МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

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

ОТЧЕТ

по лабораторной работе №5
в рамках дисциплины «Программирование»

Выполнил: А. С. Медведев
Студент группы R3237

Санкт-Петербург

2021

Цель работы:

Реализовать консольное приложение, которое реализует управление коллекцией объектов в интерактивном режиме. В коллекции необходимо хранить объекты класса Dragon.

Программа должна удовлетворять следующим требованиям:

- Класс, коллекцией экземпляров которого управляет программа, должен реализовывать сортировку по умолчанию.
- Все требования к полям класса (указанные в виде комментариев) должны быть выполнены.
- Для хранения необходимо использовать коллекцию типа java.util.Hashtable
- При запуске приложения коллекция должна автоматически заполняться значениями из файла.
- Имя файла должно передаваться программе с помощью: аргумент командной строки.
- Данные должны храниться в файле в формате csv
- Чтение данных из файла необходимо реализовать с помощью класса java.io.BufferedInputStream
- Запись данных в файл необходимо реализовать с помощью класса java.io.OutputStreamWriter
- Все классы в программе должны быть задокументированы в формате javadoc.
- Программа должна корректно работать с неправильными данными (ошибки пользовательского ввода, отсутсвие прав доступа к файлу и т.п.).

В интерактивном режиме программа должна поддерживать выполнение следующих команд:

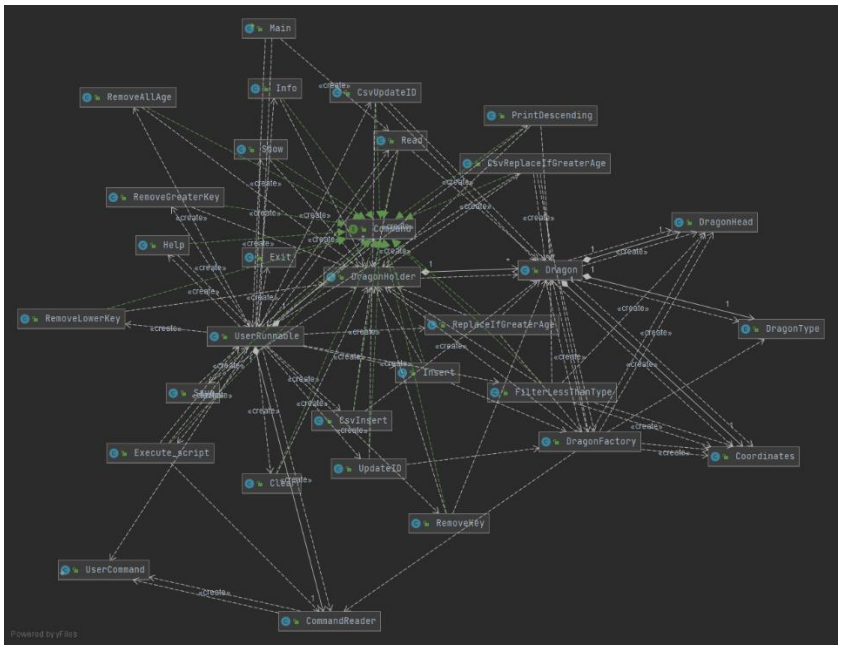
- help : вывести справку по доступным командам
- info : вывести в стандартный поток вывода информацию о коллекции (тип, дата инициализации, количество элементов и т.д.)
- show : вывести в стандартный поток вывода все элементы коллекции в строковом представлении
- insert null {element} : добавить новый элемент с заданным ключом
- update id {element} : обновить значение элемента коллекции, id которого равен заданному
- remove_key null : удалить элемент из коллекции по его ключу

- clear : очистить коллекцию
- save : сохранить коллекцию в файл
- execute_script file_name : считать и исполнить скрипт из указанного файла. В скрипте содержатся команды в таком же виде, в котором их вводит пользователь в интерактивном режиме.
- exit : завершить программу (без сохранения в файл)
- replace_if_greater null {element} : заменить значение по ключу, если новое значение больше старого
- remove_greater_key null : удалить из коллекции все элементы, ключ которых превышает заданный
- remove_lower_key null : удалить из коллекции все элементы, ключ которых меньше, чем заданный
- remove_all_by_age age : удалить из коллекции все элементы, значение поля age которого эквивалентно заданному
- filter_less_than_type type : вывести элементы, значение поля type которых меньше заданного
- print_descending : вывести элементы коллекции в порядке убывания

Выполнение работы:

Исходный код программы представлен в Git-репозитории oaleksander/Lab5 (github.com).

Диаграмма классов:



Выводы:

В ходе данной лабораторной работы реализовано консольное приложение, позволяющее управлять коллекцией объектов в интерактивном режиме, с поддержкой исполнения команд из файла и сохранения/загрузки коллекции в файл.

Файл Main.java

```java
package com.company;

import com.company.commands.Read;
import com.company.ui.UserRunnable;

/**
 * Main client class
 */
public class Main {

    /**
     * Main client function
     *
     * @param args filename
     * @see com.company.commands.Save
     * @see Read
     * @see UserRunnable
     */
    public static void main(String[] args) {
        UserRunnable userRunnable = new UserRunnable(UserRunnable.allCommands, System.out, System.in);

        System.out.println("Welcome to interactive Dragon Hashtable manager. To get help, enter \"help\"
.");
        if (args.length == 0)
            System.out.println("Input filename not specified by command line argument. Skipping...");
        else {
            try {
                UserRunnable.setFile(args[0]);
                try {
                    System.out.println(new Read().execute());
                } catch (Exception e) {
                    System.out.println(e.getMessage() + " Skipping...");
                }
            } catch (NullPointerException e) {
                System.out.println("Input filename is empty. Skipping...");
            }
        }

        userRunnable.run();
    }
}

```

Файл UserRunnable.java

```java
package com.company.ui;

import com.company.commands.*;

import java.io.*;

/**
 * Main command execution runnable
 */
public class UserRunnable implements Runnable {

    /**
     * All possible commands
     */
    public static final Command[] 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()
    };
    /**
     * Commands that user can use
     */
    public static final Command[] userCommands = {
            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()
    };
    /**
     * Programs that execute_script can use
     */
    public static final Command[] scriptCommands = {
            new Help(),
            new Info(),
            new Show(),
            //new Insert(),
            //new UpdateID(),
            new RemoveKey(),
            new Clear(),
            new Save(),
            //new Execute_script(),
```

Файл UserRunnable.java

```
            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 CommandReader commandReader;
    private final Command[] availableCommands;

    /**
     * User runnable constructor
     *
     * @param availableCommands set of available commands
     * @param printStream       PrintStream to output to
     * @param inputStream       InputStream to input from
     */
    public UserRunnable(Command[] availableCommands, PrintStream printStream, InputStream inputStream
) {
        this.availableCommands = availableCommands;
        this.printStream = printStream;
        this.commandReader = new CommandReader(new BufferedReader(new InputStreamReader(inputStream)));
    }

    /**
     * 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 userCommand User command
     */
    public void Execute(CommandReader.UserCommand userCommand) {
        boolean commandIsFound = false;
        String response = "Command gave no response.";
        try {
            for (Command command : availableCommands) {
                if (userCommand.Command.equals(command.getLabel()) && !commandIsFound) {
                    commandIsFound = true;
                    response = command.execute(userCommand.Argument);
                }
            }
            if (!commandIsFound)
                response = "Unknown command \"" + userCommand.Command + "\". try \"help\" for list of
commands";
        } catch (IllegalArgumentException e) {
            response = e.getMessage();
        } catch (Exception e) {
            response = "Unexpected error: " + e.getMessage() + ". This is a bug!";
            e.printStackTrace();
        }
        printStream.println(response);
```

Файл UserRunnable.java

```
147     }
148
149     /**
150      * Thing that executes commands from bufferedReader until System.exit
151      */
152     @Override
153     public void run() {
154         //noinspection InfiniteLoopStatement
155         for (; ; ) {
156             Execute(commandReader.readCommandFromBufferedReader());
157         }
158     }
159 }
160
161
```

Файл CommandReader.java

```java
package com.company.ui;

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

/**
 * Class designed to get commands from buffered readers and strings
 *
 * @see UserRunnable
 */
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 singleString string to parse from
     * @return Command
     */
    public static UserCommand readCommandFromString(String singleString) {
        return (readCommandFromString(singleString.split(" ", 2)));
    }

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

    /**
     * Gets a string from buffered reader line
     *
     * @return Received string
     */
    public String getStringFromBufferedReader() {
        try {
            return bufferedReader.readLine();
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage() + ".");
            return "";
        }
    }

```

Файл CommandReader.java

```
74      /**
75       * Get a command from Buffered Reader
76       *
77       * @return Command
78       */
79      public UserCommand readCommandFromBufferedReader() {
80          return readCommandFromString(getStringFromBufferedReader());
81      }
82
83      /**
84       * User command class
85       */
86      public static class UserCommand {
87          public String Command = null;
88          public String Argument = null;
89
90          /**
91           * User command constructor with argument
92           *
93           * @param Command  Command
94           * @param Argument Argument
95           */
96          public UserCommand(String Command, String Argument) {
97              this.Command = Command;
98              this.Argument = Argument;
99          }
100
101         /**
102          * User command constructor without argument
103          *
104          * @param Command Command
105          */
106         public UserCommand(String Command) {
107             this.Command = Command;
108         }
109
110         /**
111          * Empty user command constructor
112          */
113         public UserCommand() {
114         }
115
116         @Override
117         public String toString() {
118             return "Command{" +
119                     "Command='" + Command + '\'' +
120                     ", Arguments='" + Argument + '\'' +
121                     '}';
122         }
123     }
124 }
125
```

Файл Exit.java

```java
package com.company.commands;

public class Exit implements Command {

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

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

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

```

Файл Help.java

```java
package com.company.commands;

import com.company.ui.UserRunnable;

import java.util.Arrays;

public class Help implements Command {
    String response;

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

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

    public String execute(String argument) {
        response = "Available commands:\n";
        Arrays.stream(UserRunnable.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.collectionmanagement.DragonHolder;

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

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

    public String execute(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.collectionmanagement.DragonHolder;
import com.company.storables.Dragon;
import com.company.ui.UserRunnable;

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

public class Read implements Command {

    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(String argument) {
        response = "";
        File file = UserRunnable.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.collectionmanagement.DragonHolder;
import com.company.ui.UserRunnable;

import java.io.*;

public class Save implements Command {

    @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(String argument) {
        File file = UserRunnable.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.collectionmanagement.DragonHolder;

import java.util.Arrays;

public class Show implements Command {
    String response;

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

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

    public String execute(String argument) {
        response = "Collection:\n";
        Arrays.stream(DragonHolder.getCollection().values().toArray()).forEach(element -> response += element.toString() + "\n");
        return response;
    }
}

```

Файл Clear.java

```java
package com.company.commands;

import com.company.collectionmanagement.DragonHolder;

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

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

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

```

Файл Insert.java

```java
package com.company.commands;

import com.company.collectionmanagement.DragonFactory;
import com.company.collectionmanagement.DragonHolder;

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

```

Файл Command.java

```java
package com.company.commands;

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

    /**
     * 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 argument command arguments
     * @return Command execution result
     */
    String execute(String argument);

    /**
     * Execute the command without arguments
     * Should not usually be overridden
     *
     * @return Command execution result
     */
    default String execute() {
        return execute("");
    }

}

```

Файл UpdateID.java

```java
package com.company.commands;

import com.company.collectionmanagement.DragonFactory;
import com.company.collectionmanagement.DragonHolder;

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

public class UpdateID implements Command {
    @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(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
            DragonHolder.getCollection().put(dragonKey.get(), DragonFactory.inputDragonFromConsole(dragonId.get(), dragonCreationDate.get()));
        return "Update successful.";
    }
}

```

Файл CsvInsert.java

```java
package com.company.commands;

import com.company.collectionmanagement.DragonHolder;
import com.company.storables.Dragon;

public class CsvInsert implements Command {
    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(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.collectionmanagement.DragonHolder;
import com.company.storables.Dragon;

public class RemoveKey implements Command {
    @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(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 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.collectionmanagement.DragonHolder;
import com.company.storables.Dragon;

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

public class CsvUpdateID implements Command {
    @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(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()) {
                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.collectionmanagement.DragonHolder;

import java.util.LinkedList;

public class RemoveAllAge implements Command {
    @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(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) toRemove.add(key);
        });
        toRemove.forEach(key -> DragonHolder.getCollection().remove(key));
        return "Removed " + toRemove.size() + " Dragons.";
    }
}

```

Файл Execute_script.java

```java
package com.company.commands;

import com.company.ui.CommandReader;
import com.company.ui.UserRunnable;

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

public class Execute_script implements Command {
    @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(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));
            UserRunnable userRunnable = new UserRunnable(UserRunnable.scriptCommands, printStream,
fileReader);
            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");
                        userRunnable.Execute(CommandReader.readCommandFromString(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 RemoveLowerKey.java

```java
package com.company.commands;

import com.company.collectionmanagement.DragonHolder;

import java.util.LinkedList;

public class RemoveLowerKey implements Command {
    @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(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) 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.collectionmanagement.DragonHolder;
import com.company.storables.Dragon;

import java.util.Comparator;

public class PrintDescending implements Command {
    String response;

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

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

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

```

Файл RemoveGreaterKey.java

```java
package com.company.commands;

import com.company.collectionmanagement.DragonHolder;

import java.util.LinkedList;

public class RemoveGreaterKey implements Command {
    @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(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) 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.collectionmanagement.DragonHolder;
import com.company.storables.Coordinates;
import com.company.storables.Dragon;
import com.company.storables.DragonHead;

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 Command {
    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(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) -> {
                    if (element.getCoordinates().compareTo(compareToCoordinates) < 0)
```

Файл FilterLessThanType.java

```
                        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.collectionmanagement.DragonFactory;
import com.company.collectionmanagement.DragonHolder;
import com.company.storables.Dragon;

public class ReplaceIfGreaterAge implements Command {
    @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(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 + "\".");
        else {
            Dragon newDragon = DragonFactory.inputNewDragonFromConsole();
            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.collectionmanagement.DragonFactory;
import com.company.collectionmanagement.DragonHolder;
import com.company.storables.Dragon;

import java.util.Date;

public class CsvReplaceIfGreaterAge implements Command {
    @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(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 + "\".");
                Dragon newDragon = new Dragon(splitLine[1]);
                newDragon.setId(DragonFactory.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.collectionmanagement.DragonFactory;

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;

    /**
     * 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(DragonFactory.getNewId());
            else
                try {
                    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] + ".");
            }
            try {
                setType(DragonType.valueOf(splitString[8]));
```

Файл Dragon.java

```
66              } catch (IllegalArgumentException e) {
67                  throw new IllegalArgumentException("Can't parse Dragon type from " + splitString[8] +
    ".");
68              }
69              setHead(new DragonHead(splitString[9]));
70          } catch (IndexOutOfBoundsException e) {
71              throw new IllegalArgumentException("Not enough arguments to parse Dragon (" + splitString.
    length + ").");
72          }
73
74      }
75
76      /**
77       * Default constructor
78       */
79      public Dragon() {
80      }
81
82      /**
83       * Dragon constructor with fields
84       *
85       * @param id           id
86       * @param name         name
87       * @param coordinates  coordinates
88       * @param creationDate creationDate
89       * @param age          age
90       * @param description  description
91       * @param weight       weight
92       * @param type         type
93       * @param head         head
94       */
95      public Dragon(long id, String name, Coordinates coordinates, Date creationDate, long age, String
    description, int weight, DragonType type, DragonHead head) {
96          setId(id);
97          setName(name);
98          setCoordinates(coordinates);
99          setCreationDate(creationDate);
100         setAge(age);
101         setDescription(description);
102         setWeight(weight);
103         setType(type);
104         setHead(head);
105     }
106
107     /**
108      * Dragon == Object equals
109      *
110      * @param o Object
111      * @return true if equal
112      */
113     @Override
114     public boolean equals(Object o) {
115         if (this == o) return true;
116         if (o == null || getClass() != o.getClass()) return false;
117         Dragon dragon = (Dragon) o;
118         return id == dragon.id;
119     }
120
121     /**
122      * Dragon hashcode by id
123      *
124      * @return hashcode
125      */
126     @Override
127     public int hashCode() {
128         return Objects.hash(id);
129     }
130
131     /**
132      * Dragon-to-Dragon comparator
133      *
134      * @param p Dragon
135      * @return >0 if this Dragon is older, <0 if this Dragon is younger, 0 if Dragons are of equal age
136      */
```

Файл Dragon.java

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

Файл Dragon.java

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

Файл Dragon.java

```
285     }
286
287     /**
288      * Dragon ID getter
289      *
290      * @return ID
291      */
292     public DragonHead getHead() {
293         return head;
294     }
295
296     /**
297      * Dragon head setter
298      *
299      * @param head dragon head
300      */
301     public void setHead(DragonHead head) {
302         this.head = head;
303     }
304
305     /**
306      * Dragon - to - string converter
307      *
308      * @return dragon string
309      */
310     @Override
311     public String toString() {
312         return "Dragon: " +
313                 "id=" + id +
314                 ", name='" + name + '\'' +
315                 ", coordinates=" + coordinates +
316                 ", creationDate=" + creationDate +
317                 ", age=" + age +
318                 ", description='" + description + '\'' +
319                 ", weight=" + weight +
320                 ", type=" + type +
321                 ", head=" + head;
322     }
323
324     /**
325      * Converts Dragon to CSV string
326      *
327      * @return CSV string
328      */
329     public String toCsvString() {
330         return getId() + ",\"" + getName() + "\"," + getCoordinates().getX() + "," + getCoordinates().
    getY() + "," + DateFormat.getDateInstance(DateFormat.SHORT, Locale.getDefault()).format(getCreationDate
    ()) + "," + getAge() + ",\"" + (getDescription() == null ? "" : getDescription()) + "\"," + getWeight
    () + "," + getType().getLabel() + "," + getHead().getEyesCount();
331     }
332 }
```

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
     */
    public Long getY() {
```

Файл Coordinates.java

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

Файл DragonHolder.java

```java
package com.company.collectionmanagement;

import com.company.storables.Dragon;

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;
    }
}

```

Файл DragonFactory.java

```java
package com.company.collectionmanagement;

import com.company.storables.Coordinates;
import com.company.storables.Dragon;
import com.company.storables.DragonHead;
import com.company.storables.DragonType;
import com.company.ui.CommandReader;

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


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

    /**
     * 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() {
        return inputDragonFromConsole(getNewId(), new Date());
    }

    /**
     * Update dragon from Console (System.in), with existing id and
     *
     * @param id
     * @param creationDate
     * @return
     */
    public static Dragon inputDragonFromConsole(long id, Date creationDate) {
        Dragon dragon = new Dragon();
        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]));
                }
```

Файл DragonFactory.java

```
75                  success = true;
76              } catch (IllegalArgumentException e) {
77                  System.out.println("Invalid arguments for {Name} {Age>0} [Description] {Weight>0}: " +
    e.getMessage() + ".");
78              }
79          } while (!success);
80          System.out.println("Please input coordinates: [x] {y}.");
81          success = false;
82          Coordinates coordinates = new Coordinates(.0, 0L);
83          do {
84              try {
85                  String[] input = CommandReader.getStringsFromTerminal();
86                  if (input.length < 2) {
87                      if (input.length < 1)
88                          throw new IllegalArgumentException("Not enough parameters.");
89                      else
90                          coordinates.setY(Long.parseLong(input[0]));
91                  } else {
92                      coordinates.setX(Double.parseDouble(input[0]));
93                      coordinates.setY(Long.parseLong(input[0]));
94                  }
95                  success = true;
96              } catch (IllegalArgumentException e) {
97                  System.out.println("Invalid arguments for {x} {y}: " + e.getMessage() + ".");
98              }
99          } while (!success);
100         dragon.setCoordinates(coordinates);
101         DragonType dragonType = null;
102         System.out.println("Please specify {Dragon Type} from" + Arrays.toString(DragonType.values()));
103         success = false;
104         do {
105             try {
106                 String[] input = CommandReader.getStringsFromTerminal();
107                 if (input.length < 1)
108                     throw new IllegalArgumentException("Not enough parameters.");
109                 dragonType = DragonType.valueOf(input[0]);
110                 success = true;
111             } catch (IllegalArgumentException e) {
112                 System.out.println("Invalid arguments for {Dragon Type} from " + Arrays.toString(
    DragonType.values()) + ".");
113             }
114         } while (!success);
115         dragon.setType(dragonType);
116         DragonHead dragonHead = null;
117         System.out.println("Please specify [Dragon Head] eyes count");
118         success = false;
119         do {
120             try {
121                 String[] input = CommandReader.getStringsFromTerminal();
122                 if (input.length < 1)
123                     success = true;
124                 else
125                     dragonHead = new DragonHead(Float.parseFloat(input[0]));
126                 success = true;
127             } catch (IllegalArgumentException e) {
128                 System.out.println("Invalid arguments for [Dragon Head] eyes count: " + e.getMessage
    () + ".");
129             }
130         } while (!success);
131         dragon.setHead(dragonHead);
132         dragon.setCreationDate(creationDate);
133         return dragon;
134     }
135 }
136
```