

# Projeto de testes automatizados

## Tecnologias Utilizadas:

IDE - VS CODE

Linguagem – Java

Frameworks – Junit e Selenium

## Instalando e configurando o ambiente Java e Maven

Para que o nosso projeto seja criado e funcione sem problemas, precisamos utilizar uma versão do JDK 11 ou superior. Abaixo seguiremos com a instalação e configuração do ambiente utilizando a versão 11.

### JAVA JDK 11

Para fazer o download diretamente no site da Oracle, é necessário estar logado com uma conta cadastrada. Caso ainda não possua cadastro, acesse o link abaixo e crie sua conta, o processo de cadastro é simples e gratuito.

**Link de cadastro Oracle:** <https://profile.oracle.com/myprofile/account/create-account.jspx>

Após ter realizado o cadastro e efetuado login, acesse o link abaixo para realizar o download.

**Link download JDK 11:** <https://www.oracle.com/br/java/technologies/javase-jdk11-downloads.html>

Ao abrir o link indicado acima, você será direcionado para página de download, selecione o seu sistema operacional conforme mostra o marcador azul, e em seguida faça o download da versão executável conforme indicado pelo marcador vermelho na imagem abaixo:

**Java SE Development Kit 11.0.12**

Java SE subscribers will receive JDK 11 updates until at least **September of 2026**.

These downloads can be used for development, personal use, or to run Oracle licensed products. Use for other purposes, including production or commercial use, requires a Java SE subscription or another Oracle license.

JDK 11 software is licensed under the [Oracle Technology Network License Agreement for Oracle Java SE](#).

[JDK 11.0.12 checksum](#)

[Documentation Download](#)

**Linux** macOS Solaris Windows

Product/file description	File size	Download
x64 Installer	151.83 MB	<a href="#">jdk-11.0.12_windows-x64_bin.exe</a>
x64 Compressed Archive	171.27 MB	<a href="#">jdk-11.0.12_windows-x64_bin.zip</a>

A Instalação do JDK é simples, apenas siga os passos padrão de instalação.

## Maven 3.8.2

O processo de configuração do Maven é similar ao do JAVA, porém o Maven não será instalado, nós apenas iremos baixar e descompactar o seu diretório para aponta-lo no sistema. Siga os passos abaixo para realizar o download e configurar o Maven em sua máquina.

Para realizar o download do maven, acesse o link abaixo:

**Link de download Maven:** <https://maven.apache.org/download.cgi>

Ao acessar o link acima você será direcionado para o site de download do maven, faça o download do arquivo bin.zip conforme indicado em vermelho na imagem abaixo

### Files

Maven is distributed in several formats for your convenience. Simply pick a ready-made binary distribution archive and follow the [installation instructions](#). Use a source archive if you intend to build Maven yourself.

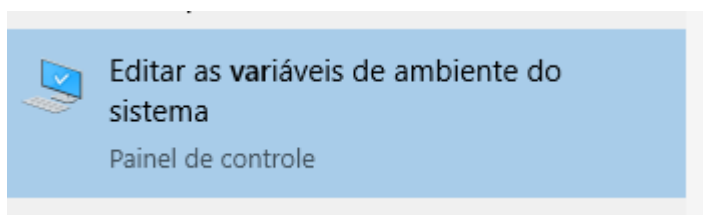
In order to guard against corrupted downloads/installations, it is highly recommended to [verify the signature](#) of the release bundles against the public [KEYS](#) used by the Apache Maven developers.

	Link	Checksums	Signature
Binary tar.gz archive	<a href="#">apache-maven-3.8.2-bin.tar.gz</a>	<a href="#">apache-maven-3.8.2-bin.tar.gz.sha512</a>	<a href="#">apache-maven-3.8.2-bin.tar.gz.asc</a>
Binary zip archive	<a href="#">apache-maven-3.8.2-bin.zip</a>	<a href="#">apache-maven-3.8.2-bin.zip.sha512</a>	<a href="#">apache-maven-3.8.2-bin.zip.asc</a>
Source tar.gz archive	<a href="#">apache-maven-3.8.2-src.tar.gz</a>	<a href="#">apache-maven-3.8.2-src.tar.gz.sha512</a>	<a href="#">apache-maven-3.8.2-src.tar.gz.asc</a>
Source zip archive	<a href="#">apache-maven-3.8.2-src.zip</a>	<a href="#">apache-maven-3.8.2-src.zip.sha512</a>	<a href="#">apache-maven-3.8.2-src.zip.asc</a>

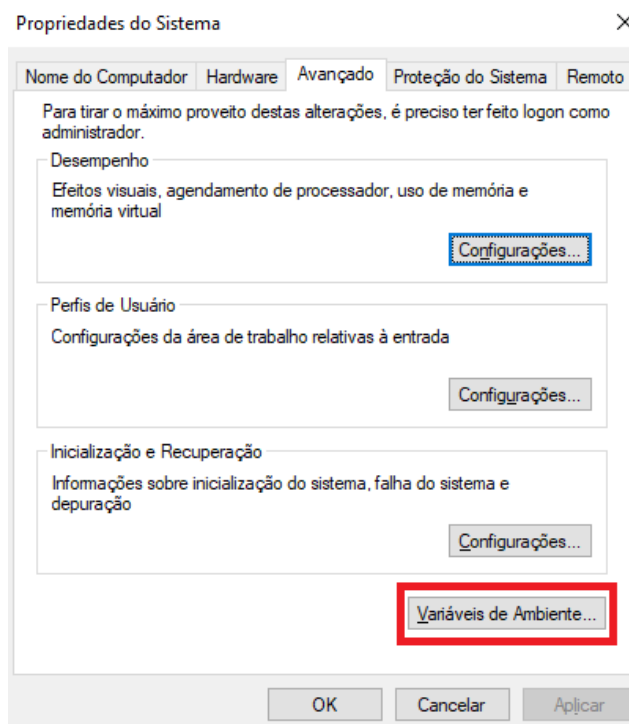
Após realizar o download extraia o arquivo em um local de sua preferência, porém de fácil acesso pois precisaremos utilizar o caminho onde o arquivo foi extraído para realizar as configurações das variáveis de ambiente Maven.

## Configurando as Variáveis de ambiente JAVA

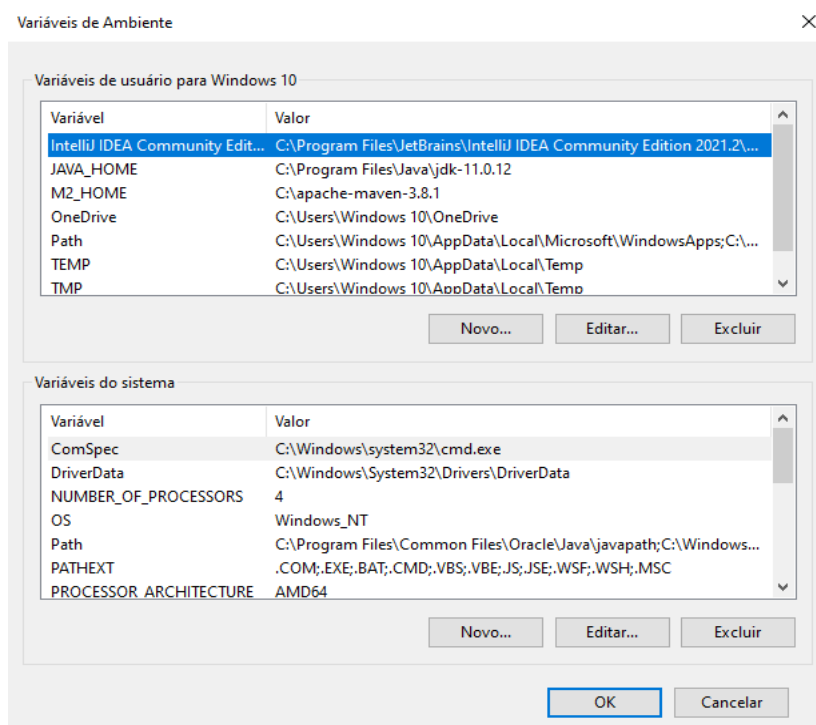
No menu iniciar do Windows pesquise por “**variáveis**” , clique na opção “**Editar as variáveis de Ambiente**” como no exemplo abaixo:



Em seguida clique em “**Variáveis de Ambiente**” como no exemplo abaixo:



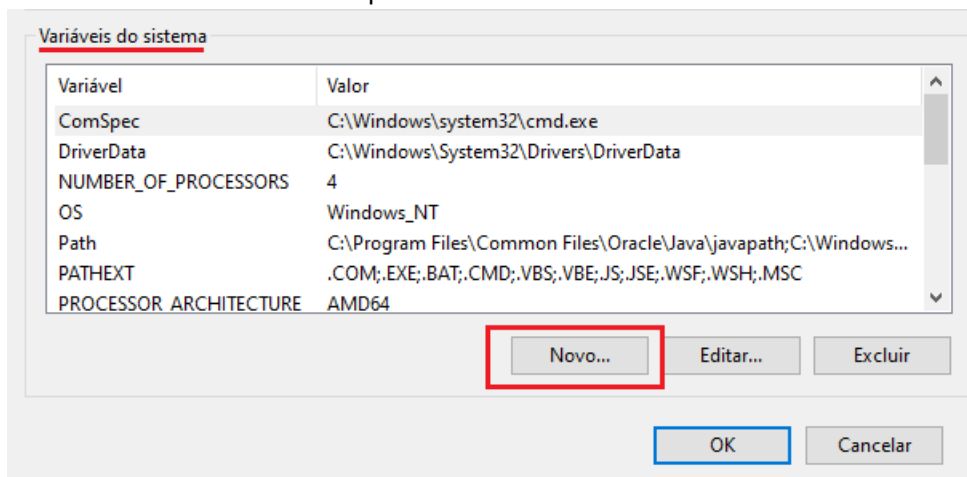
A seguinte tela irá abrir:



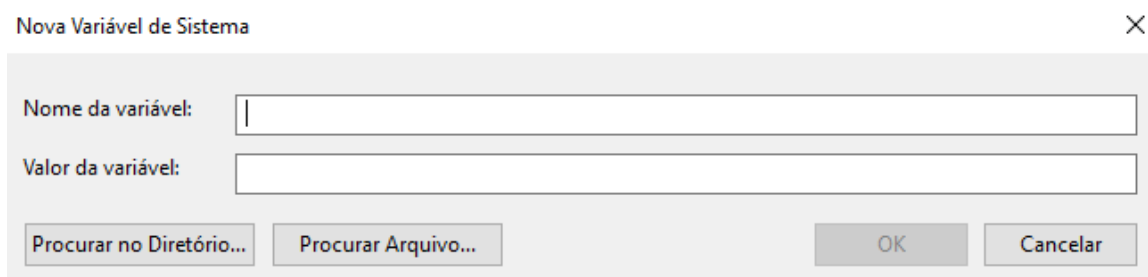
Nessa tela temos 2 opções para configurar nossas variáveis. Configurando as variáveis, em **“Variáveis de Usuário”** na parte superior da tela, nossas configurações serão válidas apenas para o usuário logado, ou seja, caso você possua mais usuários configurados em sua máquina, as variáveis não serão visíveis para os demais. Se houver necessidade de configurar as variáveis para todos os usuários existentes na máquina, então faça a configuração em **“Variáveis do sistema”** na parte inferior da tela.

As configurações e passos realizados serão os mesmos, independentemente de serem configurados apenas para um usuário ou para todo o sistema. Em nosso exemplo seguiremos com a configuração das variáveis para todo o sistema.

