Лабораторная работа 6

Выполнил: студент группы

6301-030301D

Дымченко В.Р.

Задание 1

Добавляем в класс Functions метод, возвращающий значение интеграла функции, вычисленное с помощью численного метода. Вычисление значения интеграла должно выполняться по методу трапеций.

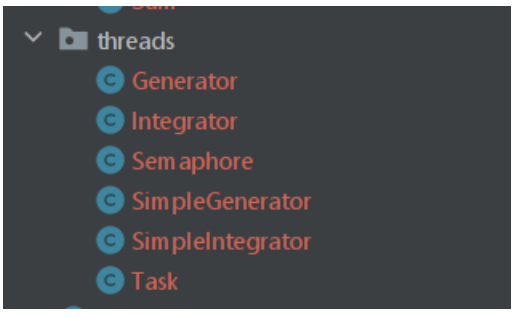
```
public static double integrate(Function function, double left, double right, double step) {  4 usages
    if (left < function.getLeftDomainBorder() || right > function.getRightDomainBorder()) {
        throw new IllegalArgumentException("Интервал интегрирования выходит за границы области определения функции");
    }
    if (step <= 0) {
        throw new IllegalArgumentException("Шаг интегрирования должен быть положительным");
    }
    if (left >= right) {
        throw new IllegalArgumentException("Левая граница должна быть меньше правой");
    }

    double integral = 0.0;
    double x = left;

    double xNext = Math.min(x + step, right);
    double y1 = function.getFunctionValue(x);
    double y2 = function.getFunctionValue(xNext);
    integral += (y1 + y2) * (xNext - x) / 2;

    x = xNext;

    while (x < right) {
        xNext = Math.min(x + step, right);
        y1 = function.getFunctionValue(x);
        y2 = function.getFunctionValue(xNext);
        integral += (y1 + y2) * (xNext - x) / 2;
        x = xNext;
    }
```

В методе main() проверили работу метода интегрирования. Для этого вычислили интеграл для экспоненты на отрезке от 0 до 1. Определили также, какой шаг дискретизации нужен, чтобы рассчитанное значение отличалось от теоретического в 7 знаке после запятой.

```java
private static void testIntegration() {  1 usage
    System.out.println("Тестирование интегрирования экспоненты на отрезке [0, 1]");

    Exp exp = new Exp();
    double left = 0;
    double right = 1;
    double theoretical = Math.E - 1;

    System.out.printf("Теоретическое значение: %.10f\n", theoretical);

    System.out.println("\nПоиск шага для точности 10^-7:");
    double step = 0.1;
    double diff;
    int iteration = 0;

    do {
        double result = Functions.integrate(exp, left, right, step);
        diff = Math.abs(result - theoretical);
        System.out.printf("Итерация %d: шаг = %.10f, результат = %.10f, разница = %.10f\n",
                iteration, step, result, diff);
        step /= 2;
        iteration++;
    } while (diff > 1e-7 && step > 1e-12);

    System.out.printf("\nДостигнута точность 10^-7 при шаге = %.10f\n", step * 2);
}
```

Задание 2

Далее в приложении один поток инструкций будет генерировать задачи для интегрирования, а второй поток – решать их. Для этого потребуется объект задания, через который эти потоки будут взаимодействовать. В объекте будут храниться параметры задания. Создаем пакет threads, в котором будут размещены классы, связанные с потоками. В пакете threads описываем 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.



Класс Task

```java
package functions.threads;

import functions.Function;
import functions.basic.Log;

public class Task {  no usages
    private Function function;  2 usages
    private double left;  2 usages
    private double right;  2 usages
    private double step;  2 usages
    private int tasksCount;  3 usages

    public Task(int tasksCount) {  12 usages
        this.tasksCount = tasksCount;
    }

    public int getTasksCount() {  no usages
        return tasksCount;
    }

    public void setTasksCount(int tasksCount) {  no usages
        this.tasksCount = tasksCount;
    }

    public Function getFunction() {
        return function;
    }

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

```java
    public double getLeft() {  no usages
        return left;
    }

    public void setLeft(double left) {  no usages
        this.left = left;
    }

    public double getRight() {  no usages
        return right;
    }

    public void setRight(double right) {  no usages
        this.right = right;
    }

    public double getStep() {  no usages
        return step;
    }

    public void setStep(double step) {  no usages
        this.step = step;
    }
}
```

В главном классе программы описываем метод nonThread(), реализующий последовательную версию программы. В методе необходимо создать объект класса Task и установить в нём количество выполняемых заданий (минимум 100). После этого в цикле нужно выполнить следующие действия:

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

Проверяем работу метода, вызвав его в методе main

