Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Кафедра “Компьютерные технологии”

С.В. Пак, А.А. Шалыто

Задача об обедающих философах

Объектно-ориентированное программирование
с явным выделением состояний

Проектная документация

Проект создан в рамках
«Движения за открытую проектную документацию»
http://is.ifmo.ru

Санкт-Петербург
2003

Содержание

Введение

Для алгоритмизации и программирования задач логического управления была предложена SWITCH-технология, которая в дальнейшем была развита применительно к событийным и объектно-ориентированным программам. Подробно ознакомиться с этой технологией и с конкретными примерами ее использования можно на сайтах http://is.ifmo.ru и http://www.softcraft.ru.

Эта технология использовалась при создании программ с иерархической структурой, в которых параллелизм отсутствовал.

Данная работа посвящена применению автоматного программирования для решения задач с параллельными процессами на примере задачи об обедающих философах, предложенной Э.Дейкстрой [1–3]. В данной задаче имеют место шесть параллельных процессов — по одному для каждого из пяти философов и слуги.

Решение рассматриваемой задачи весьма важно, так оно может быть использовано при распределении общих ресурсов в операционных системах.[4].

Предлагаемый подход базируется на применении конечных автоматов, каждый из которых задается графом переходов. В этих графах вершины — состояния, ребра — переходы, а условия, по которым совершаются переходы — события и входные переменные.

Несмотря на наличие параллелизма в постановке задачи, в данной работе реализован так называемый «псевдопараллелизм», смысл которого состоит в том, что задача выполняется в одном потоке.

Программа реализована на языке программирования Java в виде приложения.

1. Классическая формулировка задачи

Рассмотрим формулировку задачи об обедающих философах в терминологии, предложенной Э. Дейкстрой. За круглым столом расставлены пять стульев, на каждом из которых сидит философ (Ф1-Ф5) (рис. 1).

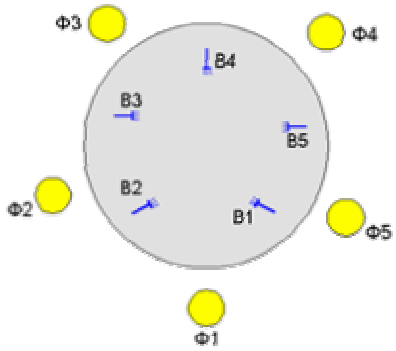

Рис. 1. Схема задачи

В центре стола размещено блюдо с макаронами. На столе лежат пять вилок (В1-В5), каждая из которых находится между двумя соседними тарелками. Каждый философ может находиться в двух состояниях: размышлять или есть макароны. Для того, чтобы начать есть, философу необходимы две вилки: одна в правой руке, а другая в левой. Закончив еду, философ кладет вилки на место и начинает размышлять до тех пор, пока снова не проголодается.

В этой задаче имеются две опасные ситуации: «заговор соседей» и «голодная смерть».

«Заговор соседей» имеет место, когда соседи слева и справа от философа строят козни. Заговорщики поочередно забирают вилки то слева, то справа от «жертвы». Такие согласованные действия злоумышленников приводят жертву к вынужденному голоданию, так как он никогда не может воспользоваться обеими вилками.

«Голодная смерть» возникает, когда философы одновременно проголодаются и одновременно попытаются взять, например, свою левую вилку. При этом возникает тупиковая ситуация, так как никто из них не может начать есть, не имея второй вилки.

2. Формулировка решаемой задачи

Для устранения неоднозначности поведения философа была предложена следующая модель его поведения:

размышляет **à** *берет левую вилку* **à** *берет правую вилку* **à** *ест* **à**

à *кладет правую вилку* **à** *кладет левую вилку* **à** *размышляет.*

При такой модели поведения философа проблема «заговора соседей» отпадает, так как каждый философ рано или поздно насытится, а значит, освободит вилку. Остается решить проблему «голодной смерти». Для этого в задачу введен еще один персонаж — «Слуга» («Servant»). Он должен следить за тем, чтобы в столовой не находилось более четырех философов одновременно.

На рис.2 – 9 представлены различные ситуации из жизни обедающих философов и обслуживающего их слуги, который на экране не отображен, а показаны лишь его действия по открытию и закрытию дверей.

На рисунках в поле «Логи» отображается протокол работы системы, который отражает ее работу в терминах автоматов.

