Session 2 : Hierarchy of partitions and spaces

5.7 Grading

E Nothing or too few new done beyond the starting point, or does not work at all.

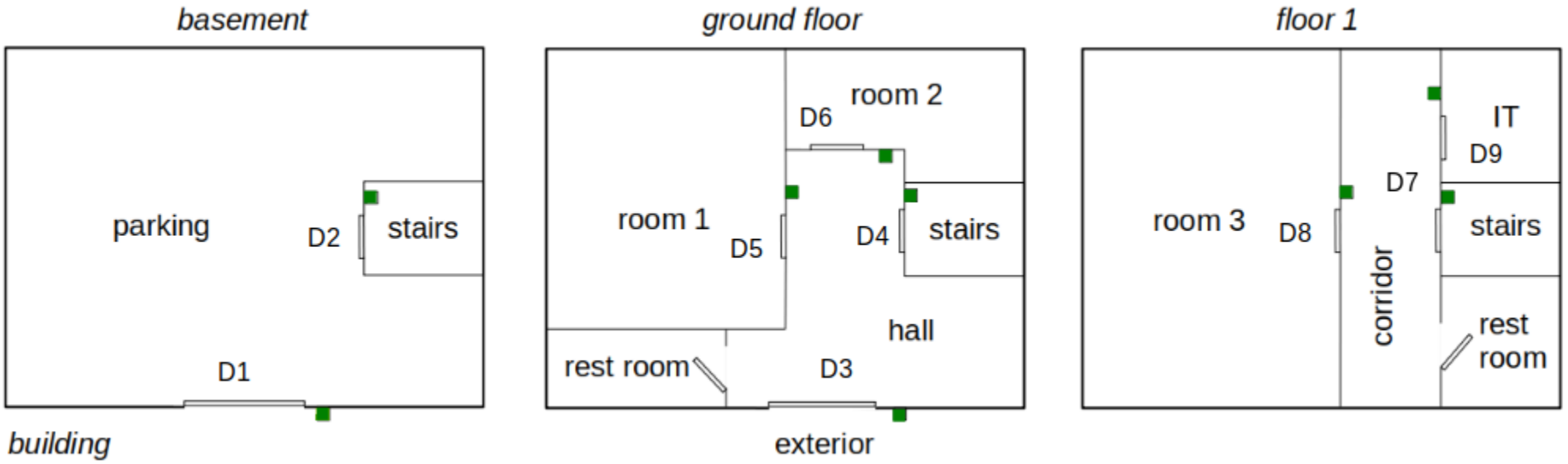
D No comments added, or too few or useless. Added door states locked and unlocked. The ACU still works at door level (no partitions or spaces), like in the starting point code.

C In addition to locked and unlocked states, added spaces and partitions (areas), but not yet user groups and schedules. A user has associated areas and upon a reader request or an area request, you check for each door involved that the user can stay in its "from" and "to" spaces. There is a sufficient number of good comments.

B C + introduced user groups and schedules, access permissions are granted to groups based on areas and schedules.

