

01

## Integrando com o Via Cep

### Transcrição

Mais uma das peculiaridades que temos nos documentos brasileiros é o trabalho com o CEP no Java. Tratando esse CEP, podemos trabalhar em uma base de dados ou passar para o front-end. No segundo caso, quando preenchemos um formulário com CEP, os campos de estado, cidade, bairro e endereço são preenchidos automaticamente.

**endereço de cadastro**

Tipo de endereço*	Apartamento
CEP*	72465-4
Não sei meu CEP	
Endereço*	QUADRA Quadra 40
Número*	
Complemento	
Informações de referência*	
Bairro*	Setor Leste (Gama)
Cidade*	Brasília
Estado*	DF

Se mudarmos o CEP, ele adequará os outros dados automaticamente.

### endereço de cadastro

Tipo de endereço*	Apartamento
CEP*	82010-340
Não sei meu CEP	
Endereço*	RUA Tobias de Macedo Júnior
Número*	<input type="text"/>
Complemento	<input type="text"/>
Informações de referência*	<input type="text"/>
Bairro*	Santo Inácio
Cidade*	Curitiba
Estado*	PR

Quem é do ramo da programação sabe que essa chamada para capturar os dados não é tão automática assim. O site dos [Correios \(<http://www.correios.com.br/para-voce>\)](http://www.correios.com.br/para-voce) disponibiliza uma pesquisa por CEP. Quando preenchemos, com um CEP, ela nos retorna o endereço correspondente, com bairro e localidade.

The screenshot shows the official website of Brazil's postal service, Correios. At the top, there's a navigation bar with links for Headmouse, Teclado Virtual, Contraste, Tamanho padrão, Ir ao conteúdo, English, Español, Acesso ao idCorreios, Fale com os Correios, Outros sites, and Correios de A a Z. Below the header is a search bar labeled "O que você procura?". The main content area features the Correios logo and a yellow banner with the text "Serviço de Impresso Agora, vender livros ficou mais fácil!". To the right, there's a section for "Busca CEP ou Endereço" where users can enter a CEP (e.g., 82010340) and a "Rastreamento de objetos" section. On the left, a sidebar titled "Você" lists links for Envio, Recebimento, Compra, Consultas e solicitações, Serviços financeiros, and Precisa de ajuda?

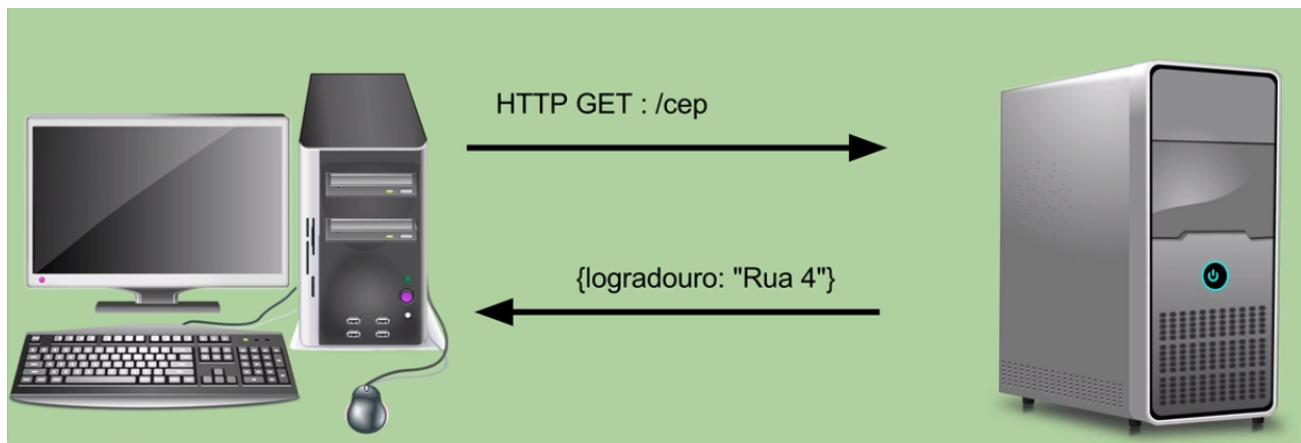
This screenshot shows the results of a CEP search for 82010-340. The search interface is similar to the one above, with the "Sistemas" tab selected. The results are displayed in a table:

Logradouro/Nome:	Bairro/Distrito:	Localidade/UF:	CEP:
Rua Tobias de Macedo Júnior	Santo Inácio	Curitiba/PR	82010-340

Below the table, there are links for Anterior, Próxima, and Nova Consulta, along with a page number indicator (1 a 1 de 1).