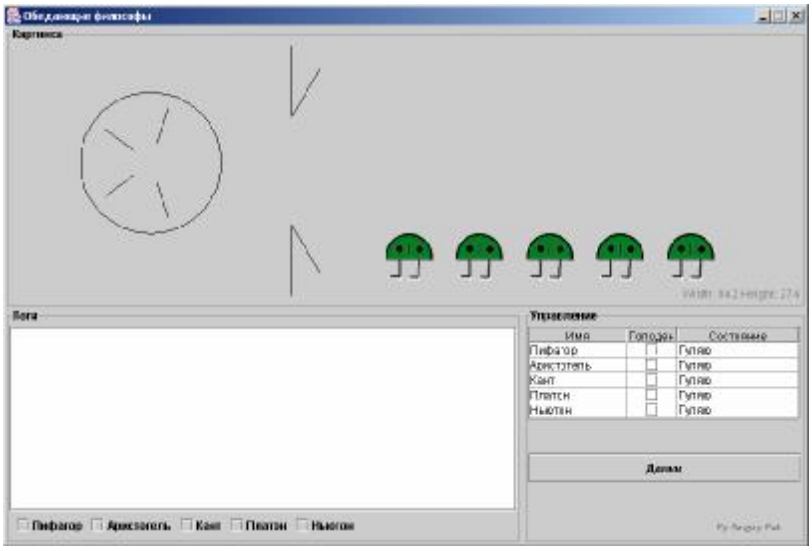

Рис. 2. Визуализатор программы. Начало

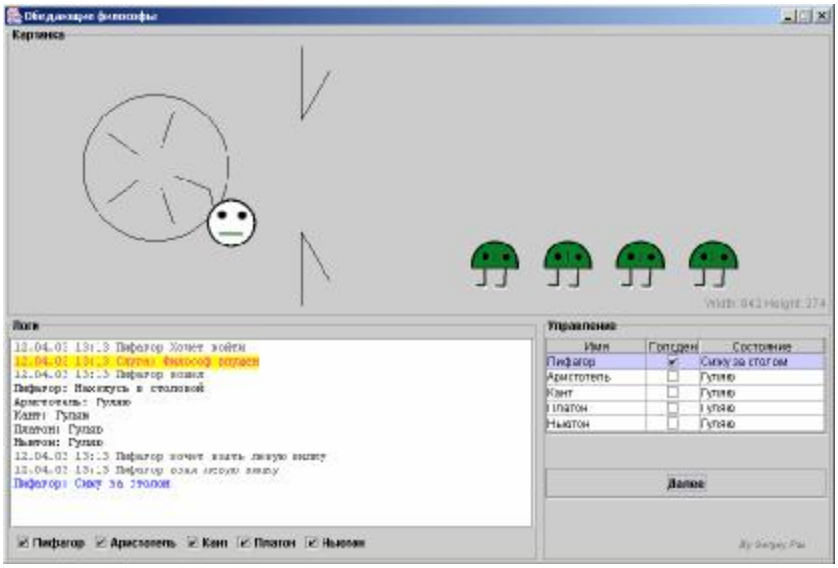

Рис. 3. В столовую запущен первый философ, который взял первую вилку

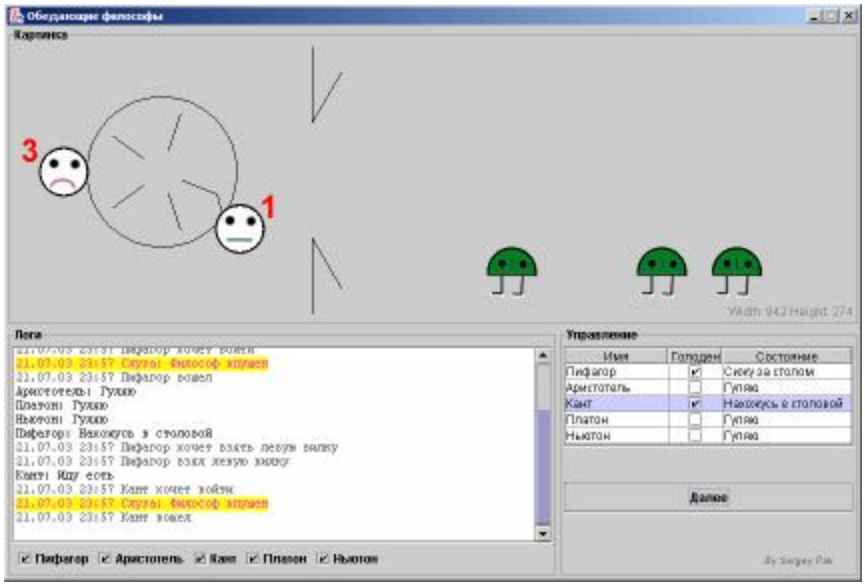

Рис. 4. Первый философ взял левую вилку. В столовую запущен второй философ

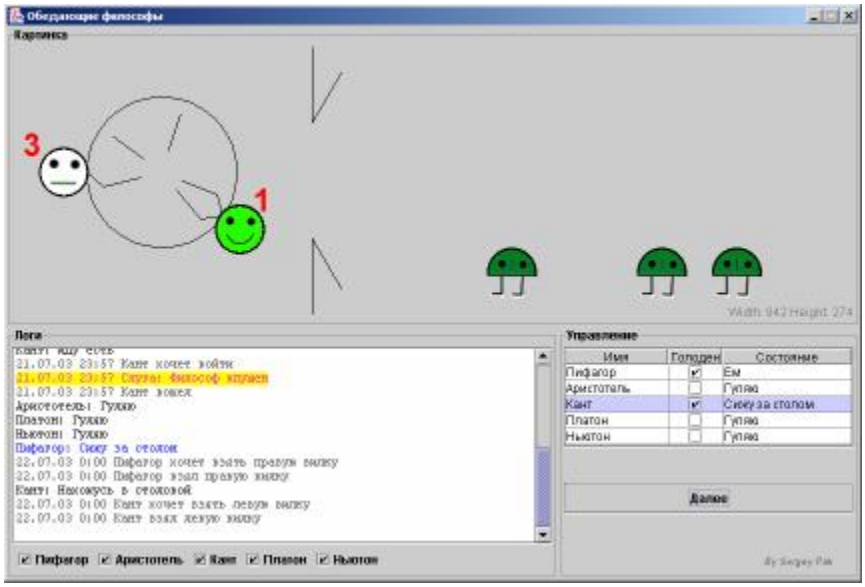

Рис. 5. Первый философ начал есть, взяв правую вилку. Второй философ философ взял левую вилку

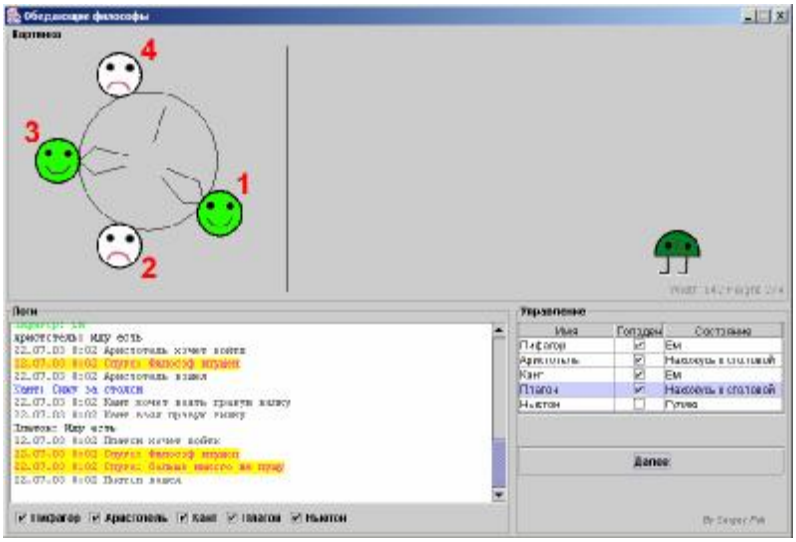

Рис. 6. За столом четыре философа, два из которых не могут есть из-за отсутствия вилок. Слуга закрыл дверь