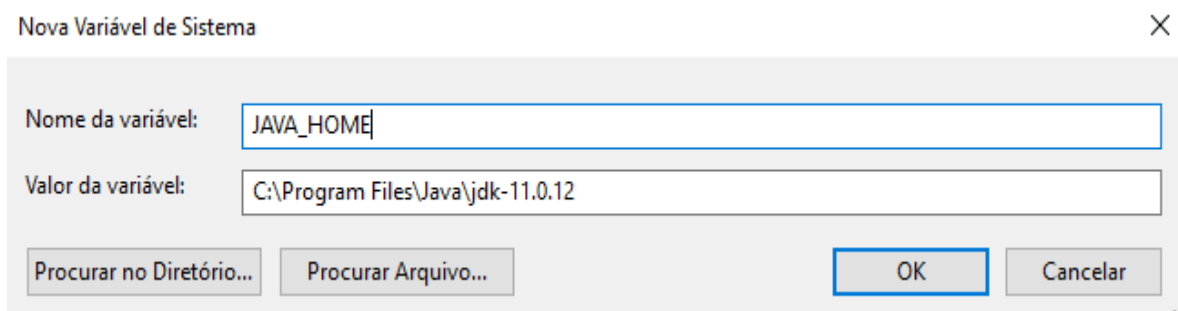
Em **“Variáveis do sistema”** clique no botão **“Novo”**



A seguinte tela será apresentada:

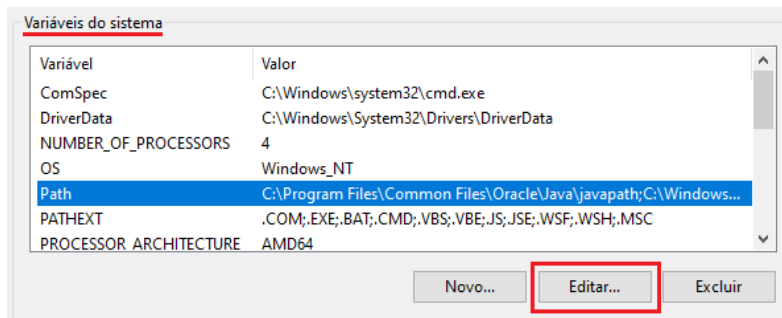


Na opção **“Nome da variável”** coloque **“JAVA\_HOME”**, na opção **“Valor da variável”** coloque o **caminho** do diretório onde o JDK foi instalado, exemplo: **“C:\Program Files\Java\jdk-11.0.12”**

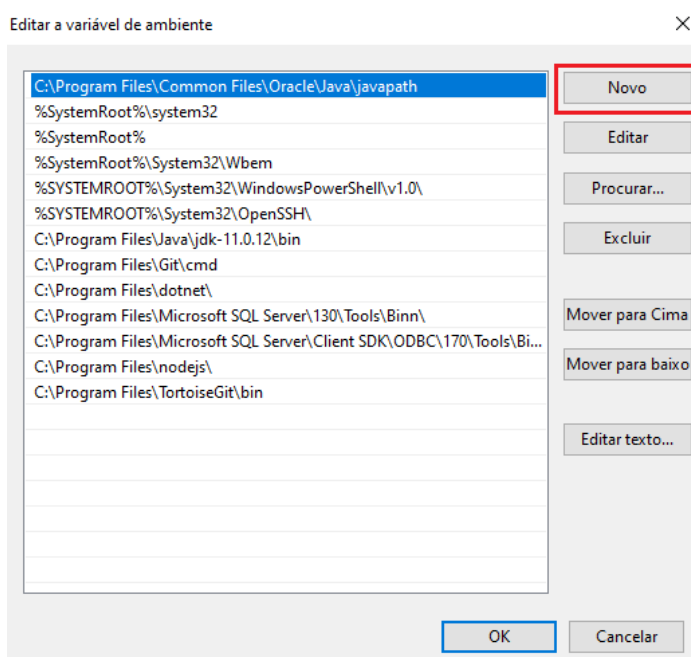


Em seguida clique em **“OK”**.

De volta a tela de variáveis do sistema, localize a variável “**Path**”, há selecione e clique no botão “**Editar**”, como mostra a imagem abaixo:

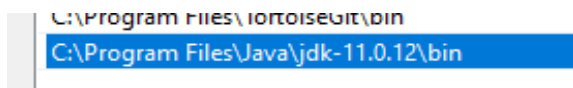


A seguinte tela será apresentada:



Clique na opção “**Novo**” e será permitido incluir um novo **Path**, inclua o caminho raiz da pasta bin onde o JDK foi instalado e clique em “**OK**”.

Exemplo:



Realizados os passos a cima, o ambiente Java deverá estar pronto. Para testar iremos abrir o **CMD** e digitar o código “ java -version “, se tudo correu bem a seguinte tela será apresentada:

```
C:\Users\Windows 10>java -version
java version "11.0.12" 2021-07-20 LTS
Java(TM) SE Runtime Environment 18.9 (build 11.0.12+8-LTS-237)
Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.12+8-LTS-237, mixed mode)
```

## Variáveis de ambiente Maven

Para adicionar as variáveis de ambiente do maven, siga as mesmas etapas de configurações das variáveis de ambiente do **JAVA**, porém na opção “**Nome da variável**” o valor informado deve ser “**M2\_HOME**” e os valores de caminho, devem ser referente ao diretório raiz onde o maven foi extraído. Também é necessário a inclusão no **Path**, adicionando um novo **Path** da mesma maneira que fizemos anteriormente com o **JAVA**, mas alterando o caminho para a raiz do diretório **bin** do **maven**.

Para verificarmos se a configuração do maven está correta, iremos abrir o **CMD** e digitar o código “**mvn -version**”, se tudo correu bem a seguinte tela será apresentada:

```
C:\Users\Windows 10>mvn -version
Apache Maven 3.8.1 (05c21c65bdfed0f71a2f2ada8b84da59348c4c5d)
Maven home: C:\apache-maven-3.8.1\bin\..
Java version: 11.0.12, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-11.0.12
Default locale: pt_BR, platform encoding: Cp1252
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
```

Obs: O CMD deve ser aberto após o término das configurações, caso você o tenha aberto antes de finalizar as configurações, feche e abra-o novamente.

## Criando o projeto maven:

Para criar um projeto de testes maven bem estruturado e padronizado, iremos seguir alguns passos que serão apresentados nesse projeto. É importante que os passos sejam seguidos corretamente e que sejam aplicados os padrões e boas práticas aqui apresentados.

*Iremos criar o nosso projeto maven por linha de comando, isso pode ser feito através do terminal do próprio **VSCode** (IDE que iremos utilizar) ou através de algum outro terminal de sua preferência como o CMD ou PowerShell.*

Com o terminal aberto iremos utilizar o seguinte código:

“**mvn archetype:generate -DarchetypeArtifactId=maven-archetype-quickstart**”

Ao Executar o código a cima, o projeto começará a ser criado mas antes de finalizar precisamos passar algumas informações para ele:

1º - **groupId**: exemplo de groupId: “*br.com.alterdata*”

2º - **artifactId**: exemplo de artifactId: “*automatizado*”

3º - **versão**: caso não tenha pré-definido, deixe a versão padrão “*1.0-SNAPSHOT*”

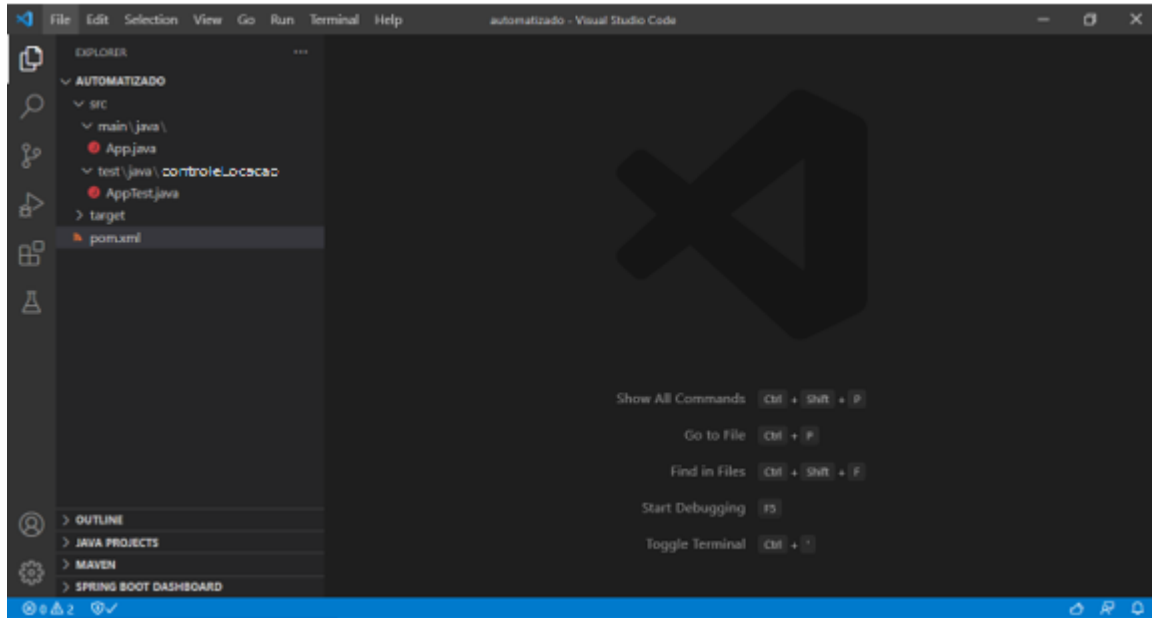
4º - **Package**: exemplo de package: “*controleLocacao*”

5º - Confirmação das informações passadas, se estiver tudo ok digite “**y**” e pressione **ENTER**

*Feito isso nosso projeto será criado com sucesso!*

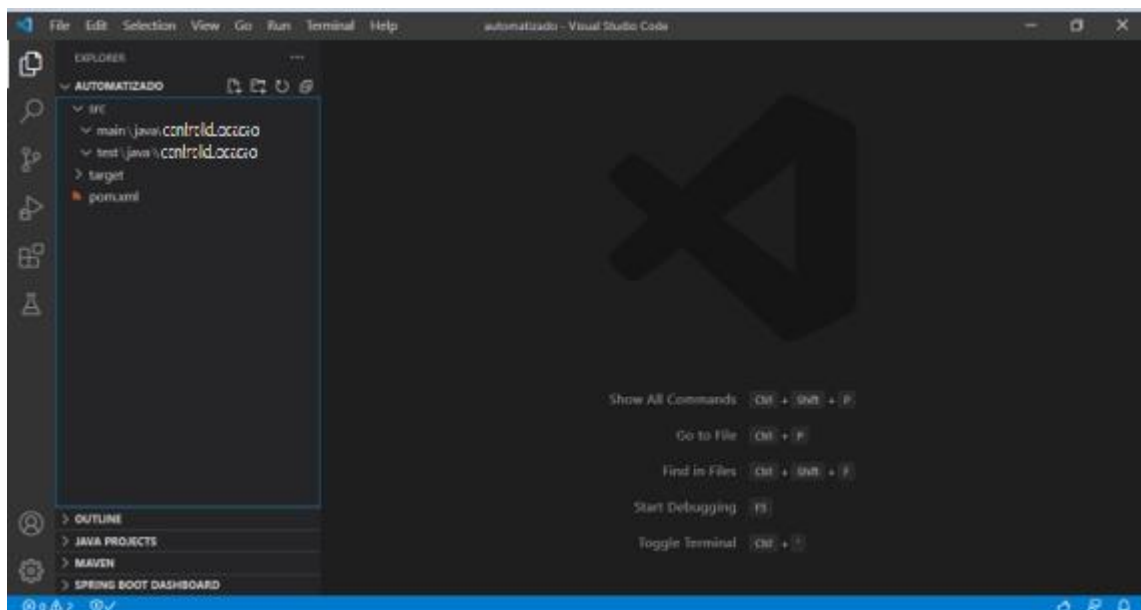
## Arrumando a casa:

Agora que já temos o nosso projeto criado, precisamos organizar a estrutura dele e definir alguns padrões que serão seguidos durante todo o período de desenvolvimento. Começaremos abrindo o projeto na IDE que iremos utilizar (**VS Code**). Na imagem abaixo vemos uma representação do nosso projeto **recém-criado**.



Note que já temos duas classes criadas, dentro da hierarquia “**main**” temos a classe “**App.java**” e dentro da hierarquia “**test**” temos a classe “**AppTest.java**”. Essas classes são criadas para que possamos testar o nosso projeto assim que o criamos, mas não as utilizaremos, por tanto delete ambas as classes.

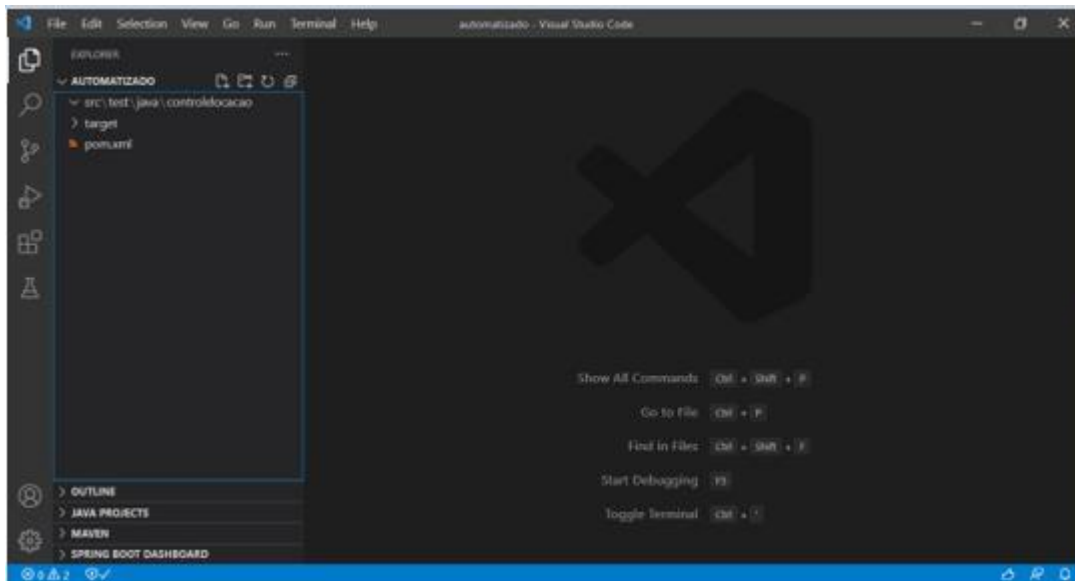
A imagem abaixo mostra como ficará nosso projeto após ambas as classes serem deletadas.



