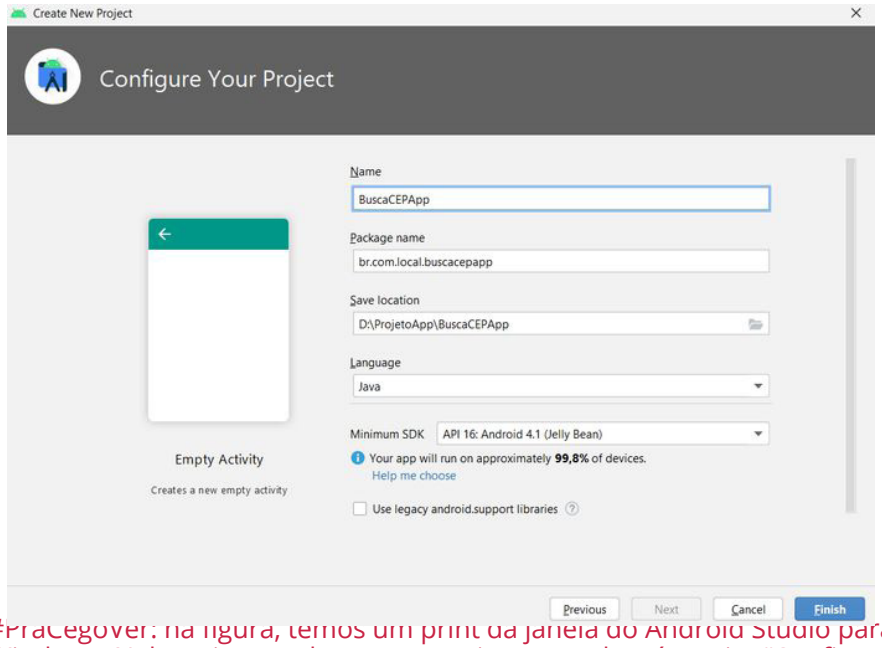
Vídeo sobre API

Uma API pode ser definida como um conjunto de regras que fornecem comunicação por meio de protocolos para diferentes finalidades. Como o acesso dos arquivos externos da aplicação dependem do uso de protocolos de comunicação para sua integridade, o uso de

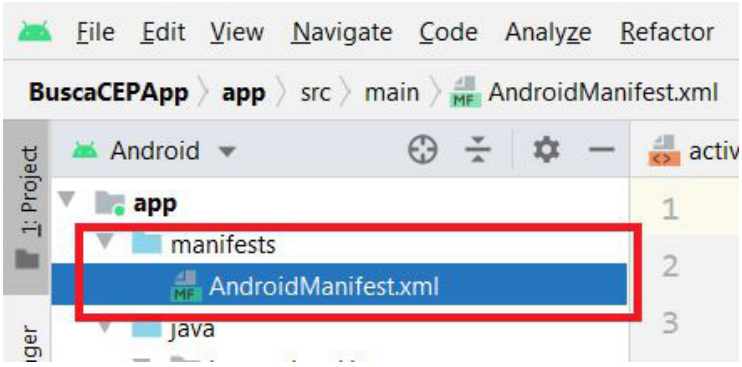
Confira um exemplo de uso da API.

Inicialmente, crie um aplicativo chamado “BuscaCEPApp”. Para tanto, elabore um projeto no Android Studio e adicione as seguintes informações.



#PraCegoVer: na figura, temos um print da janela do Android Studio para Windows. Nela, existe um banner superior no qual está escrito “Configure Your Project”. Abaixo dele, existe uma tela contendo itens de configuração do projeto

Após completado o carregamento do projeto, para acessar os arquivos externos, configure o acesso do aplicativo a internet, abrindo a pasta “*manifests*”. Agora clique em “AndroidManifest.xml”.



