REPORT

Contents

Introduction

Our company plans to develop a building management system (BMS) for maintaining different kinds of building records, However, the current exit system is not open-close. Thus, as a system analyst of the company I have been required to redesign and develop BMS with design patterns.

Assumptions regarding the problem context

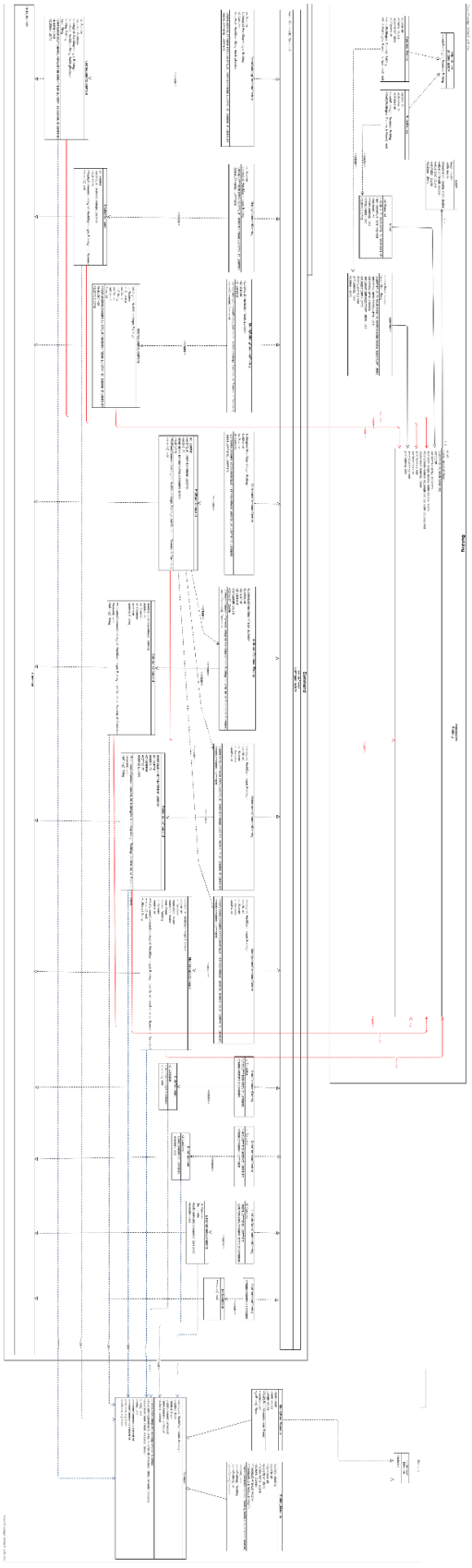
User

1. The user input a not exists command in the main system page.
2. The user input the not match the variable type.
3. The user does undo but the undo list is empty
4. The user does redo but the redo list is empty
5. The user input exists buildingNo while the building already exists.
6. The user input does not exist building id while in modify building
7. The user inputs an empty value when entering.
8. The user inputs negative about the number of rooms.
9. The user input does not exist in room id while in the editing room
10. The user wrong input the details of the building while selecting create building type.

Programmer

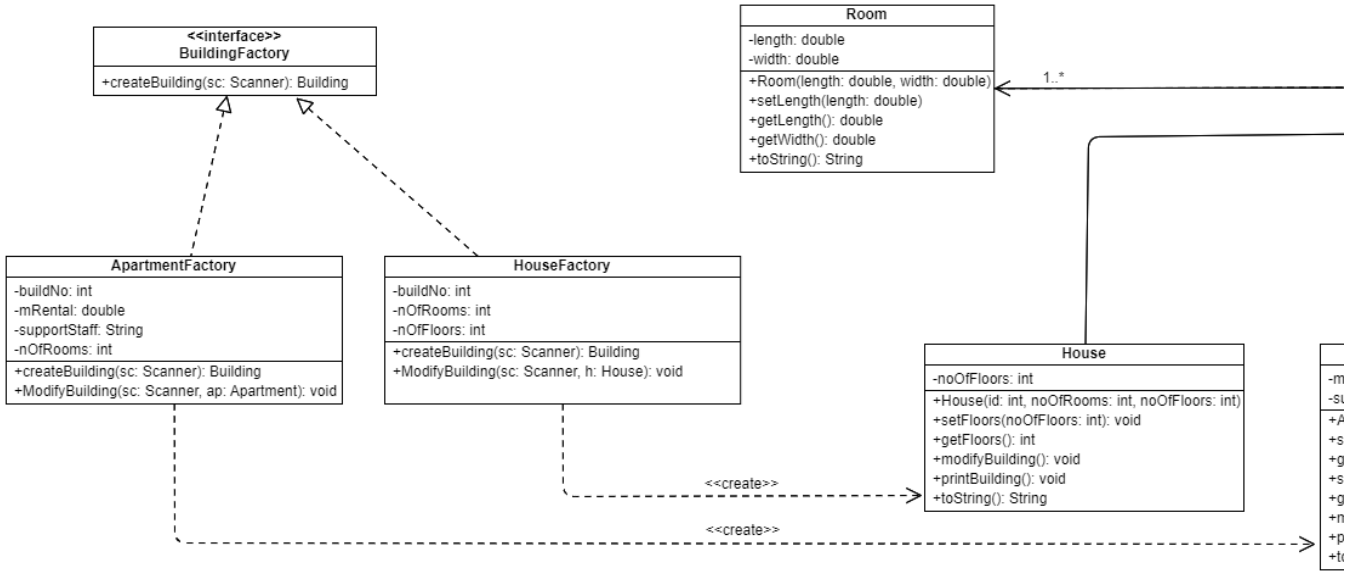
1. The programmer hard to find the bug source and fix it.
2. It needs to change many codes when adding a new function
3. It is difficult for others to take over these codes and hard to maintain
4. The system is easier to problem appear when bigger.
5. It needs to add many codes when adding a new building type.

