Санкт-Петербургский государственный институт информационных технологий, механики и оптики

Кафедра «Компьютерные технологии»

А.Р. Ишметьев, А.Н. Ситников, А.А. Шалыто

Моделирование холодильника (фрагменты)

Проектная документация

Проект создан в рамках

«Движение за открытую проектную документацию»

http://is.ifmo.ru

Санкт-Петербург

2004

Содержание

Введение

Проект является примером использования технологии автоматного программирования с явным выделением состояний [1] при моделировании работы холодильника. Применение данной технологии лучше всего отвечает поставленной задаче: в отличие от традиционного программирования, автоматное программирование позволяет четко описать поведение устройства в любой момент времени. В качестве модели был выбран холодильник по причине небольшого числа состояний и событий, что позволило обеспечить наглядную демонстрацию этого подхода.

В русскоязычном Интернете оказалось трудно найти какие-либо статьи по моделированию бытовой техники. Однако, на сайте http://is.ifmo.ru в разделе «Проекты» имеется две работы по этой тематике [2, 3]. В них производится разборка системы управления кофеваркой на основе автоматного подхода.

Проект реализован на языке *Java* в виде приложения.

Перейдем к изложению рассматриваемого проекта.

1. Внешний вид модели

На рис.1 приведен внешний вид пользовательского интерфейса холодильника.

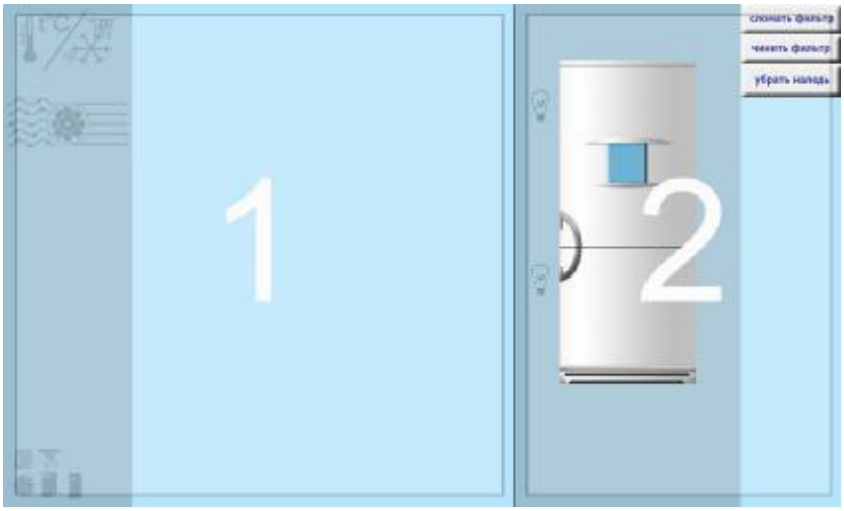


Рис.1. Вид пользовательского интерфейса

Рис.2. Панель действий

Окно интерфейса разделено на левую (1) и правую (2) части.

Левая часть – дисплей холодильника. Правая часть – панель действий, которую, в свою очередь, также можно условно разделить на две части (рис.2.):

- общий вид холодильника (1);
- кнопки (2).

«Общий вид холодильника» необходим для того, чтобы продемонстрировать работу «системы слежения за

открытыми дверями». Щелчок мыши по одной из закрытых дверей – открывает ее

Рис.3. Двери холодильника

(рис.3), если щелкнуть по открытой двери – она закроется.

Панель «Кнопки» содержит три кнопки: «Сломать фильтр», «Починить фильтр», «Убрать наледь» - с помощью которых можно изменять состояние объектов холодильника, чтобы проверить работу автоматов.

1.1. Основные части модели и ее устройство

Моделируется двухкамерный холодильник, в котором одна камера – холодильная, а другая – морозильная. По внешнему виду он ничем, кроме встроенного дисплея (рис.4) не отличается от обычного холодильника.

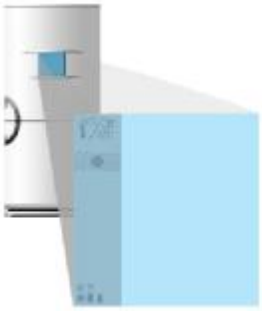


Рис.4. Укрупненное изображение дисплея

Холодильная камера содержит:

- очиститель воздуха (в качестве подобъекта имеет воздушный фильтр);
- лампу для освещения камеры;

- термометр.

Морозильная камера содержит:

- устройство, отслеживающее количество наледи;
- термометр;
- лампу для освещения камеры.

В состав холодильника входит также звуковая сигнализация. Она относится к обеим камерам и реализуется независимо от них.

«Дисплей» также нельзя отнести к определенной части холодильника, хотя он и расположен на двери холодильной камеры.

На дисплее можно получить информацию о:

- загрязненности воздушной массы в холодильной камере;
- температуре внутри камер;
- активности некоторых частей и т.д.

К техническим особенностям дисплея (рис.5) для данной модели холодильника, относится сенсорный экран, что существенно упрощает навигацию по «Меню» размещенном на этом экране.

Рис.5. Дисплей

Дисплей можно условно разделить на три части:

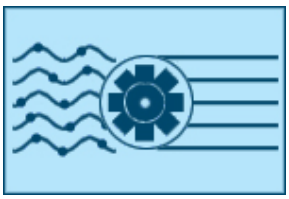
- сенсорный экран (1), меню которого состоит из двух пунктов;
- панель «Индикаторы» (2) отображает текущее состояние какого-либо из устройств холодильника;
- панель «Информационное поле» (3) отображает информацию по каждому пункту меню.

С помощью индикаторов можно быстро узнать основные сведения о состоянии отдельных частей и холодильнике в целом (например, необходимость ремонта и чистки холодильника ото льда). Однако, этого недостаточно, чтобы узнать полное состояние. В этом случае надо использовать «Меню».

1.2. Меню

Состоит из двух пунктов, которых вполне достаточно для слежения за состоянием составных частей холодильника (рис.6).

1.2.1. «Очиститель воздуха»



В этом пункте меню можно узнать информацию о рабочем состоянии очистителя, представленную в виде слов: «Включен», «Выключен» или «Сломан».

Фильтр очистителя может быть либо загрязнен, либо находиться в нормальном состоянии, не требующем замены. Уровень загрязненности фильтра тоже можно узнать из этого пункта меню.

Помимо перечисленных сведений в этом пункте меню содержится также информация о текущей загрязненности воздушной массы в холодильной камере. Загрязненность

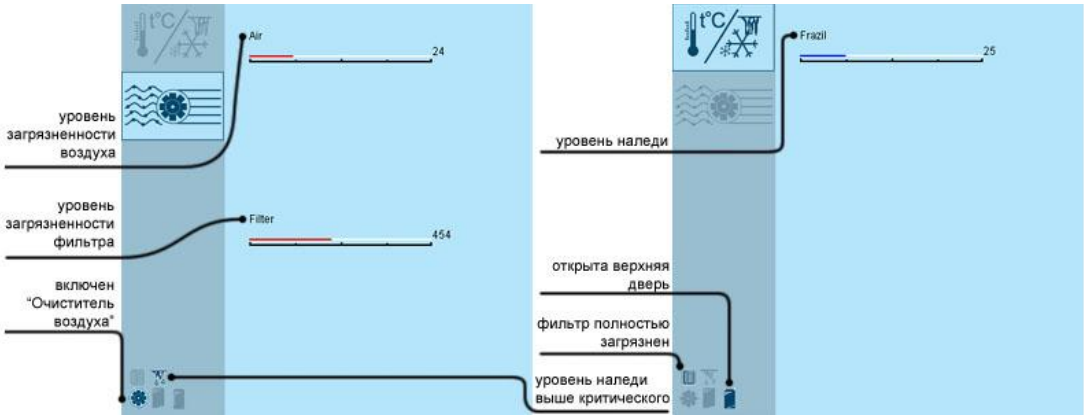
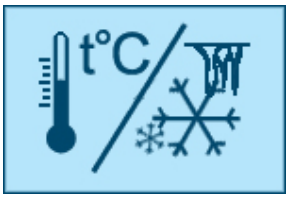


Рис.6. Вид меню модели

воздуха растет с ростом загрязненности фильтра (уменьшается пропускная способность фильтрующего элемента).

Уровень загрязненность фильтра и воздушной массы представлен в виде условных числовых значений, которые продублированы с помощью шкал. Это помогает лучше ориентироваться при оценке степени испорченности фильтра и воздуха.

1.2.2. «Температура и наледь»



Из этого пункта, в первую очередь, можно узнать такой важный для холодильника параметр, как температура (в градусах по Цельсию) в холодильной и морозильной камерах. Помимо цифровых значений есть шкалы с отметками критических значений температуры.

Еще одна полезная деталь – объем наледи в морозильной камере. Сколько скопилось наледи в объемных единицах узнать нельзя, но с помощью шкалы с метками можно предположить, когда будет достигнуто критическое значение.

1.3. *Индикаторы*

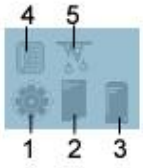


Рис.7. Индикаторы

Индикаторы (рис.7) являются источниками информации о состоянии составляющих холодильника, которая всегда должна быть перед глазами у пользователя. Перечислим индикаторы и кратко опишем их функциональность:

1 – «Очиститель воздуха работает». Индикатор горит, если очиститель работает;
2 – «Открыта дверь морозильной камеры»;
3 – «Открыта дверь холодильной камеры»;
4 – «Фильтр загрязнен». Индикатор горит, если фильтр исчерпал свой ресурс;
5 – «Скопление наледи». Индикатор горит, если в морозильнике скопилось много наледи.

2. Автоматный подход

В Приложении 1 приведена диаграмма классов моделируемой системы, состоящая из отдельных диаграмм. На диаграмме в модуле `Components` отображены автоматные классы, которые моделируют (в качестве примера) управление тремя составляющими системы.

В Приложении 2 приведен исходный код, реализующий указанную систему.

2.1. *Выделение автоматов*

Продемонстрируем эффективность автоматного подхода при моделировании не всего холодильника, а лишь трех его составных частей:

- очиститель воздуха;
- система слежения за открытыми дверями;
- датчик наледи.

