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 №6

Студент Трофимов А. В.

Группа 6301-030301D

Руководитель Борисов Д. С.

Оценка _____

САМАРА 2025

Задание №1

В класс Functions был добавлен метод, возвращающий значение интеграла функции, вычисленное с помощью численного метода. В качестве параметров метод получает ссылку типа Function на объект функции, значения левой и правой границы области интегрирования, а также шаг дискретизации. Если интервал интегрирования выходит за границы области определения функции, метод выбрасывает исключение.

Вычисление значения интеграла должно выполняться по методу трапеций.

```
public static double integrate(Function f, double left, double right, double step) {
    // Проверка, что интервал интегрирования входит в область определения
    if (left < f.getLeftDomainBorder() || right > f.getRightDomainBorder()) {
        throw new IllegalArgumentException(s: "Интервал интегрирования выходит за границы области определения функции");
    }

    if (left >= right) {
        throw new IllegalArgumentException(s: "Левая граница должна быть меньше правой границы");
    }

    if (step <= 0) {
        throw new IllegalArgumentException(s: "Шаг должен быть положительным числом");
    }

    double integral = 0.0;
    double currentX = left;

    while (currentX < right) {
        double nextX = Math.min(currentX + step, right);
        double f1 = f.getFunctionValue(currentX);
        double f2 = f.getFunctionValue(nextX);

        // Если какое-то значение NaN, возвращаем NaN
        if (Double.isNaN(f1) || Double.isNaN(f2)) {
            return Double.NaN;
        }

        // Метод трапеций
        integral += (f1 + f2) * (nextX - currentX) / 2.0;
        currentX = nextX;
    }

    return integral;
}
```

В main() была создана проверка для работы метода интегрирования

```
private static void testIntegration() {
    try {
        Function exp = new Exp();

        System.out.println(x: "Интегрирование функции exp(x) на интервале [0, 1]:");
        // Теоретическое значение интеграла exp(x) от 0 до 1 = e - 1 ≈ 1.718281828
        double theoretical = Math.E - 1;
        System.out.println("Теоретическое значение интеграла: " + theoretical + " (e - 1)");

        System.out.println(x: "\nВычисление с разными шагами дискретизации:");
        double[] steps = {0.5, 0.1, 0.01, 0.001, 0.0001, 0.00001};

        for (double step : steps) {
            double result = Functions.integrate(exp, left: 0, right: 1, step);
            double error = Math.abs(result - theoretical);
            System.out.printf(format: "   Шаг %8.5f: результат = %.10f, ошибка = %.10f",
                step, result, error);

            // Проверяем 7 знак после запятой
            if (error < 1e-7) {
                System.out.println("\nВывод: Шаг " + step + " обеспечивает точность до 7 знака после запятой");
                break;
            }
        }
```

```
Вычисление с разными шагами дискретизации:
   Шаг  0,00100: результат = 1,7182819716, ошибка = 0,0000001432   Шаг  0,00010: результат = 1,7182818299, ошибка = 0,0000000014
Вывод: Шаг 1.0E-4 обеспечивает точность до 7 знака после запятой
```

Задание 2

Был создан пакет threads и в нем был описан класс Task, объект которого хранит ссылку на объект интегрируемой функции, границы области интегрирования, шаг дискретизации, а также целочисленное поле, хранящее количество выполняемых заданий.

```java
package threads;

import functions.Function;

public class Task {
    private Function f;
    private double leftX;
    private double rightX;
    private double step;
    private int taskCount;
    private volatile boolean taskReady = false;

    public Task() {
        this.taskCount = 0;
    }
    public Function getFunction() {
        return f;
    }
    public synchronized void setFunction(Function f) {
        this.f = f;
        this.taskReady = true;
        notify();
    }
    public double getLeft() {
        return leftX;
    }
    public void setLeft (double leftX) {
        this.leftX = leftX;
    }
    public double getRight() {
        return rightX;
    }
    public void setRight (double rightX) {
        this.rightX = rightX;
    }
    public double getStep() {
        return step;
    }
    public void setStep(double step) {

        this.step = step;
    }
    public  int getTasksCount() {
        return taskCount;
    }
    public void setTasksCount(int taskCount) {
        this.taskCount = taskCount;
    }
    public synchronized boolean taskReady() {
        return taskReady;
    }
    public synchronized void resetTask() {
        this.taskReady = false;
    }

}
```

Также в классе есть поле taskReady и методы для флага,
которые далее будут необходимы для нормальной работы simpleGenerator и SimpleIntegrator.

