МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ

КАЛИНИНГРАДСКОЙ ОБЛАСТИ

ПРОФЕССИОНАЛЬНАЯ ОБРАЗОВАТЕЛЬНАЯ ОРГАНИЗАЦИЯ

«КОЛЛЕДЖ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И СТРОИТЕЛЬСТВА»

(ГБУ КО ПОО «КИТиС»)

|  |  |  |
| --- | --- | --- |
| Курсовой проект  допущен к защите  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (дата)  Зам. директора по УМР  \_\_\_\_\_\_\_\_\_\_\_\_\_Павленко Г.Я.  (подпись) |  | Курсовой проект  защищен с оценкой  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (оценка)  Руководитель работы  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись, дата) |

Пояснительная записка к курсовому проекту

по дисциплине: МДК 01.01 Разработка программных модулей

Тема: Разработка мобильного приложения для фокусировки внимания

Специальность 09.02.07 «Информационные системы и программирование»

|  |  |
| --- | --- |
| Выполнила: | студентка 3 курса,  группы ИСп 19-2к  Чекалова Елена Николаевна  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |
| Руководитель: | Большакова-Стрекалова Анна Викторовна, преподаватель ГБУ КО ПОО КИТиС  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |

Калининград

2022

Содержание

|  |  |
| --- | --- |
| Введение | 3 |
| 1. Описание предметной области | 4 |
| * 1. Аналоги разрабатываемого приложения | 4 |
| * 1. Техническое задание | 5 |
| * 1. Описание структуры приложения | 6 |
| 1. Описание разработки приложения | 8 |
| * 1. Обоснование средств разработки | 8 |
| * 1. Разработка пользовательского интерфейса | 11 |
| * 1. Разработка логики работы приложения. Схема взаимодействия компонентов проекта | 14 |
| * 1. Описание переменных, компонентов, классов и подпрограмм | 16 |
| Заключение | 22 |
| Приложение А. Листинг рабочего кода | 21 |
| Приложение Б. Руководство пользователю | 31 |

Введение

Метод Помидора в тайм-менеджменте - простая, но действенная техника, позволяющая работать эффективнее, быть производительнее, быстрее приступать к выполнению задач.

Суть метода заключается в следующем:

Включается таймер на первые 25 минут, начинается активная работа.

По истечении «помидорки» предполагается отдых: можно сделать гимнастику, подышать свежим воздухом, налить кофе, полить цветы.

По истечении четырех «помидорок» делается большой перерыв, который длится примерно 15-20 минут (иногда данный промежуток времени можно увеличить до 30-40 минут).

В данной работе была проведена разработка приложения электронного таймера, работающего по данному методу. В результате разработки схемам был построен макет устройства и проведены его испытания, подтвердившие работоспособность схемы.

Целью курсовой работы является приобретение практических навыков в области:

1. разработки:
   * Структурной и функциональной схем приложения;
   * Алгоритмов и реализующих их программ;
2. составления технического задания, содержащей обоснование принятых проектных решений;
3. При разработке программного средства следует учитывать следующие требования:

* Стандартизацию основных этапов жизненного цикла программных средств;
* Обеспечение надежности и качества функционирования программного средства.

Описание предметной области

* 1. Аналоги разрабатываемого приложения

Для обоснования актуальности и практической значимости разработанного приложения необходимо привести сравнительный анализ с программами аналогами. В целом, они схожи, так как приложения подобного типа осуществляют одинаковую деятельность.

Pomodoro Timer

Приложение содержит в себе набор функций, таких как создание списка задач, 25-минутный таймер, возможность настройки тиканья

В качестве недостатков данного приложения можно отметить то, что оно не предоставляет статистику, и не предоставляет возможность работы при выключенном дисплее. Для наглядности составим таблицу:

Таблица 1 - Плюсы и минусы Pomodoro Timer

|  |  |
| --- | --- |
| + | - |
| Возможность создания списка задач | Не предоставляет статистику |
| Возможность настройки тиканья | Не предоставляет возможность работы при выключенном дисплее |
| Поддержка нескольких языков | Есть реклама |
| Возможность автоматического старта следующего таймера |  |

Pomodoro Smart Timer

При анализе были выявлены следующие недостатки: большое количество бесполезных функций, реклама.

Таблица 2 - Плюсы и минусы Pomodoro Smart Timer

|  |  |
| --- | --- |
| **+** | **-** |
| Статистика количества и времени | Нет возможности завершить сеанс досрочно |

Таблица 3 - Продолжение таблицы 2

|  |  |
| --- | --- |
| Пользовательский интерфейс с несколькими темами | В фоновом режиме таймер помидора останавливается и сбрасывается |
| Поддержка нескольких языков | Есть реклама |
| Много разных звуков будильника | Загроможденный интерфейс |
| Разрешить сброс статистики |  |

Work Rest

При большом функционале (уведомления, резервное копирование, выбор темы) были выявлены недостатки данного приложения такие как: большое количество ошибок (исходя из истории обновлений), отсутствие выключения фоновых звуков таймера.

Таблица 4 плюсы и минусы Work Rest

|  |  |
| --- | --- |
| + | - |
| Ежедневные уведомления, напоминания | Фоновый звук таймера |
| Облачный аккаунт с резервным копированием | Приложение бесплатно только до 3й категории |
| Различные темы оформления, темная тема | Реклама в каждом разделе |
| Подсказки, как эффективно провести, перерыв | Уведомления срабатывают не всегда |
| Поддержка нескольких языков |  |

Проведя анализ существующих информационных систем, и, исходя из существующих недостатков, было принято решение о написании своей программы.

