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 №4

по дисциплине «Технология программирования»

Тема: «Многопотоковые приложения»

**Вариант №2**

Выполнил:

Студент группы АВТ-716

*Тесленко Н.Б.*

Проверил:

Ассистент кафедры

*Михайленко Дмитрий Анатольевич*

Новосибирск, 2020 г.

# **Цель работы**

* изучить особенности реализации и работы потоков в Java, управлением приоритетами потоков и синхронизацией потоков в Java;
* доработать программу, созданную в лабораторной работе №3

# **Практические задания**

1. создать абстрактный класс BaseAI, описывающий «интеллектуальное поведение» объектов. Класс должен создавать поток, обеспечивающий движения объектов коллекции;
2. реализовать класс BaseAI для каждого из видов объекта, включив в него поведение, описанное в индивидуальном задании по варианту;
3. синхронизовать работу потоков расчета интеллекта объектов, их рисования и генерации новых объектов. Рисование должно остаться в основном потоке;
4. добавить в панель управления кнопки для остановки и возобновления работы интеллекта каждого вида объектов. Реализовать через управление монитором (методы wait() и notify());
5. добавить в панель управления выпадающие списки для выставления приоритетов каждого из потоков.

## **Исходные данные, соответствующие варианту**

1. Пчелы-рабочие двигаются в один из углов области их обитания (например, [0;0]) по прямой со скоростью V, а затем возвращаться обратно в точку своего рождения с той же скоростью.
2. Трутни двигаются хаотично со скоростью V. Хаотичность достигается случайной сменой направления движения раз в N секунд.

# **Проектирование программы**

На данном этапе работы в программу внесены следующие изменения:

Создан новый абстрактный класс **BaseAI()** для создания потока путем наследования интерфейса Runnable(), в котором прописан абстрактный метод run().

import java.util.ArrayList;  
  
public abstract class BaseAI implements Runnable {  
  
 Thread thread;  
 ArrayList<Character> arrBees = new ArrayList<Character>();  
 boolean stop = true;  
  
  
 BaseAI(String name){  
 thread = new Thread(this, name);  
 }  
  
 public void setArray(ArrayList<Character> arr){  
 arrBees = new ArrayList<Character>(arr);  
 }  
 public void pause(){  
 try {  
 wait();  
 } catch (InterruptedException e) {}  
 }  
 synchronized public void \_continue(){  
 notify();  
 }  
 void setBoolStop(boolean stop){this.stop = stop;}  
}

Два класса-наследника от BaseAI() - WorkerAI() и DroneAI(). В наследниках переопределен метод run(), в котором мы проходим по коллекции, выбирая объект нужного нам вида и реализуем движение объектов. Проход по коллекции помещаем в синхронайзд блок для ее синхронизации.

Класс DroneAI()

В классе DroneAI() прописано поведение трутней. Трутни двигаются хаотично со скоростью V. Хаотичность достигается случайной сменой направления движения раз в N секунд

import java.util.Random;  
  
public class DroneAI extends BaseAI {  
 DroneAI(){  
 super("DroneAI");  
 thread.start();  
 }  
 @Override  
 synchronized public void run(){  
 int N = 2000;  
 int route = 0;  
 long timePrev1 = 0;  
 long timePrev2 = 0;  
 while (true) {  
 if(stop){  
 pause();  
 }  
 long timeCur = System.*currentTimeMillis*();  
 long step1 = timeCur - timePrev1;  
 long step2 = timeCur - timePrev2;  
 if (step1 >= 20) {  
 if(step2 >= N)  
 {  
 route =new Random().nextInt(7);  
 timePrev2 = timeCur;  
 }  
 for (Character bee : arrBees) {  
 if (bee instanceof Drone) {  
 switch (route){  
 case 0:  
 {  
 bee.sety(bee.gety() - bee.getV());  
 break;  
 }  
 case 1:  
 {  
 bee.sety(bee.gety() - bee.getV());  
 bee.setx(bee.getx() + bee.getV());  
 break;  
 }  
 case 2:  
 {  
 bee.setx(bee.getx() + bee.getV());  
 break;  
 }  
 case 3:  
 {  
 bee.sety(bee.gety() + bee.getV());  
 bee.setx(bee.getx() + bee.getV());  
 break;  
 }  
 case 4:  
 {  
 bee.sety(bee.gety() + bee.getV());  
 break;  
 }  
 case 5:  
 {  
 bee.sety(bee.gety() + bee.getV());  
 bee.setx(bee.getx() - bee.getV());  
 break;  
 }  
 case 6:  
 {  
 bee.setx(bee.getx() - bee.getV());  
 break;  
 }  
 case 7:  
 {  
 bee.sety(bee.gety() - bee.getV());  
 bee.setx(bee.getx() - bee.getV());  
 break;  
 }  
 }  
 bee.getJlb().setLocation(bee.getx(), bee.gety());  
 }  
 }  
 timePrev1 = timeCur;  
 }  
 try{  
 thread.*sleep*(1);  
 }catch (InterruptedException e){}  
 }  
 }  
}

Класс WorkerAI()

