

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:  
Студент группы R3237

A. C. Медведев

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.

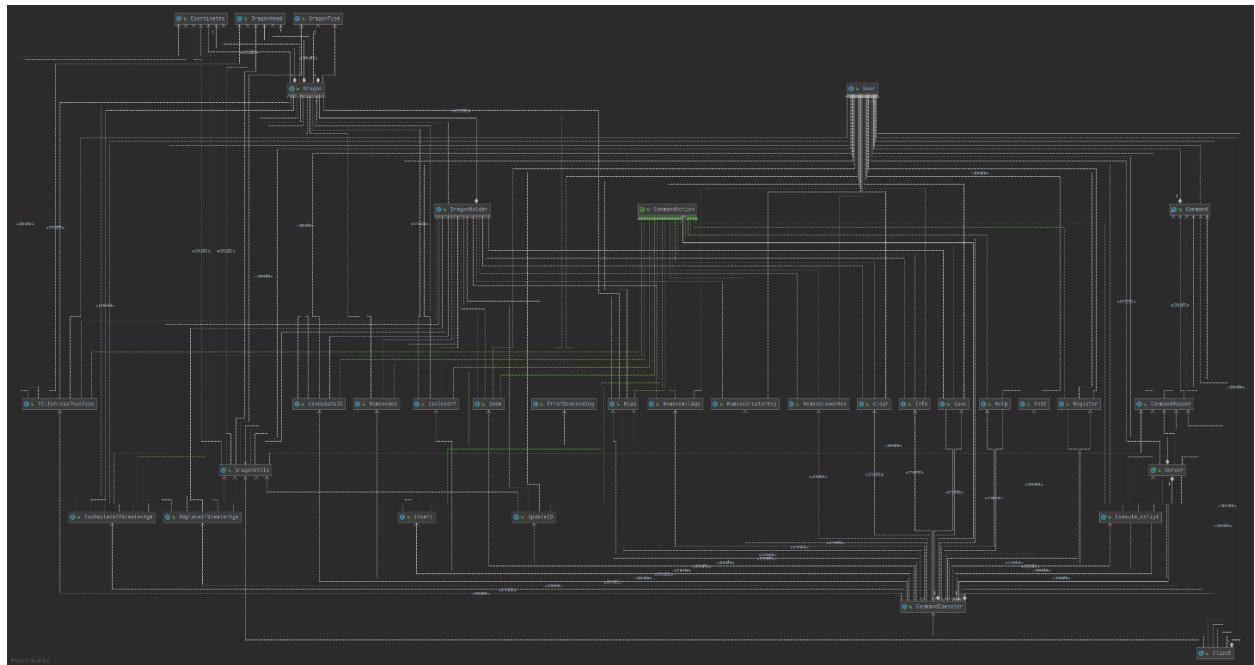
Необходимо реализовать многопоточную обработку запросов.

- Для многопоточного чтения запросов использовать создание нового потока (java.lang.Thread)
- Для многопоточной 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 \(github.com\)](https://github.com/oaleksander/Lab7).

## 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

```
1 package com.company;
2
3 import com.company.storables.DragonUtils;
4 import com.company.ui.CommandExecutor;
5 import com.company.ui.CommandReader;
6 import com.company.ui.User;
7
8 import java.io.*;
9 import java.net.DatagramPacket;
10 import java.net.DatagramSocket;
11 import java.net.InetSocketAddress;
12 import java.util.Arrays;
13 import java.util.Date;
14
15 /**
16  * Main client class
17  * @see Server
18 */
19 public class Client {
20
21     private static final String[] dragonCommands = {"insert", "update", "replace_if_greater_age"};
22     private static final InetSocketAddress address = new InetSocketAddress("localhost", 3333);
23     private static User user = new User("notYetLoggedInUser","");
24     private static ByteArrayOutputStream byteArrayOutputStream;
25     private static ObjectOutputStream objectOutput;
26     private static DatagramSocket datagramSocket = null;
27
28     /**
29      * Main client function
30      *
31      * @param args filename
32      * @see CommandExecutor
33      * @see Server
34      */
35     public static void main(String[] args) {
36         BufferedReader cin = new BufferedReader(new InputStreamReader(System.in));
37
38         String username;
39         String password;
40         try {
41             System.out.println("Username: ");
42             username = cin.readLine();
43             System.out.println("Password: ");
44             password = cin.readLine();
45             user = new User(username,password);
46         } catch (IOException e) {
47             e.printStackTrace();
48             return;
49         }
50
51         CommandReader commandReader = new CommandReader(cin);
52         byteArrayOutputStream = new ByteArrayOutputStream();
53
54         System.out.println("Logging in...");
55         try {
56             datagramSocket = new DatagramSocket();
57             datagramSocket.setBroadcast(true);
58             datagramSocket.setSoTimeout(1500);
59             byteArrayOutputStream.reset();
60             objectOutput = new ObjectOutputStream(byteArrayOutputStream);
61             sendCommand(new CommandReader.Command(user, "register", username+" "+password));
62             System.out.print(getResponse());
63         } catch (IOException e) {
64             System.out.println("Can't connect to server.");
65             System.exit(-1);
66         }
67
68         System.out.println("Welcome to interactive Dragon Hashtable server. To get help, enter \"help\"");
69         //noinspection InfiniteLoopStatement
70         while (true) {
71             CommandReader.Command command = commandReader.readCommandFromBufferedReader(user);
72             try {
73                 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 * Gets a command, verifies it and sends it to server (or exits the program if that is the case)
90 * @param userCommand Command input
91 * @return true if Command was verified and sent
92 * @throws IOException If problem occurred while sending to server
93 */
94 private static boolean sendCommand(CommandReader.Command userCommand) throws IOException {
95     if (userCommand.commandString.equals("exit")) {
96         System.out.println("Exiting...");
97         byteArrayOutputStream.close();
98         datagramSocket.close();
99         System.exit(0);
100    }
101    if (userCommand.commandString.equals("register")) {
102        String[] arguments = userCommand.argumentString.split(" ", 2);
103        if(arguments.length == 2)
104            user = new User(arguments[0],arguments[1]);
105        else if(arguments.length == 1)
106            user = new User(arguments[0]);
107    }
108    if (Arrays.stream(CommandExecutor.userCommands).parallel().noneMatch(command -> command.
109        getLabel().equals(userCommand.commandString))) {
110        System.err.println("Unknown command \"\" + userCommand.commandString + "\". Try \"help\" for
111        list of commands");
112        return false;
113    }
114    if (!Arrays.stream(CommandExecutor.userCommands).parallel().filter(command -> command.getLabel()
115        .equals(userCommand.commandString)).findAny().get().getArgumentLabel().equals(""))
116        && userCommand.argumentString.isBlank()) {
117            System.err.println("Please specify command argument.");
118            return false;
119        }
120        Date date = new Date();
121        String dragonString;
122        if (Arrays.stream(dragonCommands).parallel().anyMatch(command -> command.equals(userCommand.
123            commandString))) {
124            userCommand.commandString += "_csv";
125            if (userCommand.commandString.equals("update_id_csv")) {
126                try {
127                    dragonString = DragonUtils.inputDragonFromConsole(user,Long.parseLong(userCommand.
128                        argumentString), date).toCsvString();
129                    userCommand.argumentString = dragonString;
130                } catch (NumberFormatException e) {
131                    System.err.println("Can't parse dragon ID from " + userCommand.argumentString + ".");
132                }
133            } else {
134                dragonString = DragonUtils.inputDragonFromConsole(user,-1, date).toCsvString(); // -1
135                value will be replaced with new ID
136                userCommand.argumentString = userCommand.argumentString + "," + dragonString;
137            }
138        }
139        objectOutput.writeObject(userCommand);
140        byte[] data = byteArrayOutputStream.toByteArray();
141        datagramSocket.send(new DatagramPacket(data, data.length, address));
142        return true;
143    }
144 }
```

## Файл Client.java

```
141  /**
142   * Gets a message from a server
143   * @return message
144   * @throws IOException If problem occurred while receiving from server
145   */
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

