ГУАП

КАФЕДРА № 42

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ _____
ПРЕПОДАВАТЕЛЬ

К.т.н. Доцент		В.А.Ушаков
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №3

СОБЫТИЙНО-УПРАВЛЯЕМАЯ МОДЕЛЬ ПРОГРАММИРОВАНИЯ

Вариант 5

по курсу: КРОССПЛАТФОРМЕННОЕ ПРОГРАММИРОВАНИЕ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №	4128		В. А. Воробьев
		подпись, дата	инициалы, фамилия

Санкт-Петербург 2023

Цель работы: Разработать программу на языке JAVA, использующую графический пользовательский интерфейс для управления моделированием динамического процесса. Параметры моделирования должны задаваться до включения моделирования процесса. После включения элементы интерфейса, отвечающие за параметры моделирования, необходимо заблокировать (при окончании снова разблокировать). Предусмотреть возможность ускорения и замедления времени моделирования для изучения медленно- или быстротекущих процессов. Кроме использования стандартных графических элементов библиотеки Swing, запрограммировать визуализацию динамического процесса с помощью пользовательской графики.

При разработке моделей разрешается использовать эвристические формулы. Использование точных физических формул не требуется.

З**адание:**

Номер в журнале:

Группа 4128

N = 4128 + 5 = 4133

Вариант 6 В начальном положении маятник находится в состоянии покоя. Маятнику можно задавать направления качания в левую или правую сторону с помощью соответствующих элементов управления. В качестве закона качания можно использовать любую эвристическую формулу.

Задаваемые параметры:

- амплитуда раскачивания
- коэффициент затухания колебаний
- коэффициент изменения периода колебаний.

Элементы управления моделью: раскачать влево, раскачать вправо.

Визуализируемые значения: положение маятника (координата) и время с момента начала моделирования.

Результаты работы программы:

В ходе выполнения задания была разработана программа с графическим интерфейсом, который позволяет управлять моделированием маятника. Параметры, такие как амплитуда, коэффициент затухания, скорость изменения периода и коэффициент времени, могут быть заданы пользователем перед началом моделирования.

Программа использует библиотеку Swing для создания графического интерфейса и обеспечивает визуализацию динамического процесса с использованием пользовательской графики. Элементы управления и визуализация взаимодействуют с фоновым потоком, обеспечивая актуальное отображение результатов моделирования.

Основной метод программы, simulatePendulum, реализует эвристическую формулу для моделирования движения маятника. Этот метод используется для вычисления положения маятника в зависимости от времени. Исходный код можно посмотреть в Приложении или на GitHub (URL - https://github.com/vladcto/suai-labs/tree/355629db65b3539b04f89332b026364acbd0016d/5_semester/%D0%9A%D0%9F/src/lab3).

Интерфейс программы представлен на рисунках 1-3.

