INSTITUTO DE INFORMÁTICA
UFRGS

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
DEPARTAMENTO DE INFORMÁTICA APLICADA

INF01151 – SISTEMAS OPERACIONAIS II N
SEMESTRE 2015/1
AULA PRÁTICA: WEB SERVICES

---

1. OBJETIVO DA AULA PRÁTICA:

Esta aula prática tem por objetivo ilustrar o uso de web services utilizando a linguagem Java. Você deverá implementar uma aplicação bastante simples que utilize a API Java para web services. A aplicação permitirá que um web service disponibilize um serviço para a execução de operação matemáticas, e que um processo cliente possa fazer chamadas de invocação remota a esse serviço. Nesta aula, estaremos utilizando a Java API for XML Web Services (JAX-WS), disponível desde a versão Java SE 6.

Tutoriais de apoio: http://docs.oracle.com/javaee/6/tutorial/doc/bnayl.html
http://tomee.apache.org/examples-trunk/simple-webservice/README.html

2. CALCULADORA COMO WEB SERVICE:

Você deverá implementar um web service em Java, onde sua função será disponibilizar uma interface remota que funcione como uma calculadora. A aplicação cliente irá utilizar a URL desse web service para realizar chamadas a sua função.

Passos:

1. **Faça o download da implementação base:** baixe da página do moodle o arquivo **aula-17.zip**, e descompacte-o. Verifique no diretório descompactado quais classes estão presentes e examine o conteúdo delas.
2. Os arquivos fonte especificam um pacote Java chamado "webservice". Portanto crie um diretório chamado "webservice", e salve dentro dele os arquivos do passo anterior.
3. **Especifique a interface remota do web service:** você deverá criar uma interface chamada `Calculadora` em um arquivo chamado **Calculadora.java**. Salve este arquivo dentro da pasta webservice. Você deverá adicionar as seguintes anotações na declaração de sua interface:

```
@WebService
@SOAPBinding(style = Style.RPC)
```

A primeira delas marca esta interface Java como uma *service endpoint interface (SEI)*, a qual irá declarar métodos que o cliente poderá invocar no web service. A segunda anotação especifica o tipo de SOAP binding, o qual define o estilo de comunicação: Document ou RPC.

- **Document style**: permite estruturar o corpo da mensagem SOAP da forma que o programador desejar, desde que ele represente um XML arbitrário (mais flexível, utilizado quando clientes podem sofrer alterações frequentes)
- **RPC style**: assume uma estrutura específica do XML no corpo da mensagem SOAP (nome de operação + parâmetros), e corresponde à representação XML de uma chamada de método seguindo o modelo request/response.

Informações sobre SOAP binding :
http://www.w3schools.com/webservices/ws_wsdl_binding.asp

SOAP binding em Java:
http://docs.oracle.com/javaee/5/api/javax/jws/soap/SOAPBinding.Style.html

A interface deverá definir os métodos que serão implementados pelo web service. Comece definindo as seguintes assinaturas:

```
public int somarInt(int a, int b)
public int multiplicarInt(int a, int b)
```

Note que antes de cada método você deverá incluir anotações do tipo `@WebMethod`. As anotações colocadas sobre cada assinatura de método têm um propósito semelhante ao das anotações anteriores, ou seja, especificam um método do web service.

Ainda, lembre-se de declarar o pacote `webservice` e importar as seguintes bibliotecas:

```
javax.jws.WebMethod
javax.jws.WebService
javax.jws.soap.SOAPBinding
javax.jws.soap.SOAPBinding.Style
```

4. **Implemente o web service:** a classe chamada `CalculadoraImpl` deverá implementar a interface `Calculadora`. Você deverá indicar isso na declaração da classe `CalculadoraImpl` através de `implements Calculadora` e ainda adicionar a seguinte anotação sobre a declaração da classe:

```
@WebService(endpointInterface="webservice.Calculadora")
```

Esta anotação tem o propósito de especificar que a classe é uma implementação para a interface definida no passo anterior, e instruir o WSDL gerado a usar os parâmetros definidos na interface.

Além disso, para cada uma das assinaturas de métodos definidas na interface `Calculadora`, deverá existir uma implementação de método correspondente na classe `CalculadoraImpl`. A especificação do comportamento de cada método é intuitiva e pode ser inferida pelos seus nomes. Implemente esses métodos na classe `CalculadoraImpl` não esquecendo de colocar a anotação `@Override` sobre cada um dos métodos. **Atenção**: a classe `CalculadoraImpl` também pertence ao pacote `webservice` e precisa importar `javax.jws.WebService`.

5. **Primeira compilação:** nesse ponto, compile o código produzido até aqui utilizando o comando abaixo. Note que essa compilação (e todas as outras) deve ser feita a partir do diretório pai do diretório "webservice".

```
%> javac webservice/*.java
```

Houve algum erro de compilação? Resolva qualquer problema no código antes de prosseguir para o próximo passo.

6. **Faça a publicação da web service:** se tudo estiver correto até este ponto, você está pronto para implementar o código que fará a publicação do web service. Isso será feito na classe chamada `CalculadoraPublisher.java`. Dentro dessa classe crie uma função `main` e adicione o seguinte comando em seu corpo:

```
Endpoint.publish("http://127.0.0.1:9000/webservice",new CalculadoraImpl());
```

Atenção: Não se esqueça de importar javax.*xml.ws.Endpoint*, e de declarar esta classe no pacote *webservice*.

7. **Segunda compilação:** neste ponto, compile novamente o código produzido até aqui utilizando o comando :

```
%> javac webservice/*.java
```

Houve algum erro de compilação? Resolva qualquer problema no código antes de prosseguir para o próximo passo.

8. **Dispare a aplicação Publisher**:

```
%> java -cp . webservice.CalculadoraPublisher
```

Esse comando irá disparar uma implementação de um HTTP server leve (*lightweight*), que está incluído na instalação do Java a partir da versão Java SE 6. A partir de agora o WSDL gerado descrevendo a interface para o web service poderá ser visto abrindo-se o seguinte link no browser:

```
http:// 127.0.0.1:9000/webservice?wsdl
```

Este WSDL define o contrato em XML que especifica em detalhes as operações disponibilizadas pelo web service. Abra a URL acima no browser e verifique a definição desse contrato. Uma das principais vantagens do uso de web services (em comparação com RMI) é sua interoperabilidade entre clientes e servidores implementados em diferentes linguagens. Essa interoperabilidade se deve em parte ao uso de XML. Dessa forma, poderíamos escrever um cliente em qualquer linguagem. Porém, nessa aula vamos continuar utilizando Java...

9. **Escreva o programa cliente:** neste ponto, você deverá implementar o código do cliente. Isso será feito na classe chamada `Client.java`. Dentro da função `main`, siga os seguintes passos:

   a. **Defina a URL do web service**: o cliente deverá especificar a URL do WSDL gerado para acessar o web service. Para isso, utilize a classe URL e adicione o seguinte comando:

   ```
   URL url = new URL("http://127.0.0.1:9000/webservice");
   ```

   Atenção: Você deverá tratar a exceção `MalformedURLException`. Tanto a classe `URL` como a exceção estão definidas no pacote `java.net`, que deverá ser importado.

   b. **Criação do `QName`**: `QName` (qualified name) é usado para referenciar elementos ou atributos em documentos XML (lembre-se que o arquivo WSDL descrevendo a interface de um web service é definido em um documento XML). Para criar o `QName`, adicione o comando:

   ```
   QName n = new QName("http://webservice/", "CalculadoraImplService");
   ```

   Neste caso, você estará referenciando o elemento XML (dentro do arquivo WSDL) que descreve a implementação da calculadora. A classe `QName` está definida no pacote `javax.xml.namespace`, que dever ser importado. **Atenção**: Note que a palavra "`Service`" foi adicionada no nome da classe `Calculadora` na declaração de `QName`.

   c. **Criação do serviço**: a partir da URL e do elemento QName referenciando a implementação da calculadora, você deverá criar o serviço:

   ```
   Service service = Service.create(url, n);
   ```

   Esse comando cria o objeto que serve como uma visão do cliente sobre um web service. (http://docs.oracle.com/javaee/6/api/javax/xml/ws/Service.html). A classe `Service` precisará ser importada do pacote `javax.xml.ws.`

   d. **Obtenção do proxy para o servidor**: você deverá obter o proxy, também chamado de `port`, para o web service através do comando `getPort`. O proxy obtido implementa os métodos definidos no SEI do web service:

   ```
   Calculadora calc = service.getPort(Calculadora.class);
   ```

   e. **Realização de chamadas**: o proxy é a referência que o cliente utilizará para acessar o endpoint. A partir desse ponto é possível invocar qualquer método definido no web service através desse objeto. A sintaxe de invocação remota é idêntica a sintaxe de invocação local. Execute algumas chamadas de teste e exiba na tela os valores retornados.

10. **Terceira compilação:** compile o código do cliente:

```
%> javac webservice/Client.java
```

Houve algum erro de compilação? Resolva qualquer problema no código antes de prosseguir para o próximo passo.

11. **Dispare a aplicação Cliente:**

```
%> java -cp . webservice.Client
```

12. **Problemas?** Revise os passos anteriores e corrija!

13. **Adicionando a funcionalidade de logging:**

Volte ao código cliente e adicione o seguinte comando logo após a declaração do método main:

```
System.setProperty("com.sun.xml.ws.transport.http.client.HttpTransportPipe.dump","true");
System.setProperty("com.sun.xml.internal.ws.transport.http.client.HttpTransportPipe.dump","true");
```

Esses comandos permitirão o monitoramento das mensagens HTTP-SOAP trocadas entre cliente e servidor. Repita os passos 10 e 11, e observe as mensagens de log geradas.

Yay! Se você chegou até aqui, well done. Mostre seu programa executando para o professor, e seu objetivo na aula de hoje terá sido cumprido.

Mas espere!!! Ainda há uma serie de aspectos interessantes que podem ser explorados utilizando aplicações Java Web Services. Caso você ainda tenha tempo em aula, ou caso prefira fazer em casa, algumas atividades adicionais são sugeridas como exercício:

- Experimente disparar mais de um processo cliente simultaneamente. Provavelmente eles executaram muito rapidamente. Você pode introduzir algum *delay* artificial na execução dos métodos implementados pelo servidor, ou implementar métodos mais computacionalmente intensivos (e.g., verificar se um numero é primo, calcular a série de Fibonacci de tamanho n, etc). Responda:
  - O servidor executa simultaneamente invocações concorrente feitas por múltiplos clientes? (**Dica**: imprima mensagens na tela para verificar se duas execuções de um método estão acontecendo ao mesmo tempo no servidor).
  - Lembre-se que todo o objeto em Java pode implementar um monitor, desde que seus métodos sejam definidos como `synchronized`. Altere o código do servidor de forma que todos os métodos sejam sincronizados, e verifique como clientes fazendo chamadas simultâneas se comportam agora.
- Ao invés de realizar chamadas no seu computador local, tente localizar e fazer chamadas no web service do colega sentado ao lado.
- Existem mais mensagens de log disponíveis que podem ser vistas em https://metro.java.net/nonav/1.2/guide/Logging.html. Faça uma lista de informações de log que podem ser obtidas através da API utilizada nesta aula prática.