Министерство науки и высшего образования Российской Федерации

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

Институт информатики и кибернетики

Кафедра технической кибернетики

Отчет по лабораторной работе №6

Дисциплина: «ООП»

Тема «Многопоточное приложение»

Выполнил: Чечеткин Д.А.

Группа: 6201-120303D

Самара, 2025

Задание 1

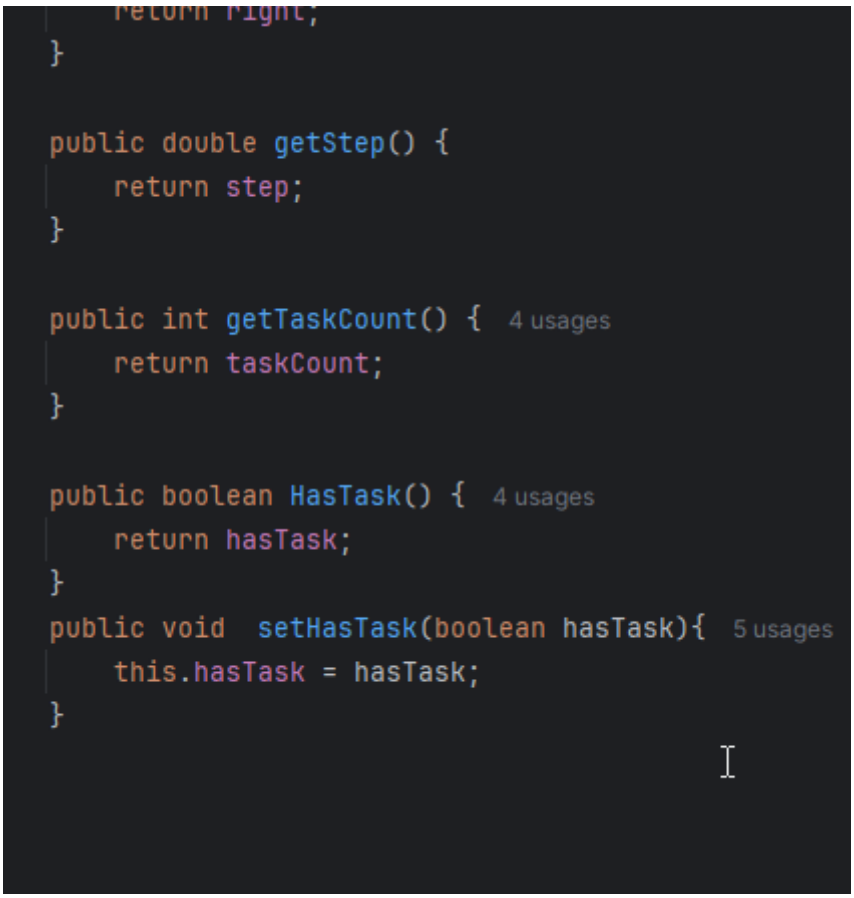
В класс **Functions** был добавлен метод, возвращающий значение интеграла функции, вычисленное с помощью численного метода. Метод получает ссылку типа Function на объект функции, значения границ и шаг дискретизации.

```
public static double integrate(Function f, double left, double right, double step) {  3 usages
    if (step <= 0) {
        throw new IllegalArgumentException("Шаг должен быть положительным");
    }
    if (left > right) {
        throw new IllegalArgumentException("Левая граница больше правой");
    }

    if (left < f.getLeftDomainBorder() || right > f.getRightDomainBorder()) {
        throw new IllegalArgumentException("Интервал интегрирования выходит за границы области определения функции");
    }

    double sum = 0.0;
    double x = left;

    while (x + step < right) {
        double y1 = f.getFunctionValue(x);
        double y2 = f.getFunctionValue( x: x + step);
        sum += 0.5 * (y1 + y2) * step;
        x += step;
    }

    if (x < right) {
        double y1 = f.getFunctionValue(x);
        double y2 = f.getFunctionValue(right);
        sum += 0.5 * (y1 + y2) * (right - x);
    }

    return sum;
}
```

Задание 2

Создан пакет **threads,** в котором описан класс **Task**, содержащий ссылку на объект интегрируемой функции, границы интегрирования, шаг дискретизации, количество выполняемых заданий.

```java
package threads;
import functions.*;
public class Task {  14 usages
    private Function function;  2 usages
    private double left;  2 usages
    private double right;  2 usages
    private double step;  2 usages
    private int taskCount;  2 usages
    private boolean hasTask = false;  2 usages

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

    public void setLeft(double left) {
        this.left = left;
    }

    public void setRight(double right) {
        this.right = right;
    }

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

    public void setTaskCount(int taskCount) {  3 usages
        this.taskCount = taskCount;
    }

    public Function getFunction() {
        return function;
    }

    public double getLeft() {
        return left;
    }

    public double getRight() {
        return right;
```



Описан метод **nonThread**, реализующий последовательную версию программы.

