МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Лабораторная работа № 2

по дисциплине «Современные информационные технологии»
на тему ”Разработка графического интерфейса. Классы-коллекции. Паттерны проектирования поведения объектов”

Студент Кузьмин Д.С.
Группа АВТ-318
Преподаватель Васюткина И.А.
Вариант 8

Новосибирск 2015 г.

Цель работы

1. Познакомиться с основными компонентами построения графических интерфейсов библиотек AWT и Swing в программах на Java. Изучить классы менеджеров компоновки.
2. Изучить назначение классов-коллекций, их виды, и методы работы с классами-коллекциями.

Задание варианта

Вариант задания:

Список транспортных средств на дороге состоит из двух категорий: автомобили и мотоциклы. Автомобили генерируются каждые N_1 секунд с вероятностью P_1. Мотоциклы генерируются каждые N_2 секунд с вероятностью P_2.

Задание

Доработать программу, созданную в лабораторной работе № 1:

1. Поделить рабочую область окна приложения на 2 части. Визуализация переносится в одну часть окна, панель управления в другую;
2. Добавить кнопки «Старт» и «Стоп» в панель управления. Они должны запускать и останавливать симуляцию соответственно. Если симуляция остановлена, то кнопка «Стоп» должна блокироваться. Если симуляция идет, то блокируется кнопка «Старт». Клавиши B и E должны функционировать по-прежнему;
3. Добавить переключатель «Показывать информацию», который разрешает отображение модального диалога из 7 пункта задания;
4. Добавить группу из 2 исключающих переключателей: «Показывать время симуляции» и «Скрывать время симуляции». Клавиша T должна функционировать по-прежнему;
5. Используя различные менеджеры компоновки, сформировать интерфейс пользователя согласно индивидуальному заданию;
6. Добавить в программу главное в меню и панель инструментов, в которых продублировать основные команды вашего интерфейса пользователя;
7. При остановке симуляции должно появляться модальное диалоговое окно (при условии, что оно разрешено) с информацией о количестве и типе сгенерированных объектов, а также времени симуляции. Вся информация выводится в элементе TextArea, недоступном для редактирования. В диалоговом окне должно быть 2 кнопки: «ОК» и «Отмена». При нажатии на «ОК» симуляции останавливается, а при нажатии на «Отмена», соответственно продолжается;
8. Предусмотреть проверку данных вводимых пользователем. При вводе неверного значения обрабатывать исключительную ситуацию: выставлять значение по умолчанию и выводить диалоговое окно с сообщением об ошибке;
9. Реализовать следующие элементы управления:
    - Периоды рождения объектов – текстовые поля;

- Для задания вероятностей рождения объектов комбобокс и список (шаг значений 10%);
- Дополнить интерфейс поясняющими метками.

Приложение А. Листинг программы

SideBarView.java

```
package com.xotonic.lab.sit;

import com.xotonic.lab.sit.settings.SettingsController;
import com.xotonic.lab.sit.settings.SettingsView;

import javax.swing.*;


public class SideBarView implements SettingsView {

    private SettingsController controller;

    private JButton sideBarStart;
    private JButton sideBarStop;
    private JCheckBox sideBarInfoToggle;
    private JRadioButton sideBarTimeShow;
    private JRadioButton sideBarTimeHide;

    public SideBarView(
            JButton sideBarStart,
            JButton sideBarStop,
            JCheckBox sideBarInfoToggle,
            JRadioButton sideBarTimeShow,
            JRadioButton sideBarTimeHide

            // TODO
        /*,
        JPanel factoriesSettingsContainer,
        JPanel factoryItemPrototype*/
    ) {
        this.sideBarStart = sideBarStart;
        this.sideBarStop = sideBarStop;
        this.sideBarInfoToggle = sideBarInfoToggle;
        this.sideBarTimeShow = sideBarTimeShow;
        this.sideBarTimeHide = sideBarTimeHide;

        ButtonGroup group = new ButtonGroup();
        group.add(sideBarTimeShow);
        group.add(sideBarTimeHide);

        setListeners();
    }

    private void setListeners() {
```

```
        sideBarStart.addActionListener(a -> controller.
          setStart());
        sideBarStop.addActionListener(a -> controller.
          setStop());
        sideBarInfoToggle.addActionListener(a ->
                controller.setShowInfo(sideBarInfoToggle.
                  isSelected()));
        sideBarTimeShow.addActionListener(a -> controller
          .setShowTime(true));
        sideBarTimeHide.addActionListener(a -> controller
          .setShowTime(false));
    }

    @Override
    public void OnSimulationStart() {
        sideBarStart.setEnabled(false);
        sideBarStop.setEnabled(true);
    }

    @Override
    public void OnSimulationStop() {
        sideBarStop.setEnabled(false);
        sideBarStart.setEnabled(true);
    }

    @Override
    public void OnShowInfo() {
        sideBarInfoToggle.setSelected(true);
    }

    @Override
    public void OnHideInfo() {
        sideBarInfoToggle.setSelected(false);

    }

    @Override
    public void OnShowTime() {
        sideBarTimeShow.setSelected(true);
    }

    @Override
    public void OnHideTime() {
        sideBarTimeHide.setSelected(true);

    }

    public void setController(SettingsController
      controller) {
        this.controller = controller;
    }
}
```

