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

Лабораторная работа № 2

Дисциплина: Проектирование мобильных приложений

Тема: Жизненный цикл приложения, альтернативные ресурсы

Выполнил студент гр. 3530901/90201 _____С.А. Федоров
(подпись)

Принял старший преподаватель _____А.Н. Кузнецов
(подпись)

“____” _____ 2021 г.

Санкт-Петербург
2021

Оглавление

Цели

- Познакомиться 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 согласно индивидуальному варианту
- Для заданного набора альтернативных ресурсов, предоставляемых приложением, и заданной конфигурацией устройства (оба параметра согласно варианту) объяснить, какой ресурс будет выбран в конечном итоге. Объяснение должно включать пошаговое исполнением 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.
    - Найти и описать все ошибки в этом приложении, которые можете найти
    - Исправить неправильный подсчет времени в приложении Continuewatch с использованием onSaveInstaceState /onRestoreInstanceState
    - Исправить неправильный подсчет времени в приложении Continuewatch с использованием Activity Lifecycle callbacks и Shared Preferences
    - Продемонстрировать, что приложения из 4.2 и 4.3имею разное поведение. Объяснить поведение в каждом случае

Жизненный цикл Activity

Activity – один из основных компонентов Android-приложения, который представляет собой схему представления Android-приложений. Например, экран, который видит пользователь. Android-приложение может иметь несколько Activity и может переключаться между ними во время выполнения приложения.

Для управления переходами между Activity все приложения Android имеют строго определенную систему жизненного цикла. При запуске пользователем приложения система дает этому приложению высокий приоритет. Каждое приложение запускается в виде отдельного процесса, что позволяет системе давать одним процессам более высокой приоритет, в отличие от других. Благодаря этому, например, при работе с одними приложениями Android позволяет не блокировать входящие звонки. После прекращения работы с приложением, система освобождает все связанные ресурсы и переводит приложение в разряд низкоприоритетного и закрывает его.

Все объекты Activity, которые есть в приложении, управляются системой в виде стека Activity, который называется **back stack**. При запуске новой Activity она помещается поверх стека и выводится на экран устройства, пока не появится новая Activity. Когда текущая Activity заканчивает свою работу (например, пользователь уходит из приложения), то она удаляется из стека, и возобновляет работу та Activity, которая ранее была второй в стеке.

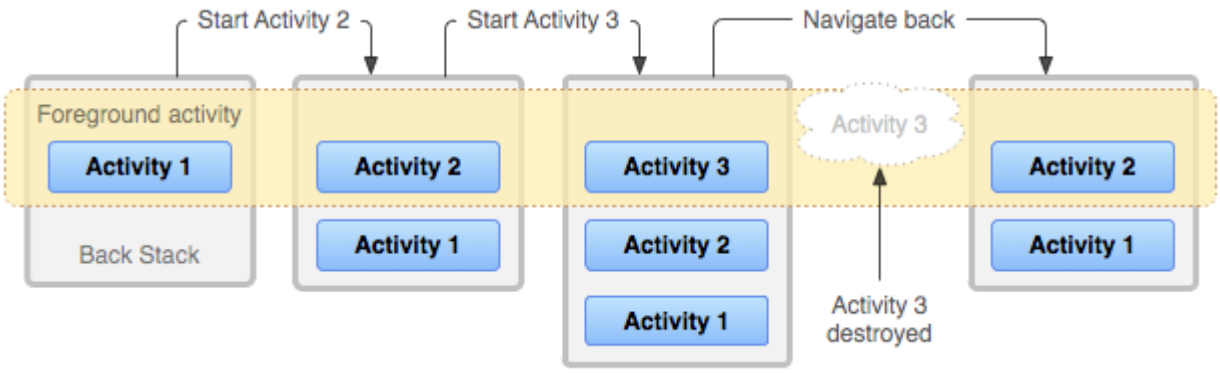


Рис. 1 back stack

После запуска Activity проходит через ряд событий, которые обрабатываются системой и для обработки которых существует шесть функций обратных вызовов:

- onCreate() – первый метод, с которого начинается выполнение Activity. В этом методе Activity переходит в состояние Created. Этот метод выполняется только один раз и обязательно должен быть определен в классе Activity: первоначальная настройки Activity, восстановление предыдущего состояния через Bundle (объект, который содержит прежнее состояние activity, если оно сохранено; null – если activity создается заново), инициализация данных для приложения. После выполнения переводит Activity в состояние Started.
- onStart() – метод, который вызывается в состоянии Started, в котором осуществляется подготовка к выводу Activity на экран устройства (видимости пользователю). После завершения работы данного метода вызывается метод onResume, а Activity переходит в состояние Resumed.
- onResume() – метод, который вызывается в состоянии Resumed. Здесь Activity отображается на экране устройства, и пользователь может с ней взаимодействовать.
- onPause() – метод, который вызывается в состоянии Paused. В этом методе освобождаются используемые ресурсы, приостанавливаются процессы, чтобы снизить нагрузку на производительность системы. Здесь надо учитывать, что Activity по-прежнему остается видимой для пользователя и времени на данный метод отводится очень мало, поэтому не стоит выполнять долгие операции или что-то хранить. Такие действия лучше выполнять в следующем методе.
- onStop() – метод, который вызывается в состоянии Stopped. В этом состоянии Activity полностью невидима. Здесь следует освобождать используемые ресурсы, которые не требуются пользователю, когда он

не взаимодействует с Activity. Теперь можно или вернуться в Activity (система вызовет метод onRestart) или завершить ее работу (метод onDestroy).

- onDestroy() – метод, который вызывается в том случае, если система решит убить Activity, либо при вызове метода finish().

На Рис.2 представлена структура жизненного цикла Activity со всеми вышеописанными методами.

