МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра вычислительной техники

Отчет по лабораторной работе №3

по дисциплине «Технология программирования»

Тема: «Классы-коллекции.»

Вариант №5

Выполнила: Проверил:

студентка группы АВТ-716 Михайленко Дмитрий

Качесова Кристина Васильевна Анатольевич

Новосибирск,

2019 г.

**Цель работы:**

1. Изучить особенности реализации классов-коллекций в Java.
2. Доработать программу, созданную в лабораторной работе № 2:
3. добавить генерируемым объектам понятия «время рождения» и «время жизни». Время рождения устанавливается в момент генерации объекта и по значению соответствует времени, прошедшему от начала симуляции. Время жизни – время, через которое объект должен исчезнуть, считая от времени рождения;
4. вынести установку параметров времени жизни объектов в пользовательский интерфейс. Для каждого типа объекта должно задаваться собственное время. Рекомендуется использовать текстовые поля, но следуют помнить о проверке на ввод некорректных данных;
5. добавить генерируемым объектам уникальные целочисленные идентификаторы (случайные числа), которые назначаются при генерации объекта. Для хранения сгенерированных идентификаторов используйте коллекцию удобную для поиска по варианту;
6. использовать коллекции по варианту. При генерации объекта происходит добавление его в коллекцию (в класс добавить поле идентификатора), а во вторую коллекцию: идентификаторы существующих объектов, в третью идентификатор + время рождения. При возникновении события по таймеру обойдите коллекцию и удалите все объекты, время жизни которых истекло, а также все данные во вспомогательных коллекциях;
7. добавьте в панель управления кпопку «Текущие объекты». По нажатию на эту кнопку появляется модальное диалоговое окно, содержащее список всех «живых» объектов на момент нажатия со временем их рождения (тип объекта, время рождения, идентификатор). В класс диалогового окна должна передаваться коллекция с хранением объектов по времени рождения. Типы коллекций задаются вариантом.

**Задание по варианту**

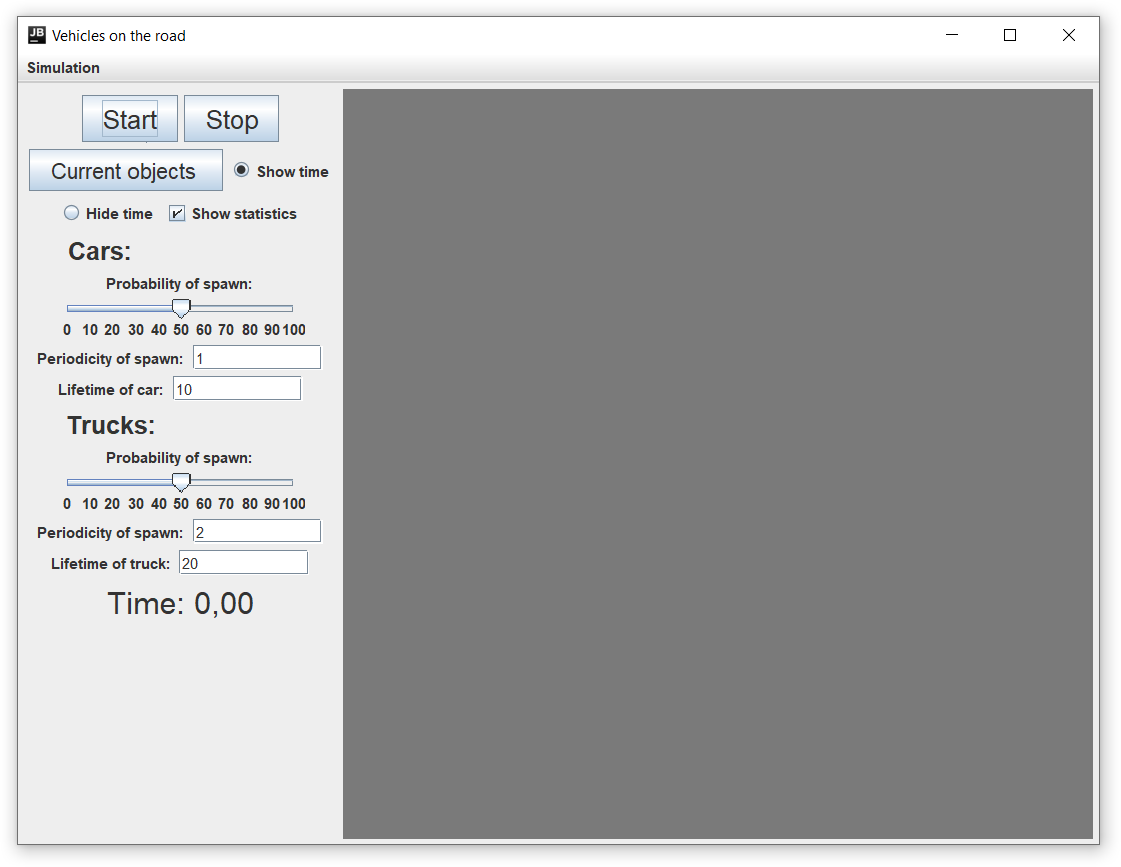
Коллекция для хранения объектов: ArrayList