```
1 package com.company;
2
3 import com.company.commands.Read;
4 import com.company.ui.CommandExecutor;
5 import com.company.ui.CommandReader;
6 import com.company.ui.User;
7
8 import java.io.*;
9 import java.net.InetSocketAddress;
10 import java.net.SocketAddress;
11 import java.nio.ByteBuffer;
12 import java.nio.channels.DatagramChannel;
13 import java.nio.channels.SelectionKey;
14 import java.nio.channels.Selector;
15 import java.text.SimpleDateFormat;
16 import java.util.Date;
17 import java.util.concurrent.*;
18
19 /**
20 * Main server class
21 * @see Client
22 */
23 public class Server {
24
25     private static final ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
26     private static final PrintStream serverResponseStream = new PrintStream(outputStream);
27     private static final CommandExecutor localExecutor = new CommandExecutor(CommandExecutor.allCommands
, System.out);
28     private static final CommandExecutor userExecutor = new CommandExecutor(CommandExecutor.userCommands
, serverResponseStream);
29     public static final User internalUser = new User("admin", "admin");
30
31 /**
32 * Main server function
33 *
34 * @param args filename
35 * @see CommandExecutor
36 * @see Client
37 */
38 public static void main(String[] args) {
39     System.out.println("Starting Server...");
40     System.out.println("Reading collection from file...");
41     readCollectionFromFile(args);
42     BufferedReader localInput = new BufferedReader(new InputStreamReader(System.in));
43     Selector selector = null;
44     ByteBuffer byteBuffer = ByteBuffer.allocate(65536);
45     try {
46         selector = Selector.open();
47         selector.wakeup();
48         DatagramChannel datagramChannel = DatagramChannel.open();
49         datagramChannel.bind(new InetSocketAddress("localhost", 3333));
50         datagramChannel.configureBlocking(false);
51         datagramChannel.register(selector, SelectionKey.OP_READ);
52     } catch (IOException e) {
53         System.err.println("Failed to start server.");
54         e.printStackTrace();
55         System.exit(-1);
56     }
57     ExecutorService responseExecutorService = Executors.newCachedThreadPool();
58     System.out.println("Server is active.");
59     try {
60         while (true) {
61             try {
62                 String line;
63                 if (localInput.ready())
64                     if ((line = localInput.readLine()) != null)
65                         localExecutor.executeCommand(CommandReader.readCommandFromString(
internalUser, line));
66                     selector.select(1500);
67                     selector.selectedKeys().stream().parallel().forEach(selectionKey ->
{
68                         try {
69                             if (selectionKey.isValid())
70                                 if (selectionKey.isReadable())
71
72
73
74
75
76
77
78
79
7
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
25
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
28
290
291
292
293
294
295
296
297
298
299
29
300
301
302
303
304
305
306
307
308
309
30
310
311
312
313
314
315
316
317
318
319
31
320
321
322
323
324
325
326
327
328
329
32
330
331
332
333
334
335
336
337
338
339
33
340
341
342
343
344
345
346
347
348
349
34
350
351
352
353
354
355
356
357
358
359
35
360
361
362
363
364
365
366
367
368
369
36
370
371
372
373
374
375
376
377
378
379
37
380
381
382
383
384
385
386
387
388
38
389
390
391
392
393
394
395
396
397
398
39
399
39
400
39
401
39
402
39
403
39
404
39
405
39
406
39
407
39
408
39
409
39
40
410
39
411
39
412
39
413
39
414
39
415
39
416
39
417
39
418
39
419
39
41
420
39
421
39
422
39
423
39
424
39
425
39
426
39
427
39
428
39
429
39
42
430
39
431
39
432
39
433
39
434
39
435
39
436
39
437
39
438
39
439
39
43
440
39
441
39
442
39
443
39
444
39
445
39
446
39
447
39
448
39
449
39
44
450
39
451
39
452
39
453
39
454
39
455
39
456
39
457
39
458
39
459
39
45
460
39
461
39
462
39
463
39
464
39
465
39
466
39
467
39
468
39
469
39
46
470
39
471
39
472
39
473
39
474
39
475
39
476
39
477
39
478
39
479
39
47
480
39
481
39
482
39
483
39
484
39
485
39
486
39
487
39
488
39
489
39
48
490
39
491
39
492
39
493
39
494
39
495
39
496
39
497
39
498
39
49
499
39
49
500
39
49
501
39
49
502
39
49
503
39
49
504
39
49
505
39
49
506
39
49
507
39
49
508
39
49
509
39
49
510
39
49
511
39
49
512
39
49
513
39
49
514
39
49
515
39
49
516
39
49
517
39
49
518
39
49
519
39
49
520
39
49
521
39
49
522
39
49
523
39
49
524
39
49
525
39
49
526
39
49
527
39
49
528
39
49
529
39
49
530
39
49
531
39
49
532
39
49
533
39
49
534
39
49
535
39
49
536
39
49
537
39
49
538
39
49
539
39
49
540
39
49
541
39
49
542
39
49
543
39
49
544
39
49
545
39
49
546
39
49
547
39
49
548
39
49
549
39
49
550
39
49
551
39
49
552
39
49
553
39
49
554
39
49
555
39
49
556
39
49
557
39
49
558
39
49
559
39
49
560
39
49
561
39
49
562
39
49
563
39
49
564
39
49
565
39
49
566
39
49
567
39
49
568
39
49
569
39
49
570
39
49
571
39
49
572
39
49
573
39
49
574
39
49
575
39
49
576
39
49
577
39
49
578
39
49
579
39
49
580
39
49
581
39
49
582
39
49
583
39
49
584
39
49
585
39
49
586
39
49
587
39
49
588
39
49
589
39
49
590
39
49
591
39
49
592
39
49
593
39
49
594
39
49
595
39
49
596
39
49
597
39
49
598
39
49
599
39
49
500
39
49
501
39
49
502
39
49
503
39
49
504
39
49
505
39
49
506
39
49
507
39
49
508
39
49
509
39
49
510
39
49
511
39
49
512
39
49
513
39
49
514
39
49
515
39
49
516
39
49
517
39
49
518
39
49
519
39
49
520
39
49
521
39
49
522
39
49
523
39
49
524
39
49
525
39
49
526
39
49
527
39
49
528
39
49
529
39
49
530
39
49
531
39
49
532
39
49
533
39
49
534
39
49
535
39
49
536
39
49
537
39
49
538
39
49
539
39
49
540
39
49
541
39
49
542
39
49
543
39
49
544
39
49
545
39
49
546
39
49
547
39
49
548
39
49
549
39
49
550
39
49
551
39
49
552
39
49
553
39
49
554
39
49
555
39
49
556
39
49
557
39
49
558
39
49
559
39
49
560
39
49
561
39
49
562
39
49
563
39
49
564
39
49
565
39
49
566
39
49
567
39
49
568
39
49
569
39
49
570
39
49
571
39
49
572
39
49
573
39
49
574
39
49
575
39
49
576
39
49
577
39
49
578
39
49
579
39
49
580
39
49
581
39
49
582
39
49
583
39
49
584
39
49
585
39
49
586
39
49
587
39
49
588
39
49
589
39
49
590
39
49
591
39
49
592
39
49
593
39
49
594
39
49
595
39
49
596
39
49
597
39
49
598
39
49
599
39
49
500
39
49
501
39
49
502
39
49
503
39
49
504
39
49
505
39
49
506
39
49
507
39
49
508
39
49
509
39
49
510
39
49
511
39
49
512
39
49
513
39
49
514
39
49
515
39
49
516
39
49
517
39
49
518
39
49
519
39
49
520
39
49
521
39
49
522
39
49
523
39
49
524
39
49
525
39
49
526
39
49
527
39
49
528
39
49
529
39
49
530
39
49
531
39
49
532
39
49
533
39
49
534
39
49
535
39
49
536
39
49
537
39
49
538
39
49
539
39
49
540
39
49
541
39
49
542
39
49
543
39
49
544
39
49
545
39
49
546
39
49
547
39
49
548
39
49
549
39
49
550
39
49
551
39
49
552
39
49
553
39
49
554
39
49
555
39
49
556
39
49
557
39
49
558
39
49
559
39
49
560
39
49
561
39
49
562
39
49
563
39
49
564
39
49
565
39
49
566
39
49
567
39
49
568
39
49
569
39
49
570
39
49
571
39
49
572
39
49
573
39
49
574
39
49
575
39
49
576
39
49
577
39
49
578
39
49
579
39
49
580
39
49
581
39
49
582
39
49
583
39
49
584
39
49
585
39
49
586
39
49
587
39
49
588
39
49
589
39
49
590
39
49
591
39
49
592
39
49
593
39
49
594
39
49
595
39
49
596
39
49
597
39
49
598
39
49
599
39
49
500
39
49
501
39
49
502
39
49
503
39
49
504
39
49
505
39
49
506
39
49
507
39
49
508
39
49
509
39
49
510
39
49
511
39
49
512
39
49
513
39
49
514
39
49
515
39
49
516
39
49
517
39
49
518
39
49
519
39
49
520
39
49
521
39
49
522
39
49
523
39
49
524
39
49
525
39
49
526
39
49
527
39
49
528
39
49
529
39
49
530
39
49
531
39
49
532
39
49
533
39
49
534
39
49
535
39
49
536
39
49
537
39
49
538
39
49
539
39
49
540
39
49
541
39
49
542
39
49
543
39
49
544
39
49
545
39
49
546
39
49
547
39
49
548
39
49
549
39
49
550
39
49
551
39
49
552
39
49
553
39
49
554
39
49
555
39
49
556
39
49
557
39
49
558
39
49
559
39
49
560
39
49
561
39
49
562
39
49
563
39
49
564
39
49
565
39
49
566
39
49
567
39
49
568
39
49
569
39
49
570
39
49
571
39
49
572
39
49
573
39
49
574
39
49
575
39
49
576
39
49
577
39
49
578
39
49
579
39
49
580
39
49
581
39
49
582
39
49
583
39
49
584
39
49
585
39
49
586
39
49
587
39
49
588
39
49
589
39
49
590
39
49
591
39
49
592
39
49
593
39
49
594
39
49
595
39
49
596
39
49
597
39
49
598
39
49
599
39
49
500
39
49
501
39
49
502
39
49
503
39
49
504
39
49
505
39
49
506
39
49
507
39
49
508
39
49
509
39
49
510
39
49
511
39
49
512
39
49
513
39
49
514
39
49
515
39
49
516
39
49
517
39
49
518
39
49
519
39
49
520
39
49
521
39
49
522
39
49
523
39
49
524
39
49
525
39
49
526
39
49
527
39
49
528
39
49
529
39
49
530
39
49
531
39
49
532
39
49
533
39
49
534
39
49
535
39
49
536
39
49
537
39
49
538
39
49
539
39
49
540
39
49
541
39
49
542
39
49
543
39
49
544
39
49
545
39
49
546
39
49
547
39
49
548
39
49
549
39
49
550
39
49
551
39
49
552
39
49
553
39
49
554
39
49
555
39
49
556
39
49
557
39
49
558
39
49
559
39
49
560
39
49
561
39
49
562
39
49
563
39
49
564
39
49
565
39
49
566
39
49
567
39
49
568
39
49
569
39
49
570
39
49
571
39
49
572
39
49
573
39
49
574
39
49
575
39
49
576
39
49
577
39
49
578
39
49
579
39
49
580
39
49
581
39
49
582
39
49
583
39
49
584
39
49
585
39
49
586
39
49
587
39
49
588
39
49
589
39
49
590
39
49
591
39
49
592
39
49
593
39
49
594
39
49
595
39
49
596
39
49
597
39
49
598
39
49
599
39
49
500
39
49
501
39
49
502
39
49
503
39
49
504
39
49
505
39
49
506
39
49
507
39
49
508
39
49
509
39
49
510
39
49
511
39
49
512
39
49
513
39
49
514
39
49
515
39
49
516
39
49
517
39
49
518
39
49
519
39
49
520
39
49
521
39
49
522
39
49
523
39
49
524
39
49
525
39
49
526
39
49
527
39
49
528
39
49
529
39
49
530
39
49
531
39
49
532
39
49
533
39
49
534
39
49
535
39
49
536
39
49
537
39
49
538
39
49
539
39
49
540
39
49
541
39
49
542
39
49
543
39
49
544
39
49
545
39
49
546
39
49
547
39
49
548
39
49
549
39
49
550
39
49
551
39
49
552
39
49
553
39
49
554
39
49
555
39
49
556
39
49
557
39
49
558
39
49
559
39
49
560
39
49
561
39
49
562
39
49
563
39
49
564
39
49
565
39
49
566
39
49
567
39
49
568
39
49
569
39
49
570
39
49
571
39
49
572
39
49
573
39
49
574
39
49
575
39
49
576
39
49
577
39
49
578
39
49
579
39
49
580
39
49
581
39
49
582
39
49
583
39
49
584
39
49
585
39
49
586
39
49
587
39
49
588
39
49
589
39
49
590
39
49
591
39
49
592
39
49
593
39
49
594
39
49
595
39
49
596
39
49
597
39
49
598
39
49
599
39
49
500
39
49
501
39
49
502
39
49
503
39
49
504
39
49
505
39
49
506
39
49
507
39
49
508
39
49
509
39
49
510
39
49
511
39
49
512
39
49
513
39
49
514
39
49
515
39
49
516
39
49
517
39
49
518
39
49
519
39
49
520
39
49
521
39
49
522
39
49
523
39
49
524
39
49
525
39
49
526
39
49
527
39
49
528
39
49
529
39
49
530
39
49
531
39
49
532
39
49
533
39
49
534
39
49
535
39
49
536
39
49
537
39
49
538
39
49
539
39
49
540
39
49
541
39
49
542
39
49
543
39
49
544
39
49
545
39
49
546
39
49
547
39
49
548
39
49
549
39
49
550
39
49
551
39
49
552
39
49
553
39
49
554
39
49
555
39
49
556
39
49
557
39
49
558
39
49
559
39
49
560
39
49
561
39
49
562
39
49
563
39
49
564
39
49
565
39
49
566
39
49
567
39
49
568
39
49
569
39
49
570
39
49
571
39
49
572
39
49
573
39
49
574
39
49
575
39
49
576
39
49
577
39
49
578
39
49
579
39
49
580
39
49
581
39
49
582
39
49
583
39
49
584
39
49
585
39
49
586
39
49
587
39
49
588
39
49
589
39
49
590
39
49
591
39
49
592
39
49
593
39
49
594
39
49
595
39
49
596
39
49
597
39
49
598
39
49
599
39
49
500
39
49
501
39
49
502
39
49
503
39
49
504
39
49
505
39
49
506
39
49
507
39
49
508
39
49
509
39
49
510
39
49
511
39
49
512
39
49
513
39
49
514
39
49
515
39
49
516
39
49
517
39
49
518
39
49
519
39
49
520
39
49
521
39
49
522
39
49
523
39
49
524
39
49
525
39
49
526
39
49
527
39
49
528
39
49
529
39
49
530
39
49
531
39
49
532
39
49
533
39
49
534
39
49
535
39
49
536
39
49
537
39
49
538
39
49
539
39
49
540
39
49
541
39
49
542
39
49
543
39
49
544
39
49
545
39
49
546
39
49
547
39
49
548
39
49
549
39
49
550
39
49
551
39
49
552
39
49
553
39
49
554
39
49
555
39
49
556
39
49
557
39
49
558
39
49
559
39
49
560
39
49
561
39
49
562
39
49
563
39
49
564
39
49
565
39
49
566
39
49
567
39
49
568
39
49
569
39
49
570
39
49
571
39
49
572
39
49
573
39
49
574
39
49
575
39
49
576
39
49
577
39
49
578
39
49
579
39
49
580
39
49
581
39
49
582
39
49
583
39
49
584
39
49
585
39
49
586
39
49
587
39
49
588
39
49
589
39
49
590
39
49
591
39
49
592
39
49
593
39
49
594
39
49
595
39
49
596
39
49
597
39
49
598
39
49
599
39
49
500
39
49
501
39
49
502
39
49
503
39
49
504
39
49
505
39
49
506
39
49
507
39
49
508
39
49
509
39
49
510
39
49
511
39
49
512
39
49
513
39
49
514
39
49
515
39
49
516
39
49
517
39
49
518
39
49
519
39
49
520
39
49
521
39
49
522
39
49
523
39
49
524
39
49
525
39
49
526
39
49
527
39
49
528
39
49
529
39
49
530
39
49
531
39
49
532
39
49
533
39
49
534
39
49
535
39
49
536
39
49
537
39
49
538
39
49
539
39
49
540
39
49
541
39
49
542
39
49
543
39
49
544
39
49
545
39
49
546
39
49
547
39
49
548
39
49
549
39
49
550
39
49
551
39
49
552
39
49
553
39
49
554
39
49
555
39
49
556
39
49
557
39
49
558
39
49
559
39
49
560
39
49
561
39
49
562
39
49
563
39
49
564
39
49
565
39
49
566
39
49
567
39
49
568
39
49
569
39
49
570
39
49
571
39
49
572
39
49
573
39
49
574
39
49
575
39
49
576
39
49
577
39
49
578
39
49
579
39
49
580
39
49
581
39
49
582
39
49
583
39
49
584
39
49
585
39
49
586
39
49
587
39
49
588
39
49
589
39
49
590
39
49
591
39
49
592
39
49
593
39
49
594
39
49
595
39
49
596
39
49
597
39
49
598
39
49
599
39
49
500
39
49
501
39
49
502
39
49
503
39
49
504
39
49
505
39
49
506
39
49
507
39
49
508
39
49
509
39
49
510
39
49
511
39
49
512
39
49
513
39
49
514
39
49
515
39
49
516
39
49
517
39
49
518
39
49
519
39
49
520
39
49
521
39
49
522
39
49
523
39
49
524
39
49
525
39
49
526
39
49
527
39
49
528
39
49
529
39
49
530
39
49
531
39
49
532
39
49
533
39
49
534
39
49
535
39
49
536
39
49
537
39
49
538
39
49
539
39
49
540
39
49
541
39
49
542
39
49
543
39
49
544
39
49
545
39
49
546
39
49
547
39
49
548
39
49
549
39
49
550
39
49
551
39
49
552
39
49
553
39
49
554
39
49
555
39
49
556
39
49
557
39
49
558
39
49
559
39
49
560
39
49
561
39
49
562
39
49
563
39
49
564
39
49
565
39
49
5
```

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

## 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

