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 №2

по дисциплине «Технология программирования»

Тема: «Разработка графического интерфейса приложения.»

Вариант №8

Выполнила: Проверил:

студентка группы АВТ-716 Михайленко Дмитрий

Качесова Кристина Васильевна Анатольевич

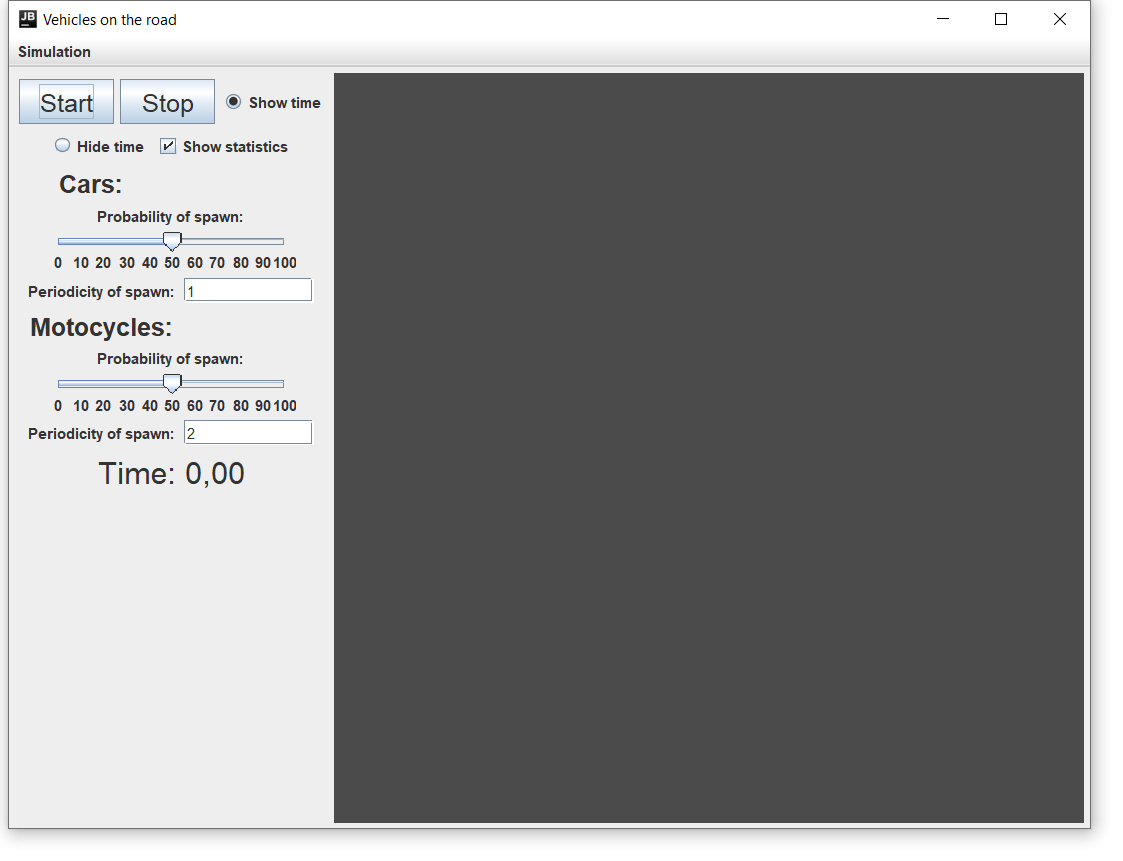
Новосибирск,

2019 г.

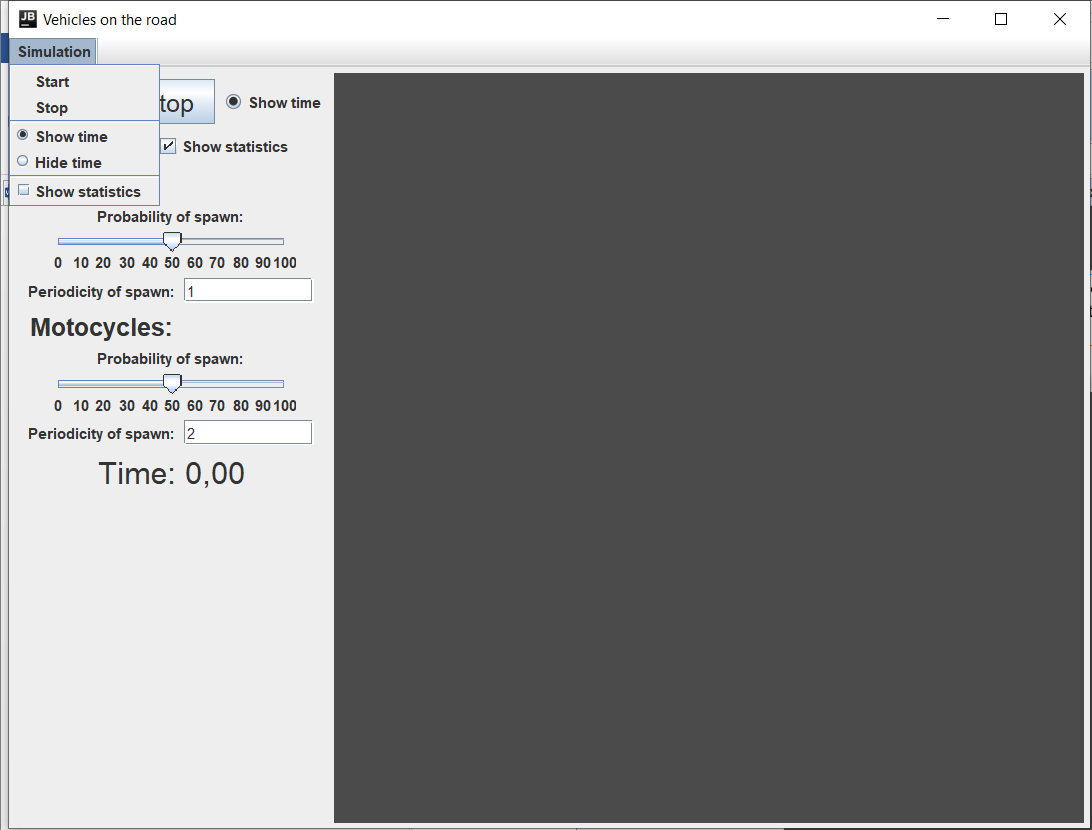
**Цель работы:**

1. Познакомиться с основными графическими библиотеками Java - AWT и Swing и их основными компонентами. Изучить классы менеджеров компоновки, классы создания меню приложения, основных и диалоговых окон и т.д.
2. Для хранения генерируемых объектов использовать динамический массив объектов.
3. Доработать программу, созданную в лабораторной работе № 1:
   1. поделить рабочую область окна приложения на 2 части. Визуализация переносится в одну часть окна, панель управления в другую;
   2. добавить кнопки «Старт» и «Стоп» в панель управления. Они должны запускать и останавливать симуляцию соответственно. Если симуляция остановлена, то кнопка «Стоп» должна блокироваться. Если симуляция идет, то блокируется кнопка «Старт». Клавиши **B** и **E** должны функционировать по-прежнему;
   3. добавить переключатель «Показывать информацию», который разрешает отображение модального диалога из 7 пункта задания;
   4. добавить группу из 2 исключающих переключателей: «Показывать время симуляции» и «Скрывать время симуляции». Клавиша **T** должна функционировать по-прежнему;
   5. используя различные менеджеры компоновки реализовать интерфейс пользователя согласно индивидуальному заданию;
   6. добавить в программу главное меню и панель инструментов, в которых продублировать основные команды интерфейса пользователя;
   7. при остановке симуляции должно появляться модальное диалоговое окно (при условии, что оно разрешено) с информацией о количестве и типе сгенерированных объектов, а также времени симуляции. Вся информация выводится в элементе TextArea, недоступном для редактирования. В диалоговом окне должно быть 2 кнопки: «ОК» и «Отмена». При нажатии на «ОК» симуляции полностью останавливается, а при нажатии на «Отмена», соответственно продолжается;
   8. предусмотреть проверку данных вводимых пользователем. При вводе неверного значения обрабатывать исключительную ситуацию: выставлять значение по умолчанию и выводить диалоговое окно с сообщением об ошибке;
   9. Реализовать следующие элементы управления:
   * Периоды рождения объектов – текстовые поля;
   * Для задания вероятностей рождения объектов - JComboBox и JSlider  (шаг значений 10);
   * Дополнить интерфейс поясняющими метками.

