

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ «САМАРСКИЙ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ имени
академика С.П. КОРОЛЁВА»

КАФЕДРА «ТЕХНИЧЕСКАЯ КИБЕРНЕТИКА»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

«Объектно-ориентированное программирование»

ЛАБОРАТОРНАЯ РАБОТА НОМЕР 5

Студент _____ Горейнов Д.В.

Группа _____ 6301-030301D

Проверил _____ Борисов Д.С.

Оценка _____

Задание 1

Добавил в класс Functions метод, возвращающий значение интеграла функции, вычисленное с помощью численного метода.

Вычисление интеграла выполняется по методу трапеций.

Основная формула: $\frac{(x_2-x_1)(y_1+y_2)}{2}$, где x_2-x_1 является высотой трапеции, а $\frac{(y_1+y_2)}{2}$ является полусуммой оснований.

```
public static double integrate(Function function, double leftBorder, double rightBorder, double step)
{
    if (leftBorder >= rightBorder)
    {
        throw new IllegalArgumentException("Левая граница должна быть меньше правой: leftBorder=" + leftBorder + ", rightBorder=" + rightBorder);
    }
    if (step <= 0)
    {
        throw new IllegalArgumentException("Шаг должен быть положительным: step=" + step);
    }

    // Проверка области определения
    if (leftBorder < function.getLeftDomainBorder() || rightBorder > function.getRightDomainBorder())
    {
        throw new IllegalArgumentException(String.format("Интервал [%2f, %2f] выходит за границы области определения [%2f, %2f]", leftBorder, rightBorder, function.getLeftDomainBorder(), function.getRightDomainBorder()));
    }

    double integral = 0;
    double x = leftBorder;
    int n = (int) Math.ceil((rightBorder - leftBorder) / step);
    for (int i = 0; i < n; i++)
    {
        double x1 = x;
        double x2 = (i == n - 1) ? rightBorder : Math.min(x + step, rightBorder);
        double y1 = function.getFunctionValue(x1);
        double y2 = function.getFunctionValue(x2);
        double S = (y1 + y2) * (x2 - x1) / 2.0;
        integral += S;
        x = x2;
    }
    return integral;
}
```

Фото 1

Была проведена проверка метода интегрирования:

```
public static void main(String[] args)
{
    Function expFunction = new Exp();
    System.out.println("Тест 1");
    double accuracy = 1e-7;
    double left = 0;
    double right = 1;
    double equivalentValue = Math.E - 1;
    double step = 0.00001;

    double integral = Functions.integrate(expFunction, left, right, step);

    System.out.printf("Вычисленный интеграл: %.10f\n", integral);
    System.out.printf("Теоретическое значение: %.10f\n", equivalentValue);
    System.out.printf("Разница: %.2e\n", Math.abs(integral - equivalentValue));

    if (Math.abs(integral - equivalentValue) <= accuracy)
    {
        System.out.println("Необходимая точность достигнута, интеграл найден верно");
    }
    else
    {
        System.out.println("Необходимая точность не достигнута");
    }
}
```

Фото 2

```
Тест 1
Вычисленный интеграл: 1,7182818285
Теоретическое значение: 1,7182818285
Разница: 1,43e-11
Необходимая точность достигнута, интеграл найден верно
```

Фото 3

Оптимальный шаг для достижения необходимой точности равен 0.0001

Следовательно метод работает корректно и может быть использован в дальнейшей работе.

Задание 2

Создал пакет threads, в котором размещены классы, связанные с потоками.

1)В пакете threads описан класс Task, объект которого должен хранить ссылку на объект интегрируемой функции, границы области интегрирования, шаг дискретизации, а также целочисленное поле, хранящее количество выполняемых заданий.

```

public class Task
{
    private Function function;
    private double leftBorder;
    private double rightBorder;
    private double step;
    private int taskCount;

    public Task()
    {
        this.taskCount = 0;
    }

    public Task(int taskCount)
    {
        this.taskCount = taskCount;
    }

    public Function getFunction() {
        return function;
    }

    public double getLeftBorder() {
        return leftBorder;
    }

    public double getRightBorder() {
        return rightBorder;
    }

    public double getStep() {
        return step;
    }

    public int getTaskCount() {
        return taskCount;
    }

    public void setFunction(Function function) {
        this.function = function;
    }

    public void setLeftBorder(double leftBorder) {
        this.leftBorder = leftBorder;
    }

    public void setRightBorder(double rightBorder) {
        this.rightBorder = rightBorder;
    }

    public void setStep(double step) {
        this.step = step;
    }

    public void setTaskCount(int taskCount) {
        this.taskCount = taskCount;
    }
}