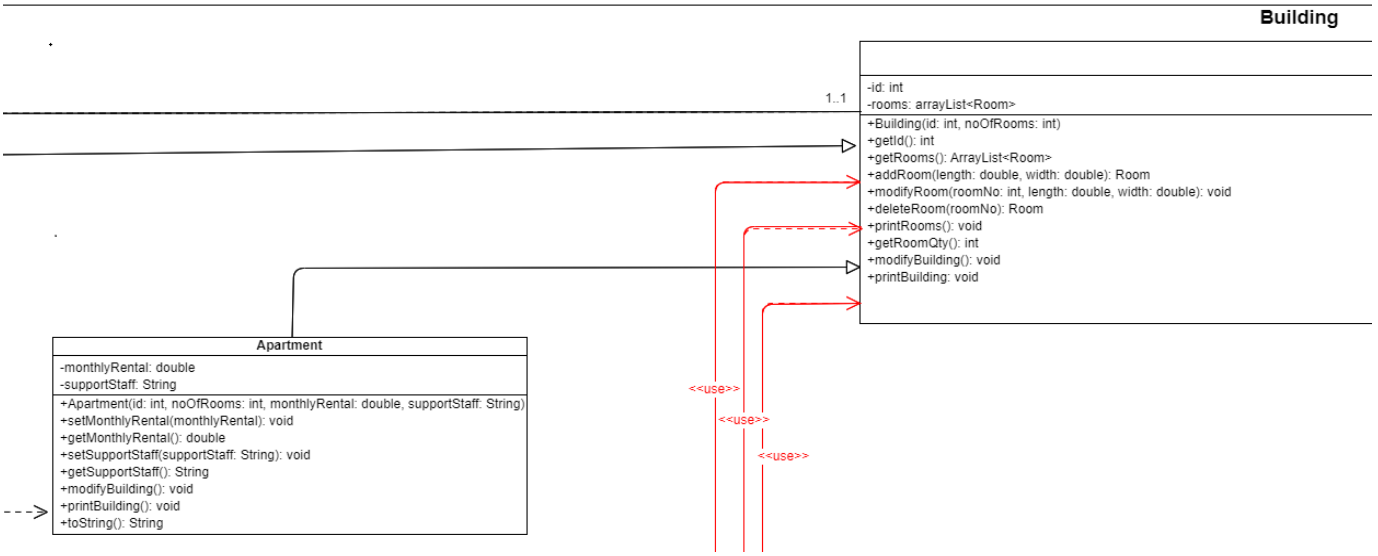
Application design with class diagram



Building package

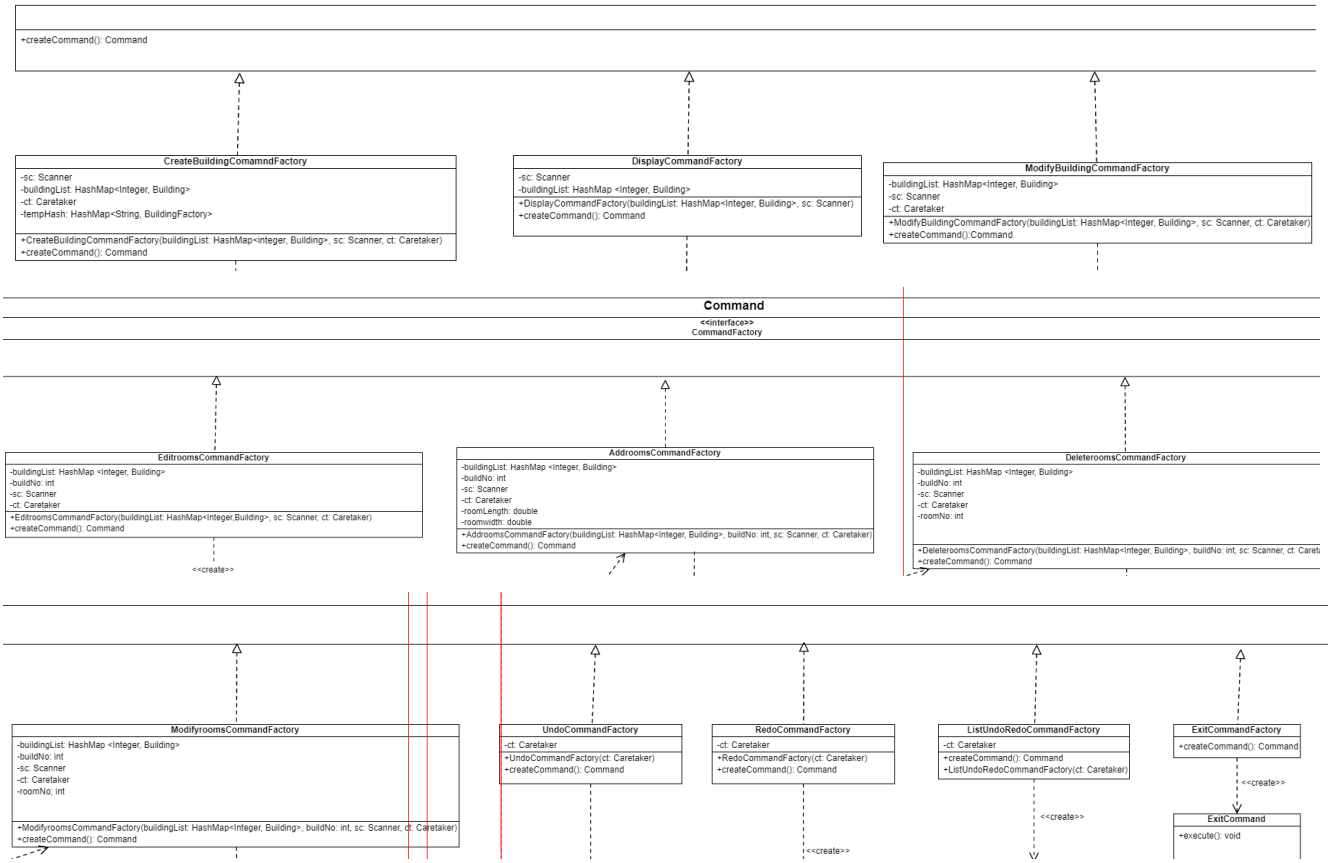
There is the building package, there are 7 classes(BuildingFactory, ApartmentFactory, HouseFactory, Room, House, Apartment, and Building), there are natively provided classes apart from BuildingFactory, ApartmentFactory and HouseFactory, the aim is to create these objects, like create much building.





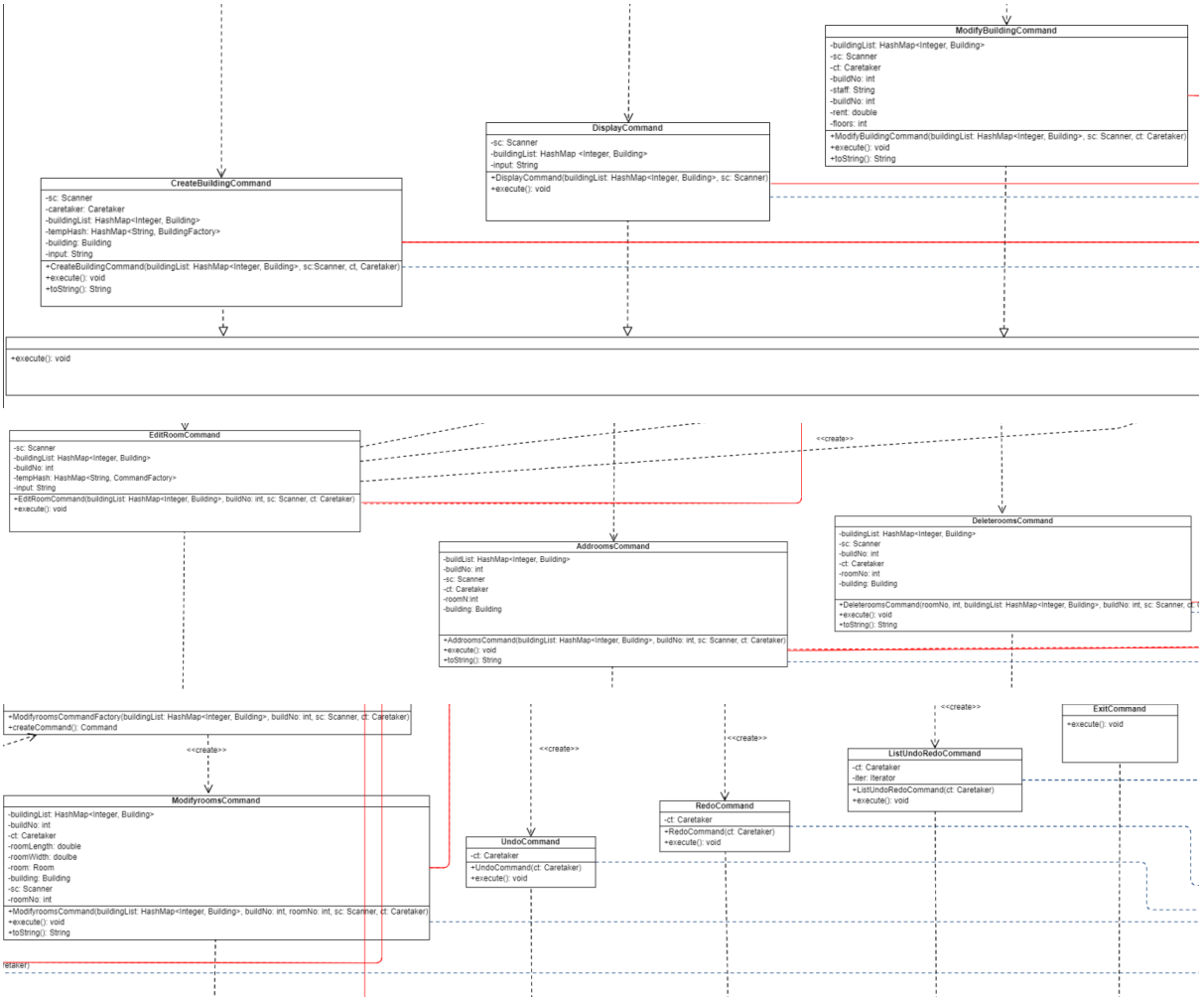
Command package

CommandFactory



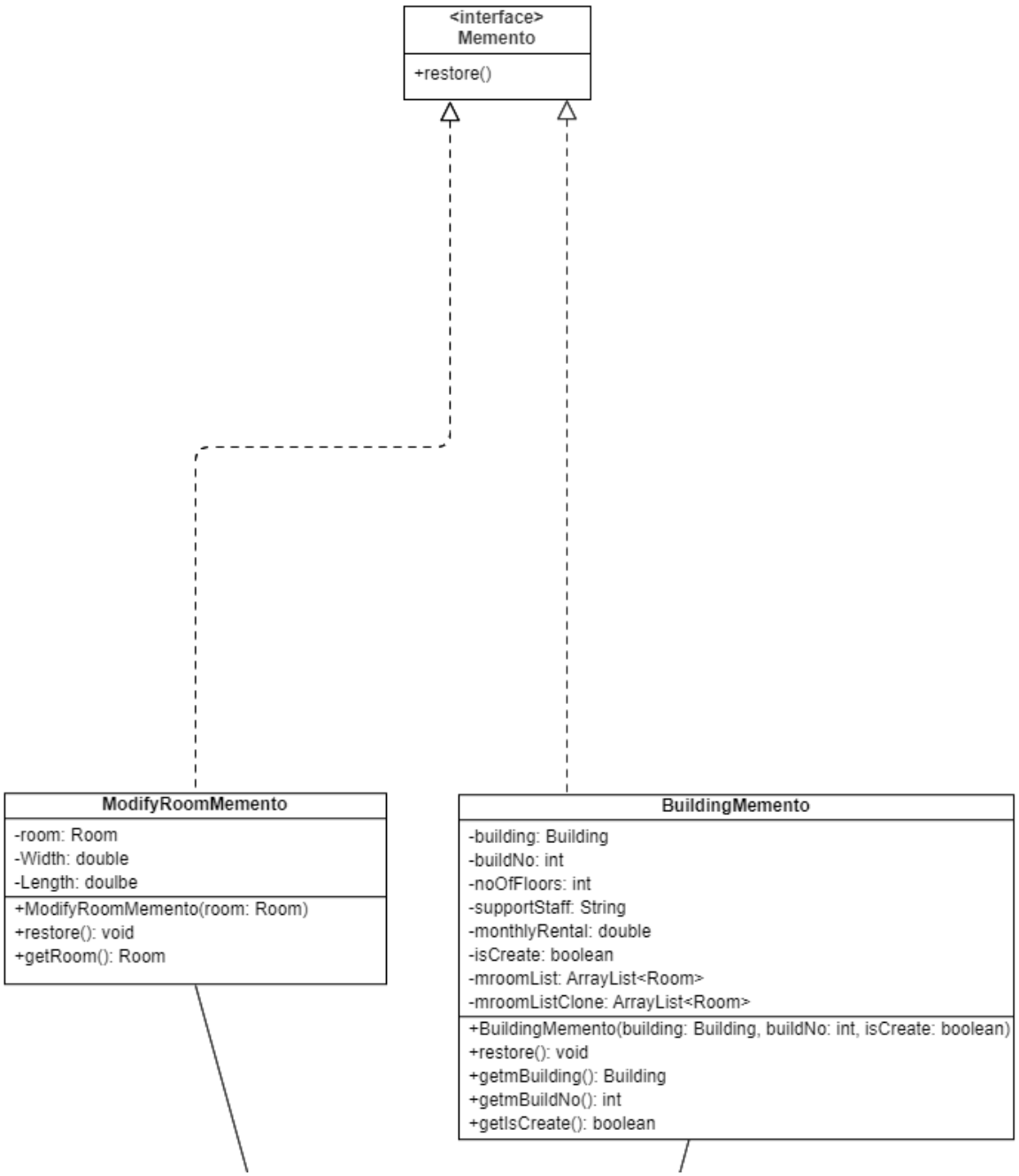
There is the command factory, these classes need to create many commands, such as create, edit, display and delete commands, etc. And these classes implement the Command Factory.

