Лабораторная работа №6

Коновалов Сергей Сергеевич 6204-010302D

Содержание

Задание 1

Добавил в класс Functions метод для вычисления интеграла функции методом трапеций

```
public static double integrate(Function f, double a, double b, double step) {
    if (a < f.getLeftDomainBorder() || b > f.getRightDomainBorder()) {
        throw new IllegalArgumentException("Интервал интегрирования выходит за границы области определения функции");
    }
    if (step <= 0) {
        throw new IllegalArgumentException("Шаг дискретизации должен быть положительным");
    }
    if (a > b) {
        // Меняем пределы интегрирования местами
        double temp = a;
        a = b;
        b = temp;
    }

    double integral = 0.0;
    double x = a;

    // Проходим по всем полным шагам
    while (x + step <= b) {
        double y1 = f.getFunctionValue(x);
        double y2 = f.getFunctionValue(x + step);
        integral += (y1 + y2) * step / 2.0;
        x += step;
    }

    // Обрабатываем последний неполный шаг (если есть)
    if (x < b) {
        double lastStep = b - x;
        double y1 = f.getFunctionValue(x);
        double y2 = f.getFunctionValue(b);
        integral += (y1 + y2) * lastStep / 2.0;
    }

    return integral;
}
```

В методе main() проверил работу метода интегрирования. Для этого вычислил интеграл для экспоненты на отрезке от 0 до 1

```
    private static void testIntegration() {
    System.out.println("=".repeat(80));
    System.out.println("Вычисление интеграла методом трапеций");
    System.out.println("=".repeat(80));

    testExponentialIntegration();
    findPrecisionForExponential();
    testIntegrationExceptions();
}

private static void testExponentialIntegration() {
    System.out.println("\n1. Вычисление интеграла экспоненты на отрезке [0, 1]");
    System.out.println("-".repeat(60));

    Function exp = new Exp();
    double a = 0.0;
    double b = 1.0;
    double theoretical = Math.E - 1; // ∫e^x dx от 0 до 1 = e - 1

    double[] steps = {0.1, 0.01, 0.001, 0.0001};

    System.out.printf("%-12s %-15s %-15s %-15s%n", "Шаг", "Численный", "Теоретический", "Погрешность");
    for (double step : steps) {
        double numerical = Functions.integrate(exp, a, b, step);
        double error = Math.abs(numerical - theoretical);
        System.out.printf("%-12.4f %-15.10f %-15.10f %-15.10f%n",
                          step, numerical, theoretical, error);
    }
}
```

```
private static void findPrecisionForExponential() {
    System.out.println("\n\n2. Поиск шага для точности 1e-7");
    System.out.println("-".repeat(50));

    Function exp = new Exp();
    double a = 0.0;
    double b = 1.0;
    double theoretical = Math.E - 1;

    double targetPrecision = 1e-7;
    double step = 0.1;
    double numerical = 0;
    double error = Double.MAX_VALUE;
    int iterations = 0;

    System.out.printf("%-6s %-12s %-15s %-15s%n", "Итер.", "Шаг", "Численный", "Погрешность");

    while (error > targetPrecision && iterations < 20) {
        numerical = Functions.integrate(exp, a, b, step);
        error = Math.abs(numerical - theoretical);

        System.out.printf("%-6d %-12.8f %-15.10f %-15.10f",
                iterations, step, numerical, error);

        if (error <= targetPrecision) {
            System.out.print(" ✓ ДОСТИГНУТО");
        }
        System.out.println();

        step /= 2.0;
        iterations++;
    }

    if (error <= targetPrecision) {
        System.out.println("\nТребуемая точность достигнута при шаге: " + step * 2);
    } else {
        System.out.println("\nТребуемая точность не достигнута за " + iterations + " итераций");
    }

    // Дополнительная проверка с очень малым шагом
    System.out.println("\n3. Проверка с очень малым шагом");
    double fineStep = 1e-6;
    numerical = Functions.integrate(exp, a, b, fineStep);
    error = Math.abs(numerical - theoretical);
    System.out.printf("Шаг: %.2e, Погрешность: %.2e%n", fineStep, error);
}
```

```
private static void testIntegrationExceptions() {
    System.out.println("\n\n3. Тестирование обработки исключений");
    System.out.println("-".repeat(50));

    Function log = new Log(Math.E);

    try {
        // Попытка интегрирования за пределами области определения
        double result = Functions.integrate(log, -1, 1, 0.1);
        System.out.println("Результат: " + result);
    } catch (IllegalArgumentException e) {
        System.out.println("Поймано исключение: " + e.getMessage());
    }

    try {
        // Неверный шаг
        double result = Functions.integrate(log, 1, 2, -0.1);
        System.out.println("Результат: " + result);
    } catch (IllegalArgumentException e) {
        System.out.println("Поймано исключение: " + e.getMessage());
    }
}
```

```
    public static void main(String[] args) {
        test1();
        test2();
        test3();
        testIntegration();
    }
}
```

Задание 2

Создал пакет threads и класс Task для хранения параметров задания.

```java
package threads;

import functions.Function;

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

    public Task() {
    }

    public Task(Function function, double leftBorder, double rightBorder, double step, int tasksCount) {
        this.function = function;
        this.leftBorder = leftBorder;
        this.rightBorder = rightBorder;
        this.step = step;
        this.tasksCount = tasksCount;
    }

    // Геттеры
    public Function getFunction() {
        return function;
    }

    public double getLeftBorder() {
        return leftBorder;
    }

    public double getRightBorder() {
        return rightBorder;
    }

    public double getStep() {
        return step;
    }

    public int getTasksCount() {
        return tasksCount;
    }

    // Сеттеры
    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 setTasksCount(int tasksCount) {
        this.tasksCount = tasksCount;
    }

    @Override
    public String toString() {
        return String.format("Task{function=%s, left=%.2f, right=%.2f, step=%.4f, tasks=%d}",
                function, leftBorder, rightBorder, step, tasksCount);
    }
}
```

Добавил метод nonThread() с последовательным выполнением 100 заданий. Реализовал генерацию случайных параметров и вывод результатов в консоль.

```java
private static void nonThread() {
    System.out.println("Запуск последовательной версии программы...");
    System.out.println("Создание и выполнение 100 заданий интегрирования");
    System.out.println("-".repeat(80));

    // Создаем объект задания
    Task task = new Task();
    task.setTasksCount(100);

    long startTime = System.currentTimeMillis();
    int successCount = 0;
    int errorCount = 0;

    // Выполняем задания в цикле
    for (int i = 0; i < task.getTasksCount(); i++) {
        try {
            // 1. Создаем логарифмическую функцию со случайным основанием от 1 до 10
            double base = 1 + Math.random() * 9; // [1, 10)
            Function logFunction = new Log(base);
            task.setFunction(logFunction);

            // 2. Левая граница от 0 до 100
            double left = Math.random() * 100; // [0, 100)
            task.setLeftBorder(left);

            // 3. Правая граница от 100 до 200
            double right = 100 + Math.random() * 100; // [100, 200)
            task.setRightBorder(right);

            // 4. Шаг дискретизации от 0 до 1
            double step = Math.random(); // [0, 1)
            task.setStep(step);

            // 5. Вывод информации о задании
            System.out.printf("Source %.4f %.4f %.4f (log base=%.4f)%n",
                              left, right, step, base);

            // 6. Вычисление интеграла
            double result = Functions.integrate(task.getFunction(),
                                                task.getLeftBorder(),
                                                task.getRightBorder(),
                                                task.getStep());
```

```
            // 7. Вывод результата
            System.out.printf("Result %.4f %.4f %.4f %.8f%n",
                              left, right, step, result);
            successCount++;

        } catch (IllegalArgumentException e) {
            System.out.printf("ERROR: %s%n", e.getMessage());
            errorCount++;
        } catch (Exception e) {
            System.out.printf("UNEXPECTED ERROR: %s%n", e.getMessage());
            errorCount++;
        }

        // Небольшая пауза для наглядности (можно убрать)
        try {
            Thread.sleep(10);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
            break;
        }
    }

    long endTime = System.currentTimeMillis();
    long duration = endTime - startTime;

    System.out.println("-".repeat(80));
    System.out.println("Статистика выполнения:");
    System.out.printf("Успешно выполнено: %d заданий%n", successCount);
    System.out.printf("Завершено с ошибкой: %d заданий%n", errorCount);
    System.out.printf("Общее время выполнения: %d мс%n", duration);
    System.out.printf("Среднее время на задание: %.2f мс%n", (double)duration / task.getTasksCount());
}
```

Задание 3

Создал классы SimpleGenerator и SimpleIntegrator, реализующие Runnable.

```
package threads;

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

public class SimpleGenerator implements Runnable {
    private final Task task;

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

    @Override
    public void run() {
        try {
            for (int i = 0; i < task.getTasksCount(); i++) {
                // Генерируем случайные параметры
                double base = 1 + Math.random() * 9; // [1, 10)
                double left = Math.random() * 100;   // [0, 100)
                double right = 100 + Math.random() * 100; // [100, 200)
                double step = Math.random();         // [0, 1)

                // Создаем функцию
                Function logFunction = new Log(base);

                // Синхронизированная установка параметров в задание
                synchronized (task) {
                    task.setFunction(logFunction);
                    task.setLeftBorder(left);
                    task.setRightBorder(right);
                    task.setStep(step);
                }

                // Вывод сообщения
                System.out.printf("Generator: Source %.4f %.4f %.4f (log base=%.4f)%n",
                                  left, right, step, base);

                // Небольшая пауза для наглядности
                Thread.sleep(10);
            }
        } catch (InterruptedException e) {
            System.out.println("Generator was interrupted");
            Thread.currentThread().interrupt();
        }
    }
}
```

```
package threads;

import functions.Functions;

public class SimpleIntegrator implements Runnable {
    private final Task task;

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

```
    @Override
    public void run() {
        try {
            for (int i = 0; i < task.getTasksCount(); i++) {
                // Локальные переменные для хранения параметров задания
                double left, right, step;
                double result;

                // Синхронизированное чтение параметров из задания
                synchronized (task) {
                    // Проверяем, что функция установлена (защита от NullPointerException)
                    if (task.getFunction() == null) {
                        System.out.println("Integrator: Function is null, waiting...");
                        Thread.sleep(50);
                        continue;
                    }

                    left = task.getLeftBorder();
                    right = task.getRightBorder();
                    step = task.getStep();

                    // Вычисляем интеграл внутри синхронизированного блока
                    result = Functions.integrate(task.getFunction(), left, right, step);
                }

                // Вывод результата (вне синхронизированного блока)
                System.out.printf("Integrator: Result %.4f %.4f %.4f %.8f%n",
                                  left, right, step, result);

                // Небольшая пауза для наглядности
                Thread.sleep(10);
            }
        } catch (InterruptedException e) {
            System.out.println("Integrator was interrupted");
            Thread.currentThread().interrupt();
        } catch (IllegalArgumentException e) {
            System.out.printf("Integrator ERROR: %s%n", e.getMessage());
        } catch (Exception e) {
            System.out.printf("Integrator UNEXPECTED ERROR: %s%n", e.getMessage());
        }
    }
}
```

Добавил метод simpleThreads() с запуском двух потоков. Использовал synchronized блоки для защиты от NullPointerException и несогласованных данных.

```
private static void simpleThreads() {
    System.out.println("Запуск простой многопоточной версии программы...");
    System.out.println("Создание и выполнение 100 заданий интегрирования");
    System.out.println("-".repeat(80));

    // Создаем объект задания
    Task task = new Task();
    task.setTasksCount(100);

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

    // Устанавливаем имена потоков для удобства отладки
    generatorThread.setName("GeneratorThread");
    integratorThread.setName("IntegratorThread");

    // Тестирование с разными приоритетами (раскомментируйте для тестирования)
    // generatorThread.setPriority(Thread.MAX_PRIORITY);
    // integratorThread.setPriority(Thread.MIN_PRIORITY);

    long startTime = System.currentTimeMillis();

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

    // Ожидаем завершения потоков
    try {
        generatorThread.join();
        integratorThread.join();
    } catch (InterruptedException e) {
        System.out.println("Main thread was interrupted");
        Thread.currentThread().interrupt();
    }

    long endTime = System.currentTimeMillis();
    long duration = endTime - startTime;

    System.out.println("-".repeat(80));
    System.out.println("Оба потока завершили работу");
    System.out.printf("Общее время выполнения: %d мс%n", duration);
}
```

Задание 4

Создал класс ReadWriteSemaphore для управления доступом.

