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

INF01151 – SISTEMAS OPERACIONAIS II N
SEMESTRE 2020/2
AULA PRÁTICA: WEB SERVICES

OBJETIVO DA AULA PRÁTICA:

Esta atividade de Programação Guiada 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 para RESTful Web Services (JAX-RS), distribuído através do framework Java Jersey.

Tutoriais de apoio: https://jersey.java.net
https://docs.oracle.com/cd/E19776-01/820-4867/6nga7f5ml/index.html
https://docs.oracle.com/javaee/6/tutorial/doc/giepu.html

INFORMAÇÕES PRELIMINARES:

Algumas linhas de código longas são exibidas neste documento com quebras de linha, para facilitar a legibilidade. As quebras de linha inseridas ao longo da linha de código não interferem no processo de compilação, podendo ser copiadas "como estão" para os arquivos de código fonte a serem desenvolvidos como parte do trabalho.

INSTALAÇÃO E CONFIGURAÇÃO DE DEPENDENCIAS:

Para essa atividade de programação guiada, vamos precisar dos softwares abaixo. Ou, se preferir, pode usar a VM preparada para a atividade (vide abaixo).

- Java EE Development Kit, sugerido: JDK 8. Para instruções de instalação, acesse https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html . No Ubuntu Linux, você pode instalar o JDK 8 com o comando:

```
apt-get install openjdk-8-jdk-headless
```

Dica: Após as instalações, pode ser necessário deslogar e logar novamente no ambiente do sistema operacional, para que as variáveis de ambiente dos recursos instalados sejam exportadas para o usuário logado atualmente.

Importante: A JDK 9 e versões posteriores não são compatíveis com esse trabalho. Caso o sistema onde você está desenvolvendo o trabalho possua outras versões de JDK instaladas, certifique-se de que as variáveis PATH, JAVA_HOME e CLASSPATH estão configuradas para usar a JDK 8.

Dica: Para facilitar o trabalho, disponibilizamos uma imagem de VM usando o VirtualBox com todo o ambiente de desenvolvimento e de testes pré-configurado, pronto para usar. O endereço da VM é: https://acdc.inf.ufrgs.br/ubuntu16.ova . Instruções para importar máquinas virtuais com o VirtualBox podem ser encontradas em https://docs.oracle.com/en/virtualization/virtualbox/6.0/user/ovf.html .

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 **PG2-WebServices.zip**, e descompacte-o. Verifique no diretório descompactado quais classes estão presentes e examine o conteúdo delas. Os arquivos fonte especificam um pacote Java chamado `webservice`. Portanto crie um diretório chamado `webservice`, e copie para dentro dele os arquivos contidos no arquivo zip do passo anterior.

2. **Faça o download do *framework* Jersey 1.19,** e copie-o para o diretório raiz (isto é, o mesmo onde o diretório `webservice` se encontra). O *framework* pode ser baixado em:

   https://repo1.maven.org/maven2/com/sun/jersey/jersey-bundle/1.19/jersey-bundle-1.19.jar

3. **Analise a classe Calculadora.** Note que ela contém atributos para guardar valores dos operandos, o tipo da operação, o resultado e uma descrição de erro, assim como métodos de `Get` e `Set` para cada um. A classe contém dois construtores, um que recebe zero parâmetros (necessário para a biblioteca JAX-RS) e um que recebe os valores dos operandos e o tipo da operação a ser realizada. Por motivos de simplicidade, é neste último construtor que o cálculo desejado é feito. Porém, em uma aplicação real, as operações são realizadas nos métodos da classe.

   Para que a classe seja vista pelo web service como um objeto a ser serializado e enviado, deve-se anotar a mesma com `@XmlRootElement` logo antes de sua declaração. Adicione o `import` de `javax.xml.bind.annotation.XmlRootElement` referente a este atributo.

4. **Implemente os métodos a serem externalizados via REST:** no arquivo `CalculadoraRest.java`, você deverá criar os métodos que indicarão quais operações serão disponibilizadas pelo web service. Para tanto, deve-se anotar a classe `CalculadoraRest` (logo antes de sua declaração) com o caminho (URI) no qual tal web service estará disponível:

   ```
   @Path("calculadora")
   ```

   Implemente, na classe `CalculadoraRest`, dois métodos: um chamado `somarInt` e outro `multiplicarInt`. Ambos vão receber como parâmetros dois inteiros e retornar uma nova instância da classe `Calculadora`. Dentro de cada método, instancie a classe `Calculadora`, passando como argumentos os valores recebidos e o tipo da operação desejada, e utilize-a como valor de retorno da função. Além disso, é necessário anotar cada função com:

   - O caminho e parâmetros necessários para o funcionamento do web service, através da anotação `@Path`:

     ```
     @Path("/somarInt/{a}/{b}")
     @Path("/multiplicarInt/{a}/{b}")
     ```

   - O tipo de dado que será gerado como retorno para o web service com `@Produces`:

     ```
     @Produces(MediaType.APPLICATION_JSON)
     ```

   - O(s) métodos(s) HTTP suportados para tal função (como GET, POST, etc.):

     ```
     @GET
     ```

   Os parâmetros de cada método também devem ser anotados, para relacionar os parâmetros contidos no `@Path` com os da função, desta maneira para ambas funções:

   ```
   public Calculadora
         somarInt(@PathParam("a") int a, @PathParam("b") int b)
   ```

Por fim, deve-se importar as seguintes definições, referentes aos objetos utilizados acima:

```
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.PathParam;
import javax.ws.rs.core.MediaType;
```

5. **Primeira compilação:** nesse ponto, compile o código produzido até aqui utilizando o comando abaixo. Note que essa compilação e todas subsequentes devem ser feitas a partir do diretório pai do diretório `webservice`. É necessário compilar juntamente a biblioteca do Jersey através do argumento *classpath* (-cp).

```
%> javac -cp "jersey-bundle-1.19.jar" webservice/*.java
```

Importante: Atenção para o estilo das aspas no comando de compilação.