```
public static void nonThread() {  1 usage
    System.out.println("Последовательная версия программы:");

    Task task = new Task( tasksCount: 100);

    for (int i = 0; i < task.getTasksCount(); i++) {
        double base = 1 + Math.random() * 9;
        Log logFunc = new Log(base);

        double left = Math.random() * 100;
        double right = 100 + Math.random() * 100;
        double step = Math.random();

        task.setFunction(logFunc);
        task.setLeft(left);
        task.setRight(right);
        task.setStep(step);

        System.out.printf("Source %.2f %.2f %.2f%n", left, right, step);

        try {
            double result = Functions.integrate(task.getFunction(), task.getLeft(), task.getRight(), task.getStep());

            System.out.printf("Result %.2f %.2f %.2f %.2f%n", left, right, step, result);
        } catch (IllegalArgumentException e) {
            System.out.printf("Error for %.2f %.2f %.2f: %s%n", left, right, step, e.getMessage());
        }
    }
}
```

Задание 3

В пакете threads создаем два класса.

Класс SimpleGenerator должен 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.

```
package functions.threads;

import functions.Function;
import functions.basic.Log;

public class SimpleGenerator implements Runnable {  1 usage
    private Task task;  12 usages
    private boolean running = true;  3 usages

    public SimpleGenerator(Task task) {  5 usages
        this.task = task;
    }

    @Override
    public void run() {
        try {
            for (int i = 0; i < task.getTasksCount() && running; i++) {
                synchronized (task) {
                    double base = 1 + Math.random() * 9;
                    Log logFunc = new Log(base);

                    double left = Math.random() * 100;
                    double right = 100 + Math.random() * 100;
                    double step = Math.random();

                    task.setFunction(logFunc);
                    task.setLeft(left);
                    task.setRight(right);
                    task.setStep(step);
```

```
                    System.out.printf("Source %.2f %.2f %.2f%n", left, right, step);

                    task.notify();

                    if (i < task.getTasksCount() - 1) {
                        task.wait();
                    }
                }

                Thread.sleep( millis: 1);
            }

            synchronized (task) {
                running = false;
                task.notify();
            }

        } catch (InterruptedException e) {
            System.out.println("Генератор прерван");
            Thread.currentThread().interrupt();
        }
    }

    public void stop() {  no usages
        running = false;
    }
```

Класс SimpleIntegrator должен 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.

```java
package functions.threads;

import functions.Function;
import functions.Functions;

public class SimpleIntegrator implements Runnable {  1 usage
    private Task task;  10 usages
    private boolean running = true;  3 usages

    public SimpleIntegrator(Task task) {  5 usages
        this.task = task;
    }

    @Override
    public void run() {
        try {
            while (running) {
                synchronized (task) {
                    task.wait();

                    if (!running) break;

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

                    try {
                        double result = Functions.integrate(func, left, right, step);
                        System.out.printf("Result %.2f %.2f %.2f %.2f%n", left, right, step, result);
                    } catch (IllegalArgumentException e) {
                        System.out.printf("Error for %.2f %.2f %.2f: %s%n", left, right, step, e.getMessage());
                    }

                    task.notify();
                }
            }
        } catch (InterruptedException e) {
            System.out.println("Интегратор прерван");
            Thread.currentThread().interrupt();
        }
    }

    public void stop() {  no usages
        running = false;
        synchronized (task) {
            task.notify();
        }
    }
}
```

В главном классе программы создайте метод simpleThreads(). В нём создайте объект задания, укажите количество выполняемых заданий (минимум 100), создайте и запустите два потока вычислений, основанных на описанных классах SimpleGenerator и SimpleIntegrator. Проверьте работу метода, вызвав его в методе main().

```java
public static void simpleThreads() {  1 usage
    System.out.println("Запуск простой многопоточной версии:");

    Task task = new Task( tasksCount: 100);

    Thread generatorThread = new Thread(new SimpleGenerator(task));
    Thread integratorThread = new Thread(new SimpleIntegrator(task));

    System.out.println("Установка приоритетов: генератор - min, интегратор - max");
    generatorThread.setPriority(Thread.MIN_PRIORITY);
    integratorThread.setPriority(Thread.MAX_PRIORITY);

    generatorThread.start();
    integratorThread.start();

    try {
        long timeout = 3000;
        long startTime = System.currentTimeMillis();

        while ((generatorThread.isAlive() || integratorThread.isAlive()) &&
                (System.currentTimeMillis() - startTime) < timeout) {
            Thread.sleep( millis: 100);
        }

        if (generatorThread.isAlive() || integratorThread.isAlive()) {
            System.out.println("\nТаймаут: Потоки зависли, принудительное прерывание.....");
            generatorThread.interrupt();
            integratorThread.interrupt();
```

```java
            generatorThread.join( millis: 500);
            integratorThread.join( millis: 500);
        }

    } catch (InterruptedException e) {
        System.out.println("Основной поток прерван");
    }

    System.out.println("Простая многопоточная версия завершена");
}
```

Задание 4

Для устранения проблемы, что не все сгенерированные задания оказываются выполнены интегрирующим потоком, потребуется дополнительный объект, представляющий собой одноместный семафор, различающий операции чтения и записи в защищаемый объект.

```java
package functions.threads;

public class Semaphore {  no usages
    private boolean canWrite = true;  3 usages
    private boolean canRead = false;  3 usages

    public synchronized void startWrite() throws InterruptedException {  no usages
        while (!canWrite) {
            wait();
        }
        canWrite = false;
    }

    public synchronized void endWrite() {  no usages
        canRead = true;
        notifyAll();
    }

    public synchronized void startRead() throws InterruptedException {  no usages
        while (!canRead) {
            wait();
        }
        canRead = false;
    }

    public synchronized void endRead() {  no usages
        canWrite = true;
        notifyAll();
    }
}
```

В пакете threads создаем два следующих класса.

Класс Generator должен 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.

```java
package functions.threads;

import functions.Function;
import functions.basic.Log;

public class Generator extends Thread {  2 usages
    private Task task;  7 usages
    private Semaphore semaphore;  5 usages

    public Generator(Task task, Semaphore semaphore) {  5 usages
        this.task = task;
        this.semaphore = semaphore;
    }

    @Override
    public void run() {
        try {
            for (int i = 0; i < task.getTasksCount(); i++) {
                if (Thread.currentThread().isInterrupted()) {
                    throw new InterruptedException();
                }

                semaphore.startWrite();

                double base = 1 + Math.random() * 9;
                Log logFunc = new Log(base);

                double left = Math.random() * 100;
                double right = 100 + Math.random() * 100;
                double step = Math.random();
```

```java
                task.setFunction(logFunc);
                task.setLeft(left);
                task.setRight(right);
                task.setStep(step);

                System.out.printf("Source %.2f %.2f %.2f%n", left, right, step);

                semaphore.endWrite();

                Thread.sleep( millis: 10);
            }

            semaphore.startWrite();
            task.setTasksCount(-1);
            semaphore.endWrite();

        } catch (InterruptedException e) {
            System.out.println("Генератор был прерван");
        }
    }
}
```

Класс Integrator должен 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 интегрирующего класса.

```java
package functions.threads;

import functions.Function;
import functions.Functions;

public class Integrator extends Thread {  2 usages
    private Task task;  6 usages
    private Semaphore semaphore;  4 usages

    public Integrator(Task task, Semaphore semaphore) {  5 usages
        this.task = task;
        this.semaphore = semaphore;
    }

    @Override
    public void run() {
        try {
            while (true) {
                if (Thread.currentThread().isInterrupted()) {
                    throw new InterruptedException();
                }

                semaphore.startRead();
                if (task.getTasksCount() == -1) {
                    semaphore.endRead();
                    break;
                }

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

                semaphore.endRead();

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

                    System.out.printf("Result %.2f %.2f %.2f %.2f%n", left, right, step, result);
                } catch (IllegalArgumentException e) {
                    System.out.printf("Error for %.2f %.2f %.2f: %s%n", left, right, step, e.getMessage());
                }

                Thread.sleep( millis: 10);
            }
        } catch (InterruptedException e) {
            System.out.println("Интегратор был прерван");
        }
    }
}
```

В главном классе программы создайте метод complicatedThreads(). В нём создайте объект задания, укажите количество выполняемых заданий (минимум 100), создайте и запустите два потока вычислений классов Generator и Integrator. Проверьте работу метода, вызвав его в методе main().