Note que ficamos agora apenas com os pacotes criados. Podemos observar que temos duas hierarquias de trabalho principais dentro de nosso projeto, a hierarquia “**main**” composta por seus pacotes filhos, netos, etc. E a hierarquia “**test**” também composta por seus pacotes filhos, netos, etc. Como o nosso projeto é voltado para realização de testes, trabalharemos dentro da hierarquia de “**test**” e não utilizaremos a hierarquia “**main**”, portanto vamos deletar também toda a hierarquia de “**main**”.

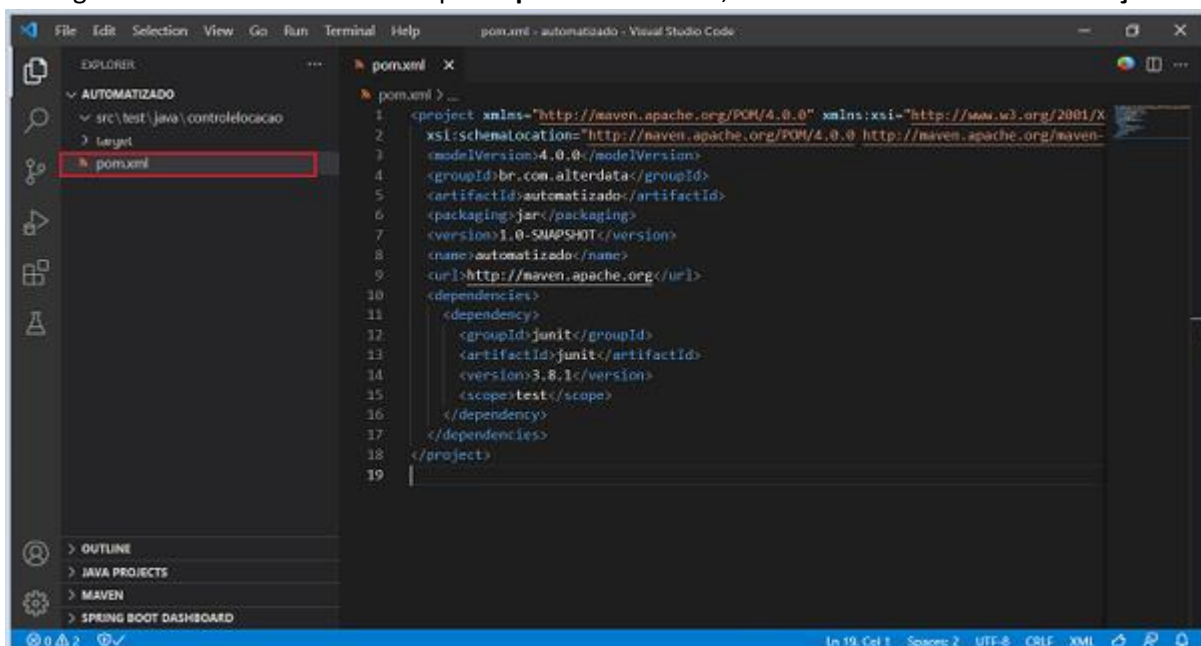
**ATENÇÃO:** devemos sempre trabalhar dentro da hierarquia “**test**”, pois algumas ferramentas que utilizaremos no projeto, não serão reconhecidas fora desta hierarquia.

A imagem abaixo mostra como ficará o nosso projeto depois de deletar a hierarquia “**main**”.



Agora estamos prontos para adicionar as ferramentas que serão utilizadas, observe que o último arquivo do nosso projeto se chama “**pom.xml**” e é nesse arquivo que iremos configurar e adicionar ferramentas ao nosso projeto.

A imagem abaixo mostra o nosso arquivo “**pom.xml**” aberto, sem ter sofrido **nenhuma alteração**.

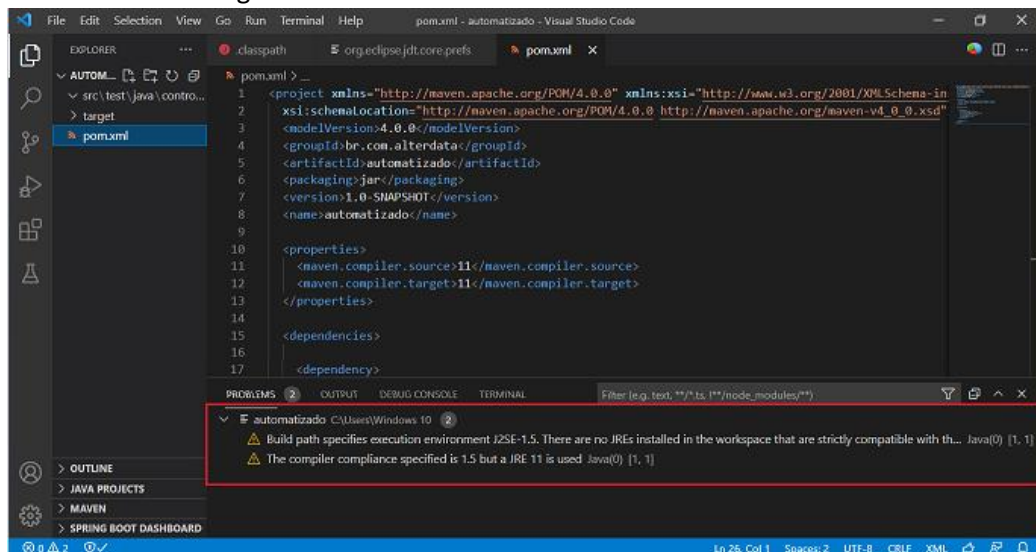




Primeiramente vamos informar o **pom.xml** a versão do JAVA que queremos utilizar em nosso compilador, para fazer isso adicionaremos código abaixo em nosso arquivo **pom.xml**.

```
<properties>
  <maven.compiler.source>11</maven.compiler.source>
  <maven.compiler.target>11</maven.compiler.target>
</properties>
```

Porém se abrirmos nosso terminal, veremos que mesmo informando no pom.xml a versão de **JAVA** que queremos utilizar, ainda teremos alertas que a versão utilizada é diferente da versão instalada como mostra a imagem abaixo:



Para resolvermos esses alertas precisamos alterar a versão do JAVA nos arquivos internos, esses arquivos estão na raiz da pasta do projeto. O primeiro arquivo a ser alterado será o **.classpath**.

Nome	Data de modificação	Tipo
.settings	24/09/2021 15:20	Pasta
src	24/09/2021 15:43	Pasta
target	24/09/2021 15:20	Pasta
.classpath	24/09/2021 16:55	Arqui
.project	24/09/2021 15:20	Arqui
pom.xml	24/09/2021 16:46	Docu

Vamos abrir esse arquivo com o próprio VS Code:

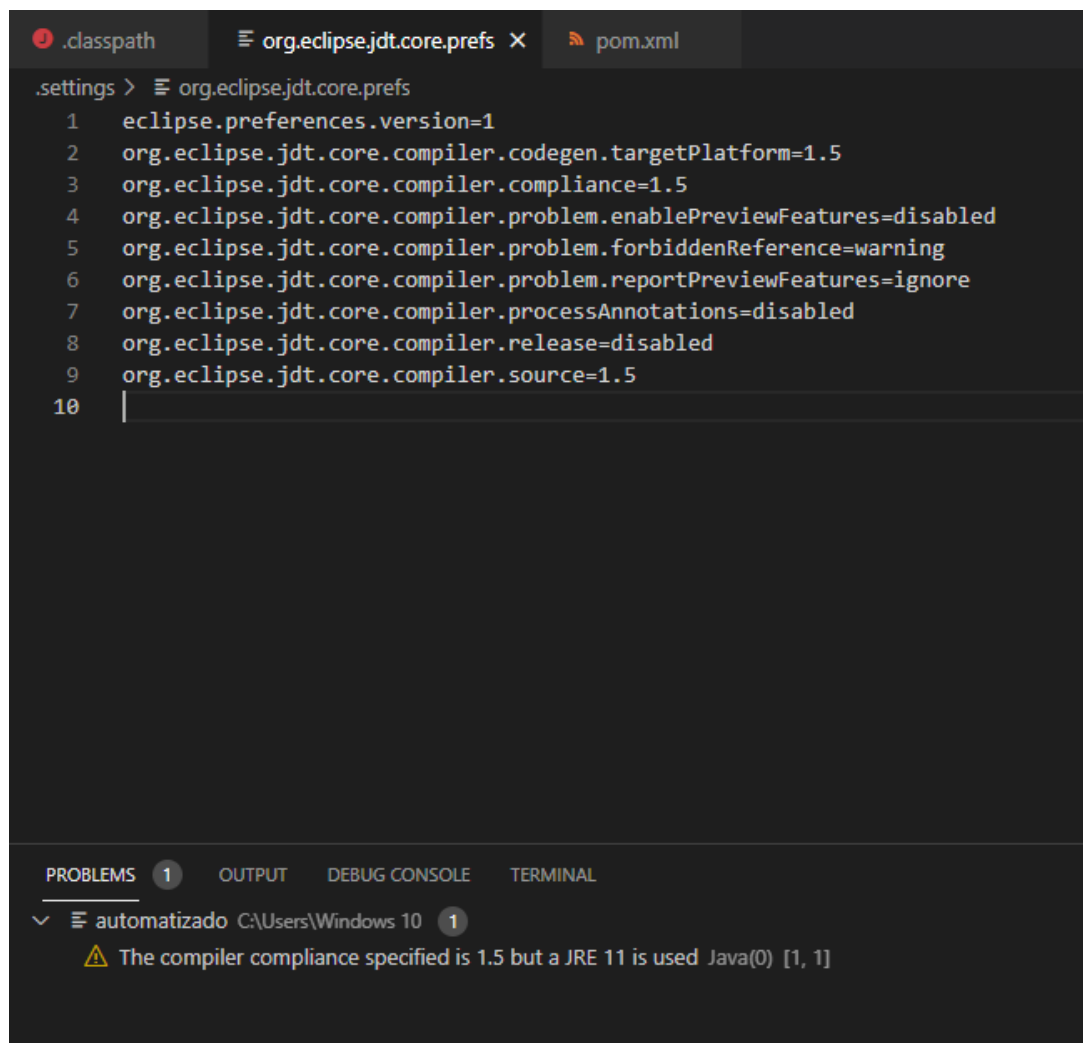
```
10 <attributes>
11   <attribute name="optional" value="true"/>
12   <attribute name="maven.pomderived" value="true"/>
13   <attribute name="test" value="true"/>
14 </attributes>
15 </classpathentry>
16 <classpathentry kind="con" path="org.eclipse.jdt.launching.JRE_CONTAINER/org.eclipse.jdt.internal.debug.ui.launcher.StandardVMType/J2SE-1.5">
17   <attributes>
18     <attribute name="maven.pomderived" value="true"/>
19   </attributes>
20 </classpathentry>
21 <classpathentry kind="con" path="org.eclipse.m2e.MAVEN2_CLASSPATH_CONTAINER">
22   <attributes>
23     <attribute name="maven.pomderived" value="true"/>
24   </attributes>
25 </classpathentry>
```

Podemos notar que no final da linha 16 temos uma versão do JSE bastante diferente da versão que instalamos, então precisamos fazer a alteração no código para versão que estamos utilizando “JavaSE-11”.



The screenshot shows an IDE with an XML file open. Line 16 contains the following code: `<classpathentry kind="con" path="org.eclipse.jdt.launching.JRE_CONTAINER/org.eclipse.jdt.internal.debug.ui.launcher.StandardVMType/JavaSE-11"/>`. The text "JavaSE-11" is highlighted with a red box. Below the code editor, the "PROBLEMS" panel is visible, showing a warning: "The compiler compliance specified is 1.5 but a JRE 11 is used Java(0) [1, 1]".

Após alterarmos para a versão correta, podemos perceber que um dos alertas não será mais exibido, pois agora estamos informando a versão correta no path. Mas continuamos com um alerta referente a versão do compilador. Para resolver esse problema vamos abrir no VS Code o arquivo “org.eclipse.jdt.core.prefs” que se encontra dentro da pasta “.settings” localizada na raiz do nosso projeto.