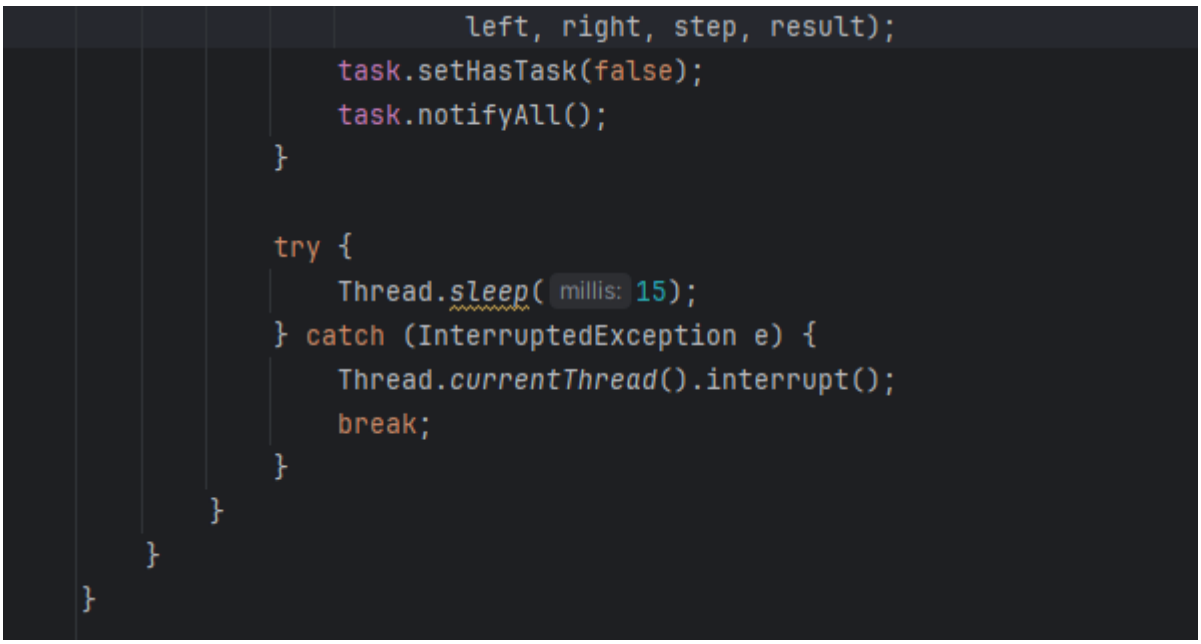
```java
public static void nonThread() {  1 usage
    System.out.println("\nnonThread");

    Task task = new Task();
    int taskCount = 100;
    task.setTaskCount(taskCount);

    for (int i = 0; i < taskCount; i++) {
        double base = 1 + random.nextDouble() * 9;
        double left = random.nextDouble() * 100;
        double right = 100 + random.nextDouble() * 100;
        double step = random.nextDouble();
        if (step == 0) step = 0.0001;

        task.setFunction(new Log(base));
        task.setLeft(left);
        task.setRight(right);
        task.setStep(step);

        System.out.printf("Source %.5f %.5f %.5f%n", left, right, step);

        double result = Functions.integrate(task.getFunction(), left, right, step);
        System.out.printf("Result %.5f %.5f %.5f %.10f%n", left, right, step, result);
    }
}
```

