

TRABALHO DE INTEROPERABILIDADE

Aluno: Bruno Silva

Sistema e Software:

Windows 10;

Eclipse IDE for Java Developers (includes Incubating components)

Version: 2020-06 (4.16.0)

Build id: 20200615-1200

Link para GitHub: <https://github.com/brunosilvaifce/SD2020.1>

Prática YAML

No projeto temos três arquivos: O cliente `CalculadoraYAMLClientSocket.java`, o servidor `CalculadoraYAMLServerSocket.java` e a classe `Equacao.java` que será enviada através do YAML, Imagem 1.

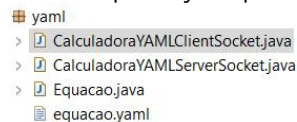


Imagem 1 Lista de arquivos do projeto

O processo de execução é parecido com o trabalho anterior com algumas mudanças.

Existe uma restrição na criação da equação que é o uso apenas de um algarismo no operador.

A classe equação é composta por duas listas de algarismo e outra de operadores, Imagem 2.

```
public class Equacao {  
    List<String> operacoes = new ArrayList<String>();  
    List<String> constantes = new ArrayList<String>();  
  
    public List<String> getOperacoes() {  
        return operacoes;  
    }  
  
    public void setOperacoes(List<String> operacoes) {  
        this.operacoes = operacoes;  
    }  
  
    public void inseriOperacoes(String arg0) {  
        this.operacoes.add(arg0);  
    }  
  
    public List<String> getConstantes() {  
        return constantes;  
    }  
  
    public void setConstantes(List<String> constantes) {  
        this.constantes = constantes;  
    }  
  
    public void inseriConstantes(String arg0) {  
        this.constantes.add(arg0);  
    }  
}
```

Imagem 2 Classe Equacao

Houve um problema entre a serialização do YAML e os operadores aritméticos, então criei um tradutor dos símbolos aritméticos, Imagem 3. No servidor transformo o inverso, Imagem 4. A linha `Equacao equac = new YAML(new Constructor(Equacao.class)).LoadAs(expressao, Equacao.class);` faz o parser da string para a classe.

```

static String fazSerializacao( String expressao ) {
    Equacao equac = new Equacao();
    for(int i=0;i<expressao.length();i++) {
        if (Character.isDigit(expressao.charAt(i))) {
            equac.inseriConstantes( ""+expressao.charAt(i) );
        }
        else {
            if (expressao.charAt(i) == '+' ) {
                equac.inseriOperacoes( "o" );//+expressao.charAt(i)
            }
            if (expressao.charAt(i) == '-' ) {
                equac.inseriOperacoes( "s" );//+expressao.charAt(i)
            }
            if (expressao.charAt(i) == '/' ) {
                equac.inseriOperacoes( "d" );//+expressao.charAt(i)
            }
            if (expressao.charAt(i) == '*' ) {
                equac.inseriOperacoes( "m" );//+expressao.charAt(i)
            }
        }
    }
}

```

Imagem 3 Transformador no cliente

```

static Equacao desFazSerializacao( String expressao ) {
    Equacao equac = new Yaml(new Constructor(Equacao.class)).loadAs(
        expressao, Equacao.class);

    List<String> operacoes = new ArrayList<String>();
    for(String str: equac.getOperacoes()) {
        if (str.equals("o")) {
            operacoes.add( "+" );//+expressao.charAt(i)
        }
        if (str.equals("s")) {
            operacoes.add( "-" );//+expressao.charAt(i)
        }
        if (str.equals("d")) {
            operacoes.add( "/" );//+expressao.charAt(i)
        }
        if (str.equals("m")) {
            operacoes.add( "*" );//+expressao.charAt(i)
        }
    }
}

```

Imagem 4 Transformador no servidor

Para montar a string no YAML é observado na imagem 5.

```

YamlMapping equacion = Yaml.createYamlDump(
    equac
).dumpMapping();

return equacion.toString();

```

Imagem 5 Função para montar string no YAML

O envio é normal via socket, ver imagem 6.

```

String df = fazSerializacao(expressao);
System.out.println("String = "+df);

socketSaidaServer.writeUTF(df);
socketSaidaServer.flush();

```

Imagem 6 Enviando via socket

No servidor, sigo o passo de fazer a tradução dos operadores e depois montar a equação a partir das duas listas, Imagem 7. Em seguida chamo a função do sistema, *shell.evaluate*, ver imagem 8, para calcular a equação.

```

Equacao equac = desFazSerializacao( equaStr );
String formula = montarEquacao(equac);
System.out.println ("Formula: " + formula);

```

Imagem 7 Figura com os passos para tradução, montagem de equação a partir da lista e execução da operação.

```

result = shell.evaluate(formula);

```

Imagem 8 Função do sistema para executar a equação

A transmissão do resultado segue o mesmo esquema do trabalho anterior.

