**Министерство образования и науки РФ**

**Санкт-Петербургский политехнический университет Петра Великого**

**Институт компьютерных наук и технологий**

**Высшая школа программной инженерии**

**К У Р С О В А Я Р А Б О Т А**

**Фитнес-трекер**

по дисциплине «Технологии разработки качественного программного обеспечения»

|  |  |  |
| --- | --- | --- |
| Выполнил  студент гр. в3530904/70321 | <*подпись*> | И.С. Гнатюк |
| Руководитель | <*подпись*> | Н.Г. Смирнов |

« » 2020г.

Санкт-Петербург

2020

**Содержание**

[Введение 3](#_Toc40038870)

[Описание функционала программы 4](#_Toc40038871)

[Ручное тестирование программы 7](#_Toc40038872)

[UML диаграмма 11](#_Toc40038873)

[Вывод 12](#_Toc40038874)

[Список используемой литературы 13](#_Toc40038875)

[Приложение 1 14](#_Toc40038876)

# **Введение**

Проблема лишнего веса существует не века, а тысячелетия. В далеком прошлом накапливать жир было эволюционным преимуществом, позволяющим человеку выживать в периоды вынужденного голодания.

В большинстве европейских странах ожирением страдает до 25 % взрослого населения. В последнее время лишним весом страдают дети и подростки в развитых странах мира.

Значимость проблемы лишнего веса определяется угрозой инвалидности людей молодого возраста и снижением общей продолжительности жизни из-за частого развития тяжелый сопутствующих заболеваний.

В связи с этим очень важно следить за своим здоровьем и телом. Именно поэтому крупные компании развивают тему спортивного оборудования. Совсем не обязательно заниматься профессиональным спортом, чтобы иметь умные часы или фитнес-трекеры. Наличие данных атрибутов в использовании - это первый шаг над контролем своей жизни и своего здоровья.

Фитнес-трекер представляет из себя программу, которая считает калории, измеряет пульс, сохраняет данные о тренировках, измеряет шаги, записывает статистику сна.

# **Описание функционала программы**

Интерфейс программы выполнен в Scene Builder (см. Рисунок 1).

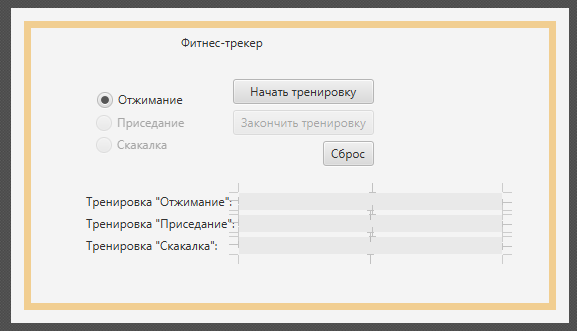


Рисунок 1. Интерфейс фитнес-трекера

Чтобы начать тренировку необходимо выбрать один вариантов: отжимание, приседание или скакалку, и нажать кнопку «Начать тренировку». Одновременно доступен только один режим тренировки, когда ни один режим не выбран, тренировку начать нельзя (см. Рисунок 2). После того как режим выбран и нажата кнопка «Начать тренировку» начинается отсчет времени в соответствующем поле снизу (см. Рисунок 3).

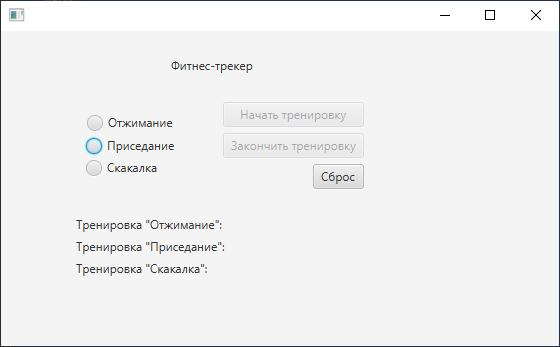


Рисунок 2. Тренировку невозможно начать

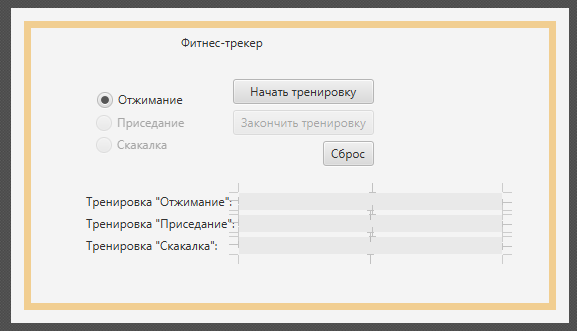


Рисунок 3. Таймер

Вместе со временем будут считаться сожженные калории. Сжигание калорий считается по формуле (см. Формула 1.1):

(1.1)

где К – количество калорий в час;

t – время.

