

```
1 package org.example;
2
3 import org.example.bootstrap.Bootstrap;
4
5 /**
6  * Главный класс приложения
7  */
8 public class App {
9     public static void main(String[] args) {
10         Bootstrap bootstrap = new Bootstrap();
11         bootstrap.start(args);
12     }
13 }
14
```

```

1 package org.example.dao;
2
3 import org.example.enums.SortStatus;
4 import org.example.exception.MusicBandNotFoundException;
5 import org.example.exception.MusicBandWrongAttributeException;
6 import org.example.model.DataStorage;
7 import org.example.model.MusicBand;
8 import org.example.model.Person;
9
10 import java.time.LocalDateTime;
11 import java.util.*;
12
13 /**
14  * Реализация объекта доступа к данным (Data Access Object)
15  * В качестве хранилища используется {@link ArrayList}
16  * Хранит {@link HashSet} паспортных данных для проверки уникальности,
17  * дату инициализации, статус сортировки {@link SortStatus}
18 */
19 public class MusicBandDAO implements IMusicBandDAO {
20     List<MusicBand> musicBandList = new ArrayList<>();
21     Set<String> passportIDUniqueValues = new HashSet<>();
22     LocalDateTime initDate = LocalDateTime.now();
23     String type = "ArrayList";
24     SortStatus sortStatus = SortStatus.ASC;
25
26     /**
27      * Проверяет объект на null, {@code passportID} на уникальность,
28      * в случае успешной проверки добавляет объект в коллекцию
29      *
30      * @param musicBand объект для сохранения
31      * @throws MusicBandWrongAttributeException при наличии невалидных
32      * атрибутов
33      */
34     @Override
35     public void save(MusicBand musicBand) {
36         if (musicBand == null || musicBand.getFrontMan() == null) {
37             throw new MusicBandWrongAttributeException("Значение поля не
38 должно быть null");
39         }
40         String passportID = musicBand.getFrontMan().getPassportID();
41         if (isPassportIDExists(passportID)) {
42             throw new MusicBandWrongAttributeException("Значение этого
43 поля должно быть уникальным");
44         } else if (passportID != null) {
45             passportIDUniqueValues.add(passportID);
46         }
47         musicBandList.add(musicBand);
48     }
49
50     @Override
51     public List<MusicBand> getAll() {
52         return musicBandList;
53     }
54
55 }
```

```

52     /**
53      * Находит объект по {@code id} и меняет его на переданный
54      * Проверяет уникальность {@code passportID} нового объекта, если он
55      * отличается от старого
56      *
57      * @param musicBand новый объект
58      * @param id        id объекта для замены
59      * @throws MusicBandWrongAttributeException при наличии невалидных
60      * атрибутов
61      */
62     @Override
63     public void update(MusicBand musicBand, Long id) {
64         int number = getNumberById(id);
65         MusicBand oldBand = musicBandList.get(number);
66         Person oldFrontMan = musicBandList.get(number).getFrontMan();
67         String passportID = musicBand.getFrontMan().getPassportID();
68         if (passportID != null &&
69             oldFrontMan != null &&
70             !passportID.equals(oldFrontMan.getPassportID()) &&
71             isPassportIDExists(passportID)) {
72             throw new MusicBandWrongAttributeException("Значение этого
73             поля должно быть уникальным");
74         } else if (passportID != null) {
75             passportIDUniqueValues.add(passportID);
76             clearPassportId(oldBand);
77         }
78         musicBandList.set(number, musicBand);
79     }
80
81     @Override
82     public MusicBand removeById(Long id) {
83         int number = getNumberById(id);
84         if (musicBandList.isEmpty() || musicBandList.size() - 1 < number
85         ) {
86             return null;
87         } else {
88             MusicBand remove = musicBandList.remove(number);
89             clearPassportId(remove);
90             return remove;
91         }
92     }
93
94     @Override
95     public List<MusicBand> removeByDescription(String description) {
96         List<MusicBand> removed = new ArrayList<>();
97         Iterator<MusicBand> it = musicBandList.iterator();
98         while (it.hasNext()) {
99             MusicBand musicBand = it.next();
100            if (musicBand != null &&
101                musicBand.getDescription() != null &&
102                musicBand.getDescription().equalsIgnoreCase(
103                    description)) {
104                    removed.add(musicBand);
105                    it.remove();
106                }
107            }
108        }
109    }
110}

```

```
101         }
102     }
103     return removed;
104 }
105
106 @Override
107 public void clear() {
108     musicBandList.clear();
109     passportIDUniqueValues.clear();
110 }
111
112 @Override
113 public boolean isPassportIDExists(String passportID) {
114     return passportID != null && passportIDUniqueValues.contains(
115         passportID);
116 }
117
118 @Override
119 public int getNumberById(Long id) {
120     for (int i = 0; i < musicBandList.size(); i++) {
121         if (musicBandList.get(i).getId().equals(id)) {
122             return i;
123         }
124     }
125     throw new MusicBandNotFoundException("Объект не найден");
126 }
127
128 @Override
129 public MusicBand getByNumber(int number) {
130     return musicBandList.get(number);
131 }
132
133 @Override
134 public MusicBand getById(Long id) {
135     int numberById = getNumberById(id);
136     return getByNumber(numberById);
137 }
138
139 @Override
140 public void init(DataStorage dataStorage) {
141     if (dataStorage.getInitDate() != null)
142         this.initDate = dataStorage.getInitDate();
143     List<MusicBand> bands = dataStorage.getBands();
144     for (MusicBand band : bands) {
145         save(band);
146     }
147 }
148
149 @Override
150 public DataStorage getData() {
151     return new DataStorage(musicBandList, initDate, type);
152 }
153
154 @Override
```

```
154     public void checkPassportIDUnique(String passportID) {
155         if (isPassportIDExists(passportID)) {
156             throw new MusicBandWrongAttributeException("Значение этого
157             поля должно быть уникальным");
158         }
159     }
160
161     @Override
162     public MusicBand getMinimal() {
163         List<MusicBand> newList = new ArrayList<>(musicBandList);
164         Collections.sort(newList);
165         if (newList.isEmpty()) {
166             return null;
167         }
168         return newList.get(0);
169     }
170
171     @Override
172     public MusicBand getByMaxId() {
173         List<MusicBand> newList = new ArrayList<>(musicBandList);
174         newList.sort(Comparator.comparingLong(MusicBand::getId));
175         if (newList.isEmpty()) {
176             return null;
177         }
178         return newList.get(newList.size() - 1);
179     }
180
181     @Override
182     public MusicBand removeLast() {
183         if (musicBandList.isEmpty()) {
184             return null;
185         } else {
186             MusicBand removed = musicBandList.remove(musicBandList.size
187             () - 1);
188             clearPassportId(removed);
189             return removed;
190         }
191     }
192
193     private void clearPassportId(MusicBand remove) {
194         if (remove != null && remove.getFrontMan() != null) {
195             passportIDUniqueValues.remove(remove.getFrontMan().
