МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

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

Лабораторная работа № 3

по дисциплине «Современные информационные технологии»
на тему ”Многопотоковые приложения. Потоки ввода-вывода. Сериализация объектов в файл.”

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

Новосибирск 2015 г.

Цель работы

1. Изучить особенности реализации и работы потоков в Java, управлением приоритетами потоков и синхронизацией потоков.
2. Доработать программу, созданную в лабораторных работах № 1-2:

Задание варианта

Вариант задания:

1. Автомобили двигаются по оси X от одного края области симуляции до другого со скоростью V.
2. Мотоциклы двигаются по оси Y от одного края области симуляции до другого со скоростью V.

Задание

Доработать программу, созданную в лабораторной работе № 2:

1. Создать абстрактный класс AI, описывающий «интеллектуальное поведение» объектов по варианту. Класс должен быть выполнен в виде отдельного потока и работать с коллекцией объектов;
2. Реализовать класс BaseAI для каждого из видов объекта, включив в него поведение, описанное в индивидуальном задании по варианту;
3. Синхронизовать работу потоков расчета интеллекта объектов с их рисованием. Рисование должно остаться в основном потоке. Синхронизация осуществляется через передачу данных в основной поток;
4. Добавить в панель управления кнопки для остановки и возобновления работы интеллекта каждого вида объектов. Реализовать через засыпание/пробуждение потоков;
5. Добавить в панель управления выпадающие списки для выставления приоритетов каждого из потоков.
6. Реализовать сохранение объектов в файл.
7. Реализовать сохранение и загрузку настроек параметров программы в локальный файл.

Вывод

Были рассмотрены особенности реализации и работы потоков в Java, управлением приоритетами потоков и синхронизацией потоков. Была разработана многооточная программа для симуляции движение транспортных средств, в которой логика, ИИ и визуализация просчитываются в различных потоках и реализовано их синхронизированное взаимодействие между собой.

Приложение А. Листинг программы

MyMath.java

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

/**
 * Вспомогательный класс с математическими операциями
 */
public class MyMath {

    /** Заключить число в интервал */
    public static float clamp(float min, float value, float max) {
        return Math.max(min, Math.min(max, value));
    }

    /** Отразить"" число от границ интервала */
    public static float reflect(float min, float value, float max, float step) {
        if (value + step >= max) {
            return max - Math.abs(step - Math.abs(max - value));
        }
        if (value + step <= min) {
            return min + Math.abs(step - Math.abs(value - min));
        }
        return value + step;
    }
}
```

Model.java

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


import java.io.Serializable;

/** Этот класс просто говорит о том,
 * что все модели должны быть сериализуемыми */
public interface Model extends Serializable {
}
```

View.java

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

/** Абстрактное представление
 * Работает только с контроллером, не имеет доступ к модели*/
public interface View<ControllerType extends Controller> {

    void setController(ControllerType controller);
}
```

TotalModel.java

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


import com.xotonic.lab.sit.settings.factory.FactoryModel;
import com.xotonic.lab.sit.settings.factory.FactoryType;

import java.util.Arrays;
```

```
import java.util.HashMap;
import java.util.Map;

/** Вся сохраняемая информация о настройках программы */
public class TotalModel implements Model {

    public SimulationState simulationState;
    public boolean showInfo = false;
    public boolean showTime = true;

    public int bikeAIThreadPriority = 2;
    public int carAIThreadPriority = 2;

    public boolean isCarAIToggled = true;
    public boolean isBikeAIToggled = true;
    public Map<FactoryType, FactoryModel> factoriesSettings = new HashMap<>();

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

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

Controller.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;

/** Абстрактный контроллер.
 * Работает с одной моделью и множеством представлений */
public abstract class Controller<ModelType extends Model, ViewType extends View>
    {


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


    protected ModelType model;
    protected Collection<ViewType> views = new ArrayList<>();

    public void addView(ViewType view) {
        log.debug("o/");
        views.add(view);
    }

    public void setModel(ModelType model) {
        log.debug("o/");
        this.model = model;

    }
}
```

HasUI.java

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

import javax.swing.*;

/** Обьект, который имеет интерфейс,
 * а значит у него должен быть корневой элемент */
public interface HasUI<RootComponent extends JComponent> {
    /** Инициализация интерфейса. Навешивание обработчиков событий */
    void initializeUI();
    RootComponent getRootComponent();
}
```

SettingsController.java

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


import com.xotonic.lab.sit.settings.Controller;
import com.xotonic.lab.sit.settings.TotalModel;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

/** Стандартный контроллер для всех настроек */
public class SettingsController extends Controller<TotalModel, SettingsView>
{

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


    @Override
    public void setModel(TotalModel model) {
        super.setModel(model);
        updateFull();
    }

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

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

    private void updateSimulationState() {
        views.forEach(model.simulationState == TotalModel.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) {
        super.addView(view);
        updateFull();
    }

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

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

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

        model.simulationState = TotalModel.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();
    }

}
```

