МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

**Лабораторная работа №1**

по дисциплине «Технология программирования»

на тему: «Основы программирования на Java. Обработка событий.

Механизм делегирования событий»

|  |  |
| --- | --- |
| Группа: АВТ-808 | Преподаватель: Михайленко Дмитрий Анатольевич |
| Студент: Кузьмина Александра |  |
| Вариант: 1 |  |
|  |  |

Новосибирск, 2020

**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**

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.

**Вариант 1**

Объект – муравей. Бывают 2 видов: рабочий и воин. Рабочие рождаются каждые N1 секунд с вероятностью P1. Воины рождаются каждые N2 секунд с вероятностью P2.

**Ход работы**

Для решения задачи были разработаны следующие классы:

Интерфейс **IBehaviour** задает поведение объекта.



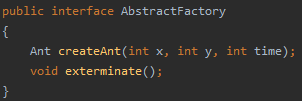
Абстрактный класс **Ant**, имплементирующий интерфейс IBehaviour. Он представляет собой модель объекта муравья и содержит в себе изображение и координаты расположения муравья.



Классы **AntWorker** и **AntWarrio**r, наследники класса Ant. Модели муравья-рабочего и муравья-воина. Каждому виду модели задается свое изображение:

Интерфейс **AbstractFactory**, содержащий метод createAnt для создания объектов классов и exterminate для очистки счетчиков созданных объектов.



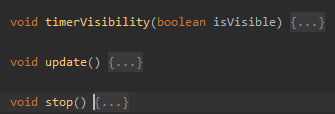
Класс **ConcreteFactory**, реализованный от интерфейса AbstractFactory и реализующий его методы createAnt и exterminate. Метод создания муравья возвращает нужный объект согласно варианту задания.



Класс **Habitat** – среда, определяющий размер рабочей и хранящий массив объектов муравьев. В данном классе реализованы следующие методы:

* update() – метод, отвечающий за таймер и генерацию новых объектов и их помещение в поле визуализации в случайном месте;
* stop() – метод, останавливающий симуляцию. При этом выводится информационное окно со статистикой по созданным объектам;
* timerVisibility(boolean isVisible) – этот метод позволяет скрывать или показывать таймер во время симуляции.





Для графической отрисовки объектов был создан класс **DrawingAnts**.



Класс **MyTimerTask**, содержащий переопределенный метод run(). С помощью него можно задать периодичность отрисовки среды.



Класс **Main** – основной класс, запускающий работу программы. Здесь перехватываются и обрабатываются такие события:

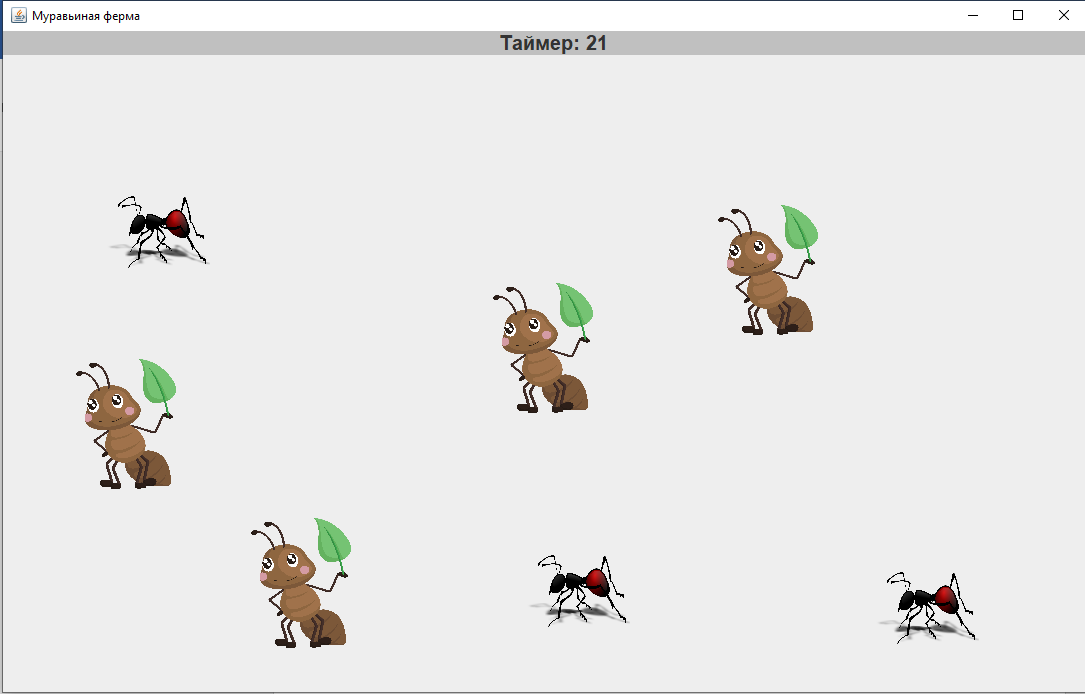
* нажатие клавиши **B** запускает симуляцию;
* по клавише **E** симуляция останавливается;
* при нажатии клавиши **T** показывается/скрывается время симуляции.