```
    public static void complicatedThreads() {  1 usage
        System.out.println("Запуск усовершенствованной многопоточной версии:");

        Task task = new Task( tasksCount: 100);
        Semaphore semaphore = new Semaphore();

        Generator generator = new Generator(task, semaphore);
        Integrator integrator = new Integrator(task, semaphore);

        generator.setPriority(Thread.NORM_PRIORITY);
        integrator.setPriority(Thread.NORM_PRIORITY);

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

        try {
            Thread.sleep( millis: 50);
        } catch (InterruptedException e) {
            System.out.println("Основной поток прерван");
        }

        System.out.println("\nПрерывание потоков после 50мс.....");

        generator.interrupt();
        integrator.interrupt();
```

```
        try {
            long timeout = 2000; // 2 секунды
            long startTime = System.currentTimeMillis();

            while ((generator.isAlive() || integrator.isAlive()) &&
                    (System.currentTimeMillis() - startTime) < timeout) {
                Thread.sleep( millis: 100);
            }

            if (generator.isAlive() || integrator.isAlive()) {
                System.out.println("Потоки не завершились за таймаут, принудительная остановка");
            }

        } catch (InterruptedException e) {
            System.out.println("Основной поток прерван при ожидании завершения");
        }

        System.out.println("Усовершенствованная многопоточная версия завершена");
    }
}
```

Изменения

Добавим флаг taskReady в Task. (Он покажет, готово ли задание к обработке и решит проблему "ложных пробуждений")

```
private boolean taskReady = false;
```

```
public boolean isTaskReady() {  2 usages  new *
    return taskReady;
}

public void setTaskReady(boolean ready) {  3 usages  new *
    this.taskReady = ready;
}
```

Используем таймауты wait(100) (Потоки не зависнут навсегда, автоматически просыпаются через 100мс)

Вычисления вне synchronized блока.

SympleIntegrator

```
@Override  Vlados-ux *
public void run() {
    try {
        while (true) {
            Function func = null;
            double left = 0, right = 0, step = 0;
            boolean shouldProcess = false;
            synchronized (task) {
                while (!task.isTaskReady() && task.getTasksCount() != -1) {
                    task.wait( timeoutMillis: 100);
                }

                if (task.getTasksCount() == -1) {
                    break;
                }

                if (!task.isTaskReady()) {
                    continue;
                }

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

                task.setTaskReady(false);
```

```
                task.notify();

            }

            try {
                double result = Functions.integrate(func, left, right, step);
                System.out.printf("SimpleInt: Result %.2f %.2f %.2f %.2f%n",
                        left, right, step, result);
            } catch (IllegalArgumentException e) {
                System.out.printf("SimpleInt: Error for %.2f %.2f %.2f: %s%n",
                        left, right, step, e.getMessage());
            }

            Thread.sleep( millis: 10);
        }
    } catch (InterruptedException e) {
        System.out.println("SimpleIntegrator: прерван");
        Thread.currentThread().interrupt();
    } catch (Exception e) {
        System.out.println("SimpleIntegrator ошибка: " + e.getMessage());
    }
  }
}
```

SimpleGenerator

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

            synchronized (task) {
                double base = 1 + Math.random() * 9;
                Log logFunc = new Log(base);

                double left = Math.random() * 100;
                double right = 100 + Math.random() * 100;
                double step = Math.random();

                task.setFunction(logFunc);
                task.setLeft(left);
                task.setRight(right);
                task.setStep(step);

                System.out.printf("SimpleGen: Source %.2f %.2f %.2f%n", left, right, step);

                task.setTaskReady(true);

                task.notify();

                if (i < task.getTasksCount() - 1) {
                    task.wait( timeoutMillis: 100);
                }
```

```
                Thread.sleep( millis: 10);
            }

            synchronized (task) {
                task.setTaskReady(false);
                task.setTasksCount(-1);
                task.notify();
            }

        } catch (InterruptedException e) {
            System.out.println("SimpleGenerator: прерван");
            Thread.currentThread().interrupt();
        } catch (Exception e) {
            System.out.println("SimpleGenerator ошибка: " + e.getMessage());
        }
    }
}
```

MainThreads

```
public static void simpleThreads() {  1 usage   Vlados-ux *
    System.out.println("Запуск простой многопоточной версии:");
    Task task = new Task( tasksCount: 100);

    Thread generatorThread = new Thread(new SimpleGenerator(task));
    Thread integratorThread = new Thread(new SimpleIntegrator(task));

    System.out.println("Установка приоритетов: генератор - MIN, интегратор - MAX");
    generatorThread.setPriority(Thread.MIN_PRIORITY);
    integratorThread.setPriority(Thread.MAX_PRIORITY);

    generatorThread.start();
    integratorThread.start();

    try {
        generatorThread.join( millis: 5000);
        integratorThread.join( millis: 1000);

        if (generatorThread.isAlive() || integratorThread.isAlive()) {
            System.out.println("\nТАЙМАУТ: Потоки зависли, принудительное прерывание...");
            generatorThread.interrupt();
            integratorThread.interrupt();

            Thread.sleep( millis: 500);
        }
```

```
        System.out.println("Потоки завершены: генератор - " +
                (generatorThread.isAlive() ? "Не завершен" : "завершен") +
                ", интегратор - " +
                (integratorThread.isAlive() ? "Не завершен" : "завершен"));

    } catch (InterruptedException e) {
        System.out.println("Основной поток прерван");
    }

    System.out.println("Простая многопоточная версия завершена");
}
```