Задание 3

В пакете **threads** созданы классы **simpleGenerator** и **simpleIntegrator,** реализующие интерфейс **Runnable,** формирующие и решающие задачи.

```java
package threads;

import functions.basic.Log;
import java.util.Random;

public class SimpleGenerator implements Runnable {  1 usage
    private final Task task;  11 usages
    private final Random rnd = new Random();  4 usages

    public SimpleGenerator(Task task) {  1 usage
        this.task = task;
    }

    @Override
    public void run() {
        for (int i = 0; i < task.getTaskCount(); i++) {
            double base = 1 + rnd.nextDouble() * 9;
            double left = rnd.nextDouble() * 100;
            double right = 100 + rnd.nextDouble() * 100;
            double step = rnd.nextDouble();
            if (step == 0) step = 0.0001;

            synchronized (task) {
                while (task.HasTask()) {
                    try {
                        task.wait();
                    } catch (InterruptedException e) {
                        Thread.currentThread().interrupt();
                        return;
                    }
                }
                task.setFunction(new Log(base));
                task.setLeft(left);
                task.setRight(right);
                task.setStep(step);
                task.setHasTask(true);

                System.out.printf("Generated: Source %.5f %.5f %.5f%n",
                        left, right, step);
                task.notifyAll();
            }

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

```java
package threads;

import functions.Function;
import functions.Functions;

public class SimpleIntegrator implements Runnable {  1 usage
    private final Task task;  12 usages

    public SimpleIntegrator(Task task) {  1 usage
        this.task = task;
    }

    @Override
    public void run() {
        for (int i = 0; i < task.getTaskCount(); i++) {
            Function func;
            double left, right, step;

            synchronized (task) {
                while (!task.HasTask()) {
                    try {
                        task.wait();
                    } catch (InterruptedException e) {
                        Thread.currentThread().interrupt();
                        return;
                    }
                }

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

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

            synchronized (task) {
                System.out.printf("Integrated: Result %.5f %.5f %.5f %.10f%n",
                        left, right, step, result);
                task.setHasTask(false);
```



Создан метод **simpleThreads**, запускающий потоки, основанные на классах **simpleGenerator** и **simpleIntegrator.**

```java
public static void simpleThreads() {  1 usage
    System.out.println("\nsimpleThreads");

    Task task = new Task();
    task.setTaskCount(100);
    task.setHasTask(false);

    Thread generator = new Thread(new SimpleGenerator(task));
    Thread integrator = new Thread(new SimpleIntegrator(task));

    // Приоритеты
    // ВАРИАНТ 1
    generator.setPriority(Thread.MAX_PRIORITY);
    integrator.setPriority(Thread.MIN_PRIORITY);

    // ВАРИАНТ 2
    //generator.setPriority(Thread.MIN_PRIORITY);
    //integrator.setPriority(Thread.MAX_PRIORITY);

    // ВАРИАНТ 3
    //generator.setPriority(Thread.NORM_PRIORITY);
    //integrator.setPriority(Thread.NORM_PRIORITY);

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

    try {
        generator.join();
        integrator.join();
    } catch (InterruptedException e) {
        Thread.currentThread().interrupt();
    }
}
```

Задание 4

Созданы классы **Generator** и **Integrator,** расширяющие класс **Threads ,** выполняющие те же действия, что и предыдущие версии и использующие возможности семафора.

```java
public class Generator extends Thread {  2 usages
    private final Task task;  8 usages
    private final Semaphore semaphore;  3 usages
    private final Random rnd = new Random();  4 usages

    public Generator(Task task, Semaphore semaphore) {  1 usage
        this.task = task;
        this.semaphore = semaphore;
    }

    @Override
    public void run() {
        try {
            for (int i = 0; i < task.getTaskCount(); i++) {
                double base = 1 + rnd.nextDouble() * 9;
                double left = rnd.nextDouble() * 100;
                double right = 100 + rnd.nextDouble() * 100;
                double step = rnd.nextDouble();
                if (step == 0) step = 0.0001;

                while (task.HasTask()) {
                    Thread.sleep( millis: 1);
                }

                semaphore.beginWrite();
                task.setFunction(new Log(base));
                task.setLeft(left);
                task.setRight(right);
                task.setStep(step);
                task.setHasTask(true);
                semaphore.endWrite();

                System.out.printf("Generated: Source %.5f %.5f %.5f%n", left, right, step);
                Thread.sleep( millis: 10);
            }
        } catch (InterruptedException e) {
            interrupt();
        }
    }
}
```

```java
package threads;

import functions.Function;
import functions.Functions;

public class Integrator extends Thread {  2 usages
    private final Task task;  8 usages
    private final Semaphore semaphore;  3 usages

    public Integrator(Task task, Semaphore semaphore) {  1 usage
        this.task = task;
        this.semaphore = semaphore;
    }