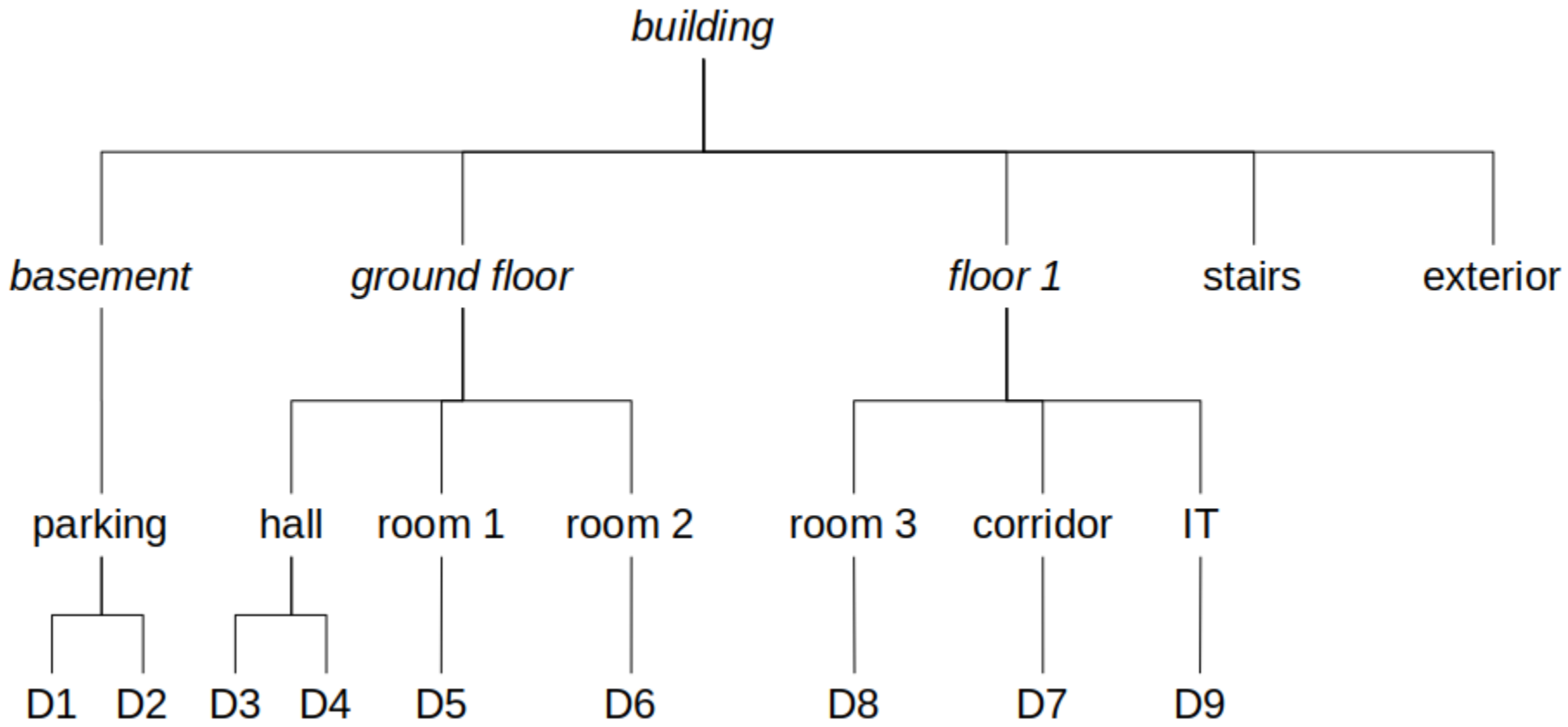
A B + introduced states propped and unlocked shortly which work well.

To all items add that the UML class diagram has to be updated according to the changes in the code.



Door name is on the space the doors gives access to (opposite to reader)

Cursive names are groups of groups and/or spaces



`Regular names` are spaces

`*Cursive names*` are groups of groups and/or spaces

`D*` are doors

Questions

- do we need a space / group to know its parent group ? eg. `parking` knows `basement`

- why do we need a space to know the doors giving acces to it ? eg. why `parking` wants to know `D1`, `D2` ?

- do we need a door to know the two spaces it communicates, the "from" and "to" spaces ? eg. `D1` : "from" = `exterior`, "to" = `parking`, `D2` : "from" = `stairs`, "to" = `parking`

Questions

- do we need a space / group to know its parent group ? eg. `parking` knows `basement`

  no at the moment

- why do we need a space to know the doors giving acces to it ? eg. why `parking` wants to know `D1`, `D2` ?