В классе WorkerAI() прописано поведение трутней - двигаются в один из углов области их обитания, а затем возвращаться обратно в точку своего рождения.

public class WorkerAI extends BaseAI {  
 WorkerAI(){  
 super("WorkerAI");  
 thread.start();  
 }  
 @Override  
 synchronized public void run(){  
 long timePrev = 0;  
 while (true) {  
 if(stop){  
 pause();  
 }  
 long timeCur = System.*currentTimeMillis*();  
 long step = timeCur - timePrev;  
 if (step >= 1000/60) {  
 for (Character bee : arrBees) {  
 if (bee instanceof Worker) {  
 if (((Worker) bee).toAngle) {  
 if (bee.getx() > 0) {  
 bee.setx(bee.getx() - bee.getV());  
 }else if (bee.gety() > 0){  
 bee.sety(bee.gety() - bee.getV());  
 }else{  
 ((Worker) bee).toAngle = false;  
 }  
 }else{  
 if (bee.getx() < ((Worker) bee).x0) {  
 bee.setx(bee.getx() + bee.getV());  
 }else if (bee.gety() < ((Worker) bee).y0){  
 bee.sety(bee.gety() + bee.getV());  
 }else{  
 ((Worker) bee).toAngle = true;  
 }  
 }  
 bee.getJlb().setLocation(bee.getx(), bee.gety());  
 }  
 }  
 timePrev = timeCur;  
 }  
 try{  
 thread.*sleep*(1);  
 }catch (InterruptedException e){}  
 }  
 }  
}

Метод pause() для управления «монитором». Метод wait() отдает монитор и поток ждет вызова функции notify(), которая изменит состояние ожидающего потока.

public void pause(){  
 try {  
 wait();  
 } catch (InterruptedException e) {}  
}

synchronized public void \_continue(){  
 notify();  
}

В классе MainProgram() добавлены 4 кнопки для остановки и возобновления работы интеллекта каждого вида объектов.

private JButton btStopWorkerThread = new JButton("Stop Workers"), btStopDroneThread = new JButton("Stop Drones"),  
 btContinueWorkerThread = new JButton("Continue Workers"), btContinueDroneThread = new JButton("Continue Drones");

Обработка событий нажатия кнопок остановки и продолжения симуляции :

btStopWorkerThread.addActionListener(e -> {  
 habitat.workerAI.setBoolStop(true);  
 btStopWorkerThread.setEnabled(false);  
 btContinueWorkerThread.setEnabled(true);  
});  
btContinueWorkerThread.addActionListener(e -> {  
 habitat.workerAI.\_continue();  
 habitat.workerAI.setBoolStop(false);  
 btStopWorkerThread.setEnabled(true);  
 btContinueWorkerThread.setEnabled(false);  
});  
btStopDroneThread.addActionListener(e -> {  
 habitat.droneAI.setBoolStop(true);  
 btStopDroneThread.setEnabled(false);  
 btContinueDroneThread.setEnabled(true);  
});  
btContinueDroneThread.addActionListener(e -> {  
 habitat.droneAI.\_continue();  
 habitat.droneAI.setBoolStop(false);  
 btStopDroneThread.setEnabled(true);  
 btContinueDroneThread.setEnabled(false);  
});

В этом же классе создаем выпадающий список для выставления приоритетов потоков и текстовое поле для того, чтобы задавать этот приоритет.

private JTextField priority = new JTextField("Введите приоритет");

private String arrPriority[] = {"Поток рабочих", "Поток трутней"};  
private JComboBox<String> comboBoxPriority = new JComboBox<String>(arrPriority);

Обработка событий с помощью лямбда-выражения(для второго вида аналогично) :

btEnterPriority.addActionListener(e -> {  
 EnterPriority();  
 this.requestFocus();  
});