SettingsView.java

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


import com.xotonic.lab.sit.settings.HasUI;
import com.xotonic.lab.sit.settings.View;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

import javax.swing.*;
/** Абстракное представление основных настроек */
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();
}
```

AISettingsController.java

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

import com.xotonic.lab.sit.settings.Controller;
import com.xotonic.lab.sit.settings.TotalModel;

/** Абстрактный контроллер настроек ИИ */
public class AISettingsController
        extends Controller<TotalModel, AISettingsView> {

    private void updateAll() {
        setBikeAIToggled(model.isBikeAIToggled);
        setBikeThreadPriority(model.bikeAIThreadPriority);
        setCarAIToggled(model.isCarAIToggled);
        setCarThreadPriority(model.carAIThreadPriority);
    }

    @Override
    public void addView(AISettingsView view) {
        super.addView(view);
        updateAll();
    }

    @Override
    public void setModel(TotalModel model) {
        super.setModel(model);
        updateAll();
    }

    public void setBikeThreadPriority(int priority) {
        model.bikeAIThreadPriority = priority;
        views.forEach(v -> v.OnBikeThreadPriorityChanged(model.
   bikeAIThreadPriority));
    }

    public void setCarThreadPriority(int priority) {
        model.carAIThreadPriority = priority;
        views.forEach(v -> v.OnCarThreadPriorityChanged(priority));
    }

    public void setBikeAIToggled(boolean on) {
        model.isBikeAIToggled = on;
        views.forEach(v -> v.OnBikeAIToggled(on));
    }

    public void setCarAIToggled(boolean on) {
        model.isCarAIToggled = on;
        views.forEach(v -> v.OnCarAIToggled(on));
    }
}
```

AISettingsView.java

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

import com.xotonic.lab.sit.settings.View;

/** Абстрактное представление настроек ИИ */
public interface AISettingsView<SettingsControllerType extends
    AISettingsController>

        extends View<SettingsControllerType> {

    void OnBikeThreadPriorityChanged(int priority);

    void OnCarThreadPriorityChanged(int priority);

    void OnCarAIToggled(boolean on);

    void OnBikeAIToggled(boolean on);
}
```

FactorySettingsView.java

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


import com.xotonic.lab.sit.settings.HasUI;
import com.xotonic.lab.sit.settings.View;

import javax.swing.*;

/** Представление настроек фабрики */
public interface FactorySettingsView
        <RootComponent extends JComponent,
         SettingsControllerType extends FactorySettingsController>

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

    FactoryType getFactoryType();

    void setFactoryType(FactoryType type);
}
```

FactoryType.java

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

/** Тип фабрики. Тут же задаются значения по умолчанию */
public enum FactoryType {
    car(new FactoryModel(100, 1.0f)),
    bike(new FactoryModel(100, 1.0f));

    private FactoryModel defaultModel;

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

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

FactorySettingsController.java

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


import com.xotonic.lab.sit.settings.Controller;
import com.xotonic.lab.sit.settings.TotalModel;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

/** Контроллер настроек фабрики */
public class FactorySettingsController
        extends Controller<TotalModel, FactorySettingsView>
{
    private static Logger log = LogManager.getLogger(FactorySettingsController.
   class.getName());


    @Override
    public void setModel(TotalModel model) {
        super.setModel(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) {
        super.addView(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.factory;


import com.xotonic.lab.sit.settings.Model;

/** Информация о фабрике для сохранения в файл */
public class FactoryModel implements Model {

    int bornPeriod;
    float bornChance;

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

MutableWorld.java

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

/**
 * Изменяемый мир, доступен только нескольким объектам
 */
public class MutableWorld extends World {

    public void setAreaWidth(int value)
    {
        areaWidth = value;
    }
```

```
    public void setAreaHeight(int value)
    {
        areaHeight = value;
    }

    public void setTimeMillis(long time)
    {
        timeMillis = time;
    }
}
```

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(World world);

    void stop();
}
```

AIManager.java

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


import java.util.Collection;

/** Управляет обьектами конкретного типа */
public class AIManager extends BasicBehavior {

    private final VehicleType vehicleType;
    private Collection<Vehicle> vehicles;
    private boolean isStarted;

    public AIManager(VehicleType type) {
        vehicleType = type;
    }

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

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

    @Override
    public void update(World world) {

        synchronized (vehicles) {
            vehicles.stream()
                    .filter(ai -> ai.getType() == vehicleType && ai instanceof
    ThinkingVehicle)
                    .forEach(ai -> ((ThinkingVehicle) ai).processAI(world));
        }
    }

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

    public AIManager setVehicles(Collection<Vehicle> brains) {
        this.vehicles = brains;
        return this;
    }
}
```

ThinkingVehicle.java

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

/**
 * ТС с ИИ
 */
public abstract class ThinkingVehicle extends Vehicle {

    /** Входные параметры */
    private transient AI.Input input = new AI.Input();
    /** Выходные параметры */
    private transient AI.Output output = new AI.Output();

    /** Текущий ИИ */
    private transient AI ai;

    private long lastUpdated = 0;

    public ThinkingVehicle(String id, float x, float y) {
        super(id, x, y);
    }

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

    public AI getAi() {
        return ai;
    }

    public void setAi(AI ai) {
        this.ai = ai;
    }
```

```
    /** Подумать */
    public void processAI(World world) {
        input.areaHeight = world.getAreaHeight() - 64.f;
        input.areaWidth = world.getAreaWidth() - 115.f;
        input.timestep = world.getTimeMillis() - lastUpdated;
        input.me = this;
        lastUpdated = world.getTimeMillis();

        output = ai.think(input);

        setX(output.x);
        setY(output.y);
    }
}
```

