Санкт-Петербургский Политехнический Университет

Институт компьютерных наук и технологий

«Высшая школа программной инженерии»

**Курсовое проектирование**

по дисциплине: «Технологии разработки качественного программного обеспечения»

по теме: «Фитнес-трекер»

Выполнил:

студент гр. В3530904/80321 Павлов С. В.

Руководитель: Смирнов Н. Г.

Санкт-Петербург

2021

СОДЕРЖАНИЕ

[1 ВВЕДЕНИЕ 3](#_Toc70352765)

[1.1 Цель работы 3](#_Toc70352766)

[1.2 Постановка задачи 3](#_Toc70352767)

[2 Архитектура приложения 4](#_Toc70352768)

[2.1 Диаграмма классов 4](#_Toc70352769)

[2.2 Диаграмма компонентов 6](#_Toc70352770)

[2.3 Диаграмма последовательностей 7](#_Toc70352771)

[3 Работа программы 8](#_Toc70352772)

[4 Тестирование 10](#_Toc70352773)

[5 Вывод 11](#_Toc70352774)

[ПРИЛОЖЕНИЕ 12](#_Toc70352775)

[Main 12](#_Toc70352776)

[Exercise 13](#_Toc70352777)

[ExerciseData 14](#_Toc70352778)

[ExerciseOption 15](#_Toc70352779)

[ExerciseOptionTest 15](#_Toc70352780)

[ExerciseDataTest 16](#_Toc70352781)

[ExerciseTest 18](#_Toc70352782)

# **1 ВВЕДЕНИЕ**

## **1.1 Цель работы**

Целью данной работы является создание программного продукта «Фитнес-трекер», удовлетворяющего следующему описанию:

Приложение позволяет указать вид тренировки: отжимание, скакалка, приседания. Доступны такие команды, как: начать тренировку (запомнить время начала) и закончить тренировку (запомнить время окончания). За отработанное время высчитывается количество потраченных калорий по формуле: K - количество калорий в секунду, затрачиваемое на определенный вид тренировки, умножается на t – продолжительность тренировки в секундах. От запуска к запуску программы данные должны сохраняться и общее количество калорий - суммироваться. Для этого необходимо создать бинарный файл с обращением к нему при помощи операций сериализации и десериализации. Добавление пользовательских профилей является опциональным.

## **1.2 Постановка задачи**

Приложение может считаться завершенным, если он удовлетворяет следующим требованиям:

* При старте проекта программа считывает бинарный файл с серилизованными данными предыдущих тренировок;
* Пользователь вводит название интересующей его тренировки при помощи клавиатуры и добавляет «start» для фиксации времени начала тренировки;
* Спустя некоторое время пользователь обязан ввести «end» для завершения тренировки и фиксации времени окончания;
* После завершения тренировки данные о ней записываются в бинарный файл;
* Программа не должна позволять начать новую тренировку без завершения предыдущей;
* При вводе «data» приложение выводит все прошедшие тренировки: вид упражнения, сожжённые калории и продолжительность в секундах;
* При вводе «exit» программа завершает свою работу;
* Проект не должен содержать критических ошибок, проводящих к некорректному выполнению вышеописанных требований.

# **2 Архитектура приложения**

## **2.1 Диаграмма классов**

UML-диаграмма классов данного приложения состоит из трёх компонентов, которые можно рассмотреть на рисунке 1.

Перечисление «ExerciseOption» содержит в себе все доступные виды тренировок: отжимание (pressUp), скакалка (skippingRope) и приседание (squats), а также два метода: caloriesPerSec() и stringToEnum(String text). caloriesPerSec позволяет определить количество калорий, которые будут сожжены за секунду выполнения данного типа тренировок. stringToEnum(String text) принимает строку введённою пользователем с клавиатуры, а возвращает вид тренировки в формате перечисления.

Класс «ExerciseData» содержит два поля: datafile (имя бинарного файла для хранения информации о всех тренировках) и pastExercises (список всех тренировок, считанных из бинарного файла). Данный класс модержит следующие методы:

ExerciseData(String datafile) – конструктор с подачей на вход имени считываемого бинарного файла.

addExercise(Exercise exercise) – метод для добавления новой тренировки в список.

readData() – импорт данных в бинарный файл из списка pastExercises.

writeData() – экспорт данных из бинарного файла в список pastExercises. toString() – вывод информации на экран.

