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 №1

по дисциплине «Технология программирования»

Тема: «Приложения с использованием потоков ввода-вывода»

Выполнила: Проверил:

студентка группы АВТ-716 Михайленко Дмитрий

Качесова Кристина Васильевна Анатольевич

Новосибирск,

2019 г.

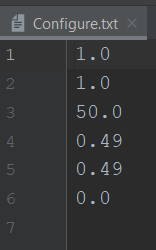
#### Практические задания

1. Изучить особенности реализации системы ввода-вывода в Java.
2. Доработать программу, созданную в лабораторной работе № 4:

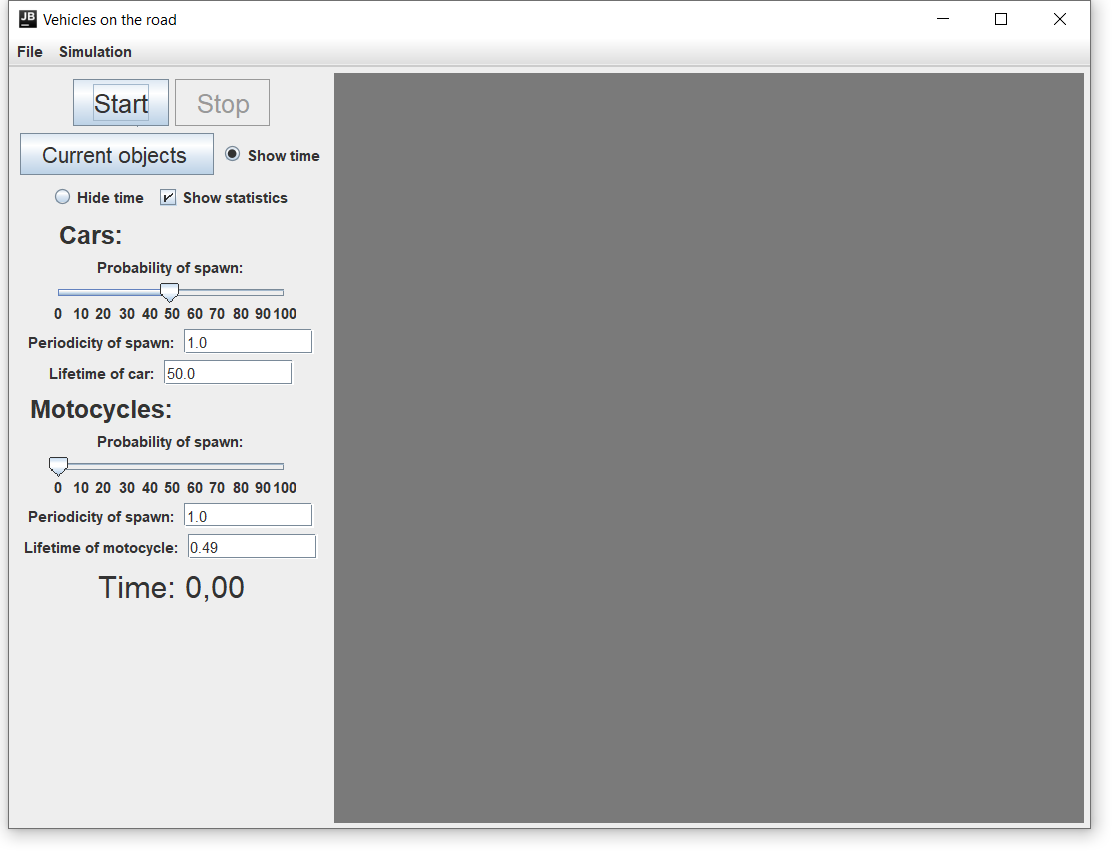
* для передачи команды в основное окно программы использовать каналы ввода-вывода;
* создать конфигурационный файл для программы. В конфигурационный файл должны сохраняться все настройки симуляции, т.е. все данные и состояния, которые задаются в панели управления программы. Конфигурационный файл должен читаться при запуске программы и записываться при выходе. Формат файла текстовый;
* добавить в главное меню пункты «Загрузить» и «Сохранить». Команда «Сохранить» вызывает сериализацию всех «живых» объектов в ней. Команда «Загрузить» останавливает текущую симуляцию (если симуляция запущена) и загружает объекты из выбранного файла. Не забудьте скорректировать время рождения объектов. После открытия симуляцию можно запустить, загруженные объекты должны вести себя естественно;
* использовать стандартные файловые диалоги.