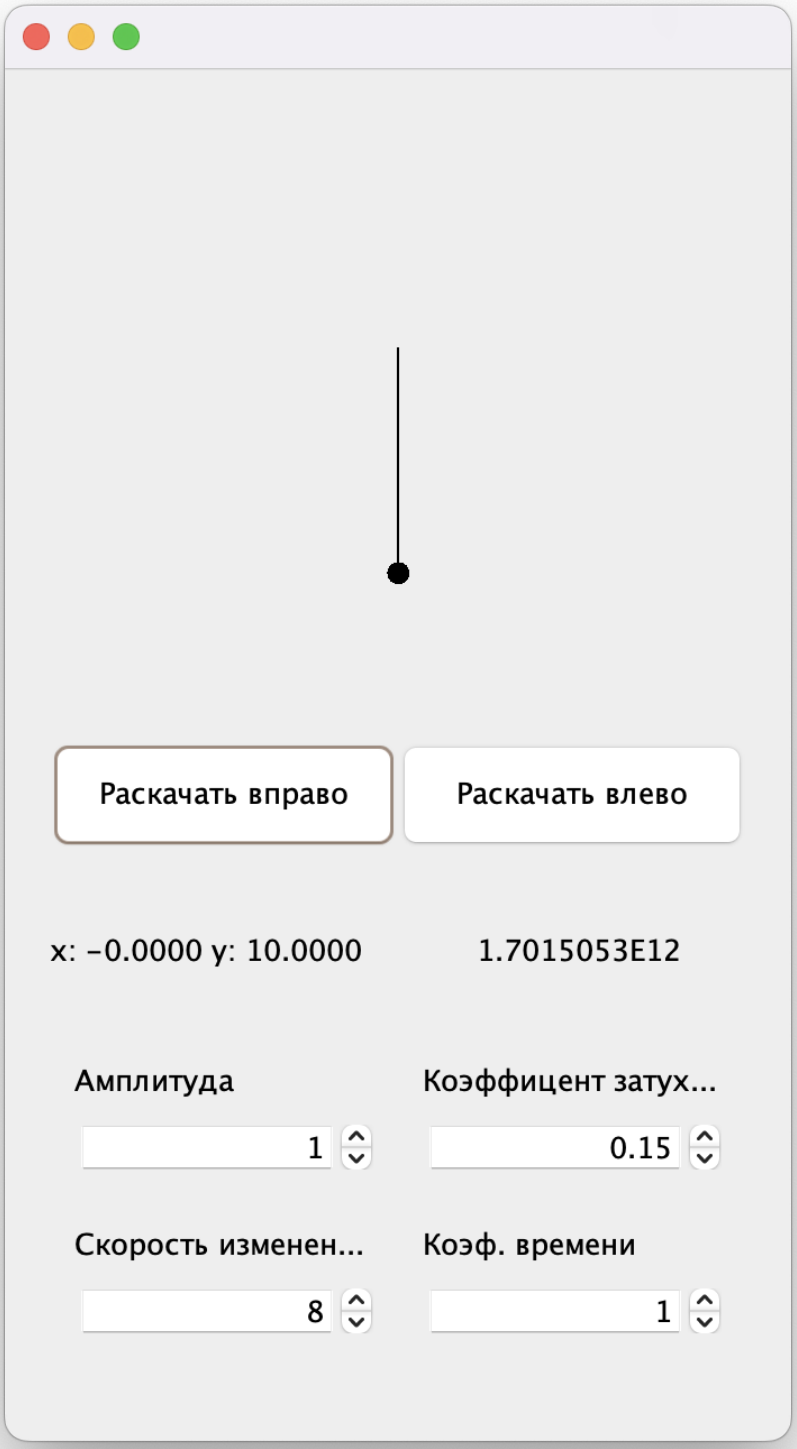

Рисунок 1 – Интерфейс при запуске

