Лабораторная работа №6

Выполнил работу
Гапанюк Антон Андреевич
Группа: 6204-010302D

Самара, 2025 г.

Оглавление (с ссылками):

Цель работы: разработать приложение, формирующее в одном потоке вычислений набор заданий для интегрирования, а во втором потоке – вычисляющее значения интегралов функций.

Ход работы

Вернуться к оглавлению

Задание 1

Добавим в класс **Functions** метод, возвращающий значение интеграла функции, вычисленное с помощью численного метода.

В качестве параметров метод будет получать ссылку типа **Function** на объект функции, значения левой и правой границы области интегрирования, а также шаг дискретизации.

Если интервал интегрирования выходит за границы области определения функции, метод будет выбрасывать исключение.

Вычисление значения интеграла будет выполняться по методу трапеций. Для этого вся область интегрирования будем разбивать на участки, длина которых (кроме одного) равна шагу дискретизации. На каждом таком участке площадь под кривой, описывающейся заданной функцией, приближается площадью трапеции, две вершины которой расположены на оси абсцисс на границах участка, а ещё две – на кривой в точках границ участка.

```
public static double integrate(Function f, double leftX, double rightX,
double step){
        // Проверка границ интегрирования
        if (leftX < f.getLeftDomainBorder() || rightX >
f.getRightDomainBorder()){
            throw new IllegalArgumentException(
                "Интервал интегрирования [" + leftX + ", " + rightX + "] " +
                "выходит за область определения функции [" +
                f.getLeftDomainBorder() + ", " + f.getRightDomainBorder() +
"]");
        }
```

```
        // Проверка на область определения
        if (leftX >= rightX){
            throw new IllegalArgumentException("Левая граница (" + leftX + ")
должна быть меньше правой (" + rightX + ")");
        }

        double totalValue = 0;
        double currentX = leftX;

        // Проходим по всем шагам
        while (currentX + step <= rightX){
            double f1 = f.getFunctionValue(currentX);
            double f2 = f.getFunctionValue(currentX + step);

            // Площадь трапеции: (f1 + f2) * step / 2
            totalValue += (f1 + f2) * step / 2;
            currentX += step;
        }

        // Обрабатываем последний неполный шаг (если есть)
        if (currentX < rightX){
            double remainingStep = rightX - currentX;
            double f1 = f.getFunctionValue(currentX);
            double f2 = f.getFunctionValue(rightX);
            totalValue += (f1 + f2) * remainingStep / 2;
        }
        return  totalValue;
    }
```

В методе main() проверим работу метода интегрирования. Вычислим интеграл для экспоненты на отрезке от 0 до 1. Определим, какой шаг дискретизации нужен, чтобы рассчитанное значение отличалось от теоретического в 7 знаке после запятой.

Вывод консоли:

Теоретическое значение интеграла: 1.718281828459045

Шаг: 1,000000, Вычисленное значение: 1,8591409142, Ошибка: 0,1408590858

Шаг: 0,100000, Вычисленное значение: 1,7197134914, Ошибка: 0,0014316629

Шаг: 0,010000, Вычисленное значение: 1,7182961475, Ошибка: 0,0000143190

Шаг: 0,001000, Вычисленное значение: 1,7182819716, Ошибка: 0,0000001432

Шаг: 0,000100, Вычисленное значение: 1,7182818299, Ошибка: 0,0000000014

Достигнута точность до 7 знака после запятой

Минимальный шаг для точности 1e-7: 0,000100

Вернуться к оглавлению

Задание 2

Создадим пакет **threads**, в котором будут размещены классы, связанные с потоками.

В пакете **threads** опишим класс **Task**, объект которого должен хранить ссылку на объект интегрируемой функции, границы области интегрирования, шаг дискретизации, а также целочисленное поле, хранящее количество выполняемых заданий. Т.е. объект описывает одно задание.

В главном классе программы опишите метод **nonThread()**, реализующий последовательную (без применения потоков инструкций) версию программы. В методе создадим объект класса **Task** и установим в нём количество выполняемых заданий (минимум 100). Выполним следующие действия:

1. создадим и поместим в объект задания объект логарифмической функции, основание которой является случайной величиной, распределённой равномерно на отрезке от 1 до 10;
2. укажем в объекте задания левую границу области интегрирования (случайно распределена на отрезке от 0 до 100);
3. укажем в объекте задания правую границу области интегрирования (случайно распределена на отрезке от 100 до 200);
4. укажем в объекте задания шаг дискретизации (случайно распределён на отрезке от 0 до 1);
5. выведем в консоль сообщение вида "Source <левая граница> <правая граница> <шаг дискретизации>";
6. вычислим значение интеграла для параметров из объекта задания;
7. выведем в консоль сообщение вида "Result <левая граница> <правая граница> <шаг дискретизации> <результат интегрирования>".

```
package threads;
```

```java
import functions.Function;

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


    // Конструкторы
    public Task(){
        this.tasksCount = 0;
    }

    public Task(int tasksCount){
        this.tasksCount = tasksCount;
    }

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

    public double getLeftBorder() {
        return leftBorder;
    }

    public double getRightBorder() {
        return rightBorder;
    }

    public double getStep() {
        return step;
    }

    public int getTasksCount() {
        return tasksCount;
    }
```

```
    // Сеттеры
    public void setFunction(Function f) {
        this.f = f;
    }

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

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

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

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

Проверим работу метода, вызвав его в методе main().

Начало вывода консоли:

Начало последовательного выполнения 100 заданий:

==========

Source 94,752198 177,642138 0,786051

Result 94,752198 177,642138 0,786051 275,811059

Source 84,760924 169,245971 0,187050

Result 84,760924 169,245971 0,187050 2641,416429

Source 47,621752 170,340750 0,239710

Result 47,621752 170,340750 0,239710 933,227130

Source 99,763737 152,755195 0,965232

Result 99,763737 152,755195 0,965232 237,312676

Source 38,684611 185,367128 0,218039

Result 38,684611 185,367128 0,218039 967,857086

Source 16,053740 172,225965 0,337641

Result 16,053740 172,225965 0,337641 311,866929

Source 64,977001 132,218882 0,939415

Result 64,977001 132,218882 0,939415 194,647855

Source 3,057518 174,653822 0,889270

Result 3,057518 174,653822 0,889270 1524936,513241

Source 11,890035 170,542669 0,687800

Result 11,890035 170,542669 0,687800 305,583891

Source 75,755687 192,903075 0,201462

Result 75,755687 192,903075 0,201462 302,043193

Source 47,537785 101,057580 0,921476

Result 47,537785 101,057580 0,921476 232,031453

Source 98,258128 173,693799 0,705547

Result 98,258128 173,693799 0,705547 180,964584

Source 98,137331 153,359658 0,263189

Result 98,137331 153,359658 0,263189 236,019010

Вернуться к оглавлению

Задание 3

В пакете **threads** создадим два следующих класса:

Класс **SimpleIntegrator** будет реализовывать интерфейс **Runnable**, получать в конструкторе и сохранять в своё поле ссылку на объект типа **Task**, а в методе **run()** в цикле должны решаться задачи, данные для которых берутся из полученного объекта задания, а также выводиться сообщения в консоль.