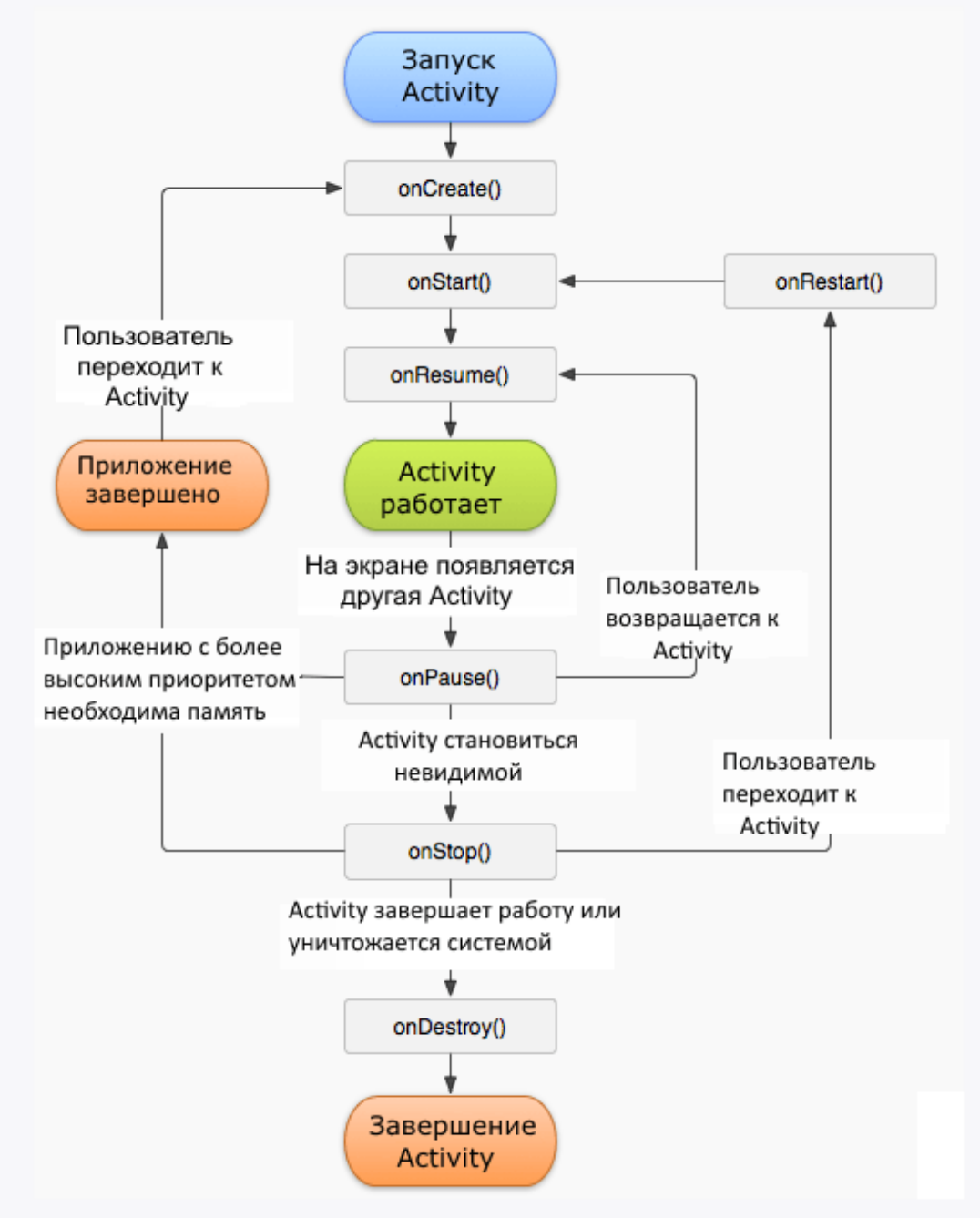


Рис. 2 Жизненный цикл Activity

Задача 1

По заданию требуется продемонстрировать жизненный цикл Activity на любом нетривиальном примере, для этого было создано простое Android приложение, в котором по нажатию кнопки изменяется текст в виджете TextView.

При создании данного приложения использовалась функция ViewBinding, чтобы создать класс привязки для каждого файла макета XML. Для демонстрации жизненного цикла были переопределены методы обратных вызовов с добавлением в них логирования при помощи инструмента LogCat.

- Запустим программу на физическом устройстве (эмуляторе) и посмотрим на результаты логирования.

```
8345-8345/com.example.lifecycle  D/MainActivity  LifeCycle: onCreate()
8345-8345/com.example.lifecycle  D/MainActivity  LifeCycle: onStart()
8345-8345/com.example.lifecycle  D/MainActivity  LifeCycle: onResume()
```

Как и ожидалось при запуске вызвались методы onCreate(), onStart() и onResume().

- Теперь попробуем позвонить на наш телефон.

```
8345-8345/com.example.lifecycle  D/MainActivity  LifeCycle: onPause()
8345-8345/com.example.lifecycle  D/MainActivity  LifeCycle: onStop()
// Здесь звонок был завершен
8345-8345/com.example.lifecycle  D/MainActivity  LifeCycle: onRestart()
8345-8345/com.example.lifecycle  D/MainActivity  LifeCycle: onStart()
8345-8345/com.example.lifecycle  D/MainActivity  LifeCycle: onResume()
```

Во время ответа на звонок приложение перешло в состояние Paused, позже на эмуляторе было открыто окно звонка, поэтому приложение перешло в состояние Stopped, так как было перекрыто наше Activity. Когда звонок завершился и окно приложения заново было открыто, произошел переход в состояние Restarted, а оттуда через Started в Resume.

- При отправке сообщения на телефон на устройстве высвечивалось данное сообщение, но Activity никак не реагировало
- При сворачивании приложения (когда открываем меню всех открытых приложений) Activity переходило в состояния Stopped.

Альтернативные ресурсы

При создании Android приложений разработчики сталкиваются с проблемой поддержки различных конфигураций устройств (телефоны, планшеты, часы, бортовые компьютеры и т.д.). Требуется сделать так, что бы на каждом устройстве язык в приложении подстраивался под определенную систему.

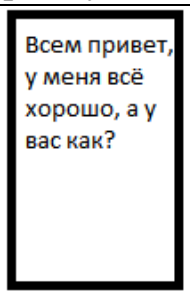
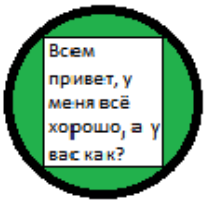
Для решения данной задачи были созданы альтернативные ресурсы, которые позволяют выбрать и загрузить именно те ресурсы, которые предусмотрены для данной конфигурации устройства.