2.2. *Автомат `A0` – «Очиститель воздуха»*

2.2.1. Словесное описание

Автомат «Очиститель воздуха» предназначен для контроля за объектом с одноименным названием.

К автомату, реализующему работу очистителя воздуха, предъявляются следующие требования:

- автомат должен реагировать на события, поступающие от пользователя, изменяющего состояние фильтра;
- автомат должен реагировать на изменение состояния воздушной массы в холодильной камере.

Для обеспечения выполнения первого требования используется панель «Кнопки» на рис. 2:

- пользователь может «сломать» очиститель воздуха, щелкнув по кнопке «Сломать фильтр». При этом процесс поломки осуществляется путем установления максимального значения загрязненности фильтра;
- пользователь может «починить» очиститель воздуха, щелкнув по кнопке «Починить фильтр». Процесс починки осуществляется путем установки нулевого значения загрязненности фильтра.

Для обеспечения выполнения второго требования используется функция, моделирующая загрязнение воздуха.

2.2.2. Нумерация и перечень состояний

Состояние 0. «Не работает». Очиститель воздуха находится в выключенном состоянии.

Состояние 1. «Работает». Очиститель воздуха находится во включенном состоянии.

Состояние 2. «Сломан». Фильтр загрязнен и требует замены.

2.2.3. Нумерация и перечень событий

e1 – загрязненность воздушной массы превысила норму. Событие формируется функцией изменения состояния воздуха.

e2 – загрязненность воздушной массы ниже нормы. Событие формируется функцией изменения состояния воздуха.

e3 – загрязненность фильтра превысила критическое значение. Событие поступает от обработчика состояния фильтра или от пользователя, нажатием на кнопку «Сломать фильтр».

e4 – фильтр заменен. Событие поступает от пользователя, нажатием на кнопку «Починить фильтр».

2.2.4. Нумерация и перечень выходных воздействий

z1 – включить очиститель воздуха и его индикатор.

z2 – выключить очиститель воздуха и его индикатор.

z3 – включить индикатор «Фильтр загрязнен».

z4 – выключить индикатор «Фильтр загрязнен».

2.2.5. Схема связей автомата

На рис. 8 показана схема связей автомата «Очиститель воздуха».

Рис.8. Схема связей автомата

2.2.6. Граф переходов

На рис. 9 показан граф переходов автомата «Очиститель воздуха».

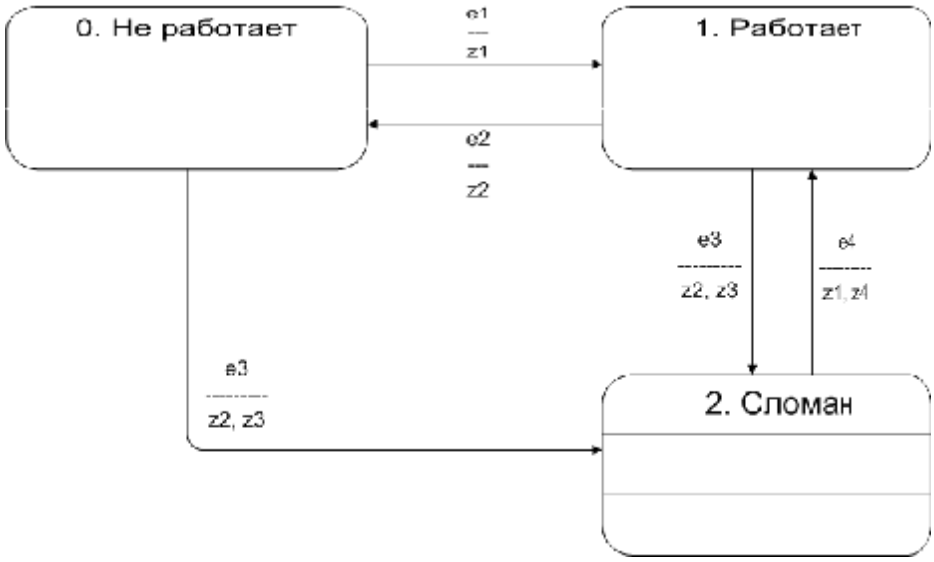


Рис.9. Граф переходов автомата «Очиститель воздуха»

2.3. Автомат A1 – «Система слежения за открытыми дверями»

2.3.1. Словесное описание

Автомат «Система слежения за открытыми дверями» предназначен для контроля за дверьми холодильника. Если дверь открыта, то на дисплее горит ее индикатор. Если после открытия прошло более 15 с, то включается звуковой сигнал. Если прошла минута, то включается громкий звуковой сигнал.

К автомату, реализующему работу данной системы, предъявляются требования:

- автомат должен реагировать на события, поступающие от пользователя, открывающего и закрывающего двери;
- автомат должен реагировать на изменение времени прошедшего с момента открытия двери.

Двери могут быть открыты или закрыты щелчком мыши по ним.

2.3.2. Нумерация и перечень состояний

Состояние 0. «Закрыта». Дверь, к которой относится автомат, закрыта.

Состояние 1. «Открыта». Дверь, к которой относится автомат, открыта.

Состояние 2. «Открыта более 15 с». Дверь, к которой относится автомат, открыта больше 15с.

Состояние 3. «Открыта более 60 с». Дверь, к которой относится автомат, открыта больше 60с.

Примечание: в программе время срабатывания звуковых сигналов уменьшено до 10 и 15 с соответственно.

2.3.3. Нумерация и перечень событий

e1 – дверь открыта. Событие поступает от пользователя.

e2 – дверь закрыта. Событие поступает от пользователя.

e3 – дверь открыта более 15 с. Событие поступает от общего обработчика событий.

e4 – дверь открыта более 60 с. Событие поступает от общего обработчика событий.

2.3.4. Нумерация и перечень выходных воздействий

z1 – включить индикатор открытой двери.

z2 – выключить индикатор открытой двери.

z3 – включить звуковой сигнал.

z4 – выключить звуковой сигнал.

z5 – включить громкий звуковой сигнал.

z6 – выключить громкий звуковой сигнал.

z7 – включить лампу освещения.

z8 – выключить лампу освещения.

2.3.5. Схема связей автомата

На рис. 10 показана схема связей автомата «Система слежения за открытыми дверьми».

Рис.10. Схема связей автомата «Система слежения за открытыми дверьми»

2.3.6. Граф переходов

На рис. 11 показан граф переходов автомата «Система слежения за открытыми дверями».

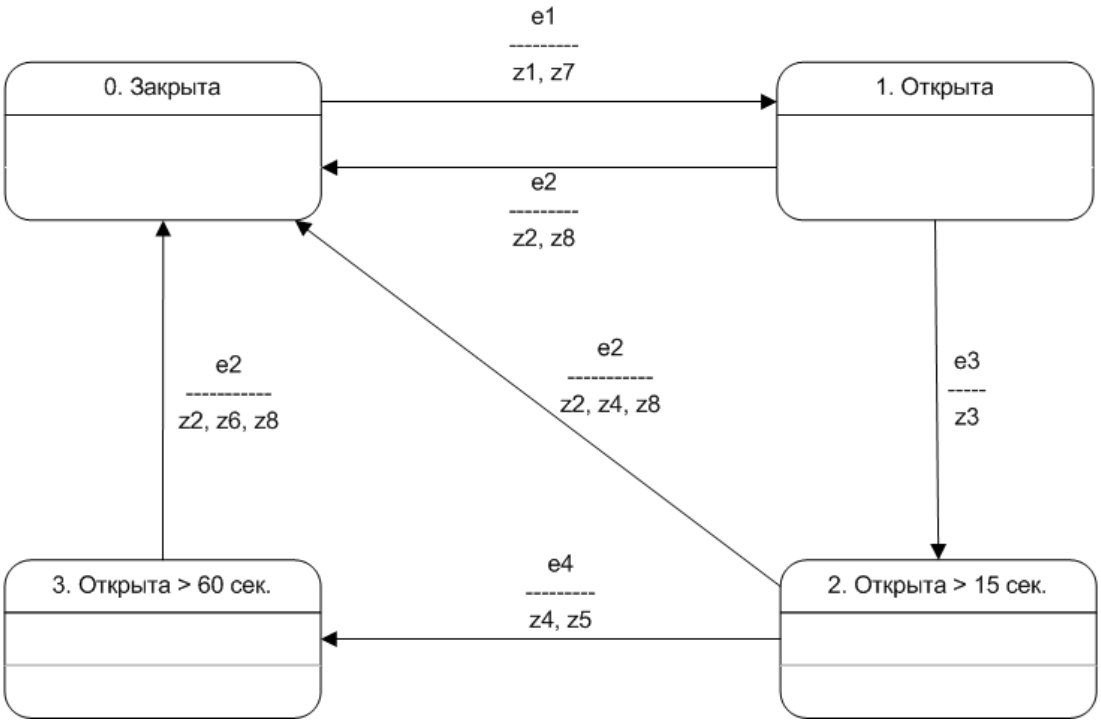


Рис.11. Граф переходов автомата «Система слежения за открытыми дверьми»

2.4. *Автомат A2 – «Датчик наледи»*

2.4.1. Словесное описание

Автомат «Датчик наледи» предназначен для контроля наледи в морозильной камере. Автомат содержит всего два состояния.

К автомату, реализующему работу датчика наледи, предъявляются требования:

- автомат должен реагировать на изменение количества наледи в морозильной камере;
- автомат должен реагировать на события, поступающие от пользователя, при нажатии на кнопку «Убрать наледь».

Как только объем наледи в морозильной камере превышает критическую отметку – загорается индикатор «Скопление наледи», говорящий о том, что камера нуждается в чистке.

2.4.2. Нумерация и перечень состояний

Состояние 0. «Норма». В морозильной камере количество наледи не превышает критическую отметку.

Состояние 1. «Много наледи». В морозильной камере количество наледи превышает критическую отметку.

2.4.3. Нумерация и перечень событий

e1 – количество наледи не превышает норму. Событие поступает от функции изменения количества наледи.

e2 – количество наледи превышает норму. Событие поступает от функции изменения количества наледи.

e3 – наледь вычищена. Событие поступает от пользователя, нажатием не кнопку «Убрать наледь».