World.java

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


/** Обьект, который будет доступен любой сущности в каждый момент времени
 *  но только для чтения
 * */
public class World {

    protected long timeMillis;
    protected int areaHeight;
    protected int areaWidth;

    public long getTimeMillis() {
        return timeMillis;
    }

    public int getAreaHeight() {
        return areaHeight;
    }

    public int getAreaWidth() {
        return areaWidth;
    }

}
```

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;

/** Единственная реализация класса среды */
public class SimpleHabitat extends Habitat {

    private ArrayList<Vehicle> vehicles;
    private Collection<Factory> factories;
```

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

/** Обновляем все
 * @param world*/
@Override
public void update(World world) {

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


    for (Vehicle v : vehicles)
        if (v.isStarted()) {
            v.update(world);
        }
        else
            v.start();

    synchronized (vehicles) {
        vehicles.sort((o1, o2) ->
                o1.getY() > o2.getY() ? 1 : o1.getY() == o2.getY() ? 0 : -1)
;
    }
}

/** Запускаем все */
@Override
public void start() {
    log.debug("SimpleHabitat start ...");

    factories.forEach(Behavior::start);

    vehicles.forEach(Vehicle::start);

}

/** Останавливаем все */
@Override
public void stop() {
    log.debug("SimpleHabitat stop ...");

    factories.forEach(Behavior::stop);

    vehicles.forEach(Vehicle::stop);

}

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

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

```
    @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;


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

    public 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(World world) {
```

```
        if (createChance > 1f | createChance < 0f)
            log.error("Chance value is not in range [0.0;1.0] (now {})",
    createChance);

        time += world.getTimeMillis() - prevTimeMillis;
        prevTimeMillis = world.getTimeMillis();
        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();
        synchronized (habitat.getVehicles()) {
            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;

import java.io.Serializable;

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

    protected ResourceId resourceId = ResourceId.DEFAULT;
    protected ResourceId resourceIdWhenMovingBack = ResourceId.DEFAULT;
    protected boolean isMovingBack = false;
    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 boolean isMovingBack() {
        return isMovingBack;
    }

    public void setMovingBack(boolean movingBack) {
        isMovingBack = movingBack;
    }

    public ResourceId getResourceId() {
        return !isMovingBack ? resourceIdWhenMovingBack : 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;
    }

    public abstract VehicleType getType();
}
```

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 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;
    }
}
```

VehicleType.java

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


public enum VehicleType {
    car, bike
}
```

AI.java

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

import java.util.Collection;

/**
 * Базовый класс для всех обьектов с ИИ
 */
public interface AI {

    /** Главный метод, тут должна быть вся логика */
    Output think(Input input);

    /** Параметры, которые будут подаваться на вход */
    class Input
    {
        /** Шаг времени */
        public long timestep;
        /** Размеры площади, по которой можно перемещаться */
        public float areaHeight, areaWidth;

        public Vehicle me;
        /** Другие виклы, кроме данной */
        public Collection<Vehicle> otherVehicles;
    }

    /** Параметры, которые будут отдаваться на выход */
    class Output
    {
```

```
        /** Координаты */
        public float x, y;
        /** Едет ли назад */
        public boolean isMovingBack;
    }

}
```

BikeAI.java

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


import com.xotonic.lab.sit.MyMath;
import com.xotonic.lab.sit.vehicle.AI;
import com.xotonic.lab.sit.vehicle.car.CarAI;
import org.apache.logging.log4j.LogManager;

public class BikeAI implements AI {

    private static org.apache.logging.log4j.Logger log =
            LogManager.getLogger(CarAI.class.getName());
    private Output o = new Output();
    private float speed = 0.1f;

    @Override
    public Output think(Input input) {

        float y = input.me.getY();
        float signedSpeed = input.me.isMovingBack() ? speed : -speed;
        float step = input.timestep * signedSpeed;
        float nextY = y + step;
        if (nextY >= input.areaHeight || nextY <= 0) {
            input.me.setMovingBack(!input.me.isMovingBack());
        }
        signedSpeed = input.me.isMovingBack() ? speed : -speed;
        step = input.timestep * signedSpeed;
        nextY = MyMath.reflect(0.f, y, input.areaHeight, step);

        o.y = nextY;
        o.x = input.me.getX();

        return o;
    }
}
```

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.bike;

import com.xotonic.lab.sit.ui.ResourceId;
import com.xotonic.lab.sit.vehicle.ThinkingVehicle;
import com.xotonic.lab.sit.vehicle.VehicleType;
import com.xotonic.lab.sit.vehicle.World;

