ОТЧЕТ ПО
ЛАБОРАТОРНОЙ РАБОТЕ №6

Выполнил:
Батуров Максим Дмитриевич
Группа: 6203-010302D

Оглавление

Задание №1

В рамках первого задания был разработан метод численного интегрирования в классе Functions. Метод integrate принимает в качестве параметров объект функции, левую и правую границы интегрирования, а также шаг дискретизации. Перед началом вычислений выполняются проверки корректности входных данных. Для вычисления интеграла применяется метод трапеций.

В методе main проведено тестирование работы метода на экспоненциальной функции. Также ручным подбором определен шаг дискретизации.

```
public static double integrate(Function function, double leftBorder, double rightBorder, double step) {  4 usages
    // проверка корректности границ интегрирования
    if (leftBorder >= rightBorder) {
        throw new IllegalArgumentException("левая граница интегрирования должна быть меньше правой");
    }

    // проверка шага дискретизации
    if (step <= 0) {
        throw new IllegalArgumentException("шаг дискретизации должен быть положительным");
    }

    // проверка, что интервал интегрирования входит в область определения
    if (leftBorder < function.getLeftDomainBorder() || rightBorder > function.getRightDomainBorder()) {
        throw new IllegalArgumentException("интервал интегрирования выходит за границы области определения функции");
    }

    double integral = 0.0;
    double currentX = leftBorder;

    // проходим по всем полным шагам
    while (currentX + step < rightBorder) {
        double f1 = function.getFunctionValue(currentX);
        double f2 = function.getFunctionValue( x: currentX + step);
        integral += (f1 + f2) * step / 2.0;
        currentX += step;
    }

    // обрабатываем последний неполный шаг
    if (currentX < rightBorder) {
        double remainingStep = rightBorder - currentX;
        double f1 = function.getFunctionValue(currentX);
        double f2 = function.getFunctionValue(rightBorder);
        integral += (f1 + f2) * remainingStep / 2.0;
    }

    return integral;
}
```

```
теоретическое значение интеграла: 1.718281828459045
шаг: 0.1, интеграл: 1.7197134913893148, разница: 0.0014316629302697503
шаг: 0.05, интеграл: 1.7186397889252205, разница: 3.579604661754221E-4
шаг: 0.025, интеграл: 1.7183713213720626, разница: 8.949291301751927E-5
шаг: 0.0125, интеграл: 1.7183042018620924, разница: 2.2373403047337703E-5
шаг: 0.00625, интеграл: 1.7182874218207358, разница: 5.593361690703347E-6
шаг: 0.003125, интеграл: 1.7182832268001362, разница: 1.3983410911411198E-6
шаг: 0.0015625, интеграл: 1.7182821780443425, разница: 3.4958529737671995E-7
шаг: 7.8125E-4, интеграл: 1.7182819158554286, разница: 8.739638346355605E-8
достигнута требуемая точность при шаге: 7.8125E-4
```

