

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.

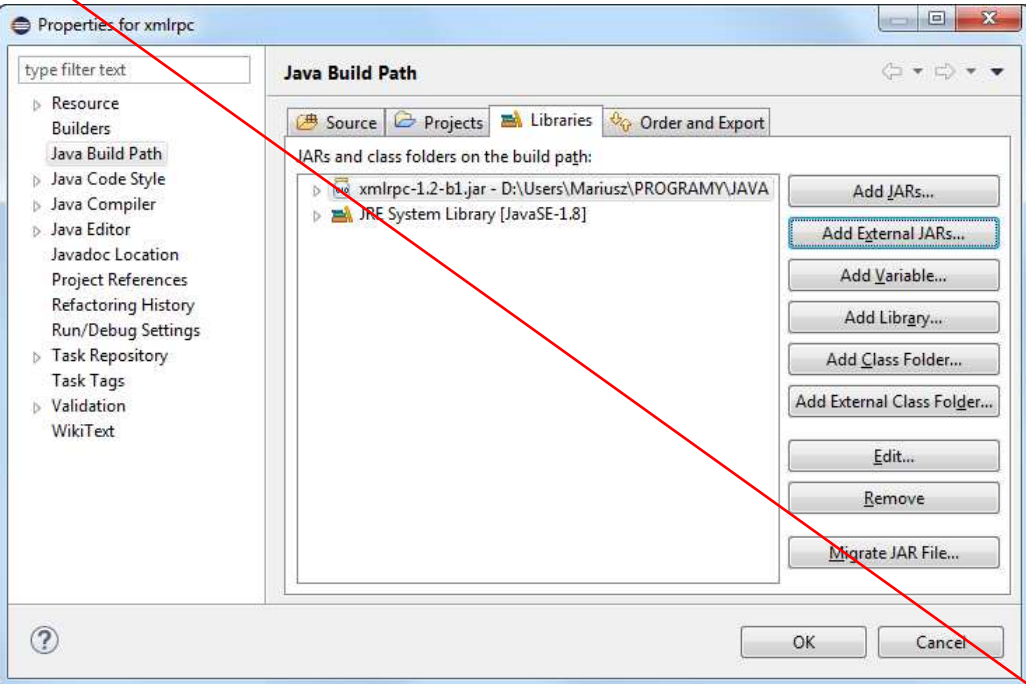
2 Środowisko deweloperskie

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

	<ul style="list-style-type: none"> • Utwórz nową klasę serwera RPC (tu nazwa: serwerRPC) • Podczas tworzenia serwera zaznacz opcję tworzenia metody statycznej <code>public static main(...)</code>. • Zaimportuj klasę serwera RPC z biblioteki RPC – dopisz na początku kodu: <pre>import org.apache.xmlrpc.WebServer;</pre> • Dopisz do klasy metodę publiczną o nazwie echo, 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> • W metodzie main 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 xxx podaj wartość 10000+nr komputera w laboratorium. • Sprawdź poprawność kodu.
3. Utworzenie kodu klienta RPC	<ul style="list-style-type: none"> • Utwórz w Eclipse drugi projekt Java o własnej nazwie (tu: <code>xmlrpc klient</code>) • Podczas tworzenia projektu lub tuż po utworzeniu dodaj do projektu pakiet xmlrpc-1.2-b1.jar – tak jak dla serwera. • Utwórz nową klasę klienta RPC • Podczas tworzenia serwera zaznacz opcję tworzenia metody statycznej <code>public static main(...)</code>.

- Zaimportuj klasę serwera RPC z biblioteki RPC – dopisz na początku kodu:

```
import org.apache.xmlrpc.XmlRpcClient;
```
- W metodzie main w bloku

```
try {
    ...
} catch (Exception exception) {
    System.err.println("Klient XML-RPC: " +exception);
}
```

dodaj:
 - kod utworzenia obiektu wywołania metody serwera,
 - upakowania parametrów dla wywołania metody echo,
 - wywołania samej metody,
t.j.:

```
XmlRpcClient srv = new
    XmlRpcClient("http://localhost:xxx");
Vector<Integer> params = new Vector<Integer>();
params.addElement(new Integer(13));
params.addElement(new Integer(21));
Object result =
    srv.execute("MojSerwer.echo", params);
```

