**ITP4507**

**Contemporary Topics in Software Engineering**

**Assignment Report**

**Name: Cheng Ka Ming**

**Student Number: 210211001**

**Application design with class diagram**

Please refer to file *assignment\_classDiagram.pdf*.

**Discussion and explanation on each of the design patterns applied to the application**

Command Pattern:

Applied to integrate different actions(commands) of the user and provide an undo/redo function.

Command is an interface class, Command\_addBuilding, Command\_addRoom… are subclasses of Command class.

In this case, Those Command objects store actions(command) the user has made, for example Command\_addBuilding stores the action of adding different buildings to the buildingList, and the undo function will remove the building this Command object has made from the buildingList. Further explanation of this pattern will be in the next part (Factory Pattern on Command objects).

For the classes Command\_modifyBuilding and Command\_modifyRoom, there are two different functions get\_memento\_from\_caretaker() and saveToMemento() which are the application of the Memento pattern, I will explain in that part.

Factory Pattern:

Applied to Command objects and Building objects, but not Room objects.

*Factory pattern on Command Object*:

CommandFactory is an interface class, Factory\_addBuilding, Factory\_addRoom… are the subclasses of CommandFactory class, they are the concrete creators of Command objects.

In this case, most of the reactions between user and the system are placed here, like inputting different data to add or modify Building / Room objects, and the Command object will never need to require the user to input data. That means the create() function of CommandFactory classes will prepare all data and pass it to Command objects, and then the Command objects will finish the last work with execute() (like putting the Building object to the buildingList, adding new room to existing building etc.).

*Factory pattern on Building Object*:

Building\_Creator is an abstract class, Concrete\_Creator\_Apartment and Concrete\_Creator\_House are the subclasses of CommandFactory class, they are the concrete creators of Building objects.

In this case, the Factory\_addBuilding class has a Building\_Creator, and when the previous code finishes the recognition of user’s input (a/h), a different creator will be created(new Concrete\_Creator\_Apartment / Concrete\_Creator\_House) to decide which type of building will be create and put into buildingList. With this structure, if a new type of Building is going to be added, only 2 new classes are needed which are the class of its own self and its concrete creator, and only 1-2 lines of codes is needed to add to the Main class and Factory\_addBuilding class finish the extension.

Memento Pattern:

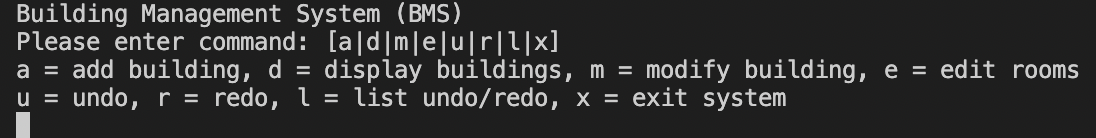
Applied to classes Command\_modifyBuiding and Command\_modifyRoom and Main, aimed to save redo data of modifying buildings and rooms.

In this case, Memento and Originator classes are put into Memento package, and the Memento class will save the old data of Building / Room objects via Originator before the modifying is made by calling saveToOriginator() method in main class, and then the returned Memento object will be saved to Caretaker class which holding a list of Memento objects. When undo/redo, the Caretaker will get saved data from the Memento object in mementoList and send it to the Command object, then the Command object can restore the old data.

I used the same Memento class to save data of both Building and Room objects. It is because I used a raw-type ArrayList to save data so that it can save any type of data and be implemented to both Building and Room objects by saving data to ArrayList in a specific order.

