Отчёт по лабораторной работе №6

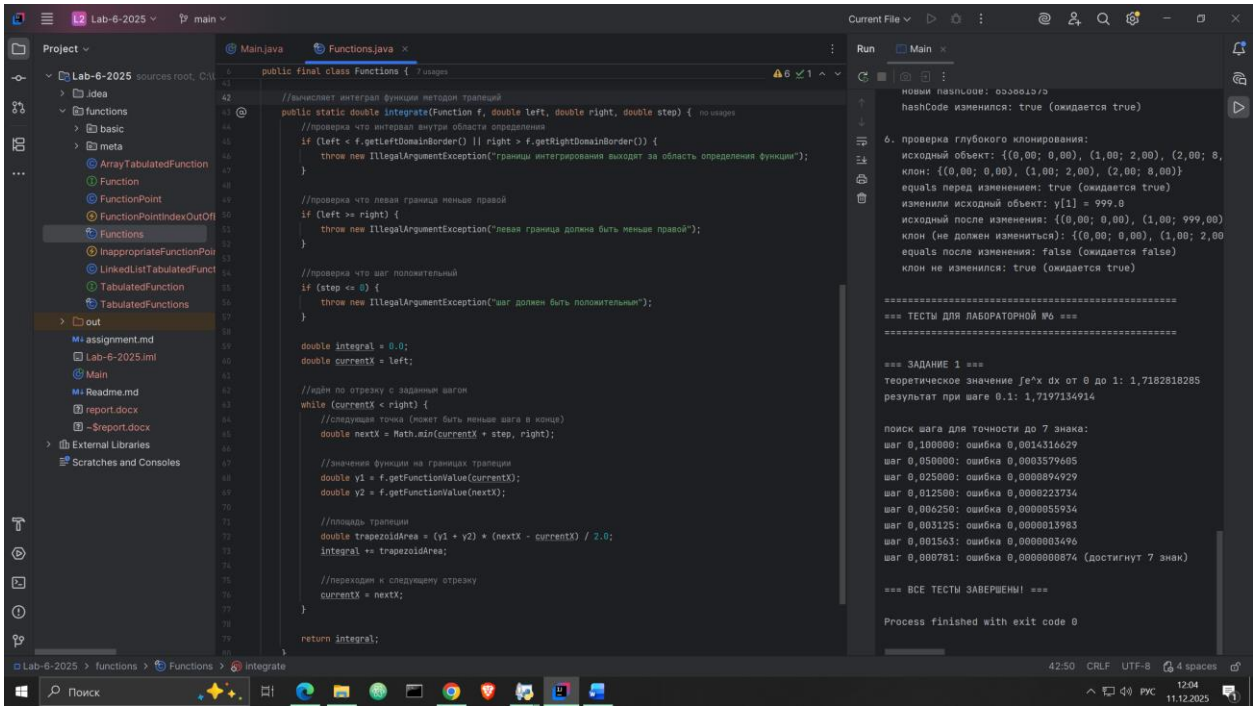
Тенигин Валерий 6204-010302D

Задание 1

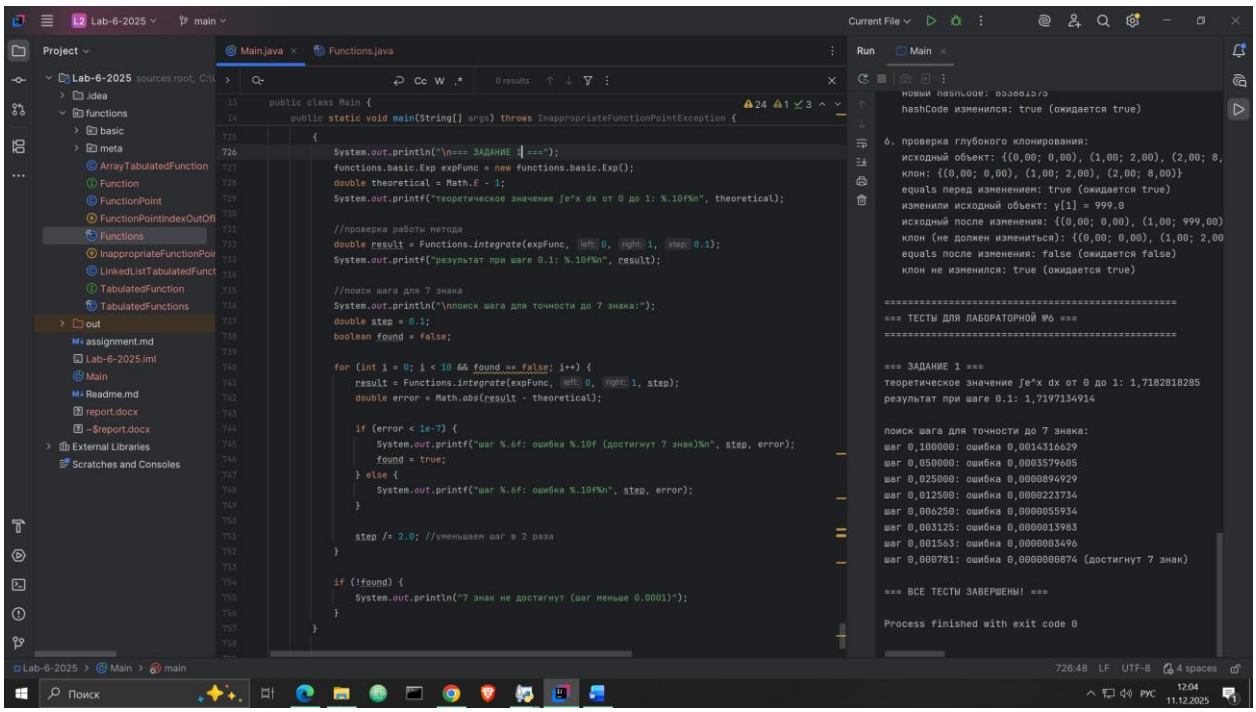
Добавил:

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

проверки - область определения, корректность границ, положительность шага

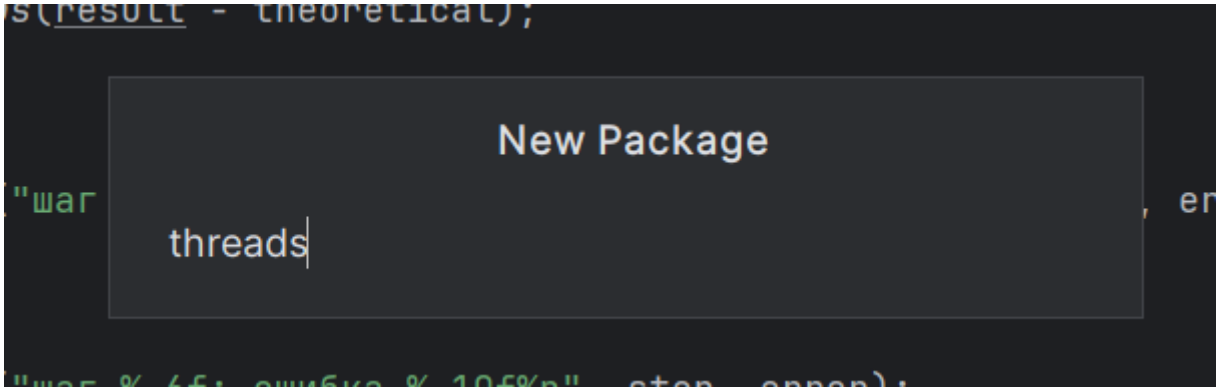


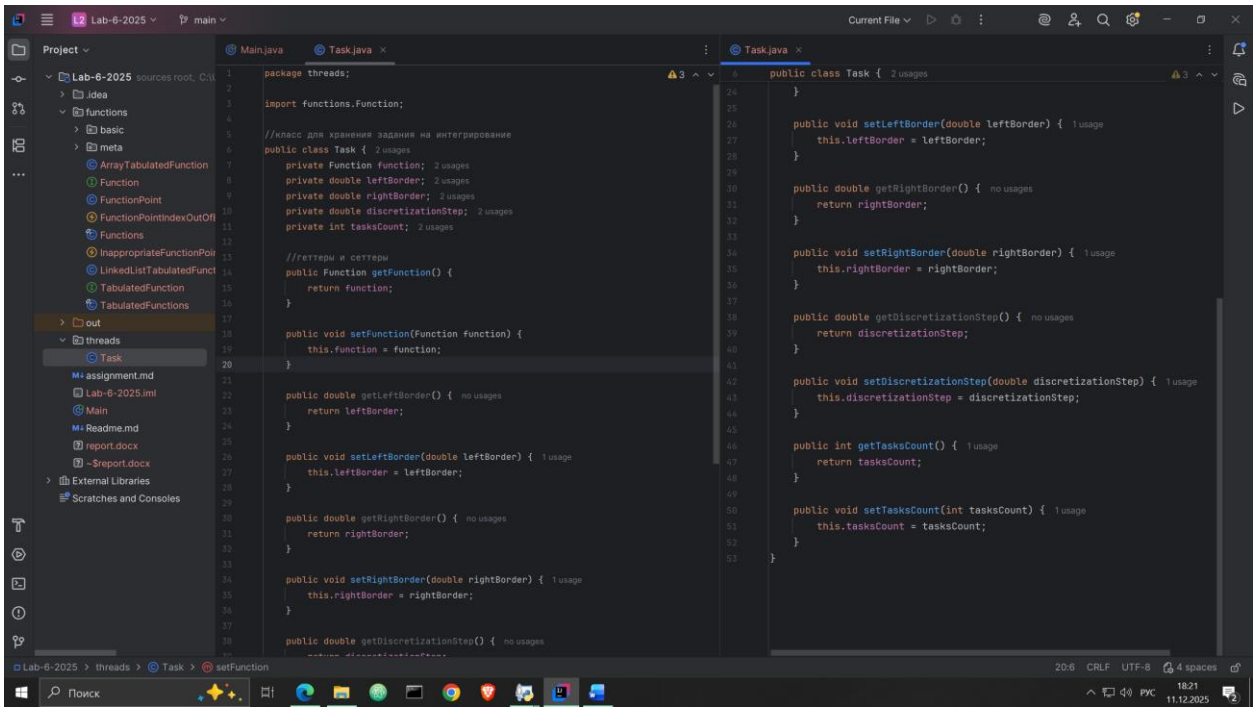
Протестировал в Main



Задание 2

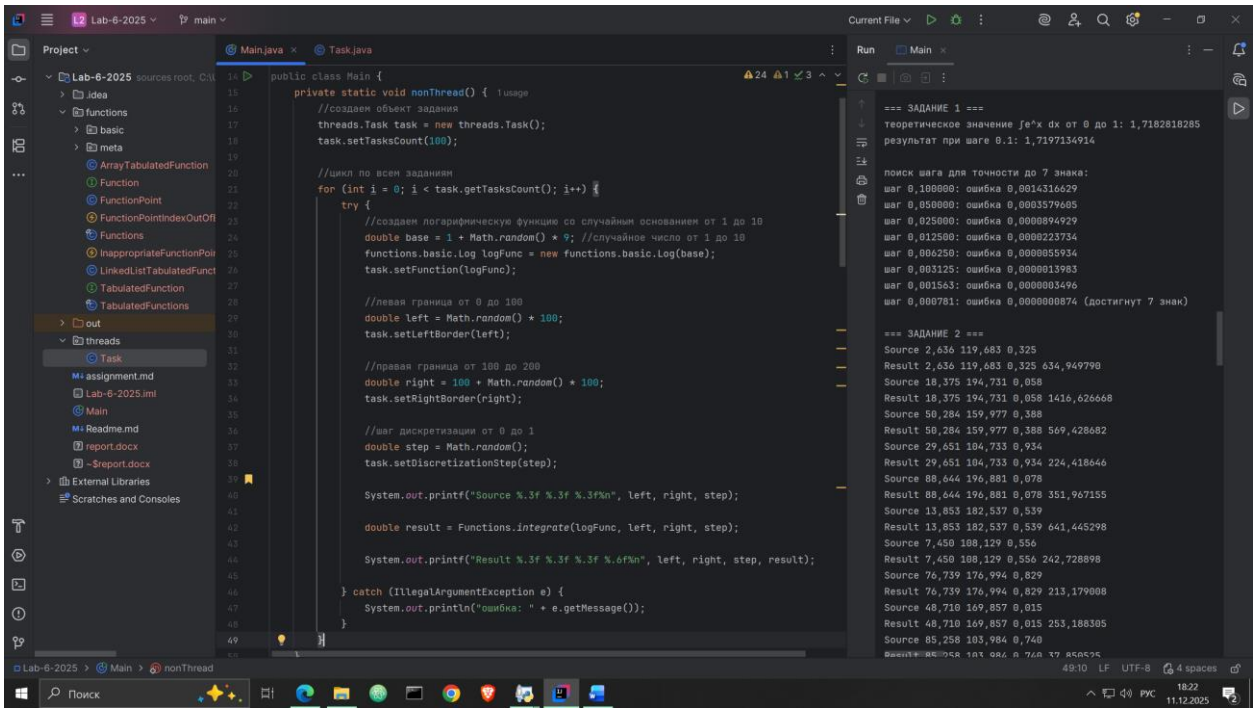
Создал класс Task для хранения задания





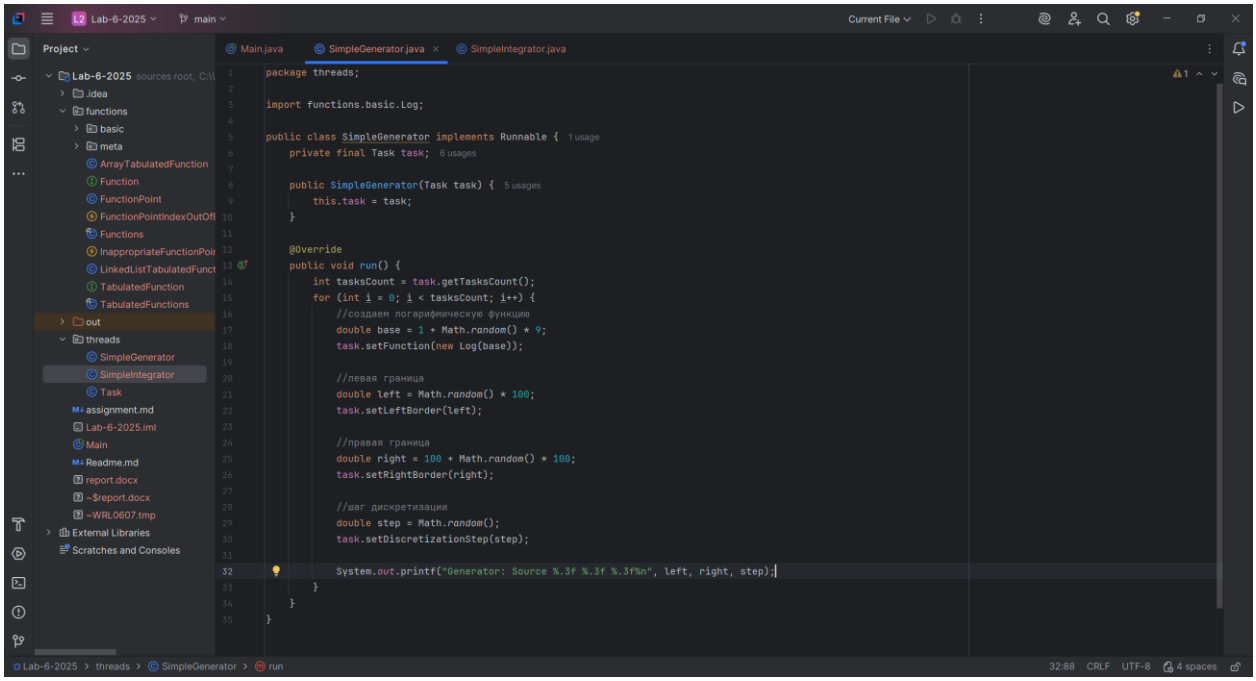
```java
//Задание 2
{
    System.out.println("\n=== ЗАДАНИЕ 2 ===");
    nonThread();
}
```

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

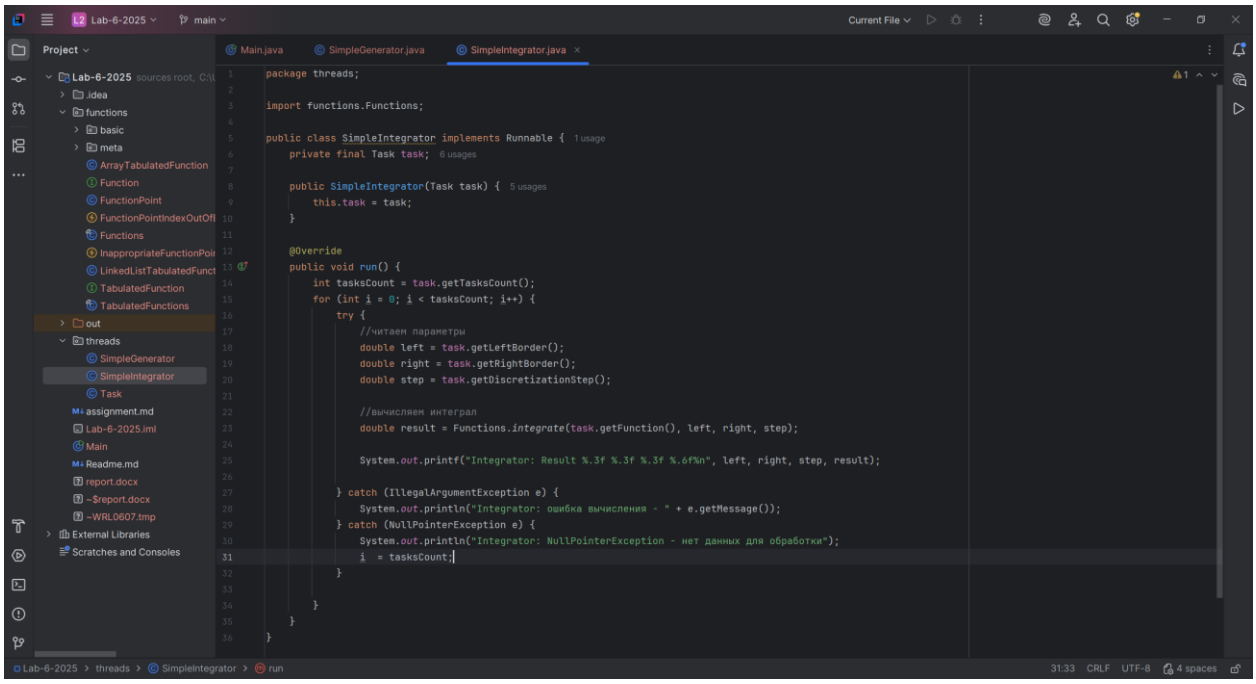


Задание 3

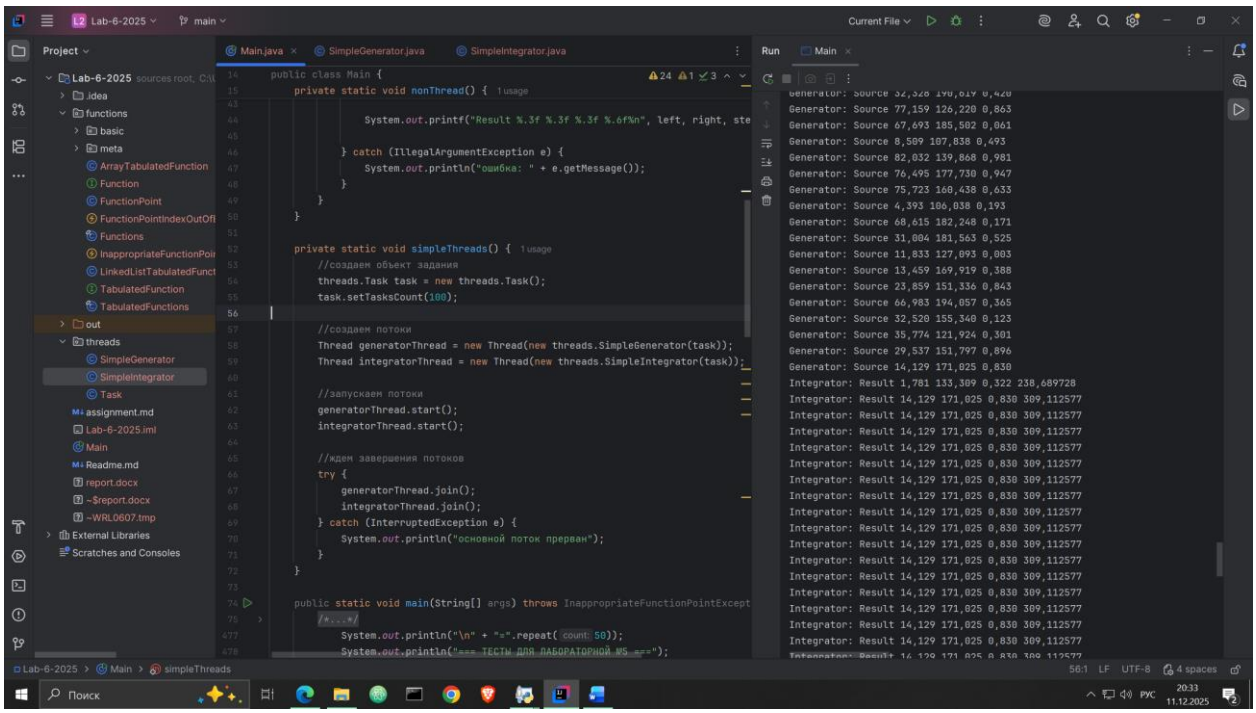
Создал два класса: SimpleGenerator (генерирует задания) и SimpleIntegrator (вычисляет интегралы)



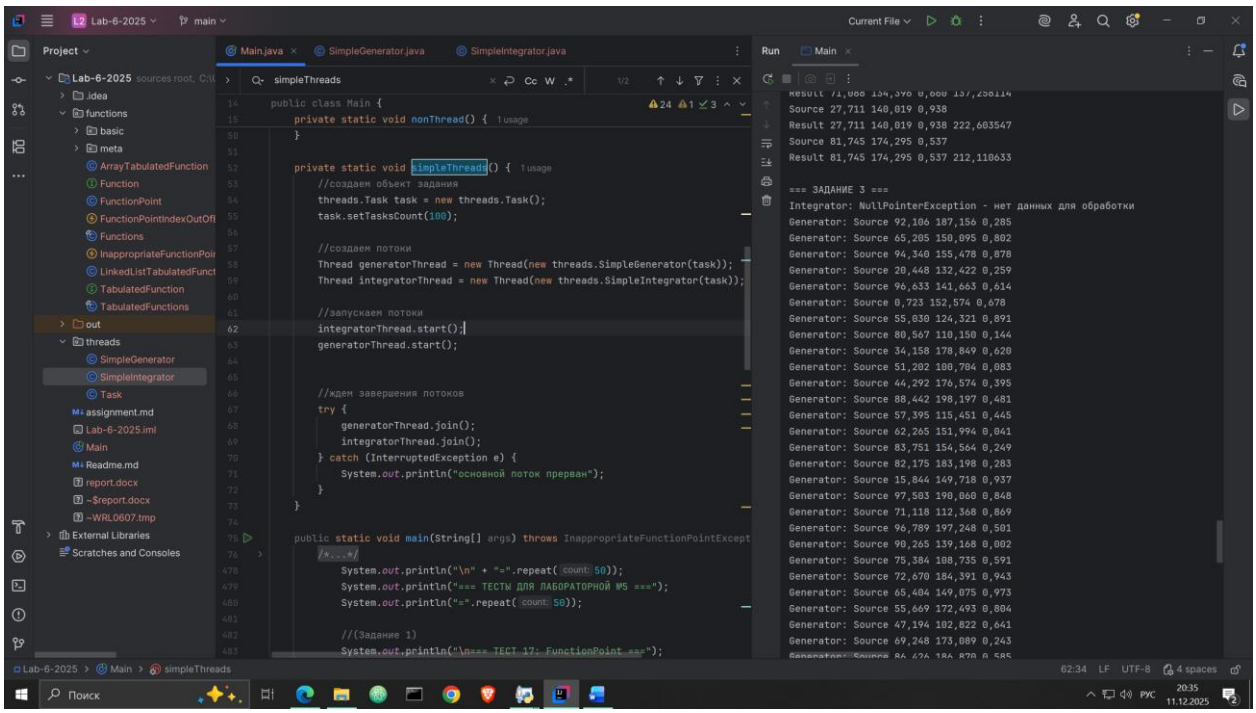
```java
package threads;

import functions.basic.Log;

public class SimpleGenerator implements Runnable {
    private final Task task;

    public SimpleGenerator(Task task) {
        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);
        }
    }
}
```



```java
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() {
        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 %.3f %.6f%n", left, right, step, 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;
            }

        }
    }
}
```

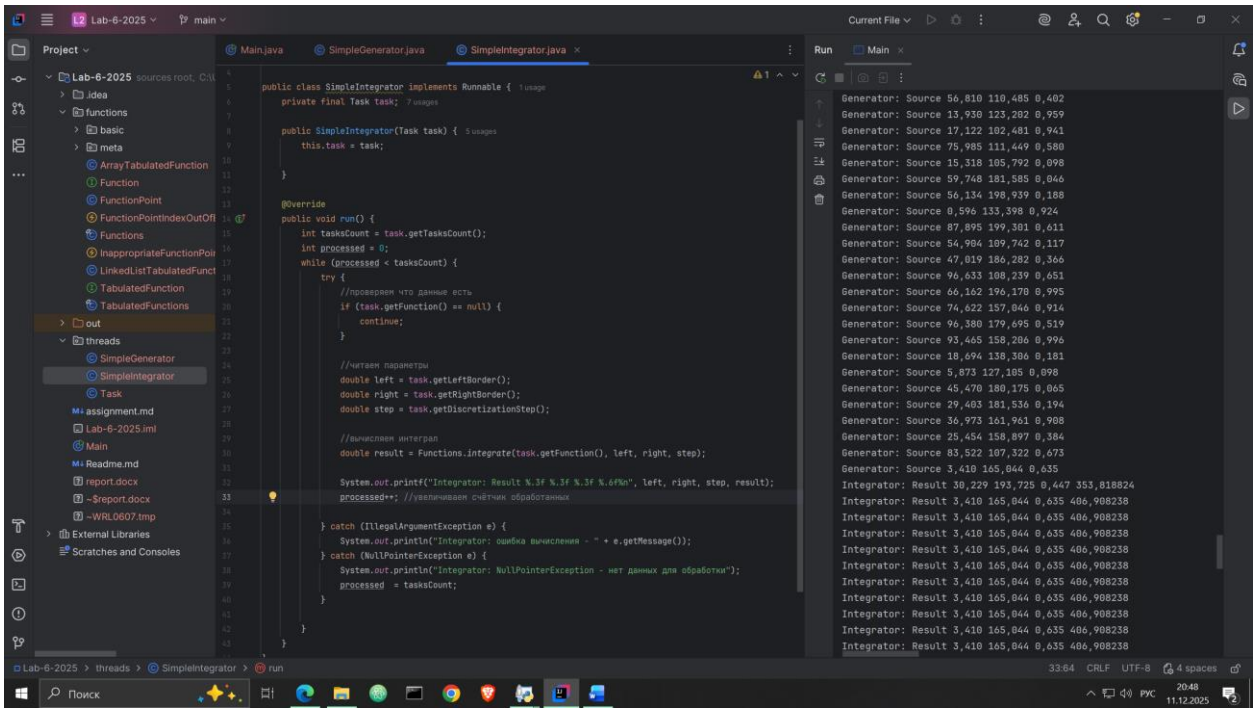


```
Generator: Source 77,159 126,220 0,863
Generator: Source 67,693 185,502 0,061
Generator: Source 8,509 107,838 0,493
Generator: Source 82,032 139,868 0,981
Generator: Source 76,495 177,730 0,947
Generator: Source 75,723 160,438 0,633
Generator: Source 4,393 106,038 0,193
Generator: Source 68,615 182,248 0,171
Generator: Source 31,004 181,563 0,525
Generator: Source 11,833 127,093 0,003
Generator: Source 13,459 169,919 0,388
Generator: Source 23,859 151,336 0,843
Generator: Source 66,983 194,057 0,365
Generator: Source 32,520 155,340 0,123
Generator: Source 35,774 121,924 0,301
Generator: Source 29,537 151,797 0,896
Generator: Source 14,129 171,025 0,830
Integrator: Result 1,781 133,309 0,322 238,689728
Integrator: Result 14,129 171,025 0,830 309,112577
Integrator: Result 14,129 171,025 0,830 309,112577
Integrator: Result 14,129 171,025 0,830 309,112577
Integrator: Result 14,129 171,025 0,830 309,112577
Integrator: Result 14,129 171,025 0,830 309,112577
Integrator: Result 14,129 171,025 0,830 309,112577
Integrator: Result 14,129 171,025 0,830 309,112577
Integrator: Result 14,129 171,025 0,830 309,112577
Integrator: Result 14,129 171,025 0,830 309,112577
Integrator: Result 14,129 171,025 0,830 309,112577
Integrator: Result 14,129 171,025 0,830 309,112577
Integrator: Result 14,129 171,025 0,830 309,112577
Integrator: Result 14,129 171,025 0,830 309,112577
```

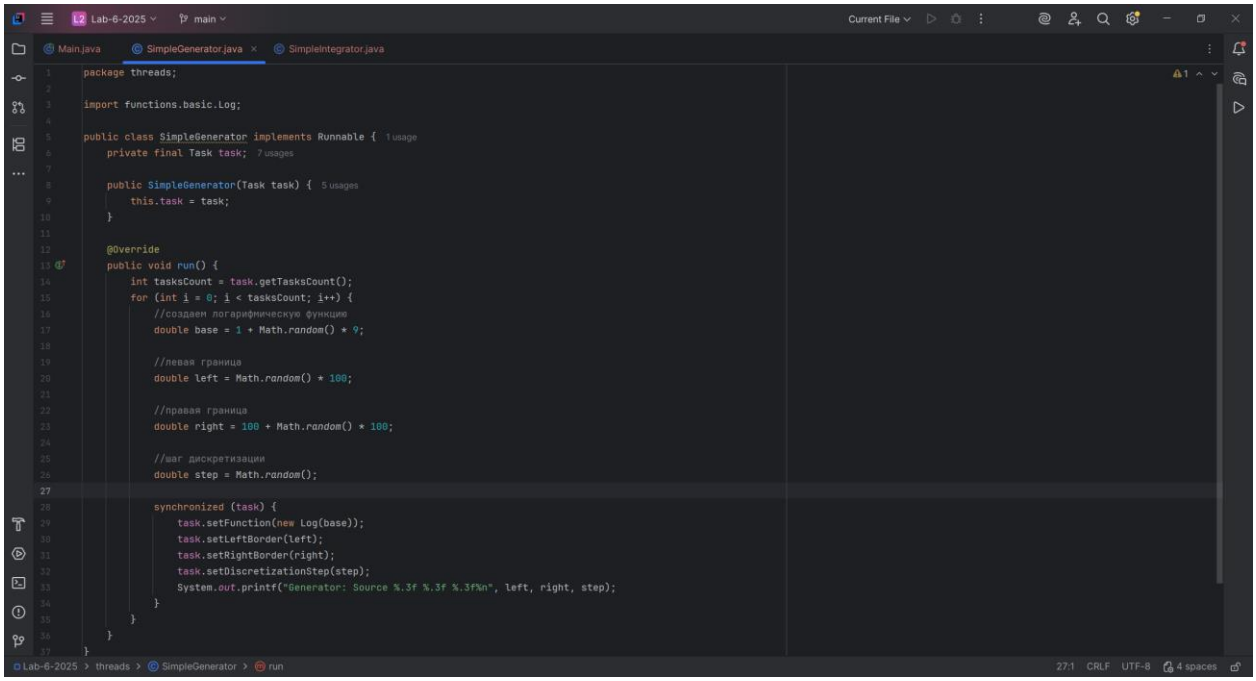


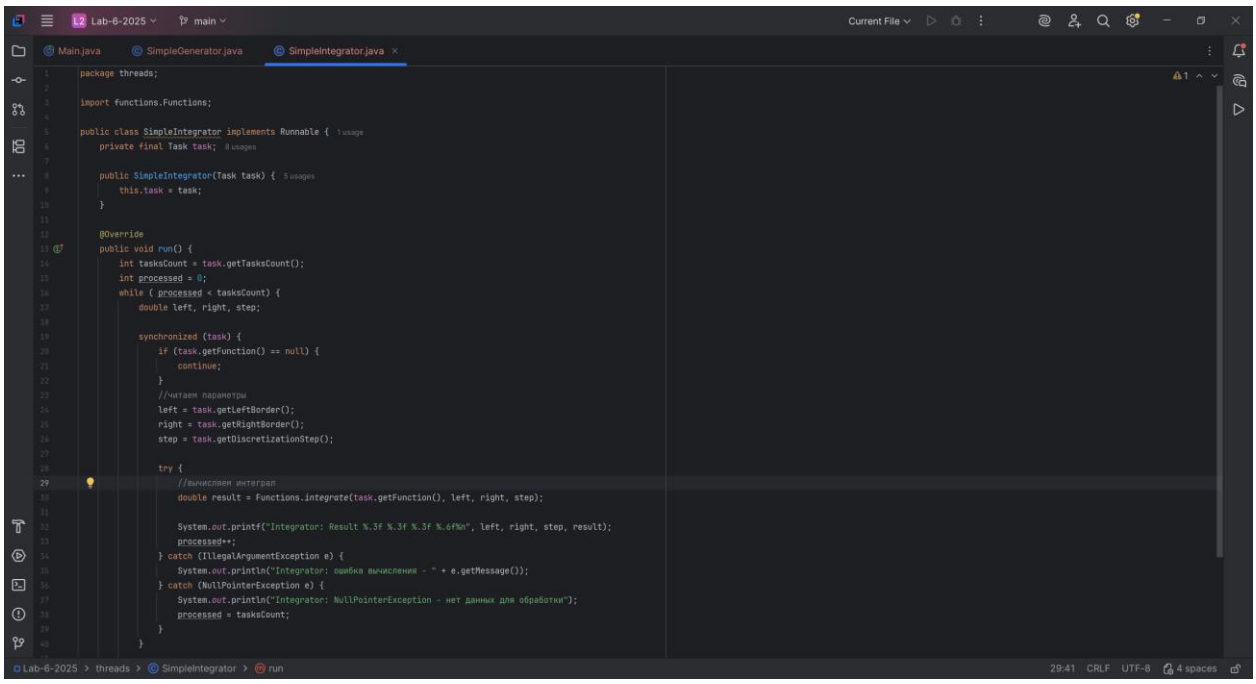
```
Source 27,711 140,019 0,938
Result 27,711 140,019 0,938 222,603547
Source 81,745 174,295 0,537
Result 81,745 174,295 0,537 212,110633

=== ЗАДАНИЕ 3 ===
Integrator: NullPointerException - нет данных для обработки
Generator: Source 92,106 187,156 0,285
Generator: Source 65,205 150,095 0,802
Generator: Source 94,340 155,478 0,878
Generator: Source 20,448 132,422 0,259
Generator: Source 96,633 141,663 0,614
Generator: Source 8,723 152,574 0,678
Generator: Source 55,830 124,321 0,891
Generator: Source 80,567 110,150 0,144
Generator: Source 34,158 178,849 0,620
Generator: Source 51,202 100,784 0,083
Generator: Source 44,292 176,574 0,395
Generator: Source 88,442 198,197 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 15,844 149,718 0,937
Generator: Source 97,503 190,860 0,848
Generator: Source 71,118 112,368 0,869
Generator: Source 96,789 197,248 0,501
Generator: Source 90,265 139,168 0,002
Generator: Source 75,384 108,735 0,591
Generator: Source 72,670 184,391 0,943
Generator: Source 65,404 149,075 0,973
Generator: Source 55,669 172,493 0,804
Generator: Source 47,194 102,822 0,641
Generator: Source 69,248 173,089 0,243
```

Исправил ошибку NullPointerException без синхронизации, проверяя что данные присутствуют



Добавил синхронизацию

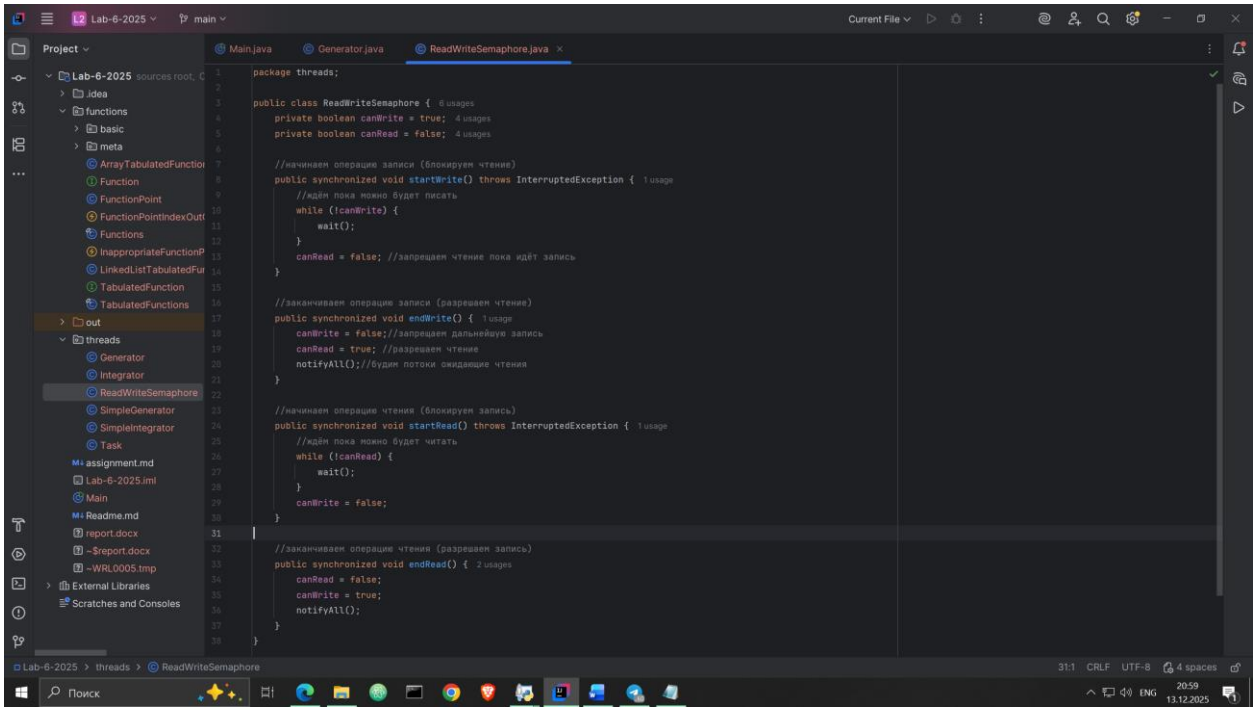




```java
package threads;

import functions.Functions;

public class SimpleIntegrator implements Runnable {
    private final Task task;

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

    @Override
    public void run() {
        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 интеграл
                    double result = Functions.integrate(task.getFunction(), left, right, step);

                    System.out.printf("Integrator: Result %.3f %.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;
                }
            }
```

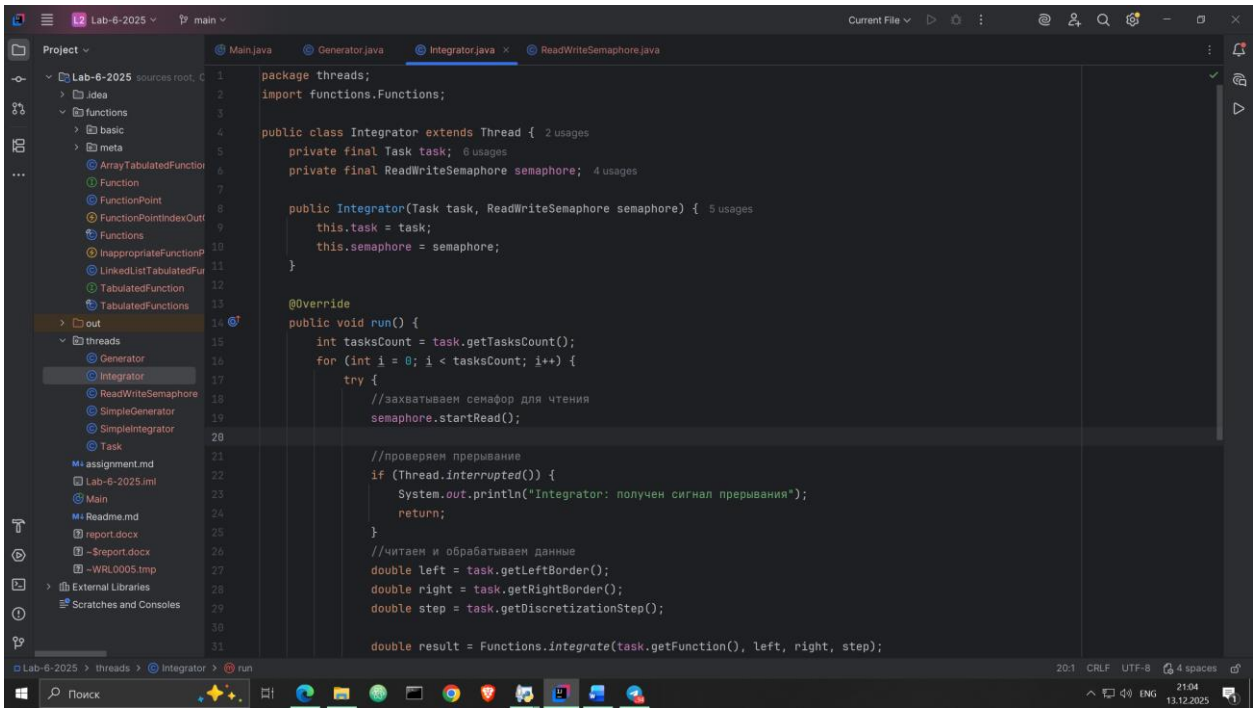
Задание 4

Сделал семафор ReadWriteSemaphore для управления доступом к данным

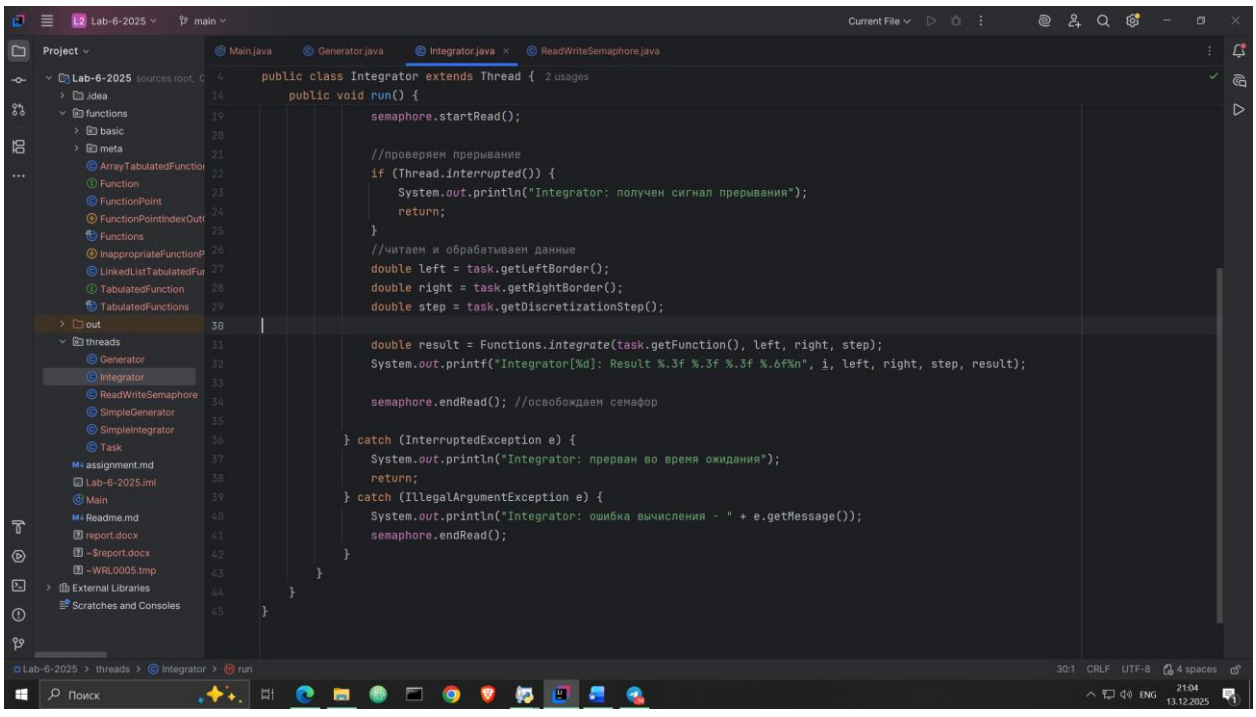


```java
package threads;

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

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

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

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

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

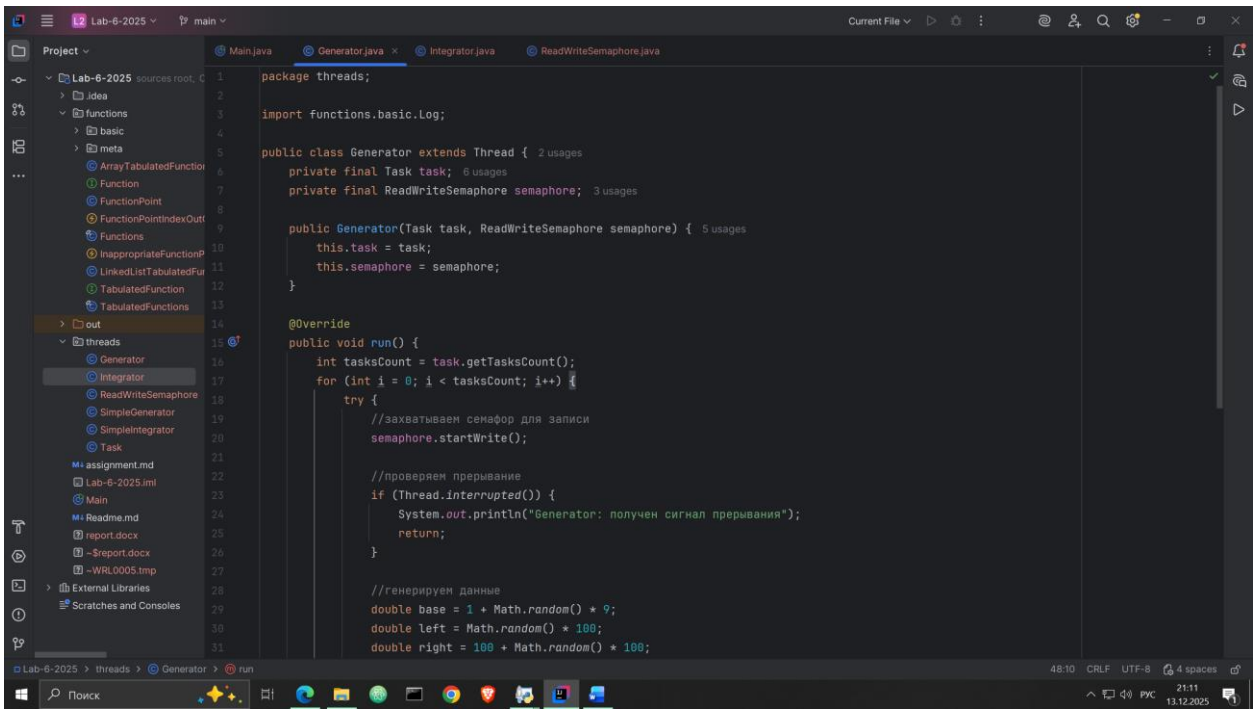
Создал классы Generator и Integrator, которые наследую от Thread



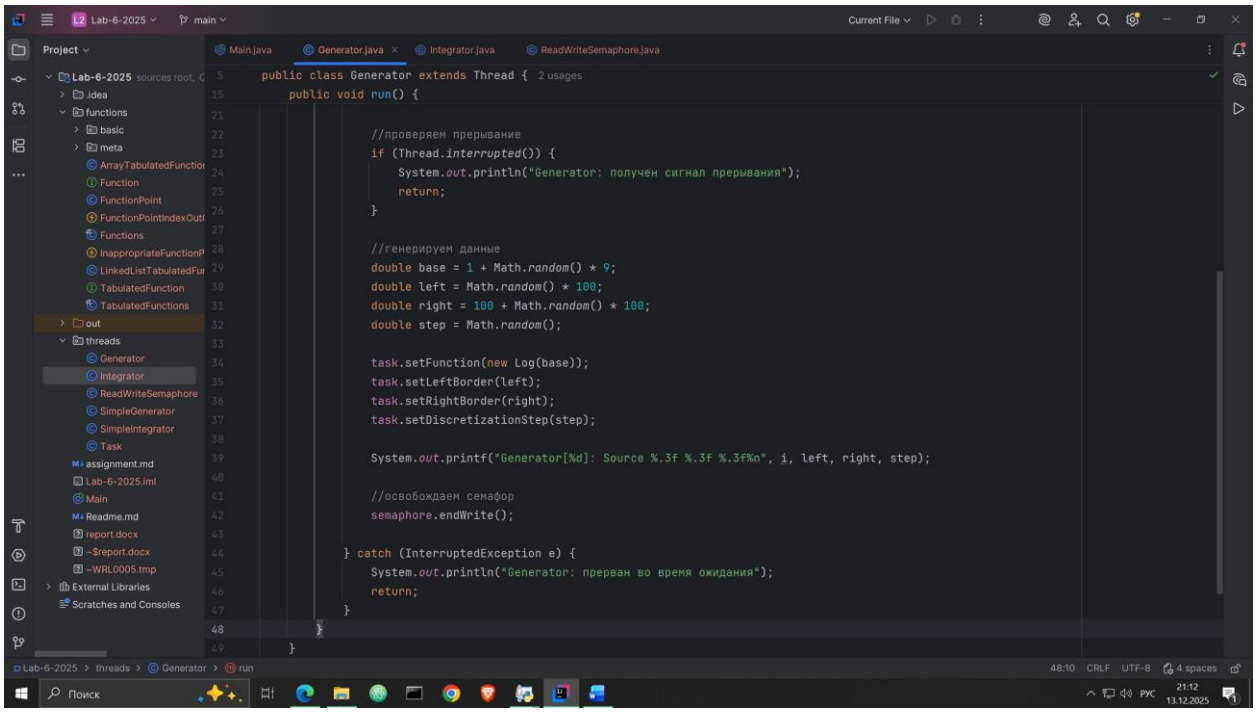
```java
package threads;
import functions.Functions;

public class Integrator extends Thread { 2 usages
    private final Task task; 6 usages
    private final ReadWriteSemaphore semaphore; 4 usages

    public Integrator(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 чтения
                semaphore.startRead();

                //проверяем прерывание
                if (Thread.interrupted()) {
                    System.out.println("Integrator: получен сигнал прерывания");
                    return;
                }
                //читаем и обрабатываем данные
                double left = task.getLeftBorder();
                double right = task.getRightBorder();
                double step = task.getDiscretizationStep();

                double result = Functions.integrate(task.getFunction(), left, right, step);
```



```java
public class Integrator extends Thread { 2 usages
    public void run() {
                semaphore.startRead();

                //проверяем прерывание
                if (Thread.interrupted()) {
                    System.out.println("Integrator: получен сигнал прерывания");
                    return;
                }
                //читаем и обрабатываем данные
                double left = task.getLeftBorder();
                double right = task.getRightBorder();
                double step = task.getDiscretizationStep();

                double result = Functions.integrate(task.getFunction(), left, right, step);
                System.out.printf("Integrator[%d]: Result %.3f %.3f %.3f %.6f%n", i, left, right, step, result);

                semaphore.endRead(); //освобождаем семафор

            } catch (InterruptedException e) {
                System.out.println("Integrator: прерван во время ожидания");
                return;
            } catch (IllegalArgumentException e) {
                System.out.println("Integrator: ошибка вычисления - " + e.getMessage());
                semaphore.endRead();
            }
        }
    }
}
```



```java
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;
```



```java
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();

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

создал метод complicatedThreads с реализацией обработки прерываний и протестировад