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: «Основы программирования на Java.

Обработка событий. Механизм делегирования событий.»

Вариант №8

Выполнила: Проверил:

студентка группы АВТ-716 Михайленко Дмитрий

Качесова Кристина Васильевна Анатольевич

Новосибирск,

2019 г.

**Цель работы:**

1. Познакомиться с особенностями технологии Java и изучить синтаксис языка Java.
2. Изучить основные понятия и термины обработки событий в модели делегирования событий на Java
3. Разработать учебную программу.

Рабочий цикл программы:

* запускается процесс симуляции по клавише, генерируются объекты классов согласно заданию;
* симуляция завершается по другой клавише, выводится статистическая информация.

Для решения задачи:

* Разработать абстрактный класс объекта, согласно варианту индивидуального задания.
* Создать интерфейс IBehaviour, задающий поведение объекта (методы: move(), getx(), gety(), sety(), setx() и другие. Далее будут реализоваться алгоритмы движения объектов в окне программы).
* Реализовать иерархию классов, определяющих объекты по варианту и реализующие интерфейс IBehaviour.
* Создать класс Habitat (среда), определяющий размер рабочей области и хранящий массив объектов, с параметрами, заданными вариантом. Предусмотреть в классе метод Update, вызывающийся по таймеру и получающий на вход время, прошедшее от начала симуляции. В данном методе должны генерироваться новые объекты и помещаться в поле визуализации в случайном месте. Визуализация объекта - использовать готовые небольшие картинки;

Рабочее окно программы – область визуализации среды обитания объектов;

Симуляция должна запускаться по клавише **B** и останавливаться по клавише **E**. При остановке симуляции список очищается. Время симуляции должно отображаться текстом в области визуализации и скрываться/показываться по клавише **T**;

По завершению симуляции область отображения объектов очищается и в поле визуализации выводится информация о количестве и типе сгенерированных объектов, а также время симуляции. Текст должен быть форматирован, т.е. выводиться с использованием разных шрифтов и цветов.