```java
package threads;

public class ReadWriteSemaphore {
    private int activeReaders = 0;
    private int activeWriters = 0;
    private int waitingReaders = 0;
    private int waitingWriters = 0;

    public synchronized void beginRead() throws InterruptedException {
        waitingReaders++;
        while (activeWriters > 0 || waitingWriters > 0) {
            wait();
        }
        waitingReaders--;
        activeReaders++;
    }

    public synchronized void endRead() {
        activeReaders--;
        if (activeReaders == 0) {
            notifyAll();
        }
    }

    public synchronized void beginWrite() throws InterruptedException {
        waitingWriters++;
        while (activeReaders > 0 || activeWriters > 0) {
            wait();
        }
        waitingWriters--;
        activeWriters++;
    }

    public synchronized void endWrite() {
        activeWriters--;
        notifyAll();
    }
}
```

Реализовал классы Generator и Integrator, наследуемые от Thread.

```java
package threads;

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

public class Generator extends Thread {
    private final Task task;
    private final ReadWriteSemaphore semaphore;

    public Generator(Task task, ReadWriteSemaphore semaphore) {
        this.task = task;
        this.semaphore = semaphore;
    }

    @Override
    public void run() {
        try {
            for (int i = 0; i < task.getTasksCount(); i++) {
                // Проверяем прерывание
                if (Thread.interrupted()) {
                    throw new InterruptedException();
                }

                // Генерируем случайные параметры
                double base = 1 + Math.random() * 9;
                double left = Math.random() * 100;
                double right = 100 + Math.random() * 100;
                double step = Math.random();

                Function logFunction = new Log(base);

                // Используем семафор для записи
                semaphore.beginWrite();
                try {
                    task.setFunction(logFunction);
                    task.setLeftBorder(left);
                    task.setRightBorder(right);
                    task.setStep(step);
                } finally {
                    semaphore.endWrite();
                }

                System.out.printf("Generator[%d]: Source %.4f %.4f %.4f (log base=%.4f)%n",
                                  i + 1, left, right, step, base);

                // Небольшая пауза для наглядности
                Thread.sleep(10);
            }
        } catch (InterruptedException e) {
            System.out.println("Generator was interrupted - stopping work");
        } finally {
            System.out.println("Generator finished work");
        }
    }
}
```

```java
package threads;

import functions.Functions;

public class Integrator extends Thread {
    private final Task task;
    private final ReadWriteSemaphore semaphore;

    public Integrator(Task task, ReadWriteSemaphore semaphore) {
        this.task = task;
        this.semaphore = semaphore;
    }

    @Override
    public void run() {
        int processedCount = 0;
        try {
            for (int i = 0; i < task.getTasksCount(); i++) {
                // Проверяем прерывание
                if (Thread.interrupted()) {
                    throw new InterruptedException();
                }

                double left, right, step, result;

                // Используем семафор для чтения
                semaphore.beginRead();
                try {
                    // Проверяем, что функция установлена
                    if (task.getFunction() == null) {
                        System.out.println("Integrator: Function is null, skipping...");
                        continue;
                    }

                    left = task.getLeftBorder();
                    right = task.getRightBorder();
                    step = task.getStep();

                    // Вычисляем интеграл
                    result = Functions.integrate(task.getFunction(), left, right, step);
                    processedCount++;
                } finally {
                    semaphore.endRead();
                }

                System.out.printf("Integrator[%d]: Result %.4f %.4f %.4f %.8f%n",
                                  processedCount, left, right, step, result);

                // Небольшая пауза для наглядности
                Thread.sleep(10);
            }
        } catch (InterruptedException e) {
            System.out.println("Integrator was interrupted - stopping work");
        } catch (IllegalArgumentException e) {
            System.out.printf("Integrator ERROR: %s%n", e.getMessage());
        } finally {
            System.out.println("Integrator finished work. Processed: " + processedCount + " tasks");
        }
    }
}
```

Вывод Main

```
================================================================================
Вычисление интеграла методом трапеций
================================================================================

