МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра вычислительной техники

**Отчёт**

**по лабораторной работе №2**

**по дисциплине «Разработка клиент-серверных приложений»**

Тема: Работа с файлами

Группа: АВТ-909

Студенты: Буйко К.А. Пилецкий Д.К.

Преподаватель: Токарев В.Г.

НОВОСИБИРСК

2022

**Цель работы**

Изучить различные методы сохранения состояния объектов и применить их на практике.

**Задание**

В базовом классе реализовать (объявить) методы (абстрактные) чтения/записи объекта в двоичный и текстовый потоки. Переопределить во всех производных классах. В оконном классе реализовать методы сохранения и загрузки вектора объектов в тестовый, двоичный и сериализуемый файл (двоичная сериализация средствами Java и XML-сериализация с использованием библиотеки xstream). Формат двоичного и текстового файла содержит начальный счетчик объектов, а затем для каждого их них – имя и содержимое. При загрузке из файла создавать объект файла произвольного класса с помощью средств рефлексии (методы Class.forName и Class.newInstance).

**Ход выполнения работы**

В лабораторной работе дополнялись ранее разработанные классы объектов: Car (базовый класс), Habitat (основной класс).

В классе Habitat реализованы следующие методы:

SaveText – Сохраняет состояние всех объектов в текстовом виде.

public void SaveText(String filename) {  
 try {  
 FileWriter f = new FileWriter(filename);  
 f.write(String.*valueOf*(cars.size()) + '\n');  
 for (Car car : cars) {  
 car.saveText(f);  
 }  
 f.close();  
 System.*out*.println("save");  
 } catch (IOException ioException) {  
 ioException.printStackTrace();  
 }  
 System.*out*.println("Text file saved.");  
}

Load Text – Загружает состояние всех объектов в текстовом виде.

public void LoadText(String filename) {  
 try {  
 FileReader f = new FileReader(filename);  
 cars.clear();  
 int count = FileActions.*readInt*(f);  
 for (int i = 0; i < count; i++) {  
 String name = FileActions.*readStr*(f);  
 Car c;  
 try {  
 Class<?> t = Class.*forName*("Main." + name);  
 c = (Car) t.newInstance();  
 c.loadText(f);  
 cars.add(c);  
 } catch (ClassNotFoundException | IllegalAccessException | InstantiationException e) {  
 e.printStackTrace();  
 }  
 }  
 f.close();  
 System.*out*.println("load");  
 } catch (IOException ioException) {  
 ioException.printStackTrace();  
 }  
 System.*out*.println("Text file loaded.");  
}

saveBin – Сохраняет состояние всех объектов в бинарном виде

public void saveBin(String filename){  
 try {  
 FileOutputStream f = new FileOutputStream(filename);  
 f.write(BinFileAction.*IntToBytes*(cars.size()));  
 for (Car car : cars) {  
 car.saveBin(f);  
 }  
 f.close();  
 System.*out*.println("Bin saved");  
 } catch (IOException ioException) {  
 ioException.printStackTrace();  
 }  
 System.*out*.println("Binary file saved.");  
}

loadBin – Загружает состояние всех объектов в бинарном виде

public void loadBin(String filename) {  
 try {  
 FileInputStream f = new FileInputStream(filename);  
 cars.clear();  
 byte[] b = new byte[f.available()];  
 byte[] buf = new byte[90];  
 f.read(b);  
 int i=0;  
 int IntNum = 8;  
 int CrB = 89;  
 int PsB = 65;  
  
 for(int j=0; j<IntNum; j++, i++)  
 buf[j] = b[i];  
 int count = BinFileAction.*ByteToInt*(buf);  
 for (int k = 0; k < count; k++) {  
 int j=0;  
 StringBuilder sb = new StringBuilder();  
 while(true){  
 if((char)b[i]=='\n')  
 break;  
 sb.append((char)b[i]);  
 i++;  
 }  
 i++;  
 String name = sb.toString();  
 Car c;  
 int numB = PsB;  
 try {  
 Class<?> t = Class.*forName*("Main." + name);  
 c = (Car) t.newInstance();  
 if (name.equals("Cargo")) numB = CrB;  
 for(j=0; j<numB; j++, i++)  
 buf[j] = b[i];  
 c.loadBin(buf);  
 cars.add(c);  
 } catch (ClassNotFoundException | IllegalAccessException | InstantiationException e) {  
 e.printStackTrace();  
 }  
 }  
 f.close();  
 System.*out*.println("Bin loaded");  
 } catch (IOException ioException) {  
 ioException.printStackTrace();  
 }  
 System.*out*.println("Binary file loaded.");  
}