```
public class SimpleIntegrator implements Runnable {
    private Task task;

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

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

        try {
            double leftBorder, rightBorder, step, result;

            // Ждем данные от генератора
            int attempts = 0;
            while (task.getFunction() == null && attempts < 10) {
                Thread.sleep(100);  // Ждем 100ms
                attempts++;
            }

            if (task.getFunction() == null) {
                System.out.println("Integrator: нет данных для задания " +
i);
                continue;
            }


            // Атомарное чтениие все данных
            synchronized (task) {
                leftBorder = task.getLeftBorder();
                rightBorder = task.getRightBorder();
                step = task.getStep();
            }

            // Вычисляем интеграл (вне блока синхронизации)
```

```
            result = Functions.integrate(task.getFunction(), leftBorder,
rightBorder, step);

            // Вывод результата (в синхронизации для красивого вывода)
            synchronized (task) {
                System.out.printf("Integrator: Result %.6f %.6f %.6f %.6f%n",
                leftBorder, rightBorder, step, result);
                }

                // Небольшая пауза
                Thread.sleep(10);

            } catch (InterruptedException e) {

                System.out.println("Integrator был прерван");
                return;
            } catch (Exception e) {
                System.out.println("Integrator ошибка: " + e.getMessage());
            }
        }

        System.out.println("Integrator завершил работу");
    }
}
```

Класс **SimpleGenerator** будет реализовывать интерфейс **Runnable**, получать в конструкторе и сохранять в своё поле ссылку на объект типа **Task**, а в методе **run()** в цикле должны формироваться задачи и заноситься в полученный объект задания, а также выводиться сообщения в консоль.

```
public class SimpleGenerator implements Runnable {
    private Task task;

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

    @Override
    public void run(){
        Random random = new Random();

        for (int i = 0; i < task.getTasksCount(); i++) {
```

```
            try {
                // Генерируем параметры задания
                double base = 1 + random.nextDouble() * 9;
                Log logFunc = new Log(base);

                double leftBorder = random.nextDouble() * 100;
                double rightBorder = 100 + random.nextDouble() * 100;
                double step = random.nextDouble();

                // Синхронизация на объекте task
                synchronized (task) {
                    // Устанавливаем параметры в объект задания
                    task.setFunction(logFunc);
                    task.setLeftBorder(leftBorder);
                    task.setRightBorder(rightBorder);
                    task.setStep(step);

                    // Вывод информации о сгенерированном задании
                    System.out.printf("Generator: Source %.6f %.6f %.6f%n",
                                      task.getLeftBorder(),
                                      task.getRightBorder(),
                                      task.getStep());
                }

                // Небольшая пауза для наглядности
                Thread.sleep(10);

            } catch (InterruptedException e) {
                System.out.println("Generator был прерван");
                return;
            } catch (Exception e) {
                System.out.println("Generator ошибка: " + e.getMessage());
            }
        }
        System.out.println("Generator завершил работу");
    }
}
```

В главном классе программы создадим метод **simpleThreads()**. Создадим и запустим два потока вычислений, основанных на описанных

классах **SimpleGenerator** и **SimpleIntegrator**. Проверьте работу метода, вызвав его в методе **main()**.

```
public static void simpleThreads() {
        // Создаем объект задания
        Task task = new Task(100);

        System.out.println("Начало многопоточного выполнения " +
task.getTasksCount() + " заданий:");
        System.out.println("==========================================================");

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

        // Устанавливаем приоритеты
        generatorThread.setPriority(Thread.MIN_PRIORITY);
        integratorThread.setPriority(Thread.MIN_PRIORITY);