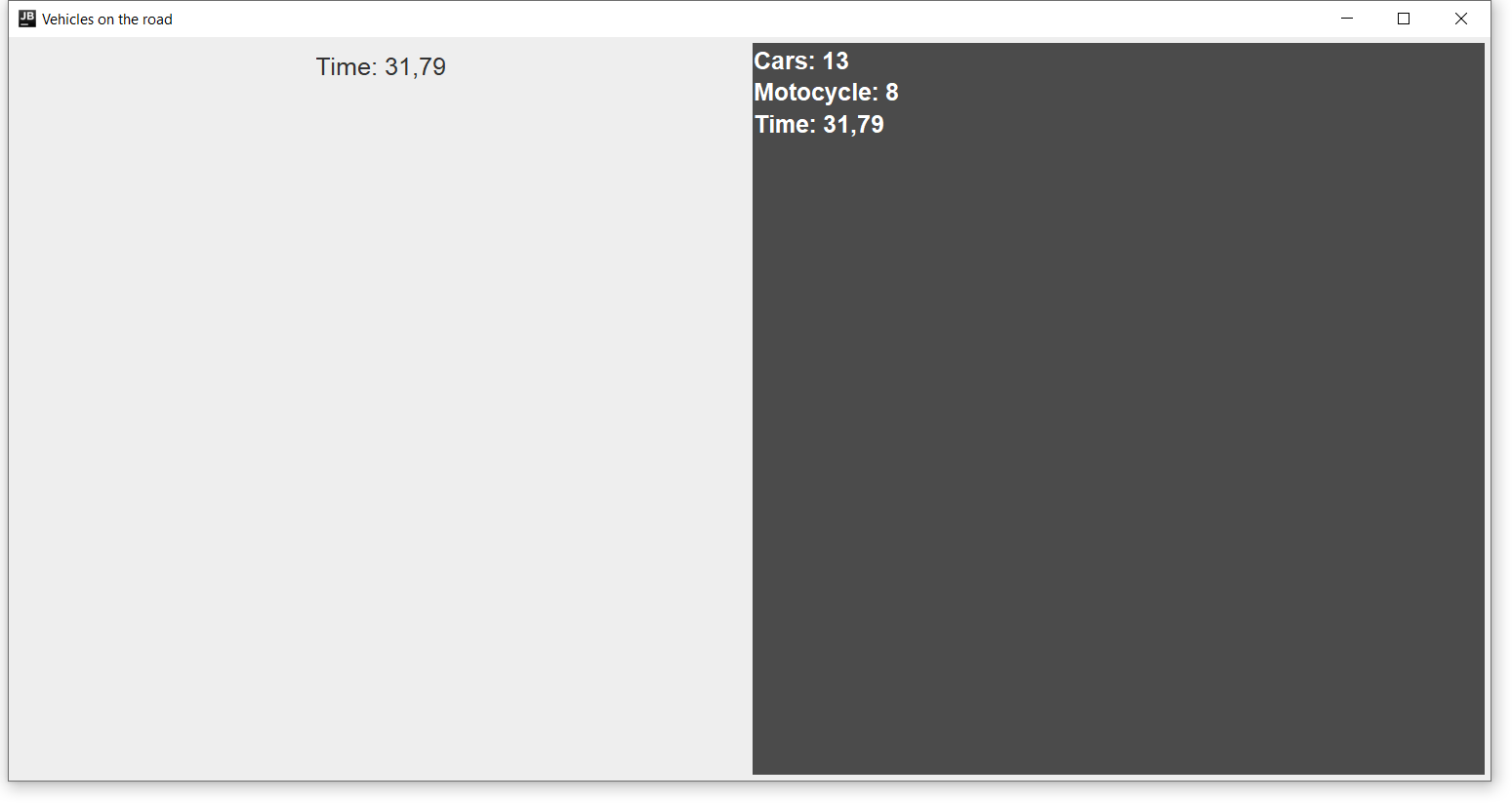
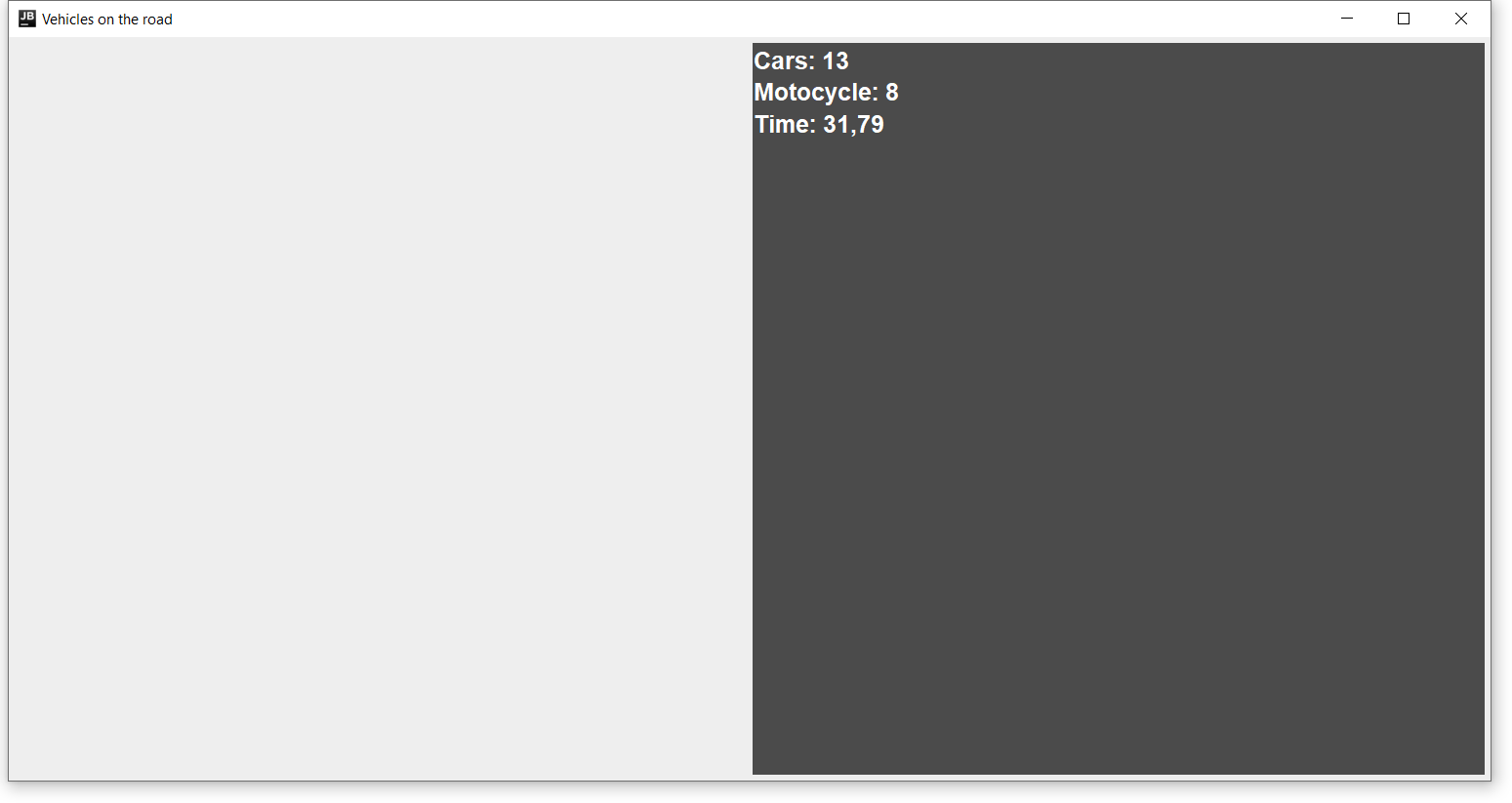
Параметры симуляции задаются в классе Habitat.

