

Лабораторная работа №6

*Выполнил: Беляев Дмитрий Михайлович
Студент 6203-010302D группы*

Ход выполнения

Первым делом было прочитано ТЗ. Спасибо за внимание. Дальше без комментариев.

Задание 1

Соответственно, напишем формулу трапеции(площади) – $S = (a+b) / 2 * h$. Соответственно, немного придется отредактировать, $h = x_1 - x_0$, а a, b – значения функции соответственно. x_1 – след шаг, x_2 – текущий шаг. См скрин 1(реализация)

The screenshot shows the IntelliJ IDEA interface with the project 'Lab6' open. The 'src/functions' package contains several classes like Cos, Exp, Log, Sin, Tan, Trigonom, and Functions. The 'Functions.java' file is selected and contains the following code:

```
package functions;

import functions.meta.*;

public class Functions {
    public static Function shift(Function f, double shiftX, double shiftY) { return new Shift(f, shiftX, shiftY); }
    public static Function scale(Function f, double scaleX, double scaleY) { return new Scale(f, scaleX, scaleY); }
    public static Function power(Function f, double power) { return new Power(f, power); }

    public static Function sum(Function f1, Function f2) { return new Sum(f1, f2); }
    public static Function mult(Function f1, Function f2) { return new Mult(f1, f2); }

    public static Function composition(Function f1, Function f2) { return new Composition(f1, f2); }

    public static double findIntegral(Function f, double leftX, double rightX, double step) {
        if (leftX < f.getLeftDomainBorder() || rightX > f.getRightDomainBorder())
            throw new IllegalArgumentException();
        double currentValue = 0;
        double currentX = leftX;
        while (currentX < rightX) {
            nextX = Math.min(currentX + step, rightX);
            currentValue += (f.getFunctionValue(currentX) + f.getFunctionValue(nextX)) / 2 *
                (nextX - currentX);
            currentX += nextX;
        }
        return currentValue;
    }
}
```

The right side of the screen shows the 'Tasks' and 'Documentation' panes for 'Zadanie 1'. The task description is:

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

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

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

Вычисление значения интеграла должно выполняться по методу трапеций. Для этого вся область интегрирования разбивается на участки, длина которых (кроме одного) равна шагу дискретизации. На каждом таком участке площадь под кривой, описываемой заданной функцией, приближается площадью трапеции, две вершины которой расположены на оси абсцисс на границах участка, а еще две – на кривой в точках границ участка. Обратите внимание на то, что в области интегрирования обязательно укладывается целое количество шагов дискретизации.

В методе main() проверьте работу метода интегрирования. Для этого вычислите интеграл для экспоненты на отрезок от 0 до 1. Определите также, какой шаг дискретизации нужен, чтобы рассчитанное значение отличалось от теоретического в 7 знаках после запятой.

(скрин 1)

Запустим, видим, ошибка... см скрин 2

The screenshot shows the IntelliJ IDEA interface with the project 'Lab6' open. The 'src/functions' package contains several classes like Cos, Exp, Log, Sin, Tan, Trigonom, and Functions. The 'Main.java' file is selected and contains the following code:

```
import functions.Functions;
import functions.basic.Exp;

public class Main {
    public static void main(String[] args) {
        Exp exp = new Exp(); // значение интеграла примерно равно 1.7182818
        System.out.println(Functions.findIntegral(exp, 0.0, 1.0, 0.0001));
    }
}
```

The bottom pane shows the terminal output:

```
*E:\Program Files\Java\jdk-21\bin\java.exe* "javaagent:E:\Program Files\JetBrains\IntelliJ IDEA 2023.1\lib\idea_rt.jar=s4SS3" -Dfile.encoding=UTF-8 -Dsun.stdout.encoding=UTF-8
Process finished with exit code 0
```

The right side of the screen shows the 'Tasks' and 'Documentation' panes for 'Zadanie 1'. The task description is:

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

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

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

Вычисление значения интеграла должно выполняться по методу трапеций. Для этого вся область интегрирования разбивается на участки, длина которых (кроме одного) равна шагу дискретизации. На каждом таком участке площадь под кривой, описываемой заданной функцией, приближается площадью трапеции, две вершины которой расположены на оси абсцисс на границах участка, а еще две – на кривой в точках границ участка. Обратите внимание на то, что в области интегрирования обязательно укладывается целое количество шагов дискретизации.

(скрин 2)

Возвращаемся обратно к коду, ищем ошибку, находим, вносим правки и заново запускаем(см скринь 3-4)

```
        (f.getFunctionValue(nextX));
    }
    currentX = nextX;
}
return currentValue;
```

(скрин 3)

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure for "Lab6". The "src" folder contains packages like "functions" which further contain classes such as "Cos", "Exp", "Log", "Sin", "Tan", and "Trigonometric".
- Main.java:** The active file contains the following code:

```
import functions.Functions;
import functions.basic.Exp;

public class Main {
    public static void main(String[] args) {
        Exp exp = new Exp(); // значение интеграла примерно равно 1.7182818
        System.out.println(functions.findIntegral(exp, 0, 1, 0.001));
    }
}
```
- Run Tab:** Shows the command used to run the application: "C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2025.1\lib\idea_rt.jar=54734" -Dfile.encoding=UTF-8 -Dsun.stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8 Main". The output shows "Process finished with exit code 0".
- Right Panel:** Displays a note about trapezoidal integration and a task description for "Задание 2".

