

Лабораторная работа №5

Технология программирования

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

Студент: Воликов М.В.

Группа: АВТ-910

Вариант: 4

Преподаватель: Дубков И.С.



НОВОСИБИРСК 2021

Оглавление

[Задание 2](#_Toc38223051)

[Ход работы 2](#_Toc38223052)

[Вывод 6](#_Toc38223053)

# Задание

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

* 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;
* для передачи команды в обрабатывающий поток использовать каналы ввода-вывода;
* 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;
* 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;

**Вариант 4**

Реализовать в консоли команду «Сократить число кроликов-альбиносов на N%». Как параметр в команду должно передаваться значение N%.

# Ход работы

## Консоль

В главное меню было добавлено соответствующее поле:

JMenuItem StartConsole = new JMenuItem("Консоль");  
StartConsole.addActionListener(e -> {  
 jDialogConsole.setVisible(true);  
});

menuBar.add(StartConsole);

Также было создано не модальное диалоговое окно с текстовой зоной, занимающей всё место.

JDialog jDialogConsole = new JDialog(this,"Консоль",false);

jDialogConsole.setSize(500, 300);  
jDialogConsole.setLocation(350,200);  
jTextAreaConsole.addKeyListener(new KeyAdapter() {  
 @Override  
 public void keyPressed(KeyEvent e) {  
 super.keyPressed(e);  
 if (e.getKeyCode() == KeyEvent.*VK\_ENTER*) {  
 try {  
 int offset = jTextAreaConsole.getLineStartOffset(jTextAreaConsole.getLineCount() - 1);  
 int length = jTextAreaConsole.getLineEndOffset(jTextAreaConsole.getLineCount() - 1) - offset + 1;  
 String string = jTextAreaConsole.getText(offset, length);  
 pipedWriter.write(string);  
 console.Start();  
 } catch (BadLocationException | IOException ex) {  
 ex.printStackTrace();  
 }  
 }  
 }  
});  
jDialogConsole.add(jTextAreaConsole);

Был создан класс Console, считывающий команды и выполняющий их.

public class Console extends Thread {  
  
 PipedReader pipedReader;  
 int N;  
  
 Console () {  
 pipedReader = new PipedReader();  
 }  
  
 @Override  
 public void run() {  
 super.run();  
 char[] buf = new char[100];  
 try {  
 pipedReader.read(buf);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 String string = new StringBuilder().append(buf).toString();  
 if (string.substring(0, 9).equals("Cut back ")) {  
 int len = string.indexOf("\n");  
 if (len == 10) {  
 N = Integer.*parseInt*(string.substring(9,10));  
 }  
 if (len == 11) {  
 N = Integer.*parseInt*(string.substring(9,11));  
 }  
 }  
 double persent = (100 - ((double) N)) / 100 \* Albino.*AlbinoQuantity*;  
  
 synchronized (Singleton.*getVector*()) {  
 while (Albino.*AlbinoQuantity* != (int) persent) {  
 int i = 0;  
 if (Singleton.*getVector*().get(i) instanceof Albino) {  
 Rabbit.*AllQuantity*--;  
 Albino.*AlbinoQuantity*--;  
 Singleton.*getHashMap*().remove(Singleton.*getVector*().get(i).ID);  
 Singleton.*getTreeSetID*().remove(Singleton.*getVector*().get(i).ID);  
 Singleton.*getVector*().remove(i++);  
 }  
 }  
 }  
 }  
  
 public void Start() {  
 run();  
 }  
  
}

Объект класса консоль создаётся в конструкторе класса GUI.

