МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

НГТУ

Лабораторная работа №5

по предмету «Технология программирования»

на тему «Потоки ввода-вывода. Сериализация объектов в файл»

Факультет: АВТ Преподаватель:

Группа: АВТ-809 Михайленко Д.А.

Студент: Швыдко Е.А.

Новосибирск 2020

Задание:

***Вариант 11***

Реализовать в консоли команды «Остановить генерацию ХХХ объектов» и «Продолжить генерацию ХХХ объектов». Чекбокс в интерфейсе должен вести себя так же, как если бы использовали его, а не исполняли команды (галочка появляется/исчезает).

Цель работы:

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 обрабатывающий поток 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.

Ход разработки:

* В главное меню было добавлена команда «Console», по нажатию на которую отображалась «консоль» - диалоговое окно с многострочным текстовым полем TextArea

**public** **class** Console **extends** JDialog {

**private** JTextArea consoleArea;

**private** JTextField inputTextField;

**private** **String** msg;

**private** MFrame \_parent;

**Console**(MFrame parent){

**super**(parent,"Консоль");

**this**.\_parent = parent;

setSize(450,460);

setResizable(**false**);

setLocationRelativeTo(parent);

**OutputStream** s = **new** **OutputStream**();

**InputStream** t = **new** **InputStream**(s.getStream(), parent);

t.start();

JPanel panelDialog = **new** JPanel();

consoleArea = **new** JTextArea(60,10);

JScrollPane consoleScrollPane = **new** JScrollPane(consoleArea);

setFocusable(**true**);

msg = "�"оступные команды:\nОстановить генерацию XXX объектов - stop dog/cat\n�-апустить генерацию XXX объектов - start dog/cat\n"***;***

consoleArea.setText(msg);

consoleArea.addKeyListener(**new** KeyAdapter() {

**@Override**

**public** void keyReleased(KeyEvent e) {

**if** (e.getKeyCode() == KeyEvent.VK\_ENTER) {

**if**(consoleArea.getText().charAt(consoleArea.getText().length() - 1) == '\n' && consoleArea.getText().charAt(consoleArea.getText().length() - 2) == '\n') {

consoleArea.setText(consoleArea.getText().substring(0, consoleArea.getText().length() - 1));

**return**;

}

msg = consoleArea.getText();

**switch** (getString(msg))

{

**case** "stop dog": {

s.goWrite(1);

msg += "�"енерация собак остановлена\n"***;***

**break**;

}

**case** "stop cat": {

s.goWrite(2);

msg += "�"енерация кошек остановлена\n"***;***

**break**;

}

**case** "start cat": {

s.goWrite(3);

msg += "�"енерация кошек запущена\n"***;***

**break**;

}

**case** "start dog": {

s.goWrite(4);

msg += "�"енерация собак запущена\n";

**break**;

}

}

consoleArea.setText(msg);

}

}

});

consoleScrollPane.setPreferredSize(**new** Dimension(400,387));

panelDialog.setLayout(**new** FlowLayout());

consoleArea.setPreferredSize(**new** Dimension(400,387));

consoleArea.setEditable(**true**);

Font font = consoleArea.getFont();

consoleArea.setFont(font.deriveFont(font.getSize()+3.5f));

panelDialog.add(consoleScrollPane);

setContentPane(panelDialog);

}

**public** **String** getString(**String** string){

char[] ch = string.toCharArray();

int fpos = 0, lastpos = 0;

**for**(int i = 0; i < ch.length; i++){

**if**(ch.length - 1 == i) {

fpos = lastpos;

lastpos = ch.length;

}

**if**(ch[i] == '\n') lastpos = i;

}

**return** string.substring(fpos + 1, lastpos);

}

}

* Был реализована работа конфигурационного файла, хранящего в себе параметры симуляции. При закрытии окна или по нажатию на «крестик» запускается метод save() сохраняющий текущие параметры симуляции в текстовый файл. При запуске программы запускается метод load(), который записывает в параметры симуляции значения, прочитанные из текстового файла

**public** void save(){

**try** {

**File** file = **new** **File**("Configs.txt");

**if** (!file.exists()) file.createNewFile();

**PrintWriter** pw = **new** **PrintWriter**(file);

pw.println(Habitat.dogTTL);

pw.println(Habitat.catTTL);

pw.println(Habitat.dogDelay);

pw.println(Habitat.catDelay);

pw.println(Habitat.dogProb);

pw.println(Habitat.catProb);

pw.close();

} **catch** (**IOException** e) {

**System**.out.print("Error: " + e);

}

}

**public** void load(){

**BufferedReader** br = **null**;

**File** file = **new** **File**("SaveFile.txt");

**if** (!file.exists()) **return**;

**try** {

br = **new** **BufferedReader**(**new** **FileReader**(file));

Habitat.dogTTL = (**Double**.parseDouble(br.readLine()));

Habitat.catTTL = (**Double**.parseDouble(br.readLine()));

Habitat.dogDelay = **Double**.parseDouble(br.readLine());

Habitat.catDelay = (**Double**.parseDouble(br.readLine()));

Habitat.dogProb = (**Double**.parseDouble(br.readLine()));

Habitat.catProb = (**Double**.parseDouble(br.readLine()));

} **catch** (**IOException** e) {

**System**.out.print("Error: " + e);

}

**finally** {

**try** {

**assert** br != **null**;

br.close();

}

**catch**(**IOException** e) {

**System**.out.print("Error: " + e);

}

}

}

* В главное меню были добавлены пункты меню «Load» и «Save». По нажатию на кнопку «Load» все текущие «живые» объекты и все сопутствующие коллекции с параметрами этих объектов сериализуются и записываются в файл. По нажатию на «Save» все коллекции десериализуются и заносятся в оперативную память.

**public** void saveObjects() {

**File** file = **new** **File**("SaveObjects.obj");

**FileOutputStream** fos = **null**;

**try** {

fos = **new** **FileOutputStream**(file);

} **catch** (**FileNotFoundException** e) {

e.printStackTrace();

}

**ObjectOutputStream** oos = **null**;

**try** {

oos = **new** **ObjectOutputStream**(fos);

} **catch** (**IOException** e) {

e.printStackTrace();

}

**try** {

**if**(oos != **null**) {

oos.writeObject(Singleton.getVector().getArray());

oos.writeObject(Singleton.getVector().getIds());

oos.writeObject(Singleton.getVector().getTTL());

oos.close();

}

}

**catch** (**IOException** e2) {

e2.printStackTrace();

}

}

**public** void loadObjects(File \_file) **throws** IOException, ClassNotFoundException {

**if**(!(\_file.exists()) || !(\_file.getName().endsWith(".obj"))) {

JOptionPane.showMessageDialog(**this**, "Choose '.obj' file");

**System**.out.println("Choose '.obj' file");

**return**;

}

**FileInputStream** fis = **null**;

**try** {

fis = **new** **FileInputStream**(\_file);

} **catch** (**FileNotFoundException** e) {

e.printStackTrace();

}

**ObjectInputStream** ois = **null**;

**try** {

ois = **new** **ObjectInputStream**(fis);

} **catch** (**IOException** e) {

e.printStackTrace();

}

Singleton.setVector((**Vector**<Animal>) ois.readObject());

Singleton.setTS((**TreeSet**<**Integer**>)ois.readObject());

Singleton.setTTL((**HashMap**<**Integer**, **Double**>) ois.readObject());

ois.close();

**for**(**HashMap**.**Entry**<**Integer**, **Double**> carsTTL : Singleton.getVector().getTTL().entrySet()) carsTTL.setValue(counter/10);

habitat.repaint();

}

Выводы:

В результате выполнения работы были изучены каналы ввода-вывода и их использование. Была рассмотрена сериализация объектов.