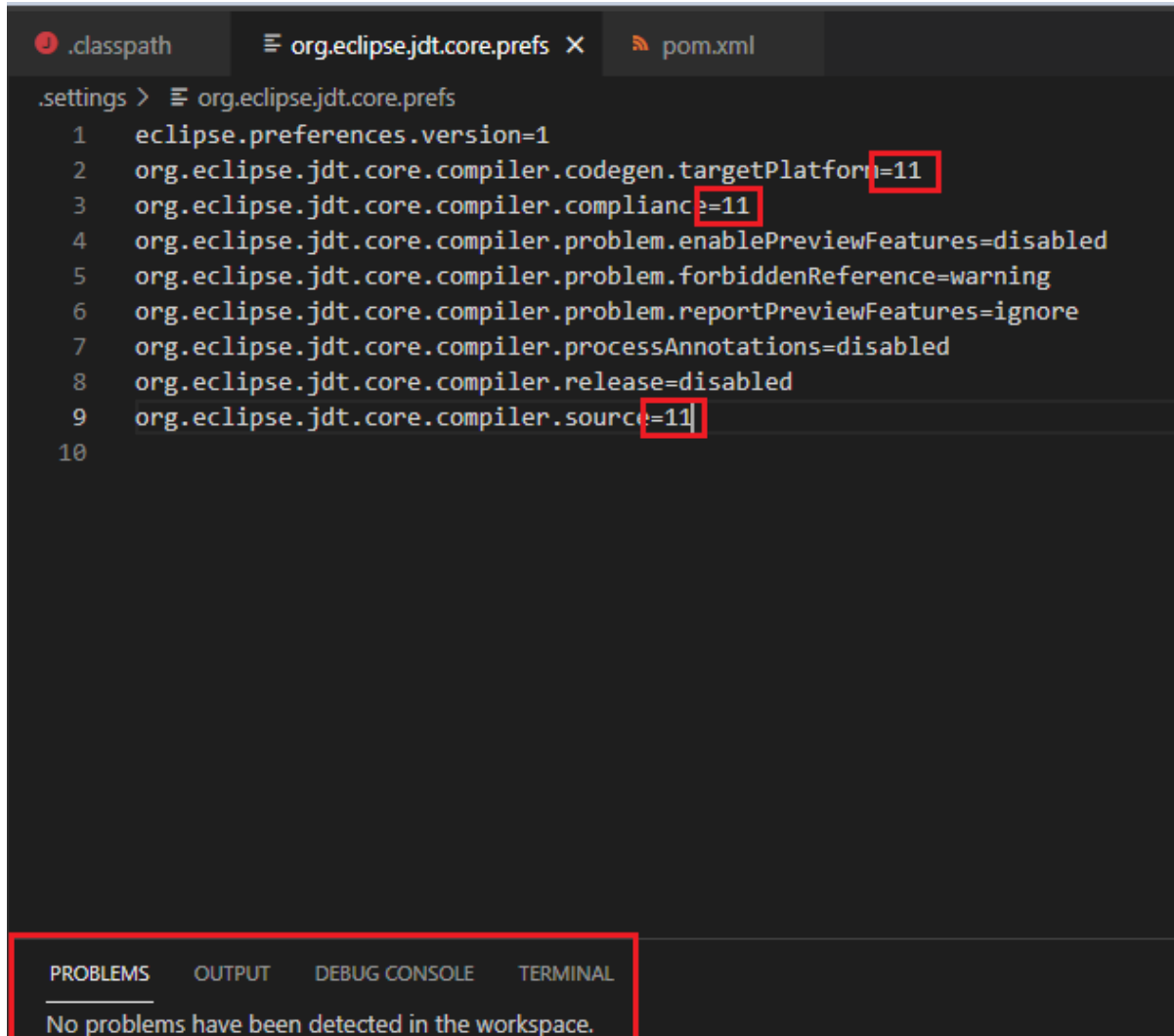


The screenshot shows the VS Code editor with the file "org.eclipse.jdt.core.prefs" open. The file contains the following properties:

```
.settings > org.eclipse.jdt.core.prefs
1  eclipse.preferences.version=1
2  org.eclipse.jdt.core.compiler.codegen.targetPlatform=1.5
3  org.eclipse.jdt.core.compiler.compliance=1.5
4  org.eclipse.jdt.core.compiler.problem.enablePreviewFeatures=disabled
5  org.eclipse.jdt.core.compiler.problem.forbiddenReference=warning
6  org.eclipse.jdt.core.compiler.problem.reportPreviewFeatures=ignore
7  org.eclipse.jdt.core.compiler.processAnnotations=disabled
8  org.eclipse.jdt.core.compiler.release=disabled
9  org.eclipse.jdt.core.compiler.source=1.5
10
```

Below the editor, the "PROBLEMS" panel shows the same warning: "The compiler compliance specified is 1.5 but a JRE 11 is used Java(0) [1, 1]".

Podemos observar na imagem a cima, que a versão passada para o compilador é a 1.5, mas nós estamos utilizando em nossa máquina a versão 11 e por isso é exibido o alerta no terminal, para corrigirmos esse alerta basta alterarmos a versão que está no arquivo para a versão que estávamos utilizando em nossa máquina.

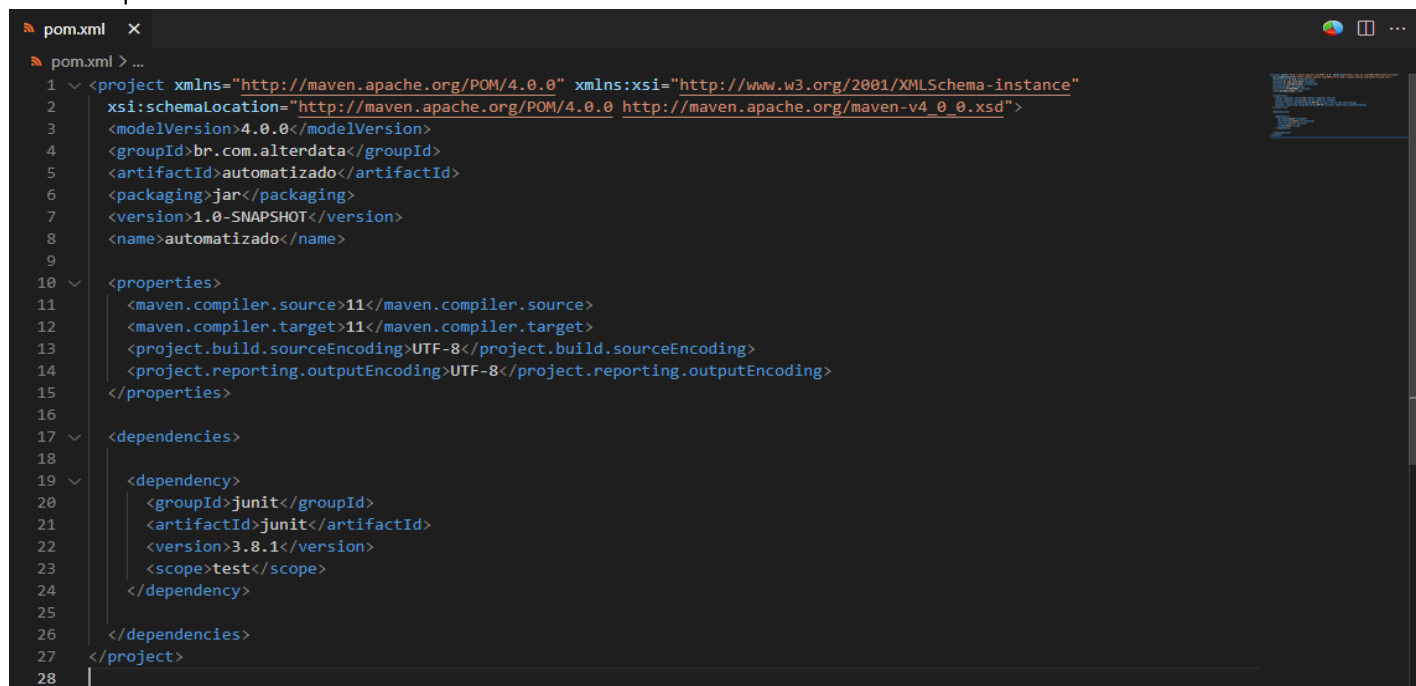


Com os ajustes feitos corretamente, nenhum alerta será exibido em nosso terminal.

Resolvido os alertas seguiremos com as configurações do nosso pom.xml, dentro das propriedades do nosso arquivo adicionaremos o tipo de codificação do nosso projeto com o seguinte trecho de código:

```
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
```

Nosso arquivo deverá ficar dessa forma:



```
1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
3   <modelVersion>4.0.0</modelVersion>
4   <groupId>br.com.alterdata</groupId>
5   <artifactId>automatizado</artifactId>
6   <packaging>jar</packaging>
7   <version>1.0-SNAPSHOT</version>
8   <name>automatizado</name>
9
10  <properties>
11    <maven.compiler.source>11</maven.compiler.source>
12    <maven.compiler.target>11</maven.compiler.target>
13    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
14    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
15  </properties>
16
17  <dependencies>
18
19    <dependency>
20      <groupId>junit</groupId>
21      <artifactId>junit</artifactId>
22      <version>3.8.1</version>
23      <scope>test</scope>
24    </dependency>
25
26  </dependencies>
27 </project>
28
```

Note que deletamos a tag `<url>http://maven.apache.org</url>` pois ela não será útil para nós.

O próximo passo é adicionar as dependências que usaremos, são através delas que teremos acesso as ferramentas necessárias para realização dos nossos testes. Vamos usar as seguintes dependências:

**Junit** – Utilizado para criação dos testes.

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.11</version>
  <scope>test</scope>
</dependency>
```

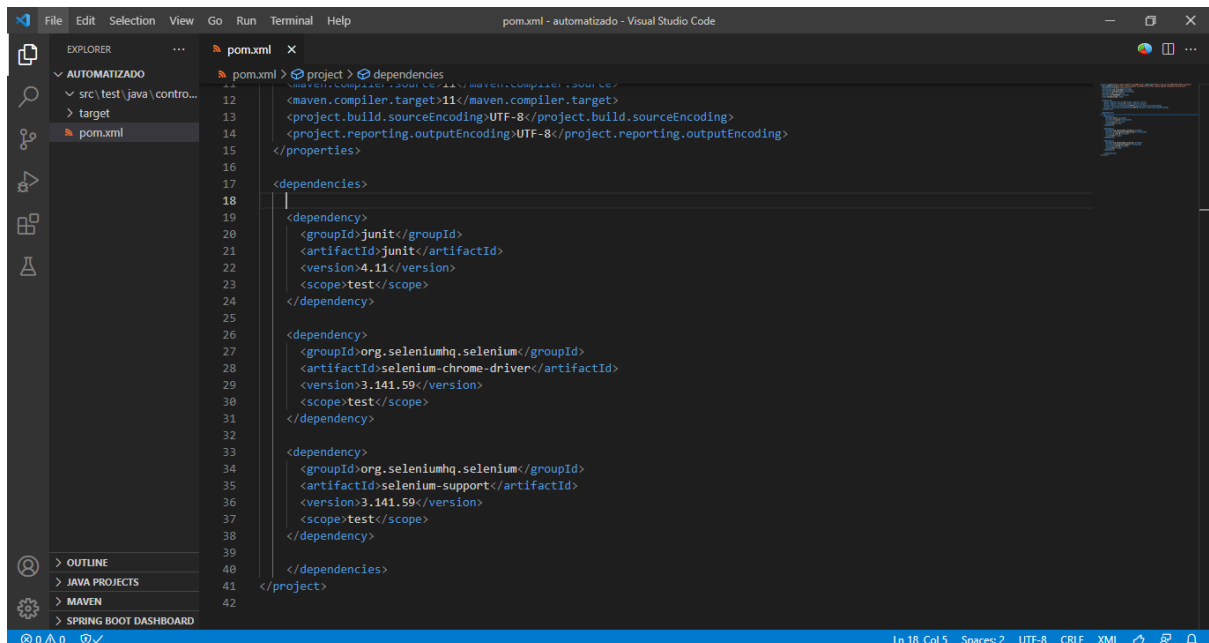
**Selenium-Chrome-Driver** – Utilizado para manipular o Google Chrome.

```
<dependency>
  <groupId>org.seleniumhq.selenium</groupId>
  <artifactId>selenium-chrome-driver</artifactId>
  <version>3.141.59</version>
  <scope>test</scope>
</dependency>
```

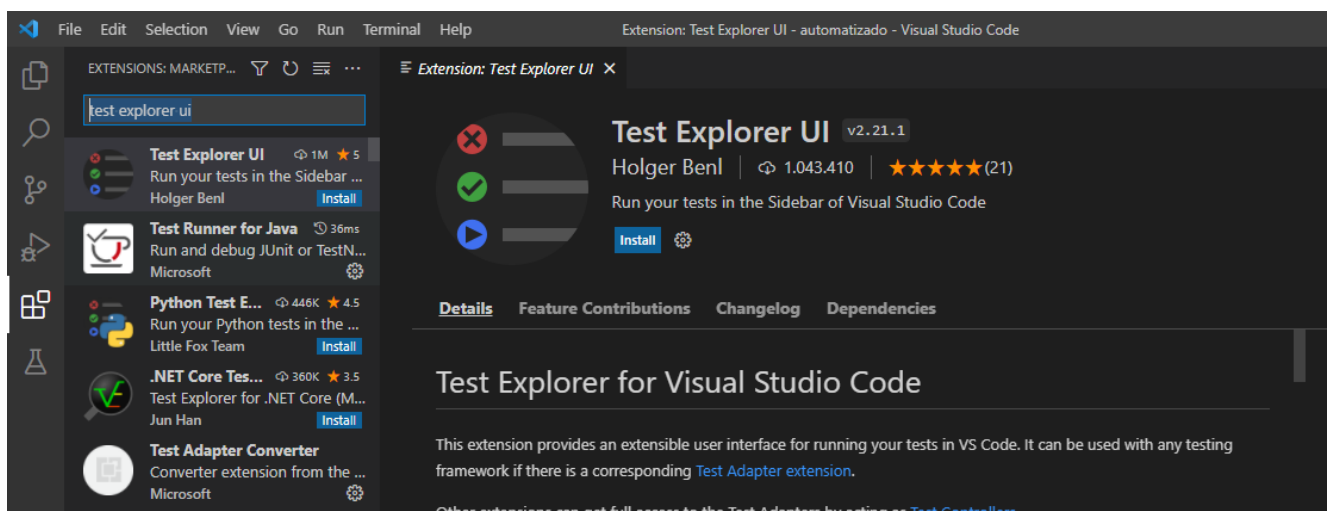
**Selenium-Support** – Ferramentas adicionais para interação com os elementos web.