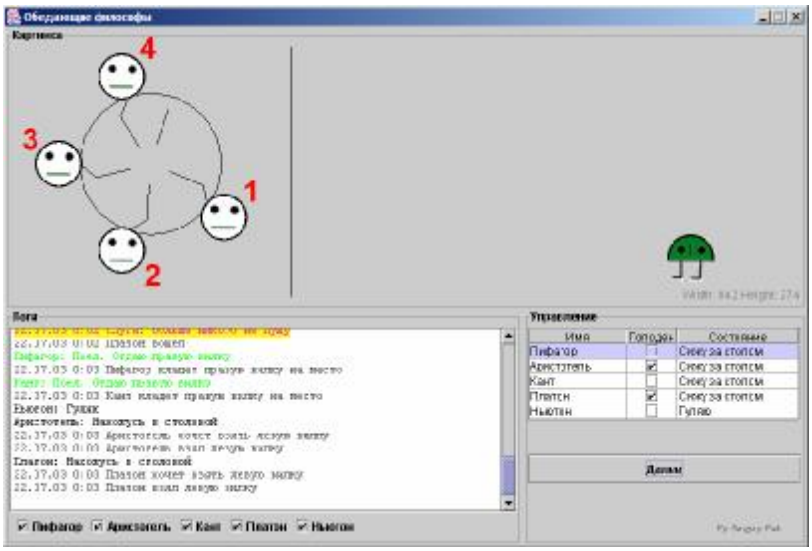

Рис. 7. Первые два философа закончили есть и освободили по одной вилке

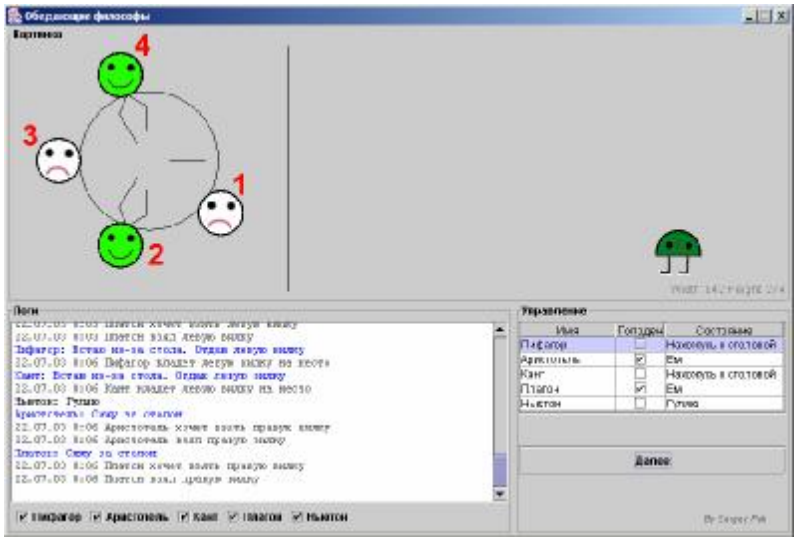

Рис. 8. Третий и четвертый философы начали обедать

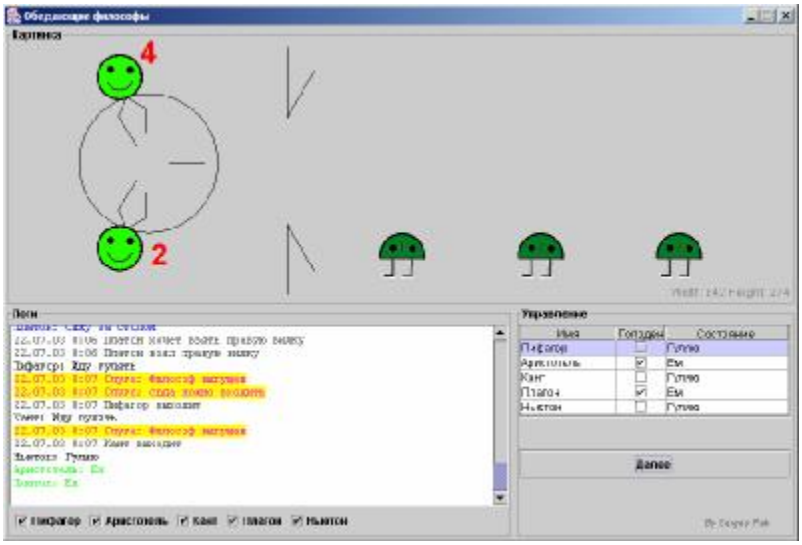

Рис. 9. Третий и четвертый философы обедают. Слуга открыл дверь. Первый и второй философы вышли погулять

После того, как третий и четвертый философы поедят, они также пойдут гулять, и система вернется в исходное состояние (рис.2).

3. Диаграмма классов

Диаграмма классов программы представлена на рис.10. Она может быть условно разбита на две части: «*интерфейс*» и «*логика*». «Интерфейс» (класс «Drawer») визуализирует задачу, а ее «логику» реализуют классы, содержащие автоматы A0 и A1.

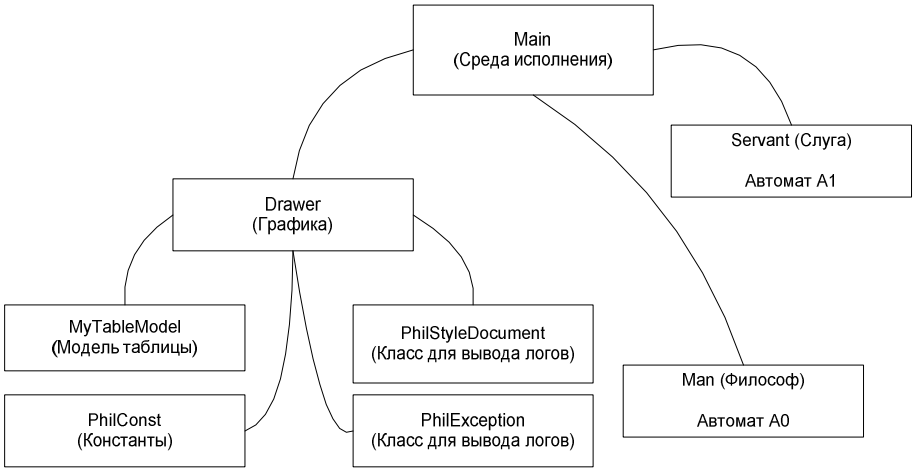

Рис. 10. Диаграмма классов

После запуска программы среда исполнения (Файл «Main.java») создает визуальную оболочку (Файл «Drawer.java»), экземпляры философов (Файл «Man.java») и слугу (Файл «Servant.java»). При нажатии на кнопку «Далее» программа выполняет один такт. Такт – один шаг автомата, который заключается в том, что все философы пытаются перейти в желаемые состояния. Если эта попытка удачна, то философ переходит в желаемое состояние, в противном случае генерируются исключение. Поведение философа описывается графом переходов (разд. 4.3) и схемой связей автомата «Философ» (разд. 4.2). Поведение слуги также описывается графом переходов (разд. 5.3) и схемой связи автомата «Слуга» (разд. 5.2)

4. Класс “Философ”

4.1. Словесное описание

Этот класс описывает поведение каждого философа (разд. 2) и протоколирует их работу. Он в качестве метода содержит автомат «Философ» (A0). Схема связей автомата представлена на рис.11, а граф переходов — на рис.12.

4.2. Схема связей автомата «Философ»

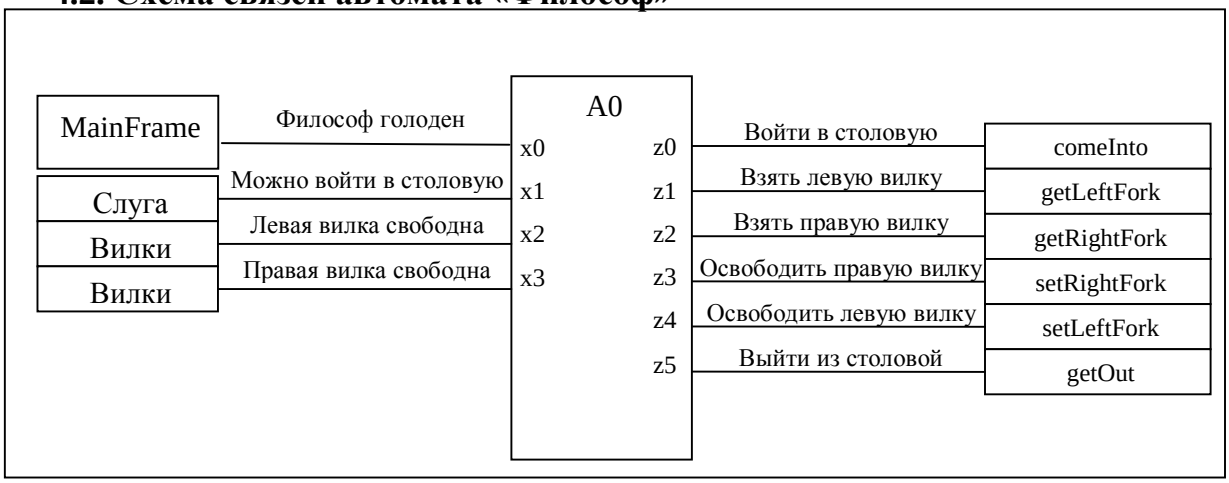

Рис. 11. Схема связей автомата «Философ»

4.3. Граф переходов

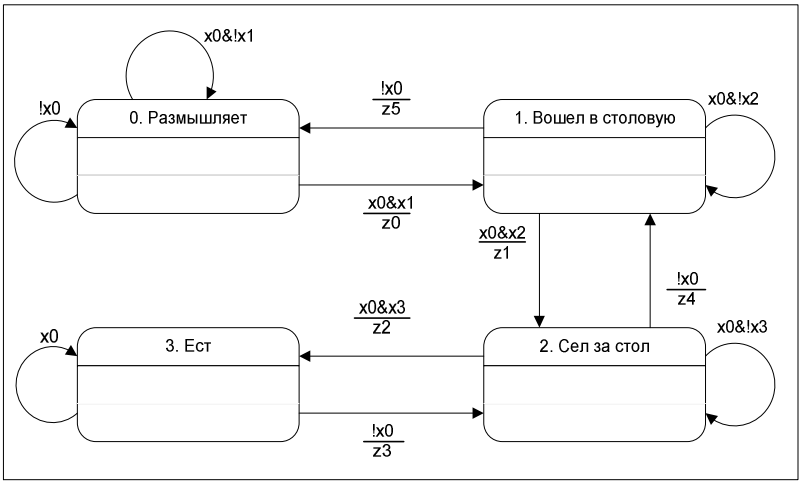

Рис. 12. Граф переходов автомата “Философ ”

5. Класс “Слуга”

5.1. Словесное описание

Этот класс описывает работу слуги, задача которого не допускать всех пятерых философов в столовую одновременно. Каждый раз, когда философ заходит в столовую, слуга увеличивает счетчик числа философов на один, и, если счетчик достигает четырех, то слуга никого больше не пускает до тех пор, пока какой-либо философ не покинет столовую. Кроме этого данный класс обеспечивает протоколирование работы слуги.

Класс в качестве метода содержит автомат «Слуга» (A1). Схема связей автомата представлена на рис.13, а граф переходов — на рис.14.

5.2. Схема связей автомата «Слуга»

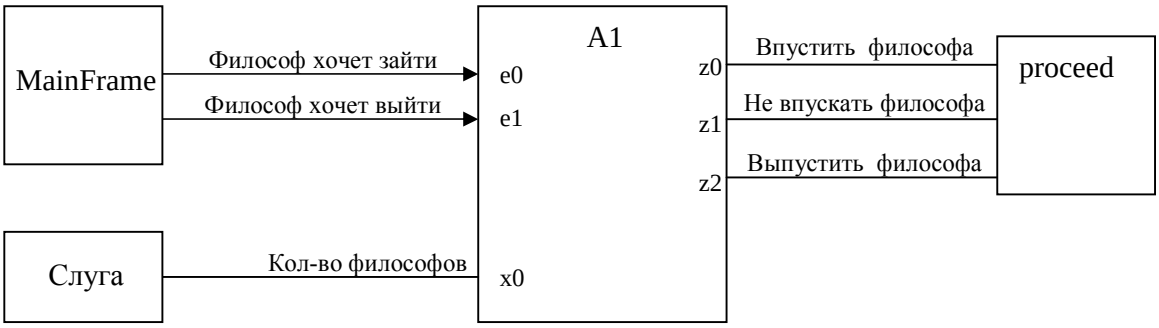

Рис. 13. Схема связей автомата «Слуга»

5.3. Граф переходов

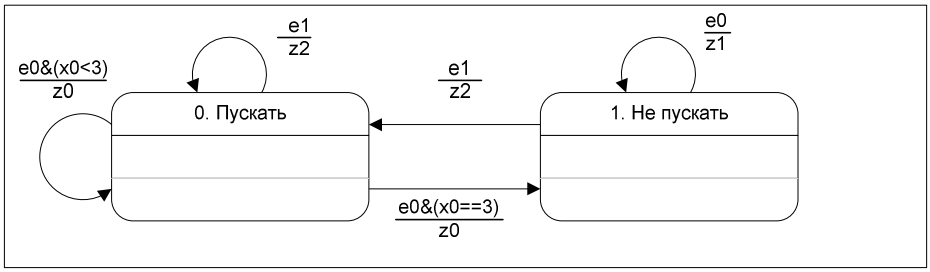

Рис. 14. Граф переходов автомата «Слуга»

В Приложении приведен исходный код программы. Отметим, что фрагменты программ, реализующих автоматы, изоморфны соответствующим графам переходов.

6. Протоколирование работы программы

Ниже приведен протокол работы программы. Его можно наблюдать в окне «Логи» визуализатора программы. Представленный ниже протокол соответствует рис 2-9.

