Лабораторная работа №6

Выполнил: Магера Никита Алексеевич

Студент группы 6203-010302D

Ход выполнения

Задание 1

В 1 задании необходимо в классе FunctionPoint добавить метод Integral, реализующий численное интегрирование методом трапеций. Метод должен принимать функцию, границы интегрирования и шаг дискретизации, проверять корректность интервала и вычислять интеграл. Затем протестировать метод на функции e^x от 0 до 1, определить шаг для точности 7 знаков после запятой. Теоретическое значение: e-1 ≈ 1.718281828. Экспериментально установлено, что для достижения точности в 7 знаке после запятой требуется шаг дискретизации порядка 0.001. (Рис.1-2)

```
public static double Integral(Function function, double left, double right, double step) {  1 usage  new *
    // не выходим ли за границы области определения
    if (left < function.getLeftDomainBorder() || right > function.getRightDomainBorder()) {
        throw new IllegalArgumentException("Интервал интегрирования выходит за границы области определения функции");
    }
    // вычисление интеграла методом трапеций
    double integral = 0.0;
    double need_x = left;
    double need_y = function.getFunctionValue(need_x);
    // идем пока не дойдем до правой границы
    while (need_x < right) {
        // следующая точка
        double next_x = need_x + step;
        if (next_x > right) {
            next_x = right;
        }
        // значение функции в следующей точке
        double next_y = function.getFunctionValue(next_x);
        // площадь трапеции: (основание1 + основание2) * высота / 2
        double trapezoid = (need_y + next_y) * (next_x - need_x) / 2.0;
        integral += trapezoid;
        // переходим к следующему отрезку
        need_x = next_x;
        need_y = next_y;
    }
    return integral;
}
```

Рис. 1

```
проверка интегрирования
теоретическое значение ∫e^x dx от 0 до 1 = 1.718281828459045
Шаг: 1.0  Результат: 1.8591409142295225, Ошибка: 0.14085908577047745
не удовлетворяет условию
Шаг: 0.1  Результат: 1.7197134913893144, Ошибка: 0.0014316629302693062
не удовлетворяет условию
Шаг: 0.01  Результат: 1.7182961474504181, Ошибка: 1.431899137305237E-5
не удовлетворяет условию
Шаг: 0.001  Результат: 1.7182819716491948, Ошибка: 1.4319014973729338E-7
не удовлетворяет условию
Шаг: 1.0E-4  Результат: 1.7182818298909435, Ошибка: 1.4318983776462346E-9
7 знаков, результат удовлетворяет условию
```

Рис. 2

Задание 2

В 2 задании нужно создать класс `Task` в пакете `threads`, который будет хранить параметры задания для вычисления интеграла. Класс должен содержать поля для функции (логарифм), левой и правой границ интегрирования, шага и результата вычислений. Функция-логарифм генерируется со случайным основанием от 1 до 10, левая граница — от 0 до 100, правая граница — от 100 до 200, шаг — от 0 до 1. Затем реализуется метод `nonThread()`, который последовательно выполняет 100 таких заданий. Для каждого задания метод вычисляет определенный интеграл функции логарифма на заданном интервале с указанным шагом, после чего выводит параметры задания в формате "Source <левая граница> <правая граница> <шаг>" и результат в формате "Result <левая граница> <правая граница> <шаг> <значение интеграла>". Все вычисления выполняются в одном потоке, что позволяет оценить базовое время выполнения без использования многопоточности.(Рис.3-4)

```
// последовательная версия программы
public static void nonThread() {  1 usage  new *
    //объект Task с 100 заданиями
    Task task = new Task( tasksCount: 100);
    for (int i = 0; i < task.getTasksCount(); i++) {
        // логарифм со случайным основанием от 1 до 10
        double log = 1 + Math.random() * 9;
        task.setFunction(new functions.basic.Log(log));
        // левая граница: 0 - 100
        task.setLeftX(Math.random() * 100);
        // правая граница: 100 - 200
        task.setRightX(100 + Math.random() * 100);
        // шаг 0 1
        task.setStep(Math.random());
        // вывод
        System.out.println("Source " + task.getLeftX() + " " + task.getRightX() + " " + task.getStep());
        try {
            // вычисление интеграла
            double integral = Functions.Integral(task.getFunction(), task.getLeftX(), task.getRightX(), task.getStep());
            System.out.println("Result " + task.getLeftX() + " " + task.getRightX() + " " + task.getStep() + " " + integral);

        } catch (Exception e) {
            System.out.println("ошибка интегрирования: " + e.getMessage());
        }
    }
    System.out.println("Выполнено " + task.getTasksCount() + " заданий.");
}
```

