Building Design(Updated)

Xin Qi, CS5004 Final Project, 2024 Spring

This is a complicated project. The purpose of this design step is to help you succeed in this project. We have asked you to build a UML diagram of the entire class structure.

Include the UML Question 9 and 10 will assess this.

Answer the following questions in this document and upload with your UML diagram.

1) How are you storing your elevators in your Building model?

I am going to use an array to store my elevators in my building model, each index of the array refers to a number(or id) for an elevator. Each elevator is an object in the array, and the data structure for the elevators in the building model would be an array of elevator objects.

2) How are you storing the incoming requests so you can distribute them to the elevators?

I will use two lists(one for up requests and one for down requests) to store the incoming requests because the elements in a list are in order, which meets the demand of an elevator. I will try to implement 2 linked lists for list distribution, because we need to let the first-in requests be first-out, and the time complexity of linked lists is better than other types of list like an arraylist.

3) How are you distributing your downRequests and your upRequests to the elevators?

Using the distributeRequests() method with the getRequests(List<Request> requests) method.

While the capacity of the elevator has not been reached, add the requests in the up request list or down request list to the list of requests for the elevator while removing the requests from the up or down list.

If our up request and down request lists are not empty:
Find the elevators in the array of the elevators in the building starting from the first index.
If the elevator is taking requests, then create a list for the request list of the specific elevator.
If the elevator is on the ground floor, then it will receive requests from the up request list and process these requests.
If the elevator is on the top floor, then it will receive requests from the down request list and process these requests.

4) How are you removing all requests when a takeOutOfService request is received?

We could use the clear() method to clear the list for up requests and the list for down requests separately.
We could use the private method clearStopRequests() to clear all requests for stop.

5) How does your step method handle updating the elevators?

The step method in the Building updates the state of each elevator based on its current state(running or stopping) and pending requests, including moving the elevators up and down, opening or closing doors(keep the door open for steps), and servicing requests (go to the top or bottom after waiting for 5 steps). We could use 3 private methods, stepOutOfService, stepDoorOpen, stepTopOrBottom for the public step method.

6) How do you start processing requests?

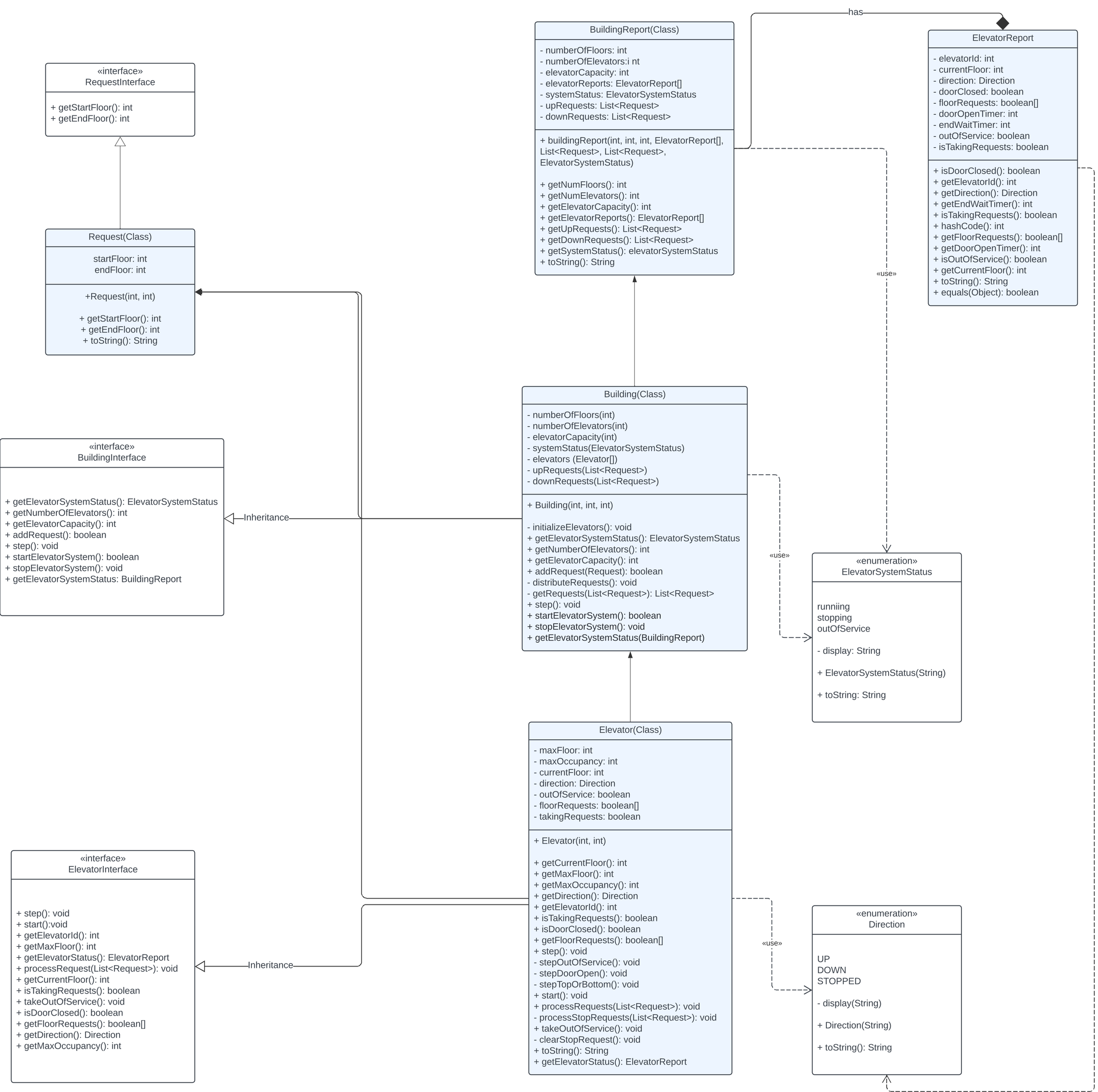
The startElevatorSystem method of the Building class initiates the processing of requests by starting all elevators.
Elevator system status changes to running.
Requests are processed by starting the elevator system.
Once the system is running, requests can be accepted(addRequest could return true and requests could be added to the request list), all the elevator status changes from stopped to running and the requests are assigned to elevators based on their capacity.

7) How do you take the building out of service?

The stopElevatorSystem method of the Building class takes the building out of service.
All new requests are rejected (addRequest returns false and does not add requests to the request list).
Call the takeOutOfService() method for each elevator.
Clear all existing requests in the request list using List.clear()method.
Set the elevatorSystemStatus to stopping.

8) How do you take the elevators out of service?

Clear all the requests for stopping on any floors.
Turn takingRequest status to false.
The wait time of stopping on the floor is cleared to 0.
The elevator is now out of service and the direction of the elevator goes down.


«interface»
RequestInterface
+ getStartFloor(): int
+ getEndFloor(): int
Request(Class)
startFloor: int
endFloor: int
+Request(int, int)
+ getStartFloor(): int
+ getEndFloor(): int
+ toString(): String
«interface»
BuildingInterface
+ getElevatorSystemStatus(): ElevatorSystemStatus
+ getNumberOfElevators(): int
+ getElevatorCapacity(): int
+ addRequest(): boolean
+ step(): void
+ startElevatorSystem(): boolean
+ stopElevatorSystem(): void
+ getElevatorSystemStatus: BuildingReport
Inheritance
BuildingReport(Class)
- numberOfFloors: int
- numberOfElevators:i nt
- elevatorCapacity: int
- elevatorReports: ElevatorReport[]
- systemStatus: ElevatorSystemStatus
- upRequests: List<Request>
- downRequests: List<Request>
+ buildingReport(int, int, int, ElevatorReport[], List<Request>, List<Request>, ElevatorSystemStatus)
+ getNumFloors(): int
+ getNumElevators(): int
+ getElevatorCapacity(): int
+ getElevatorReports(): ElevatorReport[]
+ getUpRequests(): List<Request>
+ getDownRequests(): List<Request>
+ getSystemStatus(): elevatorSystemStatus
+ toString(): String
has
ElevatorReport
- elevatorId: int
- currentFloor: int
- direction: Direction
- doorClosed: boolean
- floorRequests: boolean[]
- doorOpenTimer: int
- endWaitTimer: int
- outOfService: boolean
- isTakingRequests: boolean
+ isDoorClosed(): boolean
+ getElevatorId(): int
+ getDirection(): Direction
+ getEndWaitTimer(): int
+ isTakingRequests(): boolean
+ hashCode(): int
+ getFloorRequests(): boolean[]
+ getDoorOpenTimer(): int
+ isOutOfService(): boolean
+ getCurrentFloor(): int
+ toString(): String
+ equals(Object): boolean
«use»
Building(Class)
- numberOfFloors(int)
- numberOfElevators(int)
- elevatorCapacity(int)
- systemStatus(ElevatorSystemStatus)
- elevators (Elevator[])
- upRequests(List<Request>)
- downRequests(List<Request>)
+ Building(int, int, int)
- initializeElevators(): void
+ getElevatorSystemStatus(): ElevatorSystemStatus
+ getNumberOfElevators(): int
+ getElevatorCapacity(): int
+ addRequest(Request): boolean
- distributeRequests(): void
- getRequests(List<Request>): List<Request>
+ step(): void
+ startElevatorSystem(): boolean
+ stopElevatorSystem(): void
+ getElevatorSystemStatus(BuildingReport)
«use»
«enumeration»
ElevatorSystemStatus
runniing
stopping
outOfService
- display: String
+ ElevatorSystemStatus(String)
+ toString: String
«interface»
ElevatorInterface
+ step(): void
+ start():void
+ getElevatorId(): int
+ getMaxFloor(): int
+ getElevatorStatus(): ElevatorReport
+ processRequest(List<Request>): void
+ getCurrentFloor(): int
+ isTakingRequests(): boolean
+ takeOutOfService(): void
+ isDoorClosed(): boolean
+ getFloorRequests(): boolean[]
+ getDirection(): Direction
+ getMaxOccupancy(): int
Inheritance
Elevator(Class)
- maxFloor: int
- maxOccupancy: int
- currentFloor: int
- direction: Direction
- outOfService: boolean
- floorRequests: boolean[]
- takingRequests: boolean
+ Elevator(int, int)
+ getCurrentFloor(): int
+ getMaxFloor(): int
+ getMaxOccupancy(): int
+ getDirection(): Direction
+ getElevatorId(): int
+ isTakingRequests(): boolean
+ isDoorClosed(): boolean
+ getFloorRequests(): boolean[]
+ step(): void
- stepOutOfService(): void
- stepDoorOpen(): void
- stepTopOrBottom(): void
+ start(): void
+ processRequests(List<Request>): void
- processStopRequests(List<Request>): void
+ takeOutOfService(): void
- clearStopRequest(): void
+ toString(): String
+ getElevatorStatus(): ElevatorReport
«use»
«enumeration»
Direction
UP
DOWN
STOPPED
- display(String)
+ Direction(String)
+ toString(): String