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

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

        System.out.println("==========================================================");
        System.out.println("Многопоточное выполнение завершено");
    }
```

В процессе проверки будут появляться проблемы, которые необходимо решить для корректной работы потоков. Исправим их.

Проблема 1: NullPointerException

Причина: Интегратор пытался читать данные до того, как генератор их записал.
Решение: Добавим проверку if (task.getFunction() == null) и небольшую паузу.

Проблема 2: Несогласованные данные

Причина: Генератор мог обновить часть данных, пока интегратор читал другую часть.
Решение: добавим synchronized (task) блоки для атомарных операций.

Вывод в консоль:

Начало многопоточного выполнения 100 заданий:

==========

Generator: Source 13,200535 172,663290 0,203863

Generator: Source 85,487396 152,320643 0,500736

Generator: Source 96,673428 142,990159 0,209189

Generator: Source 40,964405 132,966617 0,250103

Generator: Source 11,373569 184,588473 0,564035

Generator: Source 85,456713 136,990992 0,169938

Generator: Source 44,596098 116,496509 0,753111

Generator: Source 53,301098 191,583343 0,274113

Integrator: Result 53,301098 191,583343 0,274113 1336,848148

Generator: Source 21,184878 159,472535 0,192895

Integrator: Result 21,184878 159,472535 0,192895 543,470363

Generator: Source 19,592795 179,594259 0,036227

Integrator: Result 19,592795 179,594259 0,036227 558,370149

Generator: Source 27,299676 175,543415 0,201502

Generator: Source 35,259442 111,606518 0,086309

Integrator: Result 27,299676 175,543415 0,201502 851,176036

Generator: Source 49,082041 112,539760 0,860701

Integrator: Result 49,082041 112,539760 0,860701 139,144759

Generator: Source 89,729021 136,000630 0,457870

Integrator: Result 89,729021 136,000630 0,457870 131,218263

Generator: Source 7,860462 125,883286 0,648983

Integrator: Result 7,860462 125,883286 0,648983 232,174418

Generator: Source 95,332425 171,780408 0,389221

Integrator: Result 95,332425 171,780408 0,389221 1181,318731

Generator: Source 98,535484 174,891994 0,974602

Integrator: Result 95,332425 171,780408 0,389221 1181,318731

Generator: Source 12,581258 173,924175 0,932958

Integrator: Result 12,581258 173,924175 0,932958 1826,228563

Generator: Source 62,785367 115,868801 0,987478

Integrator: Result 62,785367 115,868801 0,987478 105,304297

Generator: Source 60,461513 166,027212 0,877513

Integrator: Result 60,461513 166,027212 0,877513 223,523234

Generator: Source 53,732930 139,239231 0,298881

Integrator: Result 53,732930 139,239231 0,298881 198,547871

Generator: Source 80,883736 100,001116 0,734694

Integrator: Result 80,883736 100,001116 0,734694 61,202458

Generator: Source 73,714201 123,003821 0,487791

Integrator: Result 80,883736 100,001116 0,734694 61,202458

Generator: Source 51,430317 127,595637 0,797100

Integrator: Result 73,714201 123,003821 0,487791 112,359647

Generator: Source 91,789493 199,770830 0,522855

Integrator: Result 51,430317 127,595637 0,797100 154,456185

Generator: Source 65,540526 193,508686 0,519333

Integrator: Result 91,789493 199,770830 0,522855 292,561552

Generator: Source 72,637549 159,634910 0,478832

Integrator: Result 72,637549 159,634910 0,478832 289,728815

Generator: Source 24,273901 184,850068 0,726911

Integrator: Result 24,273901 184,850068 0,726911 550,851687

Generator: Source 73,473370 136,167346 0,176438

Integrator: Result 73,473370 136,167346 0,176438 184,144685

Generator: Source 8,792663 196,724630 0,406777

Integrator: Result 73,473370 136,167346 0,176438 184,144685

Generator: Source 90,630316 147,783490 0,624505

Integrator: Result 90,630316 147,783490 0,624505 220,225806

Integrator: Result 90,630316 147,783490 0,624505 220,225806

Generator: Source 27,703705 198,700391 0,083322

Generator: Source 62,447191 191,753015 0,393348

Integrator: Result 27,703705 198,700391 0,083322 394,041717

Generator: Source 58,400878 111,927330 0,033828

Integrator: Result 62,447191 191,753015 0,393348 370,069319

Generator: Source 60,100843 113,118624 0,298251

Integrator: Result 58,400878 111,927330 0,033828 175,543349

Generator: Source 26,548098 131,672419 0,847228

Integrator: Result 26,548098 131,672419 0,847228 761,296753

Integrator: Result 26,548098 131,672419 0,847228 761,296753

Generator: Source 23,024967 195,788827 0,021982

Generator: Source 92,843974 155,036549 0,080100

Integrator: Result 92,843974 155,036549 0,080100 683,909748

Generator: Source 20,555704 148,070412 0,555243

Integrator: Result 92,843974 155,036549 0,080100 683,909748

Generator: Source 41,208553 198,618955 0,013790

Integrator: Result 20,555704 148,070412 0,555243 338,639912

Generator: Source 46,444677 179,906248 0,583944

Integrator: Result 46,444677 179,906248 0,583944 324,719993

Generator: Source 87,532801 115,894798 0,766585

Integrator: Result 87,532801 115,894798 0,766585 114,855436

Generator: Source 77,880083 194,084026 0,202244

Integrator: Result 77,880083 194,084026 0,202244 919,839042

Generator: Source 53,131456 149,071475 0,264823

Integrator: Result 53,131456 149,071475 0,264823 269,334012

Generator: Source 55,656467 138,681568 0,580223

Integrator: Result 55,656467 138,681568 0,580223 194,950312

Generator: Source 72,039880 175,189344 0,826869

Integrator: Result 72,039880 175,189344 0,826869 291,429907

Generator: Source 67,133806 193,014355 0,465119

Integrator: Result 67,133806 193,014355 0,465119 291,947739

Generator: Source 26,295523 131,232440 0,363845

Integrator: Result 26,295523 131,232440 0,363845 288,117205

Generator: Source 38,626503 184,350854 0,645034

Integrator: Result 38,626503 184,350854 0,645034 423,468261

Generator: Source 28,344636 128,014826 0,451020

Integrator: Result 28,344636 128,014826 0,451020 1762,292361

Generator: Source 68,862980 157,661297 0,753998

Integrator: Result 68,862980 157,661297 0,753998 268,167651

Generator: Source 94,714854 187,608225 0,773788

Integrator: Result 94,714854 187,608225 0,773788 305,795504

Generator: Source 60,156407 145,665547 0,047317

