OOP preparation for prelims – Big Claw Adventures. Version Six Pupils

Step One – Using the class diagrams and the text file create the four classes i.e. the UI, the manager, the parent and the child class. Create the properties and fields from the class diagrams.

People book a Big Claw Adventure hike. Your program must process bookings that already exist looking for problem areas and gathering information from the booking data.

The program you write must not attempt to make bookings or add new bookings.

Trails
Properties - Id : integer - name : string - distance : integer - town : string - labelT : string
Methods – May need to be edited later when you fully read the question paper. + Constructor(int i, string n, integer d, string t, string lt) + getId( ) : string + getLabelT ( ) : string + getName ( ) : string + toString( ) : string

MountainTrails
Properties - labelM : string - climbingPattern : string
Methods – May need to be edited later when you fully read the question paper. + Constructor (int I, string n, integer d, string t, string lt, string lm, string cp) + getLableM ( ) : string + toString( ) : string

The text file "trails.txt" looks like this. Some trails have a mountaineering element.

The trail number, trail name, distance, nearest town, land or river, mountaineering difficulty, climbing pattern.

1,Gorgina,43,Panhandle,L,M3,XXXIIIIXIIXX

2,Big Beaver,65,Washing Gulsh,R

3,High Country,50,Echo Yarn,L

4,Widow's Creek,10,Creeks End,R

5,Winston's Folly,25,Breakenridge,L,M2,IIIIXX

Bare Bones

1. Create TrailsUI
2. Create TrailManager
3. Create class Trails
4. Create class MountainTrails

Q1 - Read the file into an array of trail objects – one array made up of parent and child objects.

Q2 – Create a typed method in the manager manger class called displayAllTrails. It must return the trails to the UI class for printing to the monitor. The printout should be in the following format.

```
run:
All Trails
==========
1       Georgina            43      Panhandle       L       M3      XXXIIIIXIIXX
2       Big Beaver          65      Washing Gulsh   R
3       High Country        50      Echo Yarn       L
4       Widow's Creek       10      Creeks End      R
5       Winston's Folly     25      Breakenridge    L       M2      IIIIXX
```

Q2A – Create a typed method in the manager class called displayMountainTrailOnly. This method will only print members of the MountainTrail Class. (Tip: Use the comparison operator "instanceof"). The printout should be in the following format.

```
Mountain Trails only
====================
1       Georgina            43      Panhandle       L       M3      XXXIIIIXIIXX
5       Winston's Folly     25      Breakenridge    L       M2      IIIIXX
```

SOLUTION THUS FAR:

```
 1
 2 package gr12prelimpractrails;
 3
 4
 5 public class TrailsUI {
 6
 7     public static void main(String[] args) {
 8         TrailsManager tm = new TrailsManager();
 9         String theTrails = null;
10
11         theTrails = tm.displayAllTrails();
12         System.out.println(theTrails);
13
14         theTrails = tm.displayTrailOnly();
15         System.out.println(theTrails);
16         System.out.println("trails UI");
17     }
18 }
19 ====================================================
20
21
22 package gr12prelimpractrails;
23
24 import java.io.File;
25 import java.io.FileNotFoundException;
26 import java.util.Scanner;
27 import java.util.logging.Level;
28 import java.util.logging.Logger;
29
30 public class TrailsManager {
31
32     private Trails[] trailArr = new Trails[50];
33     private int counter = 0;
34     private int id = 0, distance = 0;
35     private String name, town, labelT, labelM, climbingPattern;
36     private Trails oneTrail;
37
38     public TrailsManager() {
39         try {
40             Scanner scFile = new Scanner(new File("trails.txt"));
41
42             while(scFile.hasNext()){
43                 String line = scFile.nextLine();
44                 Scanner scLine = new Scanner(line).useDelimiter(",");
45                 id = scLine.nextInt();
46                 name = scLine.next();
47                 distance = scLine.nextInt();
48                 town = scLine.next();
49                 labelT = scLine.next();
50
51                 if(scLine.hasNext()){
52                     labelM = scLine.next();
53                     climbingPattern = scLine.next();
54                     trailArr[counter] = new
MountainTrails(id,name,distance,town, labelT, labelM, climbingPattern);
```

```
55
56                  } // end if
57                  else
58                      trailArr[counter] = new
Trails(id,name,distance,town, labelT);
59
60                  counter++;
61              } // end while
62
63
64
65          } catch (FileNotFoundException ex) {
66
Logger.getLogger(TrailsManager.class.getName()).log(Level.SEVERE, null,
ex);
67              System.out.println("Error. File not found");
68          }
69
70
71      } // end TrailsManager
72
73      public String displayAllTrails() {
74
75          System.out.println("All Trails");
76          System.out.println("===========");
77          String theTrails = "";
78
79          for(int x = 0; x < counter; x++) {
80              theTrails = theTrails + trailArr[x] + "\n";
81          }
82
83          return theTrails;
84      }
85
86          public String displayTrailOnly() {
87
88          System.out.println("Mountain Trails only");
89              System.out.println("====================");
90          String theTrails = "";
91
92          for(int x = 0; x < counter; x++) {
93
94              if(trailArr[x] instanceof MountainTrails){
95                  theTrails = theTrails + trailArr[x] + "\n";
96
97              }
98
99          } // end for
100
101         return theTrails;
102     }
103
104 }
105 ================================================================
106
107
108
```

```
package gr12prelimpractrails;
109
110 public class Trails {
111
112     private int id = 0;
113     private String name = null;
114     private int distance  = 0;
115     private String town = null;
116     private String labelT = null; // land, river
117     private Trails oneTrail = null;
118
119     public Trails (int i,String n, int d, String t, String lt) {
120         id = i;
121         name = n;
122         distance = d;
123         town = t;
124         labelT = lt;
125
126     } // end trails
127
128     public int getId() {
129         return id;
130     }
131
132     public String getName() {
133         return name;
134     }
135
136     public String getLabelT() {
137         return labelT;
138     }
139
140     @Override
141     public String toString() {
142         return id + "\t" + name +"\t" + "\t" + distance + "\t"+ town +
"\t" + labelT;
143     }
144
145
146
147 }  // end trails
148
149 ==========================================================
150
151 package gr12prelimpractrails;
152
153 public class MountainTrails extends Trails{
154
155     private String labelM;
156     private String climbingPattern;
157
158     public MountainTrails(int i, String n, int d, String t, String lt,
String lm, String cp){
159         super(i, n, d, t, lt);
160         labelM = lm;
161         climbingPattern = cp;
162
```

```
163     } // end MountainTrails
164
165     public String getLabelM() {
166         return labelM;
167     }
168
169     @Override
170     public String toString() {
171         return super.toString() +  "\t" + labelM + "\t" +
climbingPattern;
172     }
173
174
175 }
176
177 ================================================================
```

Now it is time to start reading the exam paper but focus more on the questions than the introduction. Do what the questions ask, not what you assume from the introduction.

Step Two. Develop the parent and child classes from the exam paper.

NOTE1: Getter methods (accessor methods) do not only return the actual data in the field. They can return anything – error messages, additional information, they can change the formatting of the data, they can join different fields together etc

Q3 – The getName method in the trail class. It must return the name in the following format.

The name must be in capital letters.

If the name is one word – return that one word only.

If the name is two words – the return the second word last, followed by the first word. The two words must be separated by a hyphen (see below)

Q3A - Create a typed method in the manager class that will use the getName method created above. It will return all the trail names in the format mentioned.

```
Trail names only
================

GEORGINA
BEAVER-BIG
COUNTRY-HIGH
CREEK-WIDOW'S
FOLLY-WINSTON'S
```

NOTE2: Every class from which we instantiate objects (template class) must have a toString method that can harvest the fields from the object. You will be asked to create your own toString methods using a different/unique format. Your toString methods can also have error messages and additional information.

Q4A – Comment out any existing toString method you may have for re-use later

Create an additional toString method in the trail class so that it follows this format.

<distance><(><town><)><tab><name><tab><labelT> // note the brackets.

Q4B – The toString method in the MountainTrail class must override the parent toString class. It must add its own information and follow the format below.

<distance><(><town><)><tab><name><tab><labelT><getLabelM>

NOTE: This edit will also affect the output of the child class as it too uses the toString of the parent. Run your displayAllTrails method again.

Yes, a bit messy but here is what the new version will look like.

```
All Trails
==========
43(Panhandle)   Georgina         L       M3      XXXIIIIXIIXX
€5(Washing Gulsh)       Big Beaver       R
50(Echo Yarn)   High Country     L
10(Creeks End)  Widow's Creek    R
25(Breakenridge)        Winston's Folly L       M2      IIIIXX

Mountain Trails only
====================
43(Panhandle)   Georgina         L       M3      XXXIIIIXIIXX
25(Breakenridge)        Winston's Folly L       M2      IIIIXX
```

Step Three: From the exam paper develop any further classes needed.

Q5 - Create a booking class based on the following class diagram – we also consult the relevant **text file.** Note the constants as indicated (by the capital letters). They must be **visible from outside** the program and must be **static.**

Booking
Properties: - bookingDate: Date - trail : Trail (this field is a Trail object) - clientId: int - guideId: int - trailType: char - TRAILLAND = 'l' : character (See notes on character and trail type below) - TRAILRIVER = 'r' : character - TRAILMOUNTAIN = 'm' : character - trailDescription: string
Methods: + Constructor (d : date, t: Trail, c : integer, g : int, t : character) + getTrailDesciption( ) : string + getTrailType( ) : character + getTrail( ) : Trail + writeToFile( ) : string

The text file "bookings.txt" looks like this. Booking date, trail number, clientId, guide number, trail type.

NOTE: There is NO nextChar( ) in Java. You have to use String method charAt( ) which returns a char

14/06/2023#2#2000#100#l

15/06/2023#3#2007#102#l

17/06/2023#4#2009#104#r

20/06/2023#3#2013#103#r

23/06/2023#4#2018#101#m

Notes on Character and Trail type

l – Overland, r - River hike, m - Includes mountaineering

Constants are values that do not change eg VAT at 15% is a variable that must not change. The access modifier is said to be **final.** We denote final constants using **capital letters**. See class diagram above.	
Tip but you must LEARN this	**public static final char CODE = 'm';**

Q6 – Create the constructor to accept the parameters as show and initialise the class properties. Remember that the property "trail" is a Trail object with its own fields

Q7 – Return to your manager class and create a method to read in the bookings contained in the text file "bookings" into a bookings array. Call your method from the UI class.

But there is a problem as one of the properties - is a **Trail object** (it has its own properties which we want.) We need to find the properties of that trail so that we can add them to our array. See Q9 below.

Q9 – Create a helper method to find a particular hiking trail. This method will receive **an integer** which is the unique identifier for that hiking trail. Find the trail with that matching number and then return the **object.** If the hiking trail is not found, return null.

```
int trailNum = scLine.nextInt();
Trails tempTrail = findTrail(trailNum);
```

Q7 – Return to Q7 now that you can create your array of bookings that includes the relevant details of the trail being booked.

Q10 – Create a displayAllBookings method like the one shown below. While it has all the details from the booking text file, it also has the details of the trail being booked ie name, distance, town, type

```
2023-06-14, trail=2     Big Beaver      €5     Washing Gulsh     R, client=2000, guide=100, Type=l
2023-06-15, trail=3     High Country    $0     Echo Yarn         L, client=2007, guide=102, Type=l
2023-06-17, trail=4     Widow's Creek   10     Creeks End        R, client=2009, guide=104, Type=r
2023-06-20, trail=3     High Country    $0     Echo Yarn         L, client=2013, guide=103, Type=r
2023-06-23, trail=4     Widow's Creek   10     Creeks End        R, client=2018, guide=101, Type=m
```

NOTES: Notice the problems in the bookings above. We will address these later. . .

- Big Beaver is a river trail but has been booked as a land trail.
- High Country is a land trail but has been booked as a river trail.
- Widow's Creek is river trail but has been booked as a mountain trail.

THE SOLUTION THUS FAR

```
 1
 2 package gr12prelimpractrails;
 3
 4
 5 public class TrailsUI {
 6
 7     public static void main(String[] args) {
 8         TrailsManager tm = new TrailsManager();
 9         String theTrails = null;
10         String theBookings = null;
11
12         theTrails = tm.displayAllTrails();
13         System.out.println(theTrails);
14
15         theTrails = tm.displayMountainTrailOnly();
16         System.out.println(theTrails);
17
18         theTrails = tm.displayTrailNames();
19         System.out.println(theTrails);
20
21         tm.processBookings();
22
23         theBookings = tm.displayBookings();
24         System.out.println(theBookings);
25
26     }
27 }
28
29 ================================================================
30
31
32 package gr12prelimpractrails;
33
34 import java.io.File;
35 import java.io.FileNotFoundException;
36 import java.time.LocalDate;
37 import java.time.format.DateTimeFormatter;
38 import java.util.Scanner;
39 import java.util.logging.Level;
40 import java.util.logging.Logger;
41
42 public class TrailsManager {
43
44     private Trails[] trailArr = new Trails[50];
45     private Bookings[] bookingArr = new Bookings[100];
46     private int counter = 0;    // counter for trails
47     private int size = 0;  // counter for bookings
48     private int id = 0, distance = 0;
49     private String name, town, labelT, labelM, climbingPattern;
50     private Trails oneTrail;
51
52     public TrailsManager() {
53         try {
```

```
54              Scanner scFile = new Scanner(new File("trails.txt"));
55
56              while(scFile.hasNext()){
57                  String line = scFile.nextLine();
58                  Scanner scLine = new Scanner(line).useDelimiter(",");
59                  id = scLine.nextInt();
60                  name = scLine.next();
61                  distance = scLine.nextInt();
62                  town = scLine.next();
63                  labelT = scLine.next();
64
65                  if(scLine.hasNext()){
66                      labelM = scLine.next();
67                      climbingPattern = scLine.next();
68                      trailArr[counter] = new
MountainTrails(id,name,distance,town, labelT, labelM, climbingPattern);
69
70                  } // end if
71                  else
72                      trailArr[counter] = new
Trails(id,name,distance,town, labelT);
73
74                  counter++;  // number of trails
75              } // end while
76
77
78          } catch (FileNotFoundException ex) {
79
Logger.getLogger(TrailsManager.class.getName()).log(Level.SEVERE, null,
ex);
80              System.out.println("Error. File not found");
81          }
82
83      } // end TrailsManager
84
85      public String displayAllTrails() {
86
87          System.out.println("All Trails");
88          System.out.println("==========");
89          String theTrails = "";
90
91          for(int x = 0; x < counter; x++) {
92              theTrails = theTrails + trailArr[x] + "\n";
93          }
94
95          return theTrails;
96      }
97
98      public String displayMountainTrailOnly() {
99
100         System.out.println("Mountain Trails only");
101             System.out.println("====================");
102         String theTrails = "";
103
104         for(int x = 0; x < counter; x++) {
105
106             if(trailArr[x] instanceof MountainTrails){
```

```
107                 theTrails = theTrails + trailArr[x] + "\n";
108
109             }
110
111         } // end for
112
113         return theTrails;
114     } // end display mountaintrailsonly
115
116     public String displayTrailNames(){
117             System.out.println("Trail names only");
118             System.out.println("================");
119             String theString = "";
120
121             for(int x = 0; x < counter; x++){
122                 theString = theString + trailArr[x].getName() + "\n";
123
124             }
125
126             return theString;
127     } // display trailnames
128
129     private Trails findTrail(int t){
130
131         for(int x = 0; x < counter;x++){
132             if(trailArr[x].getId() == t) {
133                 Trails trail = trailArr[x];
134                 return trail;
135             }
136         }
137       return null;
138     }
139
140     public void processBookings() {
141
142         try {
143             Scanner scFile = new Scanner(new File("bookings.txt"));
144             String dateSt = null;
145             size = 0;
146
147             while(scFile.hasNext()){
148                 String line = scFile.nextLine();
149                 Scanner scLine = new Scanner(line).useDelimiter("#");
150                 dateSt = scLine.next();
151                 LocalDate date = LocalDate.parse(dateSt,
DateTimeFormatter.ofPattern("dd/MM/yyyy"));
152
153                 int trailNum = scLine.nextInt();
154                 Trails tempTrail = findTrail(trailNum);
155                 int client = scLine.nextInt();
156                 int guide = scLine.nextInt();
157                 char trailType = scLine.next().charAt(0);
158                 bookingArr[size] = new Bookings(date, tempTrail,
client, guide, trailType);
159                 size++;  // number of bookings
160
161             } // end while
```

```
162         } catch (FileNotFoundException ex) {
163
Logger.getLogger(TrailsManager.class.getName()).log(Level.SEVERE, null,
ex);
164             System.out.println("File bookings not found");
165         }
166
167     } // end processBookings
168
169     public String displayBookings(){
170         String bookingsSt = "";
171
172         for(int x = 0; x < size; x++) {
173             bookingsSt = bookingsSt + bookingArr[x] + "\n";
174
175         } // end for
176
177         return bookingsSt;
178
179     } // end displaybookings
180
181
182 }
183
184
================================================================================
185
186
187 package gr12prelimpractrails;
188
189 public class Trails {
190
191     private int id = 0;
192     private String name = null;
193     private int distance  = 0;
194     private String town = null;
195     private String labelT = null; // land, river
196     private Trails oneTrail = null;
197
198     public Trails (int i,String n, int d, String t, String lt) {
199         id = i;
200         name = n;
201         distance = d;
202         town = t;
203         labelT = lt;
204
205     } // end trails
206
207     public int getId() {
208         return id;
209     }
210
211     public String getName() {
212         String firstName = "", secondName = "";
213         String tempName = name;
214         String finalTemp = "";
215
```

```
216         tempName = tempName.toUpperCase();
217         int space = tempName.indexOf(" ");
218
219         if(space == -1)
220             finalTemp = finalTemp + tempName;
221         else {
222             firstName = tempName.substring(0,space);
223             secondName = tempName.substring(space + 1);
224             finalTemp = secondName + "-" + firstName;
225         }
226         return finalTemp;
227
228     } // end getName
229
230     public String getLabelT() {
231         return labelT;
232     }
233
234     @Override
235     public String toString() {
236         return id + "\t" + name +"\t" + distance + "\t"+ town + "\t" +
labelT;
237         // return distance + "(" + town + ")" + "\t" + name + "\t" +
labelT;
238     }
239
240
241
242 }  // end trails
243
244
245
================================================================================
246
247
248 package gr12prelimpractrails;
249
250 public class MountainTrails extends Trails{
251
252     private String labelM;
253     private String climbingPattern;
254
255     public MountainTrails(int i, String n, int d, String t, String lt,
String lm, String cp){
256         super(i, n, d, t, lt);
257         labelM = lm;
258         climbingPattern = cp;
259
260     } // end MountainTrails
261
262     public String getLabelM() {
263         return labelM;
264     }
265
266     @Override
267     public String toString() {
268         return super.toString() +  "\t" + labelM + "\t" +
```

```
climbingPattern;
269     }
270
271
272 }
273
274
275
================================================================================
276
277
278 package gr12prelimpractrails;
279
280 import java.time.LocalDate;
281
282 public class Bookings {
283
284     private LocalDate bookingDate = null;
285     private Trails trail;
286     private int clientId = 0;
287     private int guideId = 0;
288     private char trailType;
289     public static final char TRAILLAND = 'l';
290     public static final char TRAILRIVER = 'r';
291     public static final char TRAILMOUNTAIN = 'm';
292     private String trailDescription = null;
293
294
295     public Bookings(LocalDate d, Trails t, int c, int g, char tt){
296         bookingDate = d;
297         trail = t;
298         clientId = c;
299         guideId = g;
300         trailType = tt;
301
302
303     } // end bookings
304
305     @Override
306     public String toString() {
307         return bookingDate + ", trail=" + trail + ", client=" +
clientId + ", guide=" + guideId + ", Type=" + trailType;
308     }
309
310
311
312 }
313
314
315
```

```
================================================================================
316
317 THE TEXT FILES - TRAILS
318
319 1,Georgina,43,Panhandle,L,M3,XXXIIIIXIIXX
320 2,Big Beaver,65,Washing Gulsh,R
321 3,High Country,50,Echo Yarn,L
322 4,Widow's Creek,10,Creeks End,R
323 5,Winston's Folly,25,Breakenridge,L,M2,IIIIXX
324
325 ============================================================
326
327 THE TEXT FILES - BOOKINGS
328
329 14/06/2023#2#2000#100#l
330 15/06/2023#3#2007#102#l
331 17/06/2023#4#2009#104#r
332 20/06/2023#3#2013#103#r
333 23/06/2023#4#2018#101#m
334
335 ============================================================
```