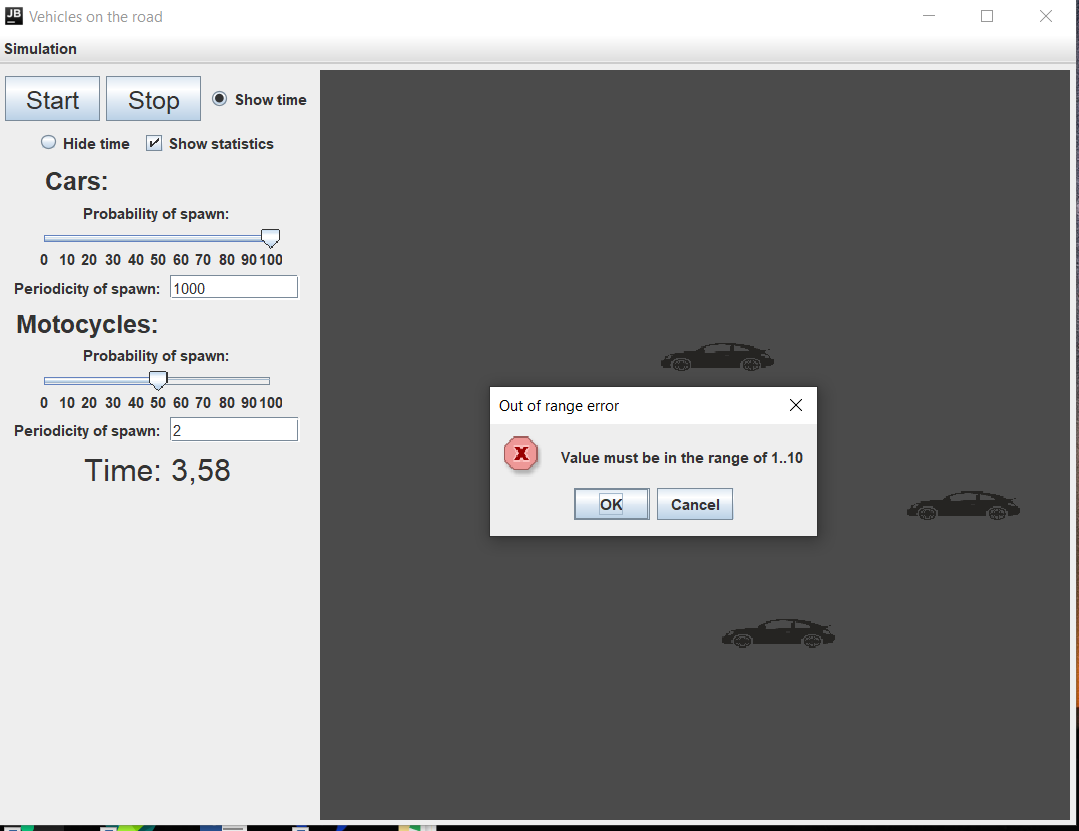
**Результаты работы программы**

При запуске программы пользователь видит следующее окно: 

