SOFTWARE ENGINEERING FOUNDATIONS

FLORIAN BERGMANN
PERSON ID: H00020398

Assessment One: Cabin Manager

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

Part I

CREATION OF A CABIN MANAGER

1 INTRODUCTION

This report shall provide detailed information about the implementation of the cabin manager application.

1.1 ATTRIBUTES

Apart from the mandatory attributes, the following attributes were chosen to be implemented as well:

SIZE: The area of available space in the cabin.

CONDITION: The condition the cabin is in. Only the following values are allowed: `PERFECT`, `GOOD`, `FAIR`, `BAD`, `IN_SHAMBLES`, `UNKNOWN`. The limitation to these attributes is achieved by utilizing an `enumeration`.

1.2 COST CALCULATION

The calculation of a cabin's cost for one night are calculated according to the following formula:

```
BASIC_COST + CONDITION_COST + FACILITIES_COST + SIZE_COST + (BED_TO_ROOM_RATIO * BED_TO_ROOM_MULTIPLIER)
```

`BASIC_COST` and `BED_TO_ROOM_MULTIPLIER` are constants that can be set in the `cabin` class.

The costs associated with the `condition`, the `facilities` and the `size` are stored in a separate class (`PriceMapping`) and can be adjusted there. The prices are stored in five discreet values in a `enumeration` (`PriceList`).

1.3 FREQUENCY REPORTS

The frequency-report provided outputs the number of cabins of a certain condition.

E.g. if two cabins are of the condition "`IN_SHAMBLES`" and one is of the condition "`GOOD`" the output would be as follows:

BAD	FAIR	GOOD	IN_SHAMBLES	PERFECT
0	0	1	2	0

1.4 STATUS REPORT

Even though not well-designed the application should meet the specification fully as all requirements were implemented and tested to function even in the case of incorrect input.

2 DIAGRAMS

2.1 CLASS DIAGRAM

The provided class diagram does not display `getters` and `setters` even though they are present for every non-final attribute present in the diagram.

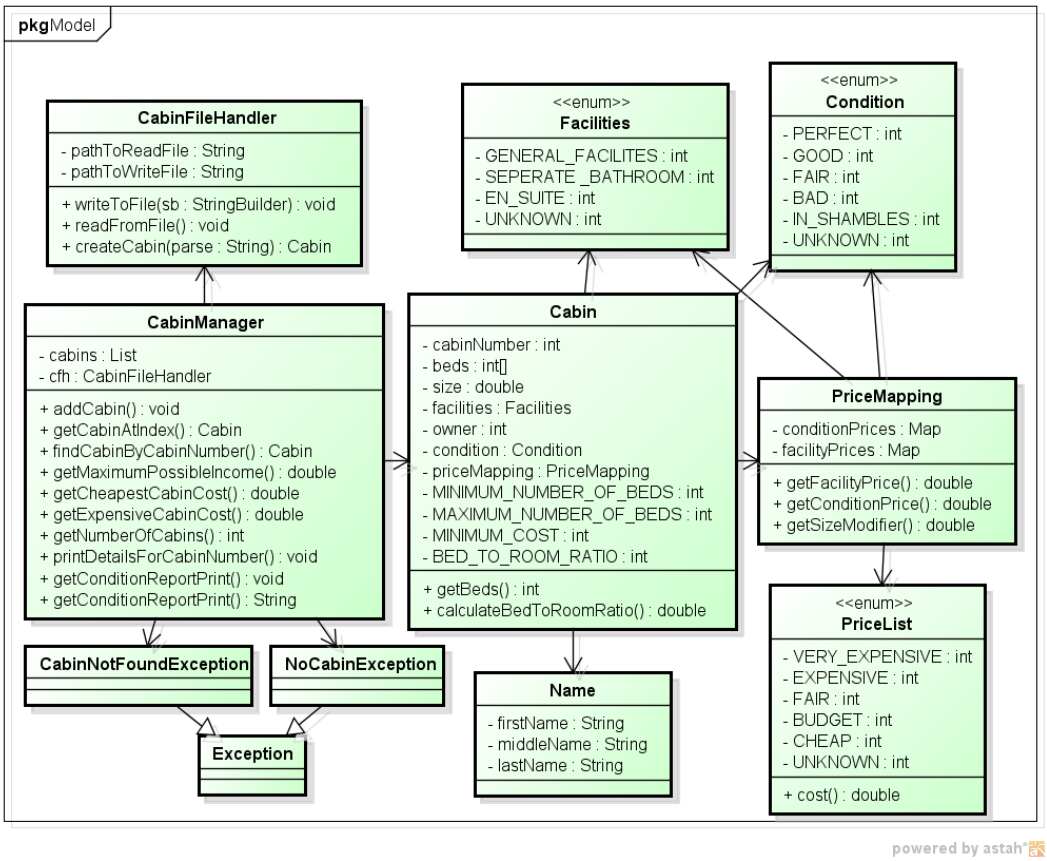

Figure 1: Class diagram of cabin manager

2.2 SEQUENCE DIAGRAM

The provided sequence diagram shows how the frequency report is generated and then printed to a file.

Even though the real implementation prints all other details as well (cabin details, overview, frequency report, most expensive & cheapest cabin) these have been left out of the sequence diagram to improve its clarity.