Console console;

console = new Console();

Связь между консолью и классом GUI поддерживается за счёт канала PipedReader/PipedWriter.

jTextAreaConsole.addKeyListener(new KeyAdapter() {  
 @Override  
 public void keyPressed(KeyEvent e) {  
 super.keyPressed(e);  
 if (e.getKeyCode() == KeyEvent.*VK\_ENTER*) {  
 try {  
 int offset = jTextAreaConsole.getLineStartOffset(jTextAreaConsole.getLineCount() - 1);  
 int length = jTextAreaConsole.getLineEndOffset(jTextAreaConsole.getLineCount() - 1) - offset + 1;  
 String string = jTextAreaConsole.getText(offset, length);  
 pipedWriter.write(string);  
 console.Start();  
 } catch (BadLocationException | IOException ex) {  
 ex.printStackTrace();  
 }  
 }  
 }  
});

## Конфигурационный файл

Для сохранения пользовательских настроек был создан текстовый файл “Configuration.txt”. Запись в этот файл производится классе Saver посредством потока FileWriter.

public class Saver {  
/\* Порядок файла:  
\* P (индекс выбранного элемента)  
\* K (индекс выбранного элемента)  
\* N обыкновенных (текстовое поле)  
\* N альбиносов (текстовое поле)  
\* Время жизни обычных  
\* Время жизни альбиносов  
\* Приоритет обычных  
\* Приоритет альбиносов  
\*/  
 private GUI gui;  
  
 public Saver(GUI gui) {  
 this.gui = gui;  
 }  
  
 public void SaveConfigurations() {  
 Runtime.*getRuntime*().addShutdownHook(new Thread() {  
 @Override  
 public void run() {  
 super.run();  
 File file = new File("Configuration.txt");  
 try {  
 FileWriter fileWriter = new FileWriter(file);  
 fileWriter.write(gui.jComboBox.getSelectedIndex() + " ");  
 fileWriter.write(gui.jList.getSelectedIndex() + " ");  
 fileWriter.write(gui.jTextFieldNOrdinary.getText() + " ");  
 fileWriter.write(gui.jTextFieldNAlbino.getText() + " ");  
 fileWriter.write(gui.jTextFieldLiveTimeOrdinary.getText() + " ");  
 fileWriter.write(gui.jTextFieldLiveTimeAlbino.getText() + " ");  
 fileWriter.write(gui.jComboBoxOrdinaryPriority.getSelectedIndex() + " ");  
 fileWriter.write(Integer.*toString*(gui.jComboBoxAlbinoPriority.getSelectedIndex()));  
 fileWriter.flush();  
 fileWriter.close();  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 });  
 }  
}

## Сериализация

Для выполнения сериализации объектов «Кролик» был создан класс Serialization. В классе созданы два метода: один для сериализации, второй для десериализации. Реализация этого класса:

public class Serializator {  
  
 public void Serialization() {  
 File file = new File("Objects.data");  
 FileOutputStream fileOutputStream = null;  
 ObjectOutputStream objectOutputStream = null;  
 try {  
 fileOutputStream = new FileOutputStream(file);  
 objectOutputStream = new ObjectOutputStream(fileOutputStream);  
 for (Rabbit rabbit : Singleton.*getVector*())  
 if (rabbit instanceof Ordinary)  
 objectOutputStream.writeObject((Ordinary) rabbit);  
 else  
 objectOutputStream.writeObject((Albino) rabbit);  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 } catch (IOException e) {  
 e.printStackTrace();  
 } finally {  
 try {  
 objectOutputStream.close();  
 fileOutputStream.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 public void Deserialization() {  
 FileInputStream fileInputStream = null;  
 ObjectInputStream objectInputStream = null;  
 try {  
 fileInputStream = new FileInputStream("Objects.data");  
 objectInputStream = new ObjectInputStream(fileInputStream);  
 Singleton.*getVector*().removeAllElements();  
 try {  
 while (true) {   
 Singleton.*getVector*().add((Rabbit) objectInputStream.readObject());  
 Singleton.*getTreeSetID*().add(Singleton.*getVector*().lastElement().ID);  
 Singleton.*getHashMap*().put(Singleton.*getVector*().lastElement().ID, Singleton.*getVector*().lastElement().BirthTime);  
 }  
 } catch (ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 } catch (IOException e) {  
 e.printStackTrace();  
 } finally {  
 try {  
 objectInputStream.close();  
 fileInputStream.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 }  
 }  
}

# Вывод

В процессе выполнения лабораторной работы были изучены потоки ввода-вывода и сериализация. Был доработан проект из предыдущих лабораторных работ.