/**
```

```
 * @author User
 */
public class Bike extends ThinkingVehicle {


    protected ResourceId resourceId = ResourceId.BIKE;
    protected ResourceId resourceIdWhenMovingBack = ResourceId.BIKE_BACK;

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

    @Override
    public ResourceId getResourceId() {
        return !isMovingBack ? resourceIdWhenMovingBack : resourceId;
    }

    @Override
    public VehicleType getType() {
        return VehicleType.bike;
    }


    @Override
    public void update(World world) {

    }
}
```

BikeFactory.java

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

import com.xotonic.lab.sit.MyMath;
import com.xotonic.lab.sit.vehicle.Habitat;
import com.xotonic.lab.sit.vehicle.TimedLuckyFactory;
import com.xotonic.lab.sit.vehicle.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(MyMath.clamp(0.1f, r.nextFloat(), 0.9f) * habitat.
   getWorldWidth());
        bike.setY(MyMath.clamp(0.1f, r.nextFloat(), 0.9f) * habitat.
   getWorldHeight());
        bike.setMovingBack(r.nextBoolean());
        bike.setAi(new BikeAI());
```

```
        return bike;
    }

}
```

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.car;

import com.xotonic.lab.sit.ui.ResourceId;
import com.xotonic.lab.sit.vehicle.ThinkingVehicle;
import com.xotonic.lab.sit.vehicle.VehicleType;
import com.xotonic.lab.sit.vehicle.World;

/**
 * @author User
 */
public class Car extends ThinkingVehicle {


    protected ResourceId resourceId = ResourceId.CAR;
    protected ResourceId resourceIdWhenMovingBack = ResourceId.CAR_BACK;

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

    @Override
    public ResourceId getResourceId() {
        return !isMovingBack ? resourceIdWhenMovingBack : resourceId;
    }

    @Override
    public VehicleType getType() {
        return VehicleType.car;
    }


    @Override
    public void update(World world) {

    }
}
```

CarAI.java

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


import com.xotonic.lab.sit.MyMath;
import com.xotonic.lab.sit.vehicle.AI;
import org.apache.logging.log4j.LogManager;


public class CarAI implements AI {
```

```
    private static org.apache.logging.log4j.Logger log =
            LogManager.getLogger(CarAI.class.getName());

    private final Output o = new Output();
    private float speed = 0.1f;

    @Override
    public Output think(Input input) {

        float x = input.me.getX();
        float signedSpeed = input.me.isMovingBack() ? speed : -speed;
        float step = input.timestep * signedSpeed;
        float nextX = x + step;

        if (nextX >= input.areaWidth || nextX <= 0)
            input.me.setMovingBack(! input.me.isMovingBack());

        signedSpeed = input.me.isMovingBack() ? speed : -speed;
        step = input.timestep * signedSpeed;
        nextX = MyMath.reflect(0.f, x, input.areaWidth, step);

        o.x = nextX;
        o.y = input.me.getY();

        return o;
    }

}
```

CarFactory.java

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

import com.xotonic.lab.sit.MyMath;
import com.xotonic.lab.sit.vehicle.Habitat;
import com.xotonic.lab.sit.vehicle.TimedLuckyFactory;
import com.xotonic.lab.sit.vehicle.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(MyMath.clamp(0.1f, r.nextFloat(), 0.9f) * habitat.getWorldWidth
   ());
        car.setY(MyMath.clamp(0.1f, r.nextFloat(), 0.9f) * habitat.
   getWorldHeight());
        car.setMovingBack(r.nextBoolean());
        car.setAi(new CarAI());
```

```
        return car;
    }


}
```

Canvas.java

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


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

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


/** Панель отрисовки */
class Canvas extends JPanel {
    Font centerFont = new Font("Consolas", 1, 36);
    Font cornerFont = new Font("Arial", 1, 12);
    private Collection<Vehicle> vehicles;
    private long lastUpdatedTime = 0;
    private boolean started = false;
    private boolean stopped = false;
    private boolean isShowTime = true;
    private Statistic statistic;

    Canvas() {
        super();
    }

    boolean isShowTime() {
        return isShowTime;
    }

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

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


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

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

        if (started) {
            drawVehicles(g);
        }
```

```
    if (started && isShowTime) {

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


    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) {

    synchronized (vehicles) {

        for (Vehicle v : vehicles) {

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


public void start() {

    started = true;
    stopped = false;
}

public void update(World world) {
    lastUpdatedTime = world.getTimeMillis();
    repaint();
}


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

public void setVehicles(Collection<Vehicle> vehicles) {
```

```
        if (this.vehicles == null) {
            this.vehicles = vehicles;
        }

    }

    /** Рисуем текст статистики */
    private void drawLinesCenter(Graphics g, String... lines) {
        Color temp = g.getColor();
        g.setFont(centerFont);
        FontMetrics metrics = g.getFontMetrics(centerFont);
        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);
    }

    /** Рисуем текст в углу */
    private void drawLinesTopLeft(Graphics g, String... lines) {
        Color temp = g.getColor();

        g.setFont(cornerFont);
        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);
    }
}
```