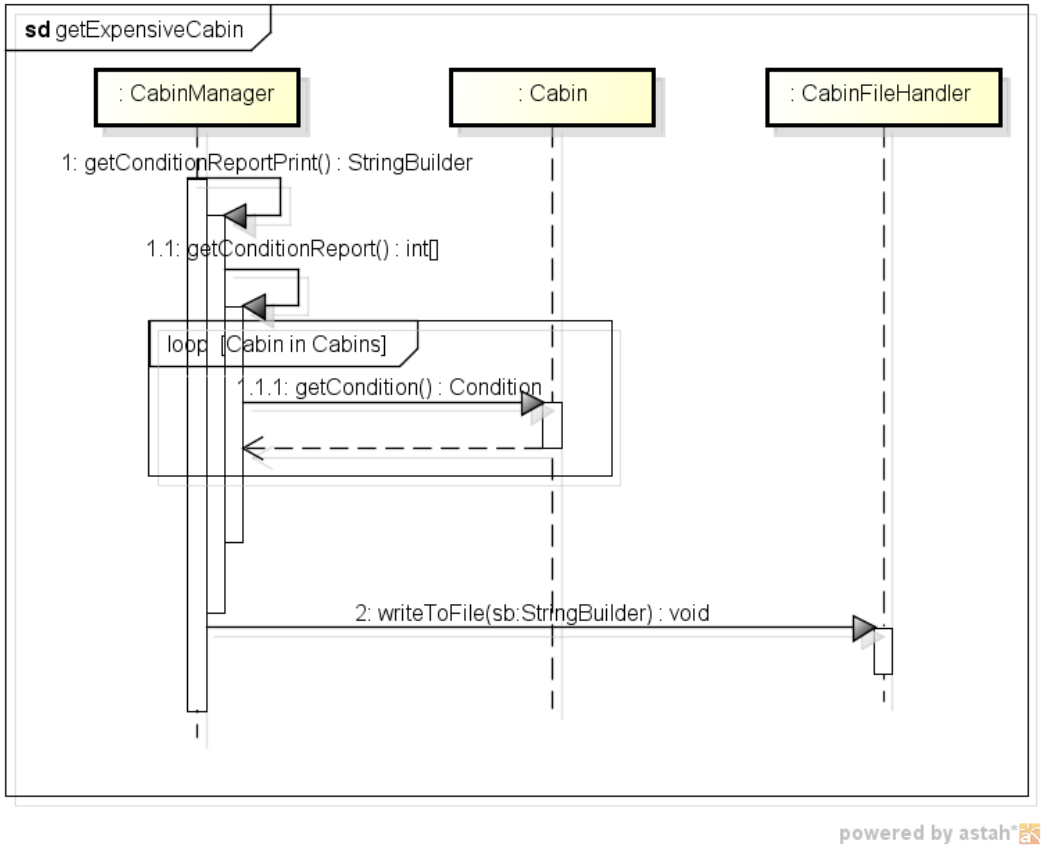

Figure 2: Sequence diagram of the printing of the frequency report

3

SOURCE CODE

Package: uk.heriotwatt.sef.model

```java
package uk.heriotwatt.sef.model;

/**
 * Stores the values associated with a cabin.
 *
 * @author Florian Bergmann
 *
 */
public class Cabin {

	private int cabinNumber;
	private int[] beds;
	private double size;
	private Facilities facilities;
	private Name owner;
	private Condition condition;

	private PriceMapping data;

	private final int MINIMUM_NUMBER_OF_BEDS = 2;
	private final int MAXIMUM_NUMBER_OF_BEDS = 8;
	private final double BASIC_COST = 10;
	private final int BED_TO_ROOM_RATIO_MULTIPLIER = 5;

	/**
	 * Creates a new cabin object without values.
	 */
	public Cabin() {
		this.data = new PriceMapping();
	}

	/**
	 * Creates a new cabin object with the specified values.
	 *
	 * @param cabinNumber
	 *            The cabin number.
	 * @param beds
	 *            The array of beds in the cabin..
	 * @param size
	 *            The size of the cabin.
	 * @param facilities
```

```
42          *            The facilites of the cabin.
43          * @param owner
44          *            The owner of the cabin.
45          * @param condition
46          *            The condition of the cabin.
47          */
48         public Cabin(int cabinNumber, int[] beds, double size,
49                         Facilities facilities, Name owner, Condition condition) {
50                 super();
51                 this.cabinNumber = cabinNumber;
52                 this.beds = beds;
53                 this.size = size;
54                 this.facilities = facilities;
55                 this.owner = owner;
56                 this.condition = condition;
57                 this.data = new PriceMapping();
58         }
59
60         /*
61          * Getters and setters
62          */
63
64         /**
65          * Returns the cabin number.
66          *
67          * @return The number of cabins stored in the manager.
68          */
69         public int getCabinNumber() {
70                 return cabinNumber;
71         }
72
73         /**
74          * Sets the cabin number.
75          *
76          * @param cabinNumber
77          *            Number to be set.
78          */
79         public void setCabinNumber(int cabinNumber) {
80                 this.cabinNumber = cabinNumber;
81         }
82
83         /**
84          * Returns the beds of the cabin as array. Each array-cell represents a
85          * room, each value of a cell the number of beds in the room.
86          *
87          * @return Array of beds.
88          */
89         public int[] getNumberOfBeds() {
90                 return beds;
91         }
92
```

```
93          /**
94           * Sets the beds.
95           *
96           * @param numberOfBeds
97           *            The new array of beds.
98           */
99          public void setNumberOfBeds(int[] numberOfBeds) {
100                 if (numberOfBeds.length > 0) {
101                         int bedsInArray = this.calculateNumberOfBeds(numberOfBeds);
102                         if (bedsInArray >= MINIMUM_NUMBER_OF_BEDS
103                                         && bedsInArray <= MAXIMUM_NUMBER_OF_BEDS) {
104                                 this.beds = numberOfBeds;
105                         } else {
106                                 throw new IllegalArgumentException(
107                                                 String.format(
108                                                                 "Only between %d and %
                                                                     d beds can be
                                                                     placed in a cabin.
                                                                     ",
109                                                                 MINIMUM_NUMBER_OF_BEDS
                                                                     ,
                                                                     MAXIMUM_NUMBER_OF_BEDS
                                                                     ));
110                         }
111                 } else {
112                         throw new IllegalArgumentException(
113                                         "The number of beds must be greater than 0.");
114                 }
115         }
116
117         /**
118          * Returns the facilities of the cabin.
119          *
120          * @return The facilities of the cabin.
121          */
122         public Facilities getFacilities() {
123                 return facilities;
124         }
125
126         /**
127          * Attempts to set the facilites of the cabin.
128          *
129          * @param facilities
130          */
131         public void setFacilities(Facilities facilities) {
132                 this.facilities = facilities;
133         }
134
135         /**
136          * Returns the owner's name.
137          *
```

```
138             * @return The owner of the cabin.
139             */
140            public Name getOwner() {
141                    return owner;
142            }
143
144            /**
145             * Sets the owner of the cabin.
146             *
147             * @param owner
148             */
149            public void setOwner(Name owner) {
150                    this.owner = owner;
151            }
152
153            /**
154             * Returns the size of the cabin.
155             *
156             * @return The size of the cabin.
157             */
158            public double getSize() {
159                    return size;
160            }
161
162            /**
163             * Sets the size of the cabin.
164             *
165             * @param size
166             *            The new size (must be bigger than 0)
167             */
168            public void setSize(double size) {
169                    if (size >= 0) {
170                            this.size = size;
171                    } else {
172                            throw new IllegalArgumentException("Size must be positive.");
173                    }
174            }
175
176            /**
177             * The cost is calculated based on different factors: - The condition. - The
178             * facilities. - The size. - The beds/rooms present (The less beds per room
179             * the more expensive). The values associated with the first three are
180             * stored in {@link PriceMapping}
181             *
182             * @return The cost if the cabin.
183             */
184            public double getCost() {
185                    double cost = BASIC_COST;
186
187                    double conditionModifier = this.data.getConditionPrice(this.condition)
                         ;
```

```
188                 double faciltiesModifier = this.data.getFacilityPrice(this.facilities)
                        ;
189                 double sizeModifier = this.data.getSizeModifier(this.size);
190                 double bedToRoomRatio = this.calculateRoomToBedRatio();
191
192                 cost = BASIC_COST + conditionModifier + faciltiesModifier
193                                 + sizeModifier
194                                 + (BED_TO_ROOM_RATIO_MULTIPLIER * bedToRoomRatio);
195
196                 return cost;
197         }
198
199         /**
200          * Returns the cabins condition.
201          *
202          * @return The condition.
203          */
204         public Condition getCondition() {
205                 return condition;
206         }
207
208         /**
209          * Sets the condition of the cabin.
210          *
211          * @param condition
212          *            New condition to be set.
213          */
214         public void setCondition(Condition condition) {
215                 this.condition = condition;
216         }
217
218         /**
219          * Returns the number of beds in a cabin.
220          *
221          * @return The number of beds in the cabin.
222          */
223         public int getBeds() {
224                 return this.calculateNumberOfBeds(this.beds);
225         }
226
227         /**
228          * Calculates the room to bed ratio.
229          *
230          * @return The room to bed ratio.
231          */
232         public double calculateRoomToBedRatio() {
233                 int rooms = this.getNumberOfBeds().length;
234                 int beds = this.calculateNumberOfBeds(this.beds);
235                 double bedToRoomRatio = rooms / beds;
236                 return bedToRoomRatio;
237         }
```

```
238
239         private int calculateNumberOfBeds(int[] numberOfBeds) {
240                 int result = 0;
241                 for (int i : numberOfBeds) {
242                         result += i;
243                 }
244                 return result;
245         }
246 }
```

