МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

ОТЧЕТ

по лабораторной работе №7
в рамках дисциплины «Программирование»

Выполнил: А. С. Медведев
Студент группы R3237

Санкт-Петербург

2021

Цель работы:

Доработать программу из лабораторной работы №6 следующим образом:

- Организовать возможность регистрации и авторизации пользователей. У пользователя есть возможность указать пароль.
- Пароли при хранении хэшировать алгоритмом MD5
- Запретить выполнение команд не авторизованным пользователям.
- При хранении объектов сохранять информацию о пользователе, который создал этот объект.
- Пользователи должны иметь возможность просмотра всех объектов коллекции, но модифицировать могут только принадлежащие им.
- Для идентификации пользователя отправлять логин и пароль с каждым запросом.

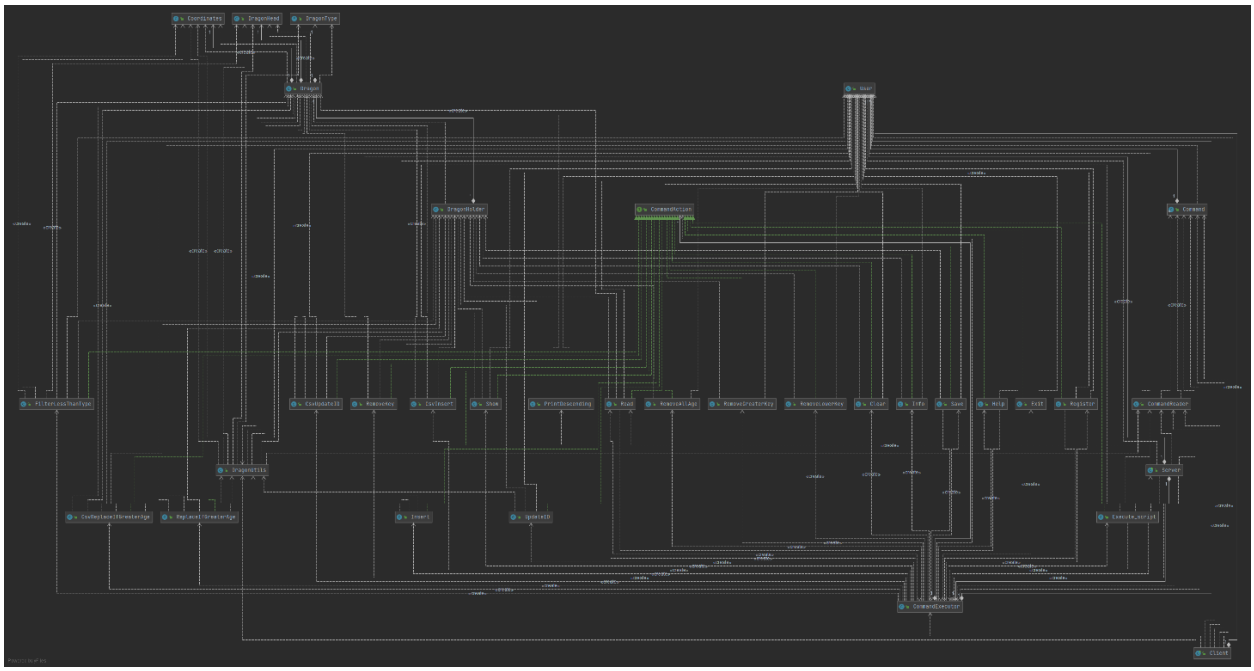
Необходимо реализовать многопоточную обработку запросов.

- Для многопоточного чтения запросов использовать создание нового потока (java.lang.Thread)
- Для многопотчной обработки полученного запроса использовать ForkJoinPool
- Для многопоточной отправки ответа использовать Cached thread pool
- Для синхронизации доступа к коллекции использовать синхронизацию чтения и записи с помощью synchronized

Выполнение работы:

Исходный код программы представлен в Git-репозитории oaleksander/Lab7 (github.com).

Диаграмма классов:



Выводы:

В ходе данной лабораторной работы было реализовано приложение для управление коллекцией объектов с разделением на клиентский и серверный модули, с поддержкой многопоточной обработки запросов и отправки ответов.

Файл Client.java

```java
package com.company;

import com.company.storables.DragonUtils;
import com.company.ui.CommandExecutor;
import com.company.ui.CommandReader;
import com.company.ui.User;

import java.io.*;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetSocketAddress;
import java.util.Arrays;
import java.util.Date;

/**
 * Main client class
 * @see Server
 */
public class Client {

    private static final String[] dragonCommands = {"insert", "update", "replace_if_greater_age"};
    private static final InetSocketAddress address = new InetSocketAddress("localhost", 3333);
    private static User user = new User("notYetLoggedInUser","");
    private static ByteArrayOutputStream byteArrayOutputStream;
    private static ObjectOutput objectOutput;
    private static DatagramSocket datagramSocket = null;

    /**
     * Main client function
     *
     * @param args filename
     * @see CommandExecutor
     * @see Server
     */
    public static void main(String[] args) {
        BufferedReader cin = new BufferedReader(new InputStreamReader(System.in));

        String username;
        String password;
        try {
            System.out.println("Username: ");
            username = cin.readLine();
            System.out.println("Password: ");
            password = cin.readLine();
            user = new User(username,password);
        } catch (IOException e) {
            e.printStackTrace();
            return;
        }

        CommandReader commandReader = new CommandReader(cin);
        byteArrayOutputStream = new ByteArrayOutputStream();

        System.out.println("Logging in...");
        try {
            datagramSocket = new DatagramSocket();
            datagramSocket.setBroadcast(true);
            datagramSocket.setSoTimeout(1500);
            byteArrayOutputStream.reset();
            objectOutput = new ObjectOutputStream(byteArrayOutputStream);
            sendCommand(new CommandReader.Command(user,"register", username+" "+password));
            System.out.print(getResponse());
        } catch (IOException e) {
            System.out.println("Can't connect to server.");
            System.exit(-1);
        }

        System.out.println("Welcome to interactive Dragon Hashtable server. To get help, enter \"help\"
.");
        //noinspection InfiniteLoopStatement
        while (true) {
            CommandReader.Command command = commandReader.readCommandFromBufferedReader(user);
            try {
                byteArrayOutputStream.reset();
```

Файл Client.java

```
74                  objectOutput = new ObjectOutputStream(byteArrayOutputStream);
75                  if (sendCommand(command))
76                      System.out.print(getResponse());
77              } catch (IOException e) {
78                  System.err.print(e.getMessage());
79                  System.err.println(". Trying again...");
80                  try {
81                      if (sendCommand(command))
82                          System.out.print(getResponse());
83                  } catch (IOException e2) {
84                      System.err.println("Failed. Please try again later.");
85                  }
86              }
87          }
88      }
89
90      /**
91       * Gets a command, verifies it and sends it to server (or exits the program if that is the case)
92       * @param userCommand Command input
93       * @return true if Command was verified and sent
94       * @throws IOException If problem occurred while sending to server
95       */
96      private static boolean sendCommand(CommandReader.Command userCommand) throws IOException {
97          if (userCommand.commandString.equals("exit")) {
98              System.out.println("Exiting...");
99              byteArrayOutputStream.close();
100             datagramSocket.close();
101             System.exit(0);
102         }
103         if (userCommand.commandString.equals("register")) {
104             String[] arguments = userCommand.argumentString.split(" ",2);
105             if(arguments.length == 2)
106                 user = new User(arguments[0],arguments[1]);
107             else if(arguments.length == 1)
108                 user = new User(arguments[0]);
109         }
110         if (Arrays.stream(CommandExecutor.userCommands).parallel().noneMatch(command -> command.
    getLabel().equals(userCommand.commandString))) {
111             System.err.println("Unknown command \"" + userCommand.commandString + "\". try \"help\" for
     list of commands");
112             return false;
113         }
114         if (!Arrays.stream(CommandExecutor.userCommands).parallel().filter(command -> command.getLabel
    ().equals(userCommand.commandString)).findAny().get().getArgumentLabel().equals("")
115                 && userCommand.argumentString.isBlank()) {
116             System.err.println("Please specify command argument.");
117             return false;
118         }
119         Date date = new Date();
120         String dragonString;
121         if (Arrays.stream(dragonCommands).parallel().anyMatch(command -> command.equals(userCommand.
    commandString))) {
122             userCommand.commandString += "_csv";
123             if (userCommand.commandString.equals("update_id_csv")) {
124                 try {
125                     dragonString = DragonUtils.inputDragonFromConsole(user,Long.parseLong(userCommand.
    argumentString), date).toCsvString();
126                     userCommand.argumentString = dragonString;
127                 } catch (NumberFormatException e) {
128                     System.err.println("Can't parse dragon ID from " + userCommand.argumentString + "."
    );
129                 }
130             } else {
131                 dragonString = DragonUtils.inputDragonFromConsole(user,-1, date).toCsvString(); //-1
    value will be replaced with new ID
132                 userCommand.argumentString = userCommand.argumentString + "," + dragonString;
133             }
134
135         }
136         objectOutput.writeObject(userCommand);
137         byte[] data = byteArrayOutputStream.toByteArray();
138         datagramSocket.send(new DatagramPacket(data, data.length, address));
139         return true;
140     }
```

