Practical Lecture 6 Developing a Java Client

Practical Session Structure

1. Introduction
2. Building a business component
3. Building an admin GUI
4. Introducing .NET remoting
5. Creating a web service and client website
6. **Developing a Java client**

2

Overview

- In order to start this session, you need to have completed all of the practical lecture 5
- In this lecture we will build a Java client to be used by team leaders, which makes calls to the web service implemented in the previous session

3

Learning Objectives

- Understand how to create proxies for a web service in NetBeans
- Create a standalone Java application that makes calls on a web service implemented in .NET, thus demonstrating cross-platform interoperability of web services

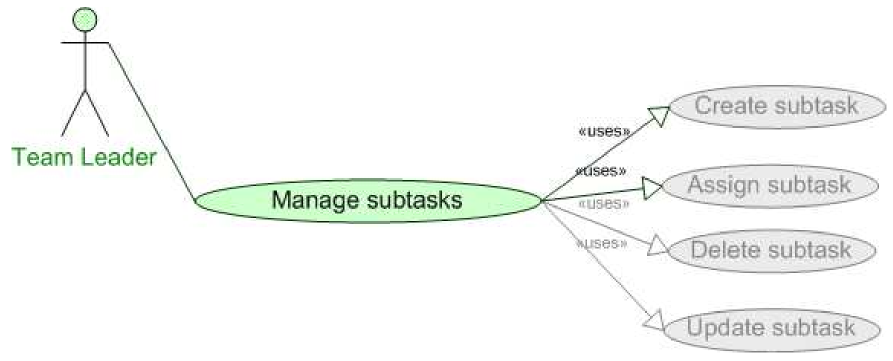
4

Introduction

- In this practical session we will:
  - Implement a GUI application that enables a team leader to log in and see a list of tasks that the project manager (administrator) has assigned to them

5

What are we building?



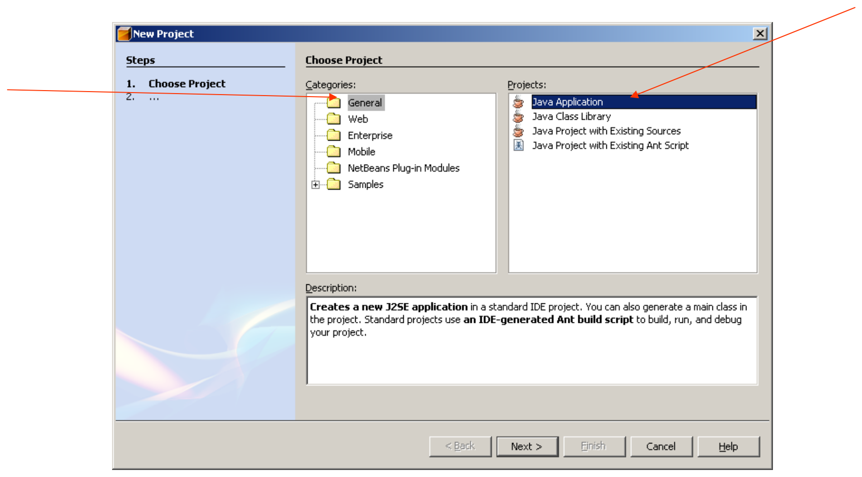
6

Creating the Project

- We will create a new project in NetBeans
  - Start NetBeans
  - Select *File -> New Project*
  - From the *Categories* pane select *General* and from the *Projects* pane select *Java Application*

7

Creating the Project /2



Creates a new J2SE application in a standard IDE project. You can also generate a main class in the project. Standard projects use **an IDE-generated Ant build script** to build, run, and debug your project.