Integrator: Result 60,156407 145,665547 0,047317 184,687137

Generator: Source 80,094267 164,076501 0,702112

Integrator: Result 80,094267 164,076501 0,702112 743,620494

Generator: Source 28,927269 111,445646 0,026102

Integrator: Result 28,927269 111,445646 0,026102 150,605849

Generator: Source 65,957618 132,282685 0,997358

Integrator: Result 65,957618 132,282685 0,997358 161,196734

Generator: Source 2,218352 160,888687 0,039884

Integrator: Result 2,218352 160,888687 0,039884 697,770060

Generator: Source 22,880235 154,534115 0,905422

Integrator: Result 22,880235 154,534115 0,905422 378,808504

Generator: Source 31,842783 147,225636 0,246224

Integrator: Result 31,842783 147,225636 0,246224 248,435697

Generator: Source 90,680347 159,831342 0,140053

Integrator: Result 31,842783 147,225636 0,246224 248,435697

Generator: Source 82,805765 114,931734 0,556227

Integrator: Result 90,680347 159,831342 0,140053 897,441280

Generator: Source 45,530372 195,486411 0,510278

Integrator: Result 82,805765 114,931734 0,556227 71,127972

Generator: Source 77,640603 155,254078 0,444861

Integrator: Result 45,530372 195,486411 0,510278 595,437337

Generator: Source 48,177857 135,876178 0,195840

Integrator: Result 48,177857 135,876178 0,195840 362,409854

Generator: Source 3,299669 170,632049 0,754987

Integrator: Result 3,299669 170,632049 0,754987 739,789874

Generator: Source 92,416626 158,096232 0,924556

Integrator: Result 92,416626 158,096232 0,924556 198,869955

Generator: Source 12,240924 170,276155 0,874012

Integrator: Result 12,240924 170,276155 0,874012 594,079553

Generator: Source 97,948819 119,281092 0,555485

Integrator: Result 97,948819 119,281092 0,555485 61,304941

Generator: Source 74,350627 144,170981 0,912067

Integrator: Result 74,350627 144,170981 0,912067 146,692739

Generator: Source 16,778422 195,356287 0,821844

Integrator: Result 16,778422 195,356287 0,821844 395,542718

Generator: Source 61,342832 187,767810 0,760619

Integrator: Result 61,342832 187,767810 0,760619 424,690673

Generator: Source 47,276914 165,987943 0,237541

Integrator: Result 47,276914 165,987943 0,237541 255,631146

Generator: Source 70,276338 174,632581 0,160416

Integrator: Result 70,276338 174,632581 0,160416 985,146968

Generator: Source 75,061655 155,495511 0,421259

Integrator: Result 75,061655 155,495511 0,421259 227,022100

Generator: Source 17,829687 199,314788 0,157680

Integrator: Result 75,061655 155,495511 0,421259 227,022100

Generator: Source 84,827649 164,493244 0,341990

Integrator: Result 17,829687 199,314788 0,157680 414,752272

Generator: Source 61,974837 137,702948 0,577456

Integrator: Result 84,827649 164,493244 0,341990 269,045862

Integrator: Result 61,974837 137,702948 0,577456 304,260775

Generator: Source 12,273299 152,026069 0,239915

Generator: Source 32,690156 197,132278 0,393005

Integrator: Result 12,273299 152,026069 0,239915 345,168467

Generator: Source 30,260779 152,965462 0,226548

Integrator: Result 32,690156 197,132278 0,393005 610,643791

Generator: Source 78,353332 153,510184 0,120175

Integrator: Result 30,260779 152,965462 0,226548 248,883874

Generator: Source 88,351655 140,430969 0,818052

Integrator: Result 88,351655 140,430969 0,818052 237,758561

Generator: Source 59,454852 197,725200 0,615631

Integrator: Result 59,454852 197,725200 0,615631 608,629680

Generator: Source 49,858360 168,002967 0,905615

Integrator: Result 49,858360 168,002967 0,905615 377,426038

Generator: Source 46,272657 138,604998 0,907869

Integrator: Result 46,272657 138,604998 0,907869 184,470231

Generator: Source 95,552199 151,492326 0,916822

Integrator: Result 95,552199 151,492326 0,916822 23850,532554

Generator: Source 2,148603 134,689101 0,939904

Integrator: Result 2,148603 134,689101 0,939904 240,130074

Generator: Source 88,168425 175,032574 0,171893

Integrator: Result 88,168425 175,032574 0,171893 200,796664

Generator: Source 59,007361 152,081912 0,352549

Integrator: Result 59,007361 152,081912 0,352549 459,075021

Generator: Source 16,849094 131,260208 0,688729

Integrator: Result 16,849094 131,260208 0,688729 214,504107

Generator: Source 16,758357 170,651690 0,687198

Integrator: Result 16,758357 170,651690 0,687198 1299,014005

Generator: Source 65,664282 130,348743 0,940649

Integrator: Result 65,664282 130,348743 0,940649 190,494276

Generator: Source 23,221414 185,841104 0,636992

Integrator: Result 23,221414 185,841104 0,636992 487,375938

Generator: Source 75,352282 137,662162 0,320728

Integrator: Result 75,352282 137,662162 0,320728 158,880847

Generator: Source 16,216691 178,354985 0,225310

Integrator: Result 16,216691 178,354985 0,225310 322,615275

Generator: Source 92,419446 124,793378 0,316648

Integrator: Result 92,419446 124,793378 0,316648 79,214486

Generator: Source 34,459541 148,990883 0,040460

Integrator: Result 34,459541 148,990883 0,040460 325,168056

Generator: Source 24,740239 143,753317 0,649971

Integrator: Result 24,740239 143,753317 0,649971 478,540104

Generator: Source 28,900958 140,878098 0,905497

Integrator: Result 28,900958 140,878098 0,905497 231,179850

Generator: Source 47,798193 159,076271 0,883726

Integrator: Result 47,798193 159,076271 0,883726 357,928772

Generator завершил работу

Integrator: Result 47,798193 159,076271 0,883726 357,928772

Integrator: Result 47,798193 159,076271 0,883726 357,928772

Integrator: Result 47,798193 159,076271 0,883726 357,928772

Integrator: Result 47,798193 159,076271 0,883726 357,928772

Integrator: Result 47,798193 159,076271 0,883726 357,928772

Integrator: Result 47,798193 159,076271 0,883726 357,928772

Integrator: Result 47,798193 159,076271 0,883726 357,928772

Integrator: Result 47,798193 159,076271 0,883726 357,928772

Integrator завершил работу

==========

Многопоточное выполнение завершено

Вернуться к оглавлению

Задание 4

Определим причину того, что не все сгенерированные задания оказываются выполнены интегрирующим потоком.

Для устранения этой проблемы потребуется дополнительный объект, представляющий собой одноместный семафор, различающий операции чтения и записи в защищаемый объект. Реализуем класс **ReadWriteSemaphore.**