```
1 package com.company.ui;
2
3 import java.io.Serializable;
4 import java.security.MessageDigest;
5 import java.security.NoSuchAlgorithmException;
6 import java.util.Arrays;
7 import java.util.Objects;
8
9 /**
10 * Class that is needed to store users
11 */
12 public class User implements Serializable {
13
14     private final String username;
15     private final byte[] passwordMD5;
16
17     /**
18      * Username-only constructor
19      * @param username username
20      */
21     public User(String username){
22         this(username,"");
23     }
24
25     /**
26      * Default constructor
27      * @param username username
28      * @param password password
29      */
30     public User(String username, String password) {
31         this.username = username;
32         byte[] finalPasswordMD5;
33         try {
34             finalPasswordMD5 = MessageDigest.getInstance("MD5").digest(password.getBytes());
35         } catch (NoSuchAlgorithmException e) {
36             finalPasswordMD5 = new byte[0];
37             e.printStackTrace();
38         }
39         this.passwordMD5 = finalPasswordMD5;
40     }
41
42     @Override
43     public boolean equals(Object o) {
44         if (this == o) return true;
45         if (!(o instanceof User)) return false;
46         User user = (User) o;
47         return username.equals(user.username) && Arrays.equals(passwordMD5, user.passwordMD5);
48     }
49
50     @Override
51     public int hashCode() {
52         int result = Objects.hash(username);
53         result = 31 * result + Arrays.hashCode(passwordMD5);
54         return result;
55     }
56
57     /**
58      * Username getter
59      * @return username
60      */
61     public String getUsername() {
62         return username;
63     }
64
65     /**
66      * Password getter
67      * @return password in MD5 format
68      */
69     public byte[] getPasswordMD5() {
70         return passwordMD5;
71     }
72
73     @Override
74     public String toString() {
```

## Файл User.java

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

## Файл CommandReader.java

```
1 package com.company.ui;
2
3 import com.company.Server;
4
5 import java.io.BufferedReader;
6 import java.io.IOException;
7 import java.io.InputStreamReader;
8 import java.io.Serializable;
9
10 /**
11 * Class designed to get commands from buffered readers and strings
12 *
13 * @see CommandExecutor
14 */
15 public class CommandReader {
16
17     BufferedReader bufferedReader;
18
19     /**
20      * Command reader constructor
21      *
22      * @param bufferedReader buffered reader to get commands from
23      */
24     public CommandReader(BufferedReader bufferedReader) {
25         this.bufferedReader = bufferedReader;
26     }
27
28     /**
29      * Get an array of strings from System.in (separated by spaces)
30      *
31      * @return Array of strings
32      */
33     public static String[] getStringsFromTerminal() {
34         return new CommandReader(new BufferedReader(new InputStreamReader(System.in))).getStringFromBufferedReader().split(" ");
35     }
36
37     /**
38      * Get a command from string
39      *
40      * @param user User
41      * @param singleString string to parse from
42      * @return Command
43      */
44     public static Command readCommandFromString(User user, String singleString) {
45         return (readCommandFromString(user,singleString.split(" ", 2)));
46     }
47
48     /**
49      * Get a command from strings
50      *
51      * @param user User
52      * @param input strings to parse from
53      * @return Command
54      */
55     public static Command readCommandFromString(User user, String[] input) {
56         if (input.length != 0) {
57             input[0] = input[0].toLowerCase();
58             if (input.length > 1)
59                 return new Command(user,input[0], input[1]);
60             else
61                 return new Command(user,input[0]);
62         } else return new Command(user);
63     }
64
65     /**
66      * Gets a string from buffered reader line
67      *
68      * @return Received string
69      */
70     public String getStringFromBufferedReader() {
71         try {
72             return bufferedReader.readLine();
73         } catch (IOException e) {
```

## Файл CommandReader.java

```
74         System.out.println("Error: " + e.getMessage() + ".");
75     }
76 }
77 */
78 /**
79  * Get a command from Buffered Reader
80  *
81  * @param user User
82  * @return Command
83  */
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

```
1 package com.company.ui;
2
3 import com.company.commands.*;
4
5 import java.io.*;
6 import java.util.ArrayList;
7 import java.util.Arrays;
8 import java.util.Collections;
9 import java.util.List;
10 import java.util.concurrent.atomic.AtomicReference;
11
12 /**
13 * Main command execution runnable
14 */
15 public class CommandExecutor {
16
17     public static final List<User> registeredUsers = Collections.synchronizedList(new ArrayList<>());
18
19     /**
20      * All possible commands
21      */
22     public static final CommandAction[] allCommands = {
23         new Help(),
24         new Info(),
25         new Show(),
26         new Insert(),
27         new UpdateID(),
28         new RemoveKey(),
29         new Clear(),
30         new Save(),
31         new Execute_script(),
32         new Exit(),
33         new ReplaceIfGreaterAge(),
34         new RemoveGreaterKey(),
35         new RemoveLowerKey(),
36         new RemoveAllAge(),
37         new FilterLessThanType(),
38         new PrintDescending(),
39         new Read(),
40         new CsvInsert(),
41         new CsvUpdateID(),
42         new CsvReplaceIfGreaterAge(),
43         new Register()
44     };
45     /**
46      * Commands that user can use
47      */
48     public static final CommandAction[] userCommands = {
49         new Help(),
50         new Info(),
51         new Show(),
52         new Insert(),
53         new UpdateID(),
54         new RemoveKey(),
55         new Clear(),
56         new Execute_script(),
57         new Exit(),
58         new ReplaceIfGreaterAge(),
59         new RemoveGreaterKey(),
60         new RemoveLowerKey(),
61         new RemoveAllAge(),
62         new FilterLessThanType(),
63         new PrintDescending(),
64         new Register()
65         //new Read(),
66         //new CsvInsert(),
67         //new CsvUpdateID(),
68         //new CsvReplaceIfGreaterAge()
69     };
70     /**
71      * Programs that execute_script can use
72      */
73     public static final CommandAction[] scriptCommands = {
74         new Help(),
```

## Файл CommandExecutor.java

```
75      new Info(),
76      new Show(),
77      //new Insert(),
78      //new UpdateID(),
79      new RemoveKey(),
80      new Clear(),
81      new Save(),
82      //new Execute_script(),
83      new Exit(),
84      //new ReplaceIfGreaterAge(),
85      new RemoveGreaterKey(),
86      new RemoveLowerKey(),
87      new RemoveAllAge(),
88      new FilterLessThanType(),
89      new PrintDescending(),
90      //new Read(),
91      new CsvInsert(),
92      new CsvUpdateID(),
93      new CsvReplaceIfGreaterAge()
94  };
95  private static File file = new File("C:\\\\Users\\\\muram\\\\IdeaProjects\\\\Lab5\\\\file.csv");
96  private final PrintStream printStream;
97  private final CommandAction[] availableCommands;
98
99 /**
100  * User runnable constructor
101  *
102  * @param availableCommands set of available commands
103  * @param printStream      PrintStream to output responses to
104  */
105 public CommandExecutor(CommandAction[] availableCommands, PrintStream printStream) {
106     this.availableCommands = availableCommands;
107     this.printStream = printStream;
108 }
109
110 /**
111  * Get file specified by command line argument
112  *
113  * @return File
114  */
115 public static File getFile() {
116     return file;
117 }
118
119 /**
120  * Set file to save/read collection from
121  *
122  * @param fileName File name
123  */
124 public static void setFile(String fileName) {
125     file = new File(fileName);
126 }
127
128 /**
129  * Execute specified user command
130  *
131  * @param command User command
132  */
133 public void executeCommand(CommandReader.Command command) {
134     AtomicReference<String> response = new AtomicReference<>("Command gave no response.");
135     if(registeredUsers.stream().noneMatch(user -> user.equals(command.user)) && !command.
136     commandString.equals("register")) {
137         response.set("Unauthorized access denied.");
138     } else {
139         if (Arrays.stream(availableCommands).parallel().noneMatch(availableCommand ->
140             availableCommand.getLabel().equals(command.commandString))) {
141             response.set("Unknown command \\\" " + command.commandString + "\\\". try \\\"help\\\" for list of
142             commands");
143         } else
144             try {
145                 Arrays.stream(availableCommands).forEach(availableCommand -> {
146                     if (command.commandString.equals(availableCommand.getLabel()))
147                         response.set(availableCommand.execute(command.user, command.argumentString));
148                 });
149             }
```

## 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

```
1 package com.company.commands;
2
3 import com.company.ui.User;
4
5 public class Exit implements CommandAction {
6
7     @Override
8     public String getLabel() {
9         return "exit";
10    }
11
12    public String getDescription() {
13        return "Exit the program (without saving).";
14    }
15
16    public String execute(User commandedUser, String argument) {
17        System.exit(0);
18        return "Exited.";
19    }
20}
21
```

## Файл Help.java

```
1 package com.company.commands;
2
3 import com.company.ui.CommandExecutor;
4 import com.company.ui.User;
5
6 import java.util.Arrays;
7
8 public class Help implements CommandAction {
9     String response;
10
11     public String getLabel() {
12         return "help";
13     }
14
15     public String getDescription() {
16         return "Gives the list of available commands.";
17     }
18
19     public String execute(User commandedUser, String argument) {
20         response = "Available commands:\n";
21         Arrays.stream(CommandExecutor.userCommands).forEach(command -> response += command.getLabel() +
22             " " + command.getArgumentLabel() + ": " + command.getDescription() + "\n");
23         response += "Collection class members have to be entered line-by-line. Standard types (including
24         primitive types) have to be entered in the same line as the command.";
25     }
26 }
```

## Файл Info.java

```
1 package com.company.commands;
2
3 import com.company.storables.DragonHolder;
4 import com.company.ui.User;
5
6 public class Info implements CommandAction {
7     public String getLabel() {
8         return "info";
9     }
10
11    public String getDescription() {
12        return "Gives the info about collection.";
13    }
14
15    public String execute(User commandedUser, String argument) {
16        return "Dragon collection, initialization date: " + DragonHolder.getInitializationDate() + ",  
number of elements: " + DragonHolder.getCollection().size() + ".";
17    }
18 }
19
```

## Файл Read.java

```
1 package com.company.commands;
2
3 import com.company.storables.Dragon;
4 import com.company.storables.DragonHolder;
5 import com.company.ui.CommandExecutor;
6 import com.company.ui.User;
7
8 import java.io.*;
9 import java.util.Arrays;
10
11 public class Read implements CommandAction {
12
13     String response;
14
15     @Override
16     public String getLabel() {
17         return "read";
18     }
19
20     @Override
21     public String getDescription() {
22         return "Clear collection and read from file (its name is specified by command line argument).";
23     }
24
25     @Override
26     public String execute(User commandedUser, String argument) {
27         response = "";
28         File file = CommandExecutor.getFile();
29         try {
30             BufferedInputStream fileReader = new BufferedInputStream(new FileInputStream(file));
31             StringBuilder stringBuilder = new StringBuilder();
32             while (fileReader.available() > 0)
33                 stringBuilder.append((char) fileReader.read());
34             DragonHolder.getCollection().clear();
35             Arrays.stream(stringBuilder.toString().split("[\\r\\n]+"))
36                 .forEach(line -> {
37                     if (!line.isBlank())
38                         try {
39                             String[] splitLine = line.split(",", 2);
40                             if (splitLine.length < 2)
41                                 throw new IllegalArgumentException("Invalid input string: '" + line
42 + ".");
43                             Dragon(splitLine[1]);
44                         } catch (NumberFormatException e) {
45                             throw new IllegalArgumentException("Can't parse key from " +
46                             splitLine[0] + ".");
47                         }
48                     response += "Error reading from CSV line " + line + ": " + e.getMessage
49 () + ".\n";
50                 });
51             fileReader.close();
52         } catch (FileNotFoundException e) {
53             throw new IllegalArgumentException("Can't find file '" + file + ".");
54         } catch (SecurityException e) {
55             throw new IllegalArgumentException("Can't access file '" + file + ".");
56         } catch (IOException e) {
57             throw new IllegalArgumentException("Error occurred accessing file '" + file + ".");
58         }
59         response += "Read collection from file '" + file + ".";
60     }
61 }
62 }
63 }
```