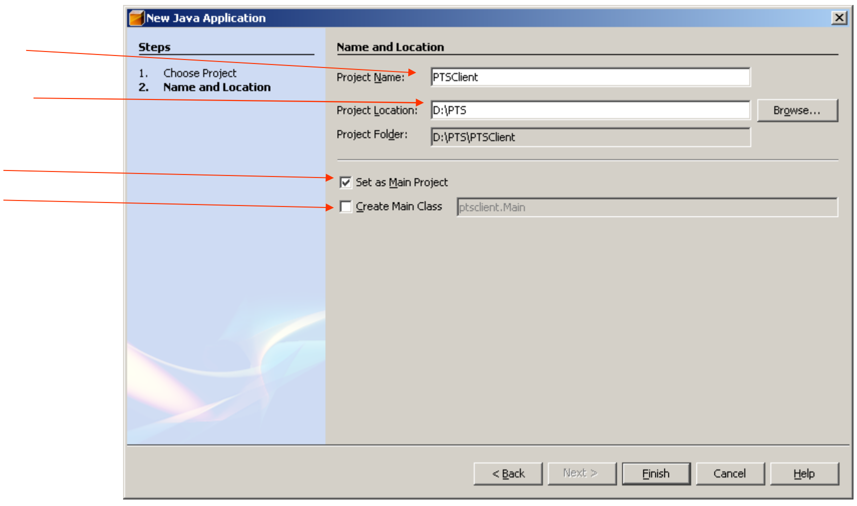
8

Creating the Project /3

- Click *Next* to go to the following screen
- On the next screen:
  - Enter project name as *PTSClient*
  - Select a suitable location
  - Make sure that the *Set as Main Project* checkbox is ticked and *Create Main Class is* unticked

9

Creating the Project /4



New Java Application

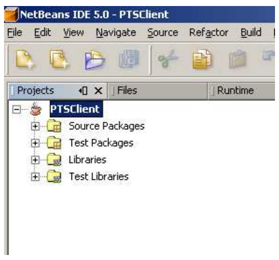
Steps
1. Choose Project
2. **Name and Location**

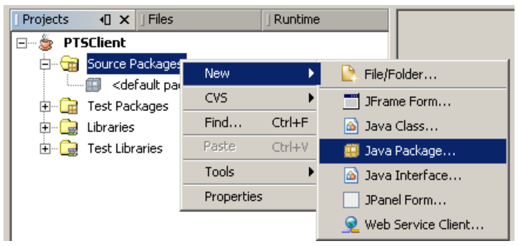
Name and Location
Project Name: PTSClient
Project Location: D:\PTS
Project Folder: D:\PTS\PTSClient
☑ Set as Main Project
☐ Create Main Class ptsclient.Main

10

Creating a Package

- NetBeans will create a project structure
- Let’s create a package for our application
  - Right-click on the *SourcePackages* folder and select *New -> Java Package*