Файл Client.java

```
141
142     /**
143      * Gets a message from a server
144      * @return message
145      * @throws IOException If problem occurred while receiving from server
146      */
147     private static String getResponse() throws IOException {
148         byte[] data = new byte[65536];
149         DatagramPacket receivePacket = new DatagramPacket(data, data.length);
150         datagramSocket.receive(receivePacket);
151         return new String(receivePacket.getData());
152     }
153 }
154
```

Файл Server.java

```java
package com.company;

import com.company.commands.Read;
import com.company.ui.CommandExecutor;
import com.company.ui.CommandReader;
import com.company.ui.User;

import java.io.*;
import java.net.InetSocketAddress;
import java.net.SocketAddress;
import java.nio.ByteBuffer;
import java.nio.channels.DatagramChannel;
import java.nio.channels.SelectionKey;
import java.nio.channels.Selector;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.concurrent.*;

/**
 * Main server class
 * @see Client
 */
public class Server {

    private static final ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
    private static final PrintStream serverResponseStream = new PrintStream(outputStream);
    private static final CommandExecutor localExecutor = new CommandExecutor(CommandExecutor.allCommands, System.out);
    private static final CommandExecutor userExecutor = new CommandExecutor(CommandExecutor.userCommands, serverResponseStream);
    public static final User internalUser = new User("admin","admin");

    /**
     * Main server function
     *
     * @param args filename
     * @see CommandExecutor
     * @see Client
     */
    public static void main(String[] args) {
        System.out.println("Starting Server...");
        System.out.println("Reading collection from file...");
        readCollectionFromFile(args);
        BufferedReader localInput = new BufferedReader(new InputStreamReader(System.in));
        Selector selector = null;
        ByteBuffer byteBuffer = ByteBuffer.allocate(65536);
        try {
            selector = Selector.open();
            selector.wakeup();
            DatagramChannel datagramChannel = DatagramChannel.open();
            datagramChannel.bind(new InetSocketAddress("localhost", 3333));
            datagramChannel.configureBlocking(false);
            datagramChannel.register(selector, SelectionKey.OP_READ);
        } catch (IOException e) {
            System.err.println("Failed to start server.");
            e.printStackTrace();
            System.exit(-1);
        }
        ExecutorService responseExecutorService = Executors.newCachedThreadPool();
        System.out.println("Server is active.");
        try {
            while (true) {
                try {
                    String line;
                    if (localInput.ready())
                        if ((line = localInput.readLine()) != null)
                            localExecutor.executeCommand(CommandReader.readCommandFromString(internalUser,line));
                    selector.select(1500);
                    selector.selectedKeys().stream().parallel().forEach(selectionKey ->
                    {
                        try {
                            if (selectionKey.isValid())
                                if (selectionKey.isReadable())
```

Файл Server.java

```
72                                      responseExecutorService.submit(processCommand(selectionKey));
73                          } catch (IOException e) {
74                              System.err.println("IOException " + e.getMessage());
75                          } catch (NullPointerException ignored) {}
76                      });
77                      Executors.newCachedThreadPool().submit(() -> {
78                      });
79                  } catch (IOException e) {
80                      System.err.println("IOException " + e.getMessage());
81                  } catch (Exception e) {
82                      System.err.println("Unexpected error: " + e.getMessage());
83                      e.printStackTrace();
84                      System.exit(-1);
85                  }
86              }
87          } finally {
88              responseExecutorService.shutdown();
89          }
90      }
91
92      /**
93       * Processes a command from client
94       * @param key Selection key
95       * @throws IOException If problem occurred while sending data to client
96       * @return Server response
97       */
98      private static Runnable processCommand(SelectionKey key)
99              throws IOException {
100         DatagramChannel client = (DatagramChannel) key.channel();
101         ByteBuffer buffer = ByteBuffer.allocate(65536);
102         SocketAddress address = client.receive(buffer);
103         if (address == null) throw new NullPointerException("Address is null");
104         byte[] data = buffer.array();
105         buffer.clear();
106         ObjectInputStream iStream = new ObjectInputStream(new ByteArrayInputStream(data));
107         byte[] response;
108         try {
109             CommandReader.Command command = (CommandReader.Command) iStream.readObject();
110             System.out.println("[" + new SimpleDateFormat("HH:mm:ss").format(new Date()) + "] " +
    command.user.getUsername() + address.toString() + ": " + command.toString() + ".");
111             //Отправляем данные клиенту
112             try {
113                 userExecutor.executeCommand(command);
114                 response = outputStream.toByteArray();
115                 outputStream.reset();
116             } catch (IllegalArgumentException e) {
117                 response = e.getMessage().getBytes();
118             }
119         } catch (StreamCorruptedException | ClassNotFoundException e) {
120             System.err.println(e.getMessage());
121             e.printStackTrace();
122             response = "Server received invalid packet. Please try again.".getBytes();
123         }
124         byte[] finalResponse = response;
125         return ()-> {
126             try {
127                 client.send(ByteBuffer.wrap(finalResponse), address);
128             } catch (IOException e) {
129                 System.err.println("IOException " + e.getMessage());
130             }
131         };
132     }
133
134     /**
135      * Reads a collection from file
136      * @see Read
137      * @param args Filename
138      */
139     private static void readCollectionFromFile(String[] args) {
140         if (args.length == 0)
141             System.out.println("Input filename not specified by command line argument. Skipping...");
142         else {
143             try {
144                 CommandExecutor.setFile(args[0]);
```

Файл Server.java

```
145                 try {
146                     System.out.println(new Read().execute(internalUser,""));
147                 } catch (Exception e) {
148                     System.out.println(e.getMessage() + " Skipping...");
149                 }
150             } catch (NullPointerException e) {
151                 System.out.println("Input filename is empty. Skipping...");
152             }
153         }
154     }
155 }
156
```

Файл User.java

```java
package com.company.ui;

import java.io.Serializable;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Arrays;
import java.util.Objects;

/**
 * Class that is needed to store users
 */
public class User implements Serializable {

    private final String username;
    private final byte[] passwordMD5;

    /**
     * Username-only constructor
     * @param username username
     */
    public User(String username){
        this(username,"");
    }

    /**
     * Default constructor
     * @param username username
     * @param password password
     */
    public User(String username, String password) {
        this.username = username;
        byte[] finalPasswordMD5;
        try {
            finalPasswordMD5 = MessageDigest.getInstance("MD5").digest(password.getBytes());
        } catch (NoSuchAlgorithmException e) {
            finalPasswordMD5 = new byte[0];
            e.printStackTrace();
        }
        this.passwordMD5 = finalPasswordMD5;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof User)) return false;
        User user = (User) o;
        return username.equals(user.username) && Arrays.equals(passwordMD5, user.passwordMD5);
    }

    @Override
    public int hashCode() {
        int result = Objects.hash(username);
        result = 31 * result + Arrays.hashCode(passwordMD5);
        return result;
    }

    /**
     * Username getter
     * @return usernamg
     */
    public String getUsername() {
        return username;
    }

    /**
     * Password getter
     * @return password in MD5 format
     */
    public byte[] getPasswordMD5() {
        return passwordMD5;
    }

    @Override
    public String toString() {
```

Файл User.java

```
75          return "User{" +
76                  "username='" + username + '\'' +
77                  ", passwordMD5=" + Arrays.toString(passwordMD5) +
78                  '}';
79      }
80  }
81
```

Файл CommandReader.java

```java
package com.company.ui;

import com.company.Server;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.Serializable;

/**
 * Class designed to get commands from buffered readers and strings
 *
 * @see CommandExecutor
 */
public class CommandReader {

    BufferedReader bufferedReader;

    /**
     * Command reader constructor
     *
     * @param bufferedReader buffered reader to get commands from
     */
    public CommandReader(BufferedReader bufferedReader) {
        this.bufferedReader = bufferedReader;
    }

    /**
     * Get an array of strings from System.in (separated by spaces)
     *
     * @return Array of strings
     */
    public static String[] getStringsFromTerminal() {
        return new CommandReader(new BufferedReader(new InputStreamReader(System.in))).
getStringFromBufferedReader().split(" ");
    }

    /**
     * Get a command from string
     *
     * @param user User
     * @param singleString string to parse from
     * @return Command
     */
    public static Command readCommandFromString(User user, String singleString) {
        return (readCommandFromString(user,singleString.split(" ", 2)));
    }

    /**
     * Get a command from strings
     *
     * @param user User
     * @param input strings to parse from
     * @return Command
     */
    public static Command readCommandFromString(User user, String[] input) {
        if (input.length != 0) {
            input[0] = input[0].toLowerCase();
            if (input.length > 1)
                return new Command(user,input[0], input[1]);
            else
                return new Command(user,input[0]);
        } else return new Command(user);
    }

    /**
     * Gets a string from buffered reader line
     *
     * @return Received string
     */
    public String getStringFromBufferedReader() {
        try {
            return bufferedReader.readLine();
        } catch (IOException e) {
```