Класс «Exercise» содержит три поля: option (тип тренировки), timeStart (время начала тренировки), timeEnd (время окончания тренировки). Данный класс имеет следующие методы:

Exercise (ExerciseOption option) – конструктор с подачей на вход типа тренировки.

static Exercise start (String input) – статический метод для обращения к конструктору класса с подачей на вход введённой пользователем строки.

void end() – фиксация времени окончания тренировки.

long getPassedTime() – потраченное на тренировку время.

long getCaloriesBurned() - количество сожжённых калорий

toString() – вывод информации об упражнении на экран

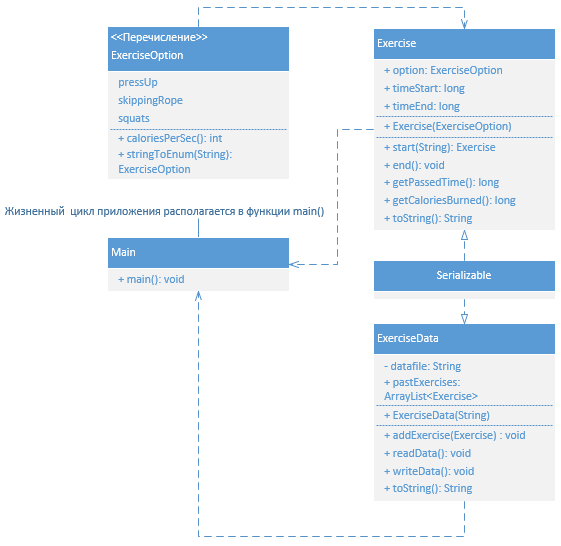


Рисунок 1 – UML-диаграмма классов

## **2.2 Диаграмма компонентов**

Данная диаграмма состоит из двух компонентов: Main, где происходит функционирование всей логики программы, Exercise Data – содержащий полную информацию по всем проведённым тренировкам. После запуска приложения Exercise Data считывает информацию из бинарного файла и хранит её, пока пользователь не запросит её при помощи специальной команды «data» через ввод клавиатуры. Вышеописанная схема представлена на рисунке 2.



Рисунок 2 – UML-диаграмма компонентов

## **2.3 Диаграмма последовательностей**

На диаграмме последовательности показан жизненный цикл программы.

В начале происходит считывание информации из бинарного файла. Далее пользователю предоставляется выбор предпочтительного упражнения. Если ввод совершён правильно, то начинается тренировка. После ввода сообщение об окончании тренировки происходит вывод информации о ней на экран, а также запись в бинарный файл. Затем, цикл работы программы может быть повторён.

Если пользователь введёт команду для получения информации о всех тренировках, тогда соответствующие данные будут десериализированы из бинарного файла и выведены на экран

Если юзер пожелает корректно завершить работу приложения, тогда следует ввести команду «exit», после чего бесконечный цикл функции Main будет окончен. Временная диаграмма продемонстрирована на рисунке 3.