196             getPassportID());
197         }
198     }
199
200     @Override
201     public SortStatus reorder() {
202         switch (sortStatus) {
203             case DESC:
204                 Collections.reverse(musicBandList);
205                 sortStatus = SortStatus.ASC;
206                 break;
207             case ASC:
```

```
205         Collections.sort(musicBandList);
206         sortStatus = SortStatus.DESC;
207         break;
208     default:
209         break;
210     }
211     return sortStatus;
212 }
213
214 @Override
215 public List<MusicBand> filterByDescription(String description) {
216     List<MusicBand> newList = new ArrayList<>();
217     for (MusicBand musicBand : musicBandList) {
218         if (musicBand != null &&
219             musicBand.getDescription() != null &&
220             musicBand.getDescription().compareTo(description) < 0
221         ) {
222             newList.add(musicBand);
223         }
224     }
225 }
226 }
227
```

```
1 package org.example.dao;
2
3 import org.example.enums.SortStatus;
4 import org.example.model.DataStorage;
5 import org.example.model.MusicBand;
6
7 import java.util.List;
8
9 /**
10  * Интерфейс сервиса доступа к данным (Data Access Object)
11 */
12 public interface IMusicBandDAO {
13
14     void save(MusicBand musicBand);
15
16     List<MusicBand> getAll();
17
18     void update(MusicBand musicBand, Long id);
19
20     void clear();
21
22     boolean isPassportIDExists(String passportID);
23
24     int getNumberById(Long id);
25
26     MusicBand getByNumber(int number);
27
28     MusicBand getById(Long id);
29
30     void init(DataStorage dataStorage);
31
32     DataStorage getData();
33
34     void checkPassportIDUnique(String passportID);
35
36     MusicBand getMinimal();
37
38     MusicBand getByMaxId();
39
40     MusicBand removeLast();
41
42     MusicBand removeById(Long id);
43
44     List<MusicBand> removeByDescription(String description);
45
46     SortStatus reorder();
47
48     List<MusicBand> filterByDescription(String description);
49 }
50
```

```

1 package org.example.util;
2
3 import org.example.exception.InterruptInputException;
4 import org.example.exception.MusicBandWrongAttributeException;
5 import org.example.model.MusicBand;
6 import org.example.service.IConsoleService;
7
8 import java.util.function.Consumer;
9
10 /**
11  * Сервис
12 */
13 public class MusicBandAttributeSetter {
14     private final IConsoleService consoleService;
15
16     public MusicBandAttributeSetter(IConsoleService consoleService) {
17         this.consoleService = consoleService;
18     }
19
20     /**
21      * Применяет функцию {@code consumer} к объекту,
22      * в случае возникновения ошибки выводит ее текст и спрашивает о
23      * повторном вводе,
24      * в случае отрицательного ответа выбрасывает {@link
25      * InterruptInputException}
26      *
27      * @param musicBand объект для заполнения
28      * @param message текст приглашения к вводу значения
29      * @param consumer функция, заполняющая нужное поле
30      */
31     public void setAttribute(MusicBand musicBand,
32                             String message,
33                             Consumer<MusicBand> consumer) {
34         boolean stop = false;
35         while (!stop) {
36             consoleService.println(message);
37             try {
38                 consumer.accept(musicBand);
39                 stop = true;
40             } catch (MusicBandWrongAttributeException e) {
41                 consoleService.println(e.getMessage());
42                 askDoAgain();
43             } catch (NumberFormatException e) {
44                 consoleService.println("Требуется ввести число");
45                 askDoAgain();
46             } catch (IllegalArgumentException e) {
47                 consoleService.println("Неверный ввод (введите значение из
48                 квадратных скобок)");
49                 askDoAgain();
50             }
51         }
52     }
53
54     private void askDoAgain() {

```

```
52         consoleService.println("Повторить ввод - любой СИМВОЛ, ВЫТИ -  
53         exit");  
54         if ("exit".equals(consoleService.read())) {  
55             throw new InterruptInputException();  
56         }  
57     }  
58 }
```

```
1 package org.example.enums;
2
3 public enum Color {
4     RED,
5     BLUE,
6     YELLOW,
7     WHITE;
8 }
9
```

```
1 package org.example.enums;
2
3 public enum MusicGenre {
4     PROGRESSIVE_ROCK,
5     PSYCHEDELIC_ROCK,
6     RAP,
7     POP;
8 }
9
```

```
1 package org.example.enums;  
2  
3 public enum SortStatus {  
4     ASC,  
5     DESC  
6 }  
7
```

```
1 package org.example.model;
2
3 import org.example.enums.Color;
4 import org.example.exception.MusicBandWrongAttributeException;
5
6 import java.io.Serializable;
7
8 public class Person implements Serializable {
9
10    private String name;
11
12    private double height;
13
14    private String passportID;
15
16    private Color eyeColor;
17
18    public String getName() {
19        return name;
20    }
21
22    public void setName(String name) {
23        if (name == null) {
24            throw new MusicBandWrongAttributeException("Поле не может быть
null");
25        }
26        if (name.trim().equals("")) {
27            throw new MusicBandWrongAttributeException("Строка не может
быть пустой");
28        }
29        this.name = name;
30    }
31
32    public double getHeight() {
33        return height;
34    }
35
36    public void setHeight(double height) {
37        if (height < 0) {
38            throw new MusicBandWrongAttributeException("Значение поля
должно быть больше 0");
39        }
40        this.height = height;
41    }
42
43    public String getPassportID() {
44        return passportID;
45    }
46
47    public void setPassportID(String passportID) {
48        if (passportID.trim().equals("")) {
49            throw new MusicBandWrongAttributeException("Строка не может
быть пустой");
50        }
51    }
52}
```

```
51         if (passportID.length() > 31) {
52             throw new MusicBandWrongAttributeException("Длина строки не
53                 должна быть больше 31");
54             }
55         this.passportID = passportID;
56     }
57
58     public Color getEyeColor() {
59         return eyeColor;
60     }
61
62     public void setEyeColor(String eyeColor) {
63         if (eyeColor == null) {
64             throw new MusicBandWrongAttributeException("Поле не может
65                 быть null");
66         }
67         this.eyeColor = Color.valueOf(eyeColor);
68     }
69
70     @Override
71     public String toString() {
72         return "Имя: " + name +
73             ", Рост: " + height +
74             ", Номер паспорта: " + passportID +
75             ", Цвет глаз: " + eyeColor;
76     }
77 }
```

```
1 package org.example.model;
2
3 import com.fasterxml.jackson.annotation.JsonFormat;
4 import org.example.enums.MusicGenre;
5 import org.example.exception.MusicBandWrongAttributeException;
6
7 import java.io.Serializable;
8 import java.time.LocalDateTime;
9 import java.time.format.DateTimeFormatter;
10
11 public class MusicBand implements Comparable<MusicBand>, Serializable {
12     public MusicBand() {
13         this.id = System.currentTimeMillis();
14         this.creationDate = LocalDateTime.now();
15     }
16
17     private Long id;
18
19     private String name;
20
21     private Coordinates coordinates;
22
23     @JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss")
24     private LocalDateTime creationDate;
25
26     private Integer numberOfParticipants;
27
28     private String description;
29
30     private MusicGenre genre;
31