Задача 2

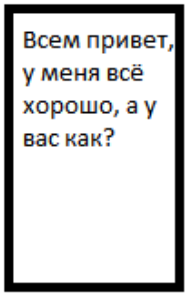
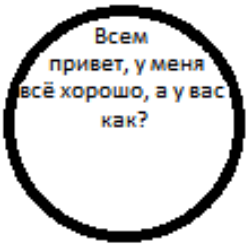
В данной задаче требуется привести пример использования альтернативного ресурса согласно индивидуальному варианту. Для 22 варианта (7 варианта) это Round screen (круглый экран).

Допустим, что нас есть часы, у которых круглый экран. Теперь представим ситуацию, что мы хотим открыть как-нибудь текстовый документ на этих часах в соответствующем приложении для чтения. Если это приложение не поддерживает альтернативный ресурс Round Screen, то часть текста окажется за пределами экрана, и мы не сможем полностью ознакомиться с документом. В случае использования Round Screen весь текст подстроится под наш экран и выберет один из двух видов представления текста:

- Выделение прямоугольной области на экране и отображение исходного текста в прямоугольном формате с уменьшением шрифта

Текст на прямоугольном экране	Текст на круглом экране
	 Зеленая область – незадействованная часть для вывода текста

- Перенос слов в том случае, если их содержание заходит за границы нашего устройства и использование выравнивания текста по центру

Текст на прямоугольном экране	Текст на круглом экране
	

Рассмотрим еще один пример использования Round screen. Сегодня практически у всех есть собственный мобильный телефон, с которым мы стараемся вести себя аккуратно, чтобы не сломать его или же разбить. Для этого многие используют защитные пленки на экран или же специальные чехлы. Представим ситуацию, что у вас есть такой чехол и вам срочно надо посмотреть время, а чехол по каким-либо причинам открыть не можете, чтобы вам был доступен целый экран. Заранее вы предусмотрели такую ситуацию и поставили приложение по отображению времени на экране в определенном месте и купили чехол с круглым вырезом. Но вы не учли того, что это приложение не поддерживает Round screen и ваше время отображается в прямоугольном формате.

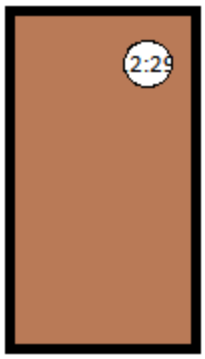


Рис. 2-1 Приложение без Round screen

Если бы у вас было приложение с поддержкой Round screen, то вы бы узнали точное время

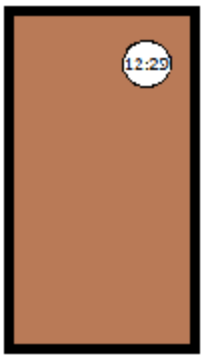


Рис. 2-2 Приложение с Round screen

Задача 3

По заданию требуется выбрать ресурс по алгоритму best-matching resource

```
==================================
Вариант 22:
==================================
Конфигурация устройства:
LOCALE_LANG: en
LOCALE_REGION: rUS
SCREEN_SIZE: small
SCREEN_ASPECT: notlong
ROUND_SCREEN: notround
ORIENTATION: land
UI_MODE: watch
NIGHT_MODE: night
PIXEL_DENSITY: tvdpi
TOUCH: finger
PRIMARY_INPUT: nokeys
NAV_KEYS: wheel
PLATFORM_VER: v27

Конфигурация ресурсов:
(default)
xlarge-long-12key
round-notouch-qwerty
round
rCA
large-round-vrheadset-night-finger-v27
en-notnight-ldpi-trackball
notlong-night-qwerty
port-desk-dpad-v26
fr-small-finger
en-large-qwerty-v26
```