Вывод

```
Потоки и интегрирование

Метод интегрирования
Тестирование интегрирования экспоненты на отрезке [0, 1]
Теоретическое значение: 1,7182818285

Поиск шага для точности 10^-7:
Итерация 0: шаг = 0,1000000000, результат = 1,7197134914, разница = 0,0014316629
Итерация 1: шаг = 0,0500000000, результат = 1,7186397889, разница = 0,0003579605
Итерация 2: шаг = 0,0250000000, результат = 1,7183713214, разница = 0,0000894929
Итерация 3: шаг = 0,0125000000, результат = 1,7183042019, разница = 0,0000223734
Итерация 4: шаг = 0,0062500000, результат = 1,7182874218, разница = 0,0000055934
Итерация 5: шаг = 0,0031250000, результат = 1,7182832268, разница = 0,0000013983
Итерация 6: шаг = 0,0015625000, результат = 1,7182821780, разница = 0,0000003496
Итерация 7: шаг = 0,0007812500, результат = 1,7182819159, разница = 0,0000000874

Достигнута точность 10^-7 при шаге = 0,0007812500
```

```
Последовательная версия
Последовательная версия программы:
Source 0,49 115,56 0,64
Result 0,49 115,56 0,64 270,00
Source 43,46 188,66 0,07
Result 43,46 188,66 0,07 350,78
Source 82,57 141,74 0,32
Result 82,57 141,74 0,32 171,18
Source 38,47 136,45 0,83
Result 38,47 136,45 0,83 255,80
Source 1,92 102,26 0,02
Result 1,92 102,26 0,02 191,17
Source 11,37 153,44 0,83
Result 11,37 153,44 0,83 405,19
Source 62,23 156,54 0,16
Result 62,23 156,54 0,16 251,34
Source 3,50 171,30 0,95
Result 3,50 171,30 0,95 4551,64
Source 43,76 120,50 0,82
Result 43,76 120,50 0,82 257,61
Source 58,38 105,51 0,70
Result 58,38 105,51 0,70 112,26
Source 62,44 173,32 0,12
Result 62,44 173,32 0,12 249,99
Source 89,63 158,19 0,27
```

.....(вывод 100 значений)

```
Простая многопоточная версия
Запуск простой многопоточной версии:
Установка приоритетов: генератор - MIN, интегратор - MAX
Source 98,27 101,83 0,75
Result 98,27 101,83 0,75 7,78
Source 81,28 166,54 0,65
Result 81,28 166,54 0,65 241,86
Source 88,28 176,70 0,12
Result 88,28 176,70 0,12 239,76
Source 70,79 144,90 0,40
Result 70,79 144,90 0,40 181,38
Source 26,34 160,06 0,06
Result 26,34 160,06 0,06 390,65
Source 65,26 117,17 0,55
Result 65,26 117,17 0,55 406,12
Source 77,72 188,66 0,53
Result 77,72 188,66 0,53 274,29
Source 76,03 169,99 0,98
Result 76,03 169,99 0,98 1591,27
Source 65,69 177,17 0,37
Result 65,69 177,17 0,37 779,89
Source 81,29 147,84 0,99
Result 81,29 147,84 0,99 147,29
Source 8,26 199,00 0,34
Result 8,26 199,00 0,34 562,86
```
...(вывод 100значений)

```
Source 10,08 110,61 0,98
Result 10,08 110,61 0,98 278,81
Потоки завершены: генератор - завершен, интегратор - завершен
Простая многопоточная версия завершена


Усовершенствованная многопоточная версия
Запуск усовершенствованной многопоточной версии:
Source 29,86 139,75 0,43
Result 29,86 139,75 0,43 348,64
Source 91,99 187,56 0,66
Result 91,99 187,56 0,66 226,39
Source 7,72 144,47 0,44
Result 7,72 144,47 0,44 329,28
Source 51,47 157,38 0,66
Result 51,47 157,38 0,66 215,06
Source 76,73 143,12 0,69
Result 76,73 143,12 0,69 137,61

Прерывание потоков после 50мс.............
Интегратор был прерван
Генератор был прерван
Усовершенствованная многопоточная версия завершена
```