Файл CommandReader.java

```
74              System.out.println("Error: " + e.getMessage() + ".");
75              return "";
76          }
77      }
78
79      /**
80       * Get a command from Buffered Reader
81       *
82       * @param user User
83       * @return Command
84       */
85      public Command readCommandFromBufferedReader(User user) {
86          return readCommandFromString(user,getStringFromBufferedReader());
87      }
88
89      /**
90       * User command class
91       */
92      public static class Command implements Serializable {
93          public User user;
94          public String commandString = "";
95          public String argumentString = "";
96
97          /**
98           * User command constructor with argument
99           *
100          * @param user User
101          * @param commandString  Command
102          * @param argumentString Argument
103          */
104         public Command(User user, String commandString, String argumentString) {
105             this.user = user;
106             this.commandString = commandString;
107             this.argumentString = argumentString;
108         }
109
110         /**
111          * User command constructor without argument
112          *
113          * @param user User
114          * @param CommandString Command
115          */
116         public Command(User user, String CommandString) {
117             this.user = user;
118             this.commandString = CommandString;
119         }
120
121         /**
122          * Empty command constructor
123          *
124          * @param user User
125          */
126         public Command(User user) {
127             this.user = user;
128         }
129
130         @Override
131         public String toString() {
132             return "Command{" + commandString + (argumentString.isBlank() ? "" : " ") + argumentString
    + "}";
133         }
134     }
135 }
136
```

Файл CommandExecutor.java

```java
package com.company.ui;

import com.company.commands.*;

import java.io.*;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;
import java.util.concurrent.atomic.AtomicReference;

/**
 * Main command execution runnable
 */
public class CommandExecutor {

    public static final List<User> registeredUsers = Collections.synchronizedList(new ArrayList<>());

    /**
     * All possible commands
     */
    public static final CommandAction[] allCommands = {
            new Help(),
            new Info(),
            new Show(),
            new Insert(),
            new UpdateID(),
            new RemoveKey(),
            new Clear(),
            new Save(),
            new Execute_script(),
            new Exit(),
            new ReplaceIfGreaterAge(),
            new RemoveGreaterKey(),
            new RemoveLowerKey(),
            new RemoveAllAge(),
            new FilterLessThanType(),
            new PrintDescending(),
            new Read(),
            new CsvInsert(),
            new CsvUpdateID(),
            new CsvReplaceIfGreaterAge(),
            new Register()
    };
    /**
     * Commands that user can use
     */
    public static final CommandAction[] userCommands = {
            new Help(),
            new Info(),
            new Show(),
            new Insert(),
            new UpdateID(),
            new RemoveKey(),
            new Clear(),
            new Execute_script(),
            new Exit(),
            new ReplaceIfGreaterAge(),
            new RemoveGreaterKey(),
            new RemoveLowerKey(),
            new RemoveAllAge(),
            new FilterLessThanType(),
            new PrintDescending(),
            new Register()
            //new Read(),
            //new CsvInsert(),
            //new CsvUpdateID(),
            //new CsvReplaceIfGreaterAge()
    };
    /**
     * Programs that execute_script can use
     */
    public static final CommandAction[] scriptCommands = {
            new Help(),
```

Файл CommandExecutor.java

```
            new Info(),
            new Show(),
            //new Insert(),
            //new UpdateID(),
            new RemoveKey(),
            new Clear(),
            new Save(),
            //new Execute_script(),
            new Exit(),
            //new ReplaceIfGreaterAge(),
            new RemoveGreaterKey(),
            new RemoveLowerKey(),
            new RemoveAllAge(),
            new FilterLessThanType(),
            new PrintDescending(),
            //new Read(),
            new CsvInsert(),
            new CsvUpdateID(),
            new CsvReplaceIfGreaterAge()
    };
    private static File file = new File("C:\\Users\\muram\\IdeaProjects\\Lab5\\file.csv");
    private final PrintStream printStream;
    private final CommandAction[] availableCommands;

    /**
     * User runnable constructor
     *
     * @param availableCommands set of available commands
     * @param printStream       PrintStream to output responses to
     */
    public CommandExecutor(CommandAction[] availableCommands, PrintStream printStream) {
        this.availableCommands = availableCommands;
        this.printStream = printStream;
    }

    /**
     * Get file specified by command line argument
     *
     * @return File
     */
    public static File getFile() {
        return file;
    }

    /**
     * Set file to save/read collection from
     *
     * @param fileName File name
     */
    public static void setFile(String fileName) {
        file = new File(fileName);
    }

    /**
     * Execute specified user command
     *
     * @param command User command
     */
    public void executeCommand(CommandReader.Command command) {
        AtomicReference<String> response = new AtomicReference<>("Command gave no response.");
        if(registeredUsers.stream().noneMatch(user -> user.equals(command.user)) && !command.
commandString.equals("register")) {
            response.set("Unauthorized access denied.");
        } else {
            if (Arrays.stream(availableCommands).parallel().noneMatch(availableCommand ->
availableCommand.getLabel().equals(command.commandString))) {
            response.set("Unknown command \"" + command.commandString + "\". try \"help\" for list of
commands");
        } else
            try {
                Arrays.stream(availableCommands).forEach(availableCommand -> {
                    if (command.commandString.equals(availableCommand.getLabel()))
                        response.set(availableCommand.execute(command.user, command.argumentString));
                });
```

Файл CommandExecutor.java

```
146             } catch (IllegalArgumentException e) {
147                 response.set(e.getMessage());
148             } catch (Exception e) {
149                 response.set("Unexpected error: " + e.getMessage() + ". This is a bug!");
150                 e.printStackTrace();
151             }
152         }
153         printStream.println(response.get());
154     }
155 }
156
157
```

Файл Exit.java

```java
package com.company.commands;

import com.company.ui.User;

public class Exit implements CommandAction {

    @Override
    public String getLabel() {
        return "exit";
    }

    public String getDescription() {
        return "Exit the program (without saving).";
    }

    public String execute(User commandedUser, String argument) {
        System.exit(0);
        return "Exited.";
    }
}

```

Файл Help.java

```java
package com.company.commands;

import com.company.ui.CommandExecutor;
import com.company.ui.User;

import java.util.Arrays;

public class Help implements CommandAction {
    String response;

    public String getLabel() {
        return "help";
    }

    public String getDescription() {
        return "Gives the list of available commands.";
    }

    public String execute(User commandedUser, String argument) {
        response = "Available commands:\n";
        Arrays.stream(CommandExecutor.userCommands).forEach(command -> response += command.getLabel() + " " + command.getArgumentLabel() + ": " + command.getDescription() + "\n");
        response += "Collection class members have to be entered line-by-line. Standard types (including primitive types) have to be entered in the same line as the command.";
        return response;
    }
}

```

Файл Info.java

```java
package com.company.commands;

import com.company.storables.DragonHolder;
import com.company.ui.User;

public class Info implements CommandAction {
    public String getLabel() {
        return "info";
    }

    public String getDescription() {
        return "Gives the info about collection.";
    }

    public String execute(User commandedUser, String argument) {
        return "Dragon collection, initialization date: " + DragonHolder.getInitializationDate() + ",
  number of elements: " + DragonHolder.getCollection().size() + ".";
    }
}

```

Файл Read.java

```java
package com.company.commands;

import com.company.storables.Dragon;
import com.company.storables.DragonHolder;
import com.company.ui.CommandExecutor;
import com.company.ui.User;

import java.io.*;
import java.util.Arrays;

public class Read implements CommandAction {

    String response;

    @Override
    public String getLabel() {
        return "read";
    }

    @Override
    public String getDescription() {
        return "Clear collection and read from file (its name is specified by command line argument).";
    }

    @Override
    public String execute(User commandedUser, String argument) {
        response = "";
        File file = CommandExecutor.getFile();
        try {
            BufferedInputStream fileReader = new BufferedInputStream(new FileInputStream(file));
            StringBuilder stringBuilder = new StringBuilder();
            while (fileReader.available() > 0)
                stringBuilder.append((char) fileReader.read());
            DragonHolder.getCollection().clear();
            Arrays.stream(stringBuilder.toString().split("[\\r\\n]+"))
                    .forEach(line -> {
                        if (!line.isBlank())
                            try {
                                String[] splitLine = line.split(",", 2);
                                if (splitLine.length < 2)
                                    throw new IllegalArgumentException("Invalid input string: \"" + line + "\".");
                                try {
                                    DragonHolder.getCollection().put(Integer.parseInt(splitLine[0]), new Dragon(splitLine[1]));
                                } catch (NumberFormatException e) {
                                    throw new IllegalArgumentException("Can't parse key from " + splitLine[0] + ".");
                                }
                            } catch (IllegalArgumentException e) {
                                response += "Error reading from CSV line " + line + ": " + e.getMessage() + ".\n";
                            }
                    });
            fileReader.close();
        } catch (FileNotFoundException e) {
            throw new IllegalArgumentException("Can't find file \"" + file + "\".");
        } catch (SecurityException e) {
            throw new IllegalArgumentException("Can't access file \"" + file + "\".");
        } catch (IOException e) {
            throw new IllegalArgumentException("Error occurred accessing file \"" + file + "\".");
        }
        response += "Read collection from file \"" + file + ".";
        return response;
    }
}