## Файл Save.java

```
1 package com.company.commands;
2
3 import com.company.storables.DragonHolder;
4 import com.company.ui.CommandExecutor;
5 import com.company.ui.User;
6
7 import java.io.*;
8
9 public class Save implements CommandAction {
10
11     @Override
12     public String getLabel() {
13         return "save";
14     }
15
16     @Override
17     public String getDescription() {
18         return "Saves collection to file (its name is specified by command line argument).";
19     }
20
21     @Override
22     public String execute(User commandedUser, String argument) {
23         File file = CommandExecutor.getFile();
24         try {
25             file.createNewFile();
26             OutputStreamWriter fileWriter = new OutputStreamWriter(new FileOutputStream(file));
27             DragonHolder.getCollection().forEach((key, value) -> {
28                 try {
29                     fileWriter.write(key + "," + value.toCsvString() + "\n");
30                 } catch (IOException e) {
31                     throw new IllegalArgumentException("Error occurred writing collection to file '" +
32                     file + ".\"");
33                 }
34             });
35             fileWriter.close();
36         } catch (FileNotFoundException e) {
37             throw new IllegalArgumentException("Can't find file '" + file + ".\"");
38         } catch (SecurityException e) {
39             throw new IllegalArgumentException("Can't access file '" + file + ".\"");
40         } catch (IOException e) {
41             throw new IllegalArgumentException("Error occurred accessing file '" + file + ".\"");
42         }
43         return "Saved collection to file '" + file + ".";
44     }
45 }
```

## Файл Show.java

```
1 package com.company.commands;
2
3 import com.company.storables.Dragon;
4 import com.company.storables.DragonHolder;
5 import com.company.ui.User;
6
7 import java.util.Comparator;
8
9 public class Show implements CommandAction {
10     String response;
11
12     public String getLabel() {
13         return "show";
14     }
15
16     public String getDescription() {
17         return "Show all collection elements.";
18     }
19
20     public String execute(User commandedUser, String argument) {
21         response = "Collection:\n";
22         DragonHolder.getCollection().values().stream().sorted(Comparator.comparing(Dragon::getName)).
23         forEach(element -> response += element.toString() + "\n");
24         return response.substring(0, response.length() - 1);
25     }
26 }
```

## Файл Clear.java

```
1 package com.company.commands;
2
3 import com.company.storables.DragonHolder;
4 import com.company.ui.User;
5
6 public class Clear implements CommandAction {
7     public String getLabel() {
8         return "clear";
9     }
10
11    public String getDescription() {
12        return "Clear the collection.";
13    }
14
15    @Override
16    public String execute(User commandedUser, String argument) {
17        DragonHolder.getCollection().clear();
18        return "Collection cleared.";
19    }
20 }
21
```

## Файл Insert.java

```
1 package com.company.commands;
2
3 import com.company.storables.DragonHolder;
4 import com.company.storables.DragonUtils;
5 import com.company.ui.User;
6
7 public class Insert implements CommandAction {
8     public String getLabel() {
9         return "insert";
10    }
11
12    public String getArgumentLabel() {
13        return "{key} {element}";
14    }
15
16    public String getDescription() {
17        return "Insert new {element} to collection with a {key}." ;
18    }
19
20    public String execute(User commandedUser, String argument) {
21        if (argument == null || argument.isEmpty())
22            throw new IllegalArgumentException("Please specify Dragon key.");
23        try {
24            DragonHolder.getCollection().put(Integer.parseInt(argument), DragonUtils.
25                inputNewDragonFromConsole(commandedUser));
26        } catch (IllegalArgumentException e) {
27            throw new IllegalArgumentException("Illegal key: " + e.getMessage() + ".");
28        }
29        return "Insert successful.";
30    }
31}
```

## Файл Register.java

```
1 package com.company.commands;
2
3 import com.company.ui.CommandExecutor;
4 import com.company.ui.User;
5
6 import java.util.Arrays;
7 import java.util.List;
8
9 public class Register implements CommandAction{
10
11     @Override
12     public String getLabel() {
13         return "register";
14     }
15
16     @Override
17     public String getArgumentLabel() {
18         return "{username} [password]";
19     }
20
21     @Override
22     public String getDescription() {
23         return "Register a new user";
24     }
25
26     @Override
27     public String execute(User commandedUser, String argument) {
28         String[] arguments = argument.split(" ",2);
29         User newUser;
30         if(arguments.length == 2)
31             newUser = new User(arguments[0],arguments[1]);
32         else if(arguments.length == 1)
33             newUser = new User(arguments[0]);
34         else
35             return "Failed to register. Did you input valid username and password? Try typing register {username} [password] to try again.";
36         if(CommandExecutor.registeredUsers.stream().anyMatch(newUser::equals))
37             return "Logged in successfully.";
38         if (CommandExecutor.registeredUsers.stream().anyMatch(user -> user.getUsername().equals(newUser.getUsername()) && !Arrays.equals(user.getPasswordMD5(), newUser.getPasswordMD5())))
39             return "Wrong password. Further commands will not be accepted. Try typing register {username} [password] to try again.";
40         CommandExecutor.registeredUsers.add(newUser);
41         return "Registered new user successfully.";
42     }
43 }
44 }
```

## Файл UpdateID.java

```
1 package com.company.commands;
2
3 import com.company.storables.DragonHolder;
4 import com.company.storables.DragonUtils;
5 import com.company.ui.User;
6
7 import java.util.Date;
8 import java.util.concurrent.atomic.AtomicBoolean;
9 import java.util.concurrent.atomic.AtomicReference;
10
11 public class UpdateID implements CommandAction {
12     @Override
13     public String getLabel() {
14         return "update";
15     }
16
17     @Override
18     public String getArgumentLabel() {
19         return "{id} {element}";
20     }
21
22     @Override
23     public String getDescription() {
24         return "Update {element} of collection by its {id}.";
25     }
26
27     @Override
28     public String execute(User commandedUser, String argument) {
29         long id;
30         try {
31             id = Long.parseLong(argument);
32         } catch (NumberFormatException e) {
33             throw new IllegalArgumentException("ID argument \"\" + argument + "\" is not an integer.");
34         }
35         AtomicBoolean found = new AtomicBoolean(false);
36         AtomicReference<Integer> dragonKey = new AtomicReference<>();
37         AtomicReference<Long> dragonId = new AtomicReference<>();
38         AtomicReference<Date> dragonCreationDate = new AtomicReference<>();
39         DragonHolder.getCollection().forEach((key, value) -> {
40             if (value.getId() == id) {
41                 dragonKey.set(key);
42                 dragonId.set(value.getId());
43                 dragonCreationDate.set(value.getCreationDate());
44                 found.set(true);
45             }
46         });
47         if (!found.get()) throw new IllegalArgumentException("Dragon with id '" + argument + "' not found");
48         else
49             if (!DragonHolder.getCollection().get(dragonKey.get()).getOwner().equals(commandedUser.getUsername()))
50                 throw new IllegalArgumentException("Unauthorized Dragon access with id " + dragonId.get() + ".");
51             else
52                 DragonHolder.getCollection().put(dragonKey.get(), DragonUtils.inputDragonFromConsole(
53                     commandedUser, dragonId.get(), dragonCreationDate.get()));
53         return "Update successful.";
54     }
55 }
```

## Файл CsvInsert.java

```
1 package com.company.commands;
2
3 import com.company.storables.Dragon;
4 import com.company.storables.DragonHolder;
5 import com.company.ui.User;
6
7 public class CsvInsert implements CommandAction {
8     public String getLabel() {
9         return "insert_csv";
10    }
11
12    public String getArgumentLabel() {
13        return "{key},{Dragon,in,csv,style}";
14    }
15
16    public String getDescription() {
17        return "Insert new {Dragon} to collection with a {key} (CSV style).";
18    }
19
20    public String execute(User commandedUser, String argument) {
21        if (!argument.isBlank()) {
22            String[] splitLine = argument.split(",", 2);
23            if (splitLine.length < 2)
24                throw new IllegalArgumentException("Invalid key,dragon input string: \'" + argument + "\'." );
25            try {
26                DragonHolder.getCollection().put(Integer.parseInt(splitLine[0]), new Dragon(splitLine[1]));
27            } catch (NumberFormatException e) {
28                throw new IllegalArgumentException("Can't parse key from " + splitLine[0] + ".");
29            }
30        } else {
31            throw new IllegalArgumentException("Please specify key,dragon.");
32        }
33        return "Insert successful.";
34    }
35}
36
```

## Файл RemoveKey.java

```
1 package com.company.commands;
2
3 import com.company.storables.Dragon;
4 import com.company.storables.DragonHolder;
5 import com.company.ui.User;
6
7 public class RemoveKey implements CommandAction {
8     @Override
9     public String getLabel() {
10         return "remove_key";
11     }
12
13     @Override
14     public String getArgumentLabel() {
15         return "{key}";
16     }
17
18     @Override
19     public String getDescription() {
20         return "Removes element from collection by its {key}";
21     }
22
23     @Override
24     public String execute(User commandedUser, String argument) {
25         int key;
26         if (argument == null || argument.isEmpty())
27             throw new IllegalArgumentException("Please specify Dragon key.");
28         try {
29             key = Integer.parseInt(argument);
30         } catch (IllegalArgumentException e) {
31             throw new IllegalArgumentException("Illegal key: " + e.getMessage() + ".");
32         }
33         if(DragonHolder.getCollection().get(key) != null)
34             if (!DragonHolder.getCollection().get(key).getOwner().equals(commandedUser.getUsername()))
35                 throw new IllegalArgumentException("Unauthorized Dragon access with id " + DragonHolder.
getCollection().get(key).getId() + ".");
36         Dragon removed = DragonHolder.getCollection().remove(key);
37         if (removed == null)
38             throw new IllegalArgumentException("No Dragon found with key \'" + key + "\'." );
39         return "Remove successful.";
40     }
41 }
42 }
```

## Файл CsvUpdateID.java

```
1 package com.company.commands;
2
3 import com.company.storables.Dragon;
4 import com.company.storables.DragonHolder;
5 import com.company.ui.User;
6
7 import java.util.Date;
8 import java.util.concurrent.atomic.AtomicBoolean;
9 import java.util.concurrent.atomic.AtomicReference;
10
11 public class CsvUpdateID implements CommandAction {
12     @Override
13     public String getLabel() {
14         return "update_csv";
15     }
16
17     @Override
18     public String getArgumentLabel() {
19         return "{Dragon,in,csv,style}";
20     }
21
22     @Override
23     public String getDescription() {
24         return "Update {element} of collection by its id (CSV version).";
25     }
26
27     @Override
28     public String execute(User commandedUser, String argument) {
29         Dragon newDragon = new Dragon(argument);
30         AtomicBoolean found = new AtomicBoolean(false);
31         AtomicReference<Integer> dragonKey = new AtomicReference<>();
32         AtomicReference<Long> dragonId = new AtomicReference<>();
33         AtomicReference<Date> dragonCreationDate = new AtomicReference<>();
34         DragonHolder.getCollection().forEach((key, value) -> {
35             if (value.getId() == newDragon.getId()) {
36                 if (!value.getOwner().equals(commandedUser.getUsername()))
37                     throw new IllegalArgumentException("Unauthorized Dragon access with id " + value.
38                         getId() +".");
39                 dragonKey.set(key);
40                 dragonId.set(value.getId());
41                 dragonCreationDate.set(value.getCreationDate());
42                 found.set(true);
43             }
44         });
45         if (!found.get()) throw new IllegalArgumentException("Dragon with id '" + argument + "' not
46         found");
47         else {
48             newDragon.setCreationDate(dragonCreationDate.get());
49             DragonHolder.getCollection().put(dragonKey.get(), newDragon);
50         }
51     }
52 }
```

## Файл RemoveAllAge.java

```
1 package com.company.commands;
2
3 import com.company.storables.DragonHolder;
4 import com.company.ui.User;
5
6 import java.util.LinkedList;
7
8 public class RemoveAllAge implements CommandAction {
9     @Override
10    public String getLabel() {
11        return "remove_all_by_age";
12    }
13
14    @Override
15    public String getArgumentLabel() {
16        return "{age}";
17    }
18
19    @Override
20    public String getDescription() {
21        return "Remove all Dragons {age} years old.";
22    }
23
24    @Override
25    public String execute(User commandedUser, String argument) {
26        long age;
27        if (argument == null || argument.isEmpty())
28            throw new IllegalArgumentException("Please specify Dragon age.");
29        try {
30            age = Long.parseLong(argument);
31        } catch (IllegalArgumentException e) {
32            throw new IllegalArgumentException("Illegal age: " + e.getMessage() + ".");
33        }
34        LinkedList<Integer> toRemove = new LinkedList<>();
35        DragonHolder.getCollection().forEach((key, dragon) -> {
36            if (dragon.getAge() == age && dragon.getOwner().equals(commandedUser.getUsername()))
37                toRemove.add(key);
38        });
39        toRemove.forEach(key -> DragonHolder.getCollection().remove(key));
40    }
41 }
42 }
```

## Файл CommandAction.java

```
1 package com.company.commands;
2
3 import com.company.ui.User;
4
5 /**
6  * Command that every program action implements
7 */
8 public interface CommandAction {
9
10    /**
11     * Command label
12     *
13     * @return string that user has to input to use the command
14     */
15    String getLabel();
16
17    /**
18     * Command argument format
19     *
20     * @return argument format
21     */
22    default String getArgumentLabel() {
23        return "";
24    }
25
26    /**
27     * Command description
28     *
29     * @return description
30     */
31    String getDescription();
32
33    /**
34     * Execute the command
35     *
36     * @param commandedUser User who issued the command
37     * @param arguments command arguments
38     * @return Command execution result
39     */
40    String execute(User commandedUser, String argument);
41
42 }
43 }
```

## Файл Execute\_script.java

```
1 package com.company.commands;
2
3 import com.company.Server;
4 import com.company.ui.CommandExecutor;
5 import com.company.ui.CommandReader;
6 import com.company.ui.User;
7
8 import java.io.*;
9 import java.nio.charset.Charset;
10 import java.util.Arrays;
11
12 public class Execute_script implements CommandAction {
13     @Override
14     public String getLabel() {
15         return "execute_script";
16     }
17
18     @Override
19     public String getArgumentLabel() {
20         return "file_name";
21     }
22
23     @Override
24     public String getDescription() {
25         return "executes script from \"file_name\"";
26     }
27
28     @Override
29     public String execute(User commandedUser, String argument) {
30         ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
31         PrintStream printStream = new PrintStream(outputStream);
32         File file;
33         try {
34             file = new File(argument);
35         } catch (NullPointerException e) {
36             throw new IllegalArgumentException("Please specify filename.");
37         }
38         if (argument.isBlank())
39             throw new IllegalArgumentException("Please specify filename.");
40         try {
41             if (!file.canRead())
42                 throw new IllegalArgumentException("Can't read file '" + argument + "'.");
43         } catch (SecurityException e) {
44             throw new IllegalArgumentException("Can't access file '" + argument + "'.");
45         }
46         try {
47             BufferedInputStream fileReader = new BufferedInputStream(new FileInputStream(file));
48             CommandExecutor commandExecutor = new CommandExecutor(CommandExecutor.scriptCommands,
49             printStream);
50             StringBuilder stringBuilder = new StringBuilder();
51             while (fileReader.available() > 0)
52                 stringBuilder.append((char) fileReader.read());
53             Arrays.stream(stringBuilder.toString().split("[\\r\\n]+"))
54                 .forEach(line -> {
55                     if (!line.isBlank())
56                         printStream.append(line).append("\n");
57                     String formattedLine = line
58                         .replaceAll("\\breplace_if_greater\\b", "replace_if_greater_csv")
59                         .replaceAll("\\bupdate\\b", "update_csv")
60                         .replaceAll("\\binsert\\b", "insert_csv");
61                     commandExecutor.executeCommand(CommandReader.readCommandFromString(Server.
internalUser, formattedLine));
62                 });
63             fileReader.close();
64         } catch (FileNotFoundException e) {
65             throw new IllegalArgumentException("Can't find file '" + file + "'.");
66         } catch (SecurityException e) {
67             throw new IllegalArgumentException("Can't access file '" + file + "'.");
68         } catch (IOException e) {
69             throw new IllegalArgumentException("Error occurred accessing file '" + file + "'.");
70         }
71         printStream.append("Executed script from file \"").append(argument).append(".");
72     }
}
```

## Файл Execute\_script.java

```
73 }  
74
```

## Файл RemoveLowerKey.java

```
1 package com.company.commands;
2
3 import com.company.storables.DragonHolder;
4 import com.company.ui.User;
5
6 import java.util.LinkedList;
7
8 public class RemoveLowerKey implements CommandAction {
9     @Override
10    public String getLabel() {
11        return "remove_lower_key";
12    }
13
14    @Override
15    public String getArgumentLabel() {
16        return "{key}";
17    }
18
19    @Override
20    public String getDescription() {
21        return "Removes elements from collection with keys less than {key}";
22    }
23
24    @Override
25    public String execute(User commandedUser, String argument) {
26        int keyThreshold;
27        if (argument == null || argument.isEmpty())
28            throw new IllegalArgumentException("Please specify Dragon key.");
29        try {
30            keyThreshold = Integer.parseInt(argument);
31        } catch (IllegalArgumentException e) {
32            throw new IllegalArgumentException("Illegal key: " + e.getMessage() + ".");
33        }
34        LinkedList<Integer> toRemove = new LinkedList<>();
35        DragonHolder.getCollection().forEach((key, dragon) -> {
36            if (key < keyThreshold && dragon.getOwner().equals(commandedUser.getUsername())) toRemove.
add(key);
37        });
38        toRemove.forEach(key -> DragonHolder.getCollection().remove(key));
39        return "Removed " + toRemove.size() + " Dragons.";
40    }
41 }
42 }
```

## Файл PrintDescending.java

```
1 package com.company.commands;
2
3 import com.company.storables.Dragon;
4 import com.company.storables.DragonHolder;
5 import com.company.ui.User;
6
7 import java.util.Comparator;
8
9 public class PrintDescending implements CommandAction {
10     String response;
11
12     public String getLabel() {
13         return "print_descending";
14     }
15
16     public String getDescription() {
17         return "Show all collection elements sorted by age.";
18     }
19
20     public String execute(User commandedUser, String argument) {
21         response = "Sorted collection:\n";
22         DragonHolder.getCollection().values().stream().sorted(Comparator.comparingLong(Dragon::getAge).
reversed())
23             .forEachOrdered(element -> response += element.toString() + "\n");
24         return response.substring(0, response.length() - 1);
25     }
26 }
27 }
```

## Файл RemoveGreaterKey.java

```
1 package com.company.commands;
2
3 import com.company.storables.DragonHolder;
4 import com.company.ui.User;
5
6 import java.util.LinkedList;
7
8 public class RemoveGreaterKey implements CommandAction {
9     @Override
10    public String getLabel() {
11        return "remove_greater_key";
12    }
13
14    @Override
15    public String getArgumentLabel() {
16        return "{key}";
17    }
18
19    @Override
20    public String getDescription() {
21        return "Removes elements from collection with keys larger than {key}";
22    }
23
24    @Override
25    public String execute(User commandedUser, String argument) {
26        int keyThreshold;
27        if (argument == null || argument.isEmpty())
28            throw new IllegalArgumentException("Please specify Dragon key.");
29        try {
30            keyThreshold = Integer.parseInt(argument);
31        } catch (IllegalArgumentException e) {
32            throw new IllegalArgumentException("Illegal key: " + e.getMessage() + ".");
33        }
34        LinkedList<Integer> toRemove = new LinkedList<>();
35        DragonHolder.getCollection().forEach((key, dragon) -> {
36            if (key > keyThreshold && dragon.getOwner().equals(commandedUser.getUsername())) toRemove.
add(key);
37        });
38        toRemove.forEach(key -> DragonHolder.getCollection().remove(key));
39        return "Removed " + toRemove.size() + " Dragons.";
40    }
41 }
42 }
```