**Скриншоты работы программы**

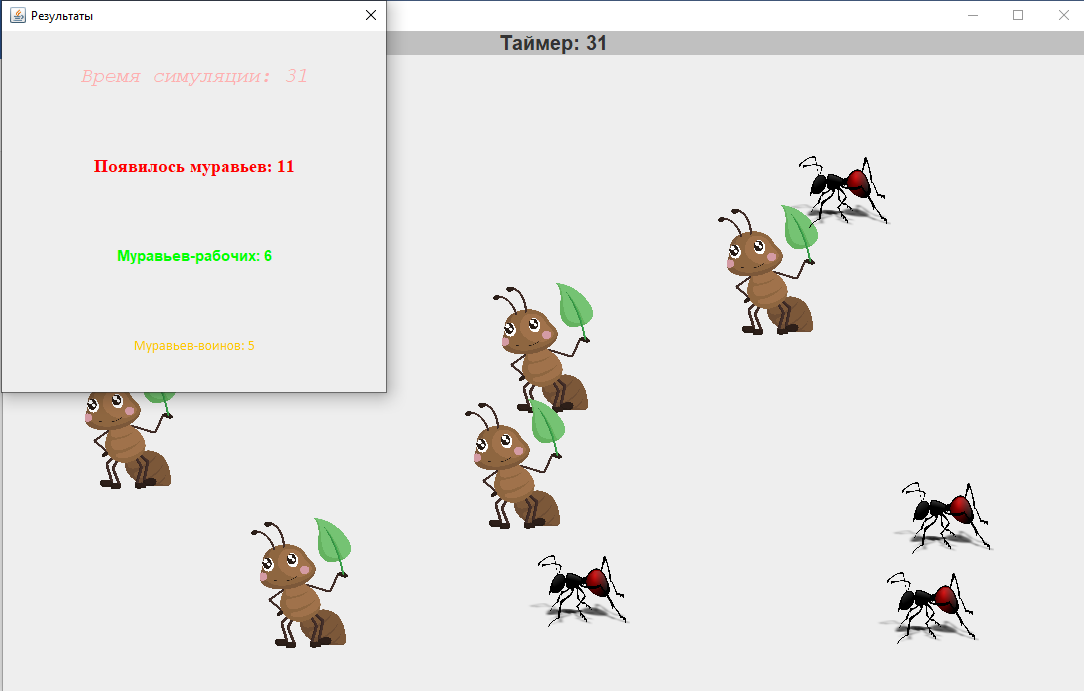
Окно до запуска симуляции:

****

Во время работы:



Информация после окончания симуляции:



**Вывод**

При выполнении данной лабораторной работы мною были изучены основные понятия и термины обработки событий в модели делегирования событий. Использовав графические библиотеки AWT и Swing, была разработана программа в соответствии с заданием. Данная программа является упрощенной имитацией поведения муравьев (воинов и рабочих).

**Листинг программы**

**IBehaviour**

package AntFarm;  
  
import java.awt.\*;  
  
public interface IBehaviour  
{  
 void setX(int x);  
 int getX();  
 void setY(int y);  
 int getY();  
 void setImage(Image image);  
 Image getImage();  
}

**Ant**

package AntFarm;  
  
import java.awt.\*;  
  
public abstract class Ant implements IBehaviour  
{  
 protected int x, y;  
 protected Image image;  
  
 protected Ant(int x, int y)  
 {  
 this.x = x;  
 this.y = y;  
 }  
  
 @Override  
 public void setX(int x)  
 {  
 this.x = x;  
 }  
  
 @Override  
 public int getX()  
 {  
 return x;  
 }  
  
 @Override  
 public void setY(int y)  
 {  
 this.y = y;  
 }  
  
 @Override  
 public int getY()  
 {  
 return y;  
 }  
  
 @Override  
 public void setImage(Image image)  
 {  
 this.image = image;  
 }  
  
 @Override  
 public Image getImage()  
 {  
 return image;  
 }  
}

**AntWarrior**

package AntFarm;  
  
import javax.imageio.ImageIO;  
import java.io.IOException;  
  
public class AntWarrior extends Ant  
{  
 AntWarrior(int x, int y) {  
 super(x, y);  
 try {  
 this.image = ImageIO.*read*(getClass().getResource("/AntWarrior.png"));  
 }  
 catch (IOException e)  
 {  
 e.printStackTrace();  
 }  
 }  
}