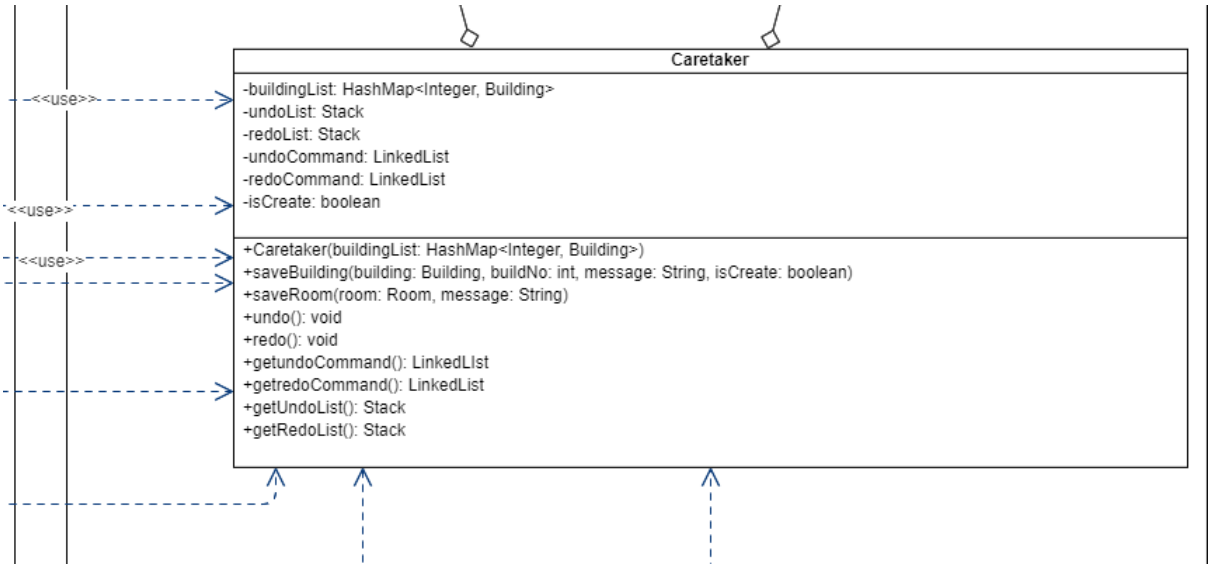
Command



There are many commands, these command classes create by the command factory, and they will execute the command, for example, the exit command will execute the exit action. the system will run what command depending on user input.
Also, some command class use building classes in Building package, such as room command, modifyBuilding command.

Memento Package





There are two memento classes to save state, BuildingMemento is saved add and modifyBuilding, also it saves add and deletes room too. And the ModifyRoomMemento is only a save modify room action.
Also, many command classes will use the caretaker class to save to memento and provide undo and redo functions.

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

Command package

- The new system used a command pattern and factory pattern and declares an interface for all commands.

- The Command interface class provides an abstract execute() method.

- The CommandFactory is an abstract factory that declares an interface for providing the createcommand() method.

- ConcreeteFactory implements the operations to create concrete command objects. For example, createBuildingCommand will execute the command to create a different building using BuildingFactory.

Why use command pattern and command factory In Command package:

The system is more flexible to manage different command if the system needs to add other commands in the future, it just needs to create a new command and command factory, and easy to maintain.

Building package

- The new system used a factory pattern for building packages.

- BuildingFactory is an AbstractFactory that declares an interface for provide createbuilding() method

- ConcreteFactory in this system will implement the operation to create concrete building objects

- Building is an Abstractclass that declares an interface for the type of building object. Therefore Apartments and houses currently exist ConcreteProject that defines a building object to be created by the corresponding ConcreteFactory.

- Building is also an Originator that is used in Memento pattern.

Why use factory pattern in Building package:

It is because the system may not only have two types of building in the future, factory patterns can let the system minimize code duplication, if the system needs to add a new building, just extend a factory class, this is faster than the now method.

Memento package

- The System uses the Memento pattern to provide save state, undo and redo functions.

- The memento is another object that saves the current state of the object, and provides restore() method to return the state of the object, and getIscreate() method to identify the state of an object that has been created.

- There are two mementos (BuildingMemento & ModifyRoomMemento). Also, there has a memento interface to give an abstract method.

- The BuildingMemento saves the state of the building, it includes creating and modify the building, and also it saves add room and deletes room of the building. The ModifyRoomMemento only saves the state when the user modifies the room.

- Caretaker manages the timing of the saving of the state of building, saves the Memento and, if needed, uses Memento to restore the state of the building.

Why use Memento pattern:

Memento pattern can provided a undo & redo function to avoid the user creating a wrong building, the user just need to enter undo command to go back to the previous action.

User Guide

Create Building

First you need to input “a” to start to add building command

```
Building Management System (BMS)
Please enter command: [a|d|m|e|u|r|l|x]
a = add building, d = display buildings, m = modify building, e = edit rooms, u = undo, r = redo, l = list undo/redo, x = exit system
```

The System will ask you for Building Type, you can type “a” or “h” to create an Apartment or house.

```
Enter Building Type (a=Apartment/h=House):
```

Don’t worry to input wrong type

```
Enter Building Type (a=Apartment/h=House):
c
Wrong input
```

When you enter “a” to create apartment, the system will ask you building no, monthly rental, and support staff, just follow the system text to input.
After that, the system will ask you the number of rooms, it will ask you “n” times length and width when you input “n” number of rooms.
Finally, the building is created and the system will print the building detail to show you.

```
Enter Building Type (a=Apartment/h=House):
a
Building No.: 1001
Monthly Rental: 21000
Support Staff: Alan Po
Number of rooms: 2
Room No. 1 :
Length: 15
Width: 20
Room No. 2 :
Length: 10
Width: 30
New Building Added:
Building No: 1001
Support Staff: Alan Po
Monthly Rental: 21000.0
Room No.: 1, Length: 15.0, Width: 20.0
Room No.: 2, Length: 10.0, Width: 30.0
```

When you enter “h” to create house, system will ask you about Building no and the number of floors, and the create room method is the same as creating an apartment.

```
Enter Building Type (a=Apartment/h=House):
h
Building No.: 1002
No. of Floors: 2
Number of rooms: 1
Room No. 1 :
Length: 1
Width: 1
New Building Added:
Building No: 1002
No of Floors: 2
Room No.: 1, Length: 1.0, Width: 1.0
```

Display Buildings

First you need to input “d” in the interface.

```
Building Management System (BMS)
Please enter command: [a|d|m|e|u|r|l|x]
a = add building, d = display buildings, m = modify building, e = edit rooms, u = undo, r = redo, l = list undo/redo, x = exit system
d
Enter Building No. (* to display all):
```

The system will ask you building no, you can enter a number or you can enter * to display all building. but display all will not show room detail.
It is display an error when the building not existing.

```
Enter Building No. (* to display all):
1
Your input is wrong, please try again!
```

```
Enter Building No. (* to display all):
1001
Building No: 1001
Support Staff: Alan Po
Monthly Rental: 21000.0
Room No.: 1, Length: 15.0, Width: 20.0
Room No.: 2, Length: 10.0, Width: 30.0
```

```
Enter Building No. (* to display all):
*
Building No.: 1001, Support Staff: Alan Po, Monthly Rental: 21000.0
Building No.: 1002, No. of Floors: 2
```

Modify Building

First you need to input “m” in the interface.

```
Building Management System (BMS)
Please enter command: [a|d|m|e|u|r|l|x]
a = add building, d = display buildings, m = modify building, e = edit rooms, u = undo, r = redo, l = list undo/redo, x = exit system
m
Building No.:
```

if the building No is a apartment, it will ask you the new value of monthly rental and support staff.

```
m
Building No.: 1001
Building No.: 1001, Support Staff: Alan Po, Monthly Rental: 21000.0
Modify Monthly Rental.: 1000
Modify Support Staff.: Tom
Building is modified:
Building No.: 1001, Support Staff: Tom, Monthly Rental: 1000.0
```

if the building No is a house, it will ask you the new value of number of floors.

```
m
Building No.: 1002
Building No.: 1002, No. of Floors: 2
No. of Floors: 3
Building is modified:
Building No.: 1002, No. of Floors: 3
```

Edit rooms

First you need to input “e” in the interface

```
Building Management System (BMS)
Please enter command: [a|d|m|e|u|r|l|x]
a = add building, d = display buildings, m = modify building, e = edit rooms, u = undo, r = redo, l = list undo/redo, x = exit system
e
Building No.:
```

The System will ask for your building no first, and the system will show add room, modify room and delete room commands.

```
Building Management System (BMS)
Please enter command: [a|d|m|e|u|r|l|x]
a = add building, d = display buildings, m = modify building, e = edit rooms, u = undo, r = redo, l = list undo/redo, x = exit system
e
Building No.: 1001
Building No: 1001
Support Staff: Tom
Monthly Rental: 1000.0
Room No.: 1, Length: 15.0, Width: 20.0
Room No.: 2, Length: 10.0, Width: 30.0

Please enter command: [a|d|m]
a = add room, d = delete room, m = modify room
```

you can enter “a” to add room, “d” to delete room and “m” to modify room

Add room

The system will ask you for the Length and Width to add a new room, and the system will show new detail when you add a room successfully.

```
Please enter command: [a|d|m]
a = add room, d = delete room, m = modify room
a
Length: 30
Width: 40
Updated Building:
Building No: 1001
Support Staff: Tom
Monthly Rental: 1000.0
Room No.: 1, Length: 15.0, Width: 20.0
Room No.: 2, Length: 10.0, Width: 30.0
Room No.: 3, Length: 30.0, Width: 40.0
```

Delete room

The system will ask you for a room number, and you can delete a existing room successfully

```
Please enter command: [a|d|m]
a = add room, d = delete room, m = modify room
d
Room No.:
3
Updated Building:
Building No: 1001
Support Staff: Tom
Monthly Rental: 1000.0
Room No.: 1, Length: 15.0, Width: 20.0
Room No.: 2, Length: 10.0, Width: 30.0
```

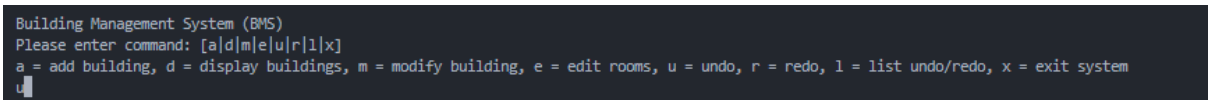
Modify room

The system will ask you for a room number to modify, and you can input a new length and new width to modify the current room.

