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

Model.java

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


import java.io.Serializable;

public interface Model extends Serializable {
}
```

View.java

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


public interface View<ControllerType extends Controller>
  {

    void setController(ControllerType controller);
}
```

SettingsModel.java

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



public class SettingsModel implements Model {

    public SimulationState simulationState;
    public boolean showInfo;
    public boolean showTime;

    enum SimulationState {start, stop, pause}
}
```

FactorySettingsView.java

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


import javax.swing.*;

public interface FactorySettingsView
        <RootComponent extends JComponent,
         SettingsControllerType extends
           FactorySettingsController>
```

```java
        extends
            HasUI<RootComponent>,
            View<SettingsControllerType>
{
    void OnBornPeriodChanged(int bornPeriod);
    void OnBornChanceChanged(float bornChance);

    FactoryType getFactoryType();

    void setFactoryType(FactoryType type);
}
```

SettingsController.java

```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 implements Controller<
  SettingsModel, SettingsView>
{

    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();
    }
```

```
}
```

Controller.java

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


public interface Controller<ModelType extends Model,
  ViewType extends View> {
    void setModel(ModelType model);
    void addView(ViewType view);
}
```

SettingsView.java

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


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

import javax.swing.*;

public interface SettingsView<RootComponent extends
  JComponent,
                              SettingsControllerType
                                extends
                                SettingsController>

        extends HasUI<RootComponent>,
                View<SettingsControllerType>
{

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

    void OnSimulationStart();
    void OnSimulationStop();

    void OnShowInfo();
    void OnHideInfo();

    void OnShowTime();
    void OnHideTime();
}
```

FactoryType.java

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


public enum FactoryType {
    car(new FactoryModel(100, 0.2f)),
    bike(new FactoryModel(200, 0.2f));
    private FactoryModel defaultModel;

    FactoryType(FactoryModel defaultModel) {
        this.defaultModel = defaultModel;
    }

    public FactoryModel getDefaultModel() {
        return defaultModel;
    }
}
```

FactorySettingsModel.java

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


import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;

public class FactorySettingsModel implements Model {

    public Map<FactoryType, FactoryModel>
      factoriesSettings = new HashMap<>();

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

}
```

HasUI.java

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

import javax.swing.*;

public interface HasUI<RootComponent extends JComponent>
   {
    void initializeUI();
    RootComponent getRootComponent();
}
```

FactorySettingsController.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 FactorySettingsController
        implements Controller<FactorySettingsModel,
          FactorySettingsView>
{
    private static Logger log = LogManager.getLogger(
      FactorySettingsController.class.getName());


    private FactorySettingsModel model;
    private Collection<FactorySettingsView> views = new
      ArrayList<>();

    @Override
    public void setModel(FactorySettingsModel model) {
        this.model = model;
        updateFullDefault();
    }

    private void updateFullDefault()
    {
        views.forEach(v -> {
            v.OnBornPeriodChanged(
                    v.getFactoryType().getDefaultModel().
                      bornPeriod);
            v.OnBornChanceChanged(
                    v.getFactoryType().getDefaultModel().
                      bornChance);
        });
    }

    @Override
    public void addView(FactorySettingsView view) {
        views.add(view);
        updateFullDefault();
    }

    public void setBornChance(FactorySettingsView sender,
       float value)
    {
        log.debug("sender: {}; type: {} value : {}",
          sender.hashCode(), sender.getFactoryType().name
          (), value);

        model.factoriesSettings.get(sender.getFactoryType
          ()).bornChance = value;
        updateBornChance(sender);
    }

    public void setBornPeriod(FactorySettingsView sender,
       int value)
    {
        log.debug("sender: {}; type: {} value : {}",
          sender.hashCode(), sender.getFactoryType().name
          (), value);
        model.factoriesSettings.get(sender.getFactoryType
          ()).bornPeriod = value;
        updateBornPeriod(sender);
    }


    private void updateBornPeriod(FactorySettingsView
      sender) {
            views.stream()
                    .filter(v -> v != sender & v.
                      getFactoryType() == sender.
                      getFactoryType())
                    .forEach(v -> v.OnBornPeriodChanged(
                            model.factoriesSettings.get(v
                              .getFactoryType()).
                              bornPeriod));

    }

    private void updateBornChance(FactorySettingsView
      sender) {
            views.stream()
                    .filter(v -> v != sender & v.
                      getFactoryType() == sender.
                      getFactoryType())
                    .forEach(v -> v.OnBornChanceChanged(
                            model.factoriesSettings.get(v
                              .getFactoryType()).
                              bornChance));

    }


}
```