**Test Plan and Test Cases**

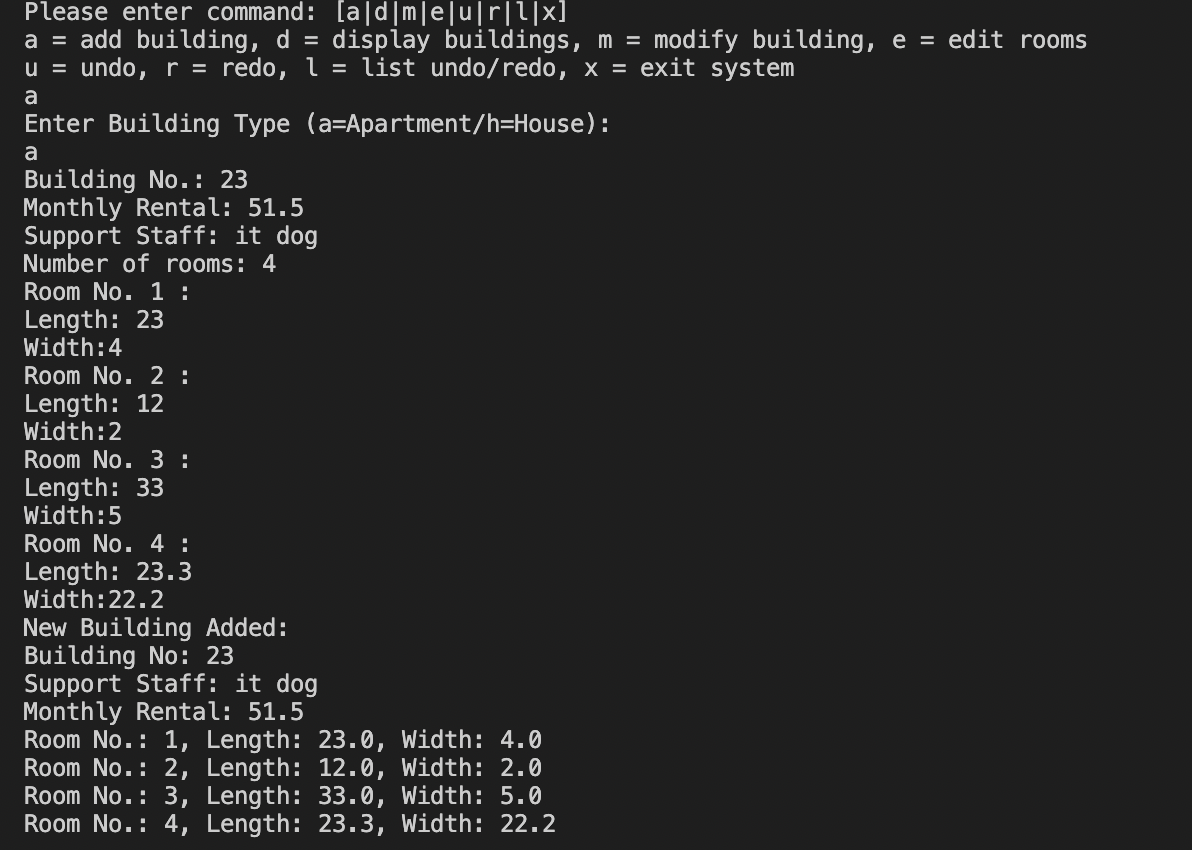
Program start:

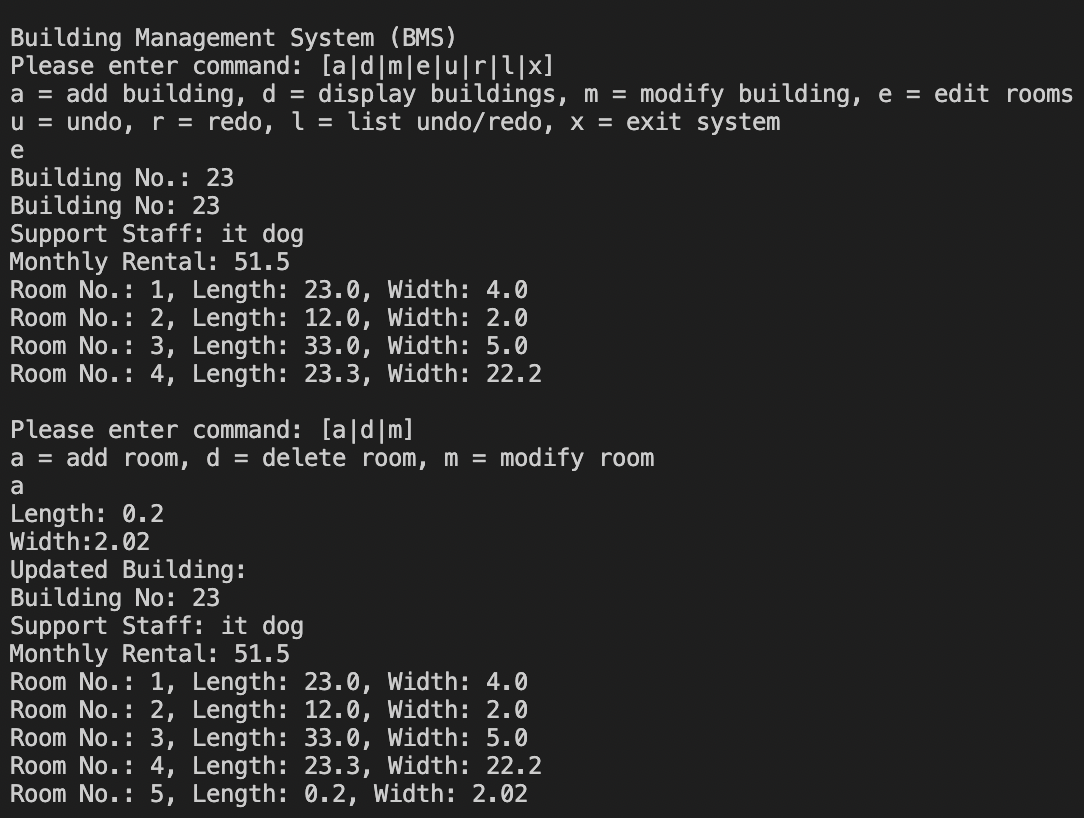
****

1. Add building and add room (apartment):

Test data: “a a 51.5 it dog 4 23 4 12 2 33 5 23.3 22.2 e 23 a 0.2 2.02”

Result:

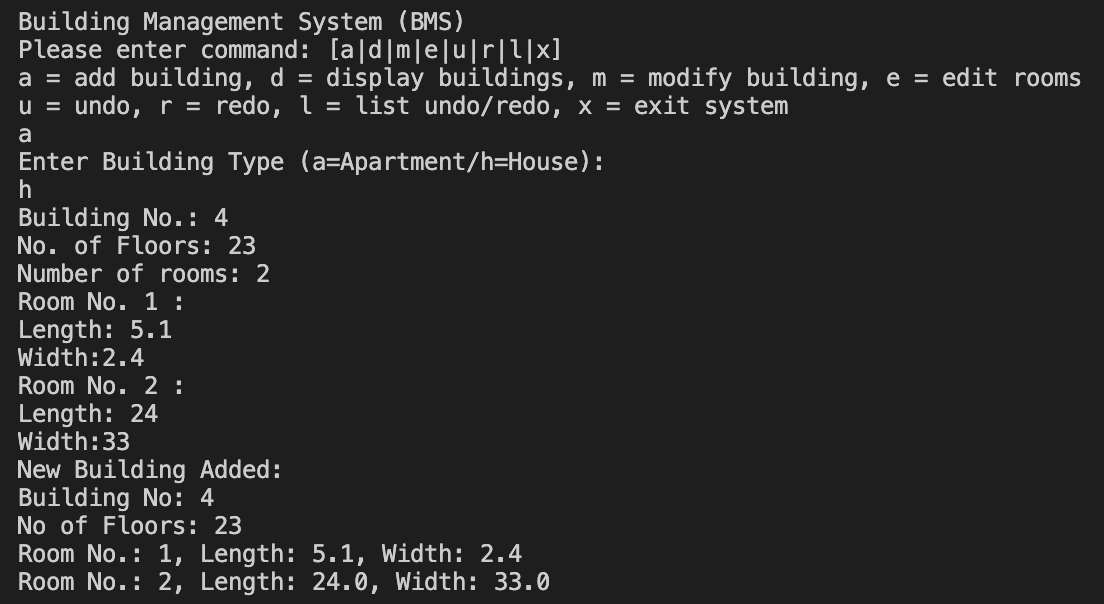


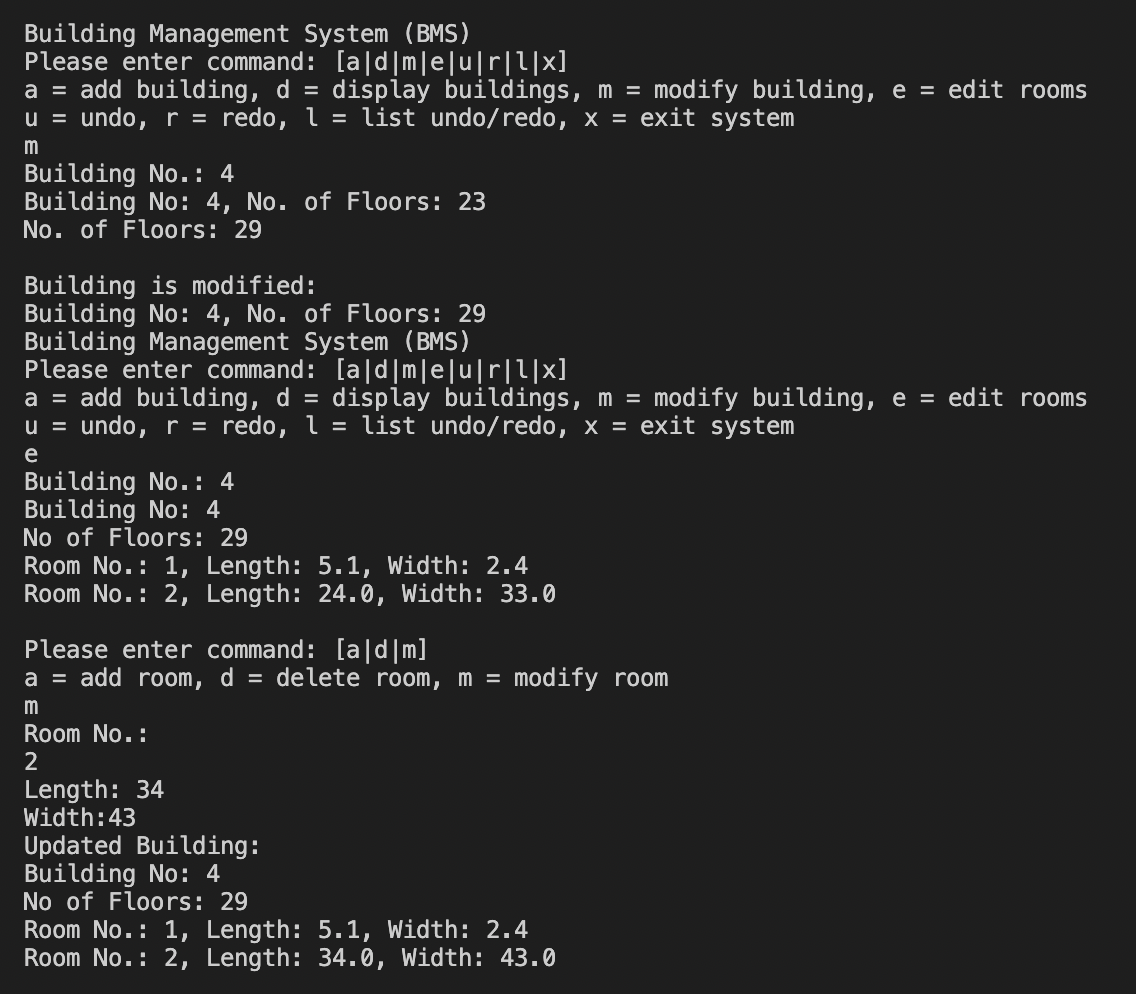


2. Add building and then modify building and room (house):

Test data: “a h 4 23 2 5.1 2.4 24 33 m 4 29 e m 2 34 43”

Result:

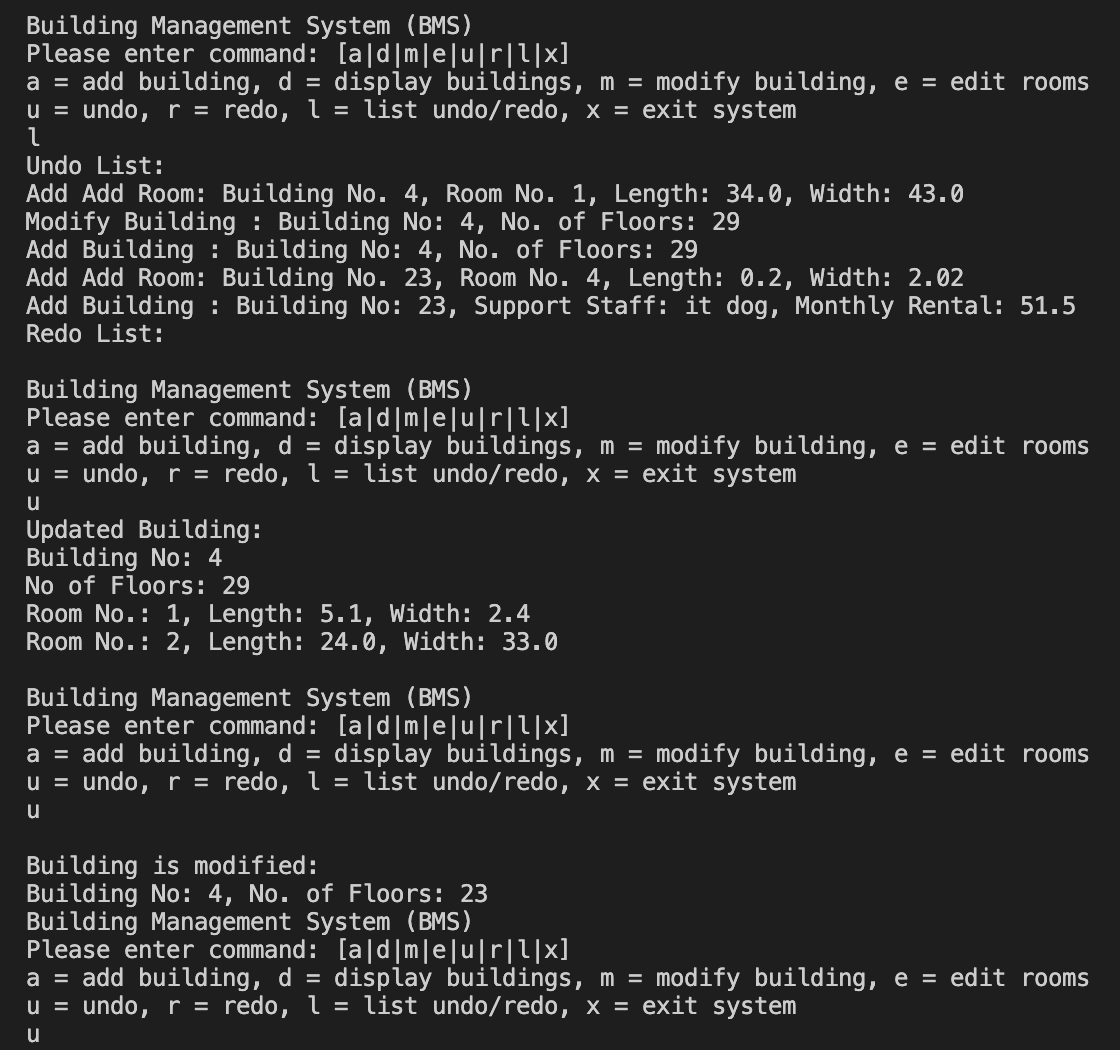




3. Show undo/redo list of previous step and then undo 3 times:

Test data: “l u u u”