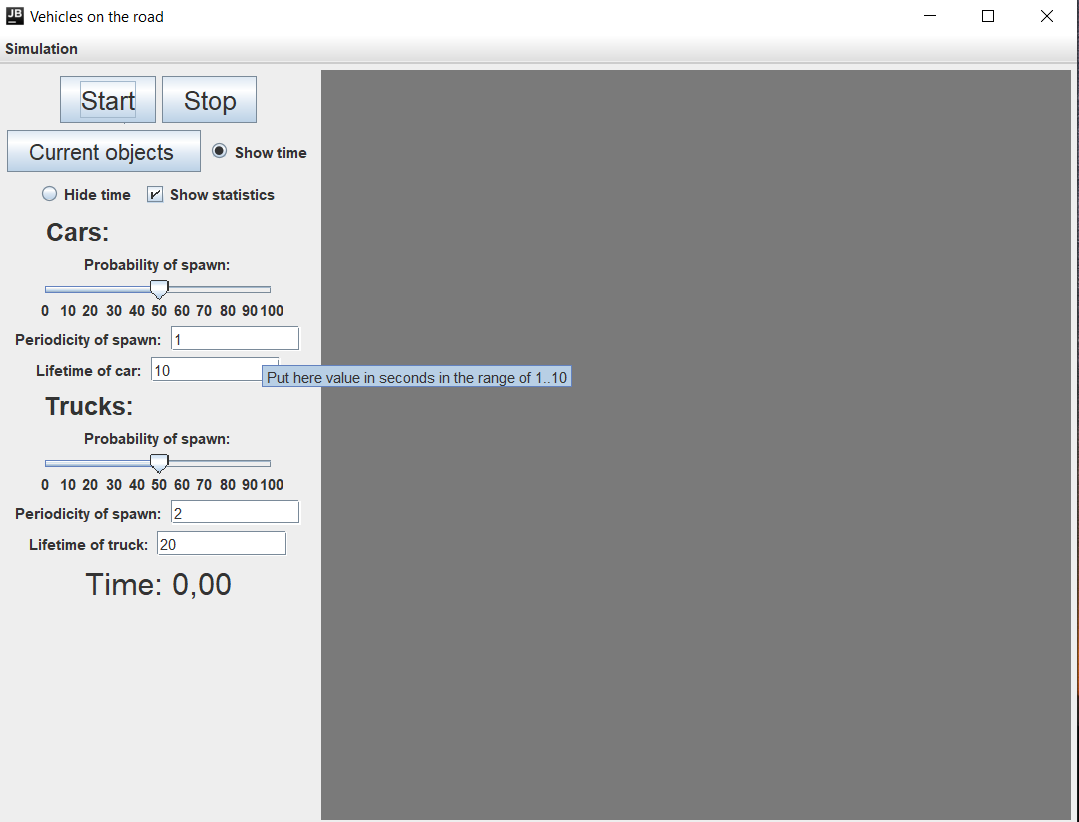
Коллекция для хранения и поиска уникальных идентификаторов: TreeSet

Коллекция для хранения времени рождения объектов: HashMap

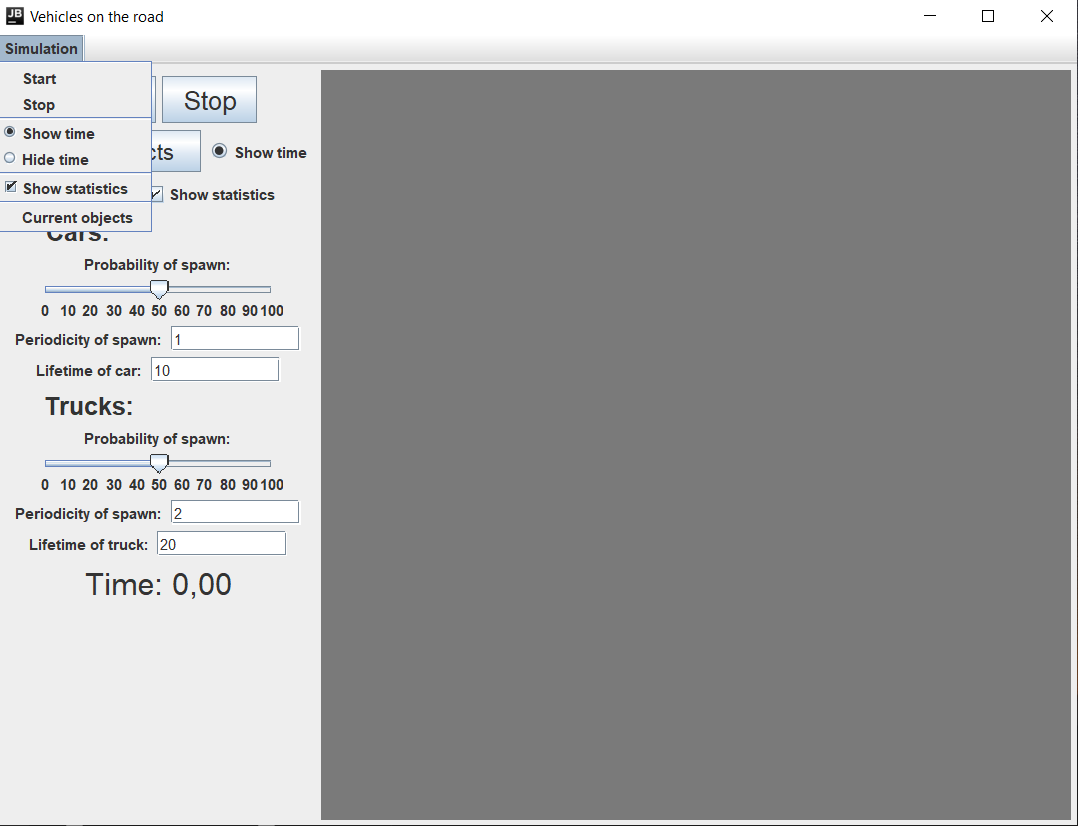
**Результаты работы программы**

Первое, что видит пользователь – окно программы, которое претерпело несколько изменений. Теперь в интерфейс добавлены два новых поля TextField для задания продолжительности жизни объектов разного типа (lifetime\_carField, lifetime\_truckField) содержащие ToolTipText, две метки типа Jlabel (lifetime\_car, lifetime\_truck), поясняющие назначение новых текстовых полей.

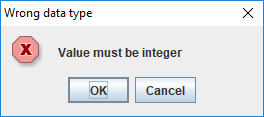
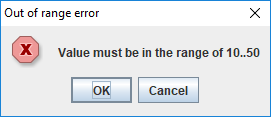




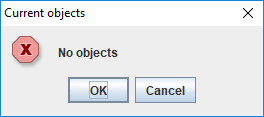
Кроме того добавлена новая кнопка “Current objects”, которая продублирована и во вложенном меню “Simulation”.