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

                while (!task.HasTask()) {
                    Thread.sleep( millis: 1);
                }

                semaphore.beginRead();
                Function f = task.getFunction();
                double left = task.getLeft();
                double right = task.getRight();
                double step = task.getStep();
                task.setHasTask(false);
                semaphore.endRead();

                double result = Functions.integrate(f, left, right, step);
                System.out.printf("Integrated: Result %.5f %.5f %.5f %.10f%n",
                        left, right, step, result);
            }
        } catch (InterruptedException e) {
            interrupt();
        }
    }
}
```

Создан метод **ComplicatedThreads**, создающий и реализующий потоки вычислений

```java
public static void complicatedThreads() {  1 usage
    System.out.println("\ncomplicatedThreads ");

    Task task = new Task();
    task.setTaskCount(100);
    Semaphore semaphore = new Semaphore();

    Generator generator = new Generator(task, semaphore);
    Integrator integrator = new Integrator(task, semaphore);
    // Приоритеты
    //  ВАРИАНТ 1
    //generator.setPriority(Thread.MAX_PRIORITY);
    //integrator.setPriority(Thread.MIN_PRIORITY);

    // ВАРИАНТ 2
    //generator.setPriority(Thread.MIN_PRIORITY);
    //integrator.setPriority(Thread.MAX_PRIORITY);

    // ВАРИАНТ 3
    generator.setPriority(Thread.NORM_PRIORITY);
    integrator.setPriority(Thread.NORM_PRIORITY);

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

    try {
        Thread.sleep( millis: 50);

        generator.interrupt();
        integrator.interrupt();
        System.out.println("Потоки прерваны после 50 мс работы");

        generator.join();
        integrator.join();
    } catch (InterruptedException e) {
        Thread.currentThread().interrupt();
    }
}
```

Реализовано прерывание после 50 миллисекунд.

Первые несколько выводов:

```
nonThread
Source 8,90303 139,34824 0,77373
Result 8,90303 139,34824 0,77373 246,8310701941
Source 93,99215 102,05667 0,09469
Result 93,99215 102,05667 0,09469 86,7976271248
Source 98,75961 115,94512 0,34277
Result 98,75961 115,94512 0,34277 102,7796700906
Source 34,47413 136,32723 0,54448
Result 34,47413 136,32723 0,54448 218,8214153684
Source 50,23600 160,64726 0,44974
Result 50,23600 160,64726 0,44974 718,9488634069
Source 43,05161 195,70697 0,81790
Result 43,05161 195,70697 0,81790 459,9709941610
Source 22,08967 119,30041 0,67029
Result 22,08967 119,30041 0,67029 397,4330111347
```

```
simpleThreads
Generated: Source 74,78139 166,14622 0,89932
Integrated: Result 74,78139 166,14622 0,89932 217,6035950447
Generated: Source 97,48060 108,34259 0,79594
Integrated: Result 97,48060 108,34259 0,79594 1732,3491484180
Generated: Source 54,05685 137,46089 0,08604
Integrated: Result 54,05685 137,46089 0,08604 15865,1807783863
Generated: Source 69,86239 151,24799 0,24489
Integrated: Result 69,86239 151,24799 0,24489 191,9829495890
Generated: Source 86,23690 192,76394 0,15413
Integrated: Result 86,23690 192,76394 0,15413 229,2756097072
Generated: Source 35,14517 153,63983 0,36190
Integrated: Result 35,14517 153,63983 0,36190 254,5019595943
Generated: Source 23,11629 165,75779 0,79214
Integrated: Result 23,11629 165,75779 0,79214 461,7817425098
```

```
complicatedThreads
Generated: Source 1,28737 188,01084 0,58620
Integrated: Result 1,28737 188,01084 0,58620 389,8353745823
Generated: Source 38,32127 115,19244 0,32449
Integrated: Result 38,32127 115,19244 0,32449 147,9206299524
Generated: Source 29,19245 166,02361 0,02892
Integrated: Result 29,19245 166,02361 0,02892 489,0938288469
Generated: Source 18,69722 189,15553 0,72130
Integrated: Result 18,69722 189,15553 0,72130 399,2535259706
Generated: Source 38,55859 136,54659 0,29278
Integrated: Result 38,55859 136,54659 0,29278 239,2749156199
Потоки прерваны после 50 мс работы
```