Listing 1: uk.heriotwatt.sef.model.Cabin.java

```
1  package uk.heriotwatt.sef.model;
2
3  import java.util.ArrayList;
4  import java.util.Formatter;
5  import java.util.List;
6  import java.util.Locale;
7
8  public class CabinManager {
9
10         private List<Cabin> cabins;
11
12         private CabinFileHandler cfh;
13
14         /**
15          * Creates a new cabin manager object from the provided arguments.
16          *
17          * @param cfh
18          *             The filehandler that loads cabins from a file and saves
19          *             reports to a file.
20          */
21         public CabinManager(CabinFileHandler cfh) {
22                 this.cabins = new ArrayList<Cabin>();
23                 this.cfh = cfh;
24                 this.cabins = this.cfh.readFromFile();
25         }
26
27         public void addCabin(Cabin cab) {
28                 this.cabins.add(cab);
29         }
30
31         /**
32          * Attempts to find a cabin with the provided cabinNumber in the cabin-List.
33          *
34          * @param cabinNumber
35          *             The cabinnumber of the cabin to be returned
36          * @return The first cabin in the list with the corresponding cabinnumber.
37          * @throws CabinNotFoundException
38          *              If no cabin with the provided number could be found.
39          */
```

```
40          public Cabin findCabinByCabinNumber(int cabinNumber)
41                          throws CabinNotFoundException {
42                  Cabin cabinFound = null;
43                  for (Cabin cabin : this.cabins) {
44                          if (cabin.getCabinNumber() == cabinNumber) {
45                                  cabinFound = cabin;
46                                  break;
47                          }
48                  }
49                  if (cabinFound != null) {
50                          return cabinFound;
51                  } else {
52                          throw new CabinNotFoundException(String.format(
53                                          "The cabin with number %d was not in the list.
                                                ",
54                                          cabinNumber));
55                  }
56          }
57
58          /**
59           * Retrieves that cabin at the specified position in the cabin list.
60           *
61           * @param index
62           *            The position for which the cabin should be returned.
63           * @return Tha cabin.
64           */
65          public Cabin getCabinAtIndex(int index) {
66                  if (index < this.getNumberOfCabins()) {
67                          return this.cabins.get(index);
68                  } else {
69                          throw new IndexOutOfBoundsException();
70                  }
71          }
72
73          /**
74           * Returns the maximum possible income that could be achieved. Therefore the
75           * cost for all cabins are added up.
76           *
77           * @return The added cost of all cabins.
78           */
79          public double getMaximumPossibleIncome() {
80                  double result = 0;
81                  for (Cabin cabin : this.cabins) {
82                          result += cabin.getCost();
83                  }
84                  return result;
85          }
86
87          /**
88           * Returns the cost for the cheapest cabin.
89           *
```

```
90          * @return The cost of the cheapest cabin.
91          * @throws NoCabinsException
92          */
93         public double getCheapestCabinCost() throws NoCabinsException {
94                 // TODO: Empty array;
95                 if (this.cabins.size() > 0) {
96                         Cabin cheapestCab = null;
97                         for (Cabin cab : this.cabins) {
98                                 if (cheapestCab == null) {
99                                         cheapestCab = cab;
100                                }
101                                if (cab.getCost() < cheapestCab.getCost()) {
102                                        cheapestCab = cab;
103                                }
104                        }
105                        return cheapestCab.getCost();
106                } else {
107                        throw new NoCabinsException(
108                                        "There are no cabins present. Insert cabins
                                            first.");
109                }
110
111        }
112
113        /**
114         * Returns the cost for the most expensive cabin.
115         *
116         * @return The cost of the most expensive cabin.
117         * @throws NoCabinsException
118         */
119        public double getExpensiveCabinCost() throws NoCabinsException {
120                if (this.cabins.size() > 0) {
121                        Cabin expensiveCab = null;
122                        for (Cabin cab : this.cabins) {
123                                if (expensiveCab == null) {
124                                        expensiveCab = cab;
125                                }
126                                if (cab.getCost() > expensiveCab.getCost()) {
127                                        expensiveCab = cab;
128                                }
129                        }
130                        return expensiveCab.getCost();
131                } else {
132                        throw new NoCabinsException(
133                                        "There are no cabins present. Insert cabins
                                            first.");
134                }
135        }
136
137        /**
138         * Returns the number of cabins currently registered in the list.
```

```
139          *
140          * @return The number of cabins.
141          */
142         public int getNumberOfCabins() {
143                 return this.cabins.size();
144         }
145 
146         /**
147          * Acquires all information from the reports and prints the to a file.
148          */
149         public void printReportsToFile() {
150                 String printString = "";
151                 printString += "OVERVIEW OF CABIN DETAILS:\n\n";
152                 StringBuilder sb = getAllCabinDetails();
153                 printString += sb.toString();
154                 printString += "SINGLE CABIN INFORMATION: \n\n";
155                 for (Cabin cab : this.cabins) {
156                         StringBuilder db = this.getCabinDetails(cab);
157                         printString += db.toString();
158                 }
159                 try {
160                         printString += "MOST EXPENSIVE CABIN: "
161                                         + this.getExpensiveCabinCost() + "\n\n";
162                         printString += "CHEAPEST CABIN: " + this.getCheapestCabinCost
                                ()
163                                         + "\n\n";
164                 } catch (NoCabinsException e) {
165                         // TODO Auto-generated catch block
166                         e.printStackTrace();
167                 }
168                 printString += "MAXIMUM INCOME PER NIGHT: "
169                                 + this.getMaximumPossibleIncome() + "\n\n";
170                 printString += "CONDITION REPORT: \n\n";
171                 printString += this.getConditionReportPrint().toString();
172                 cfh.writeToFile(printString);
173         }
174 
175         /**
176          * Prints the details of one cabin.
177          *
178          * @param cab
179          *            The cabin which details should be printed.
180          */
181         public void printCabDetails(Cabin cab) {
182                 StringBuilder sb = getCabinDetails(cab);
183                 System.out.println(sb.toString());
184         }
185 
186         /**
187          * Prints the details for all cabins to the standard output.
188          */
```

```
189         public void printAllCabins() {
190                 StringBuilder sb = getAllCabinDetails();
191                 System.out.println(sb.toString());
192         }
193 
194         /**
195          * Prints the details of a specific cabin that is specified by its cabin
196          * number.
197          *
198          * @param cabinNumber
199          *            The cabin number of the cabin whose details should be printed.
200          */
201         public void printDetailsForCabinNumber(int cabinNumber) {
202                 try {
203                         Cabin cab = this.findCabinByCabinNumber(cabinNumber);
204                         this.printCabDetails(cab);
205                 } catch (CabinNotFoundException e) {
206                         System.out
207                                         .println(String
208                                                         .format("Could not find the
                                                             cabin for number %d. No
                                                             details printed.",
209                                                                     cabinNumber));
210                 }
211         }
212 
213         /**
214          * Returns a formatted condition report.
215          *
216          * @return A stringbuilder containing the formatted condition report.
217          */
218         public StringBuilder getConditionReportPrint() {
219                 int[] conRep = getConditionReport();
220                 StringBuilder sb = new StringBuilder();
221                 Formatter formatter = new Formatter(sb, Locale.UK);
222                 formatter.format("%1$15s | %2$15s | %3$15s | %4$15s | %5$15s %n",
223                                 Condition.BAD.toString(), Condition.FAIR.toString(),
224                                 Condition.GOOD.toString(), Condition.IN_SHAMBLES.
                                     toString(),
225                                 Condition.PERFECT.toString(), Condition.UNKNOWN.
                                     toString());
226                 formatter.format("%1$15d | %2$15d | %3$15d | %4$15d | %5$15d %n",
227                                 conRep[0], conRep[1], conRep[2], conRep[3], conRep[4],
228                                 conRep[5]);
229                 formatter.format("%n");
230                 return sb;
231         }
232 
233         /**
234          * Returns the values of the condition report.
235          *
```

```
236          * @return String array containing the number of cabins of a certain
237          *         condition.
238          */
239         public int[] getConditionReport() {
240                 int size = Condition.values().length;
241                 int[] frequencyOfConditions = new int[size];
242                 for (Cabin cabin : this.cabins) {
243                         switch (cabin.getCondition()) {
244                         case BAD:
245                                 frequencyOfConditions[0]++;
246                                 break;
247                         case FAIR:
248                                 frequencyOfConditions[1]++;
249                                 break;
250                         case GOOD:
251                                 frequencyOfConditions[2]++;
252                                 break;
253                         case IN_SHAMBLES:
254                                 frequencyOfConditions[3]++;
255                                 break;
256                         case PERFECT:
257                                 frequencyOfConditions[4]++;
258                                 break;
259                         case UNKNOWN:
260                                 frequencyOfConditions[5]++;
261                                 break;
262                         default:
263                                 break;
264                         }
265                 }
266                 return frequencyOfConditions;
267         }
268
269         /**
270          * Returns the details of one cabin.
271          *
272          * @param cab
273          *            The cabin which details should be returned
274          * @return A stringbuilder with formatted output.
275          */
276         private StringBuilder getCabinDetails(Cabin cab) {
277                 StringBuilder sb = new StringBuilder();
278                 Formatter formatter = new Formatter(sb, Locale.UK);
279                 formatter
280                                 .format("%1$10s | %2$15s | %3$20s | %4$15s | %5$5s |
                                        %6$5s | %7$5s | %8$5s %n",
281                                                 "NUMBER", "OWNER", "FACILITIES", "
                                                    CONDITION", "BEDS",
282                                                 "ROOMS", "SIZE", "COST");
283                 formatter
```

```
284                                 .format("%1$10d | %2$15s | %3$20s | %4$15s | %5$5d |
                                        %6$5d | %7$5.2f | %8$5.2f %n",
285                                             cab.getCabinNumber(), cab.getOwner()
286                                                                     .getFirstAndLastName()
                                                                        , cab.
                                                                        getFacilities()
287                                                                     .toString().
                                                                        toLowerCase(), cab
                                                                        .getCondition()
288                                                                     .toString().
                                                                        toLowerCase(), cab
                                                                        .getBeds(), cab
289                                                                     .getNumberOfBeds().
                                                                        length, cab.
                                                                        getSize(), cab
290                                                                     .getCost());
291                 formatter.format("%n");
292                 return sb;
293         }
294
295         /**
296          * Returns the details about all cabins
297          *
298          * @return A stringbuilder with formatted output.
299          */
300         private StringBuilder getAllCabinDetails() {
301                 StringBuilder sb = new StringBuilder();
302                 Formatter formatter = new Formatter(sb, Locale.UK);
303                 formatter.format("%1$10s | %2$10s | %3$20s | %4$5s %n", "NUMBER",
304                                 "OWNER", "FACILITIES", "BEDS");
305                 for (Cabin cab : this.cabins) {
306                         // TODO Return the initials of the owner.
307                         formatter.format("%1$10d | %2$10s | %3$20s | %4$5d %n",
308                                         cab.getCabinNumber(), cab.getOwner().
                                                getInitials(), cab
309                                                         .getFacilities().toString().
                                                             toLowerCase(),
310                                         cab.getBeds());
311                 }
312                 formatter.format("%n");
313                 return sb;
314         }
315 }
```

Listing 2: uk.heriotwatt.sef.model.CabinManager.java

```
1  package uk.heriotwatt.sef.model;
2
3  /**
4   * Stores the different possible of conditions.
5   *
```

```
6  * @author fhb2
7  *
8  */
9  public enum Condition {
10
11         PERFECT, GOOD, FAIR, BAD, IN_SHAMBLES, UNKNOWN
12
13 }
```

Listing 3: uk.heriotwatt.sef.model.Condition.java

```
1  package uk.heriotwatt.sef.model;
2
3  /**
4   * Stores the different possible values of facilities.
5   *
6   * @author fhb2
7   *
8   */
9  public enum Facilities {
10
11         GENERAL_FACILITIES, SEPERATE_BATHROOM, EN_SUITE, UNKNOWN
12
13 }
```

Listing 4: uk.heriotwatt.sef.model.Facilities.java

```
1  package uk.heriotwatt.sef.model;
2
3  /**
4   * Stores the price modifiers for certain discrete price categories.
5   *
6   * @author fhb2
7   *
8   */
9  public enum PriceList {
10
11         VERY_EXPENSIVE(10.0), EXPENSIVE(7.5), FAIR(5.0), BUDGET(2.5), CHEAP(1.0),
               UNKNOWN(
12                         0.0);
13
14         private final double cost;
15
16         private PriceList(double cost) {
17                 this.cost = cost;
18         }
19
20         public double cost() {
21                 return this.cost;
22         }
```

```
23
24 }
```

Listing 5: uk.heriotwatt.sef.model.PriceList.java

```
1  package uk.heriotwatt.sef.model;
2
3  import java.util.HashMap;
4  import java.util.Map;
5
6  /**
7   * Class to seperate the pricing mapping from the information of the cabin.
8   *
9   * @author florian
10  *
11  */
12 public class PriceMapping {
13
14         private Map<Condition, Double> conditionPrices;
15         private Map<Facilities, Double> facilityPrices;
16
17         public PriceMapping() {
18                 this.initializeConditionPriceMapping();
19                 this.initializeFacilityPriceMapping();
20         }
21
22         public Map<Condition, Double> getConditionPrices() {
23                 return conditionPrices;
24         }
25
26         public Map<Facilities, Double> getFacilityPrices() {
27                 return facilityPrices;
28         }
29
30         /*
31          * Getters and setters
32          */
33
34         public double getFacilityPrice(Facilities facilities) {
35                 return facilityPrices.get(facilities);
36         }
37
38         public double getConditionPrice(Condition condition) {
39                 return conditionPrices.get(condition);
40         }
41
42         /**
43          * Adds Condition - Price pairs to a map. Will be used in the getCost()
44          * method.
45          */
46         private void initializeConditionPriceMapping() {
```

```
47                  this.conditionPrices = new HashMap<Condition, Double>();
48                  this.conditionPrices.put(Condition.PERFECT,
49                                  PriceList.VERY_EXPENSIVE.cost());
50                  this.conditionPrices.put(Condition.GOOD, PriceList.EXPENSIVE.cost());
51                  this.conditionPrices.put(Condition.FAIR, PriceList.FAIR.cost());
52                  this.conditionPrices.put(Condition.BAD, PriceList.BUDGET.cost());
53                  this.conditionPrices.put(Condition.IN_SHAMBLES, PriceList.CHEAP.cost()
                        );
54          }
55
56          /**
57           * Adds Facilities - Price pairs to a map. Will be used in the getCost()
58           * method.
59           */
60          private void initializeFacilityPriceMapping() {
61                  this.facilityPrices = new HashMap<Facilities, Double>();
62                  this.facilityPrices.put(Facilities.EN_SUITE,
63                                  PriceList.VERY_EXPENSIVE.cost());
64                  this.facilityPrices.put(Facilities.SEPERATE_BATHROOM,
65                                  PriceList.FAIR.cost());
66                  this.facilityPrices.put(Facilities.GENERAL_FACILITIES,
67                                  PriceList.BUDGET.cost());
68          }
69
70          /**
71           * Return the size modifier that can be used to calculate a price for a
72           * cabin.
73           *
74           * @param size
75           *            The size of the cabin.
76           * @return The size modifier according to the provided size of a room.
77           */
78          public double getSizeModifier(double size) {
79                  if (size < 20) {
80                          return PriceList.BUDGET.cost();
81                  } else if (size >= 20 && size < 30) {
82                          return PriceList.CHEAP.cost();
83                  } else if (size >= 30 && size < 40) {
84                          return PriceList.FAIR.cost();
85                  } else if (size >= 40 && size < 50) {
86                          return PriceList.EXPENSIVE.cost();
87                  } else {
88                          return PriceList.VERY_EXPENSIVE.cost();
89                  }
90          }
91  }
```

Listing 6: uk.heriotwatt.sef.model.PriceMapping.java

```
1  package uk.heriotwatt.sef.model;
2
```

```
3  import java.io.File;
4  import java.io.FileNotFoundException;
5  import java.io.IOException;
6  import java.io.PrintWriter;
7  import java.util.Date;
8  import java.util.LinkedList;
9  import java.util.List;
10 import java.util.Scanner;
11
12 public class CabinFileHandler {
13
14         private String pathToReadFile;
15         private String pathToReportFile;
16
17         /**
18          * Creates a new cabinfilehandler object from the provided arguments.
19          * @param pathReadFile The path of the file of cabins.
20          * @param pathWriteFile The path of the file to write the reports to.
21          */
22         public CabinFileHandler(String pathReadFile, String pathWriteFile) {
23                 this.pathToReadFile = pathReadFile;
24                 this.pathToReportFile = pathWriteFile;
25         }
26
27         /**
28          * Attempts to write a provided string to a file.
29          * @param sb The string to be written to a file.
30          */
31         public void writeToFile(String sb) {
32                 try {
33                         File file = new File(pathToReportFile);
34                         PrintWriter pw = new PrintWriter(file);
35                         pw.write(sb);
36                         pw.flush();
37                         pw.close();
38                 } catch (Exception e) {
39                         e.printStackTrace();
40                 }
41         }
42
43         /**
44          * Attempts to read a list of cabins from a file.
45          * @return A new list of cabin objects.
46          */
47         public List<Cabin> readFromFile() {
48                 List<Cabin> cabinList = new LinkedList<Cabin>();
49                 try {
50                         File file = new File(this.pathToReadFile);
51                         Scanner scanner = new Scanner(file);
52                         while (scanner.hasNext()) {
53                                 String nextLine = scanner.nextLine();
```

```
54                              if (nextLine.trim().startsWith("#")) {
55                                      // Ignoring commented out lines.
56                                      System.out.println("Ignoring a commented out
                                            line.");
57                              } else {
58                                      try {
59                                              Cabin cabin = this.createCabin(
                                                    nextLine);
60                                              cabinList.add(cabin);
61                                      } catch (IllegalArgumentException e) {
62                                              e.printStackTrace();
63                                      }
64                              }
65                      }
66              } catch (FileNotFoundException e) {
67                      e.printStackTrace();
68              } catch (IOException e) {
69                      e.printStackTrace();
70              }
71              return cabinList;
72      }
73
74      /**
75       * Attempts to parse a cabin from a provided string that must comply to the
76       * following format:
77       * CabinNumber,Size,Facilities,Condition,Forename,Middlename,Surname,BedsRoom1
             ,BedsRoom2,...,[BedsRoomN]
78       * @param nextLine
79       *            The string containing all arguments.
80       * @return A cabin object created from the provided arguments.
81       * @throws IllegalArgumentException
82       *             If the provided string does not comply to the needed format.
83       */
84      public Cabin createCabin(String nextLine) {
85              Cabin cabin = null;
86              String[] splitList = nextLine.split(",");
87              String errorString = "";
88
89              int cabinNumber = 0;
90              double size = 0;
91              Facilities facilities = null;
92              Condition condition = null;
93              Name name;
94              int[] beds;
95              try {
96                      cabinNumber = Integer.parseInt(splitList[0]);
97              } catch (NumberFormatException e) {
98                      errorString += splitList[0];
99              }
100             try {
101                     size = Double.parseDouble(splitList[1]);
```

```
102                 } catch (NumberFormatException e) {
103                         errorString += ", " + splitList[1];
104                 }
105                 try {
106                         facilities = Facilities.valueOf(splitList[2]);
107                 } catch (IllegalArgumentException e) {
108                         errorString += ", " + splitList[2];
109                 }
110                 try {
111                         condition = Condition.valueOf(splitList[3]);
112                 } catch (IllegalArgumentException e) {
113                         errorString += ", " + splitList[3];
114                 }
115                 name = new Name(splitList[4], splitList[5], splitList[6]);
116                 beds = new int[splitList.length - 7];
117                 for (int i = 7; i < splitList.length; i++) {
118                         try {
119                                 beds[i - 7] = Integer.parseInt(splitList[i]);
120                         } catch (Exception e) {
121                                 errorString += ", " + beds[i - 7];
122                         }
123                 }
124                 if (errorString.length() == 0) {
125                         cabin = new Cabin(cabinNumber, beds, size, facilities, name,
126                                         condition);
127                         return cabin;
128                 } else {
129                         throw new IllegalArgumentException(
130                                         String.format(
131                                                         "There were errors parsing the
                                                              line:\r%s\nThe following
                                                              arguments were violating
                                                              the format:%s",
132                                                         nextLine, errorString));
133                 }
134         }
135
136 }
```