**Результаты работы программы:**

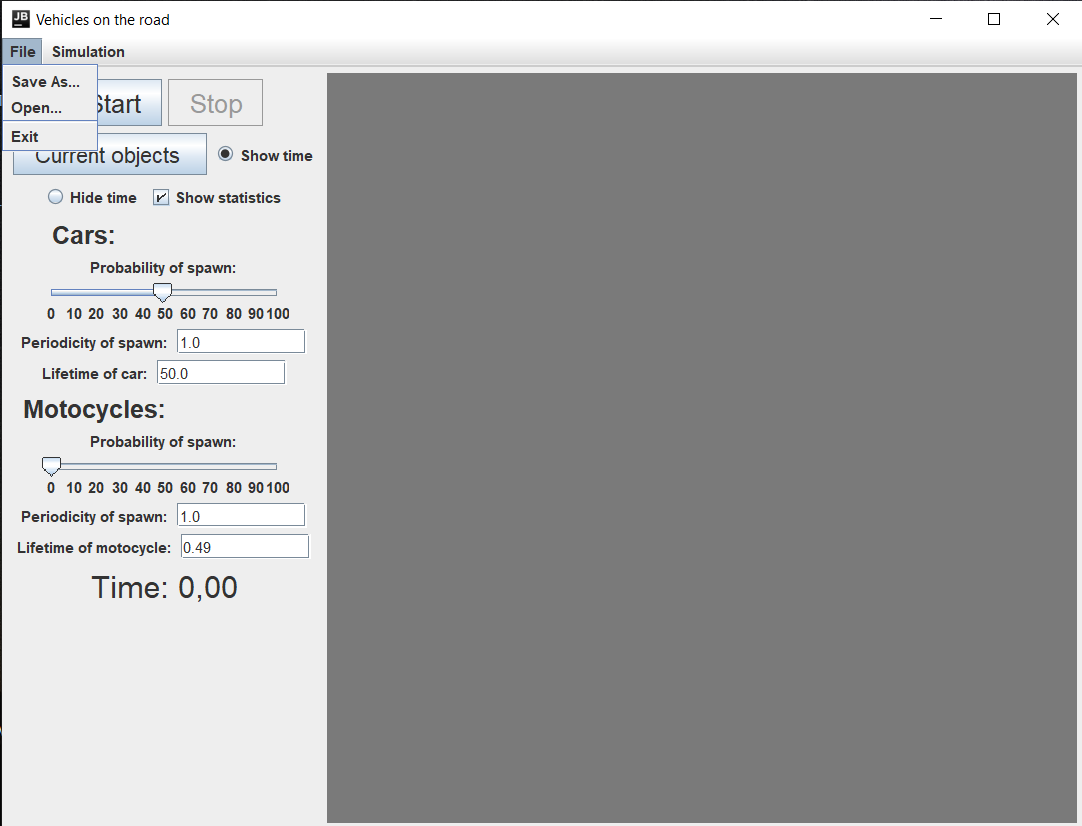
Параметры для создания окна считываются из конфигурационного файла, содержимое которого выглядит следующим образом:



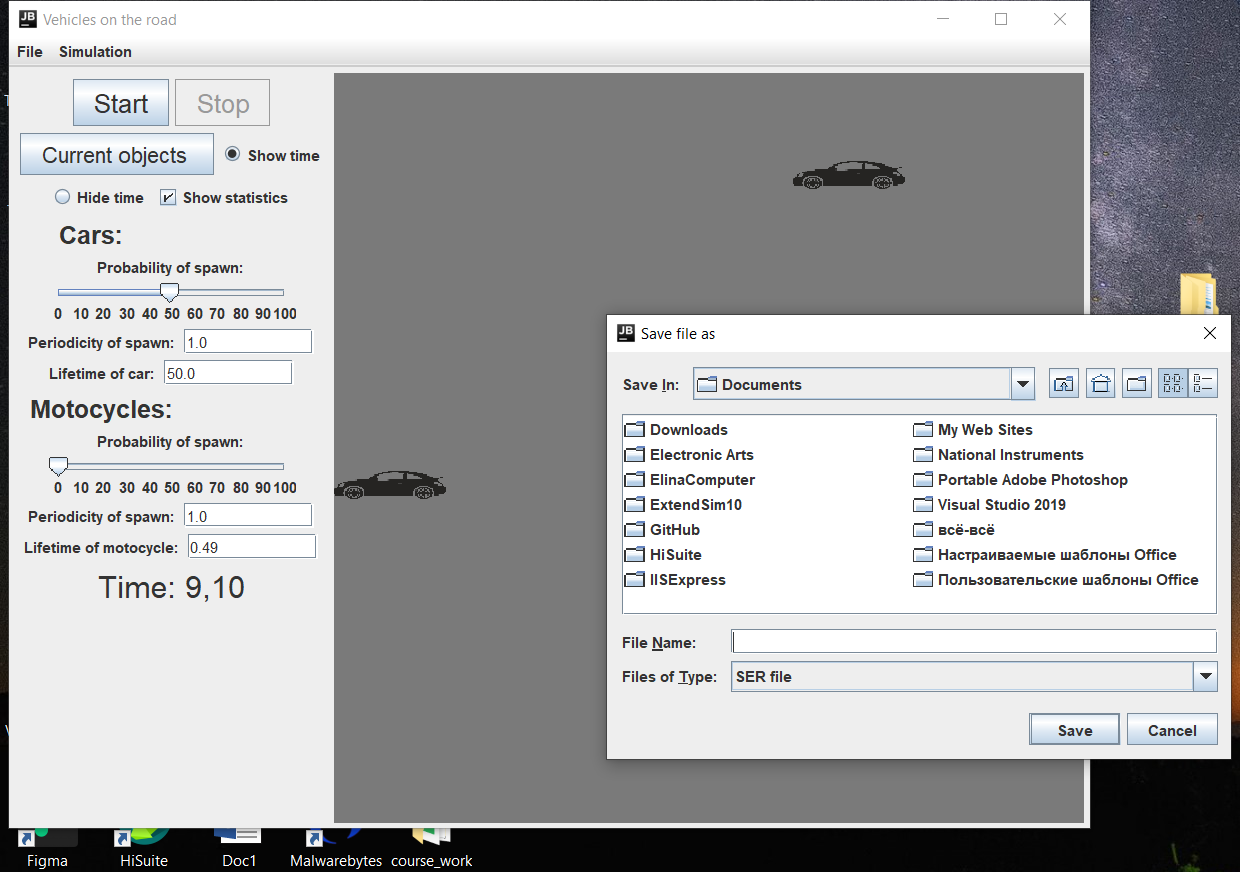
При заданных параметрах окно приложения выглядит так:



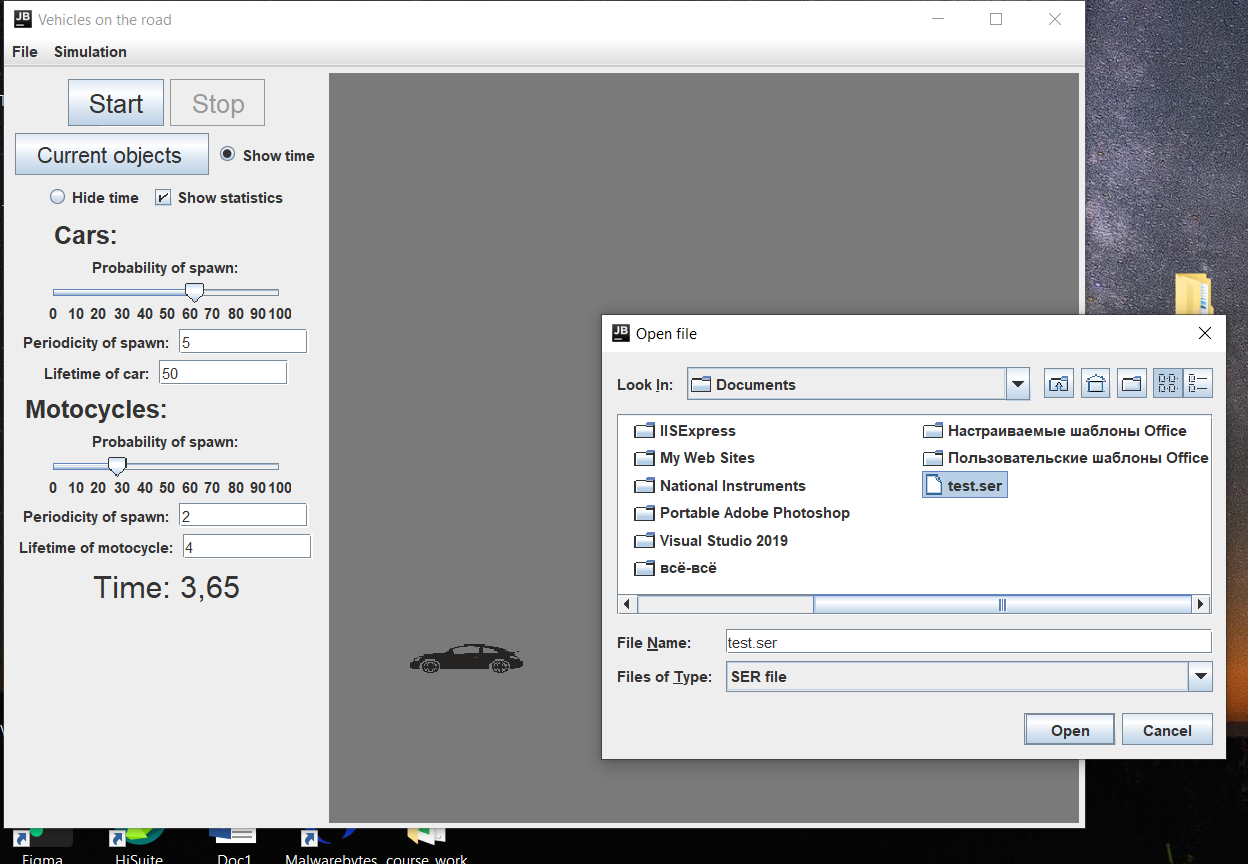
Во вложенном меню вкладки «Файл» пользователю предлагается сохранить симуляцию или же выбрать уже существующую, кроме того можно закрыть приложение.



Сохранение:

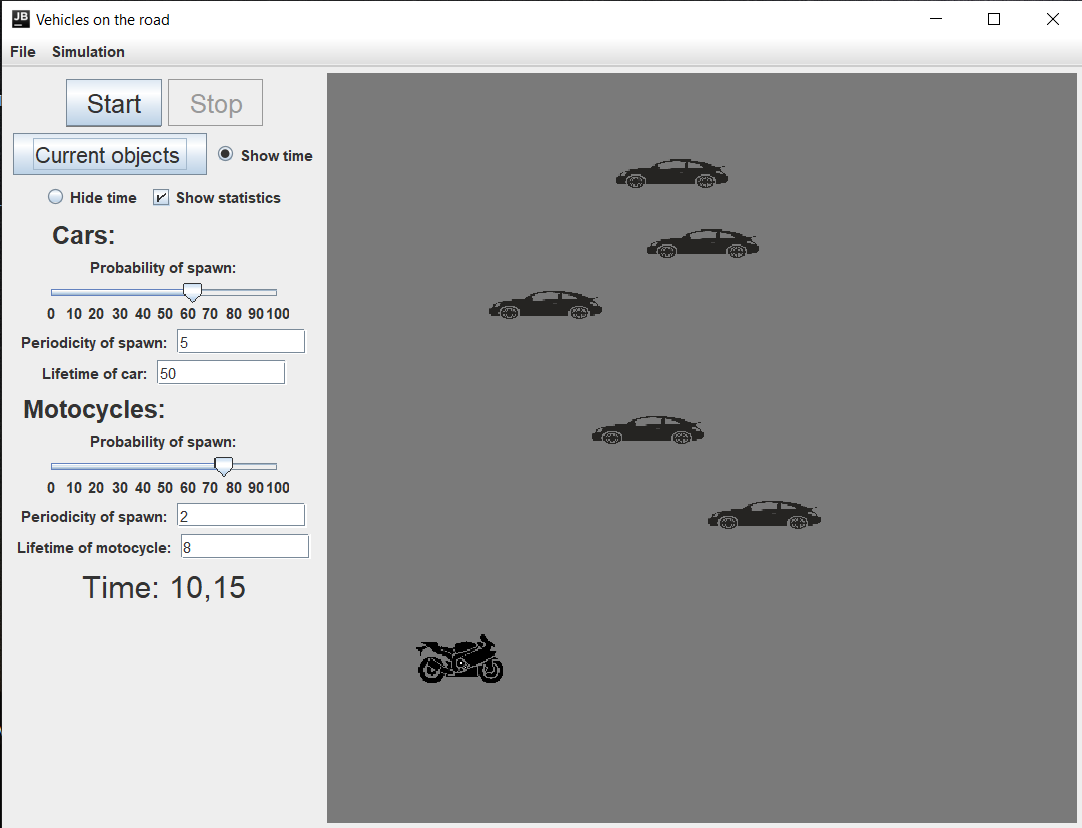


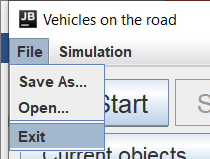
Открытие:



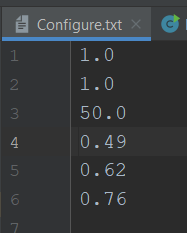
После открытия файла симуляции, список объектов очищается, после чего заполняется объектами из файла при помощи сериализации.

Если во время работы были изменены настройки, то при выходе из программы они сохранятся:

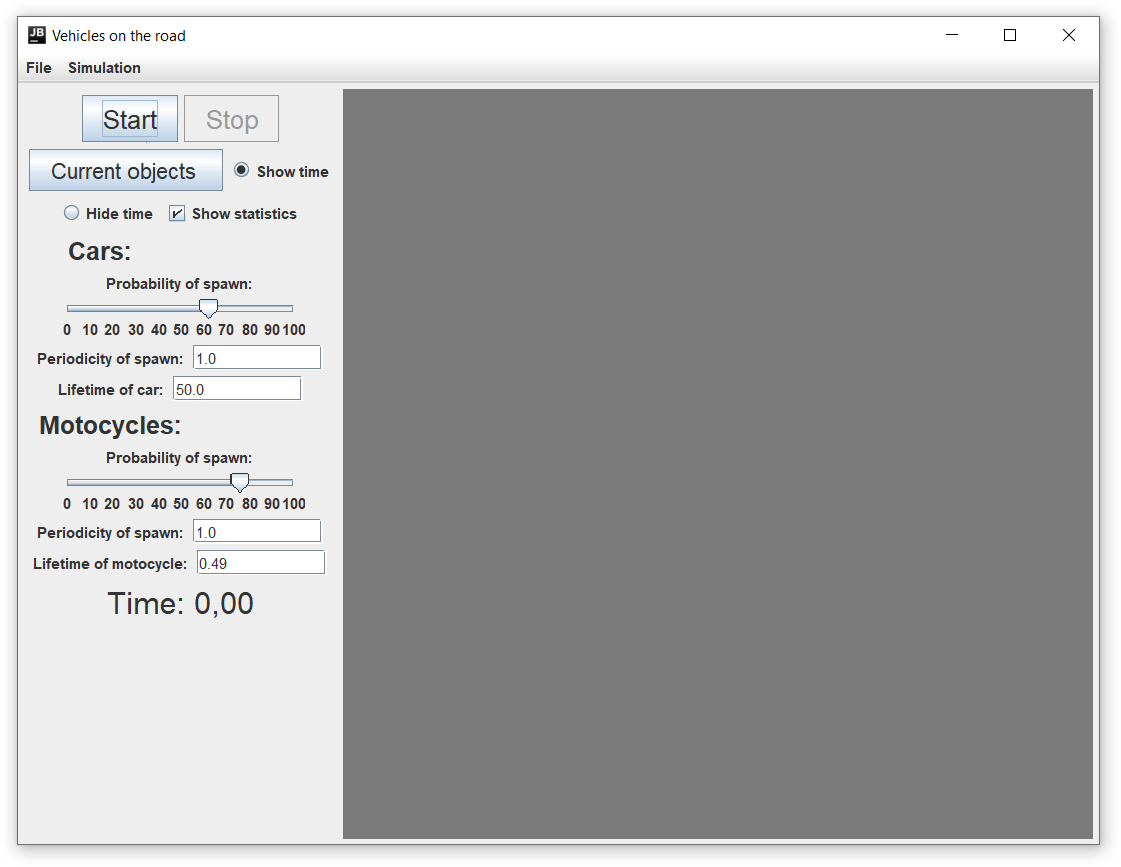




Теперь содержимое конфигурационного файла выглядит следующим образом:



И при следующем запуске, в приложение будут выставлены те же настройки, что и при выходе из программы.



**Вывод:**

Таким образом, в ходе проделанной лабораторной работы, в программу были добавлены возможность сохранения настроек симуляции, даже при выходе из программы, функционал, позволяющий пользователю сохранять симуляцию и открывать ранее сохраненную симуляцию.

В заключение можно сказать, что использование потоков ввода-вывода – очень важная часть при написании приложения, которая неизбежно ведет к оптимизированию уже написанной программы и облегчению восприятия интерфейса пользователем.

**Листинг программы:**

***Main.java***

import java.io.\*;  
import java.util.Arrays;  
  
public class Main {  
 public static void main(String[] args) {  
 Program program;  
 BufferedReader in = null;  
 try {  
 in = new BufferedReader(new FileReader("Configure.txt"));  
 String text;  
 float[] arr = new float[256];  
 for (int i = 0; (text = in.readLine()) != null; i++) {  
 arr[i] = Float.*parseFloat*(text);  
 System.*out*.println(+arr[i]);  
 }  
 in.close();  
 program = new Program(arr[0], arr[1], arr[2], arr[3], arr[4], arr[5]);  
 while (!program.isDone) {  
 program.run();  
 }  
 {  
 try (FileWriter writer = new FileWriter("Configure.txt", false)) {  
 String[] arr1 = program.getSettings();  
 for (int i = 0; i < 6; i++){  
 writer.write(arr1[i]+"\n");  
 System.*out*.println(arr1[i]);}  
 writer.flush();  
 writer.close();  
 } catch (IOException ex) {  
  
 System.*out*.println(ex.getMessage());  
 }  
  
 }  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
  
 System.*exit*(0);  
 }  
}

***Program.java***

import javax.swing.\*;  
import java.awt.\*;  
import javax.swing.event.ChangeEvent;  
import javax.swing.event.ChangeListener;  
import javax.swing.filechooser.FileNameExtensionFilter;  
import javax.swing.text.BadLocationException;  
import java.awt.event.\*;  
import java.io.File;  
import java.io.IOException;  
  
public class Program extends JFrame {  
 private final float timeStep = 1 / 60f;  
  
 private float N1\_d, N2\_d;  
 private float lifetime\_car\_d, lifetime\_motocycle\_d;  
 private float P1\_d, P2\_d;  
  
 private boolean isWorking;  
 private boolean statisticsIsVisible;  
 public boolean isDone;  
 private long startTime;  
 private long timePrev;  
 private float timeElapsed;  
  
 private JMenuBar menuBar;  
 private JMenuItem menuItemConsole;  
 private JMenuItem menuItemSave, menuItemOpen, menuItemExit;  
 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, periodicity\_motocycleField, lifetime\_carField, lifetime\_motocycleField;  
  
 private JSlider sliderProbability\_car, sliderProbability\_truck;  
  
 private FileNameExtensionFilter filter;  
  
 private Habitat habitat;  
  
 public Program(float N1, float N2, float lifetime\_car, float lifetime\_motocycle, float P1, float P2) {  
  
 N1\_d = N1;  
 N2\_d = N2;  
 lifetime\_car\_d = lifetime\_car;  
 lifetime\_motocycle\_d = lifetime\_motocycle;  
 P1\_d = P1;  
 P2\_d = P2;  
 /\* N1\_d =1;  
 N2\_d = 2;  
 lifetime\_car\_d = 10;  
 lifetime\_truck\_d = 20;  
 P1\_d = 0.5f;  
 P2\_d = 0.5f;\*/  
 isWorking = isDone = false;  
 statisticsIsVisible = true;  
 startTime = 0;  
 timePrev = System.*currentTimeMillis*();  
 timeElapsed = 0;  
  
 filter = new FileNameExtensionFilter("SER file", "ser");  
 habitat = new Habitat();  
  
 habitat.N1 = N1\_d;  
 habitat.N2 = N2\_d;  
 habitat.lifetime\_car = lifetime\_car\_d;  
 habitat.lifetime\_motocycle = lifetime\_motocycle\_d;  
 habitat.P1 = P1\_d;  
 habitat.P2 = P2\_d;  
  
 createGUI();  
 }  
  
 public String[] getSettings() {  
 String[] arr = new String[6];  
 arr[0] = Float.*toString*(N1\_d);  
 arr[1] = Float.*toString*(N2\_d);  
 arr[2] = Float.*toString*(lifetime\_car\_d);  
 arr[3] = Float.*toString*(lifetime\_motocycle\_d);  
 arr[4] = Float.*toString*(P1\_d);  
 arr[5] = Float.*toString*(P2\_d);  
 return arr;  
 }  
  
 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);  
  
  
 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.setEnabled(false);  
 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, ((int) (P1\_d \* 100)));  
 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(Float.*toString*(N1\_d));  
 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(Float.*toString*(lifetime\_car\_d));  
 lifetime\_carField.setToolTipText("Put here value in seconds in the range of 10..50");  
 guiPanel.add(lifetime\_carField);  
  
 motocycles = new JLabel("Motocycles: ");  
 guiPanel.add(motocycles);  
 motocycles.setFont(motocycles.getFont().deriveFont(20f));  
  
 probability\_motocycle = new JLabel("Probability of spawn: ");  
 guiPanel.add(probability\_motocycle);  
  
 sliderProbability\_truck = new JSlider(0, 100, ((int) (P2\_d \* 100)));  
 sliderProbability\_truck.setPaintLabels(true);  
 sliderProbability\_truck.setMajorTickSpacing(10);  
 guiPanel.add(sliderProbability\_truck);  
  