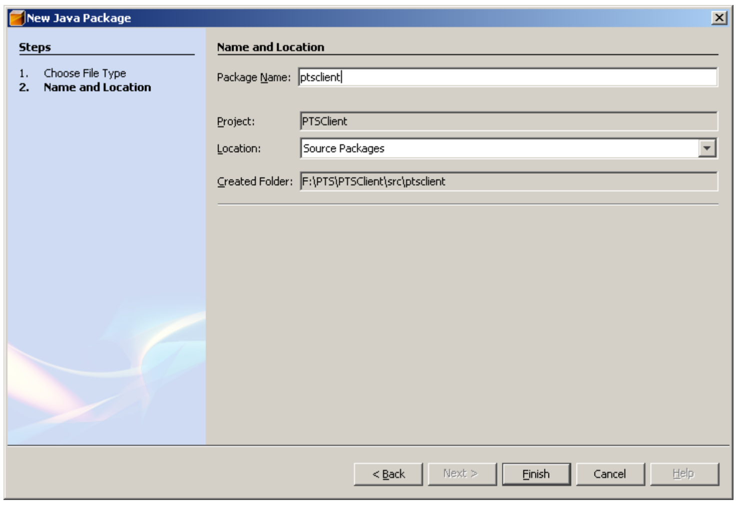




11

Creating a Package /2

- Name the package *ptsclient*


New Java Package

Steps
1. Choose File Type
2. **Name and Location**

Name and Location

Package Name: ptsclient

Project: PTSClient

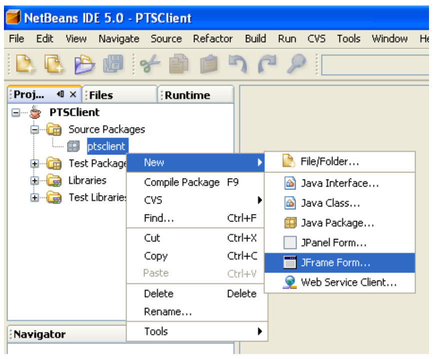
Location: Source Packages

Created Folder: F:\PTS\PTSClient\src\ptsclient

12

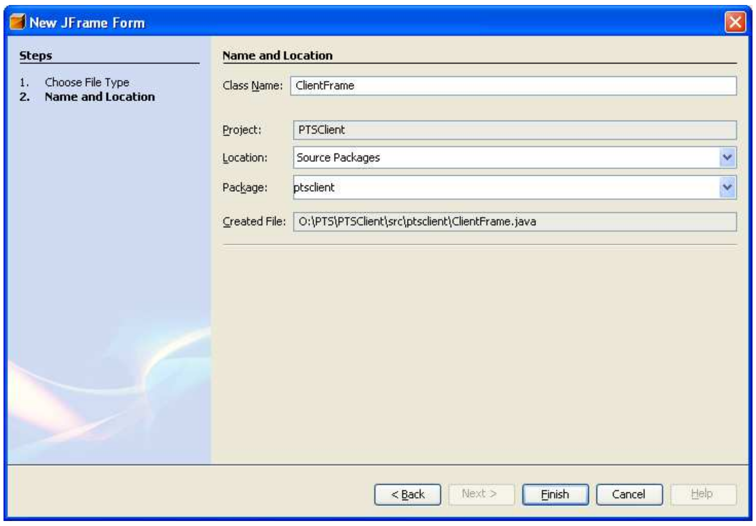
Adding a JFrame

- Let's add a user interface file to the package
  - Right-click on the newly created *ptsclient* package and select *New -> JFrame Form…*
  - Name the class *ClientFrame* and click *Finish*



13

Adding a JFrame /2


New JFrame Form

Steps
1. Choose File Type
2. Name and Location

Name and Location

Class Name: ClientFrame

Project: PTSClient

Location: Source Packages

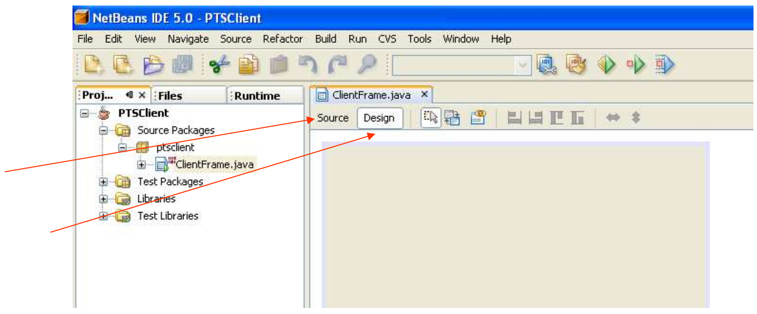
Package: ptsclient

Created File: O:\PTS\PTSClient\src\ptsclient\ClientFrame.java

14

ClientFrame Design

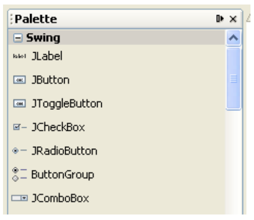
- A *ClientFram.java* file is now created
- Similar as with Visual Studio, NetBeans will allow you to change between the Source and Design view of the form



15

ClientFrame Design /2

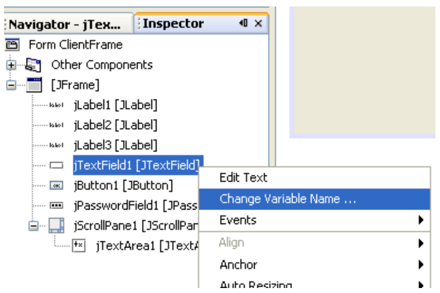
- Open the ClientFrame class in Design view
- From the *Palette* pane (Swing category) drag the following controls onto the form:
  - 3 JLabels with text set to *Username, Password* and *Your projects*
  - 1 JTextField
  - 1 JPasswordField
  - 1 JButton
  - 1 JTextArea

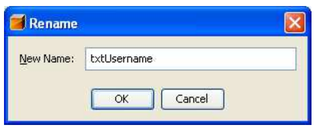


16

ClientFrame Design /3

- Rename the JTextField you added to *txtUsername*
  - Right-click on *jTextField1* in the Inspector pane and select *Change Variable Name…*
  - Enter *txtUsername* as the new name

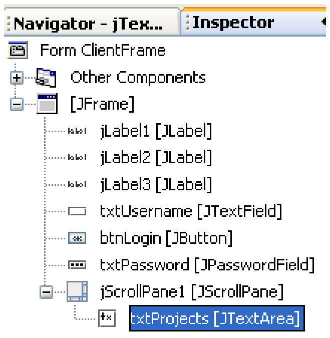




17

ClientFrame Design /4

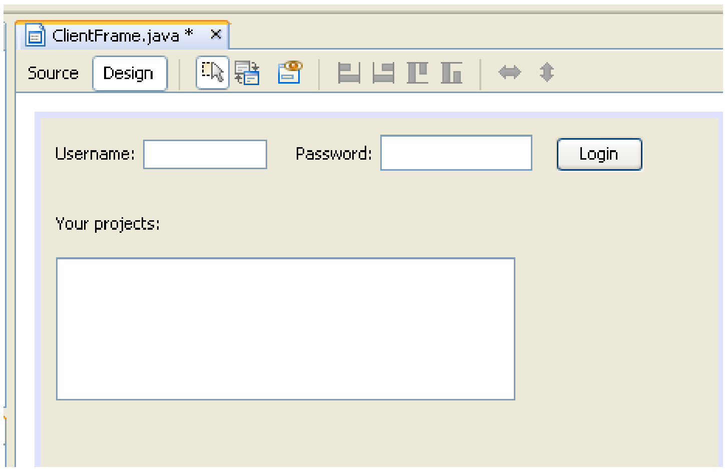
- In the same way, rename the following:
  - JPasswordField – txtPassword
  - JButton – btnLogin
  - JTextArea - txtProjects



18

ClientFrame Design /5

- When completed, the form should look like this:



19

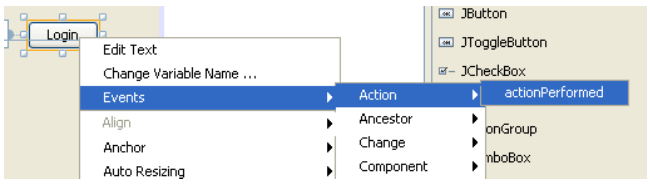
Functionality

- The only functionality we will implement is that when the button is pressed the access details are checked and, on successful authentication, the list of projects that the team leader is working on is displayed
- We will have three methods:
  - Event handling method called when the button is pressed
  - authenticate, which performs the authentication
  - showProjects, which extracts the project information and displays in in the text area

20

Adding an Action Event

- Right-click on the button on the designeer and select *Events -> Action -> actionPerformed*
  - This automatically generates skeleton code for the *actionPerformed* method and relevant event handling code is generated



