МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

*Лабораторная работа №4 по дисциплине «Технология программирования»*

*Вариант 2*

Группа: АВТ-610

Студенты: Князев Р.И., Дунаев Н.Ю.

Преподаватель: Васюткина И. А.

Новосибирск 2018

**Темы:**

***Многопотоковые приложения.***

**Задание:**

1. Изучить особенности реализации и работы потоков в Java, управлением приоритетами потоков и синхронизацией потоков.
2. Доработать программу, созданную в лабораторной работе № 3:
3. создать абстрактный класс BaseAI, описывающий «интеллектуальное поведение» объектов. Класс должен создавать поток, обеспечивающий движения объектов коллекции;
4. реализовать класс BaseAI для каждого из видов объекта, включив в него поведение, описанное в индивидуальном задании по варианту;
5. синхронизовать работу потоков расчета интеллекта объектов, их рисования и генерации новых объектов. Рисование должно остаться в основном потоке;
6. добавить в панель управления кнопки для остановки и возобновления работы интеллекта каждого вида объектов. Реализовать через управление монитором (методы wait() и notify());
7. добавить в панель управления выпадающие списки для выставления приоритетов каждого из потоков.

***Вариант 2***

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.

***Описание структуры классов:***

Для реализации программы были добавлены и изменены следующие классы:

**BasicAI -** класс, реализующий поток, который содержит все манипуляции, связанные с движением пчел.

**Habbitat** – добавлен слушатель для настройки приоритета потоков и для остановки движения пчел

***Описание функционала программы:***

myWind.**priorityStream2**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **int** priority = myWind.**priorityStream**.getSelectedIndex() + 1;  
 **trutAI**.setPriority(priority);  
 }  
});  
  
myWind.**priorityStream**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **int** priority = myWind.**priorityStream**.getSelectedIndex() + 1;  
 **workerAI**.setPriority(priority);  
 }  
});

myWind.**startStream1**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **if** (**workerAI**.**isStoped**)  
 {  
 **workerAI**.**isStoped** = **false**;  
 **synchronized** (**workerAI**) {  
 **workerAI**.notify();  
 }  
 }  
 **else  
 workerAI**.**isStoped** = **true**;  
 }  
});  
  
myWind.**startStream2**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **if** (**trutAI**.**isStoped**)  
 {  
 **trutAI**.**isStoped** = **false**;  
 **synchronized** (**trutAI**) {  
 **trutAI**.notify();  
 }  
 }  
 **else  
 trutAI**.**isStoped** = **true**;  
 }  
});

**public class** BasicAI **extends** Thread  
{  
 ArrayList<Bee> **mas**;  
 **boolean isWorker**;  
 **boolean isStoped** = **false**;  
 **boolean isFinished** = **false**;  
 BasicAI(ArrayList<Bee> mas, **boolean** isWorker)  
 {  
 **this**.**mas** = mas;  
 **this**.**isWorker** = isWorker;  
 }  
 @Override  
 **public void** run() { // метод, описывающий то, что должно выполняться потокм  
 **while** (**true**)  
 {  
 **synchronized** (**mas**)  
 { // если элемент вектора – объект рабочей пчелы, меняем координаты рабочей пчелы  
 **for** (**int** i = 0; i < **mas**.size(); i++) {  
 **if** (**isWorker**) {  
 **if** (**mas**.get(i) **instanceof** BeeWorker)  
 **mas**.get(i).move();  
 }  
 **else** { // иначе меняем координаты трутня  
 **if** (**mas**.get(i) **instanceof** BeeTruten)  
 **mas**.get(i).move();  
 }  
 }  
 }  
 **if** (**isStoped**) { // приостановка потока  
 **try**{   
 **synchronized** (**this**) {  
 wait();  
 }  
 } **catch** (InterruptedException e) {  
 System.***out***.println(e);  
 }  
 }  
 **if** (**isFinished**) // для остановки потока, т.е. он остановится, когда мы скажем ему, что задача выполнена и прервется бесконечный цикл  
 **break**;  
 **try** {  
 Thread.*sleep*(100);  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
  
 }  
}

Производные классы пчел были дополнены: переопределен метод move ( прописана логика движения)

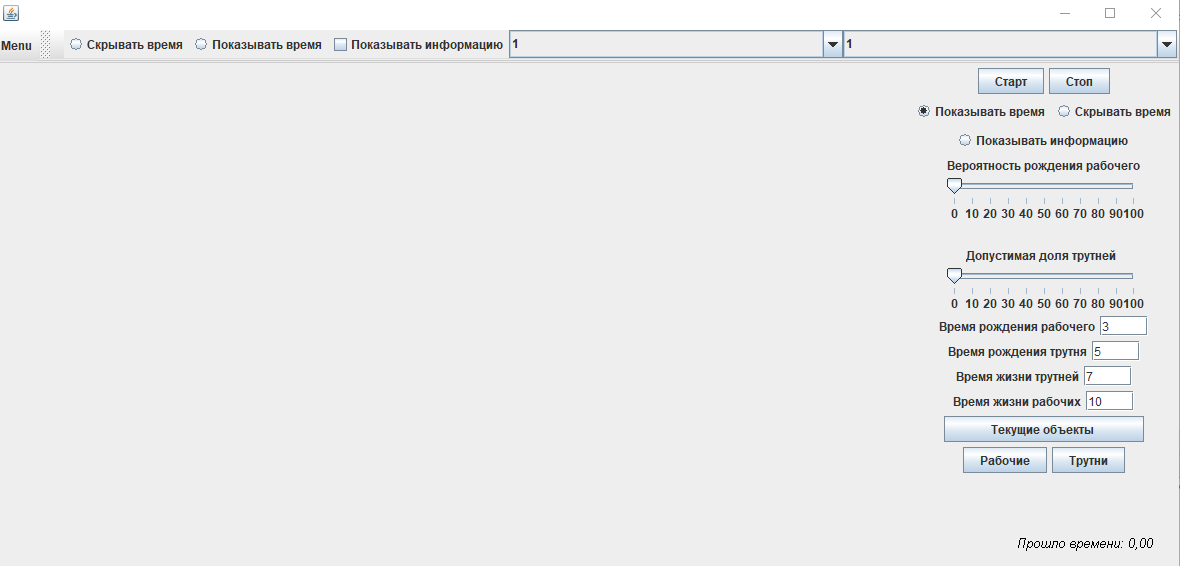
*Для пчелы рабочего:*

**public void** move()  
{  
 **double** dx;  
 **double** dy;  
 **double** targetX;  
 **double** targetY;  
 **if** (**isToStart**)  
 {  
 targetX = 20;  
 targetY = 80;  
 }  
 **else** {  
 targetX = **startX**;  
 targetY = **startY**;  
 }  
 dx = targetX - **x**;  
 dy = targetY - **y**;  
  
 **double** len = Math.*sqrt*( dx \* dx + dy \* dy);  
 **if** (len <= **speed**)  
 {  
 **x** = targetX;  
 **y** = targetY;  
 **isToStart** = !**isToStart**;  
 }  
 **else** {  
 **double** time = len / **speed**;  
 **x** += dx / time;  
 **y** += dy / time;  
 }  
}

*Для пчелы-трутня:*

**public void** move() {  
 **if** (!**isMove**)  
 {  
 **isMove** = **true**;  
 **targetX** = (**int**)(Math.*random*() \* 680) + 40;  
 **targetY** = (**int**)(Math.*random*() \* 320) + 55;  
 }  
  
 **double** dx = **targetX** - **x**;  
 **double** dy = **targetY** - **y**;  
  
 **double** len = Math.*sqrt*( dx \* dx + dy \* dy);  
 **if** (len <= **speed**)  
 {  
 **x** = **targetX**;  
 **y** = **targetY**;  
 **isMove** = !**isMove**;  
 }  
 **else** {  
 **double** time = len / **speed**;  
 **x** += dx / time;  
 **y** += dy / time;  
 }  
  
}

**Интерфейс:**



**Вывод:**

В ходе выполнения лабораторной работы мы познакомились c потоками, многопоточностью и приоритетами потоков. Узнали, как синхронизировать работу потоков. Поработали с методами, описанными в классе Thread. Научились блокировать доступ к данным при работе с ними одним потоком.