 periodicity\_motocycle= new JLabel("Periodicity of spawn: ");  
 guiPanel.add(periodicity\_motocycle);  
  
 periodicity\_motocycleField = new JTextField(10);  
 periodicity\_motocycleField.setText(Float.*toString*(N2\_d));  
 periodicity\_motocycleField.setToolTipText("Put here value in seconds in the range of 1..10");  
 guiPanel.add(periodicity\_motocycleField);  
  
 lifetime\_motocycle = new JLabel("Lifetime of motocycle: ");  
 guiPanel.add(lifetime\_motocycle);  
  
 lifetime\_motocycleField = new JTextField(10);  
 lifetime\_motocycleField.setText(Float.*toString*(lifetime\_motocycle\_d));  
 lifetime\_motocycleField.setToolTipText("Put here value in seconds in the range of 10..50");  
 guiPanel.add(lifetime\_motocycleField);  
  
 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(createFileMenu());  
 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() {  
 long timeCurrent = System.*currentTimeMillis*();  
 timeElapsed += (timeCurrent - timePrev) / 1000.f;  
 timePrev = timeCurrent;  
  
 while (timeElapsed >= timeStep) {  
 if (isWorking) {  
 long time = System.*currentTimeMillis*() - startTime;  
 habitat.update(startTime, timeStep);  
 updateTime(time);  
 }  
 timeElapsed -= timeStep;  
 }  
 }  
  
  
 private JMenu createFileMenu() {  
 JMenu fileMenu = new JMenu("File");  
  
 menuItemSave = new JMenuItem("Save As...");  
 menuItemOpen = new JMenuItem("Open...");  
 menuItemExit = new JMenuItem("Exit");  
  
 fileMenu.add(menuItemSave);  
 fileMenu.add(menuItemOpen);  
 fileMenu.add(new JSeparator());  
 fileMenu.add(menuItemExit);  
 return fileMenu;  
 }  
  
 private JMenu createSimulationMenu() {  
 JMenu simulationMenu = new JMenu("Simulation");  
  
 menuItemStart = new JMenuItem("Start");  
 menuItemStop = new JMenuItem("Stop");  
 menuItemStop.setEnabled(true);  
  
 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) {  
 int 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 = (int) N1\_d;  
 periodicity\_carField.setText(Float.*toString*(value\_periodicity\_car));  
 }  
 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 = (int) N1\_d;  
 periodicity\_carField.setText(Float.*toString*(value\_periodicity\_car));  
 } else {  
 N1\_d = habitat.N1 = value\_periodicity\_car \* 1000;  
 }  
  
 }  
  
 };  
 ActionListener actionListener\_periodicity\_truckField = new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 int value\_periodocity\_truck;  
 try {  
 value\_periodocity\_truck = Integer.*parseInt*(periodicity\_motocycleField.getText());  
 } catch (NumberFormatException exp) {  
 JOptionPane.*showConfirmDialog*(null, "Value must be integer",  
 "Wrong data type",  
 JOptionPane.*CANCEL\_OPTION*, JOptionPane.*ERROR\_MESSAGE*);  
 //periodicity\_truckField.setText(null);  
 value\_periodocity\_truck = (int) N2\_d;  
 periodicity\_motocycleField.setText(Float.*toString*(value\_periodocity\_truck));  
 }  
 if (value\_periodocity\_truck < 1 || value\_periodocity\_truck > 10) {  
 JOptionPane.*showConfirmDialog*(null, "Value must be in the range of 1..10",  
 "Out of range error",  
 JOptionPane.*CANCEL\_OPTION*, JOptionPane.*ERROR\_MESSAGE*);  
 value\_periodocity\_truck = (int) N2\_d;  
 periodicity\_motocycleField.setText(Float.*toString*(value\_periodocity\_truck));  
 } else {  
 N2\_d = habitat.N2 = value\_periodocity\_truck;  
 }  
 }  
  
 };  
 ActionListener actionListener\_lifetime\_carField = new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 float value\_lifetime\_car = lifetime\_car\_d;  
 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 = lifetime\_car\_d;  
 lifetime\_carField.setText(Float.*toString*(value\_lifetime\_car));  
 }  
 if (value\_lifetime\_car < 1 || value\_lifetime\_car > 50) {  
 JOptionPane.*showConfirmDialog*(null, "Value must be in the range of 1..50",  
 "Out of range error",  
 JOptionPane.*CANCEL\_OPTION*, JOptionPane.*ERROR\_MESSAGE*);  
 value\_lifetime\_car = lifetime\_car\_d;  
 lifetime\_carField.setText(Float.*toString*(value\_lifetime\_car));  
 } else {  
 habitat.setCarTime(value\_lifetime\_car);  
 lifetime\_car\_d = value\_lifetime\_car;  
 }  
 }  
  
 };  
 ActionListener actionListener\_lifetime\_truckField = new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 float value\_lifetime\_motocycle = lifetime\_motocycle\_d;  
 try {  
 value\_lifetime\_motocycle = Integer.*parseInt*(lifetime\_motocycleField.getText());  
 } catch (NumberFormatException exp) {  
 JOptionPane.*showConfirmDialog*(null, "Value must be integer",  
 "Wrong data type",  
 JOptionPane.*CANCEL\_OPTION*, JOptionPane.*ERROR\_MESSAGE*);  
 value\_lifetime\_motocycle = lifetime\_motocycle\_d;  
 lifetime\_motocycleField.setText(Float.*toString*(value\_lifetime\_motocycle));  
 }  
 if (value\_lifetime\_motocycle < 1 || value\_lifetime\_motocycle > 50) {  
 JOptionPane.*showConfirmDialog*(null, "Value must be in the range of 1..50",  
 "Out of range error",  
 JOptionPane.*CANCEL\_OPTION*, JOptionPane.*ERROR\_MESSAGE*);  
 value\_lifetime\_motocycle = lifetime\_motocycle\_d;  
 lifetime\_motocycleField.setText(Float.*toString*(value\_lifetime\_motocycle));  
 } else {  
 habitat.setMotoTime(value\_lifetime\_motocycle);  
 lifetime\_motocycle\_d = value\_lifetime\_motocycle;  
 }  
 }  
  
 };  
  
 lifetime\_motocycleField.addActionListener(actionListener\_lifetime\_truckField);  
 lifetime\_carField.addActionListener(actionListener\_lifetime\_carField);  
 periodicity\_carField.addActionListener(actionListener\_periodicity\_carField);  
 periodicity\_motocycleField.addActionListener(actionListener\_periodicity\_truckField);  
 }  
  
 private void slides() {  
 ChangeListener changeListener = new ChangeListener() {  
 public void stateChanged(ChangeEvent e) {  
 int value1 = sliderProbability\_car.getValue();  
 P1\_d = habitat.P1 = value1 / 100f;  
 //int value = ((JSlider) e.getSource()).getValue();  
 System.*out*.println("P1 value " + value1 + "%");  
 int value2 = sliderProbability\_truck.getValue();  
 P2\_d = habitat.P2 = value2 / 100f;  
 //int value = ((JSlider) e.getSource()).getValue();  
 System.*out*.println("P2 value " + value2 + "%");  
 }  
 };  
 sliderProbability\_car.addChangeListener(changeListener);  
 sliderProbability\_truck.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();  
 }  
  
 if (e.getSource() == menuItemSave) {  
 JFileChooser fc = new JFileChooser();  
 fc.setDialogTitle("Save file as");  
 fc.setFileFilter(filter);  
 fc.showSaveDialog(null);  
 File selFile = fc.getSelectedFile();  
 if(selFile!=null){  
 habitat.save(selFile);  
 }  
 }  
  
 if (e.getSource() == menuItemOpen) {  
  
 try {  
 JFileChooser fileopen = new JFileChooser();  
 fileopen.setDialogTitle("Open file");  
 fileopen.setFileFilter(filter);  
 fileopen.showOpenDialog(null);  
 File file = fileopen.getSelectedFile();  
 if(file!=null)habitat.open(startTime, file);  
  
 } catch (IOException e1) {  
 e1.printStackTrace();  
 }  
  
 }  
  
 if (e.getSource() == menuItemExit) {isDone = true;}  
 }  
 };  
  
  
 menuItemStart.addActionListener(actionListenerB);  
 menuItemStop.addActionListener(actionListenerB);  
 menuItemShowTime.addActionListener(actionListenerB);  
 menuItemHideTime.addActionListener(actionListenerB);  
 menuItemShowStatistics.addActionListener(actionListenerB);  
 menuItemCurrentObjects.addActionListener(actionListenerB);  
 menuItemSave.addActionListener(actionListenerB);  
 menuItemOpen.addActionListener(actionListenerB);  
 menuItemExit.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.io.\*;  
import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.TreeSet;  
  
  
public class Habitat extends JPanel {  
  
 private final int width = 600, height = 600, indent = 50;  
  
  
 public float N1, N2;  
 public float P1, P2;  
 public float lifetime\_car, lifetime\_motocycle;  
 private float countTime1, countTime2;  
 private int count1, count2;  
 private long currentSimTime;  
 private ArrayList<Transport> myObjects;  
 private TreeSet<Integer> identifiers;  
 private HashMap<Integer, Float> identifiers\_lifetime;  
  
 public Habitat() {  
 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));  
 }  
  
  
 public ArrayList<Transport> getObjects() {  
 return myObjects;  
 }  
  
 public void setCarTime(float value) {  
 lifetime\_car = value;  
 }  
  
 public void setMotoTime(float value) {  
 lifetime\_motocycle = value;  
 }  
  
 private int genRandPos(int l, int r) {  
 return l + (int) (Math.*random*() \* (r - l));  
 }  
  
 private void put\_in(Transport obj) {  
 int index = genIndex();  
  
 obj.setID(index);  
 myObjects.add(obj);  
  
 //identifiers.add(index);  
 //identifiers\_lifetime.put(index, obj.getLifeTime());  
  