Prática JSON

No projeto temos três arquivos: O cliente CalculadoraJSONClientSocket.java, o servidor CalculadoraJSONServerSocket.java e a classe Equacao.java que será enviada através do JSON, Imagem 9.



- >  CalculadoraJSONClientSocket.java
- >  CalculadoraJSONServerSocket.java
- >  Equacao.java

Imagem 9 Lista de arquivos do projeto

O processo de execução é parecido com o trabalho anterior com algumas mudanças. Existe uma restrição na criação da equação que é o uso apenas de um algarismo no operador. A classe equação é composta por duas listas de algarismo e outra de operadores, Imagem 10.

```
public class Equacao {  
    List<String> operacoes = new ArrayList<String>();  
    List<String> constantes = new ArrayList<String>();  
    public List<String> getOperacoes() {  
        return operacoes;  
    }  
    public void setOperacoes(List<String> operacoes) {  
        this.operacoes = operacoes;  
    }  
    public void inseriOperacoes(String arg0) {  
        this.operacoes.add(arg0);  
    }  
    public List<String> getConstantes() {  
        return constantes;  
    }  
    public void setConstantes(List<String> constantes) {  
        this.constantes = constantes;  
    }  
    public void inseriConstantes(String arg0) {  
        this.constantes.add(arg0);  
    }  
}
```

Imagem 10 Classe Equacao

No cliente fazemos a montagem das strings, Imagem11. Utilizamos o construtor JSONObject com entrada de parâmetro nossa classe equac(Equacao) para criar um objeto JSON que permite depois extrair a string desta classe, Imagem 11.

```
static String fazSerializacao( String expressao ) {  
    Equacao equac = new Equacao();  
    for(int i=0;i<expressao.length();i++) {  
        if (Character.isDigit(expressao.charAt(i))) {  
            equac.inseriConstantes( ""+expressao.charAt(i) );  
        }  
        else {  
            equac.inseriOperacoes( ""+expressao.charAt(i) );  
        }  
    }  
    JSONObject jo = new JSONObject(equac);  
    return jo.toString();  
}
```

Imagem 11 Monta os arrays e depois cria a String JSON.

A transmissão é via socket como visto no trabalho anterior, Imagem 12.

```
String df = fazSerializacao(expressao);  
System.out.println("String = "+df);  
socketSaidaServer.writeBytes(df+"\n");  
socketSaidaServer.flush();
```

Imagem 12 Transmissão do cliente para o servidor

No servidor o processo de recebimento e montagem é realizado, Imagem 13.

```
String equaStr = socketEntrada.readLine().toString();

System.out.println ("equaStr: " + equaStr);
Equacao equac = desFazSerializacao( equaStr );

String formula = montarEquacao(equac);
System.out.println ("Formula: " + formula);
```

Imagem 13 Ordem de recebimento e montagem da equacao

O processo de string para classe é feito na Imagem 14, neste processo as listas são extraídas do parser jsonParser, que foi construído com a expressão transmitida.

```
static Equacao desFazSerializacao( String expressao ) {

    Equacao equac = new Equacao();

    JSONTokener jsonParser = new JSONTokener(expressao);

    JSONObject content = (JSONObject) jsonParser.nextValue();
    JSONArray list = content.getJSONArray("operacoes");
    JSONArray list2 = content.getJSONArray("constantes");

    for (Object object : list.toList()) {
        equac.inseriOperacoes(object.toString());
    }

    for (Object object : list2.toList()) {
        equac.inseriConstantes(object.toString());
    }

    return equac;
}
```

Imagem 14 Processo de extração da classe

A função para montar a equação está presente em ambas as práticas JSON e YAML, Imagem 15.

```
static String montarEquacao(Equacao equac) {
    String resultado = "";
    resultado = resultado + equac.getConstantes().get(0);
    int j=0;
    for(int i = 1; i<= equac.getConstantes().size() && j<equac.getOperacoes().size();i=i+1, j=j+1){
        resultado= resultado + equac.getOperacoes().get(j);
        resultado= resultado + equac.getConstantes().get(i);
    }

    return resultado;
}
```

Imagem 15 Função para montar a equação está presente no YAML e no JSON

Depois de montar a equação a função do sistema é executada, *shell.evaluate*, ver imagem 16, para calcular a equação.

```
result = shell.evaluate(formula);
```

Imagem 16 Função do sistema para executar a equação

A transmissão do resultado segue o mesmo esquema do trabalho anterior.