FactoryModel.java

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


import java.io.Serializable;

public class FactoryModel implements Serializable {

    int bornPeriod;
    float bornChance;

    public FactoryModel() {}
    public FactoryModel(int period, float chance)
    {
        bornPeriod = period;
        bornChance = chance;
    }
}
```

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

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

    Logger log = LogManager.getLogger(Behavior.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.ui.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.ui.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);
}
```

SideBarView.java

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

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

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


public class SideBarView implements SettingsView<JPanel,
   SettingsController>{

    private SettingsController controller;



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

    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);
    }


    @Override
    public void initializeUI() {

        GridBagConstraints gbc;
        JPanel factoriesSettingsPanel;

        propertiesPanel = new JPanel();
        propertiesPanel.setLayout(new GridBagLayout());

        propertiesPanel.setBorder(BorderFactory.
          createTitledBorder("Properties"));

        final JPanel panel3 = new JPanel();
        panel3.setLayout(new GridBagLayout());
        gbc = new GridBagConstraints();
        gbc.gridx = 0;
        gbc.gridy = 0;
        gbc.weightx = 1.0;
        gbc.fill = GridBagConstraints.BOTH;
        propertiesPanel.add(panel3, gbc);
        panel3.setBorder(BorderFactory.createTitledBorder
          ("Simulation control"));

        sideBarStart = new JButton();
        sideBarStart.setText("Start");
        gbc = new GridBagConstraints();
        gbc.gridx = 0;
```

```
gbc.gridy = 0;
gbc.weightx = 1.0;
gbc.weighty = 1.0;
panel3.add(sideBarStart, gbc);
sideBarStop = new JButton();
sideBarStop.setText("Stop");
gbc = new GridBagConstraints();
gbc.gridx = 1;
gbc.gridy = 0;
gbc.weightx = 1.0;
gbc.weighty = 1.0;
panel3.add(sideBarStop, gbc);
final JPanel panel4 = new JPanel();
panel4.setLayout(new GridBagLayout());
gbc = new GridBagConstraints();
gbc.gridx = 0;
gbc.gridy = 2;
gbc.weightx = 1.0;
gbc.fill = GridBagConstraints.BOTH;
propertiesPanel.add(panel4, gbc);
panel4.setBorder(BorderFactory.createTitledBorder
  ("Simulation time"));

sideBarTimeShow = new JRadioButton();
sideBarTimeShow.setText("Show");
gbc = new GridBagConstraints();
gbc.gridx = 0;
gbc.gridy = 0;
gbc.weightx = 1.0;
gbc.weighty = 1.0;
gbc.anchor = GridBagConstraints.WEST;
panel4.add(sideBarTimeShow, gbc);

sideBarTimeHide = new JRadioButton();
sideBarTimeHide.setText("Hide");
gbc = new GridBagConstraints();
gbc.gridx = 1;
gbc.gridy = 0;
gbc.weightx = 1.0;
gbc.weighty = 1.0;
gbc.anchor = GridBagConstraints.WEST;
panel4.add(sideBarTimeHide, gbc);

final JPanel panel5 = new JPanel();
panel5.setLayout(new GridBagLayout());
gbc = new GridBagConstraints();
gbc.gridx = 0;
gbc.gridy = 1;
gbc.weightx = 1.0;
gbc.fill = GridBagConstraints.BOTH;
propertiesPanel.add(panel5, gbc);
```

```
        panel5.setBorder(BorderFactory.createTitledBorder
          ("Information"));
        sideBarInfoToggle = new JCheckBox();
        sideBarInfoToggle.setText("Show");
        gbc = new GridBagConstraints();
        gbc.gridx = 0;
        gbc.gridy = 0;
        gbc.weightx = 1.0;
        gbc.weighty = 1.0;
        gbc.anchor = GridBagConstraints.WEST;
        panel5.add(sideBarInfoToggle, gbc);

        factoriesSettingsPanel = new JPanel();
        factoriesSettingsPanel.setLayout(new
          GridBagLayout());
        gbc = new GridBagConstraints();
        gbc.gridx = 0;
        gbc.gridy = 3;
        gbc.weightx = 1.0;
        gbc.fill = GridBagConstraints.BOTH;
        propertiesPanel.add(factoriesSettingsPanel, gbc);

