Web service homework

Dr. Balázs Simon (sbalazs@iit.bme.hu), BME IIT, 2017

In the following text use your own Neptun code with capital letters instead of the word “NEPTUN”.

1 Description of the task

The task is to create and publish a SOAP web service for cinema seat reservations.

The interface of the service is described by the following pseudo-code:

```
[Uri("http://www.iit.bme.hu/soi/hw/SeatReservation")]
namespace SeatReservation
{
    exception CinemaException
    {
        int ErrorCode;
        string ErrorMessage;
    }

    struct Seat
    {
        string Row;
        string Column;
    }

    enum SeatStatus
    {
        Free,
        Locked,
        Reserved,
        Sold
    }

    interface ICinema
    {
        void Init(int rows, int columns) throws CinemaException;
        Seat[] GetAllSeats()throws CinemaException;
        SeatStatus GetSeatStatus(Seat seat) throws CinemaException;
        string Lock(Seat seat, int count) throws CinemaException;
        void Unlock(string lockId) throws CinemaException;
        void Reserve(string lockId) throws CinemaException;
        void Buy(string lockId) throws CinemaException;
    }
}
```

The operations of the service have the following responsibilities:

- Init:
  - initializes the room with the given number of rows and columns
  - number of rows: 1 <= *rows* <= 26
  - number of columns: 1 <= *columns* <= 100
  - all the seats are free, and every previous lock or reservation is deleted
  - if the number of rows or columns is outside of the given interval, a CinemaException must be thrown
- GetAllSeats:
  - returns the number of seats in the room
  - the rows are denoted by consecutive capital letters of the English alphabet starting from the letter 'A'
  - columns are denoted by consecutive integer numbers starting from 1
- GetSeatStatus:
  - returns the status (free, locked, reserved, sold) of the given *seat*
  - if the position of the given seat is invalid, a CinemaException must be thrown
- Lock:
  - locks *count* number of seats in the row starting from the given *seat*
  - if the locking cannot be performed (e.g. there are not enough remaining seats in the row or there are not enough free seats), a CinemaException must be thrown, and no seats should be locked
  - the operation must return a unique identifier based on which the service can look up the locked seats
- Unlock:
  - releases the lock with the given identifier
  - every seat belonging to this lock must be freed
  - if the identifier for the lock is invalid, a CinemaException must be thrown
- Reserve:
  - reserves the seats of the lock with the given identifier
  - if the identifier for the lock is invalid, a CinemaException must be thrown
- Buy:
  - sells the seats of the lock or reservation with the given identifier
  - if the identifier is invalid, a CinemaException must be thrown

2 The service

The service should be implemented in a Maven web application similar to the one in the tutorial. The pom.xml must be the attached **service/pom.xml**, but in this file the word NEPTUN should be replaced with your own Neptun code.

The name of the web application should be: **WebService_NEPTUN**

The service should listen on the following URL:

`http://localhost:8080/WebService_NEPTUN/Cinema`

The service has to store data between calls. In a real world application the storage should be a database. In the homework, the storage should be *static variables*.

3 The client

The service should be implemented in a Maven console application similar to the one in the tutorial. The pom.xml must be the attached **client/pom.xml**, but in this file the word NEPTUN should be replaced with your own Neptun code.

The name of the application should be: **WebServiceClient_NEPTUN**

The main class of the application should be: **cinema.Program**

The client receives four parameters:

`java cinema.Program [url] [row] [column] [task]`

The meaning of the parameters is:

- [url]: the URL of the service to be called
- [row]: the row of the seat
- [column]: the column of the seat
- [task]: what to do with the seat

The possible values of [task]:

- Lock: the seat should be locked – a single Lock() call to the service
- Reserve: the seat should be locked and then it should be reserved – a Lock()+Reserve() call to the service
- Buy: the seat should be locked and then it should be bought – a Lock()+Buy() call to the service

Examples for executing the client:

```
mvn exec:java -Dexec.mainClass=cinema.Program
    -Dexec.args="http://localhost:8080/WebService_NEPTUN/Cinema A 4 Lock"
mvn exec:java -Dexec.mainClass=cinema.Program
    -Dexec.args="http://localhost:8080/WebService_NEPTUN/Cinema H 31 Reserve"
mvn exec:java -Dexec.mainClass=cinema.Program
    -Dexec.args="http://localhost:8888/WebService_SB_Test/Cinema D 12 Buy"
```

This client always handles a single seat. The program should call only the Lock(), Reserve() and Buy() operations in the given order. It must not call any other operations! (Other client programs can be written for testing, but they should not be handed in.)

4 To be uploaded

A single ZIP file has to be uploaded. Other archive formats must not be used!

The root of the ZIP file must contain two folders:

- **WebService_NEPTUN:** the Maven application of the service
- **WebServiceClient_NEPTUN:** the Maven application of the client

The compiled class files (target folder) may be omitted from the ZIP file.

The service application must compile from the command line using the following command:

```
...\WebService_NEPTUN>mvn package
```

The service application must deploy to the server from the command line using the following command:

```
...\WebService_NEPTUN>mvn wildfly:deploy
```

The client application must compile from the command line using the following command:

```
...\WebServiceClient_NEPTUN>mvn package
```

The client application must run from the command line using the following command (the last four parameters are only examples):

```
...\WebServiceClient_NEPTUN>mvn exec:java -Dexec.mainClass=cinema.Program
            -Dexec.args="http://localhost:8080/WebService_NEPTUN/Cinema H 31 Reserve"
```

The service must implement the attached **wsdl/SeatReservation.wsdl** and **wsdl/SeatReservation.xsd**. Modifying this interface is forbidden! The client must be able to call any service implementing this interface.

The solution must compile and run in the given environment using the given pom.xml files. These pom.xml files must not be modified apart from replacing the word NEPTUN with your own Neptun code.

Important: the instructions and naming constraints described in this document must be precisely followed!

Again: use your own Neptun code with capital letters instead of the word “NEPTUN”.