```
<dependency>
  <groupId>org.seleniumhq.selenium</groupId>
  <artifactId>selenium-support</artifactId>
  <version>3.141.59</version>
  <scope>test</scope>
</dependency>
```

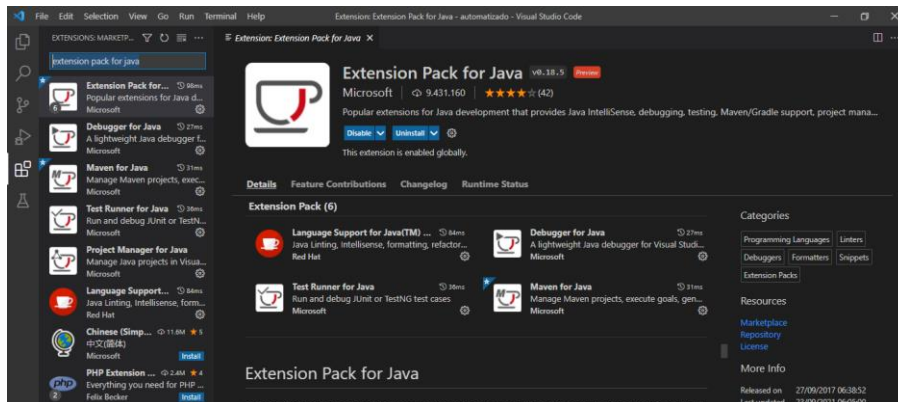
Nossas dependências devem ser adicionadas dentro do bloco <dependencies> como mostra a imagem abaixo:



Feito isso nosso arquivo pom.xml estará configurado, já teremos acesso as ferramentas necessárias para continuarmos com o projeto, porém antes de começarmos a codificar temos que adicionar no VS Code as extensões necessárias para começarmos os trabalhos, no menu de extensões iremos procurar por “test explorer ui” e instalar.



Também precisaremos do pacote de extensões para **JAVA**, sem ele nós não conseguiremos trabalhar no **VS Code** utilizando a linguagem **JAVA**, para instalar essa extensão pesquise por “ **extension pack for java** ” e instale.



É importante saber que esse pacote de extensões para JAVA já traz com ele 6 outras extensões que são:

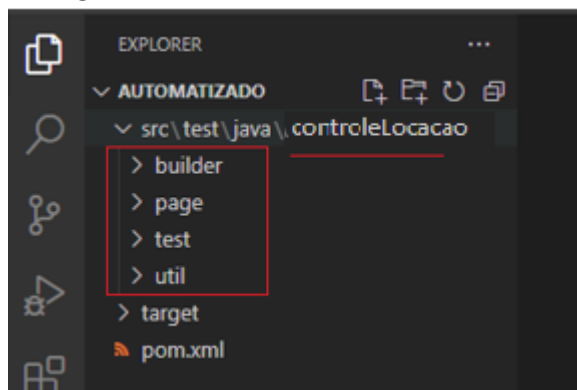
- Language Support for Java
- Debugger for Java
- Test Runner for Java
- Maven for Java
- Project Manager for Java
- Visual Studio IntelliCode

Então ao encontrar essas extensões em seu VS Code, não se preocupe, pois, elas foram instaladas junto com o **extension pack for java**.

Agora estamos prontos para começar a desenvolver nosso projeto de testes. O primeiro passo que devemos seguir é a criação da estrutura de pacotes que serão utilizados, adicionaremos então os seguintes pacotes:

- Builder: Pacote onde serão armazenados os builders do sistema.
- page: Pacote onde serão armazenadas todas as pages do nosso sistema.
- Test: Pacote onde serão armazenados todos os testes do nosso sistema.
- Útil: Pacote onde serão armazenados os uteis do sistema.

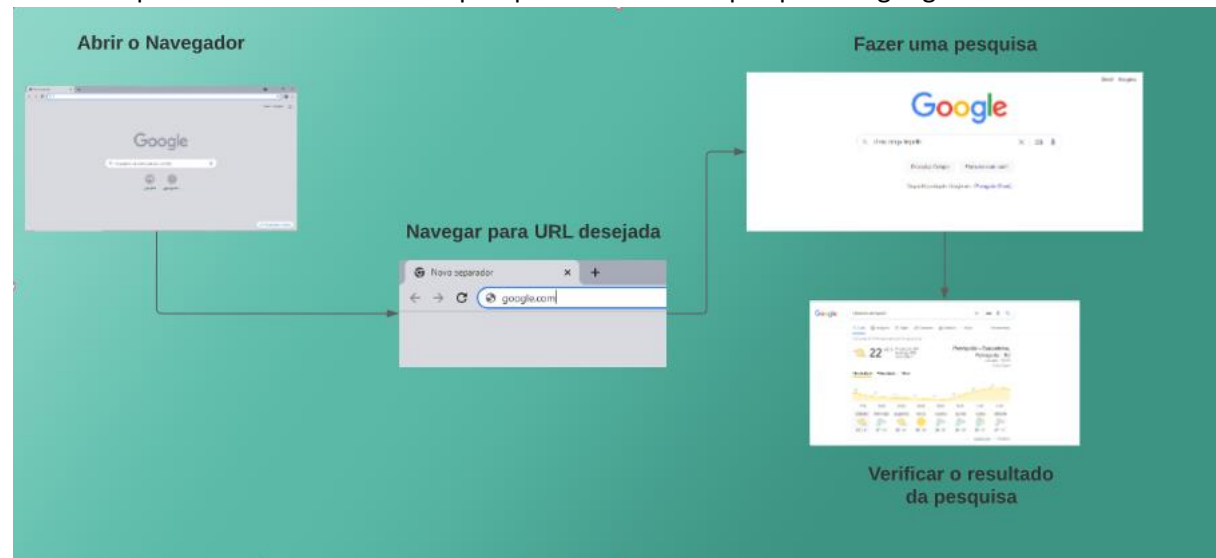
A imagem abaixo mostra como deve ficar a estrutura de pacotes após esse passo.



Note que os pacotes foram criados dentro de “controleLocacao”, que foi o pacote definido na criação do projeto.

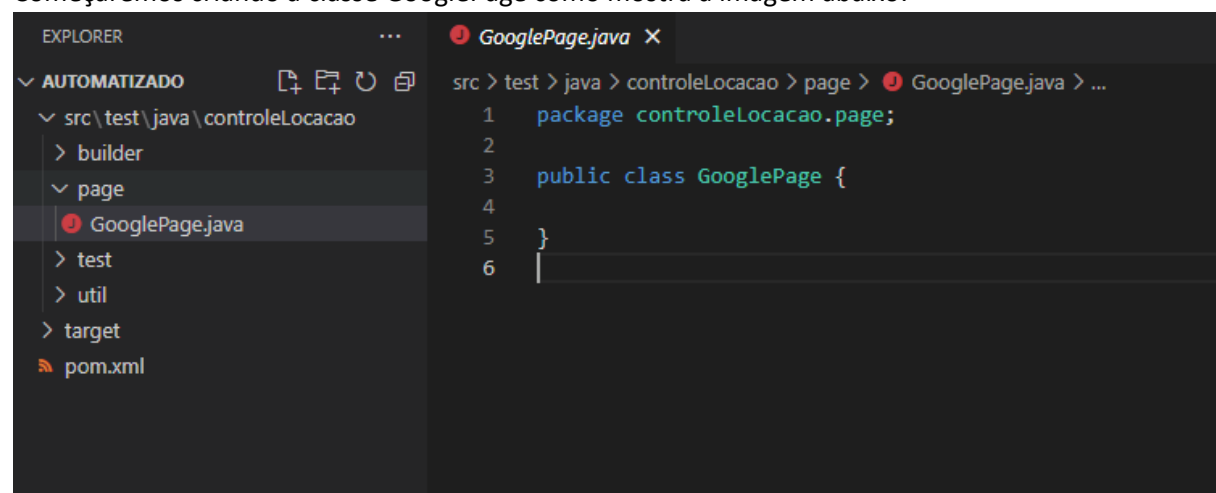
Com a estrutura de pacotes adicionada, vamos enfim criar nossa primeira Classe, mas antes precisamos entender o fluxo do nosso projeto. Estamos criando um projeto de testes automatizados para aplicações Web, logo nosso sistema deve ser capaz de acessar e interagir com páginas Web, simular a rotina do usuário e verificar se as condições passadas resultam no que é esperado.

Abaixo é apresentado uma rotina simples para realizar uma pesquisa no google.



Tendo como base o fluxograma a cima, criaremos nosso primeiro teste automatizado para ser capaz de realizar uma pesquisa simples no google.

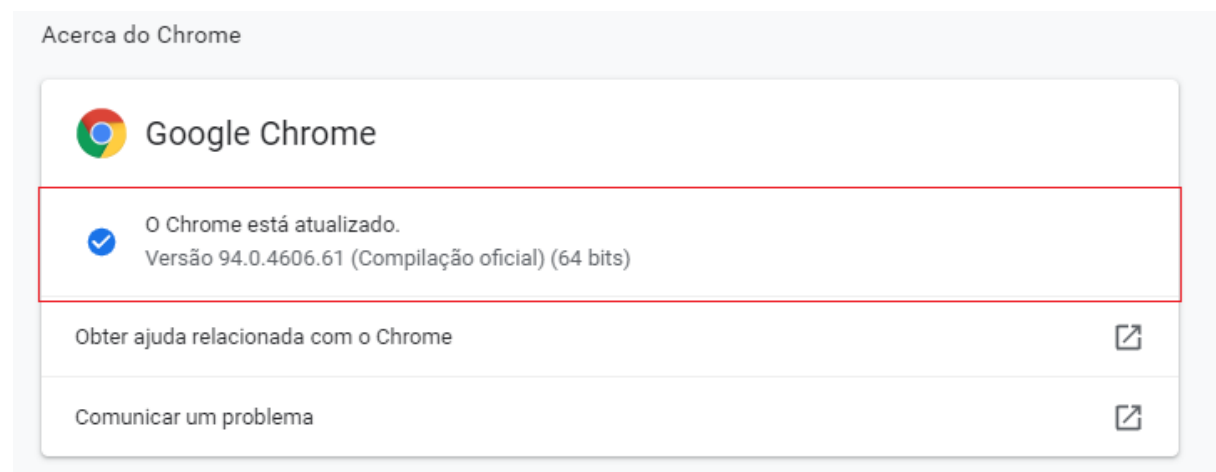
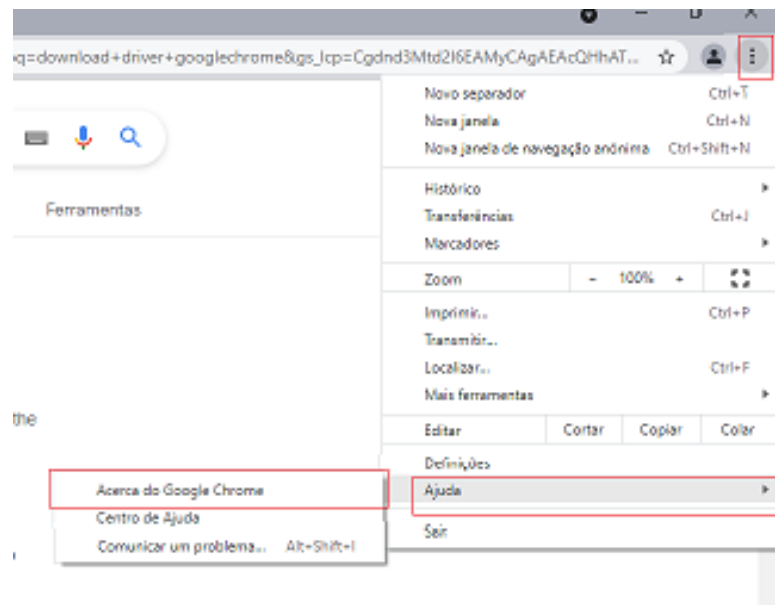
Começaremos criando a classe GooglePage como mostra a imagem abaixo:



Todas as nossas classes “**Page**” serão criadas dentro do pacote “**page**”, essas classes serão utilizadas para mapear as **páginas** web desejadas e transforma-las em objetos para serem manipulados, esse padrão é conhecido como PageObject.

Olhando novamente o fluxograma vemos que o primeiro passo é abrir o navegador, e para fazer isso precisamos do **driver** dele, pois é através do driver que seremos capazes de manipular toda a parte de interação com o nosso navegador como abrir, fechar, navegar para sites, etc.

Primeiramente vamos verificar a versão do nosso googleChrome, clicando no “**menu superior direito/ajuda/sobre o google chrome**” conseguimos ver essa informação como mostram as imagens abaixo:



Podemos verificar que estamos na versão 94, então precisamos baixar o driver da versão 94. Acesse o link abaixo e faça o download do driver na **versão equivalente ao seu navegador de acordo com o seu sistema operacional**.

Link de download do driver: <https://chromedriver.chromium.org/downloads>



No diretório do nosso projeto, dentro do pacote **src** iremos criar um pacote chamado **resource** como na imagem abaixo:

Disco Local (C:) > Usuários > Windows 10 > automatizado > src

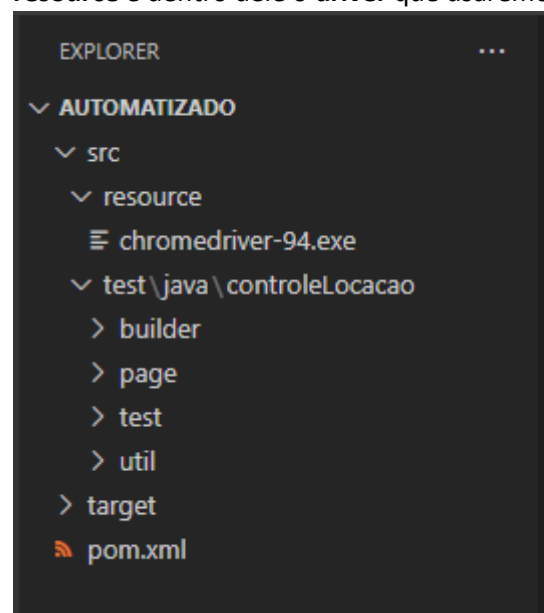
Nome	Data de modificação	Tipo	Tamanho
resource	30/09/2021 09:27	Pasta de arquivos	
test	23/09/2021 12:41	Pasta de arquivos	

Dentro do pacote que acabamos de criar (**resource**), iremos extrair o driver que baixamos, ele deve vir como nome de “**chromedriver.exe**” mas podemos o renomear para facilitar a identificação da versão do driver para outros usuários que fizerem uso do sistema.

Disco Local (C:) > Usuários > Windows 10 > automatizado > src > resource

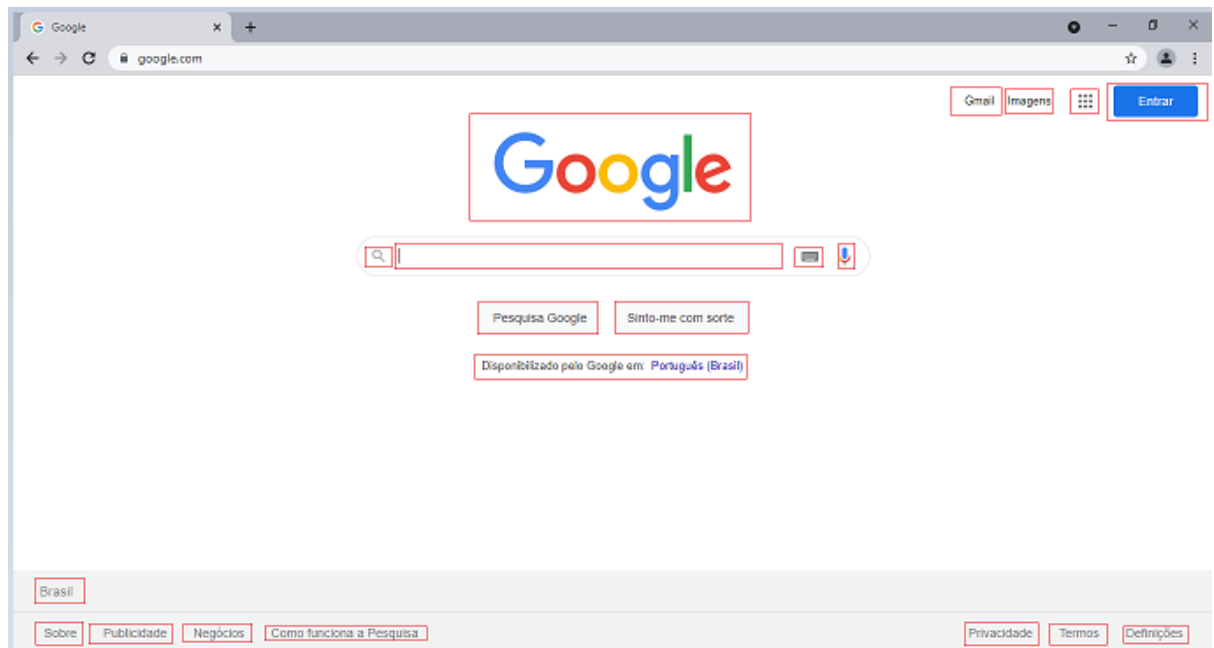
Nome	Data de modificação	Tipo	Tamanho
chromedriver-94.exe	22/09/2021 16:15	Aplicativo	10.969 KB

Se olharmos nosso projeto no **VS Code**, veremos que foi adicionado dentro de **src** o pacote **resource** e dentro dele o **driver** que usaremos.

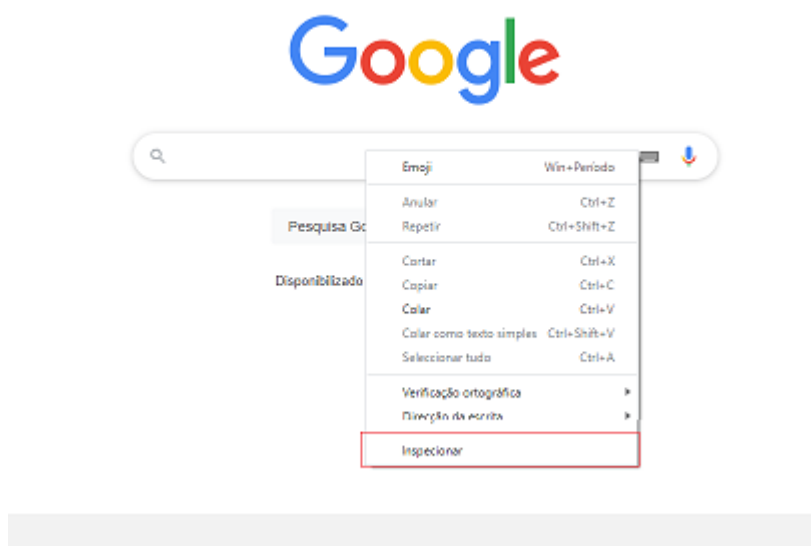


Com isso, estamos prontos para interagir com o navegador e começar nossos testes.

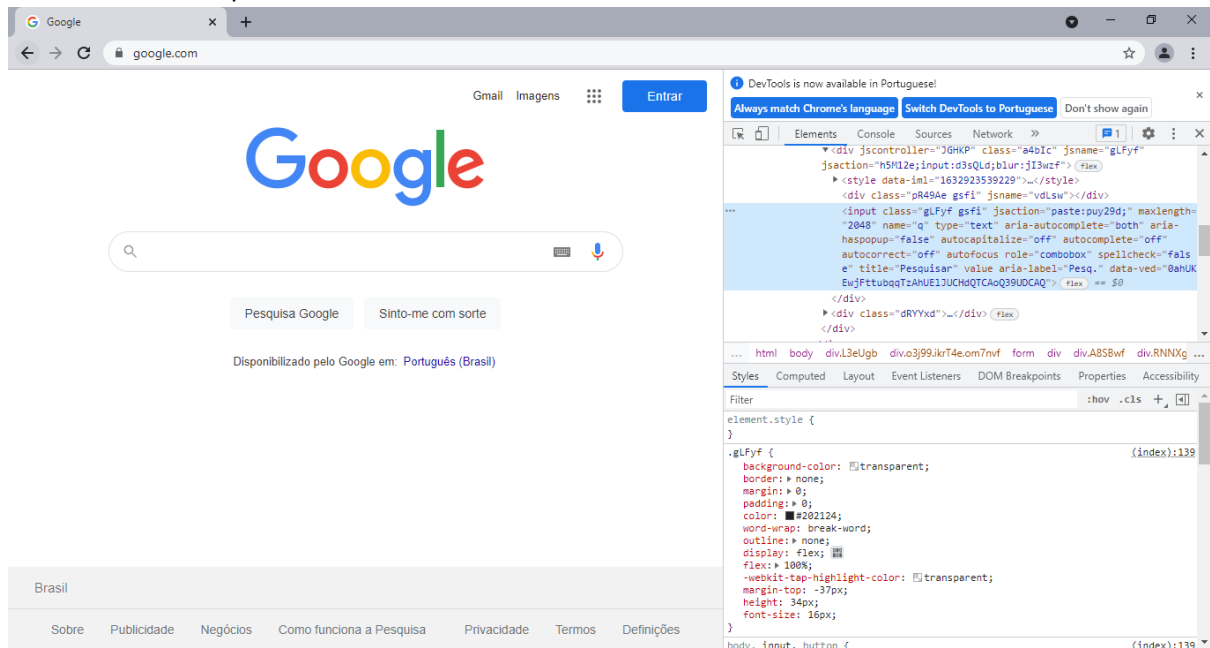
Para testarmos uma página web precisamos mapear os elementos que iremos utilizar. Na imagem abaixo vemos a página inicial do Google, temos vários elementos nessa página (como mostra os destaques em vermelho).



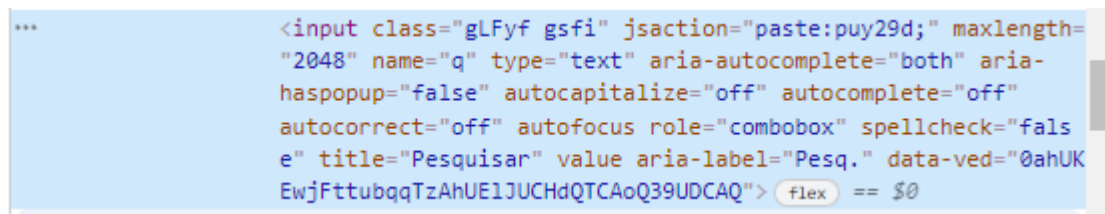
Mas para o nosso teste precisamos apenas do elemento “**campo de pesquisa**”, para mapear esse campo precisamos encontrar o seu identificador e faremos isso clicando com o botão direito do mouse em cima do elemento desejado e logo após selecionando a opção “**inspecionar**” conforme mostra a imagem abaixo:



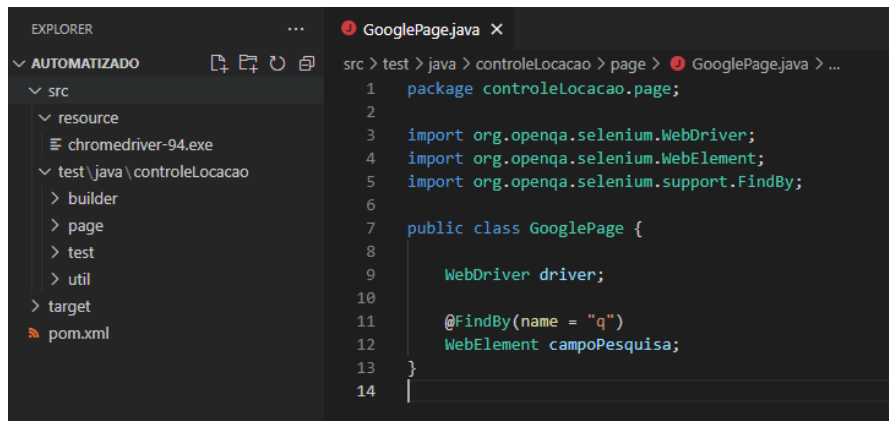
Ao fazer isso, será apresentado em tela divida o código fonte referente ao elemento que foi selecionado como podemos ver:



O que precisamos agora é encontrar algum identificador desse elemento, os identificadores podem ser vários como, ID, name, class, css entre outros, basta que seja algo único daquele elemento.



Conforme mostra a imagem a cima, nosso campo pesquisa é um campo do tipo **input** que possui alguns identificadores como **class="gLFyf gsfi"** ou **name="q"**, usaremos então o identificador **"name="q"** para mapear esse campo.

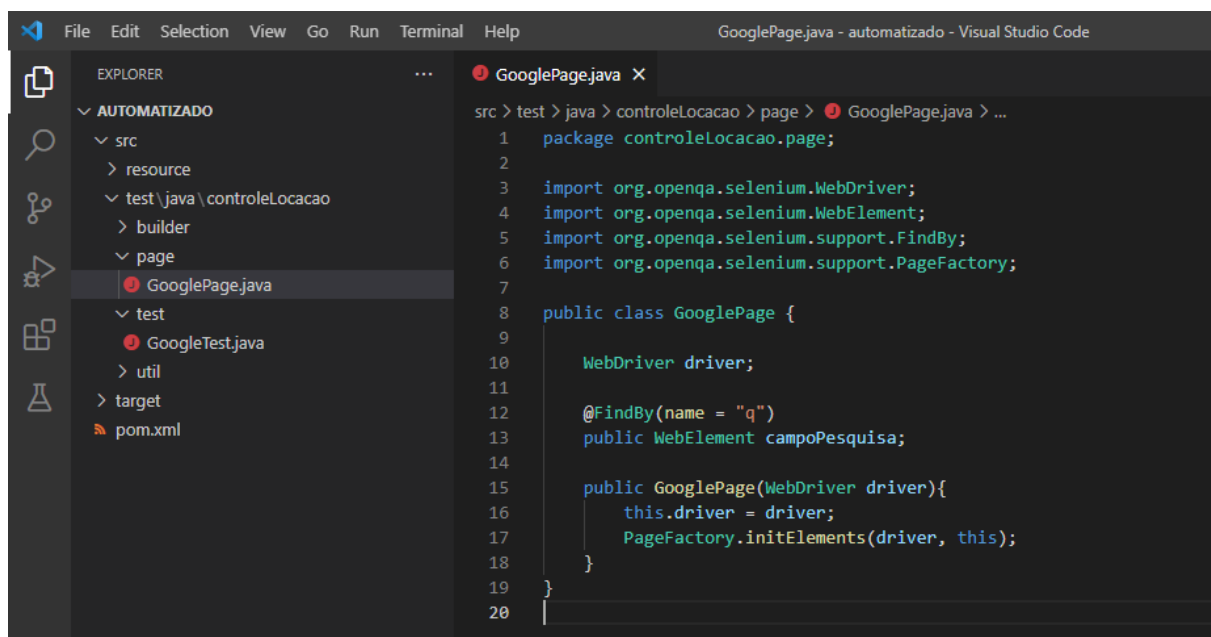