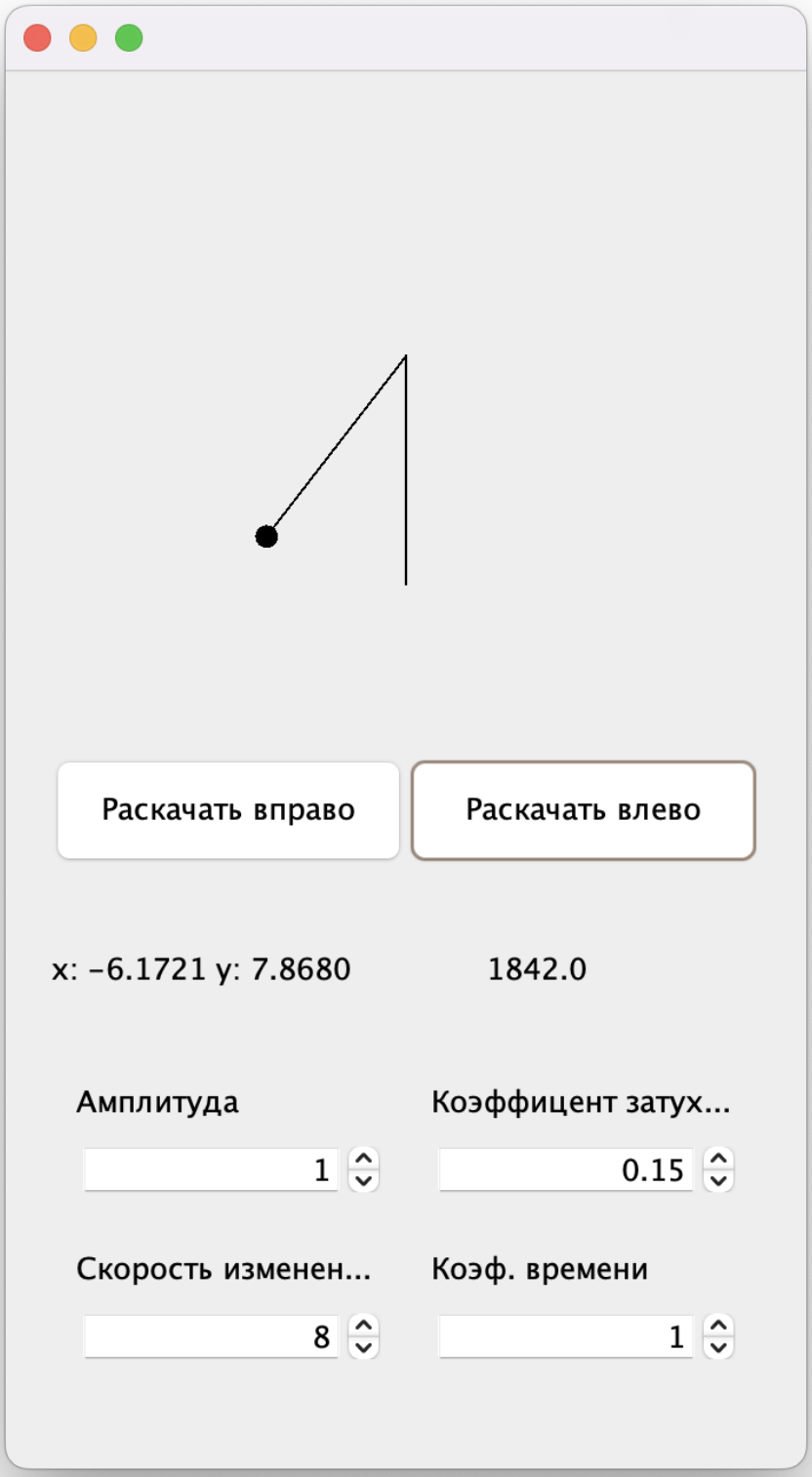

Рисунок 2 – Интерфейс после нажатия “Раскачать влево”

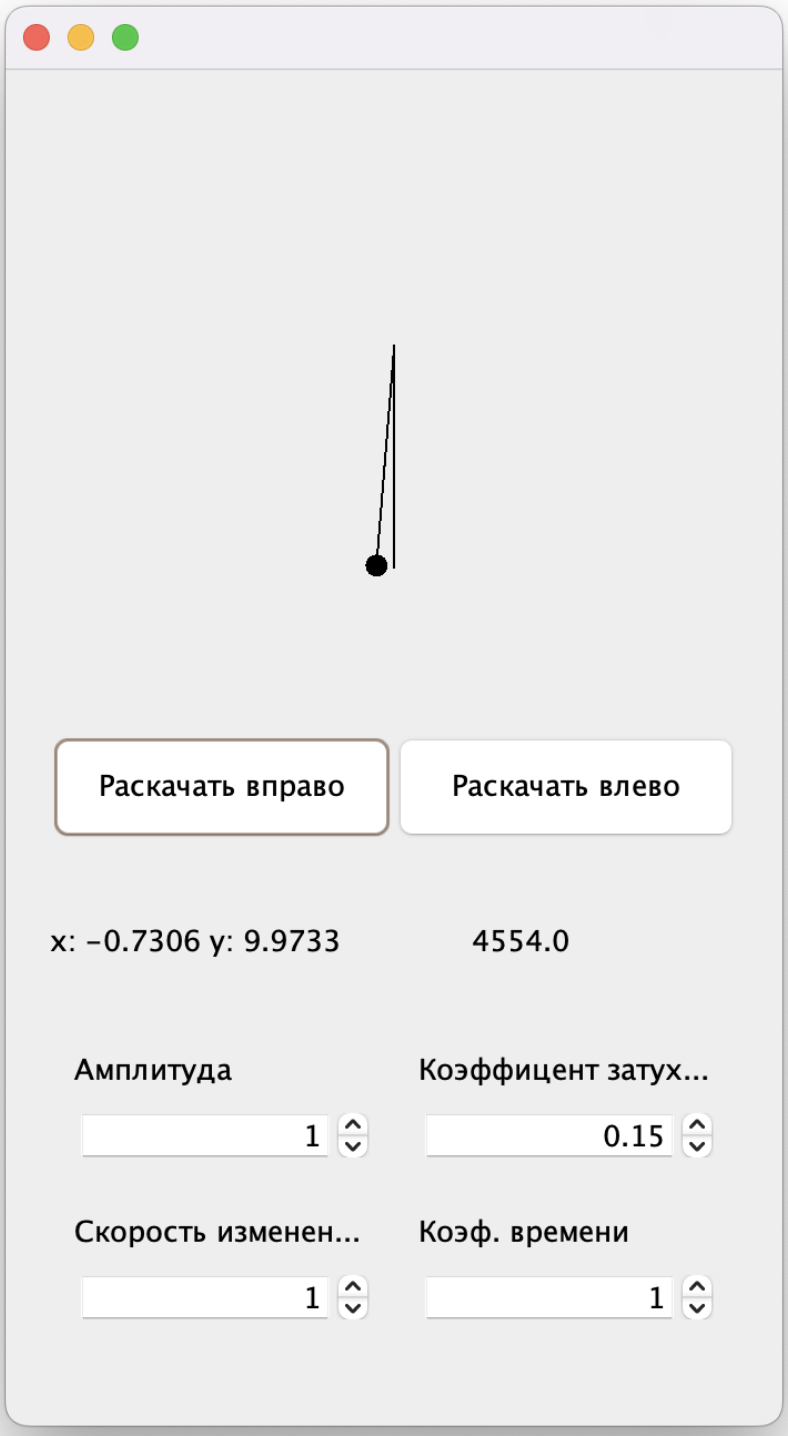

Рисунок 3 – Интерфейс через промежуток времени

Вывод:

В ходе выполнения лабораторной работы мы успешно разработали программу на языке JAVA, обеспечивающую графический пользовательский интерфейс для управления моделированием динамического процесса. Основная цель работы заключалась в создании интуитивно понятного интерфейса, который бы предоставлял пользователю удобные средства управления параметрами моделирования и визуализацию результатов в режиме реального времени.

Важным аспектом разработанной программы является возможность задания параметров моделирования до запуска процесса, а затем блокировки соответствующих элементов интерфейса. Это обеспечивает удобство и предотвращает случайные изменения параметров во время работы программы. Дополнительно, добавленная функциональность ускорения и замедления времени моделирования позволяет более детально изучать процессы с разной динамикой.

Программа также успешно использует пользовательскую графику для визуализации движения маятника, что делает процесс моделирования более наглядным и понятным для пользователя. Таким образом, лабораторная работа не только углубила наши знания в области графического программирования на языке JAVA, но и обеспечила нам опыт в создании программ с управлением динамическими процессами через графический интерфейс.