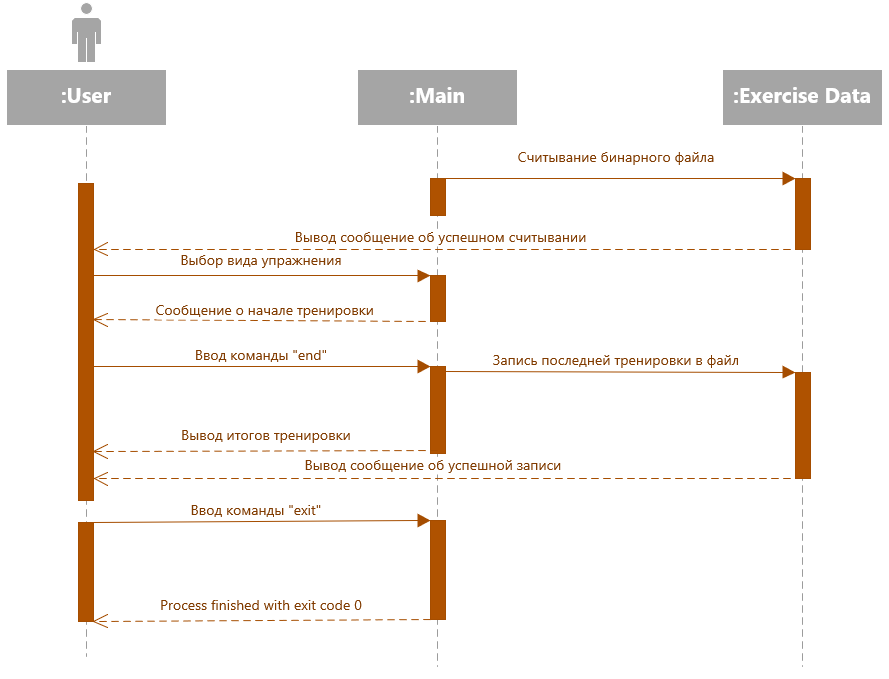


Рисунок 3 – UML-диаграмма последовательностей

# **3 Работа программы**

При запуске приложения происходит автоматическое считывание бинарного файла, содержащего информацию о всех тренировках и запись её в список упражнений. Сообщение об успешном считывании показано на  
рисунке 4. Далее пользователь вводит предпочтительное упражнение с ключевым словом «start», как продемонстрировано на рисунке 5, для того чтобы начать тренировку. После ввода ключевого слова «end» тренировка завершается и на экран выводятся её результаты, что видно из рисунка 6.

Если пользователь желает увидеть сумму сожжённых калорий за всё время, тогда вводится ключевое слово «data» прямо как на рисунке 7. Завершает работу программы ключевое слово «exit», что можно увидеть на рисунке 8.



Рисунок 4 – Успешное считывание бинарного файла

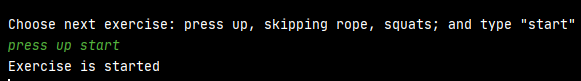


Рисунок 5 – Успешный выбор тренировки и её запуск

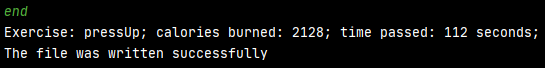


Рисунок 6 – Окончание тренировки

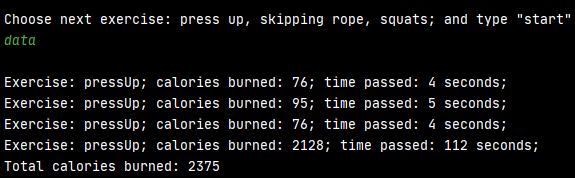


Рисунок 7 – Получение информации о всех тренировках

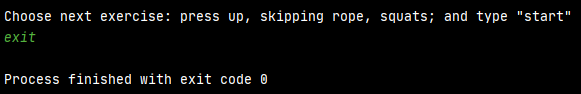


Рисунок 8 – Завершение работы приложения

# **4 Тестирование**

Для тестирования приложения были созданы следующие junit-тесты:

* ExerciseOptionTest – проверка корректного перевода String в формат перечисления ExerciseOption;
* ExerciseDataTest – проверка корректной сериализации и десериализации данных;
* ExerciseTest – проверка корректного подсчёта калорий;

Для проведения тестирования модулей приложения были использованы методы класса Assertions такие, как: assertEquals, assertNotEquals и assertNull. Код данных тестов, как сами программные модули, представлен в приложении. На рисунках 9, 10 и 11 продемонстрированы результаты прошедшего тестирования.

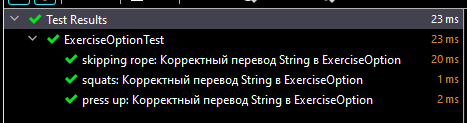


Рисунок 9 – проверка корректного перевода String в ExerciseOption

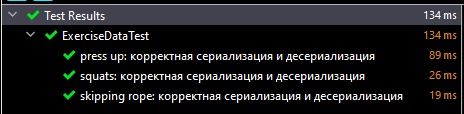


Рисунок 10 – проверка корректной сериализации и десериализации

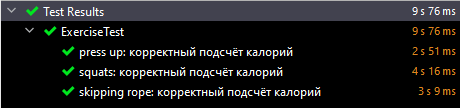


Рисунок 11 – проверка корректного подсчёта калорий

# **5 Вывод**

В результате выполнения курсового проектирования мною были получены навыки по построению диаграммы классов, диаграмм компонентов и диаграмм последовательностей, а также опыт работы по документированию программного продукта.

Во время разработки проекта «Фитнес-трекер» использовалась библиотека java.io и java.util. java.io необходим для хранения бинарных данных в сериализированном виде, а также чтения их и вывод на экран пользователя. java.util используется для подключения списков и ввода с клавиатуры.

Итоговое приложение соответствует заявленным требованиям. Наиболее важные модули программы протестированы и исправно функционируют. Проект пригоден для расширения функционала и дальнейшего сопровождения системным администратором.

