```java
/**
 * We affirm that we have carried out our academic endeavors with full academic honesty.
 * [Signed: Christina Vu & Lilah Kelly]
 * Models the process of a hotel front desk reservation.
 * @author Christina Vu and Lilah Kelly
 */

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        //Enter Hotel Info
        Hotel hotelToi = new Hotel(); //change constructor if max amount of beds exceeds 3
        hotelToi.addRooms();
        hotelToi.printHotel();

        //Enter Guest Info
        Scanner inputName = new Scanner(System.in);
        System.out.println("What is guest name?");
        String guestName = inputName.nextLine();
        Scanner inputBeds = new Scanner(System.in);
        System.out.println("How many beds do they need?");
        int bedNo = inputName.nextInt();
        Guest guest = new Guest(guestName, bedNo, 0, false);

        //Finding Room
        Room room = hotelToi.findRoom(bedNo);
        room.setGuestName(guestName);
        room.setUnavail();
        room.printRoom();

        //Update Guest Info
        guest.setRoomNum(room.getRoomNum());
        guest.setCheckedIn();
        guest.printGuest();

        //Hotel Rooms Avail
        System.out.println("Number of rooms available: " + hotelToi.numAvailRooms());
    }
```

```
40  }
41
42
```

```java
/**
 * Models a hotel room.
 * @author Christina Vu and Lilah Kelly
 */
public class Room {
    private String name;
    private int numBed;
    private int roomNum;
    private boolean isAvail;


    /**
     * constructor using guest name, number of beds, room number, and availability
     * @param name name of the guest associated with the room
     * @param numBed number of beds in the room
     * @param roomNum room number
     * @param isAvail current availability of the room
     */
    public Room(String name, int numBed, int roomNum, boolean isAvail) {
        this.name = name;
        this.numBed = numBed;
        this.roomNum = roomNum;
        this.isAvail = isAvail;
    }

    /**
     * Gets the current guest name
     * @return current guest name
     */
    public String getGuestName() {
        return name;
    }

    /**
     * Assigns a new guest name to the room
     * @param name assigns the room's guest name
     */
    public void setGuestName(String name) {
        this.name = name;
    }

```

```
42
43      /**
44       * Gets the number of beds in the room.
45       * @return number of beds
46       */
47      public int getNumBed() {
48          return numBed;
49      }
50
51      /**
52       * Assigns the room a certain number of beds
53       * @param numBed assigned number beds in the room
54       */
55      public void setNumBed(int numBed) {
56          this.numBed = numBed;
57      }
58
59
60      /**
61       * Gets the room number
62       * @return room number
63       */
64      public int getRoomNum() {
65          return roomNum;
66      }
67
68      /**
69       * Assigns a room number to a room object
70       * @param roomNum assigns a room number
71       */
72      public void setRoomNum(int roomNum) {
73          this.roomNum = roomNum;
74      }
75
76      /**
77       * Gives the status of the room.
78       * @return whether the room is available
79       */
80      public boolean isAvail() {
81          return isAvail;   }
82
83      /**
```

```
84      * Changes the status of the room to available.
85      */
86     public void setAvail() {
87         isAvail = true;
88     }
89 
90     /**
91      * Changes the status of the room as unavailable.
92      */
93     public void setUnavail() {
94         isAvail = false;
95     }
96 
97 
98     /**
99      * Changes the status of the room (to the opposite of the current state).