Рис.3

```
Result 51.33388442711282 180.4440588746545 0.2844056267499938 304.363069499722
Source 35.56841401314376 174.83462554375973 0.617394413035064
Result 35.56841401314376 174.83462554375973 0.617394413035064 300.8552666627543
Source 2.4219427518801018 190.25802920321667 0.7639419325006767
Result 2.4219427518801018 190.25802920321667 0.7639419325006767 660.9971759185038
Source 91.83043227299463 181.06590386972258 0.9330951896610484
Result 91.83043227299463 181.06590386972258 0.9330951896610484 222.93885799715076
Source 99.88633324825496 104.56632722269293 0.3164963210514423
Result 99.88633324825496 104.56632722269293 0.3164963210514423 9.98146125750764
Source 45.7593116834891 183.47287702886473 0.04314310321210768
Result 45.7593116834891 183.47287702886473 0.04314310321210768 360.1503199377836
Source 56.11658541290217 104.8878910329398 0.7709532539707321
Result 56.11658541290217 104.8878910329398 0.7709532539707321 921.0879642772388
Source 13.530342954668518 144.81226771025308 0.5601823502081682
Result 13.530342954668518 144.81226771025308 0.5601823502081682 260.7457640154454
Source 27.176062608031813 197.8967335598192 0.43253434029872395
Result 27.176062608031813 197.8967335598192 0.43253434029872395 442.52515231843915
Source 49.06694740782002 160.9972431231568 0.8098433418193903
Result 49.06694740782002 160.9972431231568 0.8098433418193903 619.4697825415116
Source 77.13617110464482 166.06976070185468 0.8474933752420433
Result 77.13617110464482 166.06976070185468 0.8474933752420433 524.7554883377562
Source 57.66993242790125 183.03735709513796 0.530282706856159
Result 57.66993242790125 183.03735709513796 0.530282706856159 394.62951386288927
Source 23.910809062809857 116.48524671039686 0.2822280871463688
Result 23.910809062809857 116.48524671039686 0.2822280871463688 186.93745760800073
Source 85.09228355834763 101.38695004384036 0.7568431475074328
Result 85.09228355834763 101.38695004384036 0.7568431475074328 45.86242962261023
Source 10.731668696844988 162.53756035369915 0.4207146181235708
Result 10.731668696844988 162.53756035369915 0.4207146181235708 596.216626093298
Source 38.70015659128998 139.96760428443264 0.5207390037033783
Result 38.70015659128998 139.96760428443264 0.5207390037033783 204.27746729173015
Source 72.4050529295221 198.33376708027146 0.03417085684227317
Result 72.4050529295221 198.33376708027146 0.03417085684227317 727.3809385953401
Source 22.360667813630243 132.13031360195075 0.24067407115692263
Result 22.360667813630243 132.13031360195075 0.24067407115692263 243.25308037905492
Source 90.49081009741266 150.5377581013842 0.6268551480473792
Result 90.49081009741266 150.5377581013842 0.6268551480473792 145.3164772012388
Source 91.69711372450631 175.8229638054374 0.9362977897304738
Result 91.69711372450631 175.8229638054374 0.9362977897304738 1847.4492192167158
Выполнено 100 заданий.
```