```
1 package controleLocacao.page;
2
3 import org.openqa.selenium.WebDriver;
4 import org.openqa.selenium.WebElement;
5 import org.openqa.selenium.support.FindBy;
6
7 public class GooglePage {
8
9     WebDriver driver;
10
11     @FindBy(name = "q")
12     WebElement campoPesquisa;
13 }
14
```

Como mostra a imagem a cima, em nossa classe **GooglePage** criaremos o atributo **driver** do tipo **WebDriver**, esse **driver** será utiliza pela nossa **Page Google** para fazer a interação com os elementos que iremos mapear.

Em seguida criamos o elemento **campoPesquisa** do tipo **WebElement**, mas como fazer o nosso sistema saber que o **campoPesquisa** que criamos é referente ao **campo de pesquisa do Google** ? Para referenciarmos um **elemento criado**, há um elemento **web real**, nós utilizamos a anotação **@FindBy()** e dentro dela colocamos o **identificador** que encontramos daquele elemento, no nosso caso já havíamos definido que iríamos utilizar o identificador **name="q"** para mapear o campo de pesquisa, mas também poderíamos ter utilizado o identificador **class="gLFyf gsfi"** nesse caso a linha 11 do nosso código ficaria **@FindBy (class="gLFyf gsfi")**. O mesmo se aplicará para outros identificadores como **id**, **css**, etc.

Com o elemento que queremos mapeado, precisamos agora iniciar esse elemento para podermos usá-lo, faremos isso dentro do nosso construtor.



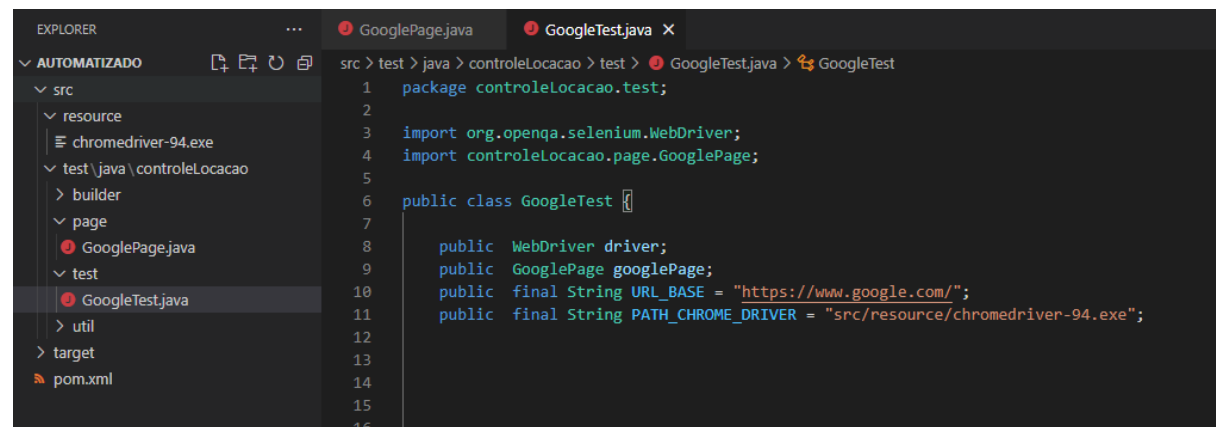
```
1 package controleLocacao.page;
2
3 import org.openqa.selenium.WebDriver;
4 import org.openqa.selenium.WebElement;
5 import org.openqa.selenium.support.FindBy;
6 import org.openqa.selenium.support.PageFactory;
7
8 public class GooglePage {
9
10     WebDriver driver;
11
12     @FindBy(name = "q")
13     public WebElement campoPesquisa;
14
15     public GooglePage(WebDriver driver){
16         this.driver = driver;
17         PageFactory.initElements(driver, this);
18     }
19 }
20
```

Criamos na linha 15 um **construtor** da nossa classe **GooglePage**, esse **construtor** recebe como **parâmetro** um **driver**, que usaremos para inicializar os elementos mapeados, na linha 16 referenciamos o **driver** que criamos na linha 10 para o **driver** que será passado no construtor da

classe como parâmetro e finalmente na linha 17 inicializamos os nossos elementos utilizando o **PageFactory**, que transforma nossa página em um objeto de fato, transformando os **elementos web** da nossa classe em atributos que podemos interagir.

Tendo feito isso, nossa Page já está pronta para ser utilizada, iremos criar então dentro do pacote **test** a classe **GoogleTest**.

Todas as nossas classes de testes devem ficar dentro do pacote **test**, e ter ao final do seu nome a palavra “ **Test** ”.