```

2) В main создал метод nonThread() реализующий последовательную (без применения потоков инструкций) версию программы:

Random-генерирует случайные числа

Для основания воспользовался циклом do while, чтобы избежать основания равному 1

Метод nextDouble() возвращает псевдослучайное число от 0.0 до 1.0. поэтому мы умножаем результат на 100

```
public static void nonThread()
{
    Random random = new Random();
    Task task = new Task();
    task.setTaskCount(taskCount: 100);

    System.out.println("ПОСЛЕДОВАТЕЛЬНАЯ ОБРАБОТКА ЗАДАНИЙ");

    for (int i = 0; i < task.getTaskCount(); i++) {
        try {
            double base;
            do {
                base = 1 + random.nextDouble() * 9; // [1, 10]
            } while (Math.abs(base - 1.0) < 1e-7); // избегаем основание = 1
            Function logFunction = new Log(base);
            task.setFunction(logFunction);

            double leftBorder = random.nextDouble() * 100; // [0, 100]
            task.setLeftBorder(leftBorder);

            double rightBorder = 100 + random.nextDouble() * 100; // [100, 200]
            task.setRightBorder(rightBorder);

            // Шаг дискретизации от 0 до 1
            double step;
            do {
                step = random.nextDouble(); // [0, 1]
            } while (step == 0.0); // избегаем нулевой шаг
            task.setStep(step);

            // Вывод информации о задании
            System.out.printf("Source %d: %.6f %.6f %.6f\n", i + 1, task.getLeftBorder(), task.getRightBorder(), task.getStep());
            double integral = Functions.integrate(task.getFunction(), task.getLeftBorder(), task.getRightBorder(), task.getStep());
            System.out.printf("Result %d: %.6f %.6f %.6f %.6f\n", i + 1, task.getLeftBorder(), task.getRightBorder(), task.getStep(), integral);
            System.out.println("_____");

        } catch (IllegalArgumentException e) {
            System.out.println("Ошибка в задании " + (i + 1) + ": " + e.getMessage());
            System.out.println("Повторяем задание...");
            i--; // Повторяем это задание
        }
    }

    System.out.println("Все задания выполнены!");
}
```

Фото 5

Была проведена проверка, метод работает верно, все задачи были решены

```
Source 43: 56,140696 190,725122 0,621940
Result 43: 56,140696 190,725122 0,621940 350,347430
-----
Source 44: 10,430328 154,609977 0,022784
Result 44: 10,430328 154,609977 0,022784 315,865883
-----
Source 45: 6,052598 163,422703 0,193168
Result 45: 6,052598 163,422703 0,193168 1344,327231
-----
Source 46: 43,550108 128,848986 0,950873
Result 46: 43,550108 128,848986 0,950873 186,280783
-----
Source 47: 9,238841 194,517268 0,581468
Result 47: 9,238841 194,517268 0,581468 431,773947
-----
Source 48: 74,744525 119,186018 0,466627
Result 48: 74,744525 119,186018 0,466627 100,334118
-----
Source 49: 45,630448 188,603895 0,844879
Result 49: 45,630448 188,603895 0,844879 362,331315
-----
Source 50: 29,041624 105,972598 0,334173
Result 50: 29,041624 105,972598 0,334173 139,101442
-----
Source 51: 52,106661 151,664882 0,104501
Result 51: 52,106661 151,664882 0,104501 229,311454
-----
Source 52: 77,780053 121,697865 0,878822
Result 52: 77,780053 121,697865 0,878822 98,428364
-----
Source 53: 32,874065 151,434373 0,261826
Result 53: 32,874065 151,434373 0,261826 1061,557448
-----
Source 54: 84,220732 106,082815 0,248623
Result 54: 84,220732 106,082815 0,248623 46,686869
-----
Source 55: 73,220963 172,788615 0,149235
Result 55: 73,220963 172,788615 0,149235 384,225036
-----
Source 56: 46,479083 120,457992 0,579781
Result 56: 46,479083 120,457992 0,579781 163,085552
-----
Source 57: 0,787609 102,927484 0,617574
Result 57: 0,787609 102,927484 0,617574 435,078129
```