(скрин 4)

Чисто технически, я очень удачно подобрал параметр шага и отличается только начиная с 7 знака...

Задание выполнено

Задание 2

Напишем Task...(см скрин 5)

```

public class Task {
    private Function function;
    private double stepIntegration;
    private double leftX, rightX;
    private int iterations;
    private int currentIteration = 0;

    public Task(Function function, double stepIntegration, double leftX, double rightX, int iterations) {
        this.function = function;
        this.stepIntegration = stepIntegration;
        this.leftX = leftX;
        this.rightX = rightX;
        this.iterations = iterations;
    }

    public Function getFunction() { return function; }

    public double getStepIntegration() { return stepIntegration; }

    public double getLeftX() { return leftX; }

    public double getRightX() { return rightX; }

    public int getIterations() { return iterations; }

    public int getCurrentIteration() { return currentIteration; }

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

    public void setStepIntegration(double stepIntegration) { this.stepIntegration = stepIntegration; }

    public void setLeftX(double leftX) { this.leftX = leftX; }

    public void setRightX(double rightX) { this.rightX = rightX; }

    public void setIterations(int iterations) { this.iterations = iterations; }

    public void setCurrentIteration(int currentIteration) { this.currentIteration = currentIteration; }

    public void incrementIteration() { this.currentIteration++; }
}

```

(скрин 5)

Честно, пока непонятно, зачем мы это делаем...

Внесем изменения в Main(см скрин 6)

```

import Functions.Functions;
import Functions.basic.Log;
import threads.Task;

public class Main {
    public static void main(String[] args) {
        nonThread();
    }

    public static void nonThread() {
        Task t = new Task();
        t.setIterations(100);
        double value;
        while (t.getCurrentIteration() < t.getIterations()) {
            t.execute(new Log(Math.random() * 9 + 1));
            t.setLeftX(Math.random() * 100);
            t.setRightX((Math.random() * 100) + 100);
            t.setStepIntegration(Math.random());
            System.out.println("Source " + t.getLeftX() + " " + t.getRightX() + " " + t.getStepIntegration());
            value = Functions.findIntegral(t.getFunction(), t.getLeftX(), t.getRightX(), t.getStepIntegration());
            System.out.println("Result " + t.getLeftX() + " " + t.getRightX() + " " + t.getStepIntegration() + " " + value);
            t.incrementIteration();
        }
    }
}

```

(скрин 6)

И запустим...(см скрин 7)

```

public class Main {
    public static void nonThread() {
        t.setFunction(new Log( base: Math.random() * 9 + 1));
        //...
        t.setLeftX(Math.random() * 100);
        t.setRightX(100 + Math.random() * 100);
        t.setStepIntegration(Math.random());
        //...
        System.out.println("Source " + t.getLeftX() + " " + t.getRightX() + " " + t.getStepIntegration());
        //...
        value = Functions.findIntegral(getFunction(), t.getLeftX(), t.getRightX(), t.getStepIntegration());
        System.out.println("Result " + t.getLeftX() + " " + t.getRightX() + " " + t.getStepIntegration() + " " + value);
        t.incrementIteration();
    }
}

```

Process finished with exit code 0

29:36 CRLF UTF-8 4 spaces

(скрин 7)

Вроде все работает, подозрительно без ошибок...

Задание выполнено

Задание 3

Напишем классы(см скрины 8-9)

```

package threads;
import functions.basic.Log;

public class SimpleGenerator implements Runnable {
    private final Task task;
    private int currentGeneratedTask = -1;

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

    @Override
    public void run() {
        while (task.getIterations() != task.getCurrentIteration()) {
            if (currentGeneratedTask != task.getCurrentIteration()) {
                currentGeneratedTask = task.getCurrentIteration();

                task.setFunction(new Log( base: Math.random() * 9 + 1));
                task.setLeftX(Math.random() * 100);
                task.setRightX(100 + Math.random() * 100);
                task.setStepIntegration(Math.random());
            }
        }
    }
}

```

25:1 CRLF UTF-8 4 spaces

(скрин 8)

```
package threads;

import functions.Functions;

public class SimpleIntegrator implements Runnable {
    private final Task task;
    private double result;

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

    @Override
    public void run() {
        while (task.getCurrentIteration() != task.getIterations()) {
            double value = Functions.integrate(task.getFunction(), task.getLeftX(), task.getRightX(), task);
            System.out.println("Result " + task.getLeftX() + " " + task.getRightX() + " " + task.getStepWidth());
            task.incrementIteration();
        }
    }
}
```

Класс `SimpleGenerator` должен реализовывать интерфейс `Runnable`, получать в конструкторе и сохранять в свое поле ссылку на объект типа `Task`, а в методе `run()` в цикле должны формироваться задачи и завершаться в полученный объект задания, а также выводиться сообщения в консоли.

Класс `SimpleIntegrator` должен реализовывать интерфейс `Runnable`, получать в конструкторе и сохранять в свое 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 консоли.

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

Попробуйте запускать программу (несколько раз). Попробуйте запускать программу, изменения приоритеты потоков перед их запуском. Убедитесь, что в некоторых случаях интегрирующий поток может прекращать свою выполнение из-за исключения `NullPointerException`. Определите причину возникновения этого исключения и сделайте так, чтобы оно не возникало (используйте какое-нибудь простое решение, без синхронизации и оповещений). Попробуйте запускать программу (несколько раз). Убедитесь, что в некоторых случаях интегрирующий поток выводит сообщения с набором данных, который не встречается в сообщениях генерирующего потока (например, левая граница области интегрирования оказывается взята из одного задания, а правая граница и шаг дискретизации – из другого). Определите причину возникновения таких ситуаций и устранимте ее, используя блоки синхронизации в методах `run()` генерирующего и интегрирующего потоков. Убедитесь, что в коде никогда не возникает необоснованное длительное блокирование объектов.

(скрин 9)

Попробуем запустить(см скрин 10)

```
import functions.Functions;
import functions.basic.Log;
import threads.SimpleGenerator;
import threads.SimpleIntegrator;
import threads.Task;

public class Main {
    public static void main(String[] args) {
        //nonThread();
        simpleThreads();
    }

    public static void simpleThreads() {
        Task t = new Task();
        t.setIterations(666);

        SimpleIntegrator si = new SimpleIntegrator(t);
        SimpleGenerator sg = new SimpleGenerator(t);

        new Thread(sg.start());
        new Thread(si.start());
    }

    public static void nonThread() {
        Task t = new Task();
        t.setIterations(666);
        double value;
        while (t.getCurrentIteration() < t.getIterations()) {
            // 1
            t.setFunction(new Log(base: Math.random() * 9 + 1));
            // 2
            t.setLeftX(Math.random() * 100);
        }
    }
}
```

Runnable , получать в конструкторе и сохранять в свое 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 консоли.

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

Попробуйте запускать программу (несколько раз). Попробуйте запускать программу, изменения приоритеты потоков перед их запуском. Убедитесь, что в некоторых случаях интегрирующий поток может прекращать свою выполнение из-за исключения `NullPointerException`. Определите причину возникновения этого исключения и сделайте так, чтобы оно не возникало (используйте какое-нибудь простое решение, без синхронизации и оповещений). Попробуйте запускать программу (несколько раз). Убедитесь, что в некоторых случаях интегрирующий поток выводит сообщения с набором данных, который не встречается в сообщениях генерирующего потока (например, левая граница области интегрирования оказывается взята из одного задания, а правая граница и шаг дискретизации – из другого). Определите причину возникновения таких ситуаций и устранимте ее, используя блоки синхронизации в методах `run()` генерирующего и интегрирующего потоков. Убедитесь, что в коде никогда не возникает необоснованное длительное блокирование объектов.

(скрин 10)

И результаты(см скрин 11)

```

import functions.Functions;
import functions.basic.Log;
import threads.SimpleGenerator;
import threads.SimpleIntegrator;
import threads.Task;

public class Main {
    public static void main(String[] args) {
        //nonThread();
        simpleThreads();
    }

    public static void simpleThreads() {
        Task t = new Task();
        t.setIterations(666);

        SimpleIntegrator si = new SimpleIntegrator(t);
        SimpleGenerator sg = new SimpleGenerator(t);
        new Thread(sg).start();
    }
}

```

Run Main

C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2021.1\lib\idea_rt.jar=50215" Source 0.0.0.0.0 Result 42.50321807014679 156.81014284371258 0.9428963148569537 254.86743373700384 Result 42.50321807014679 156.81014284371258 0.9428963148569537 254.86743373700384

(скрин 11)

Оно виснет. Ладно, отредактируем и запустим(см скрины 12-13)

```

3 usages
public class SimpleGenerator implements Runnable {
    private final Task task;
    public SimpleGenerator(Task task) {
        this.task = task;
    }
    @Override
    public void run() {
        while (task.GetCurrentIteration() != task.GetCurrentIteration()) {
            task.setFunction(new Log((base.Math.random() * 9 + 1)));
            task.setLeftX((Math.random() * 100));
            task.setRightX((Math.random() * 100));
            task.setStepIntegration(Math.random());
        }
    }
}

```

Run Main

Result 35.1165729622991 159.82189234830466 0.1250004767915921 310.4248340167867 Source 23.08340567857444 144.9955806737019 0.891059833188577 Result 89.40650222527607 161.6919549581663 0.654393177472453 1204.518548587596 Source 61.64480170287763 141.53866741713762 0.06352428799916898 Result 93.06708123522713 141.74218548284026 0.13049401674290395 164.02017753303946

Process finished with exit code 0

(скрин 12)

The screenshot shows the IntelliJ IDEA interface with the project 'Lab6' open. The code editor displays Main.java with the following content:

```

public class Main {
    public static void main(String[] args) {
        //nonThread();
        simpleThreads();
    }

    public static void simpleThreads() {
        Task t = new Task();
        t.setIterations(100);
    }
}