```
1 package controleLocacao.test;
2
3 import org.openqa.selenium.WebDriver;
4 import controleLocacao.page.GooglePage;
5
6 public class GoogleTest {
7
8     public WebDriver driver;
9     public GooglePage googlePage;
10    public final String URL_BASE = "https://www.google.com/";
11    public final String PATH_CHROME_DRIVER = "src/resource/chromedriver-94.exe";
12
13
14
15
16
```

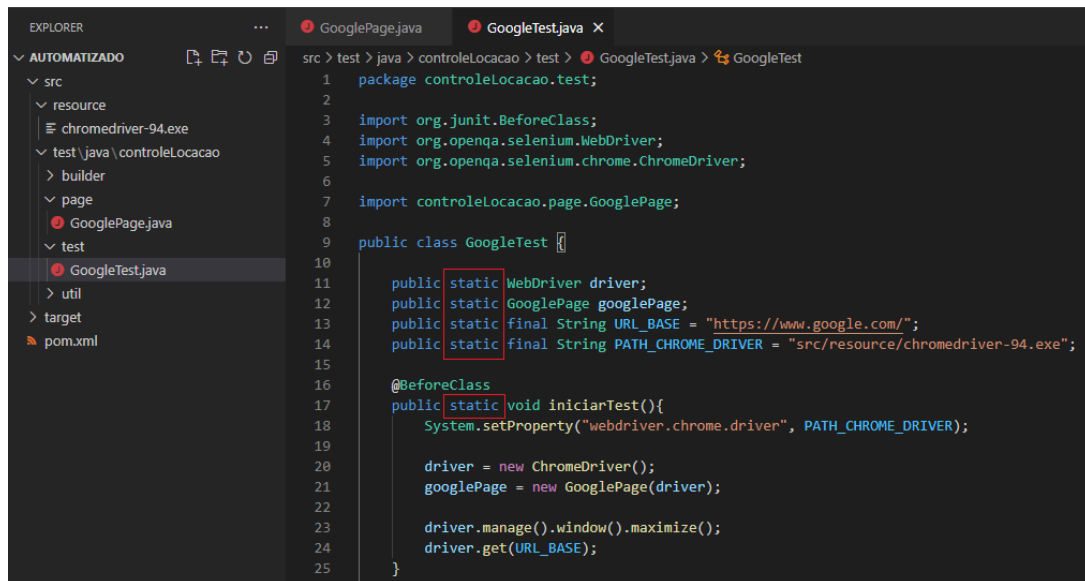
Como mostra a imagem a cima, vamos precisar criar alguns atributos em nossa classe para realizar os testes.

Primeiro vamos precisar de um **driver** para interagir com o **navegador**, então assim como criamos em nossa **page**, aqui também iremos criar um **driver** do tipo **WebDriver**.

Logo abaixo na linha 10 iremos criar uma **instancia** da nossa page, pois será através dessa **instancia** que teremos acesso aos **elementos** que lá foram **mapeados**.

Na linha 11 criamos uma **constante** que irá receber a **url base**, ou seja a **url principal** para onde nosso navegador deve ser direcionado **no início de cada teste**. **IMPORTANTE**: essa url deve estar **completa** para funcionar (**incluindo o https://**).

Por fim na linha 12 criamos outra **constante** que irá receber o **caminho** onde está o **driver** que **baixamos**, como criamos o pacote **resource** dentro do diretório **src** passamos o seguinte caminho: **src/resource/nomedodriver.exe**.



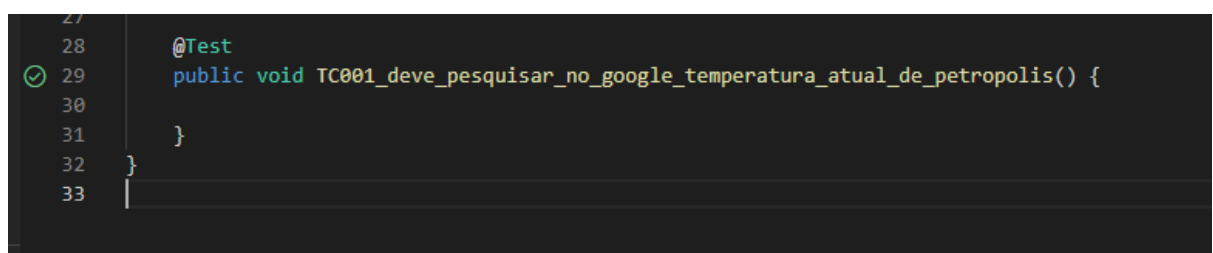
Criaremos em seguida nosso **método iniciarTest()**, esse método será executado **um a vez antes de todos os testes da classe**, mas para isso acontecer precisamos adicionar a anotação **@BeforeClass** antes do método, também precisamos colocar o método e todos os atributos e instancias que serão utilizados dentro desse método como **estáticos**.

Dentro do nosso método faremos tudo o que é necessário fazer **antes de iniciar qualquer teste**, na linha 18 iremos setar as propriedades do sistema para receber o **caminho do driver** que baixamos, esse caminho poderia ser passado diretamente como uma **string**, porém para nosso código ficar mais legível e organizado criamos uma constante que recebe esse valor e à passamos.

Em seguida nas linhas 20 e 21 **instanciamos nossos objetos**, observe que precisamos passar o **driver** ao instanciar **GooglePage**, pois criamos um construtor na classe que precisa receber o **driver**.

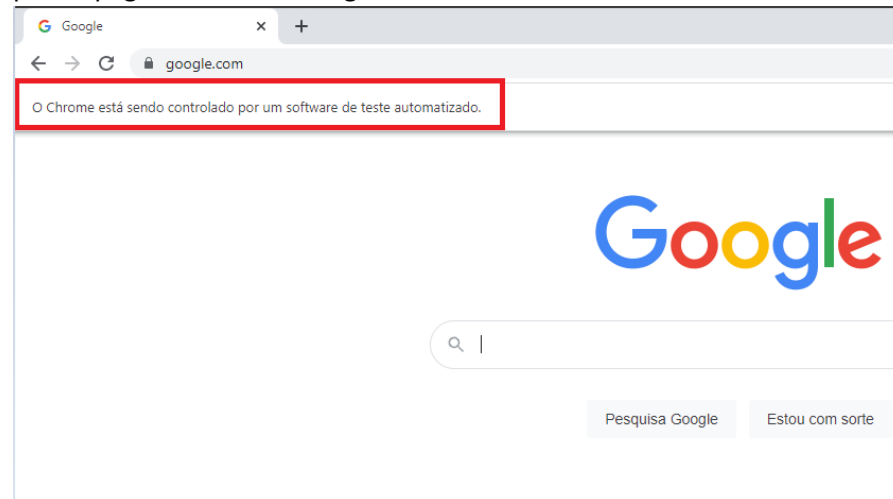
Na linha 23 configuramos nosso navegador para ser **maximizado** na tela e em seguida na linha 24 **navegamos** para a **url base**.

Agora vamos criar nosso primeiro teste. Como a imagem abaixo mostra primeiramente temos que adicionar a anotação **@Test** do junit para nosso sistema identificar que o método criado abaixo será um teste. Abaixo da anotação criamos um método seguindo o padrão de nomenclatura apresentado.



Após isso já será possível executar o teste clicando no player verde ao lado, mesmo sem adicionar nada dentro dele. Se fizermos isso o navegador irá abrir, ser maximizado e navegar

para a página inicial do Google.



Isso acontece por que em nosso método **iniciarTest()** definimos que o navegador iria abrir, ser maximizado e ir para página inicial do google. Como esse método é iniciado **antes de qualquer classe de teste**, partimos do princípio que já estaremos na página inicial do google **antes** de começar os testes.

Para fazer a pesquisa que desejamos e teste-la, basta agora dentro do método de teste incluirmos toda a lógica referente ao que queremos fazer. Para realizar uma pesquisa, precisamos escrever no campo de pesquisa e apertar a tecla ENTER por exemplo, então será exatamente isso que iremos fazer.