У каждой тренировки свое количество сожженных калорий в час:

1. Отжимание – 475 калорий в час;
2. Приседание - 600 калорий в час;
3. Скакалка - 500 калорий в час.

После завершения подхода, можно нажать на кнопку «Закончить тренировку» и таймер остановится. Если есть необходимость продолжить тренировку можно нажать кнопку «Начать тренировку» и таймер продолжит отсчет с того места, на котором остановился.

При желании можно сбросить значения, нажав соответствующую кнопку «Сброс»

# **Ручное тестирование программы**

При запуске программы пользователя встречает интерфейс фитнес-трекера. Выбираем режим «Отжимание» (см. Рисунок 4) и нажимаем «Начать тренировку» (см. Рисунок 5).

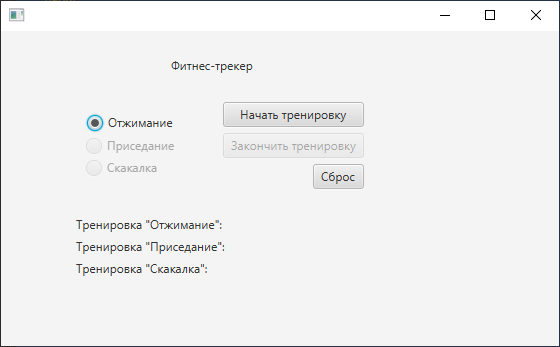


Рисунок 4. Подготовка к тренировке

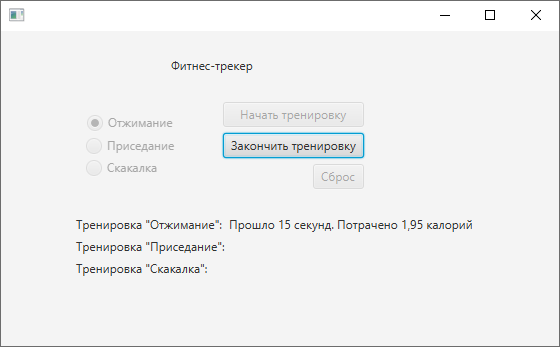


Рисунок 5. Процесс тренировки в режиме «Отжимание»

После подхода в режиме «Отжимание», меняем на следующий режим – «Приседание». Таймер режима «Отжимание» стоит на паузе. Результат можно наблюдать на рисунке 6.

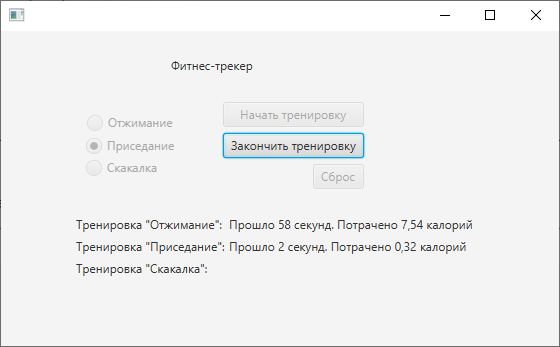


Рисунок 6. Процесс тренировки в режиме «Приседание»

Возвращаемся к режиму «Отжимание», таймер продолжает отсчет с точки на которой остановился. Результат можно наблюдать на рисунке 7.

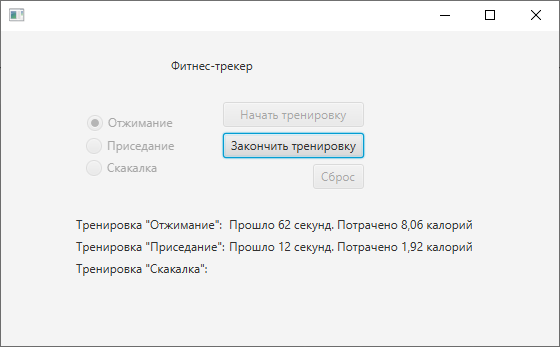


Рисунок 7. Процесс тренировки в режиме «Отжимание» продолжается

Выбираем следующий режим «Скакалка». Таймеры обоих предыдущих режимов остановились (см. Рисунок 8).

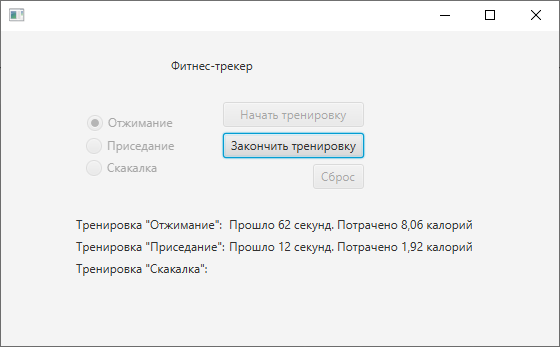


Рисунок 8. Процесс тренировки в режиме «Скакалка»

После завершения тренировки мы получаем сведения о времени и сожженных калориях каждого режима. Результат тренировки можно наблюдать на рисунке 9.

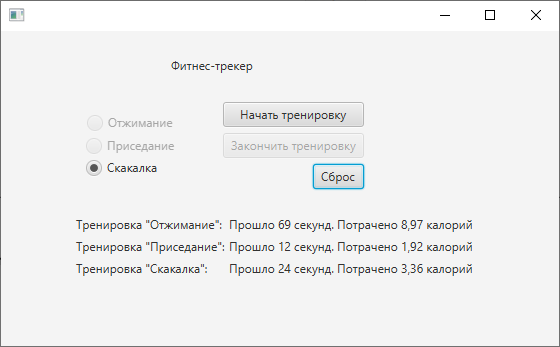


Рисунок 9. Результаты всех режимов после тренировки

Для того, чтобы очистить результаты режимов, необходимо нажать кнопку «Сброс». Результат можно наблюдать на рисунке 10.

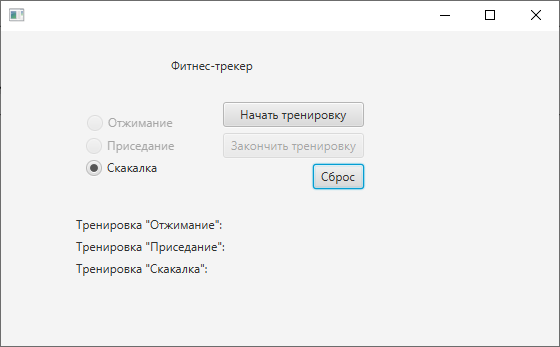


Рисунок 10. Результаты всех режимов сброшены

При желании начать тренировку снова, необходимо выбрать режим и нажать кнопку «Начать тренировку». Отсчет таймера начнется с начала. Результат можно наблюдать на рисунке 11.

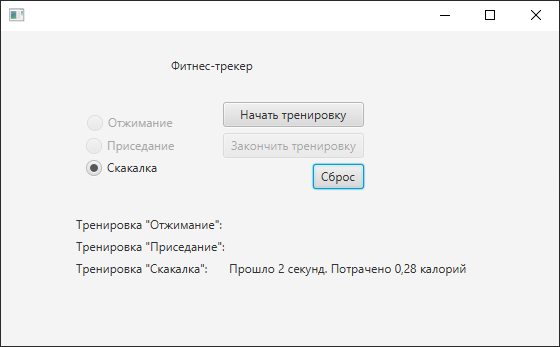


Рисунок 11. Начало новой тренировки

# **UML диаграмма**

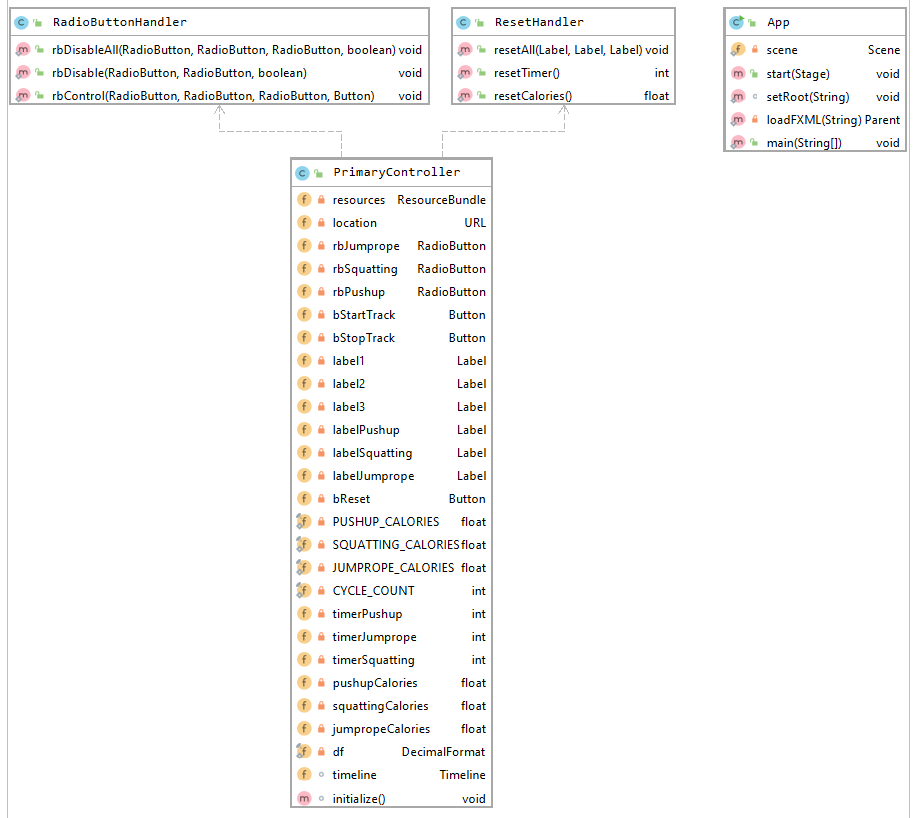


Рисунок 12. UML диаграмма

# **Вывод**

В процессе работы был программно реализован фитнес-трекер. В данной программе есть возможность тренировки в трех режимах: отжимание, приседание и скакалка. Если ни один режим не был выбран, начать тренировку не получится. У каждого режима есть таймер времени тренировки и таймер сожженных калорий. У каждого вида тренировки собственные значения сожженных калорий в час: «Отжимание» – 475 калорий в час, «Приседание» - 600 калорий в час, «Скакалка» - 500 калорий в час.

После завершения тренировки в одном из режимов, таймер становится на паузу. При желании можно продолжить со значениями от предыдущего подхода, а также можно сделать полный сброс значений и начать тренировку заново, таймер при этом начнет отсчет с начала.

Приложение создано на основе javafx.

Поставленную задачу приложение выполняет.

# **Список используемой литературы**

1. Документация [JavaFX Scene Builder](https://docs.oracle.com/javase/8/scene-builder-2/user-guide/index.html)

# **Приложение 1**

**ResetHandler.java**

package org.openjfx;

import javafx.scene.control.Label;

public class ResetHandler {

// сбросить все значения

public static void resetAll (Label labelPushup, Label labelJumprope, Label labelSquatting) {

labelPushup.setText("");

labelJumprope.setText("");

labelSquatting.setText("");

}

public static int resetTimer () {

return 0;

}

public static float resetCalories () {

return 0;

}

}

**RadioButtonHandler.java**

package org.openjfx;

import javafx.scene.control.Button;

import javafx.scene.control.RadioButton;

// отвечает за доступность переключателей

public class RadioButtonHandler {

public static void rbDisableAll(RadioButton rb1, RadioButton rb2, RadioButton rb3, boolean disable) {

if (rb1.isSelected()) {

rb1.setDisable(disable);

} else if (rb2.isSelected()) {

rb2.setDisable(disable);

} else {

rb3.setDisable(disable);

}

}

public static void rbDisable(RadioButton rb1, RadioButton rb2, boolean disable) {

rb1.setDisable(disable);

rb2.setDisable(disable);

}

public static void rbControl(RadioButton rb, RadioButton rb1, RadioButton rb2, Button button) {

if (rb.isSelected()) {

rbDisable(rb1, rb2, true);

button.setDisable(false);

} else {

rbDisable(rb1, rb2, false);

button.setDisable(true);

}

}

}

**PrimaryController.java**

package org.openjfx;

import java.net.URL;

import java.text.DecimalFormat;

import java.util.ResourceBundle;

import javafx.animation.KeyFrame;

import javafx.animation.Timeline;

import javafx.fxml.FXML;

import javafx.scene.control.Button;

import javafx.scene.control.Label;

import javafx.scene.control.RadioButton;

import javafx.util.Duration;

import static org.openjfx.RadioButtonHandler.rbControl;

import static org.openjfx.RadioButtonHandler.rbDisableAll;

import static org.openjfx.ResetHandler.\*;

public class PrimaryController {

@FXML

private ResourceBundle resources;

@FXML

private URL location;

@FXML

private RadioButton rbJumprope;

@FXML

private RadioButton rbSquatting;

@FXML

private RadioButton rbPushup;

