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

Лабораторная работа

Дисциплина: Проектирование мобильных приложений

Тема: Жизненный цикл приложения, альтернативные ресурсы

Выполнил студент гр. 3530901/90201 _____Е. В. Бурков
(подпись)

Принял старший преподаватель _____ А. Н. Кузнецов
(подпись)

“_____” _____ 2021 г.

Санкт-Петербург
2021

Содержание

Цели

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

Задачи

- Продемонстрировать жизненный цикл Activity на любом нетривиальном примере.
- Привести пример использования альтернативного ресурса (тип ресурса согласно варианту).
- Для заданного набора альтернативных ресурсов, предоставляемых приложением, и заданной конфигурации устройства (оба параметра согласно варианту) 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.
- Студент написал приложение: continuewatch. Это приложение по заданию должно считать, сколько секунд пользователь провел в этом приложении.
    - Найдите и опишите все ошибки в этом приложении, которые можете найти.
    - Исправьте неправильный подсчет времени в приложении ContinueWatch с использованием onSaveInstanceState /onRestoreInstanceState.
    - Исправьте неправильный подсчет времени в приложении ContinueWatch с использованием Activity Lifecycle callbacks и Shared Preferences.
    - Продемонстрируйте, что приложения из 4.2 и 4.3 имеют разное поведение. Объясните поведение в каждом случае.

Жизненный цикл Activity

Андройд приложение состоит из компонентов, которые операционная система может запускать независимо друг от друга. Единственным компонентом, у которого есть пользовательский интерфейс (UI), является Activity.

Во время того, как пользователь взаимодействует с приложением (открывает его, закрывает, переходит по нему и т.д.) операционная система переводит Activity в какое-либо состояние. И чтобы при перевороте экрана наше приложение не вылетало и не происходили вещи, которые нежелательны для пользователя, нужно позаботиться об обработке событий жизненного цикла.

Все Activity расположены в стеке. Когда операционная система переходит в новый Activity, то он кладётся наверх стека. А когда нам нужно выйти из приложения, то оно убирается из стека.

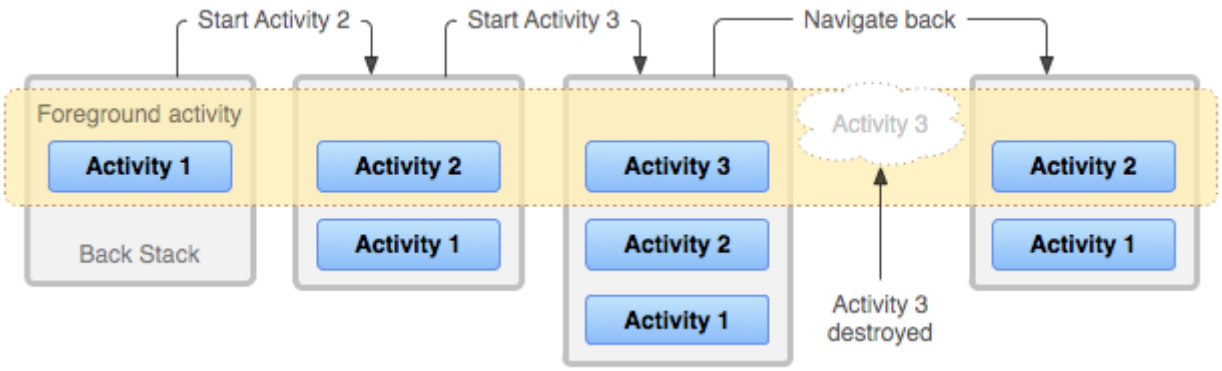

Рис. 1 Back stack

Для отслеживания перемещения по жизненному циклу класс Activity предоставляет шесть функций обратного вызова. Система вызывает каждый из них при переходе Activity в новое состояние.

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() – когда Activity входит в состояние Started система вызывает эту функцию обратного вызова. Данный вызов подготавливает Activity для видимости пользователю. После выполнения данного вызова Activity переходит в состояние Resumed и вызывает onResume().

onResume() – вызов при переход в состояние Resumed. Activity отображается на экране устройства, а пользователь может с ним взаимодействовать.

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, которые вызывается когда система решает убить Activity или при вызове метода finish().

Также есть дополнительный вызов onRestart() – вызывается после onStop(), когда текущий Activity должен быть снова отображен пользователю. После него следуют onStart() и onResume(). В продвинутых приложениях является хорошим знаком, что этот метод не вызывается.

На рисунке 2 изображена иллюстрация всех этих методов.

