

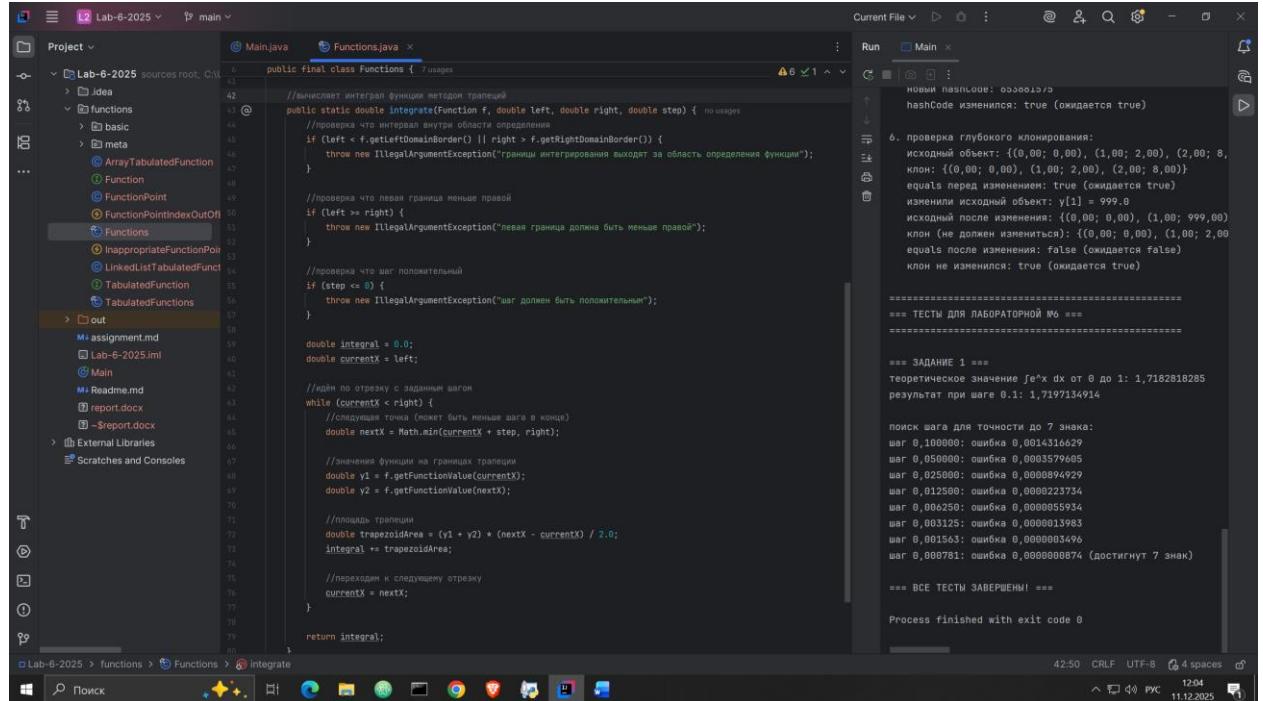
Отчёт по лабораторной работе №6
Тенигин Валерий 6204-010302D

Задание 1

Добавил:

метод integrate в клас Functions (реализовал численное интегрирование методом трапеций)

проверки - область определения, корректность границ, положительность шага



```
public final class Functions { ... }

    //вычисляет интеграл функции методом трапеций
    public static double integrate(Function f, double left, double right, double step) {
        //проверка что интеграл внутри области определения
        if (left <= f.getLeftDomainBorder() || right > f.getRightDomainBorder()) {
            throw new IllegalArgumentException("границы интегрирования выходят за область определения функции");
        }

        //проверка что левая граница меньше правой
        if (left >= right) {
            throw new IllegalArgumentException("левая граница должна быть меньше правой");
        }

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

        double integral = 0.0;
        double currentX = left;

        //идём по отрезку с заданным шагом
        while (currentX < right) {
            //следующая точка (может быть меньше шага в конце)
            double nextX = Math.min(currentX + step, right);

            //значение функции на границах трапеции
            double y1 = f.getFunctionValue(currentX);
            double y2 = f.getFunctionValue(nextX);

            //площадь трапеции
            double trapezoidArea = (y1 + y2) * (nextX - currentX) / 2.0;
            integral += trapezoidArea;

            //переходим к следующему отрезку
            currentX = nextX;
        }

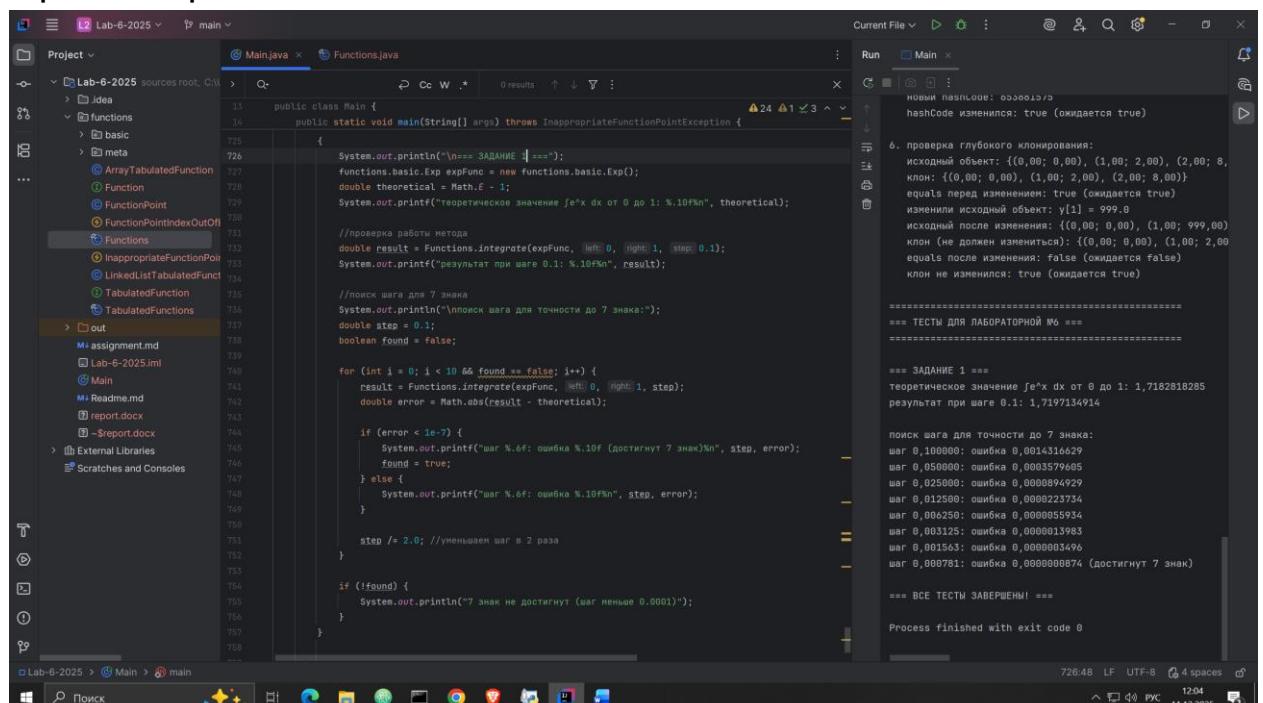
        return integral;
    }
}

==== ТЕСТЫ ДЛЯ ЛАБОРАТОРНОЙ № 6 ====
==== ЗАДАНИЕ 1 ====
теоретическое значение  $\int_0^1 e^x dx$  от 0 до 1: 1.7182818285
результат при шаге 0.1: 1.7197134914

поиск шага для точности до 7 знака:
шаг 0.100000: ошибка 0.001431629
шаг 0.050000: ошибка 0.0003579605
шаг 0.025000: ошибка 0.0000894929
шаг 0.012500: ошибка 0.00000223734
шаг 0.006250: ошибка 0.00000055934
шаг 0.003125: ошибка 0.00000013983
шаг 0.001563: ошибка 0.00000003496
шаг 0.000781: ошибка 0.00000000874 (достигнут 7 знаков)

==== ВСЕ ТЕСТЫ ЗАВЕРШЕНЫ! ====
Process finished with exit code 0
```

Протестировал в Main



```
public class Main {
    public static void main(String[] args) throws InappropriateFunctionPointException {
        System.out.println("\n==== ЗАДАНИЕ 1 ====");
        functions.basic.Exp exprFunc = new functions.basic.Exp();
        double theoretical = Math.E - 1;
        System.out.printf("теоретическое значение  $\int_0^1 e^x dx$  от 0 до 1: %.10f\n", theoretical);

        //проверка работы метода
        double result = Functions.integrate(exprFunc, left: 0, right: 1, step: 0.1);
        System.out.printf("результат при шаге 0.1: %.10f\n", result);

        //поиск шага для 7 знака
        System.out.println("поиск шага для точности до 7 знака:");
        double step = 0.1;
        boolean found = false;

        for (int i = 0; i < 10 && found == false; i++) {
            result = Functions.integrate(exprFunc, left: 0, right: 1, step);
            double error = Math.abs(result - theoretical);

            if (error < 1e-7) {
                System.out.printf("шаг %.6f: ошибка %.10f (достигнут 7 знаков)\n", step, error);
                found = true;
            } else {
                System.out.printf("шаг %.6f: ошибка %.10f\n", step, error);
            }

            step /= 2.0; //уменьшаем шаг в 2 раза
        }

        if (!found) {
            System.out.println("7 знак не достигнут (шаг меньше 0.0001)");
        }
    }
}

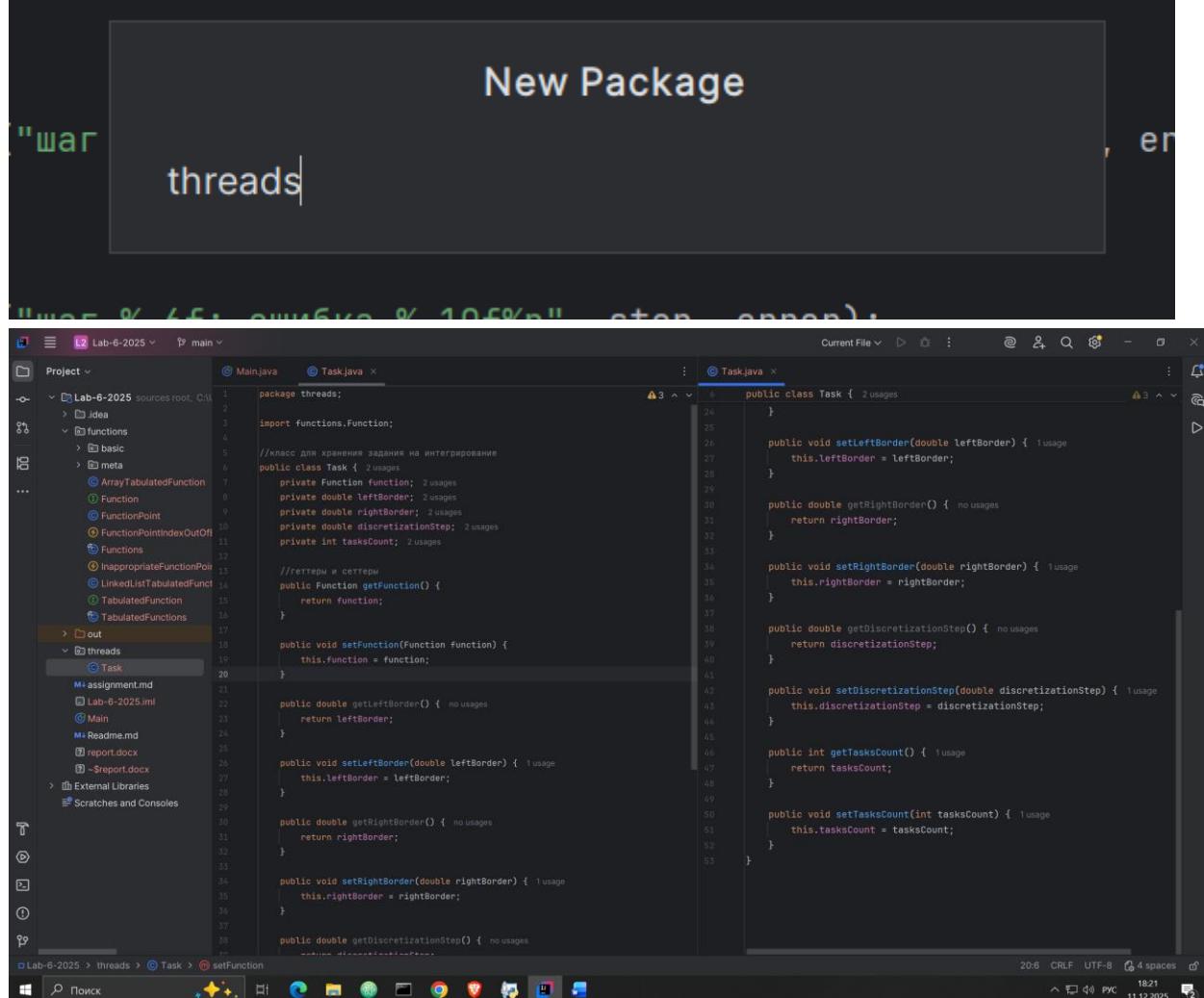
==== ТЕСТЫ ДЛЯ ЛАБОРАТОРНОЙ № 6 ====
==== ЗАДАНИЕ 1 ====
теоретическое значение  $\int_0^1 e^x dx$  от 0 до 1: 1.7182818285
результат при шаге 0.1: 1.7197134914

поиск шага для точности до 7 знака:
шаг 0.100000: ошибка 0.001431629
шаг 0.050000: ошибка 0.0003579605
шаг 0.025000: ошибка 0.0000894929
шаг 0.012500: ошибка 0.00000223734
шаг 0.006250: ошибка 0.00000055934
шаг 0.003125: ошибка 0.00000013983
шаг 0.001563: ошибка 0.00000003496
шаг 0.000781: ошибка 0.00000000874 (достигнут 7 знаков)

==== ВСЕ ТЕСТЫ ЗАВЕРШЕНЫ! ====
Process finished with exit code 0
```

Задание 2

Создал класс Task для хранения задания
~~s(result - theoretical);~~



The screenshot shows a Java code editor with the following details:

- New Package:** A modal dialog titled "New Package" is open, showing the package name "threads" being typed.
- Project Structure:** The project is named "Lab-6-2025". It contains a "sources root" folder with "Lab-6-2025" and "threads" subfolders. "threads" contains a "Task" class.
- Task.java Content:**

```
package threads;
import functions.Function;

//класс для хранения задания на интегрирование
public class Task {
    private Function function;
    private double leftBorder;
    private double rightBorder;
    private double discretizationStep;
    private int tasksCount;

    // getters и setters
    public Function getFunction() {
        return function;
    }

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

    public double getLeftBorder() {
        return leftBorder;
    }

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

    public double getRightBorder() {
        return rightBorder;
    }

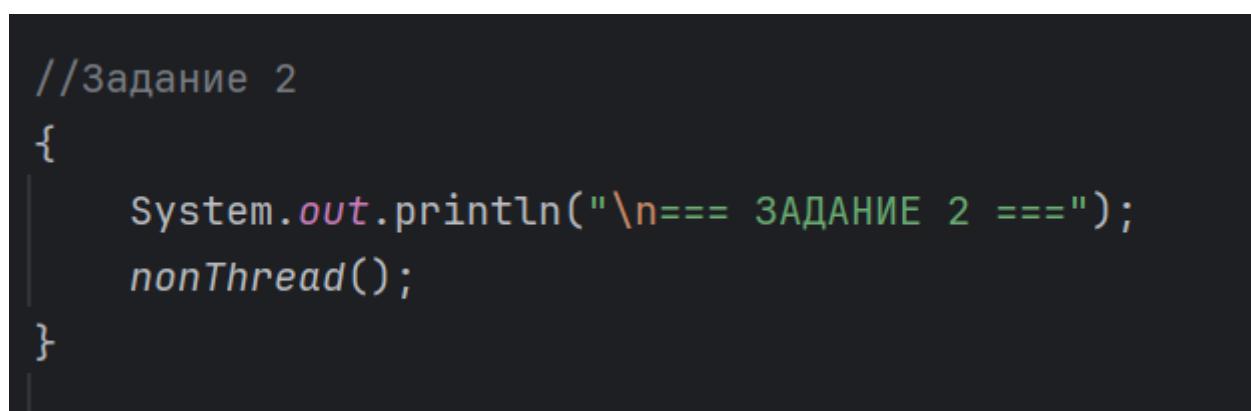
    public void setRightBorder(double rightBorder) {
        this.rightBorder = rightBorder;
    }

    public double getDiscretizationStep() {
        return discretizationStep;
    }

    public void setDiscretizationStep(double discretizationStep) {
        this.discretizationStep = discretizationStep;
    }

    public int getTasksCount() {
        return tasksCount;
    }

    public void setTasksCount(int tasksCount) {
        this.tasksCount = tasksCount;
    }
}
```
- Bottom Status Bar:** Shows file paths (Lab-6-2025 > threads > Task > MainFunction), encoding (UTF-8), and date/time (11.12.2025).



```
//Задание 2
{
    System.out.println("\n==== ЗАДАНИЕ 2 ====");
    nonThread();
}
```

Реализовал метод nonThread() - 100 заданий в одном потоке, где генерируются случайные данные

The screenshot shows a Java development environment with the following details:

- Project:** Lab-6-2025
- File:** Main.java
- Code Snippet:**

```
public class Main {  
    private static void nonThread() {  
        //создаем объект задания  
        threads.Task task = new threads.Task();  
        task.setTasksCount(100);  
  
        //цикл по всем заданиям  
        for (int i = 0; i < task.getTasksCount(); i++) {  
            try {  
                //создаем логарифмическую функцию со случайным основанием от 1 до 10  
                double base = 1 + Math.random() * 9; //случайное число от 1 до 10  
                functions.basic.Log logFunc = new functions.basic.Log(base);  
                task.setFunction(logFunc);  
  
                //левая граница от 0 до 100  
                double left = Math.random() * 100;  
                task.setLeftBorder(left);  
  
                //правая граница от 100 до 200  
                double right = 100 + Math.random() * 100;  
                task.setRightBorder(right);  
  
                //шаг дискретизации от 0 до 1  
                double step = Math.random();  
                task.setDiscretizationStep(step);  
  
                System.out.printf("Source %.3f %.3f %.3f%.6f%n", left, right, step);  
  
                double result = Functions.integrate(logFunc, left, right, step);  
  
                System.out.printf("Result %.3f %.3f %.3f%.6f%n", left, right, step, result);  
            } catch (IllegalArgumentException e) {  
                System.out.println("ошибка: " + e.getMessage());  
            }  
        }  
    }  
}
```

- Output:** The right panel displays the results of the 100 tasks. It includes:
 - == ЗАДАНИЕ 1 ==
 - теоретическое значение $\int e^x dx$ от 0 до 1: 1,7182818285
 - результат при шаге 0.1: 1,7197134914
 - поиск шага для точности до 7 знаков:
 - шаг 0,100000: ошибка 0,001431629
 - шаг 0,050000: ошибка 0,0003579605
 - шаг 0,025000: ошибка 0,0000894929
 - шаг 0,012500: ошибка 0,0000223734
 - шаг 0,006250: ошибка 0,0000055934
 - шаг 0,003125: ошибка 0,0000013983
 - шаг 0,001563: ошибка 0,0000003496
 - шаг 0,000781: ошибка 0,0000000874 (достигнут 7 знаков)- == ЗАДАНИЕ 2 ==
- Source 2,636 119,683 0,325
- Result 2,636 119,683 0,325 634,969790
- Source 18,375 194,731 0,058
- Result 18,375 194,731 0,058 1416,626668
- Source 50,284 159,977 0,388
- Result 50,284 159,977 0,388 569,428682
- Source 29,651 104,733 0,934
- Result 29,651 104,733 0,934 224,418646
- Source 88,644 196,881 0,078
- Result 88,644 196,881 0,078 351,967155
- Source 13,853 182,537 0,559
- Result 13,853 182,537 0,559 641,445298
- Source 7,450 108,129 0,556
- Result 7,450 108,129 0,556 242,728898
- Source 76,739 176,994 0,829
- Result 76,739 176,994 0,829 213,179008
- Source 48,710 169,857 0,015
- Result 48,710 169,857 0,015 253,188305
- Source 85,258 103,984 0,740

- Bottom Status Bar:** 49:10 LF UTF-8 1822 RYC 11.12.2025

Задание 3

Создал два класса: SimpleGenerator (генерирует задания) и SimpleIntegrator (вычисляет интегралы)

The image shows two side-by-side Java code editors from an IDE. Both editors have a project tree on the left and three tabs at the top: Main.java, SimpleGenerator.java, and SimpleIntegrator.java. The left editor displays the content of SimpleGenerator.java, and the right editor displays the content of SimpleIntegrator.java.

SimpleGenerator.java:

```
package threads;

import functions.basic.Log;

public class SimpleGenerator implements Runnable {
    private final Task task; 6 usages

    public SimpleGenerator(Task task) { 5 usages
        this.task = task;
    }

    @Override
    public void run() {
        int tasksCount = task.getTasksCount();
        for (int i = 0; i < tasksCount; i++) {
            //создаем логарифмическую функцию
            double base = 1 + Math.random() * 9;
            task.setFunction(new Log(base));

            //левая граница
            double left = Math.random() * 100;
            task.setLeftBorder(left);

            //правая граница
            double right = 100 + Math.random() * 100;
            task.setRightBorder(right);

            //шаг дискретизации
            double step = Math.random();
            task.setDiscretizationStep(step);
        }
        System.out.printf("Generator: Source %.3f %.3f %.3f%n", left, right, step);
    }
}
```

SimpleIntegrator.java:

```
package threads;

import functions.Functions;

public class SimpleIntegrator implements Runnable {
    private final Task task; 6 usages

    public SimpleIntegrator(Task task) { 5 usages
        this.task = task;
    }

    @Override
    public void run() {
        int tasksCount = task.getTasksCount();
        for (int i = 0; i < tasksCount; i++) {
            try {
                //читаем параметры
                double left = task.getLeftBorder();
                double right = task.getRightBorder();
                double step = task.getDiscretizationStep();

                //вычисляем интеграл
                double result = Functions.integrate(task.getFunction(), left, right, step);

                System.out.printf("Integrator: Result %.3f %.3f %.6f%n", left, right, result);
            } catch (IllegalArgumentException e) {
                System.out.println("Integrator: ошибка вычисления - " + e.getMessage());
            } catch (NullPointerException e) {
                System.out.println("Integrator: NullPointerException - нет данных для обработки");
                i = tasksCount;
            }
        }
    }
}
```

```
public class Main {
    private static void nonThread() { // usage
        System.out.printf("Result %.3f %.3f %.3f %.6f%n", left, right, step);
    } catch (IllegalArgumentException e) {
        System.out.println("ошибка: " + e.getMessage());
    }
}

private static void simpleThreads() { // usage
    //создаем объект задания
    threads.Task task = new threads.Task();
    task.setTasksCount(100);

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

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

    //ждем завершения потоков
    try {
        generatorThread.join();
        integratorThread.join();
    } catch (InterruptedException e) {
        System.out.println("основной поток прерван");
    }
}

public static void main(String[] args) throws InappropriateFunctionPointException {
    /*
     */
    System.out.println("\n" + "#".repeat(count * 50));
    System.out.println("#== ТЕСТ ДЛЯ ПАСПОРТОВ М5 ==");
}
```

Main.java

```
public class Main {
    private static void nonThread() { }

    private static void simpleThreads() {
        //создаем объект задания
        threads.Task task = new threads.Task();
        task.setTasksCount(100);

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

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

        //идем завершения потоков
        try {
            generatorThread.join();
            integratorThread.join();
        } catch (InterruptedException e) {
            System.out.println("основной поток прерван");
        }
    }

    public static void main(String[] args) throws InappropriateFunctionPointException {
        System.out.println("\n" + "=" .repeat(count: 50));
        System.out.println("== ТЕСТ ДЛЯ ЛАБОРАТОРНОЙ №5 ==");
        System.out.println("=".repeat(count: 50));

        // [Задание 1]
        System.out.println("\n== TECT 17- FunctionPoint ==");
    }
}
```

SimpleGenerator.java

```
public class SimpleGenerator implements threads.Task {
    private int count;

    public void setTasksCount(int count) {
        this.count = count;
    }

    @Override
    public void generate() {
        for (int i = 0; i < count; i++) {
            System.out.println("Task " + i);
        }
    }
}
```

SimpleIntegrator.java

```
public class SimpleIntegrator implements threads.Task {
    private int count;

    public void setTasksCount(int count) {
        this.count = count;
    }

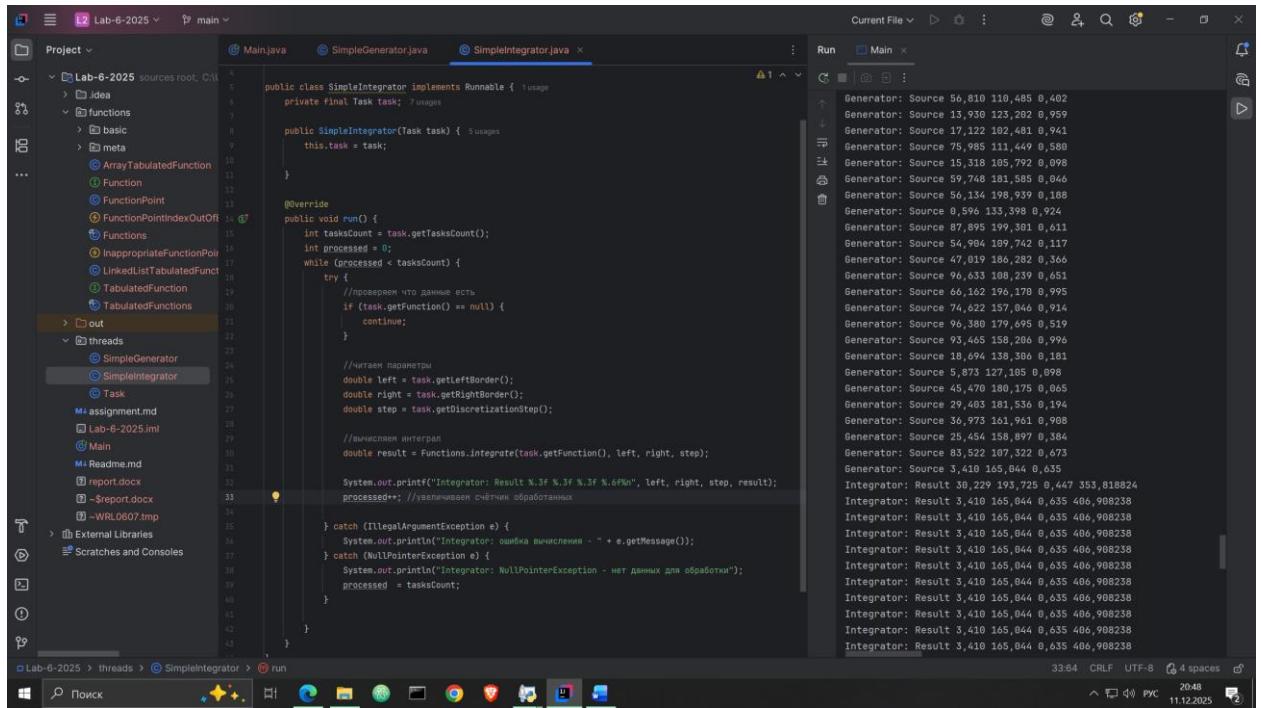
    @Override
    public void integrate() {
        int sum = 0;
        for (int i = 0; i < count; i++) {
            sum += i;
        }
        System.out.println("Sum = " + sum);
    }
}
```

Run

Main

RESULT 71.190 134,320 0,000 127,220114
Source 27,711 146,019 0,938
Result 27,711 146,019 0,938 222,603547
Source 81,745 174,295 0,537
Result 81,745 174,295 0,537 212,110633
== ЗАДАНИЕ 3 ==
Integrator: NullPointerException - нет данных для обработки
Generator: Source 92,106 187,156 0,285
Generator: Source 65,285 150,090 0,802
Generator: Source 94,346 158,478 0,978
Generator: Source 20,448 132,422 0,259
Generator: Source 96,633 141,663 0,614
Generator: Source 0,723 152,574 0,678
Generator: Source 55,839 124,321 0,893
Generator: Source 80,567 110,150 0,144
Generator: Source 34,158 178,849 0,620
Generator: Source 51,282 180,784 0,883
Generator: Source 44,292 176,576 0,995
Generator: Source 88,442 198,179 0,481
Generator: Source 57,395 115,451 0,445
Generator: Source 62,265 151,994 0,041
Generator: Source 83,751 154,564 0,249
Generator: Source 82,175 183,198 0,283
Generator: Source 15,844 149,710 0,937
Generator: Source 97,583 190,860 0,848
Generator: Source 71,118 112,368 0,699
Generator: Source 96,789 197,248 0,501
Generator: Source 90,265 139,168 0,802
Generator: Source 75,384 188,735 0,591
Generator: Source 72,670 184,391 0,943
Generator: Source 65,404 149,075 0,973
Generator: Source 55,669 172,493 0,884
Generator: Source 47,194 182,822 0,841
Generator: Source 67,248 173,089 0,243
Generator: Source 84,474 184,870 0,585

Исправил ошибку NullPointerException без синхронизации, проверяя что данные присутствуют



```
public class SimpleIntegrator implements Runnable { 1 usage
    private final Task task; 7 usages

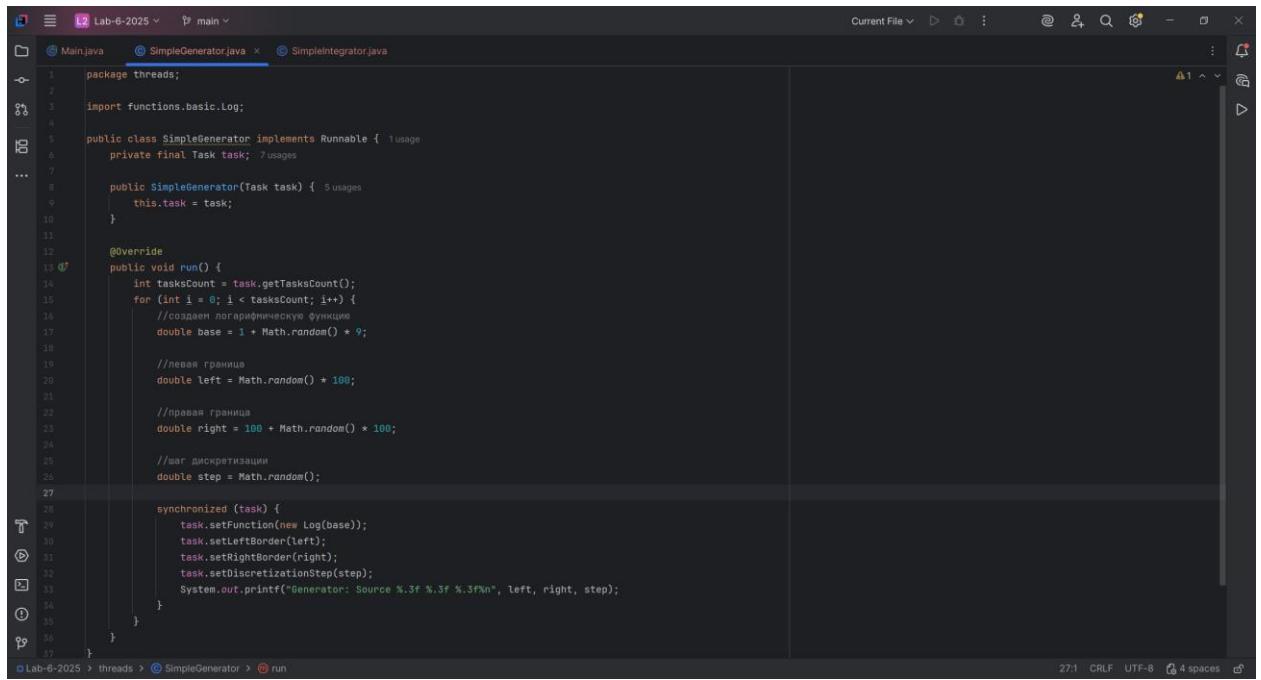
    public SimpleIntegrator(Task task) { 5 usages
        this.task = task;
    }

    @Override
    public void run() {
        int tasksCount = task.getTasksCount();
        int processed = 0;
        while (processed < tasksCount) {
            try {
                //проверим что данные есть
                if (task.getFunction() == null) {
                    continue;
                }

                //читаем параметры
                double left = task.getLeftBorder();
                double right = task.getRightBorder();
                double step = task.getDiscretizationStep();

                //вычисляем интеграл
                double result = Functions.integrate(task.getFunction(), left, right, step);
                System.out.printf("Integrator: Result %.3f %.3f %.6fn", left, right, step, result);
                processed++;
                //увеличиваем счётчик обработанных
            } catch (IllegalArgumentException e) {
                System.out.println("Integrator: ошибка вычисления - " + e.getMessage());
            } catch (NullPointerException e) {
                System.out.println("Integrator: NullPointerException - нет данных для обработки");
                processed = tasksCount;
            }
        }
    }
}
```

Добавил синхронизацию



```
package threads;

import functions.basic.Log;

public class SimpleGenerator implements Runnable { 1 usage
    private final Task task; 7 usages

    public SimpleGenerator(Task task) { 5 usages
        this.task = task;
    }

    @Override
    public void run() {
        int tasksCount = task.getTasksCount();
        for (int i = 0; i < tasksCount; i++) {
            //создаем логарифмическую функцию
            double base = 1 + Math.random() * 9;

            //нижая граница
            double left = Math.random() * 100;

            //правая граница
            double right = 100 + Math.random() * 100;

            // шаг дискретизации
            double step = Math.random();

            synchronized (task) {
                task.setFunction(new Log(base));
                task.setLeftBorder(left);
                task.setRightBorder(right);
                task.setDiscretizationStep(step);
                System.out.printf("Generator: Source %.3f %.3f %.6fn", left, right, step);
            }
        }
    }
}
```

The screenshot shows a Java code editor with the file `SimpleIntegrator.java` open. The code implements a `Runnable` interface, processing multiple tasks. It uses synchronization to ensure thread safety when reading function values and performing integration calculations. Error handling includes catching `IllegalArgumentException` and `NullPointerException`.

```
package threads;

import functions.Functions;

public class SimpleIntegrator implements Runnable {
    private final Task task; // 1 usage

    public SimpleIntegrator(Task task) { // 5 usages
        this.task = task;
    }

    @Override
    public void run() {
        int tasksCount = task.getTasksCount();
        int processed = 0;
        while (processed < tasksCount) {
            double left, right, step;

            synchronized (task) {
                if (task.getFunction() == null) {
                    continue;
                }
                //читаем параметры
                left = task.getLeftBorder();
                right = task.getRightBorder();
                step = task.getDiscretizationStep();
            }

            try {
                //Вычисляем интегрант
                double result = Functions.integrate(task.getFunction(), left, right, step);

                System.out.printf("Integrator: Result %.3f %.3f %.6f\n", left, right, step, result);
                processed++;
            } catch (IllegalArgumentException e) {
                System.out.println("Integrator: ошибка вычисления - " + e.getMessage());
            } catch (NullPointerException e) {
                System.out.println("Integrator: NullPointerException - нет данных для обработки");
                processed = tasksCount;
            }
        }
    }
}
```

Задание 4

Сделал семафор `ReadWriteSemaphore` для управления доступом к данным

The screenshot shows a Java code editor with the file `ReadWriteSemaphore.java` open. This class implements a semaphore for managing access to shared data. It provides methods for starting and ending both write and read operations, using `synchronized` blocks and `wait()`/`notifyAll()` to control access.

```
package threads;

public class ReadWriteSemaphore {
    private boolean canWrite = true; // 4 usages
    private boolean canRead = false; // 4 usages

    //начинаем операцию записи (блокируем чтение)
    public synchronized void startWrite() throws InterruptedException {
        //когда пока можно будет писать
        while (!canWrite) {
            wait();
        }
        canRead = false; //запрещаем чтение пока идет запись
    }

    //заканчиваем операцию записи (разрешаем чтение)
    public synchronized void endWrite() { //1 usage
        canWrite = false; //запрещаем дальнейшую запись
        canRead = true; //разрешаем чтение
        notifyAll(); //будим потоки ожидающие чтения
    }

    //начинаем операцию чтения (блокирует запись)
    public synchronized void startRead() throws InterruptedException {
        //когда пока можно будет читать
        while (!canRead) {
            wait();
        }
        canWrite = false;
    }

    //заканчиваем операцию чтения (разрешаем запись)
    public synchronized void endRead() { //2 usages
        canRead = false;
        canWrite = true;
        notifyAll();
    }
}
```