# **ПРИЛОЖЕНИЕ**

## **Main**

package lab37**;**import java.util.Scanner**;**public class Main {  
 public static void main(String[] args) {  
 Exercise performingExercise = null**;** //текущее упражнение  
 Scanner scan = new Scanner(System.*in*)**;** //ввод типа упражнения  
 ExerciseData data**;** data = new ExerciseData("data")**;** //журнал упражнений  
 data.readData()**;** //импорт сохранённых данных  
  
 while (true) {  
 if (performingExercise == null) {  
 System.*out*.println("\nChoose next exercise: press up, skipping rope, squats; and type \"start\"")**;** }  
 String input = scan.nextLine()**;** //ввод типа упражнения  
 if (input.contains("start")) {  
 if (performingExercise == null) {  
 performingExercise = Exercise.*start*(input)**;** //начать упражнение  
 System.*out*.println("Exercise is started")**;** } else {  
 System.*out*.println("Other exercise is already in progress. Type \"end\" to finish it.")**;** }  
 } else if (input.contains("end") || input.contains("finish")) {  
 if (performingExercise == null) {  
 System.*out*.print("You are doing nothing")**;** } else {  
 performingExercise.end()**;** //зафиксировать время окончания  
 System.*out*.println(performingExercise)**;** data.addExercise(performingExercise)**;** data.writeData()**;** //протоколирование последнего упражнения  
 performingExercise = null**;** //текущее упражнение обнуляется  
 }  
 } else if (input.contains("quit") || input.contains("exit")) { //завершение приложения  
 break**;** } else if (input.contains("output") || input.contains("data")) { //вывести данные на экран  
 System.*out*.println(data.toString())**;** }  
 }  
 }  
}

## **Exercise**

package lab37**;**import java.io.Serializable**;**class Exercise implements Serializable {  
 ExerciseOption option**;** long timeStart**;** long timeEnd**;** Exercise(ExerciseOption option) {  
 this.option = option**;** this.timeStart = System.*currentTimeMillis*() / **1000;** }  
  
 static Exercise start(String input) {  
 ExerciseOption opt = ExerciseOption.*stringToEnum*(input)**;** //распознавание типа упражнения  
 if (opt != null) {  
 return new Exercise(opt)**;** } else {  
 System.*out*.println("Your option of exercise is not recognised!")**;** return null**;** }  
 }  
  
 //зафиксировать время окончания  
 void end() {  
 timeEnd = System.*currentTimeMillis*() / **1000;** }  
  
 //потраченное время  
 long getPassedTime() {  
 return timeEnd - timeStart**;** }  
  
 //количество сожжённых калорий  
 long getCaloriesBurned() {  
 return getPassedTime() \* option.caloriesPerSec()**;** }  
  
 //вывод информации об упражнении на экран  
 @Override  
 public String toString() {  
 return "Exercise: " + option + "; calories burned: " + getCaloriesBurned() +  
 "; time passed: " + getPassedTime() + " seconds;"**;** }  
}

## **ExerciseData**

package lab37**;**import java.io.\***;**import java.util.ArrayList**;**class ExerciseData implements Serializable {  
 String datafile**;** ArrayList<Exercise> pastExercises**;** ExerciseData(String datafile) {  
 this.datafile = datafile**;** pastExercises = new ArrayList<>()**;** }  
  
 void addExercise(Exercise exercise) {  
 pastExercises.add(exercise)**;** }  
  
 //импорт сохранённых данных  
 void readData() {  
 try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(datafile))) {  
 pastExercises = (ArrayList<Exercise>) ois.readObject()**;** } catch (Exception e) {  
 e.printStackTrace()**;** System.*out*.println("Error: readData")**;** }  
 }  
  
 //экспорт данных  
 void writeData() {  
 try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(datafile))) {  
 oos.writeObject(this.pastExercises)**;** } catch (Exception e) {  
 e.printStackTrace()**;** System.*out*.println("Error: writeData")**;** }  
 }  
  
 //вывод информации об упражнении на экран  
 @Override  
 public String toString() {  
 long totalCaloriesBurned = pastExercises.stream().map(Exercise::getCaloriesBurned).reduce(**0L,** Long::*sum*)**;** return pastExercises.stream().map(Exercise::toString).reduce(""**,** (s1**,** s2) -> s1 + "\n" + s2)  
 + "\nTotal calories burned: " + totalCaloriesBurned**;** }  
}