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

        setListeners();
    }

    @Override
    public JPanel getRootComponent() {
        return propertiesPanel;
    }


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

    public void addFactorySettingsView(FactoryOptionsView
       panel)
    {
        GridBagConstraints gbc = new GridBagConstraints()
          ;
        gbc.gridx = 0;
        gbc.gridy = GridBagConstraints.RELATIVE;
        gbc.anchor = GridBagConstraints.NORTH;
        gbc.fill = GridBagConstraints.HORIZONTAL;
        propertiesPanel.add(panel.getRootComponent(), gbc
          );
    }
}
```

MenuView.java

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


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<JMenuBar,
  SettingsController> {

    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;


    @Override
    public void initializeUI() {
        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");
        startItem.setAccelerator(KeyStroke.getKeyStroke('
          b'));
        menuSimulation.add(startItem);
        stopItem = new JMenuItem("Stop");
        stopItem.setAccelerator(KeyStroke.getKeyStroke('e
          '));
        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");
        hideTimeItem.setAccelerator(KeyStroke.
          getKeyStroke('t'));
        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));
    }



    @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);
    }

    @Override
    public JMenuBar getRootComponent() {
        return menuBar;
    }


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

ResourceId.java

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

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

import javax.imageio.ImageIO;
import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.IOException;

public enum ResourceId {
```

```
DEFAULT("default.png"),
CAR("car.png"),
BIKE("bike.png");

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());
    log.debug("Loading resource '{}' with path '{}'",
       name(), resourcePath);
    try {
        BufferedImage image;
        image = ImageIO.read(getClass().getResource(
          resourcePath));
        return image;
    } catch (IOException ex) {
        ex.printStackTrace();
        return getFailedLoadingImage();
    }
    catch (Exception e)
    {
        log.error("Exception during loading resource
          ", e);
        return getFailedLoadingImage();
    }
}


private BufferedImage getFailedLoadingImage() {
    Logger log = LogManager.getLogger(ResourceId.
      class.getName());

    log.debug("o/");

    BufferedImage image = new BufferedImage(64, 64,
      BufferedImage.TYPE_INT_ARGB);
```

```
        Graphics2D g = image.createGraphics();
        g.setColor(Color.RED);
        g.drawString("fail " + name(), 5, 20);
        g.drawRect(1, 1,62, 62);
        image.flush();
        return image;
    }


}
```

ToolBarView.java

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

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

import javax.swing.*;


public class ToolBarView implements SettingsView<JToolBar
  , SettingsController> {
    private SettingsController controller;

    private JButton toolbarStartStop;
    private JButton toolbarInfo;
    private JButton toolbarTime;

    private boolean started, isShowTime, isShowInfo;
    private JToolBar toolBar;

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

    }

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

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

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

    }

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

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

    }

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

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

    @Override
    public void initializeUI() {
        toolBar = new JToolBar();
        toolbarStartStop = new JButton();
        toolbarStartStop.setText("Start");
        toolBar.add(toolbarStartStop);
        toolbarInfo = new JButton();
        toolbarInfo.setText("Info");
        toolBar.add(toolbarInfo);
        toolbarTime = new JButton();
        toolbarTime.setText("Time");
        toolBar.add(toolbarTime);

        setListeners();
    }

    @Override
    public JToolBar getRootComponent() {
        return toolBar;
    }
}
```

SimulationTimer.java

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

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.ui;

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;
    }
}
```

FactoryManipulator.java

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

import com.xotonic.lab.sit.settings.
  FactorySettingsController;
import com.xotonic.lab.sit.settings.FactorySettingsView;
import com.xotonic.lab.sit.settings.FactoryType;
import com.xotonic.lab.sit.vehicle.TimedLuckyFactory;

import javax.swing.*;


class FactoryManipulator
        implements FactorySettingsView<JComponent,
          FactorySettingsController> {
    private TimedLuckyFactory factory;
    private FactoryType ftype;

    FactoryManipulator(TimedLuckyFactory factory,
            FactoryType ftype) {
            this.factory = factory;
            this.ftype = ftype;
    }

    @Override
    public void setController(FactorySettingsController
        controller) {
    }

