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: «Технологии и методы программирования»

Основы программирования на Java.

Обработка событий. Механизм делегирования событий.

Вариант 8

Выполнили: Павлюк А.С. Преподаватель: Копылова О.А.

Казанцев К.О.

Группа: АВТ-008

Факультет: АВТФ

Новосибирск, 2022

**Цель работы:**

Познакомиться с особенностями технологии Java и изучить синтаксис языка Java. Изучить основные понятия и термины обработки событий в модели делегирования событий на Java.

**Задание к лабораторной работе:**

Разработать программу. Основная задача – разработка упрощенной имитации поведения объектов (все последующие лабораторные работы будут расширять это задание). Объекты реализуются через наследование: абстрактный класс+ интерфейс → наследники.

Рабочий цикл программы:

- запускается процесс симуляции по клавише, генерируются объекты классов согласно заданию;

- симуляция завершается по другой клавише, выводится статистическая информация.

- Разработать абстрактный класс объекта, согласно варианту индивидуального задания.

- Создать интерфейс IBehaviour, задающий поведение объекта (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 – схематично, плюсом будет, если объект будет похож на оригинал (можно 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 уничтожается. Время симуляции должно отображаться текстом в области визуализации и скрываться/показываться по клавише T;

По завершению симуляции в поле визуализации должна выводиться 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.

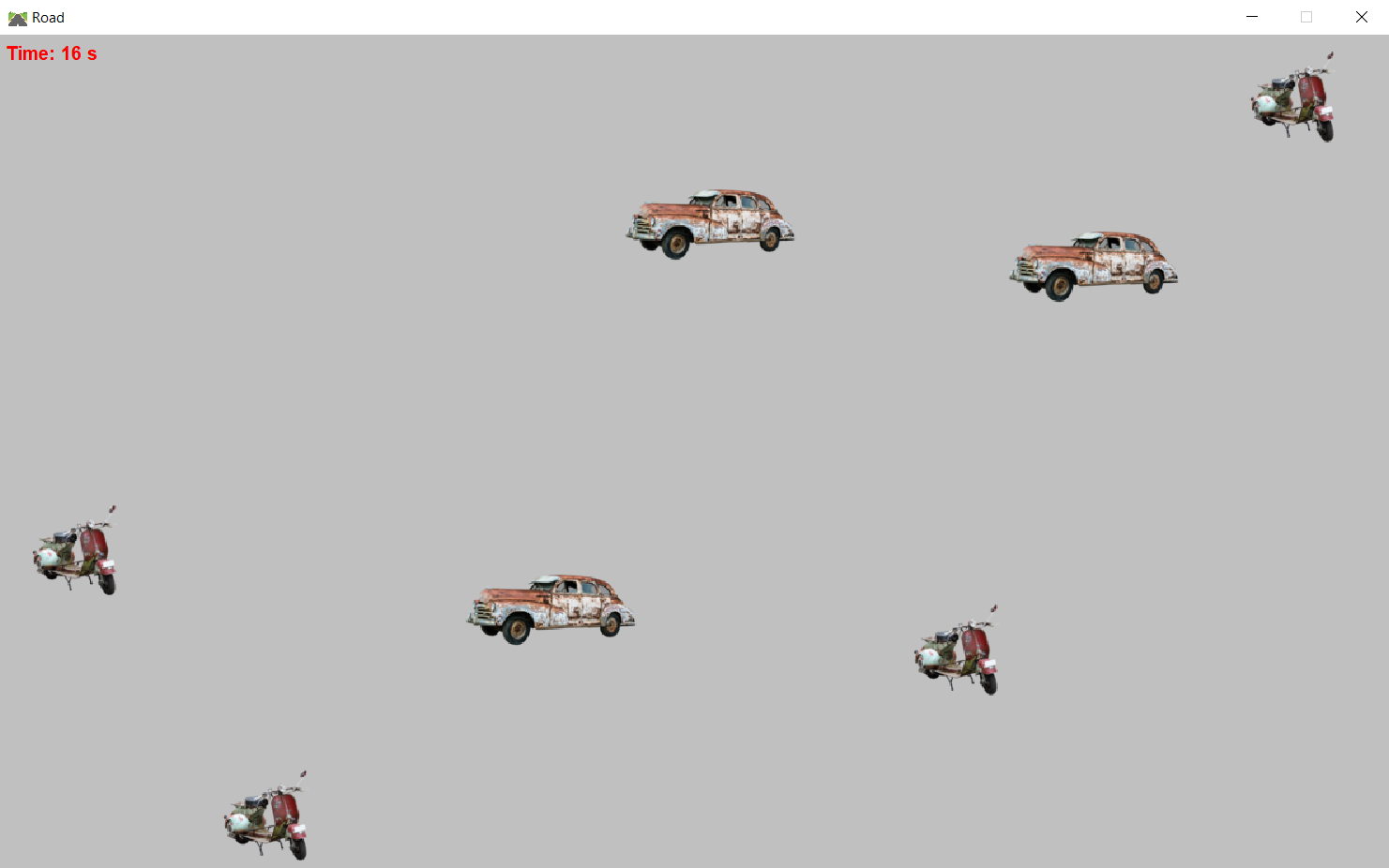
*Вариант 8*

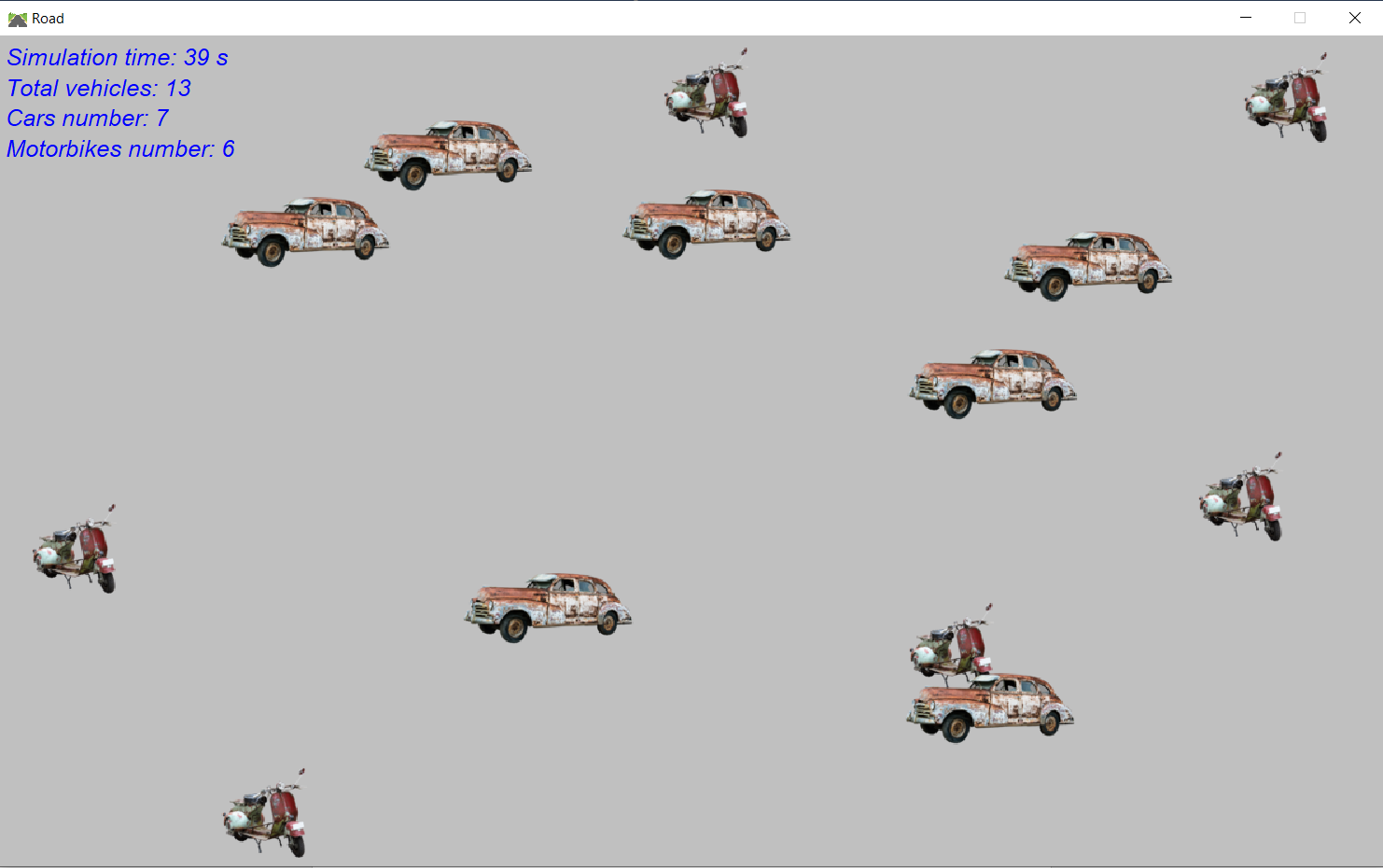
Список транспортных средств на дороге состоит из двух категорий: автомобили и мотоциклы. Автомобили генерируются каждые N1 секунд с вероятностью P1. Мотоциклы генерируются каждые N2 секунд с вероятностью P2.

**Структурное описание:**

* интерфейс IBehaviour;
* абстрактный класс Vehicle – в нем определены базовые поля и методы генерируемых объектов;
* класс машин Car, наследуемый от класса Vehicle и реализующий интерфейс IBehaviour;
* класс мотоциклов Motorbike, наследуемый от класса Vehicle и реализующий интерфейс IBehaviour;
* класс Habitat, задающий параметры рабочей области. В нем написан метод update, использующийся для таймера.
* класс MyPanel – наследник JPanel, в котором добавлена логика отрисовки объектов;
* класс MyFrame. В нем прописана визуализация симуляции и модель делегирования событий;
* класс Main – первичный класс приложения.

**Пример выполнения работы**

****



**Вывод:** в ходе выполнения лабораторной работы мы изучили синтаксис языка Java, познакомились с его особенностями. Нами были изучены основные понятия и термины обработки событий в модели делегирования событий на Java. Также мы разработали упрощенную имитацию поведения транспортных средств.

**Приложение**

**Интерфейс IBehaviour**

**package** nstu.vehicles;  
  
**public interface** IBehaviour {}

**Абстрактный класс Vehicle**

**package** nstu.vehicles;  
  
**import** javax.swing.\*;  
  
**public abstract class** Vehicle {  
 **int x**, **y**;  
 ImageIcon **image**;  
  
 **public int** getX() {  
 **return x**;  
 }  
  
 **public void** setX(**int** x) {  
 **this**.**x** = x;  
 }  
  
 **public int** getY() {  
 **return y**;  
 }  
  
 **public void** setY(**int** y) {  
 **this**.**y** = y;  
 }  
  
 **public** ImageIcon getImg() {  
 **return image**;  
 }  
}

**Класс Car**

**package** nstu.vehicles;  
  
**import** javax.swing.\*;  
  
**public class** Car **extends** Vehicle **implements** IBehaviour {  
 **public** Car(**int** x, **int** y) {  
 **this**.**image** = **new** ImageIcon(**"JavaLabs/src/nstu/imgs/car.png"**);  
 **this**.**x** = x;  
 **this**.**y** = y;  
 }  
}

**Класс Motorbike**

**package** nstu.vehicles;  
  
**import** javax.swing.\*;  
  
**public class** Motorbike **extends** Vehicle **implements** IBehaviour {  
 **public** Motorbike(**int** x, **int** y) {  
 **this**.**image** = **new** ImageIcon(**"JavaLabs/src/nstu/imgs/moto.png"**);  
 **this**.**x** = x;  
 **this**.**y** = y;  
 }  
}

**Класс Habitat**

**package** nstu;  
  
**import** nstu.vehicles.Car;  
**import** nstu.vehicles.Motorbike;  
**import** nstu.vehicles.Vehicle;  
  
**import** java.util.ArrayList;  
**import** java.util.List;  
**import** java.util.Random;  
  
**public class** Habitat {  
 **final int WIDTH** = 1200;  
 **final int HEIGHT** = 750;  
 **final int N1** = 3, **N2** = 4;  
 **final int P1** = 70, **P2** = 60;  
 **int carCount** = 0;  
 **int motoCount** = 0;  
 Random **random** = **new** Random();  
  
 **static** List<Vehicle> *vehicles* = **new** ArrayList<>();  
  
 **public void** update(**double** time) {  
 **if** (**random**.nextInt(100) < **P1** && (**int**) time % **N1** == 0) {  
 Car car = **new** Car(**random**.nextInt(**WIDTH**-150), **random**.nextInt(**HEIGHT**-98));  
 *vehicles*.add(car);  
 System.***out***.println(**"+car{"** + car.getX() + **"; "** + car.getY() + **"}"**);  
 **carCount**++;  
 }  
 **if** (**random**.nextInt(100) < **P2** && (**int**) time % **N2** == 0) {  
 Motorbike motorbike = **new** Motorbike(**random**.nextInt(**WIDTH**-100), **random**.nextInt(**HEIGHT**-114));  
 *vehicles*.add(motorbike);  
 System.***out***.println(**"+motorbike{"** + motorbike.getX() + **"; "** + motorbike.getY() + **"}"**);  
 **motoCount**++;  
 }  
 }  
}

**Класс MyPanel**

**package** nstu;  
  
**import** nstu.vehicles.Vehicle;  
  
**import** javax.swing.\*;  
**import** java.awt.\*;  
  
**class** MyPanel **extends** JPanel {  
 @Override  
 **protected void** paintComponent(Graphics g){  
 **super**.paintComponent(g);  
 **for** (Vehicle v : Habitat.*vehicles*) {  
 g.drawImage(v.getImg().getImage(), v.getX(), v.getY(), **null**);  
 }  
 }  
  
}

**Класс MyFrame**

**package** nstu;  
  
**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.awt.event.KeyAdapter;  
**import** java.awt.event.KeyEvent;  
**import** java.util.Timer;  
**import** java.util.TimerTask;  
  
