МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

**Лабораторная работа №3**

по дисциплине «Технология программирования»

на тему: «Классы-коллекции»

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

Новосибирск, 2020

**Цель работы**

Изучить особенности реализации классов-коллекций в Java. Доработать программу, созданную в лабораторной работе № 2:

**Задание**

1. Добавить генерируемым объектам понятия «время рождения» и «время жизни». Время рождения устанавливается в момент генерации объекта, и по значению соответствует времени, прошедшему от начала симуляции. Время жизни – время, через которое объект должен исчезнуть, считая от времени рождения;
2. Вынести установку параметров времени жизни объектов в пользовательский интерфейс. Для каждого типа объекта должно задаваться собственное время. Рекомендуется использовать текстовые поля, но следуют помнить о проверке на ввод некорректных данных;
3. Добавить генерируемым объектам уникальные целочисленные идентификаторы (случайные числа), которые назначаются при генерации объекта. Для хранения сгенерированных идентификаторов используйте коллекцию удобную для поиска по варианту;
4. Использовать коллекции по варианту. При генерации объекта происходит добавление его в коллекцию (в класс добавить поле идентификатора), а во вторую коллекцию: идентификаторы существующих объектов, в третью идентификатор + время рождения. При возникновении события по таймеру обойдите коллекцию и удалите все объекты, время жизни которых истекло, а также все данные во вспомогательных коллекциях; Для обеспечения доступа к коллекциям используйте структурный паттерн Singleton.
5. Добавьте в панель управления кпопку «Текущие объекты». По нажатию на эту кнопку появляется модальное диалоговое окно, содержащее список всех «живых» объектов на момент нажатия со временем их рождения (время рождения – ключ). В класс диалогового окна должна передаваться коллекция с хранением объектов по времени рождения. Типы коллекций задаются вариантом.

**Вариант 1**

Коллекция для хранения объектов: *Vector*

Коллекция для хранения и поиска уникальных идентификаторов: *HashSet*

Коллекция для хранения времени рождения объектов: *TreeMap*

**Ход работы**

В ходе решения поставленных задач доработки программы были внесены следующие изменения:

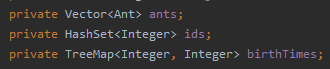
1. Добавлен класс **CurrentObject** – модальное диалоговое окно, которое отображает список всех “живых” объектов на момент нажатия соответствующей кнопки на панели управления.



2. В класс Ant добавлены новые поля: **id** – уникальный идентификатор, **birthTime** – время рождения муравья, **deathTime** – время, при наступлении которого данный объект будет удален.

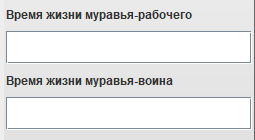


3. В класс Singleton добавлены новые поля: объекты хранятся в коллекции класса **Vector**, идентификаторы – в коллекции **HashSet**, а также реализовано дерево **TreeMap**, в котором хранятся пары идентификатор + время рождения объекта.



4. На панель управления были добавлены поля для установки времени жизни муравья-рабочего и муравья-воина и кнопка “**Текущие объекты**”, вызывающая диалоговое окно со списком объектов.