```
public class ReadWriteSemaphore {
    private int readers = 0; // Текущее кол-во активных читателей
    private int writers = 0; // Текущее количество активных писателей (0 или
1)
    private int writeRequests = 0; // Кол-во потоков, ожидающих возможность
записи

    private boolean dataReady = false; // Флаг - данные готовы для чтения

    public synchronized void readLock() throws InterruptedException {
        // Пока есть писатели или запросы на запись - ждем
        while (writers > 0 || writeRequests > 0 || !dataReady){
            wait(); // Освобождаем монитор и ждем уведомления
        }
        // Получаем доступ для чтения
        readers++;
    }

    public synchronized void readUnlock() {
        readers--;
        dataReady = false;
        notifyAll(); // Если читателей не осталось, уведомляем ожидающих
писателей
    }

    public synchronized void writeLock() throws InterruptedException {
        // Регистрируем запрос на запись
        writeRequests++;
        while (readers > 0 || writers > 0 || dataReady){ // Ждем, пока не
освободятся все читатели и писатели
            wait(); // Освобождаем монитор и ждем уведомления
        }
```

```
        // Получаем эксклюзивный доступ для записи
        writeRequests--; // Убираем наш запрос из очереди
        writers++; // Становимся активным писателем
    }

    public synchronized void writeUnlock(){
        writers--;
        dataReady = true;
        notifyAll(); // Уведомляем все ожидающие потоки (и читатели, и
писатели)
    }

}
```

В пакете **threads** создадим два класса.

Класс **Generator** будет расширять класс **Thread**, получать в конструкторе и сохранять в свои поля ссылки на объект типа **Task** и на объект семафора, а в методе **run()** должны выполняться те же действия, что и в предыдущей версии генерирующего класса.

```
public class Generator extends Thread {

    private Task task; // Общий объект задания, в который записываются параметры
    private ReadWriteSemaphore semaphore;  // Семафор для синхронизации доступа к
task

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

    @Override
    public void run() {
        Random random = new Random();

        try {
            for (int i = 0; i < task.getTasksCount(); ++i){
                if (Thread.interrupted()){ // Если поток был прерван, выходим из
цикла
```

```
                    throw new InterruptedException();
                }

                // Генерируем параметры задания
                double base = 1 + random.nextDouble() * 9;
                Log logFunc = new Log(base);

                double leftBorder = random.nextDouble() * 100;
                double rightBorder = 100 + random.nextDouble() * 100;
                double step = random.nextDouble();

                // Захват семафора для записи. Блокируем доступ для других писателей
и новых читателей
                semaphore.writeLock();
                try {
                    // Запись данных в общий объект
                    // Атомарно устанавливаем все параметры задания
                    task.setFunction(logFunc);
                    task.setLeftBorder(leftBorder);
                    task.setRightBorder(rightBorder);
                    task.setStep(step);

                    System.out.printf("Generator: Source %.6f %.6f %.6f%n",
                                      task.getLeftBorder(),
                                      task.getRightBorder(),
                                      task.getStep());
                } finally {
                    semaphore.writeUnlock();
                }

                Thread.sleep(10);
            }

        System.out.println("Generator завершил работу. Сгенерировано: " +
task.getTasksCount() + " заданий");

        } catch (InterruptedException e) {
            System.out.println("Generator был прерван");
        } catch (Exception e) {
            System.out.println("Generator ошибка: " + e.getMessage());
```

```
        }
    }
}
```

Класс **Integrator** будет расширять класс **Thread**, получать в конструкторе и сохранять в свои поля ссылки на объект типа Task и на объект семафора, а в методе **run()** будут выполняться те же действия, что и в предыдущей версии интегрирующего класса.

```
public class Integrator extends Thread {
    private Task task; // Общий объект задания, из которого читаются параметры
    private ReadWriteSemaphore semaphore; // Семафор для синхронизации доступа к task

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

    @Override
    public void run(){
        int processedCount = 0; // Счетчик успешно обработанных заданий

        try {
            // Цикл по всем заданиям, которые нужно обработать
            for (int i = 0; i < task.getTasksCount(); i++) {

                // Проверка прерывания потока
                if (Thread.interrupted()) {
                    throw new InterruptedException();
                }

                // Переменные для хранения параметров задания
                double leftBorder, rightBorder, step, result;
                Function function;

                // Захват семафора для чтения. Разрешает параллельное чтение, но
блокирует запись
                semaphore.readLock();
                try {
                    // Чтение параметров из общего объекта
                    // Атомарно читаем все параметры задания
                    leftBorder = task.getLeftBorder();
```

```
                    rightBorder = task.getRightBorder();
                    step = task.getStep();
                    function = task.getFunction();
                } finally { // Код выполнится ВСЕГДА, назависимо от того, что
произошло в try
                    semaphore.readUnlock(); // Освобождаем сразу после чтения, чтобы
не блокировать генератор
                }

                // Проверка наличия данных.
                // Если функция еще не установлена генератором
                if (task.getFunction() == null) {
                    // Ждем пока генератор подготовит данные
                    Thread.sleep(50);
                    continue; // Переходим к следующей итерации
                }

                // Вычисление интеграла
                // Это "тяжелая" операция, выполняем без блокировки семафора
                result = Functions.integrate(function, leftBorder, rightBorder,
step);

                System.out.printf("Integrator: Result %.6f %.6f %.6f %.6f%n",
                    leftBorder, rightBorder, step, result);

                processedCount++; // Увеличиваем счетчик успешных вычислений

                Thread.sleep(10);
            }

            System.out.println("Integrator завершил работу. Обработано: " +
processedCount + " заданий");

        } catch (InterruptedException e) {
            System.out.println("Integrator был прерван. Обработано: " +
processedCount + " заданий");
        } catch (Exception e) {
            System.out.println("Integrator ошибка: " + e.getMessage() + ".
Обработано: " + processedCount + " заданий");
        }
```

```
    }
}
```

В главном классе программы создадидм метод **complicatedThreads().** В нём создадим объект задания, укажем количество выполняемых заданий, создадим и запустим два потока вычислений классов **Generator** и **Integrator**.