 }  
  
 private boolean check(int index) {  
 for(int i=0; i<myObjects.size(); i++){  
 if(myObjects.get(i).identifier==index)  
 return false;  
 }  
 return true;  
 }  
 private int genIndex() {  
 int index = (int) (Math.*random*() \* (count1 + count2 + 10));  
 while (!check(index))  
 index = (int) (Math.*random*() \* (count1 + count2 + 10));  
 return index;  
 }  
  
 public void save(File file) {  
 try {  
 FileOutputStream outputStream = new FileOutputStream(file);  
 ObjectOutputStream objectOutputStream = new ObjectOutputStream(outputStream);  
 SavedSimulation savedSimulation = new SavedSimulation(myObjects);  
 objectOutputStream.writeObject(savedSimulation);  
 objectOutputStream.close();  
 outputStream.close();  
 } catch (IOException ioe) {  
 ioe.printStackTrace();  
 }  
 }  
  
 public void open(long startTime, File file) throws IOException {  
  
 try {  
 FileInputStream fileInputStream = new FileInputStream(file);  
 ObjectInputStream objectInputStream = new ObjectInputStream(fileInputStream);  
 SavedSimulation savedSimulation = (SavedSimulation) objectInputStream.readObject();  
 myObjects.clear();  
 myObjects = savedSimulation.myObjects;  
 } catch (ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
  
 removeAll();  
 repaint();  
 countTime1=countTime2=0;  
 count1=count2=0;  
 currentSimTime = System.*currentTimeMillis*() - startTime;  
  
 for(int i=0;i<myObjects.size();i++)  
 {  
 myObjects.get(i).time\_of\_birth=currentSimTime;  
 if(myObjects.get(i).getName()=="Car")  
 count1++;  
 else  
 count2++;  
 add(myObjects.get(i));  
 }  
 }  
  
 public void update(long startTime, float delta) {  
 countTime1 += delta;  
 countTime2 += delta;  
  
 currentSimTime = System.*currentTimeMillis*() - startTime;  
  
 if (countTime1 >= N1) {  
 if (Math.*random*() <= P1) {  
  
 Car car = new Car(genRandPos(0, width - indent), genRandPos(0, height - indent), currentSimTime, lifetime\_car);  
 add(car);  
 put\_in(car);  
  
 revalidate();  
 count1++;  
 }  
  
 countTime1 -= N1;  
 }  
  
 if (countTime2 >= N2) {  
 if (Math.*random*() <= P2) {  
 Motocycle motocycle = new Motocycle(genRandPos(0, width - indent), genRandPos(0, height - indent), currentSimTime, lifetime\_motocycle);  
 add(motocycle);  
 put\_in(motocycle);  
  
 revalidate();  
 count2++;  
 }  
  
 countTime2 -= N2;  
 }  
  
 updateTransports(delta);  
 this.repaint();  
 }  
  
 public void showCurrentObjects() {  
 int upd\_table = JOptionPane.*OK\_OPTION*;  
 UIManager.*put*("OptionPane.okButtonText", "Update");  
 if (myObjects.size() != 0) {  
 while (upd\_table == JOptionPane.*OK\_OPTION*) {  
 Table table = new Table(myObjects);  
 upd\_table = JOptionPane.*showConfirmDialog*(this,  
 table.createTable(),  
 "Current objects",  
 JOptionPane.*DEFAULT\_OPTION*, JOptionPane.*INFORMATION\_MESSAGE*);  
 }  
 } else {  
 UIManager.*put*("OptionPane.okButtonText", "Ok");  
 JOptionPane.*showConfirmDialog*(null, "No objects",  
 "Current objects",  
 JOptionPane.*DEFAULT\_OPTION*, JOptionPane.*ERROR\_MESSAGE*);  
 }  
  
 UIManager.*put*("OptionPane.okButtonText", "Ok");  
 }  
  
 public void updateTransports(float delta) {  
 for (int i = 0; i < myObjects.size(); i++) {  
 if (myObjects.get(i).update(delta)) {  
 delete(myObjects.get(i), i);  
 }  
 }  
 }  
  
 private void delete(Transport obj, int i) {  
 if (obj.getName().equals("Car")) count1--;  
 else count2--;  
  
 remove(obj);  
 validate();  
  
// identifiers.remove(obj.getID());  
// identifiers\_lifetime.remove(obj.getID());  
 myObjects.remove(i);  
 }  
  
 public boolean fin(long time, boolean statisticsIsVisible) {  
 if (statisticsIsVisible) {  
 int selectedValue = JOptionPane.*showConfirmDialog*(this,  
 String.*format*("<html>Cars: %d<br> Motocycle: %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;  
 }  
 }  
 myObjects.clear();  
 return false;  
  
 }  
  
 public void start() {  
 count1 = count2 = 0;  
 countTime1 = countTime2 = 0;  
 removeAll();  
 setLayout(null);  
 revalidate();  
 }  
  
  
}

***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;  
 }  
  
}

***SavedSimulation.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, long time, float lifetime) {  
 this.x = x;  
 this.y = y;  
 time\_of\_birth = time;  
 this.lifetime = lifetime;  
 this.setBounds((int) this.x, (int) this.y, 90, 65);  
 this.setVisible(true);  
 }  
 public boolean update(float delta) {  
 lifetime -= delta;  
 if (lifetime <= 0)  
 return true;  
 return false;  
 }  
  
 public float getTime() {  
 return time\_of\_birth;  
 }  
  
 public float getLifeTime() {  
 return lifetime;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public void setID(int ID) {  
 identifier = ID;  
 }  
  
 public int getID() {  
 return identifier;  
 }  
 @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, long time, float lifetime) {  
 super(x, y, time, lifetime);  
 this.setName("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, long time, float lifetime) {  
 super(x, y, time, lifetime);  
 name="Motocycle";  
 Image img = *image*.getScaledInstance(this.getWidth(), this.getHeight(), Image.*SCALE\_DEFAULT*);  
 this.setIcon(new ImageIcon(img));  
 System.*out*.println("Motocycle: (" + x + ";" + y + ") ");  
 }  
  
 }