    @Override
    public void initializeUI() {

    }

    @Override
    public JComponent getRootComponent() {
        return null;
    }

    @Override
    public void OnBornPeriodChanged(int bornPeriod) {
        factory.setCooldown(bornPeriod);
    }

    @Override
    public void OnBornChanceChanged(float bornChance) {
        factory.setCreateChance(bornChance);
    }

    @Override
    public FactoryType getFactoryType() {
        return ftype;
    }

    @Override
    public void setFactoryType(FactoryType type) {
    }
}
```

Form.java

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

import com.xotonic.lab.sit.settings.*;
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 JFrame
        implements KeyListener,
                   SettingsView<JPanel,
                     SettingsController>
{

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


    private JPanel contentPane;
    private JPanel drawPanel;

    private Habitat habitat = new SimpleHabitat();
    private TimedLuckyFactory carFactory = new CarFactory
      (habitat);
    private TimedLuckyFactory bikeFactory = new
      BikeFactory(habitat);

    private Painter painter;
    private DrawPanel drawer;
    private SimulationTimer timer;
    private StatisticDialog statisticDialog;

    private SettingsModel settingsModel;
    private SettingsController settingsController;

    private FactorySettingsModel factoriesModel;
    private FactorySettingsController factoriesController
      ;

    private MenuView menuView;
    private ToolBarView toolBarView;
    private SideBarView sideBarView;
    private FactoryOptionsView carsSettingsView;
    private FactoryOptionsView bikesSettingsView;

    /*
    TODO управлениеклавишаминеработает
     */

    public Form() {
        setDefaultCloseOperation(WindowConstants.
          EXIT_ON_CLOSE);
        addKeyListener(this);

        createDrawPanel();

        habitat.getPainters().add(painter);
```

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

statisticDialog = new StatisticDialog(this);
statisticDialog.setOnConfirmListener( () -> timer
  .reset()); // controller.setStop();
statisticDialog.setOnCancelListener(() ->
  settingsController.setStart());
settingsModel = new SettingsModel();
factoriesModel = new FactorySettingsModel();

menuView = new MenuView();
toolBarView = new ToolBarView();
sideBarView = new SideBarView();
carsSettingsView = new FactoryOptionsView(
  FactoryType.car);
bikesSettingsView = new FactoryOptionsView(
  FactoryType.bike);

log.debug("Initializing UI");
menuView.initializeUI();
toolBarView.initializeUI();
sideBarView.initializeUI();
carsSettingsView.initializeUI();
bikesSettingsView.initializeUI();
initializeUI();

log.debug("Initializing settings system");
settingsController = new SettingsController();
settingsController.setModel(settingsModel);
settingsController.addView(menuView);
settingsController.addView(toolBarView);
settingsController.addView(sideBarView);
settingsController.addView(this);

factoriesController = new
  FactorySettingsController();
factoriesController.setModel(factoriesModel);
factoriesController.addView(carsSettingsView);
factoriesController.addView(bikesSettingsView);

menuView.setController(settingsController);
toolBarView.setController(settingsController);
sideBarView.setController(settingsController);
carsSettingsView.setController(
  factoriesController);
bikesSettingsView.setController(
  factoriesController);

FactoryManipulator carFactoryManipulator = new
  FactoryManipulator(carFactory, FactoryType.car)
```

```
            ;
          FactoryManipulator bikeFactoryManipulator = new
            FactoryManipulator(bikeFactory, FactoryType.
            bike);
          factoriesController.addView(carFactoryManipulator
            );
          factoriesController.addView(
            bikeFactoryManipulator);

      }

      public static void main(String[] args) {

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

          SwingUtilities.invokeLater( () ->
          {
              final Form dialog = new Form();
              dialog.pack();
              dialog.setVisible(true);
          });
          log.debug("Program exit");
      }

      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 createDrawPanel() {
          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) {
        log.debug("KEY %s", e.getKeyChar());
        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() {
        Statistic stats =  getStatistic();

        showCanvasStatistic(stats);

        if (settingsModel.showInfo)
            showStatisticDialog(stats);
        else timer.reset();
    }

    private Statistic getStatistic() {
        log.debug("o/");
        Statistic statistic = new Statistic();
        statistic.setTotalCarsCreated(carFactory.
          getTotalCreated());
        statistic.setTotalBikesCreated(bikeFactory.
          getTotalCreated());
        statistic.setTotalTime(timer.getSimulationTime())
          ;
        return statistic;
    }

    private void showCanvasStatistic(Statistic statistic)
       {
        drawer.setStatistic(statistic);
    }

    private void showStatisticDialog(Statistic statistic)
       {
        statisticDialog.setStatistic(statistic);
        statisticDialog.show();
    }

    @Override
    public void OnShowInfo() {
    }

    @Override
    public void OnHideInfo() {

    }

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

    }

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

    }


    @Override
    public void initializeUI() {


        contentPane = new JPanel();
        contentPane.setLayout(new GridBagLayout());
        contentPane.setInheritsPopupMenu(false);
        contentPane.setPreferredSize(new Dimension(800,
          600));
        GridBagConstraints gbc1 = new GridBagConstraints
          ();
        gbc1.gridx = 0;
        gbc1.gridy = 0;
        gbc1.weightx = 1.0;
        gbc1.fill = GridBagConstraints.HORIZONTAL;

        contentPane.add(toolBarView.getRootComponent(),
          gbc1);

        final JPanel panel1 = new JPanel();
        panel1.setLayout(new GridBagLayout());
        GridBagConstraints gbc = new GridBagConstraints()
          ;
        gbc.gridx = 1;
        gbc.gridy = 0;
        gbc.weighty = 1.0;
        gbc.anchor = GridBagConstraints.NORTH;
        gbc.fill = GridBagConstraints.HORIZONTAL;
         panel1.add(sideBarView.getRootComponent(), gbc);

        GridBagConstraints gbc0 = new GridBagConstraints
          ();
        gbc0.gridx = 0;
        gbc0.gridy = 1;
        gbc0.weightx = 1.0;
        gbc0.weighty = 1.0;
        gbc0.fill = GridBagConstraints.BOTH;
        contentPane.add(panel1, gbc0);


        final JPanel panel2 = new JPanel();
        panel2.setLayout(new GridBagLayout());
        GridBagConstraints gbc3 = new GridBagConstraints
           ();
        gbc3.gridx = 0;
        gbc3.gridy = 0;
        gbc3.weightx = 1.0;
        gbc3.weighty = 1.0;
        gbc3.fill = GridBagConstraints.BOTH;
        panel1.add(panel2, gbc3);

        GridBagConstraints gbc4 = new GridBagConstraints
           ();
        gbc4.gridx = 0;
        gbc4.gridy = 0;
        gbc4.weightx = 1.0;
        gbc4.weighty = 1.0;
        gbc4.fill = GridBagConstraints.BOTH;
        panel2.add(drawPanel, gbc4);

        setJMenuBar(menuView.getRootComponent());

        sideBarView.addFactorySettingsView(
           carsSettingsView);
        sideBarView.addFactorySettingsView(
           bikesSettingsView);

        setContentPane(contentPane);
    }

    public JPanel getRootComponent() {
        return contentPane;
    }


    @Override
    public void setController(SettingsController
       controller) {
        this.settingsController = controller;
    }