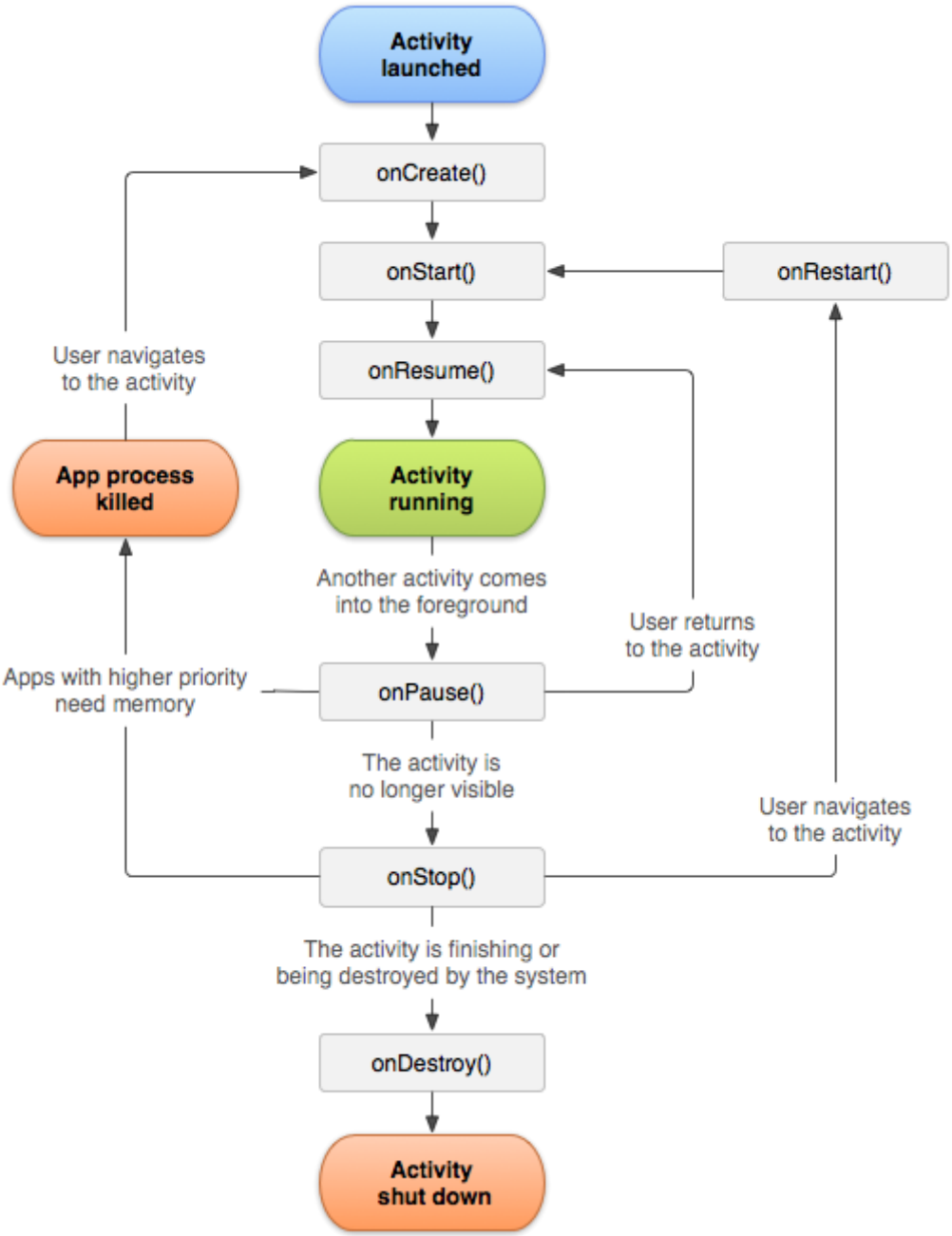

Рис. 2 Жизненный цикл Activity

Задание 1

Для выполнения данного задания было создано простое Андройд приложение. В нашем Activity были перегружены методы обратных вызовов для добавления логирования. Использовался инструмент LogCat. Запустим программу на физическом устройстве.