Listing 7: uk.heriotwatt.sef.model.CabinFileHandler.java

```
1  package uk.heriotwatt.sef.model;
2
3  /**
4   * Exception to clarify the missing of a cabin.
5   *
6   * @author florian
7   *
8   */
9  public class CabinNotFoundException extends Exception {
10
```

```
11          /**
12           * Generated serialVersionUID to allow serialisation.
13           */
14          private static final long serialVersionUID = -7740644730079198039L;
15
16          public CabinNotFoundException(String msg) {
17                  super(msg);
18          }
19  }
```

Listing 8: uk.heriotwatt.sef.model.CabinNotFoundException.java

```
1   package uk.heriotwatt.sef.model;
2
3   /**
4    * Shows that no cabins are currently loaded.
5    *
6    * @author florian
7    *
8    */
9   public class NoCabinsException extends Exception {
10
11          /**
12           * Generated serialVersionUID to allow serialisation.
13           */
14          private static final long serialVersionUID = 2274177224545932291L;
15
16          public NoCabinsException(String msg) {
17                  super(msg);
18          }
19  }
```

Listing 9: uk.heriotwatt.sef.model.NoCabinsException.java

```
1   package uk.heriotwatt.sef.model;
2
3   //First Name class
4   //F21SF - Monica
5   public class Name {
6           private String firstName;
7           private String middleName;
8           private String lastName;
9
10          // constructor to create object with first, middle and last name
11          // if there isn't a middle name, that parameter could be an empty String
12          public Name(String fName, String mName, String lName) {
13                  firstName = fName;
14                  middleName = mName;
15                  lastName = lName;
16          }
```

```
17
18          // returns the first name
19          public String getFirstName() {
20                  return firstName;
21          }
22
23          // returns the last name
24          public String getLastName() {
25                  return lastName;
26          }
27
28          // change the last name to the value provided in the parameter
29          public void setLastName(String ln) {
30                  lastName = ln;
31          }
32
33          // returns the first name then a space then the last name
34          public String getFirstAndLastName() {
35                  return firstName + " " + lastName;
36          }
37
38          // returns the last name followed by a comma and a space
39          // then the first name
40          public String getLastCommaFirst() {
41                  return lastName + ", " + firstName;
42          }
43
44          // returns name in the format initial, period, space, lastname
45          public String getInitPeriodLast() {
46                  return firstName.charAt(0) + ". " + lastName;
47          }
48
49          public String getInitials() {
50                  return firstName.charAt(0) + ". " + lastName.charAt(0) + ".";
51          }
52  }
```

Listing 10: uk.heriotwatt.sef.model.Name.java

4

EXAMPLE OUTPUT

The output printed to a file will look like the following:

```
OVERVIEW OF CABIN DETAILS:

    NUMBER |      OWNER |           FACILITIES |  BEDS
         1 |      J. S. |             en_suite |     2
         4 |      C. N. |   general_facilities |     4
         5 |      S. G. |             en_suite |     3
         6 |      U. O. |   general_facilities |     6
         7 |      P. N. |    seperate_bathroom |     8
         8 |      R. C. |    seperate_bathroom |     4
         9 |      B. W. |   general_facilities |     4
        10 |      G. J. |             en_suite |     8
         2 |      B. F. |             en_suite |     3
         3 |      H. S. |   general_facilities |     6

SINGLE CABIN INFORMATION:

    NUMBER |           OWNER |           FACILITIES |       CONDITION |  BEDS | ROOMS
        |  SIZE |  COST
         1 |        John Sly |             en_suite |         perfect |     2 |     2
            | 10.00 | 37.50

    NUMBER |           OWNER |           FACILITIES |       CONDITION |  BEDS | ROOMS
        |  SIZE |  COST
         4 | Conchobar Nessa |   general_facilities |            good |     4 |     3
            | 35.00 | 25.00

    NUMBER |           OWNER |           FACILITIES |       CONDITION |  BEDS | ROOMS
        |  SIZE |  COST
         5 |        Sie Gurd |             en_suite |            fair |     3 |     1
            | 45.00 | 32.50

    NUMBER |           OWNER |           FACILITIES |       CONDITION |  BEDS | ROOMS
        |  SIZE |  COST
         6 | Uthuce Odysseus |   general_facilities |     in_shambles |     6 |     3
            | 55.00 | 23.50

    NUMBER |           OWNER |           FACILITIES |       CONDITION |  BEDS | ROOMS
        |  SIZE |  COST
         7 |  Perseus Nestor |    seperate_bathroom |         perfect |     8 |     4
            | 39.00 | 30.00

    NUMBER |           OWNER |           FACILITIES |       CONDITION |  BEDS | ROOMS
        |  SIZE |  COST
```

```
        8 |   Robert Cotton |    seperate_bathroom |           fair |     4 |     3 | 38.00 | 25.00

    NUMBER |           OWNER |           FACILITIES |       CONDITION |  BEDS | ROOMS |  SIZE |  COST
        9 |        Beo Wulf |   general_facilities |            bad |     4 |     2 | 38.50 | 20.00

    NUMBER |           OWNER |           FACILITIES |       CONDITION |  BEDS | ROOMS |  SIZE |  COST
       10 |  Grimur Jonsson |            en_suite |            good |     8 |     3 | 27.00 | 28.50

    NUMBER |           OWNER |           FACILITIES |       CONDITION |  BEDS | ROOMS |  SIZE |  COST
        2 |    Ben Franklin |            en_suite |         perfect |     3 |     3 | 35.75 | 40.00

    NUMBER |           OWNER |           FACILITIES |       CONDITION |  BEDS | ROOMS |  SIZE |  COST
        3 |   Homer Simpson |   general_facilities |            fair |     6 |     3 | 53.95 | 27.50

MOST EXPENSIVE CABIN: 40.0

CHEAPEST CABIN: 20.0

MAXIMUM INCOME PER NIGHT: 289.5

CONDITION REPORT:

           BAD |           FAIR |           GOOD |     IN_SHAMBLES |            PERFECT
             1 |              3 |              2 |              1 |                  3
```

Listing 11: Example output-file

5 TESTING REPORT

The application is delivered with a set of test cases that all pass: they are printed in Appendix A[1].

A short list of noteworthy test cases shall be provided as an overview:

FILE HANDLING:
- Create new Cabin from valid input string.
- Create new cabin from invalid input string (throws IllegalArgumentException).
- Read from non present file (catch with stack trace).
- Write to non present file (catch with stack trace).

CABIN:
- Set number of beds (too few → IllegalArgumentException, too many → IllegalArgumentException, valid numbers).
- Test cost calculation algorithm.

CABIN MANAGER:
- Add cabin.
- Get cabin at index (with and without any cabins → IndexOutOfBoundsException).
- Find cabin by number (cabin is in list and cabin is not in list → CabinNotFoundException).
- Get cheapest and most expensive cabin (without cabins → NoCabinsException).

The error handling of incorrect input in the file handler is performed by checking each attribute and - if an error occurs - add this attribute to an error-string that will be returned via an IllegalArgumentException. This way it is possible to inform the user about the concrete argument that caused the error.

1 In order to run the test cases it is necessary that the JUnit (`http://www.junit.org/`) and the Mockito (`http://code.google.com/p/mockito/`) framework are present.

Part II

APPENDIX

APPENDIX

Package: uk.heriotwatt.sef.model.tests

```
1  package uk.heriotwatt.sef.model.tests;
2
3  import junit.framework.Assert;
4
5  import org.junit.After;
6  import org.junit.Before;
7  import org.junit.Test;
8
9  import uk.heriotwatt.sef.model.Cabin;
10 import uk.heriotwatt.sef.model.CabinFileHandler;
11
12 public class CabinFileHandlerTests {
13
14         private CabinFileHandler fileHandler;
15
16         @Before
17         public void setUp() throws Exception {
18                 this.fileHandler = new CabinFileHandler("", "");
19         }
20
21         @After
22         public void tearDown() throws Exception {
23         }
24
25         @Test
26         public void testCreateCabin()
27         {
28                 String toParse = "1,10,EN_SUITE,IN_SHAMBLES,John,Jack,MasterMind
                       ,2,2,2,2";
29                 Cabin cabin = fileHandler.createCabin(toParse);
30                 Assert.assertNotNull(cabin);
31         }
32
33         @Test(expected=IllegalArgumentException.class)
34         public void testCreateCabinFaultyValues()
35         {
36                 String toParse = "1,10,EN_SITE,IN_SHAMBLES,John,Jack,MasterMind
                       ,2,2,2,2";
37                 Cabin cabin = fileHandler.createCabin(toParse);
38         }
39
```

```
40          @Test
41          public void testReadFromNonPresentFile()
42          {
43                  this.fileHandler.readFromFile();
44                  /*
45                   * Testing that the test runner does not crash but the false input is
                         caught.
46                   */
47                  Assert.assertTrue(true);
48          }
49
50          @Test
51          public void testWriteToNonPresentFile()
52          {
53                  this.fileHandler.writeToFile("Nothing");
54                  /*
55                   * Testing that the test runner does not crash but the false input is
                         caught.
56                   */
57                  Assert.assertTrue(true);
58          }
59  }
```

