

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

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

ОТЧЕТ ПО
ЛАБОРАТОРНОЙ РАБОТЕ № 6

«Работа с потоками»

по курсу
Объектно-ориентированное программирование

Выполнила: Гонтарь Анастасия Вячеславовна,
студент группы 6203-010302D

Содержание

<u>Задание №1</u>	3
<u>Задание №2</u>	4
<u>Задание №3</u>	6
<u>Задание №4</u>	8

Задание №1

Для выполнения этого задания я написала в классе Functions метод integration, который возвращает значение интеграла функции на промежутке с заданным шагом дискретизации.

Вычислила интеграл при помощи площади трапеции. Чтобы при расчете не выйти за правую границу интегрирования, я нахожу минимальное между правой границей текущего отрезка и правой границей области интегрирования

```
public static double integration(Function func, double leftInteg, double rightInteg, double samplingStep) { 5 usages
    // проверка
    double leftFunc = func.getLeftDomainBorder();
    double rightFunc = func.getRightDomainBorder();
    if (leftInteg < leftFunc || rightInteg > rightFunc)
        throw new IllegalArgumentException("область границы интегрирования выходит за область определения функции");

    // вычисление интеграла
    double current = leftInteg;
    double res = 0;
    while (current < rightInteg){
        double next = Math.min(rightInteg, current+samplingStep);

        double a = func.getFunctionValue(current);
        double b = func.getFunctionValue(next);
        double h = next - current;

        res+= ((a+b)*h)/2;

        current = next;
    }
    return res;
}
```

Тестирований в Main

```
public static void testIntegration(){ 1 usage
    double step =1;
    double theoreticalResult = Math.E - 1;
    double error;

    while (true){
        double result = Functions.integration(new Exp(), leftInteg: 0, rightInteg: 1, step);
        error = Math.abs(theoreticalResult-result);

        if (error < 1e-7)
            break;

        step /= 2;    // уменьшаем шаг
    }
    System.out.println("шаг = " + step);
}
```

Получился такой шаг дискретизации, при котором разница между теоретическим и рассчитанным значением не менее 7 знаков после запятой

```
шаг = 4.8828125E-4
```

Задание №2

Далее я создала пакет threads и в нем класс Task, который хранит информацию о задании: функцию интегрирования, левую и правую границы интегрирования, количество шагов дискретизации и количество заданий.

```
public class Task { 15 usages
    private double leftBorder; 3 usages
    private double rightBorder; 3 usages
    private double samplingStep; 3 usages
    private Function function; 3 usages
    private int countTask; 3 usages

    public Task(Function func, double left, double right, double step, int count) { 15 usages
        this.function = func;
        this.leftBorder = left;
        this.rightBorder = right;
        this.samplingStep = step;
        this.countTask = count;
    }
}
```

Для тестирования в Main я написала метод nonThread, который последовательно создает задачу, на основе рандомно определенных параметров, и решает ее

```
public static void nonThread(){ 1 usage
    Task task = new Task( func: null, left: 0, right: 0, step: 0, count: 100);

    for (int i = 0; i < task.getTaskCount(); i++){
        // работа с рандом
        double base = Math.random() * 9 + 1; // основание на отрезке от 1 до 10
        Function log = new Log(base);
        double leftB = Math.random() * 100; // левая граница от 0 до 100
        double rightB = Math.random() *100+100; // правая граница от 100 до 200
        double step = Math.random(); // шаг дискретизации от 0 до 1

        System.out.printf("Source %.2f %.2f %.2f", leftB, rightB, step);

        // устанавливаем значения
        task.setFunc(log);
        task.setLeftBorder(leftB);
        task.setRightBorder(rightB);
        task.setStep(step);

        // вычисляем интеграл
        double result = Functions.integration(log, leftB, rightB, step);
        System.out.printf("\nResult %.2f %.2f %.2f\n", leftB, rightB, step,result);
    }
}
```

Тестирование задания 2:

Source 33,81 196,07 0,18
Result 33,81 196,07 0,18 409,88

Source 83,23 158,44 0,32
Result 83,23 158,44 0,32 185,82

Source 44,78 111,76 0,05
Result 44,78 111,76 0,05 675,29

Source 80,26 166,29 0,52
Result 80,26 166,29 0,52 299,77

Source 50,25 109,37 0,22
Result 50,25 109,37 0,22 172,83

Source 47,76 118,72 0,59
Result 47,76 118,72 0,59 159,12

Source 31,68 135,75 0,47
Result 31,68 135,75 0,47 236,15

Задание №3

В пакете threads были реализованы два класса, реализующие интерфейс Runnable, SimpleGenerator и SimpleIntegrator.

