P03 Room Reservations

Overview

In the Before Times, we had a bit of flexibility with room capacity. I mean, not *technically* -- the fire code is the fire code, after all -- but who among us hasn't been in an overcrowded lecture hall during the first week of classes, with people sitting in the aisles or leaning against the walls?

All of us, now, apparently. Thanks COVID.

To make sure we can effectively social distance while on campus, you've been asked to implement a room reservation system using Object-Oriented design, to make sure we're effectively tracking how people are utilizing the spaces on campus for learning and studying.

We'll revisit this design later this semester as we learn data structures, but for now the application will consist of:

- a **Person**, who has a name and a status (waiting/in-room);
- a **Room**, which contains an array of **Persons**; and
- a **TrackingApp**, which checks **Persons** in and out of **Rooms**

In this specification, we will walk you through the creation of these objects and get you started with your test methods.

Grading Rubric

5 points	**Pre-assignment Quiz**: accessible through Canvas until 11:59PM on **02/22**.
20 points	**Immediate Automated Tests**: accessible by submission to Gradescope. You will receive feedback from these tests *before* the submission deadline and may make changes to your code in order to pass these tests. Passing all immediate automated tests does **not** guarantee full credit for the assignment.
15 points	**Additional Automated Tests**: these will also run on submission to Gradescope, but you will not receive feedback from these tests until after the submission deadline.
10 points	**Manual Grading Feedback**: TAs or graders will manually review your code, focusing on algorithms, use of programming constructs, and style/readability.

©2021 Mouna Ayari Ben Hadj Kacem and Hobbes LeGault — University of Wisconsin–Madison

Learning Objectives

The goals of this assignment are:

- Familiarize yourself with the typical setup of Object-Oriented Programming
- Create and use a series of interrelated objects
- Create constructors, accessors, and mutators to set, examine, and modify private data fields
- Familiarize yourself with the difference between (static) class methods and data, and (non-static) instance methods and data

Additional Assignment Requirements and Notes

Keep in mind:

- You are allowed to define any local variables you may need to implement the methods in this specification.
- All methods, public or private, must have their own Javadoc-style method header comments in accordance with the CS 300 Course Style Guide.
- Any source code provided in this specification may be included verbatim in your program without attribution.
- Do NOT include any import statements besides java.util.ArrayList and any relevant exceptions.
- Do NOT define any additional class or instance fields beyond the ones detailed in this specification.
- You are allowed to define any **local** variables you may need to implement the methods in this specification.
- You are allowed to define additional **private** helper methods to help implement the methods in this specification if you like.

©2021 Mouna Ayari Ben Hadj Kacem and Hobbes LeGault — University of Wisconsin–Madison

CS 300 Assignment Requirements

You are responsible for following the requirements listed on both of these pages on all CS 300 assignments, whether you've read them recently or not. Take a moment to review them if it's been a while:

- Academic Conduct Expectations and Advice, which addresses such questions as:
  - How much can you talk to your classmates?
  - How much can you look up on the internet?
  - What do I do about hardware problems?
  - and more!
- Course Style Guide, which addresses such questions as:
  - What should my source code look like?
  - How much should I comment?
  - and more!

Getting Started

1. Create a new project in Eclipse, called something like **P03 Room Reservations**.
   a. Ensure this project uses Java 11. Select "JavaSE-11" under "Use an execution environment JRE" in the New Java Project dialog box.
   b. Do NOT create a project-specific package; use the default package.
2. Create four (4) Java source files within that project's src folder:
   a. **Person**.java (does NOT include a main method)
   b. **Room**.java (does NOT include a main method)
   c. **TrackingApp**.java (download it here; includes a main method)
   d. **OccupancyTester**.java (includes a main method)

The methods in the **Person** and **TrackingApp** classes will NOT be static methods; all methods in **OccupancyTester** should be static; the **Room** class will contain a mix of static and non-static methods.

©2021 Mouna Ayari Ben Hadj Kacem and Hobbes LeGault — University of Wisconsin–Madison

Implementation Requirements Overview

Your **Person** class must contain the following **private** instance variables:

- `name`, a String representing the Person's name
- `isWaiting`, a boolean which has the value true if and only if the Person is NOT currently in a Room

Your Person class must contain the following instance methods:

- `public Person(String name)`
  - A one-argument constructor, which initializes the instance variables for the object
  - Person objects should be assumed to NOT be in a Room at the time of creation
- Accessor methods for both of the object's instance variables:
  - `public String getName()` to get the name.
  - `public boolean isWaiting()` to get the waiting status.
- `public void toggleWaiting()`
  - Sets isWaiting to true if it is currently false, and to false if it is currently true
- ```
  public boolean equals(Object o) {
    if (o instanceof Person) {
      return this.name.equals(((Person) o).name);
    }
    return false;
  }
  ```

Your **Room** class must contain the following **private** class variable:

- `names`, an ArrayList of Strings containing the currently-used name identifiers across all instances of Room objects. This must be initialized to an empty ArrayList when defined (outside of any methods).

Your **Room** class must contain the following class method:

- `public static String[] getNames()`
  - Returns the current list of Room names as an array of Strings
  - These names MUST appear in the order they were created!

Your **Room** class must contain the following **private** instance variables:

- `name`, a String representing the name of the Room, which must be unique
- `occupants`, a **perfect-size** array of Persons currently in the Room, with social distancing enforced: Person objects may only occupy elements with even indexes, all odd indexes must always contain null
- `currentOccupancy`, an int counting the current number of Persons in the Room

©2021 Mouna Ayari Ben Hadj Kacem and Hobbes LeGault — University of Wisconsin–Madison
  ```

Your **Room** class must contain the following instance methods:

- `public Room(String name, int capacity)`
  - A two-argument constructor, which initializes the instance variables for the object
  - Note that capacity is the maximum number of Persons who can occupy the room ***without*** social distancing enforced
  - If the provided capacity is 0 or less, or if the name is already in use by another Room object, this constructor should throw an IllegalArgumentException with a descriptive error message; do not make any changes to the Room.names ArrayList
  - If the Room is created successfully, add its name to the Room.names ArrayList

- Accessor methods:
  - `public String getName()` returns the name of this Room
  - `public int getOccupancy()` returns the current number of people in the Room
  - `public int getCOVIDCapacity()` returns the number of people allowed in the Room under COVID protocols, defined for our purposes as half of normal capacity
  - `public int getCapacity()` returns the number of people allowed in the Room under normal conditions
  - `public boolean contains(Person p)` returns true if and only if the provided Person is present in the Room's occupants array

- Mutator methods:
  - `public boolean checkIn(Person in)` returns true if and only if the provided Person was successfully added to the room.
    - You should add Persons to the array **optimally** -- that is, to *only the first available **even-indexed*** element.
    - If the currentOccupancy is equal to the COVID capacity, do NOT check the Person into the Room, and return false.
    - If the Person is added successfully, remember to increment currentOccupancy AND toggle their isWaiting field before you return true.
    - If the Person passed as input to this method is null OR IS ALREADY IN THE ROOM, throw an IllegalArgumentException with a descriptive error message.

    Example of normal (non-error) behavior:
    - occupants: {"Hobbes", null, null, null, "Mouna", null, null, null, null}
    - currentOccupancy: 2

    After checkIn(new Person("Michelle")) is called, the state of the room will be:
    - occupants: {"Hobbes", null, "Michelle", null, "Mouna", null, null, null, null}
    - currentOccupancy: 3

  - `public boolean checkOut(Person out)` returns true if and only if the provided Person was successfully removed from the Room.

©2021 Mouna Ayari Ben Hadj Kacem and Hobbes LeGault — University of Wisconsin–Madison

  - If the Person is located in the Room, their isWaiting value should be toggled to true, the currentOccupancy of the Room should be decremented, and their index in the occupants array should be set to null before returning true.
  - If the provided Person was NOT present in the Room, the occupants array and currentOccupancy should NOT change and the method should return false.
  - If the Person passed as input to this method is null, throw an IllegalArgumentException with a descriptive error message.

- `public String toString()`
  - Returns a String representation of this Room and its occupants
  - For example, a valid Room named "CS 1240" with a capacity of 9 and a currentOccupancy of 4 might return a String like this, where Persons are represented with their names and empty occupants are represented with a single dash:

```
CS 1240
===
Mouna
-
Hobbes
-
-
-
Michelle
-
Sulong
```

The **TrackingApp** class (download it here) is a basic text-based menu loop built on top of Person and Room, which you can use for interactive testing and Just Messing Around with your code.

Be warned, Hobbes was not feeling particularly charitable when she wrote it, so it breaks easily with careless keyboard input. You're welcome to make it more robust if you like; we will not be grading this class at all and it is provided solely for your convenience and entertainment.

Your **OccupancyTester** class must contain boolean test methods for each of *your* classes (do not write test methods for the **TrackingApp** class). You must write:

- `public static boolean testPerson()` to test the constructor, accessors, mutator, and equals method of your **Person** class
- `public static boolean testRoomConstructor()` to test the constructor of your **Room** class

©2021 Mouna Ayari Ben Hadj Kacem and Hobbes LeGault — University of Wisconsin–Madison

- `public static boolean testRoomAccessors()` to test the accessor methods of your **Room** class -- this includes both the static AND instance accessor methods!
- `public static boolean testRoomCheckIn()` to test the check-in functionality and its effects on both the **Room** and the **Person** being checked in
- `public static boolean testRoomCheckOut()` to test the check-out functionality and its effects on both the Room and the Person being checked out
- `public static boolean testRoomToString()` to verify the result of **Room**'s toString method

If you like, you may break these into further private helper methods as permitted in the Additional Assignment Requirements and Notes at the beginning of this document, but these methods must exist as described in your tester class.

You may add any printed output you wish to these methods, but they MUST return true if the test(s) is/are successful and false otherwise.

Implementation Details and Suggestions

We recommend that you begin your implementation with the **Person** class, as it's simple and doesn't depend on any of the other classes.

Person: Implement and Test

Follow the implementation requirements outlined above, and test your class as follows:

- Create two (or more) instances of the **Person** object by calling the constructor with different names.
- Use the accessor methods to verify that each object returns different values (and that they're the ones you expect for that particular instance).
- Verify that the toggleWaiting() method flips the isWaiting value from false to true and back.
- Verify that two **Person** instances with different names are NOT considered equal, and that two **Person** instances with the same name ARE considered equal. A **Person** and any other object (say, a String) should NOT be considered equal.

Room: Implement and Test

Once you have verified that your **Person** class works as expected, move on to the **Room** class. Start with the class (static) variable and method. The instance variables and their accessors are very similar to those of the **Person** class, and the constructor only has its one little uniqueness twist.

STOP: write the tests for these. Create at least two different **Rooms**, verify that their accessors give you different values. Try creating two **Rooms** with the same name. Make sure you can't. Make sure the names array is in the correct order.

©2021 Mouna Ayari Ben Hadj Kacem and Hobbes LeGault — University of Wisconsin–Madison

Next, write your toString() method, so you can see what's happening in the **Room**.

Work on the checkIn() method. Where does the next **Person** go? How many can you fit? Use the toString() method to monitor the **Room** as you check more **Persons** in. (These should be tests in your OccupancyTester class, btw.) Remember to toggle the **Person**'s isWaiting variable when you check them in.

Finally, implement and test checkOut(). Make sure you flip the checked-out **Person** back to waiting when they leave the **Room**!

Assignment Submission

Hooray, you've finished this CS 300 programming assignment!

Once you're satisfied with your work, both in terms of adherence to this specification and the academic conduct and style guide requirements, submit your source code through Gradescope.

For full credit, please submit **ONLY** the following files (source code, *NOT* .class files):

- **Person**.java
- **Room**.java
- **OccupancyTester**.java

Your score for this assignment will be based on the submission marked "**active**" prior to the deadline. You may select which submission to mark active at any time, but by default this will be your **most recent** submission.

Copyright Notice

This assignment specification is the intellectual property of Mouna Ayari Ben Hadj Kacem, Hobbes LeGault, and the University of Wisconsin–Madison and may not be shared without express, written permission.

Additionally, students are not permitted to share source code for their CS 300 projects on any public site.

Appendix: Sample Output

Here's some quick sample output from the **TrackingApp** main method, if you'd like to compare your usage to ours!

```
TRACKING APP MENU
=================
A) Admin functions
B) Check in/out
C) Quick example setup
D) Quit
> c
TRACKING APP MENU
=================
A) Admin functions
B) Check in/out
C) Quick example setup
D) Quit
> a
ADMIN FUNCTIONS
===============
1) Add new room
2) Check room occupancy
3) Create new person
> 2
  1) CS 1240
  2) AG 100
Which room: 1
CS 1240
===
Mouna
-
Hobbes
-
Michelle
-
-
TRACKING APP MENU
=================
A) Admin functions
B) Check in/out
C) Quick example setup
```

©2021 Mouna Ayari Ben Hadj Kacem and Hobbes LeGault — University of Wisconsin–Madison

```
D) Quit
> b
CHECK IN/OUT MENU
=================
1) Check person in
2) Check person out
3) List all people
4) List all rooms
> 2
Which room number? (-1 to see a list of all rooms): 1
Which person? (-1 to see a list of all people): -1
  1) Mouna
  2) Hobbes
  3) Michelle
  4) Sulong
Which person? (-1 to see a list of all people): 2
Successfully checked Hobbes out of room CS 1240
TRACKING APP MENU
=================
A) Admin functions
B) Check in/out
C) Quick example setup
D) Quit
> a
ADMIN FUNCTIONS
===============
1) Add new room
2) Check room occupancy
3) Create new person
> 2
  1) CS 1240
  2) AG 100
Which room: 1
CS 1240
===
Mouna
-
-
-
Michelle
-
-
TRACKING APP MENU
=================
A) Admin functions
```

©2021 Mouna Ayari Ben Hadj Kacem and Hobbes LeGault — University of Wisconsin–Madison

```
B) Check in/out
C) Quick example setup
D) Quit
> d
Good-bye!
```

Note this is very brief and does not explore the functionality of manually adding Person or Room objects, or checking anyone *into* a Room.

©2021 Mouna Ayari Ben Hadj Kacem and Hobbes LeGault — University of Wisconsin–Madison