## **ExerciseOption**

package lab37**;**enum ExerciseOption {  
 *pressUp***,** *skippingRope***,** *squats***;** //количество сожжённых калорий в секунду  
 int caloriesPerSec() {  
 return switch (this) {  
 case *pressUp* -> **19;** case *skippingRope* -> **11;** case *squats* -> **13;** default -> **0;** }**;** }  
  
 //распознавание типа упражнения  
 static ExerciseOption stringToEnum(String text) {  
 if (text.contains("press up") || text.contains("press-up")) {  
 return *pressUp***;** } else if (text.contains("rope") || text.contains("skipping")) {  
 return *skippingRope***;** } else if (text.contains("squats")) {  
 return *squats***;** } else {  
 return null**;** }  
 }  
}

## **ExerciseOptionTest**

package lab37**;**import org.junit.jupiter.api.Assertions**;**import org.junit.jupiter.api.DisplayName**;**import org.junit.jupiter.api.Test**;**import java.util.ArrayList**;**import java.util.Arrays**;**public class ExerciseOptionTest {  
  
 @Test  
 @DisplayName("press up: Корректный перевод String в ExerciseOption")  
 void stringToEnumPressUp() {  
 ArrayList<String> listOptions = new ArrayList<>(Arrays.*asList*("press up start"**,** "press-up start"**,** "start press up"**,** "press start up"**,** "start press-up"**,** "press-start-up"))**;** Assertions.*assertEquals*(ExerciseOption.*pressUp***,** ExerciseOption.*stringToEnum*(listOptions.get(**0**)))**;** Assertions.*assertEquals*(ExerciseOption.*pressUp***,** ExerciseOption.*stringToEnum*(listOptions.get(**1**)))**;** Assertions.*assertEquals*(ExerciseOption.*pressUp***,** ExerciseOption.*stringToEnum*(listOptions.get(**2**)))**;** Assertions.*assertEquals*(null**,** ExerciseOption.*stringToEnum*(listOptions.get(**3**)))**;** Assertions.*assertEquals*(ExerciseOption.*pressUp***,** ExerciseOption.*stringToEnum*(listOptions.get(**4**)))**;** Assertions.*assertEquals*(null**,** ExerciseOption.*stringToEnum*(listOptions.get(**5**)))**;** }  
  
 @Test  
 @DisplayName("skipping rope: Корректный перевод String в ExerciseOption")  
 void stringToEnumSkippingRope() {  
 ArrayList<String> listOptions = new ArrayList<>(Arrays.*asList*("squats start"**,** "start squats"**,** "start quotes"))**;** Assertions.*assertEquals*(ExerciseOption.*squats***,** ExerciseOption.*stringToEnum*(listOptions.get(**0**)))**;** Assertions.*assertEquals*(ExerciseOption.*squats***,** ExerciseOption.*stringToEnum*(listOptions.get(**1**)))**;** Assertions.*assertEquals*(null**,** ExerciseOption.*stringToEnum*(listOptions.get(**2**)))**;** }  
  
 @Test  
 @DisplayName("squats: Корректный перевод String в ExerciseOption")  
 void stringToEnumSquats() {  
 ArrayList<String> listOptions = new ArrayList<>(Arrays.*asList*("skipping rope start"**,** "start skipping rope"**,** "skipping start rope"**,** "skipping-start-rope"**,** "rope start skipping"))**;** Assertions.*assertEquals*(ExerciseOption.*skippingRope***,** ExerciseOption.*stringToEnum*(listOptions.get(**0**)))**;** Assertions.*assertEquals*(ExerciseOption.*skippingRope***,** ExerciseOption.*stringToEnum*(listOptions.get(**1**)))**;** Assertions.*assertEquals*(ExerciseOption.*skippingRope***,** ExerciseOption.*stringToEnum*(listOptions.get(**2**)))**;** Assertions.*assertEquals*(ExerciseOption.*skippingRope***,** ExerciseOption.*stringToEnum*(listOptions.get(**3**)))**;** Assertions.*assertEquals*(ExerciseOption.*skippingRope***,** ExerciseOption.*stringToEnum*(listOptions.get(**4**)))**;** }  
}

## **ExerciseDataTest**

package lab37**;**import org.junit.jupiter.api.Assertions**;**import org.junit.jupiter.api.DisplayName**;**import org.junit.jupiter.api.Test**;**public class ExerciseDataTest {  
  
 @Test  
 @DisplayName("press up: корректная сериализация и десериализация")  
 void serializeAndDeserializePressUp() {  
 String filename1 = "test1"**;** ExerciseData data1**;** data1 = new ExerciseData(filename1)**;** Exercise exer1 = Exercise.*start*("press up")**;** assert exer1 != null**;** exer1.end()**;** data1.addExercise(exer1)**;** data1.writeData()**;** data1.readData()**;** Assertions.*assertEquals*(exer1.option**,** data1.getExercise(**0**).option)**;** Assertions.*assertEquals*(exer1.timeStart**,** data1.getExercise(**0**).timeStart)**;** Assertions.*assertEquals*(exer1.timeEnd**,** data1.getExercise(**0**).timeEnd)**;** }  
  
 @Test  
 @DisplayName("skipping rope: корректная сериализация и десериализация")  
 void serializeAndDeserializeSkippingRope() {  
 String filename2 = "test2"**;** ExerciseData data2 = new ExerciseData(filename2)**;** Exercise exer2 = Exercise.*start*("skipping rope")**;** assert exer2 != null**;** exer2.end()**;** data2.addExercise(exer2)**;** data2.writeData()**;** data2.readData()**;** Assertions.*assertEquals*(exer2.option**,** data2.getExercise(**0**).option)**;** Assertions.*assertEquals*(exer2.timeStart**,** data2.getExercise(**0**).timeStart)**;** Assertions.*assertEquals*(exer2.timeEnd**,** data2.getExercise(**0**).timeEnd)**;** }  
  
 @Test  
 @DisplayName("squats: корректная сериализация и десериализация")  
 void serializeAndDeserializeSquats() {  
 String filename3 = "test3"**;** ExerciseData data3 = new ExerciseData(filename3)**;** Exercise exer3 = Exercise.*start*("squats")**;** assert exer3 != null**;** exer3.end()**;** data3.addExercise(exer3)**;** data3.writeData()**;** data3.readData()**;** Assertions.*assertEquals*(exer3.option**,** data3.getExercise(**0**).option)**;** Assertions.*assertEquals*(exer3.timeStart**,** data3.getExercise(**0**).timeStart)**;** Assertions.*assertEquals*(exer3.timeEnd**,** data3.getExercise(**0**).timeEnd)**;** }  
}

## **ExerciseTest**

package lab37**;**import org.junit.jupiter.api.Assertions**;**import org.junit.jupiter.api.DisplayName**;**import org.junit.jupiter.api.Test**;**public class ExerciseTest {  
 @Test  
 @DisplayName("press up: корректный подсчёт калорий")  
 void calorieCountingPressUp() {  
 Exercise exer = Exercise.*start*("press up")**;** long SleepSeconds = **2;** try  
 {  
 Thread.*sleep*(SleepSeconds \* **1000**)**;** }  
 catch(InterruptedException ex)  
 {  
 Thread.*currentThread*().interrupt()**;** }  
 assert exer != null**;** exer.end()**;** Assertions.*assertEquals*(SleepSeconds \* ExerciseOption.*pressUp*.pressUpCalories**,** exer.getCaloriesBurned())**;** }  
  
 @Test  
 @DisplayName("skipping rope: корректный подсчёт калорий")  
 void calorieCountingSkippingRope() {  
 Exercise exer = Exercise.*start*("skipping rope")**;** long SleepSeconds = **3;** try  
 {  
 Thread.*sleep*(SleepSeconds \* **1000**)**;** }  
 catch(InterruptedException ex)  
 {  
 Thread.*currentThread*().interrupt()**;** }  
 assert exer != null**;** exer.end()**;** Assertions.*assertEquals*(SleepSeconds \* ExerciseOption.*skippingRope*.skippingRopeCalories**,** exer.getCaloriesBurned())**;** }  
  
 @Test  
 @DisplayName("squats: корректный подсчёт калорий")  
 void calorieCountingSquats() {  
 Exercise exer = Exercise.*start*("squats")**;** long SleepSeconds = **4;** try  
 {  
 Thread.*sleep*(SleepSeconds \* **1000**)**;** }  
 catch(InterruptedException ex)  
 {  
 Thread.*currentThread*().interrupt()**;** }  
 assert exer != null**;** exer.end()**;** Assertions.*assertEquals*(SleepSeconds \* ExerciseOption.*squats*.squatsCalories**,** exer.getCaloriesBurned())**;** }  
}