100     * @return changes the availability/unavailability of the room
101     */
102    public boolean changeStatus() {
103        return !isAvail;
104    }
105 
106    /**
107     * Prints the room object with the info
108     */
109    public void printRoom(){
110        System.out.println("Room guest name: " + this.getGuestName() + ",
    number of beds: " + this.getNumBed()
111                + ", room number: " + this.getRoomNum() + ", availability: "
     + this.isAvail());
112    }
113 
114    /**
115     * prints room number
116     */
117    public String printRoomNo() {
118        return "" + this.getRoomNum();
119    }
120 
121    /**
122     * Converts room object to string for testing purposes
123     * @return String of room object
```

```
124      */
125     public String roomToString(){
126         String toReturn = "";
127         toReturn += "Room guest name: " + this.getGuestName() + ",
    number of beds: " + this.getNumBed()
128                 + ", room number: " + this.getRoomNum() + ", current status
    : " + this.isAvail();
129         return toReturn;
130     }
131 }
132
133
134
135
```

```
 1  /**
 2   * Models a Guest for a hotel
 3   * @author: Christina Vu and Lilah Kelly
 4   */
 5  public class Guest{
 6      private String name;
 7      private int numPeople;
 8      private int roomNum;
 9      private boolean checkedIn;
10
11      /**
12       * constructor for Guest class
13       * @param name: name of guest
14       * @param numPeople: number of people intending to stay
15       * @param roomNum the assigned room number
16       * @param checkedIn: indicates if guest is checked in or out
17       */
18      public Guest(String name, int numPeople, int roomNum,boolean
    checkedIn) {
19          this.name = name;
20          this.numPeople = numPeople;
21          this.roomNum = roomNum;
22          this.checkedIn = checkedIn;}
23
24
25      /**
26       * getter method for guest's name
27       * @return String name
28       */
29      public String getName() {    return name;}
30
31      /**
32       * setter method for guest's name
33       * @param newName String name to add to guest info
34       */
35      public void setName(String newName) { this.name = newName; }
36
37      /**
38       * getter method for number of people for guest
39       * @return int number of people
40       */
41      public int getNumPeople() {return numPeople;}
```

```
42
43      /**
44       * setter method for number of people for guest
45       * @param newIdNum int new id number
46       */
47      public void setNumPeople(int newIdNum) { this.numPeople =
    newIdNum; }
48
49      /**
50       * getter method for indicating if guest is checked in
51       * @return boolean of guest is/is not checked in
52       */
53      public boolean getCheckedIn(){ return checkedIn;}
54
55      /**
56       * switches status to checked in
57       * @return changes checkedIn to true
58       */
59      public boolean setCheckedIn(){return checkedIn = true;}
60
61      /**
62       * switches status to checked out
63       * @return changes checkedIn to false
64       */
65      public boolean setCheckedOut() {return checkedIn = false;}
66
67
68      /**
69       * getter method for room number guest is staying in
70       * @return int room number of hotel room
71       */
72      public int getRoomNum() { return roomNum; }
73
74      /**
75       * setter method for room number guest is staying in
76       * @param roomNum int room number of hotel room
77       */
78      public void setRoomNum(int roomNum) { this.roomNum = roomNum; }
79
80      /**
81       * Prints the info for Guest
82       */
```

```
83      public void printGuest(){
84          System.out.println("Guest name: " + this.getName() + ", number of
    occupants: " + this.getNumPeople() + ", room number: " + this.
    getRoomNum() + ", checked in: " + this.getCheckedIn());
85      }
86  }
87
```

```
1  /**
2   * Models a hotel.
3   * @author Christina Vu and Lilah Kelly
4   */
5
6  import java.util.*;
7
8  public class Hotel {
9      int maxCap;
10     Hashtable<Integer, LinkedList<Room>> hotel = new Hashtable<Integer,
   LinkedList<Room>>();
11
12     /**
13      * default constructor
14      */
15     public Hotel(){
16         this(3);
17         maxCap = 3;
18     }
19
20     /**
21      * constructor using the maximum number of beds the hotel offers
22      * @param n the maximum number of beds the hotel offers
23      */
24     public Hotel(int n) {
25         maxCap = n;
26         for(int i = 1; i <= maxCap; i++){
27             LinkedList<Room> listRooms = new LinkedList<Room>();
28             hotel.put(i, listRooms );
29         }
30     }
31
32
33     /**
34      * adds Room object to LinkedList of Room objects based on user's
35      *     preference for amount of beds
36      */
37     public void addRooms(){
38         int roomNo = 1;
39
40         for (int i = 1; i <= maxCap; i++){
41             Scanner inputBedsPerList = new Scanner(System.in);
```

```
42              System.out.println("Enter number of rooms that have " + i + "
    beds.");
43              int bedsPerList = inputBedsPerList.nextInt();
44
45              LinkedList<Room> listRooms = hotel.get(i);
46
47              for (int j = 0; j < bedsPerList; j++){
48
49                  Room aRoom = new Room("None", i, roomNo, true);
50                  listRooms.add(aRoom);
51                  roomNo ++;
52              }
53          }
54      }
55
56      /**
57       * searches LinkedList of Room objects based on number of beds
    wanted. if
58       * the first room in the LinkedList
59       * is available, changes room to not available and returns room.