NOTE: Data made up of a single letter or character (a code letter) is a great solution for an app but is awful for people that must read them. For the reason the best programs have both = a code letter for the app, and a then a matching (and linked) description for people to use.

Q11 – In the Bookings class the **constants** (l for land, r for river and m for mountains) are not user friendly. Therefore, In the Bookings class, create a helper method called getTrailDescription that based on the constant, will enable the Bookings toString to return a useful trail description to the displayBookings method in the manager class.

The constants are public static and final.

```
private LocalDate bookingDate = null;
private Trails trail;
private int clientId = 0;
private int guideId = 0;
private char trailType;
public static final char TRAILLAND = 'l';
public static final char TRAILRIVER = 'r';
public static final char TRAILMOUNTAIN = 'm';
private String trailDescription = null;

public Bookings(LocalDate d, Trails t, int c, int g, char tt){
    bookingDate = d;
    trail = t;
    clientId = c;
    guideId = g;
    trailType = tt;
    trailDescription = getTrailDesc();

} // end booking constructor
```

NOTE: For equivalence we use == when comparing characters (not equals as this is a String method)

Edit your displayAllBookings and toStrings so as to achieve the following output

```
2023-06-14, trail=2     Big Beaver      65      Washing Gulsh   R, client=2000, guide=100, Desc=Land based
2023-06-15, trail=3     High Country    50      Echo Yarn       L, client=2007, guide=102, Desc=Land based
2023-06-17, trail=4     Widow's Creek   10      Creeks End      R, client=2005, guide=104, Desc=River Based
2023-06-20, trail=3     High Country    50      Echo Yarn       L, client=2013, guide=103, Desc=River Based
2023-06-23, trail=4     Widow's Creek   10      Creeks End      R, client=2018, guide=101, Desc=Land based with mountaineering
```

Notice the problems below in the bookings. We will address these now.

- Big Beaver is a river trail but has been booked as a land trail.
- High Country is a land trail but has been booked as a river trail.
- Widow's Creek is river trail but has been booked as a mountain trail.

Q12 – Create a method in the manager class to look for booking errors where the trail holiday booked, does not match the trail itself (see above) i.e. the trail label (trail class) does not match the trail type (booking class)

NOTES: The trail label is **uppercase** and **String**, whereas the trail type is **lowercase** and is **char**.

Q13 – Once you have got the method above to report the booking errors to the monitor change this so the booking errors are written to a file. Use **PrintWriter** to do this.

Here is the general layout of how to write to a file.

```
1 public void checkBookingsToFile(){
2
3 // After typing line 10 below, NetBeans will assist with the try . catch
4
5         try {
6             // We must check that the trail type in the booking,
7             // matches the actual trail type.
8             // Problem bookings are written to file
9
10            PrintWriter pw = new PrintWriter(new File("problemBookings.txt"));
11
12            for(int x = 0; x < size; x++){
13
14               *** YOUR LOGIC GOES HERE ***
15
16               // If the trail label  does not match the trial type.
17               // Write the object to file
18               if(trailLabelC != trailTypeB){
19                           pw.println("Problem booking: " + bookingArr[x]);
20
21               } // end if
22
23
24            } // end outer for
25
26            pw.close();  // Without 'close' the file is empty.
27
28        } // end checkBookingsToFile
29        catch (FileNotFoundException ex) {
30            Logger.getLogger(TrailsManager.class.getName()).log(Level.SEVERE,
null, ex);
31        }
32
33 }
```

After running successfully, you will find the "problemBookings.txt" file in your project folder. If you open it, (and you remembered to close the file), the file will look like this.

```
Problem booking: 2023-06-14, trail=2    Big Beaver      65      Washing Gulsh   R, client=2000, guide=100, Desc=Land based
Problem booking: 2023-06-20, trail=3    High Country    50      Echo Yarn       L, client=2013, guide=103, Desc=River Based
Problem booking: 2023-06-23, trail=4    Widow's Creek   10      Creeks End      R, client=2018, guide=101, Desc=Land based
```

THE FINAL SOLUTION LOOKS LIKE THIS . . .

```
1     public void checkBookingsToFile(){
2
3         try {
4             // We must check that the trail type in the booking,
5             // matches the actual trail type.
6             // Problem bookings are written to file
7
8             PrintWriter pw = new PrintWriter(new File("problemBookings.txt"));
9
10            for(int x = 0; x < size; x++){
11                // Get the trail type and trail number from the booking array.
12                char trailTypeB = bookingArr[x].getTrailTypeB();
13                Trails tempTrailB = bookingArr[x].getTrailB();
14                int trailIdB = tempTrailB.getId();
15
16                // We must now compare the booking trail type with the actual
17                // trail type.
18                // Trail label in trails is a string, whereas the trail type
19                // in bookings is a char.
20                // The trail label is uppercase - the trail type is lowercase
21                // We convert trail label (the String) to lowercase to compare
22
23                for(int y = 0; y < counter; y++) {
24                    int trailNumber = trailArr[y].getId();
25                    if(trailNumber == trailIdB){
26
27                        String trailLabelSt =
trailArr[y].getLabelT().toLowerCase();
28                        char trailLabelC = trailLabelSt.charAt(0);
29
30                        if(trailLabelC != trailTypeB){
31                            pw.println("Problem booking: " + bookingArr[x]);
32
33                        } // end if
34
35                    } // end if
36
37                } // end inner for
38
39            } // end outer for
40
41            pw.close();  // Without 'close' the file is empty.
42
43        } // end checkBookingsToFile


44        catch (FileNotFoundException ex) {
45            Logger.getLogger(TrailsManager.class.getName()).log(Level.SEVERE,
null, ex);
46        }
47    }
48
49 }
50
```