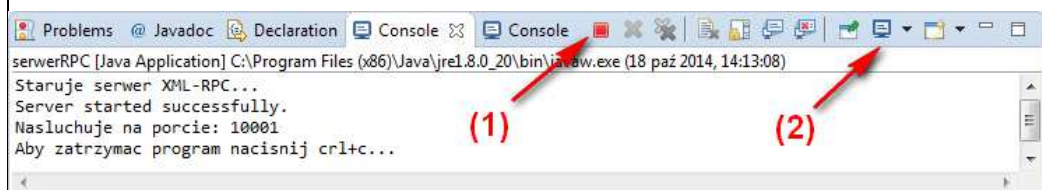
Zamiast xxx podaj port = 10000 + numer komputera w laboratorium.
- Dodaj także wyświetlanie otrzymanej wartości. W tym celu najpierw przekształć rezultat na wartość o odpowiednim typie:

```
int wynik = ((Integer) result).intValue();
```

i wyświetl wynik na ekranie;

4. Testowanie działania aplikacji

- Uruchom serwer w Eclipse
 - Uruchom klienta w Eclipse
 - Skontroluj w konsolach wyniki działania.
 - Zatrzymaj serwer.
- 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



Przełączanie między konsolami jest możliwe za pomocą przycisku (2) lub wybierając odpowiednią zakładkę.

<p>5. Procedura asynchroniczna</p>	<ul style="list-style-type: none"> 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"> Utwórz w projekcie klasę Zdefiniuj klasę publiczną implementującą interfejs AsyncCallback (tu klasę AC). <pre>public class AC implements AsyncCallback { ... }</pre> 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> W obu metodach dopisz wyświetlanie na ekranie odpowiednich (łatwo rozpoznawalnych) komunikatów.
	<ul style="list-style-type: none"> W kodzie klienta (w klasie głównej, w funkcji main) dopisz kod <ul style="list-style-type: none"> deklaracja i utworzenie obiektu typu implementującego interfejs AsyncCallback (tu obiektu typu AC): <pre>AC cb = new AC();</pre> utworzenia obiektu parametrów (tu: vector) dla wywołania nowej metody asynchronicznej execAsy, wywołania metody asynchronicznej execAsy z parametrami: nazwa procedury, obiekt parametrów procedury, wskazanie na obiekt którego metody będą wywołane po zakończeniu procedury. <pre>... Vector<Integer> params2 = new Vector<Integer>(); params2.addElement(new Integer(3000)); srv.executeAsync("MojSerwer.execAsy", params2, cb); System.out.println("Wywolano asynchronicznie"); ...</pre> Zwróć potem uwagę na komunikaty po wywołaniu procedury. Dopisz w kodzie serwera jeszcze jedną metodę – wywoływaną asynchronicznie procedurę execAsy. 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>.

	<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"> • Uruchom serwer a następnie klienta w Eclipse • 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 handleResult i wypisywany jest jej komunikat. • Można zamienić kolejność wywoływani metod w kliencie (lub dopisać kolejne) aby zaobserwować działanie aplikacji.
7. Kompilacja, uruchamianie i testowanie aplikacji w konsoli Windows	<ul style="list-style-type: none"> • Przejdź do konsoli Windows (Wiersz poleceń) • Skopiuj do katalogu roboczego następujące pliki: <ul style="list-style-type: none"> - xmlrpc-1.2-b1.jar - pliki *.java (źródła aplikacji) • Skompiluj najpierw serwer, a następnie klienta poleceniem javac – aby dodać bibliotekę jar wykorzystaj parametr -classpath podając po parametrze nazwę pliku jar poprzedzoną znakami "·;" (cudzysłówów nie podawać). • Uruchom serwer w oddzielnym oknie poleceniem start java - parametr classpath użyj analogicznie jak wyżej. • Uruchom klienta poleceniem java - parametr classpath użyj analogicznie jak wyżej. <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"> • Skompiluj klienta i serwera tak, aby można je było uruchomić na dwóch różnych komputerach w sieci (podając adres IP) <p><i>lub</i></p> <ul style="list-style-type: none"> • Ustal z wybraną osobą wykonującą ćwiczenie wzajemne wykorzystanie utworzonych serwerów. Skompiluj odpowiednio klienta tak, jak dla pierwszej opcji. • Uruchom klienta i serwer na dwóch różnych komputerach w laboratorium i przetestuj działanie.

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).