Ćwiczenia z XML-RPC

Plik z zadaniami z zajęć z XML-RPC

Poniżej znajdują się zadania do wykonania na zajęciach. Oprócz nich na tym repozytorium jest także prezentacja oraz zadania w wersji pdf.

Wstęp

Do rozwiązania poniższych zadań wykorzystamy język Java, pakiet org.apache.xmlrpc będący implementacją protokołu XML-RPC oraz narzędzie Maven. Zalecanym środowiskiem wykonywania zadań jest IntelliJ IDEA. W celu przetestowania implementacji serwera przydatny będzie również program Postman.

Zadania

1. Na początek zaimplementujemy prosty serwer. Uruchom proszę IntelliJ IDEA i utwórz nowy projekt. Jako typ projektu wybierz Maven.
2. Zaimportuj bibliotekę org.apache.xmlrcp za pomocą Mavena. W tym celu otwórz plik pom.xml i sekcji w sekcji dependiences wklej następujący kod.

```
<!-- https://mvnrepository.com/artifact/org.apache.xmlrpc/xmlrpc -->
<dependency>
    <groupId>org.apache.xmlrpc</groupId>
    <artifactId>xmlrpc-server</artifactId>
    <version>3.1.3</version>
    <type>pom</type>
</dependency>
```

Uwaga! Jeśli Maven nie wygenerował sekcji dependiences utwórz ją wpisując <dependiences></dependiences>

3. Zapoznaj się z poniższym kodem i skopuj metodę *Main* do swojej klasy

```
public static void main (String [] args){
        try {
            // Uruchomienie serwera HTTP na porcie 80
            // Ten serwer obsługuje tylko żądania XML-RPC
            WebServer webServer = new WebServer(80);
            XmlRpcServer xmlRpcServer = webServer.getXmlRpcServer();

            // Konfiguracja uchwytów do klas implementujących metody, które
            // chcemy wystawić naszym klientom.
            PropertyHandlerMapping phm = new PropertyHandlerMapping();
            // Pierwszy argument to nazwa, której użyjemy do identyfikowania
            // danej klasy
            // Drugi argument to klasa, której metody publiczne chcemy
            // udostępnić
            phm.addHandler("hello", JavaServer.class);

            // Po ustawieniu wszystkich uchwytów możemy przypisać mapowanie do
            // serwera
            xmlRpcServer.setHandlerMapping(phm);
            // uruchomienie serwera
```

```
            webServer.start();

            System.out.println("Serwer pracuje...");

        } catch (Exception exception){
            System.err.println("Nie można uruchomić serwera: " + exception);
        }//catch()
    }//main()
```

4. Zaimplementuj teraz w tej samej klasie metodę *sayHello*, która nie przyjmuje żadnych parametrów i zwraca *String*. Możesz w niej zwrócić dowolny napis witający użytkownika.
5. Uruchom program. Na konsoli powinien zostać wyświetlony napis

Serwer pracuje...

6. Uruchom proszę program *Postman* i zaimportuj do niego następującą kolekcję zapytań: klik. Będą one nam potrzebne do przetestowania serwera.
7. Zapoznaj się z żądaniem *sayHelloNoParams*, wyślij je do serwera i wynikliwie przestudiuj odpowiedź serwera.
8. Wróć do kodu serwera. Zmodyfikuj metodę *sayHello* tak aby przyjmowała *String* jako parametr i zwracała go w odpowiedzi.
9. Uruchom ponownie serwer i przetestuj go żądaniem *sayhelloWithParam* z programu Postman. Zapoznaj się z ciałem żądania HTTP przed jego wysłaniem i wprowadź w nim swój dowolny ciąg znaków.
10. Serwer już działa. Zaimplementujemy teraz klienta naszej rozbudowanej usługi. Wróć do IDE i utwórz proszę nowy projekt. Ponownie wykorzystaj do tego Maven. Do sekcji *dependiences* dodaj poniższy kod.

```
<!-- https://mvnrepository.com/artifact/org.apache.xmlrpc/xmlrpc-client -->
<dependency>
    <groupId>org.apache.xmlrpc</groupId>
    <artifactId>xmlrpc-client</artifactId>
    <version>3.1.3</version>
</dependency>
```

11. Dodaj nową klasę i wklej do niej podany niżej kod.

```
public static void main (String [] args) {
        try {
            // Utworzenie i konfiguracja klienta
            XmlRpcClient client = new XmlRpcClient();
            XmlRpcClientConfigImpl config = new XmlRpcClientConfigImpl();
            // dodanie adresu URL serwera
            config.setServerURL(new URL("http://localhost/RPC2"));
            client.setConfig(config);

            // Inicjalizacja parametrów
            Vector params = new Vector();
            params.add(new String("Dawid nie jest wcale takim dobrym programistą
jak sam uważa."));
            // kolejne parametry przesyła się dodając je w powyższy sposób

            // Zdalne wykonanie metody na serwerze i zapisanie odpowiedzi
            Object result = client.execute("hello.sayHello", params);

            // Wyświetlenie odpowiedzi serwera na ekranie
            String res = (String) result;
```

```
            System.out.println(res);

        } catch (Exception exception) {
            System.err.println("Coś poszło nie tak: " + exception);
        }//catch
    }//main()
```

12. Po zapoznaniu się z powyższym kodem i zaimportowaniu odpowiednich bibliotek przejdź do uruchomienia programu. Jeśli wszystko zrobiłeś prawidłowo na konsoli powinieneś otrzymać tekst, który wysyłasz do serwera.
13. Zaimplementujemy teraz przesyłanie tablicy. Będzie to tablica obiektów, którą wyślemy od klienta do serwera. Serwer w odpowiedzi odeśle napis złożony z elementów tej tablicy. Zacznij proszę od implementacji obsługi tego żądania po serwera. Dodaj metodę iterującą po otrzymanej tablicy, sklejającą elementy tablicy i zwracającą sklejony ciąg znaków. W ramach tego ćwiczenia ogranicz się do obsługi elementów typu *int*, *double* i *string*. Podpowiedź 1: Twoja metoda powinna przyjmować tablicę obiektów (*Object[] elements*). Podpowiedź 2: Będziesz musiał sprawdzić jaki jest typ obiektu, żeby zrzutować go na *String*. Pomocny może okazać się tutaj operator *instanceof*.
14. W celu przetestowania swojej implementacji wykorzystaj zapytanie concatArray z wcześniej zaimportowanej kolekcji w programie *Postman*.
15. Zmodyfikuj klienta tak, aby uruchamiał zdalnie zaimplementowaną procedurę serwera. Wysyłaj jako parametr tablicę obiektów. Następnie przetestuj działanie aplikacji klienckiej. Podpowiedź 1: W punkcie jedenastym klient wysyła tak naprawdę tablicę parametrów. Jeśli teraz wyślemy do serwera tablicę to jej poszczególne elementy zostaną zinterpretowane jako kolejne parametry co poskutkuje tym, że serwer zwróci błąd.
16. Dodaj do serwera klasę wewnętrzną Grade używając poniższego kodu.

```
class Grade {
   String name;
   Double value;
}
```

17. Dopiszemy teraz do serwera metodę przyjmującą obiekt klasy Grade, modyfikującą i zwracającą go. Kod metody znajduje się niżej. Skopiuj go i przetestuj swoją implementację za pomocą *Postmana*. Wykorzystaj zapytanie *modifyObject*

```
public Object modifyObject(Map<String, Object> o) {
        Grade g = new Grade();
        g.name = (String) o.get("name");
        g.value = (Double) o.get("value");
        g.value -= 1;

        Map<String, Object> map = new HashMap<String, Object>();
        map.put("name", g.name);
        map.put("value", g.value);
        return map;
    }
```

18. Na podstawie powyższego kodu modyfikuj klienta tak, aby wysyłał do serwera obiekt typu Grade i wyświetlał na ekranie wartość pól otrzymanej w odpowiedzi struktury.

Dla chętnych:

19. Sprawdź co się stanie jeśli rzucimy w jakieś metodzie wyjątek. Przetestuj zdalne wywołanie tej metody z wykorzystaniem Postmana i aplikacji klienta.
20. Rozbuduj klasę Grade tak, aby przechowywała tablicę kilku ocen. Obsłuż taką strukturę w kodzie serwera i klienta.