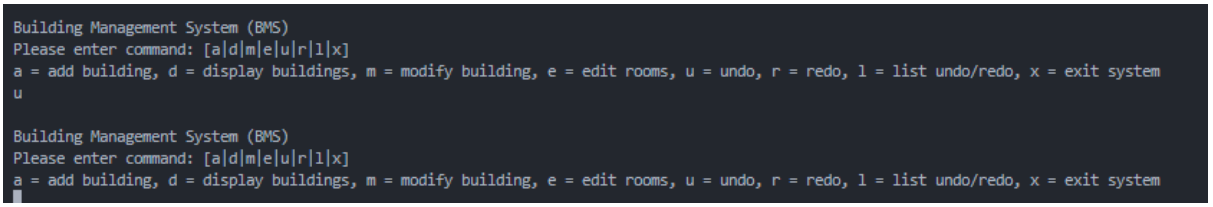
```
Please enter command: [a|d|m]
a = add room, d = delete room, m = modify room
m
Room No.: 1
Length: 20
Width: 50
Updated Building:
Building No: 1001
Support Staff: Tommer
Monthly Rental: 2000.0
Room No.: 1, Length: 20.0, Width: 50.0
Room No.: 2, Length: 10.0, Width: 30.0
```

Undo

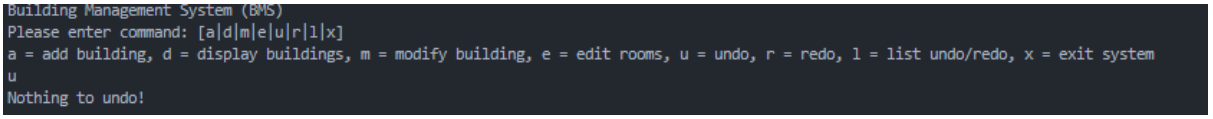
First you need to input “u” in interface



It is successful if there is no any message

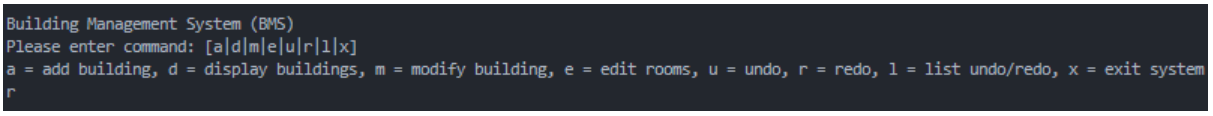


it will show “nothing undo” when you have never done add building, edit rooms and modify building.

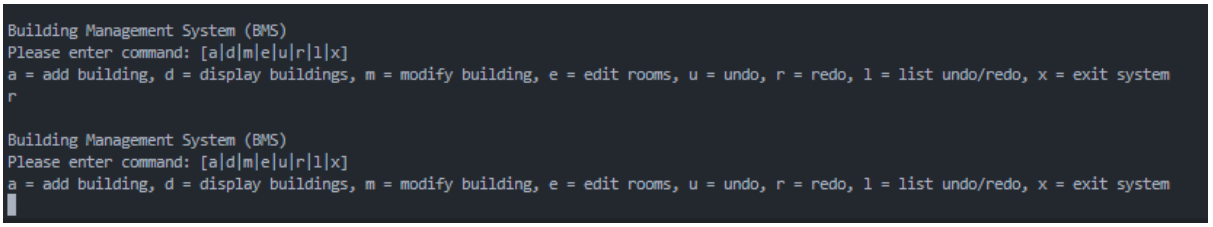


Redo

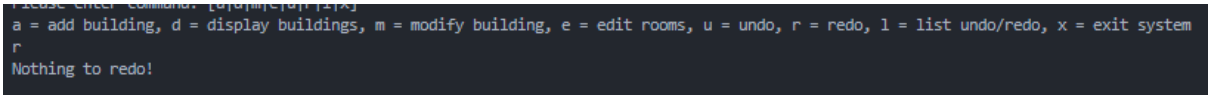
First you need to input “r” in interface



It is successful if there is no any message



it will show “nothing redo” when you have never done undo command



List undo/redo

First you need to input “l” in interface

```
Building Management System (BMS)
Please enter command: [a|d|m|e|u|r|l|x]
a = add building, d = display buildings, m = modify building, e = edit rooms, u = undo, r = redo, l = list undo/redo, x = exit system
l
```

It will show all of undo and redo, you can see there are 8 things wait for undo in this image

```
Undo List :
Modify Room : Building No. 1001 ,Room No. 1, Length : 20.0, Width : 50.0
Modify Building: Building No.: 1001, Support Staff: Tommer, Monthly Rental:2000.0
Delete Room : Building No. 1001 ,Room No. 3, Length : 30.0, Width : 40.0
Add Room : Building No. 1001 ,Room No. 3, Length : 30.0, Width : 40.0
Modify Building: Building No.: 1002, No. of Floors: 3
Modify Building: Building No.: 1001, Support Staff: Tom, Monthly Rental:1000.0
Add Building: Building No.: 1002, No. of Floors: 2
Add Building: Building No.: 1001, Support Staff: Alan Po, Monthly Rental: 21000.0

Redo List :
Nothing in Redo List
```

It will show “Nothing in Undo/Redo List” when no thing to undo or redo

```
Undo List :
Nothing in Undo List

Redo List :
Nothing in Redo List
```

exit system

First you need to input “x” in interface

```
Building Management System (BMS)
Please enter command: [a|d|m|e|u|r|l|x]
a = add building, d = display buildings, m = modify building, e = edit rooms, u = undo, r = redo, l = list undo/redo, x = exit system
x
```

You can see the interface has been closed when exit system

```
Building Management System (BMS)
Please enter command: [a|d|m|e|u|r|l|x]
a = add building, d = display buildings, m = modify building, e = edit rooms, u = undo, r = redo, l = list undo/redo, x = exit system
x
PS D:\Desktop\210020835-ITP4507-assignment>
```

Test Plan and Test Cases

––general test –

a
dddd ←-Wrong input
a
1000
5000
Tommer
1
10
20
r ←-should be empty
d
buildno ←-Wrong input
e
1000
c ←-Wrong input
j ←-Wrong input
a
30
40
a
h
1001
2
2
5
6
7
8

—-display building test—

d
100 ←-Wrong input
d
1000
d
*

—-modify building test—

m
1000
50000

Tom
m
1001
3
—-edit rooms test—
e
1000
a
50
50
e
1000
m
3
60
60
e
1000
d
3
—-undo/redo test—
l
u
u
u
l
d
1000
r
r
r
l
d
1000
x

Expected Output of normal flow

general test

```
Building Management System (BMS)
Please enter command: [a|d|m|e|u|r|l|x]
a = add building, d = display buildings, m = modify building, e = edit rooms, u = undo, r = redo, l = list undo/redo, x = exit system
a
Enter Building Type (a=Apartment/h=House):
dddd
Wrong input
Enter Building Type (a=Apartment/h=House):
a
Building No.: 1000
Monthly Rental: 5000
Support Staff: Tommer
Number of rooms: 1
Room No. 1 :
Length: 10
Width: 20
New Building Added:
Building No: 1000
Support Staff: Tommer
Monthly Rental: 5000.0
Room No.: 1, Length: 10.0, Width: 20.0

Building Management System (BMS)
Please enter command: [a|d|m|e|u|r|l|x]
a = add building, d = display buildings, m = modify building, e = edit rooms, u = undo, r = redo, l = list undo/redo, x = exit system
r
Nothing to redo!

Building Management System (BMS)
Please enter command: [a|d|m|e|u|r|l|x]
a = add building, d = display buildings, m = modify building, e = edit rooms, u = undo, r = redo, l = list undo/redo, x = exit system
d
Enter Building No. (* to display all):
buildno
Your input is wrong, please try again!

Building Management System (BMS)
Please enter command: [a|d|m|e|u|r|l|x]
a = add building, d = display buildings, m = modify building, e = edit rooms, u = undo, r = redo, l = list undo/redo, x = exit system
e
Building No.: 1000
Building No: 1000
Support Staff: Tommer
Monthly Rental: 5000.0
Room No.: 1, Length: 10.0, Width: 20.0

Please enter command: [a|d|m]
a = add room, d = delete room, m = modify room
```

```
Please enter command: [a|d|m]
a = add room, d = delete room, m = modify room
c
Wrong input
Please enter command: [a|d|m]
a = add room, d = delete room, m = modify room
j
Wrong input
Please enter command: [a|d|m]
a = add room, d = delete room, m = modify room
a
Length: 30
Width: 40
Updated Building:
Building No: 1000
Support Staff: Tommer
Monthly Rental: 5000.0
Room No.: 1, Length: 10.0, Width: 20.0
Room No.: 2, Length: 30.0, Width: 40.0

Building Management System (BMS)
Please enter command: [a|d|m|e|u|r|l|x]
a = add building, d = display buildings, m = modify building, e = edit rooms, u = undo, r = redo, l = list undo/redo, x = exit system
a
Enter Building Type (a=Apartment/h=House):
h
Building No.: 1001
No. of Floors: 2
Number of rooms: 2
Room No. 1 :
Length: 5
Width: 6
Room No. 2 :
Length: 7
Width: 8
New Building Added:
Building No: 1001
No of Floors: 2
Room No.: 1, Length: 5.0, Width: 6.0
Room No.: 2, Length: 7.0, Width: 8.0
```

display building test

```
Building Management System (BMS)
Please enter command: [a|d|m|e|u|r|l|x]
a = add building, d = display buildings, m = modify building, e = edit rooms, u = undo, r = redo, l = list undo/redo, x = exit system
d
Enter Building No. (* to display all):
100
Your input is wrong, please try again!

Building Management System (BMS)
Please enter command: [a|d|m|e|u|r|l|x]
a = add building, d = display buildings, m = modify building, e = edit rooms, u = undo, r = redo, l = list undo/redo, x = exit system
d
Enter Building No. (* to display all):
1000
Building No: 1000
Support Staff: Tommer
Monthly Rental: 5000.0
Room No.: 1, Length: 10.0, Width: 20.0
Room No.: 2, Length: 30.0, Width: 40.0

Building Management System (BMS)
Please enter command: [a|d|m|e|u|r|l|x]
a = add building, d = display buildings, m = modify building, e = edit rooms, u = undo, r = redo, l = list undo/redo, x = exit system
d
Enter Building No. (* to display all):
*
Building No.: 1000, Support Staff: Tommer, Monthly Rental: 5000.0
Building No.: 1001, No. of Floors: 2
```

modify building test

```
Building Management System (BMS)
Please enter command: [a|d|m|e|u|r|l|x]
a = add building, d = display buildings, m = modify building, e = edit rooms, u = undo, r = redo, l = list undo/redo, x = exit system
m
Building No.: 1000
Building No.: 1000, Support Staff: Tommer, Monthly Rental: 5000.0
Modify Monthly Rental.: 50000
Modify Support Staff.: Tom
Building is modified:
Building No.: 1000, Support Staff: Tom, Monthly Rental: 50000.0

Building Management System (BMS)
Please enter command: [a|d|m|e|u|r|l|x]
a = add building, d = display buildings, m = modify building, e = edit rooms, u = undo, r = redo, l = list undo/redo, x = exit system
m
Building No.: 1001
Building No.: 1001, No. of Floors: 2
No. of Floors: 3
Building is modified:
Building No.: 1001, No. of Floors: 3
```

edit rooms test

```
Building Management System (BMS)
Please enter command: [a|d|m|e|u|r|l|x]
a = add building, d = display buildings, m = modify building, e = edit rooms, u = undo, r = redo, l = list undo/redo, x = exit system
e
Building No.: 1000
Building No: 1000
Support Staff: Tom
Monthly Rental: 50000.0
Room No.: 1, Length: 10.0, Width: 20.0
Room No.: 2, Length: 30.0, Width: 40.0

Please enter command: [a|d|m]
a = add room, d = delete room, m = modify room
a
Length: 50
Width: 50
Updated Building:
Building No: 1000
Support Staff: Tom
Monthly Rental: 50000.0
Room No.: 1, Length: 10.0, Width: 20.0
Room No.: 2, Length: 30.0, Width: 40.0
Room No.: 3, Length: 50.0, Width: 50.0

Building Management System (BMS)
Please enter command: [a|d|m|e|u|r|l|x]
a = add building, d = display buildings, m = modify building, e = edit rooms, u = undo, r = redo, l = list undo/redo, x = exit system
e
Building No.: 1000
Building No: 1000
Support Staff: Tom
Monthly Rental: 50000.0
Room No.: 1, Length: 10.0, Width: 20.0
Room No.: 2, Length: 30.0, Width: 40.0
Room No.: 3, Length: 50.0, Width: 50.0

Please enter command: [a|d|m]
a = add room, d = delete room, m = modify room
m
Room No.: 3
Length: 60
Width: 60
Updated Building:
Building No: 1000
Support Staff: Tom
Monthly Rental: 50000.0
Room No.: 1, Length: 10.0, Width: 20.0
Room No.: 2, Length: 30.0, Width: 40.0
Room No.: 3, Length: 60.0, Width: 60.0
```

```
Building Management System (BMS)
Please enter command: [a|d|m|e|u|r|l|x]
a = add building, d = display buildings, m = modify building, e = edit rooms, u = undo, r = redo, l = list undo/redo, x = exit system
e
Building No.: 1000
Building No: 1000
Support Staff: Tom
Monthly Rental: 50000.0
Room No.: 1, Length: 10.0, Width: 20.0
Room No.: 2, Length: 30.0, Width: 40.0
Room No.: 3, Length: 60.0, Width: 60.0

Please enter command: [a|d|m]
a = add room, d = delete room, m = modify room
d
Room No.:
3
Updated Building:
Building No: 1000
Support Staff: Tom
Monthly Rental: 50000.0
Room No.: 1, Length: 10.0, Width: 20.0
Room No.: 2, Length: 30.0, Width: 40.0

Building Management System (BMS)
Please enter command: [a|d|m|e|u|r|l|x]
a = add building, d = display buildings, m = modify building, e = edit rooms, u = undo, r = redo, l = list undo/redo, x = exit system
l

Undo List :
Delete Room : Building No. 1000 ,Room No. 3, Length : 60.0, Width : 60.0
Modify Room : Building No. 1000 ,Room No. 3, Length : 60.0, Width : 60.0
Add Room : Building No. 1000 ,Room No. 3, Length : 50.0, Width : 50.0
Modify Building: Building No.: 1001, No. of Floors: 3
Modify Building: Building No.: 1000, Support Staff: Tom, Monthly Rental:50000.0
Add Building: Building No.: 1001, No. of Floors: 2
Add Room : Building No. 1000 ,Room No. 2, Length : 30.0, Width : 40.0
Add Building: Building No.: 1000, Support Staff: Tommer, Monthly Rental: 5000.0

Redo List :
Nothing in Redo List
```

undo/redo test

```
Building Management System (BMS)
Please enter command: [a|d|m|e|u|r|l|x]
a = add building, d = display buildings, m = modify building, e = edit rooms, u = undo, r = redo, l = list undo/redo, x = exit system
u

Building Management System (BMS)
Please enter command: [a|d|m|e|u|r|l|x]
a = add building, d = display buildings, m = modify building, e = edit rooms, u = undo, r = redo, l = list undo/redo, x = exit system
u

Building Management System (BMS)
Please enter command: [a|d|m|e|u|r|l|x]
a = add building, d = display buildings, m = modify building, e = edit rooms, u = undo, r = redo, l = list undo/redo, x = exit system
u

Building Management System (BMS)
Please enter command: [a|d|m|e|u|r|l|x]
a = add building, d = display buildings, m = modify building, e = edit rooms, u = undo, r = redo, l = list undo/redo, x = exit system
l

Undo List :
Modify Building: Building No.: 1001, No. of Floors: 3
Modify Building: Building No.: 1000, Support Staff: Tom, Monthly Rental:50000.0
Add Building: Building No.: 1001, No. of Floors: 2
Add Room : Building No. 1000 ,Room No. 2, Length : 30.0, Width : 40.0
Add Building: Building No.: 1000, Support Staff: Tommer, Monthly Rental: 5000.0

Redo List :
Add Room : Building No. 1000 ,Room No. 3, Length : 50.0, Width : 50.0
Modify Room : Building No. 1000 ,Room No. 3, Length : 60.0, Width : 60.0
Delete Room : Building No. 1000 ,Room No. 3, Length : 60.0, Width : 60.0

Building Management System (BMS)
Please enter command: [a|d|m|e|u|r|l|x]
a = add building, d = display buildings, m = modify building, e = edit rooms, u = undo, r = redo, l = list undo/redo, x = exit system
d
Enter Building No. (* to display all):
1000
Building No: 1000
Support Staff: Tom
Monthly Rental: 50000.0
Room No.: 1, Length: 10.0, Width: 20.0
Room No.: 2, Length: 30.0, Width: 40.0

Building Management System (BMS)
Please enter command: [a|d|m|e|u|r|l|x]
a = add building, d = display buildings, m = modify building, e = edit rooms, u = undo, r = redo, l = list undo/redo, x = exit system
r

Building Management System (BMS)
Please enter command: [a|d|m|e|u|r|l|x]
a = add building, d = display buildings, m = modify building, e = edit rooms, u = undo, r = redo, l = list undo/redo, x = exit system
r

Building Management System (BMS)
Please enter command: [a|d|m|e|u|r|l|x]
a = add building, d = display buildings, m = modify building, e = edit rooms, u = undo, r = redo, l = list undo/redo, x = exit system
r

Building Management System (BMS)
Please enter command: [a|d|m|e|u|r|l|x]
a = add building, d = display buildings, m = modify building, e = edit rooms, u = undo, r = redo, l = list undo/redo, x = exit system
l
```

```
Undo List :
Delete Room : Building No. 1000 ,Room No. 3, Length : 60.0, Width : 60.0
Modify Room : Building No. 1000 ,Room No. 3, Length : 60.0, Width : 60.0
Add Room : Building No. 1000 ,Room No. 3, Length : 50.0, Width : 50.0
Modify Building: Building No.: 1001, No. of Floors: 3
Modify Building: Building No.: 1000, Support Staff: Tom, Monthly Rental:50000.0
Add Building: Building No.: 1001, No. of Floors: 2
Add Room : Building No. 1000 ,Room No. 2, Length : 30.0, Width : 40.0
Add Building: Building No.: 1000, Support Staff: Tommer, Monthly Rental: 5000.0

Redo List :
Nothing in Redo List

Building Management System (BMS)
Please enter command: [a|d|m|e|u|r|l|x]
a = add building, d = display buildings, m = modify building, e = edit rooms, u = undo, r = redo, l = list undo/redo, x = exit system
d
Enter Building No. (* to display all):
1000
Building No: 1000
Support Staff: Tom
Monthly Rental: 50000.0
Room No.: 1, Length: 10.0, Width: 20.0
Room No.: 2, Length: 30.0, Width: 40.0

Building Management System (BMS)
Please enter command: [a|d|m|e|u|r|l|x]
a = add building, d = display buildings, m = modify building, e = edit rooms, u = undo, r = redo, l = list undo/redo, x = exit system
x
PS D:\Desktop\210020835-ITP4507-assignment>
```

Well documented Source Code

main.java

```
import java.util.*;

import Building.*;
import Command.*;
import Memento.*;

public class main {

    public static Scanner sc = new Scanner(System.in);

    public static void main(String[] args) throws ClassNotFoundException,
InstantiationException, IllegalAccessException {

        //a hashmap to store all building, the key is building number
        HashMap<Integer, Building> buildingList = new HashMap<>();

        Caretaker ct = new Caretaker(buildingList);
        HashMap<String, CommandFactory> f = new HashMap<>();
        f.put("a", new CreateBuildingCommandFactory(buildingList, sc, ct));
        f.put("d", new DisplayCommandFactory(buildingList, sc));
        f.put("m", new ModifyBuildingCommandFactory(buildingList, sc, ct));
        f.put("e", new EditRoomCommandFactory(buildingList, sc, ct));
        f.put("u", new UndoCommandFactory(ct));
        f.put("r", new RedoCommandFactory(ct));
        f.put("l", new ListUndoRedoCommandFactory(ct));
        f.put("x", new ExitCommandFactory());

        Command com;
        String input;
        while (true) {
            try{

                System.out.println(
                "\nBuilding Management System (BMS) \n"+
                "Please enter command: [a|d|m|e|u|r|l|x] \n"+
                "a = add building, d = display buildings, m = modify building, e = edit
rooms, u = undo, r = redo, l = list undo/redo, x = exit system"
                );

                input = sc.nextLine();
                com = f.get(input).createCommand();
                com.execute();

            } catch(Exception e){
                System.out.println("Your input is wrong, please try again!");
            }


        }
    }
}
```

Building package

apartment.java

```
/*
Student:   Ching Chun Hung 210020835 2B
Last Edit  13/11/2022
*/
package Building;
public class Apartment extends Building{
    private double monthlyRental;
    private String supportStaff;

    public Apartment(int id, int noOfRooms, double monthlyRental, String supportStaff){
        super(id, noOfRooms);
        this.monthlyRental = monthlyRental;
        this.supportStaff = supportStaff;
    }
    public void setMonthlyRental(double monthlyRental){
        this.monthlyRental = monthlyRental;
    }
    public double getMonthlyRental(){
        return this.monthlyRental;
    }
    public void setSupportStaff(String supportStaff){
        this.supportStaff = supportStaff;
    }
    public String getSupportStaff(){
        return this.supportStaff;
    }
    public void modifyBuilding(){
        //the code was move to factory
    }
    public void printBuilding(){    //print with room and \n (long)
        System.out.println("Building No: "+super.getId());
        System.out.println("Support Staff: "+supportStaff);
        System.out.println("Monthly Rental: "+monthlyRental);
        super.printRooms();

    }
    public String toString(){       //print without room (short)
        return "Building No.: "+super.getId()+", Support Staff: "+supportStaff+", Monthly
Rental: "+monthlyRental;
    }

}
```

ApartmentFactory.java

```
package Building;

import java.util.Scanner;

public class ApartmentFactory implements BuildingFactory {

    private int buildNo;
    private double mRental;
    private String supportStaff;
    private int nOfRooms;

    @Override
    public Building createBuilding(Scanner sc) {

        System.out.print("Building No.: ");
        buildNo = sc.nextInt();

        System.out.print("Monthly Rental: ");
        mRental = sc.nextDouble();

        System.out.print("Support Staff: ");
        sc.nextLine();
        supportStaff = sc.nextLine();

        System.out.print("Number of rooms: ");
        nOfRooms = sc.nextInt();

        Building apartmentBuilding = new Apartment(buildNo, nOfRooms, mRental,
supportStaff);

        //Since you need to addRoom immediately after creating a building, and the Building
class has this one method

        for (int i = 1; i <= nOfRooms; i++) {

            System.out.println("Room No. " + i + " :");

            System.out.print("Length: ");
            double Length = sc.nextDouble();

            System.out.print("Width: ");
            double Width = sc.nextDouble();

            apartmentBuilding.addRoom(Length, Width);
        }

        System.out.println("New Building Added:");
        apartmentBuilding.printBuilding();
        sc.nextLine();

        return apartmentBuilding;

    }

    //this factory provided modify building too (override the origin method)
```

```
    public void ModifyBuilding(Scanner sc, Apartment ap) {

        System.out.print("Modify Monthly Rental.: ");
        ap.setMonthlyRental(sc.nextDouble());

        System.out.print("Modify Support Staff.: ");
        sc.nextLine();
        ap.setSupportStaff(sc.nextLine());

        System.out.println("Building is modified: ");
        System.out.println(toString());

    }
}
```

Building.java

```
package Building;
import java.util.*;

public abstract class Building{
    private int id;
    private ArrayList<Room> rooms;

    public Building(int id, int noOfRooms){
        this.id = id;
        this.rooms = new ArrayList<Room>(noOfRooms);
    }

    public int getId(){
        return this.id;
    }

    public ArrayList<Room> getRooms(){
        return this.rooms;
    }

    public Room addRoom(double length, double width){
        Room roomId = new Room(length, width);
        rooms.add(roomId);
        return roomId;
    }

    //since the room index is 1 less than userinput
    public void modifyRoom(int roomNo, double length, double width){
        rooms.get(roomNo-1).setLength(length);
        rooms.get(roomNo-1).setWidth(width);

    }

    public Room deleteRoom(int roomNo){
        return rooms.remove(roomNo-1);
    }
```

```
    //loop the arraylist and print
    public void printRooms(){

        for(int i = 1; i <= rooms.size(); i++){
            Room room = rooms.get(i-1);
            System.out.println("Room No.: " + i + room);

        }

    }

    public int getRoomQty(){
        return rooms.size();
    }

    public abstract void modifyBuilding();
    public abstract void printBuilding();

}
```

BuildingFactory.java

```
package Building;
import java.util.*;

public interface BuildingFactory {
    public abstract Building createBuilding(Scanner sc);

}
```

House.java

```
package Building;
//import java.util.scanner;
public class House extends Building{
    private int noOfFloors;

    public House(int id, int noOfRooms, int noOfFloors){
        super(id, noOfRooms);
        this.noOfFloors = noOfFloors;
    }
    public void setFloors(int noOfFloors){
        this.noOfFloors = noOfFloors;
    }
    public int getFloors(){
        return noOfFloors;
    }
    public void modifyBuilding(){
        //the code was move to factory
    }
    public void printBuilding(){
        System.out.println("Building No: "+super.getId());
        System.out.println("No of Floors: "+noOfFloors);
```

```
        super.printRooms();

    }
    public String toString(){
        return "Building No.: "+super.getId()+", No. of Floors: "+noOfFloors;
    }

}
```

HouseFactory.java

```
package Building;
import java.util.*;

public class HouseFactory implements BuildingFactory {

    private int buildNo;
    private int nOfRooms;
    private int nOfFloors;

    public Building createBuilding(Scanner sc) {
        System.out.print("Building No.: ");
        buildNo = sc.nextInt();

        System.out.print("No. of Floors: ");
        nOfFloors = sc.nextInt();

        System.out.print("Number of rooms: ");
        nOfRooms = sc.nextInt();

        Building houseBuilding = new House(buildNo, nOfRooms, nOfFloors);

        for (int i = 1; i <= nOfRooms; i++) {
            System.out.println("Room No. " + i + " :");

            System.out.print("Length: ");
            double roomLength = sc.nextDouble();

            System.out.print("Width: ");
            double roomWidth = sc.nextDouble();

            houseBuilding.addRoom(roomLength, roomWidth);
        }

        System.out.println("New Building Added:");

        houseBuilding.printBuilding();
        sc.nextLine();

        return houseBuilding;
    }

    //this factory provided modify building too (override the origin method)

    public void ModifyBuilding(Scanner sc, House h) {
```

```
        System.out.print("No. of Floors: ");
        h.setFloors(sc.nextInt());

        System.out.println("Building is modified: ");
        System.out.println(toString());

        sc.nextLine();

    }
}
```

Room.java

```
package Building;
public class Room{
    private double length;
    private double width;

    public Room(double length, double width){
        this.length = length;
        this.width = width;
    }
    public void setLength(double length){
        this.length = length;
    }
    public void setWidth(double width){
        this.width = width;
    }
    public double getLength(){
        return length;
    }
    public double getWidth(){
        return width;
    }
    public String toString(){
        return ", Length: " + getLength() + ", Width: " + getWidth();
    }

}
```

Command Package

AddroomsCommand

```
package Command;

import Building.*;
import Memento.*;
import java.util.*;

public class AddroomsCommand implements Command {
    private double roomLength;
    private double roomWidth;
    private HashMap<Integer, Building> buildingList;
    private int buildNo;
    private Scanner sc;
    private Caretaker ct;
    private Building building;

    //local
    private int roomNo;

    public AddroomsCommand(double roomLength, double roomWidth, HashMap<Integer, Building>
buildingList, int buildNo, Scanner sc, Caretaker ct) {
        this.buildingList = buildingList;
        this.buildNo = buildNo;
        this.sc = sc;
        this.ct = ct;
        this.roomLength = roomLength;
        this.roomWidth = roomWidth;
        building = buildingList.get(buildNo);
    }

    public void execute() {

        sc.nextLine();
        roomNo = buildingList.get(buildNo).getRoomQty() + 1;
        ct.saveBuidling(building, buildNo, toString(), false);

        System.out.println("Updated Building:");
        buildingList.get(buildNo).addRoom(roomLength, roomWidth);
        buildingList.get(buildNo).printBuilding();


    };

    public String toString(){
        return "Add Room : Building No. " + buildNo + " ,Room No. " + roomNo + ", Length :
" + roomLength + ", Width : " + roomWidth;
    }
}
```

AddroomsCommandFactory

```
package Command;

import java.util.*;
import Building.*;
import Memento.Caretaker;

public class AddroomsCommandFactory implements CommandFactory {
    private HashMap<Integer, Building> buildingList;
    private int buildNo;
    private Scanner sc;
    private Caretaker ct;

    //local
    private double roomLength;
    private double roomWidth;

    public AddroomsCommandFactory(HashMap<Integer, Building> buildingList, int buildNo,
Scanner sc, Caretaker ct) {
        this.sc = sc;
        this.buildingList = buildingList;
        this.buildNo = buildNo;
        this.ct = ct;
    }

    public Command createCommand() {
        System.out.print("Length: ");
        roomLength = sc.nextDouble();

        System.out.print("Width: ");
        roomWidth = sc.nextDouble();
        Command c = new AddroomsCommand(roomLength, roomWidth, buildingList, buildNo, sc,
ct);
        return c;
    }


}
```

Command

```
package Command;

public interface Command{
    public abstract void execute();
}
```

CommandFactory

```
package Command;

public interface CommandFactory{
    public abstract Command createCommand();
}
```

CreateBuildingCommand

```
package Command;

import java.util.*;
import Building.*;
import Memento.*;

public class CreateBuildingCommand implements Command {
    private Scanner sc;
    private Caretaker ct;
    private HashMap<Integer, Building> buildingList;

    //local
    private Building building;
    private String input;
    private HashMap<String, BuildingFactory> tempHash;

    public CreateBuildingCommand(HashMap<String, BuildingFactory> tempHash,
HashMap<Integer, Building> buildingList, Scanner sc, Caretaker ct, String input) {
        this.buildingList = buildingList;
        this.tempHash = tempHash;
        this.sc = sc;
        this.ct = ct;
        this.input = input;
    }

    public void execute() {

        building = tempHash.get(input).createBuilding(sc);
        buildingList.put(building.getId(), building);

        ct.saveBuidling(building, building.getId(), this.toString(), true);
    }

    public String toString(){
        return "Add Building: "+buildingList.get(building.getId()).toString();
    }

}
```