}
```

StatisticDialog.java

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


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

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

public class StatisticDialog {

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

    private Statistic statistic;
    private OnConfirmListener onConfirmListener;
    private OnCancelListener onCancelListener;

    private JDialog dialog;
    private JTextPane area;
    private JButton ok;
    private JButton cancel;
    private Frame parent;

    public StatisticDialog(Frame parent)
    {
        this.parent = parent;
        onConfirmListener = () -> log.debug("Confirmed");
        onCancelListener = () -> log.debug("Canceled");

        setupUI();
    }

    private void setupUI() {
        dialog = new JDialog(parent);
        dialog.setLocationRelativeTo(parent);
        dialog.setTitle("Simulation statistic");
        dialog.pack();
        dialog.setModal(false);
        dialog.setSize(300, 300);
        JPanel rootPanel = new JPanel();
        rootPanel.setLayout(new GridBagLayout());
        GridBagConstraints c = new GridBagConstraints();

        area = new JTextPane();
        area.setEnabled(false);
        c.gridx = 0;
        c.gridy = 0;
        c.gridwidth = 2;
        c.fill = GridBagConstraints.BOTH;
        rootPanel.add(area, c);
        ok = new JButton("Stop");
        c = new GridBagConstraints();
        c.gridx = 0;
        c.gridy = 1;
        rootPanel.add(ok, c);
        ok.addActionListener( e ->  { onConfirmListener.
           OnConfirm(); close(); });

        cancel = new JButton("Cancel");
        c = new GridBagConstraints();
        c.gridx = 1;
        c.gridy = 1;
        rootPanel.add(cancel, c);
        cancel.addActionListener( e -> { onCancelListener
           .OnCancel(); close(); });

        dialog.setContentPane(rootPanel);
    }

    public void setOnConfirmListener(OnConfirmListener
       onConfirmListener) {
        this.onConfirmListener = onConfirmListener;
    }

    public void setOnCancelListener(OnCancelListener
       onCancelListener) {
        this.onCancelListener = onCancelListener;
    }

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

    void show()
    {
        log.debug("o/");
        area.setContentType("text/html");
        String text = String.format("<b><font size=\"5\"
           face=\"Arial\">Total cars: %s</font><br></b>"+
                "<font size=\"5\"><u>Total bikes: %s</u
                   ></font><br>"+
                "<font size=\"5\"><i>Total time:%s</i></
                   font>",
                statistic.getTotalCarsCreated(),
                statistic.getTotalBikesCreated(),
                statistic.getTotalTime());
        area.setText(text);
        dialog.setVisible(true);
    }

    void close()
    {
        dialog.setVisible(false);
    }

    public interface OnConfirmListener
    {
        void OnConfirm();
    }

    public interface OnCancelListener
    {
        void OnCancel();
    }
}
```

FactoryOptionsView.java

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


import com.xotonic.lab.sit.settings.
  FactorySettingsController;
import com.xotonic.lab.sit.settings.FactorySettingsView;
import com.xotonic.lab.sit.settings.FactoryType;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

import javax.swing.*;
import java.awt.*;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class FactoryOptionsView implements
  FactorySettingsView<JPanel, FactorySettingsController>
  {

    private static final Map<FactoryType, String>
      localizedFactoryNames = new HashMap<>();
    private static Logger log = LogManager.getLogger(
      FactoryOptionsView.class.getName());

    static {
        localizedFactoryNames.put(FactoryType.car, "Cars
          options");
        localizedFactoryNames.put(FactoryType.bike, "Bike
          options");
    }

    private FactorySettingsController controller;
    private FactoryType factoryType;
    private JPanel root;
    private JTextField bornPeriodField;

    private JComboBox<Float> bornChanceCombo;
    private Float[] chances = new Float[] {
            0.0f, 0.1f, 0.2f, 0.3f, 0.4f, 0.5f, 0.6f, 0.7
              f, 0.8f, 0.9f, 1.0f};

    public FactoryOptionsView(FactoryType type)
    {
        setFactoryType(type);
    }

    private void success(JTextField bornPeriodField) {
        bornPeriodField.setForeground(Color.BLACK);
        bornPeriodField.setBackground(Color.GREEN);
    }

    private void fail(JTextField bornChanceField) {
        bornChanceField.setForeground(Color.BLACK);
        bornChanceField.setBackground(Color.RED);

        JOptionPane.showMessageDialog(root, "Error in " +
           localizedFactoryNames.get(factoryType));
    }

    private void initial(JTextField bornPeriodField) {
        bornPeriodField.setForeground(Color.BLACK);
        bornPeriodField.setBackground(Color.WHITE);
    }

    @Override
    public void setController(FactorySettingsController
      controller) {
        this.controller = controller;
    }

    @Override
    public void initializeUI() {

        assert factoryType != null;

        root = new JPanel();
        root.setLayout(new GridBagLayout());

        bornChanceCombo = new JComboBox<>();
        bornChanceCombo.setEditable(false);
        for (int chance = 0; chance < chances.length;
          chance++) {
            bornChanceCombo.addItem(chances[chance]);
        }

        bornPeriodField = new JTextField();
        bornPeriodField.setColumns(5);
        GridBagConstraints gbc = new GridBagConstraints()
          ;
        gbc.gridx = 0;
        gbc.gridy = 0;
        gbc.anchor = GridBagConstraints.NORTH;
        gbc.fill = GridBagConstraints.HORIZONTAL;

        root.add(bornPeriodField , gbc);
        gbc.gridy = 1;
        root.add(bornChanceCombo , gbc);

        root.setBorder(BorderFactory.createTitledBorder(
          localizedFactoryNames.get(factoryType)));


        bornChanceCombo.addActionListener(evt -> {
            log.debug("chance");
            updateBornChance();
        });
        bornPeriodField.addActionListener(evt -> {
            log.debug("period");
            updateBornPeriod();
        });

