

ITMD 510 MP 3 – Hotel Registration

Due: November 19, 2015 6:25pm (**No Late Submissions**)

1 Objectives

The objectives of this MP are to demonstrate mastery of the following course objectives:

- Implement Java applications that leverage the object-oriented features of the Java language
- Understand and implement applications that use the Java collections framework.
- Implement an application that has a GUI interface.

2 Introduction

For this MP, you will implement an application that is used by a clerk at a hotel to register guests. The main interface displays all of the available rooms, whether or not there is a vacancy in the room and how many rooms of the various types are available.

When a guest arrives at the hotel, the clerk enters the name and address of the guest, what type of room the guest wants and then selects a room based on the desired type and availability. When a guest is ready to depart the hotel, the clerk will then check out the guest from the room.

3 The Hotel

The hotel consists of 15 rooms on a single floor. The rooms come in the following types:

- One King Bed
- One Queen Bed
- Two Double Beds
- Two Double Beds and One Cot

The room numbers, the room types and the maximum occupancy for the room is given in the table below:

Room Number	Room Type	Maximum Occupancy
100	One King Bed	2
101	Two Double Beds	4
102	One King Bed	2
103	Two Double Beds and One Cot	5
104	One Queen Bed	2
105	One King Bed	2
106	One Queen Bed	2
107	Two Double Beds	4
108	One Queen Bed	4
109	Two Double Beds and One Cot	5
110	Two Double Beds	4
111	Two Double Beds	4
112	Two Double Beds	4
113	Two Double Beds	4
114	One King Bed	2

In a completely vacant hotel, there are

- 4 One King Bed rooms
- 3 One Queen Bed rooms
- 6 Two Double Beds rooms
- 2 Two Double Beds and One Cot rooms

4 Interface

This section describes the user interface for the hotel registry application.

4.1 Main Frame

The main frame of the hotel registry has the following components:

- a menu bar that allows access to registry functionality,
- a hotel registry display that shows all of the rooms and who is registered in that room
- a display that shows all of the room types and how many rooms are vacant

The layout of the main frame is shown here:

Room	Guest	Vacancies
100:	Mickey Mouse	One King 3
101:	Donald Duck	
102:	Vacant	
103:		
104:		One Queen 3
105:		
106:		
107:		
108:		Two Double Beds: 5
109:		
110:		
111:		
112:		Two Double Beds w/Cot 2
113:		
114:		

The functionality in the main frame is described below.

Menu Bar

The menu bar has two menu, Registry and Help.

The Registry menu has the following menu items:

- “Check In” - Used to check a guest into a room. When selected, the Check In form will pop up. The Check In form is detailed below.
- “Check Out” - Used to check a guest out of the room. When selected, the Check Out form will pop up. The Check Out form is detailed below.
- “Exit” - Exits the application

The Help menu has one menu item called “About”. Selecting this menu item should show a pop-up dialog with the following text:

- Your Name

- Your Student ID Number

The About dialog should have an “OK” button which when pressed dismisses the dialog.

Registry Display

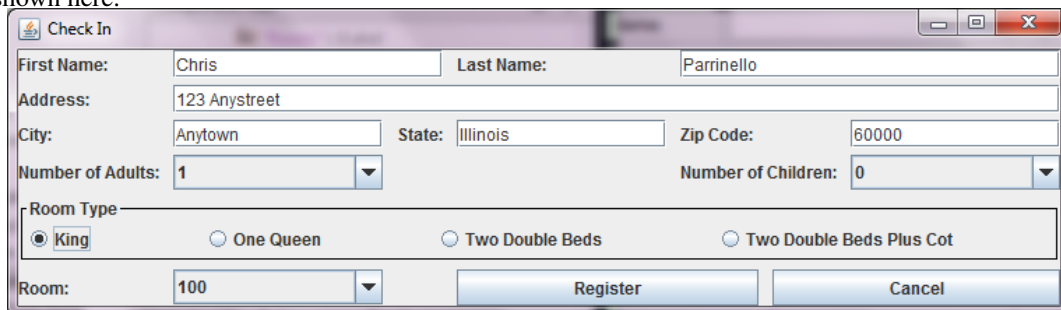
The registry display appears on the left side of the frame. There is a label for each room number and a text field which displays either the name of the guest currently checked into that room (for example, Mickey Mouse) or “Vacant” if nobody is checked into that room. None of the text fields in this display should be editable. This display should be updated dynamically based on the clerk checking in and checking out guests.

Vacancy Display

The vacancy display appears on the right side of the frame. There is a label for each room type and a text field which displays the total number of rooms that are vacant for that type. None of the text fields in this display should be editable. This display should be updated dynamically based on the clerk checking in and checking out guests.

4.2 Check In Form

The Check In form is used to gather information about the guest checking in, select a room type and select an available room. When the form is completed, the guest is checked into the room. The layout of the form is shown here:



The screenshot shows a "Check In" dialog box with the following fields and controls:

- First Name:** Text field containing "Chris"
- Last Name:** Text field containing "Parrinello"
- Address:** Text field containing "123 Anystreet"
- City:** Text field containing "Anytown"
- State:** Text field containing "Illinois"
- Zip Code:** Text field containing "60000"
- Number of Adults:** Spin box set to "1"
- Number of Children:** Spin box set to "0"
- Room Type:** Radio button group with four options:
 - ☒ King
 - ☐ One Queen
 - ☐ Two Double Beds
 - ☐ Two Double Beds Plus Cot
- Room:** Spin box set to "100"
- Buttons:** "Register" and "Cancel" buttons at the bottom right.

The form has the following editable text fields:

- First Name

- Last Name
- Address
- City
- State
- Zip Code

The form has two combo boxes which are used by the clerk to select how many adults and children are staying in the room. For the Number of Adults, the valid values are 1, 2, 3, 4, 5. For the Number of Children, the valid values are 0, 1, 2, 3, 4.

The form has a radio button group called Room Type used to select the room type. The room types available are described in section 2.

The form has a combo box called Room which lists the room numbers of the available rooms. A room number should appear in this combo box if and only if:

- The room is vacant.
- The room type matches the room type selected in the radio button group.
- The total number of adults and children in the form is less than or equal to the maximum occupancy for the room.

The Room combo box is dynamically updated based on the selections made in the Number of Adults, Number of Children and Room Type components. If no rooms match the criteria above, no room numbers should be selectable in the combo box.

To complete the form, the clerk presses the “Register” button. When the “Register” button is selected, the following validations are performed:

- Validate that First Name, Last Name, Address, City, State and Zip Code are not blank.
- Validate that a Room Type was selected.
- Validate that a Room was selected.

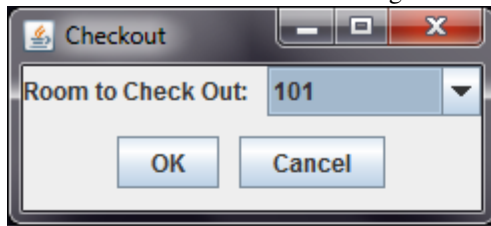
If a validation fails, display a popup dialog stating “Validation Failure!” which can be dismissed with an OK button. Once the “Validation Failure” dialog is dismissed, the clerk is returned to the Check In form to correct the validation mistakes.

If validation is successful, the guest is checked into the room. The Check In form is dismissed and the clerk is returned to the main frame. The main frame should be updated to display the name of the guest checked into the selected room and the vacancy totals should also be updated as well.

If the clerk decides to cancel the check in, pressing the “Cancel” button dismisses the Check In form and the clerk is returned to the main frame.

4.3 Check Out Form

The Check Out form is used to check a guest out of a room. The layout of the form is shown here:

A screenshot of a software window titled "Checkout". The window has a standard Windows-style title bar with minimize, maximize, and close buttons. Inside the window, there is a label "Room to Check Out:" followed by a text box containing the number "101" and a small downward-pointing arrow, indicating a dropdown menu. Below the text box are two buttons: "OK" and "Cancel".

The form has a combo box called Room to Check Out. The combo box is dynamically updated to contain the room numbers of all of the occupied rooms. The clerk selects the room to check out and then presses the “OK” button. The guest is then checked out of the room. The Check Out form is dismissed and the clerk is returned to the main frame. Like in the Check In form, the main frame should be updated to reflect the guest checking out of the room by updating the display for that room to display “Vacant” and the vacancy totals should be updated as well.

If the clerk decides to cancel the check out, pressing the “Cancel” button dismisses the Check In form and the clerk is returned to the main frame.

5 Requirements

5.1 Functional Requirements

You will write a Java program that implements the hotel registration application described above. The program is started by running the Java interpreter and providing the classpath and main class name:

```
java -classpath <class files directory> HotelRegistry
```

5.2 Non-Functional Requirements

Implementation Requirements

1. Implementation should consist of a good object-oriented design where:
 - (a) The program is decomposed into multiple classes.
 - (b) Those classes use encapsulation, inheritance and polymorphism to implement the application where appropriate.
 - (c) The Model-View-Controller pattern is utilized in the design of the application.
2. The class should not depend on any third-party libraries or classes other than what comes in the Java SE API.
3. The source code should follow standard Java conventions and style:
 - (a) Proper indentation and brace usage.
 - (b) Proper use of vertical spacing. Each method definition should be separated by a space.
 - (c) Variable names should be descriptive and relevant. One letter variable names are only appropriate in loops.
 - (d) Methods should be:
 - i. Not overly long. More than a half a screenful is probably too long. Break it up into smaller methods.
 - ii. Named properly. Method names should be verbs or predicates.
 - (e) Comments should be used liberally to explain your code or what the program is doing.

GUI Requirements

1. The GUI should implement all of the functionality and non-functional (titles, borders, labels, etc.) GUI components just like the pictures of the GUI above. The layout of the elements should match their relative positions with one another (i.e. First Name and Last Name are on the same row like in the picture). However you will not be graded on whether or not the sizes of the various elements match the example pictures. Nor will you be graded on whether or not your applications' "whitespace" around the components match the example pictures.
2. The GUI frames and dialogs should not be resizable.
3. Read-only elements should not be modifyable.

4. The GUI should dynamically update based on the actions of the user. If the user checks-in a guest, the Registry display should update. If the user selects One Queen Bed, the available rooms in the check in form should update, etc.

Documentation Requirements

1. In addition to the source code, you will also create a `README.txt` file with the following contents:
 - (a) Your name and student ID number
 - (b) Answers to the following questions:
 - i. How do you run your program (i.e. what is the command line)?
 - ii. Describe your object-oriented design for the program:
 - A. How are you implementing the hotel registry?
 - B. How are you implementing the different types of rooms?
 - C. How are you dynamically updating the forms and main frame based on the clerk's actions?
 - iii. What specific problems or challenges did you have implementing your solution? For example, was there a particular requirement that you had difficulty implementing? Or perhaps there was a particularly nasty bug in your implementation you had problems tracking down.
 - iv. Were there any requirements that were not implemented or not implemented properly in your solution? If so, please elaborate.
 - v. Were there any requirements that were vague or open to interpretation that you had to make a decision on how to implement? How did you elect to interpret them?
 - vi. How would you rate the complexity of this MP on a scale of 1 to 10 where 1 is very easy and 10 is very difficult. Why did you give this rating?

Submission Requirements

To submit your solution to this MP, please place the following files into a file in the ZIP file format (i.e. I should be able to unzip the file using Windows' extract tool, or the `unzip` command in Linux. No **RAR**, no **7z**. You will lose points if you do not follow this requirement):

- `README.txt`
- A `src` directory containing the directory structure that has all of your `.java` source code files. If you use package declarations, make sure your `.java` files are in a directory structure that reflects the package declarations (i.e. a class called `HotelRegistry` that is in the `edu.iit.itmd510` package should have a path of `src/edu/iit/itmd510/HotelRegistry.java` in you zip file).
- A `bin`, `target` or `classes` directory (depending on your IDE) containing all of the compiled `.class` files for your program. If you use package declarations, make sure your `.class` files are in a directory structure that reflects the package declarations (i.e. a class called `HotelRegistry` that is in the `edu.iit.itmd510` package should have a path of `bin/edu/iit/itmd510/HotelRegistry.class` in your zip file).

Please name this zip file in the following format `Name-StudentID.zip` (you will lose points for not following this format).

6 Grading

Your implementation will be graded on a scale of 100 points. The breakdown of the points is as follows:

- **40 points for correctness.** Correctness is whether or not your program compiles, executes with no errors and implements the requirements correctly. The functionality of the GUI should be implemented based on the requirements above.
- **35 points for completeness.** Completeness is whether or not your program implements all of the functional and non-functional requirements above.
- **15 points for object-oriented design.** Program should use classes, objects, encapsulation, inheritance and polymorphism where appropriate. The program should use the Model-View-Controller design pattern.
- **5 points for GUI look-and-feel requirements.** Your program should implement the main frame and forms as described in the non-functional requirements with respect to layout.
- **5 points for coding conventions, style and documentation.** Submission follows coding standards per requirements above, follows the documentation requirements and the submission requirements.

7 Plagiarism and Copying

- **All solutions will be checked for plagiarism and copying via automated tools and manual inspection.**
- **Students found to have copied code from another student or the Internet will receive a zero for this assignment and will be reported for an Academic Code Violation.**
- **This is not a group assignment, this is an individual assignment.** You may discuss the assignment with other students but you are not allowed to give code to, share code with or show your code to your fellow ITMD 510 students. When code is shared, there is no way for me to determine which student copied from whom so all parties involved might receive a zero.

8 Suggestions

- Start working on the MP early so that you have enough time to ask questions during class, during office hours, on Blackboard, etc.
- Do not spend too much time on the layout of the GUI. Concentrate your time on implementing the dynamic nature of the GUI. You can always come back to polish the look-and-feel after the core functionality is complete.
- Since this MP has no console output, feel free to use the console for logging messages to help debug the program.
- To implement the dynamic GUI functionality, recall callbacks, the observer pattern, etc. from our lectures. How does this fit with a Model-View-Controller design pattern? What would be the Model? What would be the View? What would be the Controller?

- Although there is no unit test requirement, consider writing unit tests to make sure the functionality of determining how many vacancies there are, what rooms are available, etc. works without the need for testing through the GUI.