CreateBuildingCommandFactory

```
package Command;
import java.util.*;
import Building.*;
import Memento.*;

public class CreateBuildingCommandFactory implements CommandFactory{

    private HashMap<Integer, Building> buildingList;
    private Scanner sc;
    private Caretaker ct;
    private String input;
    private HashMap<String, BuildingFactory> tempHash;

    public CreateBuildingCommandFactory(HashMap<Integer, Building> buildingList, Scanner
sc, Caretaker ct) {
        this.sc = sc;
        this.buildingList = buildingList;
        this.ct = ct;
        this.tempHash = new HashMap<>();

        tempHash.put("a",new ApartmentFactory());
        tempHash.put("h",new HouseFactory());
    }

    @Override
    public Command createCommand() {

        System.out.println("Enter Building Type (a=Apartment/h=House):");
        input = sc.nextLine();

        while(!input.equals("a")&&!input.equals("h")){
            System.out.println("Wrong input");
            System.out.println("Enter Building Type (a=Apartment/h=House):");
            input = sc.nextLine();
        }

        Command c = new CreateBuildingCommand(tempHash, buildingList, sc, ct, input);
        return c;
    }

}
```

DeleteroomsCommand

```
package Command;

import Building.*;
import Memento.*;

import java.util.*;

public class DeleteroomsCommand implements Command {

    private HashMap<Integer, Building> buildingList;
    private Scanner sc;
    private int buildNo;
    private Caretaker ct;
    private int roomNo;

    private Building building;

    public DeleteroomsCommand(int roomNo, HashMap<Integer, Building> buildingList, int
buildNo, Scanner sc, Caretaker ct) {
        this.roomNo = roomNo;
        this.buildingList = buildingList;
        this.buildNo = buildNo;
        this.sc = sc;
        this.ct = ct;
        building = buildingList.get(buildNo);
    }

    public void execute() {

        ct.saveBuidling(building, buildNo, toString(), false);

        System.out.println("Updated Building:");
        buildingList.get(buildNo).deleteRoom(roomNo);
        buildingList.get(buildNo).printBuilding();

        sc.nextLine();

    };
    public String toString() {
        return "Delete Room : Building No. " + buildNo + " ,Room No. " + roomNo + ", Length
: "+ buildingList.get(buildNo).getRooms().get(roomNo-1).getLength() + ", Width : " +
buildingList.get(buildNo).getRooms().get(roomNo-1).getWidth();
    }
}
```

DeleteroomsCommandFactory

```
package Command;

import java.util.*;
import Building.*;
import Memento.Caretaker;

public class DeleteroomsCommandFactory implements CommandFactory {

    private HashMap<Integer, Building> buildingList;
    private int buildNo;
    private Scanner sc;
    private Caretaker ct;
    private int roomNo;

    public DeleteroomsCommandFactory(HashMap<Integer, Building> buildingList, int buildNo,
Scanner sc, Caretaker ct) {
        this.buildingList = buildingList;
        this.buildNo = buildNo;
        this.sc = sc;
        this.ct = ct;

    }

    public Command createCommand() {

        System.out.println("Room No.: ");
        roomNo = sc.nextInt();

        Command c = new DeleteroomsCommand(roomNo, buildingList, buildNo, sc, ct);
        return c;
    }
}
```

DisplayCommand

```
package Command;

import Building.*;
import java.util.*;

public class DisplayCommand implements Command {
    private HashMap <Integer,Building> buildingList;
    private Scanner sc;
    private String input;

    public DisplayCommand(HashMap<Integer,Building> buildingList, Scanner sc) {
        this.buildingList = buildingList;
        this.sc = sc;
        this.input = "";
    }

    public void execute() {

        System.out.println("Enter Building No. (* to display all):");
        input = sc.nextLine();

        if (input.equals("*")) {
            //use Treemap to sort the hasmap by BuildingNo integer
            Map<Integer, Building> sorted = new TreeMap<>(buildingList);

            //use for each loop the hashmap
            for (Map.Entry i : sorted.entrySet()) {

                System.out.println(sorted.get(i.getKey()).toString());
            }
        } else {
            buildingList.get(Integer.parseInt(input)).printBuilding();
        }
    }

}
```

DisplayCommandFactory

```
package Command;
import java.util.*;
import Building.*;

public class DisplayCommandFactory implements CommandFactory {
    private Scanner sc;
    private HashMap<Integer, Building> buildingList;

    public DisplayCommandFactory( HashMap<Integer, Building> buildingList, Scanner sc){
        this.buildingList = buildingList;
        this.sc = sc;
    }

    public Command createCommand(){
        return new DisplayCommand(buildingList, sc);
    }

}
```

EditRoomCommand

```
package Command;

import java.util.*;
import Building.*;
import Memento.*;

public class EditRoomCommand implements Command {
    private Scanner sc;
    private HashMap<Integer, Building> buildingList;
    private int buildNo;

    private HashMap<String, CommandFactory> tempHash = new HashMap<>();
    private String input;

    public EditRoomCommand(HashMap<Integer, Building> buildingList, int buildNo, Scanner
sc, Caretaker ct) {
        this.sc = sc;
        this.buildingList = buildingList;
        this.buildNo = buildNo;

        tempHash.put("a",new AddroomsCommandFactory(buildingList, buildNo, sc, ct));
        tempHash.put("d",new DeleteroomsCommandFactory(buildingList, buildNo, sc, ct));
        tempHash.put("m",new ModifyroomsCommandFactory(buildingList, buildNo, sc, ct));
    }

    public void execute() {

        buildingList.get(buildNo).printBuilding();
        System.out.println("");
        System.out.println("Please enter command: [a|d|m]");
        System.out.println("a = add room, d = delete room, m = modify room");

        input = sc.next();
```

```
        sc.nextLine();


        while(!input.equals("a")&&!input.equals("d")&&!input.equals("m")){
            System.out.println("Wrong input");
            System.out.println("Please enter command: [a|d|m]");
            System.out.println("a = add room, d = delete room, m = modify room");
            input = sc.nextLine();
        }
        tempHash.get(input).createCommand().execute();
    }

}
```

EditRoomCommandFactory

```
package Command;

import java.util.*;
import Building.*;
import Memento.Caretaker;

public class EditRoomCommandFactory implements CommandFactory {
    private HashMap<Integer, Building> buildingList;
    private Scanner sc;
    private Caretaker ct;

    private int buildNo;

    public EditRoomCommandFactory(HashMap<Integer, Building> buildingList, Scanner sc,
Caretaker ct) {
        this.sc = sc;
        this.buildingList = buildingList;
        this.ct = ct;
    }

    @Override
    public Command createCommand() {
        System.out.print("Building No.: ");
        buildNo = sc.nextInt();

        return new EditRoomCommand(buildingList, buildNo, sc, ct);
    }
}
```

ExitCommand

```
package Command;

public class ExitCommand implements Command {

    public void execute() {
        System.exit(0);
    }

}
```

ExitCommandFactory

```
package Command;

public class ExitCommandFactory implements CommandFactory{

    public Command createCommand(){
        return new ExitCommand();
    }

}
```

ListUndoRedoCommand

```
package Command;

import java.util.*;
import Memento.*;

public class ListUndoRedoCommand implements Command {
    private Caretaker ct;
    private Iterator iter;

    public ListUndoRedoCommand(Caretaker ct) {
        this.ct = ct;
    }

    public void execute() {
        System.out.println("");
        System.out.println("Undo List :");

        if (!ct.getundoCommand().isEmpty()) {
            iter = ct.getundoCommand().iterator();
            while (iter.hasNext()) {
                String m = (String) iter.next();
                System.out.println(m);
            }
        } else {
            System.out.println("Nothing in Undo List");
        }
        System.out.println("");
```

```
        System.out.println("Redo List :");
        if (!ct.getredoCommand().isEmpty()) {
            iter = ct.getredoCommand().iterator();

            while (iter.hasNext()) {
                String m = (String) iter.next();
                System.out.println(m);
            }
        } else {
            System.out.println("Nothing in Redo List");
        }

    }

}
```

ListUndoRedoCommandFactory

```
package Command;

import Memento.*;

public class ListUndoRedoCommandFactory implements CommandFactory {
    private Caretaker ct;

    public ListUndoRedoCommandFactory(Caretaker ct) {
        this.ct = ct;
    }

    public Command createCommand() {
        return new ListUndoRedoCommand(ct);
    }

}
```

ModifyBuildingCommand

```
package Command;

import Building.*;
import Memento.*;

import java.util.*;

public class ModifyBuildingCommand implements Command {
    private HashMap<Integer, Building> buildingList;
    private Scanner sc;
    private Caretaker ct;
    private String staff;
    private int buildNo;
    private double rent;
    private int floors;
```

```
    public ModifyBuildingCommand(HashMap<Integer, Building> buildingList, Scanner sc,
Caretaker ct) {
        this.buildingList = buildingList;
        this.sc = sc;
        this.ct = ct;
    }

    public void execute() {

        System.out.print("Building No.: ");
        buildNo = sc.nextInt();

        if (buildingList.get(buildNo) instanceof Apartment) {

            System.out.println(buildingList.get(buildNo).toString());

            System.out.print("Modify Monthly Rental.: ");
            rent = sc.nextDouble();

            System.out.print("Modify Support Staff.: ");
            sc.nextLine();

            staff = sc.nextLine();

        } else if (buildingList.get(buildNo) instanceof House) {

            System.out.println(buildingList.get(buildNo).toString());
            System.out.print("No. of Floors: ");

            floors = sc.nextInt();
        }

        ct.saveBuidling(buildingList.get(buildNo), buildNo, this.toString(), false);

        // check building type to modify
        if (buildingList.get(buildNo) instanceof Apartment) {

            Apartment apartment = (Apartment) buildingList.get(buildNo);

            apartment.setMonthlyRental(rent);

            apartment.setSupportStaff(staff);

            System.out.println("Building is modified: ");
            System.out.println(apartment.toString());

        } else if (buildingList.get(buildNo) instanceof House) {

            House house = (House) buildingList.get(buildNo);

            house.setFloors(floors);

            System.out.println("Building is modified:");
            System.out.println(house.toString());

            sc.nextLine();
```

```
        }

    }

    public String toString() {
        if (buildingList.get(buildNo) instanceof Apartment) {
            return "Modify Building: " + "Building No.: " + buildNo + ", Support Staff: " +
staff + ", Monthly Rental:"
                    + rent;
        } else {
            return "Modify Building: " + "Building No.: " + buildNo + ", No. of Floors: " +
floors;
        }

    }
}
```