@FXML

private Button bStartTrack;

@FXML

private Button bStopTrack;

@FXML

private Label label1;

@FXML

private Label label2;

@FXML

private Label label3;

@FXML

private Label labelPushup;

@FXML

private Label labelSquatting;

@FXML

private Label labelJumprope;

@FXML

private Button bReset;

// Формула каллорий в час = затрачиваемое кол-во калорий \* Timer

private static final float PUSHUP\_CALORIES = 0.13f; // 475 калорий в час

private static final float SQUATTING\_CALORIES = 0.16f; // 600 калорий в час

private static final float JUMPROPE\_CALORIES = 0.14f; // 500 калорий в час

private static final int CYCLE\_COUNT = 3600; // цикл тренировки максимум 1 час

private int timerPushup = 0;

private int timerJumprope = 0;

private int timerSquatting = 0;

private float pushupCalories = 0.0f;

private float squattingCalories = 0.0f;

private float jumpropeCalories = 0.0f;

private static final DecimalFormat df = new DecimalFormat("#.###");

Timeline timeline;

@FXML

void initialize() {

// нажата кнопка "Сброс"

bReset.setOnAction(actionEvent -> {

resetAll(labelPushup, labelJumprope, labelSquatting);

timerPushup = resetTimer();

timerJumprope = resetTimer();

timerSquatting = resetTimer();

pushupCalories = resetCalories();

jumpropeCalories = resetCalories();

squattingCalories = resetCalories();

});

// нажата кнопка "Начать тренировку"

bStartTrack.setOnAction(actionEvent -> {

bStopTrack.setDisable(false);

bStartTrack.setDisable(true);

bReset.setDisable(true);

rbDisableAll(rbPushup, rbSquatting, rbJumprope, true);

// запуск таймера

timeline = new Timeline(

new KeyFrame(

Duration.seconds(1),

// каждую секунду повторять

actionEvent1 -> {

// если тренировка "Отжимания"

if (rbPushup.isSelected()) {

timerPushup += 1; // увеличить на 1 секунду

pushupCalories = PUSHUP\_CALORIES \* timerPushup; // каллорий потрачено

labelPushup.setText("Прошло " + timerPushup + " секунд. " +

"Потрачено " + df.format(pushupCalories) + " калорий"); // вывести статистику

}

// если тренировка "Приседания"

else if (rbSquatting.isSelected()) {

timerSquatting += 1;

squattingCalories = SQUATTING\_CALORIES \* timerSquatting;

labelSquatting.setText("Прошло " + timerSquatting + " секунд. " +

"Потрачено " + df.format(squattingCalories) + " калорий");

}

// если тренировка "Скакалка"

else {

timerJumprope += 1;

jumpropeCalories = JUMPROPE\_CALORIES \* timerJumprope;

labelJumprope.setText("Прошло " + timerJumprope + " секунд. " +

"Потрачено " + df.format(jumpropeCalories) + " калорий");

}

}

)

);

timeline.setCycleCount(CYCLE\_COUNT);

timeline.play();

});

// нажата кнопка "Закончить тренировку"

bStopTrack.setOnAction(actionEvent -> {

bStopTrack.setDisable(true);

bStartTrack.setDisable(false);

bReset.setDisable(false);

rbDisableAll(rbPushup, rbSquatting, rbJumprope, false);

// остановить таймер

timeline.stop();

});

rbPushup.setOnAction(actionEvent -> {

rbControl(rbPushup, rbJumprope, rbSquatting, bStartTrack);

});

rbJumprope.setOnAction(actionEvent -> {

rbControl(rbJumprope, rbSquatting, rbPushup, bStartTrack);

});

rbSquatting.setOnAction(actionEvent -> {

rbControl(rbSquatting, rbJumprope, rbPushup, bStartTrack);

});

}

}

**App.java**

package org.openjfx;

import javafx.application.Application;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.stage.Stage;

import java.io.IOException;

/\*\*

\* JavaFX App

\*/

public class App extends Application {

private static Scene scene;

@Override

public void start(Stage stage) throws IOException {

scene = new Scene(loadFXML("primary"));

stage.setScene(scene);

stage.show();

}

static void setRoot(String fxml) throws IOException {

scene.setRoot(loadFXML(fxml));

}

private static Parent loadFXML(String fxml) throws IOException {

FXMLLoader fxmlLoader = new FXMLLoader(App.class.getResource(fxml + ".fxml"));

return fxmlLoader.load();

}

// точка входа

public static void main(String[] args) {

launch();

}

}