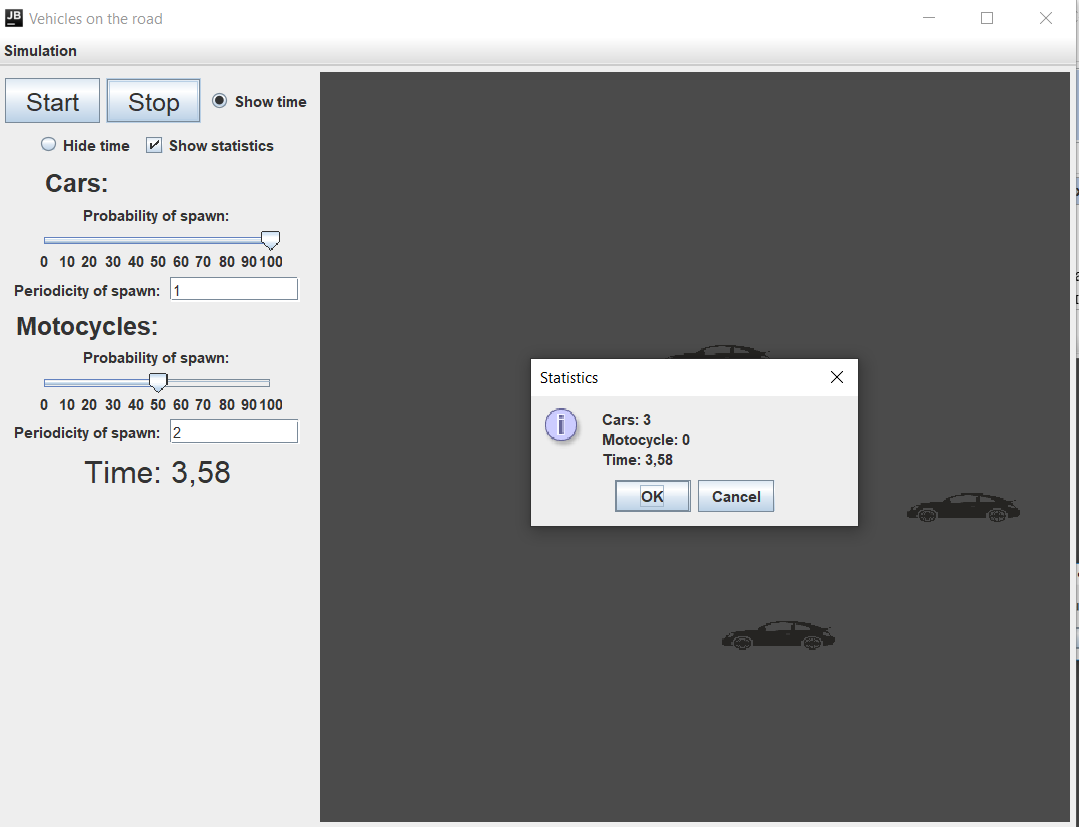
Есть возможность управлять функционалом при помощи вложенного меню в основном меню с элементами: Start, Stop, Show time, Hide time, Show statistics. Кнопки в панели и меню сообщаются между собой.



Если ввести в TextField недопустимые значения, то пользователь увидит модальное окно, а сами значения будут изменены на значения по умолчанию.



После окончания моделирования пользователь видит статистику по выполнению программы. Если нажать «Отмена», то симуляция продолжится, иначе - закончится.



**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 menuItemStart, menuItemStop;  
 private JCheckBoxMenuItem menuItemShowStatistics;  
 private JRadioButtonMenuItem menuItemShowTime, menuItemHideTime;  
  
 private JLabel timeTextLabel, cars, motocycles, p1, p2, b1, b2;  
 private JButton buttonStart, buttonStop;  
 private ButtonGroup timeVisible;  
 private JRadioButton buttonShowTime, buttonHideTime;  
 private JCheckBox buttonShowStatistics;  
 private JTextField N1Field, N2Field;  
  
 private JSlider sliderP1, sliderP2;  
  
 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(20f));  
 buttonStart.setFont(buttonStart.getFont().deriveFont(Font.*PLAIN*));  
 guiPanel.add(buttonStart);  
  
 buttonStop = new JButton("Stop");  
 buttonStop.setVisible(true);  
 buttonStop.setFont(buttonStop.getFont().deriveFont(20f));  
 buttonStop.setFont(buttonStop.getFont().deriveFont(Font.*PLAIN*));  
 guiPanel.add(buttonStop);  
  