MenuView.java

```
package com.xotonic.lab.sit;


import com.xotonic.lab.sit.settings.SettingsController;
import com.xotonic.lab.sit.settings.SettingsView;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

import javax.swing.*;

public class MenuView implements SettingsView {
    private Logger log = LogManager.getLogger(MenuView.
      class.getName());

    private SettingsController controller;
    private JMenuBar menuBar;
    private JMenuItem startItem;
    private JMenuItem stopItem;
    private JCheckBoxMenuItem showInfoItem;
    private JRadioButtonMenuItem showTimeItem;
    private JRadioButtonMenuItem hideTimeItem;


    public MenuView() {
        createMenuBar();
    }


    private void createMenuBar() {
        JMenu menuFile, menuSimulation;

        //Create the menu bar.
        menuBar = new JMenuBar();

        //Build the first menu.
        menuFile = new JMenu("File");
        menuBar.add(menuFile);
        JMenuItem nopeItem = new JMenuItem("Not
          implemented");
        nopeItem.setEnabled(false);
        menuFile.add(nopeItem);

        menuSimulation = new JMenu("Simulation");
        menuBar.add(menuSimulation);

        //a group of JMenuItems
        startItem = new JMenuItem("Start");
        menuSimulation.add(startItem);
        stopItem = new JMenuItem("Stop");
        menuSimulation.add(stopItem);

        //a group of radio button menu items
```

```
        menuSimulation.addSeparator();
        ButtonGroup group = new ButtonGroup();

        showTimeItem = new JRadioButtonMenuItem("Show
          simulation time");
        group.add(showTimeItem);
        menuSimulation.add(showTimeItem);

        hideTimeItem = new JRadioButtonMenuItem("Hide
          simulation time");
        group.add(hideTimeItem);
        menuSimulation.add(hideTimeItem);
        //a group of check box menu items
        menuSimulation.addSeparator();
        showInfoItem = new JCheckBoxMenuItem("Show
          information");
        menuSimulation.add(showInfoItem);

        setActionListeners();
    }

    private void setActionListeners() {
        startItem.addActionListener(a -> controller.
          setStart());
        stopItem.addActionListener(a -> controller.
          setStop());
        showInfoItem.addActionListener(a -> controller.
          setShowInfo(showInfoItem.getState()));
        showTimeItem.addActionListener(a -> controller.
          setShowInfo(true));
        hideTimeItem.addActionListener(a -> controller.
          setShowTime(false));
    }

    public JMenuBar getMenuBar() {
        return menuBar;
    }

    public void setController(SettingsController c) {
        controller = c;
    }

    @Override
    public void OnSimulationStart() {
        log.debug("o/");
        startItem.setEnabled(false);
        stopItem.setEnabled(true);
    }

    @Override
    public void OnSimulationStop() {
        log.debug("o/");
```

```
        stopItem.setEnabled(false);
        startItem.setEnabled(true);
    }

    @Override
    public void OnShowInfo() {
        showInfoItem.setState(true);
    }

    @Override
    public void OnHideInfo() {
        showInfoItem.setState(false);
    }

    @Override
    public void OnShowTime() {
        showTimeItem.setSelected(true);
    }

    @Override
    public void OnHideTime() {
        hideTimeItem.setSelected(true);
    }

}
```

ResourceId.java

```
package com.xotonic.lab.sit;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.net.URISyntaxException;
import java.net.URL;

public enum ResourceId {
    DEFAULT("default.png"),
    CAR("car.png"),
    BIKE("bike.png");

    static Logger log = LogManager.getLogger(ResourceId.
      class.getName());
    private String resourcePath;
    private BufferedImage image;

    ResourceId(String resourcePath) {
        this.resourcePath = resourcePath;
        image = loadResource(resourcePath);
```

```
    }

    public String getResourcePath() {
        return resourcePath;
    }

    public BufferedImage getImage() {
        return image;
    }

    private BufferedImage loadResource(String
      resourcePath) {
        Logger log = LogManager.getLogger(ResourceId.
          class.getName());
        try {
            URL url = getClass().getClassLoader().
              getResource(resourcePath);
            BufferedImage image;
            log.debug("Loading resource '{}' with path
              '{}'", name(), resourcePath);
            image = ImageIO.read(new File((url.toURI().
              getPath())));
            return image;
        } catch (IOException | URISyntaxException ex) {
            ex.printStackTrace();
            return new BufferedImage(64, 64,
              BufferedImage.TYPE_INT_ARGB);
        }
    }
}
```

ToolBarView.java

```
package com.xotonic.lab.sit;

import com.xotonic.lab.sit.settings.SettingsController;
import com.xotonic.lab.sit.settings.SettingsView;

import javax.swing.*;


public class ToolBarView implements SettingsView {
    private SettingsController controller;

    private JButton startStopButton;
    private JButton infoButton;
    private JButton timeButton;

    private boolean started, isShowTime, isShowInfo;

    public ToolBarView(JButton startStopButton, JButton
      infoButton, JButton timeButton) {
        this.startStopButton = startStopButton;
        this.infoButton = infoButton;
```

```
        this.timeButton = timeButton;
        setListeners();
    }

    private void setListeners() {
        startStopButton.addActionListener(a -> {
            if (started) controller.setStop();
            else controller.setStart();
        });
        infoButton.addActionListener(a -> {
            controller.setShowInfo(!isShowInfo);
        });
        timeButton.addActionListener(a -> {
            controller.setShowTime(!isShowTime);
        });

    }

    public void setController(SettingsController c) {
        controller = c;
    }

    @Override
    public void OnSimulationStart() {
        startStopButton.setText("Stop");
        started = true;
    }

    @Override
    public void OnSimulationStop() {
        startStopButton.setText("Start");
        started = false;

    }

    @Override
    public void OnShowInfo() {
        infoButton.setText("Hide info");
        isShowInfo = true;
    }

    @Override
    public void OnHideInfo() {
        infoButton.setText("Show info");
        isShowInfo = false;

    }

    @Override
    public void OnShowTime() {
        timeButton.setText("Hide time");
        isShowTime = true;
```

```
    }

    @Override
    public void OnHideTime() {
        timeButton.setText("Show time");
        isShowTime = false;
    }
}
```

SimulationTimer.java

```
package com.xotonic.lab.sit;

import com.xotonic.lab.sit.vehicle.Behavior;
import com.xotonic.lab.sit.vehicle.Habitat;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

import javax.swing.*;

/**
 * Created by xotonic on 01.10.2016.
 */
public class SimulationTimer {

    private static Logger log = LogManager.getLogger(Form
      .class.getName());

    private Timer timer;
    private Habitat target;
    private boolean started = false;
    private int delay = 30;
    private long simulationTime = 0;
    private long simulationStartTime = -1;
    public SimulationTimer() {
        timer = new Timer(delay, null);
    }

    public long getSimulationTime() {
        return simulationTime;
    }

    public boolean isStarted() {
        return started;
    }

    public Behavior getTarget() {
        return target;
    }

    public void setTarget(Habitat target) {
        this.target = target;
        timer.addActionListener(e -> {
            if (simulationStartTime == -1)
```

```
                simulationStartTime = System.
                    currentTimeMillis();
            simulationTime = System.currentTimeMillis() -
                simulationStartTime;
            this.target.update(simulationTime);
        });
    }

    public int getDelay() {
        return delay;
    }

    public void setDelay(int delay) {
        log.debug("Set delay {} ms", delay);
        this.delay = delay;
        timer.setDelay(delay);
    }

    public void start() {
        log.debug("Start");
        if (!started) {
            target.start();
            timer.start();
            started = true;
        } else log.warn("Already started");
    }

    public void pause() {
        log.debug("Pause");
        if (started) {
            timer.stop();
            target.stop();
            started = false;
        } else log.warn("Not started, but trying pause");
    }

    public void reset() {
        log.debug("Reset");
        if (started) {
            timer.stop();
            target.reset();
            target.stop();
            simulationTime = 0;
            simulationStartTime = -1;
            started = false;
        } else log.warn("Already stopped");
    }

}
```

Statistic.java

```
package com.xotonic.lab.sit;
```

```
import java.io.Serializable;


public class Statistic implements Serializable {

    private int totalCarsCreated;
    private int totalBikesCreated;
    private long totalTime;

    public int getTotalCarsCreated() {
        return totalCarsCreated;
    }

    public void setTotalCarsCreated(int totalCarsCreated)
        {
        this.totalCarsCreated = totalCarsCreated;
    }

    public int getTotalBikesCreated() {
        return totalBikesCreated;
    }

    public void setTotalBikesCreated(int
       totalBikesCreated) {
        this.totalBikesCreated = totalBikesCreated;
    }

    public long getTotalTime() {
        return totalTime;
    }

    public void setTotalTime(long totalTime) {
        this.totalTime = totalTime;
    }
}
```

Form.java

```
package com.xotonic.lab.sit;

import com.xotonic.lab.sit.settings.SettingsController;
import com.xotonic.lab.sit.settings.SettingsModel;
import com.xotonic.lab.sit.settings.SettingsView;
import com.xotonic.lab.sit.vehicle.*;
import com.xotonic.lab.sit.vehicle.Painter;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

import javax.swing.*;
import javax.swing.plaf.nimbus.NimbusLookAndFeel;
import java.awt.*;
import java.awt.event.ComponentEvent;
import java.awt.event.ComponentListener;
```

```
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;

public class Form extends JDialog implements KeyListener,
    SettingsView {

    static Logger log = LogManager.getLogger(Form.class.
      getName());


    private JPanel contentPane;
    private JPanel drawPanel;
    private JPanel propertiesPanel;
    private JButton sideBarStart;
    private JButton sideBarStop;
    private JCheckBox sideBarInfoToggle;
    private JRadioButton sideBarTimeShow;
    private JRadioButton sideBarTimeHide;
    private JButton toolbarStartStop;
    private JButton toolbarInfo;
    private JButton tollbarTime;
    private JPanel factoriesSettingsPanel;

    private DrawPanel drawer;
    private Habitat habitat = new SimpleHabitat();
    private TimedLuckyFactory carFactory = new CarFactory
      (habitat);
    private TimedLuckyFactory bikeFactory = new
      BikeFactory(habitat);
    private Painter painter;
    private SimulationTimer timer;
    private SettingsModel settingsModel;
    private SettingsController settingsController;
    private MenuView menuView;
    private ToolBarView toolBarView;
    private SideBarView sideBarView;

    public Form() {
        setContentPane(contentPane);
        setModal(true);
        addKeyListener(this);

        habitat.getPainters().add(painter);

        timer = new SimulationTimer();
        timer.setTarget(habitat);

        settingsModel = new SettingsModel();

        settingsController = new SettingsController();
        settingsController.setModel(settingsModel);

        menuView = new MenuView();
```

```
        menuView.setController(settingsController);
        settingsController.addView(menuView);
        setJMenuBar(menuView.getMenuBar());

        toolBarView = new ToolBarView(toolbarStartStop,
          toolbarInfo, tollbarTime);
        toolBarView.setController(settingsController);
        settingsController.addView(toolBarView);

        sideBarView = new SideBarView(
                sideBarStart,
                sideBarStop,
                sideBarInfoToggle,
                sideBarTimeShow,
                sideBarTimeHide
        );

        sideBarView.setController(settingsController);
        settingsController.addView(sideBarView);

        settingsController.addView(this);

    }

    public static void main(String[] args) {

        log.debug("Program start");
        setLookAndFeel();

        final Form dialog = new Form();
        dialog.pack();
        dialog.setVisible(true);
        log.debug("Program exit");
        System.exit(0);
    }

    private static void setLookAndFeel() {
        UIManager.put("nimbusBase", new Color(49, 247,
          255));
        UIManager.put("nimbusBlueGrey", new Color(49, 51,
           53));
        UIManager.put("control", new Color(49, 51, 53));
        UIManager.put("nimbusFocus", new Color(53, 255,
          253));
        UIManager.put("text", new Color(189, 189, 189));
        try {
            UIManager.setLookAndFeel(new
              NimbusLookAndFeel());
        } catch (UnsupportedLookAndFeelException e) {
            e.printStackTrace();
        }
    }
```

```
private void createUIComponents() {
    DrawPanel panel = new DrawPanel();
    drawPanel = panel;
    painter = panel;
    drawer = panel;

    panel.addComponentListener(new ComponentListener
       () {
         public void componentResized(ComponentEvent e
            ) {
             habitat.setWorldWidth(drawer.getWidth());
             habitat.setWorldHeight(drawer.getHeight()
                );
         }

         @Override
         public void componentMoved(ComponentEvent e)
            {
         }

         @Override
         public void componentShown(ComponentEvent e)
            {
         }

         @Override
         public void componentHidden(ComponentEvent e)
             {
         }
    });
}

@Override
public void keyTyped(KeyEvent e) {

}

@Override
public void keyPressed(KeyEvent e) {
    switch (e.getKeyChar()) {
         case 'b':
             startSimulation();
             break;
         case 'e': {
             stopSimulation();
         }
         break;
         case 't': {
             toggleShowTime();
         }
         break;
```

```
    }
}

private void startSimulation() {
    timer.start();
}

private void toggleShowTime() {
    drawer.setShowTime(!drawer.isShowTime());
}

@Override
public void keyReleased(KeyEvent e) {

}


@Override
public void OnSimulationStart() {
    startSimulation();
}

@Override
public void OnSimulationStop() {

    stopSimulation();
}

private void stopSimulation() {
    reportStatistic();
    timer.reset();
}

private void reportStatistic() {
    Statistic statistic = new Statistic();
    statistic.setTotalCarsCreated(carFactory.
       getTotalCreated());
    statistic.setTotalBikesCreated(bikeFactory.
       getTotalCreated());
    statistic.setTotalTime(timer.getSimulationTime())
       ;
    drawer.setStatistic(statistic);
}

@Override
public void OnShowInfo() {

}

@Override
public void OnHideInfo() {

}
```

```
    @Override
    public void OnShowTime() {
        drawer.setShowTime(true);

    }

    @Override
    public void OnHideTime() {
        drawer.setShowTime(false);

    }
}
```

StatisticDialog.java

```
package com.xotonic.lab.sit;

/**
 * Created by xotonic on 07.11.2016.
 */
public class StatisticDialog {
}
```

DrawPanel.java

```
package com.xotonic.lab.sit;


import com.xotonic.lab.sit.vehicle.Painter;
import com.xotonic.lab.sit.vehicle.Vehicle;

import javax.swing.*;
import java.awt.*;
import java.awt.image.BufferedImage;
import java.util.Arrays;
import java.util.Collection;
import java.util.Optional;


class DrawPanel extends JPanel implements Painter {
    private Collection<Vehicle> vehicles;
    private long lastUpdatedTime = 0;
    private boolean started = false;
    private boolean stopped = false;
    private boolean isShowTime = true;
    private Statistic statistic;

    DrawPanel() {
        super();

    }

    boolean isShowTime() {
        return isShowTime;
```

```
}

void setShowTime(boolean showTime) {
    isShowTime = showTime;
}

public void setStatistic(final Statistic statistic) {
    this.statistic = statistic;
}


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

    ((Graphics2D) g).setRenderingHint(
            RenderingHints.KEY_TEXT_ANTIALIASING,
            RenderingHints.VALUE_TEXT_ANTIALIAS_ON);

    if (vehicles != null)
        drawVehicles(g);

    if (isShowTime) {

        drawLinesTopLeft(g,
                String.format("Time : %d",
                  lastUpdatedTime),
                started ? "Simulation start" : "
                  Simulation stop"
        );
    }


    if (stopped) {

        assert statistic != null;

        drawLinesCenter(g,
                "Simulation stopped",
                String.format("Total cars : %d",
                  statistic.getTotalCarsCreated()),
                String.format("Total bikes: %d",
                  statistic.getTotalBikesCreated()),
                String.format("Total time : %d",
                  statistic.getTotalTime())
        );
    }

    g.drawRoundRect(0, 0, getWidth() - 1, getHeight()
        - 1, 20, 20);
}

private void drawVehicles(Graphics g) {
```

```
    for (Vehicle v : vehicles) {

        BufferedImage img = v.getResourceId().
          getImage();
        g.drawImage(img, Math.round(v.getX()), Math.
          round(v.getY()), this);
    }
}


@Override
public void start() {

    started = true;
    stopped = false;
}

@Override
public void update(long timeMillis) {
    log.trace("DrawPanel update");
    lastUpdatedTime = timeMillis;
    repaint();
}

@Override
public void stop() {
    started = false;
    stopped = true;
    repaint();
}

@Override
public void onRepaint(Collection<Vehicle> vehicles) {
    if (this.vehicles == null) {
        this.vehicles = vehicles;
    }

}

private void drawLinesCenter(Graphics g, String...
  lines) {
    Color temp = g.getColor();
    Font font = new Font("Consolas", 1, 36);
    g.setFont(font);
    FontMetrics metrics = g.getFontMetrics(font);
    Optional<String> longest = Arrays.stream(lines).
      max((l1, l2) -> l1.length() > l2.length() ? 1 :
       -1);
    if (longest.isPresent()) {
        boolean isOdd = false;
        int currentX = getWidth() / 2 - metrics.
          stringWidth(longest.get()) / 2;
```

```
            int currentY = getHeight() / 2 - lines.length
                * metrics.getHeight() / 2;
            for (String s : lines) {
                g.setColor(isOdd ? new Color(135, 255,
                   52) : new Color(0, 167, 255));
                isOdd = !isOdd;
                g.drawString(s, currentX, currentY);
                currentY += metrics.getHeight();
            }
        }

        g.setColor(temp);
    }

    private void drawLinesTopLeft(Graphics g, String...
       lines) {
        Color temp = g.getColor();
        Font font = new Font("Arial", 1, 12);
        g.setFont(font);
        FontMetrics metrics = g.getFontMetrics(g.getFont
           ());
        int currentX = 10;
        int currentY = 20;
        for (String s : lines) {
            g.drawString(s, currentX, currentY);
            currentY += metrics.getHeight();
        }

        g.setColor(temp);
    }
}
```

FactorySettings.java

```
package com.xotonic.lab.sit.settings;


import java.io.Serializable;

public class FactorySettings implements Serializable {
    float bornPeriod;
    float bornChance;
}
```

SettingsModel.java

```
package com.xotonic.lab.sit.settings;


import java.io.Serializable;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;
```

```
public class SettingsModel implements Serializable {
    public SimulationState simulationState;
    boolean showInfo;
    boolean showTime;
    Map<FactoryType, FactorySettings> factoriesSettings =
        new HashMap<>();

    {
        Arrays.stream(FactoryType.values())
                .forEach(type -> factoriesSettings.put(
                  type, new FactorySettings()));
    }

    enum SimulationState {start, stop, pause}

    enum FactoryType {car, bike;}
}
```

SettingsController.java

```
package com.xotonic.lab.sit.settings;


import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

import java.util.ArrayList;
import java.util.Collection;

public class SettingsController {
    Logger log = LogManager.getLogger(SettingsController.
      class.getName());

    private Collection<SettingsView> views = new
      ArrayList<>();
    private SettingsModel model;

    public void setModel(SettingsModel model) {
        log.debug("o/");

        this.model = model;

        updateFull();
    }

    private void updateFull() {
        log.debug("o/");

        updateShowInfo();
        updateShowTime();
        updateSimulationState();
    }
```

```
private void updateSimulationState() {
    views.forEach(model.simulationState ==
      SettingsModel.SimulationState.start ?
            SettingsView::OnSimulationStart :
              SettingsView::OnSimulationStop);
}

private void updateShowTime() {
    log.debug("o/");

    views.forEach(model.showTime ? SettingsView::
      OnShowTime : SettingsView::OnHideTime);
}

private void updateShowInfo() {
    log.debug("o/");

    views.forEach(model.showInfo ? SettingsView::
      OnShowInfo : SettingsView::OnHideInfo);
}

public void addView(SettingsView view) {
    log.debug("o/");

    views.add(view);
    updateFull();
}

public void setStart() {
    log.debug("o/");

    model.simulationState = SettingsModel.
      SimulationState.start;
    updateSimulationState();
}

public void setStop() {
    log.debug("o/");

    model.simulationState = SettingsModel.
      SimulationState.stop;
    updateSimulationState();
}

public void setShowTime(boolean show) {
    log.debug("o/ show = {}", show);

    model.showTime = show;
    updateShowTime();
}

public void setShowInfo(boolean show) {
    log.debug("o/ show = {}", show);
```

```
        model.showInfo = show;
        updateShowInfo();
    }

    public void setFactoryPeriod(SettingsModel.
      FactoryType type, float cooldown) {
        model.factoriesSettings.get(type).bornPeriod =
          cooldown;
    }

    public void setFactoryChance(SettingsModel.
      FactoryType type, float chance) {
        model.factoriesSettings.get(type).bornChance =
          chance;
    }

}
```

SettingsView.java

```
package com.xotonic.lab.sit.settings;


import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public interface SettingsView {
    Logger log = LogManager.getLogger(SettingsView.class.
      getName());

    void OnSimulationStart();

    void OnSimulationStop();

    void OnShowInfo();

    void OnHideInfo();

    void OnShowTime();

    void OnHideTime();
}
```