60       * else, moves on to next room in LinkedList
61       * @param bedNo: number of beds in
62       * @return Room object
63       */
64
65      public Room findRoom(int bedNo){
66          LinkedList<Room> roomList = hotel.get(bedNo);
67          Room currRoom = null;
68          int i = 0;
69          boolean found = false;
70
71          while(!found) {
72              currRoom = roomList.get(i);
73              if (currRoom.isAvail()) {
74                  currRoom.changeStatus();
75                  found = true;
76              }
77              else {
78                  i++;
79              }
80          }
81          return currRoom;
```

```
 82      }
 83
 84
 85      /**
 86       * Gets string of room number from specified LinkedList
 87       * @param list the specific linkedlist of a certain room capacity
 88       * @param index the specific index of the room from the list
 89       * @return string of room number
 90       */
 91      private String getRoom(LinkedList<Room> list, int index) {
 92          Room room = list.get(index);
 93          return "Room no: " + room.printRoomNo();
 94      }
 95
 96
 97      /**
 98       * Gets the list of rooms with a certain capacity
 99       * @param bedCap: bed capacity
100       * @return list of string of rooms for a certain capacity of beds.
101       */
102      private LinkedList<String> getList(int bedCap) {
103          LinkedList<String> newList = new LinkedList<String>();
104          LinkedList<Room> currList = hotel.get(bedCap);
105          for (int i = 0; i < currList.size(); i++) {
106              newList.add(this.getRoom(currList, i));
107          }
108          return newList;
109      }
110
111      /**
112       * Gets the complete lists of bed capacity linked lists
113       * @return string of list of bed capacity lists
114       */
115      private String getLists() {
116          String toReturn = "";
117          ArrayList<LinkedList<String>> array = new ArrayList<LinkedList<String
      >>(maxCap);
118          for(int i = 1; i <= maxCap; i++) {
119              array.add(this.getList(i));
120              toReturn += " Cap: " + i + " "+ this.getList(i);
121          }
122          return toReturn;
```

```
123     }
124
125
126     /**
127      * Prints the hotel rooms
128      */
129     public void printHotel(){
130         System.out.println(this.getLists());
131
132     }
133
134     /**
135      * Gives the number of available rooms
136      * @return the integer of available rooms in the hotel
137      */
138     public int numAvailRooms() {
139         LinkedList<Room> roomList;
140         Room currRoom = null;
141         int numRooms = 0;
142
143         for (int h = 1; h <= hotel.size(); h++) {
144             roomList = hotel.get(h);
145
146             for (int i = 0; i < roomList.size(); i++) {
147                 currRoom = roomList.get(i);
148
149                 if (currRoom.isAvail()) {
150                     numRooms++;
151                 }
152             }
153         }
154         return numRooms;
155     }
156 }
157
```

```
1  /**
2   * This class contains a collection of methods that help with testing.  All
   methods
3   * here are static so there's no need to construct a Testing object.  Just call
   them
4   * with the class name like so:
5   * <p></p>
6   * <code>Testing.assertEquals("test description", expected, actual)</code>
7   *
8   * @author Kristina Striegnitz, Aaron Cass, Chris Fernandes
9   * @version 5/28/18
10  */
11 public class Testing {
12
13    private static boolean VERBOSE = false;
14    private static int numTests;
15    private static int numFails;
16
17    /**
18     * Toggles between a lot of output and little output.
19     *
20     * @param verbose
21     *          If verbose is true, then complete information is printed,
22     *          whether the tests passes or fails. If verbose is false, only
23     *          failures are printed.
24     */
25    public static void setVerbose(boolean verbose)
26    {
27       VERBOSE = verbose;
28    }
29
30    /**
31     * Each of the assertEquals methods tests whether the actual
32     * result equals the expected result. If it does, then the test
33     * passes, otherwise it fails.
34     *
35     * The only difference between these methods is the types of the
36     * parameters.
37     *
38     * All take a String message and two values of some other type to
39     * compare:
40     *
```

```
41      * @param message
42      *          a message or description of the test
43      * @param expected
44      *          the correct, or expected, value
45      * @param actual
46      *          the actual value
47      */
48     public static void assertEquals(String message, boolean expected,
49                                     boolean actual)
50     {
51         printTestCaseInfo(message, "" + expected, "" + actual);
52         if (expected == actual) {
53             pass();
54         } else {
55             fail(message);
56         }
57     }
58
59     public static void assertEquals(String message, int expected, int actual
   )
60     {
61         printTestCaseInfo(message, "" + expected, "" + actual);
62         if (expected == actual) {
63             pass();
64         } else {
65             fail(message);
66         }
67     }
68
69     public static void assertEquals(String message, Object expected,
70                                     Object actual)
71     {
72         String expectedString = "<<null>>";
73         String actualString = "<<null>>";
74         if (expected != null) {
75             expectedString = expected.toString();
76         }
77         if (actual != null) {
78             actualString = actual.toString();
79         }
80         printTestCaseInfo(message, expectedString, actualString);
81
```

```
 82          if (expected == null) {
 83              if (actual == null) {
 84                  pass();
 85              } else {
 86                  fail(message);
 87              }
 88          } else if (expected.equals(actual)) {
 89              pass();
 90          } else {
 91              fail(message);
 92          }
 93      }
 94
 95      /**
 96       * Asserts that a given boolean must be true.  The test fails if
 97       * the boolean is not true.
 98       *
 99       * @param message The test message
100       * @param actual The boolean value asserted to be true.
101       */
102      public static void assertTrue(String message, boolean actual)
103      {
104          assertEquals(message, true, actual);
105      }
106
107      /**
108       * Asserts that a given boolean must be false. The test fails if
109       * the boolean is not false (i.e. if it is true).
110       *
111       * @param message The test message
112       * @param actual The boolean value asserted to be false.
113       */
114      public static void assertFalse(String message, boolean actual)
115      {
116          assertEquals(message, false, actual);
117      }
118
119      private static void printTestCaseInfo(String message, String expected,
120                                            String actual)
121      {
122          if (VERBOSE) {
123              System.out.println(message + ":");
```

```
124             System.out.println("expected: " + expected);
125             System.out.println("actual:   " + actual);
126         }
127     }
128
129     private static void pass()
130     {
131         numTests++;
132
133         if (VERBOSE) {
134             System.out.println("--PASS--");
135             System.out.println();
136         }
137     }
138
139     private static void fail(String description)
140     {
141         numTests++;
142         numFails++;
143
144         if (!VERBOSE) {
145             System.out.print(description + "  ");
146         }
147         System.out.println("--FAIL--");
148         System.out.println();
149     }
150
151     /**
152      * Prints a header for a section of tests.
153      *
154      * @param sectionTitle The header that should be printed.
155      */
156     public static void testSection(String sectionTitle)
157     {
158         if (VERBOSE) {
159             int dashCount = sectionTitle.length();
160             System.out.println(sectionTitle);
161             for (int i = 0; i < dashCount; i++) {
162                 System.out.print("-");
163             }
164             System.out.println();
165             System.out.println();
```

```
166         }
167     }
168 
169     /**
170      * Initializes the test suite. Should be called before running any
171      * tests, so that passes and fails are correctly tallied.
172      */
173     public static void startTests()
174     {
175         System.out.println("Starting Tests");
176         System.out.println();
177         numTests = 0;
178         numFails = 0;
179     }
180 
181     /**
182      * Prints out summary data at end of tests.  Should be called
183      * after all the tests have run.
184      */
185     public static void finishTests()
186     {
187         System.out.println("==============");
188         System.out.println("Tests Complete");
189         System.out.println("==============");
190         int numPasses = numTests - numFails;
191 
192         System.out.print(numPasses + "/" + numTests + " PASS ");
193         System.out.printf("(pass rate: %.1f%s)\n",
194                           100 * ((double) numPasses) / numTests,
195                           "%");
196 
197         System.out.print(numFails + "/" + numTests + " FAIL ");
198         System.out.printf("(fail rate: %.1f%s)\n",
199                           100 * ((double) numFails) / numTests,
200                           "%");
201     }
202 
203 }
204 
```

```
1  import java.util.*;
2
3  public class TestingClass {
4
5      public static void main(String[] args){
6          Testing.startTests();
7          testingFindRoom();
8          testingNumAvail();
9          Testing.finishTests();
10     }
11     public static Hotel makeHotel(){
12         Hotel hotel = new Hotel();
13         System.out.println("Input 2 for first prompt, 2 for second, and 2
   for third");
14         hotel.addRooms();
15         return hotel;
16         }
17
18
19     public static void testingFindRoom(){
20         Hotel hotel = makeHotel();
21         int bedNo = 1;
22         boolean expected = true;
23         Room roomExpected = new Room("None", bedNo, 2, true);
24         Room room = hotel.findRoom(1);
25         String strRoom = room.roomToString();
26         String strRoomExpected = roomExpected.roomToString();
27         boolean actual = strRoom.equals(strRoomExpected);
28         Testing.assertEquals("Testing Find Room function in Hotel Class:"
   , expected, actual);
29     }
30
31     public static void testingNumAvail() {
32         Hotel hotel = makeHotel();
33         int bedNo = 1;
34         int roomsAvail = 5;
35         boolean expected = true;
36         Room room = hotel.findRoom(1);
37         room.setUnavail();
38         boolean actual = roomsAvail == hotel.numAvailRooms();
39         Testing.assertEquals("Testing number of rooms available function
    in Hotel Class", expected, actual);
```

```
40      }
41  }
42
43
```