SideBarView.java

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

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

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

/** Боковая панель */
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);
    }

  /** Создать интерфейс */
@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);
    }

    public void addAISettingsView(AIOptionsView view) {
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.gridx = 0;
        gbc.gridy = GridBagConstraints.RELATIVE;
        gbc.anchor = GridBagConstraints.NORTH;
        gbc.fill = GridBagConstraints.HORIZONTAL;
        propertiesPanel.add(view.getRootComponent(), gbc);
```

```
    }
}
```

MenuView.java

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


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

import javax.swing.*;
import java.io.File;

/** Меню */
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;
    private JMenuItem openFileItem;
    private JMenuItem saveFileItem;
    private JMenuItem saveVehiclesItem;
    private Form form;

    public MenuView(Form form) {
        this.form = form;
    }

    /**
     * Создать интерфейс
     */
 @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);

     openFileItem = new JMenuItem("Open");
     saveFileItem = new JMenuItem("Save");
     saveVehiclesItem = new JMenuItem("Save objects");
     menuFile.add(openFileItem);
     menuFile.add(saveFileItem);
     menuFile.add(saveVehiclesItem);
```

```
        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));

        openFileItem.addActionListener(a -> {
            JFileChooser fileopen = new JFileChooser();
            int ret = fileopen.showOpenDialog(null);
            if (ret == JFileChooser.APPROVE_OPTION) {
                File file = fileopen.getSelectedFile();
                form.loadModelsFromFile(file);
            }
        });

        saveFileItem.addActionListener(a -> {
            JFileChooser fileopen = new JFileChooser();
            int ret = fileopen.showSaveDialog(null);
            if (ret == JFileChooser.APPROVE_OPTION) {
                File file = fileopen.getSelectedFile();
                form.saveModelsToFile(file);
            }
        });

        saveVehiclesItem.addActionListener(a -> {
            JFileChooser fileopen = new JFileChooser();
            int ret = fileopen.showSaveDialog(null);
            if (ret == JFileChooser.APPROVE_OPTION) {
```

```
                File file = fileopen.getSelectedFile();
                form.saveObjectsToFile(file);
            }
        });
    }



    @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;

/** Загрузчик ресурсов */
public enum ResourceId {

    DEFAULT("default.png"),
    CAR("car.png"),
    CAR_BACK("car_back.png"),
    BIKE("bike_back.png"),
    BIKE_BACK("bike.png");

    private String resourcePath;
    private BufferedImage image;

    ResourceId(String resourcePath) {
        System.setProperty("sun.java2d.opengl", "true");
        this.resourcePath = resourcePath;
        image = loadResource(resourcePath);
    }

    public String getResourcePath() {
        return resourcePath;
    }

    public BufferedImage getImage() {
        return image;
    }

    /** Загрузить картинку из файла ресурсов */
    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 toCompatibleImage(image);
        } catch (IOException ex) {
            ex.printStackTrace();
            return getFailedLoadingImage();
        }
        catch (Exception e)
        {
            log.error("Exception during loading resource", e);
            return getFailedLoadingImage();
        }
    }

    /** Получить аварийное изображение */
    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;
    }

    /** Оптимизация изображения для видеокарты */
    private BufferedImage toCompatibleImage(BufferedImage image)
    {
        // obtain the current system graphical settings
        GraphicsConfiguration gfx_config = GraphicsEnvironment.
                getLocalGraphicsEnvironment().getDefaultScreenDevice().
                getDefaultConfiguration();

    /*
     * if image is already compatible and optimized for current system
     * settings, simply return it
     */
        if (image.getColorModel().equals(gfx_config.getColorModel()))
            return image;

        // image is not optimized, so create a new image that is
        BufferedImage new_image = gfx_config.createCompatibleImage(
                image.getWidth(), image.getHeight(), image.getTransparency());

        // get the graphics context of the new image to draw the old image on
        Graphics2D g2d = (Graphics2D) new_image.getGraphics();

        // actually draw the image and dispose of context no longer needed
        g2d.drawImage(image, 0, 0, null);
        g2d.dispose();

        // return the new optimized image
        return new_image;
    }


}
```

ToolBarView.java

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

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

import javax.swing.*;

/** Панель инструментов */
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;
    }

  /** Создать интерфейс */
@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;
    }
}
```

Statistic.java

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

import java.io.Serializable;

/** Бин статистики */
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.factory.FactorySettingsController;
```

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

import javax.swing.*;

/** Класс который слушает контроллер фабрик и управляет ей фабрикой() */
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) {
    }