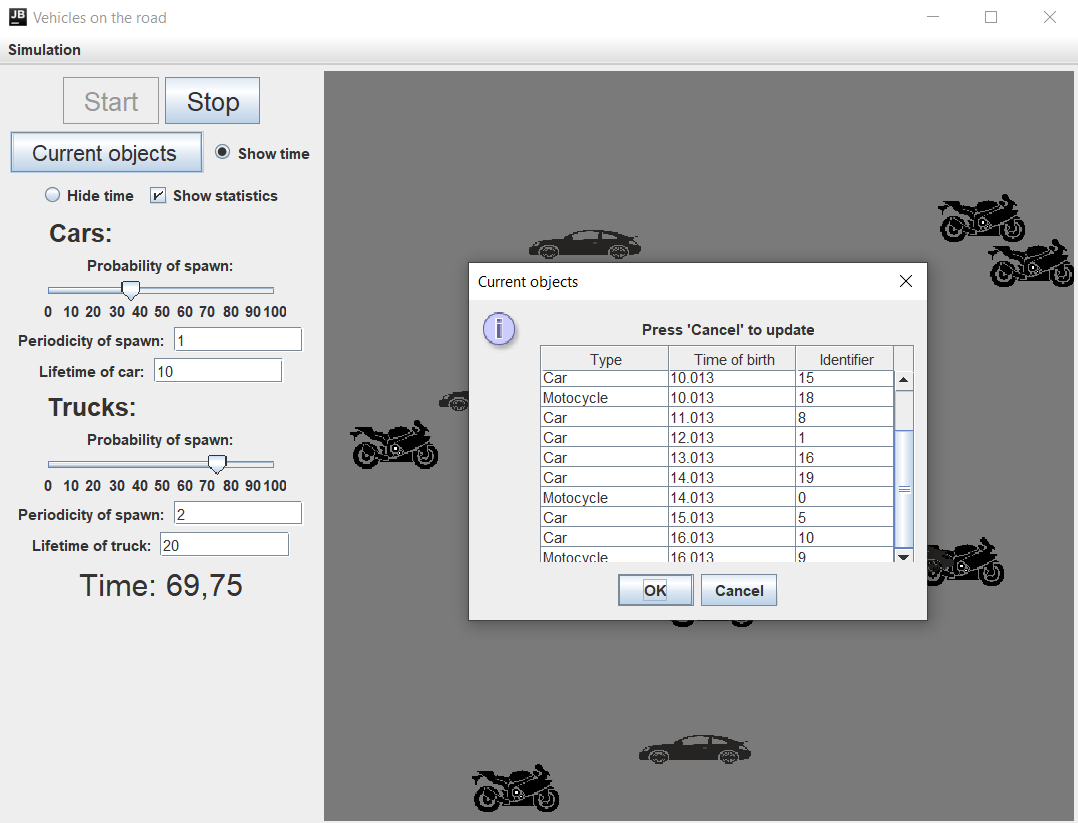
На новых текстовых полях стоит обработка исключений, если ввести значение за пределами допустимого диапазона, то пользователь одно из двух модальных окон (см. ниже), после чего будет выставлено значение по умолчанию.

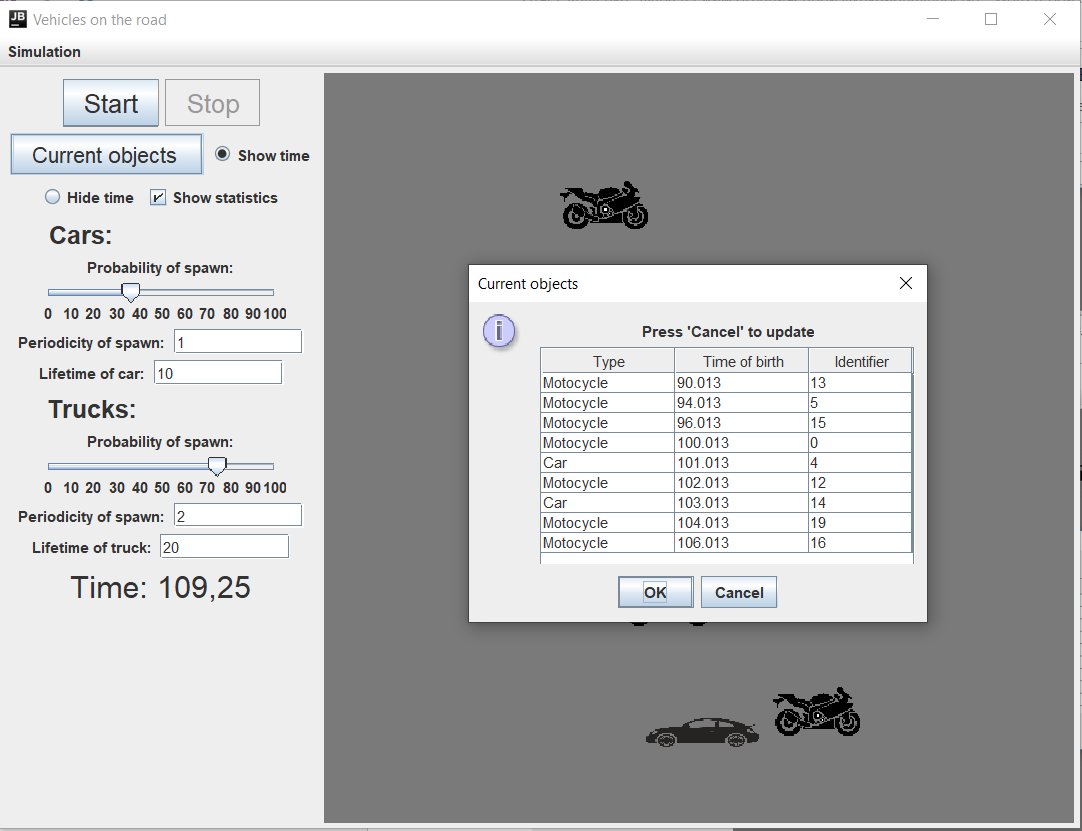
 

Если нажать на кнопку “Current objects” до начала симуляции, то пользователь увидит предупреждение:



Если нажать на кнопку во время симуляции, то пользователю будет предоставлена информация об объектах, находящихся на панели (не умерших). Данные отображаются в виде таблицы, которую можно обновить, нажав кнопку “Cancel” на модальном окне, в котором отображается данная таблица.