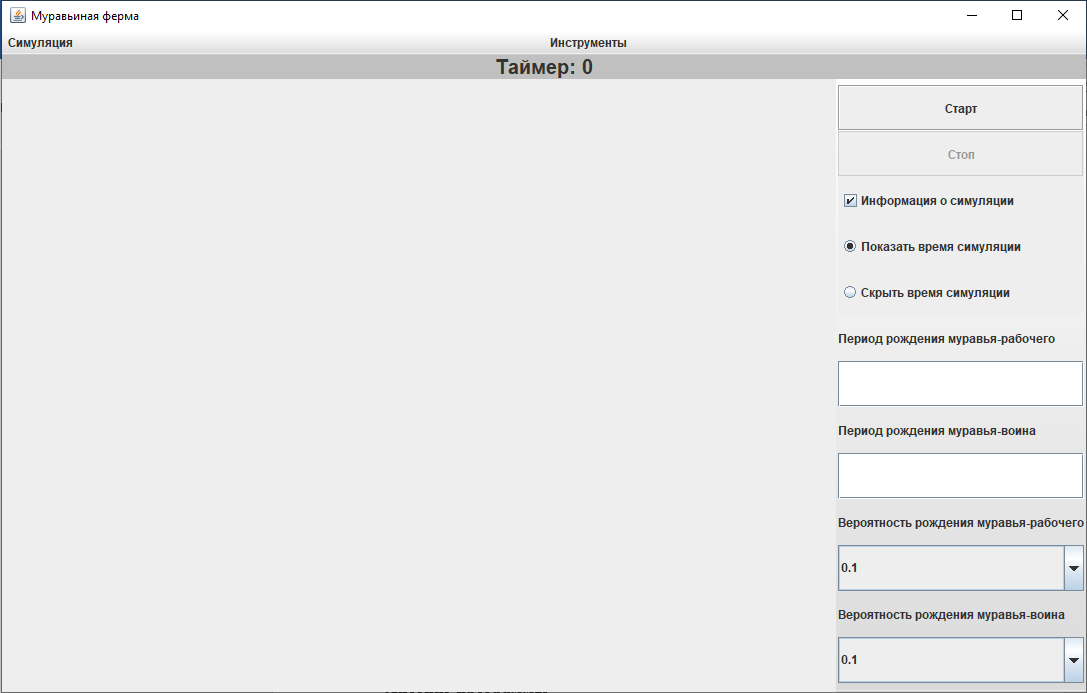


5. Реализовано удаление объектов с рабочей области при истечении времени жизни.

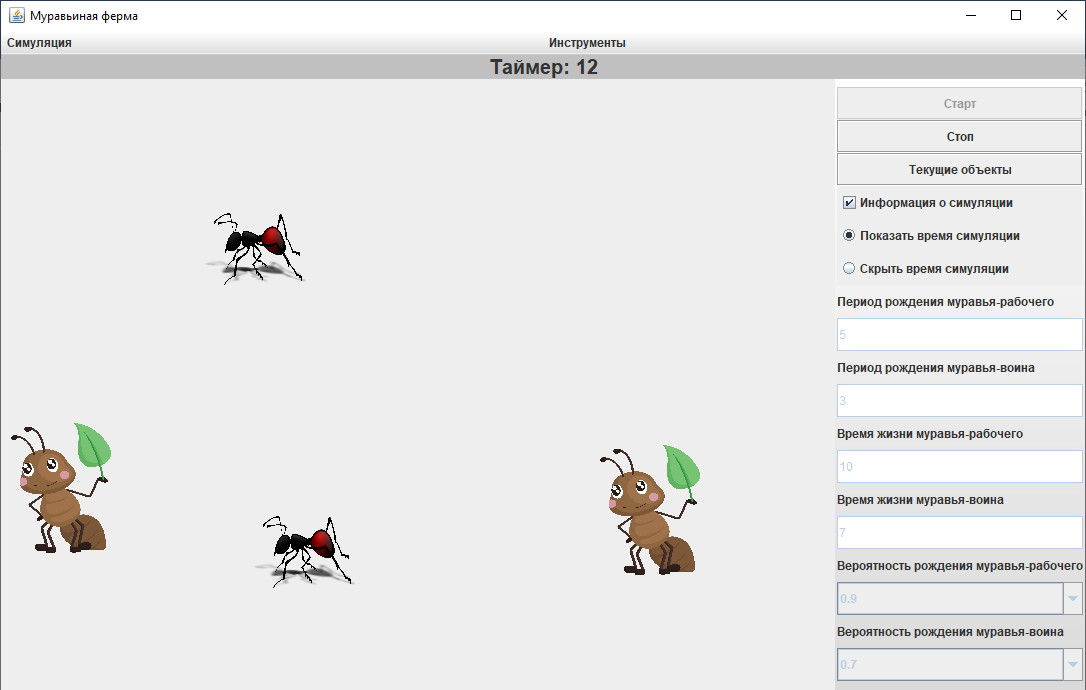
Остальные классы программы были переработаны в соответствии с новыми задачами.