В main() был создан метод последовательно генерирующий и решающий task-и со всеми необходимыми условиями, поставленными в задании.

```
private static void nonThread() {
    System.out.println(x: "\nПоследовательная версия программы (без потоков):");

    Task task = new Task();
    task.setTasksCount(taskCount: 100); // Для демонстрации 100 заданий

    for (int i = 0; i < task.getTasksCount(); i++) {
        System.out.println("\n--- Задание " + (i+1) + " ---");

        // Генерируем случайные параметры
        double base = 1 + Math.random() * 9;   // основание логарифма от 1 до 10
        double left = Math.random() * 100;     // левая граница от 0 до 100
        double right = 100 + Math.random() * 100; // правая граница от 100 до 200
        double step = Math.random();           // шаг от 0 до 1

        System.out.println("Параметры: log_" + String.format(format: "%.2f", base) +
                           "(x) на [" + String.format(format: "%.2f", left) +
                           ", " + String.format(format: "%.2f", right) +
                           "] с шагом " + String.format(format: "%.4f", step));

        // Создаем логарифмическую функцию
        Function logFunc = new Log(base);

        // Устанавливаем задание
        task.setFunction(logFunc);
        task.setLeft(left);
        task.setRight(right);
        task.setStep(step);

        // Выводим сообщение о задании
        System.out.println("Сообщение: Source " +
            String.format(format: "%.2f", left) + " " +
            String.format(format: "%.2f", right) + " " +
            String.format(format: "%.4f", step));

        try {
            // Вычисляем интеграл
            double result = Functions.integrate(logFunc, left, right, step);

            // Выводим результат
            System.out.println("Результат: Result " +
                String.format(format: "%.2f", left) + " " +
                String.format(format: "%.2f", right) + " " +
                String.format(format: "%.4f", step) + " " +
                String.format(format: "%.6f", result));

        } catch (Exception e) {
            System.out.println("Ошибка вычисления: " + e.getMessage());
        }
    }
}
```

```
Последовательная версия программы (без потоков):

--- Задание 1 ---
Параметры: log_4,46(x) на [13,91, 197,32] с шагом 0,0927
Сообщение: Source 13,91 197,32 0,0927
Результат: Result 13,91 197,32 0,0927 549,892896

--- Задание 2 ---
Параметры: log_8,93(x) на [50,77, 180,51] с шагом 0,2547
Сообщение: Source 50,77 180,51 0,2547
Результат: Result 50,77 180,51 0,2547 278,063778

--- Задание 3 ---
Параметры: log_5,65(x) на [75,98, 109,29] с шагом 0,6441
Сообщение: Source 75,98 109,29 0,6441
Результат: Result 75,98 109,29 0,6441 86,965693

--- Задание 4 ---
Параметры: log_5,67(x) на [45,20, 106,62] с шагом 0,8581
Сообщение: Source 45,20 106,62 0,8581