Car.java

```
/*
 * To change this license header, choose License Headers
   in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package com.xotonic.lab.sit.vehicle;

import com.xotonic.lab.sit.ResourceId;
```

```
/**
 * @author User
 */
public class Car extends Vehicle {


    protected ResourceId resourceId = ResourceId.CAR;

    public Car(String id) {
        super(id);
    }

    @Override
    public ResourceId getResourceId() {
        return resourceId;
    }

    @Override
    public void update(long timeMillis) {
    }

    @Override
    public void start() {
        super.start();
    }

    @Override
    public void stop() {
        super.stop();
    }

}
```

Behavior.java

```
/*
 * To change this license header, choose License Headers
   in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package com.xotonic.lab.sit.vehicle;


import com.xotonic.lab.sit.Form;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

/**
 * @author User
 */
public interface Behavior {
```

```
    Logger log = LogManager.getLogger(Form.class.getName
      ());

    void start();

    void update(long timeMillis);

    void stop();
}
```

Bike.java

```
/*
 * To change this license header, choose License Headers
   in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package com.xotonic.lab.sit.vehicle;

import com.xotonic.lab.sit.ResourceId;

/**
 * @author User
 */
public class Bike extends Vehicle {


    protected ResourceId resourceId = ResourceId.BIKE;

    public Bike(String id) {
        super(id);
    }

    @Override
    public ResourceId getResourceId() {
        return resourceId;
    }

    @Override
    public void update(long timeMillis) {
    }

    @Override
    public void start() {
        super.start();
    }

    @Override
    public void stop() {
        super.stop();
    }

}
```

BikeFactory.java

```
package com.xotonic.lab.sit.vehicle;

import java.util.Random;


public class BikeFactory extends TimedLuckyFactory {

    private Random r = new Random();

    public BikeFactory(Habitat habitat) {
        super(habitat);
        cooldown = 200;
        setCreateChance(0.2f);
    }

    @Override
    public Vehicle create() {
        Bike bike = new Bike(Bike.class.getSimpleName() +
            "_" + getNextId());

        bike.setX(r.nextFloat() * habitat.getWorldWidth()
           );
        bike.setY(r.nextFloat() * habitat.getWorldHeight
           ());
        log.debug("Created car {}", bike.getId());
        return bike;
    }

}
```

SimpleHabitat.java

```
/*
 * To change this license header, choose License Headers
   in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package com.xotonic.lab.sit.vehicle;

import java.util.ArrayList;
import java.util.Collection;

public class SimpleHabitat extends Habitat {

    private Collection<Vehicle> vehicles;
    private Collection<Factory> factories;
    private Collection<Painter> painters;

    public SimpleHabitat() {
        vehicles = new ArrayList<>();
        factories = new ArrayList<>();
        painters = new ArrayList<>();
```

```
}

@Override
public void update(long timeMillis) {
    log.trace("SimpleHabitat update ...");

    for (Factory f : factories)
        f.update(timeMillis);

    for (Vehicle v : vehicles)
        if (v.isStarted())
            v.update(timeMillis);
        else
            v.start();

    for (Painter p : painters) {
        p.update(timeMillis);
        p.onRepaint(vehicles);
    }

}

@Override
public void start() {
    log.debug("SimpleHabitat start ...");

    factories.forEach(Behavior::start);

    vehicles.forEach(Vehicle::start);

    painters.forEach(Behavior::start);
}

@Override
public void stop() {
    log.debug("SimpleHabitat stop ...");

    factories.forEach(Behavior::stop);

    vehicles.forEach(Vehicle::stop);

    painters.forEach(Behavior::stop);
}

public Collection<Vehicle> getVehicles() {
    return vehicles;
}

public Collection<Factory> getFactories() {
    return factories;
}

public Collection<Painter> getPainters() {
```

```
        return painters;
    }

    @Override
    public void reset() {
        log.debug("Reset");
        vehicles.clear();
    }

}
```

TimedLuckyFactory.java

```
/*
 * To change this license header, choose License Headers
   in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package com.xotonic.lab.sit.vehicle;

import java.util.Random;

public abstract class TimedLuckyFactory extends Factory {

    private static int id = 1;


    int cooldown = 1000;
    private float createChance = 0.5f;
    private Random r = new Random();
    private long time;
    private long prevTimeMillis = 0;
    private int totalCreated = 0;

    TimedLuckyFactory(Habitat habitat) {
        super(habitat);
    }

    public int getTotalCreated() {
        return totalCreated;
    }

    public float getCreateChance() {
        return createChance;
    }

    public void setCreateChance(float createChance) {
        this.createChance = createChance;
    }

    @Override
    public void stop() {
        totalCreated = 0;
```

```
        time = 0;
    }

    @Override
    public void start() {
        time = 0;
        prevTimeMillis = 0;
    }

    @Override
    public void update(long timeMillis) {
        if (createChance > 1f | createChance < 0f)
            log.error("Chance value is not in range
              [0.0;1.0] (now {})", createChance);

        time += timeMillis - prevTimeMillis;
        prevTimeMillis = timeMillis;
        if (time >= cooldown) {
            time -= cooldown;
            if (r.nextFloat() < createChance)
                build();
        }
    }

    protected String getNextId() {
        return String.format("%d", id++);
    }

    protected long getCooldown() {
        return time;
    }

    public void setCooldown(int cooldown) {
        this.cooldown = cooldown;
    }
    @Override
    public void build() {
        Vehicle v = create();
        habitat.getVehicles().add(v);
        totalCreated++;
    }
}
```

BasicBehavior.java

```
/*
 * To change this license header, choose License Headers
   in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package com.xotonic.lab.sit.vehicle;
```

```
/**
 * @author User
 */
public abstract class BasicBehavior implements Behavior {

    private String id = getClass().getSimpleName();

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

}
```

Factory.java

```
/*
 * To change this license header, choose License Headers
   in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package com.xotonic.lab.sit.vehicle;

/**
 * @author User
 */
public abstract class Factory extends BasicBehavior {


    protected Habitat habitat;

    public Factory(Habitat habitat) {
        this.habitat = habitat;
        habitat.getFactories().add(this);
    }

    abstract public Vehicle create();

    abstract public void build();

}
```

Vehicle.java

```
/*
 * To change this license header, choose License Headers
   in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
```

```
package com.xotonic.lab.sit.vehicle;

import com.xotonic.lab.sit.ResourceId;

/**
 * @author User
 */
public abstract class Vehicle extends BasicBehavior {

    protected ResourceId resourceId = ResourceId.DEFAULT;
    private float x = 0f;
    private float y = 0f;
    private boolean isStarted = false;

    public Vehicle(String id, float x, float y) {
        this(id);
        this.x = x;
        this.y = y;
    }

    public Vehicle(String id) {
        setId(id);
    }

    public ResourceId getResourceId() {
        return resourceId;
    }

    public float getX() {
        return x;
    }

    public void setX(float x) {
        this.x = x;
    }

    public float getY() {
        return y;
    }

    public void setY(float y) {
        this.y = y;
    }

    @Override
    public void stop() {
        isStarted = false;
    }

    @Override
    public void start() {
        isStarted = true;
    }
```

```
    public boolean isStarted() {
        return isStarted;
    }
}
```

CarFactory.java

```
package com.xotonic.lab.sit.vehicle;

import java.util.Random;


public class CarFactory extends TimedLuckyFactory {

    private Random r = new Random();

    public CarFactory(Habitat habitat) {
        super(habitat);
        cooldown = 100;
        setCreateChance(0.2f);
    }

    @Override
    public Vehicle create() {
        Car car = new Car(Car.class.getSimpleName() + "-"
            + getNextId());

        car.setX(r.nextFloat() * habitat.getWorldWidth())
           ;
        car.setY(r.nextFloat() * habitat.getWorldHeight()
           );
        log.debug("Created car {}", car.getId());
        return car;
    }

}
```

Habitat.java

```
package com.xotonic.lab.sit.vehicle;

import java.util.Collection;

/**
 * Created by xotonic on 16.09.2016.
 */
public abstract class Habitat extends BasicBehavior {
    private int worldWidth;
    private int worldHeight;

    public abstract Collection<Vehicle> getVehicles();

    public abstract Collection<Factory> getFactories();
```

```
    public abstract Collection<Painter> getPainters();

    public abstract void reset();

    public int getWorldWidth() {
        return worldWidth;
    }

    public void setWorldWidth(int worldWidth) {
        this.worldWidth = worldWidth;
    }

    public int getWorldHeight() {
        return worldHeight;
    }

    public void setWorldHeight(int worldHeight) {
        this.worldHeight = worldHeight;
    }
}
```

Painter.java

```
/*
 * To change this license header, choose License Headers
   in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package com.xotonic.lab.sit.vehicle;

import java.util.Collection;

/**
 * @author User
 */
public interface Painter extends Behavior {

    void onRepaint(Collection<Vehicle> vehicles);
}
```

Вывод

Произошло ознакомление с особенностями технологии Java и была изучена часть синтаксиса языка Java. Была разработана программа для упрощенной имитации поведения объектов.