```java
package server;

import server.commands.*;
import server.managers.CollectionManager;
import java.io.*;
import java.net.Socket;
import java.net.SocketException;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Objects;

/**
 * Класс {@code Connection} представляет объект подключённому к серверу, который манипулирует {@link CollectionManager}.
 * @author Артемий Кульбако
 * @version 1.4
 * @since 15.05.19
 */
public class Connection extends Thread {

    private Socket incoming;
    private int masterID;
    private HashMap<String, AbstractCommand> availableCommands = new HashMap<>();

    /**
     * @param incoming активное соединение с клиентской программой.
     * Команды, доступные клиенту, являются объектами {@link AbstractCommand}, хранящимися в
     * {@code HashMap <String, AbstractCommand> availableCommands}.
     */
    Connection(Socket incoming) {
        this.incoming = incoming;
        availableCommands.put("login", new LoginCommand(this));
        availableCommands.put("register", new RegisterCommand());
        availableCommands.put("delete_account", new DeleteAccountCommand(this));
        availableCommands.put("man", new ManCommand(availableCommands));
        availableCommands.put("help", new HelpCommand(availableCommands));
    }


    /**
     * Поддерживает активное соединение с клиентом.
     */
    @Override
    public void run() {
        try (ObjectInputStream getFromClient = new ObjectInputStream(incoming.getInputStream());
             ObjectOutputStream sendToClient = new ObjectOutputStream(incoming.getOutputStream())) {
            sendToClient.writeObject("Соединение установлено.\nВы можете вводить команды.");
            AbstractCommand error = new AbstractCommand() {
                @Override
                public synchronized String execute() {
                    return "Неизвестная команда. Введите 'help' для получения списка команд.";
                }
            };
            while (true) {
                try {
                    String requestFromClient = (String) getFromClient.readObject();
                    System.out.print("Получено [" + requestFromClient + "] от " + incoming + ". ");
                    String[] parsedCommand = requestFromClient.trim().split(" ");
                    if (parsedCommand.length == 1)
                        sendToClient.writeObject(availableCommands.getOrDefault(parsedCommand[0], error).execute());
                    else sendToClient.writeObject(availableCommands.getOrDefault(parsedCommand[0], error)
                            .execute(Arrays.copyOfRange(parsedCommand, 1, parsedCommand.length)));
                    System.out.println("Ответ успешно отправлен.");
                } catch (SocketException e) {
```

```
64                      System.out.println(incoming + " отключился от сервера."); //Windows
65                      break;
66                  }
67              }
68          } catch (IOException | ClassNotFoundException ex) {
69              System.err.println(incoming + " отключился от сервера."); //Unix
70          }
71      }
72
73      public int getMasterID() {
74          return masterID;
75      }
76
77      public void setMasterID(int masterID) {
78          this.masterID = masterID;
79      }
80
81      public Socket getSocket() {
82          return incoming;
83      }
84
85      public HashMap<String, AbstractCommand> getCommands() {
86          return availableCommands;
87      }
88
89      public void setCommands(HashMap<String, AbstractCommand> availableCommands) {
90          this.availableCommands = availableCommands;
91      }
92
93      @Override
94      public boolean equals(Object o) {
95          if (this == o) return true;
96          if (!(o instanceof Connection)) return false;
97          Connection that = (Connection) o;
98          return masterID == that.masterID &&
99                  Objects.equals(incoming, that.incoming) &&
100                 Objects.equals(availableCommands, that.availableCommands);
101     }
102
103     @Override
104     public int hashCode() {
105         return Objects.hash(incoming, masterID, availableCommands);
106     }
107
108     @Override
109     public String toString() {
110         return "Connection{" +
111                 "incoming=" + incoming +
112                 ", masterID=" + masterID +
113                 ", availableCommands=" + availableCommands +
114                 '}';
115     }
116 }
117
118 //TODO имена с пробелами
```

```java
package server;

import server.managers.CollectionManager;
import server.managers.DatabaseManager;
import java.io.IOException;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;

public class ServerSide {
    /**
    * Точка входа в программу. Управляет подключением к клиентам и созданием потоков для каждого из них.
     * @param args массив по умолчанию в основном методе. Не используется здесь.
     */
    public static void main(String[] args) {
        DatabaseManager.getInstance(); //Загрузить БД
        CollectionManager.getInstance(); //Загрузить коллекцию
        try (ServerSocket server = new ServerSocket(8800)) {
            System.out.print("Сервер начал слушать клиентов: " + "Порт " + server.getLocalPort() +
                    " / Адрес " + InetAddress.getLocalHost() + ".\nОжидаем подключения клиентов");
            Thread pointer = new Thread(() -> {
                while (!Thread.currentThread().isInterrupted()) {
                    System.out.print(" .");
                    try {
                        Thread.sleep(2500);
                    } catch (InterruptedException e) {
                        Thread.currentThread().interrupt();
                    }
                }
            });
            pointer.setDaemon(true);
            pointer.start();
            while (true) {
                Socket incoming = server.accept();
                pointer.interrupt();
                System.out.println(incoming + " подключился к серверу.");
                new Connection(incoming).start();
            }
        } catch (IOException ex) {
            ex.printStackTrace();
            System.exit(1);
        }
    }
}
```