ПРИЛОЖЕНИЕ

PendulumSimulation.java

```
package lab3;


import javax.swing.*;


import java.awt.*;


public class PendulumSimulation extends JFrame {
    final PendulumManager manager = new PendulumManager(this);
    final JLabel resLabel = new JLabel();
    final JLabel time = new JLabel();
    final JLabel amplitudeLabel = new JLabel("Амплитуда");
    final JLabel dampingLabel = new JLabel("Коэффицент затухания");
    final JLabel periodChangeLabel = new JLabel("Скорость изменения
периода");


    final JSpinner amplitudeSpinner = new JSpinner(new SpinnerNumberModel(1,
0.2, 2, 0.2));
    final JSpinner dampingSpinner = new JSpinner(new SpinnerNumberModel(0.1,
0.01, 100, 0.01));
    final JSpinner periodChangeSpinner = new JSpinner(new
SpinnerNumberModel(8D, 1D, 10D, 1D));
```

```
    final PendulumPanel pendulum = new PendulumPanel();
    final JButton forceLeftBtn = new JButton("Раскачать вправо");
    final JButton forceRightBtn = new JButton("Раскачать влево");

    public PendulumSimulation() {
        setLayout(null);
        setSize(350, 640);

        pendulum.setBounds(20, 0, 310, 250);
        add(pendulum);

        resLabel.setBounds(20, 369, 310, 49);
        add(resLabel);

        time.setBounds(210, 369, 120, 49);
        add(time);

        forceLeftBtn.setBounds(20, 300, 155, 49);
        forceLeftBtn.addActionListener((x) ->
manager.applyForce(PendulumManager.ForceSide.LEFT));
        add(forceLeftBtn);
```

```
        forceRightBtn.setBounds(175, 300, 155, 49);
        forceRightBtn.addActionListener((x) ->
manager.applyForce(PendulumManager.ForceSide.RIGHT));
        add(forceRightBtn);

        // Change model labels
        amplitudeLabel.setBounds(31, 431, 133, 40);
        add(amplitudeLabel);
        amplitudeSpinner.addChangeListener((x) -> {
            manager.setAmplitude((double) amplitudeSpinner.getValue());
        });
        amplitudeSpinner.setBounds(31, 471, 133, 20);
        add(amplitudeSpinner);

        dampingLabel.setBounds(186, 431, 133, 40);
        add(dampingLabel);
        dampingSpinner.addChangeListener((x) -> {
            manager.setDampingCoefficient((double) dampingSpinner.getValue());
        });
        dampingSpinner.setBounds(186, 471, 133, 20);
        add(dampingSpinner);

        periodChangeLabel.setBounds(112, 504, 133, 40);
```

```
        add(periodChangeLabel);
        periodChangeSpinner.addChangeListener((x) -> {
            manager.setPeriodChangeRate((double) periodChangeSpinner.getValue());
        });
        periodChangeSpinner.setBounds(112, 544, 133, 20);
        add(periodChangeSpinner);
        manager.start();
    }

    private void showSimulationResult() {
        final var resModel = manager.getResult();
        final var cords = resModel.getPendulumPoint();
        resLabel.setText(String.format("x: %.4f y: %.4f", cords.x, cords.y));
        pendulum.setPendulumAngle(resModel.angle);
        time.setText(String.valueOf(resModel.msPassed));
    }

    public void update() {
        showSimulationResult();
    }

    private static class PendulumPanel extends JPanel {
```

```
    private double pendulumAngle;

    public void setPendulumAngle(double pendulumAngle) {
        this.pendulumAngle = pendulumAngle;
        repaint();
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        drawPendulum(g, getWidth() >> 1, getHeight() >> 1, pendulumAngle);
    }

    private void drawPendulum(Graphics g, double centerX, double centerY,
double pendulumAngle) {
        g.drawLine((int) centerX, (int) centerY, (int) centerX, (int) (centerY +
100));

        double pendulumLength = 100;
        double positionX = centerX + pendulumLength *
Math.sin(pendulumAngle);
        double positionY = centerY + pendulumLength *
Math.cos(pendulumAngle);
```

```
            g.drawLine((int) centerX, (int) centerY, (int) positionX, (int) positionY);


            g.fillOval((int) positionX - 5, (int) positionY - 5, 10, 10);
        }
    }

    public static void main(String[] args) {
        PendulumSimulation pendulumSimulation = new PendulumSimulation();
        pendulumSimulation.setVisible(true);
    }
}
```