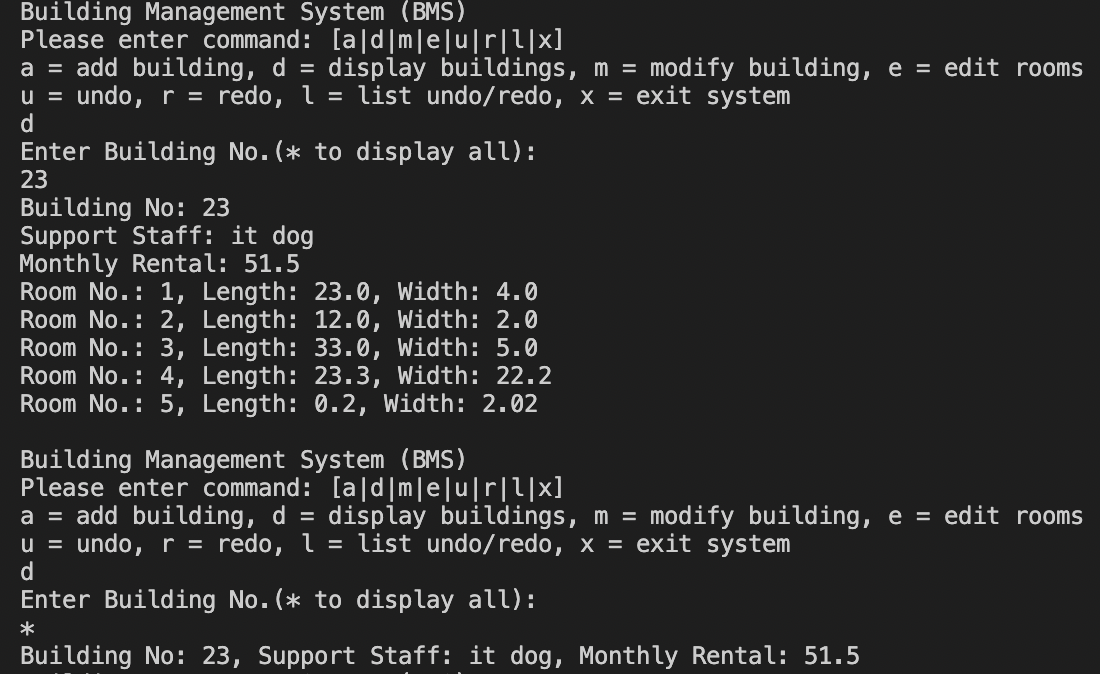
Result:



4. Display building with building number and \*:

Test data: “d 23 d \*”

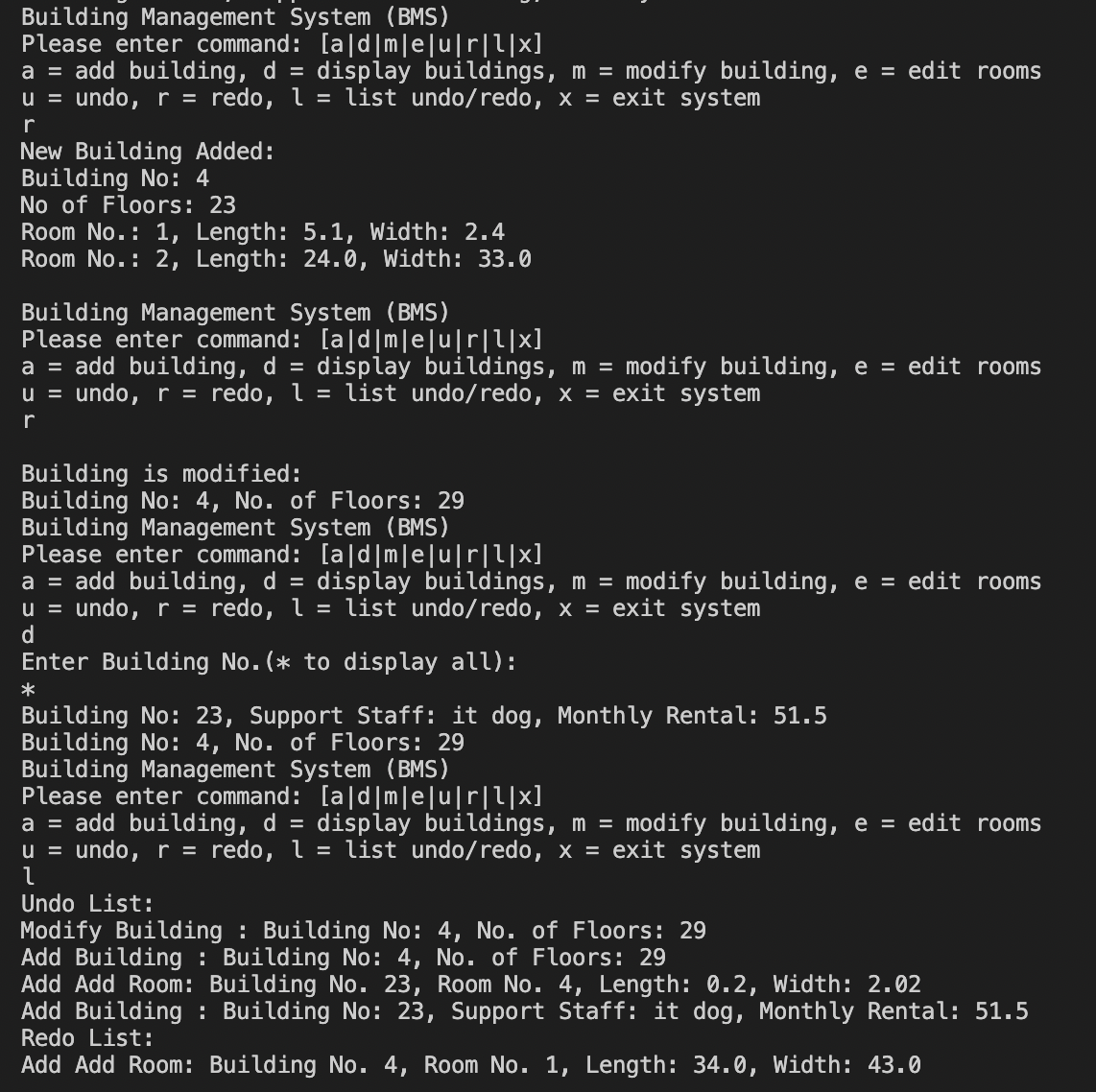
Result:



5. redo 2 times and then show undo/redo list and building:

Test data: “r r d \* l”

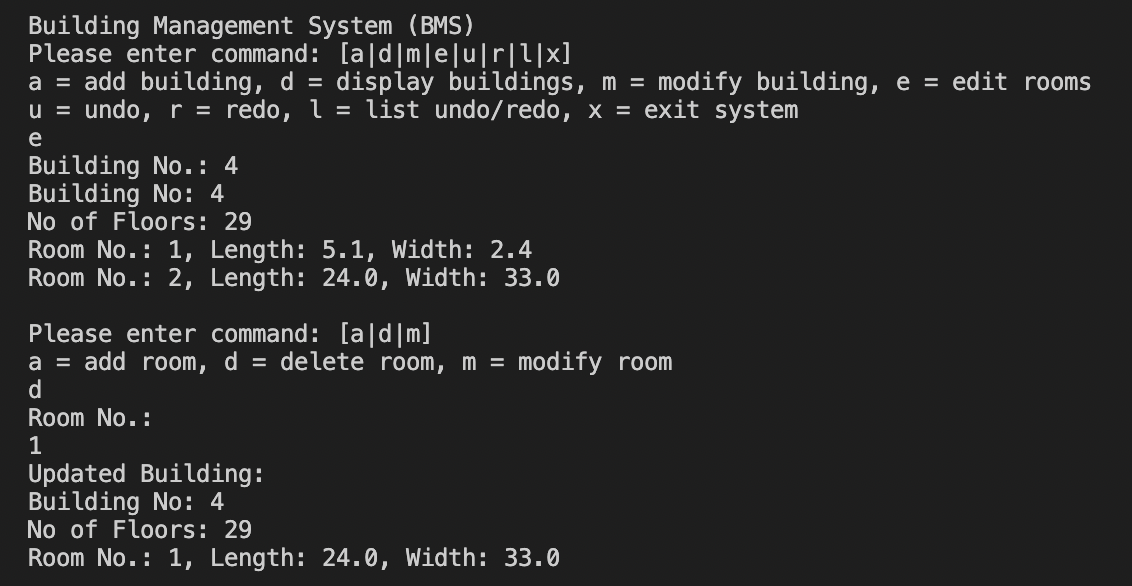
Result:



6. Delete room:

Test data: “e d 1”

Result:



7. Wrong input:

Test data: “3”

Result:

