Министерство науки и высшего образования Российской Федерации

федеральное государственное автономное
образовательное учреждение высшего образования
«Самарский национальный исследовательский университет
имени академика С.П. Королева»

Институт информатики и кибернетики

Кафедра технической кибернетики

Отчет по лабораторной работе №6

Дисциплина: «ООП»

Тема «Многопоточное приложение»

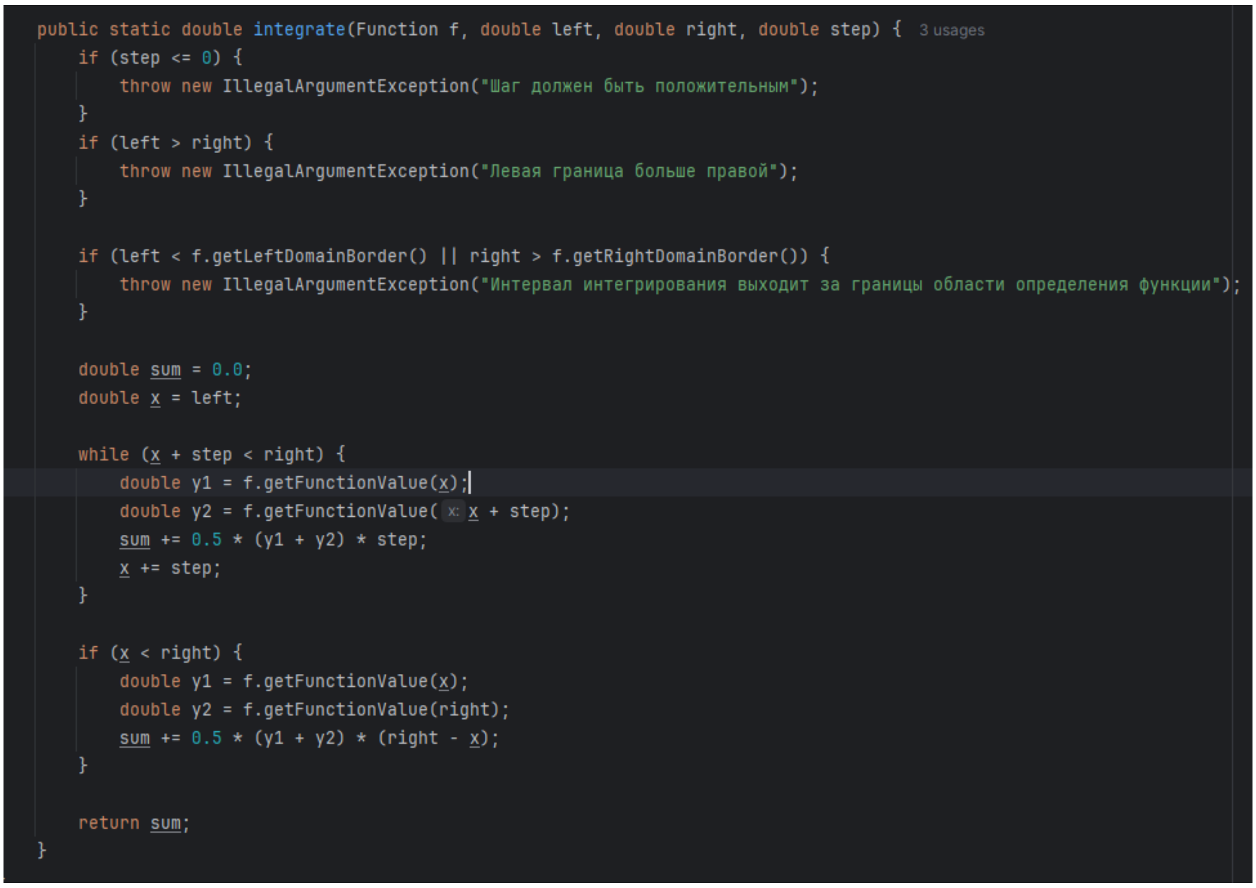
Выполнил: Чечеткин Д.А.

Группа: 6201-120303D

Самара, 2025

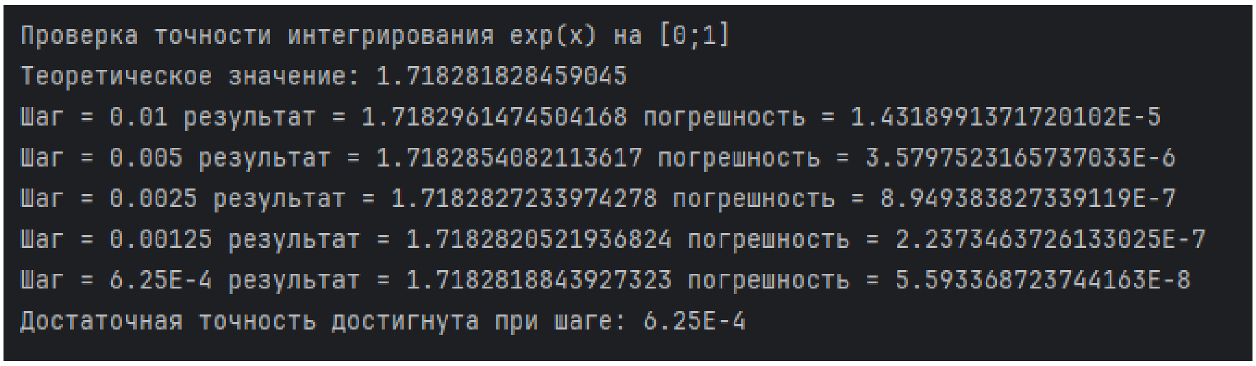
Задание 1

В класс **Functions** был добавлен метод, возвращающий значение интеграла функции, вычисленное с помощью численного метода. Метод получает ссылку типа Function на объект функции, значения границ и шаг дискретизации.



```java
public static double integrate(Function f, double left, double right, double step) {  3 usages
    if (step <= 0) {
        throw new IllegalArgumentException("Шаг должен быть положительным");
    }
    if (left > right) {
        throw new IllegalArgumentException("Левая граница больше правой");
    }

    if (left < f.getLeftDomainBorder() || right > f.getRightDomainBorder()) {
        throw new IllegalArgumentException("Интервал интегрирования выходит за границы области определения функции");
    }

    double sum = 0.0;
    double x = left;

    while (x + step < right) {
        double y1 = f.getFunctionValue(x);
        double y2 = f.getFunctionValue( x: x + step);
        sum += 0.5 * (y1 + y2) * step;
        x += step;
    }

    if (x < right) {
        double y1 = f.getFunctionValue(x);
        double y2 = f.getFunctionValue(right);
        sum += 0.5 * (y1 + y2) * (right - x);
    }

    return sum;
}
```

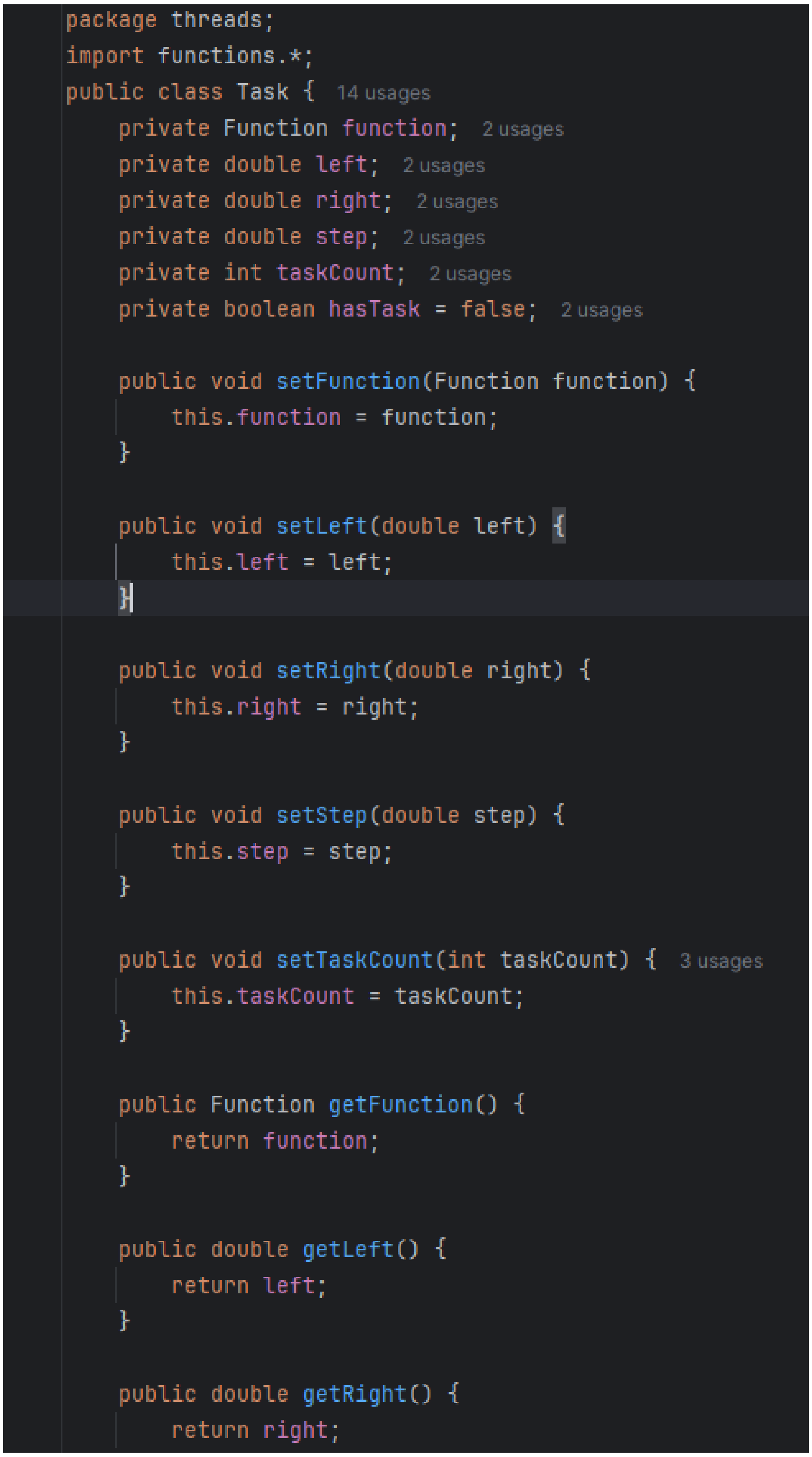
Проверка точности интегрирования exp.



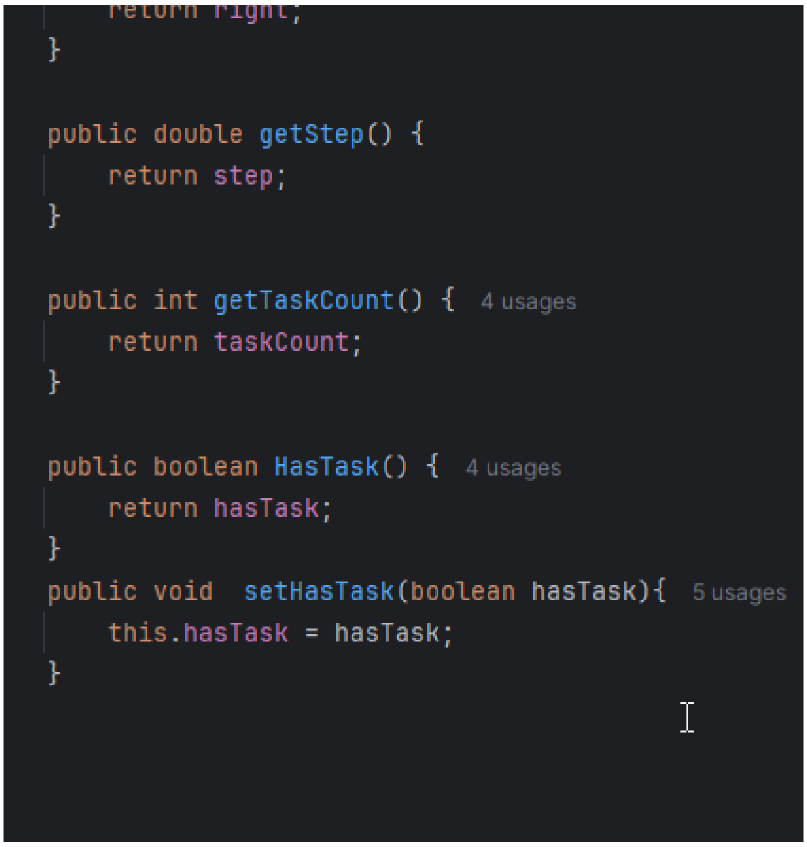
```
Проверка точности интегрирования exp(x) на [0;1]
Теоретическое значение: 1.718281828459045
Шаг = 0.01 результат = 1.7182961474504168 погрешность = 1.4318991371720102E-5
Шаг = 0.005 результат = 1.7182854082113617 погрешность = 3.5797523165737033E-6
Шаг = 0.0025 результат = 1.7182827233974278 погрешность = 8.949383827339119E-7
Шаг = 0.00125 результат = 1.7182820521936824 погрешность = 2.2373463726133025E-7
Шаг = 6.25E-4 результат = 1.7182818843927323 погрешность = 5.593368723744163E-8
Достаточная точность достигнута при шаге: 6.25E-4
```

Задание 2

Создан пакет **threads,** в котором описан класс **Task**, содержащий ссылку на объект интегрируемой функции, границы интегрирования, шаг дискретизации, количество выполняемых заданий.

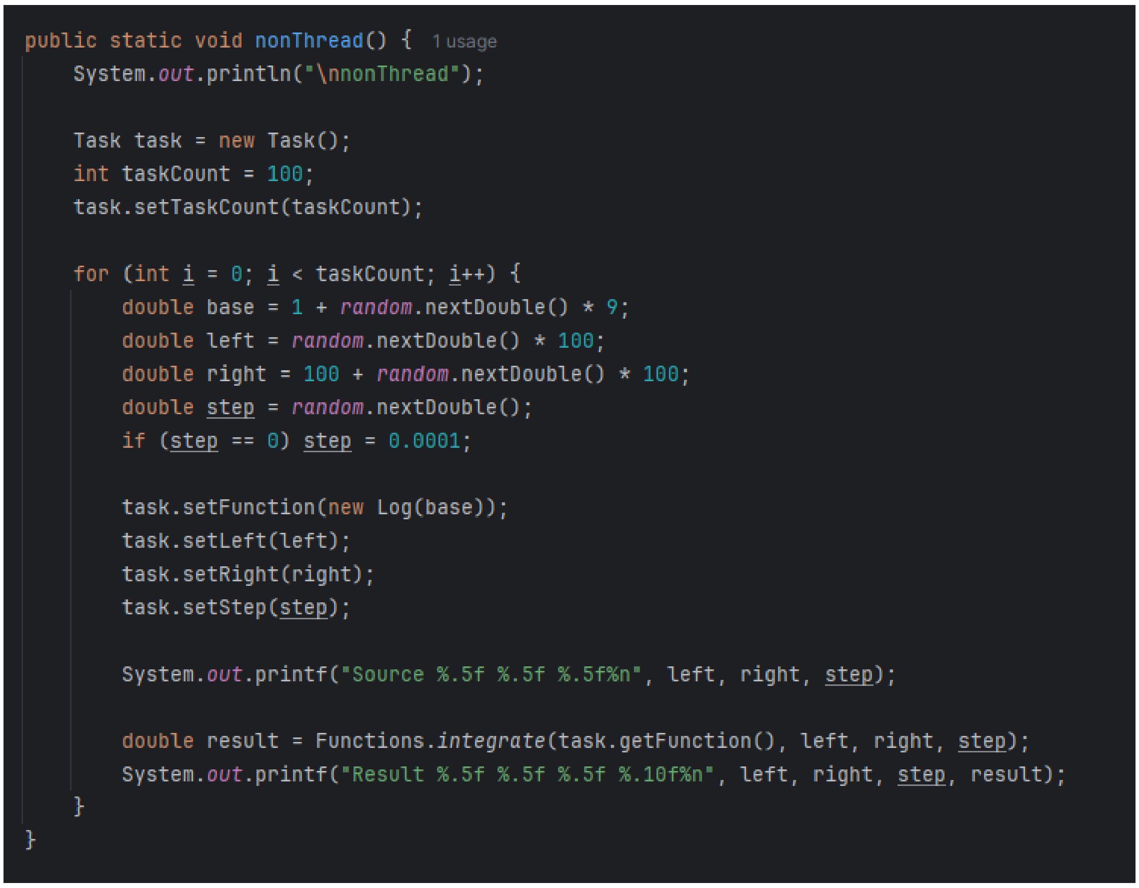


```
package threads;
import functions.*;
public class Task {  14 usages
    private Function function;  2 usages
    private double left;  2 usages
    private double right;  2 usages
    private double step;  2 usages
    private int taskCount;  2 usages
    private boolean hasTask = false;  2 usages

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

    public void setLeft(double left) {
        this.left = left;
    }

    public void setRight(double right) {
        this.right = right;
    }

    public void setStep(double step) {
        this.step = step;
    }

    public void setTaskCount(int taskCount) {  3 usages
        this.taskCount = taskCount;
    }

    public Function getFunction() {
        return function;
    }

    public double getLeft() {
        return left;
    }

    public double getRight() {
        return right;
```



```
        return right;
    }

    public double getStep() {
        return step;
    }

    public int getTaskCount() {  4 usages
        return taskCount;
    }

    public boolean HasTask() {  4 usages
        return hasTask;
    }
    public void  setHasTask(boolean hasTask){  5 usages
        this.hasTask = hasTask;
    }
```

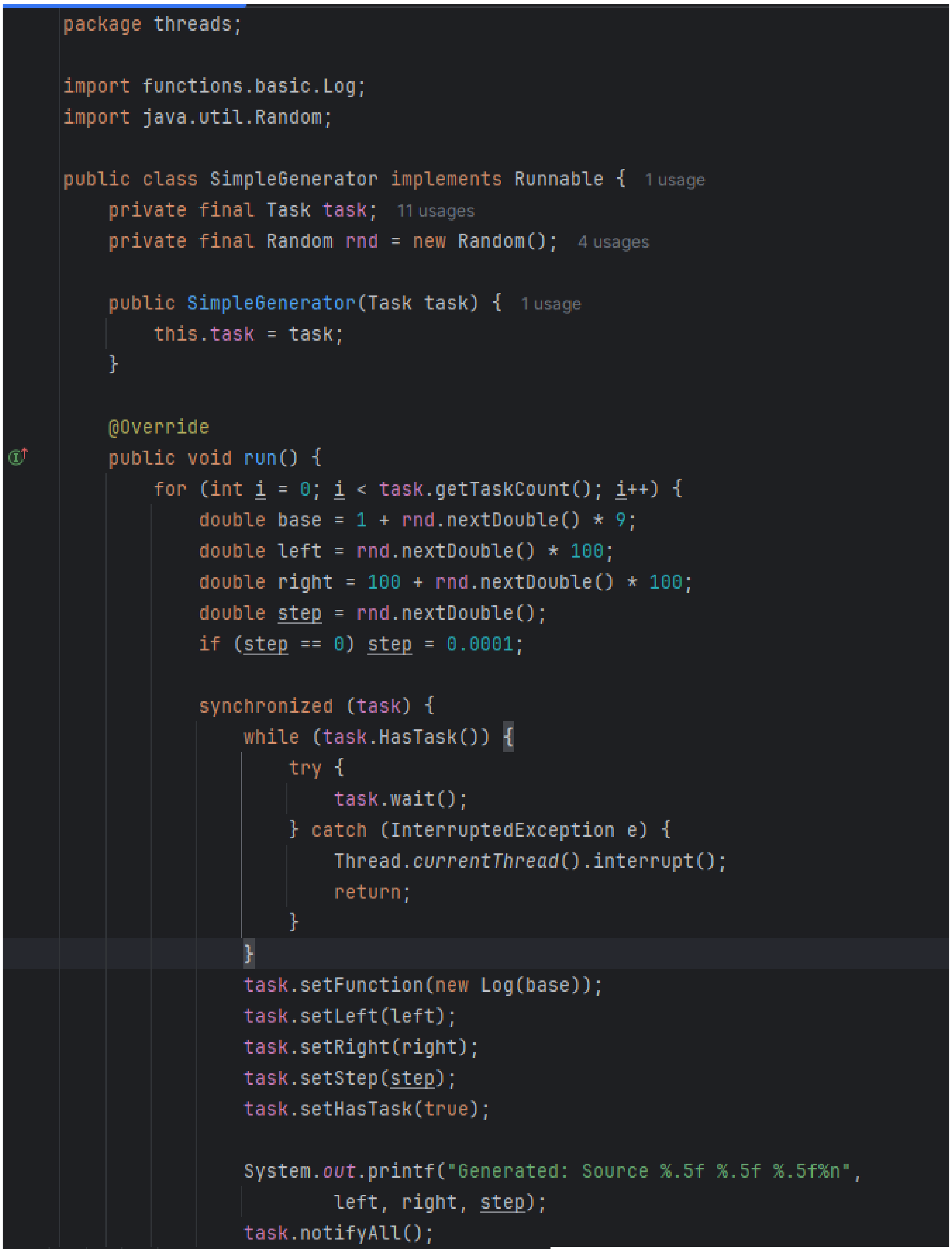
Описан метод **nonThread**, реализующий последовательную версию программы.



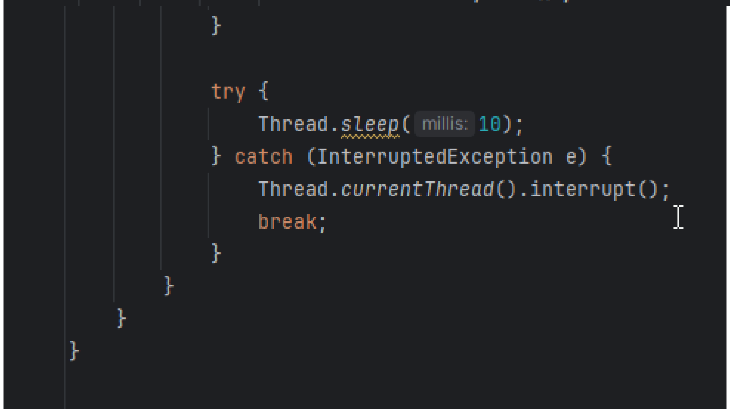
```
public static void nonThread() {  1 usage
    System.out.println("\nnonThread");

    Task task = new Task();
    int taskCount = 100;
    task.setTaskCount(taskCount);

    for (int i = 0; i < taskCount; i++) {
        double base = 1 + random.nextDouble() * 9;
        double left = random.nextDouble() * 100;
        double right = 100 + random.nextDouble() * 100;
        double step = random.nextDouble();
        if (step == 0) step = 0.0001;

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

        System.out.printf("Source %.5f %.5f %.5f%n", left, right, step);

        double result = Functions.integrate(task.getFunction(), left, right, step);
        System.out.printf("Result %.5f %.5f %.5f %.10f%n", left, right, step, result);
    }
}
```

Задание 3

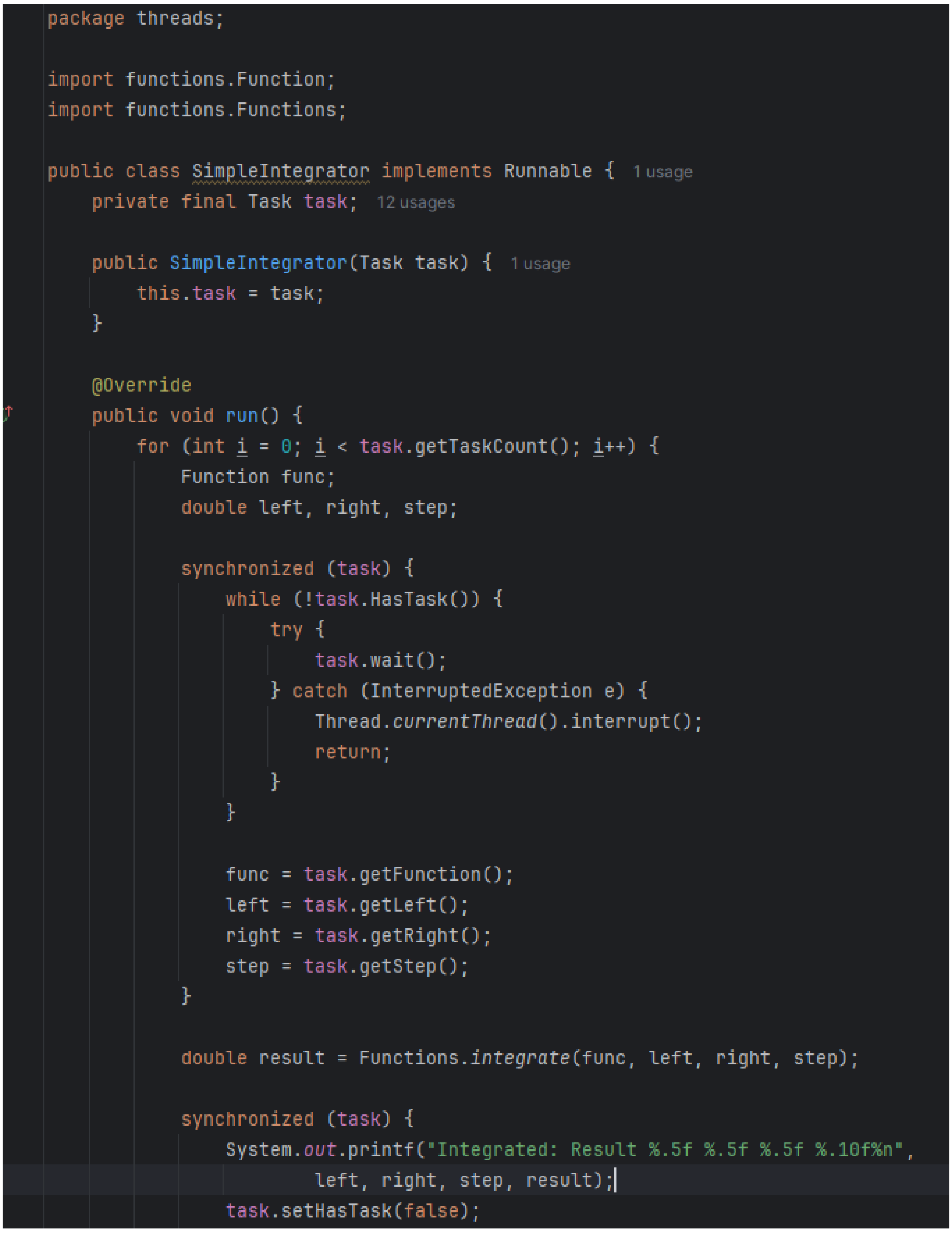
В пакете **threads** созданы классы **simpleGenerator** и **simpleIntegrator,** реализующие интерфейс **Runnable,** формирующие и решающие задачи.



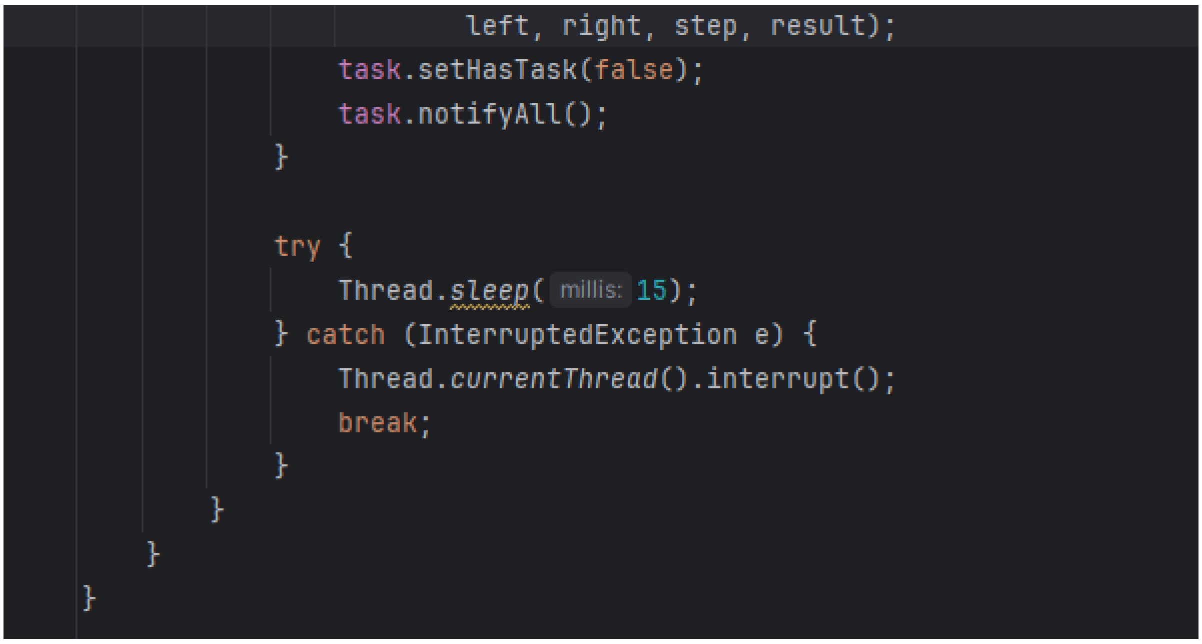
```java
package threads;

import functions.basic.Log;
import java.util.Random;

public class SimpleGenerator implements Runnable {  1 usage
    private final Task task;  11 usages
    private final Random rnd = new Random();  4 usages

    public SimpleGenerator(Task task) {  1 usage
        this.task = task;
    }

    @Override
    public void run() {
        for (int i = 0; i < task.getTaskCount(); i++) {
            double base = 1 + rnd.nextDouble() * 9;
            double left = rnd.nextDouble() * 100;
            double right = 100 + rnd.nextDouble() * 100;
            double step = rnd.nextDouble();
            if (step == 0) step = 0.0001;

            synchronized (task) {
                while (task.HasTask()) {
                    try {
                        task.wait();
                    } catch (InterruptedException e) {
                        Thread.currentThread().interrupt();
                        return;
                    }
                }
                task.setFunction(new Log(base));
                task.setLeft(left);
                task.setRight(right);
                task.setStep(step);
                task.setHasTask(true);

                System.out.printf("Generated: Source %.5f %.5f %.5f%n",
                        left, right, step);
                task.notifyAll();
```



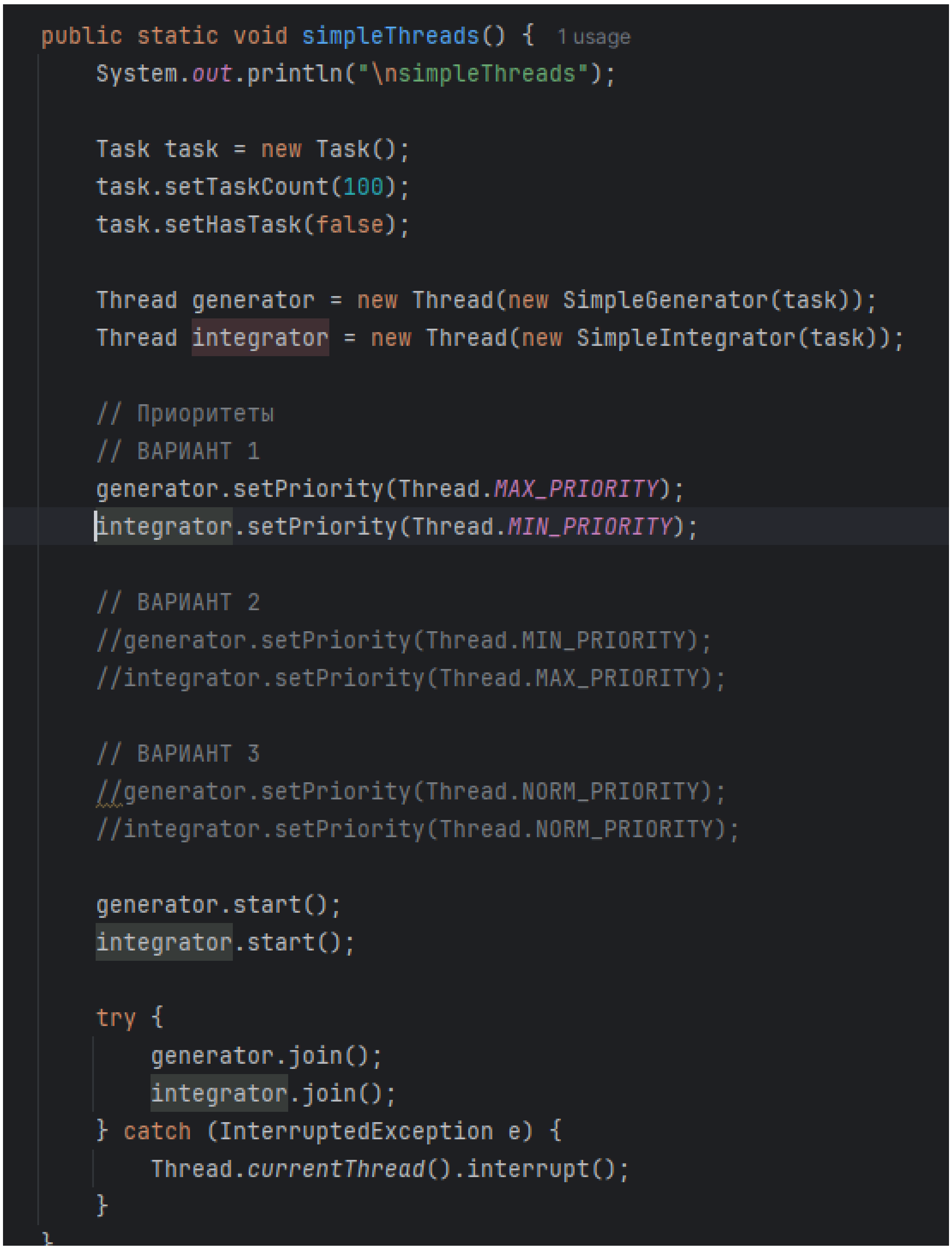
```java
            }

            try {
                Thread.sleep( millis: 10);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                break;
            }
        }
    }
}
```



```
package threads;

import functions.Function;
import functions.Functions;

public class SimpleIntegrator implements Runnable {  1 usage
    private final Task task;  12 usages

    public SimpleIntegrator(Task task) {  1 usage
        this.task = task;
    }

    @Override
    public void run() {
        for (int i = 0; i < task.getTaskCount(); i++) {
            Function func;
            double left, right, step;

            synchronized (task) {
                while (!task.HasTask()) {
                    try {
                        task.wait();
                    } catch (InterruptedException e) {
                        Thread.currentThread().interrupt();
                        return;
                    }
                }

                func = task.getFunction();
                left = task.getLeft();
                right = task.getRight();
                step = task.getStep();
            }

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

            synchronized (task) {
                System.out.printf("Integrated: Result %.5f %.5f %.5f %.10f%n",
                        left, right, step, result);
                task.setHasTask(false);
```


```
                            left, right, step, result);
                    task.setHasTask(false);
                    task.notifyAll();
                }

                try {
                    Thread.sleep( millis: 15);
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                    break;
                }
            }
        }
    }
```

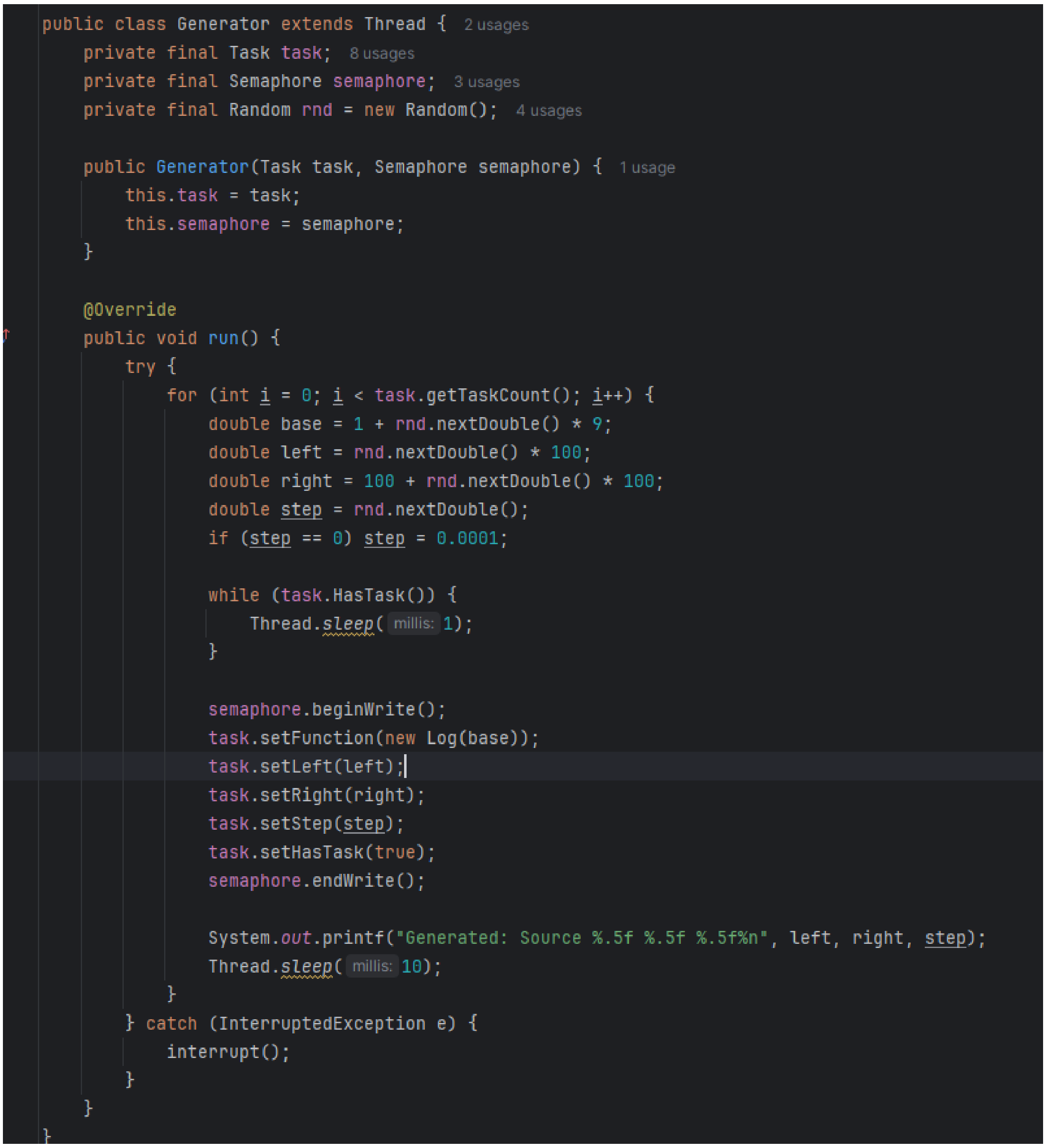
Создан метод **simpleThreads**, запускающий потоки, основанные на классах **simpleGenerator** и **simpleIntegrator.**

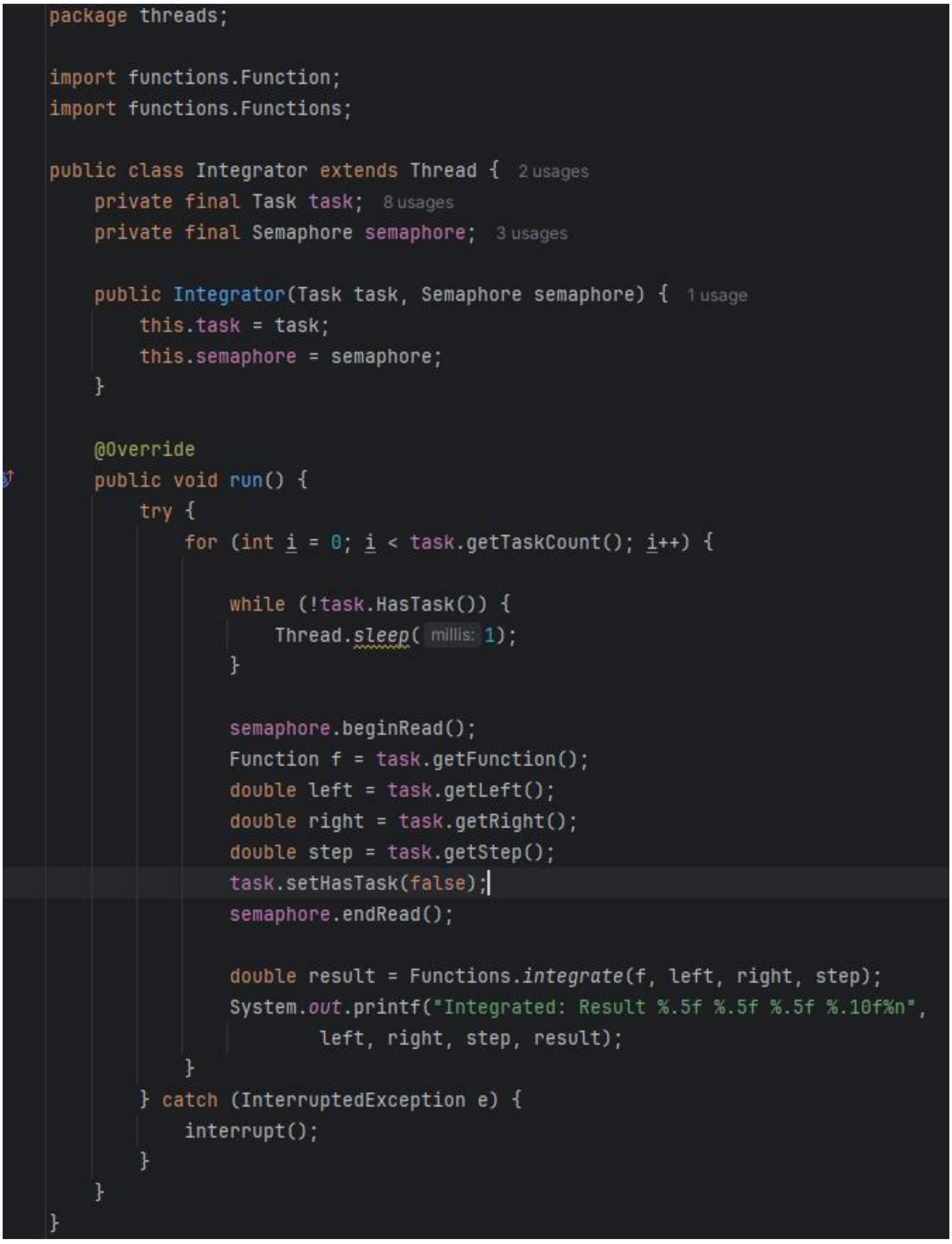


```java
public static void simpleThreads() {  1 usage
    System.out.println("\nsimpleThreads");

    Task task = new Task();
    task.setTaskCount(100);
    task.setHasTask(false);

    Thread generator = new Thread(new SimpleGenerator(task));
    Thread integrator = new Thread(new SimpleIntegrator(task));

    // Приоритеты
    // ВАРИАНТ 1
    generator.setPriority(Thread.MAX_PRIORITY);
    integrator.setPriority(Thread.MIN_PRIORITY);

    // ВАРИАНТ 2
    //generator.setPriority(Thread.MIN_PRIORITY);
    //integrator.setPriority(Thread.MAX_PRIORITY);

    // ВАРИАНТ 3
    //generator.setPriority(Thread.NORM_PRIORITY);
    //integrator.setPriority(Thread.NORM_PRIORITY);

    generator.start();
    integrator.start();

    try {
        generator.join();
        integrator.join();
    } catch (InterruptedException e) {
        Thread.currentThread().interrupt();
    }
}
```