По завершении симуляции данные об объектах все еще остаются доступны для просмотра 

**Листинг программы:**

***Main.java***

public class Main {  
 public static void main(String[] args) {  
 Program program = new Program();  
 while (!program.isDone) {  
 program.run();  
 }  
 System.*exit*(0);  
 }  
  
}

***Program.java***

import javax.swing.\*;  
import java.awt.\*;  
import javax.swing.event.ChangeEvent;  
import javax.swing.event.ChangeListener;  
import java.awt.event.\*;  
  
public class Program extends JFrame {  
  
 private boolean isWorking;  
 private boolean statisticsIsVisible;  
 public boolean isDone;  
 private long startTime;  
 private long timePrev;  
 private float timeElapsed;  
  
 private JMenuBar menuBar;  
 private JMenuItem menuItemCurrentObjects;  
 private JMenuItem menuItemStart, menuItemStop;  
 private JCheckBoxMenuItem menuItemShowStatistics;  
 private JRadioButtonMenuItem menuItemShowTime, menuItemHideTime;  
  
 private JLabel timeTextLabel, cars, motocycles, probability\_car, probability\_motocycle, periodicity\_car, periodicity\_motocycle;  
 private JLabel lifetime\_car, lifetime\_motocycle;  
 private JButton buttonStart, buttonStop;  
 private JButton buttonCurrentObjects;  
 private ButtonGroup timeVisible;  
 private JRadioButton buttonShowTime, buttonHideTime;  
 private JCheckBox buttonShowStatistics;  
 private JTextField periodicity\_carField, probability\_motocycleField, lifetime\_carField, lifetime\_motocyclesField;  
  
 private JSlider sliderProbability\_car, sliderProbability\_motocycle;  
  
 private Habitat habitat;  
  
 public Program() {  
 isWorking = isDone = false;  
 statisticsIsVisible = true;  
 startTime = 0;  
 timePrev = System.*currentTimeMillis*();  
 timeElapsed = 0;  
  
 createGUI();  
 }  
  
 private void createGUI() {  
 setTitle("Vehicles on the road");  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 getContentPane().setLayout(new FlowLayout());  
  
 JPanel guiPanel = new JPanel();  
 guiPanel.setPreferredSize(new Dimension(250, 600));  
 add(guiPanel);  
  
 habitat = new Habitat();  
 add(habitat);  
  
 buttonStart = new JButton("Start");  
 buttonStart.setVisible(true);  
 buttonStart.setFont(buttonStart.getFont().deriveFont(21f));  
 buttonStart.setFont(buttonStart.getFont().deriveFont(Font.*PLAIN*));  
 guiPanel.add(buttonStart);  
  
 buttonStop = new JButton("Stop");  
 buttonStop.setVisible(true);  
 buttonStop.setFont(buttonStop.getFont().deriveFont(21f));  
 buttonStop.setFont(buttonStop.getFont().deriveFont(Font.*PLAIN*));  
 guiPanel.add(buttonStop);  
  
  
 buttonCurrentObjects = new JButton("Current objects ");  
 buttonCurrentObjects.setVisible(true);  
 buttonCurrentObjects.setFont(buttonCurrentObjects.getFont().deriveFont(18f));  
 buttonCurrentObjects.setFont(buttonCurrentObjects.getFont().deriveFont(Font.*PLAIN*));  
 guiPanel.add(buttonCurrentObjects);  
  
 buttonShowTime = new JRadioButton("Show time");  
 buttonShowTime.setVisible(true);  
 buttonShowTime.setSelected(true);  
 guiPanel.add(buttonShowTime);  
  
 buttonHideTime = new JRadioButton("Hide time");  
 buttonHideTime.setVisible(true);  
 guiPanel.add(buttonHideTime);  
  
 buttonShowStatistics = new JCheckBox("Show statistics");  
 buttonShowStatistics.setSelected(true);  
 guiPanel.add(buttonShowStatistics);  
  
 timeVisible = new ButtonGroup();  
 timeVisible.add(buttonShowTime);  
 timeVisible.add(buttonHideTime);  
  
 cars = new JLabel("Cars: ");  
 guiPanel.add(cars);  
 cars.setFont(cars.getFont().deriveFont(20f));  
  
 probability\_car = new JLabel("Probability of spawn: ");  
 guiPanel.add(probability\_car);  
  
 sliderProbability\_car = new JSlider(0, 100, 50);  
 sliderProbability\_car.setPaintLabels(true);  
 sliderProbability\_car.setMajorTickSpacing(10);  
 guiPanel.add(sliderProbability\_car);  
  
 periodicity\_car = new JLabel("Periodicity of spawn: ");  
 guiPanel.add(periodicity\_car);  
  
 periodicity\_carField = new JTextField(10);  
 periodicity\_carField.setText("1");  
 periodicity\_carField.setToolTipText("Put here value in seconds in the range of 1..10");  
 guiPanel.add(periodicity\_carField);  
  
 lifetime\_car = new JLabel("Lifetime of car: ");  
 guiPanel.add(lifetime\_car);  
  
 lifetime\_carField = new JTextField(10);  
 lifetime\_carField.setText("10");  
 lifetime\_carField.setToolTipText("Put here value in seconds in the range of 10..50");  
 guiPanel.add(lifetime\_carField);  
  
 motocycles = new JLabel("Trucks: ");  
 guiPanel.add(motocycles);  
 motocycles.setFont(motocycles.getFont().deriveFont(20f));  
  
 probability\_motocycle = new JLabel("Probability of spawn: ");  
 guiPanel.add(probability\_motocycle);  
  
 sliderProbability\_motocycle = new JSlider(0, 100, 50);  
 sliderProbability\_motocycle.setPaintLabels(true);  
 sliderProbability\_motocycle.setMajorTickSpacing(10);  
 guiPanel.add(sliderProbability\_motocycle);  
  