**AntWorker**

package AntFarm;  
  
import javax.imageio.ImageIO;  
import java.io.IOException;  
  
public class AntWorker extends Ant  
{  
 AntWorker(int x, int y) {  
 super(x, y);  
 try {  
 this.image = ImageIO.*read*(getClass().getResource("/AntWorker.png"));  
 }  
 catch (IOException e)  
 {  
 e.printStackTrace();  
 }  
 }  
}

**AbstractFactory**

package AntFarm;  
  
public interface AbstractFactory  
{  
 Ant createAnt(int x, int y, int time);  
 void exterminate();  
}

**ConcreteFactory**

package AntFarm;  
  
import java.util.Random;  
  
public class ConcreteFactory implements AbstractFactory  
{  
 private int N1 = 5;  
 private int N2 = 3;  
 private double P1 = 0.9;  
 private double P2 = 0.7;  
 private int antsAmount = 0;  
 private int workersAmount = 0;  
 private int warriorsAmount = 0;  
  
 public ConcreteFactory() {}  
  
 @Override  
 public Ant createAnt(int x, int y, int time) {  
 if (time % N1 == 0)  
 {  
 Random random = new Random();  
 double probability = random.nextDouble();  
 if (P1 > probability)  
 {  
 Ant ant = new AntWorker(x, y);  
 workersAmount++; antsAmount++;  
 return ant;  
 }  
 }  
 if (time % N2 == 0) {  
 Random random = new Random();  
 double probability = random.nextDouble();  
 if (P2 > probability) {  
 Ant ant = new AntWarrior(x, y);  
 warriorsAmount++;  
 antsAmount++;  
 return ant;  
 }  
 }  
 return null;  
 }  
  
 @Override  
 public void exterminate()  
 {  
 antsAmount = 0;  
 workersAmount = 0;  
 warriorsAmount = 0;  
 }  
}

**MyTimeTask**

package AntFarm;  
  
import java.util.TimerTask;  
  
public class MyTimerTask extends TimerTask {  
  
 private Habitat h;  
  
 MyTimerTask(Habitat h)  
 {  
 this.h = h;  
 }  
  
 @Override  
 public void run()  
 {  
 h.update();  
 }  
}

**DrawingAnts**

package AntFarm;  
  
import javax.swing.\*;  
import java.awt.\*;  
  
public class DrawingAnts extends JPanel  
{  
 private Ant[] ants;  
  
 public DrawingAnts(Ant[] ants)  
 {  
 this.ants = ants;  
 }  
  
 @Override  
 protected void paintComponent(Graphics g)  
 {  
 super.paintComponent(g);  
 if (ants[0] == null)  
 return;  
 for (int i = 0; ants[i] != null && i<500; i++)  
 {  
 if (ants[i].getClass() == AntWorker.class)  
 g.drawImage(ants[i].getImage(), ants[i].getX(), ants[i].getY(), 100, 130, null);  
 else  
 g.drawImage(ants[i].getImage(), ants[i].getX(), ants[i].getY(), 150, 170, null);  
 }  
 }  
}

**Habitat**

package AntFarm;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.io.IOException;  
import java.util.Random;  
  
public class Habitat extends JFrame {  
 private int time;  
 private int antsAmount, workersAmount, warriorsAmount;  
 private Ant[] ants;  
 private AbstractFactory factory;  
 private JLabel timer;  
 private DrawingAnts antDraw;  
  
 Habitat() {  
 super("Муравьиная ферма");  
 int width = 1100, height = 700;  
 setSize(width, height);  
 setLayout(null);  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
  
 antsAmount = 0;  
 workersAmount = 0;  
 warriorsAmount = 0;  
 ants = new Ant[500];  
 factory = new ConcreteFactory();  
 time = 0;  
  
