МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

*Лабораторная работа №5 по дисциплине «Технология программирования»*

*Вариант 2*

Группа: АВТ-610

Студенты: Князев Р.И., Дунаев Н.Ю.

Преподаватель: Васюткина И. А.

Новосибирск 2018

**Темы:**

***Потоки ввода-вывода. Сериализация объектов в файл.***

**Задание:**

1. Изучить особенности реализации системы ввода-вывода в Java.
2. Доработать программу, созданную в лабораторной работе № 4:
3. добавить в главное меню команду «Консоль». По этой команде должно появляться немодальное диалоговое окно с многострочным текстовым полем, занимающим всю область окна. В это окно можно вводить команды по варианту. В это же окно выводится реакция программы на команду;
4. для передачи команды в основное окно программы использовать каналы ввода-вывода;
5. создать конфигурационный файл для программы. В конфигурационный файл должны сохраняться все настройки симуляции, т.е. все данные и состояния, которые задаются в панели управления программы. Конфигурационный файл должен читаться при запуске программы и записываться при выходе. Формат файла текстовый;
6. добавить в главное меню пункты «Загрузить» и «Сохранить». Команда «Сохранить» вызывает сериализацию всех «живых» объектов в ней. Команда «Загрузить» останавливает текущую симуляцию (если симуляция запущена) и загружает объекты из выбранного файла. Не забудьте скорректировать время рождения объектов. После открытия симуляцию можно запустить, загруженные объекты должны вести себя естественно;
7. использовать стандартные файловые диалоги.

***Вариант 2***

Реализовать в консоли команду «Вернуть количество живых пчел-рабочих/трутней». Как параметр в команду должен передаваться идентификатор вида объекта.

***Описание структуры классов:***

Для реализации программы были добавлены и изменены следующие классы:

* Добавлен класс **ConsoleThread –** класс, реализующий консоль
* В класс **Habitat** были добавлены методы, реализующие сохранение параметров симуляции и сериализацию объектов.

***Описание функционала программы:***

// кнопка «сохранить», вызывает метод save, описанный ниже

myWind.**save**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 myWind.**saveOrLoad**.showOpenDialog(**wind**);  
 **file** = myWind.**saveOrLoad**.getSelectedFile();  
 **try** {  
 save(**objectOutputStream**, **file**);  
 } **catch** (IOException e1) {  
 e1.printStackTrace();  
 } **catch** (ClassNotFoundException e1) {  
 e1.printStackTrace();  
 }  
 }  
});

// сериализация объектов при помощи ObjectOutputStream

**public void** save(File file) **throws** IOException, ClassNotFoundException  
{  
 **synchronized** (**mas**) {  
 ObjectOutputStream objectOutputStream = **new** ObjectOutputStream(**new** FileOutputStream(file));  
 objectOutputStream.writeObject(**time**);  
 objectOutputStream.writeObject(**mas**);  
 objectOutputStream.flush();  
 objectOutputStream.close();  
 }  
}

// кнопка «загрузить». Вызывает метод load, описанный ниже, а также очищает все состояния параметров пчел  
myWind.**load**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 myWind.**saveOrLoad**.showOpenDialog(**wind**);  
 **file** = myWind.**saveOrLoad**.getSelectedFile();  
 **try** {  
 load(**objectInputStream**, **file**);  
 } **catch** (IOException e1) {  
 e1.printStackTrace();  
 } **catch** (ClassNotFoundException e1) {  
 e1.printStackTrace();  
 }  
 BeeTruten.*trNumber*=0;  
 **for**(**int** i = 0; i < **mas**.size(); i++)  
 {  
 **mas**.get(i).**bornTime** -= **time**;  
 **if**(**mas**.get(i) **instanceof** BeeTruten) BeeTruten.*trNumber*++;  
 }  
 **wind**.setSimulationState(**true**);  
 **time** = 0;  
 **secForTrut** =0;  
 **secForWorker**=0;  
 Bee.*n* = **mas**.size();  
 startSimulation(**wind**);  
 }  
});

// десериализация

**public** ArrayList<Bee> load(File file) **throws** IOException, ClassNotFoundException  
{  
 ObjectInputStream objectInputStream;  
 **try** {  
 objectInputStream = **new** ObjectInputStream(**new** FileInputStream(file));  
 }**catch** (StreamCorruptedException e){  
 System.***out***.println(**"Задан некорректный файл!"**);  
 **return null**;  
 }  
 **double** dt = (**double**) objectInputStream.readObject();  
 **mas** =(ArrayList<Bee>) objectInputStream.readObject();  
 objectInputStream.close();  
 **return mas**;  
}

// чтение параметров симуляции из файла при помощи DataInputStream

**public void** in(DataInputStream dataInputStream){  
 **try**{  
 dataInputStream = **new** DataInputStream(**new** FileInputStream(**"config.txt"**));  
 **timeSpawnTruten** = dataInputStream.readInt();  
 **timeSpawnWorker** = dataInputStream.readInt();  
 **timeTrutLiving** = dataInputStream.readInt();  
 **timeWorkLiving** = dataInputStream.readInt();  
 **trutenPersent** = dataInputStream.readFloat();  
 **workerCreateChance** = dataInputStream.readFloat();  
 dataInputStream.close();  
 }**catch** (IOException e){  
 e.printStackTrace();  
 }  
  
 System.***out***.println(**"Time spawn trut: "** + **timeSpawnTruten** + **" Time spawn work"** + **timeSpawnWorker** + **" Work create chance: "** + **workerCreateChance** + **" truten peesent: "** + **trutenPersent**);  
}

// запись параметров симуляции в файл при помощи DataOutputStream  
**public void** out(DataOutputStream dataOutputStream)  
{ **try** {  
 dataOutputStream = **new** DataOutputStream(**new** FileOutputStream(**"config.txt"**));  
}**catch** (IOException e){  
 e.printStackTrace();  
}  
  
 System.***out***.println(**"Time spawn trut: "** + **timeSpawnTruten** + **" Time spawn work"** + **timeSpawnWorker** + **" Work create chance: "** + **workerCreateChance** + **" truten peesent: "** + **trutenPersent**);  
 **try**{  
 dataOutputStream.writeInt(**timeSpawnTruten**);  
 dataOutputStream.writeInt(**timeSpawnWorker**);  
 dataOutputStream.writeInt(**timeTrutLiving**);  
 dataOutputStream.writeInt(**timeWorkLiving**);  
 dataOutputStream.writeFloat(**trutenPersent**);  
 dataOutputStream.writeFloat(**workerCreateChance**);  
 dataOutputStream.close();  
 }**catch** (IOException e){  
 e.printStackTrace();  
 }  
}

// консоль  
**public class** ConsoleThread **extends** Thread {  
 JTextArea **area**;  
 **int n**;  
 **int num**;  
 **private int lines** = 2;  
 String **lastStr**;  
 **boolean isInt** = **false**;  
  
 PipedWriter **wr** = **new** PipedWriter();  
 ConsoleThread(JTextArea area){  
 **this**.**area** = area;  
 }  
  
 **public void** run()  
 {  
 **while** (**true**)  
 {  
 **n** = **area**.getLineCount();  
 **if** (**n** > **lines**) {  
 **lines** = **n**;  
 **try** {  
 **lastStr** = **area**.getText();  
 **lastStr** = **lastStr**.substring(**area**.getLineEndOffset(**n**-3));  
 Pattern re = Pattern.*compile*(**"^ \*?returnNum ([a-zA-Z]{4})$"**,Pattern.***MULTILINE***);  
 Matcher match = re.matcher(**lastStr**);  
 **if** ((match.find()) && ((match.group(1).equals(**"work"**))|| (match.group(1).equals(**"trut"**))))  
 {

//в зависимости от команды возвращаем кол-во нужных пчел  
 **if** (match.group(1).equals(**"work"**))  
 **wr**.write(**"w"**);  
 **if** (match.group(1).equals(**"trut"**))  
 **wr**.write(**"t"**);  
 **synchronized** (**this**)  
 {  
 **isInt** = **true**;  
 wait();  
 }  
 **isInt** = **false**;  
 **area**.setText(**area**.getText() + **num** + **"\n"**);  
 **lines** += 1;  
 **area**.setCaretPosition(**area**.getLineEndOffset(**lines**-3)+2);  
 }  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 **try** {  
 Thread.*sleep*(50);  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
}

**if** (**consThread** != **null**)  
{  
 **int** item;  
 **if** (**consThread**.**isInt**) {  
 **try** {  
 item = **red**.read();  
 **if** ((**char**) item == **'w'**)  
 **consThread**.**num** = Bee.*n* - BeeTruten.*trNumber*;  
 **if** ((**char**) item == **'t'**)  
 **consThread**.**num** = BeeTruten.*trNumber*;  
 **synchronized** (**consThread**) {  
 **consThread**.notify();  
 }  
  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

**Вывод:**

В ходе выполнения лабораторной работы мы изучили различные потоки ввода-вывода, научились записывать данные в файл в разных форматах, а также производить сериализацию и десериализацию объектов. Научились реализовывать передачу данными между потоками.