```

Файл Save.java

```java
package com.company.commands;

import com.company.storables.DragonHolder;
import com.company.ui.CommandExecutor;
import com.company.ui.User;

import java.io.*;

public class Save implements CommandAction {

    @Override
    public String getLabel() {
        return "save";
    }

    @Override
    public String getDescription() {
        return "Saves collection to file (its name is specified by command line argument).";
    }

    @Override
    public String execute(User commandedUser, String argument) {
        File file = CommandExecutor.getFile();
        try {
            file.createNewFile();
            OutputStreamWriter fileWriter = new OutputStreamWriter(new FileOutputStream(file));
            DragonHolder.getCollection().forEach((key, value) -> {
                try {
                    fileWriter.write(key + "," + value.toCsvString() + "\n");
                } catch (IOException e) {
                    throw new IllegalArgumentException("Error occurred writing collection to file \"" +
file + "\".");
                }
            });
            fileWriter.close();
        } catch (FileNotFoundException e) {
            throw new IllegalArgumentException("Can't find file \"" + file + "\".");
        } catch (SecurityException e) {
            throw new IllegalArgumentException("Can't access file \"" + file + "\".");
        } catch (IOException e) {
            throw new IllegalArgumentException("Error occurred accessing file \"" + file + "\".");
        }
        return "Saved collection to file \"" + file + ".";
    }
}

```

Файл Show.java

```java
package com.company.commands;

import com.company.storables.Dragon;
import com.company.storables.DragonHolder;
import com.company.ui.User;

import java.util.Comparator;

public class Show implements CommandAction {
    String response;

    public String getLabel() {
        return "show";
    }

    public String getDescription() {
        return "Show all collection elements.";
    }

    public String execute(User commandedUser, String argument) {
        response = "Collection:\n";
        DragonHolder.getCollection().values().stream().sorted(Comparator.comparing(Dragon::getName)).forEach(element -> response += element.toString() + "\n");
        return response.substring(0, response.length() - 1);
    }
}

```

Файл Clear.java

```java
package com.company.commands;

import com.company.storables.DragonHolder;
import com.company.ui.User;

public class Clear implements CommandAction {
    public String getLabel() {
        return "clear";
    }

    public String getDescription() {
        return "Clear the collection.";
    }

    @Override
    public String execute(User commandedUser, String argument) {
        DragonHolder.getCollection().clear();
        return "Collection cleared.";
    }
}

```

Файл Insert.java

```java
package com.company.commands;

import com.company.storables.DragonHolder;
import com.company.storables.DragonUtils;
import com.company.ui.User;

public class Insert implements CommandAction {
    public String getLabel() {
        return "insert";
    }

    public String getArgumentLabel() {
        return "{key} {element}";
    }

    public String getDescription() {
        return "Insert new {element} to collection with a {key}.";
    }

    public String execute(User commandedUser, String argument) {
        if (argument == null || argument.isEmpty())
            throw new IllegalArgumentException("Please specify Dragon key.");
        try {
            DragonHolder.getCollection().put(Integer.parseInt(argument), DragonUtils.
inputNewDragonFromConsole(commandedUser));
        } catch (IllegalArgumentException e) {
            throw new IllegalArgumentException("Illegal key: " + e.getMessage() + ".");
        }
        return "Insert successful.";
    }
}

```

Файл Register.java

```java
package com.company.commands;

import com.company.ui.CommandExecutor;
import com.company.ui.User;

import java.util.Arrays;
import java.util.List;

public class Register implements CommandAction{

    @Override
    public String getLabel() {
        return "register";
    }

    @Override
    public String getArgumentLabel() {
        return "{username} [password]";
    }

    @Override
    public String getDescription() {
        return "Register a new user";
    }

    @Override
    public String execute(User commandedUser, String argument) {
        String[] arguments = argument.split(" ",2);
        User newUser;
        if(arguments.length == 2)
            newUser = new User(arguments[0],arguments[1]);
        else if(arguments.length == 1)
            newUser = new User(arguments[0]);
        else
            return "Failed to register. Did you input valid username and password? Try typing register {username} [password] to try again.";
        if(CommandExecutor.registeredUsers.stream().anyMatch(newUser::equals))
            return "Logged in successfully.";
        if (CommandExecutor.registeredUsers.stream().anyMatch(user -> user.getUsername().equals(newUser.getUsername()) && !Arrays.equals(user.getPasswordMD5(), newUser.getPasswordMD5())))
            return "Wrong password. Further commands will not be accepted. Try typing register {username} [password] to try again.";
        CommandExecutor.registeredUsers.add(newUser);
        return "Registered new user successfully.";
    }
}

```

Файл UpdateID.java

```java
package com.company.commands;

import com.company.storables.DragonHolder;
import com.company.storables.DragonUtils;
import com.company.ui.User;

import java.util.Date;
import java.util.concurrent.atomic.AtomicBoolean;
import java.util.concurrent.atomic.AtomicReference;

public class UpdateID implements CommandAction {
    @Override
    public String getLabel() {
        return "update";
    }

    @Override
    public String getArgumentLabel() {
        return "{id} {element}";
    }

    @Override
    public String getDescription() {
        return "Update {element} of collection by its {id}.";
    }

    @Override
    public String execute(User commandedUser, String argument) {
        long id;
        try {
            id = Long.parseLong(argument);
        } catch (NumberFormatException e) {
            throw new IllegalArgumentException("ID argument \"" + argument + "\" is not an integer.");
        }
        AtomicBoolean found = new AtomicBoolean(false);
        AtomicReference<Integer> dragonKey = new AtomicReference<>();
        AtomicReference<Long> dragonId = new AtomicReference<>();
        AtomicReference<Date> dragonCreationDate = new AtomicReference<>();
        DragonHolder.getCollection().forEach((key, value) -> {
            if (value.getId() == id) {
                dragonKey.set(key);
                dragonId.set(value.getId());
                dragonCreationDate.set(value.getCreationDate());
                found.set(true);
            }
        });
        if (!found.get()) throw new IllegalArgumentException("Dragon with id '" + argument + "' not found");
        else
            if(!DragonHolder.getCollection().get(dragonKey.get()).getOwner().equals(commandedUser.getUsername()))
                throw new IllegalArgumentException("Unauthorized Dragon access with id " + dragonId.get() +".");
            else
            DragonHolder.getCollection().put(dragonKey.get(), DragonUtils.inputDragonFromConsole(commandedUser, dragonId.get(), dragonCreationDate.get()));
        return "Update successful.";
    }
}

```

Файл CsvInsert.java

```java
package com.company.commands;

import com.company.storables.Dragon;
import com.company.storables.DragonHolder;
import com.company.ui.User;

public class CsvInsert implements CommandAction {
    public String getLabel() {
        return "insert_csv";
    }

    public String getArgumentLabel() {
        return "{key},{Dragon,in,csv,style}";
    }

    public String getDescription() {
        return "Insert new {Dragon} to collection with a {key} (CSV style).";
    }

    public String execute(User commandedUser, String argument) {
        if (!argument.isBlank()) {
            String[] splitLine = argument.split(",", 2);
            if (splitLine.length < 2)
                throw new IllegalArgumentException("Invalid key,dragon input string: \"" + argument + "
\".");
            try {
                DragonHolder.getCollection().put(Integer.parseInt(splitLine[0]), new Dragon(splitLine[1
]));
            } catch (NumberFormatException e) {
                throw new IllegalArgumentException("Can't parse key from " + splitLine[0] + ".");
            }
        } else {
            throw new IllegalArgumentException("Please specify key,dragon.");
        }
        return "Insert successful.";
    }
}

```

Файл RemoveKey.java

```java
package com.company.commands;

import com.company.storables.Dragon;
import com.company.storables.DragonHolder;
import com.company.ui.User;

public class RemoveKey implements CommandAction {
    @Override
    public String getLabel() {
        return "remove_key";
    }

    @Override
    public String getArgumentLabel() {
        return "{key}";
    }

    @Override
    public String getDescription() {
        return "Removes element from collection by its {key}";
    }

    @Override
    public String execute(User commandedUser, String argument) {
        int key;
        if (argument == null || argument.isEmpty())
            throw new IllegalArgumentException("Please specify Dragon key.");
        try {
            key = Integer.parseInt(argument);
        } catch (IllegalArgumentException e) {
            throw new IllegalArgumentException("Illegal key: " + e.getMessage() + ".");
        }
        if(DragonHolder.getCollection().get(key) != null)
            if (!DragonHolder.getCollection().get(key).getOwner().equals(commandedUser.getUsername()))
                throw new IllegalArgumentException("Unauthorized Dragon access with id " + DragonHolder.
getCollection().get(key).getId() +".");
        Dragon removed = DragonHolder.getCollection().remove(key);
        if (removed == null)
            throw new IllegalArgumentException("No Dragon found with key \"" + key + "\".");
        return "Remove successful.";
    }
}