Задание 4

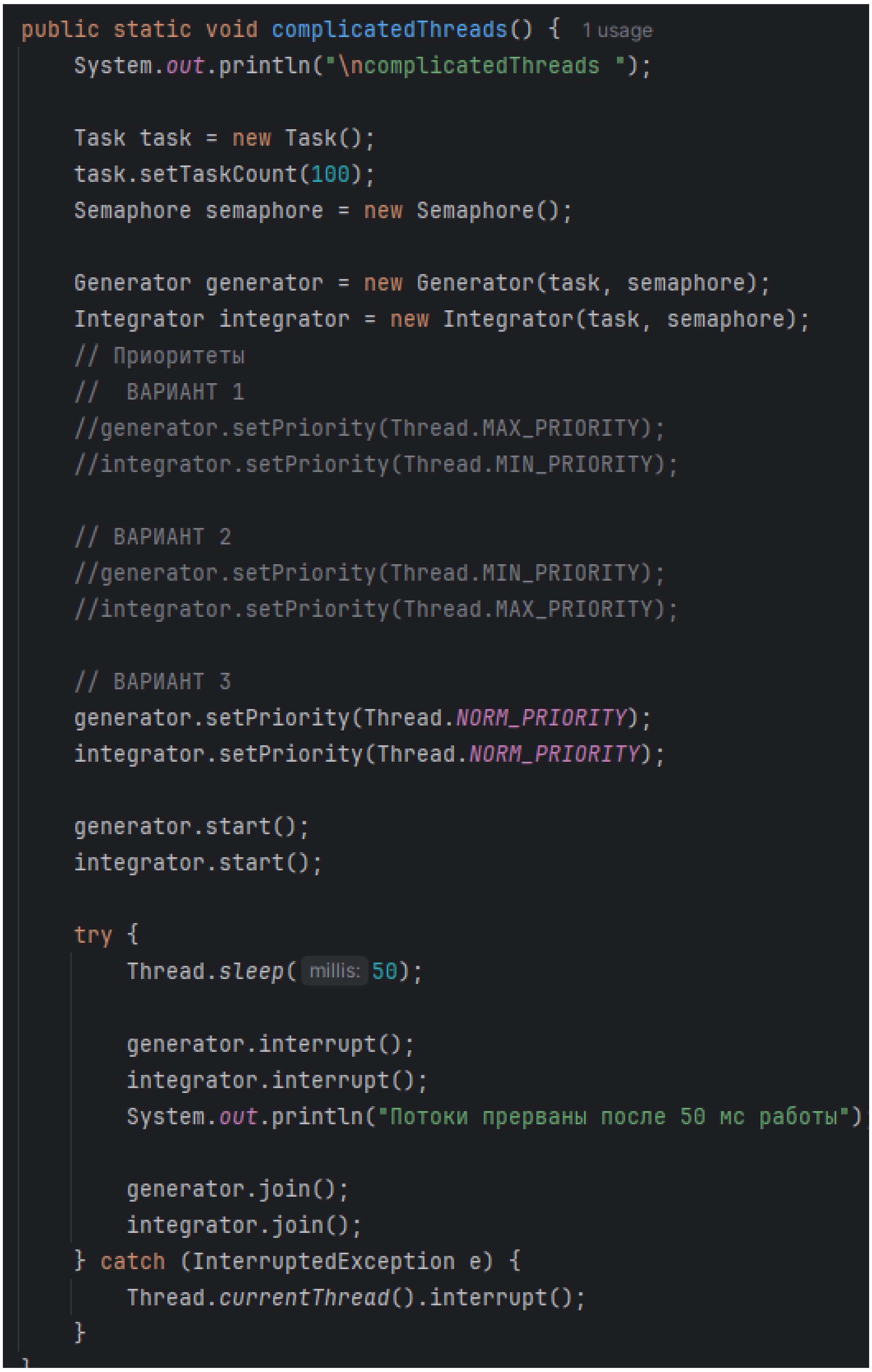
Созданы классы **Generator** и **Integrator,** расширяющие класс **Threads ,** выполняющие те же действия, что и предыдущие версии и использующие возможности семафора.


```java
public class Generator extends Thread {  2 usages
    private final Task task;  8 usages
    private final Semaphore semaphore;  3 usages
    private final Random rnd = new Random();  4 usages

    public Generator(Task task, Semaphore semaphore) {  1 usage
        this.task = task;
        this.semaphore = semaphore;
    }

    @Override
    public void run() {
        try {
            for (int i = 0; i < task.getTaskCount(); i++) {
                double base = 1 + rnd.nextDouble() * 9;
                double left = rnd.nextDouble() * 100;
                double right = 100 + rnd.nextDouble() * 100;
                double step = rnd.nextDouble();
                if (step == 0) step = 0.0001;

                while (task.HasTask()) {
                    Thread.sleep( millis: 1);
                }

                semaphore.beginWrite();
                task.setFunction(new Log(base));
                task.setLeft(left);
                task.setRight(right);
                task.setStep(step);
                task.setHasTask(true);
                semaphore.endWrite();

                System.out.printf("Generated: Source %.5f %.5f %.5f%n", left, right, step);
                Thread.sleep( millis: 10);
            }
        } catch (InterruptedException e) {
            interrupt();
        }
    }
}
```



```java
package threads;

import functions.Function;
import functions.Functions;

public class Integrator extends Thread {  2 usages
    private final Task task;  8 usages
    private final Semaphore semaphore;  3 usages

    public Integrator(Task task, Semaphore semaphore) {  1 usage
        this.task = task;
        this.semaphore = semaphore;
    }

    @Override
    public void run() {
        try {
            for (int i = 0; i < task.getTaskCount(); i++) {

                while (!task.HasTask()) {
                    Thread.sleep( millis: 1);
                }

                semaphore.beginRead();
                Function f = task.getFunction();
                double left = task.getLeft();
                double right = task.getRight();
                double step = task.getStep();
                task.setHasTask(false);
                semaphore.endRead();

                double result = Functions.integrate(f, left, right, step);
                System.out.printf("Integrated: Result %.5f %.5f %.5f %.10f%n",
                        left, right, step, result);
            }
        } catch (InterruptedException e) {
            interrupt();
        }
    }
}
```

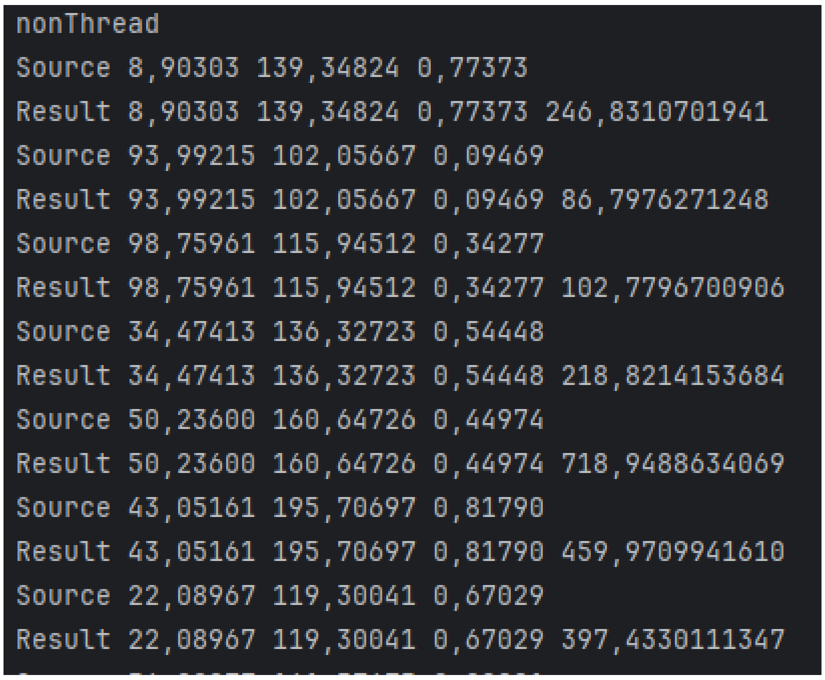
Создан метод **ComplicatedThreads**, создающий и реализующий потоки вычислений



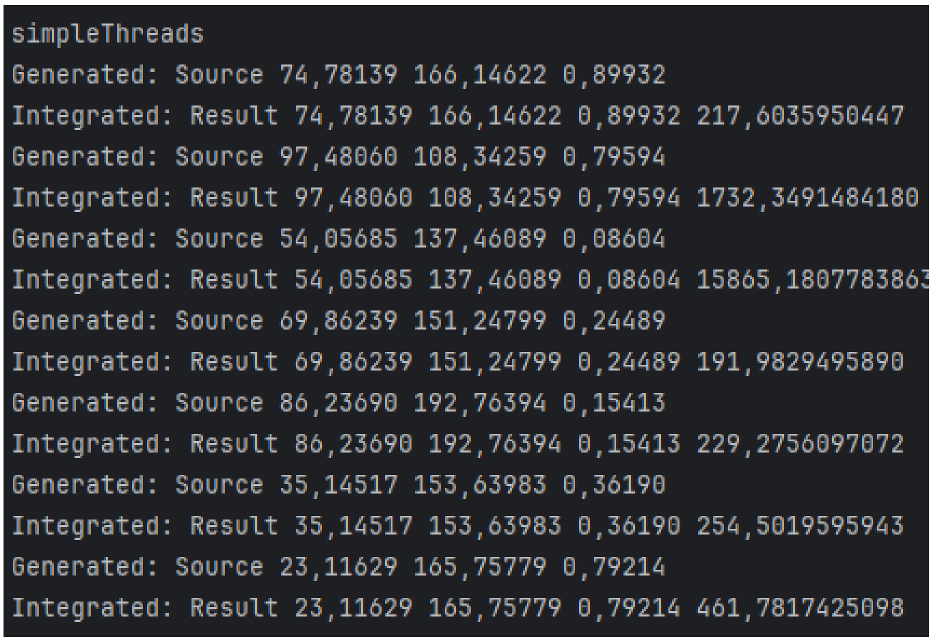
```java
public static void complicatedThreads() {  1 usage
    System.out.println("\ncomplicatedThreads ");

    Task task = new Task();
    task.setTaskCount(100);
    Semaphore semaphore = new Semaphore();

    Generator generator = new Generator(task, semaphore);
    Integrator integrator = new Integrator(task, semaphore);
    // Приоритеты
    //  ВАРИАНТ 1
    //generator.setPriority(Thread.MAX_PRIORITY);
    //integrator.setPriority(Thread.MIN_PRIORITY);

    // ВАРИАНТ 2
    //generator.setPriority(Thread.MIN_PRIORITY);
    //integrator.setPriority(Thread.MAX_PRIORITY);

    // ВАРИАНТ 3
    generator.setPriority(Thread.NORM_PRIORITY);
    integrator.setPriority(Thread.NORM_PRIORITY);

    generator.start();
    integrator.start();

    try {
        Thread.sleep( millis: 50);

        generator.interrupt();
        integrator.interrupt();
        System.out.println("Потоки прерваны после 50 мс работы")

        generator.join();
        integrator.join();
    } catch (InterruptedException e) {
        Thread.currentThread().interrupt();
    }
```

Реализовано прерывание после 50 миллисекунд.

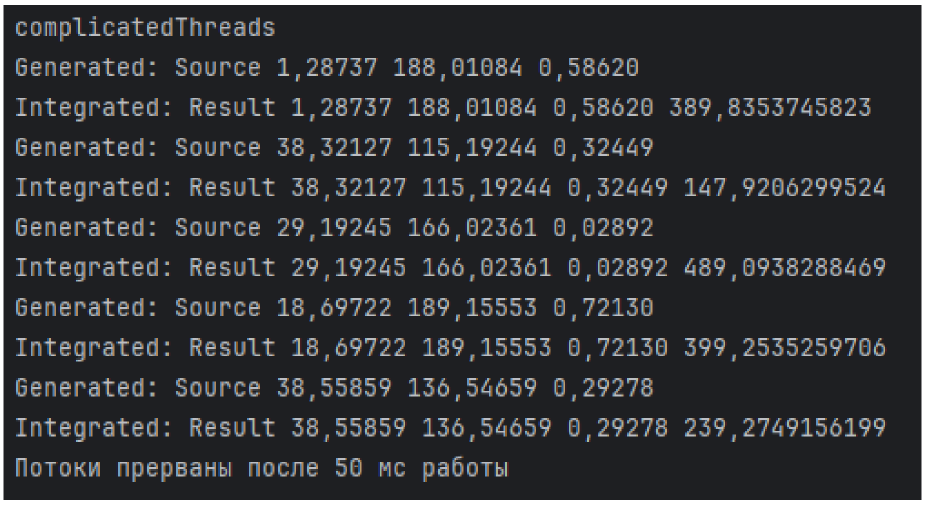
Первые несколько выводов:



```
nonThread
Source 8,90303 139,34824 0,77373
Result 8,90303 139,34824 0,77373 246,8310701941
Source 93,99215 102,05667 0,09469
Result 93,99215 102,05667 0,09469 86,7976271248
Source 98,75961 115,94512 0,34277
Result 98,75961 115,94512 0,34277 102,7796700906
Source 34,47413 136,32723 0,54448
Result 34,47413 136,32723 0,54448 218,8214153684
Source 50,23600 160,64726 0,44974
Result 50,23600 160,64726 0,44974 718,9488634069
Source 43,05161 195,70697 0,81790
Result 43,05161 195,70697 0,81790 459,9709941610
Source 22,08967 119,30041 0,67029
Result 22,08967 119,30041 0,67029 397,4330111347
```

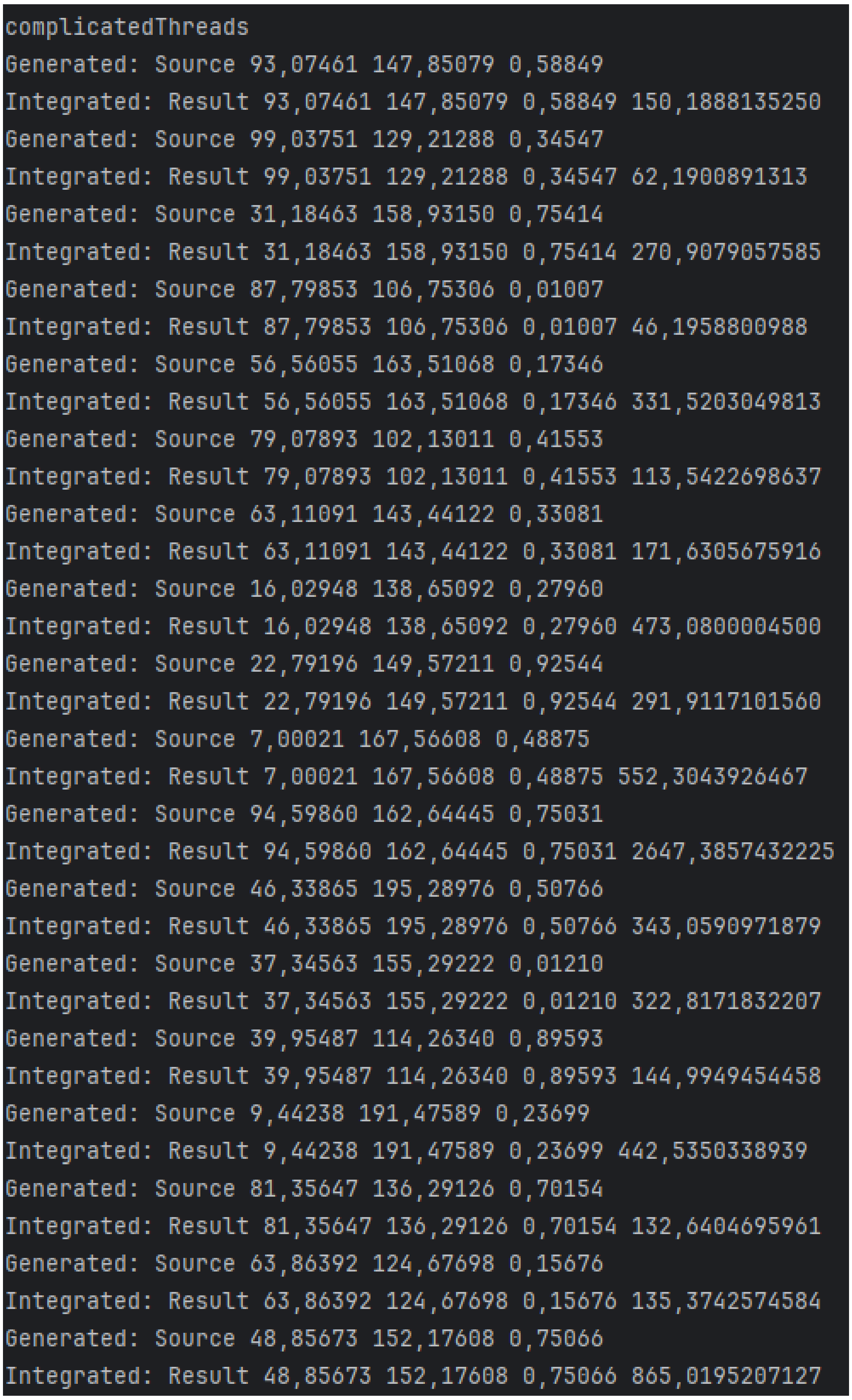


```
simpleThreads
Generated: Source 74,78139 166,14622 0,89932
Integrated: Result 74,78139 166,14622 0,89932 217,6035950447
Generated: Source 97,48060 108,34259 0,79594
Integrated: Result 97,48060 108,34259 0,79594 1732,3491484180
Generated: Source 54,05685 137,46089 0,08604
Integrated: Result 54,05685 137,46089 0,08604 15865,1807783863
Generated: Source 69,86239 151,24799 0,24489
Integrated: Result 69,86239 151,24799 0,24489 191,9829495890
Generated: Source 86,23690 192,76394 0,15413
Integrated: Result 86,23690 192,76394 0,15413 229,2756097072
Generated: Source 35,14517 153,63983 0,36190
Integrated: Result 35,14517 153,63983 0,36190 254,5019595943
Generated: Source 23,11629 165,75779 0,79214
Integrated: Result 23,11629 165,75779 0,79214 461,7817425098
```

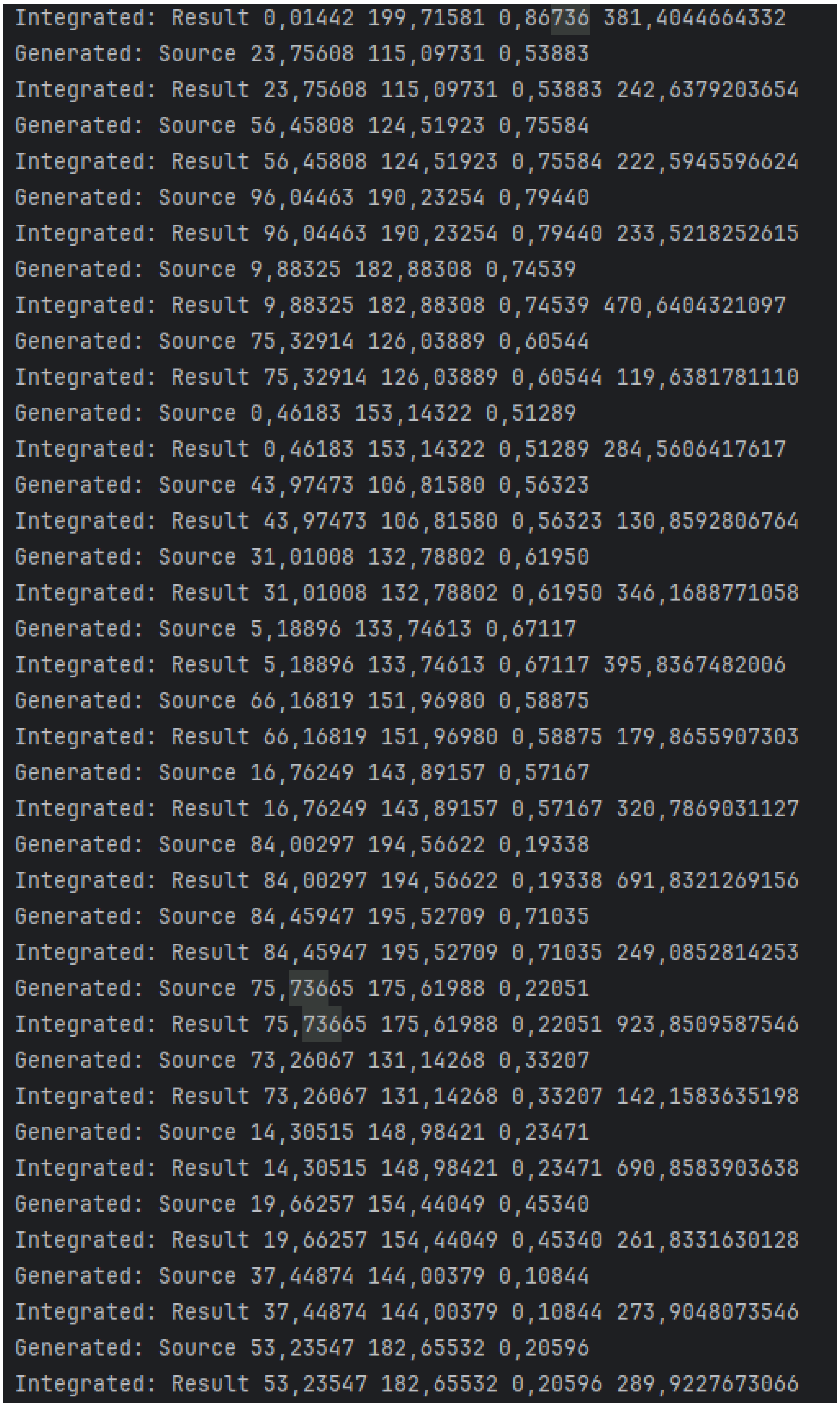


```
complicatedThreads
Generated: Source 1,28737 188,01084 0,58620
Integrated: Result 1,28737 188,01084 0,58620 389,8353745823
Generated: Source 38,32127 115,19244 0,32449
Integrated: Result 38,32127 115,19244 0,32449 147,9206299524
Generated: Source 29,19245 166,02361 0,02892
Integrated: Result 29,19245 166,02361 0,02892 489,0938288469
Generated: Source 18,69722 189,15553 0,72130
Integrated: Result 18,69722 189,15553 0,72130 399,2535259706
Generated: Source 38,55859 136,54659 0,29278
Integrated: Result 38,55859 136,54659 0,29278 239,2749156199
Потоки прерваны после 50 мс работы
```

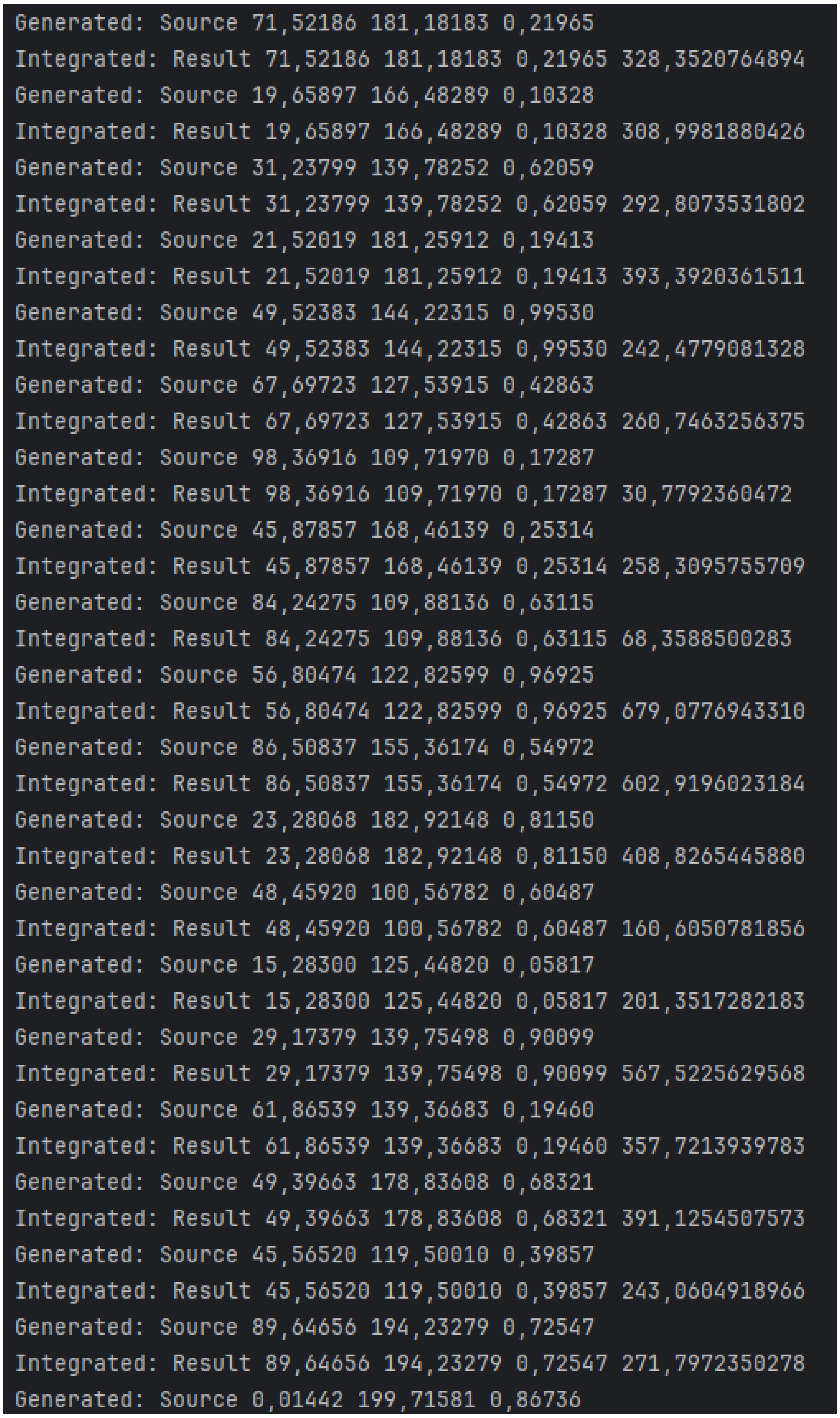
Работа **ComplicatedThreads** без прерывания:



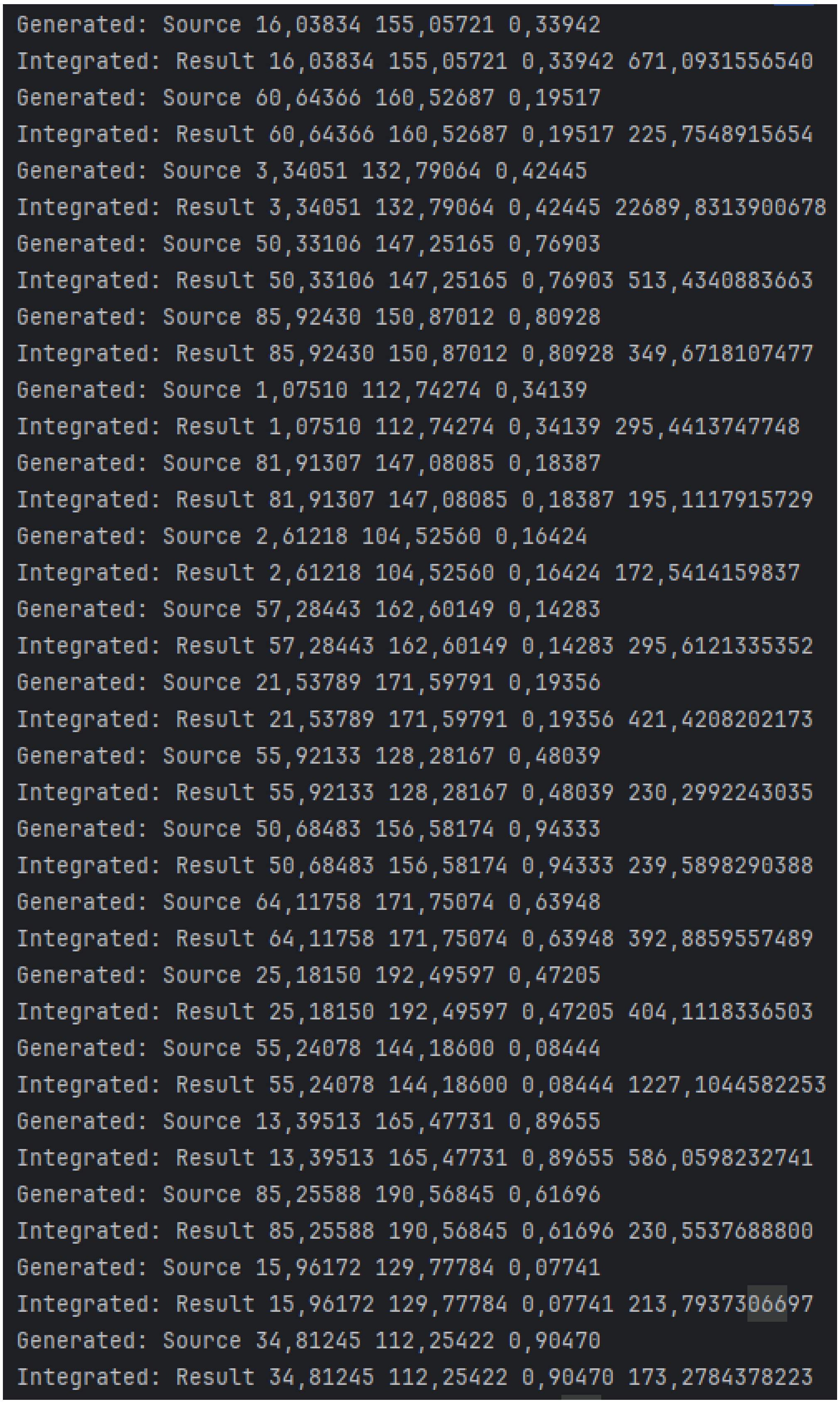
```
complicatedThreads
Generated: Source 93,07461 147,85079 0,58849
Integrated: Result 93,07461 147,85079 0,58849 150,1888135250
Generated: Source 99,03751 129,21288 0,34547
Integrated: Result 99,03751 129,21288 0,34547 62,1900891313
Generated: Source 31,18463 158,93150 0,75414
Integrated: Result 31,18463 158,93150 0,75414 270,9079057585
Generated: Source 87,79853 106,75306 0,01007
Integrated: Result 87,79853 106,75306 0,01007 46,1958800988
Generated: Source 56,56055 163,51068 0,17346
Integrated: Result 56,56055 163,51068 0,17346 331,5203049813
Generated: Source 79,07893 102,13011 0,41553
Integrated: Result 79,07893 102,13011 0,41553 113,5422698637
Generated: Source 63,11091 143,44122 0,33081
Integrated: Result 63,11091 143,44122 0,33081 171,6305675916
Generated: Source 16,02948 138,65092 0,27960
Integrated: Result 16,02948 138,65092 0,27960 473,0800004500
Generated: Source 22,79196 149,57211 0,92544
Integrated: Result 22,79196 149,57211 0,92544 291,9117101560
Generated: Source 7,00021 167,56608 0,48875
Integrated: Result 7,00021 167,56608 0,48875 552,3043926467
Generated: Source 94,59860 162,64445 0,75031
Integrated: Result 94,59860 162,64445 0,75031 2647,3857432225
Generated: Source 46,33865 195,28976 0,50766
Integrated: Result 46,33865 195,28976 0,50766 343,0590971879
Generated: Source 37,34563 155,29222 0,01210
Integrated: Result 37,34563 155,29222 0,01210 322,8171832207
Generated: Source 39,95487 114,26340 0,89593
Integrated: Result 39,95487 114,26340 0,89593 144,9949454458
Generated: Source 9,44238 191,47589 0,23699
Integrated: Result 9,44238 191,47589 0,23699 442,5350338939
Generated: Source 81,35647 136,29126 0,70154
Integrated: Result 81,35647 136,29126 0,70154 132,6404695961
Generated: Source 63,86392 124,67698 0,15676
Integrated: Result 63,86392 124,67698 0,15676 135,3742574584
Generated: Source 48,85673 152,17608 0,75066
Integrated: Result 48,85673 152,17608 0,75066 865,0195207127
```



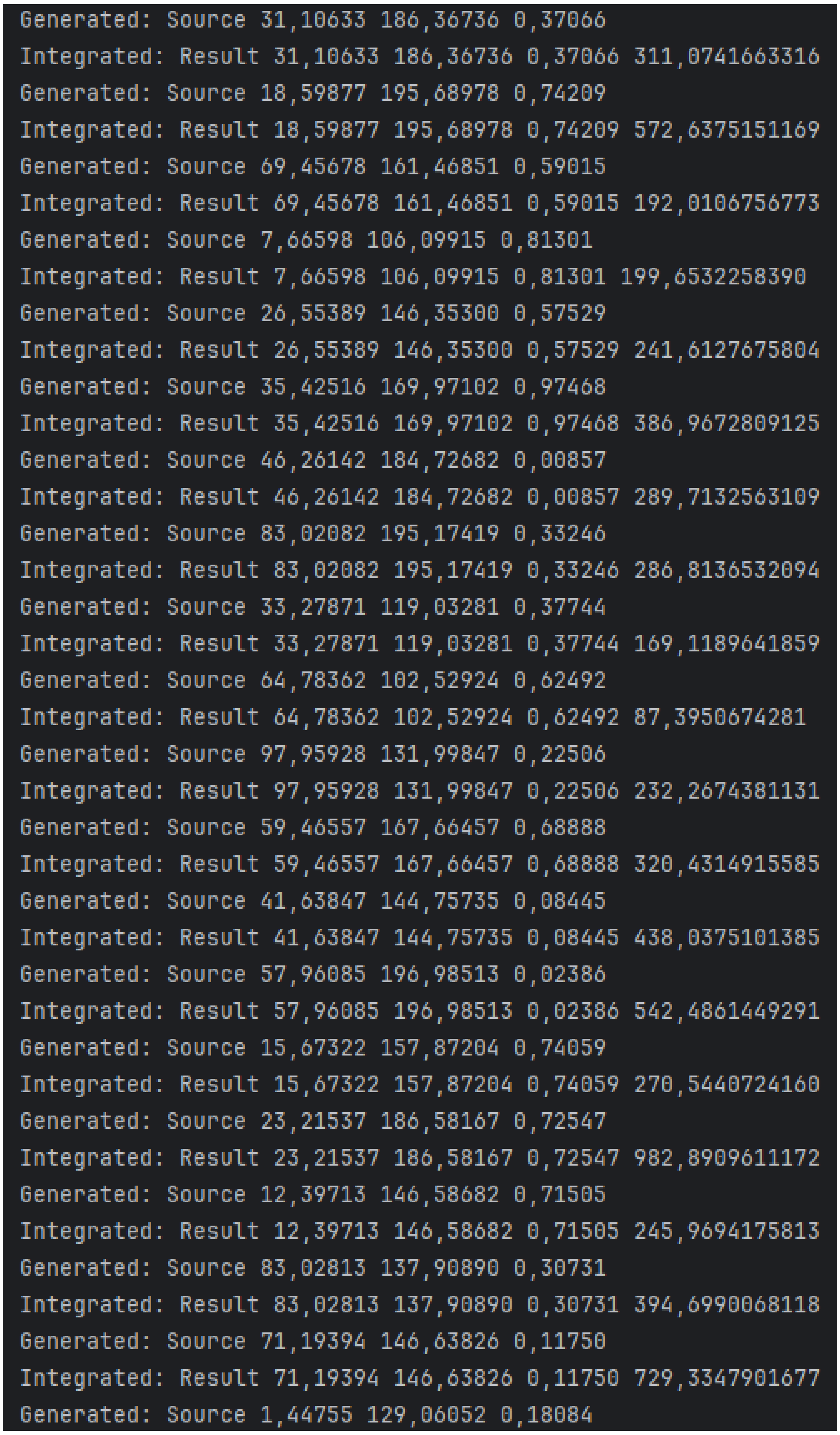
```
Integrated: Result 0,01442 199,71581 0,86736 381,4044664332
Generated: Source 23,75608 115,09731 0,53883
Integrated: Result 23,75608 115,09731 0,53883 242,6379203654
Generated: Source 56,45808 124,51923 0,75584
Integrated: Result 56,45808 124,51923 0,75584 222,5945596624
Generated: Source 96,04463 190,23254 0,79440
Integrated: Result 96,04463 190,23254 0,79440 233,5218252615
Generated: Source 9,88325 182,88308 0,74539
Integrated: Result 9,88325 182,88308 0,74539 470,6404321097
Generated: Source 75,32914 126,03889 0,60544
Integrated: Result 75,32914 126,03889 0,60544 119,6381781110
Generated: Source 0,46183 153,14322 0,51289
Integrated: Result 0,46183 153,14322 0,51289 284,5606417617
Generated: Source 43,97473 106,81580 0,56323
Integrated: Result 43,97473 106,81580 0,56323 130,8592806764
Generated: Source 31,01008 132,78802 0,61950
Integrated: Result 31,01008 132,78802 0,61950 346,1688771058
Generated: Source 5,18896 133,74613 0,67117
Integrated: Result 5,18896 133,74613 0,67117 395,8367482006
Generated: Source 66,16819 151,96980 0,58875
Integrated: Result 66,16819 151,96980 0,58875 179,8655907303
Generated: Source 16,76249 143,89157 0,57167
Integrated: Result 16,76249 143,89157 0,57167 320,7869031127
Generated: Source 84,00297 194,56622 0,19338
Integrated: Result 84,00297 194,56622 0,19338 691,8321269156
Generated: Source 84,45947 195,52709 0,71035
Integrated: Result 84,45947 195,52709 0,71035 249,0852814253
Generated: Source 75,73665 175,61988 0,22051
Integrated: Result 75,73665 175,61988 0,22051 923,8509587546
Generated: Source 73,26067 131,14268 0,33207
Integrated: Result 73,26067 131,14268 0,33207 142,1583635198
Generated: Source 14,30515 148,98421 0,23471
Integrated: Result 14,30515 148,98421 0,23471 690,8583903638
Generated: Source 19,66257 154,44049 0,45340
Integrated: Result 19,66257 154,44049 0,45340 261,8331630128
Generated: Source 37,44874 144,00379 0,10844
Integrated: Result 37,44874 144,00379 0,10844 273,9048073546
Generated: Source 53,23547 182,65532 0,20596
Integrated: Result 53,23547 182,65532 0,20596 289,9227673066
```



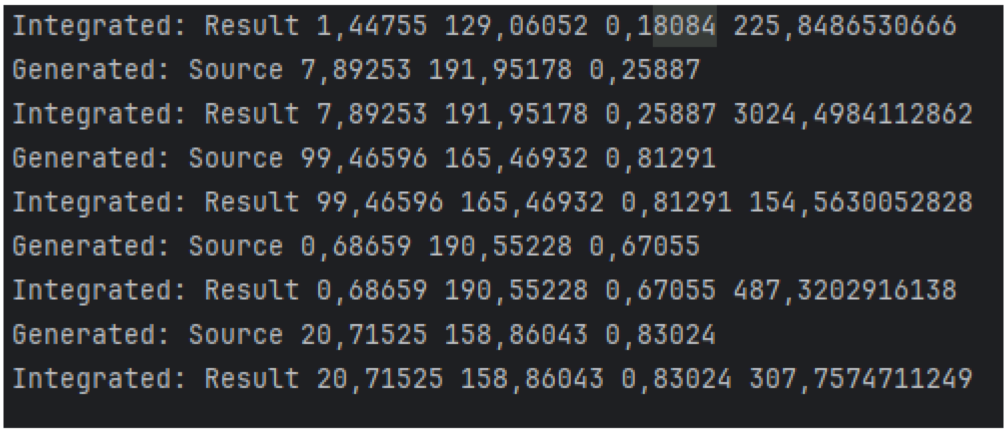
```
Generated: Source 71,52186 181,18183 0,21965
Integrated: Result 71,52186 181,18183 0,21965 328,3520764894
Generated: Source 19,65897 166,48289 0,10328
Integrated: Result 19,65897 166,48289 0,10328 308,9981880426
Generated: Source 31,23799 139,78252 0,62059
Integrated: Result 31,23799 139,78252 0,62059 292,8073531802
Generated: Source 21,52019 181,25912 0,19413
Integrated: Result 21,52019 181,25912 0,19413 393,3920361511
Generated: Source 49,52383 144,22315 0,99530
Integrated: Result 49,52383 144,22315 0,99530 242,4779081328
Generated: Source 67,69723 127,53915 0,42863
Integrated: Result 67,69723 127,53915 0,42863 260,7463256375
Generated: Source 98,36916 109,71970 0,17287
Integrated: Result 98,36916 109,71970 0,17287 30,7792360472
Generated: Source 45,87857 168,46139 0,25314
Integrated: Result 45,87857 168,46139 0,25314 258,3095755709
Generated: Source 84,24275 109,88136 0,63115
Integrated: Result 84,24275 109,88136 0,63115 68,3588500283
Generated: Source 56,80474 122,82599 0,96925
Integrated: Result 56,80474 122,82599 0,96925 679,0776943310
Generated: Source 86,50837 155,36174 0,54972
Integrated: Result 86,50837 155,36174 0,54972 602,9196023184
Generated: Source 23,28068 182,92148 0,81150
Integrated: Result 23,28068 182,92148 0,81150 408,8265445880
Generated: Source 48,45920 100,56782 0,60487
Integrated: Result 48,45920 100,56782 0,60487 160,6050781856
Generated: Source 15,28300 125,44820 0,05817
Integrated: Result 15,28300 125,44820 0,05817 201,3517282183
Generated: Source 29,17379 139,75498 0,90099
Integrated: Result 29,17379 139,75498 0,90099 567,5225629568
Generated: Source 61,86539 139,36683 0,19460
Integrated: Result 61,86539 139,36683 0,19460 357,7213939783
Generated: Source 49,39663 178,83608 0,68321
Integrated: Result 49,39663 178,83608 0,68321 391,1254507573
Generated: Source 45,56520 119,50010 0,39857
Integrated: Result 45,56520 119,50010 0,39857 243,0604918966
Generated: Source 89,64656 194,23279 0,72547
Integrated: Result 89,64656 194,23279 0,72547 271,7972350278
Generated: Source 0,01442 199,71581 0,86736
```



```
Generated: Source 16,03834 155,05721 0,33942
Integrated: Result 16,03834 155,05721 0,33942 671,0931556540
Generated: Source 60,64366 160,52687 0,19517
Integrated: Result 60,64366 160,52687 0,19517 225,7548915654
Generated: Source 3,34051 132,79064 0,42445
Integrated: Result 3,34051 132,79064 0,42445 22689,8313900678
Generated: Source 50,33106 147,25165 0,76903
Integrated: Result 50,33106 147,25165 0,76903 513,4340883663
Generated: Source 85,92430 150,87012 0,80928
Integrated: Result 85,92430 150,87012 0,80928 349,6718107477
Generated: Source 1,07510 112,74274 0,34139
Integrated: Result 1,07510 112,74274 0,34139 295,4413747748
Generated: Source 81,91307 147,08085 0,18387
Integrated: Result 81,91307 147,08085 0,18387 195,1117915729
Generated: Source 2,61218 104,52560 0,16424
Integrated: Result 2,61218 104,52560 0,16424 172,5414159837
Generated: Source 57,28443 162,60149 0,14283
Integrated: Result 57,28443 162,60149 0,14283 295,6121335352
Generated: Source 21,53789 171,59791 0,19356
Integrated: Result 21,53789 171,59791 0,19356 421,4208202173
Generated: Source 55,92133 128,28167 0,48039
Integrated: Result 55,92133 128,28167 0,48039 230,2992243035
Generated: Source 50,68483 156,58174 0,94333
Integrated: Result 50,68483 156,58174 0,94333 239,5898290388
Generated: Source 64,11758 171,75074 0,63948
Integrated: Result 64,11758 171,75074 0,63948 392,8859557489
Generated: Source 25,18150 192,49597 0,47205
Integrated: Result 25,18150 192,49597 0,47205 404,1118336503
Generated: Source 55,24078 144,18600 0,08444
Integrated: Result 55,24078 144,18600 0,08444 1227,1044582253
Generated: Source 13,39513 165,47731 0,89655
Integrated: Result 13,39513 165,47731 0,89655 586,0598232741
Generated: Source 85,25588 190,56845 0,61696
Integrated: Result 85,25588 190,56845 0,61696 230,5537688800
Generated: Source 15,96172 129,77784 0,07741
Integrated: Result 15,96172 129,77784 0,07741 213,7937306697
Generated: Source 34,81245 112,25422 0,90470
Integrated: Result 34,81245 112,25422 0,90470 173,2784378223
```



```
Generated: Source 31,10633 186,36736 0,37066
Integrated: Result 31,10633 186,36736 0,37066 311,0741663316
Generated: Source 18,59877 195,68978 0,74209
Integrated: Result 18,59877 195,68978 0,74209 572,6375151169
Generated: Source 69,45678 161,46851 0,59015
Integrated: Result 69,45678 161,46851 0,59015 192,0106756773
Generated: Source 7,66598 106,09915 0,81301
Integrated: Result 7,66598 106,09915 0,81301 199,6532258390
Generated: Source 26,55389 146,35300 0,57529
Integrated: Result 26,55389 146,35300 0,57529 241,6127675804
Generated: Source 35,42516 169,97102 0,97468
Integrated: Result 35,42516 169,97102 0,97468 386,9672809125
Generated: Source 46,26142 184,72682 0,00857
Integrated: Result 46,26142 184,72682 0,00857 289,7132563109
Generated: Source 83,02082 195,17419 0,33246
Integrated: Result 83,02082 195,17419 0,33246 286,8136532094
Generated: Source 33,27871 119,03281 0,37744
Integrated: Result 33,27871 119,03281 0,37744 169,1189641859
Generated: Source 64,78362 102,52924 0,62492
Integrated: Result 64,78362 102,52924 0,62492 87,3950674281
Generated: Source 97,95928 131,99847 0,22506
Integrated: Result 97,95928 131,99847 0,22506 232,2674381131
Generated: Source 59,46557 167,66457 0,68888
Integrated: Result 59,46557 167,66457 0,68888 320,4314915585
Generated: Source 41,63847 144,75735 0,08445
Integrated: Result 41,63847 144,75735 0,08445 438,0375101385
Generated: Source 57,96085 196,98513 0,02386
Integrated: Result 57,96085 196,98513 0,02386 542,4861449291
Generated: Source 15,67322 157,87204 0,74059
Integrated: Result 15,67322 157,87204 0,74059 270,5440724160
Generated: Source 23,21537 186,58167 0,72547
Integrated: Result 23,21537 186,58167 0,72547 982,8909611172
Generated: Source 12,39713 146,58682 0,71505
Integrated: Result 12,39713 146,58682 0,71505 245,9694175813
Generated: Source 83,02813 137,90890 0,30731
Integrated: Result 83,02813 137,90890 0,30731 394,6990068118
Generated: Source 71,19394 146,63826 0,11750
Integrated: Result 71,19394 146,63826 0,11750 729,3347901677
Generated: Source 1,44755 129,06052 0,18084
```



```
Integrated: Result 1,44755 129,06052 0,18084 225,8486530666
Generated: Source 7,89253 191,95178 0,25887
Integrated: Result 7,89253 191,95178 0,25887 3024,4984112862
Generated: Source 99,46596 165,46932 0,81291
Integrated: Result 99,46596 165,46932 0,81291 154,5630052828
Generated: Source 0,68659 190,55228 0,67055
Integrated: Result 0,68659 190,55228 0,67055 487,3202916138
Generated: Source 20,71525 158,86043 0,83024
Integrated: Result 20,71525 158,86043 0,83024 307,7574711249
```