32     private Person frontMan;
33
34     public Long getId() {
35         return id;
36     }
37
38     public String getName() {
39         return name;
40     }
41
42     public void setName(String name) {
43         if (name == null) {
44             throw new MusicBandWrongAttributeException("Поле не может быть
45             null");
46         }
47         if (name.trim().equals("")) {
48             throw new MusicBandWrongAttributeException("Строка не может
49             быть пустой");
50         }
51         this.name = name;
52     }
53
54     public Coordinates getCoordinates() {
```

```
53     return coordinates;
54 }
55
56     public void setCoordinates(Coordinates coordinates) {
57         if (coordinates == null) {
58             throw new MusicBandWrongAttributeException("Поле не может
59                 быть null");
60         }
61         this.coordinates = coordinates;
62     }
63
64     public LocalDateTime getCreationDate() {
65         return creationDate;
66     }
67
68     public Integer getNumberOfParticipants() {
69         return numberOfParticipants;
70     }
71
72     public void setNumberOfParticipants(Integer numberOfParticipants) {
73         if (numberOfParticipants == null) {
74             throw new MusicBandWrongAttributeException("Поле не может
75                 быть null");
76         }
77         if (numberOfParticipants <= 0) {
78             throw new MusicBandWrongAttributeException("Значение поля
79                 должно быть больше 0");
80         }
81         this.numberOfParticipants = numberOfParticipants;
82     }
83
84     public String getDescription() {
85         return description;
86     }
87
88     public void setDescription(String description) {
89         if ("".equals(description)) {
90             this.description = null;
91         }
92         this.description = description;
93     }
94
95     public MusicGenre getGenre() {
96         return genre;
97     }
98
99     public void setGenre(String genre) {
100        if (genre == null || "".equals(genre)) {
101            this.genre = null;
102        }
103        this.genre = MusicGenre.valueOf(genre);
104    }
105}
```

```
104     public Person getFrontMan() {
105         return frontMan;
106     }
107
108     public void setFrontMan(Person frontMan) {
109         if (frontMan == null) {
110             throw new MusicBandWrongAttributeException("Поле не может
быть null");
111         }
112         this.frontMan = frontMan;
113     }
114
115     @Override
116     public int compareTo(MusicBand o) {
117         return this.name.compareTo(o.name);
118     }
119
120     @Override
121     public String toString() {
122         return "id=" + id +
123                 ", Название: " + name +
124                 ", Дата добавления: " + creationDate.format(
125                         DateTimeFormatter.ofPattern("dd.MM.yyyy HH:mm")) +
126                 ", Количество участников: " + numberofParticipants +
127                 ", Описание: " + description +
128                 ", Жанр: " + genre +
129                 ", Солист: " + frontMan +
130                 ", Координаты: " + coordinates;
131     }
132 }
```

```
1 package org.example.model;
2
3 import org.example.exception.MusicBandWrongAttributeException;
4
5 import java.io.Serializable;
6
7 public class Coordinates implements Serializable {
8
9     private Long x;
10
11    private Float y;
12
13    public Long getX() {
14        return x;
15    }
16
17    public void setX(Long x) {
18        if (x == null) {
19            throw new MusicBandWrongAttributeException("Поле не может быть
null");
20        }
21        if (x > 565L) {
22            throw new MusicBandWrongAttributeException("Максимальное
значение поля: 565");
23        }
24        this.x = x;
25    }
26
27    public Float getY() {
28        return y;
29    }
30
31    public void setY(Float y) {
32        if (y == null) {
33            throw new MusicBandWrongAttributeException("Поле не может быть
null");
34        }
35        if (y > 900L) {
36            throw new MusicBandWrongAttributeException("Максимальное
значение поля: 900");
37        }
38        this.y = y;
39    }
40
41    @Override
42    public String toString() {
43        return "x: " + x +
44                ", y: " + y;
45    }
46 }
47
```

```
1 package org.example.model;
2
3 import com.fasterxml.jackson.annotation.JsonFormat;
4
5 import java.io.Serializable;
6 import java.time.LocalDateTime;
7 import java.time.format.DateTimeFormatter;
8 import java.util.ArrayList;
9 import java.util.List;
10
11 public class DataStorage implements Serializable {
12
13     public DataStorage(List<MusicBand> bands, LocalDateTime initDate,
14                         String type) {
15         this.bands = bands;
16         this.initDate = initDate;
17         this.type = type;
18     }
19
20     public DataStorage() {
21
22         private List<MusicBand> bands = new ArrayList<>();
23         @JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss")
24         private LocalDateTime initDate;
25
26         private String type;
27
28         public List<MusicBand> getBands() {
29             return bands;
30         }
31
32         public void setBands(List<MusicBand> bands) {
33             this.bands = bands;
34         }
35
36         public LocalDateTime getInitDate() {
37             return initDate;
38         }
39
40         public void setInitDate(LocalDateTime initDate) {
41             this.initDate = initDate;
42         }
43
44         public String getType() {
45             return type;
46         }
47
48         public void setType(String type) {
49             this.type = type;
50         }
51
52         @Override
53         public String toString() {
```

```
54         return "Размер коллекции: " + bands.size() +
55                 ", Дата инициализации: " + initDate.format(
56                     DateTimeFormatter.ofPattern("dd.MM.yyyy HH:mm")) +
57                 ", Тип коллекции: " + type;
58     }
59 }
```

```

1 package org.example.command;
2
3 import org.example.enums.Color;
4 import org.example.enums.MusicGenre;
5 import org.example.exception.InterruptInputException;
6 import org.example.model.Coordinates;
7 import org.example.model.MusicBand;
8 import org.example.model.Person;
9 import org.example.util.MusicBandAttributeSetter;
10
11 import java.util.Arrays;
12
13 public class AddCommand extends AbstractCommand {
14
15     @Override
16     public String command() {
17         return "add";
18     }
19
20     @Override
21     public String description() {
22         return "Добавить новый элемент в коллекцию";
23     }
24
25     /**
26      * Использует {@link MusicBandAttributeSetter} для наполнения объекта
27      * {@link MusicBand}
28      * и добавляет его в коллекцию
29      * @param args аргументы
30      */
31     @Override
32     public void execute(String[] args) {
33         MusicBand musicBand = new MusicBand();
34         musicBand.setFrontMan(new Person());
35         musicBand.setCoordinates(new Coordinates());
36         MusicBandAttributeSetter setter = new MusicBandAttributeSetter(
37             consoleService);
38         try {
39             setter.setAttribute(musicBand,
40                 "Введите название группы",
41                 s -> s.setName(consoleService.read()));
42             setter.setAttribute(musicBand,
43                 "Введите количество участников",
44                 s -> s.setNumberOfParticipants(Integer.valueOf(
45                 consoleService.read())));
46             setter.setAttribute(musicBand,
47                 "Введите описание группы",
48                 s -> s.setDescription(consoleService.read()));