Результат: Result 45,20 106,62 0,8581 152,242327

--- Задание 5 ---
Параметры: log_7,17(x) на [52,91, 171,37] с шагом 0,4191
Сообщение: Source 52,91 171,37 0,4191
Результат: Result 52,91 171,37 0,4191 280,825358

--- Задание 6 ---
Параметры: log_4,67(x) на [28,90, 195,23] с шагом 0,6636
Сообщение: Source 28,90 195,23 0,6636
Результат: Result 28,90 195,23 0,6636 497,126014

--- Задание 7 ---
Параметры: log_5,42(x) на [90,38, 135,58] с шагом 0,1003
Сообщение: Source 90,38 135,58 0,1003
Результат: Result 90,38 135,58 0,1003 126,298344

--- Задание 8 ---
Параметры: log_7,00(x) на [75,71, 110,06] с шагом 0,7783
Сообщение: Source 75,71 110,06 0,7783
Результат: Result 75,71 110,06 0,7783 79,881863

--- Задание 9 ---
Параметры: log_3,70(x) на [80,43, 164,64] с шагом 0,0433
Сообщение: Source 80,43 164,64 0,0433
Результат: Result 80,43 164,64 0,0433 308,075850

--- Задание 10 ---
Параметры: log_2,29(x) на [13,89, 181,16] с шагом 0,4319
Сообщение: Source 13,89 181,16 0,4319
Результат: Result 13,89 181,16 0,4319 889,251895
```

```
--- Задание 90 ---
Параметры: log_3,49(x) на [99,98, 153,17] с шагом 0,3925
Сообщение: Source 99,98 153,17 0,3925
Результат: Result 99,98 153,17 0,3925 205,665758

--- Задание 91 ---
Параметры: log_8,53(x) на [63,13, 101,48] с шагом 0,4054
Сообщение: Source 63,13 101,48 0,4054
Результат: Result 63,13 101,48 0,4054 78,766052

--- Задание 92 ---
Параметры: log_8,23(x) на [34,72, 147,77] с шагом 0,4849
Сообщение: Source 34,72 147,77 0,4849
Результат: Result 34,72 147,77 0,4849 238,160786

--- Задание 93 ---
Параметры: log_7,09(x) на [65,75, 135,79] с шагом 0,6406
Сообщение: Source 65,75 135,79 0,6406
Результат: Result 65,75 135,79 0,6406 164,249227

--- Задание 94 ---
Параметры: log_2,04(x) на [11,52, 158,19] с шагом 0,5782
Сообщение: Source 11,52 158,19 0,5782
Результат: Result 11,52 158,19 0,5782 876,240991

--- Задание 95 ---
Параметры: log_9,48(x) на [21,66, 144,08] с шагом 0,9386
Сообщение: Source 21,66 144,08 0,9386
Результат: Result 21,66 144,08 0,9386 234,361035

--- Задание 96 ---
Параметры: log_5,29(x) на [56,61, 159,85] с шагом 0,4766
Сообщение: Source 56,61 159,85 0,4766
Результат: Result 56,61 159,85 0,4766 287,651713

--- Задание 97 ---
Параметры: log_4,98(x) на [15,98, 183,47] с шагом 0,5792
Сообщение: Source 15,98 183,47 0,5792
Результат: Result 15,98 183,47 0,5792 463,969980

--- Задание 98 ---
Параметры: log_7,56(x) на [37,16, 102,08] с шагом 0,6473
Сообщение: Source 37,16 102,08 0,6473
Результат: Result 37,16 102,08 0,6473 134,907473

--- Задание 99 ---
Параметры: log_3,62(x) на [98,16, 199,60] с шагом 0,5073
Сообщение: Source 98,16 199,60 0,5073
Результат: Result 98,16 199,60 0,5073 392,698733

--- Задание 100 ---
Параметры: log_1,56(x) на [37,97, 146,66] с шагом 0,2058
Сообщение: Source 37,97 146,66 0,2058
Результат: Result 37,97 146,66 0,2058 1082,503888
```