Создал классы Generator и Integrator, которые наследую от Thread

```
1 package threads;
2 import functions.Functions;
3
4 public class Integrator extends Thread { 2 usages
5     private final Task task; 6 usages
6     private final ReadWriteSemaphore semaphore; 4 usages
7
8     public Integrator(Task task, ReadWriteSemaphore semaphore) { 5 usages
9         this.task = task;
10        this.semaphore = semaphore;
11    }
12
13    @Override
14    public void run() {
15        int tasksCount = task.getTasksCount();
16        for (int i = 0; i < tasksCount; i++) {
17            try {
18                //захватываем семафор для чтения
19                semaphore.startRead();
20
21                //проверяем прерывание
22                if (Thread.interrupted()) {
23                    System.out.println("Integrator: получен сигнал прерывания");
24                    return;
25                }
26                //читаем и обрабатываем данные
27                double left = task.getLeftBorder();
28                double right = task.getRightBorder();
29                double step = task.getDiscretizationStep();
30
31                double result = Functions.integrate(task.getFunction(), left, right, step);
32                System.out.printf("Integrator[%d]: Result %.3f %.3f %.6f%n", i, left, right, step, result);
33
34                semaphore.endRead(); //освобождаем семафор
35
36            } catch (InterruptedException e) {
37                System.out.println("Integrator: прерван во время ожидания");
38                return;
39            } catch (IllegalArgumentException e) {
40                System.out.println("Integrator: ошибка вычисления - " + e.getMessage());
41                semaphore.endRead();
42            }
43        }
44    }
45}
```

```
package threads;

import functions.basic.Log;

public class Generator extends Thread { 2 usages
    private final Task task; 6 usages
    private final ReadWriteSemaphore semaphore; 3 usages

    public Generator(Task task, ReadWriteSemaphore semaphore) { 5 usages
        this.task = task;
        this.semaphore = semaphore;
    }

    @Override
    public void run() {
        int tasksCount = task.getTasksCount();
        for (int i = 0; i < tasksCount; i++) {
            try {
                //захватываем семафор для записи
                semaphore.startWrite();

                //проверяем прерывание
                if (Thread.interrupted()) {
                    System.out.println("Generator: получен сигнал прерывания");
                    return;
                }

                //генерируем данные
                double base = 1 + Math.random() * 9;
                double left = Math.random() * 100;
                double right = 100 + Math.random() * 100;
                double step = Math.random();

                task.setFunction(new Log(base));
                task.setLeftBorder(left);
                task.setRightBorder(right);
                task.setDiscretizationStep(step);

                System.out.printf("Generator[%d]: Source %.3f %.3f %.3f%n", i, left, right, step);

                //освобождаем семафор
                semaphore.endWrite();
            } catch (InterruptedException e) {
                System.out.println("Generator: прерван во время ожидания");
                return;
            }
        }
    }
}
```

```
public class Generator extends Thread { 2 usages
    public void run() {
        //проверяем прерывание
        if (Thread.interrupted()) {
            System.out.println("Generator: получен сигнал прерывания");
            return;
        }

        //генерируем данные
        double base = 1 + Math.random() * 9;
        double left = Math.random() * 100;
        double right = 100 + Math.random() * 100;
        double step = Math.random();

        task.setFunction(new Log(base));
        task.setLeftBorder(left);
        task.setRightBorder(right);
        task.setDiscretizationStep(step);

        System.out.printf("Generator[%d]: Source %.3f %.3f %.3f%n", i, left, right, step);

        //освобождаем семафор
        semaphore.endWrite();
    }
}
```

создал метод complicatedThreads с реализацией обработки прерываний и протестировал

The screenshot shows two instances of a Java IDE (likely IntelliJ IDEA) running side-by-side. Both windows have the title 'Lab-6-2025 > Main > complicatedThreads'. The left window displays the 'Main.java' code, which contains a method named 'complicatedThreads'. This method creates a task, sets its tasks count to 100, creates a semaphore, starts a generator and integrator, sleeps for 50 ms, and then tries to join both threads. If an interrupt occurs, it prints a message and continues. The right window shows the 'Run' tab with the output of the program. The output lists 99 results from 'Integrator' and 'Generator' objects, each with a unique ID and a result value. It also includes messages about starting threads, interrupting them, and finally stating 'ВСЕ ТЕСТЫ ЗАВЕРШЕНЫ!' (All tests completed!).

```
public class Main {
    private static void complicatedThreads() { // создаем объекты
        threads.Task task = new threads.Task();
        task.setTasksCount(100);

        threads.ReadWriteSemaphore semaphore = new threads.ReadWriteSemaphore();

        // создаем потоки
        threads.Generator generator = new threads.Generator(task, semaphore);
        threads.Integrator integrator = new threads.Integrator(task, semaphore);

        System.out.println("Запускаем потока");
        generator.start();
        integrator.start();

        // ждём 50 мс
        try {
            Thread.sleep(50);
        } catch (InterruptedException e) {
            System.out.println("Основной поток прерван");
        }

        // прерываем потоки
        System.out.println("\nПрерываем потоки");
        generator.interrupt();
        integrator.interrupt();

        try {
            generator.join();
            integrator.join();
        } catch (InterruptedException e) {
            System.out.println("Основной поток прерван");
        }
    }
}
```

```
Integrator[0]: Result 0,019 104,531 0,108
Integrator[1]: Result 36,942 104,531 0,768 230,113673
Integrator[2]: Result 47,205 117,587 0,769
Integrator[3]: Result 47,205 117,587 0,769 227,068177
Integrator[4]: Result 24,185 186,677 0,466
Integrator[5]: Result 24,185 186,677 0,466 381,851764
Integrator[6]: Result 78,388 197,864 0,518
Integrator[7]: Result 78,388 197,864 0,518 2858,952163
Integrator[8]: Result 74,334 174,023 0,427
Integrator[9]: Result 74,334 174,023 0,427 629,011527
Integrator[10]: Result 48,947 149,423 0,375
Integrator[11]: Result 48,947 149,423 0,375 252,646643
Integrator[12]: Result 14,430 198,574 0,763
Integrator[13]: Result 14,430 198,574 0,763 364,728987
Integrator[14]: Result 95,208 109,069 0,536
Integrator[15]: Result 95,208 109,069 0,536 51,427286
Integrator[16]: Result 55,329 181,044 0,898
Integrator[17]: Result 55,329 181,044 0,898 343,873878
Integrator[18]: Result 40,371 195,256 0,601
Integrator[19]: Result 40,371 195,256 0,601 321,586951
Integrator[20]: Result 74,776 152,256 0,815
Integrator[21]: Result 74,776 152,256 0,815 193,695824
Integrator[22]: Result 27,412 199,074 0,448
Integrator[23]: Result 27,412 199,074 0,448 481,702706
Generator[24]: Result 82,000 168,691 0,668
Generator[25]: Result 82,000 168,691 0,668 229,623528
Generator[26]: Result 26,738 137,287 0,294
Generator[27]: Result 26,738 137,287 0,294 377,935866
Проверяю потоки
*** ВСЕ ТЕСТЫ ЗАВЕРШЕНЫ! ***
Process finished with exit code 0
```