Programação Guiada 2 - Webservices Java

Fábio de Azevedo Gomes - Cartão 00287696
Octavio Arruda - Cartão 00246622

October 2020

1 Implementação

Disponível no repositório utilizado

1.1 `class` CalculadoraRest

Para esta classe, utilizamos a anotação `@Path` para indicar ao servidor qual o caminho para as operações expostas implementadas (neste caso "calculadora"). Implementamos então 2 métodos de mesmo esqueleto, apenas com nomes diferentes e parâmetros diferentes passados ao construtor de `Calculadora`:

1.1.1 somarInt e multiplicarInt

```
@Path("somarInt/{a}/{b}")
@Produces(MediaType.APPLICATION_JSON)
@GET
public Calculadora somarInt(@PathParam("a") int op1,
    @PathParam("b") int op2)
{
    return new Calculadora(op1, op2, "+");
}
```

Fora do método precisamos colocar 3 anotações:

- `@Path`, para indicar ao servidor como acessar o método, de mesmo nome que o método correspondente.
- `@Produces`, para indicar ao servidor qual o retorno deste método. Ambos os métodos retornam um `JSON` da instância de `Calculadora` gerada.
- `@GET`, para indicar ao servidor quais operações podem ser realizadas sobre este método (neste caso apenas `GET` em ambos os métodos).

e na assinatura do método também precisamos informar ao servidor a quais parâmetros as variáveis estão associadas, através da anotação `@PathParam`.

1.2 `class` Servidor

Na classe servidora implementamos apenas o corpo da função `main`, cujo objetivo é procurar pelos métodos disponíveis e oferecê-los aos clientes.

```
// Specify address and port the server will listen at
URI endpoint = UriBuilder.fromUri("http://localhost/").
    port(9000).build();

// Seach for all available procedures (Those marked with
    @Path in other sources)
ResourceConfig calculator_rc = new PackagesResourceConfig
    ("webservice");
```

Neste primeiro trecho de código especificamos qual a `URI` e porta na qual o servidor irá escutar, e obtemos os recursos disponibilizados pela classe `CalculadoraRest`.

```
try
{
    // Create a server with specified endpoint and
        resources
    HttpServer server = HttpServerFactory.create(endpoint
        , calculator_rc);

    // Log server start
    LOGGER.log(Level.INFO, "Starting server normally");

    // Start the server
    server.start();

}
catch (IOException e)
{
    // Not able to start server
    LOGGER.log(Level.SEVERE, "Server unable to start: ",
        e);
}
```

Neste segundo momento criamos, de fato, o servidor, associando os recursos providos ao mesmo, e finalmente o iniciamos.

1.3 `class` Cliente

Na aplicação cliente também implementamos o método `main`, com objetivo de instanciar um cliente e realizar uma requisição ao serviço disponibilizado pelo

servidor.

```
// Create default client instance
ClientConfig config = new DefaultClientConfig();
Client cliente = Client.create(config);

// Create resource instance provided by the server
WebResource servico = cliente.resource("http://localhost
    :9000/calculadora");
```

De início, criamos uma nova instância do cliente, e também uma instância do serviço provido no `Path` "calculadora", como especificado no servidor.

```
// Get procedure name
operation = availableOperations.get(operation);

// Execute remote call based on operation
serviceProcedure = servico.path(operation).path(op1 + "/"
     + op2);

// Call resource GET to obtain instantiated object
procedureResponse = serviceProcedure.accept(MediaType.
    APPLICATION_JSON).get(ClientResponse.class);

// Extract json answer into a string
jsonResponse = procedureResponse.getEntity(String.class);

// Print answer to screen
System.out.println("Result: " + jsonResponse);
```

Procuramos então pela operação informada pelo usuário ao iniciar o cliente (soma ou multiplicação) e tentamos obter o serviço provido, realizando um *append* da operação e operandos ao serviço criado anteriormente.

Realizamos, então, um `GET` ao servidor, e obtemos uma resposta no formato `ClientResponse`, a qual é convertida para o formato `JSON` e mostrada na tela ao usuário.