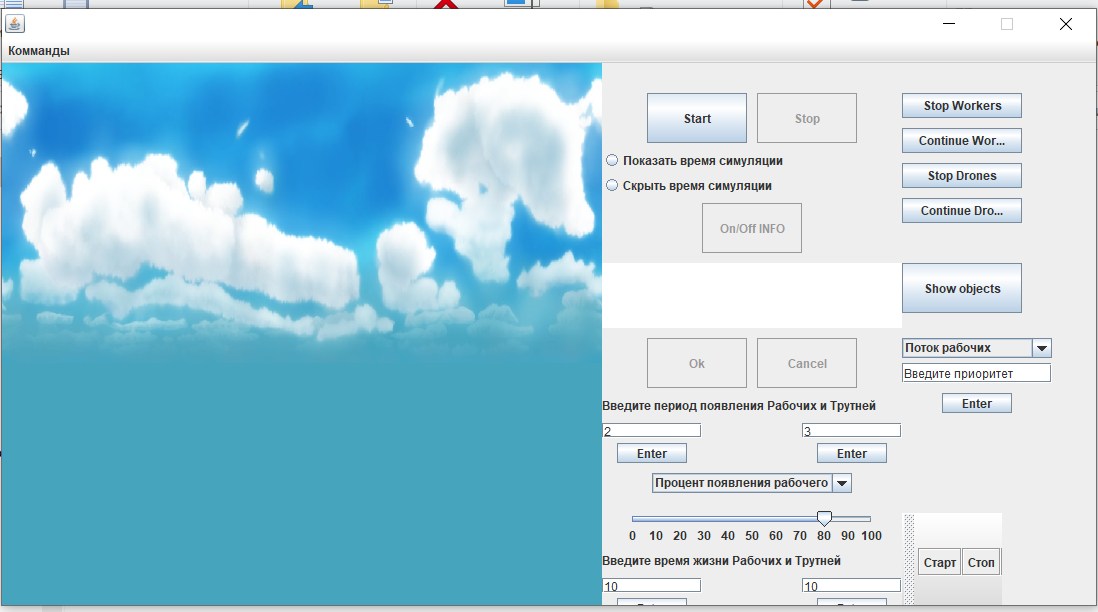
private void EnterPriority(){  
 try {  
 int priority\_ = Integer.*parseInt*(priority.getText());  
 if (priority\_ >= 1 && priority\_ <= 10) {  
 String tmp2 = (String)comboBoxPriority.getSelectedItem();  
 if(tmp2.equals(arrPriority[0])){  
 habitat.workerAI.thread.setPriority(priority\_);  
 JOptionPane.*showMessageDialog*(this, "Установлен приоритет потока\n" +  
 "рабочих, равный " + priority\_);  
 }else if(tmp2.equals(arrPriority[1])){  
 habitat.droneAI.thread.setPriority(priority\_);  
 JOptionPane.*showMessageDialog*(this, "Установлен приоритет потока\n" +  
 "трутней, равный " + priority\_);  
 }  
 } else {  
 throw new Exception();  
 }  
 }catch (Exception e1){  
 JOptionPane.*showMessageDialog*(this, "Приоритет потоков должен быть в интервале 1-10." +  
 "\nУстановлено значение по умолчанию.");  
 }  
}

В конструкторе Habitat создаем потоки и запускаем их.

workerAI = new WorkerAI();  
droneAI = new DroneAI();

Для синхронизации коллекции помещаем генерацию новых объектов и их рисование в синхронайзд блоки.

# **Пользовательский интерфейс**



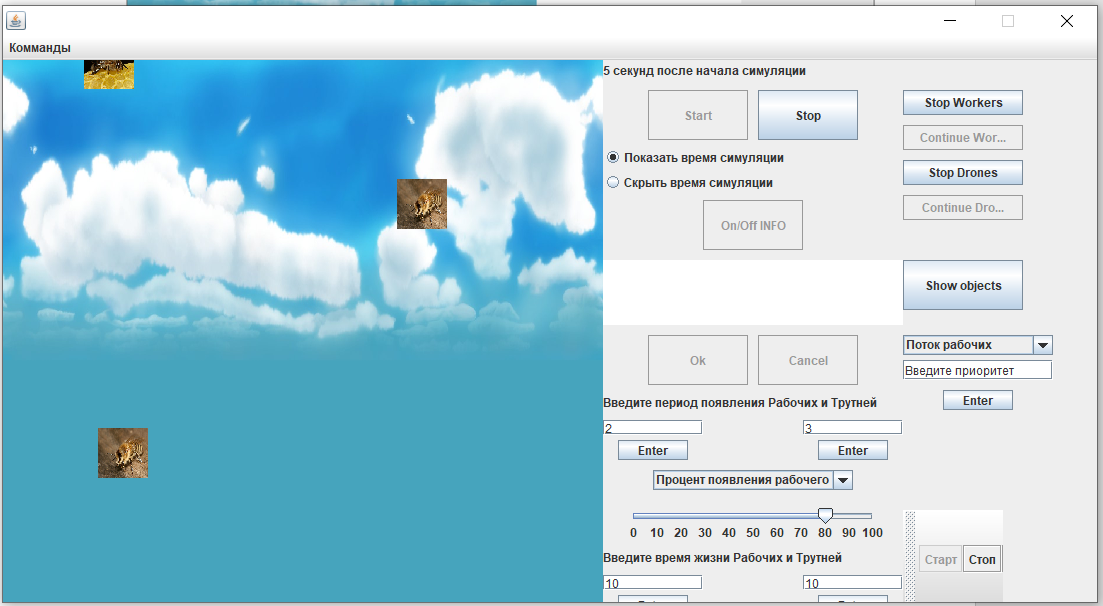


Рис. 1-2. Движение объектов

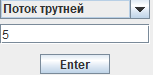
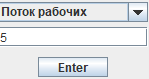
 

Рис. 3. Установка приоритета потоков

# **Анализ проделанной работы**

В ходе работы были изучены особенности реализации и работы потоков в Java, управлением приоритетами потоков и синхронизацией потоков. Была доработана программа из предыдущей лабораторной работы: создали поток для каждого из видов объектов, который отвечает за их «поведение», научились синхронизировать потоки, управлять «монитором» с помощью методов wait() и notify().

# **Литература**

* 1. Васюткина И. А. Технология разработки объектно-ориентированных программ на Java. – НГТУ, 2012.
  2. Герберт Ш. Swing: руководство для начинающих. Пер. с англ – М.: ООО «И. Д. Вильямс», 2007.