**Скриншоты работы программы**

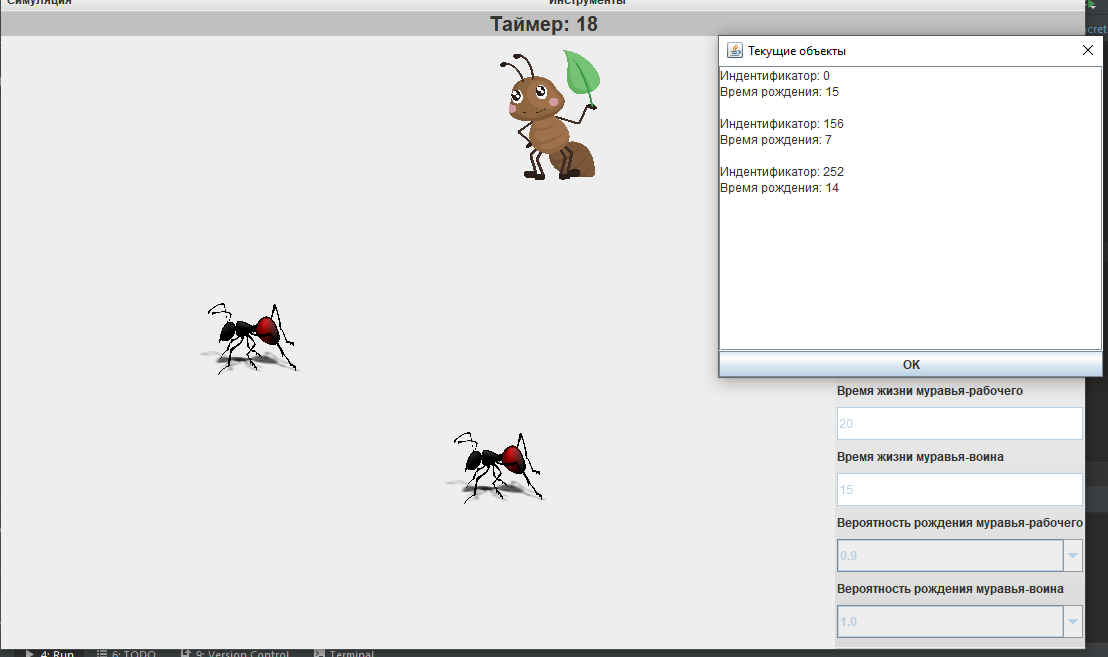
Окно до запуска симуляции:



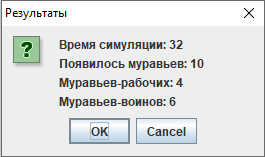
Окно во время симуляции:



Диалоговое окно со списком “живых” объектов:



Диалоговое окно после завершения симуляции:



**Вывод**

При выполнении данной работы были изученные теоретические основы и получены практические навыки по работе с классами-коллекциями. Был расширен интерфейс программы, созданной в предыдущих лабораторных работах.

**Листинг программы**

**CurrentObjects**

package AntFarm;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.MouseAdapter;  
import java.awt.event.MouseEvent;  
import java.util.Map;  
import java.util.TreeMap;  
  
public class CurrentObjects extends JDialog  
{  
 private JButton buttonOK = new JButton("OK");  
 private JTextArea textArea = new JTextArea();  
 private JScrollPane scrollPane = new JScrollPane(textArea);  
  
 public CurrentObjects (TreeMap <Integer, Integer> birthTimes)  
 {  
 super(new JFrame(), "Текущие объекты", true);  
 setSize(400,350);  
 add(buttonOK, BorderLayout.*SOUTH*);  
 add(scrollPane);  
 textArea.setEditable(false);  
  
 setDefaultCloseOperation(*DISPOSE\_ON\_CLOSE*);  
 String text = "";  
  
 for (Map.Entry<Integer,Integer> e : birthTimes.entrySet())  
 {  
 text+= "Индентификатор: " + e.getKey() + "\nВремя рождения: " + e.getValue() + "\n\n";  
 }  
 textArea.setText(text);  
  
 buttonOK.addMouseListener(new MouseAdapter() {  
 public void mousePressed(MouseEvent e) {  
 setVisible(false);  
 }  
 });  
 }  
}