Listing 12: uk.heriotwatt.sef.model.tests.CabinFileHandlerTests.java

```
1   /**
2    *
3    */
4   package uk.heriotwatt.sef.model.tests;
5
6   import static org.junit.Assert.*;
7
8   import java.util.LinkedList;
9
10  import junit.framework.Assert;
11
12  import org.junit.After;
13  import org.junit.Before;
14  import org.junit.Test;
15
16  import static org.mockito.Mockito.*;
17
18  import uk.heriotwatt.sef.model.Cabin;
19  import uk.heriotwatt.sef.model.CabinFileHandler;
20  import uk.heriotwatt.sef.model.CabinManager;
21  import uk.heriotwatt.sef.model.CabinNotFoundException;
22  import uk.heriotwatt.sef.model.Condition;
23  import uk.heriotwatt.sef.model.Facilities;
24  import uk.heriotwatt.sef.model.Name;
25  import uk.heriotwatt.sef.model.NoCabinsException;
26
```

```
27 /**
28  * @author Florian Bergmann
29  *
30  */
31 public class CabinManagerTests {
32
33         private CabinManager cabMan;
34         private Cabin mockin;
35         private Cabin mockin2;
36
37         /**
38          * @throws java.lang.Exception
39          */
40         @Before
41         public void setUp() throws Exception {
42                 CabinFileHandler mockHandler = mock(CabinFileHandler.class);
43                 when(mockHandler.readFromFile()).thenReturn(new LinkedList());
44                 cabMan = new CabinManager(mockHandler);
45                 mockin = mock(Cabin.class);
46                 mockin2 = mock(Cabin.class);
47         }
48
49         /**
50          * @throws java.lang.Exception
51          */
52         @After
53         public void tearDown() throws Exception {
54         }
55
56         /**
57          * Test method for {@link uk.heriotwatt.sef.model.CabinManager#addCabin(uk.
                heriotwatt.sef.model.Cabin)}.
58          */
59         @Test
60         public void testAddCabin() {
61                 Cabin cab = new Cabin(1, new int[] {2,3}, 55.0, Facilities.EN_SUITE,
                        new Name("Test", "Test", "Test"), Condition.GOOD);
62                 cabMan.addCabin(cab);
63                 Assert.assertEquals(1, cabMan.getNumberOfCabins());
64         }
65
66         /**
67          * Test method for {@link uk.heriotwatt.sef.model.CabinManager#getCabinAtIndex
                (int)}.
68          */
69         @Test
70         public void testGetCabinAtIndex() {
71                 Cabin cab = new Cabin(1, new int[] {2,3}, 55.0, Facilities.EN_SUITE,
                        new Name("Test", "Test", "Test"), Condition.GOOD);
72                 cabMan.addCabin(cab);
73                 Cabin cabIndex = cabMan.getCabinAtIndex(0);
```

```
74                  Assert.assertEquals(cabIndex, cab);
75          }
76
77          @Test(expected=IndexOutOfBoundsException.class)
78          public void testGetCabinAtIndexWrongIndex() {
79                  cabMan.getCabinAtIndex(1);
80          }
81
82          /**
83           * Test method for {@link uk.heriotwatt.sef.model.CabinManager#
                 findCabinByCabinNumber(int)}.
84           */
85          @Test
86          public void testFindCabinByCabinNumber() {
87                  Cabin cab = new Cabin(1, new int[] {2,3}, 55.0, Facilities.EN_SUITE,
                        new Name("Test", "Test", "Test"), Condition.GOOD);
88                  cabMan.addCabin(cab);
89                  Cabin cabFound = null;
90                  try {
91                          cabFound = cabMan.findCabinByCabinNumber(1);
92                  } catch (CabinNotFoundException e) {
93                          // TODO Auto-generated catch block
94                          e.printStackTrace();
95                  }
96                  Assert.assertEquals(cab, cabFound);
97          }
98
99          @Test(expected=CabinNotFoundException.class)
100         public void testFindCabinByCabinNumberWrongNumber() throws
                CabinNotFoundException {
101                 cabMan.findCabinByCabinNumber(1);
102         }
103
104         /**
105          * Test method for {@link uk.heriotwatt.sef.model.CabinManager#
                 getMaximumPossibleIncome()}.
106          */
107         @Test
108         public void testGetMaximumPossibleIncome() {
109                 when(mockin.getCost()).thenReturn(50.0);
110                 when(mockin2.getCost()).thenReturn(40.0);
111                 cabMan.addCabin(mockin);
112                 cabMan.addCabin(mockin2);
113                 double getMaxIncome = cabMan.getMaximumPossibleIncome();
114                 Assert.assertEquals(90, getMaxIncome, 0);
115         }
116
117         /**
118          * Test method for {@link uk.heriotwatt.sef.model.CabinManager#
                 getCheapestCabinCost()}.
119          */
```

```
120         @Test
121         public void testGetCheapestCabinCost() {
122                 when(mockin.getCost()).thenReturn(50.0);
123                 when(mockin2.getCost()).thenReturn(40.0);
124                 cabMan.addCabin(mockin);
125                 cabMan.addCabin(mockin2);
126                 double cheapest = 0;
127                 try {
128                         cheapest = cabMan.getCheapestCabinCost();
129                 } catch (NoCabinsException e) {
130                         // TODO Auto-generated catch block
131                         e.printStackTrace();
132                 }
133                 Assert.assertEquals(40, cheapest, 0);
134         }
135 
136         @Test(expected=NoCabinsException.class)
137         public void testGetCheapestCabinCostNoCabins() throws NoCabinsException {
138                 cabMan.getCheapestCabinCost();
139         }
140 
141         /**
142          * Test method for {@link uk.heriotwatt.sef.model.CabinManager#
                 getExpensiveCabinCost()}.
143          */
144         @Test
145         public void testGetExpensiveCabinCost() {
146                 when(mockin.getCost()).thenReturn(50.0);
147                 when(mockin2.getCost()).thenReturn(40.0);
148                 cabMan.addCabin(mockin);
149                 cabMan.addCabin(mockin2);
150                 double mostExpensive = 0;
151                 try {
152                         mostExpensive = cabMan.getExpensiveCabinCost();
153                 } catch (NoCabinsException e) {
154                         // TODO: handle exception
155                 }
156                 Assert.assertEquals(50, mostExpensive, 0);
157         }
158 
159         @Test(expected=NoCabinsException.class)
160         public void testGetExpensiveCabinCostNoCabins() throws NoCabinsException {
161                 cabMan.getExpensiveCabinCost();
162         }
163 }
```

Listing 13: uk.heriotwatt.sef.model.tests.CabinManagerTests.java

```
1 package uk.heriotwatt.sef.model.tests;
2 
3 
```

```
4  import org.junit.Assert;
5  import org.junit.Before;
6  import org.junit.Test;
7
8  import uk.heriotwatt.sef.model.Cabin;
9  import uk.heriotwatt.sef.model.Condition;
10 import uk.heriotwatt.sef.model.Facilities;
11 import uk.heriotwatt.sef.model.Name;
12 import uk.heriotwatt.sef.model.PriceMapping;
13
14 public class CabinTests {
15
16         private Cabin cabin;
17
18         @Before
19         public void setUp() throws Exception {
20                 cabin = new Cabin();
21         }
22
23         @Test
24         public void testSetNumberOfBeds()
25         {
26                 int[] numberToSet = new int[] {2};
27                 cabin.setNumberOfBeds(numberToSet);
28                 int[] numberSet = cabin.getNumberOfBeds();
29                 Assert.assertArrayEquals(numberToSet, numberSet);
30
31                 numberToSet = new int[] {2, 2, 4};
32                 cabin.setNumberOfBeds(numberToSet);
33                 numberSet = cabin.getNumberOfBeds();
34                 Assert.assertArrayEquals(numberToSet, numberToSet);
35         }
36
37         @Test(expected=IllegalArgumentException.class)
38         public void testSetNumberOfBedsWithTooFewBeds()
39         {
40                 int[] numberToSet = new int[] {1};
41                 cabin.setNumberOfBeds(numberToSet);
42         }
43
44         @Test(expected=IllegalArgumentException.class)
45         public void testSetNumberOfBedsWithTooManyBeds()
46         {
47                 int[] numberToSet = new int[] {9};
48                 cabin.setNumberOfBeds(numberToSet);
49         }
50
51         @Test
52         public void testCostForFacilites()
53         {
54                 PriceMapping pm = new PriceMapping();
```

```
55
56                  Facilities fac = Facilities.EN_SUITE;
57                  Condition con = Condition.FAIR;
58                  int[] beds = new int[] {2,2};
59                  double size = 49.99;
60
61                  double basePrice = 10;
62                  double conPrice = pm.getConditionPrice(con);
63                  double facPrice = pm.getFacilityPrice(fac);
64                  double sizePrice = pm.getSizeModifier(size);
65
66                  double expectedPrice = basePrice + conPrice + facPrice + sizePrice + 5
                         * (2/4);
67
68                  Cabin cab = new Cabin(1, beds,size, fac, new Name("Ho", "ho", "ho"),
                        con);
69
70                  double price = cab.getCost();
71
72                  Assert.assertEquals(expectedPrice, price, 0);
73          }
74
75  }
```

Listing 14: uk.heriotwatt.sef.model.tests.CabinTests.java