SimpleGenerator создает задания и записывает их в объект Task, а SimpleIntegrator считывает эти задания и выполняет вычисление интегралов. Для устранения проблем была внедрена синхронизация доступа к объекту Task. После этого оба потока работают согласованно: генератор блокирует объект на время записи нового задания, а интегратор на время его чтения и обработки

```
public class SimpleGenerator implements Runnable { 1 usage
    private Task task; 10 usages

    public SimpleGenerator(Task task){ 5 usages
        this.task= task;
    }
    @Override
    public void run() {
        try {
            for (int i = 0; i < task.getTaskCount(); i++) {
                double base = Math.random() * 9 + 1;
                Function log = new Log(base);
                double leftB = Math.random() * 100;
                double rightB = Math.random() * 100 + 100;
                double step = Math.random();

                synchronized (task) {
                    while (task.getFunc() != null) // ждем пока не обрабатывается предыдущее задание
                        task.wait();
                    // устанавливаем параметры
                    task.setFunc(log);
                    task.setLeftBorder(leftB);
                    task.setRightBorder(rightB);
                    task.setStep(step);
                    System.out.printf("Source %.2f %.2f %.2f\n", leftB, rightB, step);

                    task.notifyAll(); // уведомляем все ожидающие потоки о появлении данных
                }
            }
        }catch (InterruptedException e) {
            System.out.println("генератор прерван");
        }
    }
}
```

```

public class SimpleIntegrator implements Runnable { 1 usage
    private Task task; 11 usages

    public SimpleIntegrator(Task task) { 5 usages
        this.task = task;
    }
    @Override
    public void run() {
        try {
            for (int i = 0; i < task.getTaskCount(); i++) {
                synchronized (task) {
                    while (task.getFunc() == null) // ждем пока появятся данные
                        task.wait();

                    Function func = task.getFunc();
                    double leftB = task.getLeftBorder();
                    double rightB = task.getRightBorder();
                    double step = task.getStep();
                    // вычисляем интеграл
                    double result = Functions.integration(func, leftB, rightB, step);
                    System.out.printf("Result %.2f %.2f %.2f %.2f\n", leftB, rightB, step, result);

                    task.setFunc(null); // сбрасываем задание
                    task.notifyAll(); // уведомляем ждущие потоки
                }
            }
        } catch (InterruptedException e) {
            System.out.println("итератор прерван");
        }
    }
}

```

Тестирование в Main

В методе simpleThreads() были созданы и запущены два параллельных потока

```

public static void simpleThreads(){ 1 usage
    Task task = new Task(func: null, left: 0, right: 0, step: 0, count: 100);

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

    //устанавливаем приоритеты потокам
    generatorThread.setPriority(Thread.MAX_PRIORITY);
    integratorThread.setPriority(Thread.MIN_PRIORITY);

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

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

```

Тестирование задания 3:

Source 2,48 156,69 0,36
Result 2,48 156,69 0,36 410,20

Source 76,44 143,79 0,85
Result 76,44 143,79 0,85 197,70

Source 21,34 137,07 0,72
Result 21,34 137,07 0,72 253,89

Source 31,18 183,17 0,48
Result 31,18 183,17 0,48 927,10

Source 49,93 155,13 0,90
Result 49,93 155,13 0,90 285,59

Source 47,69 104,97 0,93
Result 47,69 104,97 0,93 107,78

Source 2,54 163,60 0,78
Result 2,54 163,60 0,78 5294,13

Source 35,48 189,46 0,84
Result 35,48 189,46 0,84 1267,86

Source 71,56 134,15 0,17
Result 71,56 134,15 0,17 639,00

Задание №4

Для выполнения этого задания я создала два новых класса Generator и Integrator, которые содержат объект Task и два семафора: semaphoreGen и semaphoreInteg.

