МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

**Лабораторная работа №2**

по дисциплине «Технология программирования»

на тему: «Разработка графического интерфейса приложения»

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

Новосибирск, 2020

**Цель работы**

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.

**Задание**

Для хранения генерируемых объектов использовать динамический массив объектов. Для обеспечения доступа всем элементам приложения и создания только одного экземпляра массива объектов используйте структурный паттерн Singleton.

При реализации шаблона Singleton используется класс Singleton. В этом классе определяется закрытый конструктор, имеется закрытая статическая переменная, в которой хранится ссылка на единственный экземпляр данного класса, а также определен статический метод доступа, возвращающий ссылку на этот экземпляр. Так же класс содержит объект коллекции и открытые методы доступа к ней. Таким образом можно в любом месте приложения получить ссылку на объект класса Singleton и доступ к коллекции объектов.

4. 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 в меню и панель инструментов, в которых продублировать основные команды вашего 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 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 комбобокс и список (шаг значений 10%);

- Дополнить интерфейс поясняющими метками.

**Ход работы**

В ходе решения поставленных задач доработки программы были добавлены следующие классы:

Структурный паттерн **Singleton**,используемый ля обеспечения доступа всем элементам приложения и создания только одного экземпляра массива объектов. В этом классе определяется закрытый конструктор, имеется закрытая статическая переменная, в которой хранится ссылка на единственный экземпляр данного класса, а также определен статический метод доступа, возвращающий ссылку на этот экземпляр. Так же класс содержит объект коллекции и открытые методы доступа к ней.

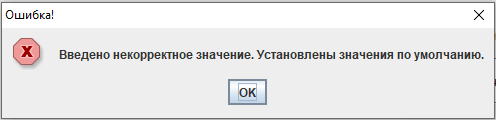


Класс **GUI**, отвечающий за графический интерфейс. Окно программы состоит из следующих структурных частей:

* Панель управления: здесь находятся кнопки “Старт” и “Стоп”, соответственно запускающие и останавливающие симуляцию; переключатель «Информация о симуляции», который разрешает отображение модального диалога; группа из 2 исключающих переключателей: «Показать время симуляции» и «Скрыть время симуляции»; текстовые поля для установки периодов рождения и комбобоксы для задания вероятностей рождения объектов с соответствующими поясняющими метками. Панель управления находится в правой части окна;
* Меню: в меню продублированы основные команды интерфейсы пользователя. Находится в верхней части программы;
* Таймер: ниже меню находится таймер, показывающий время симуляции. Можно отключить по кнопке **T** или соответствующим переключателям в меню и на панели управления;
* Окно симуляции: окно, где происходит симуляция объектов-муравьев. Занимает центр окна.

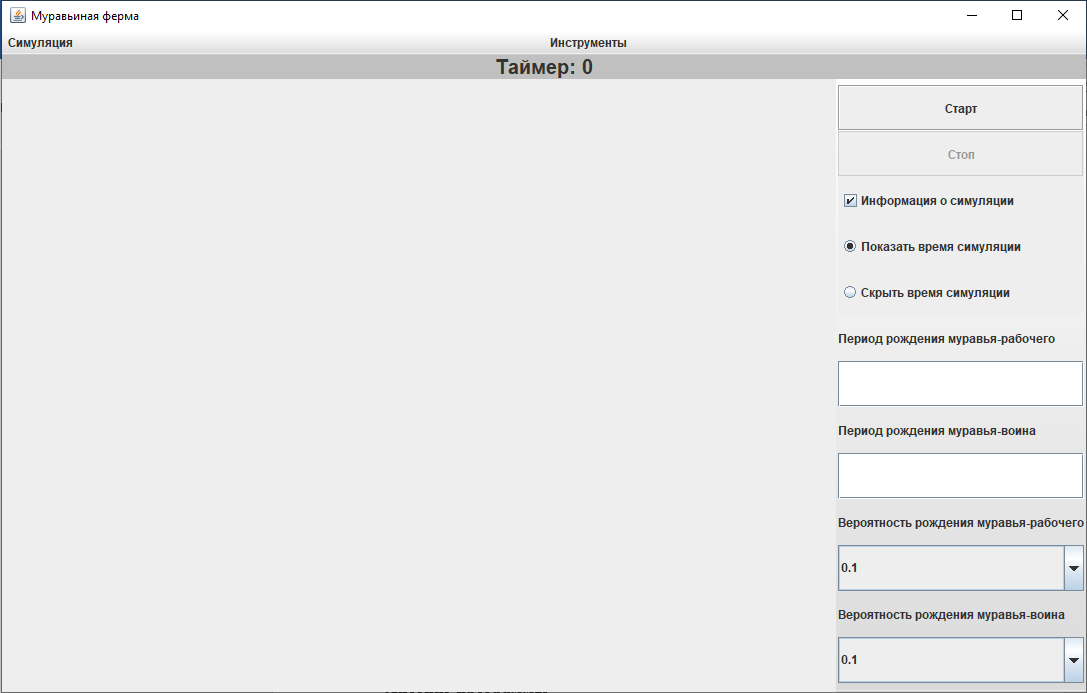
Остальные классы программы были переработаны в соответствии с новыми задачами.

