Санкт-Петербургский Национальный Исследовательский Университет

Информационных Технологий, Механики и Оптики

МФКТиУ, ФПИиКТ, СППО

Лабораторная работа №6

по дисциплине

«Программирование»

Вариант - 31121

Выполнил:

Анищенко Анатолий Алексеевич

Группа: PЗ112

Санкт-Петербург

2019 г.

**Задание:** разделить программу из лабораторной работы №5 на клиентский и серверный модули. Серверный модуль должен осуществлять выполнение команд по управлению коллекцией. Клиентский модуль должен в интерактивном режиме считывать команды, передавать их для выполнения на сервер и выводить результаты выполнения. Команда import должна использовать файл из файловой системы клиента (содержимое файла передается на сервер), load и save - сервера.

Хранящиеся в коллекции объекты должны иметь следующие характеристики:

* имя, название или аналогичный текстовый идентификатор;
* размер или аналогичный числовой параметр;
* характеристику, определяющую местоположение объекта на плоскости/в пространстве;
* время/дату рождения/создания объекта.

Если аналогичные характеристики уже есть, добавлять их не нужно.

Необходимо выполнить следующие требования:

* Коллекцию из ЛР №5 заменить на ее потокобезопасный аналог.
* Операции обработки объектов коллекции должны быть реализованы с помощью Stream API с использованием лямбда-выражений.
* Объекты между клиентом и сервером должны передаваться в сериализованном виде.
* Объекты в коллекции, передаваемой клиенту, должны быть отсортированы по названию.
* Получив запрос, сервер должен создавать отдельный поток, который должен формировать и отправлять ответ клиенту.
* Клиент должен корректно обрабатывать временную недоступность сервера.
* Обмен данными между клиентом и сервером должен осуществляться по протоколу TCP.
* На стороне сервера должен использоваться потоки ввода-вывода а на стороне клиента - сетевой канал.

Отчёт по работе должен содержать:

* Текст задания.
* Диаграмма классов разработанной программы (как клиентского, так и серверного приложения).
* Исходный код программы.
* Выводы по работе.

