

User requirements:

Dormitory System

The extensive network of dormitories is efficiently overseen by our system, which handles all the information and database. This solution enables owner to manage all dorms. Administrators of each dormitory can easily control all employees, and as well as dorm's residents can create reservations.

In our system we have an Owner (described by personal data (name, surname), health status: Disabled and Healthy, if disabled need to add list of demands and if healthy – medical card).

Dormitory registered in the system should be described by name, address (street, number) and rating (1-5).

Each dormitory has rooms, each has number, price, description, room size and minimum room size and room can't exist without dormitory.

Residents (described by personal data (name, surname), phone number, optional email address and also health status: Disabled and Healthy, if disabled need to add list of demands and if healthy – medical card).

The resident must have either a residence permit (described by type and status) or documents (described by name and content) giving the opportunity to stay in the country.

Each resident can make a report if something happened. Report should have reported room and also description, can add resident permit, add document and create a reservation (A reserved room is described by: start date of the reservation, end date of the reservation and rent price (minimum time of reservation is 3 months).

Employees (described by personal data (name, surname), working phone numbers (must have at least one), and also health status: Disabled and Healthy, if disabled need to add list of demands and if healthy – medical card) that works in the dormitory should have an employment (described by start date of the employment, end date of the employment, monthly salary, penalty and actual salary that calculates depending on month salary and penalties that employee has).

In dormitory works 3 types of employees: Administrator that has different responsibilities and can create reservations, Security that has patrol Area, Cleaning Stuff that has cleaning Area. Employee can change their types, security can become cleaning stuff, administrator can become security etc. And also, Employee can have several types, at the same time an employee can be an administrator, a security and a cleaning stuff.

In dormitory also can be working residents that have all rights and opportunities of employee and resident and additionally have discount for living.

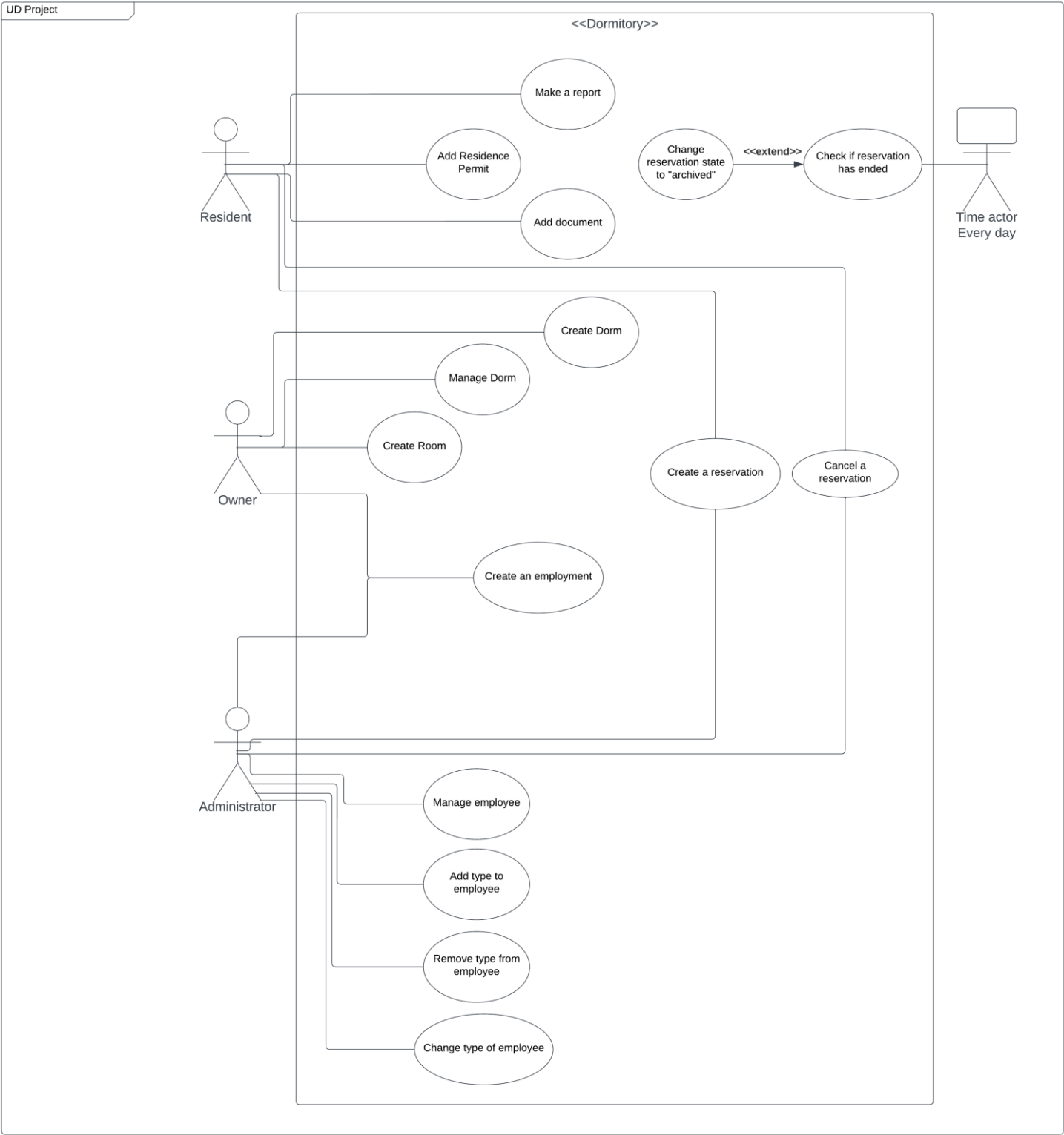
Main functionalities:

Owner can manage dormitories, create rooms and add create employments.

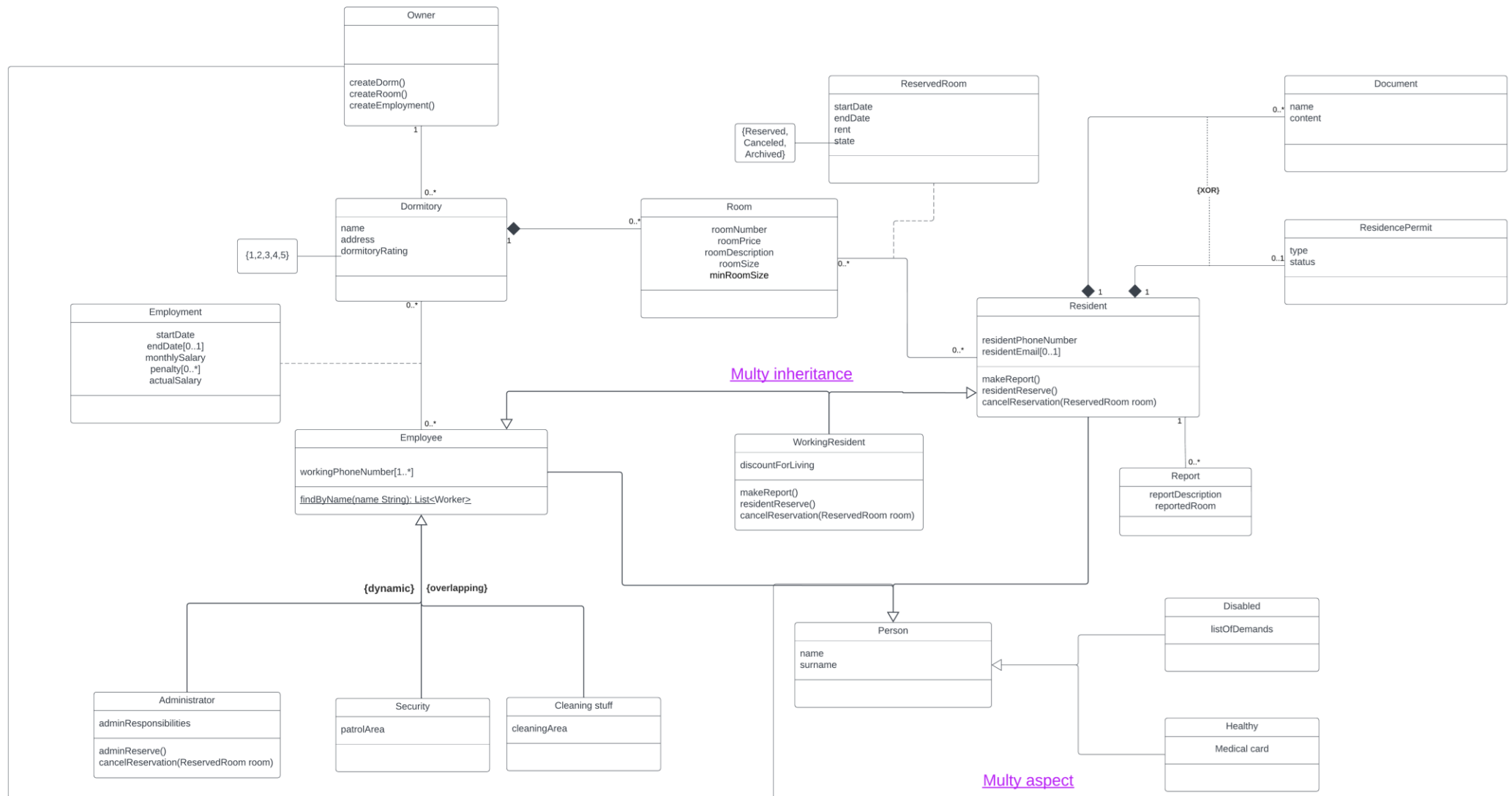
Resident can make a report, can add resident permit, add document and create, cancel a reservation.

Administrator can manage employees and create, cancel reservations.

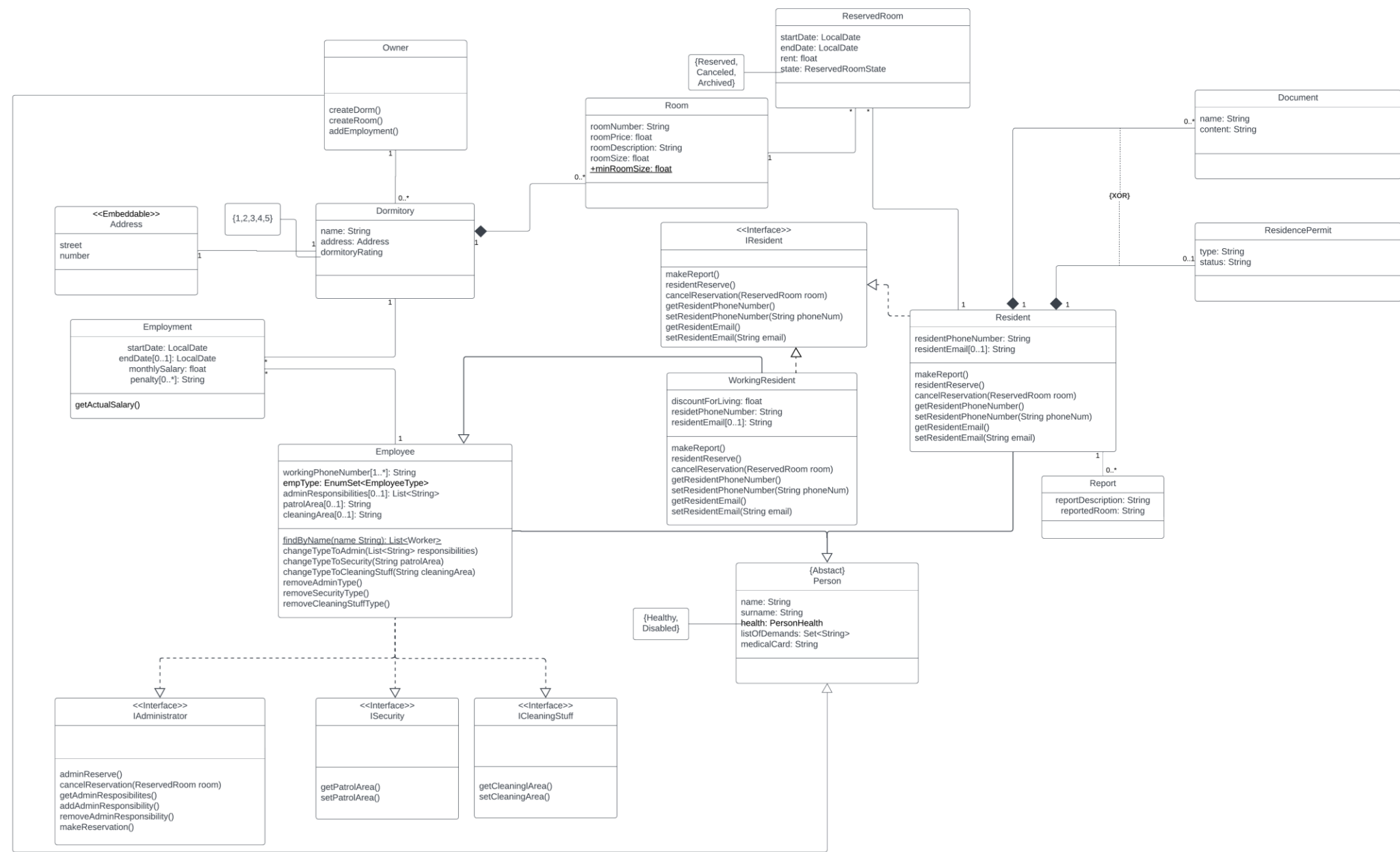
The use case diagram:



The class diagram – analytical:



The class diagram – design:



The scenario of selected use case (as text):

Use case: Make a reservation

Actor: Resident

Pre-conditions:

Resident should be in a database

Basic flow of events:

1. The actor clicks on a button "Make a reservation" and starts a use case – make a reservation
2. The system shows a new page with a first listbox with all existing dormitories.
3. The actor chooses a suitable dormitory and presses "Continue" button.
4. The system shows a new page with a second listbox with all free rooms of selected dormitory, empty Room details panel and non-interactable form with start, end date, buttons: make reservation and cancel.
5. The actor selects a room
6. The system makes a form interactable to put in start and end reserved date.
7. The actors select a period of time when wants to reserve a room and presses "Make reservation" button.
8. The system shows a confirmation window.
9. The actor confirms reservation.
10. System adds reserved room to the database with information about room and resident that made a reservation in chosen dormitory and System return to Reservation Menu page

Alternative path:

- 2a. There are no dormitories in the system
 - 2aa. Cancel the use case
- 3a. There are no free rooms of selected dormitory
 - 3aa. The actor chooses another dormitory and returns to step 3.
 - 4aa. The actor decides to cancel the use case
- 7a. The actor didn't pass validation
 - 7aa. The System show error massage. The actor selects dates again
 - 7ab. The actor selects cancel. The System shows window with text: "Do you want to make changes or quit?" and two buttons "Make changes" and "Quit".
 - 7aba. The actor decides to click Make changes and return to step 7.
 - 7abb. The actor decides to click Quit and cancel the use case.
- 9a. The actor selects back
 - 9aa. The actor returns to step 7.

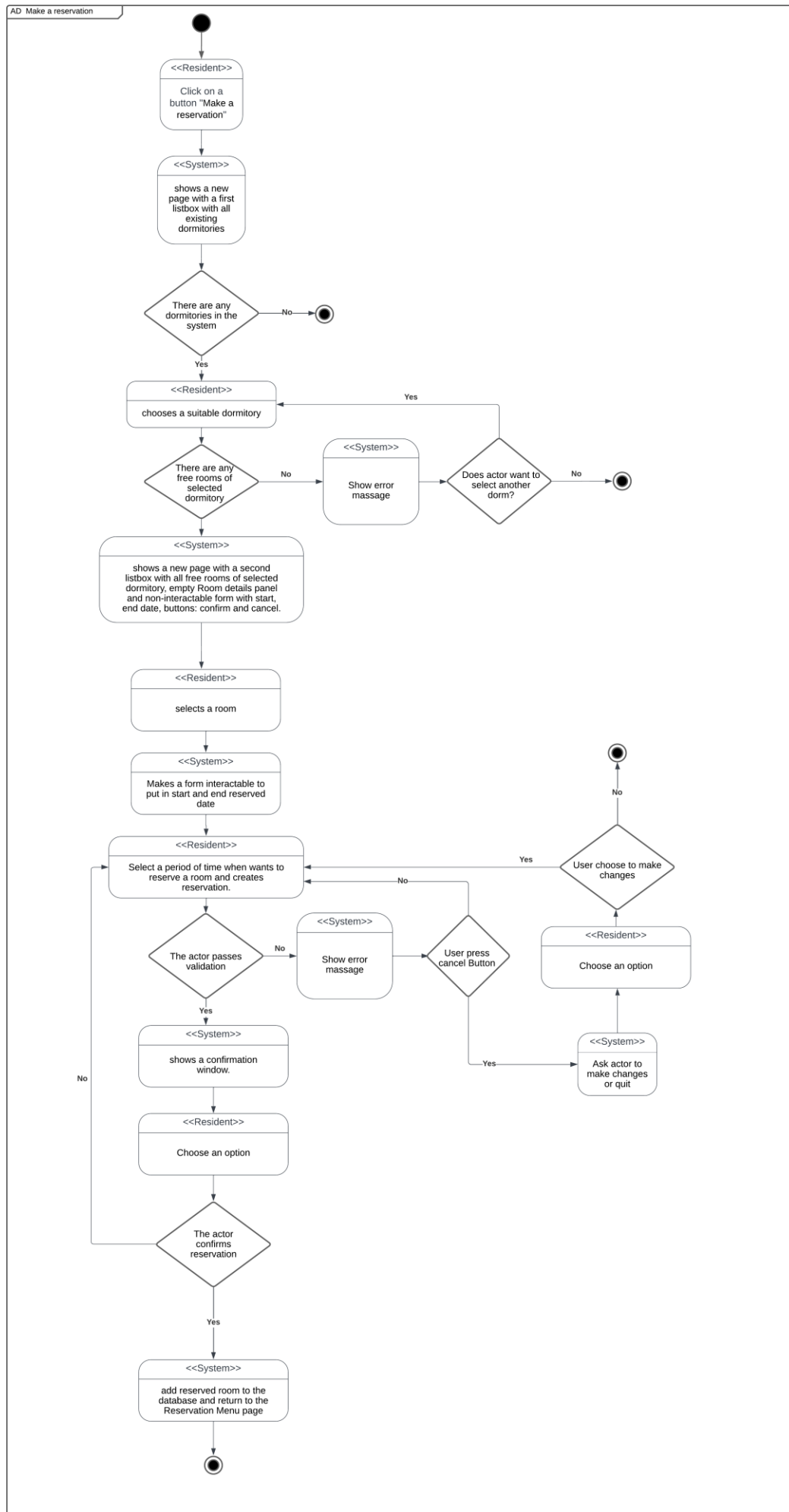
Button "Confirm" validation:

- 1) room must be selected
- 2) the end date is after the start date
- 3) period of reserve time more than 3 months

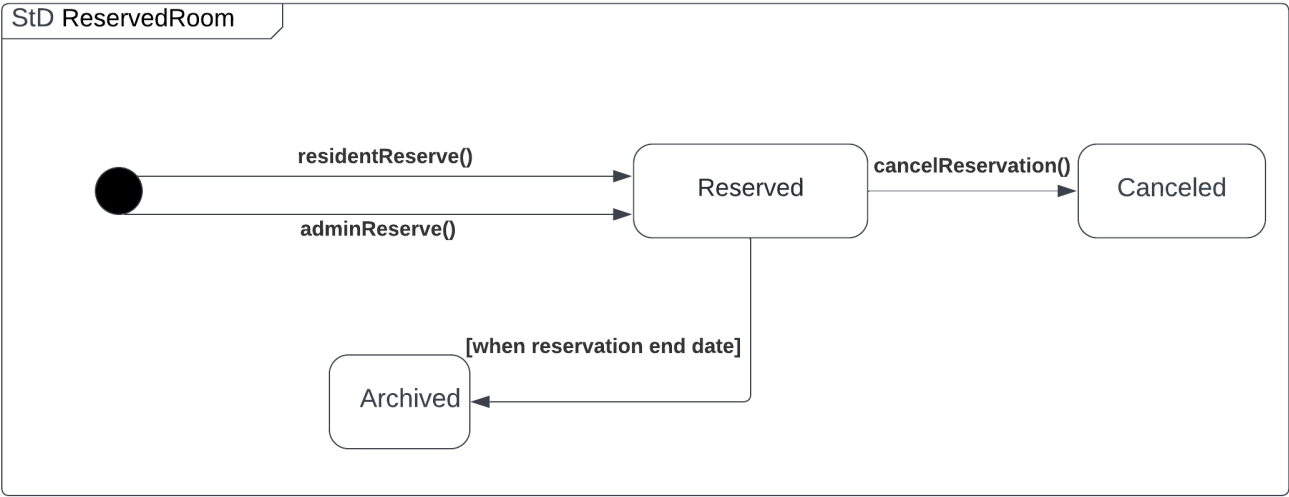
Post-conditions:

System successfully creates a reservation.

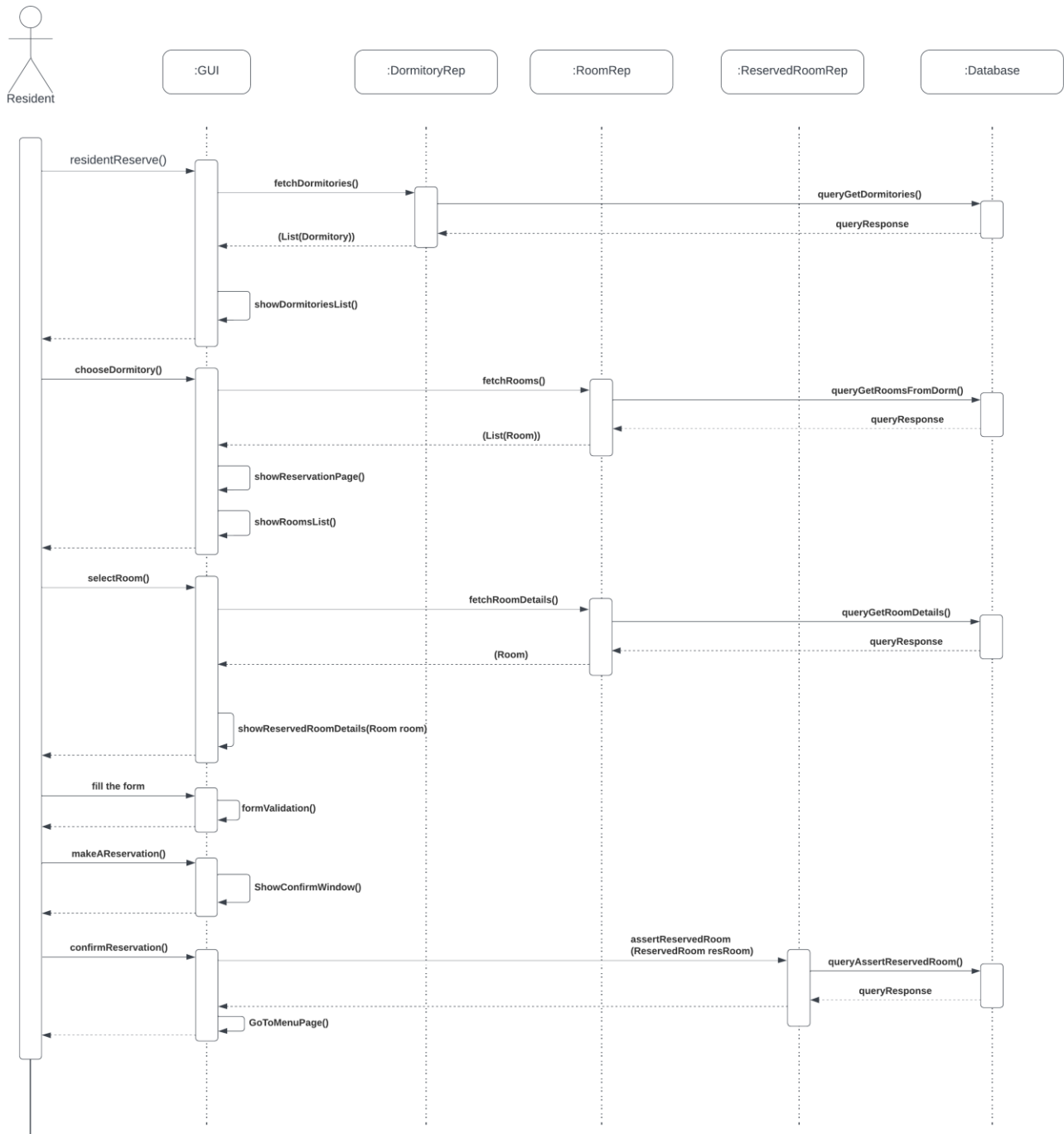
The activity diagram for picked use case



The state diagram for selected class:



The Sequence diagram:




Logo

HomeAboutFeaturesPricingContact us

Q

Welcome, Alex Marquizy



Alex

Reservation Menu





Create a reservationDelete a reservationMy reservations

HomeAboutFeaturesPricingContact usBlogSearchT & CsPrivacyCommunity

Get our newsletter

Enter your email

Subscribe




Logo

HomeAboutFeaturesPricingContact us

Q

Welcome, Alex Marquizy



Alex

Reservation process

Select a Dormitory





Continue

HomeAboutFeaturesPricingContact usBlogSearchT & CsPrivacyCommunity

Get our newsletter

Enter your email

Subscribe



Welcome, Alex Marquizy



Alex

Reservation process

Select Dormitory



Bong O' Rama

Dorm Diggity

Continue

Home
About
Features
Pricing
Contact us

Blog
Search
T & Cs
Privacy
Community

Get our newsletter

Enter your email

Subscribe



Welcome, Alex Marquizy



Alex

Reservation process

Select a Dormitory



error: there are no free rooms in selected dormitory

Continue

Home
About
Features
Pricing
Contact us

Blog
Search
T & Cs
Privacy
Community

Get our newsletter

Enter your email

Subscribe



Welcome, Alex Marquizy



Alex

Reservation process

Select a Dormitory



Bong O' Rama

Dorm Diggity

error: You must select a Dormitory

Continue

Reservation process

You selected Dorm Diggity dormitory

Select a Room



Room details:

Select Start Date:

October 2017						
Su	M	Tu	W	Th	Fr	Sa
1	2	3	4	5	6	7
8	9	1	1	1	1	1
1	1	1	1	1	2	2
2	2	2	2	2	2	2
2	3	3				

Select End Date:

October 2017						
Su	M	Tu	W	Th	Fr	Sa
1	2	3	4	5	6	7
8	9	1	1	1	1	1
1	1	1	1	1	2	2
2	2	2	2	2	2	2
2	3	3				

Make reservation

Cancel

Get our newsletter

Enter your email

Subscribe



Reservation process

You selected Dorm Diggity dormitory

B 204



B 204

C 308

Room details:

Room number: B 204

Room price: 2077 zł

Room description: Beutiful view from the window

Room size: 25 m2

Select Start Date:

October 2017						
Su	M	Tu	W	Th	Fr	Sa
1	2	3	4	5	6	7
8	9	1	1	1	1	1
1	1	1	1	1	2	2
2	2	2	2	2	2	2
2	3	3				

Select End Date:

October 2017						
Su	M	Tu	W	Th	Fr	Sa
1	2	3	4	5	6	7
8	9	1	1	1	1	1
1	1	1	1	1	2	2
2	2	2	2	2	2	2
2	3	3				

Make reservation

Cancel

Reservation process

You selected Dorm Diggity dormitory

B 204



Room details:

Room number: B 204

Room price: 2077 zł

Room description: Beutiful view from the window

Room size: 25 m2

Select Start Date:

October 2017						
Su	M	Tu	W	Th	Fr	Sa
1	2	3	4	5	6	7
8	9	1	1	1	1	1
1	1	1	1	1	2	2
2	2	2	2	2	2	2
2	3	3				

Select End Date:

October 2017						
Su	M	Tu	W	Th	Fr	Sa
1	2	3	4	5	6	7
8	9	1	1	1	1	1
1	1	1	1	1	2	2
2	2	2	2	2	2	2
2	3	3				

error: End date must be after start date

Make reservation

Cancel

Get our newsletter

Enter your email

Subscribe

Reservation process

You selected Dorm Diggity dormitory

B 204

▼

B 204

C 308

Room details:

Room number: B 204

Room price: 2077 zł

Room description: Beautiful view from the window

Room size: 25 m2

Confirm cancel

Do you want to make changes or quit?

Make changes

Quit

1	1	1	1	1	2	2
2	2	2	2	2	2	2
2	3	3				

1	1	1	1	1	2	2
2	2	2	2	2	2	2
2	3	3				

Make reservation

Cancel

Get our newsletter

Enter your email

Subscribe



Reservation process

You selected Dorm Diggity dormitory

B 204

▼

Room details:

Room number: B 204

Room price: 2077 zł

Room description: Beautiful view from the window

Confirm reservation

Dormitory: Dorm Diggity

Room: B204

Start Date: 01.01.2022

End Date: 01.01.2023

Confirm

Back

Make reservation

Cancel

Get our newsletter

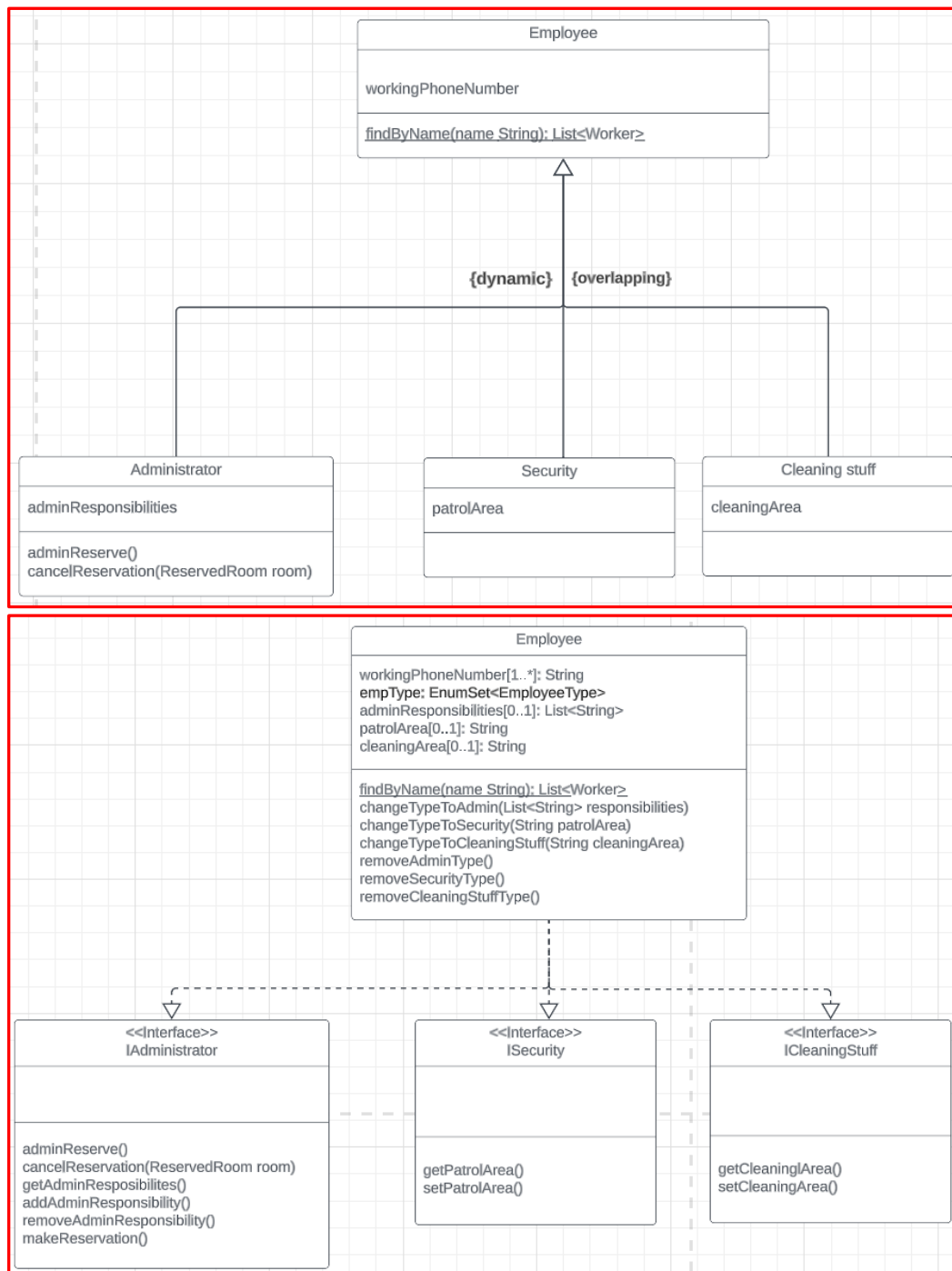
Enter your email

Subscribe



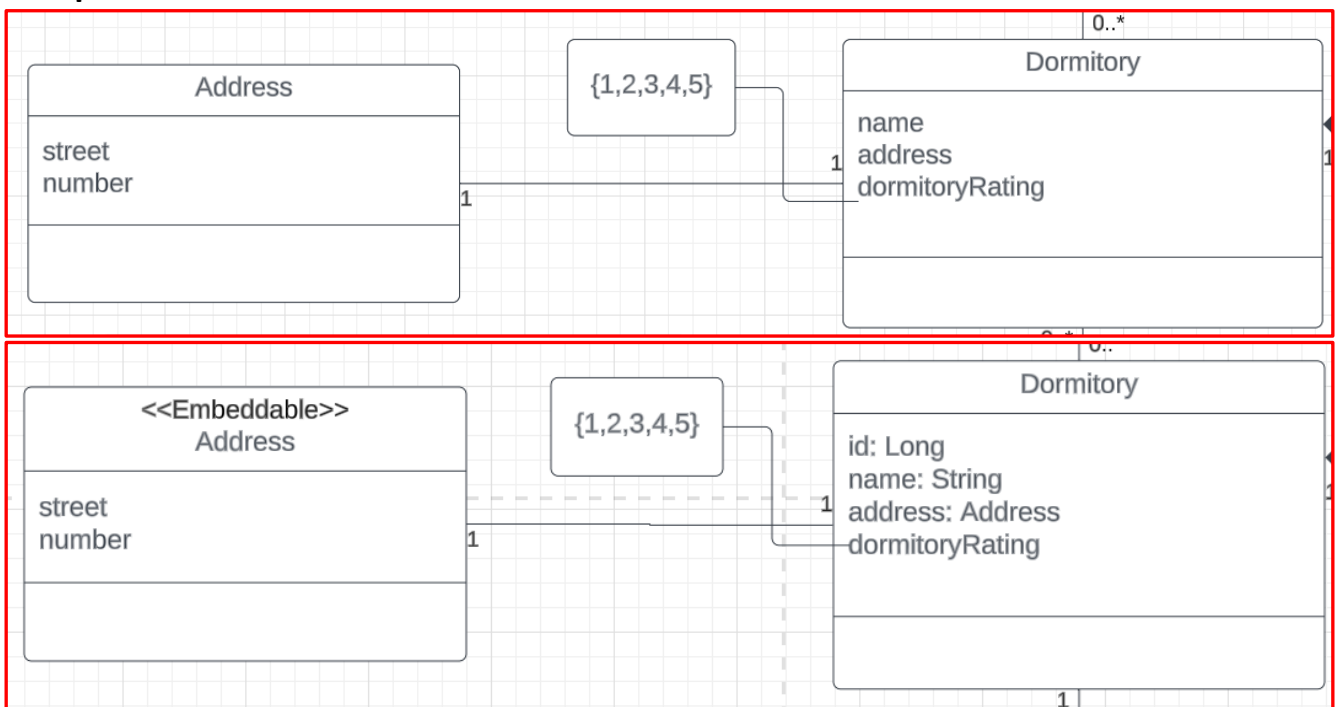
The discussion of design decisions and the effect of dynamic analysis:

- Overlapping and Dynamic



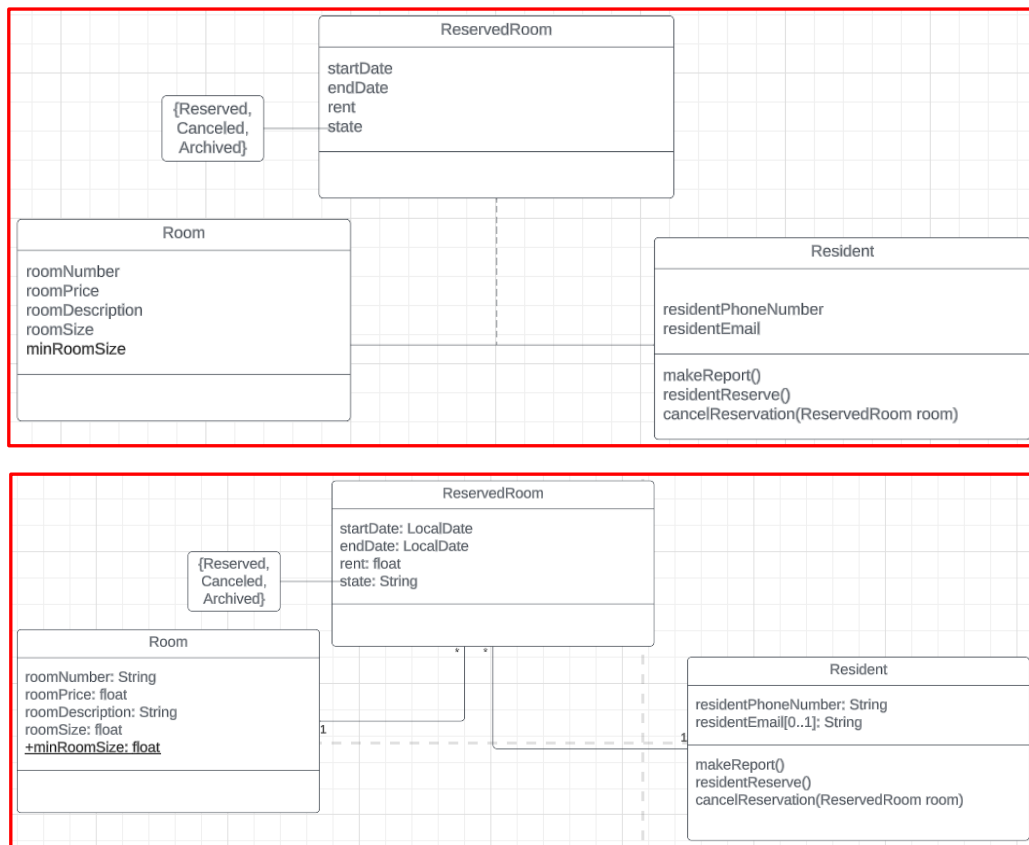
In order to implement overlapping and dynamic inheritance in my project, I used Employee class that implements three interfaces: IAdministrator, ISecurity and ICleaningStuff. Employee class has empType attribute that is a EnumSet, I use it to show what types has our Employee. Also Employee has three optional attributes: adminResponsibilites, patrolArea, cleaningArea that belong to the respective types: Administrator, Security, CleaningStuff. And in order for Employee to be able to change types, I used such methods: changeTypeToAdmin(), changeTypeToSecurity(), changeTypeToCleaningStuff() and for removing types these methods: removeAdminType(), removeSecurityType(), removeCleaningStuffType(). Constructor should take such parameters: EnumSet of Employee types, attributes that Employee has and also all attributes from each possible type of Employee(Administrator, Security, CleaningStuff). Inside constructor we should have a check for current type(s), depends on type(s) we should assign values to variables that our Employee has, to attributes that don't belong to our Employee we should set null.

- **Complex attribute**

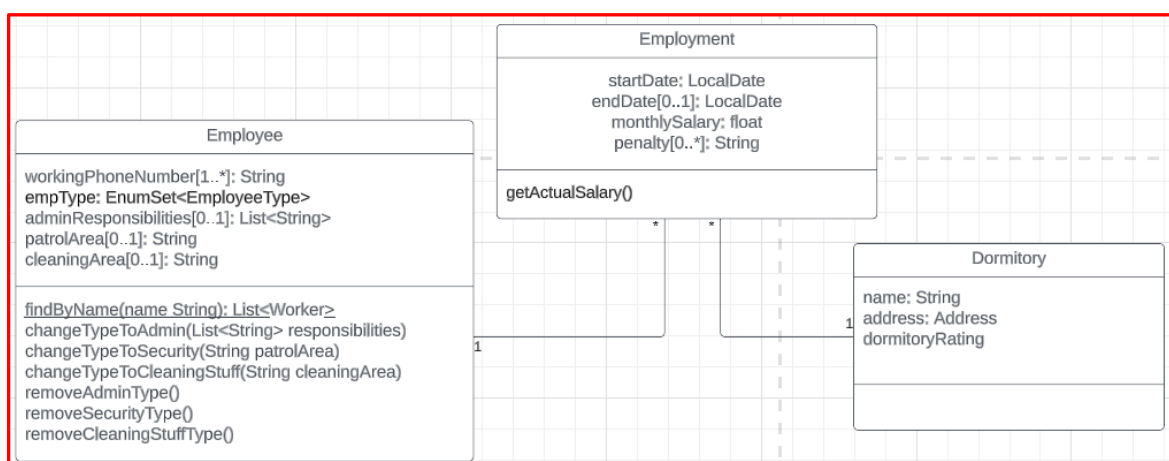
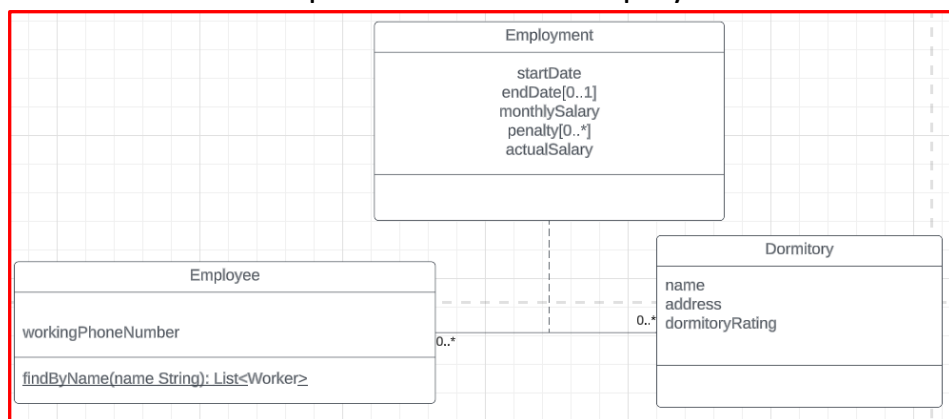


In order to implement complex attribute Address in my project, I used Address class that is described by street and number attributes with `@Embeddable` annotation and marked with `@Embedded` annotation in Dormitory class, so consequently, an entity named Dormitory is created, wherein the address details are embedded and mapped to a solitary table within the database.