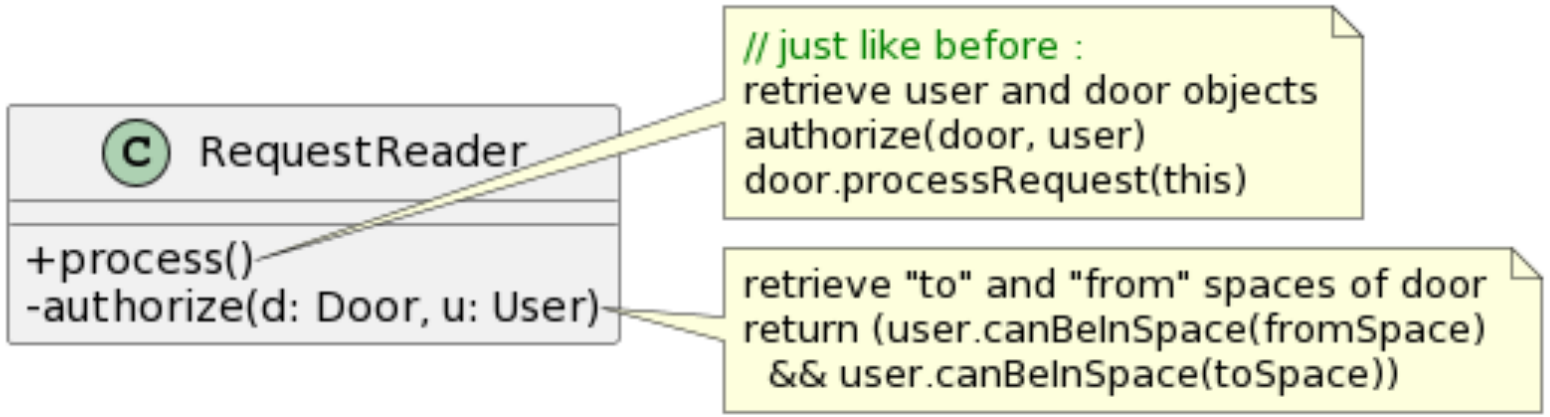
  because of area requests affect all the descendant doors of a partition or space

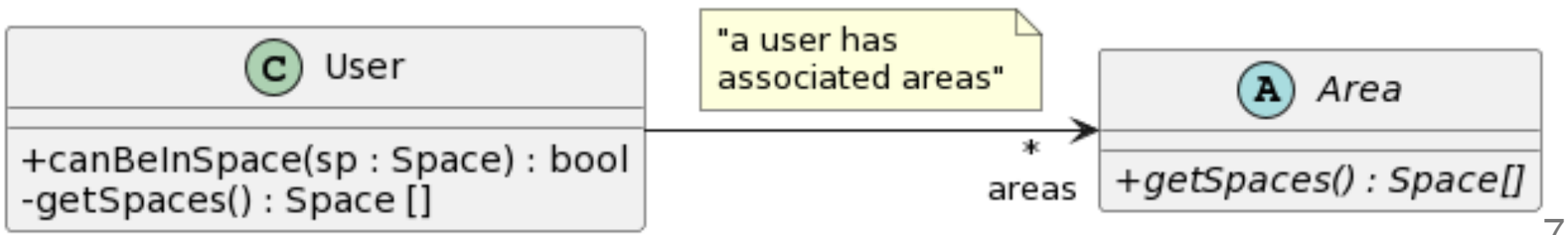
- do we need a door to know the two spaces it communicates, the "from" and "to" spaces ? eg. `D1` : "from" = `exterior`, "to" = `parking`, `D2` : "from" = `stairs`, "to" = `parking`