1. Вычисление интеграла экспоненты на отрезке [0, 1]
------------------------------------------------------------
Шаг          Численный       Теоретический   Погрешность
0,1000       1,7197134914    1,7182818285    0,0014316629
0,0100       1,7182961475    1,7182818285    0,0000143190
0,0010       1,7182819716    1,7182818285    0,0000001432
0,0001       1,7182818299    1,7182818285    0,0000000014


2. Поиск шага для точности 1e-7
------------------------------------------------
Итер.  Шаг          Численный       Погрешность
0      0,10000000   1,7197134914    0,0014316629
1      0,05000000   1,7186397889    0,0003579605
2      0,02500000   1,7183713214    0,0000894929
3      0,01250000   1,7183042019    0,0000223734
4      0,00625000   1,7182874218    0,0000055934
5      0,00312500   1,7182832268    0,0000013983
6      0,00156250   1,7182821780    0,0000003496
7      0,00078125   1,7182819159    0,0000000874    ? ДОСТИГНУТО

Требуемая точность достигнута при шаге: 7.8125E-4

3. Проверка с очень малым шагом
Шаг: 1,00e-06, Погрешность: 2,11e-11


3. Тестирование обработки исключений
------------------------------------------------
Поймано исключение: Интервал интегрирования выходит за границы области определения функции
Поймано исключение: Шаг дискретизации должен быть положительным
================================================================================
Последовательная версия (без потоков)
================================================================================
Запуск последовательной версии программы...
Создание и выполнение 100 заданий интегрирования
--------------------------------------------------------------------------------
Source 34,4276 131,6341 0,6906 (log base=6,3789)
Result 34,4276 131,6341 0,6906 228,46097171
Source 53,3155 166,3038 0,6577 (log base=1,1394)
Result 53,3155 166,3038 0,6577 4025,64325917
Source 19,3376 137,2036 0,3170 (log base=5,2797)
Result 19,3376 137,2036 0,3170 300,56187283
Source 21,7497 179,7124 0,9904 (log base=4,3250)
Result 21,7497 179,7124 0,9904 483,47506042
Source 34,1303 138,6844 0,6439 (log base=6,8992)
Result 34,1303 138,6844 0,6439 237,63991009
Source 69,9614 110,1841 0,8113 (log base=7,5190)
Result 69,9614 110,1841 0,8113 89,56279780
Source 71,6178 116,7428 0,8834 (log base=6,2663)
Result 71,6178 116,7428 0,8834 111,52195428
Source 40,6779 104,8380 0,5991 (log base=1,7351)
Result 40,6779 104,8380 0,5991 495,13474141
Source 69,9114 109,6397 0,2410 (log base=9,0813)
Result 69,9114 109,6397 0,2410 80,83569308
Source 42,8634 166,5459 0,7347 (log base=4,9633)
Result 42,8634 166,5459 0,7347 354,01798091
```

...

```
--------------------------------------------------------------------------------
Статистика выполнения:
Успешно выполнено: 100 заданий
Завершено с ошибкой: 0 заданий
Общее время выполнения: 1197 мс
Среднее время на задание: 11,97 мс
================================================================================
Простая многопоточная версия
================================================================================
Запуск простой многопоточной версии программы...
Создание и выполнение 100 заданий интегрирования
--------------------------------------------------------------------------------
Generator: Source 92,2440 189,1320 0,4183 (log base=8,9680)
Integrator: Result 92,2440 189,1320 0,4183 217,56967297
Integrator: Result 92,2440 189,1320 0,4183 217,56967297
Generator: Source 57,0046 189,2279 0,8576 (log base=7,6627)
Generator: Source 55,6029 182,3284 0,5725 (log base=5,5439)
Integrator: Result 55,6029 182,3284 0,5725 349,74854083
Generator: Source 22,2876 187,2353 0,3181 (log base=8,3542)
Integrator: Result 22,2876 187,2353 0,3181 351,21831840
Generator: Source 24,5061 162,0504 0,0650 (log base=4,6858)
Integrator: Result 24,5061 162,0504 0,0650 394,00981538
Generator: Source 83,2260 179,0004 0,7877 (log base=2,5257)
Integrator: Result 83,2260 179,0004 0,7877 501,64250734
Generator: Source 15,4461 136,1174 0,4300 (log base=6,2436)
Integrator: Result 15,4461 136,1174 0,4300 276,19310489
Generator: Source 19,7020 147,9889 0,0745 (log base=3,1713)
Integrator: Result 19,7020 147,9889 0,0745 478,72149067
Generator: Source 90,3110 126,6395 0,1371 (log base=3,4816)
Integrator: Result 90,3110 126,6395 0,1371 136,34143898
Generator: Source 64,1674 110,5804 0,2253 (log base=1,1580)
Integrator: Result 64,1674 110,5804 0,2253 1410,93401095
Generator: Source 75,0923 125,1600 0,1535 (log base=9,4675)
Integrator: Result 75,0923 125,1600 0,1535 102,36452106
Generator: Source 15,0564 160,5693 0,4924 (log base=9,4967)
Integrator: Result 15,0564 160,5693 0,4924 279,50175824
Generator: Source 79,0375 135,2075 0,9422 (log base=3,9920)
Integrator: Result 79,0375 135,2075 0,9422 189,18084334
...
--------------------------------------------------------------------------------
Оба потока завершили работу
Общее время выполнения: 1138 мс


Тестирование с разными приоритетами потоков
============================================================

Тест 1: Приоритет генератора = NORMAL
------------------------------------------------
Generator: Source 94,3784 189,2389 0,6781 (log base=2,3419)
Integrator: Result 94,3784 189,2389 0,6781 550,14003453
Generator: Source 85,9432 194,9736 0,7633 (log base=7,3212)
Integrator: Result 85,9432 194,9736 0,7633 269,37896371
Generator: Source 11,3639 118,8915 0,2753 (log base=8,3419)
Integrator: Result 11,3639 118,8915 0,2753 204,09375266
Generator: Source 50,4285 119,1399 0,7330 (log base=5,4412)
Integrator: Result 50,4285 119,1399 0,7330 178,92866373
Generator: Source 35,2039 117,0837 0,0297 (log base=4,9887)
Integrator: Result 35,2039 117,0837 0,0297 218,02767525
Integrator: Result 35,2039 117,0837 0,0297 218,02767525
Generator: Source 71,8654 190,8168 0,2116 (log base=7,6788)
Generator: Source 37,5599 135,4239 0,1345 (log base=9,6695)
Integrator: Result 37,5599 135,4239 0,1345 189,80473724
Generator: Source 35,9985 135,7224 0,7195 (log base=9,2725)
...
Время выполнения: 581 мс
Тест 2: Приоритет генератора = HIGH
------------------------------------------------
Generator: Source 92,6393 123,0370 0,5877 (log base=7,6951)
Integrator: Result 92,6393 123,0370 0,5877 69,67542318
Generator: Source 93,2623 101,2881 0,5434 (log base=8,0451)
Integrator: Result 93,2623 101,2881 0,5434 17,61867230
Generator: Source 13,6680 124,9439 0,7958 (log base=3,4009)
Integrator: Result 13,6680 124,9439 0,7958 372,69329994
Generator: Source 88,2065 111,2513 0,5941 (log base=2,8052)
Integrator: Result 88,2065 111,2513 0,5941 102,77774829
Generator: Source 4,3759 194,4246 0,3811 (log base=7,4270)
Integrator: Result 4,3759 194,4246 0,3811 412,99874935
Generator: Source 41,1467 178,1774 0,4449 (log base=7,0593)
Integrator: Result 41,1467 178,1774 0,4449 324,13578707
Generator: Source 82,7656 142,4120 0,8850 (log base=9,2914)
Integrator: Result 82,7656 142,4120 0,8850 126,07904782
Generator: Source 41,9298 120,3056 0,6455 (log base=7,5770)
Integrator: Result 41,9298 120,3056 0,6455 168,50530571
Generator: Source 59,9113 149,6291 0,8466 (log base=4,6591)
Integrator: Result 59,9113 149,6291 0,8466 269,32293480
Generator: Source 38,0306 167,8570 0,1536 (log base=6,5877)
Integrator: Result 38,0306 167,8570 0,1536 313,89250611
Generator: Source 94,8171 164,4750 0,4648 (log base=9,8731)
Integrator: Result 94,8171 164,4750 0,4648 147,61720115
Generator: Source 36,2634 125,8647 0,7666 (log base=6,1525)
Integrator: Result 36,2634 125,8647 0,7666 213,97563478
Generator: Source 72,7680 108,2411 0,6130 (log base=6,1194)
...
Время выполнения: 573 мс
```

```
Тест 3: Приоритет генератора = LOW
--------------------------------------------------
Generator: Source 47,0139 176,2484 0,0306 (log base=7,5236)
Integrator: Result 47,0139 176,2484 0,0306 297,95176179
Generator: Source 91,6687 135,2447 0,7973 (log base=6,9813)
Integrator: Result 91,6687 135,2447 0,7973 105,96034542
Generator: Source 42,6940 124,4492 0,1877 (log base=4,9095)
Integrator: Result 42,6940 124,4492 0,1877 225,18063259
Generator: Source 2,1412 111,5897 0,2504 (log base=4,6767)
Integrator: Result 2,1412 111,5897 0,2504 269,05603121
Generator: Source 99,7581 189,5677 0,2969 (log base=7,8719)
Integrator: Result 99,7581 189,5677 0,2969 215,80218778
Generator: Source 59,3432 130,5026 0,8575 (log base=7,5024)
Integrator: Result 59,3432 130,5026 0,8575 159,90873885
Generator: Source 57,0422 162,9823 0,1095 (log base=4,1838)
Integrator: Result 57,0422 162,9823 0,1095 344,85456207
Generator: Source 68,1695 171,7679 0,6871 (log base=1,4789)
Integrator: Result 68,1695 171,7679 0,6871 1258,67576565
Generator: Source 89,0834 113,0452 0,8272 (log base=3,3493)
Integrator: Result 89,0834 113,0452 0,8272 91,45405440
Generator: Source 63,7065 110,2905 0,8734 (log base=7,0219)
Integrator: Result 63,7065 110,2905 0,8734 106,44752197
Generator: Source 40,4761 145,9141 0,2620 (log base=2,4380)
Integrator: Result 40,4761 145,9141 0,2620 529,48551150
```

...

```
Время выполнения: 573 мс
```

```
================================================================================
Версия с семафором
================================================================================
Запуск версии с семафором...
Создание и выполнение 100 заданий интегрирования
--------------------------------------------------------------------------------
Generator[1]: Source 68,4963 143,0256 0,9203 (log base=9,1341)
Integrator[1]: Result 68,4963 143,0256 0,9203 156,32421089
Generator[2]: Source 3,2943 152,9420 0,9747 (log base=9,0869)
Integrator[2]: Result 3,2943 152,9420 0,9747 279,00090728
Generator[3]: Source 42,4883 105,4490 0,3584 (log base=1,9886)
Integrator[3]: Result 42,4883 105,4490 0,3584 391,23109793
Generator[4]: Source 40,3335 192,7123 0,6176 (log base=1,1635)
Integrator[4]: Result 40,3335 192,7123 0,6176 4705,66458410
Generator[5]: Source 22,9315 129,1327 0,6264 (log base=8,2342)
Integrator[5]: Result 22,9315 129,1327 0,6264 213,28086316
Generator[6]: Source 7,4320 143,9267 0,5344 (log base=7,6021)
```

...

```
Generator finished work
Integrator finished work. Processed: 100 tasks
--------------------------------------------------------------------------------
Оба потока завершили работу
Общее время выполнения: 1141 мс


Тестирование с прерыванием потоков через 50 мс
============================================================
Запуск потоков...
Generator[1]: Source 47,4723 108,5887 0,9709 (log base=1,3001)
Integrator[1]: Result 47,4723 108,5887 0,9709 1008,45398111
Integrator[2]: Result 47,4723 108,5887 0,9709 1008,45398111
Generator[2]: Source 20,0099 101,4465 0,5791 (log base=4,1815)
Generator[3]: Source 86,4011 128,7627 0,4162 (log base=6,5696)
Integrator[3]: Result 86,4011 128,7627 0,4162 105,12982573
Generator[4]: Source 75,7625 115,6346 0,3663 (log base=7,0287)
Integrator[4]: Result 75,7625 115,6346 0,3663 93,11431979
Generator[5]: Source 15,1662 165,4941 0,4390 (log base=7,2285)
Integrator[5]: Result 15,1662 165,4941 0,4390 330,59651412
Прерывание потоков...
Generator was interrupted - stopping work
Generator finished work
Integrator was interrupted - stopping work
Integrator finished work. Processed: 5 tasks
Тестирование прерывания завершено