**Singleton**

package AntFarm;  
  
import java.util.\*;  
  
public class Singleton  
{  
 private static Singleton *singleton*;  
 private Vector<Ant> ants;  
 private HashSet<Integer> ids;  
 private TreeMap<Integer, Integer> birthTimes;  
  
 private Singleton()  
 {  
 ants = new Vector<>();  
 ids = new HashSet<>();  
 birthTimes = new TreeMap<>();  
 }  
  
 public int getArraySize()  
 {  
 return ants.size();  
 }  
  
 public static Singleton getSingleton()  
 {  
 if(*singleton* == null)  
 {  
 *singleton* = new Singleton();  
 }  
 return *singleton*;  
 }  
  
 public static void addAnt(Ant ant, int id, int birthTime)  
 {  
 *singleton*.ants.add(ant);  
 *singleton*.ids.add(id);  
 *singleton*.birthTimes.put(id, birthTime);  
 }  
  
 public static void clearAnts()  
 {  
 if (!*singleton*.ants.isEmpty())  
 {  
 *singleton*.ants.clear();  
 *singleton*.ids.clear();  
 *singleton*.birthTimes.clear();  
 }  
 }  
  
 public static void removeAnt(Ant ant, int id)  
 {  
 *singleton*.ants.remove(ant);  
 *singleton*.ids.remove(id);  
 *singleton*.birthTimes.remove(id);  
 }  
  
 public static Vector<Ant> getAnts(){  
 return *singleton*.ants;  
 }  
  
 public static TreeMap<Integer, Integer> getBirthTime()  
 {  
 return *singleton*.birthTimes;  
 }  
  
 public static HashSet<Integer> getIds()  
 {  
 return *singleton*.ids;  
 }  
}

**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;  
  
 private JLabel labelWorkerLifetime;  
 private JLabel labelWarriorLifetime;  
 private JTextField workerLifetime;  
 private JTextField warriorLifetime;  
  
 private JButton buttonObjects;  
  
 //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(18,1));  
 settings.setFocusable(false);  
 settings.setFloatable(false);  
  
 buttonStart = new JButton("Старт");  
 buttonStart.addActionListener(actionListener);  
  
 buttonStop = new JButton("Стоп");  
 buttonStop.setEnabled(false);  
 buttonStop.addActionListener(actionListener);  
  
 buttonObjects = new JButton("Текущие объекты");  
 buttonObjects.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();  
  
 labelWorkerLifetime = new JLabel("Время жизни муравья-рабочего");  
 workerLifetime = new JTextField();  
  
 labelWarriorLifetime = new JLabel("Время жизни муравья-воина");  
 warriorLifetime = 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(buttonObjects);  
 settings.add(infoVisibility);  
 settings.add(timerIsVisible);  
 settings.add(timerNotVisible);  
 settings.add(labelWorkerPeriod);  
 settings.add(workerPeriod);  
 settings.add(labelWarriorPeriod);  
 settings.add(warriorPeriod);  
 settings.add(labelWorkerLifetime);  
 settings.add(workerLifetime);  
 settings.add(labelWarriorLifetime);  
 settings.add(warriorLifetime);  
 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 JButton getButtonObjects() { return buttonObjects;}  
  
 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 JTextField getWorkerLifetime() {return workerLifetime;}  
  
 public JTextField getWarriorLifetime() {return warriorLifetime;}  
  
 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 int id, birthTime, deathTime;  
  
 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, int id, int birthTime, int lifetime) {  
 super(x, y);  
 this.id = id;  
 this.birthTime = birthTime;  
 this.deathTime = birthTime + lifetime;  
  
 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, int id, int birthTime, int lifetime) {  
 super(x, y);  
 this.id = id;  
 this.birthTime = birthTime;  
 this.deathTime = birthTime + lifetime;  
  
 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 id, int time);  
 void setParameters(int N1, int N2, double P1, double P2, int L1, int L2);  
 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 L1;  
 private int L2;  
 private int antsAmount = 0;  
 private int workersAmount = 0;  
 private int warriorsAmount = 0;  
  
 public ConcreteFactory() { }  
  
 @Override  
 public Ant createAnt(int x, int y, int id, int time) {  
 if (time % N1 == 0)  
 {  
 Random random = new Random();  
 double probability = random.nextDouble();  
 if (P1 > probability)  
 {  
 Ant ant = new AntWorker(x, y, id, time, L1);  
 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, id, time, L2);  
 warriorsAmount++; antsAmount++;  
 return ant;  
 }  
 }  
 return null;  
 }  
  