49             setter.setAttribute(musicBand,
50                 String.format("Введите жанр группы %s", Arrays.
51                 toString(MusicGenre.values())),
52                 s -> s.setGenre(consoleService.read()));
53             setter.setAttribute(musicBand,
54                 "Введите координату x",
55                 s -> s.setX(consoleService.read()));
56         } catch (InterruptInputException e) {
57             System.out.println("Ввод прерван");
58         }
59     }
60 }
```

```
51             s -> s.getCoordinates().setX(Long.valueOf(
52                 consoleService.read())));
53             setter.setAttribute(musicBand,
54                 "Введите координату у",
55                 s -> s.getCoordinates().setY(Float.valueOf(
56                     consoleService.read())));
56             setter.setAttribute(musicBand,
57                 "Введите имя солиста",
58                 s -> s.getFrontMan().setName(consoleService.read()));
59             setter.setAttribute(musicBand,
60                 "Введите рост солиста",
61                 s -> s.getFrontMan().setHeight(Double.valueOf(
62                     consoleService.read())));
62             setter.setAttribute(musicBand,
63                 String.format("Введите цвет глаз солиста %s", Arrays.
64                     toString(Color.values())),
65                 s -> s.getFrontMan().setEyeColor(consoleService.read(
66)));
66             setter.setAttribute(musicBand,
67                 "Введите номер паспорта солиста",
68                 s -> {
69                     String line = consoleService.read();
70                     musicBandDAO.checkPassportIDUnique(line);
71                     s.getFrontMan().setPassportID(line);
72                 });
73         } catch (InterruptedException e) {
74             consoleService.println("Добавление элемента прервано");
75             return;
76         }
77     }
78 }
79 }
```

```
1 package org.example.command;
2
3 public class ExitCommand extends AbstractCommand {
4
5     @Override
6     public String command() {
7         return "exit";
8     }
9
10    @Override
11    public String description() {
12        return "Завершить программу (без сохранения в файл)";
13    }
14
15    @Override
16    public void execute(String[] args) {
17        consoleService.println("Программа закрывается. До свидания!");
18        System.exit(0);
19    }
20}
21
```

```
1 package org.example.command;
2
3 public class HelpCommand extends AbstractCommand {
4
5     @Override
6     public String command() {
7         return "help";
8     }
9
10    @Override
11    public String description() {
12        return "Вывести справку по доступным коммандам";
13    }
14
15    @Override
16    public void execute(String[] args) {
17        commands.forEach((key, value) -> consoleService.println(key +
18            " : " + value.description()));
19    }
20}
```

```
1 package org.example.command;
2
3 import org.example.model.DataStorage;
4
5 public class InfoCommand extends AbstractCommand {
6
7     @Override
8     public String command() {
9         return "info";
10    }
11
12    @Override
13    public String description() {
14        return "Вывести информацию о коллекции (тип, дата инициализации,
15 количества элементов и т.д)";
16    }
17
18    @Override
19    public void execute(String[] args) {
20        DataStorage data = musicBandDAO.getData();
21        consoleService.println(data.toString());
22    }
23 }
```

```
1 package org.example.command;
2
3 import org.example.model.DataStorage;
4
5 public class SaveCommand extends AbstractCommand {
6
7     @Override
8     public String command() {
9         return "save";
10    }
11
12    @Override
13    public String description() {
14        return "Сохранить коллекцию в файл";
15    }
16
17    @Override
18    public void execute(String[] args) {
19        DataStorage data = musicBandDAO.getData();
20        if (fileService.write(data)) {
21            consoleService.println("Успешно сохранено");
22        } else {
23            consoleService.println("Ошибка при сохранении в файл");
24        }
25    }
26}
27
```

```
1 package org.example.command;
2
3 import org.example.model.MusicBand;
4
5 import java.util.List;
6
7 public class ShowCommand extends AbstractCommand {
8
9     @Override
10    public String command() {
11        return "show";
12    }
13
14    @Override
15    public String description() {
16        return "Вывести в стандартный поток вывода все элементы коллекции
в строковом представлении";
17    }
18
19    @Override
20    public void execute(String[] args) {
21        List<MusicBand> all = musicBandDAO.getAll();
22        if (all.isEmpty()) {
23            consoleService.println("Коллекция пустая");
24        } else {
25            all.forEach(s -> {
26                if (s != null)
27                    consoleService.println(s.toString());
28            });
29        }
30    }
31 }
32
```

```
1 package org.example.command;
2
3 public class ClearCommand extends AbstractCommand {
4
5     @Override
6     public String command() {
7         return "clear";
8     }
9
10    @Override
11    public String description() {
12        return "Очистить коллекцию";
13    }
14
15    @Override
16    public void execute(String[] args) {
17        musicBandDAO.clear();
18        consoleService.println("Коллекция очищена");
19    }
20}
21
```

```
1 package org.example.command;
2
3 import org.example.model.MusicBand;
4
5 public class RemoveCommand extends AbstractCommand {
6
7     @Override
8     public String command() {
9         return "remove_by_id";
10    }
11
12    @Override
13    public String description() {
14        return "Удалить элемент из коллекции по его id";
15    }
16
17    @Override
18    public void execute(String[] args) {
19        if (args.length < 2 || args[1] == null) {
20            consoleService.println("Не хватает аргумента");
21            return;
22        }
23        Long id;
24        try {
25            id = Long.valueOf(args[1]);
26        } catch (NumberFormatException e) {
27            consoleService.println("Неверный формат аргумента");
28            return;
29        }
30        MusicBand musicBand = musicBandDAO.removeById(id);
31        if (musicBand == null) {
32            consoleService.println("Элемент не найден");
33        } else {
34            consoleService.println("Элемент успешно удален");
35        }
36    }
37 }
38 }
```

```
1 package org.example.command;
2
3 import org.example.enums.Color;
4 import org.example.enums.MusicGenre;
5 import org.example.exception.InterruptInputException;
6 import org.example.model.MusicBand;
7 import org.example.util.MusicBandAttributeSetter;
8
9 import java.util.Arrays;
10
11 public class UpdateCommand extends AbstractCommand {
12
13     @Override
14     public String command() {
15         return "update";
16     }
17
18     @Override
19     public String description() {
20         return "Обновить значение элемента коллекции, id которого равен
21 заданному";
22     }
23
24     @Override
25     public void execute(String[] args) {
26         if (args.length < 2 || args[1] == null) {
27             consoleService.println("Не хватает аргумента");
28             return;
29         }
30         Long id;
31         try {
