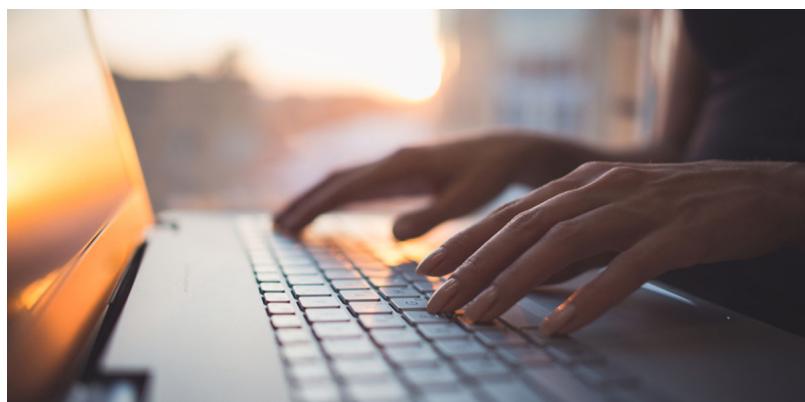


Programação para dispositivos móveis

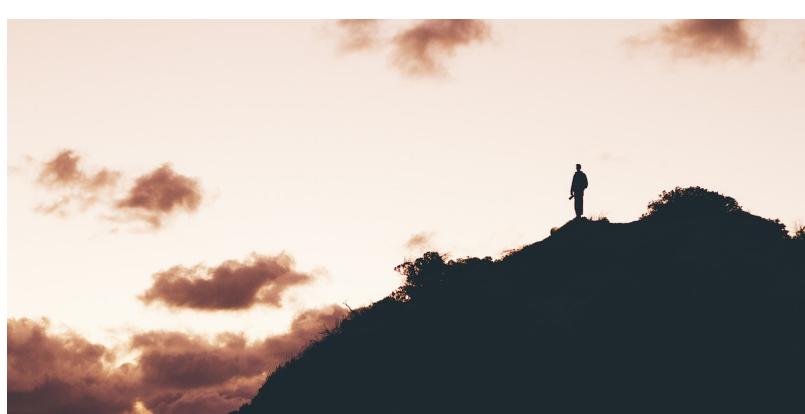
Aula 9: Web Service e o Android

INTRODUÇÃO



Web Service é uma tecnologia muito empregada para comunicação entre sistemas. Esta aula visa apresentar os conceitos básicos referentes à Web Service, bem como seu consumo por aplicativos Android.

OBJETIVOS



Descrever os fundamentos de Web Services.

Demonstrar a implementação de aplicações Android consumindo Web Services.

FUNDAMENTOS WEB SERVICE

Segundo a W3C (World Wide Web Consortium), Web Service é definido como um sistema de software projetado para suportar a interoperabilidade entre máquinas sobre a rede.

Essa tecnologia permite a comunicação entre aplicações, independente de sistema operacional e da linguagem de programação. Com isso, tem sido uma solução muito empregada para integração de sistemas.



PRINCIPAIS VANTAGENS DO USO DE WEB SERVICES

Entre as principais vantagens do uso de Web Services, podemos destacar:

- Reutilização de aplicações existentes
- Uso de padrões abertos como, por exemplo, XML e SOAP
- Interoperabilidade entre plataformas (Sistemas Operacionais) distintas
- Redução no tempo do processo de desenvolvimento

COMO ACESSAR O BANCO DE DADOS DE UM SERVIDOR?

Como estudamos anteriormente, os dispositivos móveis possuem muitas limitações, como o baixo poder de processamento e armazenamento.

Mas se precisássemos de um banco de dados mais robusto que o SQLite, como poderíamos acessar um banco de dados de um servidor?

Um aplicativo Android, até por questões de segurança dos dados, não pode se conectar a um banco de dados do servidor.

Para tanto, a tecnologia Web Service se apresenta como uma solução, sendo de sua responsabilidade acessar o banco de dados, processar as informações e retornar os dados no formato XML ou JSON para o mobile/cliente ler os dados.

VÁRIAS FORMAS DE CRIAR WEB SERVICES

Não é nosso objetivo nesta aula discutir, na íntegra, a tecnologia Web Service, pois não faz parte da ementa de nosso curso.

Porém, precisamos saber que existem várias formas de criar Web Services. Dentre elas, destacamos:

WSDL (Web Services Description Language):

- Descreve o formato das mensagens em XML;
- Tráfego de dados é feito via HTTP;
- Utiliza o protocolo SOAP (Simple Object Access Protocol);
- Maneira clássica de criar Web Services;
- Principais APIs: KSOAP2 e KXML, DOM e SAX;
- Normalmente empregado em serviços financeiros, portais de pagamento, serviços de telecomunicações.

REST (Representational State Transfer):

- Criados geralmente sobre o protocolo HTTP, utilizando os métodos GET, DELETE, POST e PUT como padronização de seu acesso;
- Utiliza os próprios métodos do protocolo HTTP para criar a lógica necessária para o Web Service;
- O formato de retorno mais comum é o JSON (JavaScript Object Notation);
- Principais APIs: Restlet, Jersey, GSON e Jackson;
- Exemplos: serviços de mídias sociais, redes sociais, serviços Web Chat, serviços móveis.

Páginas simples com Get ou Post

- Corresponde a uma página web qualquer, que, ao receber uma requisição GET ou POST, vai retornar os dados no formato XML ou JSON, em vez de retornar a uma página HTML.

EXEMPLO - formatos JSON x XML

```
{"widget": {  
    "debug": "on",  
    "window": {  
        "title": "Sample Konfabulator Widget",  
        "name": "main_window",  
        "width": 500,  
        "height": 500  
    },  
    "image": {  
        "src": "Images/Sun.png",  
        "name": "sun1",  
        "hOffset": 250,  
        "vOffset": 250,  
        "alignment": "center"  
    },  
    "text": {  
        "data": "Click Here",  
        "size": 36,  
        "style": "bold",  
        "name": "text1",  
        "hOffset": 250,  
        "vOffset": 100,  
        "alignment": "center",  
        "onMouseUp": "sun1.opacity = (sun1.opacity / 100) * 90;"  
    }  
}}
```

```
<widget>  
  <debug>on</debug>  
  <window title="Sample Konfabulator Widget">  
    <name>main_window</name>  
    <width>500</width>  
    <height>500</height>  
  </window>  
  <image src="Images/Sun.png" name="sun1">  
    <offsetx>250</offsetx>  
    <offsety>250</offsety>  
    <alignment>center</alignment>  
  </image>
```

```

<text data="Click Here" size="36" style="bold">
<name>text</name>
<hOffset>250</hOffset>
<vOffset>100</vOffset>
<alignment>center</alignment>
<onMouseUp>
    sun1.opacity = (sun1.opacity / 100) * 90;
</onMouseUp>
</text>
</widget>

```

[//docsslide.com.br/technology/consumindo-dados-via-web-service-no-android.html](http://docsslide.com.br/technology/consumindo-dados-via-web-service-no-android.html)

Comparação entre Web Services REST e SOAP

A tabela abaixo demonstra algumas das principais diferenças entre Web Services REST e o SOAP:

REST	SOAP
Estilo de arquitetura de software	Protocolo de comunicação
Abordagem mais simples	Maior complexidade
Menos burocrática	Especificações bem definidas
Fortemente relacionado ao HTTP	HTTP como meio de transporte
Suporte Json, XML, YAML e outros	Suporte XML
Métodos uniformes	Muitos métodos customizados
Amplamente e frequentemente usado	Desempenho não é tão grande quanto o REST
Pode ler cache	Invoca serviços chamando o método RPC

Tabela – Diferenças entre Web Services REST e o SOAP

Análise de desempenho entre REST e SOAP

A imagem abaixo ilustra a diferença do tempo de resposta, em função do tamanho de bytes da mensagem, entre REST e SOAP.

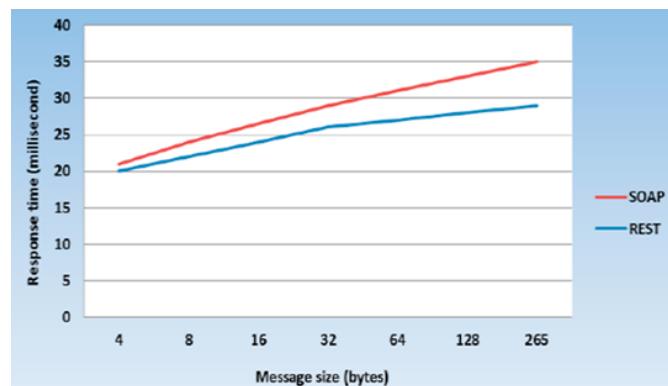


Gráfico – desempenho entre REST e SOAP

https://www.researchgate.net/figure/260632169_fig10_Fig-11-Performance-Analysis-of-REST-VS-SOAP

Os dois tipos de Web Services possuem vantagens e desvantagens.

O gráfico acima demonstra que os Web Services do tipo REST possuem melhor tempo de resposta em aplicações que envolvam manipulações de dados onde resgatam as características da Web.





Por essa razão, é muito empregado no universo mobile, onde o custo na troca de mensagens é bastante elevado como, por exemplo, sites de relacionamento e blogs.

Isso não quer dizer que os Web Services SOAP estão ultrapassados. Eles são muito utilizados em empresas e organizações de grande porte.

O link [http://www.seer.ufrgs.br/reic/article/viewFile/22140/12928 \(glossário\)](http://www.seer.ufrgs.br/reic/article/viewFile/22140/12928) efetua uma excelente comparação entre esses dois tipos de Web Services.

APLICAÇÕES ANDROID E WEB SERVICES

Vamos desenvolver um pequeno projeto, passo a passo, para facilitar o entendimento.

Primeiramente, vamos criar um projeto Android. O nome do nosso projeto é AulaWeb Service_1.

Atenção!

Não descreveremos os passos aqui, pois já fizemos isso na nossa aula 2.

Editar arquivo AndroidManifest.xml

Altere o nosso arquivo AndroidManifest.xml, incluindo a linha <uses-permission android:name="android.permission.INTERNET"/>.

Embora também já tenhamos discutido isso, vale a pena lembrar que precisamos declarar esta permissão porque nosso aplicativo acessará a Internet.

Isto é ilustrado na tela abaixo:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="professoids.com.aulawebservice_1">
    <uses-permission android:name="android.permission.INTERNET" />
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Aula WebService 3"
        android:supportRtl="true">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Implementar o layout

Implemente nosso arquivo de layout, conforme tela:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp">
    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingBottom="16dp"
        tools:context=".MainActivity" />

```

```

<Text>
    android.layout_width="wrap_content" android.layout_height="wrap_content"
    android:textAppearance="@android:attr/textAppearanceLarge" android:layout_marginTop="8dp" />
<Button>
    android.layout_width="220dp" android.layout_height="wrap_content" android:id="@+id/calcDestino"
    android:text="Calcular" android:layout_marginTop="8dp" android:layout_centerHorizontal="true" />
<EditText>
    android.layout_width="220dp" android.layout_height="wrap_content" android:id="@+id/cdServico"
    android:layout_marginTop="8dp" android:layout_alignParentLeft="true" android:layout_centerHorizontal="true" />
<EditText>
    android.layout_width="220dp" android.layout_height="wrap_content" android:id="@+id/cdPrazo"
    android:layout_marginTop="8dp" android:layout_alignParentLeft="true" android:layout_centerHorizontal="true" />
<EditText>
    android.layout_width="220dp" android.layout_height="wrap_content" android:id="@+id/cdEntrega"
    android:layout_marginTop="8dp" android:layout_alignParentLeft="true" android:layout_centerHorizontal="true" />
<Text>
    android.layout_width="wrap_content" android:text="Código da Entrega" android:layout_marginTop="8dp" />
<Text>
    android.layout_width="wrap_content" android:id="@+id/textView2" android:layout_alignParentTop="true" android:layout_centerHorizontal="true" />
</RelativeLayout>

```

Não estamos discutindo a criação desse arquivo, pois esse assunto já foi exaustivamente discutido em nossas aulas 2, 3 e 4, bem como implementado em vários exemplos.

Implemente agora o arquivo Encomenda.java, conforme ilustrado na tela abaixo:

```

package profosvaldo.com.autawebservice_1;

public class Encomenda {
    private String codige, prazoEntrega, entregaDomiciliar, entregasabado;
    public Encomenda() {
    }
    public String getCodige() {
        return codige;
    }
    public void setCodige(String codige) {
        this.codige = codige;
    }
    public String getPrazoEntrega() {
        return prazoEntrega;
    }
    public void setPrazoEntrega(String prazoEntrega) {
        this.prazoEntrega = prazoEntrega;
    }
    public String getEntregaDomiciliar() {
        return entregaDomiciliar;
    }
    public void setEntregaDomiciliar(String entregaDomiciliar) {
        this.entregaDomiciliar = entregaDomiciliar;
    }
    public String getEntregasabado() {
        return entregasabado;
    }
    public void setEntregasabado(String entregasabado) {
        this.entregasabado = entregasabado;
    }
}

```

Esse arquivo representa a encomenda para qual desejamos verificar o prazo da entrega e se há entrega domiciliar aos sábados.

Implementar a classe MainActivity.java

Agora vamos desenvolver a classe principal (MainActivity.java). Implemente, então, o código ilustrado nas telas abaixo.

Devido ao seu tamanho, precisamos capturar em três telas, conforme ilustrado a seguir:

```

package profosvaldo.com.autawebservice_1;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.app.AlertDialog;
import android.util.Log;
public class MainActivity extends AppCompatActivity{
    private Encomenda encomenda = new Encomenda();
    private Button btDestino;
    private EditText cdServico, cdPrazo;
    private String gtservico, gdprazo, gddestino, TAG = "Response";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        btDestino = (Button) findViewById(R.id.calcDestino);
        cdServico = (EditText) findViewById(R.id.cdServico);
        cdPrazo = (EditText) findViewById(R.id.cdPrazo);
        gddestino = (EditText) findViewById(R.id.cdEntrega);
        gddestino.setOnFocusChangeListener(new OnFocusChangeListener() {
            @Override
            public void onFocusChange(View v, boolean hasFocus) {
                if (!hasFocus) {
                    Log.i(TAG, "onFocusChange");
                    btDestino.setEnabled(true);
                }
            }
        });
        btDestino.setOnClickListener(v -> {
            String cdPrazo = cdPrazo.getText().toString();
            String cdServico = cdServico.getText().toString();
            String gddestino = gddestino.getText().toString();
            AsynCallWS task = new AsynCallWS();
            task.execute();
        });
    }
    private class AsynCallWS extends AsyncTask<Void, Void> {
        @Override
        protected void doInBackground(Void result) {
            Log.i(TAG, "doInBackground");
            Log.i(TAG, "Resultatado");
            dialogo.show();
            return null;
        }
        protected void onPostExecute(Void result) {
            Log.i(TAG, "onPostExecute");
            AlertDialog dialogo = new AlertDialog.Builder(MainActivity.this)
            dialogo.setTitle("Resultado")
            dialogo.setMessage(result.toString());
            dialogo.setNegativeButton("Cancelar", null);
            dialogo.show();
        }
    }
    private void calculaPrazo() {
        SoapObject Request = new SoapObject(NAMESPACE, METHOD_NAME);
        Request.addProperty("cdServico", gddestino);
        Request.addProperty("cdPrazo", gdPrazo);
        Request.addProperty("cdEntrega", gddestino);
        Request.addProperty("cdDestino", gddestino);
        SoapSerializationEnvelope soapEnvelope = new SoapSerializationEnvelope(SoapEnvelope.VERB11);
        soapEnvelope.setOutputSoapObject(Request);
    }
}

```

```

String SOAP_ACTION = "http://tempuri.org/CalcPrazo";
    @Override
    protected void onPostExecute(Void result) {
        Log.i(TAG, "onPostExecute");
        AlertDialog dialogo = new AlertDialog.Builder(MainActivity.this)
        dialogo.setTitle("Resultado")
        dialogo.setMessage(result.toString());
        dialogo.setNegativeButton("Cancelar", null);
        dialogo.show();
    }
}

```

```

String SOAP_ACTION = "http://tempuri.org/CalcPrazo";

```

```

String METHOD_NAME = "CalcularPrazo";
String NAMESPACE = "http://ws.correios.com.br/calculador/CalcPrecoPrazo.asmx";
String URL = "http://ws.correios.com.br/calculador/CalcPrecoPrazo.asmx";
try {
    SoapObject Request = new SoapObject(NAMESPACE, METHOD_NAME);
    Request.addProperty("CdServico", getServicio());
    Request.addProperty("CpDestinatario", getDestinatario());
    Request.addProperty("CpOrigem", getOrigem());
    SoapSerializationEnvelope soapEnvelope = new SoapSerializationEnvelope(SoapEnvelope.VERB11);
    soapEnvelope.dotNet = true;
    soapEnvelope.setOutputSoapObject(Request);
    HttpTransportSE transport = new HttpTransportSE(URL);
    transport.call(SOAP_ACTION, soapEnvelope);
    SoapObject response = (SoapObject) soapEnvelope.getResponse();
    response = (SoapObject) response.getProperty("Service");
    response = (SoapObject) response.getProperty("Service");
    encomenda = new Encomenda();
    encomenda.setCodigo(response.getProperty("Cedige").toString());
    encomenda.setRazaoUnidade(response.getProperty("PrazoEntrega").toString());
    encomenda.setEntregaDominical(response.getProperty("Entregabilliar").toString());
    encomenda.setEntregaSábado(response.getProperty("Entregabaldo").toString());
} catch (Exception ex) {
    Log.e(TAG, "ERRO: " + ex.getMessage());
}

```

A primeira parte de nosso programa já é bastante conhecida. A grande novidade de nosso código é a classe AsyncTask.

Seu objetivo é encapsular a implementação das classes Threads e Handler, tornando muito mais fácil a implementação desses conceitos em nosso aplicativo.

PRINCIPAIS MÉTODOS DA CLASSE ASYNCTASK

onPreExecute

- Executado antes da Thread iniciar;
- Muito comum para implementar configurações iniciais do processo, exibir uma janela de progresso ou até mesmo uma mensagem para usuário, como, por exemplo, “Aguarde.”.

onInBackground

- Executado em background por uma Thread;
- Chamado após o método `onPreExecute()` ser finalizado;
- Nesse método deve ser implementado o processamento principal, como, por exemplo, uma busca em um Web Service ou qualquer outro processamento que demande tempo;
- Pode retornar um parâmetro para o método `onPostExecute()`.

onProgressUpdate

- Informa o andamento do processo;
- Podemos, por exemplo, usar para atualizar o status de uma barra de progresso.

onPostExecute

- Chamado logo após o término do método `onInBackground()`;
- Recebe o retorno do método `onInBackground()`;
- Responsável por terminar a execução de nosso processo, como, por exemplo, fechar nossa janela de progresso.

Para maiores informações sobre as classes `AsyncTask`, `Thread` e `Handler`, você poderá consultar os links:

- <https://developer.android.com/reference/android/os/AsyncTask.html> (glossário)
- <https://developer.android.com/reference/java/lang/Thread.html> (glossário)
- <https://developer.android.com/reference/android/os/Handler.html> (glossário)

PRINCIPAIS LINHAS DO CÓDIGO

Apresentada nossa classe `AsyncTask`, vamos analisar as principais linhas de nosso código.

Vamos começar pela linha `private class AsyncCallWS extends AsyncTask<Void,Void,Void>`.

Como você pode observar, a definição da `AsyncTask< Params,Progress, Result >` possui três tipos genéricos que correspondem respectivamente a:

Params

- Tipo de parâmetro de entrada;
- Argumentos que podemos passar por parâmetro ao método `execute (params...)`;
- Devemos usar "Void", se não desejarmos passar algum parâmetro.

Progress

- Tipo de parâmetro de incremento;
- Argumentos usados para informar o progresso do processo;
- Utilizados no método `publishProgress()` e `onProgressUpdate()`.

Result

- Tipo de parâmetro de retorno;
- Usado no método `onInBackground()` como sendo o seu retorno e o `onPostExecute(Result)`.

Em nossa classe interna `AsyncCallWS`, implementamos somente os métodos `doPostExecute()` e o `doInBackground()`.

O `doPostExecute()` possui o código necessário para exibir as informações referentes à Encomenda.

Todos os recursos empregados nesse método, como você pôde observar, já foram estudados em nossas aulas.

O método `doInBackground()` também é bastante simples, pois apenas chama o método `CalcPrazo()`, definido em nossa `MainActivity`.

A grande novidade, então, está no método `CalPrazo()`. Nele está definido todo o código necessário para o consumo das informações de nosso Web Service.

Para começar, observe as linhas de código abaixo:

```
String SOAP_ACTION = "//tempuri.org/CalcPrazo";
String METHOD_NAME = "CalcPrazo";
String NAMESPACE = "//tempuri.org/";
String URL = "//ws.correios.com.br/calculador/CalcPrecoPrazo.asmx";
```

Nessas variáveis, foram definidas quadro importantes informações para acesso ao Web Service, que são:

- `SOAP_ACTION`: Possui o caminho completo do método do Web Service que será invocado
- `METHOD_NAME`: Possui o nome do método que será invocado pelo Web Service
- `NAMESPACE`: Possui o caminho onde o Web Service está hospedado
- `URL`: Possui o caminho do arquivo com o descritivo de todas as funções do Web Service.

Precisamos criar agora uma requisição SOAP, através de uma instância de um objeto do tipo `SoapObject`, para consumir os dados do Web Service.

O objeto `SoapObject` representará o caminho onde está hospedado nosso Web Service (`namespace`) e o nome do método a ser chamado (`method_name`).

Isso é feito através da linha:

```
SoapObject Request = new  
SoapObject(NAMESPACE, METHOD_NAME);
```

Esse objeto possui informações que serão passadas para o método do Web Service, como o código do serviço, CEP de origem e o CEP de destino.

Isso pode ser verificado nas linhas abaixo:

```
Request.addProperty("nCdServico", getServiço);  
Request.addProperty("sCepOrigem", getOrigem);  
Request.addProperty("sCepDestino", getDestino);
```

As próximas linhas relevantes de nosso programa são:

```
SoapSerializationEnvelope soapEnvelope = new SoapSerializationEnvelope(SoapEnvelope.VER11);  
soapEnvelope.dotNet = true;  
soapEnvelope.setOutputSoapObject(Request);
```

Vamos entender com mais detalhes:

1. A classe `SoapSerializationEnvelope` é responsável por empacotar nossa requisição SOAP. Para tanto, precisamos criar um objeto desse tipo. Isso é feito através da linha `SoapSerializationEnvelope soapEnvelope = new SoapSerializationEnvelope(SoapEnvelope.VER11)`, onde `SoapEnvelope.VER11` corresponde à versão 1.1 do SOAP.

2. O método `.dotNet = true` define a compatibilidade com o padrão de codificação `.net`. Foi necessário definir esse padrão pois o Web Service foi desenvolvido na tecnologia `.net`.

3. Outro método importante é o `.setOutputSoapObject(Request)`. Ele define o objeto de solicitação "SoapObject" para o envelope como a mensagem de saída referente à chamada de método de SOAP, ou seja, insere a requisição montada no envelope.

OBJETO RESPONSÁVEL PELA COMUNICAÇÃO

Precisamos agora criar o objeto responsável pela comunicação. Isso é feito através da linha `HttpTransportSE transport = new HttpTransportSE(URL)`.

A classe `HttpTransportSE` é responsável pela chamada ao servidor de aplicação, passando-se a URL como argumento.

Seu método `.call()`, quando executado, invocará o Web Service. Isso pode ser observado na linha `transport.call(SOAP_ACTION, soapEnvelope)`, que recebe como parâmetro a localização de nosso Web Service (SOAP_ACTION) e nossa requisição SOAP "envelopada" pelo objeto do tipo `SoapSerializationEnvelope(soapEnvelope)`.

Para finalizar, obteremos a resposta através do método `.getResponse()` de nosso objeto envelope.

Isso pode ser observado nas linhas:

```
SoapObject response = (SoapObject) soapEnvelope.getResponse();  
response = (SoapObject) response.getProperty("Servicos");  
response = (SoapObject) response.getProperty("cServiço");
```

A partir daí nosso programa é bastante simples, pois apenas criamos um objeto do tipo `Encomenda` e populamos seus atributos.

Rodando nosso programa, você observará a tela:





Para teste, selecionamos o código do serviço 40010.

Existem vários outros, que podem ser verificados na tabela abaixo:

Código	Serviço
40010	SEDEX sem contrato
40045	SEDEX a Cobrar, sem contrato
40126	SEDEX a Cobrar, com contrato
40215	SEDEX 10, sem contrato
40290	SEDEX Hoje, sem contrato
40096	SEDEX com contrato
40436	SEDEX com contrato
40444	SEDEX com contrato
40568	SEDEX com contrato
40606	SEDEX com contrato
41106	PAC sem contrato
41211 / 41068	PAC com contrato
81019	e-SEDEX, com contrato
81027	e-SEDEX Prioritário, com contrato
81035	e-SEDEX Express, com contrato
81868	(Grupo 1) e-SEDEX, com contrato
81833	(Grupo 2) e-SEDEX, com contrato
81850	(Grupo 3) e-SEDEX, com contrato

Tabela – código do serviço

//docs.sisecommerce.com.br/perguntas-item/envios-os-codigos-da-tabela-de-consulta-Web Service-dos-correios/108

Além disso, selecionamos CEPs aleatórios, sendo o primeiro de origem do envio da encomenda e o segundo de destino.

Após selecionar o botão “enviar”, será exibida a tela:



EXEMPLO - JSON (JavaScript Object Notation)

Nosso exemplo exibirá uma lista de flores. [Clique \(glossário\)](#) e veja.

CLASSE FLOWERJSONPARSER

Estamos chegando quase ao final de nosso código. Chegou a vez da classe FlowerJSONParser.

Seu código é ilustrado na imagem abaixo:

```
package com.profoswaldo.aulawebsevice_2.parser;

import java.util.ArrayList;
import java.util.List;
import org.json.JSONArray;
import org.json.JSONObject;

import com.profoswaldo.aulawebsevice_2.modelo.Flor;

public class FlowerJSONParser {
    public static List<Flor> parsefeed(String content) {
        try {
            JSONArray ar = new JSONArray(content);
            List<Flor> lista = new ArrayList();
            for(int i=0; i<ar.length(); i++){
                JSONObject obj = ar.getJSONObject(i);
                Flor flower = new Flor();
                flower.setNome(obj.getString("Nome"));
                flower.setCor(obj.getString("Cor"));
                flower.setPreco(obj.getDouble("Preco"));
                lista.add(flower);
            }
            return lista;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

```

    flower.setFlorId(obj.getInt("productId"));
    flower.setName(obj.getString("name"));
    flower.setCategoria(obj.getString("category"));
    flower.setInstrucoes(obj.getString("instructions"));
    flower.setFotoString(obj.getString("photo"));
    flower.setPreco(obj.getDouble("price"));
    flowerList.add(flower);
}

return flowerList;
} catch (Exception e) {
    e.printStackTrace();
    return null;
}
}

```

JSON nada mais é que um modelo para o armazenamento e troca de informações no formato texto.

Possui um conjunto classes e interfaces Java que podem ser estudadas, adaptadas e utilizadas livremente por qualquer desenvolvedor.

Sua principal característica é que ela trabalha lendo todo o documento JSON para a memória.

Em nosso exemplo, implementamos *JSONArray* e *JSONObject*.

Na linha *JSONArray ar = new JSONArray(content)*, criamos um objeto do tipo *JSONArray*, passando a string recebida.

Podemos comparar esse objeto a um array comum, onde cada parte do JSON se transforma em uma posição.

Logo após, este array é percorrido e, para cada interação, é criado um objeto do tipo *JSONObject*, através do qual obteremos os dados de flor, conforme demonstrado no código abaixo:

```

for(int i=0; i< ar.length(); i++) {
    JSONObject obj = ar.getJSONObject(i);
    Flor flower = new Flor();

    flower.setFlorId(obj.getInt("productId"));
    flower.setName(obj.getString("name"));
    flower.setCategoria(obj.getString("category"));
    flower.setInstrucoes(obj.getString("instructions"));
    flower.setFotoString(obj.getString("photo"));
    flower.setPreco(obj.getDouble("price"));
    flowerList.add(flower);
}

```

CLASSE PRINCIPAL (MAINACTIVITY)

Para finalizar, chegou a vez de nossa classe principal (MainActivity).

Para demonstrar o código completo dessa classe, foi necessário, devido ao seu tamanho, demonstrar o código em três tela:

```

    package com.profosaldo.aulawebsevice_2;
    import ...
    public class MainActivity extends ListActivity {
        private static final String PHOTO_BASE_URL = "http://services.hanselandpetal.com/photos/";
        private ProgressDialog progressbar;
        private List<Flor> tarefas;
        private List<Flor> listaFlores;
        ...
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main);
            progressbar = (ProgressDialog) findViewById(R.id.progressbar);
            progressbar.setProgressStyle(ProgressDialog.STYLE_INDETERMINATE);
            tarefas = new ArrayList<Flor>();
        }
        ...
        @Override
        public boolean onCreateOptionsMenu(Menu menu) {
            getMenuInflater().inflate(R.menu.main, menu);
            return true;
        }
        ...
        @Override
        public boolean onOptionsItemSelected(MenuItem item) {
            if (isOnline()) {
                requestData("http://services.hanselandpetal.com/secure/flowers.json");
            } else {
                Toast.makeText(this, "Rede indisponivel", Toast.LENGTH_LONG).show();
            }
            return false;
        }
    }

```

```

    private void requestData(String url) {
        MyTask task = new MyTask();
        task.execute(url);
    }

    protected void updateDisplay() {
        FlowerAdapter adapter = new FlowerAdapter(this, R.layout.item_flower, listaFlores);
        setListAdapter(adapter);
    }

    protected boolean isOnline() {
        ConnectivityManager cm = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
        if (netInfo == null && netInfo.isConnectedOrConnecting()) {
            return true;
        } else {
            return false;
        }
    }

    private class MyTask extends AsyncTask<String, String, List<Flor>> {
        ...
        @Override
        protected void onPreExecute() {
            if (progressbar != null) {
                progressbar.show();
            }
            progressbar.setIndeterminate(true);
            tarefas.add(this);
        }
        ...
        @Override
        protected List<Flor> doInBackground(String... params) {
            String content = HTTPManager.getDados(params[0], "feeduser", "feedpassword");
            listaFlores = FlowerJSONParser.parseFeedContent();
            for (Flor flower : listaFlores) {
                String imageurl = PHOTO_BASE_URL + flower.getFotoString();
            }
        }
        ...
    }

```

```

public class MainActivity extends ListActivity {
    ...
    @Override
    protected void doInBackground(String... params) {
        String url = params[0];
        String password = params[1];
        URL urlFlowers = new URL(url);
        listaFlares = FlowerJSONParser.parseJsonContent(urlFlowers);
        for (Flower flower : listaFlares) {
            try {
                String imgurl = PHOTO_PAGE_URL + flower.getFotoString();
                InputStream in = (InputStream) new URL(imgurl).getContent();
                Bitmap bitmap = BitmapFactory.decodeStream(in);
                flower.setFoto(bitmap);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        return listaFlares;
    }
    ...
}

```

Grande parte do código não apresenta nenhuma inovação.

Visto que já discutimos também a classe `AsyncTask` em nosso exemplo anterior, apenas é necessário destacar o trecho de código abaixo:

```

protected boolean isOnline() {
    ConnectivityManager cm = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo netInfo = cm.getActiveNetworkInfo();
    if (netInfo != null && netInfo.isConnectedOrConnecting()) {
        return true;
    }
    else {
        return false;
    }
}

```

Através da classe `ConnectivityManager`, podemos consultar o estado de conectividade da rede, notificando ao nosso aplicativo qualquer mudança ocorrida.

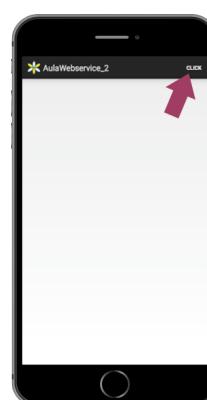
Para obter uma instância dessa classe, fazemos uso do método `getSystemService()`, que informa qual serviço do sistema será utilizado.

Em nosso exemplo, `Context.CONNECTIVITY_SERVICE`.

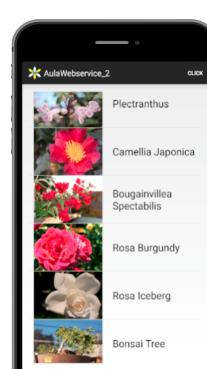
A partir daí, obtemos por retorno um objeto do tipo `NetworkInfo`, que possui detalhes sobre a rede de dados padrão ativa no momento.

Logo após, verificamos se está conectada ou conectando.

Execute nosso programa para verificar a tela abaixo:



Selecione a opção click, conforme apontado pela seta vermelha na imagem acima, para verificar a tela:



ATIVIDADE

(FVG/2009) Um Web Service é definido pela W3C como um sistema de software projetado para fornecer interoperabilidade entre máquinas em uma determinada rede. Dentro do contexto dos Web Services, assinale a alternativa CORRETA:

- a) A UDDI (Universal Description, Discovery, and Integration) é uma linguagem baseada em XML que descreve o que um Web Service pode fazer, onde ele reside e como chamá-lo.
- b) SOAP (Simple Object Access Protocol) é um protocolo, baseado em XML, para troca de informação estruturada com Web Services em redes de computadores.
- c) A interoperabilidade entre os Web Services e aplicações é garantida devido ao uso obrigatório da linguagem Java na implementação das aplicações.
- d) SOA (Simple Object Access) é uma plataforma de arquitetura orientada a serviços, utilizada como base para suportar os Web Services.
- e) A WSDL (Web Services Description Language) é uma especificação para publicar e localizar informações sobre Web Services.

Justificativa

Glossário