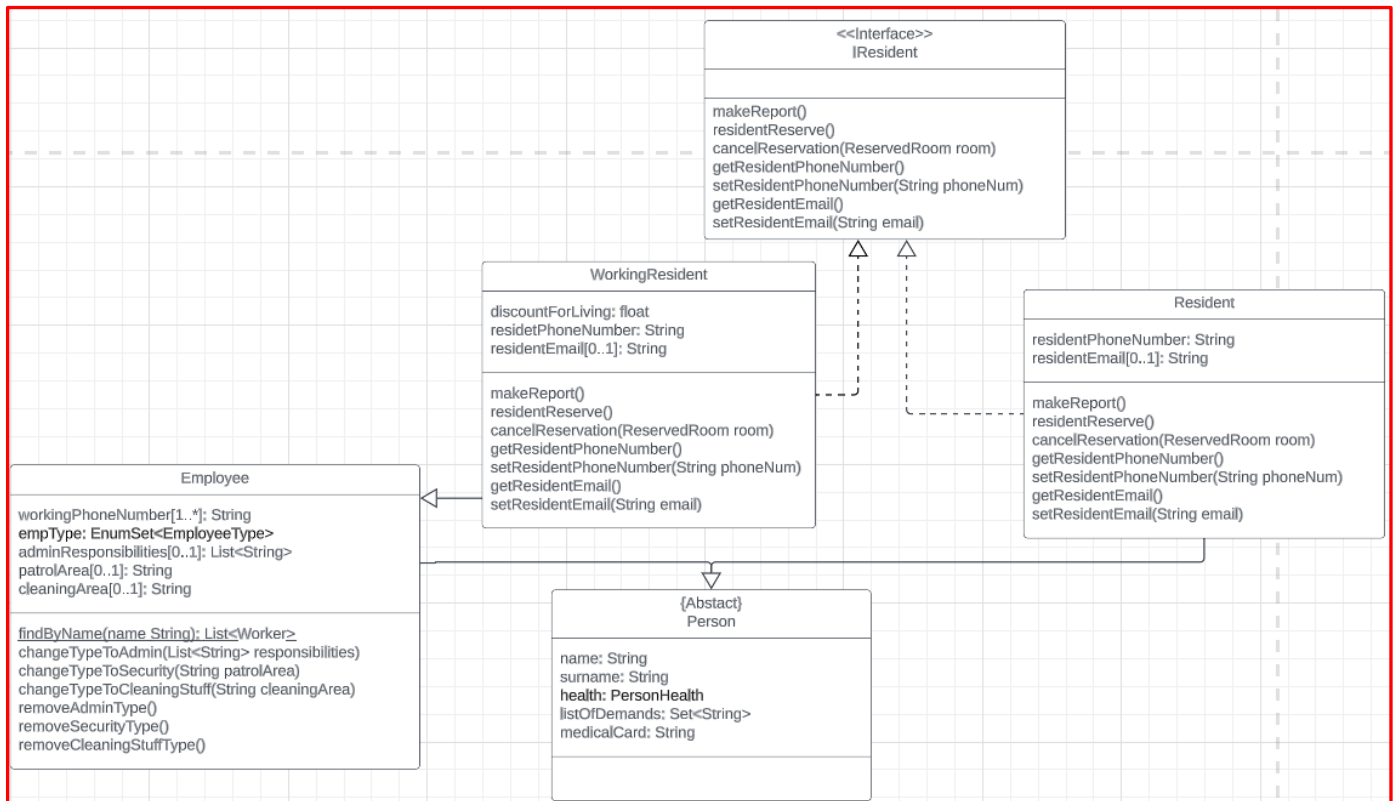
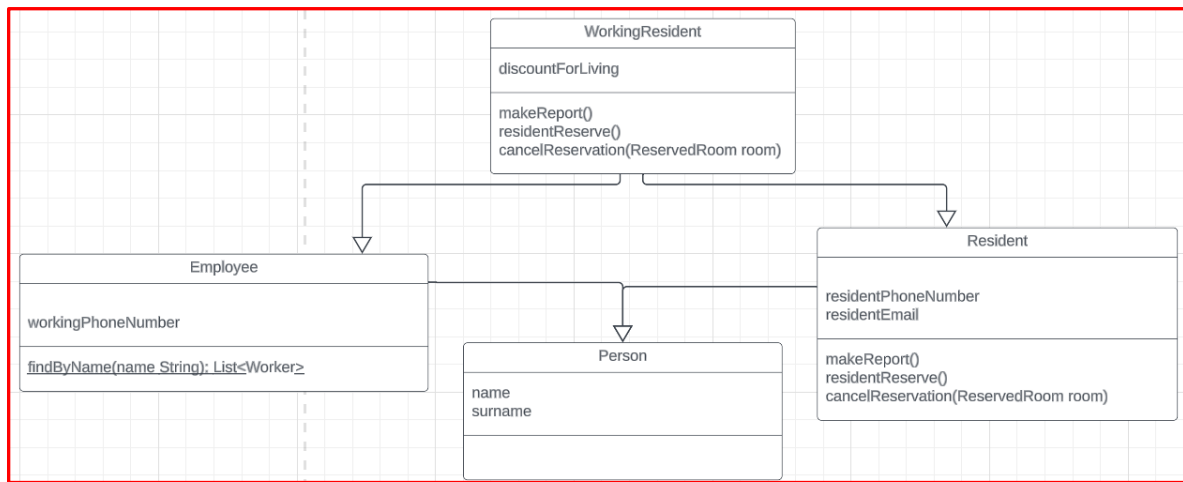
- Association with an attribute



In order to implement association with an attribute in my project, I used ReservedRoom class that is described by startDate, endDate and rent. ReservedRoom serves as a representation of the many to many relationship that exists between Room and Resident classes. Room has connection 1-* with ReservedRoom as well as class Resident. For implementation, I also used @ManyToOne, @JoinColumn annotation to connect ReservedRoom with Room, Resident classes and from other sides connected using @OneToMany annotation in Room and Resident classes. I also used this implementation for Employment class.

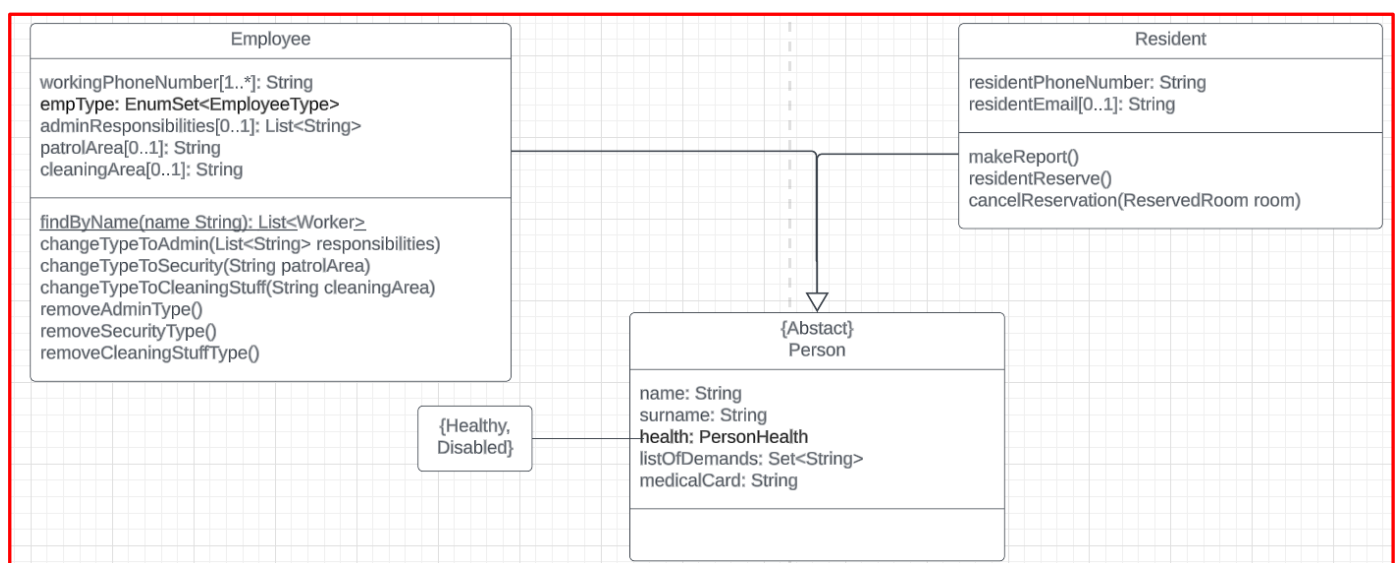
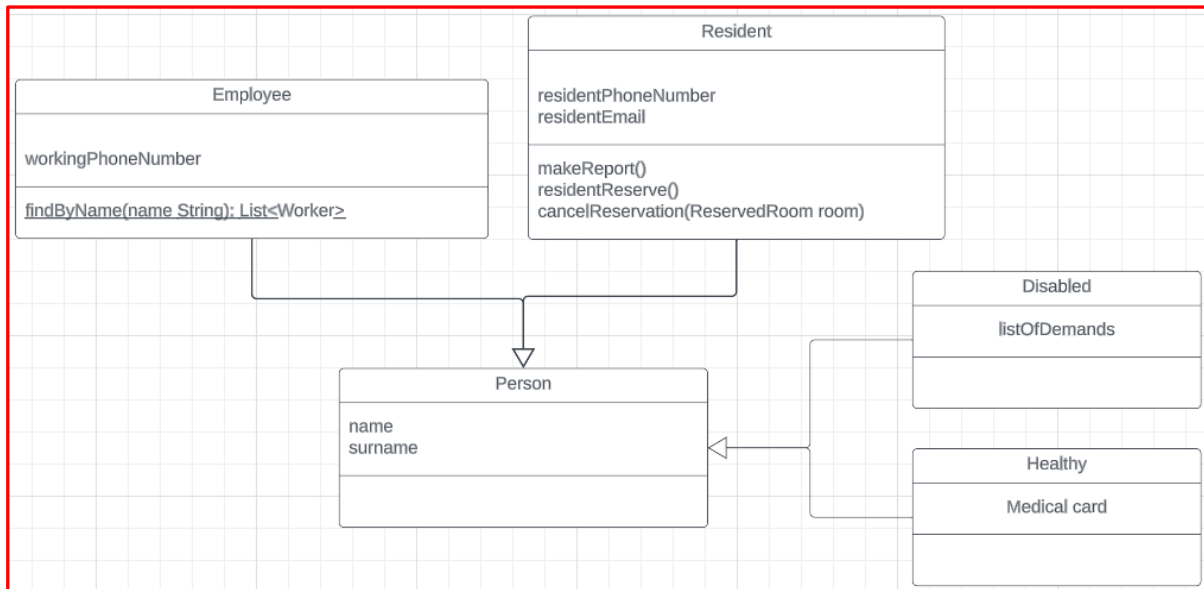


- Multi-inheritance



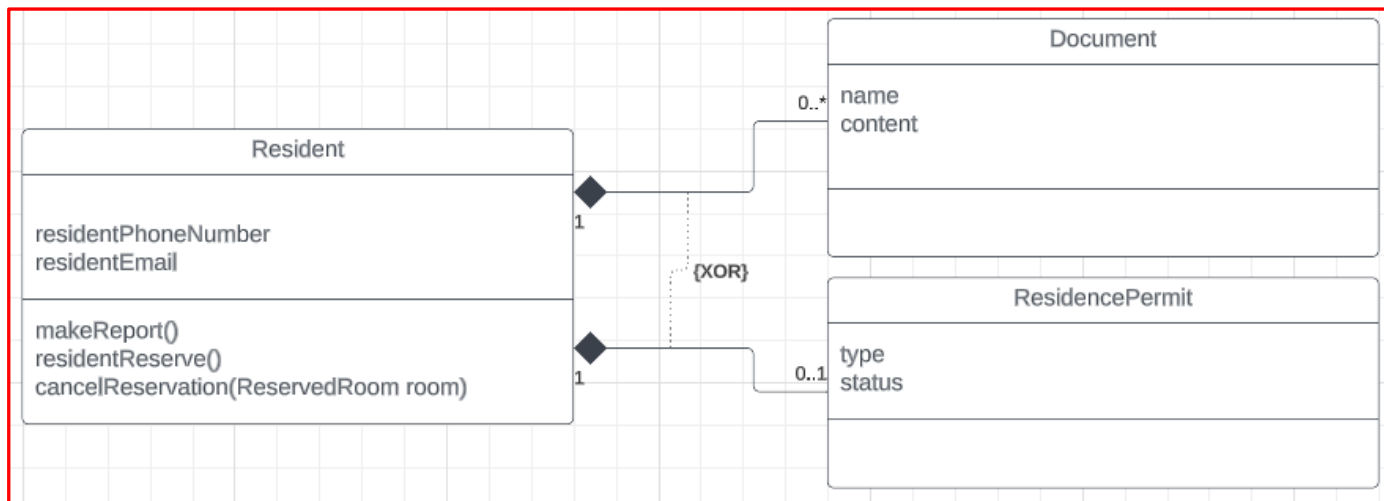
In order to implement multi-inheritance in my project, I used **WorkingResident** class that extends **Employee**, implements **IResident** Interface (described by method `makeReport()`, `residentReserve()`, `cancelReservation()` and getters and setters for resident attributes(phone number and email)), when **Employee** class and **Resident** class extends **Person** Class and also **Resident** implements **IResident** Interface. Class **WorkingResident** also has attributes that **Resident** Class has.

- **Multi-aspect**



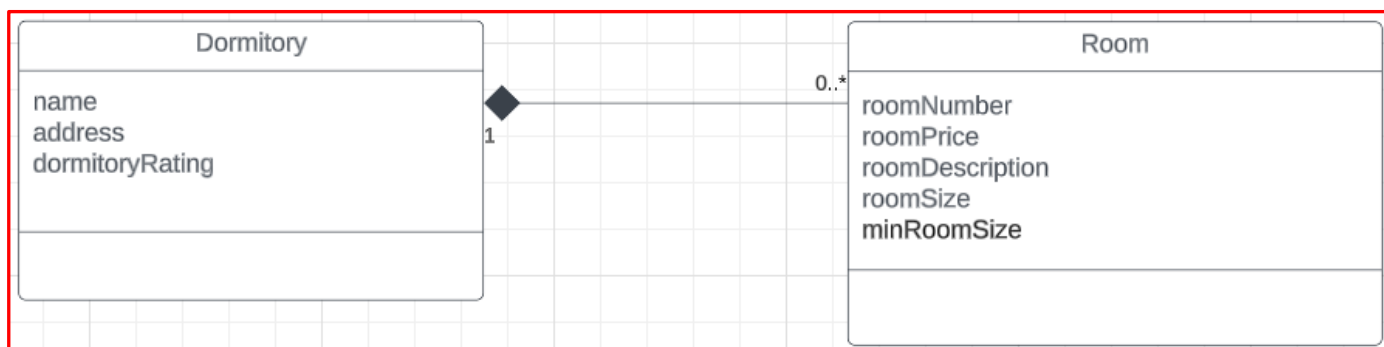
In order to implement Multi-aspect in my project, I used classes **Employee** and **Resident** that extend **Person** as first aspect, and second aspect is **Healthy** and **Disabled** that I have in Enum **PersonHealth**. Depends on this attribute value in **Person** class, **Person** has `listOfDemands` attribute when **Disabled** and `medicalCard` when **Healthy**. In order to create a **Person** with different aspect, we use constructor that take such parameters((Enum) **PersonHealth**, **Person** attributes and attributes for **Disable** and **Healthy**(`listOfDemands` and `medicalCard`)) inside constructor we have validation for **PersonHealth**, as mentioned earlier depends on this attribute we can have either `listOfDemands` attribute or `medicalCard`. For second aspect we can create **Employee** object or **Resident** object that extends **Person** class.

- **XOR**



In order to implement XOR in my project, I used two classes Document and ResidencePermit. Class Resident has excluded associations with these classes. The coexistence of these two classes is not possible, and creating one while the other exists is prohibited. I enforce this approach by implementing a validation process before object creation.

- **Composition**



In order to implement Composition association in my project, I connected Dormitory class to Room class by the composition one-to-many association, that means the object of this class cannot exist without the Dormitory. I implemented this structure by adding the annotations: @OneToMany (Dormitory), @ManyToOne (Room). A crucial attribute for the @OneToMany annotation is cascade = CascadeType.REMOVE, which ensures that if a Dormitory object is deleted, that means that if object Dormitory were deleted, all connected Rooms will also be deleted. The same association I have used in XOR constraint.

- **Keep state of the reservation**

In order to implement state of the reservation in my project, I created attribute state type Enum in ReservedRoom class, that can have such values: "Reserved", "Canceled", "Archived". In order to get state "Reserved" I created methods residentReserve() in Resident and adminReserve() in Administrator. For "Canceled" state, method cancelReservation() in Resident and Administrator, and ReservedRoom automatically gets state "Archived" when reservation end date has passed.