32             id = Long.valueOf(args[1]);
33         } catch (NumberFormatException e) {
34             consoleService.println("Неверный формат аргумента");
35             return;
36         }
37         MusicBand musicBand = musicBandDAO.getById(id);
38         MusicBandAttributeSetter setter = new MusicBandAttributeSetter(
39             consoleService);
40         try {
41             setter.setAttribute(musicBand,
42                 "Введите название группы",
43                 s -> s.setName(consoleService.read()));
44             setter.setAttribute(musicBand,
45                 "Введите количество участников",
46                 s -> s.setNumberOfParticipants(Integer.valueOf(
47                 consoleService.read())));
48             setter.setAttribute(musicBand,
49                 "Введите описание группы",
50                 s -> s.setDescription(consoleService.read()));
51             setter.setAttribute(musicBand,
52                 String.format("Введите жанр группы %s", Arrays.
53                 toString(MusicGenre.values())),
54                 s -> s.setGenre(consoleService.read())));
55         }
```

```
51         setter.setAttribute(musicBand,
52                         "Введите координату x",
53                         s -> s.getCoordinates().setX(Long.valueOf(
54                             consoleService.read())));
54         setter.setAttribute(musicBand,
55                         "Введите координату y",
56                         s -> s.getCoordinates().setY(Float.valueOf(
57                             consoleService.read())));
57         setter.setAttribute(musicBand,
58                         "Введите имя солиста",
59                         s -> s.getFrontMan().setName(consoleService.read()));
60         setter.setAttribute(musicBand,
61                         "Введите рост солиста",
62                         s -> s.getFrontMan().setHeight(Double.valueOf(
63                             consoleService.read())));
63         setter.setAttribute(musicBand,
64                         String.format("Введите цвет глаз солиста %s", Arrays.
65                             toString(Color.values())),
66                         s -> s.getFrontMan().setEyeColor(consoleService.read
67                             ()));
67         setter.setAttribute(musicBand,
68                         "Введите номер паспорта солиста",
69                         s -> {
70                             String line = consoleService.read();
71                             if (!musicBand.getFrontMan().getPassportID().
72                                 equals(line)) {
73                                 musicBandDAO.checkPassportIDUnique(line);
74                             }
75                             s.getFrontMan().setPassportID(line);
76                         });
77             } catch (InterruptedException e) {
78                 consoleService.println("Обновление элемента прервано");
79                 return;
80             }
81             musicBandDAO.update(musicBand, id);
82             consoleService.println("Элемент успешно обновлен!");
83     }
```

```

1 package org.example.command;
2
3 import org.example.bootstrap.ServiceLocator;
4 import org.example.exception.InterruptScriptException;
5 import org.example.service.IConsoleService;
6
7 import java.io.File;
8 import java.io.FileInputStream;
9 import java.io.IOException;
10 import java.io.InputStream;
11
12 public class ExecuteCommand extends AbstractCommand {
13
14     @Override
15     public String command() {
16         return "execute_script";
17     }
18
19     @Override
20     public String description() {
21         return "Считать и исполнить скрипт из указанного файла";
22     }
23
24     /**
25      * Создает {@link InputStream} из файла, устанавливает его в {@link IConsoleService}
26      * взамен {@code System.in} затем запускает исполнение команд в {@link ServiceLocator}
27      * Исполнение скрипта прерывается, когда из {@link ServiceLocator}
28      * приходит исключение
29      * {@link InterruptScriptException}, после исполнения скрипта
30      * закрывает поток файла и
31      * устанавливает в {@link IConsoleService} консольный поток {@code
32      * System.in}
33      *
34      * {@param args аргументы (1 - путь к файлу со скриптом)}
35      */
36     @Override
37     public void execute(String[] args) {
38         if (args.length < 2 || args[1] == null) {
39             consoleService.println("Не хватает аргумента");
40             return;
41         }
42         File file = new File(args[1]);
43         InputStream inputStream = null;
44         try {
45             inputStream = new FileInputStream(file);
46             consoleService.setInputStream(inputStream);
47             consoleService.println("***Началось выполнение скрипта***");
48             serviceLocator.executeCommands();
49         } catch (InterruptScriptException e) {
50             consoleService.println("***Выполнение скрипта завершено***");
51         } catch (IOException e) {
52             consoleService.println("Ошибка при чтении файла скрипта");
53         }
54     }
55 }

```

```
50     } finally {
51         consoleService.setSystemIn();
52         if (inputStream != null) {
53             try {
54                 inputStream.close();
55             } catch (IOException e) {
56                 e.printStackTrace();
57             }
58         }
59     }
60 }
61 }
62 }
```

```
1 package org.example.command;
2
3 import org.example.model.MusicBand;
4
5 public class MaxByIdCommand extends AbstractCommand {
6
7     @Override
8     public String command() {
9         return "max_by_id";
10    }
11
12    @Override
13    public String description() {
14        return "Вывести любой объект из коллекции, значение поля id
которого является максимальным";
15    }
16
17    @Override
18    public void execute(String[] args) {
19        MusicBand maxId = musicBandDAO.getByMaxId();
20        if (maxId == null) {
21            consoleService.println("Коллекция пустая");
22        } else {
23            consoleService.println(maxId.toString());
24        }
25    }
26}
27
```

```
1 package org.example.command;
2
3 import org.example.enums.SortStatus;
4
5 public class ReorderCommand extends AbstractCommand {
6
7     @Override
8     public String command() {
9         return "reorder";
10    }
11
12    @Override
13    public String description() {
14        return "Отсортировать коллекцию в порядке, обратном нынешнему";
15    }
16
17    @Override
18    public void execute(String[] args) {
19        SortStatus reorder = musicBandDAO.reorder();
20        switch (reorder) {
21            case ASC:
22                consoleService.println("Коллекция отсортирована по
возрастанию");
23                break;
24            case DESC:
25                consoleService.println("Коллекция отсортирована по
убыванию");
26                break;
27            default:
28                break;
29        }
30    }
31 }
32 }
```

```
1 package org.example.command;
2
3 import org.example.bootstrap.ServiceLocator;
4 import org.example.dao.IMusicBandDAO;
5 import org.example.service.IConsoleService;
6 import org.example.service.IFileService;
7
8 import java.util.Map;
9
10 /**
11  * Абстрактный класс консольной команды.