   /** Создать интерфейс */
 @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) {
    }
}
```

SimulationHandler.java

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

import com.xotonic.lab.sit.settings.ai.AISettingsController;
import com.xotonic.lab.sit.settings.ai.AISettingsView;
import com.xotonic.lab.sit.vehicle.*;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
```

```
import java.util.Collection;
import java.util.concurrent.atomic.AtomicBoolean;

/** Класс, который аркестрирует потоками. Обновляет отрисовку и логику. */
public class SimulationHandler implements AISettingsView<AISettingsController> {

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

    private Habitat habitat;
    private MutableWorld world;
    private AIManager bikeAIManager;
    private AIManager carAIManager;

    private Canvas canvas;
    private boolean started = false;
    /* Параметры времени обновления */
    private int delay = 20;
    private long simulationTime = 0;
    private long simulationStartTime = -1;

    /* Потоки */
    private Thread logicThread;     // Обновление среды
    private Thread renderThread;    // Рендер
    private Thread carAIThread;     // ИИ машин
    private Thread bikeAIThread;    // ИИ мотоциклов
    /* Флаги для синхронизации */
    private final AtomicBoolean isBikeAiStarted = new AtomicBoolean();
    private final AtomicBoolean isCarAiStarted = new AtomicBoolean();

    public SimulationHandler() {
        bikeAIManager = new AIManager(VehicleType.bike);
        carAIManager = new AIManager(VehicleType.car);
    }
    /** Создать потоки */
    private void createThreads() {
        logicThread = new Thread(() -> {
            while (started) {
                sleep();
                updateWorld();
                habitat.update(world);
            }
        });

        renderThread = new Thread(() ->
        {
            while (started) {
                canvas.update(world);
            }
        });

        bikeAIThread = new Thread(() -> {
            while (started) {
                {
                    if (isBikeAiStarted.get())
                        bikeAIManager.update(world);
                    else {
                        try {
                            synchronized (isBikeAiStarted) {
                                isBikeAiStarted.wait();
                            }
```

```
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            }
        }
    });

    carAIThread = new Thread(() -> {
        while (started) {
            {
                if (isCarAiStarted.get())
                    carAIManager.update(world);
                else {
                    try {
                        synchronized (isCarAiStarted) {
                            isCarAiStarted.wait();
                        }
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            }
        }
    });
}

private void sleep() {
    try {
        Thread.sleep(delay);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
/** Обновить мир для следующего шага симуляции */
private void updateWorld() {
    if (simulationStartTime == -1)
        simulationStartTime = System.currentTimeMillis();
    simulationTime = System.currentTimeMillis() - simulationStartTime;
    world.setTimeMillis(simulationTime);
    world.setAreaWidth(canvas.getWidth());
    world.setAreaHeight(canvas.getHeight());
}

public long getSimulationTime() {
    return simulationTime;
}

public void setHabitat(Habitat habitat) {
    this.habitat = habitat;

}

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

public void start() {
    log.debug("Start");
    if (!started) {
```

```
            createThreads();

            Collection<Vehicle> vehicles = habitat.getVehicles();
            canvas.setVehicles(vehicles);
            bikeAIManager.setVehicles(vehicles);
            carAIManager.setVehicles(vehicles);
            habitat.start();
            canvas.start();
            started = true;
            startThreads();

        } else {
            log.warn("Already started");
        }
    }

    private void startThreads() {
        logicThread.start();
        renderThread.start();
        bikeAIThread.start();
        carAIThread.start();
    }


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

    private void joinThreads() {
        try {
            logicThread.join();
            notifyBikeAI();
            bikeAIThread.join();
            notifyCarAi();
            carAIThread.join();
            renderThread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    private void notifyCarAi() {
        synchronized (isCarAiStarted) {
            isCarAiStarted.notifyAll();
        }
    }

    private void notifyBikeAI() {
        synchronized (isBikeAiStarted) {
            isBikeAiStarted.notifyAll();
        }
    }
```

```
    public void setWorld(MutableWorld world) {
        this.world = world;
    }

    public SimulationHandler setCanvas(Canvas canvas) {
        this.canvas = canvas;
        return this;
    }


    @Override
    public void OnBikeThreadPriorityChanged(int priority) {
        if (started)
            bikeAIThread.setPriority(priority);
    }

    @Override
    public void OnCarThreadPriorityChanged(int priority) {
        if (started)
            carAIThread.setPriority(priority);

    }

    @Override
    public void OnCarAIToggled(boolean on) {
        if (started) {
            isCarAiStarted.set(on);

            if (on) {
                notifyCarAi();
            }
        }

    }

    @Override
    public void OnBikeAIToggled(boolean on) {
        if (started) {
            isBikeAiStarted.set(on);

            if (on) {
                notifyBikeAI();
            }
        }

    }

    @Override
    public void setController(AISettingsController controller) {
        createThreads();

        controller.setBikeAIToggled(isBikeAiStarted.get());
        controller.setCarAIToggled(isCarAiStarted.get());
        controller.setBikeThreadPriority(bikeAIThread.getPriority());
        controller.setCarThreadPriority(carAIThread.getPriority());
    }
}
```

Form.java

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

import com.xotonic.lab.sit.settings.TotalModel;
import com.xotonic.lab.sit.settings.ai.AISettingsController;
import com.xotonic.lab.sit.settings.factory.FactorySettingsController;
import com.xotonic.lab.sit.settings.factory.FactoryType;
import com.xotonic.lab.sit.settings.settings.SettingsController;
import com.xotonic.lab.sit.settings.settings.SettingsView;
import com.xotonic.lab.sit.vehicle.Habitat;
import com.xotonic.lab.sit.vehicle.MutableWorld;
import com.xotonic.lab.sit.vehicle.SimpleHabitat;
import com.xotonic.lab.sit.vehicle.TimedLuckyFactory;
import com.xotonic.lab.sit.vehicle.bike.BikeFactory;
import com.xotonic.lab.sit.vehicle.car.CarFactory;
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;
import java.io.*;