  in order to authorize or not a reader request concerning a certain door

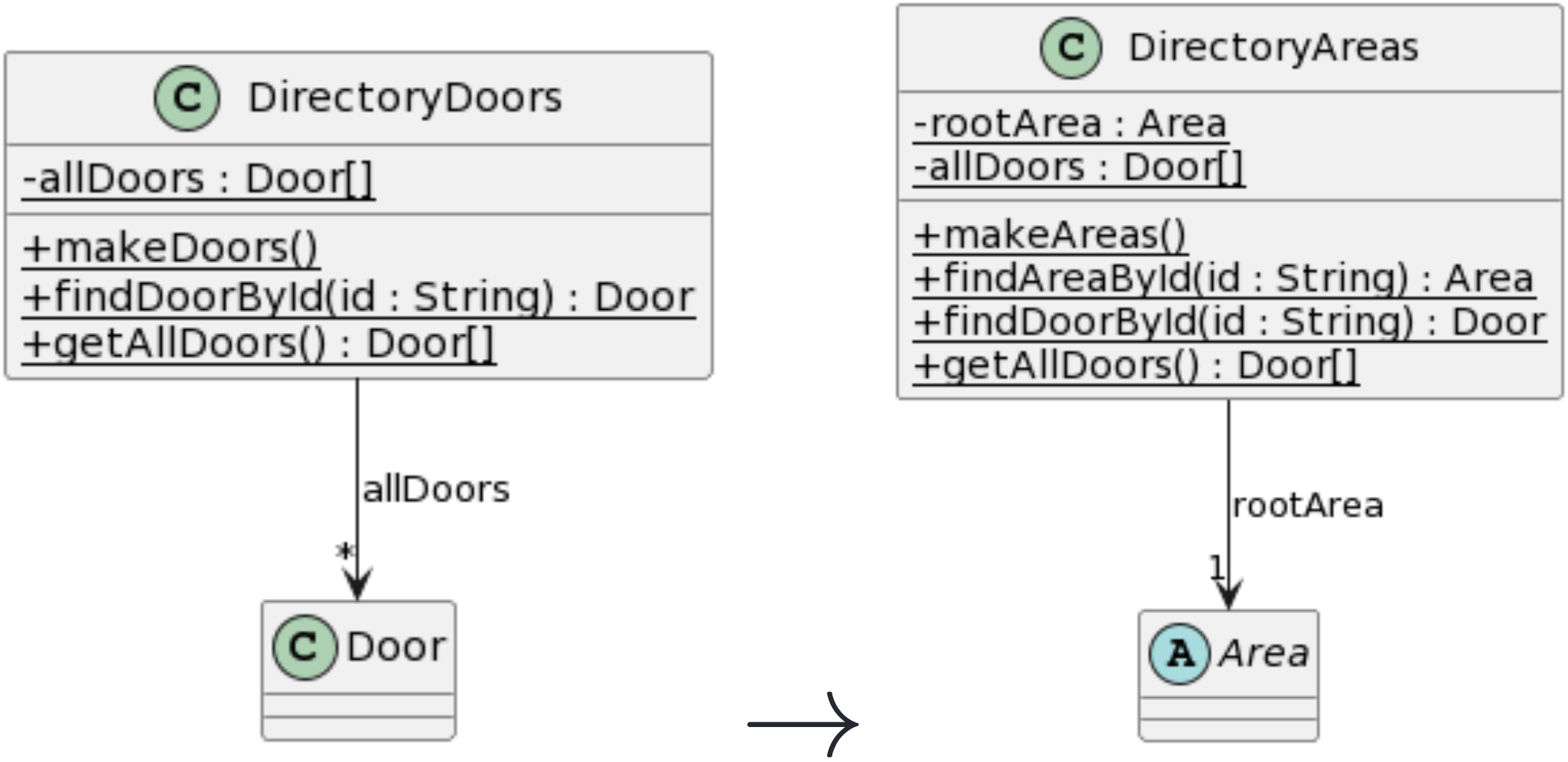
Hint 1


doors
*
Door
fromSpace
1
toSpace
1
Space
-doors : Door[]
doors giving access
to this space