12 */
13 public abstract class AbstractCommand {
14     protected IConsoleService consoleService;
15
16     protected IMusicBandDAO musicBandDAO;
17
18     protected IFileService fileService;
19
20     protected ServiceLocator serviceLocator;
21
22     protected Map<String, AbstractCommand> commands;
23
24     public void init(ServiceLocator serviceLocator) {
25         this.musicBandDAO = serviceLocator.getMusicDAO();
26         this.consoleService = serviceLocator.getConsoleService();
27         this.fileService = serviceLocator.getFileService();
28         this.serviceLocator = serviceLocator;
29         this.commands = serviceLocator.getCommands();
30     }
31
32     /**
33      * return имя команды (как она вводится в консоль)
34      */
35     public abstract String command();
36
37     /**
38      * return описание команды
39      */
40     public abstract String description();
41
42     /**
43      * Выполнение команды
44      *
45      * param args аргументы
46      * throws Exception ошибки при выполнении команды
47      */
48     public abstract void execute(String[] args) throws Exception;
49 }
50
```

```

1 package org.example.command;
2
3 import org.example.enums.Color;
4 import org.example.enums.MusicGenre;
5 import org.example.exception.InterruptInputException;
6 import org.example.model.Coordinates;
7 import org.example.model.MusicBand;
8 import org.example.model.Person;
9 import org.example.util.MusicBandAttributeSetter;
10
11 import java.util.Arrays;
12
13 public class AddIfMinCommand extends AbstractCommand {
14
15     @Override
16     public String command() {
17         return "add_if_min";
18     }
19
20     @Override
21     public String description() {
22         return "Добавить новый элемент в коллекцию, если его значение
меньше, чем у наименьшего элемента этой коллекции";
23     }
24     /**
25      * Использует {@link MusicBandAttributeSetter} для наполнения объекта
26      * {@link MusicBand}
27      * и добавляет его в коллекцию, если он самый меньший в коллекции или
28      * она пуста
29      * @param args аргументы
30     */
31     @Override
32     public void execute(String[] args) {
33         MusicBand musicBand = new MusicBand();
34         musicBand.setFrontMan(new Person());
35         musicBand.setCoordinates(new Coordinates());
36         MusicBandAttributeSetter setter = new MusicBandAttributeSetter(
37             consoleService);
38         try {
39             setter.setAttribute(musicBand,
40                 "Введите название группы",
41                 s -> s.setName(consoleService.read()));
42             setter.setAttribute(musicBand,
43                 "Введите количество участников",
44                 s -> s.setNumberOfParticipants(Integer.valueOf(
45                     consoleService.read())));
46             setter.setAttribute(musicBand,
47                 "Введите описание группы",
48                 s -> s.setDescription(consoleService.read()));
49             setter.setAttribute(musicBand,
50                 String.format("Введите жанр группы %s", Arrays.
51                     toString(MusicGenre.values())),
52                 s -> s.setGenre(consoleService.read()));
53             setter.setAttribute(musicBand,
54

```

```
49                     "Введите координату x",
50                     s -> s.getCoordinates().setX(Long.valueOf(
51                         consoleService.read())));
51                     setter.setAttribute(musicBand,
52                         "Введите координату у",
53                         s -> s.getCoordinates().setY(Float.valueOf(
54                         consoleService.read())));
54                     setter.setAttribute(musicBand,
55                         "Введите имя солиста",
56                         s -> s.getFrontMan().setName(consoleService.read()));
57                     setter.setAttribute(musicBand,
58                         "Введите рост солиста",
59                         s -> s.getFrontMan().setHeight(Double.valueOf(
60                         consoleService.read())));
60                     setter.setAttribute(musicBand,
61                         String.format("Введите цвет глаз солиста %s", Arrays.
62                         toString(Color.values())),
62                         s -> s.getFrontMan().setEyeColor(consoleService.read
63                         ()));
63                     setter.setAttribute(musicBand,
64                         "Введите номер паспорта солиста",
65                         s -> {
66                             String line = consoleService.read();
67                             musicBandDAO.checkPassportIDUnique(line);
68                             s.getFrontMan().setPassportID(line);
69                         });
70             } catch (InterruptedException e) {
71                 consoleService.println("Добавление элемента прервано");
72                 return;
73             }
74             MusicBand minimal = musicBandDAO.getMinimal();
75             if (minimal == null || musicBand.compareTo(minimal) < 0) {
76                 musicBandDAO.save(musicBand);
77                 consoleService.println("Элемент успешно добавлен!");
78             } else {
79                 consoleService.println("Введенный элемент больше минимального
");
80             }
81         }
82     }
83 }
```

```
1 package org.example.command;
2
3 import org.example.model.MusicBand;
4
5 public class RemoveLastCommand extends AbstractCommand {
6
7     @Override
8     public String command() {
9         return "remove_last";
10    }
11
12    @Override
13    public String description() {
14        return "Удалить последний элемент из коллекции";
15    }
16
17    @Override
18    public void execute(String[] args) {
19        MusicBand musicBand = musicBandDAO.removeLast();
20        if (musicBand == null) {
21            consoleService.println("Коллекция пустая");
22        } else {
23            consoleService.println("Удален элемент с id = " + musicBand.