2 Demonstração da execução

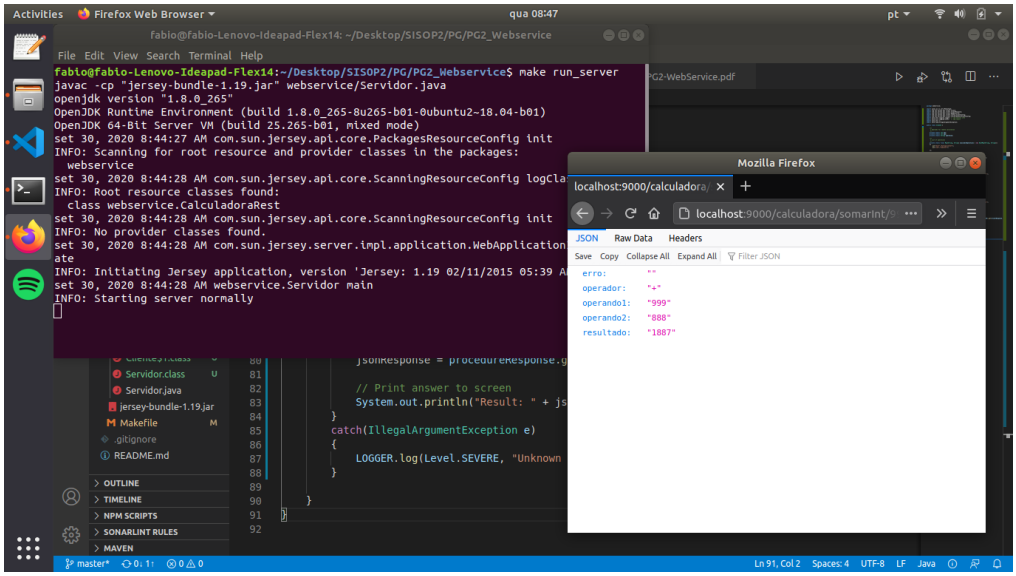

Figure 1: Objeto resultante da operação de soma visualizado no browser

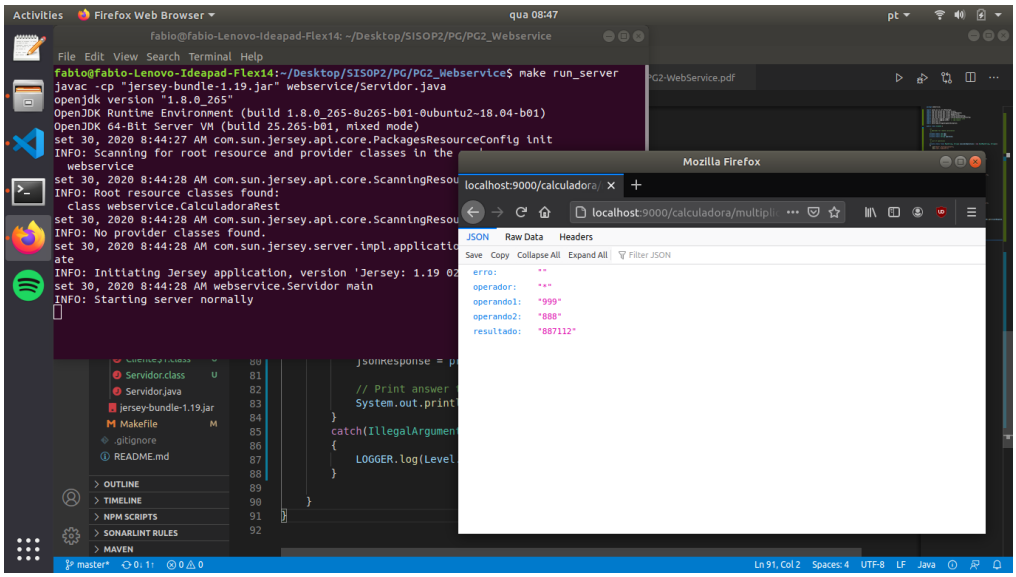

Figure 2: Objeto resultante da operação de multiplicação visualizado no browser

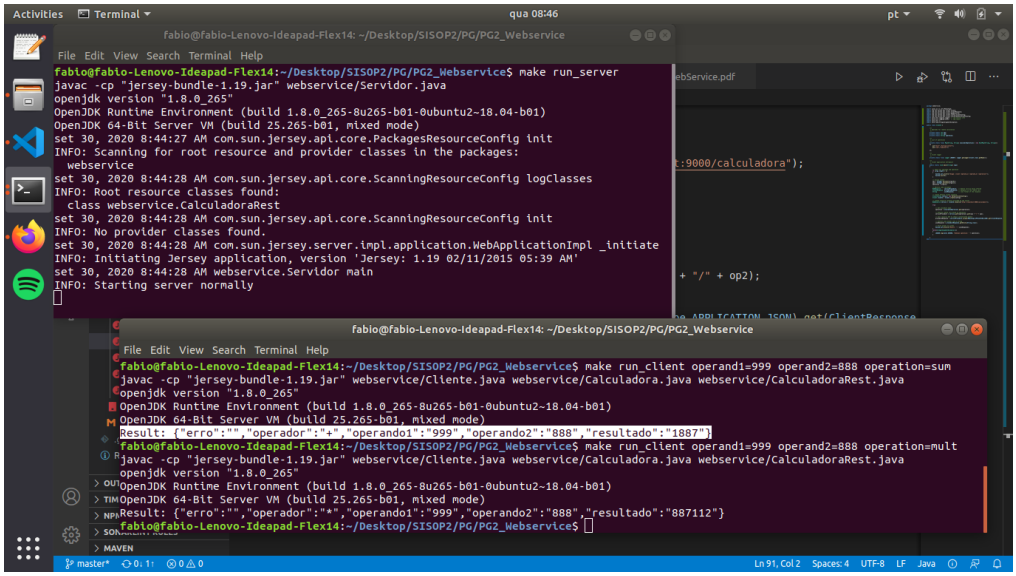

Figure 3: Resultado recebido pela aplicação cliente para a soma

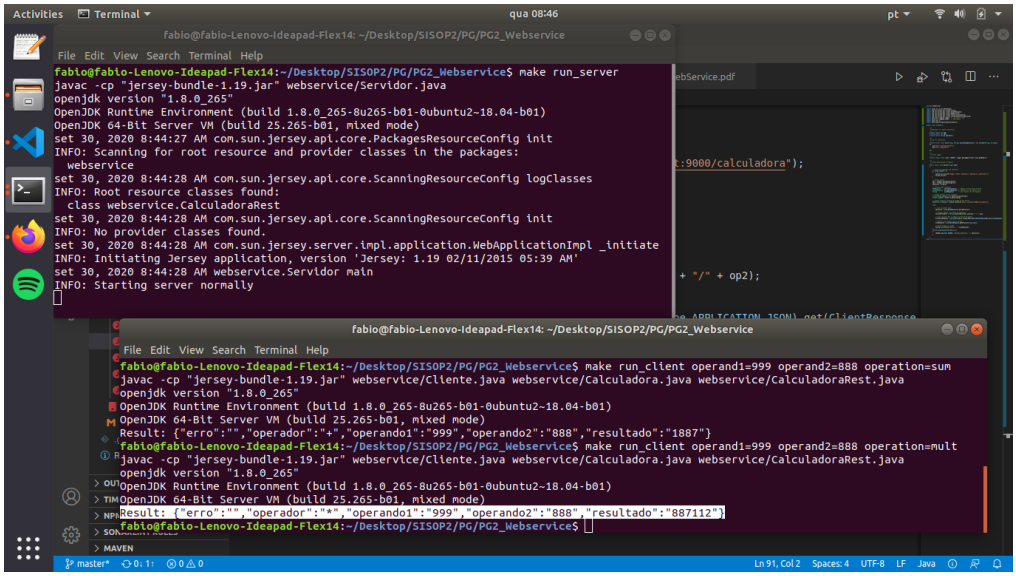

Figure 4: Resultado recebido pela aplicação cliente para a multiplicação

3 Observações

Novamente algumas dificuldades surgiram quanto ao compilador `javac`, o qual estava na versão JDK 11. Feitas algumas pesquisas, descobrimos que os `imports javax.xml.bind` foram **removidos** completamente desta versão, estando **deprecated** a partir da versão 9. Precisamos então baixar uma versão mais antiga do compilador, JDK 8, para termos acesso a estes `imports`