## Файл FilterLessThanType.java

```
1 package com.company.commands;
2
3 import com.company.storables.Coordinates;
4 import com.company.storables.Dragon;
5 import com.company.storables.DragonHead;
6 import com.company.storables.DragonHolder;
7 import com.company.ui.User;
8
9 import java.text.DateFormat;
10 import java.text.ParseException;
11 import java.util.ArrayList;
12 import java.util.Date;
13 import java.util.Locale;
14 import java.util.regex.PatternSyntaxException;
15
16 public class FilterLessThanType implements CommandAction {
17     String response;
18
19     public String getLabel() {
20         return "filter_less_than_type";
21     }
22
23     public String getArgumentLabel() {
24         return "{type} {value}";
25     }
26
27     public String getDescription() {
28         return "Filter all elements that have {type} less than {value}.";
29     }
30
31     public String execute(User commandedUser, String argument) {
32         String[] arguments;
33         try {
34             arguments = argument.split(" ", 2);
35         } catch (PatternSyntaxException | NullPointerException e) {
36             throw new IllegalArgumentException("Please specify field type and value.");
37         }
38         if (arguments.length < 1)
39             throw new IllegalArgumentException("Please specify field type and value.");
40         if (arguments.length < 2)
41             throw new IllegalArgumentException("Please specify field value.");
42         String field = arguments[0];
43         String value = arguments[1];
44         ArrayList<Dragon> result = new ArrayList<>();
45         try {
46             Dragon.class.getDeclaredField(field);
47         } catch (NoSuchFieldException e) {
48             throw new IllegalArgumentException("No field named '" + field + "'.");
49         }
50         switch (field) {
51             case "id":
52             case "age":
53                 try {
54                     long compareToLong = Long.parseLong(value);
55                     DragonHolder.getCollection().forEach((key, element) -> {
56                         if (((field.equals("id")) ? element.getId() : element.getAge()) < compareToLong)
57                             result.add(element);
58                     });
59                 } catch (NumberFormatException | NullPointerException e) {
60                     throw new IllegalArgumentException("Can't parse " + field + " from '" + value + "'"
61 : " + e.getMessage() + ".");
62                 }
63                 break;
64             case "name":
65             case "description":
66                 DragonHolder.getCollection().forEach((key, element) -> {
67                     if (((field.equals("name")) ? element.getName() : element.getDescription())
68                         compareTo(value) < 0)
69                         result.add(element);
70                 });
71                 break;
72             case "coordinates":
73                 Coordinates compareToCoordinates = new Coordinates(value);
74                 DragonHolder.getCollection().forEach((key, element) -> {
```

## Файл FilterLessThanType.java

```
73             if (element.getCoordinates().compareTo(compareToCoordinates) < 0)
74                 result.add(element);
75         );
76         break;
77     case "creationDate":
78         try {
79             Date compareToDate = DateFormat.getDateInstance(DateFormat.SHORT, Locale.getDefault())
80             .parse(value);
81             DragonHolder.getCollection().forEach((key, element) -> {
82                 if (element.getCreationDate().compareTo(compareToDate) < 0)
83                     result.add(element);
84             });
85         } catch (ParseException e) {
86             throw new IllegalArgumentException("Can't parse date from \"" + value + "\": " + e.getMessage() + ".");
87         }
88         break;
89     case "weight":
90         try {
91             int compareToWeight = Integer.parseInt(value);
92             DragonHolder.getCollection().forEach((key, element) -> {
93                 if (element.getWeight() < compareToWeight)
94                     result.add(element);
95             });
96         } catch (NumberFormatException | NullPointerException e) {
97             throw new IllegalArgumentException("Can't parse weight from \"" + value + "\": " + e.getMessage() + ".");
98         }
99         break;
100    case "type":
101        throw new IllegalArgumentException("Can't compare Dragon types. All Dragons are equal!");
102    case "head":
103        float compareToEyesCount = new DragonHead(value).getEyesCount();
104        DragonHolder.getCollection().forEach((key, element) -> {
105            if (element.getHead().getEyesCount() < compareToEyesCount)
106                result.add(element);
107        });
108        break;
109    default:
110        throw new IllegalArgumentException("No field named \"" + field + "\".");
111    }
112    response = "Result:\n";
113    result.forEach(element -> response += element.toString() + "\n");
114 }
115 }
```

## Файл ReplaceIfGreaterAge.java

```
1 package com.company.commands;
2
3 import com.company.storables.Dragon;
4 import com.company.storables.DragonHolder;
5 import com.company.storables.DragonUtils;
6 import com.company.ui.User;
7
8 public class ReplaceIfGreaterAge implements CommandAction {
9     @Override
10    public String getLabel() {
11        return "replace_if_greater";
12    }
13
14    @Override
15    public String getArgumentLabel() {
16        return "{key} {Dragon}";
17    }
18
19    @Override
20    public String getDescription() {
21        return "replaces a {Dragon} by {key} if he has a greater age";
22    }
23
24    @Override
25    public String execute(User commandedUser, String argument) {
26        int key;
27        if (argument == null || argument.isEmpty())
28            throw new IllegalArgumentException("Please specify Dragon key.");
29        try {
30            key = Integer.parseInt(argument);
31        } catch (IllegalArgumentException e) {
32            throw new IllegalArgumentException("Illegal key: " + e.getMessage() + ".");
33        }
34        Dragon currentDragon = DragonHolder.getCollection().get(key);
35        if (currentDragon == null)
36            throw new IllegalArgumentException("No Dragon found with key \\" + key + "\\".");
37        if (!currentDragon.getOwner().equals(commandedUser.getUsername()))
38            throw new IllegalArgumentException("Unauthorized Dragon access with id " + DragonHolder.
getCollection().get(key).getId() + ".");
39        else {
40            Dragon newDragon = DragonUtils.inputNewDragonFromConsole(commandedUser);
41            if (newDragon.getAge() > currentDragon.getAge()) {
42                DragonHolder.getCollection().replace(key, newDragon);
43                return "Remove successful.";
44            } else {
45                return "Dragon not replaced: new dragon is not older.";
46            }
47        }
48    }
49 }
50 }
```

## Файл CsvReplaceIfGreaterAge.java

```
1 package com.company.commands;
2
3 import com.company.storables.Dragon;
4 import com.company.storables.DragonHolder;
5 import com.company.storables.DragonUtils;
6 import com.company.ui.User;
7
8 import java.util.Date;
9
10 public class CsvReplaceIfGreaterAge implements CommandAction {
11     @Override
12     public String getLabel() {
13         return "replace_if_greater_csv";
14     }
15
16     @Override
17     public String getArgumentLabel() {
18         return "{key},{Dragon,in,csv,style}";
19     }
20
21     @Override
22     public String getDescription() {
23         return "replaces a {Dragon} by {key} if he has a greater age (CSV version).";
24     }
25
26     @Override
27     public String execute(User commandedUser, String argument) {
28         int key;
29         if (!argument.isBlank()) {
30             String[] splitLine = argument.split(",", 2);
31             if (splitLine.length < 2)
32                 throw new IllegalArgumentException("Invalid key,dragon input string: \\" + argument + "
33                                         + "\");
34             try {
35                 key = Integer.parseInt(splitLine[0]);
36                 Dragon currentDragon = DragonHolder.getCollection().get(key);
37                 if (currentDragon == null)
38                     throw new IllegalArgumentException("No Dragon found with key \\" + key + "\");
39                 if(!currentDragon.getOwner().equals(commandedUser.getUsername()))
40                     throw new IllegalArgumentException("Unauthorized Dragon access with id " +
41                                         currentDragon.getId() + ".");
42                 Dragon newDragon = new Dragon(splitLine[1]);
43                 newDragon.setId(DragonUtils.getNewId());
44                 newDragon.setCreationDate(new Date());
45                 if (newDragon.getAge() > currentDragon.getAge()) {
46                     DragonHolder.getCollection().replace(Integer.parseInt(splitLine[0]), newDragon);
47                     return "Remove successful.";
48                 } else {
49                     return "Dragon not replaced: new dragon is not older.";
50                 }
51             } catch (NumberFormatException e) {
52                 throw new IllegalArgumentException("Can't parse key from " + splitLine[0] + ".");
53             }
54         } else {
55             throw new IllegalArgumentException("Please specify key,dragon.");
56     }
57 }
```

