CS102 – Algorithms and Programming II
Lab Programming Assignment 6
Fall 2022

ATTENTION:

- Compress all of the Java program source files (.java) files into a single zip file.
- The name of the zip file should follow the below convention:

  CS102_Sec1_Asgn6_YourSurname_YourName.zip
- Replace the variables "YourSurname" and "YourName" with your actual surname and name.
- You may ask questions on Moodle.
- Upload the above zip file to Moodle by the deadline (if not significant points will be taken off). You will get a chance to update and improve your solution by consulting to the TA during the lab. You will resubmit your code once you demo your work to the TA.

GRADING WARNING:

- Please read the grading criteria provided on Moodle. The work must be done individually. Code sharing is strictly forbidden. We are using sophisticated tools to check the code similarities. The Honor Code specifies what you can and cannot do. Breaking the rules will result in disciplinary action.

High-Security Facility

You are going to implement a Java program to track security breaches in a facility made of rooms connected to each other. Implement a **Room** class to keep the information about each room. Each room has a unique name made out of 4 letters. Each room can be connected to at most 10 other rooms. You should store an integer that represents the security level of the room. You can use an ArrayList to keep a reference to the rooms connected to this room.

Implement a **Facility** class to manage all the rooms of the facility. This class should contain a **void importFacility(string filename)** method to import the room data of the facility from a text file. The first line of this text file includes the number of rooms in the facility, then each following line includes the name of the room and its security level separated by a space character. Finally, the names of the connected room pairs are included, separated by a dash character. An example input file can be as follows:

```
6
cmpt 10
recr 6
glmb 7
twrm 4
mgsm 11
kkda 7
cmpt-recr
cmpt-glmb
glmb-twrm
twrm-recr
mgsm-cmpt
mgsm-recr
kkda-mgsm
```

You may use https://csacademy.com/app/graph_editor/ for visualization of the file. Include the links only and without the dash character for this purpose, such as:

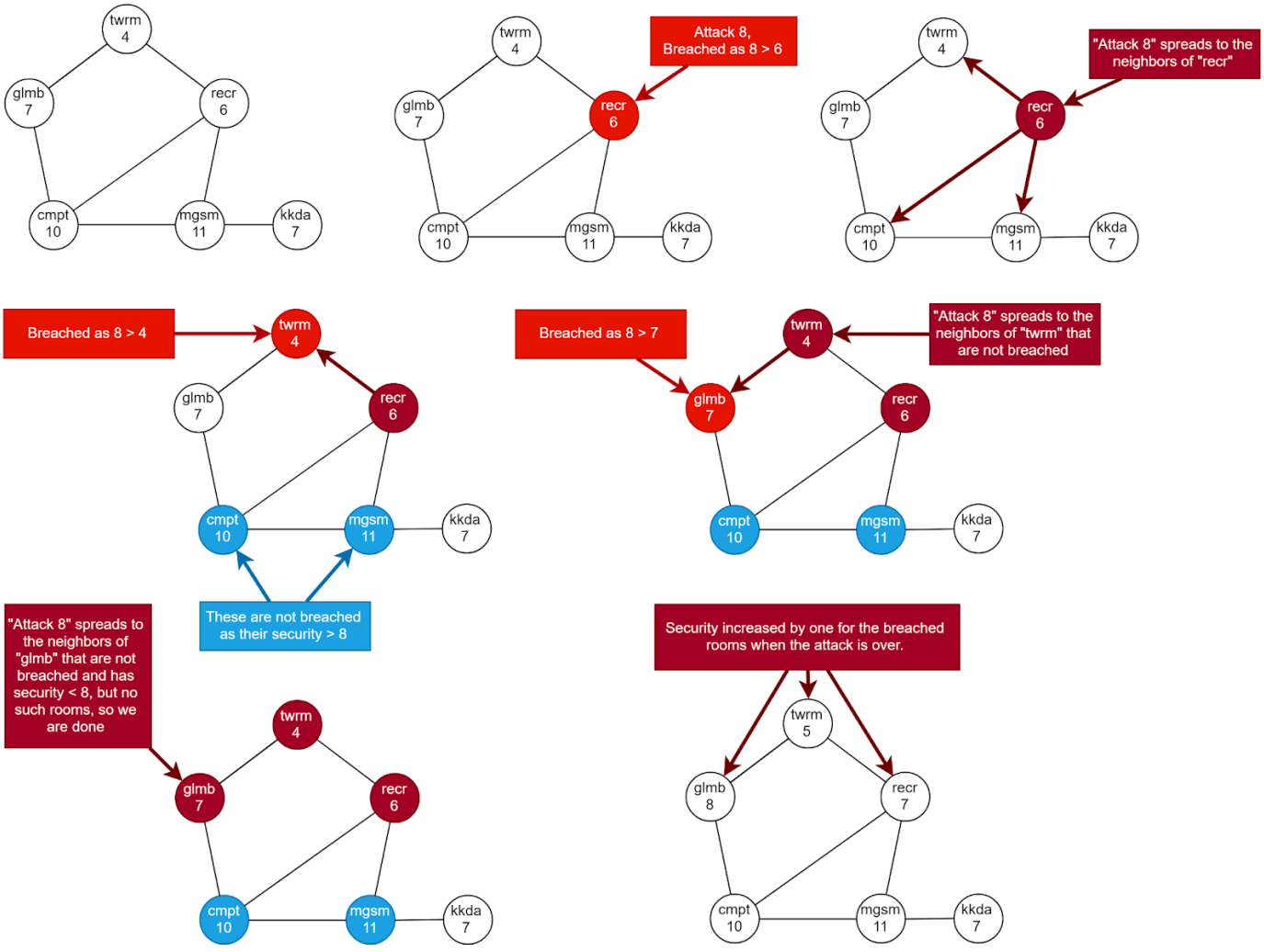
cmpt recr
cmpt glmb
glmb twrm
twrm recr
mgsm cmpt
mgsm recr
kkda mgsm

Note that connected room pairs work both ways, if room A is connected to room B, then room B is connected to room A.

The Facility class should include a **void securityAttack(string roomName, int attackLevel)** method that starts a security attack on a specific room with the given attack level. If the attack level is strictly greater than the security level of a room, we say that this room's security is breached. The attack spreads into the neighboring rooms as long as it is strictly greater than the security level of the current room and the neighboring rooms are not yet breached.

You should implement the spread of the attack using a recursive method, **securityAttack** can act as a helper method to start the recursion. Keep a Boolean variable for each room about whether it is breached or not. Print out the names of the rooms that are breached. You can do the printing during recursion or after the recursion finishes using an iterative approach, but spread of the attack should be done recursively.

For example, using the facility information above, **securityAttack("recr", 8)** should print *{"recr", "twrm", "glmb"}*. Order of the room names may change based on your implementation. This attack is illustrated on the next page. After the attack ends, Increase the security level of the rooms that are breached by 1. This should better protect the rooms during upcoming attacks. We also want to reset the breached status of the rooms after the attack.



Facility class should also include a void **simulateAttacks(int numberOfTries)** method to simulate increasingly higher-level attacks on randomly selected rooms, starting from attack level 1. For example, if the given number of tries is 4, we would like to start an attack of level 1 in a randomly selected room. After the attack is complete, some rooms may have increased security. Then we will start an attack of level 2 in a randomly selected room and so on. Each simulated attack would cause some rooms to increase their security levels.

At the end, the **simulateAttacks** method should report the rooms whose security levels change. You need to print the name of the room, its starting security level (before all the simulations), and its final security level (after all the simulations). You are free to include additional variables and methods to Room and Facility classes to support the required functionality.

Suppose we call **simulateAttacks(7)** which causes the following attacks on the randomly selected rooms:

- securityAttack("cmpt",1)
- securityAttack("mgsm",2)
- securityAttack("recr",3)
- securityAttack("cmpt",4)
- securityAttack("kkda",5)
- securityAttack("twrm",6)
- securityAttack("recr",7)

Among these attacks, securityAttack("twrm",6) will cause breach on "twrm", increasing its security from 4 to 5. Then, securityAttack("recr",7) will cause breach on "recr" and "twrm", increasing their securities from 6 to 7 and 5 to 6, respectively. At the end of **simulateAttacks(7)** we need to print:

```
The following rooms have security change:
twrm: 4 to 6
recr: 6 to 7
```

After the **simulateAttacks** method, the original security levels of the rooms should not change, as this is just a simulation and not a real attack. To do this, you may work on a separate copy of the current facility using the **securityAttack** method. Then you would report the rooms whose security levels are different in the original facility and its clone. Alternatively, you may include a simulated security level variable into the Room class to use it with an alternative securityAttack method. For example, this alternative securityAttack method could function using the simulated security level of the rooms. In the end, the important thing is that the simulation should not alter the original security levels of the rooms, it should only report the resulting security levels after the simulated attacks. Also include a method to print all the rooms and their current security levels for debugging.

Preliminary Submission: You will submit an early version of your solution before the lab. This version should at least include the following:

- Room class should be completed.
- The methods importFacility and securityAttack of the Facility class should be completed.
- Test whether your implementation works by importing the facility from a file and starting attacks at different rooms with different levels.

You will have time to complete your solution in the lab. Do not forget to make your final submission at the end. Even if you finish the assignment in the preliminary submission, you should submit for the final submission on Moodle.

Not completing the preliminary submission on time results in 50% reduction of this assignment's final grade.