RequestReader
+process()
-authorize(d: Door, u: User)
// just like before :
retrieve user and door objects
authorize(door, user)
door.processRequest(this)
retrieve "to" and "from" spaces of door
return (user.canBeInSpace(fromSpace)
&& user.canBeInSpace(toSpace))


User
+canBeInSpace(sp : Space) : bool
-getSpaces() : Space []
"a user has
associated areas"
*
areas
Area
+getSpaces() : Space[]

Hint 2


DirectoryDoors
-allDoors : Door[]
+makeDoors()
+findDoorById(id : String) : Door
+getAllDoors() : Door[]
allDoors
*
Door
DirectoryAreas
-rootArea : Area
-allDoors : Door[]
+makeAreas()
+findAreaById(id : String) : Area
+findDoorById(id : String) : Door
+getAllDoors() : Door[]
rootArea
1
Area

Hint 3: sample code

```
Partition building = new Partition("building", "...", null);
Partition basement = new Partition("basement", "...", building);

Space parking = new Space("parking", "...", basement);

Door d1 = new Door("D1", exterior, parking);
```

Hint 4

Excerpt of `RequestArea.java`

```
public class RequestArea implements Request {

    // processing the request of an area is creating the corresponding door requests
    // and forwarding them to all of its doors. For some it may be authorized and
    // action will be done, for others it won't be authorized and nothing will happen
    // to them.
    public void process() {
      Area area = DirectoryAreas.findAreaById(areaId);
      if (area != null) {
        // make all the door requests, one for each door in the area, and process them
        for (Door door : area.getDoorsGivingAccess()) {
          RequestReader requestReader =
            new RequestReader(credential, action, now, door.getId());
          requestReader.process();
        }
      }
    }
```

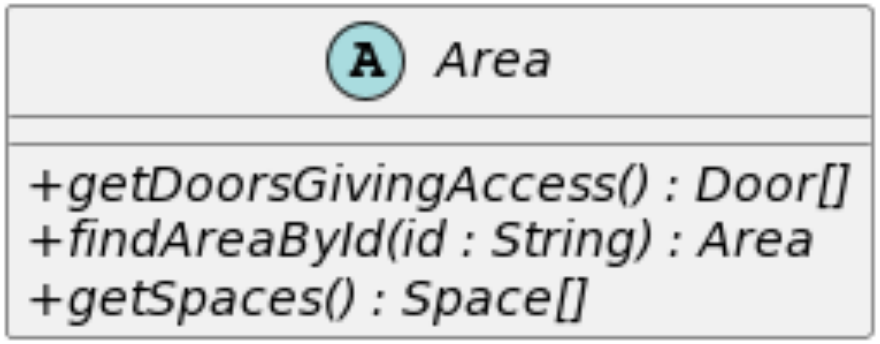
Note: `DirectoryAreas.findAreaById()`, `area.getDoorsGivingAccess()`

`DirectoryAreas` has the tree of areas = spaces and partitions and doors. Tree = root node.

```
public static Area findAreaById(String id) {
  return root.findAreaById(id); // an Area or null if not found
}
```

`area.getDoorsGivingAccess()` : list of doors that give access to some space of this area.

`area.getSpaces()` : list of spaces of this area, for the user to get all the spaces he/she can be, so as to authorize a request



Subclasses implement these 3 methods. They recursively traverse the tree with an area as root looking for the node with a certain id, or building a result list.

Try to implement them. They will be important for milestone 2.

Useful methods of `ArrayList` :

- instantiation `ArrayList<X> xs = new ArrayList<>()`
- add element `xs.add(x)`
- add elements of another list `xs.addAll(x2s)`
- has a list some object ? `xs.contains(x)`
- a list of one element `new ArrayList<>(Arrays.asList(x))`

Comments

While programming **write comments** explaining

- what is class `X` and why do we need it ? ...
- why class `X` has method `y()`
- which design patterns are you applying ? why ? which classes are involved ?