```

The terminal window shows the execution results:

```

RESULT 13.100082339833685 110.5932488677314 0.9303684566521761
Source 51.36706687918552 186.71742578173842 0.06046096847104565 301.6106360024559
Source 67.32551237844491 123.97048134458161 0.6850246551012086
Result 99.32244604801753 115.51627522474776 0.26390870649556897 1197.0784541108762
Source 77.84588764112225 106.26653000683680 0.8993680380416506
Result 92.056472465294222 104.7351804949812 0.6267841804163137 96.26525021102748
Source 83.6541497276815 168.1156042191305 0.8832239749958186
Result 15.22777434659316 123.98999749097712 0.029981609684065824 418.0598774467112
Source 73.83320596007298 164.02169182468552 0.704852381985975
Result 98.08477784045078 180.0600840012844 0.9113397228943912 367.6234308247471
Source 51.39768352685186 179.7730867879926 0.1009843591101679
Result 42.26675858778375 101.0159121533252 0.54756280039313 208.4519761453227
Source 53.598087927447689 151.9878261225833 0.4392539795066984
Result 62.623170779988804 112.86482236813245 0.212130087062264 267.8642193873781
Source 42.1579902650143 26.0616528165782 0.09142461878671393
Result 56.4652262112628 148.25645960353714 0.17686627978020808 309.8559366314625

```

The status bar at the bottom indicates 'Process finished with exit code 0'.

(скрин 13)

Вообще все меняется... Ладно, попробуем сделать базовую какую-нибудь скажем так, проверку, добавим простейший Boolean под названием lock в класс Task(см скрин 14)

The screenshot shows the IntelliJ IDEA interface with the project 'Lab6' open. The code editor displays Task.java with the following content:

```

public class Task {
    private Function function;
    private double stepIntegration;
    private double leftX, rightX;
    private int iterations;
    private int currentIteration = 0;
    private boolean lock = false;

    public boolean isLock() { return lock; }

    public void setLock(boolean lock) { this.lock = lock; }

    public Task() {}

    public Task(Function function, double stepIntegration, double leftX, double rightX) {
        this.function = function;
        this.stepIntegration = stepIntegration;
        this.leftX = leftX;
    }
}

```

The terminal window shows the execution results:

```

RESULT 13.100082339833685 110.5932488677314 0.9303684566521761
Source 51.36706687918552 186.71742578173842 0.06046096847104565 301.6106360024559
Source 67.32551237844491 123.97048134458161 0.6850246551012086
Result 99.32244604801753 115.51627522474776 0.26390870649556897 1197.0784541108762
Source 77.84588764112225 106.26653000683680 0.8993680380416506
Result 92.056472465294222 104.7351804949812 0.6267841804163137 96.26525021102748
Source 83.6541497276815 168.1156042191305 0.8832239749958186
Result 15.22777434659316 123.98999749097712 0.029981609684065824 418.0598774467112
Source 73.83320596007298 164.02169182468552 0.704852381985975
Result 98.08477784045078 180.0600840012844 0.9113397228943912 367.6234308247471
Source 51.39768352685186 179.7730867879926 0.1009843591101679
Result 42.26675858778375 101.0159121533252 0.54756280039313 208.4519761453227
Source 53.598087927447689 151.9878261225833 0.4392539795066984
Result 62.623170779988804 112.86482236813245 0.212130087062264 267.8642193873781
Source 42.1579902650143 26.0616528165782 0.09142461878671393
Result 56.4652262112628 148.25645960353714 0.17686627978020808 309.8559366314625

```

The status bar at the bottom indicates 'Process finished with exit code 0'.

(скрин 14)

И отредактируем классы(runnable). Скринь 15-16

The screenshot shows a Java project named 'Lab6' in an IDE. The 'SimpleGenerator.java' file is open, containing the following code:

```
import functions.basic.Log;

public class SimpleGenerator implements Runnable {
    private final Task task;
    public SimpleGenerator(Task task) {
        this.task = task;
    }
    @Override
    public void run() {
        while (task.getIterations() != task.getCurrentIteration()) {
            while (!task.isLock());
            task.setLock(true);

            task.setFunction(new Log( base:Math.random() * 9 + 1));
            task.setLeftX(Math.random() * 100);
            task.setRightX(100 + Math.random() * 100);
            task.setStepIntegration(Math.random());

            task.setLock(false);
        }
    }
}
```

The code defines a class `SimpleGenerator` that implements the `Runnable` interface. It has a private field `task` and a constructor that takes a `Task` object. The `run` method contains a loop that continues until the current iteration equals the total iterations. Inside the loop, it checks if the task is locked, sets it to true, generates a new function using a `Log` class, sets the left and right x-coordinates, and sets the step integration value. Finally, it releases the lock.

(скрин 15)

The screenshot shows the same Java project 'Lab6'. The 'SimpleIntegrator.java' file is now open, containing the following code:

```
public class SimpleIntegrator implements Runnable {
    private final Task task;
    public SimpleIntegrator(Task task) {
        this.task = task;
    }
    @Override
    public void run() {
        while (task.getCurrentIteration() != task.getIterations()) {
            while (!task.isLock());
            task.setLock(true);

            System.out.println("Source " + task.getLeftX() + " " + task.getRightX());
            double value = Functions.findIntegral(task.getFunction(), task.getLeftX());
            System.out.println("Result " + task.getLeftX() + " " + task.getRightX());
            task.incrementIteration();

            task.setLock(false);
        }
    }
}
```

This code defines a class `SimpleIntegrator` that implements the `Runnable` interface. It has a private field `task` and a constructor that takes a `Task` object. The `run` method contains a loop that continues until the current iteration equals the total iterations. Inside the loop, it checks if the task is locked, sets it to true, prints the source coordinates, calculates the integral using a `Functions` class, prints the result, increments the iteration, and then releases the lock.

