Homework 2

Mattias Cederlund, mcede@kth.se

Services

The web services were implemented using Netbeans 8.0.2, with Java 8.

I started off using the bottom-up approach to create the authorization service and the itinerary finding services. The authorization service will take a username and a password and if it matches against a pre-defined list of users, it will return a token used to provide access to the other services in the system. The itinerary finding service will take departure city, destination city and an optional date and will return a list of possible itineraries. Services for finding price of available itineraries were not implemented as web services, instead all the required information was supplied in the data structures and could easily be calculated on the client.

User and flight data are generated and kept in a static class to make it easily accessible across the entire web service. This class also contains all itinerary finding logic.

Booking and issuing of tickets were implemented in a top-down fashion, creating the WSDL schema and using it to generate the Java classes. In the WSDL document operations bookTicket and issueTicket were defined, with their input and output messages and their parameters. The parameters were defined using an external XML schema. bookTicket takes an itinerary and a cardNumber, creates a ticket stored on the server and returns the ticket number. issueTicket will accept this ticket number as an input parameter and will return the full ticket in response.

Creating the WSDL with the operations and defining their parameters etc. went fine. The problems actually arose after importing it to Netbeans and generating the source code. First off I could not find any way to make the top-down services use the same model classes that I defined in the bottom-up implementation. This would have been appropriate since both services share the same storage. It may have been easier if I started with the top-down approach and used those classes when creating the bottom-up services.

Also the way Netbeans was generating classes from the WSDL and schema documents made it impossible to clean the project. When I did, it would remove the generated classes which would make the web service implementation miss them. Then the entire project (including the auto-generated classes) wouldn't build because it was referring to classes that couldn't be found in a circle-like fashion.

I also had some problems with deploying the server and making sure that both the bottom-up and top-down web services were deployed. Often the top-down (generated) web services wouldn't get deployed. I solved this by creating a new project for the top-down services.

Lastly the web services that were auto-generated by Netbeans would have its parameters set to null when it was invoked. It turned out the generated methods were missing required annotations that I had to add manually to make it work.

Architecture

This homework is divided in three projects, HW2BottomUpServer, HW2TopDownServer and HW2Client.

The server projects contains one web service each, bottom-up and top-down. Both web services use an external class for storing its data (and some application logic), FlightsDataService and TicketService.

The client is a regular command-line java program which works as a client to both the web services.

It contains test cases for each individual service, as well as a test case for the entire interaction from authorization to ticket issuing.

Running instructions

The source code is provided as Netbeans projects. Import both the server and client projects to Netbeans using File > Import Project > From ZIP.

To run, deploy the server projects to GlassFish server. When it finished deployment, run the client project as a regular Java application and observe its progress in the console. If you wish to run the client with different parameters (destinations, dates etc.) feel free to edit the constants in HW2Client.

Executables are also provided in /Executables (HW2BottomUpServer.war, HW2TopDownServer.war and HW2Client.jar) Running the services this way is although not tested.

SOAP Headers

All messages (except authorization) requires an authorization token that is sent as a parameter:

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <S:Body>
        <ns2:getItinearies xmlns:ns2="http://hw2.id2208.mattec.se/"
xmlns:ns3="http://topdown.hw2.id2208.mattec.se/">
            <authToken>SecretTokenThatAllowsAccess</authToken>
            <departureCity>Stockholm</departureCity>
            <destinationCity>Paris</destinationCity>
            <date>2015-02-05</date>
        </ns2:getItinearies>
    </S:Body>
</S:Envelope>
```

Instead the authorization information can be put in the header as such (or by authToken):

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header>
            <a:authorizationInformation xmlns:a="http://hw2.id2208.mattec.se/">
                  <a:username>mattias</a:username>
                  <a:password>muchSecretPassword</a:password>
            </a:authorizationInformation>
      </SOAP-ENV:Header>
    <S:Body>
        <ns2:getItinearies xmlns:ns2="http://hw2.id2208.mattec.se/"
xmlns:ns3="http://topdown.hw2.id2208.mattec.se/">
            <departureCity>Stockholm</departureCity>
            <destinationCity>Paris</destinationCity>
            <date>2015-02-05</date>
        </ns2:getItinearies>
    </S:Body>
</S:Envelope>
```