E como pegar os dados daqui, que é um HTML, e passo para um Java? Teríamos que fazer um `parse`, procurar onde está o `table` do logradouro (a `<div>logradouro</div>`), a do Bairro e assim por diante, picotando o HTML. Seria um pouco demorado fazer dessa maneira, por isso costumamos usar REST, fazendo a chamada em algum webservice que nos retorne um `.json` ou um `.xml`, que nos trazem dados mais fáceis de tratarmos. Se usarmos o HTML, virão junto com os dados que nos interessam o cabeçalho, o menu lateral... Temos aqui na Alura um [curso](#) (<https://cursos.alura.com.br/course/webservices-rest-com-jaxrs-e-jersey>) que ensina a consumir esse tipo de dado via REST, usando o JAX-RS e o Jersey. Vou explicar aqui o básico do REST, como ele busca um dado e como o servidor nos retorna.

Estando em uma aplicação, em um celular, desktop ou navegador qualquer, faz-se uma chamada. A chamada é feita para o servidor em cima do protocolo HTTP, usando os verbos `get`, `post`, `delete` ou `update`. O servidor nos retorna só os dados solicitados no formato que especificamos, por exemplo um `.json`. O mais interessante é que o dado já vem tratado, diferente de todo um HTML, tornando muito mais fácil usá-lo no Java.



Tendo em mente que queremos o dado via `.json`, não podemos usar o site dos Correios, que só disponibiliza o HTML. Usaremos o [ViaCEP](http://viacep.com.br/) (<http://viacep.com.br/>), um webservice gratuito que disponibiliza os CEPs do correio via `.json`.



Procurando um [webservice](#) gratuito e de alto desempenho para consultar Códigos de Endereçamento Postal (CEP) do Brasil? Utilize o serviço, melhore a qualidade de suas aplicações web e colabore para manter esta base de dados atualizada.

### Acessando o webservice de CEP

Para acessar o webservice, um CEP no formato de `{8}` dígitos deve ser fornecido, por exemplo: `"01001000"`.

Após o CEP, deve ser fornecido o tipo de retorno desejado, que deve ser `"json"`, `"xml"`, `"piped"` ou `"query"`.

Exemplo: [viacep.com.br/ws/01001000/json/](http://viacep.com.br/ws/01001000/json/)

### Validação do CEP

Quando consultado um CEP de formato inválido, por exemplo: `"950100100"` (9 dígitos), `"95010A10"` (alfanumérico), `"95010 10"` (espaço), o código de retorno da consulta será um **400** (Bad Request). Antes de acessar o webservice, valide o formato do CEP e certifique-se que o mesmo possua `{8}` dígitos. Exemplo de como validar o formato do CEP em javascript está disponível nos exemplos abaixo.

Quando consultado um CEP de formato válido, porém inexistente, por exemplo: `"09999999"`, o retorno conterá um valor de `"erro"` igual a `"true"`. Isso significa que o CEP consultado não foi encontrado na base de dados. Veja como manipular este `"erro"` em javascript nos exemplos abaixo.

A pesquisa do CEP tem o seguinte padrão `viacep.com.br/ws/01001000/json/` e deve ser feita na barra do navegador. O número do CEP consta antes do formato desejado. O resultado vem em texto. Só precisamos fazer um `parse` do que QUEREMOS.

```
{
  "cep": "01001-000",
  "logradouro": "Praça da Sé",
  "complemento": "lado ímpar",
  "bairro": "Sé",
  "localidade": "São Paulo",
  "uf": "SP",
  "unidade": "",
  "ibge": "3550308",
  "gia": "1004"
}
```

Caso desejemos os dados no formato .xml , basta alterar o final da URL.

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<xmlcep>
<cep>01001-000</cep>
<logradouro>Praça da Sé</logradouro>
<complemento>lado ímpar</complemento>
<bairro>Sé</bairro>
<localidade>São Paulo</localidade>
<uf>SP</uf>
<unidade/>
<ibge>3550308</ibge>
<gia>1004</gia>
</xmlcep>
```

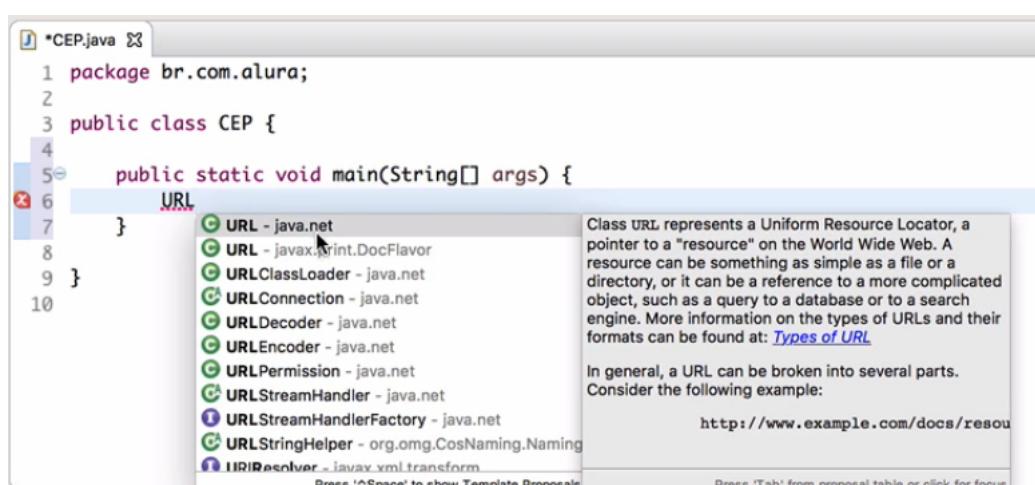
Hoje em dia é mais comum trabalhar com .json , e é o que usaremos. Faremos uma chamada, passando a URL, no Java. Criaremos uma nova classe no nosso pacote (com o botão direito sobre ela New > Class ). chamada CEP . Dentro dela, criaremos o método main para fazer a chamada dentro dele.

```
package br.com.alura

public class CEP {

    public static void main (String[] args) {
        URL
    }
}
```

Como a URL está no java.net , optaremos por ele.



A seguir, colocaremos a URL que queremos chamar, criando uma variável e a colocando em forma de string.

```

package br.com.alura

import java.net.URL;

public class CEP {

    public static void main (String[] args) {
        URL url = new URL("http://viacep.com.br/ws/01001000/xml/");
    }
}

```

O Eclipse nos avisa que há um problema de exceção e nos sugere duas soluções rápidas.



Optaremos por resolver com um `try/catch`.

```

public class CEP {

    public static void main (String[] args) {
        try {
            URL url = new URL("http://viacep.com.br/ws/01001000/xml/");
        } catch (MalformedURLException e){
            e.printStackTrace();
        }
    }
}

```

Agora é preciso abrir a conexão, com o `openConnection()`, que nos retorna um `HttpURLConnection`.

```

public class CEP {

    public static void main (String[] args) {
        try {
            URL url = new URL("http://viacep.com.br/ws/01001000/xml/");
            HttpURLConnection conexao = url.openConnection();
        } catch (MalformedURLException e){

```

```

        e.printStackTrace();

    }

}

}

```

Precisaremos fazer um `cast`, pois a `conexao` na verdade retorna uma `URLConnection`. E aqui também há uma exceção, então precisaremos acrescentar mais um `catch`.

```

public class CEP {

    public static void main (String[] args) {
        try {
            URL url = new URL("http://viacep.com.br/ws/01001000/xml/");
            HttpURLConnection conexao = (HttpURLConnection) url.openConnection();
        } catch (MalformedURLException e){
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

}

```

Então, poderemos configurar nossa conexão, usando `setRequestMethod()`, que recebe uma string. O método que usaremos é o "GET".

```

public class CEP {

    public static void main (String[] args) {
        try {
            URL url = new URL("http://viacep.com.br/ws/01001000/xml/");
            HttpURLConnection conexao = (HttpURLConnection) url.openConnection();
            conexao.setRequestMethod("GET");
        } catch (MalformedURLException e){
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

}

```

Precisamos pegar os dados do servidor com o `getInputStream()`, que é mais um trabalho. No começo do curso, mencionamos que esse é o dia a dia do programador brasileiro. Será que não tem algo mais fácil?

Como muitas pessoas passam por isso, foi criada uma API para facilitar o trabalho, também chamada ViaCEP. Está disponível no [GitHub](https://github.com/gilberto-torrezan/viacep) (<https://github.com/gilberto-torrezan/viacep>). Vamos baixá-la via `<a href=` <https://mvnrepository.com/artifact/com.github.gilberto-torrezan/viacep/1.2.0>

(<https://mvnrepository.com/artifact/com.github.gilberto-torrezan/viacep/1.2.0>) target="blank">Maven, na versão mais atual. Basta copiar o código correspondente da dependência.

The screenshot shows the MVNRepository website. On the left, there's a sidebar with 'Indexed Artifacts (5.75M)' and a line graph showing the number of indexed artifacts over time from 2004 to 2017. Below the graph is a list of 'Popular Categories' including Aspect Oriented, Actor Frameworks, Application Metrics, Build Tools, Bytecode Libraries, Command Line Parsers, Cache Implementations, Cloud Computing, Code Analyzers, and Collections. The main content area shows the 'ViaCEP Java API > 1.2.0' dependency page. It includes a logo of a cartoon character, a brief description 'Java API to access ViaCEP webservices', and a table with details like License (MIT), HomePage (<https://github.com/gilberto-torrezan/viacep>), Date (Dec 05, 2015), and Files (Download (JAR) (10 KB)). It also lists Repositories (Central, Sonatype Releases) and supported build tools (Maven, Gradle, SBT, Ivy, Grape, Leiningen, Buildr). Below this is the Maven dependency code:

```
<!-- https://mvnrepository.com/artifact/com.github.gilberto-torrezan/viacep -->
<dependency>
    <groupId>com.github.gilberto-torrezan</groupId>
    <artifactId>viacep</artifactId>
    <version>1.2.0</version>
</dependency>
```

Include comment with link to declaration

Ele será colado no `pom.xml`, depois da dependência do Moneta.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-:
  <modelVersion>4.0.0</modelVersion>
  <groupId>br.com.alura.brasileirice</groupId>
  <artifactId>Brasileirice</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <dependencies>
    <dependency>
      <groupId>br.com.caelum.stella</groupId>
      <artifactId>caelum-stella-core</artifactId>
      <version>2.1.2</version>
    </dependency>
    <dependency>
      <groupId>org.javamoney</groupId>
      <artifactId>moneta</artifactId>
      <version>1.1</version>
    </dependency>
    <dependency>
      <groupId>com.github.gilberto-torrezan</groupId>
      <artifactId>viacep</artifactId>
      <version>1.2.0</version>
    </dependency>
  </dependencies>

</project>
```

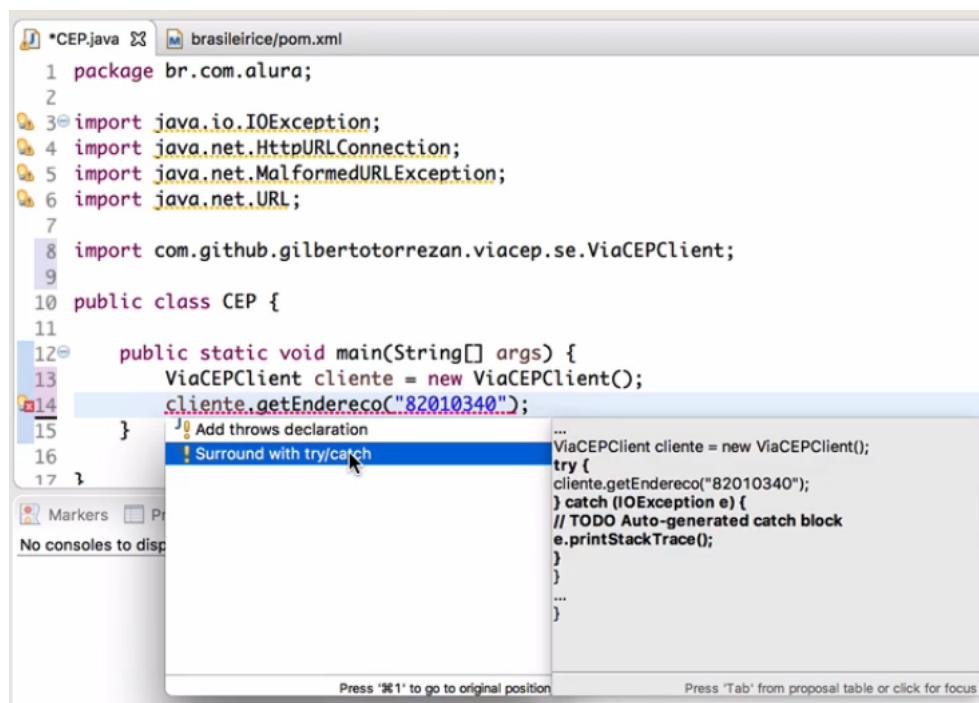
Assim que o ViaCEP aparecer dentre as dependências do Maven, poderemos substituir todo aquele código que digitamos. Usaremos `ViaCEPClient`, pois somos os consumidores de um servidor.

```
public static void main(String[] args) {
    ViaCEPClient cliente = new ViaCEPClient();
}
```

Agora usaremos o `getEndereco(cep)`, para que possamos pegar o endereço com base no CEP.

```
public static void main(String[] args) {
    ViaCEPClient cliente = new ViaCEPClient();
    cliente.getEndereco("82010340");
}
```

O Eclipse novamente nos avisa de um erro. Colocaremos a solicitação dentro de um `try/catch`. O importante é que ele retorna um `ViaCEPEndereco`, que é efetivamente o endereço da chamada, e a partir dele conseguimos pegar o logradouro, o bairro, e as demais informações.



```
public static void main(String[] args) {
    ViaCEPClient cliente = new ViaCEPClient();
    try {
        ViaCEPEndereco endereco = cliente.getEndereco("82010340");
    } catch (IOException E) {
        e.printStackTrace();
    }
}
```

Quando clicamos em `ViaCEPEndereco` com o `Ctrl` pressionado, vemos a classe como um todo.

```
public class ViaCEPEndereco implements Serializable {

    private static final long serialVersionUID = 1L;

    private String cep;
```

```

private String logradouro;
private String complemento;
private String bairro;
private String localidade;
private String uf;
private String ibge;

public String getCep() {
    return cep;
}
}

```

Todos os dados ali listados podem ser trazidos para o método `main`. Vamos imprimir em nosso código o bairro.

```

public static void main(String[] args) {
    ViaCEPClient cliente = new ViaCEPClient();
    try {
        ViaCEPEndereco endereco = cliente.getEndereco(82010340);
        System.out.println(endereco.getBairro());
    } catch (IOException E) {
        e.printStackTrace();
    }
}

```

Agora basta salvar e rodar para obter a informação desejada. O console nos mostrará o seguinte:

Santo Inácio

Vamos testar também com o logradouro.

```

public static void main(String[] args) {
    ViaCEPClient cliente = new ViaCEPClient();
    try {
        ViaCEPEndereco endereco = cliente.getEndereco("82010340");
        System.out.println(endereco.getLogradouro());
    } catch (IOException E) {
        e.printStackTrace();
    }
}

```

Ao rodar, o console nos mostrará:

Rua Tobias de Macedo Júnior

Todo o procedimento complexo é feito pelo simples `getEndereco`, facilitando o trabalho para nós. Quando clicamos em `ViaCEPClient` com o `Ctrl` pressionado, vemos que todo o código que estávamos criando está contido na classe.

```

...
String urlString = getHost() + cep + "/json/";
URL url = new URL(urlString);

```

```
HttpURLConnection URLConnection = (HttpURLConnection) url.openConnection;
try{
    InputStream in = new BufferedInputStream(URLConnection.getInputStream());
    ViaCEPEndereco obj = getService().benFrom(ViaCEPEndereco.class, in);
    if(obj == null || obj.getCep() == null){
        return null;
    }
    return obj;
}
```

A API serve justamente para que não tenhamos que criar todo aquele código. Ela cria o chamado para nós, e fica bem mais simples consumir a API que implementar tudo. Até a próxima!