```

Файл CsvUpdateID.java

```java
package com.company.commands;

import com.company.storables.Dragon;
import com.company.storables.DragonHolder;
import com.company.ui.User;

import java.util.Date;
import java.util.concurrent.atomic.AtomicBoolean;
import java.util.concurrent.atomic.AtomicReference;

public class CsvUpdateID implements CommandAction {
    @Override
    public String getLabel() {
        return "update_csv";
    }

    @Override
    public String getArgumentLabel() {
        return "{Dragon,in,csv,style}";
    }

    @Override
    public String getDescription() {
        return "Update {element} of collection by its id (CSV version).";
    }

    @Override
    public String execute(User commandedUser, String argument) {
        Dragon newDragon = new Dragon(argument);
        AtomicBoolean found = new AtomicBoolean(false);
        AtomicReference<Integer> dragonKey = new AtomicReference<>();
        AtomicReference<Long> dragonId = new AtomicReference<>();
        AtomicReference<Date> dragonCreationDate = new AtomicReference<>();
        DragonHolder.getCollection().forEach((key, value) -> {
            if (value.getId() == newDragon.getId()) {
                if(!value.getOwner().equals(commandedUser.getUsername()))
                    throw new IllegalArgumentException("Unauthorized Dragon access with id " + value.getId() +".");
                dragonKey.set(key);
                dragonId.set(value.getId());
                dragonCreationDate.set(value.getCreationDate());
                found.set(true);
            }
        });
        if (!found.get()) throw new IllegalArgumentException("Dragon with id '" + argument + "' not found");
        else {
            newDragon.setCreationDate(dragonCreationDate.get());
            DragonHolder.getCollection().put(dragonKey.get(), newDragon);
        }
        return "Update successful.";
    }
}
```

Файл RemoveAllAge.java

```java
package com.company.commands;

import com.company.storables.DragonHolder;
import com.company.ui.User;

import java.util.LinkedList;

public class RemoveAllAge implements CommandAction {
    @Override
    public String getLabel() {
        return "remove_all_by_age";
    }

    @Override
    public String getArgumentLabel() {
        return "{age}";
    }

    @Override
    public String getDescription() {
        return "Remove all Dragons {age} years old.";
    }

    @Override
    public String execute(User commandedUser, String argument) {
        long age;
        if (argument == null || argument.isEmpty())
            throw new IllegalArgumentException("Please specify Dragon age.");
        try {
            age = Long.parseLong(argument);
        } catch (IllegalArgumentException e) {
            throw new IllegalArgumentException("Illegal age: " + e.getMessage() + ".");
        }
        LinkedList<Integer> toRemove = new LinkedList<>();
        DragonHolder.getCollection().forEach((key, dragon) -> {
            if (dragon.getAge() == age && dragon.getOwner().equals(commandedUser.getUsername()))
toRemove.add(key);
        });
        toRemove.forEach(key -> DragonHolder.getCollection().remove(key));
        return "Removed " + toRemove.size() + " Dragons.";
    }
}

```

Файл CommandAction.java

```java
package com.company.commands;

import com.company.ui.User;

/**
 * Command that every program action implements
 */
public interface CommandAction {

    /**
     * Command label
     *
     * @return string that user has to input to use the command
     */
    String getLabel();

    /**
     * Command argument format
     *
     * @return argument format
     */
    default String getArgumentLabel() {
        return "";
    }

    /**
     * Command description
     *
     * @return description
     */
    String getDescription();

    /**
     * Execute the command
     *
     * @param commandedUser User who issued the command
     * @param argument command arguments
     * @return Command execution result
     */
    String execute(User commandedUser, String argument);

}

```

Файл Execute_script.java

```java
package com.company.commands;

import com.company.Server;
import com.company.ui.CommandExecutor;
import com.company.ui.CommandReader;
import com.company.ui.User;

import java.io.*;
import java.nio.charset.Charset;
import java.util.Arrays;

public class Execute_script implements CommandAction {
    @Override
    public String getLabel() {
        return "execute_script";
    }

    @Override
    public String getArgumentLabel() {
        return "file_name";
    }

    @Override
    public String getDescription() {
        return "executes script from \"file_name\"";
    }

    @Override
    public String execute(User commandedUser, String argument) {
        ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
        PrintStream printStream = new PrintStream(outputStream);
        File file;
        try {
            file = new File(argument);
        } catch (NullPointerException e) {
            throw new IllegalArgumentException("Please specify filename.");
        }
        if (argument.isBlank())
            throw new IllegalArgumentException("Please specify filename.");
        try {
            if (!file.canRead())
                throw new IllegalArgumentException("Can't read file \"" + argument + "\".");
        } catch (SecurityException e) {
            throw new IllegalArgumentException("Can't access file \"" + argument + "\".");
        }
        try {
            BufferedInputStream fileReader = new BufferedInputStream(new FileInputStream(file));
            CommandExecutor commandExecutor = new CommandExecutor(CommandExecutor.scriptCommands,
printStream);
            StringBuilder stringBuilder = new StringBuilder();
            while (fileReader.available() > 0)
                stringBuilder.append((char) fileReader.read());
            Arrays.stream(stringBuilder.toString().split("[\\r\\n]+"))
                    .forEach(line -> {
                        if (!line.isBlank())
                            printStream.append(line).append("\n");
                        String formattedLine = line
                                .replaceAll("\\breplace_if_greater\\b", "replace_if_greater_csv")
                                .replaceAll("\\bupdate\\b", "update_csv")
                                .replaceAll("\\binsert\\b", "insert_csv");
                        commandExecutor.executeCommand(CommandReader.readCommandFromString(Server.
internalUser, formattedLine));
                    });
            fileReader.close();
        } catch (FileNotFoundException e) {
            throw new IllegalArgumentException("Can't find file \"" + file + "\".");
        } catch (SecurityException e) {
            throw new IllegalArgumentException("Can't access file \"" + file + "\".");
        } catch (IOException e) {
            throw new IllegalArgumentException("Error occurred accessing file \"" + file + "\".");
        }
        printStream.append("Executed script from file \"").append(argument).append(".");
        return outputStream.toString(Charset.defaultCharset());
    }
```

Файл Execute_script.java

```
73 }
74
```

Файл RemoveLowerKey.java

```java
package com.company.commands;

import com.company.storables.DragonHolder;
import com.company.ui.User;

import java.util.LinkedList;

public class RemoveLowerKey implements CommandAction {
    @Override
    public String getLabel() {
        return "remove_lower_key";
    }

    @Override
    public String getArgumentLabel() {
        return "{key}";
    }

    @Override
    public String getDescription() {
        return "Removes elements from collection with keys less than {key}";
    }

    @Override
    public String execute(User commandedUser, String argument) {
        int keyThreshold;
        if (argument == null || argument.isEmpty())
            throw new IllegalArgumentException("Please specify Dragon key.");
        try {
            keyThreshold = Integer.parseInt(argument);
        } catch (IllegalArgumentException e) {
            throw new IllegalArgumentException("Illegal key: " + e.getMessage() + ".");
        }
        LinkedList<Integer> toRemove = new LinkedList<>();
        DragonHolder.getCollection().forEach((key, dragon) -> {
            if (key < keyThreshold && dragon.getOwner().equals(commandedUser.getUsername())) toRemove.
add(key);
        });
        toRemove.forEach(key -> DragonHolder.getCollection().remove(key));
        return "Removed " + toRemove.size() + " Dragons.";
    }
}

```

Файл PrintDescending.java

```java
package com.company.commands;

import com.company.storables.Dragon;
import com.company.storables.DragonHolder;
import com.company.ui.User;

import java.util.Comparator;

public class PrintDescending implements CommandAction {
    String response;

    public String getLabel() {
        return "print_descending";
    }

    public String getDescription() {
        return "Show all collection elements sorted by age.";
    }

    public String execute(User commandedUser, String argument) {
        response = "Sorted collection:\n";
        DragonHolder.getCollection().values().stream().sorted(Comparator.comparingLong(Dragon::getAge).reversed())
                .forEachOrdered(element -> response += element.toString() + "\n");
        return response.substring(0, response.length() - 1);
    }
}

```

Файл RemoveGreaterKey.java

```java
package com.company.commands;

import com.company.storables.DragonHolder;
import com.company.ui.User;

import java.util.LinkedList;

public class RemoveGreaterKey implements CommandAction {
    @Override
    public String getLabel() {
        return "remove_greater_key";
    }

    @Override
    public String getArgumentLabel() {
        return "{key}";
    }

    @Override
    public String getDescription() {
        return "Removes elements from collection with keys larger than {key}";
    }

    @Override
    public String execute(User commandedUser, String argument) {
        int keyThreshold;
        if (argument == null || argument.isEmpty())
            throw new IllegalArgumentException("Please specify Dragon key.");
        try {
            keyThreshold = Integer.parseInt(argument);
        } catch (IllegalArgumentException e) {
            throw new IllegalArgumentException("Illegal key: " + e.getMessage() + ".");
        }
        LinkedList<Integer> toRemove = new LinkedList<>();
        DragonHolder.getCollection().forEach((key, dragon) -> {
            if (key > keyThreshold && dragon.getOwner().equals(commandedUser.getUsername())) toRemove.
add(key);
        });
        toRemove.forEach(key -> DragonHolder.getCollection().remove(key));
        return "Removed " + toRemove.size() + " Dragons.";
    }
}

```

Файл FilterLessThanType.java

```java
package com.company.commands;

import com.company.storables.Coordinates;
import com.company.storables.Dragon;
import com.company.storables.DragonHead;
import com.company.storables.DragonHolder;
import com.company.ui.User;

import java.text.DateFormat;
import java.text.ParseException;
import java.util.ArrayList;
import java.util.Date;
import java.util.Locale;
import java.util.regex.PatternSyntaxException;

public class FilterLessThanType implements CommandAction {
    String response;

    public String getLabel() {
        return "filter_less_than_type";
    }

    public String getArgumentLabel() {
        return "{type} {value}";
    }

    public String getDescription() {
        return "Filter all elements that have {type} less than {value}.";
    }

    public String execute(User commandedUser, String argument) {
        String[] arguments;
        try {
            arguments = argument.split(" ", 2);
        } catch (PatternSyntaxException | NullPointerException e) {
            throw new IllegalArgumentException("Please specify field type and value.");
        }
        if (arguments.length < 1)
            throw new IllegalArgumentException("Please specify field type and value.");
        if (arguments.length < 2)
            throw new IllegalArgumentException("Please specify field value.");
        String field = arguments[0];
        String value = arguments[1];
        ArrayList<Dragon> result = new ArrayList<>();
        try {
            Dragon.class.getDeclaredField(field);
        } catch (NoSuchFieldException e) {
            throw new IllegalArgumentException("No field named \"" + field + "\".");
        }
        switch (field) {
            case "id":
            case "age":
                try {
                    long compareToLong = Long.parseLong(value);
                    DragonHolder.getCollection().forEach((key, element) -> {
                        if (((field.equals("id")) ? element.getId() : element.getAge()) < compareToLong)
                            result.add(element);
                    });
                } catch (NumberFormatException | NullPointerException e) {
                    throw new IllegalArgumentException("Can't parse " + field + " from \"" + value + "\"
: " + e.getMessage() + ".");
                }
                break;
            case "name":
            case "description":
                DragonHolder.getCollection().forEach((key, element) -> {
                    if (((field.equals("name")) ? element.getName() : element.getDescription()).
compareTo(value) < 0)
                        result.add(element);
                });
                break;
            case "coordinates":
                Coordinates compareToCoordinates = new Coordinates(value);
                DragonHolder.getCollection().forEach((key, element) -> {
```

Файл FilterLessThanType.java

```
                    if (element.getCoordinates().compareTo(compareToCoordinates) < 0)
                        result.add(element);
                });
                break;
            case "creationDate":
                try {
                    Date compareToDate = DateFormat.getDateInstance(DateFormat.SHORT, Locale.getDefault
()).parse(value);
                    DragonHolder.getCollection().forEach((key, element) -> {
                        if (element.getCreationDate().compareTo(compareToDate) < 0)
                            result.add(element);
                    });
                } catch (ParseException e) {
                    throw new IllegalArgumentException("Can't parse date from \"" + value + "\": " + e.
getMessage() + ".");
                }
                break;
            case "weight":
                try {
                    int compareToWeight = Integer.parseInt(value);
                    DragonHolder.getCollection().forEach((key, element) -> {
                        if (element.getWeight() < compareToWeight)
                            result.add(element);
                    });
                } catch (NumberFormatException | NullPointerException e) {
                    throw new IllegalArgumentException("Can't parse weight from \"" + value + "\": " +
e.getMessage() + ".");
                }
                break;
            case "type":
                throw new IllegalArgumentException("Can't compare Dragon types. All Dragons are equal!"
);
            case "head":
                float compareToEyesCount = new DragonHead(value).getEyesCount();
                DragonHolder.getCollection().forEach((key, element) -> {
                    if (element.getHead().getEyesCount() < compareToEyesCount)
                        result.add(element);
                });
                break;
            default:
                throw new IllegalArgumentException("No field named \"" + field + "\".");
        }
        response = "Result:\n";
        result.forEach(element -> response += element.toString() + "\n");
        return response;
    }
}
```

Файл ReplaceIfGreaterAge.java

```java
package com.company.commands;

import com.company.storables.Dragon;
import com.company.storables.DragonHolder;
import com.company.storables.DragonUtils;
import com.company.ui.User;

public class ReplaceIfGreaterAge implements CommandAction {
    @Override
    public String getLabel() {
        return "replace_if_greater";
    }

    @Override
    public String getArgumentLabel() {
        return "{key} {Dragon}";
    }

    @Override
    public String getDescription() {
        return "replaces a {Dragon} by {key} if he has a greater age";
    }

    @Override
    public String execute(User commandedUser, String argument) {
        int key;
        if (argument == null || argument.isEmpty())
            throw new IllegalArgumentException("Please specify Dragon key.");
        try {
            key = Integer.parseInt(argument);
        } catch (IllegalArgumentException e) {
            throw new IllegalArgumentException("Illegal key: " + e.getMessage() + ".");
        }
        Dragon currentDragon = DragonHolder.getCollection().get(key);
        if (currentDragon == null)
            throw new IllegalArgumentException("No Dragon found with key \"" + key + "\".");
        if(!currentDragon.getOwner().equals(commandedUser.getUsername()))
            throw new IllegalArgumentException("Unauthorized Dragon access with id " + DragonHolder.
getCollection().get(key).getId() +".");
        else {
            Dragon newDragon = DragonUtils.inputNewDragonFromConsole(commandedUser);
            if (newDragon.getAge() > currentDragon.getAge()) {
                DragonHolder.getCollection().replace(key, newDragon);
                return "Remove successful.";
            } else {
                return "Dragon not replaced: new dragon is not older.";
            }
        }
    }
}

```

Файл CsvReplaceIfGreaterAge.java

```java
package com.company.commands;

import com.company.storables.Dragon;
import com.company.storables.DragonHolder;
import com.company.storables.DragonUtils;
import com.company.ui.User;

import java.util.Date;

public class CsvReplaceIfGreaterAge implements CommandAction {
    @Override
    public String getLabel() {
        return "replace_if_greater_csv";
    }

    @Override
    public String getArgumentLabel() {
        return "{key},{Dragon,in,csv,style}";
    }

    @Override
    public String getDescription() {
        return "replaces a {Dragon} by {key} if he has a greater age (CSV version).";
    }

    @Override
    public String execute(User commandedUser, String argument) {
        int key;
        if (!argument.isBlank()) {
            String[] splitLine = argument.split(",", 2);
            if (splitLine.length < 2)
                throw new IllegalArgumentException("Invalid key,dragon input string: \"" + argument + "
\".");
            try {
                key = Integer.parseInt(splitLine[0]);
                Dragon currentDragon = DragonHolder.getCollection().get(key);
                if (currentDragon == null)
                    throw new IllegalArgumentException("No Dragon found with key \"" + key + "\".");
                if(!currentDragon.getOwner().equals(commandedUser.getUsername()))
                    throw new IllegalArgumentException("Unauthorized Dragon access with id " +
currentDragon.getId() +".");
                Dragon newDragon = new Dragon(splitLine[1]);
                newDragon.setId(DragonUtils.getNewId());
                newDragon.setCreationDate(new Date());
                if (newDragon.getAge() > currentDragon.getAge()) {
                    DragonHolder.getCollection().replace(Integer.parseInt(splitLine[0]), newDragon);
                    return "Remove successful.";
                } else {
                    return "Dragon not replaced: new dragon is not older.";
                }
            } catch (NumberFormatException e) {
                throw new IllegalArgumentException("Can't parse key from " + splitLine[0] + ".");
            }
        } else {
            throw new IllegalArgumentException("Please specify key,dragon.");
        }
    }
}
```