Аристотель: Гуляю
Кант: Гуляю
Платон: Гуляю
Ньютон: Гуляю
Пифагор: Иду есть
21.07.03 23:57 Пифагор хочет войти
21.07.03 23:57 Слуга: Философ впущен
21.07.03 23:57 Пифагор вошел
Аристотель: Гуляю
Платон: Гуляю
Ньютон: Гуляю
Пифагор: Нахожусь в столовой
21.07.03 23:57 Пифагор хочет взять левую вилку
21.07.03 23:57 Пифагор взял левую вилку
Кант: Иду есть
21.07.03 23:57 Кант хочет войти
21.07.03 23:57 Слуга: Философ впущен
21.07.03 23:57 Кант вошел
Аристотель: Гуляю
Платон: Гуляю
Ньютон: Гуляю
Пифагор: Сижу за столом
22.07.03 0:00 Пифагор хочет взять правую вилку
22.07.03 0:00 Пифагор взял правую вилку
Кант: Нахожусь в столовой
22.07.03 0:00 Кант хочет взять левую вилку
22.07.03 0:00 Кант взял левую вилку
Ньютон: Гуляю
Пифагор: Ем
Аристотель: Иду есть
22.07.03 0:02 Аристотель хочет войти
22.07.03 0:02 Слуга: Философ впущен
22.07.03 0:02 Аристотель вошел
Кант: Сижу за столом
22.07.03 0:02 Кант хочет взять правую вилку
22.07.03 0:02 Кант взял правую вилку
Платон: Иду есть
22.07.03 0:02 Платон хочет войти
22.07.03 0:02 Слуга: Философ впущен
22.07.03 0:02 Слуга: больше никого не пущу
22.07.03 0:02 Платон вошел
Пифагор: Поел. Отдаю правую вилку
22.07.03 0:03 Пифагор кладет правую вилку на место
Кант: Поел. Отдаю правую вилку
22.07.03 0:03 Кант кладет правую вилку на место
Ньютон: Гуляю
Аристотель: Нахожусь в столовой
22.07.03 0:03 Аристотель хочет взять левую вилку
22.07.03 0:03 Аристотель взял левую вилку
Платон: Нахожусь в столовой
22.07.03 0:03 Платон хочет взять левую вилку
22.07.03 0:03 Платон взял левую вилку
Пифагор: Встаю из-за стола. Отдаю левую вилку
22.07.03 0:06 Пифагор кладет левую вилку на место
Кант: Встаю из-за стола. Отдаю левую вилку
22.07.03 0:06 Кант кладет левую вилку на место
Ньютон: Гуляю
Аристотель: Сижу за столом

22.07.03 0:06 Аристотель хочет взять правую вилку
22.07.03 0:06 Аристотель взял правую вилку
Платон: Сижу за столом
22.07.03 0:06 Платон хочет взять правую вилку
22.07.03 0:06 Платон взял правую вилку
Пифагор: Иду гулять
22.07.03 0:07 Слуга: Философ выпущен
22.07.03 0:07 Слуга: сюда можно входить
22.07.03 0:07 Пифагор выходит
Кант: Иду гулять
22.07.03 0:07 Слуга: Философ выпущен
22.07.03 0:07 Кант выходит
Ньютон: Гуляю
Аристотель: Ем
Платон: Ем

Заключение

Данная работа, в отличие от работы [2], содержит как проект программы, так и ее реализацию. Использование «псевдопараллелизма» (автоматы, соответствующие философам, вызываются по очереди) упрощает как построение, так и понимание программы. Правильность ее работы подтверждается не только визуально, но и с помощью протоколирования.

Источники

1. *Дубинин В. Н., Зинкин С. А.* Языки логического программирования в проектировании вычислительных систем и сетей. Пенза: Пензенский государственный университет. http://alice.stup.ac.ru/~dvn/prolog/articls/33.htm
2. *Любченко В.С.* Обедающие философы Дейкстры (о модели параллельных процессов). http://www.softcraft.ru/auto/ka/fil/fil.shtml
3. *Хоар Ч.* Взаимодействующие последовательные процессы. М.: Мир, 1989.
4. Операционные системы и системы программирования. http://www.kolasc.net.ru/cdo/programmes/os/26.htm

Приложение. Листинги программы с комментариями

Файл «Main.java»

Основная форма. Содержит экземпляры философов и слугу

```
package gui;

import logic.PhilConst;
import logic.Man;
import logic.PhilException;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Main extends JPanel{

    // public static members
    public static final String header = "Обедающие философы";
    public static final String drawAreaHeader = "Картинка";
    public static final String controlsHeader = "Управление";
    public static final String loggerHeader = "Логи";

    // private members
    private boolean[] philStates = {false, false, false, false, false};
    private boolean servant = true;

    private JPanel drawArea = null;
    private Drawer drawer = null;
    private JPanel controls = null;
    private JPanel logger   = null;
    private JButton next = null;
    private JButton reset = null;
    private JCheckBox[] philLogs   = {null, null, null, null, null};
    private JCheckBox srvButton = null;
    private JTextPane logPane = null;
    private PhilStyleDocument log = null;
    private JTable table = null;

    public Man[] men;

    public Main()
    {
        super(new BorderLayout(), true);
        initLogic();
        logger = createLogger();
        Man.prn = log;
        controls = createControls();
        PhilException.prn = log;
        this.add(controls, BorderLayout.EAST);
        this.add(logger, BorderLayout.WEST);
        drawArea = createDrawArea();
        this.add(drawArea, BorderLayout.NORTH);
        drawArea.repaint();
    }

    private JPanel createDrawArea()
    {
        JPanel panel = new JPanel(new BorderLayout());
        panel.setBorder(BorderFactory.createTitledBorder(drawAreaHeader));
        drawer = new Drawer(men);
```

```
        panel.add(drawer, BorderLayout.CENTER);

        panel.setPreferredSize(new Dimension(800, 300));
        return panel;
    }

    private JPanel createControls()
    {
        JPanel panel = new JPanel(new BorderLayout(25, 35));
        panel.setBorder(BorderFactory.createTitledBorder(controlsHeader));
        table = new JTable(new MyTableModel());
        table.getColumnModel().getColumn(0).setPreferredWidth(75);
        table.getColumnModel().getColumn(1).setPreferredWidth(25);
        table.getColumnModel().getColumn(2).setPreferredWidth(100);

        JScrollPane scroller = new JScrollPane(table);
        scroller.setPreferredSize(new Dimension(245, 100));

        panel.add(scroller, BorderLayout.NORTH);

        next = new JButton("Далее");
        next.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                for (int i=0; i<5; i++)
                     men[i].setH(((Boolean)table.getValueAt(i,
1)).booleanValue());

                for (int i=0; i<5; i++)
                {
                    if (men[i].getH())
                        continue;
                    men[i].proceed();

table.setValueAt(logic.PhilConst.stateMap[men[i].getState()],i, 2);
                }

                for (int i=0; i<5; i++)
                {
                    if (!men[i].getH())
                        continue;
                    men[i].proceed();

table.setValueAt(logic.PhilConst.stateMap[men[i].getState()],i, 2);
                }
                drawer.repaint();
            }
        });
        next.setSize(new Dimension(100, 25));

        panel.add(next, BorderLayout.CENTER);

        JLabel label = new JLabel("By Sergey Pak");
        label.setPreferredSize(new Dimension(90, 25));
        label.setAlignmentX(0.5f);
        label.setFont(new Font("Courier", Font.ITALIC, 10));
        label.setForeground(new Color(104, 104, 104));
        JPanel pane = new JPanel(new BorderLayout());
        pane.add(label, BorderLayout.EAST);
        panel.add(pane, BorderLayout.SOUTH);

        panel.setPreferredSize(new Dimension(300, 250));
        return panel;
    }
```

```
private JPanel createLogger()
{
    // Размер панели: 400x250
    JPanel panel = new JPanel(new BorderLayout());
    panel.setBorder(BorderFactory.createTitledBorder(loggerHeader));
    log = new PhilStyleDocument();
    logPane = new JTextPane(log);
    logPane.setEditable(false);
    JScrollPane scroller = new JScrollPane(logPane);

    panel.add(scroller, BorderLayout.CENTER);
    panel.add(createLogBar(), BorderLayout.SOUTH);
    panel.setPreferredSize(new Dimension(550, 250));
    return panel;
}

private JToolBar createLogBar()
{
    JToolBar bar = new JToolBar("Sergey");
    for(int i=0; i<5; i++)
    {
        final int a = i;
        philLogs[i] = new JCheckBox(PhilConst.names[i]);
        philLogs[i].addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e)
            {
                philStates[a] = philLogs[a].isSelected();
                men[a].setLogged(philStates[a]);
            }
        });
        bar.add(philLogs[i]);
    }
    bar.setFloatable(false);
    return bar;
}

private void initLogic()
{
    men = new Man[5];
    for (int i=0; i< 5;i++)
    {
        men[i] = new Man(i, PhilConst.names[i], false, philStates[i]);
    }
}

public static void main(String[] args)
{
    JFrame frame = new JFrame(header);
    frame.addWindowListener(new WindowAdapter()
    {
        public void windowClosing(WindowEvent e)
        {
            exit(0);
        }
    });
    Main panel = new Main();
    frame.getContentPane().add(panel, BorderLayout.CENTER);
    frame.pack();
    frame.setResizable(false);
    frame.setLocation(100, 100);
    frame.show();
}
```

```
    private static void exit(int errorCode)
    {
        System.exit(errorCode);
    }
}
```

Файл «Drawer.java»

Используется для рисования

```
package gui;

import logic.Man;

import javax.swing.*;
import java.awt.*;
import java.awt.image.BufferedImage;

public class Drawer extends Canvas
{

    private Man[] mm;

    public Drawer(Man[] men)
    {
        mm = men;
    }

    private void drawForks(Graphics g)
    {
        int xCenter =  150;
        int yCenter =  125;
        int radix = 75;

        for (int i=0; i<5; i++)
        {

            double xShift  = radix * Math.cos(i*Math.PI/2.5);
            double yShift  = radix * Math.sin(i*Math.PI/2.5);
            double xShift1 = radix * Math.cos((i+0.3)*Math.PI/2.5);
            double yShift1 = radix * Math.sin((i+0.3)*Math.PI/2.5);
            double xShift2 = radix * Math.cos((i-0.3)*Math.PI/2.5);
            double yShift2 = radix * Math.sin((i-0.3)*Math.PI/2.5);
            double xShifta = radix * Math.cos((i-0.5)*Math.PI/2.5);
            double yShifta = radix * Math.sin((i-0.5)*Math.PI/2.5);
            double xShiftb = radix * Math.cos((i+0.5)*Math.PI/2.5);
            double yShiftb = radix * Math.sin((i+0.5)*Math.PI/2.5);


            if (Man.chkFork(i) == 0)
            {
                g.drawLine((int) (xCenter + xShift*0.3),
                           (int) (yCenter + yShift*0.3),
                           (int) (xCenter + xShift*0.8),
                           (int) (yCenter + yShift*0.8)
                );
            }
            else if (Man.chkFork(i) == (i+1))
            {
                g.drawLine((int) (xCenter + xShift1*0.3),
                           (int) (yCenter + yShift1*0.3),
                           (int) (xCenter + xShift1*0.8),
```

```
                        (int) (yCenter + yShift1*0.8)
            );
            g.drawLine((int) (xCenter + xShift1*0.8),
                        (int) (yCenter + yShift1*0.8),
                        (int) (xCenter + xShiftb),
                        (int) (yCenter + yShiftb)
            );
        }
        else
        {
            g.drawLine((int) (xCenter + xShift2*0.3),
                        (int) (yCenter + yShift2*0.3),
                        (int) (xCenter + xShift2*0.8),
                        (int) (yCenter + yShift2*0.8)
            );
            g.drawLine((int) (xCenter + xShift2*0.8),
                        (int) (yCenter + yShift2*0.8),
                        (int) (xCenter + xShifta),
                        (int) (yCenter + yShifta)
            );
        }
    }
}

private void drawPeople(Graphics g)
{
    int xCenter =  150;
    int yCenter =  125;
    int radix = 75;
    Image walk = new ImageIcon("img\\walk.gif").getImage();
    int walkh = walk.getHeight(this)/2;
    int walkw = walk.getWidth(this)/2;
    Image stand = new ImageIcon("img\\stand.gif").getImage();
    int standh = stand.getHeight(this)/2;
    int standw = stand.getWidth(this)/2;
    Image sit = new ImageIcon("img\\sit.gif").getImage();
    int sith = sit.getHeight(this)/2;
    int sitw = sit.getWidth(this)/2;
    Image eat = new ImageIcon("img\\eat.gif").getImage();
    int eath = eat.getHeight(this)/2;
    int eatw = eat.getWidth(this)/2;

    for (int i=0; i<5; i++)
    {

        double xShift  = radix * Math.cos((i+0.5)*Math.PI/2.5);
        double yShift  = radix * Math.sin((i+0.5)*Math.PI/2.5);
        if (mm == null)
            continue;

        switch (mm[i].getState())
        {
        case 0:
            g.drawImage(walk, 400 + i*3*walkw, 200, this);
        break;

        case 1:
            g.drawImage(stand,
                        (int) (xCenter + xShift*1.3 - standw),
                        (int) (yCenter + yShift*1.3 - standh),
                        this);
        break;

        case 2:
            g.drawImage(sit,
```

```
                            (int) (xCenter + xShift*1.3 - sitw),
                            (int) (yCenter + yShift*1.3 - sith),
                            this);
                break;

                case 3:
                case 4:
                    g.drawImage(eat,
                            (int) (xCenter + xShift*1.3 - eatw),
                            (int) (yCenter + yShift*1.3 - eath),
                            this);
                break;
                }
            }
        }


        private void drawDoor(Graphics g)
        {
            g.drawLine(300, 0, 300, 75);
            g.drawLine(300, 191, 300, 266);
            switch(logic.Servant.getState())
            {
            case 0:
                g.drawLine(300, 75, 329, 25);
                g.drawLine(300, 191, 329, 241);
                break;
            case 1:
                g.drawLine(300, 75, 300, 191);
                break;
            }
        }

        // Рисуем картинку
        public void paint(Graphics g2)
        {
            BufferedImage img = new BufferedImage(this.getWidth(), this.getHeight(),
BufferedImage.TYPE_3BYTE_BGR);
            Graphics g = img.getGraphics();
            int width = this.getWidth();
            int height = this.getHeight();
            g.setColor(this.getBackground());
            g.fillRect(0, 0, width, height);

            g.setColor(Color.gray);
            g.drawString("Width: " + width + " Height: " + height, width-125, height
- 5);

            g.setColor(Color.black);
            int xCenter =  150;
            int yCenter =  125;
            int radix = 75;
            g.drawOval(xCenter-radix, yCenter-radix, radix*2, radix*2);
            // Drawing forks and people
            drawForks(g);
            drawPeople(g);
            drawDoor(g);

            g2.drawImage(img, 0, 0, this);
        }
}
```

Файл «MyTableModel.java»

Модель таблицы

```
package gui;

import javax.swing.table.*;
import javax.swing.*;
import logic.PhilConst;

public class MyTableModel extends AbstractTableModel {
    final String[] columnNames = {"Имя", "Голоден", "Состояние"};
    final Object[][] data = {
            {PhilConst.names[0], new Boolean(false), PhilConst.stateMap[0]},
            {PhilConst.names[1], new Boolean(false), PhilConst.stateMap[0]},
            {PhilConst.names[2], new Boolean(false), PhilConst.stateMap[0]},
            {PhilConst.names[3], new Boolean(false), PhilConst.stateMap[0]},
            {PhilConst.names[4], new Boolean(false), PhilConst.stateMap[0]}
        };

    public int getColumnCount() {
        return columnNames.length;
    }

    public int getRowCount() {
        return data.length;
    }

    public String getColumnName(int col) {
        return columnNames[col];
    }

    public Object getValueAt(int row, int col) {
        return data[row][col];
    }

    public Class getColumnClass(int c) {
        return getValueAt(0, c).getClass();
    }

    public boolean isCellEditable(int row, int col) {
        //Note that the data/cell address is constant,
        //no matter where the cell appears onscreen.
        if (col == 1)
            return true;

        return false;
    }

    public void setValueAt(Object value, int row, int col) {
        if (data[0][col] instanceof Integer
                && !(value instanceof Integer)) {
            try {
                data[row][col] = new Integer(value.toString());
                fireTableCellUpdated(row, col);
            } catch (NumberFormatException e) {
                JOptionPane.showMessageDialog(null,
                    "The \"" + getColumnName(col)
                    + "\" column accepts only integer values.");
            }
        } else {
            data[row][col] = value;
            fireTableCellUpdated(row, col);
        }
```

```
    }
}
```

Файл «PhilStyleDocument.java»

Документ для хранения лога

```
package gui;

import javax.swing.text.DefaultStyledDocument;
import javax.swing.text.SimpleAttributeSet;

public class PhilStyleDocument extends DefaultStyledDocument{
    public PhilStyleDocument()
    {
        super();
    }
}
```

Файл «Man.java»

Класс философ

```
package logic;

import javax.swing.text.Document;
import javax.swing.text.SimpleAttributeSet;
import javax.swing.text.StyleConstants;
import javax.swing.text.MutableAttributeSet;
import java.awt.Color;
import java.text.DateFormat;

public class Man
{
    private static final String philMessage = "Исключение в Философе #";

    private static MutableAttributeSet set1 = null;
    private static MutableAttributeSet set2 = null;
    private static MutableAttributeSet set3 = null;
    private static MutableAttributeSet set = null;
    private static int[] forks = new int[5];
    private int state = 0;
    private int num = 0;
    private int time = 0;
    private boolean x0 = false;
    private boolean isLogged = true;
    private boolean messagePrinted;
    private String name = null;
    private String philStateMessage = null;
    public static Document prn = null;

    protected Man()
    {
        // Константы для вывода цветного текста в поле
        set1 = new SimpleAttributeSet();
        StyleConstants.setForeground(set1, Color.black);
        StyleConstants.setFontFamily(set1, "Courier New");
        set2 = new SimpleAttributeSet();
        StyleConstants.setForeground(set2, Color.blue);
        StyleConstants.setFontFamily(set2, "Courier New");
        set3 = new SimpleAttributeSet();
        StyleConstants.setForeground(set3, Color.green);
        StyleConstants.setFontFamily(set3, "Courier New");
        set = new SimpleAttributeSet();
        StyleConstants.setForeground(set, Color.darkGray);
```

```
    StyleConstants.setFontFamily(set, "Courier New");
    new Servant();
}

// Основной конструктор
public Man(int n, String str, boolean isH, boolean isL)
{
    this();
    setNum(n);
    this.philStateMessage = PhilConst.names[n] + ":";
    this.state = 0;
    this.isLogged = isL;
    setName(str);
    setX0(isH);
}

public int getNum()     {return num;}
public int getState()   {return state;}
public String getName() {return name;}
public boolean getX0()   {return x0;}
public int getTime()    {return time;}
public boolean isLogged(){return isLogged;}

public void setLogged(boolean newLogState) {isLogged = newLogState;}
public void setNum(int n)       {num = n;}
public void setName(String str) {name = str;}
public void setX0(boolean isH)   {x0 = isH;}
public void setTime(int t)      {time = t;}

// Автомат A0
public void proceed()
{
    messagePrinted = false;
    try
    {
        switch(state)
        {
            case 0:     // Гуляет
                if (x0 && x1())
                {
                    z0();
                    state = 1;
                    break;
                }
                if (x0 && !x1())
                {
                    createPhilException(3);     // Выводим логи
                    state = 0;
                    break;
                }
                if (!x0)
                {
                    state = 0;
                    break;
                }
            break;
            case 1:     // Находится в столовой
                if (x0 && x2())
                {
                    z1();
                    state = 2;
                    break;
                }
```

```
                if (x0 && !x2())
                {
                    createPhilException(1); // Выводим логи
                    state = 1;
                    break;
                }

                if (!x0)
                {
                    z5();
                    state = 0;
                    break;
                }
            break;
            case 2:     // Сидит за столом (вилка в левой руке)
                if (x0 && x3())
                {
                    z2();
                    state = 3;
                    break;
                }

                if (x0 && !x3())
                {
                    createPhilException(2);     // Выводим логи
                    state = 2;
                    break;
                }
                if (!x0)
                {
                    z4();
                    state = 1;
                    break;
                }
            break;
            case 3:     // Ест
                if (x0)
                {

                    state = 3;
                    break;
                }

                if (!x0)
                {
                    z3();
                    state = 2;
                    break;
                }
            break;
            default:    // Ошибка
            {
                throw createPhilException(0);
            }
        }
        printStateMessage();
    }
    catch(PhilException e)
    {
    }
}
```

```
/*
   Входные воздействия автомата A0
*/

// Сыт философ или голоден (x0)
private boolean x1()
{
    if (!messagePrinted)
        locprn(4);
    messagePrinted = true;
    return (Servant.getState() == 0);
}

// Пытаемся открыть дверь (x1)
private boolean x2()
{
    if (!messagePrinted)
        locprn(0);
    messagePrinted = true;
    return (forks[num] == 0);
}

// Проверяем свободна ли левая вилка (x2)
private boolean x3()
{
    if (!messagePrinted)
        locprn(2);
    messagePrinted = true;
    return (forks[(num+1)%5] != 0);
}

// Проверяем свободна ли правая вилка (x3)
public static int chkFork(int num)
{
    return forks[num];
}

// Философ пытается войти
private void z0()
{
    Servant.proceed(true);
    locprn(5);
}

// Пытаемся взять левую вилку
private void z1()
{
    forks[num] = num+1;  // Это вилку взял данный философ
    locprn(1);
}

// Пытаемся взять правую вилку
private void z2()
{
    forks[(num+1)%5] = num+1;  // Это вилку взял данный философ
    locprn(3);
}

// Положили правую вилку обратно
private void z3()
{
    forks[(num+1)%5] = 0;
    locprn(7);
}
```

```
// Положили левую вилку обратно
private void z4()
{
    forks[num] = 0;
    locprn(6);
}

// Выходим из столовой
private void z5()
{
    Servant.proceed(false);
    locprn(8);
}

private PhilException createPhilException(int number)
{
    return new PhilException("Исключение: " + name + ": " +
            PhilConst.exceptionMap[number] + '\n');
}

private void printStateMessage()
{
    String[] map = PhilConst.stateMessageMap;
    if (!isLogged)
        return;
    try
    {
        switch (state)
        {
        case 0:
            if (x0)
                stateprn(map[0], set1);
            else
                stateprn(map[1], set1);
            break;

        case 1:
            if (x0)
                stateprn(map[2], set1);
            else
                stateprn(map[3], set1);
            break;

        case 2:
            if (x0)
                stateprn(map[4], set2);
            else
                stateprn(map[5], set2);
            break;

        case 3:
            if (x0)
                stateprn(map[6], set3);
            else
                stateprn(map[7], set3);
            break;
        }
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}

public static void prn(String str)
```

```
    {
        try
        {
            String data =  DateFormat.getInstance().format(new
java.util.Date());
            prn.insertString(prn.getLength(), new String(data +" "+ str + "\n"),
set);
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }

    public static void prn(String str, MutableAttributeSet set)
    {
        try
        {
            String data =  DateFormat.getInstance().format(new
java.util.Date());
            prn.insertString(prn.getLength(), new String(data +" "+ str + "\n"),
set);
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }

    public void locprn(int n)
    {
        if (isLogged)
            prn(name + " " + PhilConst.messageMap[n]);
    }

    public void stateprn(String str, MutableAttributeSet set)
    {
        try
        {
            prn.insertString(prn.getLength(), name + ": " + str + "\n", set);
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

Файл «Servant.java»

Класс слуга

```
package logic;

import javax.swing.text.MutableAttributeSet;
import javax.swing.text.SimpleAttributeSet;
import javax.swing.text.StyleConstants;
import java.awt.*;

public class Servant
{
    private static int state = 0;
    private static int x0=0;
    private static MutableAttributeSet set = null;
```

```
public Servant()
{
    set = new SimpleAttributeSet();
    StyleConstants.setForeground(set, new Color(255, 0, 255));
    StyleConstants.setFontFamily(set, "Courier New");
    StyleConstants.setBold(set, false);
    StyleConstants.setBackground(set, Color.yellow);
}

public static int getState()
{
    return state;
}

public static int getQuanity()
{
    return x0;
}
// Автомат A1
public static void proceed(boolean in)
{
    switch(state)
    {
    case 0:
        if (in && (x0 < 3))     // Еще есть свободные места
        {
            z0();
            state = 0;
            break;
        }

        if (in && (x0 == 3))    // Свободных мест больше не будет
        {
            z0();
            state = 1;
            break;
        }

        if (!in)                // Философ хочет выйти
        {
            z1();
            state = 0;
            break;
        }
    case 1:
        if (!in)                // Философ хочет выйти
        {
            z1();
            state = 0;
            break;
        }
    }
}

private static void z0()       // Впустить философа
{
    x0++;
    Man.prn("Слуга: Философ впущен", set);
    if (x0 == 4)
    {
        Man.prn("Слуга: больше никого не пущу", set);
    }
}
```

```
    private static void z1()       // Выпустить философа
    {
        x0--;
        Man.prn("Слуга: Философ выпущен", set);
        if (x0 == 3)
        {
            Man.prn("Слуга: сюда можно входить", set);
        }
    }
}
```

Файл «PhilConst.java»

Этот файл содержит константы, необходимые для работы программы

```
package logic;
/*
      Этот файл содержит значения констант, используемых в программе
*/
public class PhilConst
{
    public static final String[] names =
    {
        "Пифагор",
        "Аристотель",
        "Кант",
        "Платон",
        "Ньютон"
    };

    public static final String[] stateMessageMap =
    {
        "Иду есть",                                 // 0
        "Гуляю",                                    // 1
        "Нахожусь в столовой",                      // 2
        "Иду гулять",                               // 3
        "Сижу за столом",                           // 4
        "Встаю из-за стола. Отдаю левую вилку",     // 5
        "Ем",                                       // 6
        "Поел. Отдаю правую вилку",                 // 7
    };

    public static final String[] messageMap =
    {
        "хочет взять левую вилку",                  // 0
        "взял левую вилку",                         // 1
        "хочет взять правую вилку",                 // 2
        "взял правую вилку",                        // 3
        "Хочет войти",                              // 4
        "вошел",                                    // 5
        "кладет левую вилку на место",              // 6
        "кладет правую вилку на место",             // 7
        "выходит"                                   // 8
    };

    public static final String[] exceptionMap =
    {
        "Неверное состояние",
        "Не могу взять левую вилку",
        "Не могу взять правую вилку",
        "Не могу войти",
    };
```

```
    public static final String[] stateMap =
    {
        "Гуляю",
        "Нахожусь в столовой",
        "Сижу за столом",
        "Ем"
    };

    public static final String[] servantMessageMap =
    {
        "Философ %name хочет войти",
        "Философ %name хочет выйти",
        "Философ %name заходит",
        "Философ %name не может зайти"
    };

    protected PhilConst()
    {
    }
}
```

Файл «PhilException.java»

Этот класс используется для генерации исключений, которые затем перехватываются и отображаются в виде логов

```
package logic;
import javax.swing.*;
import javax.swing.text.*;
import java.awt.*;

/*
      Файл используется для вывода информации об ошибках (вилка занята, нельзя
      войти и т.п.) в панель «Логи» визуализатора
*/

public class PhilException extends Exception
{
    public static Document prn = null;
    public PhilException(String str)
    {
        super(str);
        MutableAttributeSet set1 = new SimpleAttributeSet();
        StyleConstants.setForeground(set1, Color.red);
        StyleConstants.setFontFamily(set1, "Courier New");
        try
        {
            prn.insertString(prn.getLength(),
                    str,
                    set1);
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }
}
```