

# Ćwiczenie 1

## XML-RPC

*Autor: Mariusz Fraś*

### 1 Cele ćwiczenia

Celem ćwiczenia jest:

1. Poznanie ogólnej architektury aplikacji XML-RPC.
2. Opanowanie wybranej technologii tworzenia aplikacji XML-RPC.

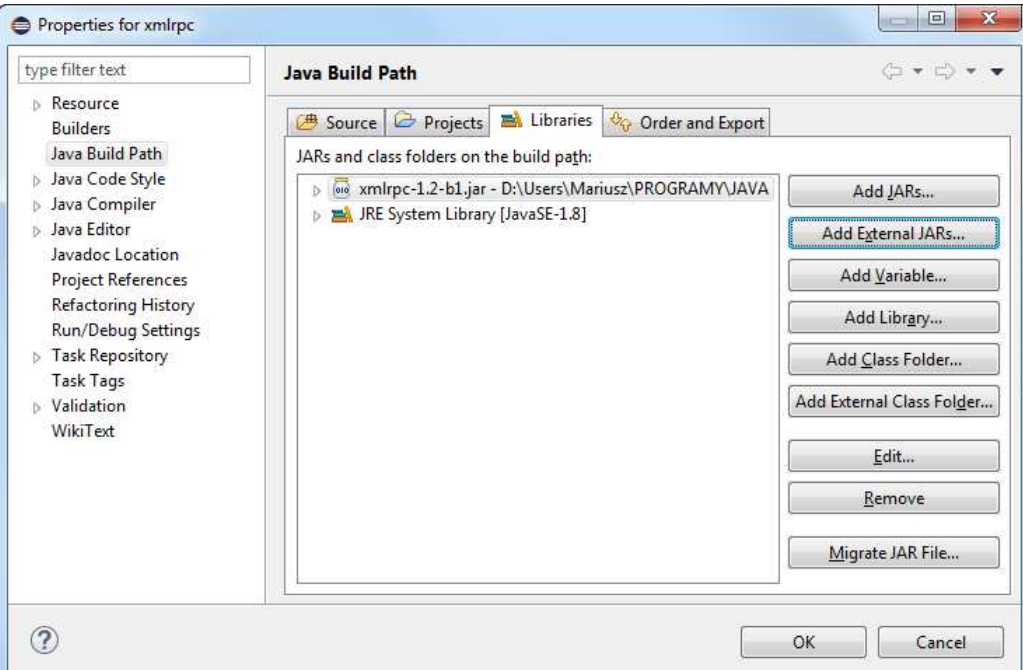
### 2 Środowisko deweloperskie

Standardowe (i zarazem wymagane) oprogramowanie do tworzenia i testowania aplikacji XML-RPC stosowane w laboratorium to:

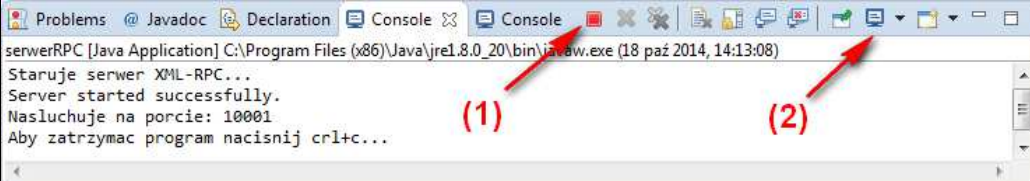
- System operacyjny Windows 7 lub wyższy
- Java 2 Software Development Kit (JDK).
- Środowisko programistyczne Java – np. Eclipse lub IntelliJ IDEA.
- Pakiet **org.apache.xmlrpc**  
Informacje o pakiecie są w kursie na eportalu.