 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));  
  
 p1 = new JLabel("Probability of spawn: ");  
 guiPanel.add(p1);  
  
 sliderP1 = new JSlider(0, 100, 50);  
 sliderP1.setPaintLabels(true);  
 sliderP1.setMajorTickSpacing(10);  
 guiPanel.add(sliderP1);  
  
 b1 = new JLabel("Periodicity of spawn: ");  
 guiPanel.add(b1);  
  
 N1Field = new JTextField(10);  
 N1Field.setText("1");  
 N1Field.setToolTipText("Put here value in seconds in the range of 1..10");  
 guiPanel.add(N1Field);  
  
 motocycles = new JLabel("Motocycles: ");  
 guiPanel.add(motocycles);  
 motocycles.setFont(motocycles.getFont().deriveFont(20f));  
  
 p2 = new JLabel("Probability of spawn: ");  
 guiPanel.add(p2);  
  
 sliderP2 = new JSlider(0, 100, 50);  
 sliderP2.setPaintLabels(true);  
 sliderP2.setMajorTickSpacing(10);  
 guiPanel.add(sliderP2);  
  
 b2 = new JLabel("Periodicity of spawn: ");  
 guiPanel.add(b2);  
  
 N2Field = new JTextField(10);  
 N2Field.setText("2");  
 N2Field.setToolTipText("Put here value in seconds in the range of 1..10");  
 guiPanel.add(N2Field);  
  
 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) {  
 startTime = System.*currentTimeMillis*();  
 habitat.start();  
 isWorking = true;  
 System.*out*.println("Begin");  
 }  
 }  
  
 private void end() {  
 if (isWorking) {  
 isWorking = false;  
 long stopTime = System.*currentTimeMillis*() - startTime;  
 isWorking = habitat.fin(System.*currentTimeMillis*() - startTime, statisticsIsVisible);  
 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);  
 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");  
  
 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);  
  
 simulationMenu.add(menuItemStart);  
 simulationMenu.add(menuItemStop);  
 simulationMenu.add(new JSeparator());  
 simulationMenu.add(menuItemShowTime);  
 simulationMenu.add(menuItemHideTime);  
 simulationMenu.add(new JSeparator());  
 simulationMenu.add(menuItemShowStatistics);  
  
 return simulationMenu;  
 }  
  
 private void fields() {  
 ActionListener actionListenerF1 = new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 long value\_N1;  
 try {  
 value\_N1 = Integer.*parseInt*(N1Field.getText());  
 } catch (NumberFormatException exp) {  
 JOptionPane.*showConfirmDialog*(null, "Value must be integer",  
 "Wrong data type",  
 JOptionPane.*CANCEL\_OPTION*, JOptionPane.*ERROR\_MESSAGE*);  
 value\_N1 = 1;  
 N1Field.setText("1");  
 }  
 if (value\_N1 < 1 || value\_N1 > 10) {  
 JOptionPane.*showConfirmDialog*(null, "Value must be in the range of 1..10",  
 "Out of range error",  
 JOptionPane.*CANCEL\_OPTION*, JOptionPane.*ERROR\_MESSAGE*);  
 value\_N1 = 1;  
 N1Field.setText("1");  
 } else {  
 habitat.N1 = value\_N1 \* 1000;  
 }  
  
 }  
  
 };  
 N1Field.addActionListener(actionListenerF1);  
  
 ActionListener actionListenerF2 = new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 long value\_N2;  
 try {  
 value\_N2 = Integer.*parseInt*(N2Field.getText());  
 } catch (NumberFormatException exp) {  
 JOptionPane.*showConfirmDialog*(null, "Value must be integer",  
 "Wrong data type",  
 JOptionPane.*CANCEL\_OPTION*, JOptionPane.*ERROR\_MESSAGE*);  
 //N2Field.setText(null);  
 value\_N2 = 2;  
 N2Field.setText("2");  
 }  
 if (value\_N2 < 1 || value\_N2 > 10) {  
 JOptionPane.*showConfirmDialog*(null, "Value must be in the range of 1..10",  
 "Out of range error",  
 JOptionPane.*CANCEL\_OPTION*, JOptionPane.*ERROR\_MESSAGE*);  
 value\_N2 = 2;  
 N2Field.setText("2");  
 } else {  
 habitat.N2 = value\_N2 \* 1000;  
 }  
 }  
  
 };  
 N2Field.addActionListener(actionListenerF2);  
 }  
  
 private void slides() {  
 ChangeListener changeListener = new ChangeListener() {  
 public void stateChanged(ChangeEvent e) {  
 int value1 = sliderP1.getValue();  
 habitat.P1 = value1 / 100f;  
 //int value = ((JSlider) e.getSource()).getValue();  
 System.*out*.println("P1 value " + value1 + "%");  
 int value2 = sliderP2.getValue();  
 habitat.P2 = value2 / 100f;  
 //int value = ((JSlider) e.getSource()).getValue();  
 System.*out*.println("P2 value " + value2 + "%");  
 }  
 };  
 sliderP1.addChangeListener(changeListener);  
 sliderP2.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;  
 }  
  
 }  
  
 }  
 };  
  
 menuItemStart.addActionListener(actionListenerB);  
 menuItemStop.addActionListener(actionListenerB);  
 menuItemShowTime.addActionListener(actionListenerB);  
 menuItemHideTime.addActionListener(actionListenerB);  
 menuItemShowStatistics.addActionListener(actionListenerB);  
  
 buttonStart.addActionListener(actionListenerB);  
 buttonStop.addActionListener(actionListenerB);  
 buttonShowTime.addActionListener(actionListenerB);  
 buttonHideTime.addActionListener(actionListenerB);  
 buttonShowStatistics.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()) {  
 //e.setSource(buttonHideTime);  
 //buttons();  
 buttonHideTime.setSelected(true);  
 timeTextLabel.setVisible(false);  
 } else {  
 //e.setSource(buttonShowTime);  
 buttonShowTime.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.Arrays;  
  
  
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;  
 private int countTime1, countTime2;  
 protected int count1, count2;  
 private Transport[] myObjects;  
  
  
 public Habitat() {  
 N1 = 1000;  
 N2 = 2000;  
 P1 = P2 = 0.5f;  
 count1 = count2 = 0;  
 countTime1 = countTime2 = 0;  
 myObjects = new Transport[size];  
  
 setPreferredSize(new Dimension(width, height));  
 setLayout(null);  
 setBackground(new Color(75, 75, 75));  
 }  
  
 private int setPos(int len) {  
 return (int) (Math.*random*() \* (len - 50));  
 }  
  
 private void PutIn(Transport obj) {  
 if (count1 + count2 + 1 == myObjects.length) {  
 System.*out*.println("I'm here and I gonna copy your array");  
 myObjects = Arrays.*copyOf*(myObjects, myObjects.length \* 2);  
 }  
 myObjects[count1 + count2] = obj;  
 }  
  
 private void check() {  
 System.*out*.println("myObjects:");  
 for (int i = 0; myObjects[i] != null; i++)  
 System.*out*.println(myObjects[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);  
 PutIn(car);  
 validate();  
 repaint();  
 count1++;  
 }  
 countTime1++;  
 }  
  
 if (time - (countTime2 \* N2) >= N2) {  
 if (Math.*random*() <= P2) {  
 Motocycle truck = new Motocycle(setPos(width), setPos(height));  
 add(truck);  
 PutIn(truck);  
 validate();  
 repaint();  
 count2++;  
 }  
 countTime2++;  
 }  
 }  
  
 public boolean fin(long time, boolean statisticsIsVisible) {  
 check();  
 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;  
 }  
 }  
 return false;  
  
 //revalidate();  
 //repaint();  
 }  
  
 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();  
}

***Transport.java***

import javax.swing.\*;  
  
public abstract class Transport extends JLabel implements IBehaviour {  
 public float x, y;  
  
 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);  
 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);  
 Image img = *image*.getScaledInstance(this.getWidth(), this.getHeight(), Image.*SCALE\_DEFAULT*);  
 this.setIcon(new ImageIcon(img));  
 System.*out*.println("Motocycle: (" + x + ";" + y + ") ");  
 }  
  
 }