ModifyBuildingCommandFactory

```
package Command;

import Building.*;
import Memento.*;

import java.util.*;

public class ModifyBuildingCommandFactory implements CommandFactory {
    private HashMap<Integer, Building> buildingList;
    private Scanner sc;
    private Caretaker ct;

    public ModifyBuildingCommandFactory(HashMap<Integer, Building> buildingList, Scanner
sc, Caretaker ct) {
        this.buildingList = buildingList;
        this.sc = sc;
        this.ct = ct;
    }

    public Command createCommand() {
        Command c = new ModifyBuildingCommand(buildingList, sc, ct);
        return c;

    }
}
```

ModifyroomsCommand

```
*/
package Command;

import Building.*;
import Memento.*;

import java.util.*;

public class ModifyroomsCommand implements Command {
    private HashMap<Integer, Building> buildingList;
    private int buildNo;
    private Caretaker ct;
    private int roomNo;
    private double roomLength;
    private double roomWidth;
    private Room room;
    private Scanner sc;
    private Building building;

    public ModifyroomsCommand(HashMap<Integer, Building> buildingList, int buildNo, int
roomNo, Scanner sc, Caretaker ct) {
        this.buildingList = buildingList;
        this.buildNo = buildNo;
        this.roomNo = roomNo;
        this.ct = ct;
        this.sc = sc;
        building = buildingList.get(buildNo);
        room = building.getRooms().get(roomNo);
    }

    public void execute() {

        System.out.print("Room No.: ");
        roomNo = sc.nextInt();

        System.out.print("Length: ");
        roomLength = sc.nextDouble();

        System.out.print("Width: ");
        roomWidth = sc.nextDouble();
        sc.nextLine();

        ct.saveRoom(room, toString());

        System.out.println("Updated Building:");
        buildingList.get(buildNo).modifyRoom(roomNo, roomLength, roomWidth);
        buildingList.get(buildNo).printBuilding();


    }
    public String toString(){
        return "Modify Room : Building No. " + buildNo + " ,Room No. " + roomNo + ", Length
: "+ roomLength + ", Width : " + roomWidth;
    }
}
```

ModifyroomsCommandFactory

```
package Command;

import java.util.*;
import Building.*;
import Memento.Caretaker;

public class ModifyroomsCommandFactory implements CommandFactory {

    private HashMap<Integer, Building> buildingList;
    private int buildNo;
    private Scanner sc;
    private Caretaker ct;
    private int roomNo;

    public ModifyroomsCommandFactory(HashMap<Integer, Building> buildingList, int buildNo,
Scanner sc, Caretaker ct) {
        this.buildingList = buildingList;
        this.buildNo = buildNo;
        this.sc = sc;
        this.ct = ct;
    }

    public Command createCommand() {

        Command c = new ModifyroomsCommand(buildingList, buildNo, roomNo,sc, ct);
        return c;
    }
}
```

RedoCommand

```
package Command;
import Memento.*;

public class RedoCommand implements Command {
  private Caretaker ct;

  public RedoCommand(Caretaker ct) {
    this.ct = ct;
  }

  public void execute() {
    if (!ct.getRedoList().isEmpty()) {
      ct.redo();
    } else {
      System.out.println("Nothing to redo!");
    }
  }

}
```

RedoCommandFactory

```
package Command;

import Memento.*;

public class RedoCommandFactory implements CommandFactory {
    private Caretaker ct;

    public RedoCommandFactory(Caretaker ct) {
        this.ct = ct;
    }

    public Command createCommand() {
        return new RedoCommand(ct);
    }

}
```

UndoCommand

```
package Command;

import Memento.*;

public class UndoCommand implements Command {
  private Caretaker ct;

  public UndoCommand(Caretaker ct) {
    this.ct = ct;
  }

  public void execute() {
    if (!ct.getUndoList().isEmpty()) {
      ct.undo();
    } else {
      System.out.println("Nothing to undo!");
    }
  };

}
```

UndoCommandFactory

```
package Command;

import Memento.*;

public class UndoCommandFactory implements CommandFactory {
    private Caretaker ct;

    public UndoCommandFactory(Caretaker ct) {
        this.ct = ct;
    }

    @Override
    public Command createCommand() {
        return new UndoCommand(ct);
    }

}
```

Memento Package

BuildingMemento

```
package Memento;

import Building.*;
import java.util.*;

public class BuildingMemento implements Memento {

    private Building building;
    private int buildNo;

    // house
    private int noOfFloors;

    // apartment
    private String supportStaff;
    private double monthlyRental;
    private boolean IsCreate;

    //
    private ArrayList<Room> mroomList;
    private ArrayList<Room> mroomListClone;

    public BuildingMemento(Building building, int buildNo, boolean IsCreate) {
        this.buildNo = buildNo;
        this.building = building;
        this.IsCreate = IsCreate;

        if (this.building instanceof House) {
            this.noOfFloors = ((House) this.building).getFloors();
        } else if (this.building instanceof Apartment) {
            this.supportStaff = ((Apartment) this.building).getSupportStaff();
            this.monthlyRental = ((Apartment) this.building).getMonthlyRental();
        }

        this.mroomListClone = (ArrayList) building.getRooms().clone();
        this.mroomList = building.getRooms();

    }

    // save the state
    public void restore() {

        if (building instanceof House) {
            ((House) building).setFloors(noOfFloors);
        } else if (building instanceof Apartment) {
            ((Apartment) building).setSupportStaff(supportStaff);
            ((Apartment) building).setMonthlyRental(monthlyRental);
        }
        mroomList.clear();
        mroomList.addAll(this.mroomListClone);
    }
```

```
    public Building getmbuilding() {
        return building;
    }

    public int getmbuildingNo() {
        return buildNo;
    }

    public boolean getIsCreate() {
        return IsCreate;
    }
}
```

Caretaker

```
package Memento;

import java.util.*;
import Building.*;

public class Caretaker {

    private HashMap<Integer, Building> buildingList;

    private Stack undoList;
    private Stack redoList;

    private LinkedList<String> undoCommand;
    private LinkedList<String> redoCommand;

    private boolean IsCreate;


    public Caretaker(HashMap<Integer, Building> buildingList) {
        this.buildingList = buildingList;

        undoList = new Stack();
        redoList = new Stack();

        undoCommand = new LinkedList<String>();
        redoCommand = new LinkedList<String>();
    }

    public void saveBuidling(Building building, int buildingNo, String message, boolean
IsCreate) {
        undoList.push(new BuildingMemento(building, buildingNo, IsCreate));
        undoCommand.push(message);
    }

    public void saveRoom(Room mroom, String message) {
        undoList.push(new ModifyRoomMemento(mroom));
        undoCommand.push(message);
    }

    public void undo() {
        if (!undoList.isEmpty()) {
```

```
            if (undoList.peek() instanceof BuildingMemento) {

                BuildingMemento undom = (BuildingMemento) undoList.pop();
                BuildingMemento remember = new BuildingMemento(undom.getmbuilding(),
undom.getmbuildingNo(), IsCreate);

                if (undom.getIsCreate()) {
                    redoList.push(undom);
                    buildingList.remove(undom.getmbuildingNo());
                } else {
                    redoList.push(remember);
                    undom.restore();
                }
                if (!undoCommand.isEmpty()) {
                    String message = (String) undoCommand.pop();
                    redoCommand.push(message);
                }

            }

            else if (undoList.peek() instanceof ModifyRoomMemento) {

                ModifyRoomMemento undom = (ModifyRoomMemento) undoList.pop();
                ModifyRoomMemento remember = new ModifyRoomMemento(undom.getRoom());

                redoList.push(remember);
                undom.restore();

                if (!undoCommand.isEmpty()) {
                    String message = (String) undoCommand.pop();
                    redoCommand.push(message);
                }
            }

        } else {
            System.out.println("\nNothing to Undo");
        }

    }

    public void redo() {
        if (!redoList.isEmpty()) {
            if (redoList.peek() instanceof BuildingMemento) {
                BuildingMemento redom = (BuildingMemento) redoList.pop();
                BuildingMemento remember = new BuildingMemento(redom.getmbuilding(),
redom.getmbuildingNo(),
                        IsCreate);
                if (redom.getIsCreate()) {
                    undoList.push(redom);
                    buildingList.put(redom.getmbuildingNo(), redom.getmbuilding());
                } else {
                    undoList.push(remember);
                    redom.restore();
                }
                if (!redoCommand.isEmpty()) {
                    String message = (String) redoCommand.pop();
                    undoCommand.push(message);
```

```
                }
            }

            else if (redoList.peek() instanceof ModifyRoomMemento) {
                ModifyRoomMemento redom = (ModifyRoomMemento) redoList.pop();
                ModifyRoomMemento remember = new ModifyRoomMemento(redom.getRoom());
                undoList.push(remember);
                redom.restore();

                if (!redoCommand.isEmpty()) {
                    String message = (String) redoCommand.pop();
                    undoCommand.push(message);
                }

            }
        } else {
            System.out.println("\nNothing to Redo");
        }
    }

    public LinkedList getundoCommand() {
        return (LinkedList) this.undoCommand.clone();
    }

    public LinkedList getredoCommand() {
        return (LinkedList) this.redoCommand.clone();
    }

    public Stack getRedoList() {
        return redoList;
    }

    public Stack getUndoList() {
        return undoList;
    }
}
```

Memento

```
package Memento;

public interface Memento {

    public void restore();

}
```

ModifyRoomMemento

```
package Memento;

import Building.*;

public class ModifyRoomMemento implements Memento {

    //room content
    private Room room;
    private double Width;
    private double Length;

    public ModifyRoomMemento(Room room) {
        this.room = room;
        this.Width = room.getWidth();
        this.Length = room.getLength();
    }

    public void restore() {
        room.setLength(Length);
        room.setWidth(Width);

    }

    public Room getRoom() {
        return room;
    }
}
```