21

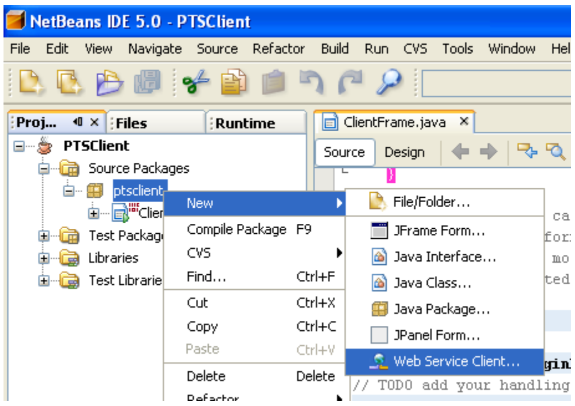
Adding Web Service Client

- The Java application will need to make calls on the .NET web service, so we will need to add a web service client
- Ensure that the SQL Server database is running and available
- Further, open the PTSLibrary solution, right-click on the PTSClientWebService.asmx file and select *View in Browser*
  - The web service methods should be exposed in the browser
  - Copy the URL displayed in the browser as we will soon need it
    - e.g. http://localhost:2844/PTSWebService/PTSClientWebService.asmx

22

Adding Web Service Client /2

- Right-click on the *ptsclient* package and select *New -> Web Service Client…*



```
// TODO add your handling
```

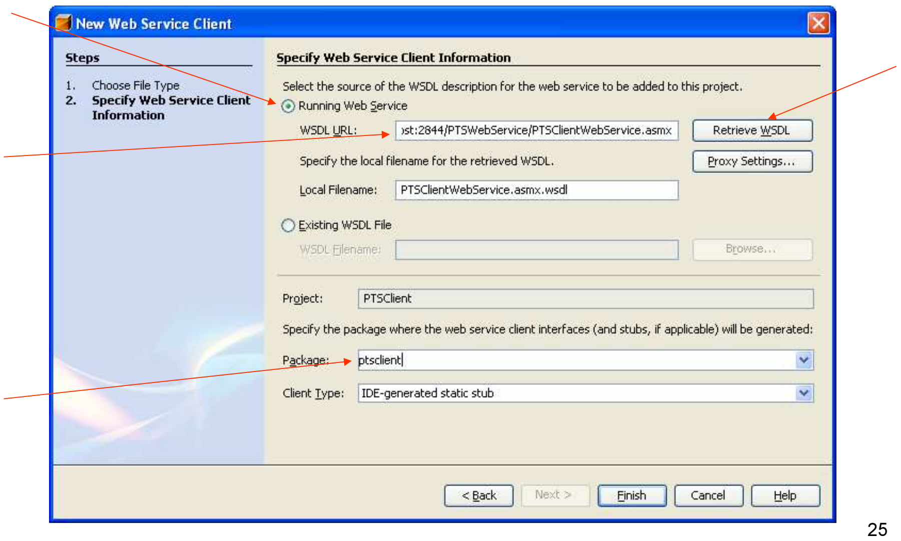
23

Adding Web Service Client /3

- On the dialog window, do the following:
  - Make sure the *Running Web Service* option is selected
  - Paste the URL of the web service which you copied from the browser into the *WSDL URL* field
  - Click *Retrieve WSDL* – this will automatically set the *Local Filename* field
  - Select the *pstclient* package from the *Package* dropdown

24

Adding Web Service Client /4


New Web Service Client

Steps

1. Choose File Type
2. **Specify Web Service Client Information**

Specify Web Service Client Information

Select the source of the WSDL description for the web service to be added to this project.

(●) Running Web Service

WSDL URL: ›st:2844/PTSWebService/PTSClientWebService.asmx

Specify the local filename for the retrieved WSDL.

Local Filename: PTSClientWebService.asmx.wsdl

( ) Existing WSDL File

WSDL Filename:

Project: PTSClient

Specify the package where the web service client interfaces (and stubs, if applicable) will be generated:

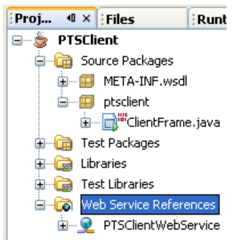
Package: ptsclient

Client Type: IDE-generated static stub

25

Adding Web Service Client /5

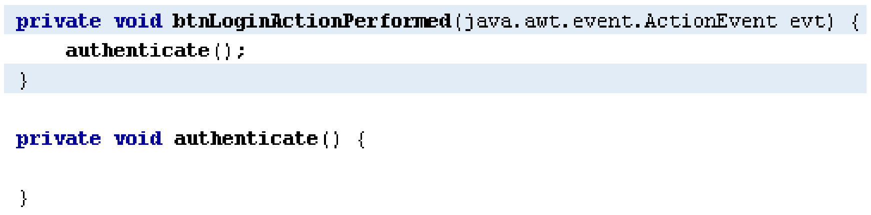
- When you click *Finish*, NetBeans will automatically generate code for the proxy classes and compile it for you
- There is now a new package in *Source Packages* called *META-INF.wsdl* and when you expand *Web Service References*, the *PTSClientWebService* should be listed there



26

btnLoginActionPerformed

- Open the *ClientFrame* file and find the *btnLoginActionPerformed* skeleton method, which was generated
- Add a call to the *authenticate* method
- Further, create the *authenticate* method

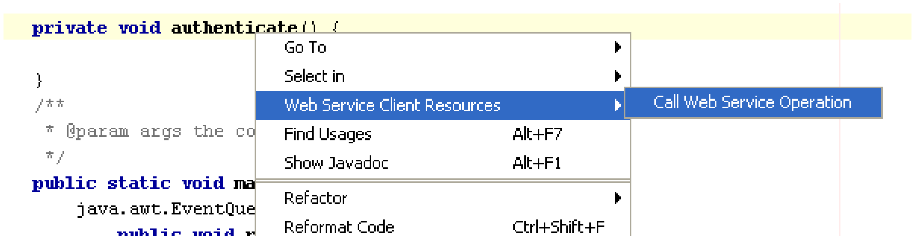


```
private void btnLoginActionPerformed(java.awt.event.ActionEvent evt) {
    authenticate();
}

private void authenticate() {

}
```

27

authenticate

- Right-click on the *authenticate* method and select *Web Service Client Resources -> Call Web Service Operation*



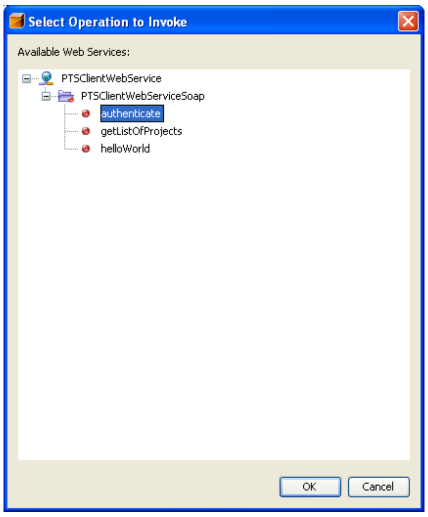
```
private void authenticate() {

}
/**
 * @param args the co
 */
public static void ma
    java.awt.EventQue
```

Go To
Select in
Web Service Client Resources
Call Web Service Operation
Find Usages Alt+F7
Show Javadoc Alt+F1
Refactor
Reformat Code Ctrl+Shift+F

28

btnLoginActionPerformed /2

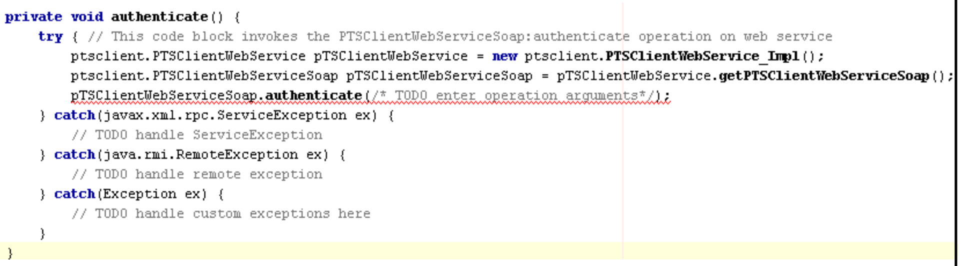
- Select the *authenticate* method and click *OK*



29

authenticate /2

- Code is automatically generated to invoke the *authenticate* web method on the web service



```
private void authenticate() {
    try { // This code block invokes the PTSClientWebServiceSoap:authenticate operation on web service
        ptsclient.PTSClientWebService pTSClientWebService = new ptsclient.PTSClientWebService_Impl();
        ptsclient.PTSClientWebServiceSoap pTSClientWebServiceSoap = pTSClientWebService.getPTSClientWebServiceSoap();
        pTSClientWebServiceSoap.authenticate(/* TODO enter operation arguments*/);
    } catch(javax.xml.rpc.ServiceException ex) {
        // TODO handle ServiceException
    } catch(java.rmi.RemoteException ex) {
        // TODO handle remote exception
    } catch(Exception ex) {
        // TODO handle custom exceptions here
    }
}
```

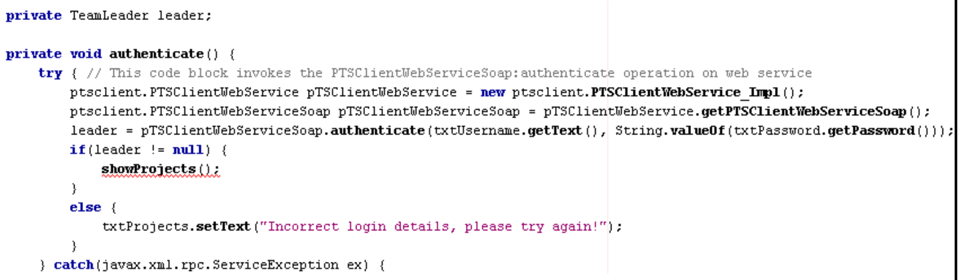
30

authenticate /3

- The call to authenticate returns a object of type TeamLeader
  - Declare an instance of TeamLeader outside of the *authenticate* method
  - Assign what the call to the web service returns to it
  - Pass the contents from the username and password fields to the method invocation on the web service
  - If an instance is returned, call the *showProjects* method

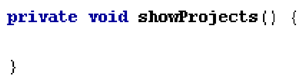
31

authenticate /4



```
private TeamLeader leader;

private void authenticate() {
    try { // This code block invokes the PTSClientWebServiceSoap:authenticate operation on web service
        ptsclient.PTSClientWebService pTSClientWebService = new ptsclient.PTSClientWebService_Impl();
        ptsclient.PTSClientWebServiceSoap pTSClientWebServiceSoap = pTSClientWebService.getPTSClientWebServiceSoap();
        leader = pTSClientWebServiceSoap.authenticate(txtUsername.getText(), String.valueOf(txtPassword.getPassword()));
        if(leader != null) {
            showProjects();
        }
        else {
            txtProjects.setText("Incorrect login details, please try again!");
        }
    } catch(javax.xml.rpc.ServiceException ex) {
```

- Next we need to create the *showProjects* method

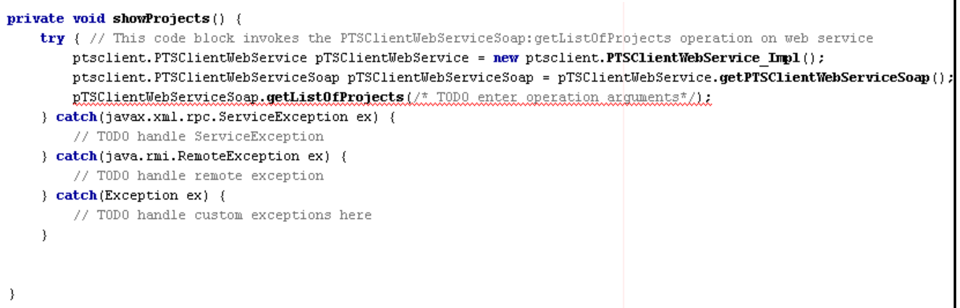


```
private void showProjects() {

}
```

32

showProjects

- Similarly to the *authenticate* method, generate the code for calling the *getListOfProjects* method