Рис.4

Задание 3

В третьем задании должны быть созданы классы SimpleGenerator и SimpleIntegrator, реализующие интерфейс Runnable. SimpleGenerator формирует задания в цикле, занося параметры в объект Task и выводя сообщения "Source". SimpleIntegrator в цикле извлекает данные из Task, вычисляет интегралы через Functions.Integral и выводит "Result". При тестировании выявлены проблемы многопоточности: NullPointerException возникал при попытке интегратора получить данные до их генерации, также наблюдалось смешивание параметров из разных заданий. Для устранения добавлена проверка на null с ожиданием и блоки синхронизации synchronized(task) в методах run(). Проведены эксперименты с приоритетами потоков: при высоком приоритете генератора задания создавались быстрее, чем обрабатывались; при высоком приоритете интегратора учащались попытки чтения неготовых данных. После синхронизации исключения исчезли, данные передаются целостно, потоки корректно завершают все 100 заданий.(Рис.5-7)

```
public class SimpleGenerator implements Runnable {  1 usage  new *
    private Task task;  13 usages
    public SimpleGenerator(Task task) {  1 usage  new *
        this.task = task;
    }

    public void run() {  new *
        for (int i = 0; i < task.getTasksCount(); i++) {
            // synchronized
            synchronized (task) {
                // генерируем задание
                double base = 1 + Math.random() * 9;
                task.setFunction(new Log(base));
                task.setLeftX(Math.random() * 100);
                task.setRightX(100 + Math.random() * 100);
                task.setStep(Math.random());
                System.out.println("Source " + task.getLeftX() + " " + task.getRightX() + " " + task.getStep());

                // будим интегратор
                task.notifyAll();

                // ждем пока интегратор обработает
                try {
                    task.wait();
                } catch (InterruptedException e) {
                    break;
                }
            }

            try {
                Thread.sleep( millis: 10);
            } catch (InterruptedException e) {
                break;
            }
        }
        System.out.println("Выполнено " + task.getTasksCount() + " заданий.");
    }
}
```