(скрин 16)

Запустим, взглянем(скрины 17-18)

```

import functions.Functions;
import functions.basic.Log;
import threads.SimpleGenerator;
import threads.SimpleIntegrator;
import threads.Task;

public class Main {
    public static void main(String[] args) {
        //nonThread();
        simpleThreads();
    }

    Usage
    public static void simpleThreads() {
        Task t = new Task();
        t.setIterations(666);

        SimpleIntegrator si = new SimpleIntegrator(t);
    }
}

```

Source 16.2509337116301 112.95685005083931 0.03282070409131088
Result 60.651841230870765 155.77792941831802 0.046255525474241166 309.3758110323449
Source 72.93035516692983 137.180109998588 0.20196806476178208
Result 99.14756004478399 118.1449847332057 0.27717725068535003 244.91089448391193
Source 96.79582812941199 143.2412925421196 0.6578886757398849
Result 42.399903906506175 149.08460680621718 0.8480639897668821 416.462916505076
Source 96.11150835997748 122.136709122468519 0.6722003585088391
Result 25.093701735214857 134.11307578032427 0.7274983238918562 223.25777019493924

Process finished with exit code 0

(скрин 17, не помогло)

```

import functions.Functions;
import functions.basic.Log;
import threads.SimpleGenerator;
import threads.SimpleIntegrator;
import threads.Task;

public class Main {
    public static void main(String[] args) {
        //nonThread();
        simpleThreads();
    }

    Usage
    public static void simpleThreads() {
        Task t = new Task();
        t.setIterations(666);

        SimpleIntegrator si = new SimpleIntegrator(t);
    }
}

```

Source 89.66091276420217 102.20435251489754 0.05927810344920681
Result 89.66091276420217 102.20435251489754 0.05927810344920681 28.233183718206618
Source 89.66091276420217 102.20435251489754 0.05927810344920681
Result 89.66091276420217 102.20435251489754 0.05927810344920681 28.233183718206618
Source 89.66091276420217 102.20435251489754 0.05927810344920681
Result 89.66091276420217 102.20435251489754 0.05927810344920681 28.233183718206618
Source 89.66091276420217 102.20435251489754 0.05927810344920681
Result 89.66091276420217 102.20435251489754 0.05927810344920681 28.233183718206618

Process finished with exit code 0

(скрин 18, хуже стало)

Нет. В общем с этим применять – плохая затея. Возможно и сработает, возможно и нет. Перестановки приоритетов тоже ничего не дает. Еще один раз было, что все зависло – бесконечный цикл...

Попробуем synchronized использовать(см скрины 19-20)

The screenshot shows a Java IDE interface with the following details:

- Project:** Lab6
- File:** SimpleGenerator.java
- Code Content:**

```
package threads;

import functions.basic.Log;

public class SimpleGenerator implements Runnable {
    private final Task task;
    public SimpleGenerator(Task task) {
        this.task = task;
    }
    @Override
    public void run() {
        while (task.getCurrentIteration() != task.getIterations()) {
            synchronized (task) {
                task.setFunction(new Log( base: Math.random() * 9 + 1));
                task.setLeftX(Math.random() * 100);
                task.setRightX(100 + Math.random() * 100);
                task.setStepIntegration(Math.random());
            }
        }
    }
}
```

- Right Panel:** Readme (4).md
- Bottom Status Bar:** 22:1 CRLF UTF-8 4 spaces

(скрин 19)

The screenshot shows a Java IDE interface with the following details:

- Project:** Lab6
- File:** SimpleIntegrator.java
- Code Content:**

```
package threads;

import functions.Functions;

public class SimpleIntegrator implements Runnable {
    private final Task task;
    public SimpleIntegrator(Task task) {
        this.task = task;
    }
    @Override
    public void run() {
        while (task.getCurrentIteration() != task.getIterations()) {
            synchronized (task) {
                System.out.println("Source " + task.getLeftX() + " " + task.getRightX());
                double value = Functions.findIntegral(task.getFunction(), task.getLeftX(),
                        task.getRightX());
                System.out.println("Result " + task.getLeftX() + " " + task.getRightX());
                task.incrementIteration();
            }
        }
    }
}
```

- Right Panel:** Readme (4).md
- Bottom Status Bar:** 18:43 CRLF UTF-8 4 spaces

(скрин 20)

Запустим, лучше не стало(см скрин 21)

```

public class SimpleGenerator implements Runnable {
    private final Task task;
    private int iterations;
    private Function function;

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

    @Override
    public void run() {
        while (!task.getIterations() != task.getCurrentIteration()) {
            synchronized (task) {
                task.setFunction(new Log( base: Math.random() * 9 + 1));
                task.setLeftX(Math.random() * 100);
                task.setRightX(100 + Math.random() * 100);
                task.setStepIntegration(Math.random());
            }
        }
    }
}

```

The screenshot shows the IntelliJ IDEA interface with the SimpleGenerator.java file open. The code implements the Runnable interface and runs a loop until the task's getIterations() method returns false. It uses synchronized blocks and random numbers for step integration.

(скрин 21)

Точней, как, обновляются данные корректно, но мало. Всего лишь пару раз. Давайте попробуем серьезно изменить логику(см скрин 22-24)

```

public class Task {
    private Function function;
    private double stepIntegration;
    private double leftX, rightX;
    private int iterations;
    private int currentIteration = 0;
    private boolean isExecuted = true;

    public Task() {}

    public Task(Function function, double stepIntegration, double leftX, double rightX) {
        this.function = function;
        this.stepIntegration = stepIntegration;
        this.leftX = leftX;
        this.rightX = rightX;
        this.iterations = iterations;
    }

    public void incrementIteration() {
        this.currentIteration++;
        isExecuted = true;
    }

    public Function getFunction() { return function; }
}

```

The screenshot shows the IntelliJ IDEA interface with the Task.java file open. The Task class has a flag 'isExecuted' which is set to true in the incrementIteration() method. This flag is used to check if the task has been generated.

(скрин 22, переменная-флаг, отвечает за то, что была ли сгенерирована задача или нет(true – значит не сгенерирована новая задача))

```
public class SimpleGenerator implements Runnable {
    private final Task task;
    private boolean running = true;

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

    @Override
    public void run() {
        while (running) {
            synchronized (task) {
                if (!task.isExecuted()) {
                    task.setFunction(new Log( base: Math.random() * 9 + 1));
                    task.setLeftX(Math.random() * 100);
                    task.setRightX(100 + Math.random() * 100);
                    task.setStepIntegration(Math.random());
                    task.setExecuted(true);
                }
            }
            running = task.getCurrentIteration() != task.getIterations();
        }
    }
}
```

Задание 4

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

Для устранения этой проблемы потребуется дополнительный объект, представляющий собой односемафор, различающий операции чтения и записи в защищаемый объект (пример класса таких объектов был приведён в лекции).

В пакете threads создайте два следующих класса. При их реализации воспользуйтесь фрагментами классов из предыдущего задания.

Класс Generator должен расширять класс Thread, получать в конструкторе и сохранять в своем поле ссылки на объект типа Task, и на объект семафора, а в методе run() должны выполняться те же блоки синхронизации в методах run() генерирующего и интегрирующего потоков. Убедитесь, что в коде никогда не возникает необоснованно длительной блокировки объектов.

(скрин 23)

```
import functions.Functions;

public class SimpleIntegrator implements Runnable {
    private final Task task;
    private boolean running = true;

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

    @Override
    public void run() {
        while (running) {
            synchronized (task) {
                if (!task.isExecuted()) {
                    System.out.println("Source " + task.getLeftX() + " " + task.getRightX());
                    double value = Functions.findIntegral(task.getFunction(), task.getLeftX(), task.getRightX());
                    System.out.println("Result " + task.getLeftX() + " " + task.getRightX());
                    task.incrementIteration();
                }
            }
            running = task.getCurrentIteration() != task.getIterations();
        }
    }
}
```

Задание 4

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

Для устранения этой проблемы потребуется дополнительный объект, представляющий собой односемафор, различающий операции чтения и записи в защищаемый объект (пример класса таких объектов был приведён в лекции).

В пакете threads создайте два следующих класса. При их реализации воспользуйтесь фрагментами классов из предыдущего задания.

Класс Generator должен расширять класс Thread, получать в конструкторе и сохранять в своем поле ссылки на объект типа Task, и на объект семафора, а в методе run() должны выполняться те же блоки синхронизации в методах run() генерирующего и интегрирующего потоков. Убедитесь, что в коде никогда не возникает необоснованно длительной блокировки объектов.

(скрин 24)

Запустим...(см скрин 25)

The screenshot shows a Java development environment with the following details:

- Project Structure:** Lab6 > src > Main > simpleThreads
- Main.java:** Contains the main class Main with a static method simpleThreads(). It creates a SimpleIntegrator and a SimpleGenerator, starts them in new threads, and prints their results.
- Terminal Output:** Shows the execution of the program, printing multiple source and result pairs. The results are floating-point numbers ranging from 11.19027563628673 to 295.374758775906.
- Right Panel:** Displays a Readme (4).md file with instructions about running the program and handling NullPointerExceptions.

(скрин 25)

О чудо, заработало. Попробуем поставить между запуском 1 сек задержки(см скрин 26)

The screenshot shows the same Java development environment as before, but with a modification to the code:

- Main.java:** The static method simpleThreads() now includes a Thread.sleep(1000) call after the threads are started.
- Terminal Output:** Shows the execution of the program followed by the message "Process finished with exit code 0".
- Right Panel:** Displays a Readme (4).md file with instructions about running the program and handling NullPointerExceptions.

(скрин 26)

Решительность программы это не поколебало.

Ошибка Null{} тоже была(когда отдельно ставил задержку искусственно), когда запускали генератор после исполнителя, но у меня компьютер слишком быстро работает...

Задание выполнено

Задание 4

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

Честно, это нигде не отображалось...

Я даже расставил номера(см скрин 27)

The screenshot shows an IDE interface with a dark theme. On the left, the project structure for 'Lab6' is visible, showing a 'src' folder containing 'functions', 'basic', 'meta', 'ArrayTabulator', 'Function', 'FunctionPair', 'Inappropriate', 'LinkedListTable', and 'TabulatedFunction'. A file named 'Main.java' is open in the editor, containing the following code:

```
public class Main {
    simpleThreads();
}

public static void simpleThreads() {
    Task t = new Task();
    t.setIterations(100);

    SimpleIntegrator si = new SimpleIntegrator(t);
    SimpleGenerator sg = new SimpleGenerator(t);

    new Thread(si).start();
    try {
        Thread.sleep( millis: 1000 );
    } catch (Exception err) { err.printStackTrace(); }
    new Thread(sg).start();
}
```

The 'Run' tab is selected, and the output window shows the results of the execution:

```
Source 85.80709499511099 189.83490213548288 0.0930452851609162
#96. Result 85.80709499511099 189.83490213548288 0.0930452851609162 358.66601089312434
Source 56.00800400121011 103.19883440387373 0.9910390906114029
#97. Result 56.00800400121011 103.19883440387373 0.9910390906114029 149.96391342406082
Source 77.92651198450338 172.3095941864056 0.06090887994755523
#98. Result 77.92651198450338 172.3095941864056 0.06090887994755523 525.9438363332786
Source 47.201572396617934 123.05517421708932 0.16015703166550377
#99. Result 47.201572396617934 123.05517421708932 0.16015703166550377 877.0741520184861
```

Below the output, a message says 'Process finished with exit code 0'.

(скрин 27)

Проверил вручную, пропусков нигде не было...

Несколько перезапусков тоже результата не дало

Чисто технически, я встроил семафор(причем исключительно для двух классов) в Task...

Но это не то, что делает настоящий семафор(кстати, по ней у нас лекций не было)

Добавим классы(см скрин 28-29)

```

public class Generator extends Thread {
    private final Semaphore semaphore;
    private boolean running = true;

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

    @Override
    public void run() {
        while (running) {
            semaphore.acquireUninterruptibly();
            if (task.isExecuted()) {
                task.setFunction(new Log( base: Math.random() * 9 + 1));
                task.setLeftX(Math.random() * 100);
                task.setRightX(100 + Math.random() * 100);
                task.setStepIntegration(Math.random());
            }
            task.setExecuted(false);
        }
        running = task.getIterations() != task.getCurrentIteration();
        semaphore.release();
    }

    @Override
    public void interrupt() {
        this.running = false;
    }
}

```

(скрин 28)

```

public class Integrator extends Thread {
    private Task task;
    private Semaphore semaphore;
    private boolean running = true;

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

    @Override
    public void run() {
        while (running) {
            semaphore.acquireUninterruptibly();
            System.out.println("Source " + task.getLeftX() + " " + task.getRightX() +
                double value = Functions.findIntegral(task.getFunction(), task.getLeftX(),
                System.out.println(" " + task.getCurrentIteration() + ". Result " + task.x);

            task.incrementIteration();
            running = task.getCurrentIteration() != task.getIterations();

            semaphore.release();
        }
    }

    @Override
    public void interrupt() {
        this.running = false;
    }
}

```

(скрин 29)

Отредактируем Main и запустим...(см скрин 30)

The screenshot shows a Java IDE interface with a dark theme. On the left, the project structure for 'Lab6' is visible, showing files like Main.java, Generator.java, and Integrator.java. The main editor window displays Main.java with the following code:

```
public class Main {
    public static void main(String[] args) {
        //nonthreads();
        //simpleThreads();
        complicatedThreads();
    }
    usage
    public static void complicatedThreads() {
        Task t = new Task();
        t.setIterations(1000);
        Semaphore s = new Semaphore(permits: 1, fair: true);

        Integrator i = new Integrator(t, s);
        Generator g = new Generator(t, s);

        g.start();
        i.start();
    }
}
```

The 'Run' tab is selected, and the output window shows several execution results:

```
Source 74.33809540369796 157.0927942568758 0.28463251698264513
#996. Result 74.33809540369796 157.0927942568758 0.28463251698264513 544.549286656211
Source 11.84049404575747 197.47625425207073 0.049727583275349097 633.5149730889466
#997. Result 11.84049404575747 197.47625425207073 0.049727583275349097 633.5149730889466
Source 71.57232492334406 156.81635383851653 0.993217297430123
#998. Result 71.57232492334406 156.81635383851653 0.993217297430123 366.01520578043556
Source 10.738212874346454 143.75896079280247 0.7313175167388989
#999. Result 10.738212874346454 143.75896079280247 0.7313175167388989 263.87959302205235
```

The status bar at the bottom indicates 'Process finished with exit code 0'.

(скрин 30)

Ничего не меняется в принципе, все также выполняет

Изменим и запустим(скрин 31)

This screenshot is nearly identical to the previous one, showing the same Java code and execution results. The key difference is in the 'complicatedThreads()' method where a 'Thread.sleep()' call has been added:

```
public static void complicatedThreads() throws Exception{
    Task t = new Task();
    t.setIterations(1000);
    Semaphore s = new Semaphore(permits: 1, fair: true);

    Integrator i = new Integrator(t, s);
    Generator g = new Generator(t, s);

    g.start();
    i.start();

    Thread.sleep(millis: 50);
    g.interrupt();
    i.interrupt();
}
```

The output window shows the same results as before, indicating no significant change in behavior.

(скрин 31)

В принципе, ничего не изменилось... Кроме одного, потоки не останавливаются...

Придется добавлять еще Boolean, см скрины 32-33

```
public class Generator extends Thread {
    private final Semaphore semaphore;
    private boolean running = true;
    private boolean interrupted = false;

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

    @Override
    public void run() {
        while (running && !interrupted) {
            semaphore.acquireUninterruptibly();
            if (task.isExecuted()) {
                task.setFunction(new Log(1, Math.random() * 9 + 1));
                task.setLeftX((Math.random() * 100);
                task.setRightX((Math.random() * 100));
                task.setStepIntegration(Math.random());
                task.setExecuted(false);
            }
            running = task.getCurrentIteration() != task.getIterations();
            semaphore.release();
        }
    }

    @Override
    public void interrupt() {
        this.interrupted = true;
    }
}
```

(скрин 22)

```
public class Integrator extends Thread {
    private final Task task;
    private boolean running = true;
    private boolean interrupted = false;

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

    @Override
    public void run() {
        while (running && !interrupted) {
            semaphore.acquireUninterruptibly();
            if (!task.isExecuted()) {
                System.out.println("Source " + task.getLeftX() + " " + task.getRightX());
                double value = Functions.findIntegral(task.getFunction(), task.getLeftX(), task.getRightX());
                System.out.println("[" + task.getLeftX() + ", " + task.getRightX() + "] " + value);
                task.incrementIteration();
            }
            running = task.getCurrentIteration() != task.getIterations();
            semaphore.release();
        }
    }

    @Override
    public void interrupt() {
        this.interrupted = true;
    }
}
```

(скрин 23)

В принципе, если запустить, то программа оборвётся(см скрин 24)

```

import functions.Functions;
import functions.basic.Log;
import threads.*;

import java.util.concurrent.Semaphore;

public class Main {
    public static void main(String[] args) throws Exception {
        //nonThread();
        //simpleThreads();
        complicatedThreads();
    }

    Usage
    public static void complicatedThreads() throws Exception{
        Task t = new Task();
        t.setIterations(10000);
        Semaphore s = new Semaphore(1, fair: true);

        for (int i = 0; i < t.getIterations(); i++) {
            try {
                s.acquire();
                System.out.println("Source " + t.getLeftX() + " " + t.getRightX());
                double value = Functions.findIntegral(t.getFunction(), t.getLeftX(), t.getRightX());
                System.out.println("Result " + value);
                t.incrementIteration();
            } catch (Exception e) {
                e.printStackTrace();
            } finally {
                s.release();
            }
        }
    }
}

```

Source 4.846946455745571 100.90104985493838 0.1804153531505771
#1055. Result 4.846946455745571 100.90104985493838 0.1804153531505771 171.89110206418167
#1056. Result 29.111127723765573 141.43425230611356 0.9188539250896437 6482.931794954456
#1057. Result 28.45562185044527 189.60376084472455 0.004506929240590041 344.8742329653515
#1058. Result 5.191521391525978 110.25677266381983 0.6528382316275122 230.5192301771384
#1059. Result 5.191521391525978 110.25677266381983 0.6528382316275122 230.5192301771384

Process finished with exit code 0

(скрин 24)

И я только сейчас понял, в чем заключалась ошибка. Внесу правки...(см скрин 25)

```

public class Integrator extends Thread {
    private final Semaphore semaphore;
    16 usages
    private final Task task;
    2 usages
    private boolean running = true;
    3 usages
    public Integrator(Task task, Semaphore semaphore) {
        this.task = task;
        this.semaphore = semaphore;
    }
    @Override
    public void run() {
        while (running && !isInterrupted()) {
            semaphore.acquireUninterruptibly();
            if (!task.isExecuted()) {
                System.out.println("Source " + task.getLeftX() + " " + task.getRightX());
                double value = Functions.findIntegral(task.getFunction(), task.getLeftX(), task.getRightX());
                System.out.println("Result " + value);
                task.incrementIteration();
            }
            running = task.getCurrentIteration() != task.getIterations();
            semaphore.release();
        }
    }
    @Override
    public void interrupt() {
        super.interrupt();
    }
}

```

Source 4.846946455745571 100.90104985493838 0.1804153531505771
#1055. Result 4.846946455745571 100.90104985493838 0.1804153531505771 171.89110206418167
#1056. Result 29.111127723765573 141.43425230611356 0.9188539250896437 6482.931794954456
#1057. Result 28.45562185044527 189.60376084472455 0.004506929240590041 344.8742329653515
#1058. Result 5.191521391525978 110.25677266381983 0.6528382316275122 230.5192301771384
#1059. Result 5.191521391525978 110.25677266381983 0.6528382316275122 230.5192301771384

Process finished with exit code 0

(скрин 25)

Во втором аналогично тоже самое...

Запускаем, ничего не изменилось, поток также прерывается.

Задание выполнено

Из нового для себя вынес – семафоры. Спасибо вам большое за предоставленные возможности изучения

Лабораторная работа была выполнена.