24                getId());
25        }
26    }
27}
```

```
1 package org.example.command;
2
3 public class FilterByDescriptionCommand extends AbstractCommand {
4
5     @Override
6     public String command() {
7         return null;
8     }
9
10    @Override
11    public String description() {
12        return "Вывести элементы, значение поля description которых меньше
13        заданного";
14    }
15
16    @Override
17    public void execute(String[] args) {
18        if (args.length < 2 || args[1] == null) {
19            consoleService.println("Не хватает аргумента");
20            return;
21        }
22        musicBandDAO.filterByDescription(args[1]).forEach(s -> {
23            if (s != null)
24                consoleService.println(s.toString());
25        });
26    }
27}
```

```
1 package org.example.command;
2
3 import org.example.model.MusicBand;
4
5 import java.util.List;
6
7 public class RemoveByDescriptionCommand extends AbstractCommand {
8
9     @Override
10    public String command() {
11        return "remove_any_by_description";
12    }
13
14    @Override
15    public String description() {
16        return "Удалить из коллекции один элемент, значение поля
description которого эквивалентно заданному";
17    }
18
19    @Override
20    public void execute(String[] args) {
21        if (args.length < 2 || args[1] == null) {
22            consoleService.println("Не хватает аргумента");
23            return;
24        }
25        String descr = "";
26        for (int i = 1; i < args.length; i++) {
27            descr = descr.concat(args[i]).concat(" ");
28        }
29
30        List<MusicBand> musicBands = musicBandDAO.removeByDescription(
descr.trim());
31        if (musicBands.isEmpty()) {
32            consoleService.println("Совпадений не найдено");
33        } else {
34            consoleService.println("Удалены объекты:");
35            musicBands.forEach(s -> {
36                if (s != null)
37                    consoleService.println(s.toString());
38            });
39        }
40    }
41 }
42
```

```

1 package org.example.service;
2
3 import com.fasterxml.jackson.databind.ObjectMapper;
4 import com.fasterxml.jackson.dataformat.xml.XmlMapper;
5 import com.fasterxml.jackson.datatype.jsr310.JavaTimeModule;
6 import org.example.model.DataStorage;
7
8 import java.io.*;
9
10 /**
11  * Реализация сервиса работы с файлом. Хранит путь к файлу и сервис работы
12  * с консолью
13 */
14 public class FileService implements IFileService {
15     private String path;
16     private IConsoleService consoleService;
17
18     public FileService(String path, IConsoleService consoleService) {
19         this.path = path;
20         this.consoleService = consoleService;
21     }
22
23     /**
24      * Создает {@link FileOutputStream} из файла, оборачивает его в {@link
25      * PrintWriter}
26      * использует его в {@link XmlMapper} для записи объекта в xml файл.
27      * Отлавливает возникающие ошибки и выводит их в {@link
28      * IConsoleService}
29      *
30      * @param musicBand объект для записи
31      * @return флаг успешности операции
32      */
33     @Override
34     public boolean write(DataStorage musicBand) {
35         final ObjectMapper mapper = new XmlMapper();
36         mapper.registerModule(new JavaTimeModule());
37         final File file = new File(path);
38         try (final FileOutputStream outputStream = new FileOutputStream(
39             file)) {
40             if (!file.exists()) {
41                 file.getParentFile().mkdirs();
42                 file.createNewFile();
43             }
44             PrintWriter printWriter = new PrintWriter(outputStream);
45             mapper.writerWithDefaultPrettyPrinter().writeValue(printWriter
46             , musicBand);
47             return true;
48         } catch (FileNotFoundException e) {
49             consoleService.println("Файл не найден");
50         } catch (IOException e) {
51             consoleService.println("Ошибка при записи в файл");
52         }
53         return false;
54     }

```

```
50
51     /**
52      * Создает {@link FileInputStream} из файла, оборачивает его в {@link
53      * InputStreamReader}
54      * использует его в {@link XmlMapper} для чтения объекта из xml файла
55      *
56      * @return объект, считанный из файла (null в случае возникновения
57      * ошибок)
58      */
59     @Override
60     public DataStorage readFromXml() {
61         final ObjectMapper mapper = new XmlMapper();
62         mapper.registerModule(new JavaTimeModule());
63         File file = new File(path);
64         InputStreamReader inputStreamReader;
65         try (FileInputStream inputStream = new FileInputStream(file)) {
66             inputStreamReader = new InputStreamReader(inputStream);
67             return mapper.readValue(inputStreamReader, DataStorage.class
68         );
69         } catch (FileNotFoundException e) {
70             consoleService.println("Файл не найден");
71         } catch (IOException e) {
72             consoleService.println("Ошибка при чтении файла");
73             e.printStackTrace();
74         }
75     }
76 }
```

```
1 package org.example.service;
2
3 import org.example.model.DataStorage;
4
5 /**
6  * Интерфейс сервиса работы с файлом.
7  */
8 public interface IFileService {
9
10    boolean write(DataStorage musicBand);
11
12    DataStorage readFromXml();
13 }
14
```

```
1 package org.example.service;
2
3 import org.example.exception.InterruptScriptException;
4
5 import java.io.BufferedReader;
6 import java.io.IOException;
7 import java.io.InputStream;
8 import java.io.InputStreamReader;
9
10 /**
11  * Реализация сервиса работы с пользовательским вводом-выводом.
12 */
13 public class ConsoleService implements IConsoleService {
14     private BufferedReader reader = new BufferedReader(new
15         InputStreamReader(System.in));
16     private boolean isSystemIn = true;
17
18     public void setInputStream(InputStream inputStream) {
19         this.isSystemIn = false;
20         this.reader = new BufferedReader(new InputStreamReader(inputStream
21     ));
22     }
23
24     public void setSystemIn() {
25         this.isSystemIn = true;
26         this.reader = new BufferedReader(new InputStreamReader(System.in
27     ));
28     }
29
30     @Override
31     public void print(String message) {
32         System.out.print(message);
33     }
34
35     @Override
36     public void println(String message) {
37         System.out.println(message);
38     }
39
40     @Override
41     public String read() {
42         String s = null;
43         try {
44             if (!isSystemIn && !reader.ready()) {
45                 throw new InterruptScriptException();
46             }
47             s = reader.readLine();
48             if (!isSystemIn) {
49                 println(s);
50             }
51         } catch (IOException e) {
52             println("Ошибка потока данных");
53         }
54         return s;
```

```
52      }
53  }
54
```

```
1 package org.example.service;
2
3 import java.io.InputStream;
4
5 /**
6  * Интерфейс сервиса работы с пользовательским вводом-выводом.
