**Лабораторная работа № 4**

**Цель:**

Научиться работать с классами “Timer” и “TimerTask”, а также изучить стандартные компоненты построения диаграмм в языке Java.

**Постановка задачи:**

**Задание 1**. Реализовать приложение, которое выводит на экран окружность и радиус. Радиус вращается по часовой стрелке вокруг центра со скоростью 60 сек/один оборот, т.е. как секундная стрелка часов.

**Задание 2**. Загружается графическое изображение и движется по кругу, вписанному в окно. Скорость движения (линейная) задается и изменяется бегунком и не зависит от размеров окна (использовать формулу), направление движения выбирается.

**Задание 3**. Реализовать приложение, которое строит круговую/столбчатую диаграмму. Использовать стандартные компоненты для диаграмм. Количество категорий и их значения могут быть различными. Количество категорий, их имена и значения задаются в файле формата JSON. Входные данные отображаются в виде легенды и/или подписей.

**Решение задачи:**

**Задание 1.**

Сначала создаем класс “Clock”, объекты которого будут хранить количество делений, пройденных стрелкой за промежуток времени в 1 секунду.

```
package graviwave;

class Clock {
    private double seconds;

    Clock(double seconds) {
        this.setSeconds(seconds);
    }

    void setSeconds(double seconds) {
        this.seconds = seconds % 60;
    }

    double getSeconds() {
        return seconds;
    }
}
```

*Рис.1. Класс “Clock”.*

Затем описываем класс “ClockPanel”, который наследуется от “JPanel”, то есть реализует оконное приложение. В данном классе присутствует лишь одно поле - объект класса “Clock” - которое передается в качестве параметра конструктора окна.

Переопределяем метод “paint”, который выполняет рисование в окне аплета. Метод “drawCircle” осуществляет прорисовку окружности (циферблата), в нем используется стандартный метод “fillOval”, остается лишь задать формулу для координат желаемой фигуры.

```
class ClockPanel extends JPanel{
    private Clock clock;

    ClockPanel(Clock Clock) {
        setDoubleBuffered(true);
        setPreferredSize(new Dimension( width: 600,  height: 400));
        setClock(Clock);
    }

    @Override
    public void paint(Graphics g) {
        super.paint(g);
            drawSecClock(g);

    }
    private void drawCircle(Graphics g, Point center) {
        g.setColor(Color.RED);
        g.fillOval( x: center.x - 8 / 2,  y: center.y - 8 / 2,  width: 8,  height: 8);
    }
```

*Рис.2. Конструктор с параметрами класса “ClockPanel” и методы прорисовки окружности.*

С помощью метода “getEndPoint” высчитываем текущую позицию точки, являющейся подвижным концом радиуса окружности. Вычисленные координаты используются в теле метода “drawSecClosk”, который задает движение стрелки циферблата и саму окружность в виде 12 точек. Чтобы ход не сбивался при изменении размеров окна, программу реализуем через переменные, зависящие друг от друга и от высоты и ширины окна.

```
private Point getEndPoint(double angle, int radius) {
    Point O = new Point( x: getSize().width / 2,  y: getSize().height / 2);
    int x = (int) (O.x + radius * Math.cos(angle));
    int y = (int) (O.y - radius * Math.sin(angle));
    return new Point(x, y);
}
private void drawSecClock(Graphics g){
    Point O = new Point( x: getSize().width / 2,  y: getSize().height / 2);
    int radiusClock = Math.min(O.x, O.y) - 20;
    int radiusSeconds = radiusClock - 10;
    double angle;
    for (int i = 1; i < 13; i++) {
        angle = Math.PI / 2 - i * Math.PI / 6;
        Point point = getEndPoint(angle, radiusClock);
        drawCircle(g, point);
    }
    angle = Math.PI / 2 - clock.getSeconds() * Math.PI / 30;
    Point point = getEndPoint(angle, radiusSeconds);
    g.setColor(Color.GRAY);
    g.drawLine(O.x, O.y, point.x, point.y);
}
Clock getClock() { return clock; }

private void setClock(Clock clock) { this.clock = clock; }
```

*Рис.3.Методы, реализующие прорисовку окружности и ее посекундно перемещающегося радиуса.*