Задание №3

В пакете threads были созданы два следующих класса.

Класс SimpleGenerator реализовывает интерфейс Runnable, получает в конструкторе и сохраняет в своё поле ссылку на объект типа Task, а в методе run() цикл формирует задачи и заносит в полученный объект задания, а также выводит сообщение в консоль.

```java
package threads;

import functions.Function;
import functions.basic.Log;

// Простой генератор заданий, реализующий интерфейс Runnable
public class SimpleGenerator implements Runnable {
    private final Task task;

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

    @Override
    public void run() {
        try {
            // Генерация заданий в цикле
            for (int i = 0; i < task.getTasksCount(); i++) {

                    // Генерируем случайные параметры
                    double base = 1 + Math.random() * 9;    // основание логарифма от 1 до 10
                    double left = Math.random() * 100;      // левая граница от 0 до 100
                    double right = 100 + Math.random() * 100; // правая граница от 100 до 200
                    double step = Math.random();            // шаг от 0 до 1

                    // Создаем логарифмическую функцию
                    Function logFunc = new Log(base);
                synchronized (task) {
                    // Устанавливаем задание
                    task.setFunction(logFunc);
                    task.setLeft(left);
                    task.setRight(right);
                    task.setStep(step);
                    // Выводим сообщение о созданном задании с комментариями
                    System.out.println("Создано задание: левая граница = " + String.format(format: "%.2f", left) +
                                       ", правая граница = " + String.format(format: "%.2f", right) +
                                       ", шаг = " + String.format(format: "%.4f", step) +
                                       "   [Основание логарифма: " + String.format(format: "%.2f", base) + "]");
                }
                Thread.sleep(millis: 2);
            }
        } catch (InterruptedException e) {
            System.out.println("Ошибка генерации: " + e.getMessage());
        }
    }
}
```

Была добавлена задержка для нормального вывода.

Класс SimpleIntegrator реализовывает интерфейс Runnable, получает в конструкторе и сохраняет в своё поле ссылку на объект типа Task, а в методе run() цикл решает задачи, данные для которых берутся из полученного объекта задания, а также выводит сообщения в консоль.

```java
package threads;

import functions.Function;
import functions.Functions;

// Простой интегратор, реализующий интерфейс Runnable
public class SimpleIntegrator implements Runnable {
    private final Task task;

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

    @Override
    public void run() {
        try {
            // Обработка заданий в цикле
            for (int i = 0; i < task.getTasksCount(); i++) {

                synchronized (task) {
                    while(!task.taskReady()){
                        task.wait();
                    }

                    // Ждем, пока появится задание
                    // Получаем параметры задания
                    Function func = task.getFunction();
                    double left = task.getLeft();
                    double right = task.getRight();
                    double step = task.getStep();

                    // Вычисляем интеграл
                    double result = Functions.integrate(func, left, right, step);

                    // Выводим результат с комментариями
                    System.out.println("Результат интегрирования: левая граница = " + String.format(format: "%.2f", left) +
                                       ", правая граница = " + String.format(format: "%.2f", right) +
                                       ", шаг = " + String.format(format: "%.4f", step) +
                                       ", интеграл = " + String.format(format: "%.6f", result));

                    // Помечаем задание как выполненное
                    task.resetTask();
                }
            }
        } catch (InterruptedException e) {
            System.out.println("Ошибка интегрирования: " + e.getMessage());
        }
    }
}
```