Фото 6

```
-----
Source 88: 84,668431 120,883861 0,827999
Result 88: 84,668431 120,883861 0,827999 298,614965

-----
Source 89: 17,973209 107,336996 0,670725
Result 89: 17,973209 107,336996 0,670725 557,701313

-----
Source 90: 91,080622 117,167696 0,758329
Result 90: 91,080622 117,167696 0,758329 58,616841

-----
Source 91: 40,446749 199,391161 0,066235
Result 91: 40,446749 199,391161 0,066235 531,085738

-----
Source 92: 84,992921 162,590439 0,184034
Result 92: 84,992921 162,590439 0,184034 362,870323

-----
Source 93: 94,613846 167,762241 0,343567
Result 93: 94,613846 167,762241 0,343567 349,100689

-----
Source 94: 96,391012 105,856728 0,542216
Result 94: 96,391012 105,856728 0,542216 21,719355

-----
Source 95: 63,924674 186,210674 0,973840
Result 95: 63,924674 186,210674 0,973840 367,250909

-----
Source 96: 87,638949 118,420957 0,644815
Result 96: 87,638949 118,420957 0,644815 63,965662

-----
Source 97: 29,481112 104,678034 0,748959
Result 97: 29,481112 104,678034 0,748959 166,913684

-----
Source 98: 24,127458 172,176745 0,102627
Result 98: 24,127458 172,176745 0,102627 810,885965

-----
Source 99: 0,624617 165,944365 0,700063
Result 99: 0,624617 165,944365 0,700063 1320,340458

-----
Source 100: 63,336522 191,273571 0,239150
Result 100: 63,336522 191,273571 0,239150 902,459769

-----
Все задания выполнены!
```

Фото 7

Задание 3

В пакете threads создал два следующих класса.

1)Класс SimpleGenerator реализовывает интерфейс Runnable, получает в конструкторе и сохраняет в своём поле ссылку на объект типа Task, а в методе run() в цикле формируются задачи и заносятся в полученный объект задания, а также выводятся сообщения в консоль.

```

package functions.threads;
import functions.basic.*;
import java.util.Random;
import functions.*;
public class SimpleGenerator implements Runnable {
    private Task task;

    public SimpleGenerator(Task task) {
        this.task = task;
    }
    Loading...
    @Override
    public void run() {
        Random random = new Random();
        int taskCount = task.getTaskCount();

        System.out.println("Generator started: " + taskCount + " tasks");

        for (int i = 0; i < taskCount; i++) {
            // Формируем параметры задачи
            double base;
            do {
                base = 1 + random.nextDouble() * 9; // [1, 10]
            } while (Math.abs(base - 1.0) < 1e-10);

            Function logFunction = new Log(base);
            double leftBorder = random.nextDouble() * 100; // [0, 100]
            double rightBorder = leftBorder + 10 + random.nextDouble() * 90; // [left+10, left+100]
            double step;
            do {
                step = random.nextDouble(); // [0, 1]
            } while (Math.abs(step) < 1e-10);

            // Синхронизированная установка всех параметров задачи
            synchronized (task) {
                task.setFunction(logFunction);
                task.setLeftBorder(leftBorder);
                task.setRightBorder(rightBorder);
                task.setStep(step);
            }

            System.out.printf("Generator: Task %d - function: log(%4f), [%4f, %4f], step: %6f\n", i + 1, base, leftBorder, rightBorder, step);

            try {
                Thread.sleep(10); // Небольшая задержка
            } catch (InterruptedException e) {
                System.out.println("Generator interrupted");
                return;
            }
        }

        System.out.println("Generator finished");
    }
}

```

Фото 8

2) Класс SimpleIntegrator реализовывает интерфейс Runnable, получает в конструкторе и сохраняет в своём поле ссылку на объект типа Task, а в методе run() в цикле решаются задачи, данные для которых берутся из полученного объекта задания, а также выводятся сообщения в консоль.


```

package functions.threads;

import functions.Function;
import functions.Functions;

public class SimpleIntegrator implements Runnable {
    private Task task;

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

    @Override
    public void run() {
        int tasksProcessed = 0;
        int taskCount = task.getTaskCount();

        System.out.println("Integrator started");

        while (tasksProcessed < taskCount) {
            Function function = null;
            double leftBorder, rightBorder, step;

            // Синхронизированное чтение всех параметров задачи
            synchronized (task) {
                function = task.getFunction();
                leftBorder = task.getLeftBorder();
                rightBorder = task.getRightBorder();
                step = task.getStep();
            }

            // Проверка на null для избежания NullPointerException
            if (function == null) {
                try {
                    Thread.sleep(1); // Короткая пауза если данные еще не готовы
                } catch (InterruptedException e) {
                    System.out.println("Integrator interrupted");
                    return;
                }
                continue;
            }

            try {
                // Используем метод integrate из класса Functions
                double integral = Functions.integrate(function, leftBorder, rightBorder, step);

                System.out.printf("Integrator: Task %d - [%.4f, %.4f], step: %.6f, result: %.6f\n",
                    tasksProcessed + 1, leftBorder, rightBorder, step, integral);

                tasksProcessed++;
            } catch (IllegalArgumentException e) {
                System.out.printf("Integrator: Invalid task %d - %s\n",
                    tasksProcessed + 1, e.getMessage());
                tasksProcessed++; // Все равно считаем задачу обработанной
            }

            try {
                Thread.sleep(10); // Небольшая задержка
            } catch (InterruptedException e) {
                System.out.println("Integrator interrupted");
                return;
            }
        }

        System.out.println("Integrator finished: " + tasksProcessed + " tasks processed");
    }
}

```

Фото 9

В main создал метод simpleThreads() для проверки.

В нём создал объект задания, указал количество выполняемых заданий в размере 100 штук, создал и запустил два потока вычислений, основанных на описанных

классах SimpleGenerator и SimpleIntegrator.

```
public static void simpleThreads() {
    Task task = new Task(taskCount: 100);
    // Создаем потоки
    Thread generatorThread = new Thread(new SimpleGenerator(task));
    Thread integratorThread = new Thread(new SimpleIntegrator(task));
    // Эксперименты с приоритетами
    // одинаковые приоритеты
    generatorThread.setPriority(Thread.NORM_PRIORITY);
    integratorThread.setPriority(Thread.NORM_PRIORITY);
    // разные приоритеты_1
    // generatorThread.setPriority(Thread.MAX_PRIORITY);
    // integratorThread.setPriority(Thread.MIN_PRIORITY);
    // разные приоритеты_2
    // generatorThread.setPriority(Thread.MIN_PRIORITY);
    // integratorThread.setPriority(Thread.MAX_PRIORITY);

    System.out.println("Starting threads with priorities - " + "Generator: " + generatorThread.getPriority() + ", Integrator: " + integratorThread.getPriority());

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

    // Ожидаем завершения потоков
    try {
        generatorThread.join();
        integratorThread.join();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    System.out.println("All threads completed");
}
```

Фото 10

Результаты запуска:

```
Generator started: 100 tasks
Generator: Task 1 - function: log(6,6457), [57,1415, 87,3743], step: 0,048554
Generator: Task 2 - function: log(2,2811), [71,3244, 106,5702], step: 0,290731
Integrator: Task 1 - [57,1415, 87,3743], step: 0,048554, result: 68,205862
Integrator: Task 2 - [71,3244, 106,5702], step: 0,290731, result: 191,537662
Generator: Task 3 - function: log(7,8231), [21,2525, 100,5752], step: 0,795554
Integrator: Task 3 - [21,2525, 100,5752], step: 0,795554, result: 155,298172
Generator: Task 4 - function: log(6,0707), [7,2616, 62,4025], step: 0,933213
Integrator: Task 4 - [7,2616, 62,4025], step: 0,933213, result: 104,465601
Generator: Task 5 - function: log(2,0563), [29,9701, 97,8363], step: 0,659453
Integrator: Task 5 - [29,9701, 97,8363], step: 0,659453, result: 386,527445
Generator: Task 6 - function: log(2,3982), [73,6226, 86,0858], step: 0,890698
Integrator: Task 6 - [73,6226, 86,0858], step: 0,890698, result: 62,395686
Generator: Task 7 - function: log(5,4065), [85,9439, 135,3381], step: 0,182059
Generator: Task 8 - function: log(1,7720), [52,7472, 72,6621], step: 0,652100
Integrator: Task 7 - [85,9439, 135,3381], step: 0,182059, result: 137,500368
Integrator: Task 8 - [52,7472, 72,6621], step: 0,652100, result: 143,909173
Generator: Task 9 - function: log(7,7569), [83,6089, 108,0560], step: 0,919527
Generator: Task 10 - function: log(6,4236), [96,1136, 186,2106], step: 0,669515
Integrator: Task 9 - [83,6089, 108,0560], step: 0,919527, result: 54,416074
Integrator: Task 10 - [96,1136, 186,2106], step: 0,669515, result: 238,923173
Generator: Task 11 - function: log(9,8557), [43,8823, 74,6007], step: 0,422576
Integrator: Task 11 - [10,4876, 106,7799], step: 0,351157, result: 219,596913
Integrator: Task 12 - [43,8823, 74,6007], step: 0,422576, result: 54,644536
Generator: Task 13 - function: log(6,3423), [24,1350, 95,3977], step: 0,506521
Integrator: Task 13 - [24,1350, 95,3977], step: 0,506521, result: 155,219210
Generator: Task 14 - function: log(8,3615), [26,5256, 50,0394], step: 0,796818
Integrator: Task 14 - [26,5256, 50,0394], step: 0,796818, result: 40,179326
Generator: Task 15 - function: log(4,7140), [73,8638, 161,3750], step: 0,064809
Generator: Task 16 - function: log(7,9566), [30,1844, 92,0905], step: 0,654991
Integrator: Task 15 - [73,8638, 161,3750], step: 0,064809, result: 267,711931
Integrator: Task 16 - [30,1844, 92,0905], step: 0,654991, result: 121,383653
Generator: Task 17 - function: log(5,3961), [61,3920, 159,8592], step: 0,930317
Integrator: Task 17 - [61,3920, 159,8592], step: 0,930317, result: 272,848927
Generator: Task 18 - function: log(3,4155), [32,3917, 77,3462], step: 0,730196
Integrator: Task 18 - [32,3917, 77,3462], step: 0,730196, result: 145,492833
Generator: Task 19 - function: log(7,1360), [59,5535, 79,3385], step: 0,537975
Integrator: Task 19 - [59,5535, 79,3385], step: 0,537975, result: 42,659179
Generator: Task 20 - function: log(7,3530), [39,8054, 137,4306], step: 0,219779
Integrator: Task 20 - [39,8054, 137,4306], step: 0,219779, result: 216,689408
Generator: Task 21 - function: log(9,1445), [33,2118, 64,2846], step: 0,392821
Integrator: Task 21 - [33,2118, 64,2846], step: 0,392821, result: 54,323740
Generator: Task 22 - function: log(2,5424), [84,5881, 164,5146], step: 0,708869
Integrator: Task 22 - [84,5881, 164,5146], step: 0,708869, result: 411,753045
Generator: Task 23 - function: log(4,1835), [31,5176, 127,7105], step: 0,597333
Integrator: Task 23 - [31,5176, 127,7105], step: 0,597333, result: 289,572264
Generator: Task 24 - function: log(7,9125), [61,0778, 121,0229], step: 0,302472
Integrator: Task 24 - [61,0778, 121,0229], step: 0,302472, result: 130,202526
Generator: Task 25 - function: log(5,8945), [45,6517, 84,0703], step: 0,370516
Integrator: Task 25 - [45,6517, 84,0703], step: 0,370516, result: 90,029672
Generator: Task 26 - function: log(9,9141), [9,3876, 47,7103], step: 0,095033
Integrator: Task 26 - [9,3876, 47,7103], step: 0,095033, result: 54,517959
Generator: Task 27 - function: log(7,1881), [51,1921, 140,4334], step: 0,812660
Integrator: Task 27 - [51,1921, 140,4334], step: 0,812660, result: 204,668191
Generator: Task 28 - function: log(8,1973), [77,5762, 101,7908], step: 0,356363
Integrator: Task 28 - [77,5762, 101,7908], step: 0,356363, result: 51,716579
Generator: Task 29 - function: log(2,7577), [79,2480, 131,3620], step: 0,419667
Integrator: Task 29 - [79,2480, 131,3620], step: 0,419667, result: 238,710131
Generator: Task 30 - function: log(3,7916), [19,0297, 74,2952], step: 0,062848
Generator: Task 31 - function: log(4,0359), [85,4994, 100,4201], step: 0,100862
Integrator: Task 30 - [19,0297, 74,2952], step: 0,062848, result: 156,620667
Integrator: Task 31 - [85,4994, 100,4201], step: 0,100862, result: 48,455858
Generator: Task 32 - function: log(3,2297), [6,6947, 53,3397], step: 0,652832
Integrator: Task 32 - [6,6947, 53,3397], step: 0,652832, result: 130,277536
Generator: Task 33 - function: log(4,1591), [11,2844, 88,7456], step: 0,704185
Integrator: Task 33 - [11,2844, 88,7456], step: 0,704185, result: 205,769971
Generator: Task 34 - function: log(6,3072), [70,8049, 167,4841], step: 0,135097
Generator: Task 35 - function: log(5,7565), [55,8742, 148,6149], step: 0,302672
```

Фото 11

Экспериментально было найдено лучшее соотношение:

Максимальный приоритет на интегратор и минимальный на генератор
Предварительно в интеграторе повысил время до `Thread.sleep(10);`

Задание 4

Определите причину того, что не все сгенерированные задания оказываются выполнены интегрирующим потоком:

генератор записывает данные в Task, интегратор читает эти данные ,но нет механизма подтверждения, что он читает именно этот номер задания. То есть интегратор может решить одну и ту же задачу дважды, при этом пропустив другую.

