

**Sofia University**  
**Department of Mathematics and Informatics**

Course : **Programming IT services**

Date: January 20, 2015

Student Name:

**Lab No. 15 (Web services)**

**Submit the all Java files developed to solve the problems listed below. Use comments and Modified-Hungarian notation.**

**Problem 1.**

Complete tutorial [Getting Started with JAX-WS Web Services](#)

**Problem 2.**

Modify the client in RMISample1 from Lecture 14b to use:

- a) a JAX-WS web service
- b) a XML RESTful web service
- c) a JSON RESTful web service

Create separate Netbeans Java Web applications for the web service in each one the above cases and respective Java applications for the client.

**Problem 3**

**Write a RESTful web service application and a respective JFrame client application**, where a client sends loan information (annual interest rate, number of years and loan amount) to the server. The server computes monthly payment and total payment and sends them (as a formatted string) back to the client. Use JSON to represent data returned by the web service. Consume the web service in a **Swingworker Thread**.

**Problem 4.**

(*Phone Book Server*) Create a **phone book SOAP web server** that maintains a **file of names** and **phone numbers**. Define the following **web service methods**:

```
public PhoneBookEntry[] getPhoneBook()  
public void addEntry( PhoneBookEntry entry )  
public void modifyEntry( PhoneBookEntry entry )  
public void deleteEntry( PhoneBookEntry entry )
```

Class **PhoneBookEntry** should contain **String instance variables** that represent the **first name**, **last name** and **phone number** for one person. The class should also provide appropriate **set/get**

methods and perform **validation** on the **phone number format**. Use a **HashMap** to store **PhoneBookEntry** objects employing their **phone number** as a key

**Class PhoneBookClient** should **provide a JFrame user interface** that allows the **user to scroll through entries**, add a **new entry**, **modify an existing entry** and **delete an existing entry**. The client and the **server should provide proper error handling** (e.g., the client cannot modify an entry that does not exist). . Consume the web service in a **Swingworker Thread**.

#### Problem 5.

**Write a SOAP web service application and a JFrame client program.** Consume the web service in a **Swingworker Thread**. The data sent from the client comprises the radius of a circle and the result produced by the server is the area of the circle as shown below.



#### Problem 6.

Learn to consume free registered web services available in Netbeans and in catalogs on the web.

- a) Complete tutorial [SmugMug Client in Java with NetBeans IDE](#)
- b) Complete tutorial [Developing JAX-WS Web Service Clients](#)
- c) Complete tutorial [Create a weather forecast SOAP web service client](#)

#### Problem 7.

Complete the following tutorial to learn how to use the Strikelron Web services published in Netbeans in order to validate an email address:

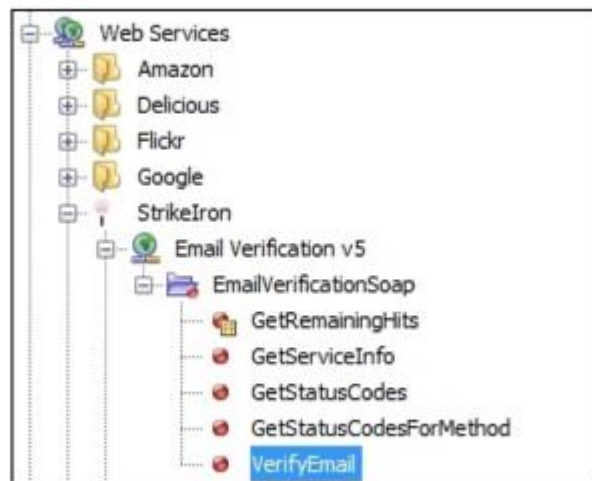
To validate an e-mail address using Strikelron, perform the following steps:

1. Click on **File** and then on **New Project...** to create a new NetBeans project. Create a Java application called `EmailValidator` ensuring that a main class called `com.davidsalter.cookbook.email.EmailValidator` is created.

### Tip

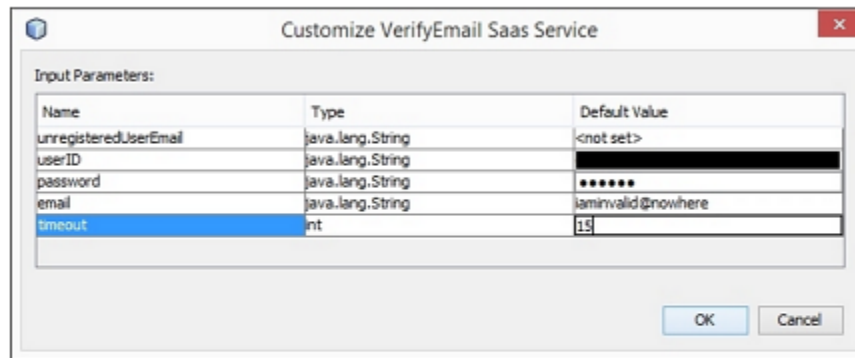
If you are having trouble creating a new project, check out the *Creating a Java application* recipe in [Chapter 1, Using NetBeans Projects](#), of this book.

2. Ensure that the `EmailValidator.java` file is open for editing.
3. Click on the **Services** explorer and expand the **Web Services** node. Locate the **Strikelron** node and expand this to show the **Email Verification v5** service. Expand this node and the **EmailVerificationSoap** node within it, as shown in the following screenshot:



4. Locate the **Verify Email** web service underneath the **Email Verification v5** web services and drag it into the `main` method of the `EmailValidator.java` class.
5. The **Customize VerifyEmail Saas Service** dialog will be displayed. Enter the following information into the dialog:
  - **userID:** Your **Strikelron** user ID
  - **password:** Your **Strikelron** password
  - **email:** `iaminvalid@nowhere`
  - **timeout:** `15`

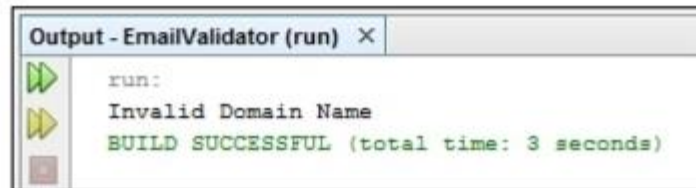
We can see the **Customize VerifyEmail Saas Service** dialog in the following screenshot:



6. Click on the **OK** button.
7. NetBeans will now add the code to the `EmailValidator.java` class to query the e-mail address we entered (`iaminvalid@nowhere`).
8. We need to change the generated code within the `main()` method within the `EmailValidator.java` class in order to see the results of the API call. Immediately after the call to `port.verifyEmail(...)`, add the following code:

```
System.out.println(verifyEmailR
result.value.getServiceStatus().
getStatusDescription());
```

9. We've now entered all of the information we need, so press **F6** to run the application and validate the e-mail address.
10. The application will launch, and a message will be displayed in the **Output** window indicating that the e-mail address has an invalid domain name (shown in the following screenshot) —Strikelron is telling us that the e-mail address is invalid:

A screenshot of the NetBeans IDE's Output window. The title bar reads "Output - EmailValidator (run) X". The window contains three lines of text: "run:" on the first line, "Invalid Domain Name" on the second line, and "BUILD SUCCESSFUL (total time: 3 seconds)" on the third line. To the left of the text are three icons: a green double arrow, a yellow double arrow, and a red square.

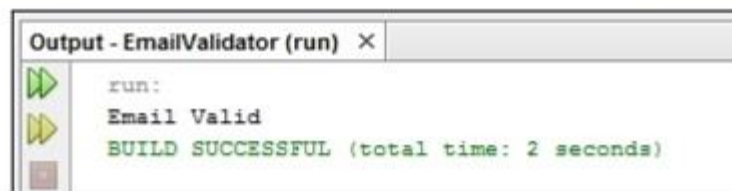
```
run:
Invalid Domain Name
BUILD SUCCESSFUL (total time: 3 seconds)
```

Now, let's change the code so we can see what happens when we try to validate a valid e-mail address with the following steps:

1. Edit the `EmailValidator.java` class and change the email variable to contain your e-mail address. So, for example, if your e-mail address is `cookbookreader@gmail.com`, change the email variable to read:

```
java.lang.String email = "cookbookreader@gmail.com";
```

2. Press *F6* to run the application again, and note that this time Strikelron has successfully validated the e-mail address:

A screenshot of the NetBeans IDE's Output window. The title bar reads "Output - EmailValidator (run) X". The window contains three lines of text: "run:" on the first line, "Email Valid" on the second line, and "BUILD SUCCESSFUL (total time: 2 seconds)" on the third line. To the left of the text are three icons: a green double arrow, a yellow double arrow, and a red square.

```
run:
Email Valid
BUILD SUCCESSFUL (total time: 2 seconds)
```

## How it works...

When validating e-mail addresses with the Strikelron tools, NetBeans creates five variables to hold the input parameters to Strikelron:

- `unregisteredUserEmail`: This variable is a hangover from when Strikelron used to allow unregistered users access to its API by specifying an e-mail address. This field is best left empty.
- `userID`: This is your user ID provided by Strikelron. Most probably, this will be the e-mail address that you registered with at Strikelron.
- `password`: This is your API password that was generated for you by Strikelron when you created an account.
- `email`: This is the e-mail address that you wish to validate.
- `timeout`: This is the timeout period in seconds for querying the Strikelron web services. This has to be within the range of 15 to 120 for the API call to succeed.

Unlike the previous recipes where NetBeans created specific classes for interacting with REST endpoints, NetBeans has used JAX-WS in this recipe to query the SOAP web services we have called. Classes representing the entities used by the web services are defined within the `EmailVerification.jar` library that NetBeans has automatically created and added to our project.

### Tip

For more information on JAX-WS, check out the documentation at <https://jax-ws.java.net/>.

NetBeans automatically generated code to instantiate the web service (`com.strikeiron.EmailVerification service`) and the port (`com.strikeiron.EmailVerificationSoap port`) for us to invoke web service operations.

We then invoked the web service operation to verify the provided e-mail address (`port.verifyEmail(...)`).

Finally, we checked the result of the web service call to establish whether the e-mail address was valid or not. We did this by checking the `statusDescription` field of the web service result.



## There's more...

So far, we've written code to query the **Strikelron** services to validate e-mail addresses. What if we just want to quickly test a **web service** without writing any code? Can this be done?

It certainly can. If we expand the node under the **Email Verification v5 web service** in the **Services** explorer and then right-click on the **Verify Email web service** (rather than dragging it into a class), we get the **Test Method** option. Selecting this option causes the **Test Web Service Method** dialog to be displayed where we can enter different data for the different **web service** parameters and get rapid feedback on how the **web service** works. The **Test Web Service Method** dialog is shown in the following screenshot:

Type	Name	Value
● java.lang.String	unregisteredUserEmail	
● java.lang.String	userID	
● java.lang.String	password	
● java.lang.String	email	
● int	timeout	0
⊕ javax.xml.ws.Holder <com.strike	verifyEmailResult	javax.xml.ws.Holder @bb85769

Submit

Results:

Close Help

In addition to verifying e-mails, **Strikelron** provides many other data and validation services, all of which can be invoked and tested in a similar fashion to the **Email Validation v5 web service** described earlier. This is shown in the following screenshot:



