

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

Задание 1

Добавил:

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

проверки - область определения, корректность границ, положительность шага

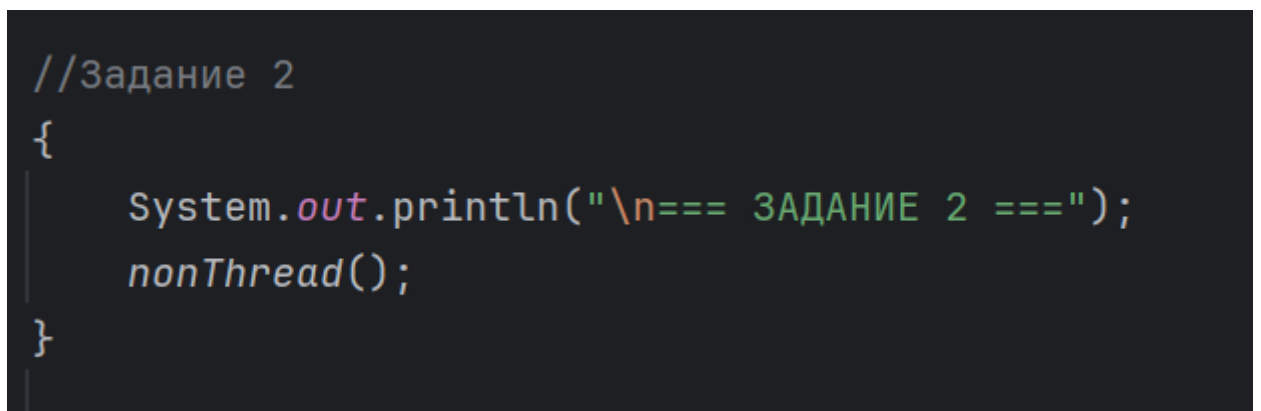
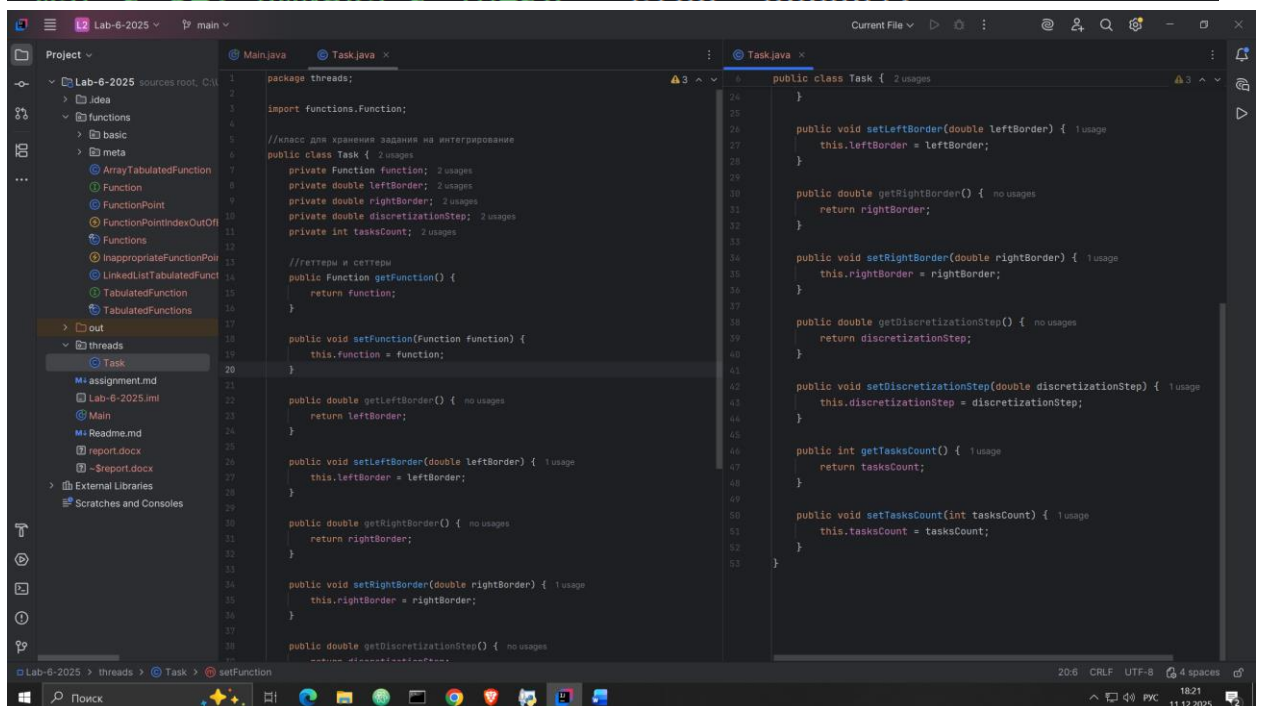
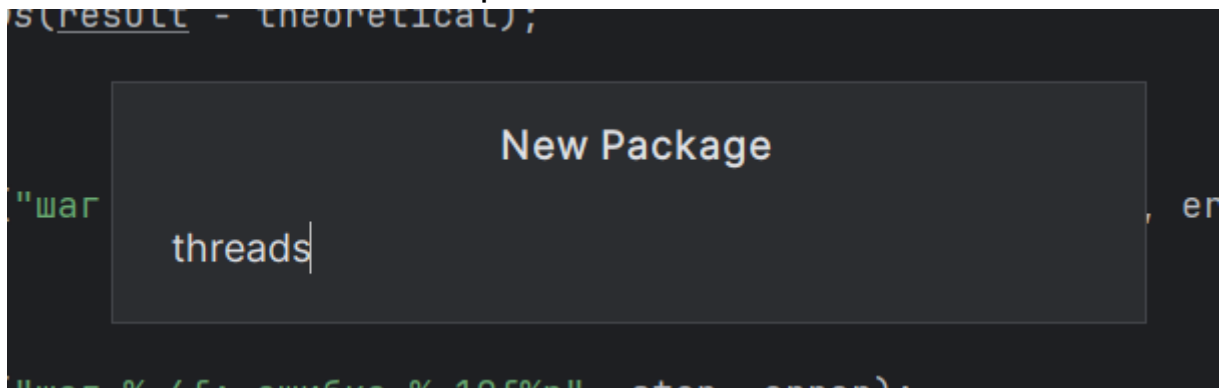
```
public final class Functions {  
    //вычисляет интеграл функции методом трапеций  
    public static double integrate(Function f, double left, double right, double step) {  
        //проверка что интеграл внутри области определения  
        if (left < f.getLeftDomainBorder() || right > f.getRightDomainBorder()) {  
            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;  
  
        //идем 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;  
    }  
}
```

Протестировал в Main

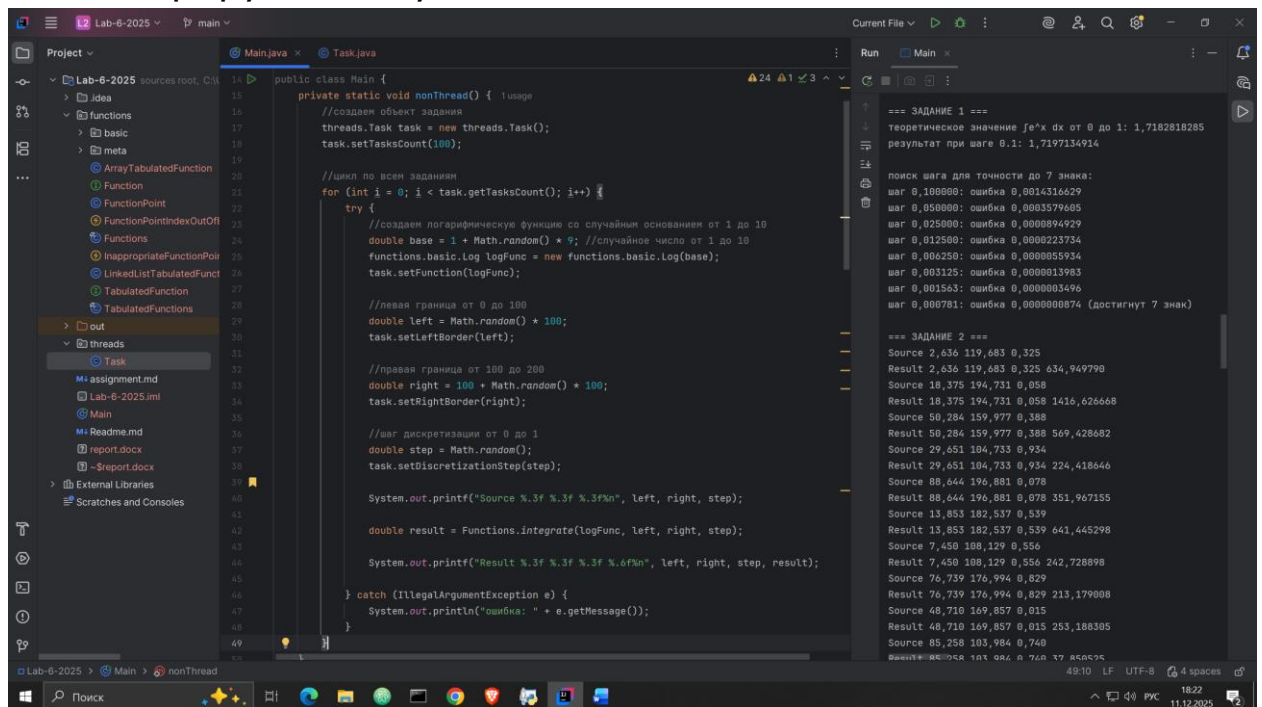
```
public class Main {  
    public static void main(String[] args) throws InappropriateFunctionPointException {  
        System.out.println("\n=== ЗАДАНИЕ 1 ===");  
        functions.basic.Exp expFunc = new functions.basic.Exp();  
        double theoretical = Math.E - 1;  
        System.out.println("теоретическое значение  $e^x - 1$  от 0 до 1: %.10f", theoretical);  
  
        //проверка работы метода  
        double result = Functions.integrate(expFunc, 0, 1, 0.1);  
        System.out.println("результат при шаге 0.1: %.10f", result);  
  
        //поиск шага для 7 знака  
        System.out.println("\nпоиск 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(expFunc, 0, 1, step);  
            double error = Math.abs(result - theoretical);  
  
            if (error < 1e-7) {  
                System.out.println("шаг %.6f: ошибка %.10f (достигнут 7 знаков)", step, error);  
                found = true;  
            } else {  
                System.out.println("шаг %.6f: ошибка %.10f", step, error);  
            }  
  
            step /= 2.0; //уменьшаем шаг в 2 раза  
        }  
  
        if (!found) {  
            System.out.println("7 знак не достигнут (шаг меньше 0.0001)");  
        }  
    }  
}
```

Задание 2

Создал класс Task для хранения задания



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



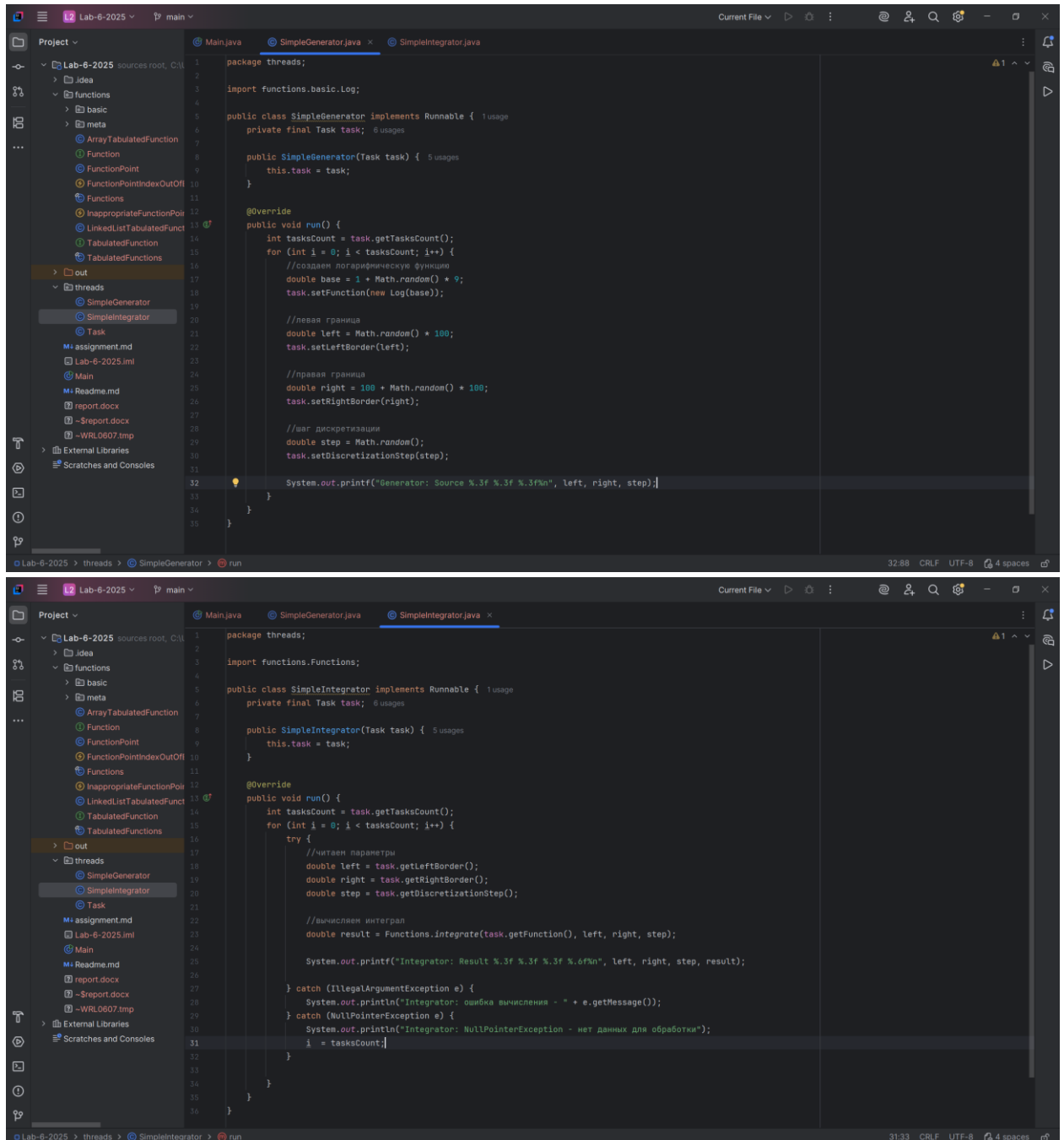
```
public class Main {  
    private static void nonThread() {  
        //создаем объект задания  
        threads.Task task = new threads.Task();  
        task.setTaskCount(100);  
  
        //цикл по всем заданиям  
        for (int i = 0; i < task.getTaskCount(); i++) {  
            try {  
                //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\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());  
            }  
        }  
    }  
}
```

Run Main

```
=== ЗАДАНИЕ 1 ===  
теоретическое значение  $\int_{0.1}^{1.7182818285} e^x dx$  от 0 до 1: 1,7182818285  
результат при шаге 0.1: 1,7197134914  
  
поиск шага для точности до 7 знака:  
шаг 0,100000: ошибка 0,0014316429  
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 знак)  
  
=== ЗАДАНИЕ 2 ===  
Source 2,636 119,683 0,325  
Result 2,636 119,683 0,325 634,949790  
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 0,539  
Result 13,853 182,537 0,539 641,445298  
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
```

Задание 3

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



The image displays two screenshots of an IDE (IntelliJ IDEA) showing the implementation of two Java classes: `SimpleGenerator` and `SimpleIntegrator`.