    }

    private void updateBornPeriod() {
        log.debug("o/");
        try {
            controller.setBornPeriod(
                    this,
                    Integer.parseInt(bornPeriodField.
                      getText()));
            success(bornPeriodField);

        }
        catch (NumberFormatException e)
        {
            fail(bornPeriodField);
        }
    }

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

        Float selected = bornChanceCombo.getItemAt(
          bornChanceCombo.getSelectedIndex());
        if (selected!=null && controller!=null)
            controller.setBornChance(this, selected);
        else
```

```
            log.debug("Skip updating");

    }

    @Override
    public JPanel getRootComponent() {
        return root;
    }

    @Override
    public void OnBornPeriodChanged(int bornPeriod) {
        bornPeriodField.setText(Integer.toString(
           bornPeriod));
        initial(bornPeriodField);

    }

    @Override
    public void OnBornChanceChanged(float bornChance) {
        int nextSelected = bornChanceCombo.getItemCount()
           ;

        List<Float> floats = Arrays.asList(chances);
        if (floats.contains(bornChance))
            bornChanceCombo.setSelectedIndex(floats.
               indexOf(bornChance));
        else
        {
            bornChanceCombo.addItem(bornChance);
            bornChanceCombo.setSelectedIndex(nextSelected
               );
        }
    }

    @Override
    public FactoryType getFactoryType() {
        return factoryType;
    }

    @Override
    public void setFactoryType(FactoryType type)
    {
        this.factoryType = type;
    }


}
```

SwingUtil.java

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

import javax.swing.*;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import javax.swing.text.Document;
import javax.swing.text.JTextComponent;
import java.beans.PropertyChangeEvent;
import java.util.Objects;
import java.util.regex.Matcher;
import java.util.regex.Pattern;


public class SwingUtil {

    public static void addChangeListener(JTextComponent
       text, ChangeListener changeListener) {
        Objects.requireNonNull(text);
        Objects.requireNonNull(changeListener);
        DocumentListener dl = new DocumentListener() {
            private int lastChange = 0,
               lastNotifiedChange = 0;

            @Override
            public void insertUpdate(DocumentEvent e) {
                changedUpdate(e);
            }

            @Override
            public void removeUpdate(DocumentEvent e) {
                changedUpdate(e);
            }

            @Override
            public void changedUpdate(DocumentEvent e) {
                lastChange++;
                SwingUtilities.invokeLater(() -> {
                    if (lastNotifiedChange != lastChange)
                       {
                        lastNotifiedChange = lastChange;
                        changeListener.stateChanged(new
                           ChangeEvent(text));
                    }
                });
            }
        };
        text.addPropertyChangeListener("document", (
           PropertyChangeEvent e) -> {
            Document d1 = (Document)e.getOldValue();
            Document d2 = (Document)e.getNewValue();
            if (d1 != null) d1.removeDocumentListener(dl)
               ;
            if (d2 != null) d2.addDocumentListener(dl);
            dl.changedUpdate(null);
        });
        Document d = text.getDocument();
        if (d != null) d.addDocumentListener(dl);
    }

    public static class RegExpInputVerifier extends
      InputVerifier {

        private String expression;

        public RegExpInputVerifier(String expression) {
            this.expression = expression;
        }

        public String getExpression() {
            return expression;
        }

        @Override
        public boolean verify(JComponent input) {
            if (input instanceof JTextComponent) {
                JTextComponent field = (JTextComponent)
                  input;
                String regNo1 = field.getText();
                Pattern pattern1 = Pattern.compile(
                  expression);
                Matcher matcher1 = pattern1.matcher(
                  regNo1);
                return matcher1.matches();
            }
            return false;
        }
    }

    public static class FloatVerifier extends
      InputVerifier {
        @Override
        public boolean verify(JComponent input) {
            String text = ((JTextField) input).getText();
            try {
                Float.parseFloat(text);
            } catch (NumberFormatException e) {
                return false;
            }

            return true;
        }
    }

}
```

DrawPanel.java

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


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);
    }
}
```
Вывод

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