Министерство науки и высшего образования РФ

**НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

Кафедра Вычислительной техники

Новосибирск 2020

|  |  |
| --- | --- |
| Факультет: АВТФ  Группа: АВТ-814  Студент: Альжанов В.А. | Преподаватель:  Васюткина Ирина Александровна |

Лабораторная работа 5

«Потоки ввода-вывода. Сериализация объектов в файл»

по дисциплине «Технологии программирования»

**Цель работы**

1. Изучить особенности реализации системы ввода-вывода в Java.
2. Доработать программу, созданную в лабораторной работе № 4.

**Задание**

Вариант 2

1. 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;
2. для передачи команды в основное окно программы использовать каналы ввода-вывода;
3. 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;
4. 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;
5. использовать стандартные файловые диалоги.

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

**Демонстрация работы программы**

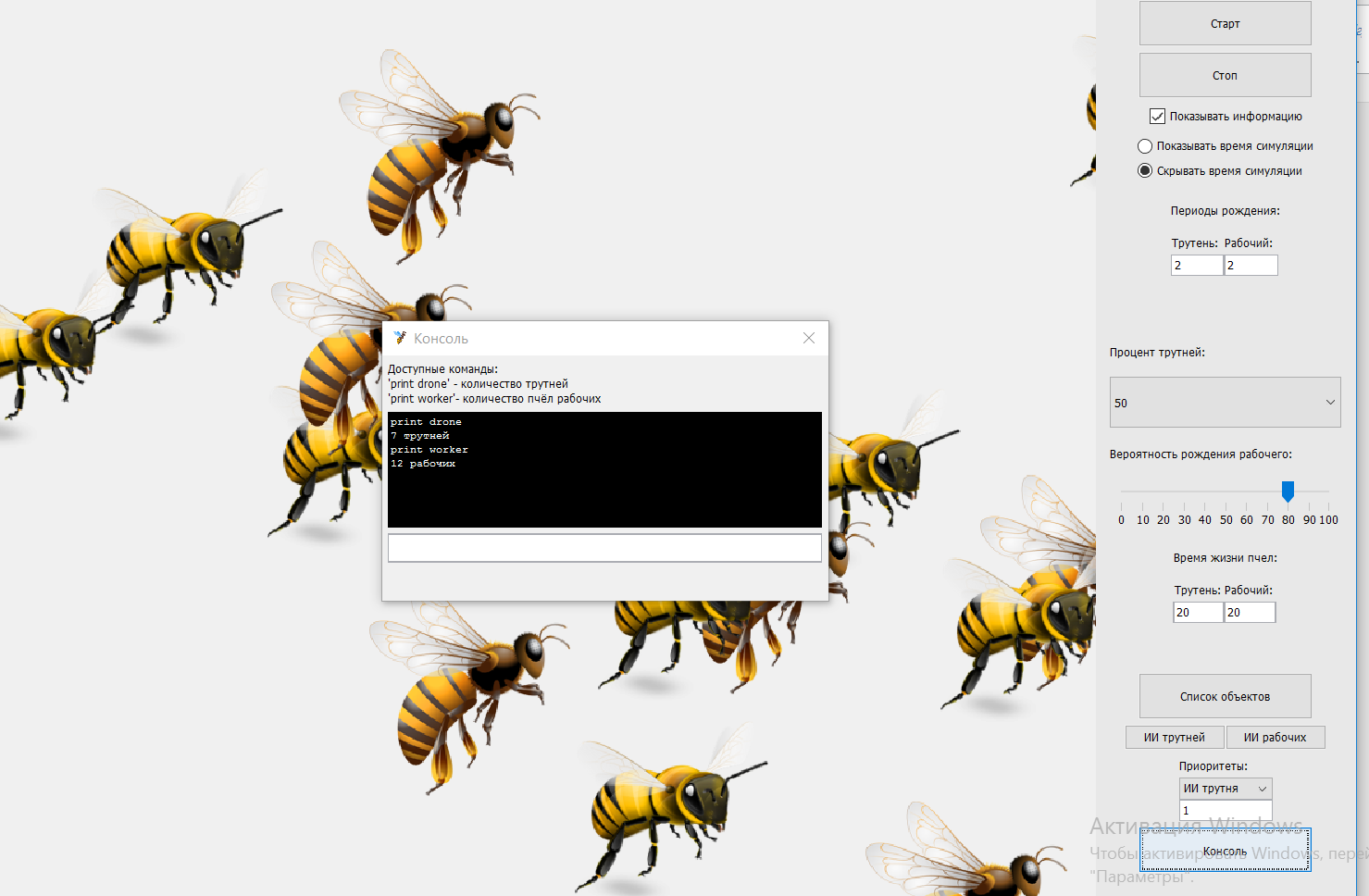


Рис. 1 – Реализация консоли

**Описание блоков программы**

Для реализации консоли был написан класс Console, унаследованный от класса JDialog. Ввод данных происходит через потоки ввода-вывода PipedInputStream/PipedOutputStream. Доступны команды “ print drone”(выводит количество трутней), “ print worker” (выводит количество пчел рабочих) .Класс унаследован от интерфейса Runnable, что позволяет реализовать работу потока.

import javax.swing.\*;  
import java.awt.\*;  
import java.io.IOException;  
import java.io.PipedInputStream;  
import java.io.PipedOutputStream;  
import java.io.PrintStream;  
import java.util.Scanner;  
  
public class Console extends JDialog implements Runnable {  
 private PipedOutputStream outputFromField;  
 private final PipedInputStream inputFromField;  
 private Scanner fieldInput;  
 private PrintStream fieldOutput;  
 private JTextField inputField;  
 private JTextArea outputArea;  
  
  
 Console(GUI owner){  
 super(owner, "Консоль", false);  
 setLayout(new FlowLayout(FlowLayout.*LEFT*));  
 setLocation(650, 200);  
 setDefaultCloseOperation(*DISPOSE\_ON\_CLOSE*);  
 setSize(400, 250);  
 setVisible(true);  
 requestFocus();  
  
 JLabel commandHint = new JLabel("<html>Доступные команды: <br>'print drone' - количество трутней " +  
 "<br>'print worker'- количество пчёл рабочих");  
 add(commandHint);  
  
 outputArea = new JTextArea();  
 outputArea.setEditable(false);  
 outputArea.setPreferredSize(new Dimension(getWidth() - 25, 100));  
 outputArea.setBackground(Color.*BLACK*);  
 outputArea.setForeground(Color.*WHITE*);  
 add(outputArea);  
  
 inputField = new JTextField();  
 inputField.setPreferredSize(new Dimension(getWidth() - 25, 25));  
 add(inputField);  
  
 outputFromField = new PipedOutputStream();  
 inputFromField = new PipedInputStream();  
 try {  
 outputFromField.connect(inputFromField);  
 } catch (IOException e) {  
 e.printStackTrace();  
 System.*exit*(1);  
 }  
 fieldInput = new Scanner(inputFromField);  
 fieldOutput = new PrintStream(outputFromField);  
  
  
 inputField.addActionListener(e -> {  
 String text = inputField.getText();  
 fieldOutput.println(text);  
 inputField.setText("");  
 synchronized (inputFromField) {  
 inputFromField.notify();  
 }  
 });  
 Thread mainProgram = new Thread(this);  
 mainProgram.start();  
 }  
 public void run() {  
 while (fieldInput.hasNextLine()) {  
 String line = fieldInput.nextLine();  
 outputArea.append(line + "\n");  
 System.*out*.println("Program recieved input: " + line);  
  
 try {  
 if (line.contains("print drone")) {  
  
 outputArea.append(Habitat.*numberOfDrones*+" трутней" + "\n");  
 } else if (line.contains("print worker")) {  
 outputArea.append(Habitat.*numberOfWorkers*+" рабочих" + "\n");  
 }  
 else {  
 throw new Exception();  
 }  
 } catch (Exception e2) {  
 JOptionPane.*showMessageDialog*(null,  
 "Введена неверная команда.",  
 "Ошибка", JOptionPane.*ERROR\_MESSAGE*);  
 }  
 }  
 }  
}

Сериализация в файл и десериализация из файла были описаны в классе GUI и помещены в меню «Файл». При сериализации значения помещаются в файл save.dat, находящийся в папке data. При десериализации происходит процедура чтения данных из этого файла, воссоздаются объекты, после чего каждому объекту присваивается состояние, записанное в файл.

JMenu menuFile = new JMenu("Файл");  
menuBar.add(menuFile);  
  
  
JMenuItem saveItem = new JMenuItem("Сохранить");  
menuFile.add(saveItem);  
saveItem.addActionListener(e -> {  
  
 FileOutputStream outputStream = null;  
 try {  
 outputStream = new FileOutputStream("data/save.dat");  
 } catch (FileNotFoundException e1) {  
 e1.printStackTrace();  
 }  
 ObjectOutputStream objectOutputStream = null;  
 try {  
 objectOutputStream = new ObjectOutputStream(outputStream);  
 } catch (IOException e1) {  
 e1.printStackTrace();  
 }  
 // сохраняем в файл  
 try {  
 if (objectOutputStream != null) {  
 objectOutputStream.writeObject(Habitat.*array*);  
 objectOutputStream.writeObject(Habitat.*identifiers*);  
 objectOutputStream.writeObject(Habitat.*birthdays*);  
 System.*out*.println("Saved successfully!");  
 }  
 } catch (IOException e1) {  
 e1.printStackTrace();  
 }  
  
 try {  
 if (objectOutputStream != null) {  
 objectOutputStream.close();  
 }  
 } catch (IOException e1) {  
 e1.printStackTrace();  
 }  
});  
  
JMenuItem loadItem = new JMenuItem("Загрузить");  
menuFile.add(loadItem);  
loadItem.addActionListener(e -> {  
 FileInputStream fileInputStream = null;  
 try {  
 fileInputStream = new FileInputStream("data/save.dat");  
 } catch (FileNotFoundException e1) {  
 e1.printStackTrace();  
 }  
 ObjectInputStream objectInputStream = null;  
 try {  
 objectInputStream = new ObjectInputStream(fileInputStream);  
 } catch (IOException e1) {  
 e1.printStackTrace();  
 }  
  
 try {  
 if (objectInputStream != null) {  
 Habitat.*array*.clear();  
 Habitat.*numberOfDrones* = 0;  
 Habitat.*numberOfWorkers* = 0;  
 time = 0;  
 Habitat.*array* = (ArrayList<Bee>) objectInputStream.readObject();  
 for (int i = 0; i < Habitat.*array*.size(); i++) {  
 Habitat.*array*.get(i).timeOfBirth = 0;  
 }  
 }  
 } catch (IOException | ClassNotFoundException e1) {  
 e1.printStackTrace();  
 }  
  
 try {  
 if (objectInputStream != null) {  
 Habitat.*identifiers* = (HashSet<Integer>) objectInputStream.readObject();  
 }  
 } catch (IOException | ClassNotFoundException e1) {  
 e1.printStackTrace();  
 }  
 int mapSize = Habitat.*array*.size();  
 Habitat.*birthdays* = new TreeMap<>();  
 for (int i = 0; i < mapSize; i++) {  
 Habitat.*birthdays*.put(Habitat.*array*.get(i).id, 0L);  
 }  
 for (int i = 0; i < Habitat.*array*.size(); i++) {  
 if (Habitat.*array*.get(i) instanceof Drone) {  
 Habitat.*numberOfDrones*++;  
 } else {  
 Habitat.*numberOfWorkers*++;  
 }  
 }  
});

Конфигурация приложения записывается в файл config.txt при закрытии приложения. При запуске приложения значениям переменных присваиваются значения, прочитанные из файла.

void saveConfig() throws IOException {  
 FileOutputStream outputStream = new FileOutputStream("data/config.txt");  
 ObjectOutputStream objectOutputStream = new ObjectOutputStream(outputStream);  
 objectOutputStream.writeObject(statShow.isSelected() + "," + showSimTime.isSelected()  
 + "," + Habitat.*N1* + "," + Habitat.*N2* + ","  
 + Habitat.*P* + "," + Habitat.*K* + "," + Habitat.*droneTimeOfLife* + "," + Habitat.*WorkerTimeOfLife*);  
 System.*out*.println("Config saved.");  
 objectOutputStream.close();  
}  
  
void loadConfig() throws Exception {  
 FileInputStream fileInputStream = new FileInputStream("data/config.txt");  
 ObjectInputStream objectInputStream = new ObjectInputStream(fileInputStream);  
 String config = objectInputStream.readObject().toString();  
 String[] parts = config.split(",", 8);  
 for (int i = 0; i < 8; i++) {  
 System.*out*.println(parts[i]);  
 }  
  
 isStatShow = Boolean.*valueOf*(parts[0]);  
 isShowSimTime = Boolean.*valueOf*(parts[1]);  
 Habitat.*N1* = Integer.*valueOf*(parts[2]);  
 Habitat.*N2* = Integer.*valueOf*(parts[3]);  
 Habitat.*P* = Integer.*valueOf*(parts[4]);  
 Habitat.*K* = Integer.*valueOf*(parts[5]);  
 Habitat.*droneTimeOfLife* = Integer.*valueOf*(parts[6]);  
 Habitat.*WorkerTimeOfLife* = Integer.*valueOf*(parts[7]);  
}

**Выводы**

В ходе выполненной работы были изучены особенности сериализации и десериализации, а также работа потоков ввода-вывода в Java.