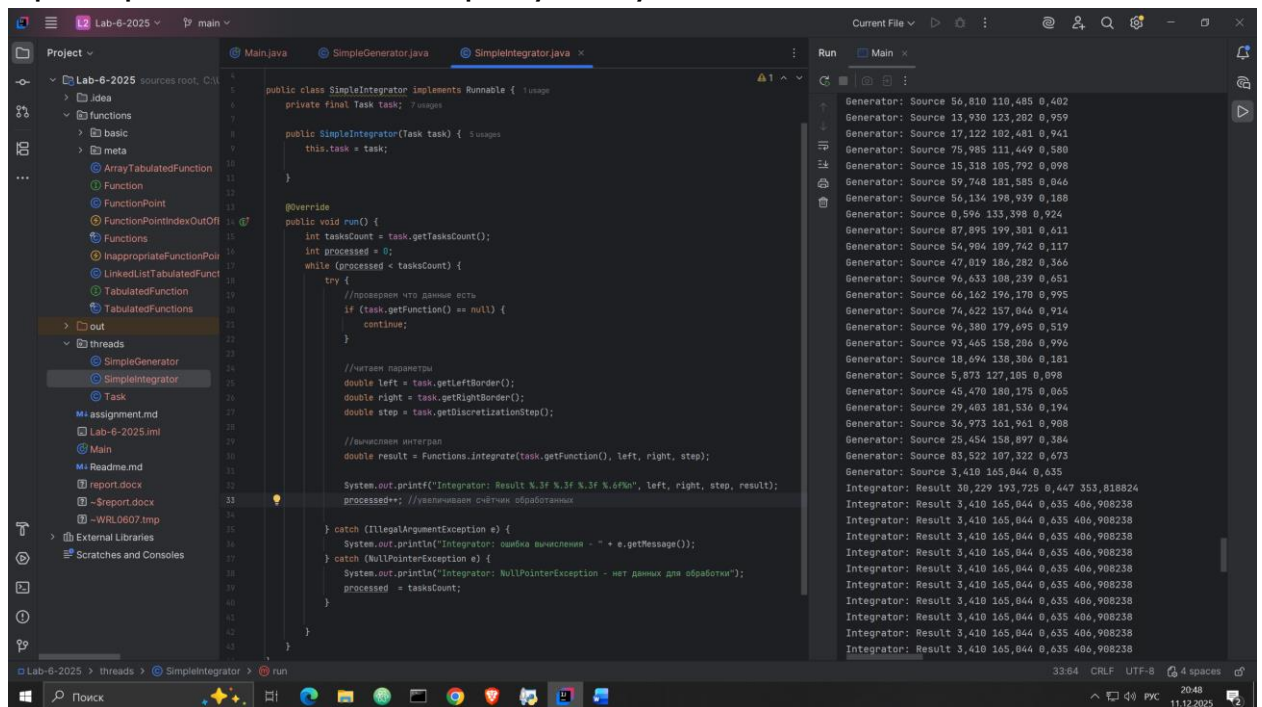
Top Screenshot: `SimpleGenerator.java`

```
1 package threads;
2
3 import functions.basic.Log;
4
5 public class SimpleGenerator implements Runnable { 1 usage
6     private final Task task; 6 usages
7
8     public SimpleGenerator(Task task) { 5 usages
9         this.task = task;
10    }
11
12    @Override
13    public void run() {
14        int tasksCount = task.getTasksCount();
15        for (int i = 0; i < tasksCount; i++) {
16            //создаем логарифмическую функцию
17            double base = 1 + Math.random() * 9;
18            task.setFunction(new Log(base));
19
20            //левая граница
21            double left = Math.random() * 100;
22            task.setLeftBorder(left);
23
24            //правая граница
25            double right = 100 + Math.random() * 100;
26            task.setRightBorder(right);
27
28            //шаг дискретизации
29            double step = Math.random();
30            task.setDiscretizationStep(step);
31
32            System.out.printf("Generator: Source %.3f %.3f %.3f\n", left, right, step);
33        }
34    }
35 }
```

Bottom Screenshot: `SimpleIntegrator.java`

```
1 package threads;
2
3 import functions.Functions;
4
5 public class SimpleIntegrator implements Runnable { 1 usage
6     private final Task task; 6 usages
7
8     public SimpleIntegrator(Task task) { 5 usages
9         this.task = task;
10    }
11
12    @Override
13    public void run() {
14        int tasksCount = task.getTasksCount();
15        for (int i = 0; i < tasksCount; i++) {
16            try {
17                //читаем параметры
18                double left = task.getLeftBorder();
19                double right = task.getRightBorder();
20                double step = task.getDiscretizationStep();
21
22                //вычисляем интеграл
23                double result = Functions.integrate(task.getFunction(), left, right, step);
24
25                System.out.printf("Integrator: Result %.3f %.3f %.3f %.6f\n", left, right, step, result);
26
27            } catch (IllegalArgumentException e) {
28                System.out.println("Integrator: ошибка вычисления - " + e.getMessage());
29            } catch (NullPointerException e) {
30                System.out.println("Integrator: NullPointerException - нет данных для обработки");
31                i = tasksCount;
32            }
33        }
34    }
35 }
```


Исправил ошибку NullPointerException без синхронизации, проверяя что данные присутствуют



```
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) {
            try {
                //проверяем что данные есть
                if (task.getFunction() == null) {
                    continue;
                }

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

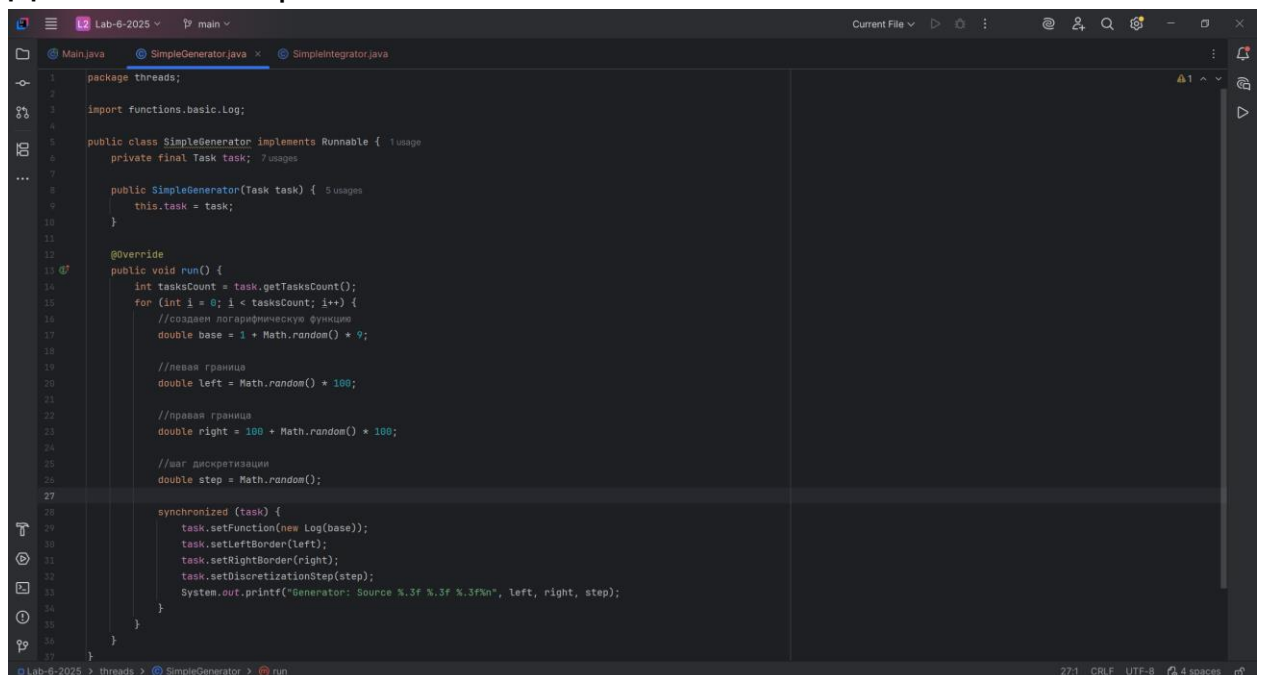
                //вычисляем интеграл
                double result = Functions.integrate(task.getFunction(), left, right, step);

                System.out.printf("Integrator: Result %.3f %.3f %.6f\n", 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;
            }
        }
    }
}
```

Run console output:

```
Generator: Source 56,810 110,485 0,402
Generator: Source 13,930 123,282 0,959
Generator: Source 17,122 182,481 0,941
Generator: Source 75,985 111,449 0,580
Generator: Source 15,318 185,792 0,098
Generator: Source 59,748 181,585 0,044
Generator: Source 56,134 198,939 0,188
Generator: Source 0,596 133,938 0,924
Generator: Source 87,895 199,301 0,611
Generator: Source 54,984 189,742 0,117
Generator: Source 47,019 186,282 0,366
Generator: Source 96,633 188,239 0,651
Generator: Source 66,162 196,178 0,995
Generator: Source 74,622 157,846 0,914
Generator: Source 96,380 179,495 0,519
Generator: Source 93,465 158,286 0,996
Generator: Source 18,694 138,386 0,181
Generator: Source 5,873 127,185 0,098
Generator: Source 45,470 180,175 0,065
Generator: Source 29,403 181,536 0,194
Generator: Source 36,973 161,961 0,908
Generator: Source 25,454 158,897 0,384
Generator: Source 83,522 187,322 0,673
Generator: Source 3,410 165,844 0,635
Integrator: Result 30,229 193,725 0,447 353,818824
Integrator: Result 3,410 165,844 0,635 406,908238
Integrator: Result 3,410 165,844 0,635 406,908238
Integrator: Result 3,410 165,844 0,635 406,908238
Integrator: Result 3,410 165,844 0,635 406,908238
Integrator: Result 3,410 165,844 0,635 406,908238
Integrator: Result 3,410 165,844 0,635 406,908238
Integrator: Result 3,410 165,844 0,635 406,908238
Integrator: Result 3,410 165,844 0,635 406,908238
Integrator: Result 3,410 165,844 0,635 406,908238
```

Добавил синхронизацию



```
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;

            //левая 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 %.3f\n", left, right, step);
            }
        }
    }
}
```

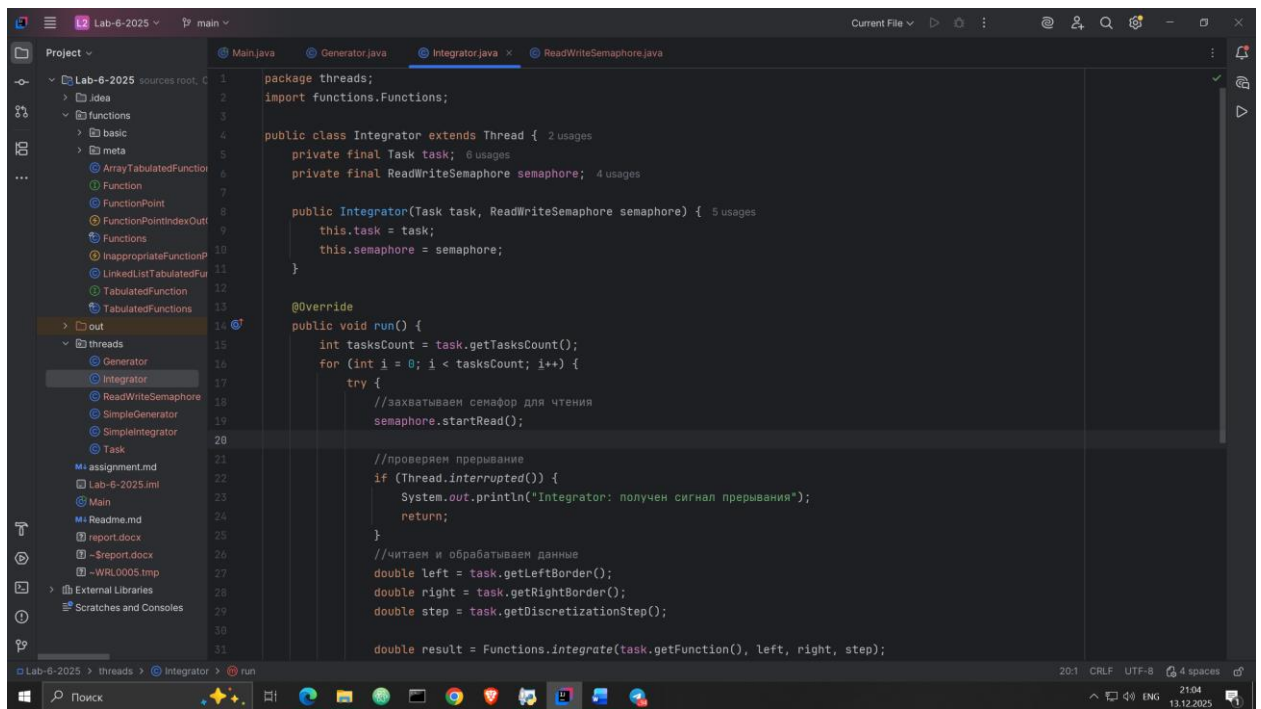
```
1 package threads;
2
3 import functions.Functions;
4
5 public class SimpleIntegrator implements Runnable { 1 usage
6     private final Task task; 8 usages
7
8     public SimpleIntegrator(Task task) { 5 usages
9         this.task = task;
10    }
11
12    @Override
13    public void run() {
14        int tasksCount = task.getTasksCount();
15        int processed = 0;
16        while (processed < tasksCount) {
17            double left, right, step;
18
19            synchronized (task) {
20                if (task.getFunction() == null) {
21                    continue;
22                }
23                //читаем параметры
24                left = task.getLeftBorder();
25                right = task.getRightBorder();
26                step = task.getDiscretizationStep();
27
28                try {
29                    //начинаем интегрировать
30                    double result = Functions.integrate(task.getFunction(), left, right, step);
31
32                    System.out.printf("Integrator: Result %.3f %.3f %.3f %.6f\n", left, right, step, result);
33                    processed++;
34                } catch (IllegalArgumentException e) {
35                    System.out.println("Integrator: ошибка вычисления - " + e.getMessage());
36                } catch (NullPointerException e) {
37                    System.out.println("Integrator: NullPointerException - нет данных для обработки");
38                }
39                processed = tasksCount;
40            }
41        }
42    }
43}
```

Задание 4

Сделал семафор ReadWriteSemaphore для управления доступом к данным

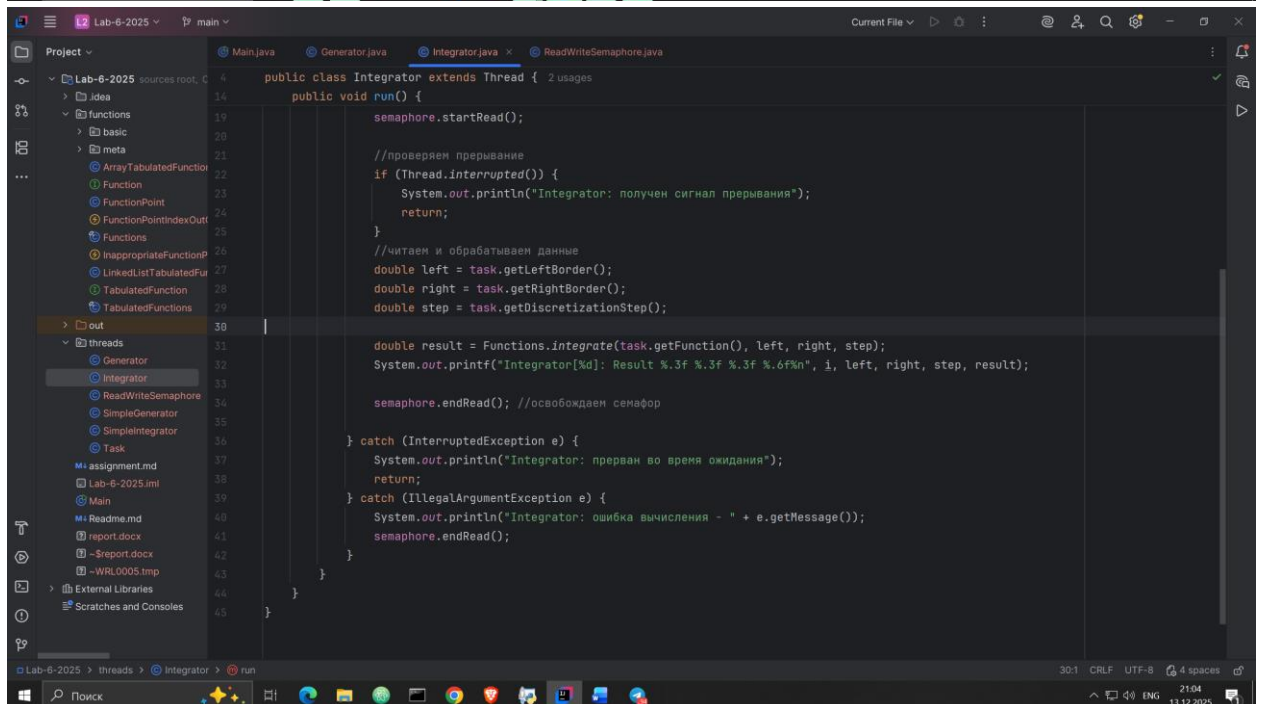
```
1 package threads;
2
3 public class ReadWriteSemaphore { 6 usages
4     private boolean canWrite = true; 4 usages
5     private boolean canRead = false; 4 usages
6
7     //начинаем операцию записи (блокируем чтение)
8     public synchronized void startWrite() throws InterruptedException { 1 usage
9         //ждем пока можно будет писать
10        while (!canWrite) {
11            wait();
12        }
13        canRead = false; //запрещаем чтение пока идет запись
14    }
15
16    //заканчиваем операцию записи (разрешаем чтение)
17    public synchronized void endWrite() { 1 usage
18        canWrite = false; //запрещаем дальнейшую запись
19        canRead = true; //разрешаем чтение
20        notifyAll(); //будит потоки ожидающие чтения
21    }
22
23    //начинаем операцию чтения (блокируем запись)
24    public synchronized void startRead() throws InterruptedException { 1 usage
25        //ждем пока можно будет читать
26        while (!canRead) {
27            wait();
28        }
29        canWrite = false;
30    }
31
32    //заканчиваем операцию чтения (разрешаем запись)
33    public synchronized void endRead() { 2 usages
34        canRead = false;
35        canWrite = true;
36        notifyAll();
37    }
38}
```


Создал классы Generator и Integrator, которые наследуют от Thread



The screenshot shows the IntelliJ IDEA IDE with the 'Integrator.java' file open. The code defines a package 'threads', imports 'functions.Functions', and declares a 'public class Integrator extends Thread'. It has two private final fields: 'Task task' and 'ReadWriteSemaphore semaphore'. The constructor 'Integrator(Task task, ReadWriteSemaphore semaphore)' initializes these fields. The 'run()' method is annotated with '@Override' and contains logic to get the task count, loop through tasks, acquire the semaphore, check for interruption, read data, and integrate it using 'Functions.integrate'.

```
1 package threads;
2 import functions.Functions;
3
4 public class Integrator extends Thread {
5     private final Task task;
6     private final ReadWriteSemaphore semaphore;
7
8     public Integrator(Task task, ReadWriteSemaphore semaphore) {
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 семфор 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);
```



This screenshot continues the code from the previous one, showing the completion of the 'run()' method. It includes the calculation of the result, printing it with formatted output, releasing the semaphore with 'semaphore.endRead()', and handling 'InterruptedException' and 'IllegalArgumentException' with appropriate error messages and semaphore release.

```
32        double result = Functions.integrate(task.getFunction(), left, right, step);
33        System.out.printf("Integrator[%d]: Result %.3f %.3f %.3f %.6f\n", i, left, right, step, result);
34
35        semaphore.endRead(); //освобождаем семфор
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 {
    private final Task task;
    private final ReadWriteSemaphore semaphore;

    public Generator(Task task, ReadWriteSemaphore semaphore) {
        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;
            }
        }
    }
}
```

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

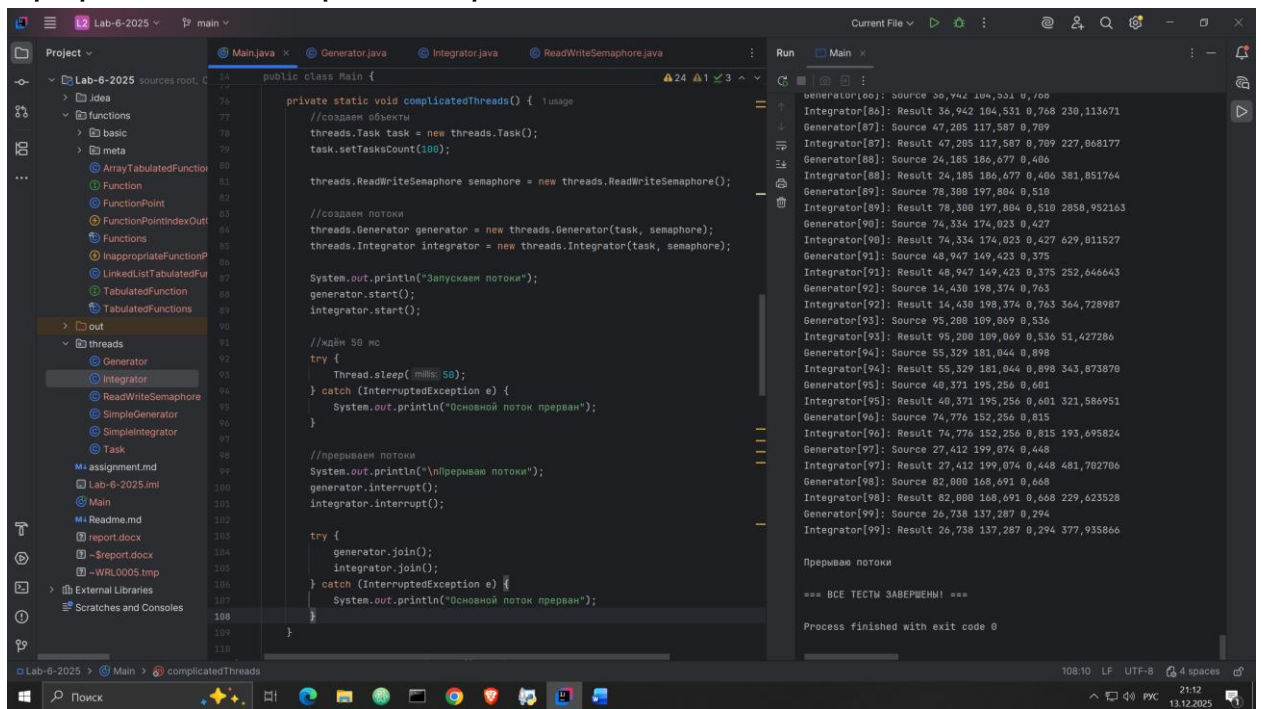
        //генерируем данные
        double base = 1 + Math.random() * 9;
        double left = Math.random() * 100;
        double right = 100 + Math.random() * 100;
        double step = Math.random();

        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;
    }
}
```

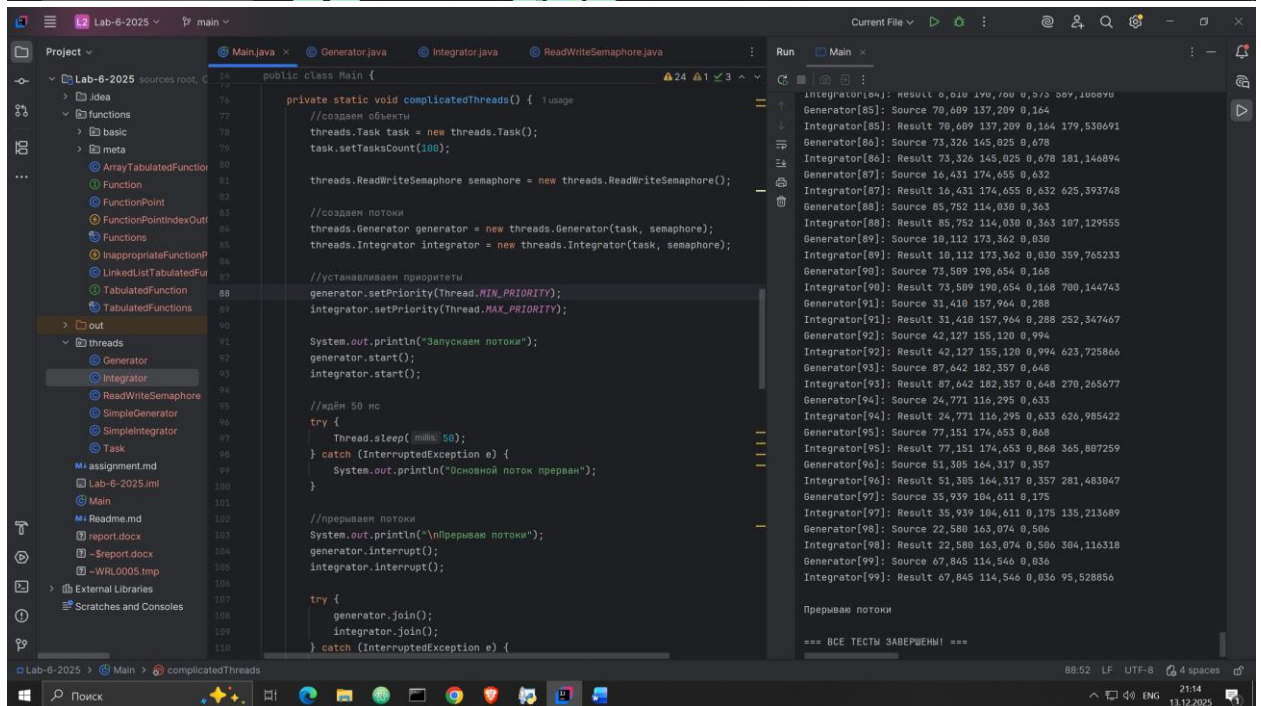
создал метод complicatedThreads с реализацией обработки прерываний и протестировал



```
public class Main {  
    private static void complicatedThreads() {  
        //создаем объекты  
        threads.Task task = new threads.Task();  
        task.setTaskCount(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 потоки");  
        generator.start();  
        integrator.start();  
  
        //ждем 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");  
        }  
    }  
}
```

Run Main

```
generator[00]: Source 30,942 104,531 0,700  
Integrator[86]: Result 36,942 104,531 0,768 230,113671  
Generator[87]: Source 47,285 117,587 0,709  
Integrator[87]: Result 47,285 117,587 0,709 227,868177  
Generator[88]: Source 24,185 186,677 0,486  
Integrator[88]: Result 24,185 186,677 0,486 381,851764  
Generator[89]: Source 78,388 197,884 0,519  
Integrator[89]: Result 78,388 197,884 0,519 2858,952163  
Generator[90]: Source 74,334 174,023 0,427  
Integrator[90]: Result 74,334 174,023 0,427 629,811527  
Generator[91]: Source 48,947 149,423 0,375  
Integrator[91]: Result 48,947 149,423 0,375 252,646643  
Generator[92]: Source 14,430 198,374 0,763  
Integrator[92]: Result 14,430 198,374 0,763 364,728987  
Generator[93]: Source 95,200 109,069 0,536  
Integrator[93]: Result 95,200 109,069 0,536 51,427286  
Generator[94]: Source 55,329 181,044 0,898  
Integrator[94]: Result 55,329 181,044 0,898 343,873870  
Generator[95]: Source 40,371 195,256 0,601  
Integrator[95]: Result 40,371 195,256 0,601 321,586951  
Generator[96]: Source 74,776 152,256 0,815  
Integrator[96]: Result 74,776 152,256 0,815 193,695824  
Generator[97]: Source 27,412 199,074 0,448  
Integrator[97]: Result 27,412 199,074 0,448 481,702706  
Generator[98]: Source 82,088 168,691 0,668  
Integrator[98]: Result 82,088 168,691 0,668 229,623528  
Generator[99]: Source 26,738 137,287 0,294  
Integrator[99]: Result 26,738 137,287 0,294 377,935866  
  
Прерываем потоки  
  
=== ВСЕ ТЕСТЫ ЗАВЕРШЕНЫ ===  
  
Process finished with exit code 0
```



```
public class Main {  
    private static void complicatedThreads() {  
        //создаем объекты  
        threads.Task task = new threads.Task();  
        task.setTaskCount(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);  
  
        //устанавливаем приоритеты  
        generator.setPriority(Thread.MIN_PRIORITY);  
        integrator.setPriority(Thread.MAX_PRIORITY);  
  
        System.out.println("Запускаем потоки");  
        generator.start();  
        integrator.start();  
  
        //ждем 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");  
        }  
    }  
}
```

Run Main

```
Integrator[04]: Result 0,010 190,700 0,373 000,000000  
Generator[85]: Source 70,609 137,209 0,164  
Integrator[85]: Result 70,609 137,209 0,164 179,530691  
Generator[86]: Source 73,326 145,025 0,678  
Integrator[86]: Result 73,326 145,025 0,678 181,146894  
Generator[87]: Source 16,431 174,655 0,632  
Integrator[87]: Result 16,431 174,655 0,632 625,393748  
Generator[88]: Source 85,752 114,030 0,363  
Integrator[88]: Result 85,752 114,030 0,363 107,129555  
Generator[89]: Source 10,112 173,362 0,030  
Integrator[89]: Result 10,112 173,362 0,030 359,765233  
Generator[90]: Source 73,509 190,654 0,168  
Integrator[90]: Result 73,509 190,654 0,168 700,144743  
Generator[91]: Source 31,410 157,964 0,288  
Integrator[91]: Result 31,410 157,964 0,288 252,347467  
Generator[92]: Source 42,127 155,120 0,994  
Integrator[92]: Result 42,127 155,120 0,994 623,725866  
Generator[93]: Source 87,642 182,357 0,648  
Integrator[93]: Result 87,642 182,357 0,648 270,265677  
Generator[94]: Source 24,771 116,295 0,633  
Integrator[94]: Result 24,771 116,295 0,633 626,985422  
Generator[95]: Source 77,151 174,653 0,868  
Integrator[95]: Result 77,151 174,653 0,868 365,887259  
Generator[96]: Source 51,305 164,317 0,357  
Integrator[96]: Result 51,305 164,317 0,357 281,483047  
Generator[97]: Source 35,939 104,611 0,175  
Integrator[97]: Result 35,939 104,611 0,175 135,213689  
Generator[98]: Source 22,588 163,074 0,586  
Integrator[98]: Result 22,588 163,074 0,586 384,116318  
Generator[99]: Source 67,845 114,546 0,836  
Integrator[99]: Result 67,845 114,546 0,836 95,528856  
  
Прерываем потоки  
  
=== ВСЕ ТЕСТЫ ЗАВЕРШЕНЫ ===
```