Задание №2

Создан пакет threads, в котором размещен класс Task. Данный класс содержит поля для хранения ссылки на интегрируемую функцию, границ области интегрирования, шага дискретизации и количества выполняемых заданий. Реализован метод nonThread. В цикле генерируется логарифмическая функция со случайным основанием. Устанавливаются случайные границы интегрирования b случайный шаг дискретизации.

```java
public static void nonThread() {  1 usage
    // создаем объект класса Task
    Task task = new Task();

    // устанавливаем количество выполняемых заданий
    task.setTasksCount(100);

    // создаем генератор случайных чисел
    Random random = new Random();

    // выполняем задания в цикле
    for (int i = 0; i < task.getTasksCount(); i++) {
        // создаем объект логарифмической ф-ции со случайным основанием от 1 до 10
        double base = 1 + random.nextDouble() * 9; // от 1 до 10
        Log logFunction = new Log(base);
        task.setFunction(logFunction);

        // левая граница области интегрирования
        double leftBorder = random.nextDouble() * 100;
        task.setLeftBorder(leftBorder);

        // правая граница области интегрирования
        double rightBorder = 100 + random.nextDouble() * 100;
        task.setRightBorder(rightBorder);

        // шаг дискретизации
        double step = random.nextDouble();
        task.setStep(step);

        System.out.printf("Source %.4f %.4f %.4f\n",
                task.getLeftBorder(), task.getRightBorder(), task.getStep());

        // вычисляем значение интеграла для параметров из объекта задания
        double result = Functions.integrate(task.getFunction(),
                task.getLeftBorder(), task.getRightBorder(), task.getStep());

        System.out.printf("Result %.4f %.4f %.4f %.4f\n",
                task.getLeftBorder(), task.getRightBorder(), task.getStep(), result);
    }
}
```

```
****** Тест nonThread ******

Source 66,6738 190,4525 0,7620
Result 66,6738 190,4525 0,7620 490,3792
Source 68,6137 195,8824 0,7273
Result 68,6137 195,8824 0,7273 360,0335
Source 39,9031 128,8917 0,7934
Result 39,9031 128,8917 0,7934 176,2216
Source 28,5451 102,1992 0,1959
Result 28,5451 102,1992 0,1959 201,9755
Source 27,2185 100,0168 0,5521
Result 27,2185 100,0168 0,5521 662,6461
Source 72,0559 142,1082 0,7265
Result 72,0559 142,1082 0,7265 195,2983
Source 61,3992 184,2588 0,0381
Result 61,3992 184,2588 0,0381 289,2704
Source 23,7813 109,3343 0,6514
Result 23,7813 109,3343 0,6514 195,1348
Source 37,1405 152,2650 0,1078
Result 37,1405 152,2650 0,1078 249,6732
```

Задание №3

В пакете threads созданы классы SimpleGenerator и SimpleIntegrator. Класс SimpleGenerator формирует задания и записывает их в объект Task, а SimpleIntegrator выполняет чтение заданий и вычисляет интегралы.

В ходе тестов обнаружено, что в некоторых случаях возникало исключение NullPointerException при обращении интегратора к функции до ее установки генератором. Происходило так, что интегратор мог получить несогласованные данные.

Чтобы избавиться от недочетов были добавлены блоки синхронизации.

```
package threads;

import functions.Functions;

public class SimpleIntegrator implements Runnable {  1 usage
    private Task task;  10 usages

    public SimpleIntegrator(Task task) {  1 usage
        this.task = task;
    }

    public void run() {
        for (int i = 0; i < task.getTasksCount(); i++) {
            synchronized (task) {
                // вычисляем значение интеграла для параметров из объекта задания
                double result = Functions.integrate(task.getFunction(),
                        task.getLeftBorder(), task.getRightBorder(), task.getStep());

                System.out.printf("Result %.4f %.4f %.4f %.4f\n",
                        task.getLeftBorder(), task.getRightBorder(), task.getStep(), result);
            }

            // пауза чтобы генерирующий поток положил новые данные
            try {
                Thread.sleep( millis: 1);
            } catch (InterruptedException e) {
                System.out.println("интегрирующий поток был прерван");
                return;
            }
        }
    }
}
```

```java
public class SimpleGenerator implements Runnable {  1 usage
    private Task task;  10 usages

    public SimpleGenerator(Task task) {  1 usage
        this.task = task;
    }

    public void run() {
        Random random = new Random();

        for (int i = 0; i < task.getTasksCount(); i++) {
            synchronized (task) {
                // создаем объект логарифмической функции со случайным основанием от 1 до 10
                double base = 1 + random.nextDouble() * 9;
                Log logFunction = new Log(base);
                task.setFunction(logFunction);

                // левая граница области интегрирования (от 0 до 100)
                double leftBorder = random.nextDouble() * 100;
                task.setLeftBorder(leftBorder);

                // правая граница области интегрирования (от 100 до 200)
                double rightBorder = 100 + random.nextDouble() * 100;
                task.setRightBorder(rightBorder);

                // шаг дискретизации (от 0 до 1)
                double step = random.nextDouble();
                task.setStep(step);

                System.out.printf("Source %.4f %.4f %.4f\n",
                        task.getLeftBorder(), task.getRightBorder(), task.getStep());
            }

            // пауза чтобы интегрирующий поток забрал данные
            try {
                Thread.sleep( millis: 1);
            } catch (InterruptedException e) {
                System.out.println("генерирующий поток был прерван");
                return;
            }
        }
    }
```

```java
// метод с многопоточной версией программы
public static void simpleThreads() {  1 usage
    // создаем объект задания
    Task task = new Task();

    // устанавливаем количество выполняемых заданий
    task.setTasksCount(100);

    // создаем потоки
    Thread generatorThread = new Thread(new SimpleGenerator(task));
    Thread integratorThread = new Thread(new SimpleIntegrator(task));

    // запускаем потоки
    generatorThread.start();
    integratorThread.start();

    // ждем завершения потоков
    try {
        generatorThread.join();
        integratorThread.join();
    } catch (InterruptedException e) {
        System.out.println("потоки были прерваны");
    }
}
```

```
****** Тест simpleThreads ******

Source 64,1355 112,5631 0,5025
Result 64,1355 112,5631 0,5025 104,5127
Source 32,6640 157,5352 0,6441
Result 32,6640 157,5352 0,6441 397,4332
Source 74,3115 127,9198 0,6449
Result 74,3115 127,9198 0,6449 1728,3246
Source 31,2673 188,7899 0,3735
Result 31,2673 188,7899 0,3735 391,4108
Source 49,3893 167,4681 0,4970
Source 85,7438 168,2572 0,7481
Result 85,7438 168,2572 0,7481 184,6260
Result 85,7438 168,2572 0,7481 184,6260
Source 89,0651 113,7313 0,0052
Result 89,0651 113,7313 0,0052 54,2774
Source 23,3984 173,4036 0,9953
Source 90,4491 142,2521 0,6785
Result 90,4491 142,2521 0,6785 150,7006
Source 94,5363 161,3653 0,1413
Result 94,5363 161,3653 0,1413 147,3774
Result 94,5363 161,3653 0,1413 147,3774
Source 85,5552 130,8818 0,1371
```

Задание №4

Обнаружена проблема неполного выполнения заданий, связанная с недостаточной синхронизацией потоков. Для решения этой проблемы созданы классы Generator и Integrator.

Классы получают в конструкторе ссылки на объект Task и семафор. В методе run реализована логика взаимодействия потоков через семафор, обеспечивающая корректное выполнение всех заданий. Добавлена обработка прерывания потоков, позволяющая корректно завершать их работу по истечении заданного времени.

```
****** Тест complicatedThreads ******

Source 71,1879 134,1090 0,3512
Source 70,9809 187,4201 0,1412
Source 61,2394 140,6875 0,6354
Source 52,9524 152,9495 0,2175
Source 62,1519 131,8162 0,5189
Source 15,0264 179,0858 0,7958
Source 20,0882 135,4876 0,0087
Source 3,6872 143,0349 0,2155
Source 60,7287 127,9450 0,0859
Source 81,1466 169,6824 0,5730
Source 74,4973 162,4002 0,0377
Source 29,9823 193,9081 0,4080
Source 17,2735 164,2615 0,2817
Result 71,1879 134,1090 0,3512 129,6131
Source 54,7071 141,1690 0,2283
Result 54,7071 141,1690 0,2283 206,7370
Result 45,7279 102,6371 0,9349 358,0478
Result 45,7279 102,6371 0,9349 358,0478
Result 45,7279 102,6371 0,9349 358,0478
Result 45,7279 102,6371 0,9349 358,0478
Result 45,7279 102,6371 0,9349 358,0478
Result 45,7279 102,6371 0,9349 358,0478
Result 45,7279 102,6371 0,9349 358,0478
Result 45,7279 102,6371 0,9349 358,0478
Result 45,7279 102,6371 0,9349 358,0478
Result 45,7279 102,6371 0,9349 358,0478
Result 45,7279 102,6371 0,9349 358,0478
Result 45,7279 102,6371 0,9349 358,0478
Result 45,7279 102,6371 0,9349 358,0478
```