```
public static void complicatedThreads(){

        Task task = new Task(100); // Общий объект задания на 100 вычислений
        ReadWriteSemaphore semaphore = new ReadWriteSemaphore(); // Семафор для
управления доступом к объекту task

        System.out.println("Начало многопоточного выполнения с семафором " +
task.getTasksCount() + " заданий:");
        System.out.println("============================================================
=========================");

        Generator generator = new Generator(task, semaphore); // Поток-генератор
заданий (операции ЗАПИСИ)
        Integrator integrator = new Integrator(task, semaphore); // Поток-вычислитель
интегралов (операции ЧТЕНИЯ)

        // Вариант 1: Одинаковые приоритеты (баланс)
        // generator.setPriority(Thread.NORM_PRIORITY);
        // integrator.setPriority(Thread.NORM_PRIORITY);

        // Вариант 2: Приоритет генератору (быстрее генерирует задания)
        // generator.setPriority(Thread.MAX_PRIORITY);
        // integrator.setPriority(Thread.MIN_PRIORITY);

        // Вариант 3: Приоритет интегратору (быстрее вычисляет)
        generator.setPriority(Thread.MIN_PRIORITY);
        integrator.setPriority(Thread.MAX_PRIORITY);

        System.out.println("Запуск потоков Generator и Integrator...");
        generator.start();    // Запускает поток генерации заданий
        integrator.start();   // Запускает поток вычисления интегралов
```

Проверим работу метода, вызвав его в методе **main().**

Начало многопоточного выполнения с семафором 100 заданий:

==========

Запуск потоков Generator и Integrator...

Generator: Source 1,489394 112,221802 0,961075

Integrator: Result 1,489394 112,221802 0,961075 240,225545

Generator: Source 16,858260 132,761826 0,502230

Integrator: Result 16,858260 132,761826 0,502230 234,762797

Generator: Source 80,252713 101,762519 0,586698

Integrator: Result 80,252713 101,762519 0,586698 57,649947

Generator: Source 74,061274 133,662646 0,970177

Integrator: Result 74,061274 133,662646 0,970177 189,995773

Generator: Source 38,165626 126,840703 0,282982

Integrator: Result 38,165626 126,840703 0,282982 204,119875

Generator: Source 44,795232 169,914764 0,608718

Integrator: Result 44,795232 169,914764 0,608718 267,507088

Generator: Source 67,593637 122,152863 0,107982

Integrator: Result 67,593637 122,152863 0,107982 192,213956

Generator: Source 32,887741 164,896309 0,137827

Integrator: Result 32,887741 164,896309 0,137827 481,568098

Generator: Source 56,092867 139,746141 0,607724

Integrator: Result 56,092867 139,746141 0,607724 190,468805

Generator: Source 34,252668 186,235419 0,634072

Integrator: Result 34,252668 186,235419 0,634072 308,862559

Generator: Source 86,904525 166,807858 0,907838

Integrator: Result 86,904525 166,807858 0,907838 307,893490

Generator: Source 11,777314 120,145826 0,573692

Integrator: Result 11,777314 120,145826 0,573692 214,929743

Generator: Source 49,269947 115,238522 0,834054

Integrator: Result 49,269947 115,238522 0,834054 199,313938

Generator: Source 30,225466 188,053679 0,937745

Integrator: Result 30,225466 188,053679 0,937745 2196,025443

Generator: Source 15,206818 130,273894 0,690481

Integrator: Result 15,206818 130,273894 0,690481 275,099114

Generator: Source 20,296948 165,281272 0,133240

Integrator: Result 20,296948 165,281272 0,133240 3487,332206

Generator: Source 64,387184 190,421771 0,574563

Integrator: Result 64,387184 190,421771 0,574563 459,815167

Generator: Source 65,984270 186,553273 0,734147

Integrator: Result 65,984270 186,553273 0,734147 260,326672

Generator: Source 63,757925 195,979067 0,357624

Integrator: Result 63,757925 195,979067 0,357624 1409,350869

Generator: Source 34,367049 148,091081 0,296205

Integrator: Result 34,367049 148,091081 0,296205 504,910134

Generator: Source 33,544262 168,394864 0,709912

Integrator: Result 33,544262 168,394864 0,709912 397,263775

Generator: Source 91,984768 141,060962 0,749420

Integrator: Result 91,984768 141,060962 0,749420 179,514397

Generator: Source 56,276947 107,745020 0,799036

Integrator: Result 56,276947 107,745020 0,799036 214,112054

Generator: Source 76,928566 176,141545 0,122178

Integrator: Result 76,928566 176,141545 0,122178 208,304002

Generator: Source 10,483776 180,583980 0,540717

Integrator: Result 10,483776 180,583980 0,540717 509,055715

Generator: Source 20,792613 183,976347 0,600857

Integrator: Result 20,792613 183,976347 0,600857 397,589230

Generator: Source 35,686005 135,439410 0,530752

Integrator: Result 35,686005 135,439410 0,530752 223,066353

Generator: Source 39,625751 101,276929 0,178863

Integrator: Result 39,625751 101,276929 0,178863 133,904299

Generator: Source 60,340359 168,215276 0,037090

Integrator: Result 60,340359 168,215276 0,037090 295,295924

Generator: Source 38,023990 177,342300 0,398987

Integrator: Result 38,023990 177,342300 0,398987 288,857385

Generator: Source 49,367190 184,768998 0,680946

Integrator: Result 49,367190 184,768998 0,680946 682,905628

Generator: Source 95,504510 160,760226 0,942986

Integrator: Result 95,504510 160,760226 0,942986 234,579091

Generator: Source 52,905463 132,928298 0,685838

Integrator: Result 52,905463 132,928298 0,685838 368,275983

Generator: Source 29,301775 151,056026 0,072311

Integrator: Result 29,301775 151,056026 0,072311 248,446920

Generator: Source 62,935162 101,213367 0,823461

Integrator: Result 62,935162 101,213367 0,823461 80,980708

Generator: Source 52,726397 125,054525 0,173860

Integrator: Result 52,726397 125,054525 0,173860 279,835720

Generator: Source 9,407753 106,530000 0,435318

Integrator: Result 9,407753 106,530000 0,435318 264,834462

Generator: Source 73,227354 172,720751 0,914696

Integrator: Result 73,227354 172,720751 0,914696 264,672857

Generator: Source 52,525342 118,099264 0,158985

Integrator: Result 52,525342 118,099264 0,158985 138,413831

Generator: Source 99,298226 102,245015 0,759606

Integrator: Result 99,298226 102,245015 0,759606 6,507138

Generator: Source 52,809120 189,889407 0,237948

Integrator: Result 52,809120 189,889407 0,237948 317,638218

Generator: Source 50,368032 199,138021 0,855426

Integrator: Result 50,368032 199,138021 0,855426 422,889715

Generator: Source 25,597756 198,815556 0,543894

Integrator: Result 25,597756 198,815556 0,543894 528,854032

Generator: Source 88,817317 149,101588 0,536033

Integrator: Result 88,817317 149,101588 0,536033 271,824543

Generator: Source 11,616337 110,972884 0,690375

