Exercise doors

Problem

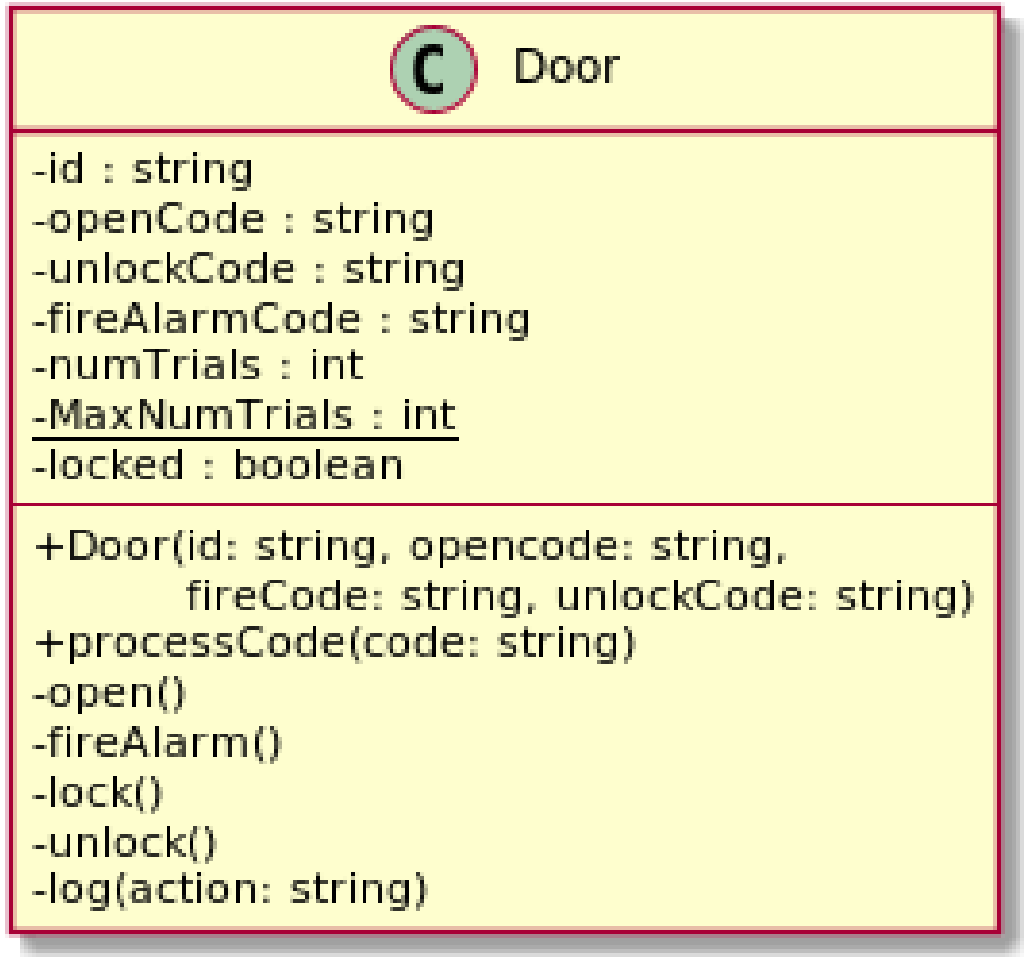
We are programming an access controller for several doors. Each door has a numeric keypad to enter a code of 4 digits. We want them to have the following behaviour:

- **open** the door if the open code for that door is entered and the door is not locked
- **lock** the door after 3 failed attempts to enter the open or fire alarm code, so that it can not be opened any more until an
- **unlock** code is entered
- if the door is not locked and a fire alarm code is entered, **open the door and fire an alarm** at some private surveillance station (in case we are being threatened to open the door)
- **log** (print) every entered code, associated door id plus date and time and door state (locked or not)

Some precisions:

- each door has its own codes
- these 5 capabilities are the most a door can do
- we want a door to have any valid subset of them, like just open and log
- want to be able to change behaviour of a door at execution time

Simple "solution"



Simple but all doors have **all the 5 capabilities, always**.

```
public class Door {
    private static int MaxNumTrials = 3;
    private String id;
    private int numTrials;
    private boolean locked;
    private String codeOpen;
    private String codeUnlock;
    private String codeFireAlarm;

    public Door(String id, String codeOpen, String codeFire, String codeUnlock) {
        this.id = id;
        this.codeOpen = codeOpen;
        this.codeFireAlarm = codeFire;
        this.codeUnlock = codeUnlock;
        numTrials = 0;
        locked = false;
    }
```

```
    public void processCode(String code) {
        log("entered code " + code);
        if (!locked) {
            if (code.equals(codeOpen)) {
                numTrials = 0;
                open();
            } else if (code.equals(codeFireAlarm)) {
                numTrials = 0;
                fireAlarm();
                open();
            } else { // not a valid code when unlocked
                numTrials++;
                if (numTrials==MaxNumTrials) {
                    lock();
                }
            }
        } else { // door is locked
            if (code.equals(codeUnlock)) {
                unlock();
            } else {
                // do nothing, invalid unlock code
            }
        }
    }
```

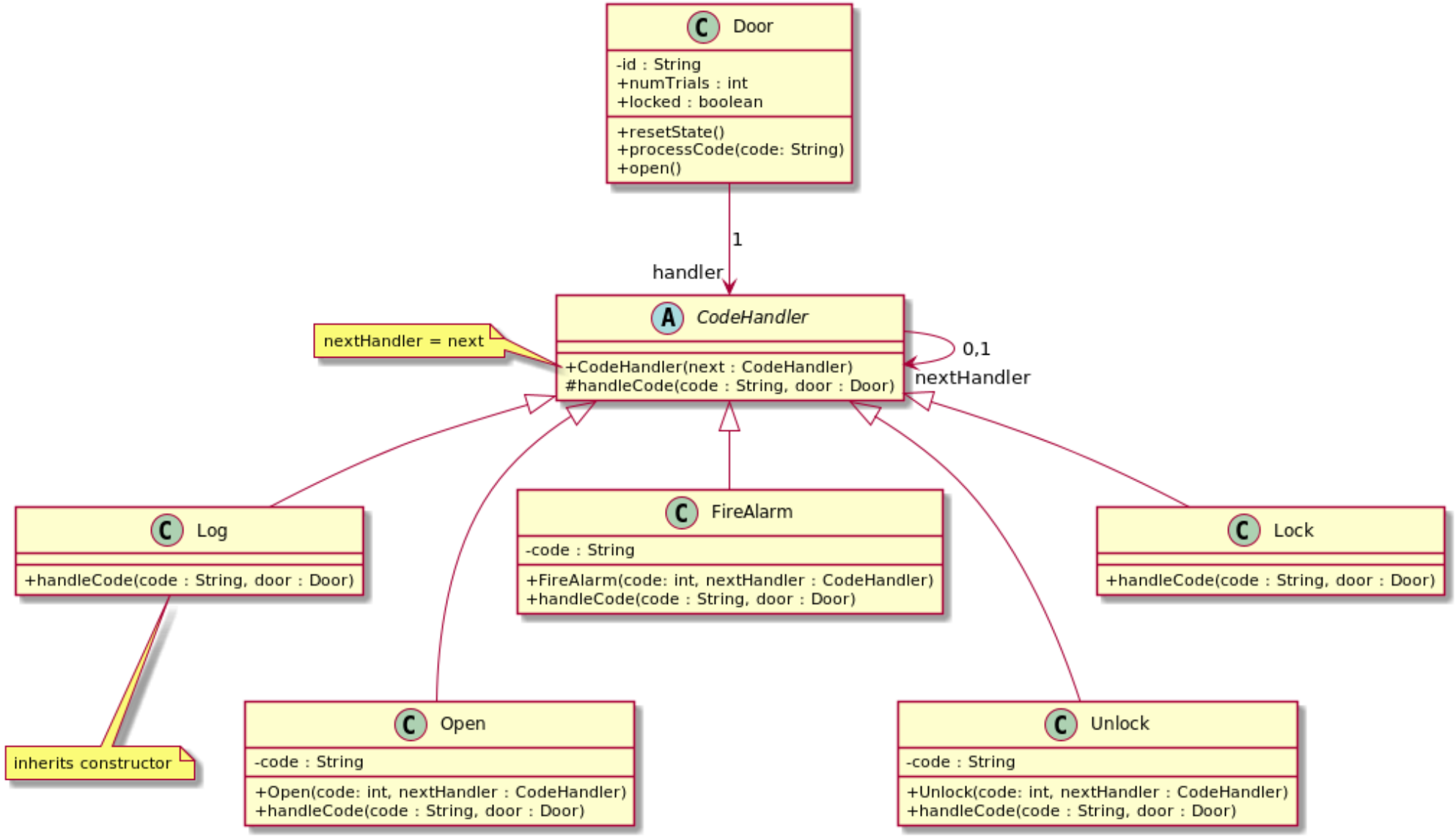
```
    private void log(String action) {
        String isLocked = locked ? "is locked, " : "";
        System.out.println("door " + id + " " + isLocked + action
                + " num. trials left " + (MaxNumTrials - numTrials)
                + ", at " + LocalDateTime.now());
    }

    private void open() {
        // sends pulse to door's electromagnet so that door can be open
        log("open");
    }

    private void fireAlarm() {
        // sends message to control station
        log("fire alarm");
    }

    private void lock() {
        locked = true;
        log("lock");
    }

    private void unlock() {
        numTrials = 0;
        locked = false;
        log("unlock");
    }
```

Advantage : **simplicity**

Problem: it is not what we want because it's **inflexible**

1. all doors behave the same way: have open, lock, unlock, fire alarm and log capabilities
2. we would like to be able to configure how *each particular door* behaves:
    - `door1` just open, log, doesn't lock after 3 trials to open it
    - `door2` open + fire alarm only
    - `door3` does everything
    - ...
3. the behavior of a door is always the same, can not be changed at execution time

Solution: **chain of responsibility**



- `door` as parameter of `handleCode()` because we may need its `id` or `open()`
- door state `numTrials` and `doorLocked` attributes shared by all handlers

class Door

```
public class Door {
    private String id;
    private CodeHandler codeHandler;

    public Door(String ident, CodeHandler ch) {
        id = ident;
        codeHandler = ch;
    }
    public void processCode(String code) { codeHandler.handleCode(code, this); }
    public void open() { System.out.println("door " + id + " opened"); }
    public String getId() { return id; }
    public void setCodeHandler(CodeHandler ch) { codeHandler = ch; }
}
```

- `Door` doesn't have anymore the codes and what to do for each code
- a door instead has a (sequence of) handler
- the chain of resp. request is to process the entered code
- order of handlers matters! not all combinations work as we intend

Client code

```
public class Client {
        public static void main(String[] args) throws IOException {

    String openCode = "1111";
    String fireAlarmCode = "2222";
    String unlockCode = "3333";

    CodeHandler chain1 =
        new Log(new Unlock(unlockCode,
            new FireAlarm(fireAlarmCode,
                new Open(openCode,
                    new Lock(null)))));
    CodeHandler chain2 = new Log(new Open(openCode, null));
    CodeHandler chain3 =
        new Log(
            new FireAlarm(code_fire_alarm,
                new Open(code_open, null)));

    Door d1 = new Door("d1", chain1);
```

```
    d1.processCode("1111"); // opens
    d1.processCode("2222"); // opens and fires alarm
    d1.processCode("1234"); // first trial
    d1.processCode("4321"); // second trial
    d1.processCode("5555"); // third trial, gets locked
    d1.processCode("6666"); // invalid unlock code
    d1.processCode("7777"); // invalid unlock code
    d1.processCode("1111"); // invalid unlock code
    d1.processCode("3333"); // valid unlock code, now can be opened or fire alarm
    d1.processCode("2222"); // opens and fires alarm

    // change behaviour of door d1 at run time
    d1.setCodeHandler(chain2);
    BufferedReader stdin = new BufferedReader(
        new InputStreamReader(System.in));
    while (true) {
        System.out.print("Input code : ");
        String inputCode = stdin.readLine();
        d1.processCode(inputCode);
    }
```

Sample handler

```
public class Open extends CodeHandler {
    private String codeOpen;
    public Open(String code, CodeHandler nextHandler) {
        super(nextHandler);
        codeOpen = code;
    }
    @Override
    public void handleCode(String code, Door door) {
        System.out.println("handle Open");
        if (!doorLocked) {
            if (codeOpen.equals(code)) {
                resetDoorState();
                door.open();
            } else {
                numTrials++;
                System.out.println(numTrials + " trials");
                super.handleCode(code, door);
            }
        } else { /* void, can't open until unlocked */ }
    }
}
```

Consequences

- programmers must take care when building the chain: order of handlers matters !
- we gain in flexibility : independent door behaviour, change it at run time
- design open to new handlers = add new behaviour to doors
- but the price is increased complexity

What to do

- implement the design (you can change it if needed)
- write test function and trace execution with prints
- show it works for `chain1`: run and copy output to console to a file and comment it
- same for `chain2`, `chain3`

Deliverables

A .zip file containing:

1. text file named `autors.txt` with name and NIU of authors responsible of this work
2. output to console for the three chains
3. a zip file with the project, including the plantUML file and a PNG image