 periodicity\_motocycle = new JLabel("Periodicity of spawn: ");  
 guiPanel.add(periodicity\_motocycle);  
  
 probability\_motocycleField = new JTextField(10);  
 probability\_motocycleField .setText("2");  
 probability\_motocycleField.setToolTipText("Put here value in seconds in the range of 1..10");  
 guiPanel.add(probability\_motocycleField);  
  
 lifetime\_motocycle = new JLabel("Lifetime of truck: ");  
 guiPanel.add(lifetime\_motocycle);  
  
 lifetime\_motocyclesField = new JTextField(10);  
 lifetime\_motocyclesField.setText("20");  
 lifetime\_motocyclesField.setToolTipText("Put here value in seconds in the range of 10..50");  
 guiPanel.add(lifetime\_motocyclesField);  
  
 timeTextLabel = new JLabel("Time: 0,00");  
 timeTextLabel.setVisible(true);  
 timeTextLabel.setFont(timeTextLabel.getFont().deriveFont(25f));  
 timeTextLabel.setFont(timeTextLabel.getFont().deriveFont(Font.*PLAIN*));  
 guiPanel.add(timeTextLabel);  
  
 menuBar = new JMenuBar();  
 menuBar.add(createSimulationMenu());  
 setJMenuBar(menuBar);  
  
 setComponentOrientation(ComponentOrientation.*LEFT\_TO\_RIGHT*);  
 pack();  
 setVisible(true);  
  
 fields();  
 slides();  
 buttons();  
 keys();  
  
 }  
  
 private void start() {  
 if (!isWorking) {  
 buttonStart.setEnabled(false);  
 menuItemStart.setEnabled(false);  
 buttonStop.setEnabled(true);  
 menuItemStop.setEnabled(true);  
 startTime = System.*currentTimeMillis*();  
 habitat.start();  
 isWorking = true;  
 System.*out*.println("Begin");  
 }  
 }  
  
 private void end() {  
 if (isWorking) {  
 buttonStop.setEnabled(false);  
 menuItemStop.setEnabled(false);  
 buttonStart.setEnabled(true);  
 menuItemStart.setEnabled(true);  
 isWorking = false;  
 long stopTime = System.*currentTimeMillis*() - startTime;  
 isWorking = habitat.fin(System.*currentTimeMillis*() - startTime, statisticsIsVisible);  
 if (isWorking) {  
 buttonStart.setEnabled(false);  
 menuItemStart.setEnabled(false);  
 buttonStop.setEnabled(true);  
 menuItemStop.setEnabled(true);  
 }  
 startTime += (System.*currentTimeMillis*() - startTime) - stopTime;  
 }  
 }  
  
 private void updateTime(long time) {  
 timeTextLabel.setText("Time: " + String.*format*("%.02f", time / 1000f));  
 }  
  
 public void run() {  
 float timeStep = 1 / 50f;  
 long timeCurrent = System.*currentTimeMillis*();  
 timeElapsed += (timeCurrent - timePrev) / 1000.f;  
 timePrev = timeCurrent;  
  
 while (timeElapsed >= timeStep) {  
 if (isWorking) {  
 long time = System.*currentTimeMillis*() - startTime;  
 habitat.update(time);  
 habitat.visitingRound(timeStep\*1000);  
 updateTime(time);  
 }  
 timeElapsed -= timeStep;  
 }  
 }  
  
 private JMenu createSimulationMenu() {  
 JMenu simulationMenu = new JMenu("Simulation");  
  
 menuItemStart = new JMenuItem("Start");  
 menuItemStop = new JMenuItem("Stop");  
  
 menuItemShowStatistics = new JCheckBoxMenuItem("Show statistics");  
 menuItemShowStatistics.setSelected(buttonShowStatistics.isSelected());  
  
 menuItemShowTime = new JRadioButtonMenuItem("Show time");  
 menuItemHideTime = new JRadioButtonMenuItem("Hide time");  
  
 ButtonGroup bg1 = new ButtonGroup();  
 menuItemShowTime.setSelected(buttonShowTime.isSelected());  
 menuItemHideTime.setSelected(buttonHideTime.isSelected());  
 bg1.add(menuItemShowTime);  
 bg1.add(menuItemHideTime);  
 menuItemCurrentObjects=new JMenuItem("Current objects");  
  
 simulationMenu.add(menuItemStart);  
 simulationMenu.add(menuItemStop);  
 simulationMenu.add(new JSeparator());  
 simulationMenu.add(menuItemShowTime);  
 simulationMenu.add(menuItemHideTime);  
 simulationMenu.add(new JSeparator());  
 simulationMenu.add(menuItemShowStatistics);  
 simulationMenu.add(new JSeparator());  
 simulationMenu.add(menuItemCurrentObjects);  
  
 return simulationMenu;  
 }  
  
 private void fields() {  
 ActionListener actionListener\_periodicity\_carField = new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 long value\_periodicity\_car;  
 try {  
 value\_periodicity\_car = Integer.*parseInt*(periodicity\_carField.getText());  
 } catch (NumberFormatException exp) {  
 JOptionPane.*showConfirmDialog*(null, "Value must be integer",  
 "Wrong data type",  
 JOptionPane.*CANCEL\_OPTION*, JOptionPane.*ERROR\_MESSAGE*);  
 value\_periodicity\_car = 1;  
 periodicity\_carField.setText("1");  
 }  
 if (value\_periodicity\_car < 1 || value\_periodicity\_car > 10) {  
 JOptionPane.*showConfirmDialog*(null, "Value must be in the range of 1..10",  
 "Out of range error",  
 JOptionPane.*CANCEL\_OPTION*, JOptionPane.*ERROR\_MESSAGE*);  
 value\_periodicity\_car = 1;  
 periodicity\_carField.setText("1");  
 } else {  
 habitat.N1 = value\_periodicity\_car \* 1000;  
 }  
  