Serialize – Сериализует объекты

public void Serialize(String file){  
 try {  
 OutputStream fstream = new FileOutputStream(file);  
 ObjectOutputStream f = new ObjectOutputStream(fstream);  
 int i=0;  
 f.write(cars.size());  
 while (i<this.cars.size()) {  
 cars.get(i).Serialize(f);  
 i++;  
 }  
 f.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 System.*out*.println("Serialized.");  
}

Deserialize – Десериализует все объекты

public void Deserialize(String file){  
 cars.clear();  
 try {  
 InputStream fstream = new FileInputStream(file);  
 ObjectInputStream f = new ObjectInputStream(fstream);  
 int n = f.read();  
 int i=0;  
 while (i<n) {  
 String name = f.readUTF();  
 Car c;  
 try {  
 Class<?> t = Class.*forName*("Main." + name);  
 if (name.equals("Cargo"))  
 c = (Cargo) t.newInstance();  
 else  
 c = (PassCar) t.newInstance();  
 c.Deserialize(f);  
 cars.add(c);  
 } catch (ClassNotFoundException | IllegalAccessException | InstantiationException e) {  
 e.printStackTrace();  
 }  
 i++;  
 }  
 f.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 System.*out*.println("Deserialized.");  
}

SaveXML - Сохраняет состояние всех объектов в виде XML файла

public void SaveXML(String file){  
 try {  
 XMLEncoder f = new XMLEncoder(new FileOutputStream(file));  
 f.writeObject(cars.size());  
 int i=0;  
 while(i<cars.size())  
 {  
 f.writeObject(cars.get(i));  
 i++;  
 }  
 f.close();  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
 System.*out*.println("XML file saved.");  
}

LoadXML - Загружает состояние всех объектов в виде XML файла

public void LoadXML(String file){  
 try {  
 XMLDecoder f = new XMLDecoder(new FileInputStream(file));  
 Car.*InitImages*();  
 int n = (int) f.readObject();  
 int i=0;  
 cars.clear();  
 while(i<n)  
 {  
 cars.add((Car)f.readObject());  
 i++;  
 }  
 f.close();  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
 System.*out*.println("XML file loaded.");  
}

В классе Car также были реализованы соответствующие методы для записи и чтения в файлы различными методами.

saveText – Сохраняет состояние объекта в текстовом виде.

public void saveText(FileWriter f) {  
 try {  
 f.write(String.*valueOf*(cx) + '\n');  
 f.write(String.*valueOf*(cy) + '\n');  
 f.write(String.*valueOf*(w) + '\n');  
 f.write(String.*valueOf*(h) + '\n');  
 f.write(String.*valueOf*(ImInd) + '\n');  
 f.write(String.*valueOf*(isMove) + '\n');  
 f.write(String.*valueOf*(color.getRed()) + ';');  
 f.write(String.*valueOf*(color.getGreen()) + ';');  
 f.write(String.*valueOf*(color.getBlue()) + '\n');  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
}

loadText – Загружает состояние объекта в текстовом виде.

public void loadText(FileReader f) {  
 cx = FileActions.*readInt*(f);  
 cy = FileActions.*readInt*(f);  
 w = FileActions.*readInt*(f);  
 h = FileActions.*readInt*(f);  
 ImInd = FileActions.*readInt*(f);  
 isMove = FileActions.*readBool*(f);  
 color = FileActions.*readColor*(f);  
}

saveBin – Сохраняет состояние объекта в бинарном виде

public void saveBin(FileOutputStream f) {  
 try {  
 f.write(BinFileAction.*IntToBytes*(cx));  
 f.write(BinFileAction.*IntToBytes*(cy));  
 f.write(BinFileAction.*IntToBytes*(w));  
 f.write(BinFileAction.*IntToBytes*(h));  
 f.write(BinFileAction.*IntToBytes*(ImInd));  
 f.write(BinFileAction.*BoolToByte*(isMove));  
 f.write(BinFileAction.*ColorToBytes*(color));  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
}

loadBin – Загружает состояние объекта в бинарном виде

Для Cargo:

public void loadBin(byte[] b) {  
 byte[] buf = new byte[24];  
 int i = 0;  
 int intNum = 8;  
 for (int j = 0; j < intNum; j++, i++)  
 buf[j] = b[i];  
 cx = BinFileAction.*ByteToInt*(buf);  
  
 for (int j = 0; j < intNum; j++, i++)  
 buf[j] = b[i];  
 cy = BinFileAction.*ByteToInt*(buf);  
  
 for (int j = 0; j < intNum; j++, i++)  
 buf[j] = b[i];  
 w = BinFileAction.*ByteToInt*(buf);  
  
 for (int j = 0; j < intNum; j++, i++)  
 buf[j] = b[i];  
 h = BinFileAction.*ByteToInt*(buf);  
  
 for (int j = 0; j < intNum; j++, i++)  
 buf[j] = b[i];  
 ImInd = BinFileAction.*ByteToInt*(buf);  
  
 isMove = BinFileAction.*ByteToBool*(b[i]);  
 i++;  
  
 for (int j = 0; j < intNum \* 3; j++, i++)  
 buf[j] = b[i];  
 color = BinFileAction.*ByteToColor*(buf);  
  
 for (int j = 0; j < intNum; j++, i++)  
 buf[j] = b[i];  
 t = BinFileAction.*ByteToInt*(buf);  
  
 for (int j = 0; j < intNum; j++, i++)  
 buf[j] = b[i];  
 W = BinFileAction.*ByteToInt*(buf);  
  
 for (int j = 0; j < intNum; j++, i++)  
 buf[j] = b[i];  
 H = BinFileAction.*ByteToInt*(buf);  
}

Для PassCar:

public void loadBin(byte[] b) {  
 byte[] buf = new byte[24];  
 int i = 0;  
 int intNum = 8;  
 for (int j = 0; j < intNum; j++, i++)  
 buf[j] = b[i];  
 cx = BinFileAction.*ByteToInt*(buf);  
  
 for (int j = 0; j < intNum; j++, i++)  
 buf[j] = b[i];  
 cy = BinFileAction.*ByteToInt*(buf);  
  
 for (int j = 0; j < intNum; j++, i++)  
 buf[j] = b[i];  
 w = BinFileAction.*ByteToInt*(buf);  
  
 for (int j = 0; j < intNum; j++, i++)  
 buf[j] = b[i];  
 h = BinFileAction.*ByteToInt*(buf);  
  
 for (int j = 0; j < intNum; j++, i++)  
 buf[j] = b[i];  
 ImInd = BinFileAction.*ByteToInt*(buf);  
  
 isMove = BinFileAction.*ByteToBool*(b[i]);  
 i++;  
  
 for (int j = 0; j < intNum \* 3; j++, i++)  
 buf[j] = b[i];  
 color = BinFileAction.*ByteToColor*(buf);  
}

Serialize – Сериализует объекты

void Serialize(ObjectOutputStream f)  
{  
 try {  
 f.writeObject(this);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
}

Deserialize – Десериализует все объекты

Для Cargo:

void Deserialize(ObjectInputStream f) {  
 try {  
 Cargo base = (Cargo) f.readObject();  
 this.cx = base.cx;  
 this.cy = base.cy;  
 this.w = base.w;  
 this.h = base.h;  
 this.ImInd = base.ImInd;  
 this.isMove = base.isMove;  
 this.color = base.color;  
  
 this.W = base.W;  
 this.H = base.H;  
 this.t = base.t;  
 } catch (IOException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
}

Для PassCar:

void Deserialize(ObjectInputStream f)  
{  
 try {  
 PassCar base = (PassCar) f.readObject();  
 this.cx = base.cx;  
 this.cy = base.cy;  
 this.w = base.w;  
 this.h = base.h;  
 this.ImInd = base.ImInd;  
 this.isMove = base.isMove;  
 this.color = base.color;  
 } catch (IOException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
}

Для работы с новым функционалом был обновлён слушатель нажатий на клавиши:

public void keyTyped(KeyEvent e) {  
 switch (e.getKeyChar()) {  
 case ' ': {  
 Check.*isMove* = !Check.*isMove*;  
 break;  
 }  
 case 't': {  
 h.SaveText("Cars.txt");  
 break;  
 }  
 case 'r': {  
 h.LoadText("Cars.txt");  
 break;  
 }  
 case 'b': {  
 h.saveBin("BinCars.txt");  
 break;  
 }  
 case 'v': {  
 h.loadBin("BinCars.txt");  
 break;  
 }  
 case 's': {  
 h.Serialize("SerCar.dat");  
 break;  
 }  
 case 'a': {  
 h.Deserialize("SerCar.dat");  
 break;  
 }  
 case 'x': {  
 h.SaveXML("XMLCars.xml");  
 break;  
 }  
 case 'z': {  
 h.LoadXML("XMLCars.xml");  
 break;  
 }  
 }  
}

**Вывод**

В ходе выполнения лабораторной работы были практически закреплены навыки сохранения состояния объектов.

Также была создана программа моделирования создания, удаления, движения, сохранения и загрузки различными методами объектов.