Как и ожидалось при запуске приложения вызвались следующие методы:

```
25423-25423/com.wooftown.loglifecycle D/MainActivity LC: onCreate()
25423-25423/com.wooftown.loglifecycle D/MainActivity LC: onStart()
25423-25423/com.wooftown.loglifecycle D/MainActivity LC: onResume()
```

Позвоним на телефон и посмотрим, что произошло:

```
25423-25423/com.wooftown.loglifecycle D/MainActivity LC: onPause()
25423-25423/com.wooftown.loglifecycle D/MainActivity LC: onStop()
// Сбросил звонок
25423-25423/com.wooftown.loglifecycle D/MainActivity LC: onRestart()
25423-25423/com.wooftown.loglifecycle D/MainActivity LC: onStart()
25423-25423/com.wooftown.loglifecycle D/MainActivity LC: onResume()
```

Можно подтвердить вывод, что, когда Activity не отображается на экране приложение переходит в состояние Stopped. А когда мы возвращаемся в него, то мы проходим через состояния Started и попадаем обратно в Resumed. В нашем случае звонок перекрыл наше Activity.

При получении уведомлений (строка сверху экрана) Activity никак на это не реагировало.

Когда я делал скриншот приложения, Activity переходило в состояние Stopped. Т. к. его часть была закрыта анимацией съемки.

Так же при создании приложения были проверены возможности уведомлений посредствам Toast и Snackbar. Были выявлены главные их отличия:

Toast	Snackbar
Не привязан к activity	Привязан к activity из которого вызван
Не может быть закрыт	Может быть закрыт пользователем
Показывается определённое время	Может висеть бесконечно

Альтернативные ресурсы

При разработке Андройд приложений возникает проблема : существует огромное количество различных конфигураций устройств (от наручных часов, до автомобильных бортовых компьютеров), также нам хочется, чтобы язык в приложении подстраивался под систему или, например некоторые действия можно было бы сделать только в определённом регионе. Для решения этой проблемы были созданы альтернативные ресурсы. Они позволяют выбрать те ресурсы, которые созданы для какой-либо характеристики устройства.

Задание 2

К примеру, соотношение сторон экрана.

Пусть у нас есть приложение со списком продуктов. Их список образует два столбца. Тогда на устройствах с соотношением 9:16 все будет выглядеть хорошо, но, например, на планшетах с соотношением 16:9 эти два столбца будут выглядеть нелепо и будет правильно сделать их количество больше.

Или, к примеру, у нас есть игра. Один из ресурсов представляет собой игровое поле (неважно png или svg). И при изменении соотношения сторон не очень хотелось бы, чтобы поле было растянуто. Тогда есть смысл создать несколько ресурсов, чтобы всё выглядело красиво и не сжато/растянуто.

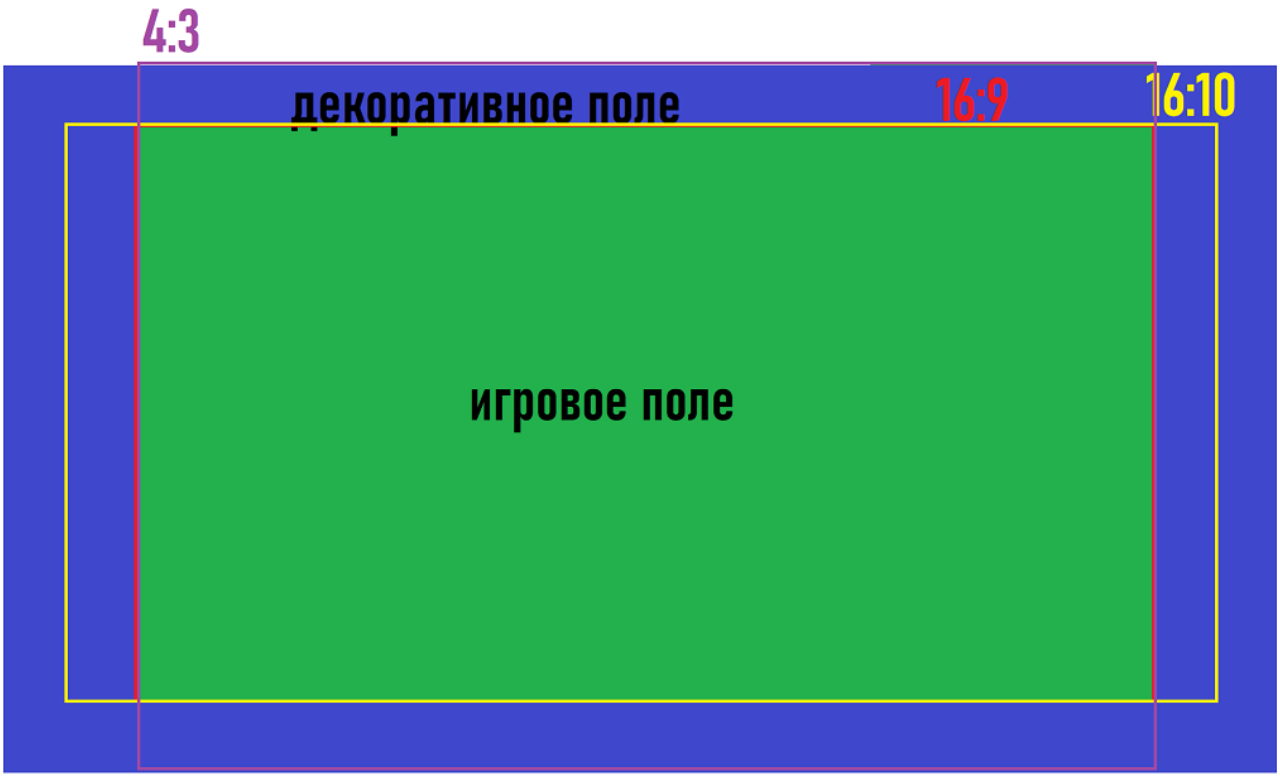

Рис. 3 Пример зависимости поля от соотношения сторон

На примере изображена иллюстрация с соотношениями сторон экрана. Можно создать один фон, зеленая часть игровая, а синяя декоративная. И в зависимости от разрешения будем выводить разную часть поля. Тогда проблемы со сжатием фона не будет и у любого девайса будет доступ к каждой точке поля.

Задание 3

Выбрать ресурс по алгоритму для best-matching resource.

==========

Вариант 6:

==========

Конфигурация устройства:
LOCALE_LANG: fr
LOCALE_REGION: rFR
SCREEN_SIZE: small
SCREEN_ASPECT: long
ROUND_SCREEN: round
ORIENTATION: port
UI_MODE: vrheadset
NIGHT_MODE: notnight
PIXEL_DENSITY: mdpi
TOUCH: notouch
PRIMARY_INPUT: qwerty
NAV_KEYS: wheel
PLATFORM_VER: v25

Конфигурация ресурсов:
(default)
rFR-notround
rFR-notnight
xhdpi-qwerty-v25
fr-rFR-small-mdpi-nonav
rUS-wheel-v25
notlong
round-notnight-nonav
fr-finger-12key-v25
notround-port-night
port-watch-xxxhdpi-finger-qwerty

Для начала выбросим все конфигурации ресурсов противоречащие конфигурации устройства.

(default)
~~rFR-notround~~ (ROUND_SCREEN: round)
rFR-notnight
xhdpi-qwerty-v25 (можем оставить из-за исключения для PD)
~~fr-rFR-small-mdpi-nonav~~ (NAV_KEYS: wheel)

~~rUS-wheel-v25~~ (LOCALE_REGION: rFR)

~~notlong~~ (SCREEN_ASPECT: long)

~~round-notnight-nonav~~ (NAV_KEYS: wheel)

~~fr-finger-12key-v25~~ (TOUCH: notouch)

~~notround-port-night~~ (ROUND_SCREEN: round)

~~port-watch-xxxhdpi-finger-qwerty~~ (UI_MODE: vrheadset)

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

(default)

rFR-notnight

xhdpi-qwerty-v25

Нету ни одного совпадения, поэтому мы ничего не исключаем. Далее “Language”(в контексте данного задания язык и регион никак не связаны между собой). Т. к. нету совпадений, то продолжаем.

“Region”:

~~(default)~~

rFR-notnight

~~xhdpi-qwerty-v25~~

Будет выбран ресурс для французского региона с незатемнённым режимом (у меня в телефоне это называется так).

Задание 4.1

Необходимо провести тестирование приложения continuewatch. Это было выполнено на физическом устройстве.
В ходе тестирования выявлены следующие ошибки:

- При выходе из приложения секунды всё равно считаются.
- При смене ориентации положения экрана счётчик сбрасывается.
- В начале вместо “Second elapsed” написано “Hello, world”, а если повернуть, то “TextView”.
- В альбомной ориентации текст съехал в угол экрана.

Исправим приложение. Для начала разберёмся с подсчётом только когда приложение на экране. Для этого добавим булеву, которая будет указывать, что приложение прямо сейчас отображается на экране. Нам подойдёт метод обратного вызова onResume() и onStop() для её изменения. Так же была изменена вёрстка приложения.

P.S. Я считаю, что это часть работы нужна для лучшего понятия жизненного цикла Activity и получения навыков сохранения его состояния. Поэтому код был изменён минимально, без внесения различных view/data bindings или корутин.

Задание 4.2

Теперь в рамках одного Activity приложение работает корректно, но существует та же проблема переворота экрана (когда Activity разрушается). Используем перегрузку методов onSaveInstanceState /onRestoreInstanceState.

Листинг 1 onSaveInstanceState /onRestoreInstanceState

```
override fun onSaveInstanceState(outState: Bundle) {
    outState.run {
        Log.d(TAG,"Saving state SEC=$secondsElapsed")
        putInt(SEC,secondsElapsed)
    }
    super.onSaveInstanceState(outState)
}

override fun onRestoreInstanceState(savedInstanceState: Bundle) {
    super.onRestoreInstanceState(savedInstanceState)
    savedInstanceState.run {
        secondsElapsed = getInt(SEC)
        Log.d(TAG,"Restore state SEC=$secondsElapsed")
    }
}
```

Метод onSaveInstanceState вызывается, когда операционная система разрушает наше Activity с определёнными целями. Например, изменение положения ориентации экрана. Если пользователь просто закроет приложение или будет вызван finish(), то метод onSaveInstanceState вызван не будет.

Записываем в Bundle outState количество секунд. Bundle представляет собой ассоциативный массив ключ (String) – значение (Parcelable).

Метод onRestoreInstanceState вызывается, когда при запуске приложения в методе onCreate() аргумент типа Bundle не равен null(когда он равен null, то создаётся новая сущность Activity) то происходит восстановления при помощи переданного Bundle. При создании Activity будем забирать из словаря количество секунд.

Данный подход позволил нам реализовать приложение так, как оно было задумано. При завершении приложения при следующем заходе счётчик сбрасывался в 0.

Задание 4.3

Эту же задачу можно решить с помощью “общих настроек”. В документации написано : “Если у вас относительно маленькая коллекция ключ-значение, которую вы хотите сохранить, вам следует воспользоваться SharedPreferences API”

Листинг 2 Использование SharedPreference

```
private lateinit var sharedPref : SharedPreferences

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

}
override fun onStop() {
    super.onStop()
    visibility = false
    with(sharedPref.edit()){
        Log.d(TAG,"put $secondsElapsed to SEC of sharedPref")
        putInt(SEC,secondsElapsed)
        apply()
    }
    Log.d(TAG,"visibility == FALSE")

}
override fun onResume() {
    super.onResume()
    visibility = true
    secondsElapsed = sharedPref.getInt(SEC,0)
    Log.d(TAG,"get $secondsElapsed from SEC of sharedPref")
    Log.d(TAG,"visibility == TRUE")

}
```

В начале работы приложения создаём новый файл с помощью метода getSharedPreferences (String name, int mode). С помощью него мы можем получить файл настроек ‘name’ и объект SharedPreferenced с помощью которого мы можем извлекать и изменять его значения. Любому вызвавшего getSharedPreferences с одинаковым именем будет возвращён один и тот же экземпляр и изменения будут видны у всех. Отсюда и Shared в названии.

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

Данный подход позволил нам реализовать приложение так, как оно было задумано. При завершении приложения при следующем заходе счётчик не сбрасывался в 0.

Задание 4.4

Поведения двух предложенных программ разное.

Приложение 4.2 использует объект Bundle. Он содержит состояние Activity, если оно было сохранено. И например, при перевороте экрана, когда прежнее Activity разрушает в новое передаётся наш Bundle и мы можем восстановить число секунд.

Приложение 4.3 использует общие настройки. По сути, это просто файл со словарём. Из-за этого, мы можем сохранять состояние таймера и при следующем заходе приложение число секунд будет прежним.

Выводы

Все проекты находятся в репозитории: https://github.com/wooftown/spbstu-android

В ходе данной лабораторной работы был изучен жизненный цикл компонента Activity. Понимание его работы очень важен при разработке Android приложений. Были получены навыки работы с callback-методами и применены на учебных примерах.

Была выполнена работа по сохранению состояния Activity. Для этого взялись самые тривиальные способы. При помощи Bundle мы восстанавливали состояние по переданному объекту Bundle. А в случае SharedPreferences мы работали с файлом. Хоть с первого взгляда можно сказать, что эти варианты полностью идентичны, на практике это оказалось не так. Я считаю, что при разработке более сложных приложений будет полезным использовать эти два подхода вместе.

Проведено изучение альтернативных ресурсов. На примере показана их важность. Если бы такой концепции не было, то программирование могло превратиться в мешанину операторов выбора. Также было выполнено упражнение по best-matching resource для понимая алгоритма выбора лучшего ресурса.

Список источников

- https://developer.android.com
- https://github.com/andrei-kuznetsov/android-lectures

Время на выполнение работы

- 1 – 45 мин
- 2 – 20 мин
- 3 – 10 мин
- 4 – 150 мин
- Отчёт – 60 мин (+параллельно заполнял, вставляя сначала рисунки и листинги)