7  */
8 public interface IConsoleService {
9
10    void print(String message);
11
12    void println(String message);
13
14    String read();
15
16    void setInputStream(InputStream inputStreamReader);
17
18    void setSystemIn();
19 }
20
```

```

1 package org.example.bootstrap;
2
3 import org.example.command.*;
4 import org.example.dao.IMusicBandDAO;
5 import org.example.dao.MusicBandDAO;
6 import org.example.exception.InterruptScriptException;
7 import org.example.model.DataStorage;
8 import org.example.service.ConsoleService;
9 import org.example.service.FileService;
10 import org.example.service.IConsoleService;
11 import org.example.service.IFileService;
12
13 import java.util.HashMap;
14 import java.util.Map;
15
16 /**
17 * Класс-загрузчик, используется для старта приложения и доступа к
18 * необходимым сервисам.
19 */
20 public class Bootstrap implements ServiceLocator {
21     private IMusicBandDAO musicDAO = new MusicBandDAO();
22     private IConsoleService consoleService = new ConsoleService();
23     private IFileService fileService = new FileService("", consoleService
24 );
25     private Map<String, AbstractCommand> commands = new HashMap<>();
26
27     /**
28      * Запускает приложение
29      *
30      * @param args аргументы приложения
31      */
32     public void start(String[] args) {
33         if (args != null && args.length > 0) {
34             initCollection(args[0]);
35         } else {
36             consoleService.println("Отсутствует аргумент с адресом файла,
37 коллекция не загружена");
38         }
39         initCommands();
40         consoleService.println("***WELCOME TO MUSIC BAND COLLECTION***");
41         executeCommands();
42     }
43
44     /**
45      * Начинает исполнение введенных с консоли команд в цикле с условием
46      * выхода введенная строка = "exit"
47      *
48      * @throws InterruptScriptException перебрасывается с уровня выше
49      */
50     public void executeCommands() {
51         String command = "";
52         while (!command.equals("exit")) {
53             try {
54                 String line = consoleService.read();
55             }
56         }
57     }
58 }
```

```

51             if (line != null) {
52                 String[] params = line.split(" ");
53                 command = params[0];
54                 if (!commands.containsKey(command)) {
55                     consoleService.println("Такой команды не
56                     существует, наберите help для справки");
57                 } else {
58                     commands.get(command).execute(params);
59                 }
60             } catch (InterruptScriptException e) {
61                 throw e;
62             } catch (Exception e) {
63                 consoleService.println(e.getMessage());
64                 e.printStackTrace();
65             }
66         }
67     }
68
69 /**
70 * Инициализирует коллекцию данными из файла
71 *
72 * @param arg путь к файлу
73 */
74 private void initCollection(String arg) {
75     fileService = new FileService(arg, consoleService);
76     DataStorage dataStorage = fileService.readFromXml();
77     if (dataStorage != null) {
78         musicDAO.init(dataStorage);
79     }
80 }
81
82 /**
83 * Инициализирует команды
84 */
85 private void initCommands() {
86     registryCommand(new HelpCommand());
87     registryCommand(new ExitCommand());
88     registryCommand(new AddCommand());
89     registryCommand(new AddIfMinCommand());
90     registryCommand(new ClearCommand());
91     registryCommand(new ExecuteCommand());
92     registryCommand(new FilterByDescriptionCommand());
93     registryCommand(new InfoCommand());
94     registryCommand(new MaxByIdCommand());
95     registryCommand(new RemoveByDescriptionCommand());
96     registryCommand(new RemoveCommand());
97     registryCommand(new RemoveLastCommand());
98     registryCommand(new ReorderCommand());
99     registryCommand(new SaveCommand());
100    registryCommand(new ShowCommand());
101    registryCommand(new UpdateCommand());
102 }
103

```

```
104     /**
105      * Добавляет объект команды в мапу
106     */
107    private void registryCommand(final AbstractCommand command) {
108        if (command.command() == null || command.command().isEmpty()) {
109            return;
110        }
111        command.init(this);
112        commands.put(command.command(), command);
113    }
114
115    public IConsoleService getConsoleService() {
116        return consoleService;
117    }
118
119    public void setConsoleService(IConsoleService consoleService) {
120        this.consoleService = consoleService;
121    }
122
123    public Map<String, AbstractCommand> getCommands() {
124        return commands;
125    }
126
127    public void setCommands(Map<String, AbstractCommand> commands) {
128        this.commands = commands;
129    }
130
131    public IFileService getFileService() {
132        return fileService;
133    }
134
135    public void setFileService(IFileService fileService) {
136        this.fileService = fileService;
137    }
138
139    public IMusicBandDAO getMusicDAO() {
140        return musicDAO;
141    }
142
143    public void setMusicDAO(IMusicBandDAO musicDAO) {
144        this.musicDAO = musicDAO;
145    }
146 }
147
```

```
1 package org.example.bootstrap;
2
3 import org.example.command.AbstractCommand;
4 import org.example.dao.IMusicBandDAO;
5 import org.example.service.IConsoleService;
6 import org.example.service.IFileService;
7
8 import java.util.Map;
9
10 public interface ServiceLocator {
11     IConsoleService getConsoleService();
12
13     IFileService getFileService();
14
15     IMusicBandDAO getMusicDAO();
16
17     Map<String, AbstractCommand> getCommands();
18
19     void executeCommands();
20 }
21
```

```
1 package org.example.exception;  
2  
3 public class InterruptInputException extends RuntimeException{  
4 }  
5
```

```
1 package org.example.exception;  
2  
3 public class InterruptScriptException extends RuntimeException{  
4 }  
5
```

```
1 package org.example.exception;
2
3 public class MusicBandNotFoundException extends RuntimeException{
4     public MusicBandNotFoundException(String s) {
5         super(s);
6     }
7 }
8
```

```
1 package org.example.exception;
2
3 public class MusicBandWrongAttributeException extends RuntimeException{
4     public MusicBandWrongAttributeException(String s) {
5         super(s);
6     }
7 }
8
```