Тестирование семафора с разными приоритетами
============================================================

Тест 1: Приоритет генератора = NORM
--------------------------------------------------
Generator[1]: Source 32,1126 168,9570 0,2935 (log base=2,7598)
Integrator[1]: Result 32,1126 168,9570 0,2935 609,20587924
Integrator[2]: Result 32,1126 168,9570 0,2935 609,20587924
Generator[2]: Source 44,4215 182,8710 0,5438 (log base=4,0964)
Integrator[3]: Result 44,4215 182,8710 0,5438 457,80990365
Generator[3]: Source 97,2133 102,8308 0,9807 (log base=2,4317)
Integrator[4]: Result 97,2133 102,8308 0,9807 29,11359563
Generator[4]: Source 65,3410 165,0286 0,5750 (log base=7,6080)
Integrator[5]: Result 65,3410 165,0286 0,5750 231,55352412
Generator[5]: Source 51,4544 104,0297 0,8839 (log base=5,3832)
Integrator[6]: Result 51,4544 104,0297 0,8839 135,35537835
```

...

```
Integrator finished work. Processed: 30 tasks
Generator finished work
Время выполнения: 348 мс

Тест 2: Приоритет генератора = MAX
--------------------------------------------------
Generator[1]: Source 9,4377 196,2269 0,4547 (log base=4,1902)
Integrator[1]: Result 9,4377 196,2269 0,4547 577,88498908
Generator[2]: Source 3,0934 150,2397 0,5875 (log base=7,4794)
Integrator[2]: Result 3,0934 150,2397 0,5875 299,37463439
Generator[3]: Source 99,1587 131,8195 0,5270 (log base=1,4724)
Integrator[3]: Result 99,1587 131,8195 0,5270 400,65796964
Integrator[4]: Result 99,1587 131,8195 0,5270 400,65796964
Generator[4]: Source 38,5791 196,4887 0,7561 (log base=3,9749)
Integrator[5]: Result 38,5791 196,4887 0,7561 535,32948896
Generator[5]: Source 64,6495 143,8978 0,3412 (log base=3,2989)
Integrator[6]: Result 64,6495 143,8978 0,3412 306,86665596
Generator[6]: Source 49,6440 105,7141 0,8535 (log base=9,5561)
Integrator[7]: Result 49,6440 105,7141 0,8535 107,55967744
Generator[7]: Source 21,8619 141,8770 0,2568 (log base=3,9930)
Integrator[8]: Result 21,8619 141,8770 0,2568 372,35419978
```
...
```
Integrator finished work. Processed: 30 tasks
Generator finished work
Время выполнения: 345 мс

Тест 3: Приоритет генератора = MIN
--------------------------------------------------
Generator[1]: Source 4,2402 153,5372 0,0160 (log base=8,2160)
Integrator[1]: Result 4,2402 153,5372 0,0160 293,18593621
Generator[2]: Source 56,8001 181,8642 0,7637 (log base=9,8300)
Integrator[2]: Result 56,8001 181,8642 0,7637 258,93338516
Integrator[3]: Result 56,8001 181,8642 0,7637 258,93338516
Generator[3]: Source 19,2486 139,9084 0,4714 (log base=7,2709)
Generator[4]: Source 79,5787 122,4510 0,9834 (log base=6,2517)
Integrator[4]: Result 79,5787 122,4510 0,9834 107,77713322
Generator[5]: Source 24,7347 175,6252 0,8698 (log base=3,3196)
Integrator[5]: Result 24,7347 175,6252 0,8698 564,61348153
Integrator[6]: Result 24,7347 175,6252 0,8698 564,61348153
Generator[6]: Source 31,6325 185,1497 0,5334 (log base=1,4486)
Integrator[7]: Result 31,6325 185,1497 0,5334 1899,53958359
Generator[7]: Source 2,3405 150,4126 0,2352 (log base=7,7895)
```
...
```
Integrator finished work. Processed: 30 tasks
Generator finished work
Время выполнения: 344 мс
```