Класс “Main” содержит в себе основную функцию запуска программы “main”, но кроме задания дизайна окна создаем объект стандартного класса “Timer” и “TimerTask” - задачу, которая будет периодически выполняться. Таким образом, с периодом в 100 миллисекунд производится перерисовка панели так, что количество пройденных секунд увеличивается на единицу.

```
public class Main {
    public static void main(String[] args) {
        JFrame f = new JFrame( title: "Clock");
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setLayout(new BorderLayout());
        ClockPanel clockPanel = new ClockPanel(new Clock( seconds: -0.1));
        clockPanel.setBorder(BorderFactory.createLineBorder(Color.DARK_GRAY,  thickness: 2));
        f.add(clockPanel, BorderLayout.CENTER);

        Timer t = new Timer();
        t.schedule(() -> {
                clockPanel.getClock().setSeconds(clockPanel.getClock().getSeconds() + 0.1);
                clockPanel.repaint();
        },  delay: 0,  period: 100);
        f.pack();
        f.setVisible(true);
    }

}
```

*Рис.4. Класс “Main” с описанием перерисовки “ClockPanel”.*

**Задание 2.**

Класс “Form” наследуется от “JFrame” и является основой оконного приложения. Добавляем панель JPanel “mPanel”, которая будет содержать объект класса “PointOnCircle”.

```
class Form extends JFrame {
    Form(){
        super( title: "Practice 4.2");
        setResizable(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (ClassNotFoundException | InstantiationException | IllegalAccessException
                | UnsupportedLookAndFeelException ignored) {
        }
        JPanel mPanel = new JPanel();
        mPanel.setBackground(Color.BLACK);
        mPanel.add(new PointOnCircle());
        add(mPanel);
        pack();
        setSize( width: 800,  height: 800);
        setVisible(true);
    }
}
```

*Рис.5.Конструктор класса “Form”.*

В классе “PointOnCircle” реализуются все основные методы, требуемые в задании. Первым делом создаем группу радиокнопок таким образом, что пользователь, нажав определенную из них, выбирает, какая планета будет вращаться по кругу (ее изображение помещаем в “image”). Также необходимо инициализировать кнопки “left” и “right”, которые задают направление движения планеты. Вспомогательные переменные “m” и “val” отвечают за характер и скорость вращения соответственно.

```
class PointOnCircle extends JPanel {
    private int m = 1;
    private int val;
    private double angle;
    private Image image;
    private JRadioButton moon = new JRadioButton( text: "Moon",  selected: true);
    private JRadioButton earth = new JRadioButton( text: "Earth");
    private JRadioButton jupiter = new JRadioButton( text: "Jupiter");
    private JRadioButton mercury = new JRadioButton( text: "Mercury");
    private JRadioButton mars = new JRadioButton( text: "Mars");
    private JRadioButton venus = new JRadioButton( text: "Venus");
    private JRadioButton sun = new JRadioButton( text: "Sun");
    private JRadioButton saturn = new JRadioButton( text: "Saturn");
    private JButton left = new JButton( text: "Left");
    private JButton right = new JButton( text: "Right");
    private String imPath = "Moon";
```

*Рис.6.Поля класса “PointOnCircle”.*

В конструкторе указываем все, что будет отображаться на “mPanel”. Для этого добавляем панель настроек “setPanel”, на которой находятся все перечисленные выше кнопки и регулятор скорости движения изображения - объект класса JSlider “slider”.

```
PointOnCircle() {
    setPreferredSize(new Dimension( width: 800,  height: 800));
    JPanel setPanel = new JPanel();
    setPanel.setPreferredSize(new Dimension( width: 700,  height: 100));
    setPanel.setLayout(new GridLayout( rows: 3,  cols: 4));
    ButtonGroup planets = new ButtonGroup();
    planets.add(moon);
    planets.add(earth);
    planets.add(jupiter);
    planets.add(mercury);
    planets.add(sun);
    planets.add(saturn);
    planets.add(mars);
    planets.add(venus);

    setPanel.add(mercury);
    setPanel.add(venus);
    setPanel.add(moon);
    setPanel.add(earth);
    setPanel.add(mars);
    setPanel.add(jupiter);
    setPanel.add(saturn);
    setPanel.add(sun);

    JSlider slider = new JSlider( min: 0,  max: 10,  value: 0);
    slider.setMajorTickSpacing(1);
    slider.setPaintLabels(true);
    setPanel.add(slider);
    left.setPreferredSize(new Dimension( width: 50,  height: 20));
    right.setPreferredSize(new Dimension( width: 50,  height: 20));
    setPanel.add(left);
    setPanel.add(right);
    left.addActionListener(this::actionPerformed);
    right.addActionListener(this::actionPerformed);
```