#PraCegoVer: na figura, temos um print da janela do Android Studio para Windows. Nela, existe a barra de ferramentas da plataforma. Abaixo dele, existe um dashboard com um trecho destacado em vermelho contendo dois ícones, um de uma pasta com o nome “manifests” e o segundo com um ícone de um programa “xml” cujo nome é “AndroidManifest.xml”.

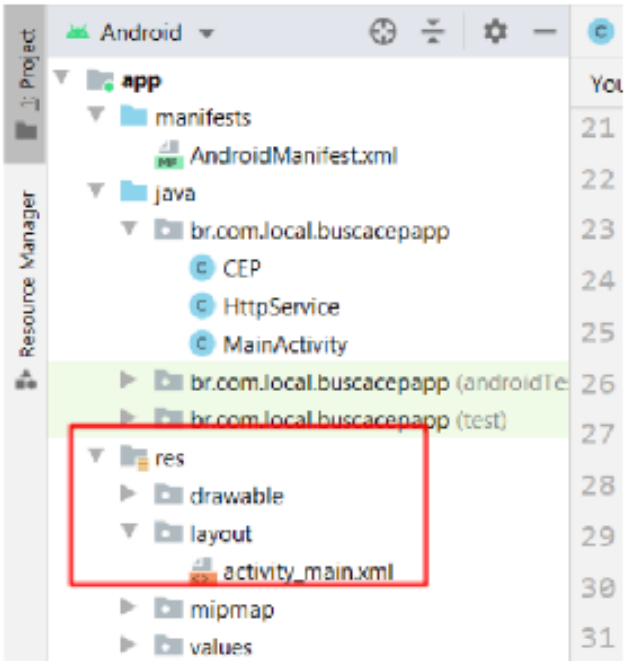
Na sequência, faça a inserção do código destacado.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="br.com.local.buscacepapp">

    <uses-permission android:name="android.permission.INTERNET"/>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="BuscaCEPApp"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.BuscaCEPApp">
        <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>
```

Com as configurações realizadas, monte o aplicativo para acessar a API de busca por CEP.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="50dp"
    tools:context=".MainActivity">

    <EditText
        android:id="@+id/txtCep"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:digits="0123456789"
        android:hint="Insira seu CEP"
        android:inputType="number"
        android:textColor="#595959"
        android:textSize="25sp" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:orientation="horizontal">

        <Button
            android:id="@+id/btnBuscaCep"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="20dp"
            android:layout_marginBottom="10dp"
            android:background="@color/design_default_color_on_primary"
            android:padding="10dp"
            android:text="Buscar CEP"
            android:textSize="22sp" />

    </LinearLayout>

    <TextView
        android:id="@+id/lblResposta"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:padding="5dp"
        android:textColor="#000000"
        android:textSize="20sp" />

</LinearLayout>
```

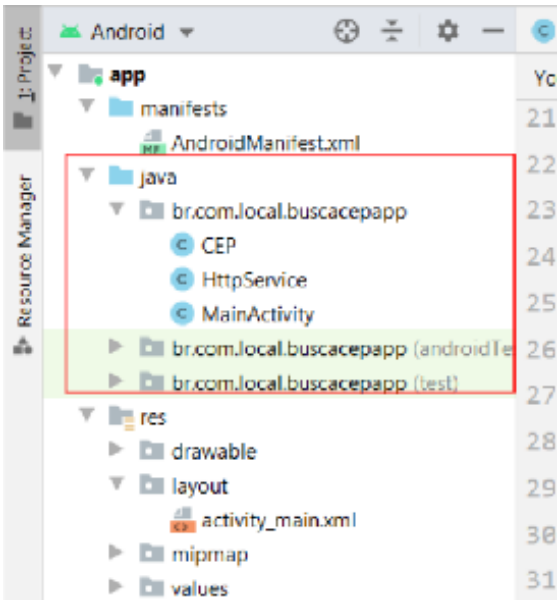
Lembre-se de que é possível acessar o arquivo de layout seguindo pelo caminho "res-""*layout-*" e "*activity_main.xml*".



#PraCegoVer: na figura, temos um print da janela do Android Studio para Windows. Nela, existe a barra de ferramentas da plataforma.

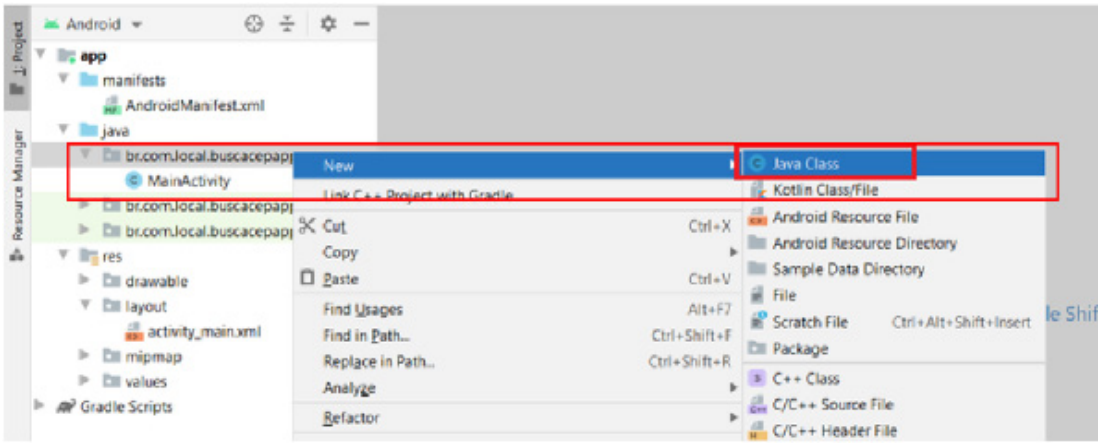
Finalizada a codificação do layout, avance para o código da parte que acessa os arquivos e retorna o resultado para o aplicativo. Faça isso em “java” e “br.com.local.buscacepapp”.

Nesse caso, será apresentado três arquivos para o desenvolvimento do trabalho.



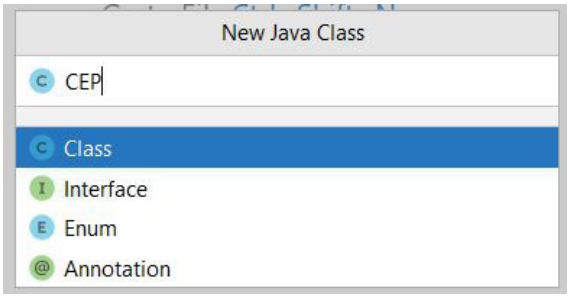
#PraCegoVer: na figura, temos um print da janela do Android Studio para Windows. Nela, existe a barra de ferramentas da plataforma.

Para criar os arquivos, clique com o botão direito do mouse.



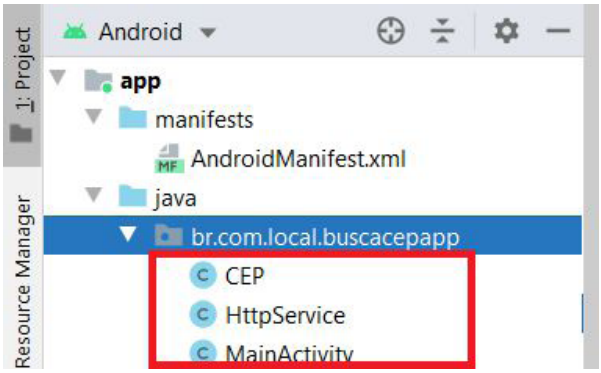
#PraCegoVer: na figura, temos um print da tela do Android Studio. Nela, há um dashboard à esquerda e, do meio até à direita, há duas janelas com opções de configuração.

Após clicar Java Class abrirá a aba Class e você poderá criar o “CEP.java”.



#PraCegoVer: na figura, temos um print da tela do Android Studio. Nela, há uma janela com o título “New Java Class” no topo e opções de configuração abaixo.

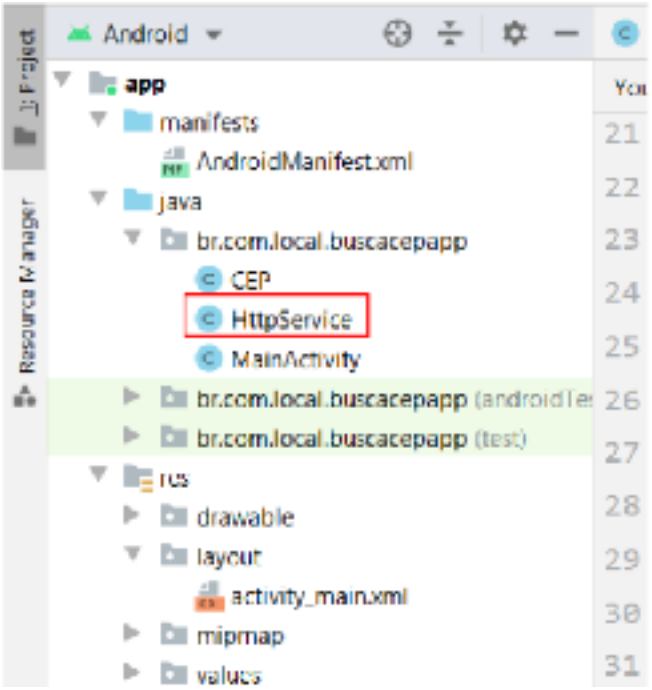
Observe com atenção o resultado. É importante destacar que o “**CEP.java**” é o objeto para troca de informações.



#PraCegoVer: na figura, temos um print da janela do Android Studio para Windows. Nela, existe a barra de ferramentas da plataforma. Abaixo dele, existe um dashboard com um trecho destacado em vermelho contendo três ícones, cada um formado por um círculo com a letra "c" dentro, com os seguintes trechos escritos: "CEP". "HttpService" e "MainActivity".

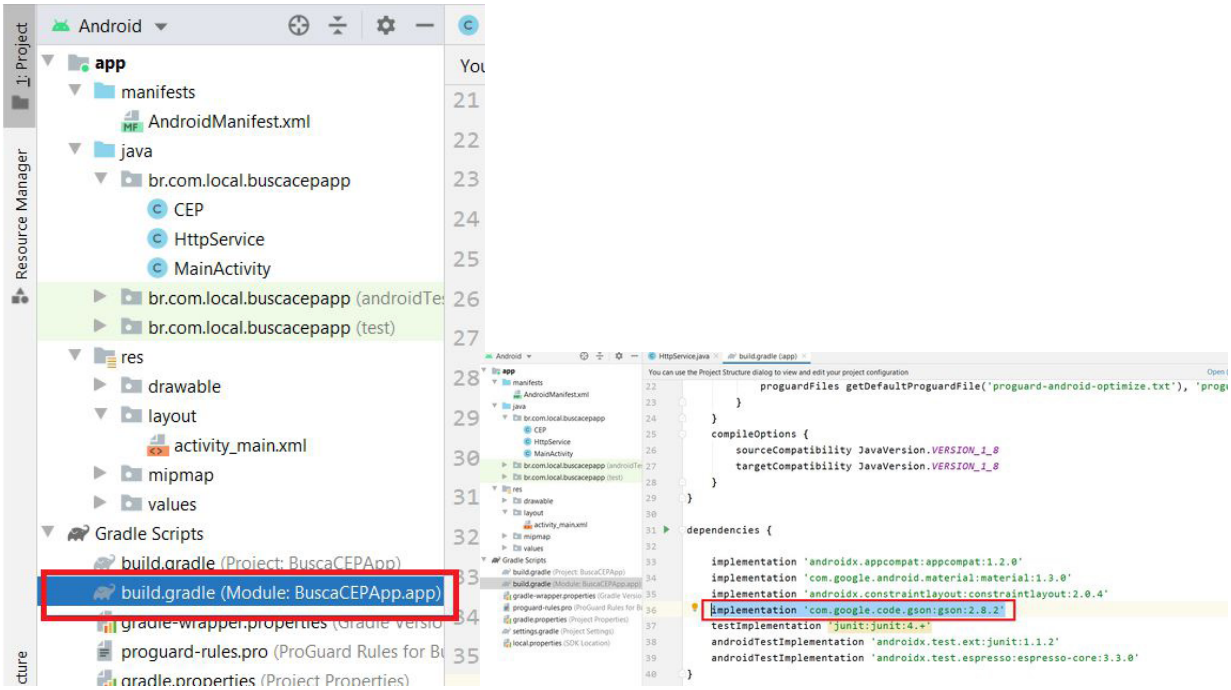
```
package br.com.local.buscacepapp;

public class CEP {
        private String cep;
        private String logradouro;
        private String complemento;
        private String bairro;
        private String localidade;
        private String uf;

    public CEP() {
    }

    public String getCep() { return cep; }

    public void setCep(String cep) { this.cep = cep; }

    public String getLogradouro() { return logradouro; }

    public void setLogradouro(String logradouro) { this.logradouro = logradouro; }

    public String getComplemento() { return complemento; }

    public void setComplemento(String complemento) { this.complemento = complemento; }

    public String getBairro() { return bairro; }

    public void setBairro(String bairro) { this.bairro = bairro; }

    public String getLocalidade() { return localidade; }

    public void setLocalidade(String localidade) { this.localidade = localidade; }

    public String getUf() { return uf; }
        public void setUf(String uf) { this.uf = uf; }

        @Override
        public String toString() {
            return "CEP: " + getCep()
                    + "\nLogradouro: " + getLogradouro()
                    + "\nComplemento: " + getComplemento()
                    + "\nBairro: " + getBairro()
                    + "\nCidade:" + getLocalidade()
                    + "\nEstado: " + getUf();
        }
}
```

Já o ***HTTPService*** traz as configurações necessárias para o acesso à API CEP.



#PraCegoVer: na figura, temos um print da janela do Android Studio para Windows. Nela, existe a barra de ferramentas da plataforma.

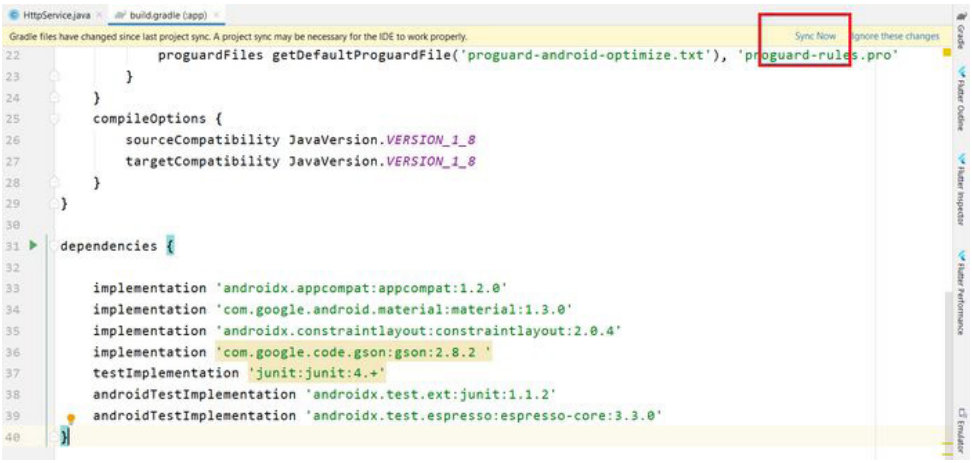
A fim de utilizar essa classe, deve-se implementar uma API externa, onde, no projeto do Google, é a GSON, o que requer a configuração do arquivo.



#PraCegoVer: na figura, temos um print da janela do Android Studio para Windows. Nela, existe a barra de ferramentas da plataforma. Ao

lado, temos um print da janela do Android Studio para Windows. Nela, existe o dashboard à esquerda. Do centro para a esquerda, há o painel de modelagem com várias linhas de código escrita.

Após inserir essa linha, conforme indicado aqui, sincronize o arquivo com o projeto, clicando em "*Sync Now*", localizado na parte superior à direita da mesma janela.



#PraCegoVer: temos um print da janela do Android Studio para Windows. Nela, existe o painel de modelagem com várias linhas de código escrita. No canto superior direito está um destaque no botão de "Sync Now".

Em seguida, configure a Classe "*HTTPService.class*".

```
package br.com.local.buscacepapp;

import ...

public class HttpService extends AsyncTask<Void, Void, CEP> {

    private final String cepInserido;

    public HttpService(String cep) { this.cepInserido = cep; }

@Override
protected CEP doInBackground(Void... voids) {
    StringBuilder resposta = new StringBuilder();

    if (this.cepInserido != null && this.cepInserido.length() == 8) {

        try {

            URL url = new URL( spec: "https://viacep.com.br/ws/" + this.cepInserido + "/json/");

            HttpURLConnection connection = (HttpURLConnection) url.openConnection();
            connection.setRequestMethod("GET");
            connection.setRequestProperty("Content-type", "application/json");
            connection.setRequestProperty("Accept", "application/json");
            connection.setDoOutput(true);
            connection.setConnectTimeout(5000);
            connection.connect();

            Scanner scanner = new Scanner(url.openStream());
```

```
            while (scanner.hasNext()) {
                resposta.append(scanner.next());
            }

        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    return new Gson().fromJson(resposta.toString(), CEP.class);
}

}
```

Note que o ***MainActivity*** é o objeto responsável pelos eventos e pela montagem das informações.

```
package br.com.local.buscacepapp;

import ...

public class MainActivity extends AppCompatActivity {

    Button btnBuscarCep;
    EditText txtCep;
    TextView lblResposta;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    txtCep = findViewById(R.id.txtCep);
    lblResposta = findViewById(R.id.lblResposta);
    btnBuscarCep = findViewById(R.id.btnBuscaCep);
}
```

```
btnBuscarCep.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        try {
            CEP retorno = new HttpService(txtCep.getText().toString().trim()).execute().get();
            lblResposta.setText(retorno.toString());
        } catch (ExecutionException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

    }
});
```

Agora, é só executar o emulador e realizar os testes.

#PraCegoVer: na figura, temos a tela de um celular com design do app "BuscaCEPApp". Aparecem seus botões e outras particularidades, como horário e nível de bateria. No centro, encontramos uma caixa de texto "insira seu CEP" e, abaixo dela, um botão com o trecho escrito "buscar CEP".

#PraCegoVer: na figura, temos a tela de um celular com design do app "BuscaCEPApp". Aparecem seus botões e outras particularidades, como horário e nível de bateria. No centro, encontramos uma caixa de texto

"insira seu CEP" e, abaixo dela, um botão com o trecho escrito "buscar CEP". Abaixo disso, há um teclado numérico.

#PraCegoVer: na figura, temos a tela de um celular com design do app "BuscaCEPApp". Aparecem seus botões e outras particularidades, como horário e nível de bateria. No centro, encontramos uma caixa de texto "04029000" e, abaixo dela, um botão com o trecho escrito "buscar CEP". Abaixo disso, há dados de CEP, Logradouro, Complemento, Bairro, Cidade e Estado.