 JPanel timerPanel = new JPanel();  
 timerPanel.setLayout(new GridLayout(1, 1));  
 timerPanel.setBackground(Color.*lightGray*);  
 timerPanel.setBounds(0, 0, width, 24);  
  
 timer = new JLabel("Таймер: 0");  
 timer.setHorizontalAlignment(SwingConstants.*CENTER*);  
 timer.setFont(new Font("Arial", Font.*BOLD*, 20));  
 timerPanel.add(timer);  
 add(timerPanel);  
  
 antDraw = new DrawingAnts(ants);  
 antDraw.setBounds(0, 24, width, height - 24);  
 add(antDraw);  
 }  
  
 void timerVisibility(boolean isVisible) {  
 if (isVisible)  
 timer.setVisible(false);  
 else  
 timer.setVisible(true);  
 }  
  
 void update() {  
 time++;  
 timer.setText("Таймер: " + time);  
 Random random = new Random();  
 int xcord = random.nextInt(antDraw.getWidth() - 200);  
 int ycord = random.nextInt(antDraw.getHeight() - 200);  
 Ant ant = factory.createAnt(xcord, ycord, time);  
 if (ant != null) {  
 if (ant.getClass() == AntWorker.class)  
 workersAmount++;  
 else  
 warriorsAmount++;  
 ants[antsAmount] = ant;  
 antsAmount++;  
 antDraw.repaint();  
 }  
 }  
  
 void stop() {  
 JDialog results = new JDialog(this, "Результаты");  
  
 JLabel timerCount = new JLabel("Время симуляции: " + time);  
 timerCount.setFont(new Font("Courier New", Font.*ITALIC*,20));  
 timerCount.setForeground(Color.*PINK*);  
 timerCount.setHorizontalAlignment(SwingConstants.*CENTER*);  
 results.add(timerCount);  
  
 JLabel str = new JLabel("Появилось муравьев: " + antsAmount);  
 str.setFont(new Font("Times New Roman", Font.*BOLD*, 18));  
 str.setForeground(Color.*RED*);  
 str.setHorizontalAlignment(SwingConstants.*CENTER*);  
 results.add(str);  
  
 JLabel workersCount = new JLabel("Муравьев-рабочих: " + workersAmount);  
 workersCount.setFont(new Font("Arial", Font.*BOLD*,15));  
 workersCount.setForeground(Color.*GREEN*);  
 workersCount.setHorizontalAlignment(SwingConstants.*CENTER*);  
 results.add(workersCount);  
  
 JLabel warriorsCount = new JLabel("Муравьев-воинов: " + warriorsAmount);  
 warriorsCount.setFont(new Font("Calibri", Font.*PLAIN*,14));  
 warriorsCount.setForeground(Color.*ORANGE*);  
 warriorsCount.setHorizontalAlignment(SwingConstants.*CENTER*);  
 results.add(warriorsCount);  
  
 results.setDefaultCloseOperation(WindowConstants.*DISPOSE\_ON\_CLOSE*);  
 results.setSize(400, 400);  
 results.setLayout(new GridLayout(4,1));  
 results.setModal(true);  
 results.setVisible(true);  
  
 factory.exterminate();  
 time = 0;  
 timer.setText("Таймер: " + time);  
 for (int i = 0; i <antsAmount; i++)  
 ants[i] = null;  
 antsAmount = 0;  
 workersAmount = 0; warriorsAmount = 0;  
 antDraw.repaint();  
 }  
}

**Main**

package AntFarm;  
  
import java.awt.event.KeyAdapter;  
import java.awt.event.KeyEvent;  
import java.util.Timer;  
  
public class Main extends KeyAdapter  
{  
 private Habitat antFarm;  
 private MyTimerTask myTimerTask;  
 private Timer timer;  
 private boolean isStarted, isVisible;  
  
 public static void main(String[] args)  
 {  
 new Main();  
 }  
  
 Main()  
 {  
 antFarm = new Habitat();  
 antFarm.setVisible(true);  
 antFarm.addKeyListener(this);  
 isStarted = false;  
 isVisible = true;  
 }  
  
 @Override  
 public void keyPressed(KeyEvent keyEvent)  
 {  
 int key = keyEvent.getKeyCode();  
 if (key == keyEvent.*VK\_B* && !isStarted)  
 {  
 isStarted = true;  
 timer = new Timer();  
 myTimerTask = new MyTimerTask(antFarm);  
 timer.schedule(myTimerTask, 0, 1000);  
 }  
 if (key == keyEvent.*VK\_E* && isStarted)  
 {  
 timer.cancel();  
 timer.purge();  
 myTimerTask.cancel();  
 antFarm.stop();  
 isStarted = false;  
 }  
 if (key == keyEvent.*VK\_T*)  
 {  
 antFarm.timerVisibility(isVisible);  
 isVisible = !isVisible;  
 }  
 }  
}