**public class** MyFrame **extends** JFrame {  
 **boolean willShowStatsLabel** = **false**;  
 **boolean willShowTime** = **false**;  
 **boolean isStarted** = **false**;  
 Habitat **h** = **new** Habitat();  
 **long time**;  
  
 **public** MyFrame() {  
 **super**(**"Road"**);  
 setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  
 setIconImage(**new** ImageIcon(**"JavaLabs/src/nstu/imgs/icon.png"**).getImage());  
 Toolkit toolkit = Toolkit.*getDefaultToolkit*();  
 Dimension dimension = toolkit.getScreenSize();  
 setBounds(dimension.**width**/2 - **h**.**WIDTH**/2, dimension.**height**/2 - **h**.**HEIGHT**/2, **h**.**WIDTH**, **h**.**HEIGHT**);  
 setFocusable(**true**);  
  
 setResizable(**false**);  
 JPanel scene = **new** JPanel();  
 add(scene);  
 scene.setLayout(**new** BorderLayout());  
  
 MyPanel road = **new** MyPanel();  
 road.setLayout(**new** FlowLayout(FlowLayout.***LEFT***));  
 road.setBackground(Color.***LIGHT\_GRAY***);  
 scene.add(road);  
  
 JLabel timeLabel = **new** JLabel();  
 timeLabel.setFont(**new** Font(**"JavaLabs/fonts/ttf/JetBrainsMono-Regular.ttf"**, Font.***BOLD***, 16));  
 timeLabel.setForeground(Color.***RED***);  
 timeLabel.setVisible(**false**);  
 road.add(timeLabel);  
  
 JLabel statsLabel = **new** JLabel();  
 statsLabel.setFont(**new** Font(**"JavaLabs/fonts/ttf/JetBrainsMono-Regular.ttf"**, Font.***ITALIC***, 20));  
 statsLabel.setForeground(Color.***BLUE***);  
 statsLabel.setVisible(**false**);  
 road.add(statsLabel);  
  
 addKeyListener(**new** KeyAdapter() {  
 Timer **timer**;  
  
 @Override  
 **public void** keyPressed(KeyEvent e) {  
 **switch** (e.getKeyChar()) {  
 **case 'b'** -> {  
 **if** (!**isStarted**) {  
 System.***out***.println(**"---------------------------"**);  
 System.***out***.println(**"Car: chance - "** + **h**.**P1** + **"%, time - "** + **h**.**N1** +  
 **"\nBike: chance - "** + **h**.**P2** + **"%, time - "** + **h**.**N2**);  
 repaint();  
 **if** (**willShowStatsLabel**) {  
 statsLabel.setVisible(**false**);  
 **willShowStatsLabel** = **false**;  
 }  
  
 **isStarted** = **true**;  
 **h**.**carCount** = 0;  
 **h**.**motoCount** = 0;  
 **timer** = **new** Timer();  
 **time** = 0;  
 **timer**.schedule(**new** TimerTask() {  
 @Override  
 **public void** run() {  
 **h**.update(**time**);  
 timeLabel.setText(**"<html><p>Time: "** + **time** + **" s</html>"**);  
 add(**new** JComponent(){});  
 **time**++;  
 }  
 }, 0, 1000);  
 }  
 }  
 **case 'e'** -> {  
 **if** (**isStarted**) {  
 **timer**.cancel();  
 **isStarted** = **false**;  
 }  
 **if** (!**willShowStatsLabel**) {  
 **if** (**willShowTime**) {  
 timeLabel.setVisible(**false**);  
 **willShowTime** = **false**;  
 }  
 statsLabel.setText(**"<html>"** +  
 **"<p>Simulation time: "** + **time** + **" s"** +  
 **"<p>Total vehicles: "** + Habitat.*vehicles*.size() +  
 **"<p>Cars number: "** + **h**.**carCount** +  
 **"<p>Motorbikes number: "** + **h**.**motoCount** +  
 **"</html>"** );  
 statsLabel.setVisible(**true**);  
 **willShowStatsLabel** = **true**;  
 Habitat.*vehicles*.clear();  
 }  
 }  
 **case 't'** -> {  
 **if** (!**willShowStatsLabel**) {  
 **if** (!**willShowTime**) {  
 timeLabel.setVisible(**true**);  
 **willShowTime** = **true**;  
 } **else** {  
 timeLabel.setVisible(**false**);  
 **willShowTime** = **false**;  
 }  
 }  
 }  
 **default** -> {}  
 }  
 }  
 });  
  
 setVisible(**true**);  
 }  
}

**Класс Main**

**package** nstu;  
  
**public class** Main {  
  
 **public static void** main(String[] args) {  
 **new** MyFrame();  
 }  
}