**Задание по варианту:**

Список транспортных средств на дороге состоит из двух категорий: автомобили и мотоциклы. Автомобили генерируются каждые N1 секунд с вероятностью P1. Мотоциклы генерируются каждые N2 секунд с вероятностью P2.

**Результат работы программы:**

****

**** ****

**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 java.awt.event.KeyAdapter;  
import java.awt.event.KeyEvent;  
  
public class Program {  
 private boolean isWorking;  
 public boolean isDone;  
 private long startTime;  
 private long timePrev;  
 private float timeElapsed;  
  
 private JLabel timeTextLabel;  
 private Habitat habitat;  
  
 public Program() {  
 createGUI();  
 }  
  
  
 private void createGUI() {  
 isWorking = isDone = false;  
 startTime = 0;  
 timePrev = System.*currentTimeMillis*();  
 timeElapsed = 0;  
 JFrame frame = new JFrame("Vehicles on the road");  
 frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 frame.getContentPane().setLayout(new FlowLayout());  
 JPanel guiPanel = new JPanel();  
 guiPanel.setPreferredSize(new Dimension(600, 600));  
 frame.add(guiPanel);  
  
 habitat = new Habitat();  
 frame.add(habitat);  
  
 timeTextLabel = new JLabel("Time: 0,00");  
 timeTextLabel.setVisible(true);  
 timeTextLabel.setFont(timeTextLabel.getFont().deriveFont(20f));  
 timeTextLabel.setFont(timeTextLabel.getFont().deriveFont(Font.*PLAIN*));  
 guiPanel.add(timeTextLabel);  
  
 frame.setComponentOrientation(ComponentOrientation.*LEFT\_TO\_RIGHT*);  
 frame.pack();  
 frame.setVisible(true);  
 frame.addKeyListener(new KeyAdapter() {  
 @Override  
 public void keyPressed(KeyEvent evt) {  
 switch (evt.getKeyCode()) {  
 case KeyEvent.*VK\_B*:  
 if (!isWorking) {  
 startTime = System.*currentTimeMillis*();  
 habitat.start();  
 isWorking = true;  
 System.*out*.println("Begin");  
 }  
 break;  
 case KeyEvent.*VK\_T*:  
 timeTextLabel.setVisible(!timeTextLabel.isVisible());  
 break;  
 case KeyEvent.*VK\_E*:  
 if (isWorking) {  
 isWorking = false;  
 habitat.fin(System.*currentTimeMillis*() - startTime);  
 System.*out*.println("End");  
 }  
 break;  
 case KeyEvent.*VK\_ESCAPE*:  
 isDone = true;  
 break;  
 }  
 }  
 });  
 }  
  
 private void updateTime(long time) {  
 timeTextLabel.setText("Time: " + String.*format*("%.02f", time / 1000f));  
 }  
  
 public void run() {  
 float timeStep = 1 / 60f;  
 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;  
 }  
 }  
}

***Habitat.java***

import javax.swing.\*;  
import java.awt.\*;  
  
public class Habitat extends JPanel {  
 private final int width = 600, height = 600;  
 private final float P1 = 0.5f, P2 = 0.5f;  
 private final long N1 = 1000, N2 = 2000;  
 private int countTime1, countTime2;  
 private int count1, count2;  
  
  
 public Habitat() {  
 count1 = count2 = 0;  
 countTime1 = countTime2 = 0;  
 this.setPreferredSize(new Dimension(width, height));  
 this.setLayout(null);  
 this.setBackground(new Color(75, 75, 75));  
 }  
  
 public void update(long time) {  
 if (time - (countTime1 \* N1) >= N1) {  
 if (Math.*random*() <= P1) {  
 this.add(new Car((int) (Math.*random*() \* width), (int) (Math.*random*() \* height)));  
 this.validate();  
 this.repaint();  
 count1++;  
 }  
 countTime1++;  
 }  
  
 if (time - (countTime2 \* N2) >= N2) {  
 if (Math.*random*() <= P2) {  
 this.add(new Motocycle((int) (Math.*random*() \* width), (int) (Math.*random*() \* height)));  
 this.validate();  
 this.repaint();  
 count2++;  
 }  
 countTime2++;  
 }  
 }  
  
 public void fin(long time) {  
 this.removeAll();  
 String text = String.*format*("<html>Cars: %d<br> Motocycle: %d<br>Time: %.02f</html>", count1, count2, time / 1000.f);  
 JLabel textLabel = new JLabel(text);  
 textLabel.setLocation(0, 0);  
 textLabel.setFont(textLabel.getFont().deriveFont(20f));  
 textLabel.setForeground(Color.*WHITE*);  
 this.setLayout(new BorderLayout());  
 this.add(textLabel, BorderLayout.*NORTH*);  
 count1 = count2 = 0;  
 countTime1 = countTime2 = 0;  
 this.revalidate();  
 this.repaint();  
 }  
  
 public void start() {  
 this.removeAll();  
 this.setLayout(null);  
 this.revalidate();  
 this.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 + ") ");  
 }  
  
 }