Первый семафор контролирует доступ генератора к записи данных, а второй - доступ интегратора к чтению. Генератор захватывает semaphoreGen.acquire() в run() и блокирует доступ к общим данным на время записи данных в Task. Завершая свою часть, генератор освобождает semaphoreInteg.release(), сигнализируя интегратору о готовности данных. Интегратор захватывает semaphoreInteg.acquire() и

считывает параметры из Task для вычисления интеграла. Затем он освобождает semaphoreGen.release(), позволяя генератору создать следующее задание и тд.

```
public class Generator extends Thread{ 2 usages
    private Task task; 6 usages
    private Semaphore semaphoreGen; 2 usages
    private Semaphore semaphoreInteg; 2 usages

    public Generator(Task task, Semaphore semGen, Semaphore semInteg){ 5 usages
        this.task = task;
        this.semaphoreGen = semGen;
        this.semaphoreInteg = semInteg;
    }

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

                semaphoreGen.acquire();
                try{
                    double base = Math.random() * 9 + 1;
                    Function log = new Log(base);
                    double leftB = Math.random() * 100;
                    double rightB = Math.random() * 100 + 100;
                    double step = Math.random();

                    // устанавливаем параметры
                    task.setFunc(log);
                    task.setLeftBorder(leftB);
                    task.setRightBorder(rightB);
                    task.setStep(step);
                    System.out.printf("Source %.2f %.2f %.2f\n", leftB, rightB, step);

                } catch (Exception e) {
                    System.out.println("ошибка генерации задачи");
                }finally {
                    semaphoreInteg.release();
                }
            }
        }catch (InterruptedException e) {
            System.out.println("генератор прерван");
            Thread.currentThread().interrupt();
        }
    }
}
```

```

public class Integrator extends Thread { 2 usages
    private Task task; 5 usages
    private Semaphore semaphoreGen; 2 usages
    private Semaphore semaphoreInteg; 2 usages

    public Integrator(Task task, Semaphore semGen, Semaphore semInteg){ 5 usages
        this.task = task;
        this.semaphoreGen = semGen;
        this.semaphoreInteg = semInteg;
    }

    @Override
    public void run() {
        try {
            while (!isInterrupted()) {
                semaphoreInteg.acquire();
                try{

                    Function func = task.getFunc();
                    double leftB = task.getLeftBorder();
                    double rightB = task.getRightBorder();
                    double step = task.getStep();

                    // вычисляем интеграл
                    double result = Functions.integration(func, leftB, rightB, step);
                    System.out.printf("Result %.2f %.2f %.2f\n\n", leftB, rightB, step, result);

                }catch (Exception e){
                    System.out.println("ошибка при вычислении");
                }finally {
                    semaphoreGen.release();
                }
            }
        }catch (InterruptedException e){
            System.out.println("интегратор прерван");
            Thread.currentThread().interrupt();
        }
    }
}

```

Тестирование в Main

В Main был реализован метод complicatedThreads(). Потоки генератора и интегратора создаются и запускаются. Основной поток ожидает 50 миллисекунд с помощью Thread.sleep(50), после чего прерывает оба потока вызовом interrupt().

```
public static void complicatedThreads() { 1 usage
    Task task = new Task( func: null, left: 0, right: 0, step: 0, count: 100);

    Semaphore semaphoreGen = new Semaphore( permits: 1);
    Semaphore semaphoreInteg = new Semaphore( permits: 0);

    Generator generator = new Generator(task, semaphoreGen, semaphoreInteg);
    Integrator integrator = new Integrator(task, semaphoreGen, semaphoreInteg);

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

    try {
        Thread.sleep( millis: 50);

        // Прерываем оба потока
        generator.interrupt();
        integrator.interrupt();

        // Ждём завершения
        generator.join();
        integrator.join();
    } catch (InterruptedException e) {
        throw new RuntimeException(e);
    }
}
```

Тестирование задания 4:

```
Source 98,12 180,73 0,54
Result 98,12 180,73 0,54 209,07
```

```
Source 68,83 128,90 0,89
Result 68,83 128,90 0,89 129,97
```

```
Source 12,15 157,71 0,36
Result 12,15 157,71 0,36 343,15
```

```
Source 3,23 190,00 0,17
Result 3,23 190,00 0,17 426,72
```

```
Source 57,52 192,69 0,61
Result 57,52 192,69 0,61 429,94
```

```
Source 47,93 134,50 0,89
Result 47,93 134,50 0,89 13009,53
```