1. По алгоритму best-matching отбросить файлы ресурсов, противоречащие конфигурации устройства

default)	
~~xlarge-long-12key~~	SCREEN_SIZE: small
~~round-notouch-qwerty~~	ROUND_SCREEN: notround
~~round~~	ROUND_SCREEN: notround
~~rCA~~	LOCALE_REGION: rUS
~~large-round-vrheadset-night-finger-v27~~	SCREEN_SIZE: small
~~en-notnight-ldpi-trackball~~	NIGHT_MODE: night
notlong-night-qwerty	
~~port-desk-dpad-v26~~	ORIENTATION: land
~~fr-small-finger~~	LOCALE_LANG: en
~~en-large-qwerty-v26~~	SCREEN_SIZE: small

2. Теперь воспользуемся таблицей квалификаторов альтернативных ресурсов “MCC and MNC”

default)	
notlong-night-qwerty	

3.1. “Language and region” отсутствует у обоих, поэтому ничего не делаем и идем к следующему конфигуратору.
3.2. “Layout Direction” тоже отсутствует
3.3. “smallestWidth” тоже отсутствует
3.4. “Available width” тоже отсутствует
3.5. “Available height” тоже отсутствует
3.6. “Screen size” тоже отсутствует
3.7. “Screen aspect” присутствует у второго (**notlong**-night-qwerty), поэтому переходим к 4 пункту

Ничего не исключаем, так как нет ни одного совпадения и сразу переходим к 4ому пункту.

4. Удалим каталоги, у которых нет аспекта экрана (Screen aspect)

~~default)~~	
notlong-night-qwerty	

5. Так как остался один каталог, то завершаем алгоритм.

По итогу был выбран ресурс notlong-night-qwerty. Это ресурс с не длинным аспектом экрана, с ночным режимом и основным методом ввода текста при помощи аппаратной клавиатуры.

Задача 4_1

По заданию требуется провести ручное тестирование приложения continuewatch на устройстве или эмуляторе, а также выявить ошибки.

В ходе тестирования были найдены следующие ошибки:

- При сворачивании приложения (приложение уже не отображается на экране) подсчет секунд продолжался, когда по заданию счетчик должен был остановиться
- При повороте телефона на альбомную ориентацию текст отображался не там, где его задали в XML файле (в XML файле по центру, а текст отображается в левом верхнем углу)
- При повороте экрана счетчик сбрасывался
- В начале работы приложения на экране отобразилась надпись “Hello World!”, а должна была появиться надпись “Second elapsed”.
- При работе в альбомной раскладке высветилась надпись “TextView”, вместо “Second elapsed”.

Дополнительная ошибка в написании XML файла:

Для альбомной раскладки (activity_main_land.xml) в XML файле использовался вид компоновки ConstraintLayout, при использовании которого требуется указывать хотя бы 2 привязки, а в XML файле не было указано ни одной.

Задача 4_2

По заданию требуется исправить неправильный подсчет времени в приложении Continuewatch с использованием onSaveInstaceState /onRestoreInstanceState.

Для начала следует устранить проблему подсчета времени в том случае, когда приложение не отображается на экране. Для этого:

- Введем булеву переменную *visibility* со значением false, а в цикле подсчет времени поставим условие на if (visibility == true)
- В предыдущих пунктах изучался жизненный цикл Activity и теперь мы знаем, что:
    - при запуске приложения Activity проходит через состояния Created, Started и Resumed
    - существует метод onStop(), который вызывается в состоянии Activity Stopped, когда Activity полностью невидима, то есть приложения свернуто
    - после возвращения в приложение Activity опять проходит через состояние Resumed
- Исходя из предыдущего пункта для решения проблемы следует переопределить метод onResume(), где мы будем visibility присваивать значение true, чтобы при открытом приложении велся подсчет. Также переопределим метод onStop(), где visibility = false, чтобы при свернутом приложении подсчет останавливался

Листинг 1. onResume/onStop

```
override fun onStop() {
    super.onStop()
    visibility = false
    Log.d(TAG,"visibility == FALSE")

}
```

```
override fun onResume() {
    super.onResume()
    visibility = true
    Log.d(TAG,"visibility == TRUE")

}
```

Теперь можно перейти к решению проблемы разрушения Activity при повороте экрана (счетчик сбрасывается). Для этого переопределим методы onSaveInstaceState() и onRestoreInstanceState().

Листинг 2. onSaveInstaceState / onRestoreInstanceState

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

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

companion object{
    const val TAG = "ContinueWatch"
    const val SEC = "SecondsElapsed"
}
```

Метод onSaveInstanceState вызывается тогда, когда приложение останавливает свою работу и Activity разрушается, поэтому требуется сохранить временную информацию о состоянии иерархии представлений Activity. Например, когда надо изменить ориентацию экрана. Данный метод не будет вызван, если приложение будет закрыто или же будет вызван метод finish().

Для сохранения дополнительной информации о состоянии экземпляра требуется добавить в метод onSaveInstanceState пары ключ-значение к Bundle объекту, который сохраняется даже в том случае, если Activity будет

случайно уничтожено. Bundle – это ассоциативный массив, где ключ (String), а значение (Parcelable).

Метод onRestoreInstanceState вызывается тогда, когда Activity воссоздается (когда в методе onCreate() значение Bundle не равно null) после того, как оно ранее было уничтожено. Восстановление происходит при помощи переданного Bundle.

В приложении Continuewatch в onSaveInstanceState() мы запоминаем количество секунд в переменной SEC, а при восстановлении в методе onRestoreInstanceState() передаем это значение.

Теперь приложение работает корректно (по заданию). Счетчик не сбрасывается при повороте экрана, при сворачивании приложения счётчик останавливается. При закрытии приложения счетчик сбрасывается.

Задача 4_3

По заданию требуется решить ту же самую проблему с подсчетом времени, но уже другим способом. Здесь требуется использовать объект обратные вызовы Activity LifeCycle и общих настроек. В таком случае, если у нас относительно небольшая коллекция пар “ключ-значение”, которую требуется сохранить, следует использовать SharedPreferences API.

Листинг 3 Использование объекта SharedPreferences

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

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

}

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

}
```

Метод getSharedPrefernces() используется, когда требуется несколько общих файлов настроек, изменения в которых будут видны всем. (shared - общий)

В методе onCreate() следует создать новый файл при помощи метода getSharedPreferences с указанием двух параметров (String – имя, int - модификация).В методе onStop() при помощи edit() запоминаем количество секунд. В методе onResume() при помощи getInt получаем запомненное значение количества секунд. Теперь приложение работает корректно.

Задача 4_4

Приложение 4.2 использует объект Bundle, который содержит состояние Activity, если оно было сохранено. При решении задачи 4.2 во время сворачивания экрана Activity разрушалось. Когда мы обратно возвращались в приложение, Activity создавалось с параметром Bundle, где хранилось количество секунд.

Приложение 4.3 использует общие настройки. Здесь использовала файл-словарь с общим доступом, куда можно записать значение количества секунд, а потом извлечь это значение в методе при возвращении в приложение (метод onResume()), так как Activity пройдет через состояние Resumed.

Выводы

- В ходе выполнения данной лабораторной работы был изучен жизненный цикл Activity
- Были изучены методы обратных вызовов
- Были рассмотрены альтернативные ресурсы и приведен пример их востребованности на примере ресурса Round Screen. Также при помощи best-matching resources алгоритма была решена задача по выбору лучшего ресурса
- Было выполнено задание по устранение ошибок подсчета времени в готовом приложении Continuewatch. Здесь требовалось сохранить состояние Activity двумя способами:
    - Использование объекта Bundle для восстановления данных. Для этого были переопределены методы onSaveInstaceState() и onRestoreInstanceState()
    - Использование общих настроек, а именно SharedPreferences, где была произведена работа с файлом

Ссылка на github: https://github.com/sergeyfedorov02/Android_labs.git

Список источников

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

Время выполнения лабораторной работы

- Задача 1 – 60 минут
- Задача 2 – 25 минут
- Задача 3 – 25 минут
- Задача 4 – 150 минут
- Заполнение отчета – 260 минут (отчет заполнялся по мере решения задач, поэтому время решения задачи = отчет + написание кода)