```
28     }
29
30     @Test
31     public void TC001_deve_pesquisar_no_google_temperatura_atual_de_petropolis() {
32
33         googlePage.campoPesquisa.sendKeys("Temperatura atual em petropópolis", Keys.ENTER);
34
35         Assert.assertEquals(driver.getTitle(), "Temperatura atual em petropópolis - Pesquisa Google");
36     }
37 }
38 }
```

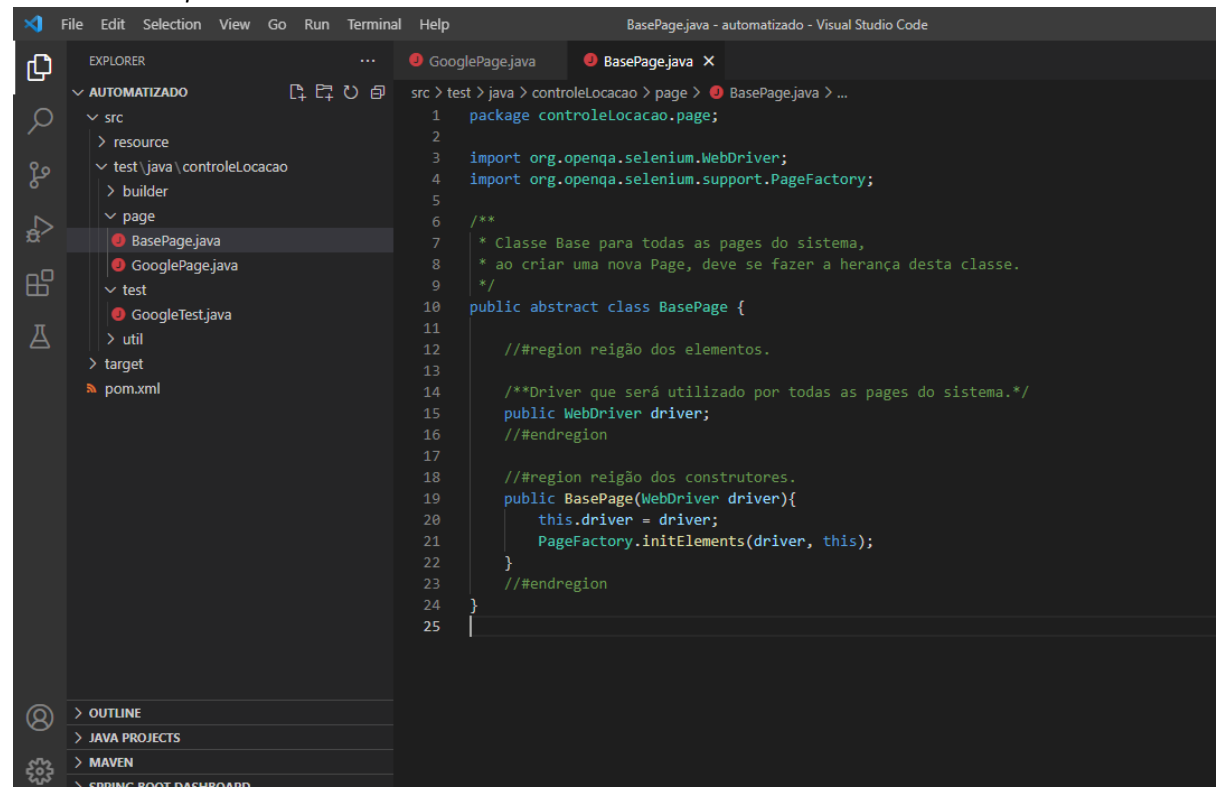
Observe que na linha 33 fazemos exatamente o que descrevemos a cima, nós pegamos o campo de pesquisa da página do google, escrevemos nele e apertamos ENTER, e para testar se nossa pesquisa funcionou de fato, na linha 35 verificamos se o título da página foi alterado de acordo com a pesquisa que fizemos como mostra a imagem abaixo:

```
gRzz45CNpt624audv+wHOJwFAAEAAABieyJvcmlnaW4iOiJodHRwczovL2dzb2dsZS5jb206NDQ
zIiwZmVhdHVyZSI6I1RydXN0VG9rZW5zIiwZbXhwaXJ5IjoxNjI2MjIwNzk5LCJpc1N1YmRvbW
Fpb1I6dHJ1ZX0=" http-equiv="origin-trial">
<meta content="/images/branding/googleg/1x/googleg_standard_color_128dp.png"
itemprop="image">
*** <title>Temperatura atual em petropópolis - Pesquisa Google</title> == $0
<script src="https://apis.google.com/_/scs/abc-static/_/js/k=gapi.gapi.en.M
5RD9.../sv=1/d=1/ed=1/rs=AHp0oo8z3ZIGbS4Q1hdx100-i7gQCAheug/cb=gapi.loaded_0"
nonce async></script>
▶ <script nonce>...</script>
```

Se o título tiver sido alterado de acordo com a pesquisa que fizemos isso quer dizer que o teste funcionou, poderíamos ter validado esse teste de diversas formas diferentes, verificando algum elemento presente na página, ou pela URL, temos diversas formas de validar nosso código no exemplo a cima.

*Mas e se tivermos dezenas de testes e páginas, teremos sempre que digitar todo o bloco que código que utilizamos nas classes anteriores ?*

*Podemos otimizar nosso código para reaproveitar o máximo possível e evitar código repetido, em todas as nossas Pages, precisaremos de um driver então ao invés de criar um driver para cada Page, podemos criar uma classe pai que passará esse driver para todas as suas filhas como no exemplo abaixo:*



```
1 package controleLocacao.page;
2
3 import org.openqa.selenium.WebDriver;
4 import org.openqa.selenium.support.PageFactory;
5
6 /**
7  * Classe Base para todas as pages do sistema,
8  * ao criar uma nova Page, deve se fazer a herança desta classe.
9  */
10 public abstract class BasePage {
11
12     /**#region reigão dos elementos.
13
14     /**Driver que será utilizado por todas as pages do sistema.*/
15     public WebDriver driver;
16     /**#endregion
17
18     /**#region reigão dos construtores.
19     public BasePage(WebDriver driver){
20         this.driver = driver;
21         PageFactory.initlements(driver, this);
22     }
23     /**#endregion
24 }
25
```

Criamos então nossa **BasePage**, essa será a classe **pai** de todas as **Pages** do sistema, sempre que uma **nova Page** for criada a mesma deve **herdar** desta classe, e tudo o que for **de uso comum** entre as pages, deve ficar dentro desta classe. **Todo Page Object criado, terá que ter no final do seu nome a palavra Page**, pois será assim que **identificaremos os Page Objects dentro do sistema**. Essa classe **não poderá ser instanciada** pois só queremos compartilhar o que há de comum entre as pages e não instanciar a base, então observe que à deixamos como **abstrata**, observe também que iniciamos a implementação do **JavaDoc** para deixar nosso projeto melhor documentado e mais fácil de ser compreendido, também iniciamos a separação das **regiões** dentro das classes, para deixar nosso código mais organizado, seguindo **sempre** a seguinte ordem: (**Atributos e Elementos, Construtores e Métodos**).



Agora que criamos uma classe pai para todas as Pages do nosso sistema, nossa GooglePage irá mudar um pouco.

```
GooglePage.java X
src > test > java > controleLocacao > page > GooglePage.java > GooglePage > GooglePage(WebDriver)
1  package controleLocacao.page;
2
3  import org.openqa.selenium.Keys;
4  import org.openqa.selenium.WebDriver;
5  import org.openqa.selenium.WebElement;
6  import org.openqa.selenium.support.FindBy;
7
8  /**Page Object da página google*/
9  public class GooglePage extends BasePage{
10
11      /**#region região dos elementos.
12
13      /**Elemento referente ao campo de pesquisa do google*/
14      @FindBy(name = "q")
15      public WebElement campoPesquisa;
16      /**#endregion
17
18      /**#region região dos construtores.
19
20      /**Construtor da GooglePage*/
21      public GooglePage(WebDriver driver) {
22          super(driver);
23      }
24      /**#endregion
25
26      /**#region região dos métodos.
27
28      /**
29       * Método criado para realizar uma pesquisa no google.
30       * @param textoPesquisa texto que será pesquisado.
31       */
32      public void pesquisarNoGoogle(String textoPesquisa) {
33          campoPesquisa.sendKeys(textoPesquisa, Keys.ENTER);
34      }
35      /**#endregion
36  }
37
```

Note que passamos por herança a BasePage, e com isso nós não precisamos mais criar um driver pois sempre usaremos o driver do pai, também implementamos o JavaDoc e separamos por regiões, faremos isso em todas as classes do sistema para manter nosso código sempre organizado e de fácil leitura. Nossa ultima região é a região dos métodos, temos apenas um método criado por enquanto, esse método terá como responsabilidade fazer uma pesquisa no google com qualquer texto passado a ele.

Agora em nossa classe de teste, não passamos mais o texto direto para o elemento “campoPesquisa”, nós criamos um método que sabe fazer a pesquisa então precisamos apenas chamar esse método e passar o texto que queremos pesquisar.

```
27     }
28
29     @Test
30     public void TC001_deve_pesquisar_no_google_temperatura_atual_de_petropolis() {
31
32         googlePage.pesquisarNoGoogle("Temperatura atual em petropópolis");
33
34         Assert.assertEquals(driver.getTitle(), "Temperatura atual em petropópolis - Pesquisa Google");
35     }
36 }
37
```

Mas ainda podemos otimizar mais a nossa classe teste, criando uma *BaseTest* que terá a mesma utilidade da *BasePage*. Em nossa *BaseTest*, criaremos tudo o que for comum entre nossos testes, e reaproveitaremos os códigos por herança.

```
... GooglePage.java GoogleTest.java BaseTest.java X
src > test > java > controleLocacao > test > BaseTest.java > ...
1 package controleLocacao.test;
2
3 import org.junit.AfterClass;
4 import org.junit.BeforeClass;
5 import org.openqa.selenium.WebDriver;
6 import org.openqa.selenium.chrome.ChromeDriver;
7 /**
8  * Classe Base para todos os testes do sistema.
9  * Ao criar uma nova classe de teste, deve-se herdar desta classe.
10  */
11 public abstract class BaseTest {
12
13     //region reigão dos atributos.
14
15     /**Driver que será utilizado por todos os testes do sistema.*/
16     public static WebDriver driver;
17     public static final String URL_BASE = "https://www.google.com/";
18     public static final String PATH_CHROME_DRIVER = "src/resource/chromedriver-94.exe";
19     //endregion
20
21     //region reigão dos métodos.
22
23     /**Método que inicia todos os pontos necessários antes de um teste ser iniciado*/
24     @BeforeClass
25     public static void iniciarTest(){
26         System.setProperty("webdriver.chrome.driver", PATH_CHROME_DRIVER);
27
28         driver = new ChromeDriver();
29
30         driver.manage().window().maximize();
31         driver.get(URL_BASE);
32     }
33     //endregion
34
35     /**Método que fecha o navegador após os testes serem executados.*/
36     @AfterClass
37     public static void encerrarTest() {
38         driver.quit();
39     }
40 }
```

Criamos então nossa **BaseTest**, essa será a classe **pai** de todas os **Testes** do sistema, sempre que um **novo Teste** for criado, o mesmo deve **herdar** desta classe, e tudo o que for **de uso comum** entre os testes, deve ficar dentro desta classe. **Todo Teste criado, terá que ter no final do seu nome a palavra Test**, pois será assim que **identificaremos os Testes dentro do sistema**. Essa classe **não poderá ser instanciada** pois só queremos compartilhar o que há de comum entre os testes e não instanciar a base, então observe que a deixamos como **abstrata**. Também criamos um método novo, que recebe a anotação **@AfterClass**, esse método será responsável por fechar o navegador ao final dos testes de cada classe.

Com essas alterações feitas, nossa classe teste ficará da seguinte forma:

```
GooglePage.java  GoogleTest.java X  BaseTest.java
src > test > java > controleLocacao > test > GoogleTest.java > GoogleTest
1  package controleLocacao.test;
2
3  import org.junit.Assert;
4  import org.junit.BeforeClass;
5  import org.junit.Test;
6
7  import controleLocacao.page.GooglePage;
8
9  /**Classe de teste para a pagina do google*/
10 public class GoogleTest extends BaseTest {
11
12     //region região dos atributos.
13
14     public static GooglePage googlePage;
15     //endregion
16
17     //region região dos métodos;
18
19     /**Método que inicia todos os pontos necessários antes dos testes serem executados*/
20     @BeforeClass
21     public static void iniciarTeste() {
22         googlePage = new GooglePage(driver);
23     }
24
25     //region região dos testes.
26
27     @Test
28     public void TC001_deve_pesquisar_no_google_temperatura_atual_de_petropolis() {
29
30         googlePage.pesquisarNoGoogle("Temperatura atual em petropópolis");
31
32         Assert.assertEquals(driver.getTitle(), "Temperatura atual em petropópolis - Pesquisa Google");
33     }
34     //endregion
35 }
36
```