2.4.4. Нумерация и перечень выходных воздействий

z1 – выключить индикатор «Скопление наледи».

z2 – включить индикатор «Скопление наледи».

2.4.5. Схема связей автомата

На рис. 12 показана схема связей автомата «Датчик наледи».

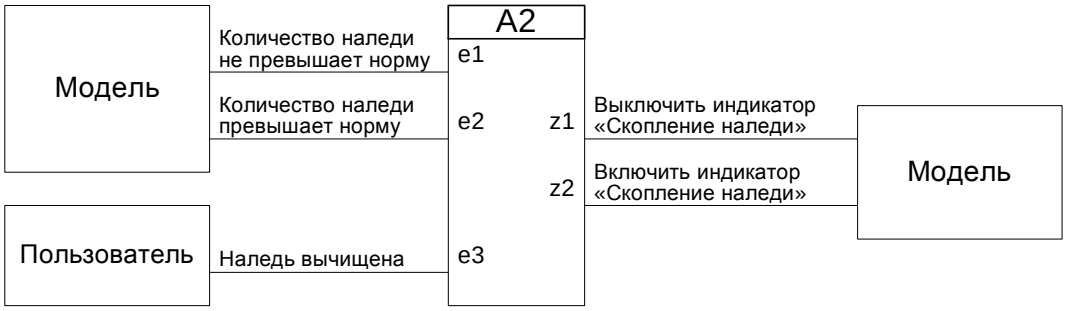


Рис.12. Схема связей автомата «Датчик наледи»

2.4.6. Граф переходов

На рис. 13 показан граф переходов автомата «Датчик наледи».

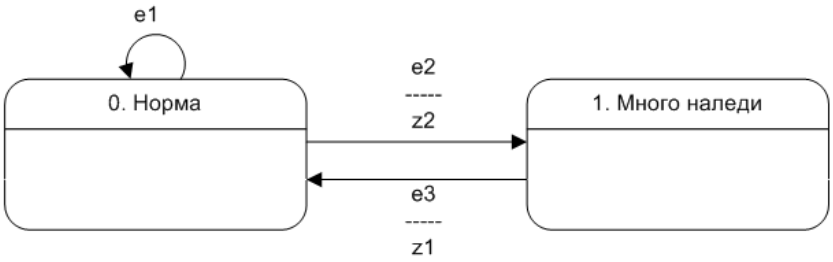


Рис.13. Граф переходов автомата «Датчик наледи»

3. Заключение

Как выше было отмечено, данная модель не исчерпывается тремя объектами (автоматами), которые выделены авторами проекта. Можно было бы с помощью автоматного программирования реализовать также и другие виды управления:

- дисплеем с управляющим меню;
- компрессором;
- системой охлаждения.

Однако авторы не преследовали цель сделать полноценную программную модель холодильника. Цель проекта – показать удобство использования автоматного программирования для задач рассматриваемого класса.

К положительным чертам автоматного программирования следует отнести:

- наглядность работы программируемого объекта. Достаточно одного взгляда на схему связей и граф переходов автомата, чтобы понять, как он устроен или что должно произойти в тот или иной момент;

- простота отладки. По сравнению с традиционным программированием, автоматный подход проще для анализа за счет ведения «лог-файла» в терминах автоматов (Приложение 3).

Пожалуй, единственной сложностью применения рассматриваемого автомата является процесс построения автомата на первых стадиях, когда он «рисуется на бумаге». Именно в этот момент надо учесть все детали и нюансы, чтобы потом не пришлось перестраивать весь код программы. Однако это сложности проектирования, а реализация при этом резко упрощается.

Литература

1. *Туккель Н.И., Шалыто А.А.* Система управления танком для игры **"Robocode".** Вариант **1.** Объектно-ориентированное программирование с явным выделением состояний**.** **http://is.ifmo.ru,** раздел **«**Проекты**».**
2. *Алексеев В.А., Ларионов А.В.* Сравнение программ управления кофеваркой **"Mark 4 Special Coffee Maker",** реализованных на основе нотации Буча и **SWITCH-**технологии**.** **http://is.ifmo.ru,** раздел **«**Проекты**».**
3. *Кессель С.В.* Разработка системы управления кофеваркой на основе автоматного подхода**.** **http://is.ifmo.ru,** раздел **«**Проекты**».**

Приложения

Приложение 1. Диаграммы классов

На рис.14 приведена схема пакета `refrigerator`, которая содержит три модуля: `Components, Debug, UI` - и основной класс `Refrigerator`.

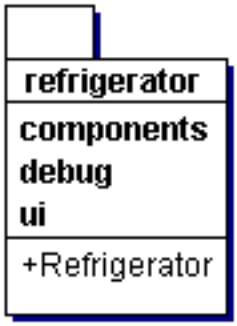


Рис.14. Пакет `refrigerator`

На рис.15 – 18 приведены диаграммы классов для модулей.

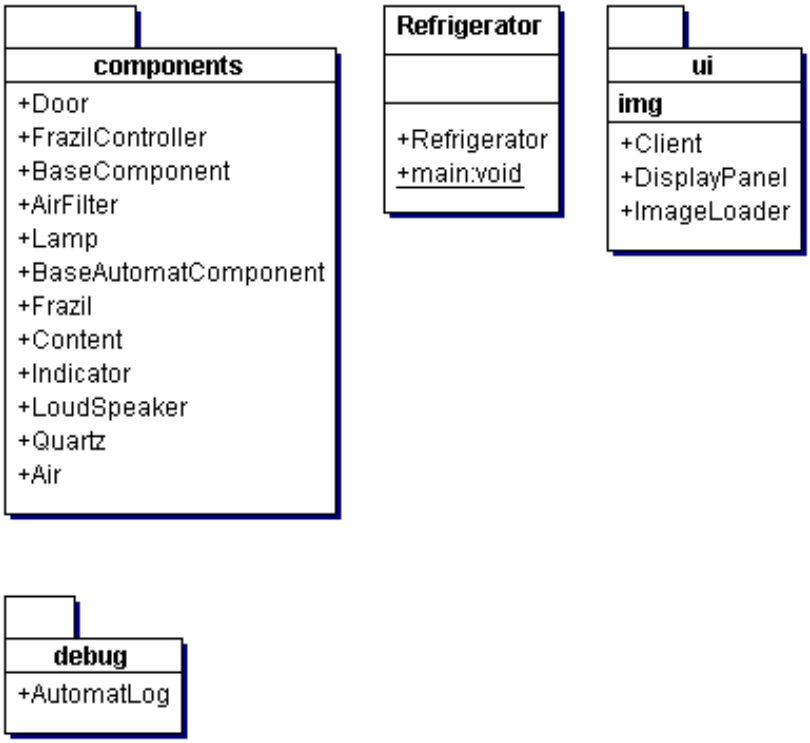


Рис.15. Модуль `Refrigerator`

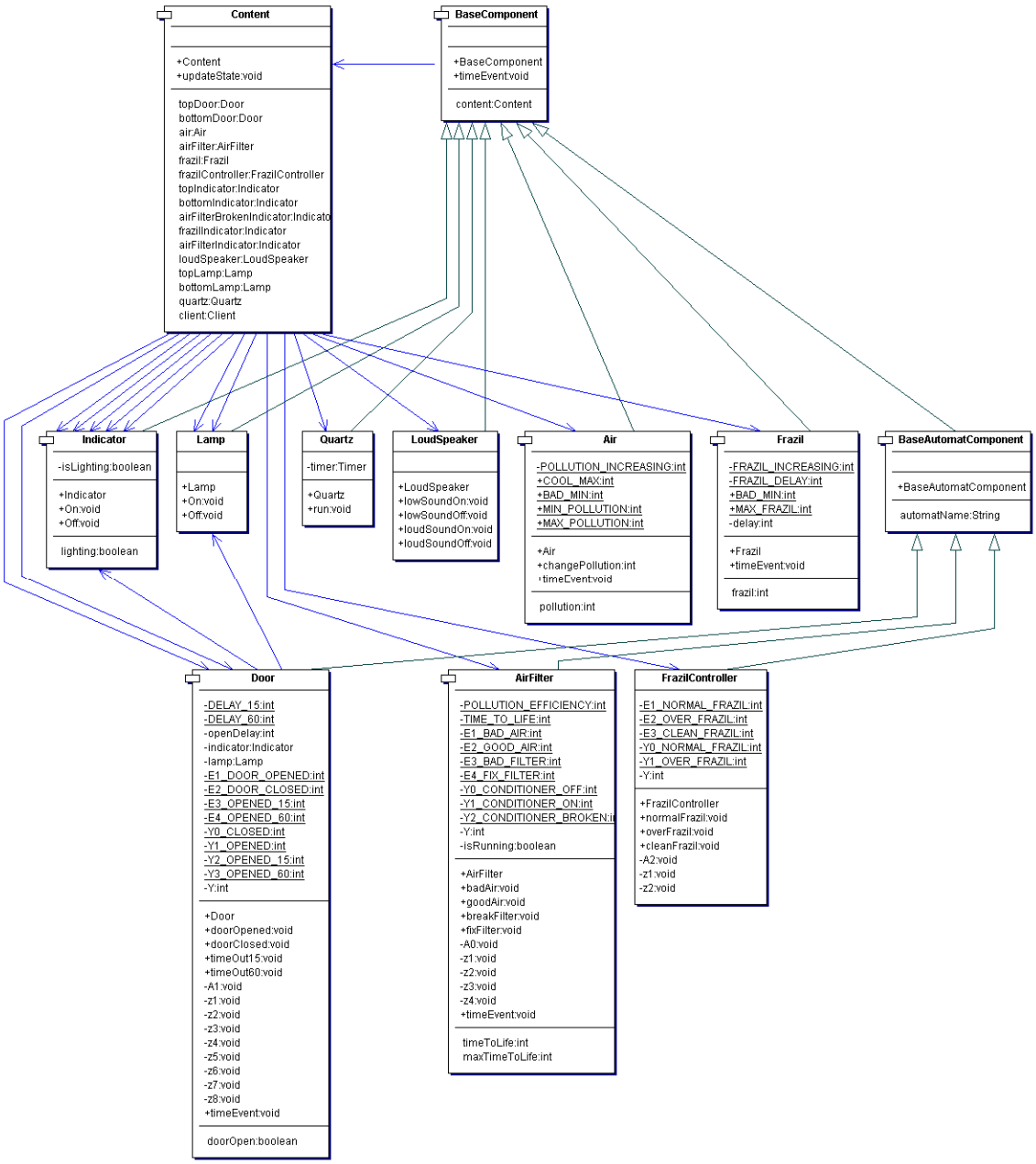


Рис.16. Модуль `Components`

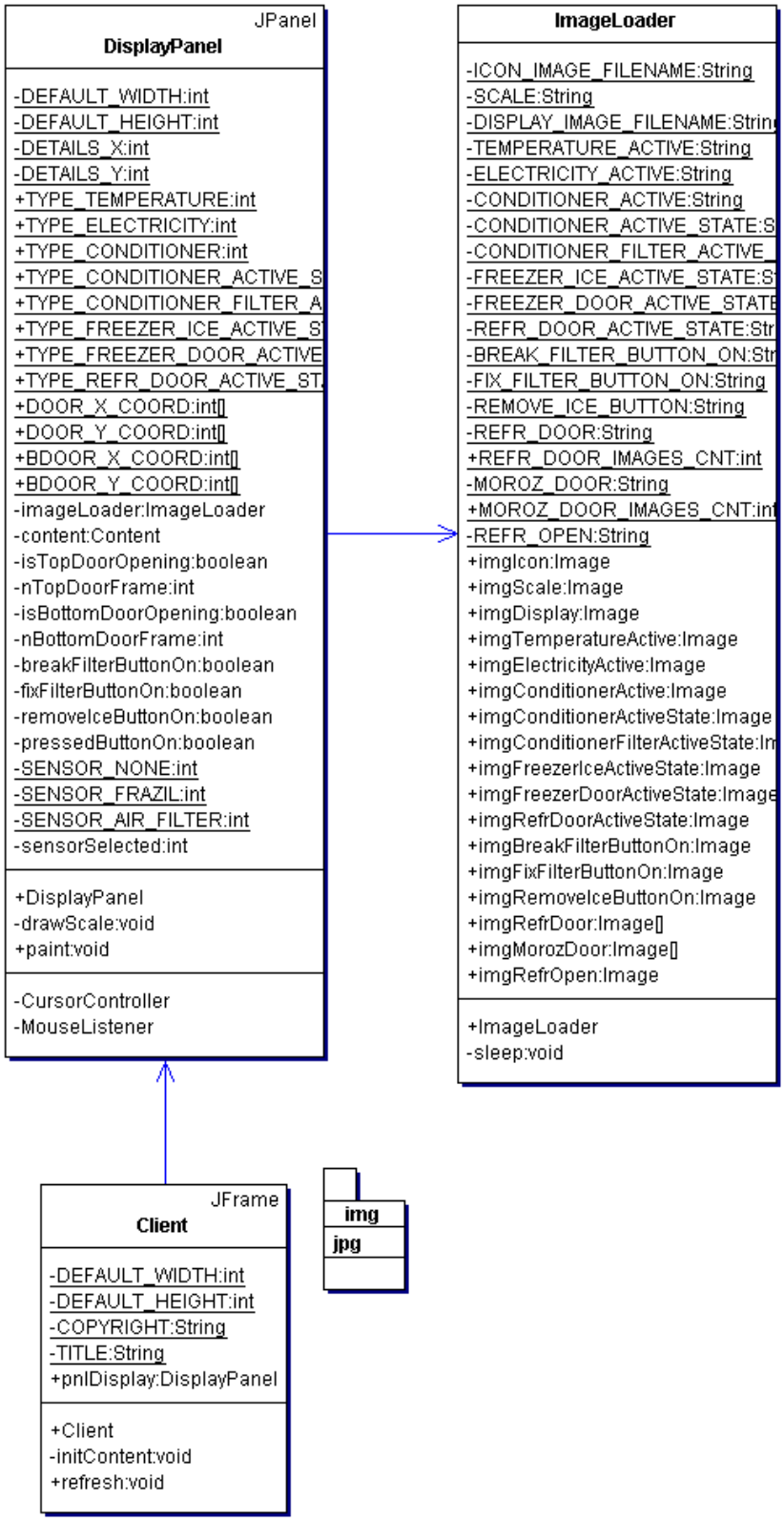


Рис.17. Модуль UI

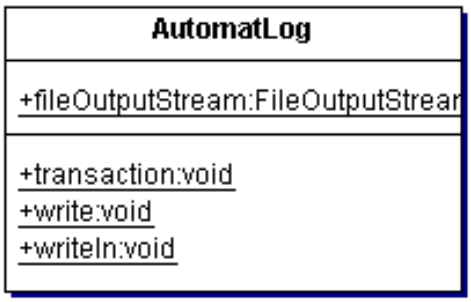


Рис.18. Модуль Debug

Приложение 2. Листинг модулей

Здесь приведены листинги модулей, реализующих автоматы и объекты, управляемые автоматами.

Листинг модуля **Refrigerator**

```
import org.ifmo.refrigerator.ui.Client;
import org.ifmo.refrigerator.components.Content;

/**
 * Базовый класс холодильника.
 * Создает окно пользователя "Client"
 */
public class Refrigerator {

    public Refrigerator() {
        (new Client(new Content())).show();
    }

    public static void main(String args[]) {
        new Refrigerator();
    }
}
```

Листинг модуля **Client**

```
import org.ifmo.refrigerator.components.Content;
import javax.swing.*;
import java.awt.*;

/**
 * Класс окна, на котором размещается DisplayPanel
 */
public class Client extends JFrame {
    private static final int DEFAULT_WIDTH = 750;
    private static final int DEFAULT_HEIGHT = 450;
```

```
    private static final String COPYRIGHT = "\u00A9";
    private static final String TITLE = "Refrigerator " + COPYRIGHT + " 2003 by
A.Ishmetyev & A.Sitnikov";

    public DisplayPanel pnlDisplay = null;

/**
 * Создание базового окна, на котором будут размещен DisplayPanel.
 * Загрузка картинок и иконок.
 */
    public Client(Content content) {
        super(TITLE);
        content.setClient(this);
        ImageLoader imageLoader = new ImageLoader(getToolkit());
        setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
        setResizable(false);
        setIconImage(imageLoader.imgIcon);
        initContent(imageLoader, content);
    }

/**
 * Создание и размещение DisplayPanel на базовом окне
 */
    private void initContent(ImageLoader imageLoader, Content content) {
        DisplayPanel pnlDisplay = new DisplayPanel(imageLoader, content);
        JPanel contentPanel = new JPanel();
        contentPanel.setPreferredSize(new Dimension(DEFAULT_WIDTH,
DEFAULT_HEIGHT));
        contentPanel.setLayout(new BorderLayout());
        contentPanel.add(pnlDisplay, BorderLayout.CENTER);
        setContentPane(contentPanel);
        pack();
    }

    public void refresh() {
        this.repaint();
    }

}
```

Листинг модуля `DisplayPanel`

```
import org.ifmo.refrigerator.components.Content;

import java.util.Timer;
import javax.swing.*;
import java.awt.*;
import java.awt.event.MouseMotionAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.MouseAdapter;
import java.util.TimerTask;
```

```
/**
 * Класс, отвечающий за отображение холодильника на экране.
 */
public class DisplayPanel extends JPanel {
   private static final int DEFAULT_WIDTH = 600;
   private static final int DEFAULT_HEIGHT = 600;
   private static final int DETAILS_X = 140;
   private static final int DETAILS_Y = 40;

   public static final int TYPE_TEMPERATURE = 0;
   public static final int TYPE_ELECTRICITY = 1;
   public static final int TYPE_CONDITIONER = 2;
   public static final int TYPE_CONDITIONER_ACTIVE_STATE = 3;
   public static final int TYPE_CONDITIONER_FILTER_ACTIVE_STATE = 4;
   public static final int TYPE_FREEZER_ICE_ACTIVE_STATE = 5;
   public static final int TYPE_FREEZER_DOOR_ACTIVESTATE = 6;
   public static final int TYPE_REFR_DOOR_ACTIVE_STATE = 7;
   public static final int DOOR_X_COORD[] = new int[] {490, 515, 563, 594};
   public static final int DOOR_Y_COORD[] = new int[] {65, 65, 65, 63};
   public static final int BDOOR_X_COORD[] = new int[] {490, 515, 563, 594};
   public static final int BDOOR_Y_COORD[] = new int[] {225, 225, 225, 223};

   private ImageLoader imageLoader = null;
   private Content content = null;

   private boolean isTopDoorOpening = true;
   private int nTopDoorFrame = 0;
   private boolean isBottomDoorOpening = true;
   private int nBottomDoorFrame = 0;

   private boolean breakFilterButtonOn = false;
   private boolean fixFilterButtonOn = false;
   private boolean removeIceButtonOn = false;
   private boolean pressedButtonOn = false;

   private final static int SENSOR_NONE = 0;
   private final static int SENSOR_FRAZIL = 1;
   private final static int SENSOR_AIR_FILTER = 2;
   private int sensorSelected = SENSOR_NONE;

/**
 * Контроллер курсора мыши.
 */
   private class CursorController extends MouseMotionAdapter {
/**
 * Проверка попадания мыши в область смены указателя курсора.
 */
        public void mouseMoved(MouseEvent e) {
           if ((e.getX() > 650 && e.getY() < 90)  ||
              (e.getX() < 115 && e.getY() < 155) ||
              (e.getX() > 490 && e.getX() < 610 && e.getY() > 60 && e.getY() < 340))
{
              setCursor(new Cursor(Cursor.HAND_CURSOR));
           } else {
              setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
           }
        }
```

```
/**
 * Проверка попадания мыши в область кнопок.
 */
        public void mouseDragged(MouseEvent e) {
           pressedButtonOn = false;
           if (e.getX() > 650 && e.getY() < 30)
              pressedButtonOn = breakFilterButtonOn;
           else if (e.getX() > 650 && e.getY() >= 30 && e.getY() < 60)
              pressedButtonOn = fixFilterButtonOn;
           else if (e.getX() > 650 && e.getY() >= 60 && e.getY() < 90)
              pressedButtonOn = removeIceButtonOn;
           repaint();
        }
     }

/**
 * Класс обработки событий от мыши.
 */
   private class MouseListener extends MouseAdapter {

   /**
    * Попадание курсора мыши в сенсорную панель.
    */
      private void testSensorButtons(Point mouse) {
         final Rectangle rectFrazil = new Rectangle(0, 0, 112, 75);
         final Rectangle rectAirFilter = new Rectangle(0, 76, 112, 75);
         final Rectangle rectNone = new Rectangle(0, 150, 112, 450);
         if (rectFrazil.contains(mouse))
            sensorSelected = SENSOR_FRAZIL;
         else if (rectAirFilter.contains(mouse))
            sensorSelected = SENSOR_AIR_FILTER;
         else if (rectNone.contains(mouse))
            sensorSelected = SENSOR_NONE;
      }

/**
 * Попадание курсора мыши в область дверей холодильника.
 */
      private void testDoors(Point mouse) {
         final Rectangle topDoor = new Rectangle(490, 60, 120, 165);
         final Rectangle bottomDoor = new Rectangle(490, 225, 120, 115);
         if (topDoor.contains(mouse)) {
            isTopDoorOpening = !content.getTopDoor().isDoorOpen();
            if (isTopDoorOpening)
               content.getTopDoor().doorOpened();
            else
               content.getTopDoor().doorClosed();

            nTopDoorFrame = 0;
            Timer timer = new Timer();
            timer.schedule(new TimerTask() {
               public void run() {
                  while (nTopDoorFrame < (imageLoader.REFR_DOOR_IMAGES_CNT - 1))
{
                     repaint();
                     nTopDoorFrame++;
                     try {
```

```
                        Thread.sleep(100);
                     } catch (Exception e) {
                        e.printStackTrace();
                     }
                  }
                  repaint();
               }
            }, 0);
         } else if (bottomDoor.contains(mouse)) {
            isBottomDoorOpening = !content.getBottomDoor().isDoorOpen();
            if (isBottomDoorOpening)
               content.getBottomDoor().doorOpened();
            else
               content.getBottomDoor().doorClosed();
            nBottomDoorFrame = 0;
            Timer timer = new Timer();
            timer.schedule(new TimerTask() {
               public void run() {
                  while (nBottomDoorFrame <
(imageLoader.MOROZ_DOOR_IMAGES_CNT - 1)) {
                     repaint();
                     nBottomDoorFrame++;
                     try {
                        Thread.sleep(100);
                     } catch (Exception e) {
                        e.printStackTrace();
                     }
                  }
                  repaint();
               }
            }, 0);
         }
      }

/**
 * Нажать кнопку.
 */
      public void mousePressed(MouseEvent e) {
         testDoors(e.getPoint());
         testSensorButtons(e.getPoint());
         pressedButtonOn = true;
         if (e.getX() > 650 && e.getY() < 30)
            breakFilterButtonOn = true;
         else if (e.getX() > 650 && e.getY() >= 30 && e.getY() < 60)
            fixFilterButtonOn = true;
         else if (e.getX() > 650 && e.getY() >= 60 && e.getY() < 90)
            removeIceButtonOn = true;
         else
            pressedButtonOn = false;
         repaint();
      }

/**
 * Отпустить кнопку.
 */
      public void mouseReleased(MouseEvent e) {
         if (pressedButtonOn) {
```

```
            if (breakFilterButtonOn)
               content.getAirFilter().breakFilter();
            else if (fixFilterButtonOn)
               content.getAirFilter().fixFilter();
            else if (removeIceButtonOn)
               content.getFrazilController().cleanFrazil();
         }
         breakFilterButtonOn = false;
         fixFilterButtonOn = false;
         removeIceButtonOn = false;
         repaint();
      }
   }

/**
 * Инициализация объекта класса DisplayPanel.
 */
   public DisplayPanel(ImageLoader imgLoader, Content content) {
      this.imageLoader = imgLoader;
      this.content = content;
      this.isTopDoorOpening = !content.getTopDoor().isDoorOpen();
      this.isBottomDoorOpening = !content.getBottomDoor().isDoorOpen();
      this.setPreferredSize(new Dimension(DEFAULT_WIDTH, DEFAULT_HEIGHT));
      this.addMouseMotionListener(new CursorController());
      this.addMouseListener(new MouseListener());
   }

/**
 * Отрисовка шкалы.
 */
   private void drawScale(Graphics g, String text, int x, int y, int value, int maxValue,
Color lineColor) {
      final int scaleWidth = imageLoader.imgScale.getWidth(null);
      g.drawString(text, x, y);
      y += g.getFontMetrics().getHeight();
      g.drawImage(imageLoader.imgScale, x, y, null);
      Color tColor = g.getColor();
      g.setColor(lineColor);
      g.fillRect(x, y, (scaleWidth * value) / maxValue, 2);
      g.setColor(tColor);
      g.drawString(Integer.toString(value), x + scaleWidth, y);
   }

/**
 * Перерисовка.
 */
   public void paint(Graphics g) {
      //---------Задний фон----------
      g.drawImage(imageLoader.imgDisplay, 0, 0, null);

      //---------Холодильник----------
      g.drawImage(imageLoader.imgRefrOpen, 490, 60, null);
      if (isTopDoorOpening)
         g.drawImage(imageLoader.imgRefrDoor[nTopDoorFrame],
DOOR_X_COORD[nTopDoorFrame], DOOR_Y_COORD[nTopDoorFrame], null);
      else
```

```
            g.drawImage(imageLoader.imgRefrDoor[3 - nTopDoorFrame],
DOOR_X_COORD[3 - nTopDoorFrame], DOOR_Y_COORD[3 - nTopDoorFrame], null);
        if (isBottomDoorOpening)
            g.drawImage(imageLoader.imgMorozDoor[nBottomDoorFrame],
BDOOR_X_COORD[nBottomDoorFrame], BDOOR_Y_COORD[nBottomDoorFrame], null);
        else
            g.drawImage(imageLoader.imgMorozDoor[3 - nBottomDoorFrame],
BDOOR_X_COORD[3 - nBottomDoorFrame], BDOOR_Y_COORD[3 - nBottomDoorFrame],
null);

        //------------Кнопки-----------------
        if (pressedButtonOn) {
            if (breakFilterButtonOn) g.drawImage(imageLoader.imgBreakFilterButtonOn,
650, 0, null);
            if (fixFilterButtonOn) g.drawImage(imageLoader.imgFixFilterButtonOn, 650, 30,
null);
            if (removeIceButtonOn) g.drawImage(imageLoader.imgRemoveIceButtonOn,
650, 60, null);
        }

        //-------------Индикаторы-------------
        if (content.getTopIndicator().isLighting())
            g.drawImage(imageLoader.imgFreezerDoorActiveState, 30, 420, null);
        if (content.getBottomIndicator().isLighting())
            g.drawImage(imageLoader.imgRefrDoorActiveState, 52, 420, null);
        if (content.getAirFilterBrokenIndicator().isLighting())
            g.drawImage(imageLoader.imgConditionerFilterActiveState, 8, 398, null);
        if (content.getAirFilterIndicator().isLighting())
            g.drawImage(imageLoader.imgConditionerActiveState, 8, 420, null);
        if (content.getFrazilIndicator().isLighting())
            g.drawImage(imageLoader.imgFreezerIceActiveState, 30, 398, null);

        //-----------Датчики-----------
        switch (sensorSelected) {
            case SENSOR_FRAZIL:
                g.drawImage(imageLoader.imgTemperatureActive, 0, 0, null);
                drawScale(g, "Frazil", DETAILS_X, DETAILS_Y,
                        content.getFrazil().getFrazil(),
                        content.getFrazil().MAX_FRAZIL,
                        Color.blue);
                break;
            case SENSOR_AIR_FILTER:
                g.drawImage(imageLoader.imgConditionerActive, 0, 75, null);
                drawScale(g, "Air", DETAILS_X, DETAILS_Y,
                        content.getAir().getPollution(),
                        content.getAir().MAX_POLLUTION,
                        Color.red);
                drawScale(g, "Filter", DETAILS_X, DETAILS_Y + 200,
                        content.getAirFilter().getMaxTimeToLife() -
content.getAirFilter().getTimeToLife(),
                        content.getAirFilter().getMaxTimeToLife(),
                        Color.red);
                break;
            case SENSOR_NONE:
                break;
        }
    }
```

```
}
```

Листинг модуля **ImageLoader**

```
import java.awt.*;

/**
 * Класс "Загрузка картинок".
 */
public class ImageLoader {

    private static final String ICON_IMAGE_FILENAME =
"src/org/ifmo/refrigerator/ui/img/Burger.gif";
    private static final String SCALE = "src/org/ifmo/refrigerator/ui/img/scale.jpg";
    private static final String DISPLAY_IMAGE_FILENAME =
"src/org/ifmo/refrigerator/ui/img/display.jpg";
    private static final String TEMPERATURE_ACTIVE =
"src/org/ifmo/refrigerator/ui/img/TempIceMenuActive.gif";
    private static final String CONDITIONER_ACTIVE =
"src/org/ifmo/refrigerator/ui/img/ConditionerActive.gif";
    private static final String CONDITIONER_ACTIVE_STATE =
"src/org/ifmo/refrigerator/ui/img/ConditionerActiveState.gif";
    private static final String CONDITIONER_FILTER_ACTIVE_STATE =
"src/org/ifmo/refrigerator/ui/img/ConditionerFilterActive.gif";
    private static final String FREEZER_ICE_ACTIVE_STATE =
"src/org/ifmo/refrigerator/ui/img/FreezerChamberIceActive.gif";
    private static final String FREEZER_DOOR_ACTIVE_STATE =
"src/org/ifmo/refrigerator/ui/img/FreezerDoorOpenActive.gif";
    private static final String REFR_DOOR_ACTIVE_STATE =
"src/org/ifmo/refrigerator/ui/img/RefrDoorOpenActive.gif";
    private static final String BREAK_FILTER_BUTTON_ON =
"src/org/ifmo/refrigerator/ui/img/brakeFilterP.gif";
    private static final String FIX_FILTER_BUTTON_ON =
"src/org/ifmo/refrigerator/ui/img/fixFilterP.gif";
    private static final String REMOVE_ICE_BUTTON =
"src/org/ifmo/refrigerator/ui/img/removeIceP.gif";
    private static final String REFR_DOOR = "src/org/ifmo/refrigerator/ui/img/TdoorOp";
    public static final int REFR_DOOR_IMAGES_CNT = 4;
    private static final String MOROZ_DOOR =
"src/org/ifmo/refrigerator/ui/img/BdoorOp";
    public static final int MOROZ_DOOR_IMAGES_CNT = 4;
    private static final String REFR_OPEN =
"src/org/ifmo/refrigerator/ui/img/refrigeratorOpen.jpg";
    private static final String LIGHT_ON = "src/org/ifmo/refrigerator/ui/img/lighton.gif";
    private static final String LIGHT_OFF = "src/org/ifmo/refrigerator/ui/img/lightoff.gif";

    public Image imgIcon = null;
    public Image imgScale = null;
    public Image imgDisplay = null;
    public Image imgTemperatureActive = null;
    public Image imgConditionerActive = null;
    public Image imgConditionerActiveState = null;
    public Image imgConditionerFilterActiveState = null;
    public Image imgFreezerIceActiveState = null;
    public Image imgFreezerDoorActiveState = null;
    public Image imgRefrDoorActiveState = null;
```

```
    public Image imgBreakFilterButtonOn = null;
    public Image imgFixFilterButtonOn = null;
    public Image imgRemoveIceButtonOn = null;
    public Image imgRefrDoor[] = new Image[REFR_DOOR_IMAGES_CNT];
    public Image imgMorozDoor[] = new Image[MOROZ_DOOR_IMAGES_CNT];
    public Image imgRefrOpen = null;
    public Image imgLightOn = null;
    public Image imgLightOff = null;

    // Загрузка картинок из файлов.
    public ImageLoader(Toolkit toolkit) {
        imgDisplay = toolkit.createImage(DISPLAY_IMAGE_FILENAME);
        imgIcon = toolkit.createImage(ICON_IMAGE_FILENAME);
        imgScale = toolkit.createImage(SCALE);
        imgTemperatureActive = toolkit.createImage(TEMPERATURE_ACTIVE);
        imgConditionerActive = toolkit.createImage(CONDITIONER_ACTIVE);
        imgConditionerActiveState = toolkit.createImage(CONDITIONER_ACTIVE_STATE);
        imgConditionerFilterActiveState =
toolkit.createImage(CONDITIONER_FILTER_ACTIVE_STATE);
        imgFreezerIceActiveState = toolkit.createImage(FREEZER_ICE_ACTIVE_STATE);
        imgFreezerDoorActiveState =
toolkit.createImage(FREEZER_DOOR_ACTIVE_STATE);
        imgRefrDoorActiveState = toolkit.createImage(REFR_DOOR_ACTIVE_STATE);
        imgBreakFilterButtonOn = toolkit.createImage(BREAK_FILTER_BUTTON_ON);;
        imgFixFilterButtonOn = toolkit.createImage(FIX_FILTER_BUTTON_ON);
        imgRemoveIceButtonOn = toolkit.createImage(REMOVE_ICE_BUTTON);
        imgLightOn = toolkit.createImage(LIGHT_ON);
        imgLightOff = toolkit.createImage(LIGHT_OFF);

        for (int i = 0; i < REFR_DOOR_IMAGES_CNT; i++) {
            imgRefrDoor[i] = toolkit.createImage(REFR_DOOR + i + ".jpg");
        }
        for (int i = 0; i < MOROZ_DOOR_IMAGES_CNT; i++) {
            imgMorozDoor[i] = toolkit.createImage(MOROZ_DOOR + i + ".jpg");
        }
        imgRefrOpen = toolkit.createImage(REFR_OPEN);

        while (!toolkit.prepareImage(imgScale, 1, 1, null)) {
            sleep();
        }
        while (!toolkit.prepareImage(imgDisplay, 1, 1, null)) {
            sleep();
        }
        while (!toolkit.prepareImage(imgIcon, 1, 1, null)) {
            sleep();
        }
        while (!toolkit.prepareImage(imgTemperatureActive, 1, 1, null)) {
            sleep();
        }
        while (!toolkit.prepareImage(imgConditionerActive, 1, 1, null)) {
            sleep();
        }
        while (!toolkit.prepareImage(imgConditionerActiveState, 1, 1, null)) {
            sleep();
        }
        while (!toolkit.prepareImage(imgConditionerFilterActiveState, 1, 1, null)) {
            sleep();
```

```
        }
        while (!toolkit.prepareImage(imgFreezerIceActiveState, 1, 1, null)) {
            sleep();
        }
        while (!toolkit.prepareImage(imgFreezerDoorActiveState, 1, 1, null)) {
            sleep();
        }
        while (!toolkit.prepareImage(imgRefrDoorActiveState, 1, 1, null)) {
            sleep();
        }
        while (!toolkit.prepareImage(imgBreakFilterButtonOn, 1, 1, null)) {
            sleep();
        }
        while (!toolkit.prepareImage(imgFixFilterButtonOn, 1, 1, null)) {
            sleep();
        }
        while (!toolkit.prepareImage(imgRemoveIceButtonOn, 1, 1, null)) {
            sleep();
        }
        while (!toolkit.prepareImage(imgLightOn, 1, 1, null)) {
            sleep();
        }
        while (!toolkit.prepareImage(imgLightOff, 1, 1, null)) {
                sleep();
        }

        for (int i = 0; i < REFR_DOOR_IMAGES_CNT; i++) {
            while (!toolkit.prepareImage(imgRefrDoor[i], 1, 1, null)) {
                sleep();
            }
        }
        for (int i = 0; i < REFR_DOOR_IMAGES_CNT; i++) {
            while (!toolkit.prepareImage(imgMorozDoor[i], 1, 1, null)) {
                sleep();
            }
        }
        while (!toolkit.prepareImage(imgRefrOpen, 1, 1, null)) {
            sleep();
        }
    }

    private void sleep() {
        try {
            Thread.sleep(20);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

Листинг модуля **Content**

```
import org.ifmo.refrigerator.ui.Client;
```

```
/**
 * Класс, содержащий все себе все компоненты холодильника.
 */
public class Content {
    // индикатор открытости верхнего отсека
    private Indicator topIndicator = new Indicator(this);
    // индикатор открытости нижнего отсека
    private Indicator bottomIndicator = new Indicator(this);
    // лампа верхнего отсека
    private Lamp topLamp = new Lamp(this);
    // лампа нижнего отсека
    private Lamp bottomLamp = new Lamp(this);
    // верхняя дверь
    private Door topDoor = new Door(this, topIndicator, topLamp);
    // нижняя дверь
    private Door bottomDoor = new Door(this, bottomIndicator, bottomLamp);
    // воздух
    private Air air = new Air(this);
    // фильтр
    private AirFilter airFilter = new AirFilter(this);
    // наледь
    private Frazil frazil = new Frazil(this);
    // датчик наледи
    private FrazilController frazilController = new FrazilController(this);
    // индикатор загрязненности фильтра
    private Indicator airFilterBrokenIndicator = new Indicator(this);
    // индикатор наледи
    private Indicator frazilIndicator = new Indicator(this);
    // индикатор работы воздушного фильтра
    private Indicator airFilterIndicator = new Indicator(this);
    // динамик
    private LoudSpeaker loudSpeaker = new LoudSpeaker(this);
    // кварц
    private Quartz quartz = new Quartz(this);

    // Конструктор.
    public Content() {
        quartz.run();
        airFilter.setAutomatName("Air Filter");
        topDoor.setAutomatName("Top Door");
        bottomDoor.setAutomatName("Bottom Door");
        frazilController.setAutomatName("Frazil Controller");
    }

    public Door getTopDoor() {
        return topDoor;
    }

    public Door getBottomDoor() {
        return bottomDoor;
    }

    public Air getAir() {
        return air;
    }

    public AirFilter getAirFilter() {
```

```
    return airFilter;
}

public Frazil getFrazil() {
    return frazil;
}

public FrazilController getFrazilController() {
    return frazilController;
}

public Indicator getTopIndicator() {
    return topIndicator;
}

public Indicator getBottomIndicator() {
    return bottomIndicator;
}

public Indicator getAirFilterBrokenIndicator() {
    return airFilterBrokenIndicator;
}

public Indicator getFrazilIndicator() {
    return frazilIndicator;
}

public Indicator getAirFilterIndicator() {
    return airFilterIndicator;
}

public LoudSpeaker getLoudSpeaker() {
    return loudSpeaker;
}

public Lamp getTopLamp() {
    return topLamp;
}

public Lamp getBottomLamp() {
    return bottomLamp;
}

public Quartz getQuartz() {
    return quartz;
}

// Функция обновления.
private Client client = null;

public void updateState() {
    if (client != null)
        client.repaint();
}

public void setClient(Client client) {
    this.client = client;
```

```
    }
}
```

Листинг модуля **BaseAutomatComponent**

```
/**
 * Базовый класс компонентов, которые содержат внутри себя автоматы.
 */
public class BaseAutomatComponent extends BaseComponent {
    private String automatName = "";

    public BaseAutomatComponent(Content content) {
        super(content);
    }

    public String getAutomatName() {
        return automatName;
    }

    public void setAutomatName(String automatName) {
        this.automatName = automatName;
    }
}
```

Листинг модуля **BaseComponent**

```
/**
 * Базовый класс компонентов, которые не содержат внутри себя автоматы.
 */
public class BaseComponent {
    private Content content;

    public BaseComponent(Content content) {
        this.content = content;
    }

    public Content getContent() {
        return content;
    }

    public void timeEvent() {
    }
}
```

Листинг модуля **Air**

```
/**
 * Класс создания объекта "Воздух".
 */
public class Air extends BaseComponent {
    private final static int POLLUTION_INCREASING = 1;
```

```
    public final static int COOL_MAX = 10;
    public final static int BAD_MIN = 50;
    public final static int MIN_POLLUTION = 0;
    public final static int MAX_POLLUTION = 100;

    private int pollution = 0;

/**
 * Конструктор класса.
 * Получает ссылку на родителя – Content.
 */
    public Air(Content content) {
        super(content);
    }

/**
 * Получить текущий уровень загрязнения воздуха.
 */
    public int getPollution() {
        return pollution;
    }

/**
 * Функция изменения состояния воздуха.
 */
    public int changePollution(int delta) {
        pollution += delta;
        if (pollution < MIN_POLLUTION)
            pollution = MIN_POLLUTION;
        else if (pollution > MAX_POLLUTION)
            pollution = MAX_POLLUTION;
        if (pollution <= COOL_MAX)
            getContent().getAirFilter().goodAir();
        else if (pollution >= BAD_MIN)
            getContent().getAirFilter().badAir();
        return pollution;
    }

/**
 * Изменение состояния воздуха со временем.
 */
    public void timeEvent() {
        changePollution(POLLUTION_INCREASING);
    }
}
```

Листинг модуля `AirFilter`

```
import org.ifmo.refrigerator.debug.AutomatLog;


/**
 * Класс "Воздушный фильтр".
 */
public class AirFilter extends BaseAutomatComponent {
```

```
    private final static int POLLUTION_EFFICIENCY = -2;
    private final static int TIME_TO_LIFE = 1000;

/**
 * Конструктор класса.
 * Получает ссылку на родителя – Content.
 */
    public AirFilter(Content content) {
        super(content);
    }
/**
 * Событие "Загрязненность воздушной массы превысила норму".
 */
    public void badAir() {
        A0(E1_BAD_AIR);
    }
/**
 * Событие "Загрязненность воздушной массы ниже нормы".
 */
    public void goodAir() {
        A0(E2_GOOD_AIR);
    }
/**
 * Событие "Загрязненность фильтра превысила критическое значение".
 */
    public void breakFilter() {
        timeToLife = 0;
        A0(E3_BAD_FILTER);
    }
/**
 * Событие "Фильтр заменен".
 */
    public void fixFilter() {
        timeToLife = TIME_TO_LIFE;
        A0(E4_FIX_FILTER);
    }

    // События.
    private static final int E1_BAD_AIR = 1;
    private static final int E2_GOOD_AIR = 2;
    private static final int E3_BAD_FILTER = 3;
    private static final int E4_FIX_FILTER = 4;

    // Состояния.
    private static final int Y0_CONDITIONER_OFF = 0;
    private static final int Y1_CONDITIONER_ON = 1;
    private static final int Y2_CONDITIONER_BROKEN = 2;

    // Текущее состояние.
    private int Y = Y0_CONDITIONER_OFF;

/**
 * Автомат A0.
 */
    private void A0(int e) {
        int oldY = Y;
        switch (Y) {
```

```
        case Y0_CONDITIONER_OFF:
            if (e == E1_BAD_AIR) {
                Y = Y1_CONDITIONER_ON;
                z1();
            } else
            if (e == E3_BAD_FILTER) {
                Y = Y2_CONDITIONER_BROKEN;
                z2();
                z3();
            }
            break;

        case Y1_CONDITIONER_ON:
            if (e == E2_GOOD_AIR) {
                Y = Y0_CONDITIONER_OFF;
                z2();
            } else
            if (e == E3_BAD_FILTER) {
                Y = Y2_CONDITIONER_BROKEN;
                z2();
                z3();
            }
            break;

        case Y2_CONDITIONER_BROKEN:
            if (e == E4_FIX_FILTER) {
                Y = Y1_CONDITIONER_ON;
                z1();
                z4();
            }
            break;
    }
    if (Y != oldY)
        AutomatLog.transaction(0, getAutomatName(), oldY, Y);
}

// Исходящее воздействие - "Включить очиститель воздуха и его индикатор".
private void z1() {
    isRunning = true;
    getContent().getAirFilterIndicator().On();
}
// Исходящее воздействие - "Выключить очиститель воздуха и его индикатор".
private void z2() {
    isRunning = false;
    getContent().getAirFilterIndicator().Off();
}
// Исходящее воздействие - "Включить индикатор "Фильтр загрязнен".
private void z3() {
    getContent().getAirFilterBrokenIndicator().On();
}
// Исходящее воздействие - "Выключить индикатор "Фильтр загрязнен".
private void z4() {
    getContent().getAirFilterBrokenIndicator().Off();
}

private boolean isRunning = false;
private int timeToLife = TIME_TO_LIFE;
```

```
    // Измененеие состояния фильтра со временем.
    public void timeEvent() {
        if ((timeToLife > 0) && isRunning) {
            getContent().getAir().changePollution(POLLUTION_EFFICIENCY);
            timeToLife--;
            if (timeToLife == 0) {
                breakFilter();
            }
        }
    }

    // Получить время жизни фильтра.
    public int getTimeToLife() {
        return timeToLife;
    }

    public int getMaxTimeToLife() {
        return TIME_TO_LIFE;
    }
}
```

Листинг модуля **Door**

```
import org.ifmo.refrigerator.debug.AutomatLog;


/**
 * Класс двери холодильника.
 */
public class Door extends BaseAutomatComponent {
    private final static int DELAY_15 = 150;
    private final static int DELAY_60 = 600;
    private int openDelay = 0;

    private Indicator indicator = null;
    private Lamp lamp = null;

/**
 * Конструктор.
 */
    public Door(Content content, Indicator indicator, Lamp lamp) {
        super(content);
        this.indicator = indicator;
        this.lamp = lamp;
        indicator.Off();
        lamp.Off();
    }

    // Проверка открытости двери.
    public boolean isDoorOpen() {
        return (Y != Y0_CLOSED);
    }
```

```
    // Событие "Дверь открыта".
    public void doorOpened() {
        A1(E1_DOOR_OPENED);
    }
    // Событие "Дверь закрыта".
    public void doorClosed() {
        A1(E2_DOOR_CLOSED);
    }

    // Событие "Дверь открыта больше 15с.".
    public void timeOut15() {
        A1(E3_OPENED_15);
    }

    // Событие "Дверь открыта больше 60с.".
    public void timeOut60() {
        A1(E4_OPENED_60);
    }

    // События.
    private static final int E1_DOOR_OPENED = 1;
    private static final int E2_DOOR_CLOSED = 2;
    private static final int E3_OPENED_15 = 3;
    private static final int E4_OPENED_60 = 4;

    // Состояния.
    private static final int Y0_CLOSED = 0;
    private static final int Y1_OPENED = 1;
    private static final int Y2_OPENED_15 = 2;
    private static final int Y3_OPENED_60 = 3;

    // Текущее состояние.
    private int Y = Y0_CLOSED;

/**
 * Автомат A1.
 */
    private void A1(int e) {
        int oldY = Y;
        switch (Y) {
            case Y0_CLOSED:
                if (e == E1_DOOR_OPENED) {
                    Y = Y1_OPENED;
                    z1();
                    z7();
                }
                break;

            case Y1_OPENED:
                if (e == E2_DOOR_CLOSED) {
                    Y = Y0_CLOSED;
                    z2();
                    z8();
                } else if (e == E3_OPENED_15) {
                    Y = Y2_OPENED_15;
                    z3();
                }
```

```
            break;

        case Y2_OPENED_15:
            if (e == E2_DOOR_CLOSED) {
                Y = Y0_CLOSED;
                z2();
                z4();
                z8();
            } else if (e == E4_OPENED_60) {
                Y = Y3_OPENED_60;
                z4();
                z5();
            }
            break;

        case Y3_OPENED_60:
            if (e == E2_DOOR_CLOSED) {
                Y = Y0_CLOSED;
                z2();
                z6();
                z8();
            }
            break;
    }
    if (Y != oldY)
        AutomatLog.transaction(1, getAutomatName(), oldY, Y);
}

// Исходящее воздействие "Включить индикатор открытой двери".
private void z1() {
    indicator.On();
}

// Исходящее воздействие "Выключить индикатор открытой двери".
private void z2() {
    indicator.Off();
}

// Исходящее воздействие "Включить звуковой сигнал".
private void z3() {
    getContent().getLoudSpeaker().lowSoundOn();
}

// Исходящее воздействие "Выключить звуковой сигнал".
private void z4() {
    getContent().getLoudSpeaker().lowSoundOff();
}

// Исходящее воздействие "Включить громкий звуковой сигнал".
private void z5() {
    getContent().getLoudSpeaker().loudSoundOn();
}

// Исходящее воздействие "Выключить громкий звуковой сигнал".
private void z6() {
    getContent().getLoudSpeaker().loudSoundOff();
}
```

```
    // Исходящее воздействие "Включить лампу освещения".
    private void z7() {
        lamp.On();
    }

    // Исходящее воздействие "Выключить лампу освещения".
    private void z8() {
        lamp.Off();
    }

    // Обновление по времени.
    public void timeEvent() {
        if (isDoorOpen()) {
            openDelay++;
            if (openDelay == DELAY_15)
                timeOut15();
            else if (openDelay == DELAY_60)
                timeOut60();
        } else {
            openDelay = 0;
        }
    }
}
```

Листинг модуля **Frazil**

```
package org.ifmo.refrigerator.components;

/**
 * Класс "Наледь".
 */
public class Frazil extends BaseComponent {
    private final static int FRAZIL_INCREASING = 1;
    private final static int FRAZIL_DELAY = 5;
    public final static int BAD_MIN = 90;
    public final static int MAX_FRAZIL = 100;

    private int frazil = 0;
    private int delay = 10;

    // Конструктор.
    public Frazil(Content content) {
        super(content);
    }

    // Текущий уровень наледи.
    public int getFrazil() {
        return frazil;
    }

    // Изменить количество наледи.
    public void setFrazil(int frazil) {
        if (frazil > MAX_FRAZIL)
            frazil = MAX_FRAZIL;
```

```
        if (frazil > BAD_MIN)
            getContent().getFrazilController().overFrazil();
        else
            getContent().getFrazilController().normalFrazil();
        this.frazil = frazil;
    }

    // Обновление по времени.
    public void timeEvent() {
        delay--;
        if (delay <= 0) {
            setFrazil(getFrazil() + FRAZIL_INCREASING);
            delay = FRAZIL_DELAY;
        }
    }
}
```

Листинг модуля `FrazilController`

```
import org.ifmo.refrigerator.debug.AutomatLog;

/**
 * Класс "Дачтик наледи".
 */
public class FrazilController extends BaseAutomatComponent {

    // Конструктор.
    public FrazilController(Content content) {
        super(content);
    }

    // Событие "Количество наледи не превышает норму".
    public void normalFrazil() {
        A2(E1_NORMAL_FRAZIL);
    }

    // Событие "Количество наледи превышает норму".
    public void overFrazil() {
        A2(E2_OVER_FRAZIL);
    }

    // Событие "Наледь вычищена".
    public void cleanFrazil() {
        getContent().getFrazil().setFrazil(0);
        A2(E3_CLEAN_FRAZIL);
    }

    // События.
    private static final int E1_NORMAL_FRAZIL = 1;
    private static final int E2_OVER_FRAZIL = 2;
    private static final int E3_CLEAN_FRAZIL = 3;

    // Состояния.
    private static final int Y0_NORMAL_FRAZIL = 0;
    private static final int Y1_OVER_FRAZIL = 1;
```

```
   // Текущее состояние.
   private int Y = Y0_NORMAL_FRAZIL;

/**
 * Автомат A2.
 */
   private void A2(int e) {
      int oldY = Y;
      switch (Y) {
         case Y0_NORMAL_FRAZIL:
            if (e == E1_NORMAL_FRAZIL)
               ;
            else if (e == E2_OVER_FRAZIL) {
               Y = Y1_OVER_FRAZIL;
               z2();
            }
            break;

         case Y1_OVER_FRAZIL:
            if (e == E3_CLEAN_FRAZIL) {
               Y = Y0_NORMAL_FRAZIL;
               z1();
            }
            break;
      }
      if (Y != oldY)
         AutomatLog.transaction(2, getAutomatName(), oldY, Y);
   }

   // Исходящее воздействие "Выключить индикатор "Скопление наледи".
   private void z1() {
      getContent().getFrazilIndicator().Off();
   }

   // Исходящее воздействие "Включить индикатор "Скопление наледи".
   private void z2() {
      getContent().getFrazilIndicator().On();
   }
}
```

Листинг модуля `Indicator`

```
/**
 * Класс "Индикатор".
 */
public class Indicator extends BaseComponent {
   private boolean isLighting = false;

   // Конструктор.
   public Indicator(Content content) {
      super(content);
   }
```

```
    // Включить индикатор.
    public void On() {
        isLighting = true;
    }

    //Выключить индикатор.
    public void Off() {
        isLighting = false;
    }

    // Получить состояние индикатора.
    public boolean isLighting() {
        return isLighting;
    }
}
```

Листинг модуля **Lamp**

```
/**
 * Класс "Лампа".
 */
public class Lamp extends BaseComponent {

    public boolean lamp;
    // Конструктор
    public Lamp(Content content) {
        super(content);
        lamp = false;
    }

    // Включить лампу.
    public void On() {
        lamp = true;
    }

    // Выключить лампу.
    public void Off() {
        lamp = false;
    }

    // Получить состояние лампы.
    public boolean isOn() {
        return lamp;
    }
}
```

Листинг модуля **LoudSpeaker**

```
import java.io.*;
import javax.sound.midi.*;
```

```
/**
 * Класс "Звуковая сигнализация".
 */
public class LoudSpeaker extends BaseComponent {

   public boolean lowSound;
   public boolean loudSound;
   private static final String
lowSoundSrc="src/org/ifmo/refrigerator/ui/sound/benny.mid";
   private static final String loudSoundSrc="src/org/ifmo/refrigerator/ui/sound/lion
king.mid";
   private Sequencer playLoud, playLow;

   // Загрузка звукового файла в объект.
   private Sequencer LoadMidi(String fName) {
      Sequencer play = null;
      try {
         play = MidiSystem.getSequencer();
         play.open();
         File MidiFile = new File(fName);
         Sequence seq = MidiSystem.getSequence(MidiFile);
         play.setSequence(seq);
      } catch (Exception e) {
         System.out.println("NO PLAYER");
      }
      return play;
   }

   // Конструкотр.
   public LoudSpeaker(Content content) {
      super(content);
      lowSound = false;
      loudSound = false;
      playLow = LoadMidi(lowSoundSrc);
      playLoud = LoadMidi(loudSoundSrc);
   }

   // Включить слабый звуковой сигнал.
   public void lowSoundOn() {
      playLow.setMicrosecondPosition(0);
      playLow.start();
      lowSound = true;
   }

   // Выключить слабый звуковой сигнал.
   public void lowSoundOff() {
      lowSound = false;
      playLow.stop();
   }

   // Включить громкий звуковой сигнал.
   public void loudSoundOn() {
      playLow.stop();
      playLoud.setMicrosecondPosition(0);
      playLoud.start();
      loudSound = true;
   }
```

```
    // Выключить громкий звуковой сигнал.
    public void loudSoundOff() {
        playLoud.stop();
        loudSound = false;
    }
}
```

Листинг модуля Quartz

```
import java.util.Timer;
import java.util.TimerTask;

/**
 * Класс таймера, производит обновление состояний компонент системы и делает
запрос на перерисовку.
 */
public class Quartz extends BaseComponent {
    // Таймер.
    private Timer timer = new Timer();

    public Quartz(Content content) {
        super(content);
    }

    //  Обновление состояний компонент системы.
    public void run() {
        timer.schedule(new TimerTask() {
            public void run() {
                while (true) {
                    Content content = getContent();
                    content.getAir().timeEvent();
                    content.getAirFilter().timeEvent();
                    content.getAirFilterBrokenIndicator().timeEvent();
                    content.getBottomDoor().timeEvent();
                    content.getBottomIndicator().timeEvent();
                    content.getAirFilterIndicator().timeEvent();
                    content.getFrazil().timeEvent();
                    content.getFrazilController().timeEvent();
                    content.getFrazilIndicator().timeEvent();
                    content.getTopDoor().timeEvent();
                    content.getLoudSpeaker().timeEvent();
                    content.getTopIndicator().timeEvent();
                    content.getTopLamp().timeEvent();
                    content.getBottomLamp().timeEvent();
                    content.updateState();
                    try {
                        Thread.sleep(100);
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            }
        }, 0);
    }
```

```
}
```

Листинг модуля **AutomatLog**

```
import java.io.FileNotFoundException;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;

/**
 * Класс "Лог".
 */
public class AutomatLog {
   public static FileOutputStream fileOutputStream;

   // Формирование строки лог-файла.
   synchronized public static void transaction(int autoNum, String autoName, int from,
int to) {
      java.util.Date time = new java.util.Date();
      time = java.util.Calendar.getInstance(java.util.Locale.getDefault()).getTime();
      write(time.toString() + "| ");
      write("A" + autoNum + " ");
      write("<" + autoName + ">: ");
      writeln("has changed state from " + from + " to " + to);
   }

   // Написать строку символов.
   synchronized public static void write(String l) {
      try {
         for (int i = 0; i < l.length(); i++) {
            fileOutputStream.write((byte)(l.charAt(i)));
         };
      } catch (IOException e) { };
   };

   // Написать строку символов и сделать переход на новую строку.
   synchronized public static void writeln(String l) {
      try {
         for (int i = 0; i < l.length(); i++) {
            fileOutputStream.write((byte)(l.charAt(i)));
         };
         fileOutputStream.write(13);
      } catch (IOException e) { };
   };

   static {
      try {
         fileOutputStream = new FileOutputStream(new File("log.txt"));
      } catch (FileNotFoundException e) { };
   }
}
```

Приложение 3. Пример лог-файла

Во время работы программы ведется лог-файл, который хранит информацию об изменении состояния автоматов в следующем формате:

<Дата события**>|<**ИД_Автомата**><**Имя_Автомата**>:has changed state from <**Бывшее_Состояние_Автомата**> to <**Новое_Состояние_Автомата**>**. Все изменения хранятся в файле log.txt.

Ниже приведен пример лог-файла работы программы.

Fri Oct 08 16:11:20 GMT+04:00 2004| A0 <Air Filter>: has changed state from 0 to 1
Fri Oct 08 16:11:23 GMT+04:00 2004| A1 <Top Door>: has changed state from 0 to 1
Fri Oct 08 16:11:25 GMT+04:00 2004| A0 <Air Filter>: has changed state from 1 to 0
Fri Oct 08 16:11:26 GMT+04:00 2004| A1 <Top Door>: has changed state from 1 to 0
Fri Oct 08 16:11:27 GMT+04:00 2004| A1 <Bottom Door>: has changed state from 0 to 1
Fri Oct 08 16:11:29 GMT+04:00 2004| A0 <Air Filter>: has changed state from 0 to 1
Fri Oct 08 16:11:30 GMT+04:00 2004| A1 <Bottom Door>: has changed state from 1 to 0
Fri Oct 08 16:11:33 GMT+04:00 2004| A0 <Air Filter>: has changed state from 1 to 0
Fri Oct 08 16:11:34 GMT+04:00 2004| A0 <Air Filter>: has changed state from 0 to 2
Fri Oct 08 16:11:48 GMT+04:00 2004| A0 <Air Filter>: has changed state from 2 to 1
Fri Oct 08 16:11:57 GMT+04:00 2004| A0 <Air Filter>: has changed state from 1 to 0
Fri Oct 08 16:12:02 GMT+04:00 2004| A0 <Air Filter>: has changed state from 0 to 1
Fri Oct 08 16:12:05 GMT+04:00 2004| A2 <Frazil Controller>: has changed state from 0 to 1
Fri Oct 08 16:12:06 GMT+04:00 2004| A0 <Air Filter>: has changed state from 1 to 0
Fri Oct 08 16:12:10 GMT+04:00 2004| A0 <Air Filter>: has changed state from 0 to 1
Fri Oct 08 16:12:12 GMT+04:00 2004| A2 <Frazil Controller>: has changed state from 1 to 0
Fri Oct 08 16:12:15 GMT+04:00 2004| A0 <Air Filter>: has changed state from 1 to 0
Fri Oct 08 16:12:19 GMT+04:00 2004| A0 <Air Filter>: has changed state from 0 to 1