/** Главная форма */

public class Form extends JFrame
        implements KeyListener,
        SettingsView<JPanel, SettingsController>
{

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

    /**
     * Главная панель
     */
    private JPanel contentPane;
    /**
     * Панель для отрисовки
     */
    private JPanel drawPanel;

    /** Параметры мира */
    private MutableWorld world;
    /** Окружение */
    private Habitat habitat;
    /** Фабрика машин */
    private TimedLuckyFactory carFactory;
    /** Фабрика мотоциклов */
    private TimedLuckyFactory bikeFactory;

    /* Вспомогательные классы */
    private Canvas canvas;
    private SimulationHandler simulation;
    private StatisticDialog statisticDialog;

    /* — Система MVC — */
```

```
    /* Модели */
    private TotalModel totalModel;

    /* Контроллеры */
    private FactorySettingsController factoriesController;
    private AISettingsController aiSettingsController;
    private SettingsController settingsController;

    /* Вьюшки */
    /** Меню */
    private MenuView menuView;
    /** Панель инструментов */
    private ToolBarView toolBarView;
    /** Боковая панель */
    private SideBarView sideBarView;
    /** Панель настройки фабрики машин */
    private FactoryOptionsView carsSettingsView;
    /** Панель настройки фабрики байков */
    private FactoryOptionsView bikesSettingsView;
    /**
     * Панель с настройками ИИ
     */
    private AIOptionsView aiOptionsView;

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

        /*
         СОЗДАНИЕ И СВЯЗЫВАНИЕ ВСЕХ КОМПОНЕНТОВ В ОДНОМ МЕСТЕ
         */

        world = new MutableWorld();
        world.setAreaHeight(600);
        world.setAreaWidth(800);
        world.setTimeMillis(0);

        habitat = new SimpleHabitat();

        carFactory = new CarFactory(habitat);
        bikeFactory = new BikeFactory(habitat);

        createDrawPanel();

        simulation = new SimulationHandler();
        simulation.setHabitat(habitat);
        simulation.setWorld(world);
        simulation.setCanvas(canvas);


        statisticDialog = new StatisticDialog(this);
        statisticDialog.setOnConfirmListener(() -> simulation.reset()); //
controller.setStop();
        statisticDialog.setOnCancelListener(() -> settingsController.setStart())
;
        totalModel = new TotalModel();

        menuView = new MenuView(this);
        toolBarView = new ToolBarView();
        sideBarView = new SideBarView();
```

```
        carsSettingsView = new FactoryOptionsView(FactoryType.car);
        bikesSettingsView = new FactoryOptionsView(FactoryType.bike);
        aiOptionsView = new AIOptionsView();

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

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

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

        aiSettingsController = new AISettingsController();
        aiOptionsView.setController(aiSettingsController);
        aiSettingsController.setModel(totalModel);
        aiSettingsController.addView(aiOptionsView);
        aiSettingsController.addView(simulation);
        simulation.setController(aiSettingsController);

        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");
    }
```

```
/** Установка цветовой схемы */
private static void setLookAndFeel() {

    try {
        UIManager.setLookAndFeel(new NimbusLookAndFeel());
    } catch (UnsupportedLookAndFeelException e) {
        e.printStackTrace();
    }
}

/** Создать панель отрисовки */
private void createDrawPanel() {
    Canvas panel = new Canvas();

    drawPanel = panel;
    canvas = panel;

    panel.addComponentListener(new ComponentListener() {
        public void componentResized(ComponentEvent e) {
            habitat.setWorldWidth(canvas.getWidth());
            habitat.setWorldHeight(canvas.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() {
    simulation.start();
}

private void toggleShowTime() {
    canvas.setShowTime(!canvas.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 (totalModel.showInfo)
        showStatisticDialog(stats);
    else simulation.reset();
}
/** Собрать статистику */
private Statistic getStatistic() {
    log.debug("o/");
    Statistic statistic = new Statistic();
    statistic.setTotalCarsCreated(carFactory.getTotalCreated());
    statistic.setTotalBikesCreated(bikeFactory.getTotalCreated());
    statistic.setTotalTime(simulation.getSimulationTime());
    return statistic;
}

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

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

@Override
public void OnShowInfo() {

}

@Override
public void OnHideInfo() {
```

```
    }

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

    }

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

    }


  /** Создать интерфейс */
@Override
    public void initializeUI() {

        contentPane = new JPanel();
        contentPane.setLayout(new GridBagLayout());
        contentPane.setInheritsPopupMenu(false);
     contentPane.setPreferredSize(new Dimension(
              world.getAreaWidth(),
              world.getAreaHeight()));
        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);
 sideBarView.addAISettingsView(aiOptionsView);

    setContentPane(contentPane);
}

public JPanel getRootComponent() {
    return contentPane;
}


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

/** Сохранить настройки в файл */
public void saveModelsToFile(File f) {
    try {
        FileOutputStream saveFile = new FileOutputStream(f);
        ObjectOutputStream save = new ObjectOutputStream(saveFile);
        save.writeObject(totalModel);
        save.close(); // This also closes saveFile.

    } catch (IOException e) {
        e.printStackTrace();
    }
}
/** Загрузить настройки из файла */
public void loadModelsFromFile(File f) {
    try {
        FileInputStream saveFile = new FileInputStream(f);
        ObjectInputStream save = new ObjectInputStream(saveFile);
        totalModel = (TotalModel) save.readObject();
        save.close();

        aiSettingsController.setModel(totalModel);
        settingsController.setModel(totalModel);
        factoriesController.setModel(totalModel);
    } catch (Exception e) {
        e.printStackTrace();
    }

}
/** Сохранить объекты в файл */
public void saveObjectsToFile(File f)
{
    try {
        FileOutputStream saveFile = new FileOutputStream(f);
```

```
            ObjectOutputStream save = new ObjectOutputStream(saveFile);
            save.writeObject(habitat.getVehicles());
            save.close(); // This also closes saveFile.

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

}
```

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.*;

/** Окно статистики */
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();
    }

    /** Инициализация интерфейса */
    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;
    }

    /** Показать окно */
    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.factory.FactorySettingsController;
import com.xotonic.lab.sit.settings.factory.FactorySettingsView;
import com.xotonic.lab.sit.settings.factory.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;

/** Панель с настройками фабрики */
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, 0.7f, 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;
    }

   /** Создать интерфейс */
@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;
    }


}
```

AIOptionsView.java

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


import com.xotonic.lab.sit.settings.HasUI;
import com.xotonic.lab.sit.settings.ai.AISettingsController;
import com.xotonic.lab.sit.settings.ai.AISettingsView;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

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

public class AIOptionsView implements AISettingsView<AISettingsController>,
    HasUI<JPanel> {

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

    private AISettingsController controller;
    private JPanel root;
    private JSpinner bikeSpinner;
    private JSpinner carSpinner;
    private JCheckBox bikeToggle;
    private JCheckBox carToggle;

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

    @Override
    public void initializeUI() {
        root = new JPanel();
        root.setLayout(new GridBagLayout());
        root.setBorder(BorderFactory.createTitledBorder("AI options"));
        SpinnerModel carModel =
                new SpinnerNumberModel(0, 0, 100, 1);
        SpinnerModel bikeModel =
                new SpinnerNumberModel(0, 0, 100, 1);
        bikeSpinner = new JSpinner(bikeModel);
        carSpinner = new JSpinner(carModel);

        bikeToggle = new JCheckBox("Enable");
        carToggle = new JCheckBox("Enable");

        JPanel carPanel = new JPanel(new GridBagLayout());
        carPanel.setBorder(BorderFactory.createTitledBorder("Car"));
        JPanel bikePanel = new JPanel(new GridBagLayout());
        bikePanel.setBorder(BorderFactory.createTitledBorder("Bike"));

        GridBagConstraints gbc = new GridBagConstraints();
        gbc.gridx = 0;
        gbc.gridy = GridBagConstraints.RELATIVE;
```

```
        gbc.anchor = GridBagConstraints.NORTH;
        gbc.fill = GridBagConstraints.HORIZONTAL;

        root.add(carPanel);
        root.add(bikePanel);
        carPanel.add(carToggle, gbc);
        bikePanel.add(bikeToggle, gbc);
        carPanel.add(carSpinner, gbc);
        bikePanel.add(bikeSpinner, gbc);


        bikeSpinner.addChangeListener(e -> {
            SpinnerModel dateModel = bikeSpinner.getModel();
            if (dateModel instanceof SpinnerNumberModel) {
                controller.setBikeThreadPriority(((SpinnerNumberModel) dateModel
    ).getNumber().intValue());
            }
        });

        carSpinner.addChangeListener(e -> {
            SpinnerModel dateModel = carSpinner.getModel();
            if (dateModel instanceof SpinnerNumberModel) {
                controller.setCarThreadPriority(((SpinnerNumberModel) dateModel)
    .getNumber().intValue());
            }
        });

        bikeToggle.addActionListener(a ->
                controller.setBikeAIToggled(bikeToggle.isSelected()));

        carToggle.addActionListener(a ->
                controller.setCarAIToggled(carToggle.isSelected()));

    }

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

    @Override
    public void OnBikeThreadPriorityChanged(int priority) {
        bikeSpinner.setValue(priority);
    }

    @Override
    public void OnCarThreadPriorityChanged(int priority) {
        carSpinner.setValue(priority);
    }

    @Override
    public void OnCarAIToggled(boolean on) {
        carToggle.setSelected(on);
    }

    @Override
    public void OnBikeAIToggled(boolean on) {
        bikeToggle.setSelected(on);
    }
}
```