**Код программы:**

<https://github.com/AnatoliiAnishchenko/ITMO/tree/master/Programming/Lab6>

**Client.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165 | **package** client;  **import** **commands.CommandDescriptor**;  **import** **moominClasses.Moomin**;  **import** **server.Response**;  **import** **java.io.\***;  **import** **java.net.InetSocketAddress**;  **import** **java.net.Socket**;  **import** **java.nio.ByteBuffer**;  **import** **java.nio.channels.SocketChannel**;  **import** **java.util.Scanner**;  **import** **java.util.Vector**;  **public** **class** **Client** {  **private** SocketChannel socket;  **private** Scanner scanner;  **private** **boolean** working;  **private** Vector<Moomin> moomins;  **private** **final** **long** WAITING\_TIME = **10000**;  **public** **Client**(String serverAddress, **int** port) **throws** IOException {  //Set working flag  working = **true**;  //Create socket  socket = SocketChannel.open(**new** InetSocketAddress(serverAddress, port));  scanner = **new** Scanner(System.in);  moomins = **new** Vector<>();  doCommand("connect");  }  **private** **void** **work**() **throws** IOException {  **if** (working) {  System.out.println("Client is ready to work.");  } **else** {  System.out.println("Client can't work.");  }  **while** (working){  System.out.println();  System.out.print("Your command: ");  String stringIn = scanner.nextLine().trim();  System.out.println();  doCommand(stringIn);  }  System.out.println("Session end.");  }  **private** **byte**[] **createRequest**(String description) **throws** IOException {  **byte**[] sending;  CommandDescriptor command = **new** CommandDescriptor(description);  **switch** (command.getNAME()) {  **case** "exit":  **if** (command.getARGS\_COUNT() == **0**) {  working = **false**;  }  **break**;  **case** "import":  **if** (command.getARGS\_COUNT() == **1**) {  **char**[] buf = **new** **char**[**1024**];  **try** (FileReader fr = **new** FileReader(command.getArguments())) {  fr.read(buf);  String json = String.valueOf(buf);  command.setArguments(json);  } **catch** (FileNotFoundException e){  System.err.println("File not found!");  command = **null**;  }  }  **break**;  **case** "my":  moomins.forEach(System.out::println);  **if** (moomins.size() == **0**) {  System.out.println("Collection is empty.");  }  command = **null**;  **break**;  }  **if** (command == **null**) {  **return** **null**;  }  **try** (ByteArrayOutputStream baos = **new** ByteArrayOutputStream();  ObjectOutputStream oos = **new** ObjectOutputStream(baos)){  oos.writeObject(command);  oos.flush();  sending = baos.toByteArray();  **return** sending;  } **catch** (IOException e) {  **throw** e;  }  }  **public** **void** **doCommand**(String command) **throws** IOException {  **byte**[] byteRequest = createRequest(command);  //If we need to send the command  **if** (byteRequest != **null**) {  ByteBuffer request = ByteBuffer.wrap(byteRequest);  **while**(request.hasRemaining()) {  socket.write(request);  }  ByteBuffer respBuf = ByteBuffer.allocate(**4096**);  socket.read(respBuf);  **byte**[] byteResponse = respBuf.array();  **try** (ByteArrayInputStream bais = **new** ByteArrayInputStream(byteResponse);  ObjectInputStream ois = **new** ObjectInputStream(bais)) {  Response response = (Response) ois.readObject();  System.out.println(response.getDoings());  moomins = response.getMoomins();  } **catch** (IOException | ClassNotFoundException e) {  e.printStackTrace();  }  }  }  **public** **static** **void** **main**(String[] args) {  //We need to get server port and address  String address;  **int** port;  //By arguments  **if** (args.length == **2**) {  address = args[**0**];  port = Integer.valueOf(args[**1**]);  } **else** {  //By input stream  Scanner scanner = **new** Scanner(System.in);  System.out.print("Address: ");  address=scanner.nextLine();  System.out.print("Port: ");  port=scanner.nextInt();  }  **try** {  //Creating the client.Client  Client client = **new** Client(address, port);  System.out.println("Welcome!!!");  //Start working  client.work();  } **catch** (Exception e) {  System.err.println("Something gone wrong. Please DEBUG!!!");  e.printStackTrace();  }  }  } |

**Server.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62 | **package** server;  **import** **commands.FileHandler**;  **import** **mainClasses.MoominManager**;  **import** **moominClasses.Moomin**;  **import** **java.io.IOException**;  **import** **java.net.InetAddress**;  **import** **java.net.ServerSocket**;  **import** **java.net.Socket**;  **import** **java.util.Scanner**;  **import** **java.util.Vector**;  **public** **class** **Server** {  **private** Vector<Moomin> moomins;  **private** ServerSocket serverSocket;  **public** **Server**(**int** port) **throws** IOException {  serverSocket = **new** ServerSocket(port, **0**, InetAddress.getLocalHost());  MoominManager manager = **new** MoominManager();  **if** (manager.getMoomins().addAll(FileHandler.readFile("save.json"))) {  System.out.println("Collection changed.");  } **else** {  System.out.println("Collection didn't changed.");  }  moomins = manager.getMoomins();  System.out.println("Server started");  System.out.println("IP: " + serverSocket.getLocalSocketAddress());  }  **private** **void** **listen**() **throws** IOException {  **while** (**true**) {  Socket socket = serverSocket.accept();  ClientThread clientThread = **new** ClientThread(socket, moomins);  //clientThread.start();  clientThread.run();  }  }  **public** **static** **void** **main**(String[] args) {  **int** port = **1234**;  //By arguments  **if** (args.length > **0**) {  port = Integer.valueOf(args[**0**]);  } **else** {  //By input stream  Scanner scanner = **new** Scanner(System.in);  System.out.print("Port: ");  port = scanner.nextInt();  }  **try** {  Server server = **new** Server(port);  server.listen();  } **catch** (IOException e) {  e.printStackTrace();  }  }  } |