Рис.5

```java
public class SimpleIntegrator implements Runnable {  1 usage  new *
    private Task task;  11 usages
    public SimpleIntegrator(Task task) { this.task = task; }

    public void run() {  new *
        for (int i = 0; i < task.getTasksCount(); i++) {
            // synchronized
            synchronized (task) {
                try {
                    // ждем пока генератор создаст задание
                    while (task.getFunction() == null) {
                        task.wait();
                    }

                    // берем текущие данные
                    double leftX = task.getLeftX();
                    double rightX = task.getRightX();
                    double step = task.getStep();

                    // вычисляем интеграл
                    double integral = Functions.Integral(task.getFunction(), leftX, rightX, step);

                    // выводим результат
                    System.out.println("Result " + leftX + " " + rightX + " " + step + " " + integral);

                    // очищаем и будим генератор
                    task.setFunction(null);
                    task.notifyAll();

                } catch (Exception e) {
                    System.out.println("Integrator error: " + e.getMessage());
                }
            }

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

Рис.6

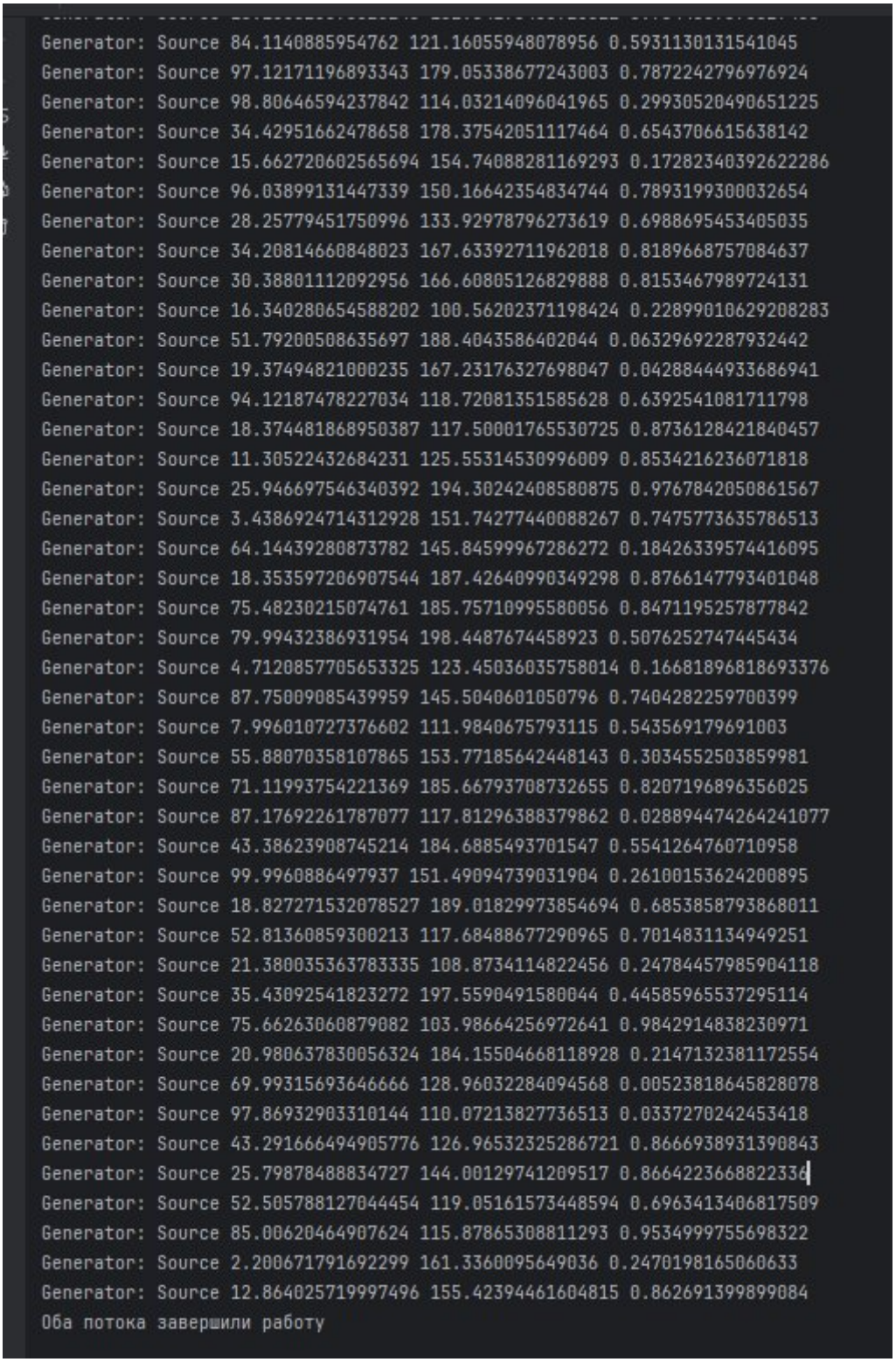
Generator: Source 84.1140885954762 121.16055948078956 0.5931130131541045
Generator: Source 97.12171196893343 179.05338677243003 0.7872242796976924
Generator: Source 98.80646594237842 114.03214096041965 0.29930520490651225
Generator: Source 34.42951662478658 178.37542051117464 0.6543706615638142
Generator: Source 15.662720602565694 154.74088281169293 0.17282340392622286
Generator: Source 96.03899131447339 150.16642354834744 0.7893199300032654
Generator: Source 28.25779451750996 133.92978796273619 0.6988695453405035
Generator: Source 34.20814660848023 167.63392711962018 0.8189668757084637
Generator: Source 30.38801112092956 166.60805126829888 0.8153467989724131
Generator: Source 16.340280654588202 100.56202371198424 0.22899010629208283
Generator: Source 51.79200508635697 188.4043586402044 0.06329692287932442
Generator: Source 19.37494821000235 167.23176327698047 0.04288444933686941
Generator: Source 94.12187478227034 118.72081351585628 0.6392541081711798
Generator: Source 18.374481868950387 117.50001765530725 0.8736128421840457
Generator: Source 11.30522432684231 125.55314530996009 0.8534216236071818
Generator: Source 25.946697546340392 194.30242408580875 0.9767842050861567
Generator: Source 3.4386924714312928 151.74277440088267 0.7475773635786513
Generator: Source 64.14439280873782 145.84599967286272 0.18426339574416095
Generator: Source 18.353597206907544 187.42640990349298 0.8766147793401048
Generator: Source 75.48230215074761 185.75710995580056 0.8471195257877842
Generator: Source 79.99432386931954 198.4487674458923 0.5076252747445434
Generator: Source 4.7120857705653325 123.45036035758014 0.16681896818693376
Generator: Source 87.75009085439959 145.5040601050796 0.7404282259700399
Generator: Source 7.996010727376602 111.9840675793115 0.543569179691003
Generator: Source 55.88070358107865 153.77185642448143 0.3034552503859981
Generator: Source 71.11993754221369 185.66793708732655 0.8207196896356025
Generator: Source 87.17692261787077 117.81296388379862 0.028894474264241077
Generator: Source 43.38623908745214 184.6885493701547 0.5541264760710958
Generator: Source 99.9960886497937 151.49094739031904 0.26100153624200895
Generator: Source 18.827271532078527 189.01829973854694 0.6853858793868011
Generator: Source 52.81360859300213 117.68488677290965 0.7014831134949251
Generator: Source 21.380035363783335 108.8734114822456 0.24784457985904118
Generator: Source 35.43092541823272 197.5590491580044 0.44585965537295114
Generator: Source 75.66263060879082 103.98664256972641 0.9842914838230971
Generator: Source 20.980637830056324 184.15504668118928 0.2147132381172554
Generator: Source 69.99315693646666 128.96032284094568 0.00523818645828078
Generator: Source 97.86932903310144 110.07213827736513 0.0337270242453418
Generator: Source 43.291666494905776 126.96532325286721 0.8666938931390843
Generator: Source 25.79878488834727 144.00129741209517 0.866422366882233
Generator: Source 52.505788127044454 119.05161573448594 0.6963413406817509
Generator: Source 85.00620464907624 115.87865308811293 0.9534999755698322
Generator: Source 2.200671791692299 161.3360095649036 0.2470198165060633
Generator: Source 12.864025719997496 155.42394461604815 0.862691399899084
Оба потока завершили работу

Рис.7

Задание 4

В 4 задании реализованы два потока Generator и Integrator, использующие общий семафор java.util.concurrent.Semaphore для синхронизации доступа к объекту Task. Generator генерирует случайные параметры задач, Integrator вычисляет интегралы. Семафор с одним разрешением обеспечивает взаимоисключающий доступ: пока один поток работает с данными, второй ожидает. Программа отрабатывает минимум 100 заданий с гарантированной синхронизацией без потерь данных.(Рис.8-9)

```java
public class Generator extends Thread {  2 usages  new *
    private final Task task;  8 usages
    private final Semaphore semaphore; // семафор для синхронизации доступа  3 usages

    public Generator(Task task, Semaphore semaphore) {  1 usage  new *
        this.task = task;
        this.semaphore = semaphore;
    }
    // основной метод потока
    public void run() {  new *
        Random random = new Random();// генератор случайных чисел
        int count = 0;//  счетчик для  100 задач

        try {
            while (count < 100 && !isInterrupted()) {
                try {
                    // захватываем семафор
                    semaphore.acquire();

                    // генерация случайных параметров для задачи:
                    double a = random.nextDouble() * 9 + 1;
                    task.setFunction(new Log(a));
                    task.setLeftX(random.nextDouble() * 100);
                    task.setRightX(100 + random.nextDouble() * 100);
                    task.setStep(random.nextDouble());

                    System.out.println("Source " + task.getLeftX() + " " + task.getRightX() + " " + task.getStep());
                    count++;

                } finally {
                    // всегда освобождаем семафор
                    semaphore.release();
                }
                // пауза между генерациями
                try {
                    sleep( millis: 1);
                } catch (InterruptedException e) {
                    if (count < 100) {
                        System.out.println("Generator прерван после " + count + " задач");
                    }
                    break;
                }
            }
        } catch (Exception e) {
            System.out.println("Generator error: " + e.getMessage());
        }

        System.out.println("Generator: " + count + "/100 задач");
    }
}
```

Рис.8

```java
public class Integrator extends Thread {  2 usages  new *
    private final Task task;  7 usages
    private final Semaphore semaphore; //семафор для синхронизации доступа  3 usages
    public Integrator(Task task, Semaphore semaphore) {  1 usage  new *
        this.task = task;
        this.semaphore = semaphore;
    }
    // основной метод потока
    public void run() {  new *
        int task_count = 0;//  счетчик обработанных задач

        try {
            // работает пока поток не прервали
            while (task_count < 100 && !isInterrupted()) {
                try {
                    // захватываем семафор
                    semaphore.acquire();

                    if (task.getFunction() != null) {
                        // чтение параметров задачи, сгенерированных Generator
                        double leftX = task.getLeftX();
                        double rightX = task.getRightX();
                        double step = task.getStep();
                        // вычисление интеграла функции на заданном интервале
                        double integral = Functions.Integral(task.getFunction(), leftX, rightX, step);
                        // вывод результата вычислений
                        System.out.println("Result " + leftX + " " + rightX + " " + step + " " + integral);

                        task.setFunction(null); // очищаем
                        task_count++;
                    }

                } finally {
                    semaphore.release();
                }
                // пауза между вычислениями
                try {
                    sleep( millis: 1);
                } catch (InterruptedException e) {
                    if (task_count < 100) {
                        System.out.println("Integrator прерван после " + task_count + " задач");
                    }
                    break;
                }
            }
        } catch (Exception e) {
            System.out.println("Integrator error: " + e.getMessage());
        }

        System.out.println("Integrator: " + task_count + "/100 задач");
    }
```

Рис.9