Point.java

```
package lab3;

public class Point {
    public final double x;
    public final double y;

    public Point(double x, double y) {
        this.x = x;
        this.y = y;
```

```
    }
}
```

PendulumModel.java

```
package lab3;

final public class PendulumModel {
    final double amplitude;
    final double dampingCoefficient;
    final double periodChangeRate;

    public PendulumModel(double amplitude, double dampingCoefficient, double periodChangeRate) {
        this.amplitude = amplitude;
        this.dampingCoefficient = dampingCoefficient;
        this.periodChangeRate = periodChangeRate;
    }

    public PendulumModel copyWith(Double amplitude, Double dampingCoefficient, Double periodChangeRate) {
        return new PendulumModel(
                amplitude != null ? amplitude : this.amplitude,
```

```
            dampingCoefficient != null ? dampingCoefficient :
this.dampingCoefficient,
            periodChangeRate != null ? periodChangeRate : this.periodChangeRate
        );
    }
}
```

PendulumSimulationResult.java

```
package lab3;

public class PendulumSimulationResult {
    final public float msPassed;
    final public double angle;

    public PendulumSimulationResult(PendulumModel model, long msPassed, int
directionForce) {
        angle = simulatePendulum(
            model,
            msPassed,
            directionForce
        );
        this.msPassed = msPassed;
    }
```

```
    public static double simulatePendulum(PendulumModel model, long
totalTimeMillis, int direction) {

        final var amplitude = model.amplitude;
        final var dampingCoefficient = model.dampingCoefficient;
        final var periodChangeRate = model.periodChangeRate;

        float totalTimeSeconds = totalTimeMillis / 1000.0f;



        return amplitude * (float) Math.exp(-dampingCoefficient * totalTimeSeconds)
                * (float) Math.cos(9 * (1 / periodChangeRate) * totalTimeSeconds +
direction * Math.PI / 2.0f);

    }

    public Point getPendulumPoint() {
        double xCoordinate = Math.sin(this.angle) * 10;
        double yCoordinate = Math.cos(this.angle) * 10;

        return new Point(xCoordinate, yCoordinate);
    }
```

```
}
```

PendulumManager.java

```
package lab3;

import javax.swing.Timer;

public class PendulumManager {
    PendulumModel model = new PendulumModel(
            1F,
            0.15F,
            10F
    );

    final PendulumSimulation simulation;

    public PendulumManager(PendulumSimulation simulation) {
        this.simulation = simulation;
    }

    public void start() {
        final var timer = new Timer(32, (x) -> simulation.update());
```

```
    timer.setRepeats(true);
    timer.start();
}

long startedTimeMs = -1;
int forceDirection = 1;

public void setAmplitude(double amplitude) {
    this.model = model.copyWith(amplitude, null, null);
    simulation.update();
}

public void setDampingCoefficient(double dampingCoefficient) {
    this.model = model.copyWith(null, dampingCoefficient, null);
    simulation.update();
}

public void setPeriodChangeRate(double periodChangeRate) {
    this.model = model.copyWith(null, null, periodChangeRate);
    simulation.update();
}
```

```
    public enum ForceSide {LEFT, RIGHT}

    public void applyForce(ForceSide side) {
        startedTimeMs = System.currentTimeMillis();
        if (side == ForceSide.RIGHT) {
            forceDirection = 1;
        } else {
            forceDirection = -1;
        }
        simulation.update();
    }

    public PendulumSimulationResult getResult() {
        final var estimatedTime = System.currentTimeMillis() - startedTimeMs;
        return new PendulumSimulationResult(model, estimatedTime,
forceDirection);
    }
}
```