### 3 Zadanie – część I

#### Podstawowe elementy tworzenia aplikacji XML-RPC

1. Wstępne przygotowanie środowiska	<ul style="list-style-type: none"> <li>• Przygotuj katalog roboczy, w którym będzie tworzona aplikacja.</li> <li>• Z eportalu pobierz pakiet <b>xmlrpc-1.2-b1.jar</b> do katalogu roboczego</li> <li>• Przygotuj kompilację programów Java z linii komend             <ul style="list-style-type: none"> <li>– sprawdź czy jest w systemie ścieżka dostępu do katalogu bin JDK – użyj poleceń <b>set</b> lub <b>path</b>, lub <b>echo %path%</b>. / ścieżka powinna być odpowiednio ustawiona /</li> <li>– jeśli nie ma jej to ustaw ścieżkę dostępu do katalogu bin JDK w zmiennej PATH                 <ul style="list-style-type: none"> <li>- dodanie nowej ścieżki dostępu wykonują się komendą: <b>set path=%path%;nowa_ścieżka_dostępu</b></li> <li>- lub w zmiennych środowiskowych, w zaawansowanych ustawieniach apletu System, w Panelu sterowania.</li> </ul> </li> </ul> <p><b>Uwaga:</b> błędne wykonanie komendy może wykasować wszystkie ścieżki, niezbędne do prawidłowej pracy systemu operacyjnego.</p> </li> <li>• Przejdź w oknie poleceń do katalogu roboczego i sprawdź efekty przez uruchomienie z wiersza poleceń kompilatora poleceniem: <b>javac</b> (wywołanie kompilatora potwierdzi poprawność ustawień).</li> </ul>
2. Utworzenie kodu serwera RPC	<ul style="list-style-type: none"> <li>• Utwórz w platformie Eclipse projekt Java o własnej nazwie (tu: xmlrpcserwer)</li> <li>• Podczas tworzenia projektu lub tuż po utworzeniu dodaj do projektu pakiet <b>xmlrpc-1.2-b1.jar</b> (właściwości projektu → <i>Java Build Path</i> → zakładka <i>Libraries</i> → <i>Add External JARs...</i>) – patrz rysunek.</li> </ul> 

	<ul style="list-style-type: none"> <li>• Utwórz nową klasę serwera RPC (tu nazwa: serwerRPC)</li> <li>• Podczas tworzenia serwera zaznacz opcję tworzenia metody statycznej <code>public static main(...)</code>.</li> <li>• Zaimportuj klasę serwera RPC z biblioteki RPC – dopisz na początku kodu: <pre>import org.apache.xmlrpc.WebServer;</pre> </li> <li>• Dopisz do klasy metodę publiczną o nazwie <b>echo</b>, która będzie zwracała sumę dwóch zmiennych całkowitych podanych jako parametry wywołania metody, np. <pre>public Integer echo(int x, int y) {     return new Integer(x+y); }</pre> </li> <li>• W metodzie <b>main</b> w bloku: <pre>try {     ... } catch (Exception exception) {     System.err.println("Serwer XML-RPC: " + exception); }</pre> dodaj kod (w miejsce kropek): <pre>System.out.println("Startuje serwer XML-RPC..."); int port = xxx; WebServer server = new WebServer(port); //ponizej tworzy się obiekt swojej klasy serwera //i uruchamia się go: server.addHandler("MojSerwer", new serwerRPC()); server.start(); System.out.println("Serwer wystartował pomyslnie."); System.out.println("Nasluchuje na porcie: " + port); System.out.println("Aby zatrzymać serwer naciśnij"); System.out.println("ctrl+c");</pre> zamiast <b>xxx</b> podaj wartość <b>10000+nr komputera w laboratorium</b>. </li> <li>• Sprawdź poprawność kodu.</li> </ul>
3. Utworzenie kodu klienta RPC	<ul style="list-style-type: none"> <li>• Utwórz w Eclipse drugi projekt Java o własnej nazwie (tu: xmlrpcklient)</li> <li>• Podczas tworzenia projektu lub tuż po utworzeniu dodaj do projektu pakiet <b>xmlrpc-1.2-b1.jar</b> – tak jak dla serwera.</li> <li>• Utwórz nową klasę klienta RPC</li> <li>• Podczas tworzenia serwera zaznacz opcję tworzenia metody statycznej <code>public static main(...)</code>.</li> </ul>

	<ul style="list-style-type: none"> <li>• Zaimportuj klasę serwera RPC z biblioteki RPC – dopisz na początku kodu:  <code>import org.apache.xmlrpc.XmlRpcClient;</code></li> <li>• W metodzie main w bloku  <code>try {</code>  <code>...</code>  <code>} catch (Exception exception) {</code>  <code>System.err.println("Klient XML-RPC: " +exception);</code>  <code>}</code>  dodaj:  - kod utworzenia obiektu wywołania metody serwera,  - upakowania parametrów dla wywołania metody echo,  - wywołania samej metody,  t.j.:  <code>XmlRpcClient srv = new</code>  <code>XmlRpcClient("http://localhost:xxx");</code>  <code>Vector&lt;Integer&gt; params = new Vector&lt;Integer&gt;();</code>  <code>params.addElement(new Integer(13));</code>  <code>params.addElement(new Integer(21));</code>  <code>Object result =</code>  <code>srv.execute("MojSerwer.echo", params);</code>  Zamiast xxx podaj port = 10000 + numer komputera w laboratorium.</li> <li>• Dodaj także wyświetlanie otrzymanej wartości. W tym celu najpierw przekształć rezultat na wartość o odpowiednim typie:  <code>int wynik = ((Integer) result).intValue();</code>  i wyświetl wynik na ekranie;</li> </ul>
4. Testowanie działania aplikacji	<ul style="list-style-type: none"> <li>• Uruchom serwer w Eclipse</li> <li>• Uruchom klienta w Eclipse</li> <li>• Skontroluj w konsolach wyniki działania.</li> <li>• Zatrzymaj serwer.</li> </ul> <p>Uwaga: w Eclipse zatrzymuje się serwer nie za pomocą sekwencji klawiszy ctrl-c (jak w konsoli) ale naciśnięciem ikony (1) okna konsoli jak na rysunku poniżej</p>  <p>Przełączanie między konsolami jest możliwe za pomocą przycisku (2) lub wybierając odpowiednią zakładkę.</p>

5. Procedura asynchroniczna	<ul style="list-style-type: none"> <li>W projekcie klienta utwórz nową klasę, która będzie zawierała metody wywoływane gdy na serwerze zakończy się przetwarzanie procedury wywoływanej asynchronicznie. <ul style="list-style-type: none"> <li>Utwórz w projekcie klasę</li> <li>Zdefiniuj klasę publiczną implementującą interfejs <b>AsyncCallback</b> (tu klasę AC). <pre>public class AC implements AsyncCallback { ... }</pre> </li> <li>Dopisz wymagane przez interfejs dwie metody <pre>public void handleResult(Object rezultat, URL url,                         String metoda)</pre> <p>oraz:</p> <pre>public void handleError(Exception e, URL url,                         String metoda)</pre> </li> <li>W obu metodach dopisz wyświetlanie na ekranie odpowiednich (łatwo rozpoznawalnych) komunikatów.</li> </ul> </li> <li>W kodzie klienta (w klasie głównej, w funkcji main) dopisz kod <ul style="list-style-type: none"> <li>deklaracja i utworzenie obiektu typu implementującego interfejs AsyncCallback (tu obiektu typu AC): <pre>AC cb = new AC();</pre> </li> <li>utworzenia obiektu parametrów (tu: vector) dla wywołania nowej metody asynchronicznej <b>execAsy</b>,</li> <li>wywołania metody asynchronicznej <b>execAsy</b> z parametrami: nazwa procedury, obiekt parametrów procedury, wskazanie na obiekt którego metody będą wywołane po zakończeniu procedury. <pre>... Vector&lt;Integer&gt; params2 = new Vector&lt;Integer&gt;(); params2.addElement(new Integer(3000)); srv.executeAsync("MojSerwer.execAsy", params2, cb); System.out.println("Wywolano asynchronicznie"); ...</pre> </li> </ul> </li> <li>Zwróć potem uwagę na komunikaty po wywołaniu procedury.</li> <li>Dopisz w kodzie serwera jeszcze jedną metodę – wywoływaną asynchronicznie procedurę <b>execAsy</b>. Procedura będzie teoretycznie długo coś wykonywać, co zasymulowane będzie wstrzymaniem wykonywania na okres x milisekund (x – parametr podawany w wywołaniu procedury) metodą <code>Thread.sleep(x)</code>.</li> </ul>
-----------------------------	--

	<pre> public int execAsy(int x) {     System.out.println("... wywołano asy - odliczam");     try {         Thread.sleep(x);     } catch (InterruptedException ex) {         ex.printStackTrace();         Thread.currentThread().interrupt();     }     System.out.println("... asy - koniec odliczania");     return (123); } </pre>
6. Testowanie działania aplikacji	<ul style="list-style-type: none"> <li>• Uruchom serwer a następnie klienta w Eclipse</li> <li>• Zwróć uwagę na kolejność i momenty wypisywania komunikatów. Zauważ, że po wywołaniu metody asynchronicznie, natychmiast wykonywane są kolejne instrukcje w kliencie i dopiero po pewnym czasie wywoływana jest zwrótnie metoda <b>handleResult</b> i wypisywany jest jej komunikat.</li> <li>• Można zamienić kolejność wywoływani metod w kliencie (lub dopisać kolejne) aby zaobserwować działanie aplikacji.</li> </ul>
7. Kompilacja, uruchamianie i testowanie aplikacji w konsoli Windows	<ul style="list-style-type: none"> <li>• Przejdź do konsoli Windows (Wiersz poleceń)</li> <li>• Skopiuj do katalogu roboczego następujące pliki:             <ul style="list-style-type: none"> <li>- xmlrpc-1.2-b1.jar</li> <li>- pliki *.java (źródła aplikacji)</li> </ul> </li> <li>• Skompiluj najpierw serwer, a następnie klienta poleceniem <b>javac</b> – aby dodać bibliotekę jar wykorzystaj parametr <b>-classpath</b> podając po parametrze nazwę pliku jar poprzedzoną znakami ".*;" (cudzysłówów nie podawać).</li> <li>• Uruchom serwer w oddzielnym oknie poleceniem <b>start java</b> <ul style="list-style-type: none"> <li>- parametr classpath użyj analogicznie jak wyżej.</li> </ul> </li> <li>• Uruchom klienta poleceniem <b>java</b> <ul style="list-style-type: none"> <li>- parametr classpath użyj analogicznie jak wyżej.</li> </ul> </li> </ul> <p>Podpowiedź: w poleceniu javac podaje się rozszerzenia plików, w poleceniu java nie podaje się rozszerzenia plików.</p>
8. Testowanie w sieci	<p><i>Ta część ćwiczenia może być zmodyfikowana przez prowadzącego.</i></p> <ul style="list-style-type: none"> <li>• Skompiluj klienta i serwera tak, aby można je było uruchomić na dwóch różnych komputerach w sieci (podając adres IP)</li> </ul> <p><i>lub</i></p> <ul style="list-style-type: none"> <li>• Ustal z wybraną osobą wykonującą ćwiczenie wzajemne wykorzystanie utworzonych serwerów. Skompiluj odpowiednio klienta tak, jak dla pierwszej opcji.</li> <li>• Uruchom klienta i serwer na dwóch różnych komputerach w laboratorium i przetestuj działanie.</li> </ul>

## 4 Zadanie – część II

**Szczegółowe wymagania i wybór zadania (A, B lub inne) określa prowadzący grupy.**

### A. Przykładowe zadanie do przećwiczenia.

Napisać aplikację XML-RPC (może to być aplikacja "konsolowa" (wiersza poleceń)) spełniającą następujące wymagania:

1. Serwer RPC wykonuje kilka działań (oferuje kilka procedur), **inne niż w podanej instrukcji ćwiczenia**, wymagających podania **więcej niż jednego parametru**.
2. Przynajmniej jedna procedura zawiera **parametry różnego typu**. Wykorzystać w tym celu w kliencie np. wektor zmiennych typu Object.
3. Przynajmniej jedna procedura jest **wywoływana asynchronicznie**.
4. Serwer zawiera usługę/procedurę **show**, która podaje informacje o dostępnych procedurach – wyświetla ich listę z opisem (nazwa procedury, parametry, krótki opis)
5. Dodać w aplikacji funkcjonalność:

przy uruchamianiu klienta można podać adres dostępowy usług serwera – adres IP lub DNS i port – i dopiero potem łączyć się ze wskazanym serwerem.

### B. Zastanowić się jak zaimplementować aplikację, żeby klientowi wystarczyła informacja o adresie serwera i metodzie *show()* – aby można było wywoływać dowolną usługę serwera bez wcześniejszej znajomości tych usług oprócz znajomości usługi *show()*.

A więc, tak żeby nie trzeba było na sztywno kodować ich wywołania w kliencie wg specyfikacji.

Czyli nie znamy wcześniej usług (oprócz adresu i usługi *show()*), i chcemy móc z nich skorzystać bez powtórnego kodowania klienta. Poznajemy usługi tylko dzięki usłudze *show()*.

**Uwaga:** na serwerze nie powinna to być jedna sparametryzowana procedura z parametrami typu string (proste rozwiązanie ale nie to jest celem), ale oddzielne procedury dla każdej usługi.

### C. Przygotować się do napisania na zajęciach aplikacji o podobnych funkcjonalnościach (testu z samodzielnego napisania klienta i serwera według podanych wskazówek).