I4SWT1 Exercise: Fakes 2 (primarily mocks)

In this exercise, you will create the core component of a *Door Control* system, which, by means of some sort of identification, controls the opening and closing of a door. You will test the components of the system using fakes, both stubs and mocks, as appropriate to ensure that the system works as advertised. Then – as always – you will set up a Jenkins project which must run build and unit tests.

A Door Control system controls the opening and closing of a door. When a user requests that the door be opened, he identifies himself with an ID (by entering a code, swiping a card, performing a retina scan, …). The system consults a user validation mechanism to check if the user may be granted access. If so, the system notifies the user, opens the door, and waits for it to close again before any more requests can be handled. The sequence diagram below depicts the main scenario, *Entry Granted*:

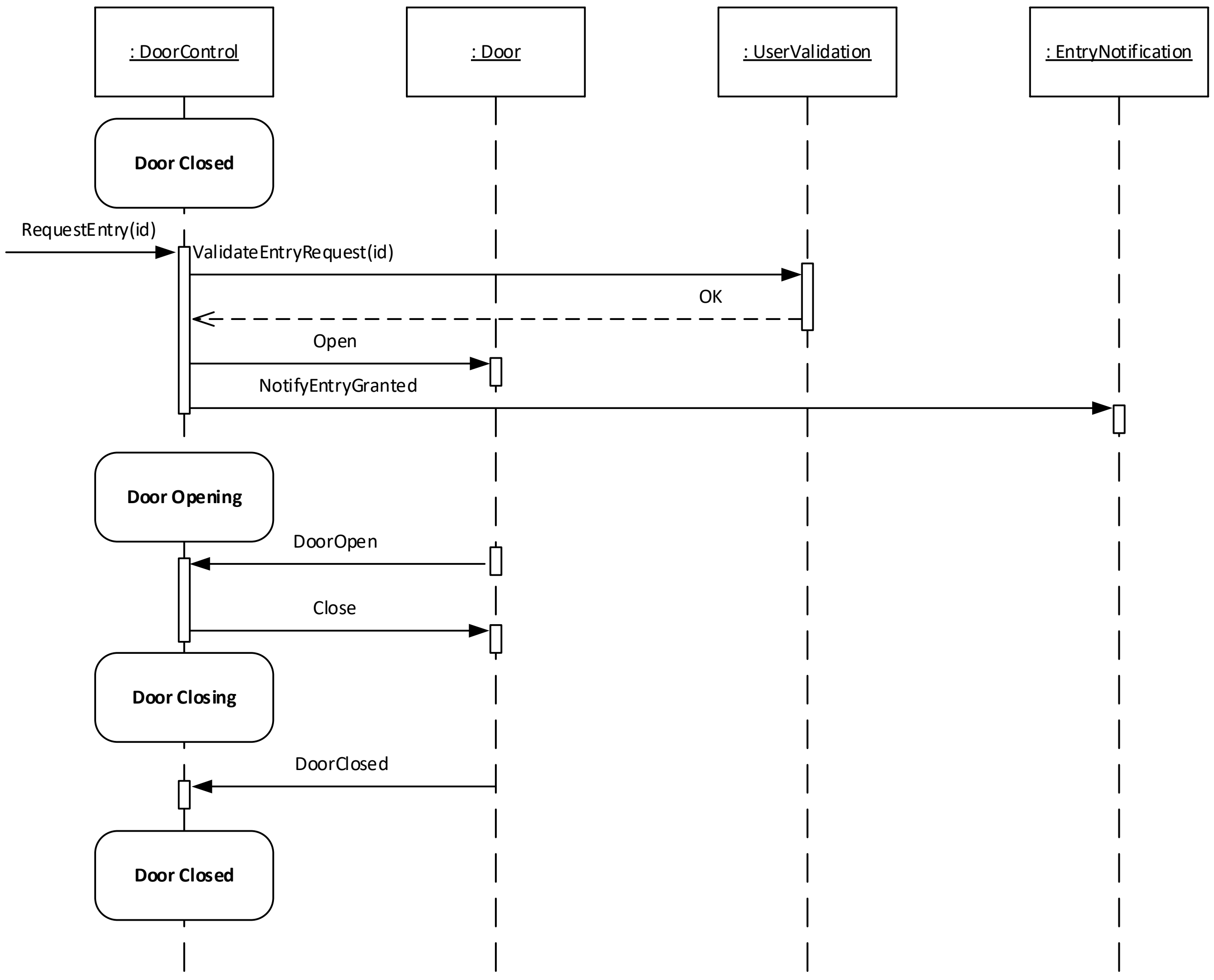

Figure 1: Main scenario: "Entry Granted"

Furthermore, two exception scenarios have been identified: *Entry Denied* and *Door Breached*, the latter when the door is opened without the prior consent of the system. The sequence diagram for each of these two exceptions is shown below:

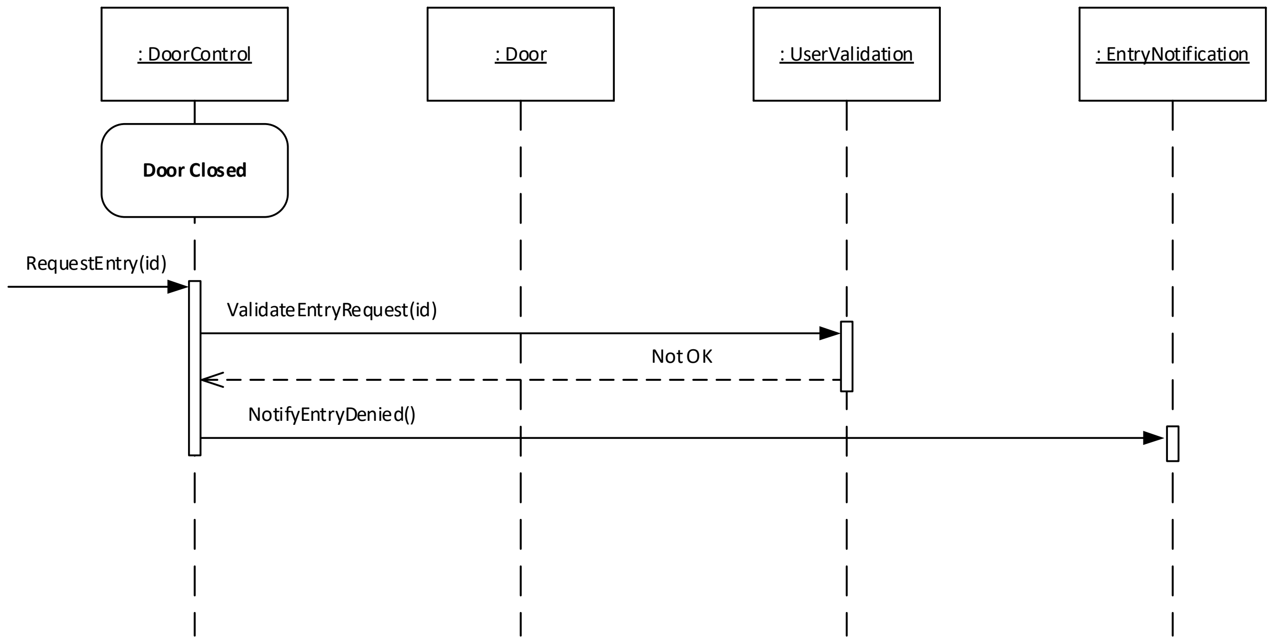

Figure 2: Sequence diagram for exception scenario "Entry denied"

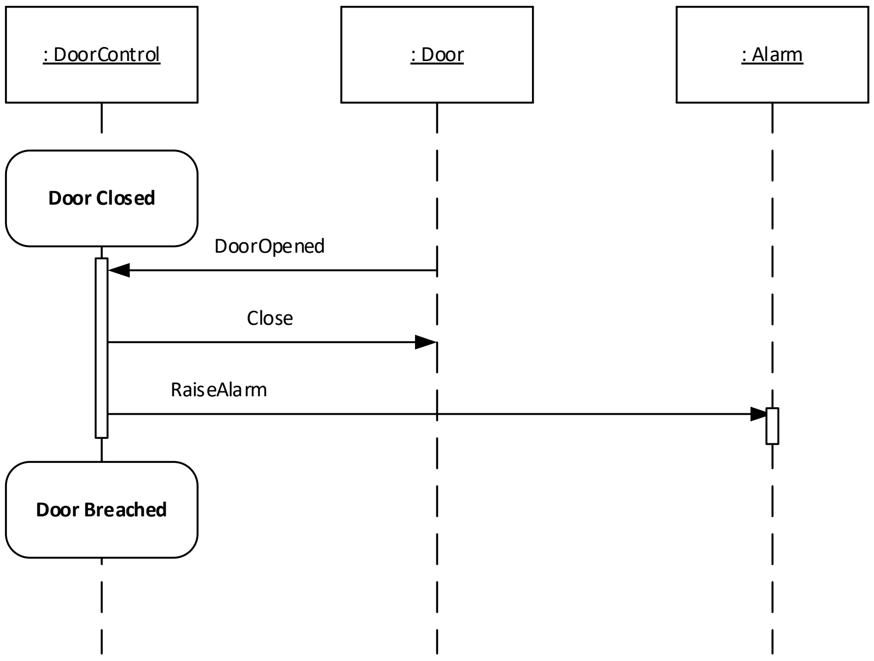

Figure 3: Sequence diagram for exception scenario "Door breached"

Exercise 1:
Analyse the system to make sure you understand each of the given scenarios:

- Draw a UML State Machine Diagram (STM) state chart for class `DoorControl`
- Draw a class diagram for the complete system
- Check and change the class diagram as necessary to make the design testable

Exercise 2:
Design, implement and test class `DoorControl` according to your testable design from the previous question, so that it satisfies the sequence given in Figure 1. You should not implement any other classes than `DoorControl` - use fakes as necessary to satisfy the dependencies of `DoorControl.`

NOTE: It is important that you think about how you can test class `DoorControl` *without* exposing any class-internal variables or properties only for the sake of testing. For example, it is tempting **but wrong** to define a variable or property `currentState` in `DoorControl` which holds the current state (`DoorOpen`, `DoorClosed`, …) in the STM for DoorControl. If you do so, you are making a white-box test, and merely renaming the states will break your test suite.

Exercise 3:
If you did not already do so, place your solution under version control, publish it to GitHub and setup a Jenkins job, that will pull, build and run the tests.

Exercise 4:
Working in parallel as you prefer, extend your implementation and test of class `DoorControl` so that it can also handle the exceptions given in Figure 2 and Figure 3. Again, test your solution using fakes. Share your results using GitHub and check that your tests are running on Jenkins.