**ClientThread.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174 | **package** server;  **import** **commands.Command**;  **import** **commands.CommandDescriptor**;  **import** **commands.CommandsManager**;  **import** **commands.FileHandler**;  **import** **mainClasses.MoominManager**;  **import** **moominClasses.Moomin**;  **import** **java.io.\***;  **import** **java.net.Socket**;  **import** **java.util.Vector**;  **public** **class** **ClientThread** **extends** Thread {  **private** **static** CommandsManager manager;  **private** **final** MoominManager moominManager;  **private** Vector<Moomin> moomins;  **private** Socket socket;  **private** **boolean** isActive;  **public** **ClientThread**(Socket socket, Vector<Moomin> moomins) {  moominManager = **new** MoominManager();  moominManager.setMoomins(moomins);  isActive = **true**;  manager = moominManager.getCommandsManager();  manager.addCommand(  **new** **Command**("connect", **0**) {  **@Override**  **public** **void** **execute**() {  manager.println("Connected");  }  **@Override**  **public** **void** **describe**() {  manager.println("Command to check connection.");  }  },  **new** **Command**("load", **1**) {  **@Override**  **public** **void** **execute**() {  **if** (moominManager.getMoomins().addAll(FileHandler.readFile(getArguments())))  manager.println("Коллекция изменена");  **else** manager.println("Коллекция не изменилась");  }  **@Override**  **public** **void** **describe**() {  manager.println("Дополняет коллекцию элементами из файла с сервера.");  }  },  **new** **Command**("wait", **0**) {  **@Override**  **public** **void** **execute**() {  **long** start = System.currentTimeMillis();  **try** {  Thread.sleep(**100**);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  manager.println(Thread.currentThread().getId() + " wait for " + (System.currentTimeMillis() - start));  }  **@Override**  **public** **void** **describe**() {  manager.println("Make a delay.");  }  },  **new** **Command**("import", -**1**) {  **@Override**  **public** **void** **execute**() {  **if** (moominManager.getMoomins().addAll(moominManager.createMoomin(getArguments())))  manager.println("Коллекция изменена");  **else** manager.println("Коллекция не изменилась");  }  **@Override**  **public** **void** **describe**() {  manager.println("Дополняет коллекцию элементами из файла клиента.");  }  },  **new** **Command**("update",**0**) {  **@Override**  **public** **void** **execute**() {  manager.println("Collection was updated.");  }  **@Override**  **public** **void** **describe**() {  manager.println("Send collection to the client.");  }  },  **new** **Command**("exit",**0**) {  **@Override**  **public** **void** **execute**() {  manager.println("Good bye!");  }  **@Override**  **public** **void** **describe**() {  manager.println("Disconnect user from server.");  }  });  **this**.socket = socket;  **this**.moomins = moomins;  }  **public** **void** **run**() {  **while**(isActive) {  **byte**[] request = **new** **byte**[**4096**];  **try** {  InputStream ins = socket.getInputStream();  **if** (ins.read(request) < **0**) {  **continue**;  }  } **catch** (IOException e) {  e.printStackTrace();  }  answer(request);  }  }  **private** **void** **answer**(**byte**[] request) {  CommandDescriptor command;  **try** (ByteArrayInputStream bais = **new** ByteArrayInputStream(request);  ObjectInputStream ois = **new** ObjectInputStream(bais);  ByteArrayOutputStream baos = **new** ByteArrayOutputStream();  ObjectOutputStream oos = **new** ObjectOutputStream(baos);  ByteArrayOutputStream bao = **new** ByteArrayOutputStream();  PrintStream printStream = **new** PrintStream(bao)) {  command = (CommandDescriptor) ois.readObject();  manager.setPrintStream(printStream);  **if** (command.getNAME().equals("exit")) {  isActive = **false**;  }  manager.doCommand(command);  printStream.flush();  String doings = **new** String(bao.toByteArray()).trim();  **synchronized** (System.out) {  System.out.println();  System.out.println("Client's command: " + command.getNAME() + " " + command.getArguments());  System.out.println();  System.out.println(doings);  }  Response response = **new** Response(doings, moominManager.getMoomins());  oos.writeObject(response);  oos.flush();  **try** {  OutputStream outs = socket.getOutputStream();  outs.write(baos.toByteArray());  } **catch** (IOException e) {  e.printStackTrace();  }  } **catch** (IOException | ClassNotFoundException e) {  e.printStackTrace();  }  }  **public** **static** CommandsManager **getManager**() {  **return** manager;  }  } |

**Выводы:** в процессе выполнения лабораторной работы были получены навыки работы с дженериками и потоками ввода/вывода.