В main() создал метод для проверки работы классов.

```
public class Main {
    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        System.out.println(x: "\n=== Лабораторная работа №6 ===");

        System.out.println(x: "\n--- Задание 1: Тестирование численного интегрирования экспоненты ---");
        testIntegration();

        //System.out.println("\n--- Задание 2: Последовательная версия программы ---");
        //nonThread();

        System.out.println(x: "\n--- Задание 3: Простая многопоточная версия с синхронизацией ---");
        simpleThreads();

        //System.out.println("\n--- Задание 4: Усовершенствованная версия с семафорами ---");
        //complicatedThreads();
    }
private static void simpleThreads() {
    System.out.println(x: "\nПростая многопоточная версия:");
    System.out.println(x: "Один поток генерирует задания, другой - решает их");

    Task task = new Task();
    task.setTasksCount(taskCount: 100); // 100 заданий для демонстрации

    System.out.println(x: "\nСоздаем два потока:");
    System.out.println(x: "1. SimpleGenerator - генерирует задания");
    System.out.println(x: "2. SimpleIntegrator - решает задания");

    Thread generatorThread = new Thread(new SimpleGenerator(task));
    Thread integratorThread = new Thread(new SimpleIntegrator(task));

    // Установка приоритетов
    generatorThread.setPriority(Thread.MIN_PRIORITY);
    integratorThread.setPriority(Thread.MAX_PRIORITY);
    System.out.println(x: "Установлены нормальные приоритеты для потоков");

    System.out.println(x: "\nЗапускаем потоки...");
    System.out.println(x: "Ожидаемые сообщения:");
    System.out.println(x: "  'Создано задание ...' - от генератора");
    System.out.println(x: "  'Результат ...' - от интегратора");

    // Запуск потоков
    integratorThread.start();
    generatorThread.start();

    System.out.println(x: "\nПростая многопоточная версия завершена");
    System.out.println(x: "Примечание: Из-за синхронизации сообщения могут выводиться в разном порядке");
}
```

```
Создано задание: левая граница = 99,13, правая граница = 159,47, шаг = 0,9598   [Основание логарифма: 7,34]
Результат интегрирования: левая граница = 99,13, правая граница = 159,47, шаг = 0,9598, интеграл = 146,878298
Создано задание: левая граница = 31,68, правая граница = 132,85, шаг = 0,2425   [Основание логарифма: 8,66]
Результат интегрирования: левая граница = 31,68, правая граница = 132,85, шаг = 0,2425, интеграл = 203,269504
Создано задание: левая граница = 62,72, правая граница = 114,22, шаг = 0,0139   [Основание логарифма: 7,28]
Результат интегрирования: левая граница = 62,72, правая граница = 114,22, шаг = 0,0139, интеграл = 115,945146
Создано задание: левая граница = 98,53, правая граница = 127,28, шаг = 0,0674   [Основание логарифма: 7,78]
Результат интегрирования: левая граница = 98,53, правая граница = 127,28, шаг = 0,0674, интеграл = 66,216168
Создано задание: левая граница = 77,69, правая граница = 116,48, шаг = 0,2283   [Основание логарифма: 7,78]
Результат интегрирования: левая граница = 77,69, правая граница = 116,48, шаг = 0,2283, интеграл = 86,374910
Создано задание: левая граница = 16,10, правая граница = 102,14, шаг = 0,1466   [Основание логарифма: 4,11]
Результат интегрирования: левая граница = 16,10, правая граница = 102,14, шаг = 0,1466, интеграл = 241,910429
Создано задание: левая граница = 54,66, правая граница = 124,85, шаг = 0,8359   [Основание логарифма: 9,78]
Результат интегрирования: левая граница = 54,66, правая граница = 124,85, шаг = 0,8359, интеграл = 137,577945
Создано задание: левая граница = 67,65, правая граница = 128,68, шаг = 0,4817   [Основание логарифма: 3,92]
Результат интегрирования: левая граница = 67,65, правая граница = 128,68, шаг = 0,4817, интеграл = 204,060654
Создано задание: левая граница = 66,13, правая граница = 150,02, шаг = 0,6862   [Основание логарифма: 5,47]
Результат интегрирования: левая граница = 66,13, правая граница = 150,02, шаг = 0,6862, интеграл = 229,933594
Создано задание: левая граница = 9,36, правая граница = 124,45, шаг = 0,5022   [Основание логарифма: 9,20]
Результат интегрирования: левая граница = 9,36, правая граница = 124,45, шаг = 0,5022, интеграл = 209,214093
Создано задание: левая граница = 19,59, правая граница = 197,37, шаг = 0,3177   [Основание логарифма: 3,51]
Результат интегрирования: левая граница = 19,59, правая граница = 197,37, шаг = 0,3177, интеграл = 643,427145
Создано задание: левая граница = 80,72, правая граница = 125,86, шаг = 0,8970   [Основание логарифма: 3,69]
Результат интегрирования: левая граница = 80,72, правая граница = 125,86, шаг = 0,8970, интеграл = 160,020219
Создано задание: левая граница = 89,66, правая граница = 189,14, шаг = 0,6589   [Основание логарифма: 7,97]
Результат интегрирования: левая граница = 89,66, правая граница = 189,14, шаг = 0,6589, интеграл = 235,572855
Создано задание: левая граница = 20,95, правая граница = 182,64, шаг = 0,5771   [Основание логарифма: 6,48]
Результат интегрирования: левая граница = 20,95, правая граница = 182,64, шаг = 0,5771, интеграл = 388,501430
Создано задание: левая граница = 59,47, правая граница = 157,06, шаг = 0,9256   [Основание логарифма: 8,10]
Результат интегрирования: левая граница = 59,47, правая граница = 157,06, шаг = 0,9256, интеграл = 216,906396
Создано задание: левая граница = 54,25, правая граница = 177,66, шаг = 0,4611   [Основание логарифма: 8,27]
Результат интегрирования: левая граница = 54,25, правая граница = 177,66, шаг = 0,4611, интеграл = 274,665255
Создано задание: левая граница = 46,83, правая граница = 121,49, шаг = 0,5303   [Основание логарифма: 2,25]
```

```
Создано задание: левая граница = 41,06, правая граница = 112,67, шаг = 0,7171   [Основание логарифма: 2,61]
Результат интегрирования: левая граница = 41,06, правая граница = 112,67, шаг = 0,7171, интеграл = 321,756343
Создано задание: левая граница = 79,89, правая граница = 173,37, шаг = 0,5250   [Основание логарифма: 5,52]
Результат интегрирования: левая граница = 79,89, правая граница = 173,37, шаг = 0,5250, интеграл = 263,483806
Создано задание: левая граница = 18,14, правая граница = 196,68, шаг = 0,8674   [Основание логарифма: 1,61]
Результат интегрирования: левая граница = 18,14, правая граница = 196,68, шаг = 0,8674, интеграл = 1701,098405
Создано задание: левая граница = 7,39, правая граница = 131,31, шаг = 0,9058   [Основание логарифма: 5,42]
Результат интегрирования: левая граница = 7,39, правая граница = 131,31, шаг = 0,9058, интеграл = 296,740701
Создано задание: левая граница = 4,14, правая граница = 175,12, шаг = 0,8988   [Основание логарифма: 3,00]
Результат интегрирования: левая граница = 4,14, правая граница = 175,12, шаг = 0,8988, интеграл = 661,413353
Создано задание: левая граница = 75,11, правая граница = 138,36, шаг = 0,7046   [Основание логарифма: 1,74]
Результат интегрирования: левая граница = 75,11, правая граница = 138,36, шаг = 0,7046, интеграл = 530,784836
Создано задание: левая граница = 70,50, правая граница = 183,33, шаг = 0,6208   [Основание логарифма: 1,28]
Результат интегрирования: левая граница = 70,50, правая граница = 183,33, шаг = 0,6208, интеграл = 2193,235742
Создано задание: левая граница = 15,81, правая граница = 109,63, шаг = 0,9843   [Основание логарифма: 1,62]
Результат интегрирования: левая граница = 15,81, правая граница = 109,63, шаг = 0,9843, интеграл = 783,580426
Создано задание: левая граница = 20,11, правая граница = 118,18, шаг = 0,8278   [Основание логарифма: 1,50]
Результат интегрирования: левая граница = 20,11, правая граница = 118,18, шаг = 0,8278, интеграл = 1005,539213
Создано задание: левая граница = 49,89, правая граница = 179,32, шаг = 0,2852   [Основание логарифма: 9,10]
Результат интегрирования: левая граница = 49,89, правая граница = 179,32, шаг = 0,2852, интеграл = 274,408758
Создано задание: левая граница = 71,40, правая граница = 151,11, шаг = 0,3699   [Основание логарифма: 1,03]
Результат интегрирования: левая граница = 71,40, правая граница = 151,11, шаг = 0,3699, интеграл = 11900,178777
Создано задание: левая граница = 21,55, правая граница = 142,04, шаг = 0,2128   [Основание логарифма: 1,47]
Результат интегрирования: левая граница = 21,55, правая граница = 142,04, шаг = 0,2128, интеграл = 1344,303113
Создано задание: левая граница = 27,85, правая граница = 189,28, шаг = 0,8374   [Основание логарифма: 3,48]
Результат интегрирования: левая граница = 27,85, правая граница = 189,28, шаг = 0,8374, интеграл = 591,897139
```

Задание №4

В пакете threads были созданы два следующих класса.

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() выполняет те же действия, что и в предыдущей версии генерирующего класса.

```java
package threads;

import functions.Function;
import functions.basic.Log;
import java.util.concurrent.Semaphore;

// Генератор заданий, расширяющий класс Thread
public class Generator extends Thread {
    private final Task task;
    private final Semaphore dataReady;
    private final Semaphore dataProcessed;

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

    @Override
    public void run() {
        try {
            // Генерация заданий в цикле
            for (int i = 0; i < task.getTasksCount(); i++) {


                // Генерируем случайные параметры
                double base = 1 + Math.random() * 9;   // основание логарифма от 1 до 10
                double left = Math.random() * 100;     // левая граница от 0 до 100
                double right = 100 + Math.random() * 100; // правая граница от 100 до 200
                double step = Math.random();           // шаг от 0 до 1
                dataProcessed.acquire();

                Function logFunc = new Log(base);
                task.setFunction(logFunc);
                task.setLeft(left);
                task.setRight(right);
                task.setStep(step);


                System.out.println("Создано задание: левая граница = " + String.format(format: "%.2f", left) +
                                   ", правая граница = " + String.format(format: "%.2f", right) +
                                   ", шаг = " + String.format(format: "%.4f", step) +
                                   "   [Основание логарифма: " + String.format(format: "%.2f", base) + "]");
                dataReady.release();
            }

        } catch (InterruptedException e) {
            System.out.println(x: "Поток генератора Generator прерван");
        }
    }
}
```

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

```java
package threads;

import functions.Function;
import functions.Functions;
import java.util.concurrent.Semaphore;

// Интегратор, расширяющий класс Thread
public class Integrator extends Thread {
    private final Task task;
    private final Semaphore dataReady;
    private final Semaphore dataProcessed;

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

    @Override
    public void run() {
        try {
            // Обработка заданий в цикле
            for (int i = 0; i < task.getTasksCount(); i++) {
                // Проверка прерывания потока

                // Начинаем чтение (могут быть несколько читателей одновременно)
                dataReady.acquire();
                // Получаем параметры задания
                Function func = task.getFunction();
                double left = task.getLeft();
                double right = task.getRight();
                double step = task.getStep();

                // Вычисляем интеграл
                double result = Functions.integrate(func, left, right, step);

                // Выводим результат с комментариями
                System.out.println("Результат интегрирования: левая граница = " + String.format(format: "%.2f", left) +
                                   ", правая граница = " + String.format(format: "%.2f", right) +
                                   ", шаг = " + String.format(format: "%.4f", step) +
                                   ", интеграл = " + String.format(format: "%.6f", result));
                dataProcessed.release();

            }
        } catch (InterruptedException e) {
            System.out.println(x: "Поток интегратора Integrator прерван");
        } catch (Exception e) {
            System.out.println("Ошибка интегрирования: " + e.getMessage());
        }
    }
}
```

В main() создал метод для проверки Generator и Integrator.

```java
public class Main {
    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        System.out.println(x: "\n=== Лабораторная работа №6 ===");

        System.out.println(x: "\n--- Задание 1: Тестирование численного интегрирования экспоненты ---");
        testIntegration();

        //System.out.println("\n--- Задание 2: Последовательная версия программы ---");
        //nonThread();

        //System.out.println("\n--- Задание 3: Простая многопоточная версия с синхронизацией ---");
        //simpleThreads();

        System.out.println(x: "\n--- Задание 4: Усовершенствованная версия с семафорами ---");
        complicatedThreads();
    }
```

```java
private static void complicatedThreads() {
    System.out.println(x: "\nУсовершенствованная многопоточная версия с семафорами:");
    System.out.println(x: "Используется семафор для более тонкой синхронизации");

    Task task = new Task();
    task.setTasksCount(taskCount: 100); // 100 заданий для демонстрации
    Semaphore dataReady = new Semaphore(permits: 0);
    Semaphore dataProcessed = new Semaphore(permits: 1);

    Generator generator = new Generator(task, dataReady, dataProcessed);
    Integrator integrator = new Integrator(task, dataReady, dataProcessed);
    // Запуск потоков
    generator.start();
    integrator.start();

    try {
        // Ждем 50 миллисекунд
        Thread.sleep(millis: 50);
        // Прерываем потоки
        generator.interrupt();
        integrator.interrupt();
        // Ожидаем завершения потоков
        generator.join();
        integrator.join();

    } catch (InterruptedException e) {
        System.out.println(x: "Основной поток был прерван");
    }

    System.out.println(x: "\nУсовершенствованная многопоточная версия завершена");
    System.out.println(x: "Семафор позволил более эффективно управлять доступом к общему ресурсу");
}
```