На панели управления можно задавать параметры симуляции. Если в текстовых полях были введены некорректные значения, будут установлены параметры по умолчанию и выведено соответствующее окно.

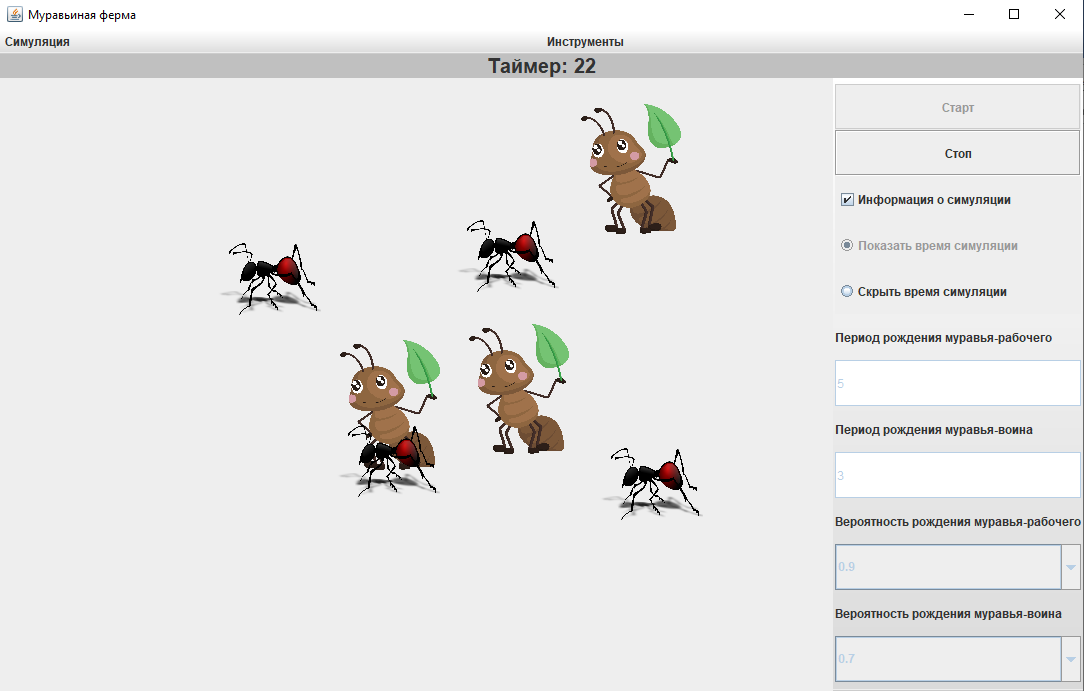


**Скриншоты работы программы**

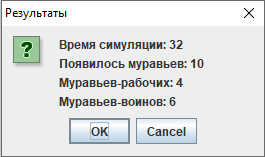
Окно до запуска симуляции:



Окно во время симуляции:



При остановке симуляции появляется (если установлен соответствующий переключатель) диалоговое окно, в котором отображается статистика по симуляции. При нажатии на клавишу “ОК” симуляция закончится, а при нажатии на клавишу “Отмена” – продолжится.



**Вывод**

При выполнении данной лабораторной работы были получены навыки по работе с графическими библиотеками Java – AWT и Swing и изучены их основные компоненты. Была доработана программа, созданная в лабораторной работе №1. Теперь пользователь может контролировать симуляцию, изменяя ее параметры.

**Листинг программы**

**Singleton**

package AntFarm;  
  
import java.util.ArrayList;  
  
public class Singleton  
{  
 private static Singleton *singleton*;  
 private ArrayList<Ant> ants;  
  
 private Singleton()  
 {  
 ants = new ArrayList<>();  
 }  
  
 public int getArraySize()  
 {  
 return ants.size();  
 }  
  
 public static Singleton getSingleton()  
 {  
 if(*singleton* == null)  
 {  
 *singleton* = new Singleton();  
 }  
 return *singleton*;  
 }  
  
 public static void addAnt(Ant ant)  
 {  
 *singleton*.ants.add(ant);  
 }  
  
 public static void clearAnts()  
 {  
 if (!*singleton*.ants.isEmpty())  
 *singleton*.ants.clear();  
 }  
  
 public static ArrayList<Ant> getAnts(){  
 return *singleton*.ants;  
 }  
}

**GUI**

package AntFarm;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionListener;  
  
class GUI extends JPanel  
{  
 protected JMenuBar menu;  
 protected JToolBar settings;  
 protected DrawingAnts draw;  
 protected JPanel timerPanel;  
 protected JPanel simulation;  
 protected JLabel timer;  
  
 //боковая панель управления  
 private JButton buttonStart;  
 private JButton buttonStop;  
 private JCheckBox infoVisibility;  
 private ButtonGroup timerVisibility;  
 private JRadioButton timerIsVisible;  
 private JRadioButton timerNotVisible;  
 private JLabel labelWorkerPeriod;  
 private JLabel labelWarriorPeriod;  
 private JLabel labelWorkerProbability;  
 private JLabel labelWarriorProbability;  
 private JTextField workerPeriod;  
 private JTextField warriorPeriod;  
 private JComboBox workerProbability;  
 private JComboBox warriorProbability;  
  
 //меню  
 private JMenu menuSimulation;  
 private JMenu menuInstruments;  
 private JMenuItem menuStart;  
 private JMenuItem menuStop;  
 private JMenuItem menuTimerVisible;  
 private JMenuItem menuInformationVisible;  
  
 public GUI(ActionListener actionListener)  
 {  
 menu = new JMenuBar();  
 settings = new JToolBar();  
  
 settings.setLayout(new GridLayout(13,1));  
 settings.setFocusable(false);  
 settings.setFloatable(false);  
  
 buttonStart = new JButton("Старт");  
 buttonStart.addActionListener(actionListener);  
  
 buttonStop = new JButton("Стоп");  
 buttonStop.setEnabled(false);  
 buttonStop.addActionListener(actionListener);  
  
 infoVisibility = new JCheckBox("Информация о симуляции", true);  
 infoVisibility.addActionListener(actionListener);  
  
 timerIsVisible = new JRadioButton("Показать время симуляции");  
 timerIsVisible.setSelected(true);  
 timerIsVisible.addActionListener(actionListener);  
 timerNotVisible = new JRadioButton("Скрыть время симуляции");  
 timerNotVisible.addActionListener(actionListener);  
  
 timerVisibility = new ButtonGroup();  
 timerVisibility.add(timerIsVisible);  
 timerVisibility.add(timerNotVisible);  
  
 labelWorkerPeriod = new JLabel("Период рождения муравья-рабочего");  
 workerPeriod = new JTextField();  
  
 labelWarriorPeriod = new JLabel("Период рождения муравья-воина");  
 warriorPeriod = new JTextField();  
  
 Double[] steps = new Double[] {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0};  
  
 labelWorkerProbability = new JLabel("Вероятность рождения муравья-рабочего");  
 workerProbability = new JComboBox(steps);  
  
 labelWarriorProbability = new JLabel("Вероятность рождения муравья-воина");  
 warriorProbability = new JComboBox(steps);  
  
 settings.add(buttonStart);  
 settings.add(buttonStop);  
 settings.add(infoVisibility);  
 settings.add(timerIsVisible);  
 settings.add(timerNotVisible);  
 settings.add(labelWorkerPeriod);  
 settings.add(workerPeriod);  
 settings.add(labelWarriorPeriod);  
 settings.add(warriorPeriod);  
 settings.add(labelWorkerProbability);  
 settings.add(workerProbability);  
 settings.add(labelWarriorProbability);  
 settings.add(warriorProbability);  
  
 menu.setLayout(new GridLayout(1, 2));  
 menu.setFocusable(false);  
  
 menuSimulation = new JMenu("Симуляция");  
 menuInstruments = new JMenu("Инструменты");  
  
 menuStart = new JMenuItem("Старт");  
 menuStart.addActionListener(actionListener);  
 menuStop = new JMenuItem("Стоп");  
 menuStop.addActionListener(actionListener);  
 menuTimerVisible = new JMenuItem("Показать/скрыть время симуляции");  
 menuTimerVisible.addActionListener(actionListener);  
 menuInformationVisible = new JMenuItem("Показать/скрыть информацию о симуляции");  
 menuInformationVisible.addActionListener(actionListener);  
  
 menuSimulation.add(menuStart);  
 menuSimulation.add(menuStop);  
 menuInstruments.add(menuTimerVisible);  
 menuInstruments.add(menuInformationVisible);  
 menu.add(menuSimulation);  
 menu.add(menuInstruments);  
  
 timerPanel = new JPanel();  
 timerPanel.setLayout(new GridLayout(1, 1));  
 timerPanel.setBackground(Color.*lightGray*);  
  
 timer = new JLabel("Таймер: 0");  
 timer.setHorizontalAlignment(SwingConstants.*CENTER*);  
 timer.setFont(new Font("Arial", Font.*BOLD*, 20));  
 timerPanel.add(timer);  
  
 simulation = new JPanel();  
 simulation.setLayout(new GridLayout(2, 1));  
 simulation.add(menu);  
 simulation.add(timerPanel);  
  
 draw = new DrawingAnts();  
  
 setLayout(new BorderLayout());  
 add(simulation, BorderLayout.*PAGE\_START*);  
 add(settings, BorderLayout.*LINE\_END*);  
  
 add(draw, BorderLayout.*CENTER*);  
 }  
  
 public JButton getButtonStart()  
 {  
 return buttonStart;  
 }  
  
 public JButton getButtonStop()  
 {  
 return buttonStop;  
 }  
  
 public JCheckBox getInfoVisibility()  
 {  
 return infoVisibility;  
 }  
  
 public JRadioButton getTimerIsVisible()  
 {  
 return timerIsVisible;  
 }  
  
 public JRadioButton getTimerNotVisible()  
 {  
 return timerNotVisible;  
 }  
  
 public JTextField getWorkerPeriod()  
 {  
 return workerPeriod;  
 }  
  
 public JTextField getWarriorPeriod()  
 {  
 return warriorPeriod;  
 }  
  
 public JComboBox getWorkerProbability()  
 {  
 return workerProbability;  
 }  
  
 public JComboBox getWarriorProbability()  
 {  
 return warriorProbability;  
 }  
  
 public JMenuItem getMenuStart()  
 {  
 return menuStart;  
 }  
  
 public JMenuItem getMenuStop()  
 {  
 return menuStop;  
 }  
  
 public JMenuItem getMenuTimerVisible()  
 {  
 return menuTimerVisible;  
 }  
  
 public JMenuItem getMenuInformationVisible()  
 {  
 return menuInformationVisible;  
 }  
}

**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 setParameters(int N1, int N2, double P1, double P2);  
 void exterminate();  
}

**ConcreteFactory**

package AntFarm;  
  
import java.util.Random;  
  
public class ConcreteFactory implements AbstractFactory  
{  
 private int N1;  
 private int N2;  
 private double P1;  
 private double P2;  
 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 setParameters(int N1, int N2, double P1, double P2)  
 {  
 this.N1 = N1;  
 this.N2 = N2;  
 this.P1 = P1;  
 this.P2 = P2;  
 }  
 @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 Singleton ants;  
  
 public DrawingAnts()  
 {  
 ants = Singleton.*getSingleton*();  
 }  
  
 @Override  
 protected void paintComponent(Graphics g)  
 {  
 super.paintComponent(g);  
 if (Singleton.*getAnts*().isEmpty())  
 return;  
 for (int i = 0; i< ants.getArraySize() && i<500; i++)  
 {  
 if (ants.*getAnts*().get(i).getClass() == AntWorker.class)  
 g.drawImage(ants.*getAnts*().get(i).getImage(), ants.*getAnts*().get(i).getX(), ants.*getAnts*().get(i).getY(), 100, 130, null);  
 else  
 g.drawImage(ants.*getAnts*().get(i).getImage(), ants.*getAnts*().get(i).getX(), ants.*getAnts*().get(i).getY(), 150, 170, null);  
 }  
 }  
}

**Habitat**

package AntFarm;  
  
import javax.swing.\*;  
import java.awt.event.ActionListener;  
import java.util.Random;  
  
public class Habitat extends JFrame {  
 private int time;  
 private int antsAmount, workersAmount, warriorsAmount;  
 private Singleton ants;  
 protected AbstractFactory factory;  
 private GUI gui;  
 private boolean isInformationShowing;  
 private boolean isOver, isPaused;  
  
 Habitat(ActionListener actionListener)  
 {  
 super("Муравьиная ферма");  
 int width = 1100, height = 700;  
 setSize(width, height);  
 setFocusable(true);  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
  
 antsAmount = 0;  
 workersAmount = 0;  
 warriorsAmount = 0;  
 ants = Singleton.*getSingleton*();  
 factory = new ConcreteFactory();  
 time = 0;  
 isInformationShowing = true;  
 isOver = true; isPaused = false;  
  
 gui = new GUI(actionListener);  
 add(gui);  
 }  
  
 void timerVisibility(boolean isTimerVisible) {  
 if (isTimerVisible)  
 gui.timer.setVisible(false);  
 else  
 gui.timer.setVisible(true);  
 }  
  
 void informationShowing()  
 {  
 if (isInformationShowing)  
 gui.getInfoVisibility().setSelected(false);  
 else  
 gui.getInfoVisibility().setSelected(true);  
 isInformationShowing = !isInformationShowing;  
 }  
  
 void update() {  
 if (!isPaused)  
 {  
 time++;  
 gui.timer.setText("Таймер: " + time);  
 Random random = new Random();  
 int xcord = random.nextInt(gui.draw.getWidth() - 200);  
 int ycord = random.nextInt(gui.draw.getHeight() - 200);  
 Ant ant = factory.createAnt(xcord, ycord, time);  
 if (ant != null) {  
 if (ant.getClass() == AntWorker.class)  
 workersAmount++;  
 else  
 warriorsAmount++;  
 ants.*addAnt*(ant);  
 antsAmount++;  
 gui.draw.repaint();  
 }  
 }  
 else  
 {  
 gui.draw.repaint();  
 }  
 }  
  
 boolean stop() {  
 if (isInformationShowing) {  
 isPaused = true;  
 switch(JOptionPane.*showConfirmDialog*(null, "Время симуляции: " + time + "\nПоявилось муравьев: "  
 + antsAmount + "\nМуравьев-рабочих: " + workersAmount + "\nМуравьев-воинов: " +  
 warriorsAmount, "Результаты", JOptionPane.*OK\_CANCEL\_OPTION*)) {  
 case JOptionPane.*OK\_OPTION*: {  
 factory.exterminate();  
 isPaused = false;  
 isOver = true;  
 time = 0;  
 gui.timer.setText("Таймер: " + time);  
 ants.*clearAnts*();  
 antsAmount = 0;  
 workersAmount = 0; warriorsAmount = 0;  
 gui.draw.repaint();  
 return isOver;  
 }  
 case JOptionPane.*CANCEL\_OPTION*:  
 {  
 isOver = false;  
 isPaused = false;  
 return isOver;  
 }  
 default:  
 {  
 isOver = false;  
 isPaused = false;  
 return false;  
 }  
 }  
 }  
 else {  
 factory.exterminate();  
 isOver = true;  
 System.*out*.println(isOver);  
 time = 0;  
 gui.timer.setText("Таймер: " + time);  
 ants.*clearAnts*();  
 antsAmount = 0;  
 workersAmount = 0; warriorsAmount = 0;  
 gui.draw.repaint();  
 }  
 return true;  
 }  
  
 public JButton getButtonStart()  
 {  
 return gui.getButtonStart();  
 }  
  
 public JButton getButtonStop()  
 {  
 return gui.getButtonStop();  
 }  
  
 public JCheckBox getInformationVisibility()  
 {  
 return gui.getInfoVisibility();  
 }  
  
 public JRadioButton getTimerIsVisible()  
 {  
 return gui.getTimerIsVisible();  
 }  
  
 public JRadioButton getTimerNotVisible()  
 {  
 return gui.getTimerNotVisible();  
 }  
  
 public JTextField getWorkerPeriod()  
 {  
 return gui.getWorkerPeriod();  
 }  
  
 public JTextField getWarriorPeriod()  
 {  
 return gui.getWarriorPeriod();  
 }  
  
 public JComboBox getWorkerProbability()  
 {  
 return gui.getWorkerProbability();  
 }  
  
 public JComboBox getWarriorProbability()  
 {  
 return gui.getWarriorProbability();  
 }  
  
 public JMenuItem getMenuStart()  
 {  
 return gui.getMenuStart();  
 }  
  
 public JMenuItem getMenuStop()  
 {  
 return gui.getMenuStop();  
 }  
  
 public JMenuItem getMenuTimerVisible()  
 {  
 return gui.getMenuTimerVisible();  
 }  
  
 public JMenuItem getMenuInformationVisible()  
 {  
 return gui.getMenuInformationVisible();  
 }  
}

**Main**

package AntFarm;  
  
import javax.swing.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.awt.event.KeyAdapter;  
import java.awt.event.KeyEvent;  
import java.util.Timer;  
  
public class Main extends KeyAdapter implements ActionListener {  
 private Habitat antFarm;  
 protected MyTimerTask myTimerTask;  
 protected Timer timer;  
 private boolean isStarted, isTimerVisible;  
 int workerPeriod, warriorPeriod;  
 double workerProbability, warriorProbability;  
  
 public static void main(String[] args)  
 {  
 new Main();  
 }  
  
 Main() {  
 antFarm = new Habitat(this);  
 antFarm.setVisible(true);  
 antFarm.addKeyListener(this);  
 isStarted = false;  
 isTimerVisible = true;  
 }  
  
 @Override  
 public void keyPressed(KeyEvent keyEvent) {  
 int key = keyEvent.getKeyCode();  
 if (key == KeyEvent.*VK\_B* && !isStarted) {  
 startSimulation();  
 }  
 if (key == KeyEvent.*VK\_E* && isStarted) {  
 stopSimulation();  
 }  
 if (key == KeyEvent.*VK\_T*) {  
 antFarm.timerVisibility(isTimerVisible);  
 if (isTimerVisible) {  
 antFarm.getTimerIsVisible().setEnabled(true);  
 antFarm.getTimerNotVisible().setEnabled(false);  
 antFarm.getTimerNotVisible().setSelected(true);  
 } else {  
 antFarm.getTimerIsVisible().setEnabled(false);  
 antFarm.getTimerIsVisible().setSelected(true);  
 antFarm.getTimerNotVisible().setEnabled(true);  
 }  
 isTimerVisible = !isTimerVisible;  
 }  
 }  
  
 @Override  
 public void actionPerformed(ActionEvent actionEvent) {  
 Object action = actionEvent.getSource();  
 if (action == antFarm.getButtonStart()) {  
 if (!isStarted)  
 startSimulation();  
 }  
 if (action == antFarm.getButtonStop())  
 {  
 if (isStarted)  
 stopSimulation();  
 }  
 if (action == antFarm.getInformationVisibility()) {  
 antFarm.informationShowing();  
 }  
 if (action == antFarm.getTimerIsVisible() || action == antFarm.getTimerNotVisible()) {  
 antFarm.timerVisibility(isTimerVisible);  
 if (isTimerVisible) {  
 antFarm.getTimerIsVisible().setEnabled(true);  
 antFarm.getTimerNotVisible().setEnabled(false);  
 antFarm.getTimerNotVisible().setSelected(true);  
 } else {  
 antFarm.getTimerIsVisible().setEnabled(false);  
 antFarm.getTimerIsVisible().setSelected(true);  
 antFarm.getTimerNotVisible().setEnabled(true);  
 }  
 isTimerVisible = !isTimerVisible;  
 }  
 if (action == antFarm.getMenuStart()) {  
 if (!isStarted)  
 startSimulation();  
 }  
 if (action == antFarm.getMenuStop()) {  
 if (isStarted)  
 stopSimulation();  
 }  
 if (action == antFarm.getMenuTimerVisible()) {  
 antFarm.timerVisibility(isTimerVisible);  
 if (isTimerVisible) {  
 antFarm.getTimerIsVisible().setEnabled(true);  
 antFarm.getTimerNotVisible().setEnabled(false);  
 antFarm.getTimerNotVisible().setSelected(true);  
 } else {  
 antFarm.getTimerIsVisible().setEnabled(false);  
 antFarm.getTimerIsVisible().setSelected(true);  
 antFarm.getTimerNotVisible().setEnabled(true);  
 }  
 isTimerVisible = !isTimerVisible;  
 }  
 if (action == antFarm.getMenuInformationVisible()) {  
 antFarm.informationShowing();  
 }  
 antFarm.requestFocus();  
 }  
  
 public void startSimulation()  
 {  
 if (checkParameters())  
 {  
 workerPeriod = Integer.*parseInt*(antFarm.getWorkerPeriod().getText());  
 warriorPeriod = Integer.*parseInt*(antFarm.getWarriorPeriod().getText());  
 workerProbability = (double)(antFarm.getWorkerProbability().getSelectedItem());  
 warriorProbability = (double)(antFarm.getWarriorProbability().getSelectedItem());  
 antFarm.factory.setParameters(workerPeriod, warriorPeriod, workerProbability, warriorProbability);  
 }  
 else  
 {  
 JOptionPane.*showMessageDialog*(null, "Введено некорректное значение. Установлены значения по умолчанию.", "Ошибка!", JOptionPane.*ERROR\_MESSAGE*);  
 workerPeriod = 5;  
 warriorPeriod = 3;  
 workerProbability = 0.9;  
 warriorProbability = 0.7;  
 antFarm.factory.setParameters(workerPeriod, warriorPeriod, workerProbability, warriorProbability);  
 antFarm.getWorkerPeriod().setText(Integer.*toString*(workerPeriod));  
 antFarm.getWarriorPeriod().setText(Integer.*toString*(warriorPeriod));  
 antFarm.getWorkerProbability().setSelectedItem(workerProbability);  
 antFarm.getWarriorProbability().setSelectedItem(warriorProbability);  
 }  
 //System.*out*.println(workerPeriod + " " + warriorPeriod + " " + workerProbability + " " + warriorProbability);  
 isStarted = true;  
 timer = new Timer();  
 myTimerTask = new MyTimerTask(antFarm);  
 timer.schedule(myTimerTask, 0, 1000);  
 antFarm.getButtonStart().setEnabled(false);  
 antFarm.getButtonStop().setEnabled(true);  
 antFarm.getWorkerPeriod().setEnabled(false);  
 antFarm.getWarriorPeriod().setEnabled(false);  
 antFarm.getWorkerProbability().setEnabled(false);  
 antFarm.getWarriorProbability().setEnabled(false);  
 }  
  
 public void stopSimulation()  
 {  
 boolean isOver = antFarm.stop();  
 if (isOver)  
 {  
 antFarm.getButtonStart().setEnabled(true);  
 antFarm.getButtonStop().setEnabled(false);  
 antFarm.getWorkerPeriod().setEnabled(true);  
 antFarm.getWarriorPeriod().setEnabled(true);  
 antFarm.getWorkerProbability().setEnabled(true);  
 antFarm.getWarriorProbability().setEnabled(true);  
 timer.cancel();  
 timer.purge();  
 myTimerTask.cancel();  
 isStarted = false;  
 }  
 }  
  
 public boolean checkParameters() {  
 try  
 {  
 Integer.*parseInt*(antFarm.getWorkerPeriod().getText());  
 }  
 catch (NumberFormatException e)  
 {  
 return false;  
 }  
 try {  
 Integer.*parseInt*(antFarm.getWarriorPeriod().getText());  
 }  
 catch (NumberFormatException e)  
 {  
 return false;  
 }  
 return true;  
 }  
}