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

6. **Implemente o servidor HTTP que irá prover o serviço:** o arquivo `Servidor.java` contém o método `main`, o qual deverá instanciar o servidor HTTP sobre o qual será provido o web service da calculadora. Primeiramente, especifica-se o endereço e porta sobre o qual o servidor irá esperar por requisições:

```
URI endpoint =
 UriBuilder.fromUri("http://localhost/").port(9000).build();
```

Para indicar ao servidor quais serviços devem ser providos, utiliza-se a classe `ResourceConfig`, a qual busca no pacote métodos e classes anotados com `@Path` (passo 5), interpretando e instanciando os métodos (seus parâmetros e retorno) desejados:

```
ResourceConfig calculadora_rc =
  new PackagesResourceConfig("webservice");
```

Em seguida, instancia-se o servidor HTTP, passando como parâmetro a URI e os recursos a serem providos. Fique atento pois será necessário realizar o tratamento, através de `try/catch`, da exceção `IOException`.

```
HttpServer server =
       HttpServerFactory.create(endpoint, calculadora_rc);
server.start();
```

Importe as seguintes definições necessárias no `Servidor.java`:

```
import java.io.IOException;
import java.net.URI;
import javax.ws.rs.core.UriBuilder;
import
com.sun.jersey.api.container.httpserver.HttpServerFactory;
import com.sun.jersey.api.core.PackagesResourceConfig;
import com.sun.jersey.api.core.ResourceConfig;
import com.sun.net.httpserver.HttpServer;
```

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

```
%> javac -cp "jersey-bundle-1.19.jar" webservice/*.java
```

8. **Dispare o servidor**:

```
%> java -cp ".:jersey-bundle-1.19.jar" webservice.Servidor
```

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 WADL gerado descrevendo a interface para o web service poderá ser visto abrindo-se o seguinte link no browser:

`http://127.0.0.1:9000/application.wadl`

Este WADL 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 e JSON. Dessa forma, poderíamos escrever um cliente em qualquer linguagem. Porém, nessa aula vamos continuar utilizando Java.

Com o servidor executando, já é possível testar os métodos REST através do navegador, utilizando como URL a localização do servidor (`http://127.0.0.1:9000/`) concatenado, nesta ordem, com: o caminho do serviço (`/calculadora`); o caminho do método desejado (i.e., `/somarInt`); e os argumentos desejados, tal como declarado no passo 4 (i.e., `/2/3`). O retorno deve ser, em formato JSON, os atributos da classe `Calculadora`.

9. **Escreva o programa cliente:** neste ponto, você deverá implementar o código do cliente. Isso será feito na classe chamada `Cliente.java`. Dentro da função `main`, crie uma instância cliente padrão de web service (também provida pela biblioteca Jersey), passando para a mesma a URL raiz do web service `calculadora` (URL do servidor concatenado com o `@Path` da classe `Calculadora`):

```
ClientConfig config = new DefaultClientConfig();
Client cliente = Client.create(config);
WebResource servico =
  cliente.resource("http://localhost:9000/calculadora");
```

Crie um serviço cliente, concatenando o caminho adicional para tal serviço, de acordo com o `@Path` do método definido na classe `CalculadoraRest.java`, assim como os parâmetros da operação desejada (isto é, os operandos 'a' e 'b' – lembre-se de declará-los), sempre no formato de URI (i.e., separados por '/'):

```
WebResource servicoSomarInt =
  servico.path("somarInt").path(a + "/" + b);
```

Para realizar a consulta ao web service, deve-se chamar o método `Get` do objeto instanciado via a instância do `WebResource` acima. Além disso, indica-se ao mesmo que se deseja receber uma resposta em formato JSON (poderíamos, caso especificado na classe `CalculadoraRest.java`, receber em outro formato, como XML). O retorno desta operação é guardado pela classe `ClientResponse`:

```
ClientResponse respostaSomarInt =
  servicoSomarInt.accept(MediaType.APPLICATION_JSON).
  get(ClientResponse.class);
```

Por fim, recupera-se a resposta recebida pelo serviço através de uma conversão para `String` (para facilitar a impressão da mesma):

```
String respostaJsonSomarInt =
  respostaSomarInt.getEntity(String.class);
System.out.println("Resposta da adicao: " +
  respostaJsonSomarInt);
```

Importe as definições utilizadas na aplicação cliente:

```
import javax.ws.rs.core.MediaType;
```

```
import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;
import com.sun.jersey.api.client.config.ClientConfig;
import com.sun.jersey.api.client.config.DefaultClientConfig;
```

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

```
%> javac -cp "jersey-bundle-1.19.jar" webservice/Cliente.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 ".:jersey-bundle-1.19.jar" webservice.Cliente
```

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

Yay! Se você chegou até aqui, well done! Faça o envio do relatório do programa no Moodle, e seu objetivo na atividade de programação guiada estará cumprido.

Mas espere!!! Ainda há uma serie de aspectos interessantes que podem ser explorados utilizando aplicações Java Web Services. Caso você prefira continuar trabalhando nos códigos, 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.
- Tente fazer com que algum dos serviços retorne, ao invés de JSON, um XML contendo a resposta do web service. Note que a aplicação cliente também deve explicitar que deseja receber a resposta em formato XML.
- Altere os métodos de modo que, ao invés de receberem os parâmetros pelo método HTTP GET, estes sejam passados via HTTP POST através de um documento JSON.