* 1. Техническое задание

Основанием для разработки служит приказ 238/1У от 01.09.2021

Проанализировав существующие программные решения, были определены следующие функциональные требования к разрабатываемой программе:

1. программа должна иметь простой, но в то же время понятный и наглядный интерфейс, который не должен перегружать ресурсы телефона
2. Программа не должна занимать большой объем памяти
3. Работоспособность приложения в среде Android
4. Отсчитывать заданное время и оповещать пользователя об этом
5. В программе должны использоваться кнопки для ввода данных и графическое меню;
6. Программа должна содержать поле для ввода данных
7. Реализовать возможность вывода на экран таймера
8. Реализовать возможность сброса результата
9. Продолжайте считать, когда приложение работает в фоновом режиме
10. Отправить уведомление о завершении
    1. Описание структуры приложения

Пользователь, заходя в приложение должен иметь возможность выставления времени сколько длится каждый период работы и сколько их будет. На выставленное пользователем время включает таймер, начинается активная работа.

По истечении первого таймера включаемся второй, который отсчитывает время отдыха (приблизительно 5 минут). Далее если пользователь указал больше 2х раундов должен включаться 3й «рабочий» таймер. И так указанное количество пользователем раз.

По истечении четырех раундов делается большой перерыв, который длится примерно 15-20 минут.

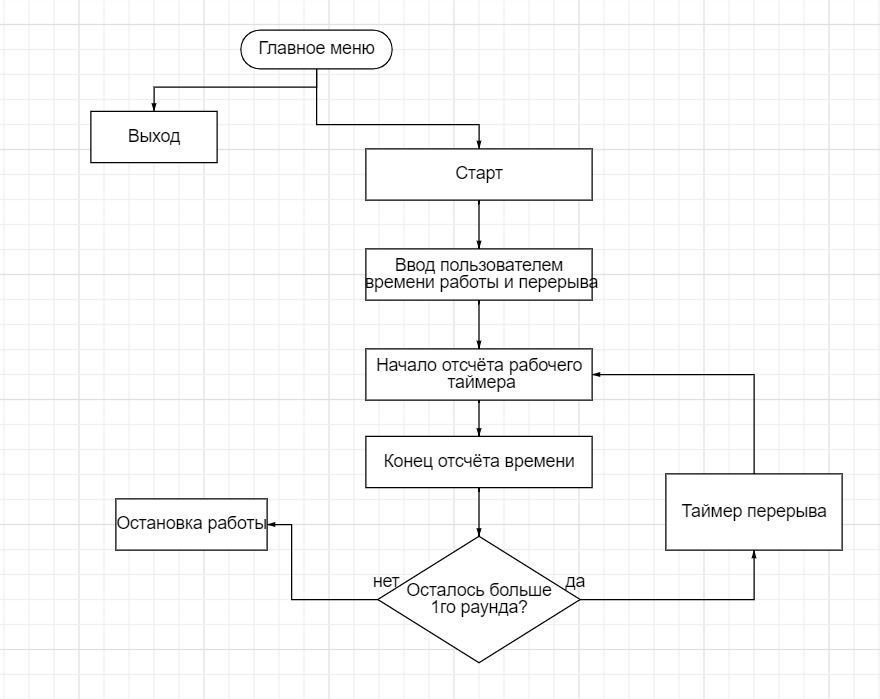


Рисунок 1 - Схема работы приложения

2. Описание разработки приложения

2.1 Обоснование средств разработки

Существует несколько причин, почему разработка приложений на Android удобна:

* **Возможность адаптировать платформу под специализированную задачу без потери поддержки**. Если разрабатывать приложение под какую-то специализированную задачу; то Android подходит для этого лучше;
* **Коммьюнити**, сообщество разработчиков под Android несколько шире, чем у iOS;
* **Для работы подойдет любой компьютер**. Неважно, на основе Windows, Linux или Mac.

У iOS-приложений есть неоспоримые плюсы: прибыль со старта, скорость разработки и более платежеспособная аудитория. Однако, большая часть мира пользуется Android, и для охвата большого количества пользователей рекомендуется разрабатывать под эту платформу. Поэтому мой выбор останавливается именно на этой платформе.Выбор того или иного языка (и вообще любого инструмента) как правило продиктован как объективными, так и субъективными факторами. В каждом проекте могут быть свои особенности, влияющие на выбор. Для написания приложения выбор стоял межу Java и Kotlin.

Kotlin – это язык программирования, разработанный компанией Jetbrains, которая знаменита IntelliJ, WebStorm, PhpStorm и плагином ReSharper для Visual Studio. Они занялись поисками нового языка программирования, который был бы лучше, чем Java, но все еще функционально совместим с ним.

Язык Java создали в компании Sun Microsystems в начале 90-х годов прошлого столетия. Этот язык появился в ответ на потребность в платформенной независимости. Java стал языком общего назначения, на котором можно писать программы под любые устройства

Чтобы разобраться, чем Kotlin отличается от Java, сравним их по нескольким критериям: количество кода, проверяемые исключения, безопасность, среда разработки и цели разработки.

Kotlin:

* Код более лаконичен, то есть можно писать и поддерживать меньшее количество кода;
* Код более безопасный, поскольку язык заставляет вас думать о (не)изменяемости, nullability и т.д. во время написания кода;

Вышесказанное также ускоряет развитие. Вам становится проще делать сложны вещи.

* Отличная поддержка IDE (будет использоваться Android Studio);
* Большое и растущее сообщество.
* Существующие библиотеки поддерживаются или мигрируют в Kotlin, интернет переполнен постами в блогах, видео, презентациями и, конечно же, вопросами/ответами
* Более продуманная и юзабельная структура классов: вместо HashMap, например, Map (иммутабельный) и MutableMap (мутабельный) — позволяет с легкостью входить в язык новичкам и концентрироваться на бизнесовой постановке, а не на способах реализации.

В отличии от Java который имеет ряд недостатков:

* Низкая скорость
* Требует много памяти
* С 2019 года обновления для бизнеса и коммерческого использования стали платными
* Для ООП нужен опыт, а планирование новой программы занимает много времени
* Низкая производительность. Из-за компиляции и абстракции с помощью виртуальной машины.

Следующим на очереди стоит выбор среды разработки:

* IntelliJ IDEA. Одна из популярных сред разработки. Доступна в двух версиях — бесплатной Community edition для самостоятельных разработчиков и платной Ultimate edition для компаний. Подходит для мобильной разработки под Android и десктопной Java для запуска на компьютере.
* Android Studio. Основана на IntelliJ IDEA, предлагает все нужные функции и заточена под проекты на Android. IDE рекомендует Google, ее используют начинающие разработчики и опытные Android-девелоперы
* IDE Eclipse. бесплатная среда разработки

Преимущества IntelliJ IDEA:

* Мультиязычная. Поддерживает разные языки программирования+++
* Интегрирована с Git и Github. Позволяет использовать систему контроля версий Git и отправлять проекты на Github в один клик.
* Позволяет настроить нужное окружение. Чтобы запускать тесты в IDE, достаточно докачать Java Devkit.

Преимущества Android Studio:

* Рекомендована Google. IDE поддерживают и развивают в корпорации
* Среда разработки оптимальна для создания приложений для Android.
* Большое сообщество разработчиков. Почти все Android-девелоперы используют Android Studio. Если у вас возникнут проблемы в работе, сможете обратиться к коллегам на форумах или тематических сайтах — вам помогут.

Eclipse, бесплатная среда разработки этой среде можно подключать дополнительные модули, позволяющие расшить ее функционал. Однако это уже устарелая среда разработки

2.2 Разработка интерфейса

Пользовательский интерфейс - это система средств для взаимодействия пользователя с компьютером, основанная на представлении всех доступных пользователю системных объектов и функций в виде графических компонентов экрана (окон, значков, меню, кнопок, списков и т.п.). При этом, в отличие от интерфейса командной строки, пользователь имеет произвольный доступ (с помощью клавиатуры или указательного устройства ввода) ко всем видимым экранным объектам

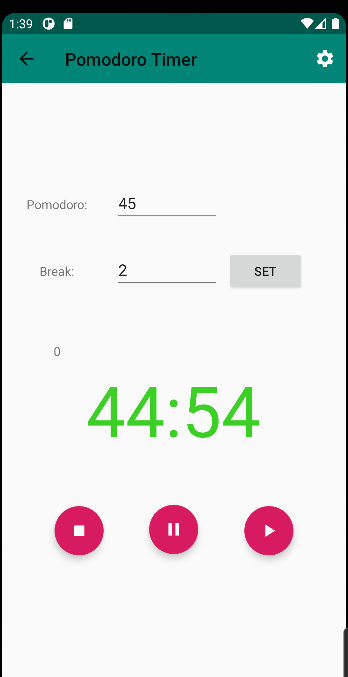
Одной из самых важных функций любой программы является ввод и вывод данных. Выводимые данные это то, что сообщается пользователю. Входные данные это то, что пользователь сообщает программе. Выводимые данные в программе представлены в виде графического отображения окна программы(рис.1)

Рисунок 2 - Окно программы

Входные данные представлены в виде программного кода, который необходимо выполнить при определенных действиях пользователя, а именно:

* + нажатие клавиш клавиатуры;
  + работа пользователя с кнопками в интерфейсе.

Компоненты, которые показаны на рисунке 1, выполняют следующие действия:

Button1 – кнопка, которая необходима для реализации в программе команды «set».



Рисунок 3 - Кнопка SET

Button2 – кнопка, которая необходима для реализации в программе команды паузы.

Рисунок 4 - Кнопка паузы

Button3 – кнопка, которая необходима для реализации в программе команды начала отсчета



Рисунок 5 - Кнопка начала отсчета

Button4 – кнопка, которая необходима для реализации в программе команды остановки и сброса отсчета.

Рисунок 6 - Кнопка сброса отсчета

Label1 – компонент, отображаемый текст заголовков, который не изменяется в течение всей работы программы «Pomodoro Timer».

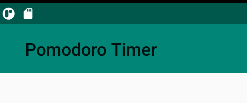


Рисунок 7 - Label1

Label2 – компонент, отображающий отсчет времени, может меняться в течение всей работы программы

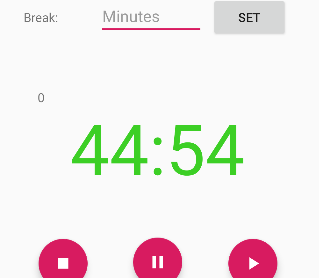


Рисунок 8 - Таймер

Label3 – компонент, отображающий текст, который не изменяется в течение всей работы программы «Pomodorо:».



Рисунок 9 - Label Pomodorо  
Label4 – компонент, отображающий текст, который не изменяется в течение всей работы программы «Break:».

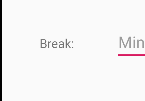


Рисунок 10 - Label Break

Label5 – компонент, отображающий текст, который отображает подсчет выполненный раундов работы

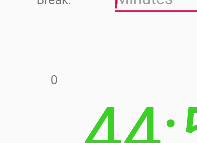


Рисунок 11 - Счетчик помидорок

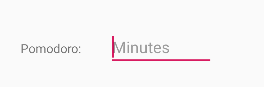
Edit1 – однострочное текстовое поле, с помощью которого пользователь вводит сколько минут будет длиться период работы

Рисунок 12 - Ввод периода работы

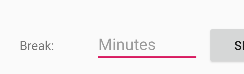
Edit2 – однострочное текстовое поле, с помощью которого пользователь вводит сколько минут будет длиться период перерыва

Рисунок 13 - Ввод периода перерыва

2.3 Разработка логики работы приложения. Схема взаимодействия компонентов проекта

В разрабатываемом приложении реализуется работа таймера в методике поморора.