## Файл Dragon.java

```
1 package com.company.storables;
2
3 import com.company.ui.User;
4
5 import java.text.DateFormat;
6 import java.text.ParseException;
7 import java.util.Date;
8 import java.util.Locale;
9 import java.util.Objects;
10
11 public class Dragon implements Comparable<Dragon> {
12     private long id; //Значение поля должно быть больше 0, Значение этого поля должно быть уникальным,
13     Значение этого поля должно генерироваться автоматически
14     private String name; //Поле не может быть null, Стока не может быть пустой
15     private Coordinates coordinates; //Поле не может быть null
16     private java.util.Date creationDate; //Поле не может быть null, Значение этого поля должно
17     генерироваться автоматически
18     private long age; //Значение поля должно быть больше 0
19     private String description; //Поле может быть null
20     private int weight; //Значение поля должно быть больше 0
21     private DragonType type; //Поле не может быть null
22     private DragonHead head;
23     private String owner;
24
25     /**
26      * Converts CSV format string to Dragon
27      *
28      * @param csvString CSV format string
29      * @throws IllegalArgumentException if CSV format string can't be parsed
30     */
31     public Dragon(String csvString) throws IllegalArgumentException {
32         String[] splitString = csvString.split(",");
33         try {
34             if (splitString[0].isBlank())
35                 setId(DragonUtils.getNewId());
36             else
37                 try {
38                     if (Long.parseLong(splitString[0]) == -1L)
39                         setId(DragonUtils.getNewId());
40                     setId(Long.parseLong(splitString[0]));
41                 } catch (NumberFormatException e) {
42                     throw new IllegalArgumentException("Can't parse Dragon id from " + splitString[0] +
43                         ".");
44                 }
45             setName(splitString[1].replaceAll("\\\"", ""));
46             try {
47                 setCoordinates(new Coordinates(splitString[2].isBlank() ? .0 : Double.parseDouble(
48                     splitString[2]), Long.parseLong(splitString[3])));
49             } catch (NumberFormatException | NullPointerException e) {
50                 throw new IllegalArgumentException("Can't parse Dragon coordinates from " + splitString[2] +
51                     "," + splitString[3] + ".");
52             }
53             if (splitString[4].isBlank())
54                 setCreationDate(new Date());
55             else
56                 try {
57                     setCreationDate(DateFormat.getDateInstance(DateFormat.SHORT, Locale.getDefault()).
58                         parse(splitString[4]));
59                 } catch (ParseException e) {
60                     throw new IllegalArgumentException("Can't parse creation date from " + splitString[4] +
61                         ".");
62                 }
63             try {
64                 setAge(Long.parseLong(splitString[5]));
65             } catch (NumberFormatException e) {
66                 throw new IllegalArgumentException("Can't parse Dragon age from " + splitString[5] + ".");
67             }
68             setDescription(splitString[6].isBlank() ? "\"" : splitString[6].replaceAll("\\\"", ""));
69             try {
70                 setWeight(Integer.parseInt(splitString[7]));
71             } catch (NumberFormatException e) {
72                 throw new IllegalArgumentException("Can't parse Dragon weight from " + splitString[7] +
73                     ".");
74         }
75     }
76 }
```

## 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            }
73            setHead(new DragonHead(splitString[9]));
74            setOwner(splitString[10].isBlank() ? "\\" : splitString[10].replaceAll("\\\"", ""));
75        } catch (IndexOutOfBoundsException e) {
76            throw new IllegalArgumentException("Not enough arguments to parse Dragon (" + splitString.
77                length + ").");
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().
355             getY() + "," + DateFormat.getDateInstance(DateFormat.SHORT).format(getCreationDate()) + "," + getAge
356             () + ",\\" + (getDescription() == null ? "" : getDescription()) + "\",\"" + getWeight() + "," + getType
357             ().getLabel() + "," + getHead().getEyesCount() + ",\"" + getOwner() + "\"";
358     }
```

## Файл Dragon.java

```
356 }
```

## Файл DragonHead.java

```
1 package com.company.storables;
2
3 /**
4 * Head of the dragon
5 *
6 * @see Dragon
7 */
8 public class DragonHead {
9
10    private float eyesCount;
11
12    /**
13     * Dragon head from string constructor
14     *
15     * @param string to parse from
16     * @throws IllegalArgumentException if string is invalid
17     */
18    public DragonHead(String fromString) throws IllegalArgumentException {
19        try {
20            this.eyesCount = Float.parseFloat(fromString);
21        } catch (NumberFormatException | NullPointerException e) {
22            throw new IllegalArgumentException("Can't parse Dragon Head from \"" + fromString + "\".");
23        }
24    }
25
26    /**
27     * Dragon head constructor
28     *
29     * @param eyesCount count of eyes
30     */
31    public DragonHead(float eyesCount) {
32        this.eyesCount = eyesCount;
33    }
34
35    /**
36     * Get number of eyes
37     *
38     * @return number of eyes
39     */
40    public float getEyesCount() {
41        return eyesCount;
42    }
43
44    /**
45     * Set number of eyes
46     *
47     * @param eyesCount new number of eyes
48     */
49    public void setEyesCount(float eyesCount) {
50        this.eyesCount = eyesCount;
51    }
52
53    @Override
54    public String toString() {
55        return "DragonHead: " +
56                " eyesCount=" + eyesCount;
57    }
58 }
59 }
```

## Файл DragonType.java

```
1 package com.company.storables;
2
3 /**
4 * Type of the dragon
5 *
6 * @see Dragon
7 */
8 public enum DragonType {
9     AIR("AIR"),
10    FIRE("FIRE"),
11    UNDERGROUND("UNDERGROUND"),
12    WATER("WATER");
13
14    private final String label;
15
16    /**
17     * Dragon type constructor
18     *
19     * @param label name
20     */
21    DragonType(String label) {
22        this.label = label;
23    }
24
25    /**
26     * Gets dragon type name
27     *
28     * @return dragon type name
29     */
30    public String getLabel() {
31        return label;
32    }
33}
34
```

## Файл Coordinates.java

```
1 package com.company.storables;
2
3 import java.util.regex.PatternSyntaxException;
4
5 /**
6  * Coordinates where Dragon can be at
7  *
8  * @see Dragon
9 */
10 public class Coordinates implements Comparable<Coordinates> {
11
12     private double x;
13     private Long y; //Can't be null
14
15     /**
16      * Coordinates from string constructor
17      *
18      * @param fromString string to parse from
19      * @throws IllegalArgumentException if string is invalid
20      */
21     public Coordinates(String fromString) throws IllegalArgumentException {
22         String[] arguments;
23         try {
24             arguments = fromString.split(" ", 2);
25         } catch (PatternSyntaxException e) {
26             throw new IllegalArgumentException("Error in argument: " + e.getMessage() + ".");
27         }
28         try {
29             if (arguments.length > 1) {
30                 this.x = Double.parseDouble(arguments[0]);
31                 this.y = Long.parseLong(arguments[1]);
32             } else if (arguments.length > 0) {
33                 this.y = Long.parseLong(arguments[0]);
34             }
35         } catch (NumberFormatException | NullPointerException e) {
36             throw new IllegalArgumentException("Can't parse Coordinates from \""
+ fromString + "\": "
+ e.getMessage() + ".");
37         }
38     }
39
40     /**
41      * Coordinates constructor
42      *
43      * @param x x coordinate
44      * @param y y coordinate
45      */
46     public Coordinates(double x, Long y) {
47         this.x = x;
48         this.y = y;
49     }
50
51     /**
52      * Gets x coordinate
53      *
54      * @return x coordinate
55      */
56     public double getX() {
57         return x;
58     }
59
60     /**
61      * Sets x coordinate
62      *
63      * @param x new x coordinate
64      */
65     public void setX(double x) {
66         this.x = x;
67     }
68
69     /**
70      * Gets y coordinate
71      *
72      * @return y coordinate
73      */
```

## 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.
100             getY() * o.getY() + o.getX() * o.getX()));
101    }
```

## Файл DragonUtils.java

```
1 package com.company.storables;
2
3 import com.company.ui.CommandReader;
4 import com.company.ui.User;
5
6 import java.util.ArrayList;
7 import java.util.Arrays;
8 import java.util.Date;
9
10
11 /**
12 * Class that is needed to create dragons
13 */
14 public class DragonUtils {
15
16     /**
17      * Gets unique Dragon ID
18      *
19      * @return ID
20     */
21     synchronized public static long getNewId() {
22         ArrayList<Long> usedIDs = new ArrayList<>();
23         DragonHolder.getCollection().values().forEach(dragon -> usedIDs.add(dragon.getId()));
24         long id = 1;
25         while (usedIDs.contains(id)) id++;
26         usedIDs.add(id);
27         return id;
28     }
29
30     /**
31      * Inputs new dragon from Console (System.in)
32      *
33      * @return new Dragon
34     */
35     public static Dragon inputNewDragonFromConsole(User owner) {
36         return inputDragonFromConsole(owner, getNewId(), new Date());
37     }
38
39     /**
40      * Update dragon from Console (System.in), with existing id and
41      *
42      * @param owner      creator
43      * @param id         id
44      * @param creationDate creation date
45      * @return new dragon
46     */
47     public static Dragon inputDragonFromConsole(User owner, long id, Date creationDate) {
48         Dragon dragon = new Dragon();
49         dragon.setOwner(owner.getUsername());
50         dragon.setId(id);
51         System.out.println("Please input {Name} {Age} [Description] {Weight}.");
52         String name = "";
53         long age = 0;
54         String description = null;
55         int weight = 0;
56         boolean success = false;
57         do {
58             try {
59                 String[] input = CommandReader.getStringsFromTerminal();
60                 if (input.length < 4) {
61                     if (input.length == 3) {
62                         dragon.setName(input[0]);
63                         dragon.setAge(Integer.parseInt(input[1]));
64                         dragon.setWeight(Integer.parseInt(input[2]));
65                     } else {
66                         throw new IllegalArgumentException(Arrays.toString(input) + input.length);
67                     }
68                 } else {
69                     dragon.setName(input[0]);
70                     dragon.setAge(Integer.parseInt(input[1]));
71                     dragon.setDescription(input[2]);
72                     dragon.setWeight(Integer.parseInt(input[3]));
73                 }
74             success = true;
75         }
```

## 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 }
```

## Файл DragonHolder.java

```
1 package com.company.storables;
2
3 import java.util.Date;
4 import java.util.Hashtable;
5
6 /**
7  * Class that holds the collection
8  */
9 public class DragonHolder {
10     private static final Date initializationDate = new Date();
11
12     private static final Hashtable<Integer, Dragon> collection = new Hashtable<>();
13
14     /**
15      * Access the collection
16      *
17      * @return Dragon collection
18      */
19     synchronized public static Hashtable<Integer, Dragon> getCollection() {
20         return collection;
21     }
22
23     /**
24      * Get the time when collection was initialized
25      *
26      * @return Date time
27      */
28     public static Date getInitializationDate() {
29         return initializationDate;
30     }
31 }
32 }
```