Integrator: Result 11,616337 110,972884 0,690375 402,489916

Generator: Source 18,260103 159,440650 0,951907

Integrator: Result 18,260103 159,440650 0,951907 381,543893

Generator: Source 20,256491 123,168833 0,866749

Integrator: Result 20,256491 123,168833 0,866749 216,251930

Generator: Source 17,232347 138,782116 0,578174

Integrator: Result 17,232347 138,782116 0,578174 2070,956567

Generator: Source 27,208320 152,020045 0,271298

Integrator: Result 27,208320 152,020045 0,271298 318,981835

Generator: Source 2,427509 119,866272 0,605031

Integrator: Result 2,427509 119,866272 0,605031 282,962606

Generator: Source 33,084767 167,879616 0,272443

Integrator: Result 33,084767 167,879616 0,272443 301,972244

Generator: Source 40,619856 135,176293 0,464471

Integrator: Result 40,619856 135,176293 0,464471 271,099797

Generator: Source 14,679032 195,860283 0,676945

Integrator: Result 14,679032 195,860283 0,676945 419,082404

Generator: Source 53,416154 180,171525 0,997193

Integrator: Result 53,416154 180,171525 0,997193 302,674307

Generator: Source 25,495669 131,219205 0,734013

Integrator: Result 25,495669 131,219205 0,734013 227,955964

Generator: Source 83,976343 123,890150 0,013627

Integrator: Result 83,976343 123,890150 0,013627 80,551109

Generator: Source 27,195998 141,369917 0,754214

Integrator: Result 27,195998 141,369917 0,754214 337,834279

Generator: Source 19,843831 151,686863 0,252966

Integrator: Result 19,843831 151,686863 0,252966 364,456812

Generator: Source 58,598449 151,787136 0,472103

Integrator: Result 58,598449 151,787136 0,472103 381,831584

Generator: Source 91,026027 149,871175 0,994109

Integrator: Result 91,026027 149,871175 0,994109 132,633894

Generator: Source 73,874452 138,069977 0,531462

Integrator: Result 73,874452 138,069977 0,531462 130,326092

Generator: Source 87,465763 167,011991 0,822435

Integrator: Result 87,465763 167,011991 0,822435 168,351191

Generator: Source 6,580514 171,429276 0,722081

Integrator: Result 6,580514 171,429276 0,722081 312,351165

Generator: Source 65,044186 111,674055 0,306303

Integrator: Result 65,044186 111,674055 0,306303 108,770584

Generator: Source 41,167108 189,864596 0,846954

Integrator: Result 41,167108 189,864596 0,846954 311,127117

Generator: Source 34,255790 171,351923 0,587184

Integrator: Result 34,255790 171,351923 0,587184 337,373839

Generator: Source 98,604987 192,296544 0,348823

Integrator: Result 98,604987 192,296544 0,348823 210,565769

Generator: Source 50,696171 162,367050 0,137336

Integrator: Result 50,696171 162,367050 0,137336 347,616740

Generator: Source 67,769555 177,920057 0,074966

Integrator: Result 67,769555 177,920057 0,074966 433,154298

Generator: Source 51,503169 145,493664 0,188934

Integrator: Result 51,503169 145,493664 0,188934 255,302553

Generator: Source 39,223925 178,044240 0,809136

Integrator: Result 39,223925 178,044240 0,809136 412,056054

Generator: Source 6,892000 167,078991 0,588542

Integrator: Result 6,892000 167,078991 0,588542 304,612519

Generator: Source 23,041277 152,684427 0,449552

Integrator: Result 23,041277 152,684427 0,449552 251,066586

Generator: Source 80,924995 110,423527 0,080853

Integrator: Result 80,924995 110,423527 0,080853 69,784552

Generator: Source 57,307152 149,172026 0,961858

Integrator: Result 57,307152 149,172026 0,961858 199,352828

Generator: Source 20,656608 167,186906 0,275594

Integrator: Result 20,656608 167,186906 0,275594 962,213257

Generator: Source 38,942301 145,047851 0,485953

Integrator: Result 38,942301 145,047851 0,485953 610,692770

Generator: Source 17,794262 163,585063 0,801503

Integrator: Result 17,794262 163,585063 0,801503 278,100503

Generator: Source 33,486713 122,504500 0,225593

Integrator: Result 33,486713 122,504500 0,225593 221,863801

Generator: Source 12,015716 101,144257 0,401286

Integrator: Result 12,015716 101,144257 0,401286 9781,313554

Generator: Source 45,485461 146,643444 0,999677

Integrator: Result 45,485461 146,643444 0,999677 540,661847

Generator: Source 30,083532 196,777617 0,971459

Integrator: Result 30,083532 196,777617 0,971459 1505,798631

Generator: Source 56,116961 144,812427 0,584121

Integrator: Result 56,116961 144,812427 0,584121 178,360136

Generator: Source 79,897303 131,949820 0,800977

Integrator: Result 79,897303 131,949820 0,800977 157,784058

Generator: Source 59,704305 109,288822 0,223226

Integrator: Result 59,704305 109,288822 0,223226 116,849125

Generator: Source 43,624236 145,695521 0,512573

Integrator: Result 43,624236 145,695521 0,512573 354,248777

Generator: Source 55,750263 170,574566 0,939450

Integrator: Result 55,750263 170,574566 0,939450 1156,416766

Generator: Source 39,048737 131,257181 0,460685

Integrator: Result 39,048737 131,257181 0,460685 207,158945

Generator: Source 78,136602 113,568263 0,520482

Integrator: Result 78,136602 113,568263 0,520482 185,137964

Generator: Source 43,281610 121,107893 0,589904

Integrator: Result 43,281610 121,107893 0,589904 176,079056

Generator: Source 68,128332 133,880150 0,312534

Integrator: Result 68,128332 133,880150 0,312534 225,077324

Generator: Source 97,969506 170,124855 0,007838

Integrator: Result 97,969506 170,124855 0,007838 181,704507

Generator: Source 22,823501 156,555474 0,198749

Integrator: Result 22,823501 156,555474 0,198749 489,011123

Generator: Source 40,145237 171,679084 0,776294

Integrator: Result 40,145237 171,679084 0,776294 275,384254

Generator: Source 75,689188 164,471967 0,391640

Integrator: Result 75,689188 164,471967 0,391640 253,014364

Generator: Source 27,859584 166,407039 0,674692

Integrator: Result 27,859584 166,407039 0,674692 858,982981

Generator: Source 90,824287 108,677002 0,918453

Integrator: Result 90,824287 108,677002 0,918453 39,827155

Generator: Source 54,735231 128,429729 0,939611

Integrator: Result 54,735231 128,429729 0,939611 264,979125

Generator: Source 77,589738 197,930079 0,928378

Integrator: Result 77,589738 197,930079 0,928378 1294,101385

Generator: Source 72,583425 138,784978 0,234386

Integrator: Result 72,583425 138,784978 0,234386 518,836206

Generator завершил работу. Сгенерировано: 100 заданий

Integrator завершил работу. Обработано: 100 заданий

Вернуться к оглавлению

Попробуем запускать программу, изменяя приоритеты потоков перед их запуском. Также сделаем так, чтобы после создания потоков основной поток программы выжидал 50 миллисекунд, после чего прерывал работу потоков путём вызова метода **interrupt().**

```
        try {
            System.out.println("Главный поток ждет 50ms...");
            Thread.sleep(50);  // Главный поток "засыпает" на 50 миллисекунд
        } catch (InterruptedException e) {
            System.out.println("Главный поток был прерван во время ожидания");
        }

        // ПРЕРЫВАЕМ РАБОТУ ПОТОКОВ
        System.out.println("Прерывание потоков Generator и Integrator...");
        generator.interrupt();   // Устанавливает флаг прерывания в потоке Generator
        integrator.interrupt();  // Устанавливает флаг прерывания в потоке Integrator

        // ИНФОРМАЦИОННОЕ СООБЩЕНИЕ
        System.out.println("Потоки прерваны главным потоком через 50ms работы");

        // ОЖИДАЕМ КОРРЕКТНОГО ЗАВЕРШЕНИЯ ПОТОКОВ
        try {
            System.out.println("Ожидание завершения потоков (таймаут 1 секунда)...");

            // join(1000) - ждем завершения потока до 1 секунды
            generator.join(1000);   // Ждем завершения Generator
            integrator.join(1000);  // Ждем завершения Integrator

        } catch (InterruptedException e) {
            System.out.println("Главный поток был прерван во время ожидания
завершения потоков");
        }

        // ПРОВЕРЯЕМ СТАТУС ПОТОКОВ ПОСЛЕ ПРЕРЫВАНИЯ

        if (generator.isAlive()) {
```

```
            System.out.println("Generator все еще выполняется после прерывания и
таймаута");
        } else {
            System.out.println("Generator корректно завершился");
        }

        if (integrator.isAlive()) {
            System.out.println("Integrator все еще выполняется после прерывания и
таймаута");
        } else {
            System.out.println("Integrator корректно завершился");
        }

        System.out.println("============================================================
=========================");
        System.out.println("Многопоточное выполнение с семафором завершено");

    }
```

Вывод в консоль:

Начало многопоточного выполнения с семафором 100 заданий:

==========

Запуск потоков Generator и Integrator...

Главный поток ждет 50ms...

Generator: Source 64,328275 174,312989 0,237544

Integrator: Result 64,328275 174,312989 0,237544 5331,550585

Generator: Source 36,698817 161,691215 0,858529

Integrator: Result 36,698817 161,691215 0,858529 330,351738

Generator: Source 6,207735 158,585605 0,670514

Integrator: Result 6,207735 158,585605 0,670514 574,746457

Generator: Source 36,781387 195,454567 0,412117

Integrator: Result 36,781387 195,454567 0,412117 414,007349

Прерывание потоков Generator и Integrator...

Потоки прерваны главным потоком через 50ms работы

Generator был прерван

Ожидание завершения потоков (таймаут 1 секунда)...

Integrator был прерван. Обработано: 4 заданий

Generator корректно завершился

Integrator корректно завершился

==========

Многопоточное выполнение с семафором завершено