Есть несколько вариантов исправления, для примера можно сделать флажки(сколько задач создано и сколько прочитано) или семафор.

Семафор работает как светофор:

Семафор:

- Красный (`writers > 0`): "Идет запись" - все читатели ждут
- Желтый (`writeRequests > 0`): "Очередь на запись" - новые читатели ждут
- Зеленый (`readers > 0`): "Идет чтение" - новые писатели ждут

(Было решено использовать стандартную реализацию семафора)

В пакете `threads` создал два следующих класса. При их реализации воспользовался фрагментами классов из предыдущего задания.

Класс `Generator` расширяет класс `Thread`, получает в конструкторе и сохраняет в свои поля ссылки на объект типа `Task` и на объект семафора, а в методе `run()` выполняются те же действия, что и в предыдущей версии генерирующего класса.

```

package functions.threads;
import functions.basic.*;
import java.util.Random;
import java.util.concurrent.Semaphore;
public class Generator extends Thread {
    private Task task;
    Semaphore dataReady;
    Semaphore dataProcessed;

    public Generator(Task task, Semaphore dataReady, Semaphore dataProcessed)
    {
        this.task = task;
        this.dataReady = dataReady;
        this.dataProcessed = dataProcessed;
        this.setName("Generator-Thread");
    }

    @Override
    public void run() {
        Random random = new Random();
        int taskCount = task.getTaskCount();

        System.out.println(getName() + " started: " + taskCount + " tasks");

        try {
            for (int i = 0; i < taskCount && !isInterrupted(); i++) {
                try {
                    dataProcessed.acquire();

                    if (isInterrupted())
                    {
                        // Освобождаем семафор для завершения другого потока
                        dataReady.release();
                        break;
                    }

                    // Генерируем параметры
                    double base;
                    do {
                        base = 1 + random.nextDouble() * 9;
                    } while (Math.abs(base - 1.0) < 1e-10);

                    Log logFunc = new Log(base);

                    double leftBorder = random.nextDouble() * 100;
                    double rightBorder;
                    do {
                        rightBorder = leftBorder + 1 + random.nextDouble() * 100;
                    } while (rightBorder <= leftBorder);

                    double step;
                    do {
                        step = random.nextDouble();
                    } while (Math.abs(step) < 1e-10);

                    if (leftBorder <= 0) {
                        leftBorder = 0.1;
                        rightBorder = Math.max(rightBorder, leftBorder + 1);
                    }

                    try {
                        task.setFunction(logFunc);
                        task.setLeftBorder(leftBorder);
                        task.setRightBorder(rightBorder);
                        task.setStep(step);

                        System.out.printf("%s %d: Source %.6f %.6f %.6f (base=%.4f)%n",
                            getName(), i + 1,
                            leftBorder, rightBorder, step, base);
                    }
                }
            }
        }
    }
}

```

Фото 13

```

        leftborder, rightborder, step, base);
    } finally {
        dataReady.release();
    }

    // Пауза
    try {
        Thread.sleep(10);
    } catch (InterruptedException e) {
        System.out.println(getName() + " interrupted during sleep");
        Thread.currentThread().interrupt();
        break; // Выходим из цикла
    }

    catch (InterruptedException e) {
        System.out.println(getName() + " interrupted in semaphore operation");
        break; // Выходим из цикла
    } catch (Exception e) {
        System.out.println(getName() + " error: " + e.getMessage());
        dataProcessed.release();
        i--; // Повторяем итерацию
    }
} finally {
    System.out.println(getName() + " finished");
}
}
}

```

Фото 14

Аналогично сделал Integrator

```

public class Integrator extends Thread
{
    private Task task;
    private Semaphore dataReady;
    private Semaphore dataProcessed;

    public Integrator(Task task, Semaphore dataReady, Semaphore dataProcessed)
    {
        this.task = task;
        this.dataReady = dataReady;
        this.dataProcessed = dataProcessed;
        this.setName("Integrator-Thread");
    }

    @Override
    public void run() {
        int tasksProcessed = 0;
        int taskCount = task.getTaskCount();

        System.out.println(getName() + " started");

        try {
            while (tasksProcessed < taskCount && !isInterrupted())
            {
                try {
                    // Используем семафор для чтения
                    dataReady.acquire();
                    if (isInterrupted()) {
                        // Освобождаем семафор для завершения другого потока
                        dataProcessed.release();
                        break;
                    }

                    Function function = task.getFunction();
                    double leftBorder = task.getLeftBorder();
                    double rightBorder = task.getRightBorder();
                    double step = task.getStep();

                    try {
                        double integral = Functions.integrate(function, leftBorder, rightBorder, step);
                        System.out.printf("%s: Task %d - [%.4f, %.4f], step: %.6f, result: %.6f\n", getName(), tasksProcessed + 1, leftBorder, rightBorder, step, integral);

                        tasksProcessed++;

                    } catch (IllegalArgumentException e) {
                        System.out.printf("%s: Invalid task %d - %s\n", getName(), tasksProcessed + 1, e.getMessage());
                        tasksProcessed++;
                    }

                    dataProcessed.release();
                    // Пауза между задачами
                    try {
                        Thread.sleep(10);
                    } catch (InterruptedException e) {
                        System.out.println(getName() + " interrupted between tasks");
                        Thread.currentThread().interrupt();
                        break; // Выходим из цикла
                    }
                } catch (InterruptedException e) {
                    System.out.println(getName() + " interrupted in semaphore operation");
                    break; // Выходим из цикла
                }
            }
        } finally {
            System.out.println(getName() + " finished: " + tasksProcessed + " tasks processed");
        }
    }
}

```

Фото 15

Отличие этих классов от предыдущих версий заключается в том, что вместо средств синхронизации в методах run() должны использоваться возможности семафора.

В главном классе программы создал метод complicatedThreads(). В нём создал объект задания

Регулярная проверка: if (isInterrupted()) или while (!isInterrupted())

Обработка InterruptedException: Немедленный выход из метода

Восстановление статуса (опционально):

```

complicatedThreads
Всего задач: 100
Integrator-Thread started
Generator-Thread started: 100 tasks
Generator-Thread 1: Source 68,856855 90,464019 0,443736 (base=8,0774)
Integrator-Thread: Task 1 - [68,8569, 90,4640], step: 0,443736, result: 45,247243
Generator-Thread 2: Source 75,806052 94,619415 0,090959 (base=8,0924)
Integrator-Thread: Task 2 - [75,8061, 94,6194], step: 0,090959, result: 39,977505
Generator-Thread 3: Source 55,522981 145,707146 0,657837 (base=6,3120)
Integrator-Thread: Task 3 - [55,5230, 145,7071], step: 0,657837, result: 223,965437
Generator-Thread 4: Source 46,479053 94,368421 0,589217 (base=2,6842)
Integrator-Thread: Task 4 - [46,4791, 94,3684], step: 0,589217, result: 205,383016

Main thread: Interrupting threads after 50ms...
Generator-Thread 5: Source 70,543544 147,721045 0,540512 (base=4,5959)
Integrator-Thread interrupted in semaphore operation
Generator-Thread interrupted during sleep
Integrator-Thread finished: 4 tasks processed
Generator-Thread finished

```

Фото 17(Thread.sleep(50))

Thread.sleep(2000)

```

Всего задач: 100
Integrator-Thread started
Generator-Thread started: 100 tasks
Generator-Thread 1: Source 20,558266 58,482965 0,802691 (base=5,3096)
Integrator-Thread: Task 1 - [20,5583, 58,4830], step: 0,802691, result: 82,581739
Generator-Thread 2: Source 10,904538 75,015401 0,107465 (base=9,8870)
Integrator-Thread: Task 2 - [10,9045, 75,0154], step: 0,107465, result: 102,010918
Generator-Thread 3: Source 61,670186 80,621996 0,717144 (base=5,0954)
Integrator-Thread: Task 3 - [61,6702, 80,6220], step: 0,717144, result: 49,601705
Generator-Thread 4: Source 12,448076 85,080323 0,976644 (base=7,0896)
Integrator-Thread: Task 4 - [12,4481, 85,0803], step: 0,976644, result: 139,912589
Generator-Thread 5: Source 59,520126 117,695100 0,715695 (base=4,9344)
Integrator-Thread: Task 5 - [59,5201, 117,6951], step: 0,715695, result: 162,751325
Generator-Thread 6: Source 96,100640 134,799334 0,357050 (base=7,8695)
Integrator-Thread: Task 6 - [96,1006, 134,7993], step: 0,357050, result: 88,992373
Generator-Thread 7: Source 0,107067 35,326820 0,520118 (base=5,3253)
Integrator-Thread: Task 7 - [0,1071, 35,3268], step: 0,520118, result: 54,281308
Generator-Thread 8: Source 27,224884 90,541143 0,807569 (base=1,8494)
Integrator-Thread: Task 8 - [27,2249, 90,5411], step: 0,807569, result: 414,215718
Generator-Thread 9: Source 52,287236 80,920499 0,705582 (base=3,3649)
Integrator-Thread: Task 9 - [52,2872, 80,9205], step: 0,705582, result: 98,896758
Generator-Thread 10: Source 90,368884 151,361639 0,302871 (base=2,4675)
Integrator-Thread: Task 10 - [90,3689, 151,3616], step: 0,302871, result: 323,051639
Generator-Thread 11: Source 20,497886 45,280333 0,839236 (base=8,7949)
Integrator-Thread: Task 11 - [20,4979, 45,2803], step: 0,839236, result: 39,534140
Generator-Thread 12: Source 29,517378 58,849818 0,931347 (base=4,8496)
Integrator-Thread: Task 12 - [29,5174, 58,8498], step: 0,931347, result: 70,025336
Generator-Thread 13: Source 91,383600 168,532732 0,691601 (base=8,0438)
Integrator-Thread: Task 13 - [91,3837, 168,5327], step: 0,691601, result: 179,545869
Generator-Thread 14: Source 98,065930 128,226136 0,855315 (base=7,1476)
Integrator-Thread: Task 14 - [98,0659, 128,2261], step: 0,855315, result: 72,468033
Generator-Thread 15: Source 62,216275 106,753249 0,462191 (base=8,7689)
Integrator-Thread: Task 15 - [62,2163, 106,7532], step: 0,462191, result: 90,762505
Generator-Thread 16: Source 18,908610 42,099586 0,599898 (base=3,9725)
Integrator-Thread: Task 16 - [18,9086, 42,0996], step: 0,599898, result: 57,037894
Generator-Thread 17: Source 63,997755 125,705330 0,210758 (base=2,6508)
Integrator-Thread: Task 17 - [63,9978, 125,7053], step: 0,210758, result: 286,996894

```

Фото 18


```
Integrator-Thread: Task 88 - [80,2003, 103,2370], step: 0,014332, result: 177,333100
Generator-Thread 89: Source 82,083469 129,304992 0,026305 (base=1,1928)
Integrator-Thread: Task 89 - [82,0835, 129,3050], step: 0,026305, result: 1245,981556
Generator-Thread 90: Source 47,363527 145,399024 0,906170 (base=6,1318)
Integrator-Thread: Task 90 - [47,3635, 145,3990], step: 0,906170, result: 244,420048
Generator-Thread 91: Source 96,970280 106,278411 0,653012 (base=7,1316)
Integrator-Thread: Task 91 - [96,9703, 106,2784], step: 0,653012, result: 21,894339
Generator-Thread 92: Source 24,311520 81,375940 0,770989 (base=5,8298)
Integrator-Thread: Task 92 - [24,3115, 81,3759], step: 0,770989, result: 126,681583
Generator-Thread 93: Source 1,448234 66,061321 0,225472 (base=5,0088)
Integrator-Thread: Task 93 - [1,4482, 66,0613], step: 0,225472, result: 131,382083
Generator-Thread 94: Source 11,674218 57,646693 0,430059 (base=6,0150)
Integrator-Thread: Task 94 - [11,6742, 57,6467], step: 0,430059, result: 88,648176
Generator-Thread 95: Source 90,047906 131,992717 0,313733 (base=4,9612)
Integrator-Thread: Task 95 - [90,0479, 131,9927], step: 0,313733, result: 123,183655
Generator-Thread 96: Source 94,486883 100,238034 0,179142 (base=3,2739)
Integrator-Thread: Task 96 - [94,4869, 100,2380], step: 0,179142, result: 22,201515
Generator-Thread 97: Source 90,289474 162,088795 0,056122 (base=3,1516)
Integrator-Thread: Task 97 - [90,2895, 162,0888], step: 0,056122, result: 301,730274
Generator-Thread 98: Source 94,326678 140,269131 0,779019 (base=1,5423)
Integrator-Thread: Task 98 - [94,3267, 140,2691], step: 0,779019, result: 504,516393
Generator-Thread 99: Source 91,651641 180,147437 0,072004 (base=1,3204)
Generator-Thread 100: Source 6,902794 29,217426 0,513535 (base=7,4432)
Integrator-Thread: Task 99 - [91,6516, 180,1474], step: 0,072004, result: 1558,021539
Integrator-Thread: Task 100 - [6,9028, 29,2174], step: 0,513535, result: 31,360074
Generator-Thread finished
Integrator-Thread finished: 100 tasks processed
```

Фото 19