 @Override  
 public void setParameters(int N1, int N2, double P1, double P2, int L1, int L2)  
 {  
 this.N1 = N1;  
 this.N2 = N2;  
 this.P1 = P1;  
 this.P2 = P2;  
 this.L1 = L1;  
 this.L2 = L2;  
 }  
  
 @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++) {  
 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, id, lifetime;  
 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();  
 id = random.nextInt(500);  
 int xcord = random.nextInt(gui.draw.getWidth() - 200);  
 int ycord = random.nextInt(gui.draw.getHeight() - 200);  
 Ant ant = factory.createAnt(xcord, ycord, id, time);  
 if (ant != null) {  
 if (ant.getClass() == AntWorker.class)  
 workersAmount++;  
 else  
 warriorsAmount++;  
 ants.*addAnt*(ant, ant.id, time);  
 antsAmount++;  
 }  
 for (int i=0; i<ants.getArraySize(); i++)  
 {  
 if (ants.*getAnts*().get(i).deathTime == time)  
 ants.*removeAnt*(ants.*getAnts*().get(i), ants.*getAnts*().get(i).id);  
 }  
 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;  
 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 JButton getButtonObjects()  
 {  
 return gui.getButtonObjects();  
 }  
  
 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 JTextField getWorkerLifetime()  
 {  
 return gui.getWorkerLifetime();  
 }  
  
 public JTextField getWarriorLifetime()  
 {  
 return gui.getWarriorLifetime();  
 }  
  
 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;  
 private CurrentObjects obj;  
 protected MyTimerTask myTimerTask;  
 protected Timer timer;  
 private boolean isStarted, isTimerVisible;  
 int workerPeriod, warriorPeriod, workerLifetime, warriorLifetime;  
 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.getButtonObjects())  
 {  
 obj = new CurrentObjects(Singleton.*getBirthTime*());  
 obj.setVisible(true);  
 }  
 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());  
 workerLifetime = Integer.*parseInt*(antFarm.getWorkerLifetime().getText());  
 warriorLifetime = Integer.*parseInt*(antFarm.getWarriorLifetime().getText());  
 workerProbability = (double)(antFarm.getWorkerProbability().getSelectedItem());  
 warriorProbability = (double)(antFarm.getWarriorProbability().getSelectedItem());  
 antFarm.factory.setParameters(workerPeriod, warriorPeriod, workerProbability, warriorProbability, workerLifetime, warriorLifetime);  
 }  
 else  
 {  
 JOptionPane.*showMessageDialog*(null, "Введено некорректное значение. Установлены значения по умолчанию.", "Ошибка!", JOptionPane.*ERROR\_MESSAGE*);  
 workerPeriod = 5;  
 warriorPeriod = 3;  
 workerLifetime = 10;  
 warriorLifetime = 7;  
 workerProbability = 0.9;  
 warriorProbability = 0.7;  
 antFarm.factory.setParameters(workerPeriod, warriorPeriod, workerProbability, warriorProbability, workerLifetime, warriorLifetime);  
 antFarm.getWorkerPeriod().setText(Integer.*toString*(workerPeriod));  
 antFarm.getWarriorPeriod().setText(Integer.*toString*(warriorPeriod));  
 antFarm.getWorkerLifetime().setText(Integer.*toString*(workerLifetime));  
 antFarm.getWarriorLifetime().setText(Integer.*toString*(warriorLifetime));  
 antFarm.getWorkerProbability().setSelectedItem(workerProbability);  
 antFarm.getWarriorProbability().setSelectedItem(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.getWorkerLifetime().setEnabled(false);  
 antFarm.getWarriorLifetime().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.getWorkerLifetime().setEnabled(true);  
 antFarm.getWarriorLifetime().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());  
 Integer.*parseInt*(antFarm.getWarriorPeriod().getText());  
 Integer.*parseInt*(antFarm.getWorkerLifetime().getText());  
 Integer.*parseInt*(antFarm.getWarriorLifetime().getText());  
 }  
 catch (NumberFormatException e)  
 {  
 return false;  
 }  
 return true;  
 }  
}