*Рис.7.Конструктор класса “PointOnCircle”.*

Для удобства создаем “одного слушателя” для всех кнопок. Если выбрана кнопка “left”, то “m” принимает отрицательное значение, что влечет за собой изменение направления движения планеты с “по часовой стрелки” на “против часовой стрелки”, у “right” эффект противоположный. Для радиокнопок предусмотрена загрузка соответствующих изображений планет. Если пользователь кликает на “Earth”, то значение переменной “imPath”, входящей в наименование пути, меняется автоматически на “Earth”, из чего напрямую следует видоизменение космического тела.

```
private void actionPerformed(ActionEvent e) {
    if (e.getSource() == left){
        m = -1;
    }
    else if(e.getSource() == right){
        m = 1;
    }
    if (e.getSource() == moon) {
        imPath = "Moon";
    }
    else if (e.getSource() == jupiter){
        imPath = "Jupiter";
    }
    else if (e.getSource() == earth){
        imPath = "Earth";
    }
    else if (e.getSource() == mercury){
        imPath = "Mercury";
    }
    else if (e.getSource() == venus){
        imPath = "Venus";
    }
    else if (e.getSource() == mars){
        imPath = "Mars";
    }
    else if (e.getSource() == saturn){
        imPath = "Saturn";
    }
    else if (e.getSource() == sun){
        imPath = "Sun";
    }
}
```

*Рис.8.“Слушатель” для всех типов кнопок.*

```
    add(setPanel);
    moon.addActionListener(this::actionPerformed);
    earth.addActionListener(this::actionPerformed);
    jupiter.addActionListener(this::actionPerformed);
    mercury.addActionListener(this::actionPerformed);
    mars.addActionListener(this::actionPerformed);
    sun.addActionListener(this::actionPerformed);
    saturn.addActionListener(this::actionPerformed);
    venus.addActionListener(this::actionPerformed);
    angle = 0;
    Timer t1 = new Timer();
    val = 1;
    TimerTask timerTask = new TimerTask() {
        @Override
        public void run() {
            angle += m * 0.05 * val;
            repaint();
        }
    };
    t1.schedule(timerTask, delay: 10, period: 100);
    slider.addChangeListener(e -> {
        JSlider source = (JSlider) e.getSource();
        if(!source.getValueIsAdjusting()){
            val = source.getValue();
        }
    });
}
private void loadImage() {
    try {
        image = ImageIO.read(new File(
                pathname: "C:\\Users\\Лиза\\IdeaProjects\\Practice4.2\\src\\plain\\" + imPath +".png"));
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

*Рис.9.Метод “loadImage” и способ задания кругового движения планеты.*

На рисунке 9 можно увидеть “Timer” и “TimerTask”, которые задают вращение объекта, подобно реализации предыдущего задания. Однако теперь с помощью метода “getValue” в переменную “val” передается значение скорости, которую выбрал пользователь. И “val”, и “m” участвуют в задании угла, то есть расстояния, которое пройдет планета за

определенный промежуток времени: чем больше “val”, тем выше скорость (об “m” было написано выше).

В методе “paintComponent” загружаем изображение космического тела и задаем формулу его вращения, где координаты высчитываются относительно размеров панели.

```
@Override
protected void paintComponent(Graphics g) {
    loadImage();
    int width = getWidth();
    int height = getHeight();
    g.setColor(Color.black);
    g.fillRect( x: 0,  y: 0, width, height);
    Graphics2D g2d = (Graphics2D) g;
    int x =  width / 2 - 100;
    int y = height / 2 -100;
    double r = 0.65 * Math.min(x, y);
    x += r * Math.cos(angle);
    y += r * Math.sin(angle);
    g2d.drawImage(image,x, y,  observer: null);
}
```

*Рис.10.Метод “paintComponent”, который рисует изображение объекта согласно его движению.*

**Задание 3.**

Подключив через настройки проекта библиотеки, содержащие все необходимое для работы с “JSON” и диаграммами, создаем класс “Cosmetics”, объекты которого будут хранить информацию, считанную из файла “statisticData”

```
class Cosmetics {
    private int count;
    private String brand;

    int getCount() { return count; }

    String getBrand() { return brand; }
}
```

```
{
  "brand": "MAC",
  "count": 580
},
{
  "brand": "L'Oreal",
  "count": 500
},
{
  "brand": "Chanel",
  "count": 450
},
```

*Рис.11.Класс “Cosmetics” и формат входных данных из файла “JSON”.*

Для предупреждения ошибок вставляем блок “try-catch”, который ловит исключения, если файл пуст или данные в нем заданы неверно.
Создаем контейнер объектов класса “Cosmetics”, куда добавляем информацию из файла. Генерируем формат подписей (“labelFormat”) и крепим их на “теле” диаграммы.

```
private MainWindow() {
    setVisible(true);
    setPreferredSize(new Dimension( width: 800,  height: 600));

    try {
        JsonReader reader = new JsonReader(new FileReader( fileName:
                "C:\\Users\\Лиза\\IdeaProjects\\Practice4.3\\src\\statisticData"));
        Gson g = new Gson();
        Cosmetics[] cosmetics = g.fromJson(reader, Cosmetics[].class);
        for (var elem : cosmetics) {
            if (elem.getCount() <= 0)
                throw new NumberFormatException("Number of women cannot be less than 1");
        }
        PieDataset pieDataset = createDataSet(cosmetics);
        chart = createChart(pieDataset);
        PiePlot plot = (PiePlot) chart.getPlot();
        PieSectionLabelGenerator gen = new StandardPieSectionLabelGenerator(
                labelFormat: "{0}: {1} women, {2} ");
        plot.setLabelGenerator(gen);
        JPanel panel = new JPanel(new BorderLayout());
        panel.add(new ChartPanel(chart), BorderLayout.CENTER);
        add(panel);
        panel.validate();
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    } catch (IOException | NumberFormatException | JsonSyntaxException exc) {
        JOptionPane.showMessageDialog( parentComponent: null, exc.getMessage());
    }
    pack();
}
```

*Рис.12.Конструктор оконного приложения и реализация считывания из файла.*

Метод “PieDataset” заполняет DefaultPieDataset данными из множества объектов класса “Cosmetics”. Затем в функции “createChart” вызываем стандартный метод создания круговой диаграммы “createPieChart”, куда передаем “dataSet”, а также ставим булевы значения “true” на места параметров, которые хотим видеть в диаграмме (“legend”, “tooltips”).
Запуск программы происходит в функции “main” с вызовом конструктора класса “MainWindow”.

```
private JFreeChart createChart(final PieDataset dataSet) {
    chart = ChartFactory.createPieChart( title: "The best cosmetics",
            dataSet,  legend: true,  tooltips: true,  urls: false);
    return chart;
}

private PieDataset createDataSet(Cosmetics[] container) {
    DefaultPieDataset dataSet = new DefaultPieDataset();
    for (var elem : container) {
        dataSet.setValue(elem.getBrand(), elem.getCount());
    }
    return dataSet;
}
public static void main(String[] args) { new MainWindow(); }
```

*Рис.13.Класс "Cosmetics" и формат входных данных из файла "JSON".*

**Результат:**

**Задание 1.**

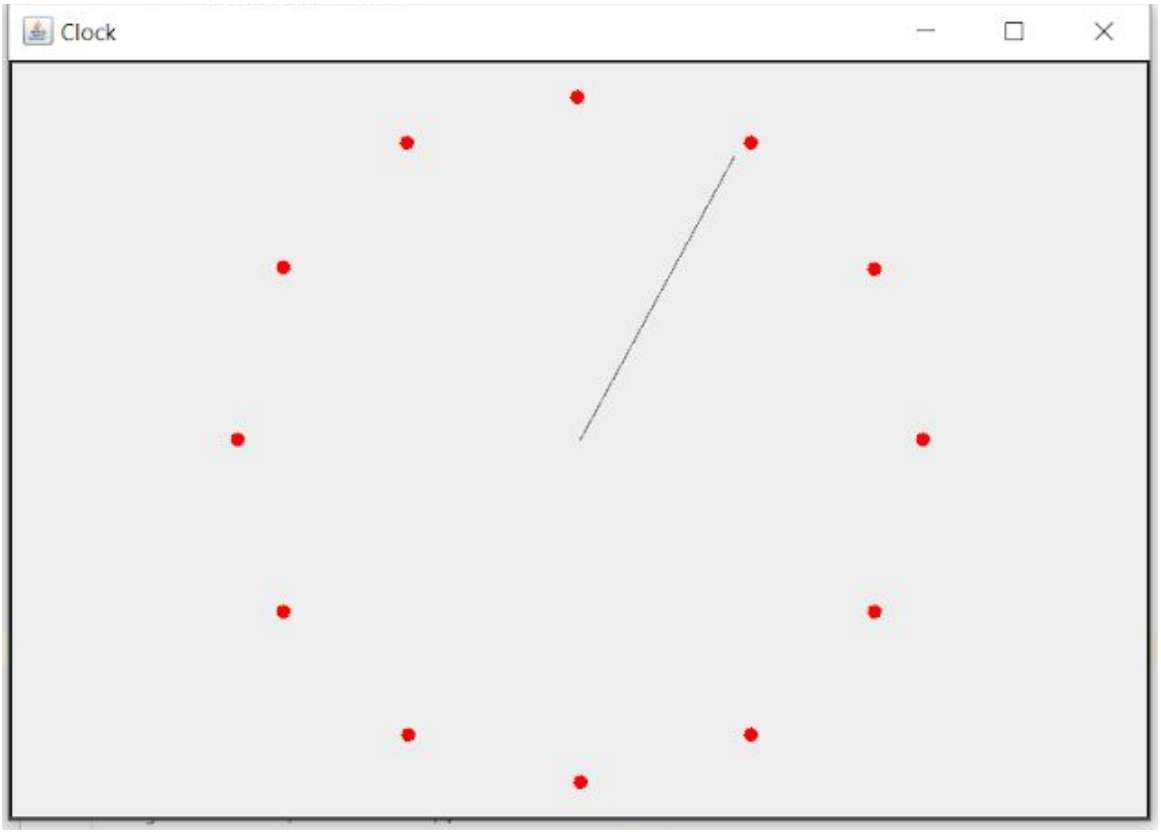


*Рис.14.Результат работы оконного приложения.*

## Задание 2.

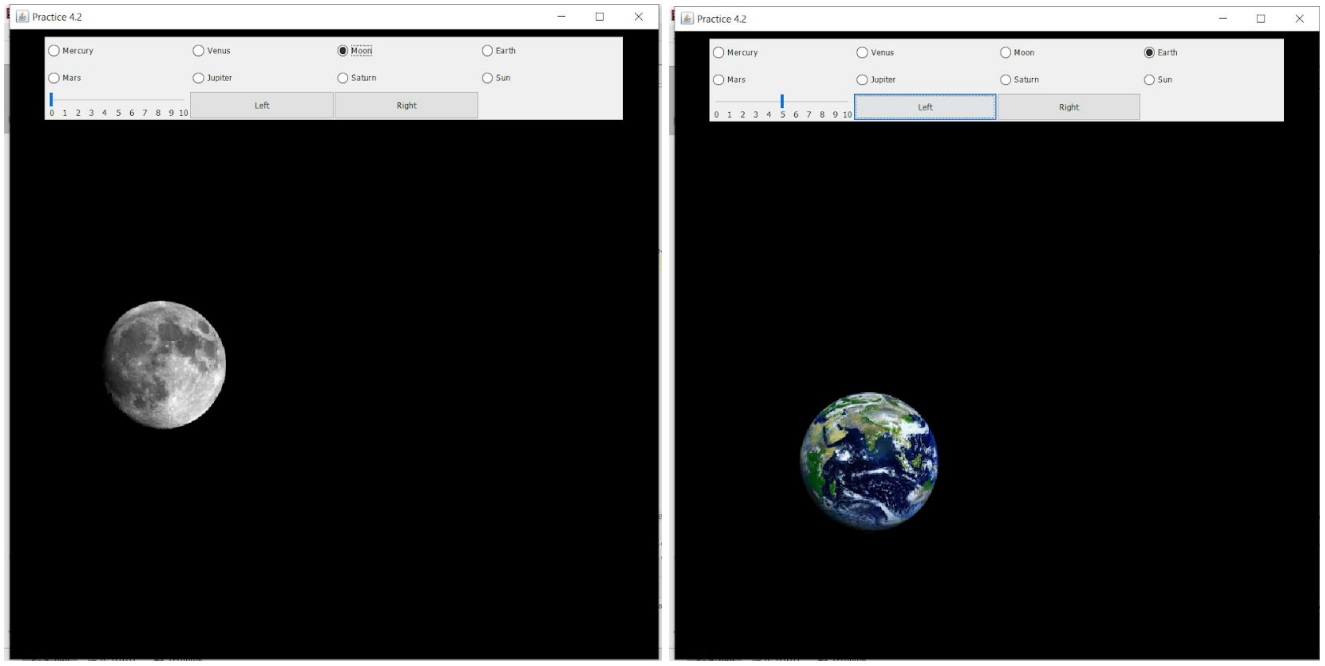


*Рис.15.Результат работы оконного приложения.*

## Задание 3.

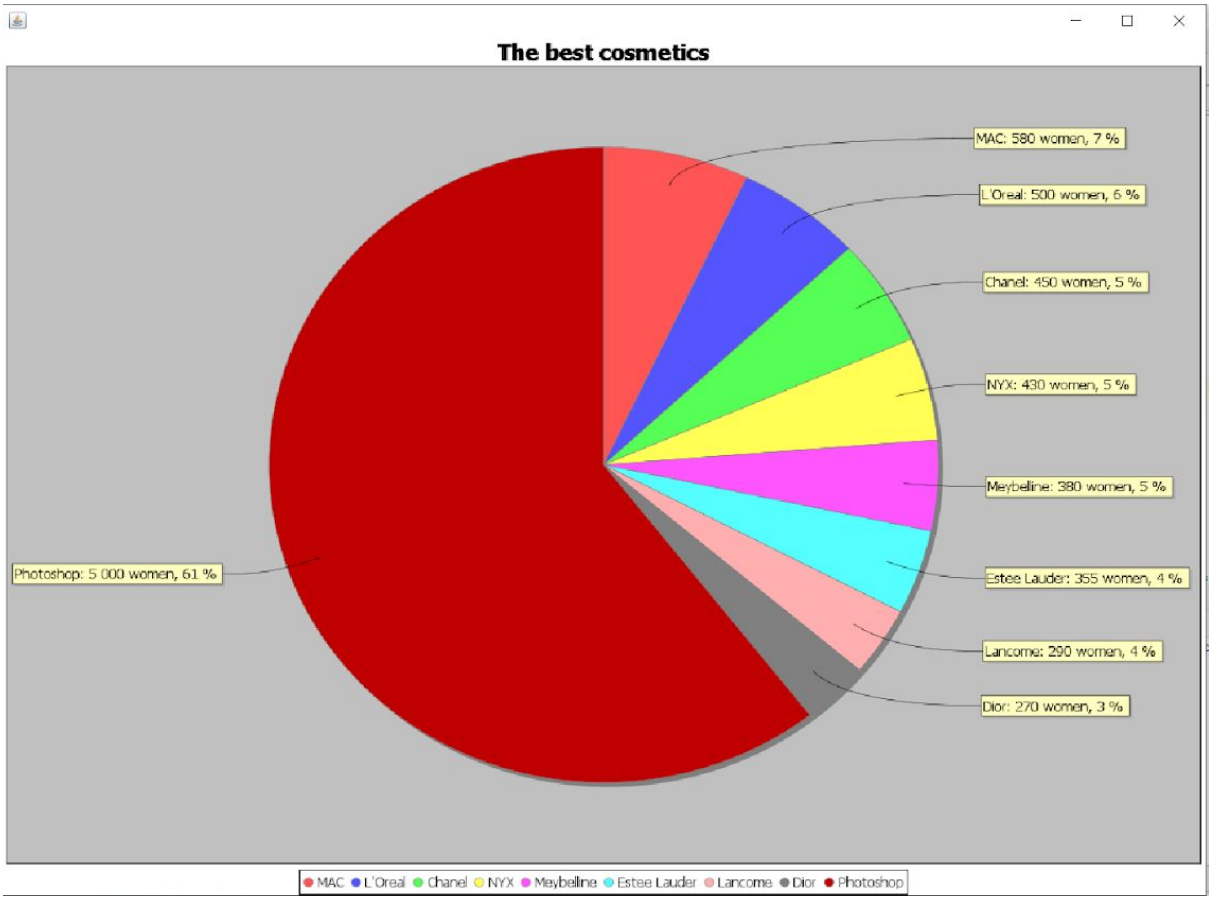


*Рис.16.Результат работы оконного приложения.*