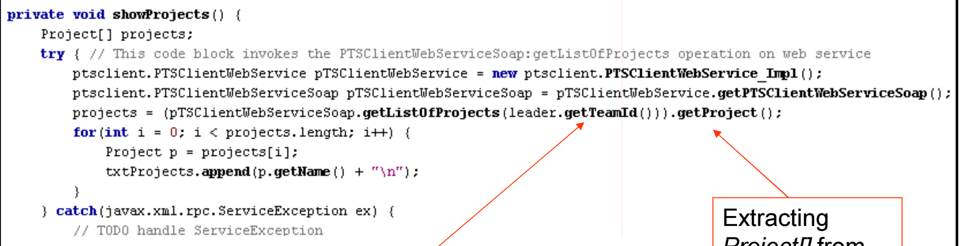


```
private void showProjects() {
    try { // This code block invokes the PTSClientWebServiceSoap:getListOfProjects operation on web service
        ptsclient.PTSClientWebService pTSClientWebService = new ptsclient.PTSClientWebService_Impl();
        ptsclient.PTSClientWebServiceSoap pTSClientWebServiceSoap = pTSClientWebService.getPTSClientWebServiceSoap();
        pTSClientWebServiceSoap.getListOfProjects(/* TODO enter operation arguments*/);
    } catch(javax.xml.rpc.ServiceException ex) {
        // TODO handle ServiceException
    } catch(java.rmi.RemoteException ex) {
        // TODO handle remote exception
    } catch(Exception ex) {
        // TODO handle custom exceptions here
    }


}
```

33

showProjects /2

- Add code to assign the returning array of *Project*
- Loop through the array and extract the names for display



```
private void showProjects() {
    Project[] projects;
    try { // This code block invokes the PTSClientWebServiceSoap:getListOfProjects operation on web service
        ptsclient.PTSClientWebService pTSClientWebService = new ptsclient.PTSClientWebService_Impl();
        ptsclient.PTSClientWebServiceSoap pTSClientWebServiceSoap = pTSClientWebService.getPTSClientWebServiceSoap();
        projects = (pTSClientWebServiceSoap.getListOfProjects(leader.getTeamId())).getProject();
        for(int i = 0; i < projects.length; i++) {
            Project p = projects[i];
            txtProjects.append(p.getName() + "\n");
        }
    } catch(javax.xml.rpc.ServiceException ex) {
        // TODO handle ServiceException
```

Passing the teamId extracted from the TeamLeader object obtained at authentication

Extracting *Project[]* from *ArrayOfProject*

34

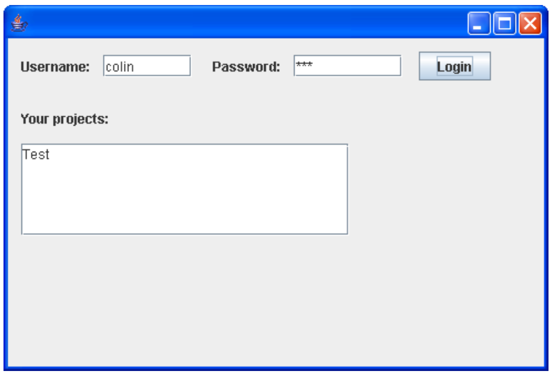
Adding Required Data to DB

- Open the database and enter some data manually such that:
  - There is a project entry
  - For which there are task entries
  - Who point at a team
  - Whose TeamLeaderId points at a person entry
- Use the login details for this person in order to test the Java application

35

Testing the application

- Run the application by clicking the button
- If all goes well, you should authenticate and get the name of project(s) displayed



36

Implementation

- Thus far we implemented:
  - Database
  - Most business classes
  - Some of the use cases (using different technologies)

37

Possible Further Work

- Exercise: try to add the missing classes and functionality to the system you have been building
  - Some classes that have not been implemented are:
    - TeamMember
    - Subtask
  - There are many use cases which you could implement (especially on the Java client and admin tool)
  - Add validation
  - Create a .NET version of the Java client
  - Other improvements you can think of

38

Summary

- By this point you should have successfully completed the Java client which connects to a .NET web service
- This demonstrates the cross-platform interoperability of web services