Пользователь имеет возможность запуска потока, приостановки и возобновления потока, а также остановки запущенного потока с сохранением текущего состояния в файл и последующей загрузки и продолжения потока с сохраненного состояния.

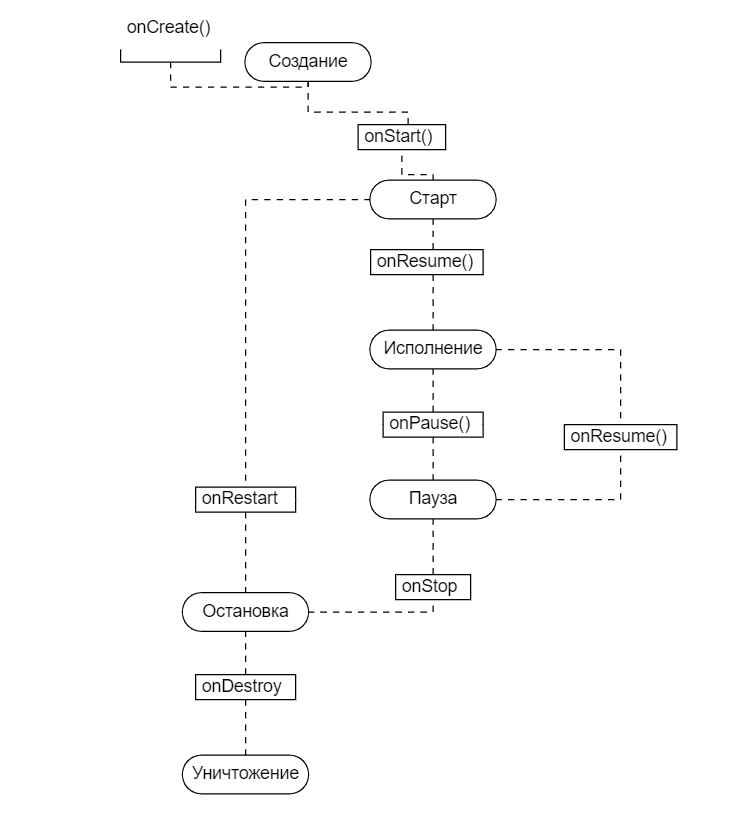


Рисунок 14 - Схема приложения

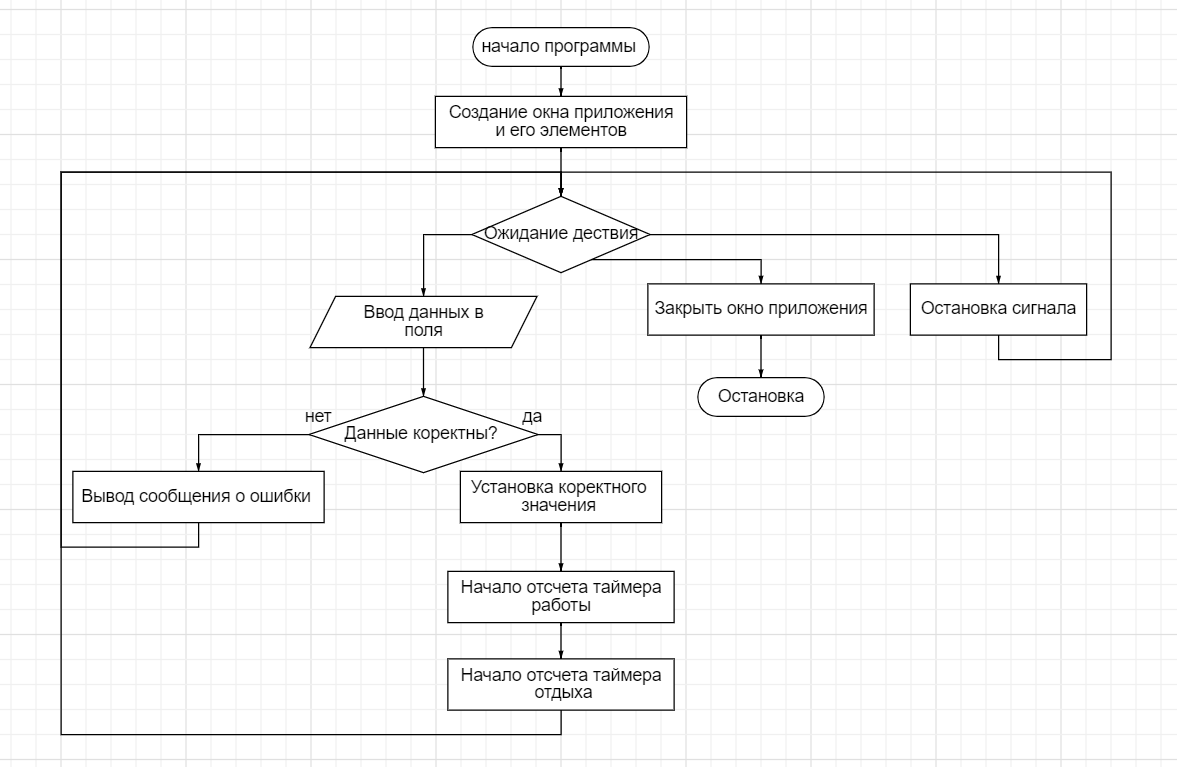


Рисунок 15 - Схема работы приложения

2.4 Описание переменных, компонентов, классов и подпрограмм

Таблица 5 - Методы

|  |  |
| --- | --- |
| Методы | Описание |
| onCreate | Привязка к активностям. |
| onStartCommand | Необходим для возвращения констант. |
| onDestroy | Для удаления таймера. |
| onTick | Показывает оставшиеся время. |
| onFinish | Форма рекорда |
| onBind | Привязка к активностям. |
| createNotificationChannel | Для создания уведомлений. |
| makeToast | Для уведомлений сверху. |
| onPause | Для паузы таймера. |
| onStop | Для остановки таймера. |
| onResume | Для продолжения таймера. |
| handleCountDown | Обновление графического интерфейса каждую секунду. |
| setTimerColor | Выставление цвета цифр таймера. |
| onSaveInstanceState | Сохранение последнего результата. |

Таблица 6 – Классы

|  |  |
| --- | --- |
| Методы | Описание |
| Class CountDownService | Обратный отсчёт. |
| Class ForegroundService | Сохранение последнего режима работы. |
| Class MainActivity | Главное меню и действия в нём. |
| Class Timer | Класс таймера |

3. Тестирование и установка приложения

Тестирование программы - это этап, на котором проверяется, как ведет себя программа на как можно большем количестве входных наборов данных, в том числе и на заведомо неверных.

Во время выполнения отладки, были использованы следующие методы:

* + Тестирование с помощью Monkey;
  + анализ кода без исполнения программы;
  + запуск программы из-под отладчика.

После установки приложения на устройство запускаем приложение тестировать приложение, используя функциональное тестирование

В данном пункте важно убедиться, что продукт соответствует нужной функциональной спецификации, упомянутой в техническом задании по разработке.  
 Тестирование с помощью Monkey. Monkey, входящая в состав Android SDK, отправляет поток случайных действий пользователя. В командной строке можно указать количество действий пользователя, долю действий каждого типа и имя пакета Автоматическое тестирование с помощью Monkey можно считать неплохой начальной проверкой для любого приложения. Этот метод может дать достаточно полезные результаты для определенного приложения. Но при низком качестве тестирования следует использовать другие методы.

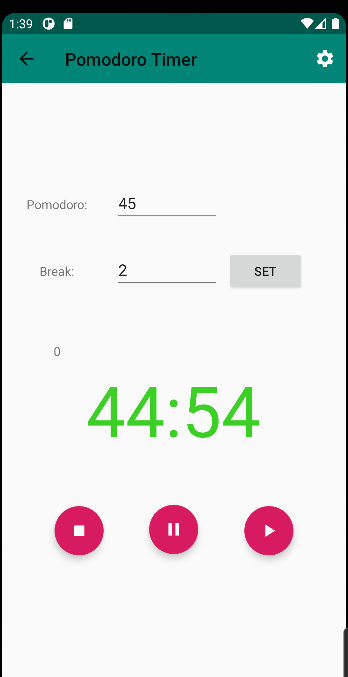


Рисунок 16. Запуск приложения

Так же в проекте были с нуля созданы интерфейс и дизайн приложения согласно техническому заданию.

Из этого следует, что создание мобильного приложения процесс трудоемкий и затрачивает он очень много времени из-за необходимости тестов и перестройки модели приложения.

После проведения данного тестирования было выявлено, что тестирование прошло успешно, были предусмотрены некоторые исключительные ситуации. Приложение соответствует техническому заданию.

Заключение

В результате выполнения данного курсового проекта был разработан игровой программный продукт "PomodoroTimer». При помощи этой программы можно высчитывать время по методу помидора.

Программа не занимает много места, не требовательна к установленному программному обеспечению. Также было проведено исследование полученного программного продукта. В результате были выявлены следующие достоинства и недостатки полученного программного продукта:

Достоинства:

* + Существует возможность сбрасывать результат;
  + Программный продукт малотребователен к системным ресурсам компьютера.

Недостатки:

* + Невозможность сохранения результатов в памяти программы;

В целом, поставленная в начале курсового проекта цель была достигнута. В программе выполняются все необходимые функциональные требования.

Список литературы

1 Дон Гриффитс, Дэвид Гриффитс. «Head First. Kotlin», СПб.: Питер 2019 год 464 стр.  
2 Голощапов А.Л. Android. Создание приложений для смартфонов и планшетных ПК, СПб: БХВ-Петербург, 2016 г.

3 Хашими С., Коматинени С., Маклин Д. Разработка приложений для Android. - СПб.: Питер, 2015. – 736 с.

4 Вейл, Эстель HTML5. Разработка приложений для мобильных устройств / Эстель Вейл. - М.: Питер, 2015. - 818 c

5 Дэвид Гриффитс, Дон Гриффитс «Head First. Программирование для Android» Питер 2018год 912 стр.

6 Бондарев В.М., Рублинецкий В.И., Качко Е.Г. Основы программирования. —Харьков: Фолио, Ростов н/Д: Феникс, 2019.

7 Гниденко, И. Г. Технология разработки программного обеспечения : учеб. пособие для СПО / И. Г. Гниденко, Ф. Ф. Павлов, Д. Ю. Федоров. — М. : Издательство Юрайт, 2017. — 235 с.

8 Трофимов, В. В. Основы алгоритмизации и программирования : учебник для СПО / В. В. Трофимов, Т. А. Павловская ; под ред. В. В. Трофимова. — М. :Издательство Юрайт, 2019. — 137 с.

9 Баженова, И.Ю. Языки программирования: Учебник для студентов учреждений высш. проф. образования / И.Ю. Баженова; Под ред. В.А. Сухомлин. — М.: ИЦ Академия, 2018. — 368 c.

10 Довек, Ж. Введение в теорию языков программирования / Ж. Довек, Ж.-Ж. Леви. — М.: ДМК, 2016. — 134 c.

## Приложения

Приложение А. Листинг кода

Timer

package com.example.pomodorotimer

class Timer{

private var hourToCount: Int = 0 /\*час\*/

private var minToCount: Int = 0 /\*минута\*/

private var secondToCount: Int = 0 /\*секунда\*/

var isCounting = false /\*\*/

var isPause = false

/\*нам не нужна третья переменная, хранящая isStop, потому что isStop будет истинным, когда таймер не считает и таймер не останавливается\*/

var workState = WorkState.Work /\*по умолчанию запуск с таймером работы\*/

/\*подсчёт в минутах\*/

var breakTimer: Int = 0

var workTimer: Int = 0

/\*таймер перерыва\*/

fun loadBreakTimer() {

hourToCount = 0

minToCount = breakTimer

secondToCount = 0

}

/\*таймер работы\*/

fun loadWorkTimer() {

hourToCount = 0

minToCount = workTimer

secondToCount = 0

}

/\*вывод времени на дисплей\*/

fun displayTime(): String {

var result = ""

var hr = hourToCount.toString() /\*присвоение\*/

var min = minToCount.toString() /\*присвоение\*/

var sec = secondToCount.toString() /\*присвоение\*/

if (hourToCount > 0) {

result += (if (hr.length < 2) hr.padStart(2, '0') else hr) + ":"

}

result += (if (min.length < 2) min.padStart(2, '0') else min) + ":"

result += if (sec.length < 2) sec.padStart(2, '0') else sec

return result

}

/\*расчёт секунд\*/

fun toSeconds(): Long {

return hourToCount.toLong() \* 60 \* 60 + minToCount.toLong() \* 60 + secondToCount.toLong()

}

/\*расчёт от секунд\*/

fun restoreFromSeconds(s:Long){

secondToCount = (s % 60).toInt()

minToCount = (((s-secondToCount)/60) % 60).toInt()

hourToCount = ((s-secondToCount-minToCount\*60)/3600).toInt()

}

/\*расчёт минут,секунд, часов\*/

fun minusOneSecond() {

if (hourToCount == 0 && minToCount == 0 && secondToCount <= 1) {

secondToCount = 0

isCounting = false

return

}

if (secondToCount == 0 && minToCount > 0) {

secondToCount = 60

minToCount--

} else if (secondToCount == 0 && minToCount == 0) {

secondToCount = 60

minToCount = 60

hourToCount--

}

secondToCount--

}

/\*Перезагрузка таймера\*/

fun resetTimer(){

isCounting = false

isPause = false

/\*сброс таймера и возврат в стандартное сострояние\*/

workState = WorkState.Work

secondToCount=0

minToCount= workTimer

hourToCount=0

}

}

MainActivity

class MainActivity : AppCompatActivity() { /\*шаблон для создания объекта\*/

private var timer = Timer() /\*переменные\*/

private val channelId = "pomodoroTimer" /\*переменные\*/

lateinit var wakeLock: PowerManager.WakeLock /\*переменные\*/

private var completed = 0 /\*переменные\*/

private fun createNotificationChannel() { /\*возможность реализовать метод так, чтобы он имел идентичную сигнатуру с методом класса-предка, Объявление функции начинается с ключевого слова fun, затем идёт имя функции, в круглых скобках указываются параметры.\*/

if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) { /\*создание NotificationChannel выше API 26+ потому что это новый класс который не поддерживает старые версии\*/

val name = getString(R.string.channel\_name) /\*неизменяемые переменные\*/

val descriptionText = getString(R.string.channel\_description) /\*неизменяемые переменные\*/

val importance = NotificationManager.IMPORTANCE\_DEFAULT /\*неизменяемые переменные\*/

val channel = NotificationChannel(channelId, name, importance).apply { /\*неизменяемые переменные\*/

description = descriptionText /\*описание\*/

}

/\*Регистрация канала в системе\*/

val notificationManager: NotificationManager = /\*Регистрация канала в системе\*/

getSystemService(Context.NOTIFICATION\_SERVICE) as NotificationManager /\*Регистрация канала в системе\*/

notificationManager.createNotificationChannel(channel) /\*Регистрация канала в системе\*/

}

}

private fun sendNotification() { /\*возможность реализовать метод так, чтобы он имел идентичную сигнатуру с методом класса-предка, Объявление функции начинается с ключевого слова fun, затем идёт имя функции, в круглых скобках указываются параметры.\*/

with(NotificationManagerCompat.from(this)) { /\*Отмена последнего уведомления\*/

cancel(completed - 1) /\*Отмена последнего уведомления\*/

}

val notificationIntent = Intent(this, MainActivity::class.java).apply { /\*неизменяемые переменные\*/

flags = Intent.FLAG\_ACTIVITY\_NEW\_TASK or Intent.FLAG\_ACTIVITY\_CLEAR\_TASK /\*True если одна из переменных true\*/

}

val pendingIntent: PendingIntent = PendingIntent.getActivity(this, 0, notificationIntent, 0)

val notification: Notification = /\*настройка уведомлений\*/

NotificationCompat.Builder(this, channelId)

.setContentTitle("Session $completed Completed")

.setContentText("Yippee ki yay")

.setSmallIcon(R.drawable.timericon)

.setPriority(NotificationCompat.PRIORITY\_DEFAULT)

.setContentIntent(pendingIntent)

.setAutoCancel(true)

.build()

with(NotificationManagerCompat.from(this)) {

notify(completed, notification) /\*это уникальный int для каждого уведомления, которое вы должны определить\*/

}

}

private fun makeToast(message: String) { /\*Показывает уведомление сверху\*/

val toast = Toast.makeText(applicationContext, message, Toast.LENGTH\_SHORT)

toast.setGravity(Gravity.TOP, 0, 200)

toast.show()

}

override fun onCreate(savedInstanceState: Bundle?) { /\*возможность реализовать метод так, чтобы он имел идентичную сигнатуру с методом класса-предка, Объявление функции начинается с ключевого слова fun, затем идёт имя функции, в круглых скобках указываются параметры.\*/

super.onCreate(savedInstanceState) /\*используется тогда, когда подклассу требуется сослаться на его непосредственный супер класс\*/

setContentView(R.layout.activity\_main) /\*устанавливает activity\*/

setSupportActionBar(toolbar) /\*добавляет доп активность\*/

supportActionBar?.setDisplayHomeAsUpEnabled(true) /\*включает функцию\*/

setRequestedOrientation(ActivityInfo.SCREEN\_ORIENTATION\_PORTRAIT); /\*Вертикальная ориентация\*/

window.addFlags(WindowManager.LayoutParams.FLAG\_KEEP\_SCREEN\_ON) /\*оставлять на экране уведомление\*/

textView\_completed.setText(completed.toString()) /\*показывать уведомление о завершении задачи\*/

createNotificationChannel() /\*показывает уведомление\*/

val powerManager = getSystemService(Context.POWER\_SERVICE) as PowerManager /\*позволяет работать приложению после удаления из даспетчера задач\*/

wakeLock = powerManager.newWakeLock(

PowerManager.PARTIAL\_WAKE\_LOCK,

"pomodoroTimer::wakeLock"

)

/\*Отображает выбранные варианты\*/

val sharedPref = getPreferences(Context.MODE\_PRIVATE) ?: return

val savedWorkTimer = sharedPref.getInt("WORK",0)

val savedBreakTimer = sharedPref.getInt("BREAK",0)

editText\_pomodoro.setText(savedWorkTimer.toString())

editText\_break.setText(savedBreakTimer.toString())

Log.i("sharedPref","work $savedWorkTimer")

Log.i("sharedPref","break $savedBreakTimer")

timer.workTimer = savedWorkTimer /\*присваивание значения переменной\*/

timer.breakTimer = savedBreakTimer /\*присваивание значения переменной\*/

timer.loadWorkTimer() /\*загрузка таймера\*/

textView\_countdown.text = timer.displayTime() /\*показывает оставшиеся время таймера\*/

setTimerTextColor() /\*выставление цвета\*/

fab\_play.setOnClickListener { /\*проигрывать звук при нажатии\*/

Log.i("timerapp", "clicked timer start")

/\*не запускает новый таймер если уже есть активный\*/

if (timer.isCounting) {

Log.i("timerapp", "ignore duplicate starting request") /\*\*/

makeToast("Already started")

return@setOnClickListener

}

startTimer() /\*запуск таймера\*/

}

/\*при нажатии паузы удаляет прошный таймер\*/

fab\_pause.setOnClickListener {

Log.i("timerapp", "clicked timer pause")

if (timer.isCounting) {

makeToast("Pause timer")

destroyTimer()

timer.isCounting = false

timer.isPause = true

} else {

/\*Ничего не делает если таймер не запущен, нажатие на паузу при неактивном таймере невозможно\*/

makeToast("Already pause")

}

}

/\*остановка таймера означает отмену таймера\*/

fab\_stop.setOnClickListener {

Log.i("timerapp", "clicked timer stop(cancel)")

makeToast("Cancel timer")

/\*если таймер запущен и нажата кнопка отмена, он удаляется\*/

if (timer.isCounting) {

destroyTimer()

}

/\*если пауза уже нажата, приложение останавливается, и обновляет подсчёт\*/

timer.resetTimer()

timer.loadWorkTimer()

textView\_countdown.text = timer.displayTime()

setTimerTextColor()

}

button\_set.setOnClickListener {

Log.i("timerapp", "clicked set button")

val workTime = editText\_pomodoro.text.toString() /\*неизменяемые переменные\*/

val breakTime = editText\_break.text.toString() /\*неизменяемые переменные\*/

val sharedPref = getPreferences(Context.MODE\_PRIVATE) /\*неизменяемые переменные\*/

/\*сохранение в общих настройках\*/

val editor = sharedPref.edit() /\*неизменяемые переменные\*/

editor.putInt("WORK",workTime.toInt())

editor.putInt("BREAK",breakTime.toInt())

editor.commit()

timer.workTimer = if (workTime.equals("")) 2 else workTime.toInt()

timer.breakTimer = if (breakTime.equals("")) 2 else breakTime.toInt()

Log.i(

"timerapp",

"workTimer set to ${timer.workTimer}, breakTimer set to ${timer.breakTimer}"

)

/\*если таймер не работает и таймер не поставлен на паузу, отображать время обратного отсчета\*/

if (!timer.isCounting and !timer.isPause){

timer.loadWorkTimer()

textView\_countdown.text = timer.displayTime()

textView\_countdown.setTextColor(resources.getColor(R.color.colorWork))

makeToast("Current session: Work ${timer.workTimer} min, break ${timer.breakTimer} min") /\*отображение надписи о том сколько проведено в работе и когда перерыв\*/

}else{

makeToast("Next session: Work ${timer.workTimer} min, break ${timer.breakTimer} min") /\*отображение надписи о том сколько остальсь отдыхать и когда будет второй круг\*/

}

editText\_pomodoro.setText(timer.workTimer.toString()) /\*вывод текста\*/

editText\_break.setText(timer.breakTimer.toString()) /\*вывод текста\*/

CountDownService

package com.example.pomodorotimer

import android.app.Service

import android.content.Intent

import android.os.CountDownTimer

import android.os.IBinder

import android.util.Log

class CountDownService : Service() { /\*шаблон для создания объекта\*/

var bi = Intent(COUNTDOWN\_BR) /\*Для обычных изменяемых переменных\*/

private lateinit var timer: CountDownTimer /\*Общедоступный класс, переменные которого нельзя инициализировать сразу\*/

override fun onCreate() { /\*возможность реализовать метод так, чтобы он имел идентичную сигнатуру с методом класса-предка, Объявление функции начинается с ключевого слова fun, затем идёт имя функции, в круглых скобках указываются параметры.\*/

super.onCreate() /\*обозначает суперкласс, т. е. класс, производным от которого является текущий класс.\*/

return super.onCreate() /\*Возвращает к суперклассу\*/

}

override fun onDestroy() { /\*возможность реализовать метод так, чтобы он имел идентичную сигнатуру с методом класса-предка, Объявление функции начинается с ключевого слова fun, затем идёт имя функции, в круглых скобках указываются параметры.\*/

super.onDestroy() /\*обозначает суперкласс, т. е. класс, производным от которого является текущий класс.\*/

timer.cancel() /\*нужен для удаления себя из очереди задач\*/

bi.putExtra("forceStopped", true) /\*запись в bi\*/

sendBroadcast(bi) /\*отправляет в обьект\*/

return super.onDestroy() /\*Возвращает к суперклассу\*/

}

override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int { /\*возможность реализовать метод так, чтобы он имел идентичную сигнатуру с методом класса-предка, Объявление функции начинается с ключевого слова fun, затем идёт имя функции, в круглых скобках указываются параметры.\*/

var secToCount: Long = intent?.getLongExtra("toCount", -1) ?: -1 /\*Для обычных изменяемых переменных, \*/

secToCount++ /\* +1 секунда, чтобы избежать завершения некоторых сервисов телефона до того, как наш собственный класс таймера достигнет 0 \*/

try { /\*определяет блок кода, в котором может произойти исключение\*/

timer = object:CountDownTimer(secToCount\*1000, 1000) { /\* обратный отсчёт по секунде за раз\*/

override fun onTick(millisUntilFinished: Long) { /\*показавыет сколько миллисекунд осталось\*/

var msRemain:Long = millisUntilFinished /\*сколько осталось миллисекунд до конца\*/

Log.i("timerapp", msRemain.toString())

bi.putExtra("toCount", msRemain) /\*запись в bi\*/

sendBroadcast(bi) /\*отправляет в обьект\*/

}

override fun onFinish() { /\*возможность реализовать метод так, чтобы он имел идентичную сигнатуру с методом класса-предка, Объявление функции начинается с ключевого слова fun, затем идёт имя функции, в круглых скобках указываются параметры.\*/

Log.i("timerapp", "timer finish")

bi.putExtra("toCount", -1.toLong()) /\*запись в bi\*/

sendBroadcast(bi) /\*отправляет в обьект\*/

}

}

timer.start() /\*для старта таймера\*/

} catch (e: InterruptedException) {

Thread.currentThread().interrupt()

}

return super.onStartCommand(intent, flags, startId) /\*возврат к суперклассу\*/

}

override fun onBind(arg0: Intent?): IBinder? { /\*возможность реализовать метод так, чтобы он имел идентичную сигнатуру с методом класса-предка, Объявление функции начинается с ключевого слова fun, затем идёт имя функции, в круглых скобках указываются параметры.\*/

return null /\*говорит что мы намеренно возвращаем null и у нас для этого есть причины.\*/

}

companion object { /\*это объект, который объявлен в том же файле, что и класс, и имеет то же имя, что и класс.\*/

const val COUNTDOWN\_BR = "CountDownService.countdown\_br" /\*COUNTDOWN\_BR - станет приватной переменной, для доступа к которой будет создан геттер. переменная будет заинлайнена, то есть компилятор заменит все полученные значения этой переменной на само значение.\*/

}

Приложение Б. Руководство пользователя.

Руководство пользователя мобильного приложения для устройств на ОС Android

1 Требования к аппаратуре и программному обеспечению

Операционная система: Android 8.0 и выше

Размер ОЗУ: от 2 ГБ

Поддержка 3G, 4G

2. Установка, обновление, удаление мобильного приложения

Для удаления мобильного приложения необходимо на мобильном устройстве в настройках удалить мобильное приложение.

3. Запуск мобильного приложения

Для запуска приложения нужно найти его в списке и нажать на иконку. При этом будет открыто основное окно мобильного приложения.

4. Интерфейс системы

Интерфейс мобильной платформы ориентирован на то, что в один момент времени на экране отображается одна форма. Под нее освобождается максимум свободного места на экране. Первой видимой формой является главный экран. Команды, открывающие другие экраны, находятся в главном меню приложения. Также это меню содержит команду возврата на рабочий стол, команды отображения информации о программе и перехода к списку приложений. Вызов главного меню приложения осуществляется с помощью системной кнопки вызова меню.

1. Выход из приложения

Для выхода из приложения достаточно свайпнуть по экрану согласно системе или нажать кнопку назад на нижней панели управления.

Рецензия

Курсовой проект на тему: Разработка мобильного приложения для фокусировки внимания.

В курсовом проэкте содержится 23 листа формата А4, и также имеются приложения, которые более подробно раскрывают тему работы. По содержанию курсовой проэкт целостная, раскрывает все пункты. Объем и оформление курсового проекта соответствуют заявленным требованиям.

Структура курсового проект:

Курсовой проект состоит из трех разделов, первые два содержат по 3 пункта. Заключения, списка литературы (4 наименований) и приложений (16 единиц).

Во введении обусловлена тема исследования, представлено описание метода, корректно сформулирована цель.

Выводы подтверждают практическую значимость результатов исследования. Поэтому проект закончилась логично, где была продемонстрирована компетентность.

https://github.com/washraava/Pomodoro.git