 }  
  
 };  
 ActionListener actionListener\_probability\_truckField = new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 long value\_probability\_motocycle;  
 try {  
 value\_probability\_motocycle = Integer.*parseInt*(probability\_motocycleField.getText());  
 } catch (NumberFormatException exp) {  
 JOptionPane.*showConfirmDialog*(null, "Value must be integer",  
 "Wrong data type",  
 JOptionPane.*CANCEL\_OPTION*, JOptionPane.*ERROR\_MESSAGE*);  
 //probability\_truckField.setText(null);  
 value\_probability\_motocycle = 2;  
 probability\_motocycleField.setText("2");  
 }  
 if (value\_probability\_motocycle < 1 || value\_probability\_motocycle > 10) {  
 JOptionPane.*showConfirmDialog*(null, "Value must be in the range of 1..10",  
 "Out of range error",  
 JOptionPane.*CANCEL\_OPTION*, JOptionPane.*ERROR\_MESSAGE*);  
 value\_probability\_motocycle = 2;  
 probability\_motocycleField.setText("2");  
 } else {  
 habitat.N2 = value\_probability\_motocycle \* 1000;  
 }  
 }  
  
 };  
 ActionListener actionListener\_lifetime\_carField = new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 long value\_lifetime\_car;  
 try {  
 value\_lifetime\_car = Integer.*parseInt*(lifetime\_carField.getText());  
 } catch (NumberFormatException exp) {  
 JOptionPane.*showConfirmDialog*(null, "Value must be integer",  
 "Wrong data type",  
 JOptionPane.*CANCEL\_OPTION*, JOptionPane.*ERROR\_MESSAGE*);  
 value\_lifetime\_car = 10;  
 lifetime\_carField.setText("10");  
 }  
 if (value\_lifetime\_car < 10 || value\_lifetime\_car > 50) {  
 JOptionPane.*showConfirmDialog*(null, "Value must be in the range of 10..50",  
 "Out of range error",  
 JOptionPane.*CANCEL\_OPTION*, JOptionPane.*ERROR\_MESSAGE*);  
 lifetime\_carField.setText("10");  
 } else {  
 habitat.lifetime\_car = value\_lifetime\_car \* 1000;  
 }  
 }  
  
 };  
 ActionListener actionListener\_lifetime\_truckField = new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 long value\_lifetime\_motocycle;  
 try {  
 value\_lifetime\_motocycle = Integer.*parseInt*(lifetime\_motocyclesField.getText());  
 } catch (NumberFormatException exp) {  
 JOptionPane.*showConfirmDialog*(null, "Value must be integer",  
 "Wrong data type",  
 JOptionPane.*CANCEL\_OPTION*, JOptionPane.*ERROR\_MESSAGE*);  
 //probability\_truckField.setText(null);  
 value\_lifetime\_motocycle = 20;  
 lifetime\_motocyclesField.setText("20");  
 }  
 if (value\_lifetime\_motocycle < 10 || value\_lifetime\_motocycle > 50) {  
 JOptionPane.*showConfirmDialog*(null, "Value must be in the range of 10..50",  
 "Out of range error",  
 JOptionPane.*CANCEL\_OPTION*, JOptionPane.*ERROR\_MESSAGE*);  
 lifetime\_motocyclesField.setText("20");  
 } else {  
 habitat.lifetime\_motocycle = value\_lifetime\_motocycle \* 1000;  
 }  
 }  
  
 };  
  
 lifetime\_motocyclesField.addActionListener(actionListener\_lifetime\_truckField);  
 lifetime\_carField.addActionListener(actionListener\_lifetime\_carField);  
 periodicity\_carField.addActionListener(actionListener\_periodicity\_carField);  
 lifetime\_motocyclesField.addActionListener(actionListener\_probability\_truckField);  
 }  
  
 private void slides() {  
 ChangeListener changeListener = new ChangeListener() {  
 public void stateChanged(ChangeEvent e) {  
 int value1 = sliderProbability\_car.getValue();  
 habitat.P1 = value1 / 100f;  
 //int value = ((JSlider) e.getSource()).getValue();  
 System.*out*.println("P1 value " + value1 + "%");  
 int value2 = sliderProbability\_motocycle.getValue();  
 habitat.P2 = value2 / 100f;  
 //int value = ((JSlider) e.getSource()).getValue();  
 System.*out*.println("P2 value " + value2 + "%");  
 }  
 };  
 sliderProbability\_car.addChangeListener(changeListener);  
 sliderProbability\_motocycle.addChangeListener(changeListener);  
  
 }  
  
 private void buttons() {  
 ActionListener actionListenerB = new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 if (e.getSource() == buttonStart || e.getSource() == menuItemStart) {  
 start();  
 }  
 if (e.getSource() == buttonStop || e.getSource() == menuItemStop) {  
 end();  
 }  
 if (e.getSource() == buttonShowTime || e.getSource() == menuItemShowTime) {  
 buttonShowTime.setSelected(true);  
 menuItemShowTime.setSelected(true);  
 timeTextLabel.setVisible(true);  
 }  
 if (e.getSource() == buttonHideTime || e.getSource() == menuItemHideTime) {  
 buttonHideTime.setSelected(true);  
 menuItemHideTime.setSelected(true);  
 timeTextLabel.setVisible(false);  
 }  
  
 if (e.getSource() == buttonShowStatistics) {  
 if (buttonShowStatistics.isSelected()) {  
 buttonShowStatistics.setSelected(true);  
 menuItemShowStatistics.setSelected(true);  
 statisticsIsVisible = true;  
 } else {  
 buttonShowStatistics.setSelected(false);  
 menuItemShowStatistics.setSelected(false);  
 statisticsIsVisible = false;  
 }  
 }  
  
 if (e.getSource() == menuItemShowStatistics) {  
 if (menuItemShowStatistics.isSelected()) {  
 buttonShowStatistics.setSelected(true);  
 menuItemShowStatistics.setSelected(true);  
 statisticsIsVisible = true;  
 } else {  
 buttonShowStatistics.setSelected(false);  
 menuItemShowStatistics.setSelected(false);  
 statisticsIsVisible = false;  
 }  
  
 }  
  
 if (e.getSource() == buttonCurrentObjects || e.getSource() == menuItemCurrentObjects) {  
 habitat.showCurrentObjects();  
 }  
  
