Fazendo integrações entre micro-serviços em api’s

Introdução

Uma API, em uma definição formal, é um conceito para um **conjunto de rotinas e padrões** estabelecidos por um **software** para a **utilização das suas funcionalidades por outros softwares**.

O conceito de API nada mais é do que uma **forma de comunicação entre sistemas utilizando JSON ou XML**. Ou seja, elas permitem a **integração entre dois ou mais sistemas**, em que um deles fornece informações e serviços que podem ser utilizados pelo outro, sem a necessidade de algum dos sistemas conhecer detalhes de sua implementação.

Em outras palavras, é uma forma bem segura pela qual dois softwares trocam dados. Assim, as APIs cuidam dessa comunicação em tempo real.

Definindo uma URL em Java

Uma URL é uma referência ou um endereço para um recurso na web. Simplificando, o código Java que se comunica pela rede utiliza a classe java.net.URL para representar os endereços dos recursos. A plataforma Java vem com suporte de rede embutido, agrupado no pacote java.net, para importá-lo:

```
import java.net.*;
```

Definindo uma URL em Java

Vamos criar um objeto java.net.URL usando seu construtor e passando uma String representando o endereço web que queremos acessar:

```
URL url = new URL(String.format("https://viacep.com.br/ws/%cep/json/", cep);
```

O método .format da classe String nos permite concatenar a String passada por no primeiro parâmetro com a segunda String passada no segundo parâmetro. Essa concatenação é feita utilizando o pseudônimo %cep presente na primeira String com o atributo cep. Vamos supor que 'cep' tenha por valor "01001000", a String gerada e passada para a URL seria: `https://viacep.com.br/ws/01001000/json/`

Referencia: https://www.baeldung.com/java-url

Abrindo uma conexão

A classe UrlConnection nos permite realizar requisições HTTP básicas sem o uso de nenhuma biblioteca adicional. Todas as classes que precisamos fazem parte do pacote java.net. Para abrirmos uma conexão, vamos criar um objeto java.net.URLConnection utilizando o método .openConnection( ) do Objeto URL criado anteriormente:

```
URLConnection connection = url.openConnection( );
```

Referencia: https://docs.oracle.com/javase/7/docs/api/java/net/URLConnection.html

Entrada de dados

Um programa que precise ler algum dado de algum local (uma fonte) precisa de um **InputStream**, por outro lado um programa que precise escrever um dado em algum local (destino) precisa de um **OutputStream**. (Iremos abordar somente o InputStream durante a aula de hoje).

Em Java, a palavra “Stream” representa um fluxo de dados, seja para leitura ou para escrita. Imagine um Stream como uma conexão com uma fonte ou destino de dados, onde esses dados podem ser passados via byte ou character. Por exemplo, um arquivo de texto pode representar um Stream, onde o seu programa irá ler esse Stream via byte ou character usando InputStream.

InputStream

A classe InputStream nos permite receber um fluxo de dados em binário. Esta classe faz parte do pacote java.io, para importá-lo:

```
import java.io.*;
```

Recebendo um fluxo de dados (InputStream)

Para recebermos um fluxo de dados (em bytes), vamos criar um objeto java.io.InputStream utilizando o método .getInputStream() do Objeto URLConnection criado anteriormente:

```
InputStream input = connection.getInputStream();
```

Referencia: https://docs.oracle.com/javase/7/docs/api/java/io/InputStream.html

InputStreamReader

A classe InputStreamReader funciona como uma ponte entre o fluxo de bytes e fluxo de caracteres. Isso ocorre porque o InputStreamReader lê bytes do fluxo de entrada como caracteres.

Por exemplo, alguns caracteres exigiam 2 bytes para serem armazenados no armazenamento. Para ler tais dados, podemos usar o InputStreamReader que lê os 2 bytes juntos e converter no caractere correspondente.

Referencia: https://www.programiz.com/java-programming/inputstreamreader

BufferedReader

A classe BufferedReader simplifica a leitura de texto de um fluxo de entrada (InputStreamReader) de caracteres. Ele armazena os caracteres em buffer para permitir a leitura eficiente dos dados de texto. Esta classe também faz parte do pacote java.io, para importá-lo:

```
import java.io.*;
```

Referencia: https://www.baeldung.com/java-buffered-reader

“Lendo” um fluxo de caracteres

Vamos criar um objeto java.io.BufferedReader usando seu construtor e passando uma nova instância de um java.io.InputStreamReader que fará a conversão do fluxo de dados em bytes do InputStream definido anteriormente, em um fluxo de caracteres que queremos acessar:

```
BufferedReader buffer = new BufferedReader(new InputStreamReader(input, “UTF-8”));
```

StringBuilder

A classe StringBuilder fornece uma matriz de utilitários para a construção de strings. Essa matriz de utilitários facilitam o trabalho de manipulação e criação de strings personalizadas.

Referencia: https://www.baeldung.com/java-strings-concatenation

A biblioteca GSON

Uma API pode se comunicar utilizando JSON ou XML, quando queremos nos comunicar com uma API utilizando o formato JSON, é necessário “lermos” o JSON de entrada de dados gerada pela API que estamos estabelecendo a conexão. Para isso, vamos utilizar a biblioteca GSON, do Google.

Gson é uma biblioteca Java que pode ser usada para converter objetos Java em sua representação JSON. Também pode ser usado para converter uma string JSON em um objeto Java equivalente. O Gson pode trabalhar com objetos Java arbitrários, incluindo objetos pré-existentes dos quais você não possui o código-fonte.

Referencia: https://github.com/google/gson

Adicionando a biblioteca GSON ao Pom.xml

Para adicionarmos a biblioteca a nossa API, vamos adicionar a seguinte dependência ao Pom.xml

```
<!-- https://mvnrepository.com/artifact/com.google.code.gson/gson -->
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
</dependency>
```