Если закомментировать блок с искусственной остановкой, то классы корректно генерируют и решают task-и.

```
Создано задание: левая граница = 78,28, правая граница = 194,07, шаг = 0,4545   [Основание логарифма: 3,63]
Результат интегрирования: левая граница = 78,28, правая граница = 194,07, шаг = 0,4545, интеграл = 438,300998
Создано задание: левая граница = 83,76, правая граница = 127,05, шаг = 0,5186   [Основание логарифма: 6,18]
Результат интегрирования: левая граница = 83,76, правая граница = 127,05, шаг = 0,5186, интеграл = 110,573921
Создано задание: левая граница = 28,28, правая граница = 101,98, шаг = 0,0749   [Основание логарифма: 5,01]
Результат интегрирования: левая граница = 28,28, правая граница = 101,98, шаг = 0,0749, интеграл = 188,345821
Создано задание: левая граница = 2,02, правая граница = 176,04, шаг = 0,3858   [Основание логарифма: 2,36]
Результат интегрирования: левая граница = 2,02, правая граница = 176,04, шаг = 0,3858, интеграл = 857,183546
Создано задание: левая граница = 10,03, правая граница = 134,68, шаг = 0,2074   [Основание логарифма: 4,92]
Результат интегрирования: левая граница = 10,03, правая граница = 134,68, шаг = 0,2074, интеграл = 321,773629
Создано задание: левая граница = 1,44, правая граница = 149,66, шаг = 0,9833   [Основание логарифма: 1,21]
Результат интегрирования: левая граница = 1,44, правая граница = 149,66, шаг = 0,9833, интеграл = 3178,122847
Создано задание: левая граница = 73,42, правая граница = 141,77, шаг = 0,1900   [Основание логарифма: 3,00]
Создано задание: левая граница = 15,33, правая граница = 192,74, шаг = 0,8266   [Основание логарифма: 7,92]
Результат интегрирования: левая граница = 15,33, правая граница = 192,74, шаг = 0,8266, интеграл = 384,019198
Создано задание: левая граница = 15,39, правая граница = 125,64, шаг = 0,1934   [Основание логарифма: 1,13]
Результат интегрирования: левая граница = 15,39, правая граница = 125,64, шаг = 0,1934, интеграл = 3620,302463
Создано задание: левая граница = 49,38, правая граница = 137,33, шаг = 0,4422   [Основание логарифма: 9,35]
Результат интегрирования: левая граница = 49,38, правая граница = 137,33, шаг = 0,4422, интеграл = 176,900014
Создано задание: левая граница = 48,51, правая граница = 180,77, шаг = 0,7253   [Основание логарифма: 6,55]
Результат интегрирования: левая граница = 48,51, правая граница = 180,77, шаг = 0,7253, интеграл = 329,378585
Создано задание: левая граница = 28,05, правая граница = 189,16, шаг = 0,6240   [Основание логарифма: 8,49]
Результат интегрирования: левая граница = 28,05, правая граница = 189,16, шаг = 0,6240, интеграл = 344,680497
Создано задание: левая граница = 93,33, правая граница = 100,69, шаг = 0,4116   [Основание логарифма: 3,58]
Результат интегрирования: левая граница = 93,33, правая граница = 100,69, шаг = 0,4116, интеграл = 26,351445
Создано задание: левая граница = 18,61, правая граница = 158,24, шаг = 0,9095   [Основание логарифма: 9,66]
Результат интегрирования: левая граница = 18,61, правая граница = 158,24, шаг = 0,9095, интеграл = 267,724212

Усовершенствованная многопоточная версия завершена
Семафор позволил более эффективно управлять доступом к общему ресурсу
```

Для обработки корректного прерывания потоков, в Generator и Integrator есть блоки try-catch, которые позволяют поймать исключение InterruptedException об остановке.