 }  
 };  
  
  
 menuItemStart.addActionListener(actionListenerB);  
 menuItemStop.addActionListener(actionListenerB);  
 menuItemShowTime.addActionListener(actionListenerB);  
 menuItemHideTime.addActionListener(actionListenerB);  
 menuItemShowStatistics.addActionListener(actionListenerB);  
 menuItemCurrentObjects.addActionListener(actionListenerB);  
  
 buttonStart.addActionListener(actionListenerB);  
 buttonStop.addActionListener(actionListenerB);  
 buttonShowTime.addActionListener(actionListenerB);  
 buttonHideTime.addActionListener(actionListenerB);  
 buttonShowStatistics.addActionListener(actionListenerB);  
 buttonCurrentObjects.addActionListener(actionListenerB);  
 }  
  
 private void keys() {  
 habitat.getInputMap(JComponent.*WHEN\_IN\_FOCUSED\_WINDOW*).put(KeyStroke.*getKeyStroke*("B"), "menuItemStart");  
 habitat.getActionMap().put("menuItemStart", new AbstractAction() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 start();  
 }  
 });  
 habitat.getInputMap(JComponent.*WHEN\_IN\_FOCUSED\_WINDOW*).put(KeyStroke.*getKeyStroke*("E"), "finish");  
 habitat.getActionMap().put("finish", new AbstractAction() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 end();  
 }  
 });  
 habitat.getInputMap(JComponent.*WHEN\_IN\_FOCUSED\_WINDOW*).put(KeyStroke.*getKeyStroke*("T"), "time");  
 habitat.getActionMap().put("time", new AbstractAction() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 if (timeTextLabel.isVisible()) {  
 buttonHideTime.setSelected(true);  
 menuItemHideTime.setSelected(true);  
 timeTextLabel.setVisible(false);  
 } else {  
 buttonShowTime.setSelected(true);  
 menuItemShowTime.setSelected(true);  
 timeTextLabel.setVisible(true);  
 }  
 }  
 });  
 habitat.getInputMap(JComponent.*WHEN\_IN\_FOCUSED\_WINDOW*).put(KeyStroke.*getKeyStroke*("ESCAPE"), "exit");  
 habitat.getActionMap().put("exit", new AbstractAction() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 isDone = true;  
 }  
 });  
 }  
}

***Habitat.java***

import javax.swing.\*;  
import java.awt.\*;  
import java.util.HashMap;  
import java.util.TreeSet;  
import java.util.ArrayList;  
  
  
public class Habitat extends JPanel {  
  
 private final int width = 600, height = 600;  
 private final int size = 5;  
  
 public long N1, N2;  
 public float P1, P2;  
 public float lifetime\_car, lifetime\_motocycle;  
 private int countTime1, countTime2;  
 private int count1, count2;  
 public ArrayList<Transport> myObjects;  
 private TreeSet<Integer> identifiers;  
 private HashMap<Integer, Float> identifiers\_lifetime;  
  
  
 public Habitat() {  
 N1 = 1000;  
 N2 = 2000;  
 lifetime\_car = 10000;  
 lifetime\_motocycle = 20000;  
 P1 = P2 = 0.5f;  
 count1 = count2 = 0;  
 countTime1 = countTime2 = 0;  
 myObjects = new ArrayList<Transport>();  
 identifiers = new TreeSet<Integer>();  
 identifiers\_lifetime = new HashMap<Integer, Float>();  
  
 setPreferredSize(new Dimension(width, height));  
 setLayout(null);  
 setBackground(new Color(122, 122, 122));  
 }  
  
 private int setPos(int len) {  
 return (int) (Math.*random*() \* (len - 50));  
 }  
  
 private void put\_in(Transport obj) {  
 int index = index();  
  
 obj.identifier = index;  
  
 myObjects.add(obj);  
 identifiers.add(index);  
 identifiers\_lifetime.put(index, obj.lifetime);  
  
 }  
  
 private int index() {  
 int index = (int) (Math.*random*() \* (count1 + count2 + 10));  
 while (identifiers.contains(index))  
 index = (int) (Math.*random*() \* (count1 + count2 + 10));  
 return index;  
 }  
  
 private void check() {  
 for (int i = 0; i < myObjects.size(); i++)  
 System.*out*.println(myObjects.get(i));  
 }  
  
 public void update(long time) {  
 if (time - (countTime1 \* N1) >= N1) {  
 if (Math.*random*() <= P1) {  
 Car car = new Car(setPos(width), setPos(height));  
 add(car);  
 car.time\_of\_birth = time;  
 car.lifetime = lifetime\_car;  
 put\_in(car);  
 validate();  
 repaint();  
 count1++;  
 }  
 countTime1++;  
 }  
  
 if (time - (countTime2 \* N2) >= N2) {  
 if (Math.*random*() <= P2) {  
 Motocycle motocycle = new Motocycle(setPos(width), setPos(height));  
 add(motocycle);  
 motocycle.time\_of\_birth = time;  
 motocycle.lifetime = lifetime\_motocycle;  
 put\_in(motocycle);  
 validate();  
 repaint();  
 count2++;  
 }  
 countTime2++;  
 }  
  
 }  
  
 public void showCurrentObjects() {  
 int upd\_table=JOptionPane.*CANCEL\_OPTION*;  
 if (myObjects.size() != 0) {  
 while( upd\_table==JOptionPane.*CANCEL\_OPTION*)  
 { Table table = new Table(myObjects);  
 upd\_table=JOptionPane.*showConfirmDialog*(this,  
 table.createTable(),  
 "Current objects",  
 JOptionPane.*OK\_CANCEL\_OPTION*, JOptionPane.*INFORMATION\_MESSAGE*);  
 }  
 } else  
 JOptionPane.*showConfirmDialog*(null, "No objects",  
 "Current objects",  
 JOptionPane.*CANCEL\_OPTION*, JOptionPane.*ERROR\_MESSAGE*);  
 }  
  
  
 public void visitingRound(float timeStep) {  
 for (int i = 0; i < myObjects.size(); i++) {  
 Transport obj = myObjects.get(i);  
 obj.lifetime -= timeStep;  
 if (obj.lifetime <= 0)  
 delete(obj, i);  
 else  
 myObjects.set(i, obj);  
 }  
 }  
  
 private void delete(Transport obj, int i) {  
 if(obj.name=="Car") count1--;  
 else count2--;  
  
 remove(obj);  
 validate();  
 repaint();  
  
 identifiers.remove(obj.identifier);  
 identifiers\_lifetime.remove(obj.identifier);  
 myObjects.remove(i);  
 }  
  
 public boolean fin(long time, boolean statisticsIsVisible) {  
 check();  
 if (statisticsIsVisible) {  
 int selectedValue = JOptionPane.*showConfirmDialog*(this,  
 String.*format*("<html>Cars: %d<br> Motocycles: %d<br>Time: %.02f</html>", count1, count2, time / 1000.f),  
 "Statistics",  
 JOptionPane.*OK\_CANCEL\_OPTION*, JOptionPane.*INFORMATION\_MESSAGE*);  
 if (selectedValue == JOptionPane.*CANCEL\_OPTION*) {  
 return true;  
 }  
 }  
 return false;  
  
 }  
  
 public void start() {  
 count1 = count2 = 0;  
 countTime1 = countTime2 = 0;  
 removeAll();  
 setLayout(null);  
 revalidate();  
 repaint();  
 }  
}

***IBehaviour.java***

public interface IBehaviour {  
 void move();  
  
 void setx(float x);  
  
 void sety(float y);  
  
 float getx();  
  
 float gety();  
}

***Table.java***

import javax.swing.\*;  
import javax.swing.table.DefaultTableModel;  
import java.awt.\*;  
import java.util.ArrayList;  
  
  
public class Table extends DefaultTableModel {  
 private String[][] data;  
 private String[] columnNames;  
 private ArrayList<Transport> copyMyObjects;  
  
  
 public Table(ArrayList myObjects) {  
 copyMyObjects = (ArrayList<Transport>) myObjects.clone();  
 columnNames = new String[3];  
 data = new String[myObjects.size()][3];  
  
 }  
  
 public JPanel createTable() {  
  
 int width = 300, height = 200;  
  
 JPanel tablePanel = new JPanel();  
 tablePanel.setPreferredSize(new Dimension(width, height));  
  
 columnNames[0] = "Type";  
 columnNames[1] = "Time of birth";  
 columnNames[2] = "Identifier";  
  
  
 for (int i = 0; i < copyMyObjects.size(); i++) {  
 Transport obj = copyMyObjects.get(i);  
 data[i][0] = obj.name;  
 data[i][1] = Float.*toString*(obj.time\_of\_birth / 1000f);  
 data[i][2] = Integer.*toString*(obj.identifier);  
 }  
  
 JLabel text = new JLabel("Press 'Cancel' to update");  
 tablePanel.add(text);  
  
 JTable table = new JTable(data, columnNames);  
 table.setAutoResizeMode(JTable.*AUTO\_RESIZE\_OFF*);  
 table.getColumnModel().getColumn(0).setPreferredWidth(100);  
 table.getColumnModel().getColumn(1).setPreferredWidth(100);  
 table.setAutoResizeMode(JTable.*AUTO\_RESIZE\_LAST\_COLUMN*);  
  
  
 table.setFillsViewportHeight(true);  
 JScrollPane scrollPane = new JScrollPane(table);  
 scrollPane.setPreferredSize(new Dimension(width, height-20));  
  
 tablePanel.add(scrollPane);  
  
 return tablePanel;  
 }  
  
}

***Transport.java***

import javax.swing.\*;  
import java.awt.\*;  
  
public abstract class Transport extends JLabel implements IBehaviour {  
 private float x, y;  
 public long time\_of\_birth;  
 public float lifetime;  
 public int identifier;  
 public String name;  
  
 public Transport(float x, float y) {  
 this.x = x;  
 this.y = y;  
 this.setBounds((int) this.x, (int) this.y, 90 , 65 );  
 this.setVisible(true);  
 }  
  
 @Override  
 public void move() {  
  
 }  
  
 @Override  
 public float getx() {  
 return 0;  
 }  
  
 @Override  
 public float gety() {  
 return 0;  
 }  
  
 @Override  
 public void setx(float x) {  
  
 }  
  
 @Override  
 public void sety(float y) {  
  
 }  
}

***Car.java***

import javax.imageio.ImageIO;  
import javax.swing.ImageIcon;  
import java.awt.\*;  
import java.io.File;  
import java.awt.image.BufferedImage;  
  
public class Car extends Transport {  
  
 private static final BufferedImage *image*;  
  
 static {  
 BufferedImage img = null;  
 try {  
 img = ImageIO.*read*(new File("car.jpg"));  
 } catch (Exception e) {  
 System.*out*.println("Image load error!");  
 }  
 *image* = img;  
 }  
  
 public Car(float x, float y) {  
 super(x, y);  
 name="Car";  
 Image img = *image*.getScaledInstance(this.getWidth(), this.getHeight(), Image.*SCALE\_DEFAULT*);  
 this.setIcon(new ImageIcon(img));  
 System.*out*.println("Car: (" + x + ";" + y + ") ");  
 }  
}

***Motocycle.java***

import javax.imageio.ImageIO;  
import javax.swing.ImageIcon;  
import java.awt.\*;  
import java.io.File;  
import java.awt.image.BufferedImage;  
  
 public class Motocycle extends Transport {  
  
 private static final BufferedImage *image*;  
  
 static {  
 BufferedImage img = null;  
 try {  
 img = ImageIO.*read*(new File("moto.png"));  
 } catch (Exception e) {  
 System.*out*.println("Image load error!");  
 }  
 *image* = img;  
 }  
  
 public Motocycle(float x, float y) {  
 super(x, y);  
 name="Motocycle";  
 Image img = *image*.getScaledInstance(this.getWidth(), this.getHeight(), Image.*SCALE\_DEFAULT*);  
 this.setIcon(new ImageIcon(img));  
 System.*out*.println("Motocycle: (" + x + ";" + y + ") ");  
 }  
  
 }