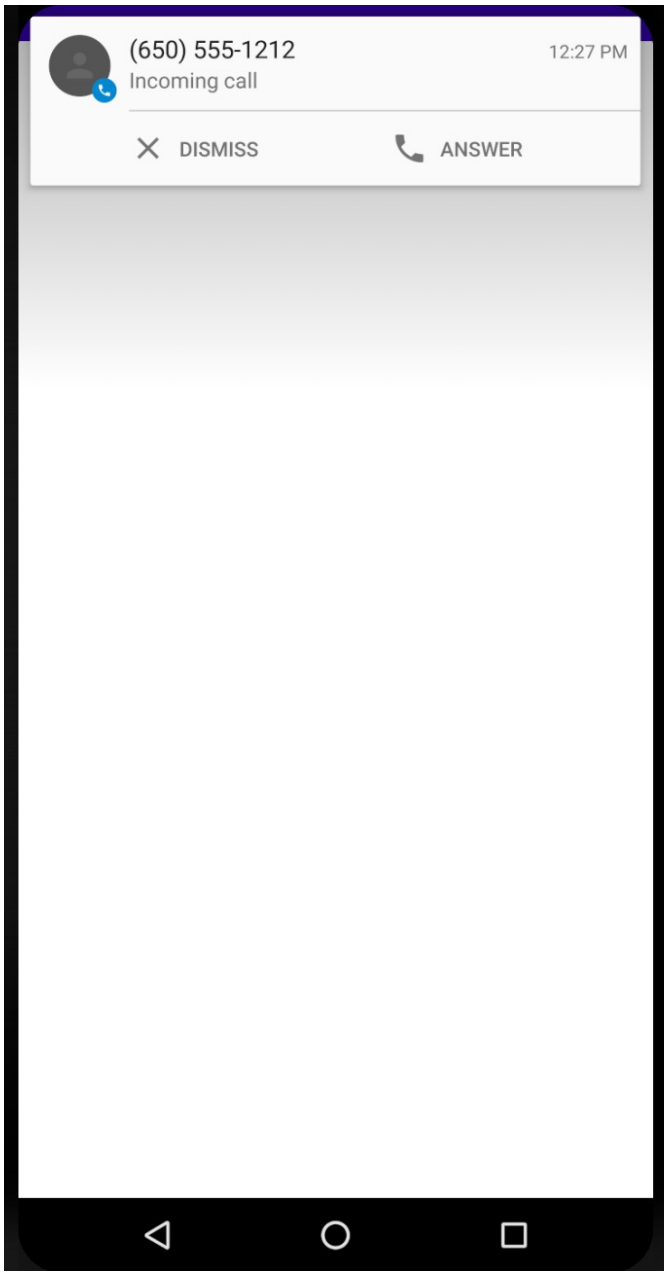

Рис. 1 Отправим звонок на эмулятор.

На данный момент никакого изменения в состояние нашего приложения не произошло. Попробуем принять звонок.

```
D/MainActivity: onPause
```

Рис. 2 Колбэк при принятие звонка.

А при окончание разговора произойдет вполне очевидный onResume.

Теперь отправим СМСку и посмотрим, что будет если открыть ее.

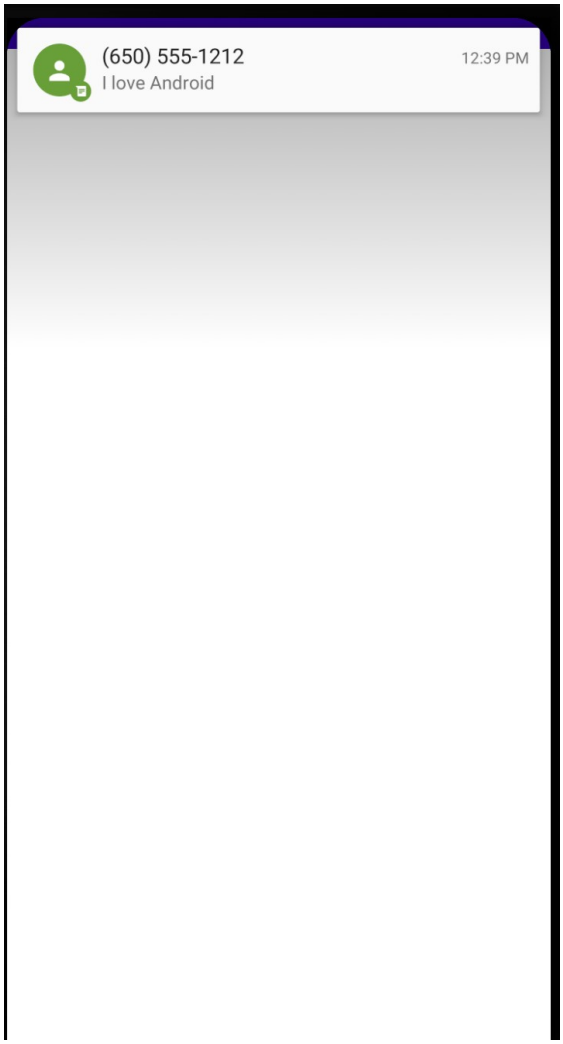

Рис. 3 Отправка СМС на эмулятор.

Откроем СМС и взглянем как изменилось наше состояние.

```
D/MainActivity: onPause
D/EGL_emulation: eglMakeCurre
E/Surface: getSlotFromBufferL
D/MainActivity: onStop
```

Рис. 4 Колбэк при открытие СМСок

Из рисунка выше заметим, что в это случае, для нашего активити сработал onStop, что достаточно интересно.
Вспоминания некоторые телефоны, видимо на более новых версия Андроида, во многих из них есть ф-ция где, при нажатие на подобные уведомления, можно отправить быстрый ответ, мне кажется в подобном случае отработало бы только onPause.

При блокировки экрана и разблокировки, порядок коллбеков следующий: Рис. 5

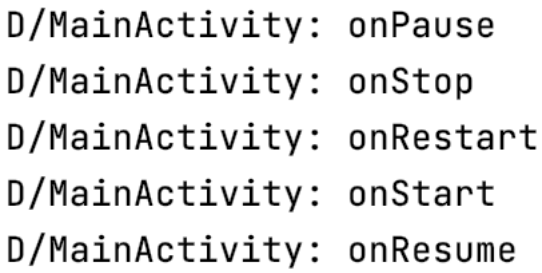

Рис. 5 Порядок коллбеков для блокировки и разблокировки

Благодаря полученным знаниям, теперь при разработке нашего приложения, мы можем более грамотно выстраивать взаимоотношение пользователя с нашим приложением. Так например – освобождать некоторую часть ресурсов в onPause и большую в onStop.

Alternative Resources. *Привести пример использования альтернативного ресурса Navigation key availability*

У данной конфигурации есть две опции:

- **navexposed** – навигации клавиши доступны пользователю
- **navhidden** – навигации выключены для пользователя

Если у пользователя отключена панель навигации, то за управление, как правило, происходит жестами по бокам экрана “Swipe”. Для удобства использования пользователем нашего приложения, возможно стоит как-то по другому расположить кликабелбные “блоки”, “текста”, которые находились по бокам. Так как из-за этого может произойти неосознанное использование навигации, когда например наш пользователь хотел как-то провзавимодействовать с приложением.

Best-matching Resource.

Для следующей конфигурации выберем ресурс.

```
LOCALE_LANG: en
LOCALE_REGION: rUS
SCREEN_SIZE: normal
SCREEN_ASPECT: long
ROUND_SCREEN: notround
ORIENTATION: port
UI_MODE: vrheadset
NIGHT_MODE: notnight
PIXEL_DENSITY: tvdpi
TOUCH: finger
PRIMARY_INPUT: nokeys
NAV_KEYS: trackball
PLATFORM_VER: v27
```

Конфигурация ресурсов:

```
(default)
land-appliance-notnight-ldpi-dpad-v27
fr-xlarge-notlong-round-land-watch-night-xxhdpi
ldpi-notouch-trackball
land-notouch-v27
port-12key-trackball
rCA-small-dpad
en-rCA-notround-dpad
v27
notlong-desk-trackball
rUS-normal
```

1. Отбросим все альтернативные ресурсы имеющие противоречущие параметры

```
(default)
~~land-appliance-notnight-ldpi-dpad-v27~~
~~fr-xlarge-notlong-round-land-watch-night-xxhdpi~~
~~ldpi-notouch-trackball~~
~~land-notouch-v27~~
~~port-12key-trackball~~
~~rCA-small-dpad~~
~~en-rCA-notround-dpad~~
v27
~~notlong-desk-trackball~~
rUS-normal
```

2. Пройдемся заново по параметрам в порядке приоритета. Из имеющихся, регион – rUS является самым приоритетным в данном случае, теперь удалим все альтернативные ресурсы не имеющие данного параметра.

```
(default)
rUS-normal
```

По итогу у нас остается два ресурса. Какой из них выберет Андроид, описано в документации. Итого имеем, что для устройства с характеристиками описанными выше будет выбрана конфигурация: **rUS-normal**

Также, возможен случай, когда у нас не будет подходящего ресурса, тогда будет использован (default)

Сохранение состояние Activity.

Поиск ошибок

1. Не сохраняем состояние Activity → данные пропадают (при повороте экрана например)
2. Отсутствие флага для остановки потока → когда мы не в приложение счет продолжается
3. Отсутствие модификатора volatile для переменной secondsElapsed → есть вероятность, что backgroundThread и UI Thread прочитают разные значения.
4. Конкатенация строк в цикле .setText(s1 + s2) → лишние аллокации.

5. Строка не вынесена в ресурсы → сложность переиспользования и смены языка (скорее хорошая практика)
6. Thread.sleep происходит первым действием в потоке → мы теряем 1 секунды при старте
7. Отстуствует ресурс для горизонтального ориентирования → некорректное отображение элемента текста

Сохранение состояния Activity(1).

Выполним сохранение состояния Activity при помощи onSaveInstanceState, onRestoreInstanceState.

Листинг 2

```
override fun onSaveInstanceState(outState: Bundle) {
    outState.putInt(SECONDS_ELAPSED_TEXT, secondsElapsed)
    Log.d(TAG, "onSaveInstanceState: " + secondsElapsed)
    super.onSaveInstanceState(outState)
}
override fun onRestoreInstanceState(savedInstanceState: Bundle) {
    secondsElapsed = savedInstanceState.getInt(SECONDS_ELAPSED_TEXT)
    Log.d(TAG, "Loading seconds: " + secondsElapsed)
    super.onRestoreInstanceState(savedInstanceState)
}
```

Проверим работоспособность данного кода на примере переворота экрана. Рис 6., Рис. 7

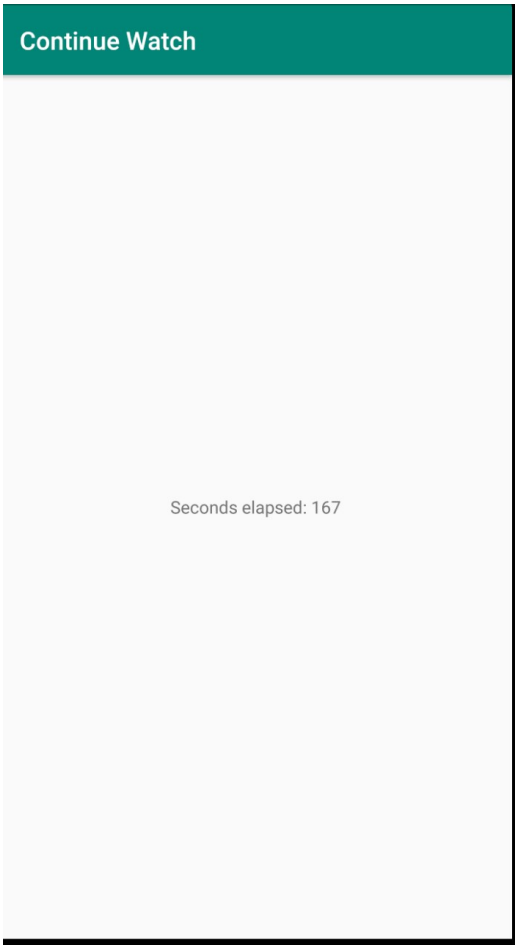

Рис 6. Приложение по подсчету секунд

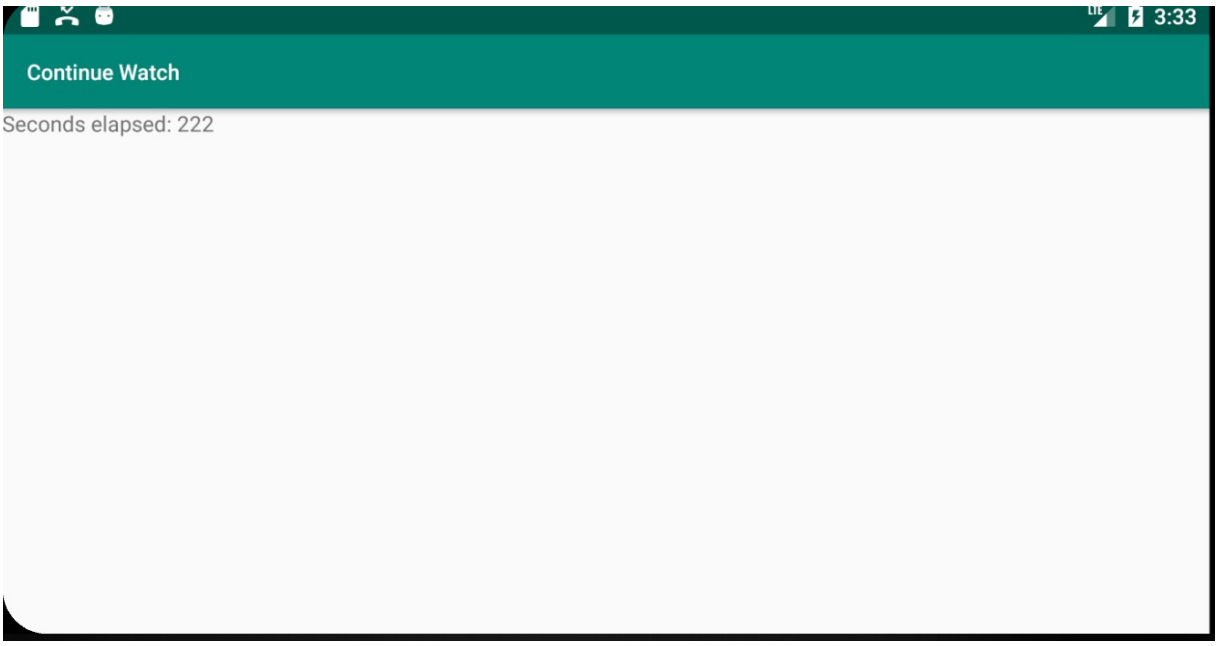

Рис. 7 Приложение в другой ориентации

Между переворачиванием экрана и созданием Рис.6 прошло некоторое время. Но становится ясно, что наше приложение начало отчет не заново, а продолжило, что говорит о корректности нашего решения.

Сохранение состояния Activity(2).

Выполним сохранение состояния Activity при помощи Activity Lifecycle callbacks и Shared Preferences.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    val sharedPref = getPreferences(Context.MODE_PRIVATE) ?: return
    secondsElapsed = sharedPref.getInt(SECONDS_ELAPSED_TEXT, 0)
    setContentView(R.layout.activity_main)
    textSecondsElapsed = findViewById(R.id.textSecondsElapsed)
    backgroundThread.start()
}
override fun onPause() {
    val sharedPref = getPreferences(Context.MODE_PRIVATE) ?: return
    with (sharedPref.edit()) {
        putInt(SECONDS_ELAPSED_TEXT, secondsElapsed)
        apply()
    }
    super.onPause()
}
override fun onResume() {
    val sharedPref = getPreferences(Context.MODE_PRIVATE) ?: return
    secondsElapsed = sharedPref.getInt(SECONDS_ELAPSED_TEXT, 0)
    super.onResume()
}
```

Выполняя данный код получим аналогичные результаты с рис. 6 и рис. 7, что говорит о корректности работы нашего приложения.

Что произойдет если мы будем сохранять счетчик в sharedPref на коллбэке onStop, а восстанавливать в onResume. На самом деле, помня жизненный цикл Activity, описать проблемную ситуацию – тривиально. OnResume – OnStop не являются парными, это означает, что существует такой кейс, когда onStop будет вызван 0 раз, а onResume будет вызываться N > 0, что будет причиной постоянной подгрузки значения, сохраненного последний раз в коллбеке onStop. Пример такой ситуации (API 23): принятие звонка, вызывает только onPause (Рис.2), соответственно вернувшись в приложение, мы подгрузим старое значение.

Сравнение решегний.

Главным отличием этих двух подходов будет то, что при закрытие приложения onSaveInstanceState не сохранит данные, когда SharedPreference это сделает. Поэтому для хранения какой-то информации мы однозначно будем использовать SharedPref.

4. Вывод

В данной лабораторной работе были выполнены все поставленные цели и решены требуемые задачи.
При помощи переопределения основных коллбеков Activitity и вставки туда сообщений для логирования, мы познакомились с жизненным циклом. А также, узнали предназначение альтернативных ресурсов и вручную воспроизвели алгоритм выполняемый Андроидом автоматически для подборки лучшего ресурса для конкретной конфигурации устройства.