Файл Dragon.java

```java
package com.company.storables;

import com.company.ui.User;

import java.text.DateFormat;
import java.text.ParseException;
import java.util.Date;
import java.util.Locale;
import java.util.Objects;

public class Dragon implements Comparable<Dragon> {
    private long id; //Значение поля должно быть больше 0, Значение этого поля должно быть уникальным, Значение этого поля должно генерироваться автоматически
    private String name; //Поле не может быть null, Строка не может быть пустой
    private Coordinates coordinates; //Поле не может быть null
    private java.util.Date creationDate; //Поле не может быть null, Значение этого поля должно генерироваться автоматически
    private long age; //Значение поля должно быть больше 0
    private String description; //Поле может быть null
    private int weight; //Значение поля должно быть больше 0
    private DragonType type; //Поле не может быть null
    private DragonHead head;
    private String owner;

    /**
     * Converts CSV format string to Dragon
     *
     * @param csvString CSV format string
     * @throws IllegalArgumentException if CSV format string can't be parsed
     */
    public Dragon(String csvString) throws IllegalArgumentException {
        String[] splitString = csvString.split(",");
        try {
            if (splitString[0].isBlank())
                setId(DragonUtils.getNewId());
            else
                try {
                    if (Long.parseLong(splitString[0]) == -1L)
                        setId(DragonUtils.getNewId());
                    setId(Long.parseLong(splitString[0]));
                } catch (NumberFormatException e) {
                    throw new IllegalArgumentException("Can't parse Dragon id from " + splitString[0] + ".");
                }
            setName(splitString[1].replaceAll("\"", ""));
            try {
                setCoordinates(new Coordinates(splitString[2].isBlank() ? .0 : Double.parseDouble(splitString[2]), Long.parseLong(splitString[3])));
            } catch (NumberFormatException | NullPointerException e) {
                throw new IllegalArgumentException("Can't parse Dragon coordinates from " + splitString[2] + "," + splitString[3] + ".");
            }
            if (splitString[4].isBlank())
                setCreationDate(new Date());
            else
                try {
                    setCreationDate(DateFormat.getDateInstance(DateFormat.SHORT, Locale.getDefault()).parse(splitString[4]));
                } catch (ParseException e) {
                    throw new IllegalArgumentException("Can't parse creation date from " + splitString[4] + ".");
                }
            try {
                setAge(Long.parseLong(splitString[5]));
            } catch (NumberFormatException e) {
                throw new IllegalArgumentException("Can't parse Dragon age from " + splitString[5] + ".");
            }
            setDescription(splitString[6].isBlank() ? "\"\"" : splitString[6].replaceAll("\"", ""));
            try {
                setWeight(Integer.parseInt(splitString[7]));
            } catch (NumberFormatException e) {
                throw new IllegalArgumentException("Can't parse Dragon weight from " + splitString[7] + ".");
```

Файл Dragon.java

```
66              }
67              try {
68                  setType(DragonType.valueOf(splitString[8]));
69              } catch (IllegalArgumentException e) {
70                  throw new IllegalArgumentException("Can't parse Dragon type from " + splitString[8] +
    ".");
71              }
72              setHead(new DragonHead(splitString[9]));
73              setOwner(splitString[10].isBlank() ? "\"\"" : splitString[10].replaceAll("\"", ""));
74          } catch (IndexOutOfBoundsException e) {
75              throw new IllegalArgumentException("Not enough arguments to parse Dragon (" + splitString.
    length + ").");
76          }
77
78      }
79
80      /**
81       * Default constructor
82       */
83      public Dragon() {
84      }
85
86      /**
87       * Dragon constructor with fields
88       * @param id           id
89       * @param name         name
90       * @param coordinates  coordinates
91       * @param creationDate creationDate
92       * @param age          age
93       * @param description  description
94       * @param weight       weight
95       * @param type         type
96       * @param head         head
97       * @param owner        owner
98       */
99      public Dragon(long id, String name, Coordinates coordinates, Date creationDate, long age, String
    description, int weight, DragonType type, DragonHead head, String owner) {
100         setId(id);
101         setName(name);
102         setCoordinates(coordinates);
103         setCreationDate(creationDate);
104         setAge(age);
105         setDescription(description);
106         setWeight(weight);
107         setType(type);
108         setHead(head);
109         setOwner(owner);
110     }
111
112     /**
113      * Dragon == Object equals
114      *
115      * @param o Object
116      * @return true if equal
117      */
118     @Override
119     public boolean equals(Object o) {
120         if (this == o) return true;
121         if (o == null || getClass() != o.getClass()) return false;
122         Dragon dragon = (Dragon) o;
123         return id == dragon.id;
124     }
125
126     /**
127      * Dragon hashcode by id
128      *
129      * @return hashcode
130      */
131     @Override
132     public int hashCode() {
133         return Objects.hash(id);
134     }
135
136     /**
```

Файл Dragon.java

```
137      * Dragon-to-Dragon comparator
138      *
139      * @param p Dragon
140      * @return >0 if this Dragon is older, <0 if this Dragon is younger, 0 if Dragons are of equal age
141      */
142     @Override
143     public int compareTo(Dragon p) {
144         return (int) (this.getAge() - p.getAge());
145     }
146 
147     /**
148      * Dragon ID getter
149      *
150      * @return ID
151      */
152     public long getId() {
153         return id;
154     }
155 
156     public void setId(long id) {
157         this.id = id;
158     }
159 
160     /**
161      * Dragon ID getter
162      *
163      * @return ID
164      */
165     public String getName() {
166         return name;
167     }
168 
169     /**
170      * Dragon name setter
171      *
172      * @param name name
173      */
174     public void setName(String name) {
175         this.name = name;
176     }
177 
178     /**
179      * Dragon ID getter
180      *
181      * @return ID
182      */
183     public Coordinates getCoordinates() {
184         return coordinates;
185     }
186 
187     /**
188      * Dragon coordinates setter
189      *
190      * @param coordinates coordinates
191      */
192     public void setCoordinates(Coordinates coordinates) {
193         this.coordinates = coordinates;
194     }
195 
196     /**
197      * Dragon creation date getter
198      *
199      * @return creation date
200      */
201     public Date getCreationDate() {
202         return creationDate;
203     }
204 
205     /**
206      * Dragon creation date setter
207      *
208      * @param creationDate creationDate
209      */
210     public void setCreationDate(Date creationDate) {
```

Файл Dragon.java

```
211         this.creationDate = creationDate;
212     }
213 
214     /**
215      * Dragon age getter
216      *
217      * @return age
218      */
219     public long getAge() {
220         return age;
221     }
222 
223     /**
224      * Dragon age setter
225      *
226      * @param age age
227      * @throws IllegalArgumentException if age is invalid
228      */
229     public void setAge(long age) throws IllegalArgumentException {
230         if (age < 0)
231             throw new IllegalArgumentException("Age can't be less than 0");
232         this.age = age;
233     }
234 
235     /**
236      * Dragon description getter
237      *
238      * @return description
239      */
240     public String getDescription() {
241         return description;
242     }
243 
244     /**
245      * Dragon description setter
246      *
247      * @param description description
248      */
249     public void setDescription(String description) {
250         this.description = description;
251     }
252 
253     /**
254      * Dragon ID getter
255      *
256      * @return ID
257      */
258     public int getWeight() {
259         return weight;
260     }
261 
262     /**
263      * Dragon weight setter
264      *
265      * @param weight weight
266      * @throws IllegalArgumentException if weight is illegal
267      */
268     public void setWeight(int weight) throws IllegalArgumentException {
269         if (weight < 0)
270             throw new IllegalArgumentException("Weight can't be less than 0");
271         this.weight = weight;
272     }
273 
274     /**
275      * Dragon ID getter
276      *
277      * @return ID
278      */
279     public DragonType getType() {
280         return type;
281     }
282 
283     /**
284      * Dragon type setter
```

Файл Dragon.java

```
285      *
286      * @param type dragon type
287      */
288     public void setType(DragonType type) {
289         this.type = type;
290     }
291 
292     /**
293      * Dragon ID getter
294      *
295      * @return ID
296      */
297     public DragonHead getHead() {
298         return head;
299     }
300 
301     /**
302      * Dragon head setter
303      *
304      * @param head dragon head
305      */
306     public void setHead(DragonHead head) {
307         this.head = head;
308     }
309 
310     /**
311      * Dragon owner getter
312      *
313      * @return owner
314      */
315     public String getOwner() {
316         return owner;
317     }
318 
319     /**
320      * Dragon owner setter
321      *
322      * @param owner new owner;
323      */
324     public void setOwner(String owner) {
325         this.owner = owner;
326     }
327 
328     /**
329      * Dragon - to - string converter
330      *
331      * @return dragon string
332      */
333     @Override
334     public String toString() {
335         return "Dragon: " +
336                 "id=" + id +
337                 ", name='" + name + '\'' +
338                 ", coordinates=" + coordinates +
339                 ", creationDate=" + creationDate +
340                 ", age=" + age +
341                 ", description='" + description + '\'' +
342                 ", weight=" + weight +
343                 ", type=" + type +
344                 ", head=" + head +
345                 ", owner=" + owner;
346     }
347 
348     /**
349      * Converts Dragon to CSV string
350      *
351      * @return CSV string
352      */
353     public String toCsvString() {
354         return getId() + ",\"" + getName() + "\"," + getCoordinates().getX() + "," + getCoordinates().
    getY() + "," + DateFormat.getDateInstance(DateFormat.SHORT).format(getCreationDate()) + "," + getAge
    () + ",\"" + (getDescription() == null ? "" : getDescription()) + "\"," + getWeight() + "," + getType
    ().getLabel() + "," + getHead().getEyesCount() + ",\"" + getOwner() + "\"";
355     }
```

Файл Dragon.java

```
356 }
```

Файл DragonHead.java

```java
package com.company.storables;

/**
 * Head of the dragon
 *
 * @see Dragon
 */
public class DragonHead {

    private float eyesCount;

    /**
     * Dragon head from string constructor
     *
     * @param fromString string to parse from
     * @throws IllegalArgumentException if string is invalid
     */
    public DragonHead(String fromString) throws IllegalArgumentException {
        try {
            this.eyesCount = Float.parseFloat(fromString);
        } catch (NumberFormatException | NullPointerException e) {
            throw new IllegalArgumentException("Can't parse Dragon Head from \"" + fromString + "\".");
        }
    }

    /**
     * Dragon head constructor
     *
     * @param eyesCount count of eyes
     */
    public DragonHead(float eyesCount) {
        this.eyesCount = eyesCount;
    }

    /**
     * Get number of eyes
     *
     * @return number of eyes
     */
    public float getEyesCount() {
        return eyesCount;
    }

    /**
     * Set number of eyes
     *
     * @param eyesCount new number of eyes
     */
    public void setEyesCount(float eyesCount) {
        this.eyesCount = eyesCount;
    }

    @Override
    public String toString() {
        return "DragonHead: " +
                " eyesCount=" + eyesCount;
    }
}

```

Файл DragonType.java

```java
package com.company.storables;

/**
 * Type of the dragon
 *
 * @see Dragon
 */
public enum DragonType {
    AIR("AIR"),
    FIRE("FIRE"),
    UNDERGROUND("UNDERGROUND"),
    WATER("WATER");

    private final String label;

    /**
     * Dragon type constructor
     *
     * @param label name
     */
    DragonType(String label) {
        this.label = label;
    }

    /**
     * Gets dragon type name
     *
     * @return dragon type name
     */
    public String getLabel() {
        return label;
    }
}

```

Файл Coordinates.java

```java
package com.company.storables;

import java.util.regex.PatternSyntaxException;

/**
 * Coordinates where Dragon can be at
 *
 * @see Dragon
 */
public class Coordinates implements Comparable<Coordinates> {

    private double x;
    private Long y; //Can't be null

    /**
     * Coordinates from string constructor
     *
     * @param fromString string to parse from
     * @throws IllegalArgumentException if string is invalid
     */
    public Coordinates(String fromString) throws IllegalArgumentException {
        String[] arguments;
        try {
            arguments = fromString.split(" ", 2);
        } catch (PatternSyntaxException e) {
            throw new IllegalArgumentException("Error in argument: " + e.getMessage() + ".");
        }
        try {
            if (arguments.length > 1) {
                this.x = Double.parseDouble(arguments[0]);
                this.y = Long.parseLong(arguments[1]);
            } else if (arguments.length > 0) {
                this.y = Long.parseLong(arguments[0]);
            }
        } catch (NumberFormatException | NullPointerException e) {
            throw new IllegalArgumentException("Can't parse Coordinates from \"" + fromString + "\": "
    + e.getMessage() + ".");
        }
    }

    /**
     * Coordinates constructor
     *
     * @param x x coordinate
     * @param y y coordinate
     */
    public Coordinates(double x, Long y) {
        this.x = x;
        this.y = y;
    }

    /**
     * Gets x coordinate
     *
     * @return x coordinate
     */
    public double getX() {
        return x;
    }

    /**
     * Sets x coordinate
     *
     * @param x new x coordinate
     */
    public void setX(double x) {
        this.x = x;
    }

    /**
     * Gets y coordinate
     *
     * @return y coordinate
     */
```

Файл Coordinates.java

```
 74     public Long getY() {
 75         return y;
 76     }
 77
 78     /**
 79      * Sets y coordinate
 80      *
 81      * @param y new y coordinate
 82      */
 83     public void setY(Long y) {
 84         if (y == null)
 85             throw new IllegalArgumentException("y can't be null");
 86         this.y = y;
 87     }
 88
 89     @Override
 90     public String toString() {
 91         return "{" +
 92                 "x=" + x +
 93                 ";y=" + y +
 94                 '}';
 95     }
 96
 97     @Override
 98     public int compareTo(Coordinates o) {
 99         return (int) (Math.sqrt(this.getY() * this.getY() + this.getX() * this.getX()) - Math.sqrt(o.
    getY() * o.getY() + o.getX() * o.getX()));
100     }
101 }
```

Файл DragonUtils.java

```java
package com.company.storables;

import com.company.ui.CommandReader;
import com.company.ui.User;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Date;


/**
 * Class that is needed to create dragons
 */
public class DragonUtils {

    /**
     * Gew unique Dragon ID
     *
     * @return ID
     */
    synchronized public static long getNewId() {
        ArrayList<Long> usedIDs = new ArrayList<>();
        DragonHolder.getCollection().values().forEach(dragon -> usedIDs.add(dragon.getId()));
        long id = 1;
        while (usedIDs.contains(id)) id++;
        usedIDs.add(id);
        return id;
    }

    /**
     * Inputs new dragon from Console (System.in)
     *
     * @return new Dragon
     */
    public static Dragon inputNewDragonFromConsole(User owner) {
        return inputDragonFromConsole(owner,getNewId(), new Date());
    }

    /**
     * Update dragon from Console (System.in), with existing id and
     *
     * @param owner        creator
     * @param id           id
     * @param creationDate creation date
     * @return new dragon
     */
    public static Dragon inputDragonFromConsole(User owner, long id, Date creationDate) {
        Dragon dragon = new Dragon();
        dragon.setOwner(owner.getUsername());
        dragon.setId(id);
        System.out.println("Please input {Name} {Age} [Description] {Weight}.");
        String name = "";
        long age = 0;
        String description = null;
        int weight = 0;
        boolean success = false;
        do {
            try {
                String[] input = CommandReader.getStringsFromTerminal();
                if (input.length < 4) {
                    if (input.length == 3) {
                        dragon.setName(input[0]);
                        dragon.setAge(Integer.parseInt(input[1]));
                        dragon.setWeight(Integer.parseInt(input[2]));
                    } else {
                        throw new IllegalArgumentException(Arrays.toString(input) + input.length);
                    }
                } else {
                    dragon.setName(input[0]);
                    dragon.setAge(Integer.parseInt(input[1]));
                    dragon.setDescription(input[2]);
                    dragon.setWeight(Integer.parseInt(input[3]));
                }
                success = true;
```

Файл DragonUtils.java

```
75              } catch (IllegalArgumentException e) {
76                  System.out.println("Invalid arguments for {Name} {Age>0} [Description] {Weight>0}: " +
    e.getMessage() + ".");
77              }
78          } while (!success);
79          System.out.println("Please input coordinates: [x] {y}.");
80          success = false;
81          Coordinates coordinates = new Coordinates(.0, 0L);
82          do {
83              try {
84                  String[] input = CommandReader.getStringsFromTerminal();
85                  if (input.length < 2) {
86                      if (input.length < 1)
87                          throw new IllegalArgumentException("Not enough parameters.");
88                      else
89                          coordinates.setY(Long.parseLong(input[0]));
90                  } else {
91                      coordinates.setX(Double.parseDouble(input[0]));
92                      coordinates.setY(Long.parseLong(input[0]));
93                  }
94                  success = true;
95              } catch (IllegalArgumentException e) {
96                  System.out.println("Invalid arguments for {x} {y}: " + e.getMessage() + ".");
97              }
98          } while (!success);
99          dragon.setCoordinates(coordinates);
100         DragonType dragonType = null;
101         System.out.println("Please specify {Dragon Type} from" + Arrays.toString(DragonType.values()));
102         success = false;
103         do {
104             try {
105                 String[] input = CommandReader.getStringsFromTerminal();
106                 if (input.length < 1)
107                     throw new IllegalArgumentException("Not enough parameters.");
108                 dragonType = DragonType.valueOf(input[0]);
109                 success = true;
110             } catch (IllegalArgumentException e) {
111                 System.out.println("Invalid arguments for {Dragon Type} from " + Arrays.toString(
    DragonType.values()) + ".");
112             }
113         } while (!success);
114         dragon.setType(dragonType);
115         DragonHead dragonHead = null;
116         System.out.println("Please specify [Dragon Head] eyes count");
117         success = false;
118         do {
119             try {
120                 String[] input = CommandReader.getStringsFromTerminal();
121                 if (input.length < 1)
122                     success = true;
123                 else
124                     dragonHead = new DragonHead(Float.parseFloat(input[0]));
125                 success = true;
126             } catch (IllegalArgumentException e) {
127                 System.out.println("Invalid arguments for [Dragon Head] eyes count: " + e.getMessage
    () + ".");
128             }
129         } while (!success);
130         dragon.setHead(dragonHead);
131         dragon.setCreationDate(creationDate);
132         return dragon;
133     }
134 }
135
```

Файл DragonHolder.java

```java
package com.company.storables;

import java.util.Date;
import java.util.Hashtable;

/**
 * Class that holds the collection
 */
public class DragonHolder {
    private static final Date initializationDate = new Date();

    private static final Hashtable<Integer, Dragon> collection = new Hashtable<>();

    /**
     * Access the collection
     *
     * @return Dragon collection
     */
    synchronized public static Hashtable<Integer, Dragon> getCollection() {
        return collection;
    }

    /**
     * Get the time when collection was initialized
     *
     * @return Date time
     */
    public static Date getInitializationDate() {
        return initializationDate;
    }
}

```