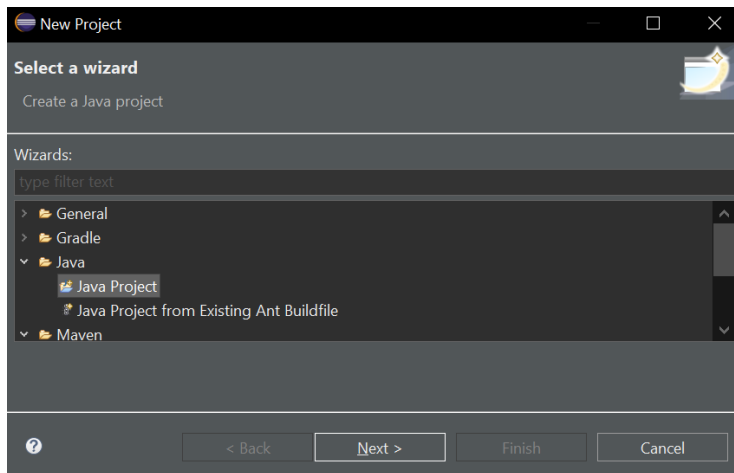
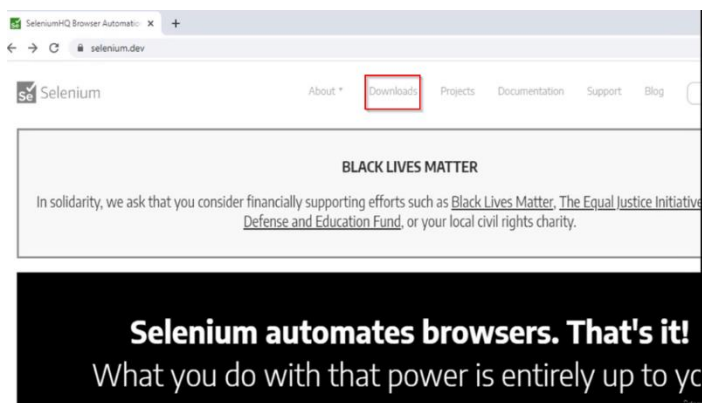


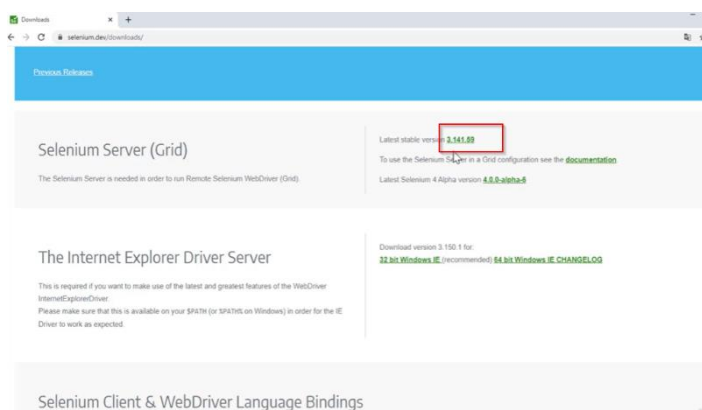
Creación de proyecto y primer script



Se creara un proyecto con java tradicional sin utilizar repositorios externo, para eso lo primero que hacemos es crear un Java Project

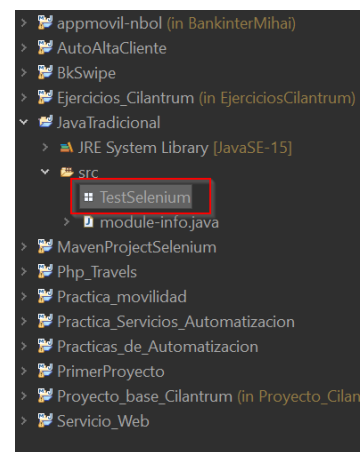


A continuación se pasara a descargar la librería de Selenium desde la propia web para eso vamos a Selenium.dev y dentro pulsamos Downloads



Nos descargamos la ultima versión que exista sobre Selenium Server(Grid)

Dentro del proyecto se crea un paquete que se llama TestSelenium

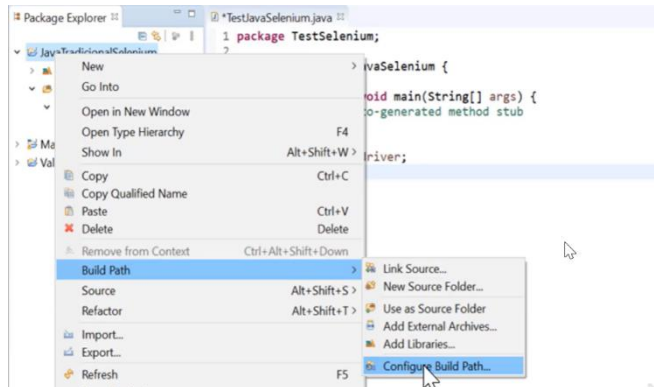


```

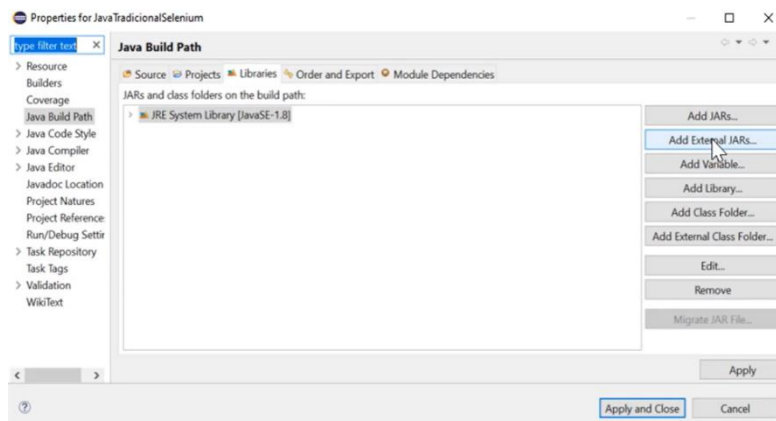
1 package TestSelenium;
2
3 public class TestJavaSelenium {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7
8     }
9
10
11
12
13 }
14
15 }
16

```

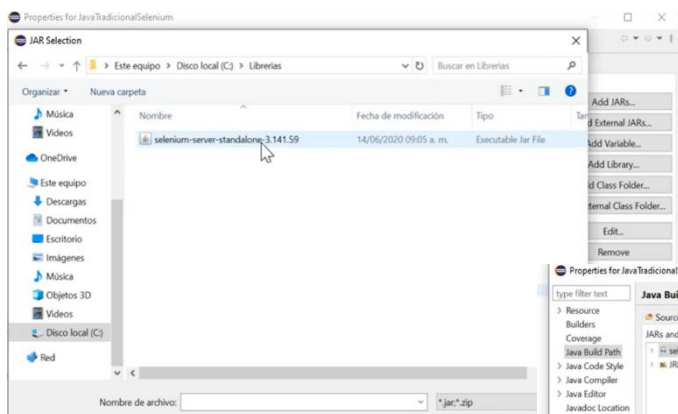
Dentro del paquete se crea la clase main



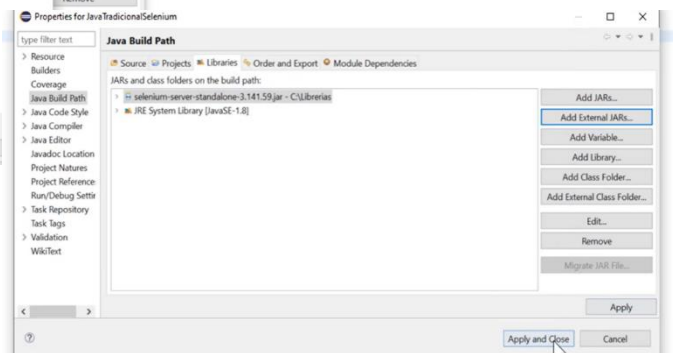
Para importar las librerías de Selenium, se hace click derecho sobre el proyecto y nos vamos a Build Path/Configure Build Path..



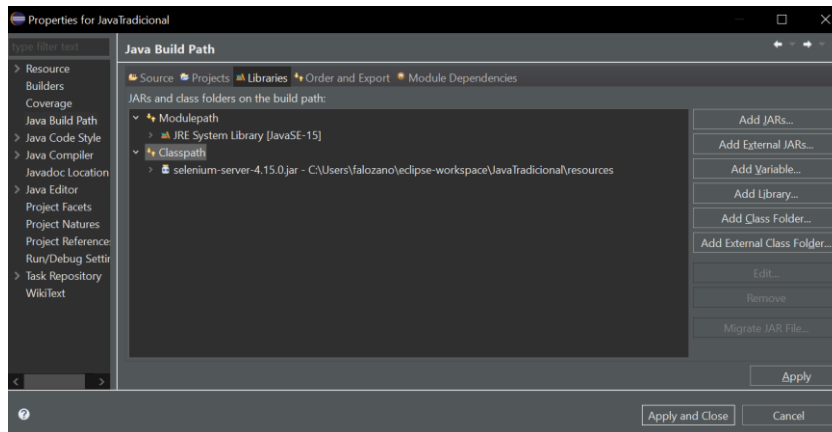
En la ventana que se abre pulsamos sobre la pestaña de Libraries y luego a Add External JARs...



Se busca la librería descargada con anterioridad

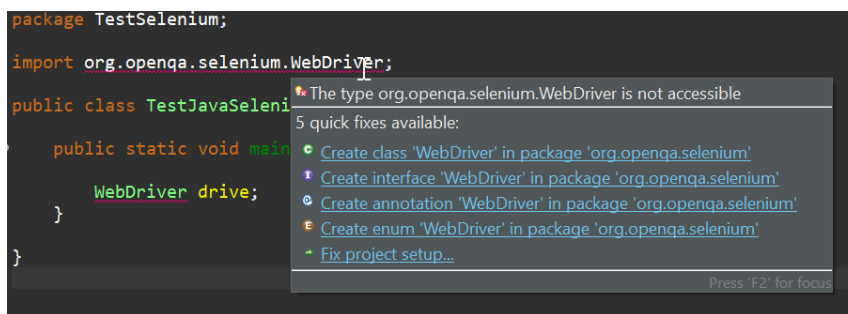


Para finalizar Apply and Close

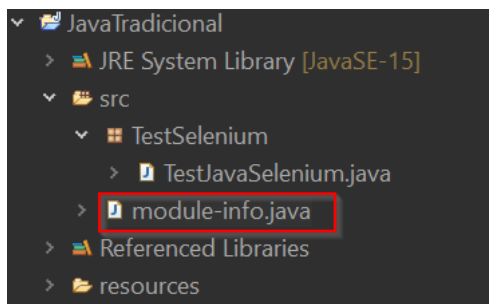


Si se muestra
Classpath, se añade
ahí

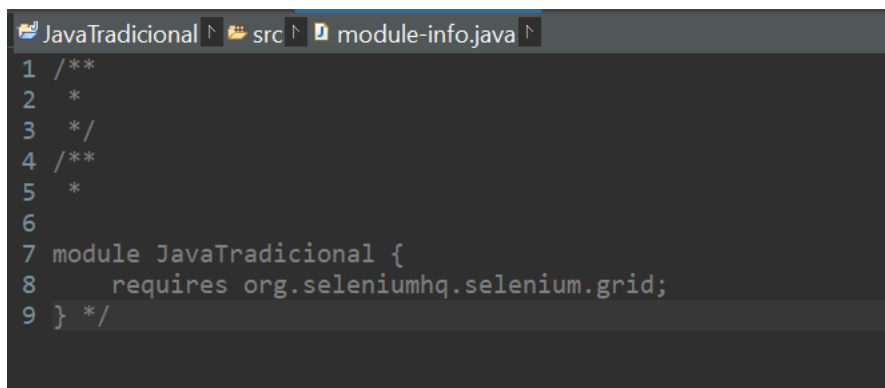
Si al importar la librería te da este error



En module-info.java



Comentamos todas las líneas y guardamos



Descargar driver de firefox

El siguiente paso es descargar el drive de Firefox:













[Releases · mozilla/geckodriver \(github.com\)](https://github.com/mozilla/geckodriver/releases)

Es el que se encargara de controlar el navegador firefox en la ejecución, se descarga desde el repositorio git

Se descarga la ultima version

▼ Assets

12

 geckodriver-v0.33.0-linux-aarch64.tar.gz	2.93 MB	Apr 3
 geckodriver-v0.33.0-linux32.tar.gz	3.02 MB	Apr 3
 geckodriver-v0.33.0-linux32.tar.gz.asc	833 Bytes	Apr 3
 geckodriver-v0.33.0-linux64.tar.gz	2.93 MB	Apr 3
 geckodriver-v0.33.0-linux64.tar.gz.asc	833 Bytes	Apr 3
 geckodriver-v0.33.0-macos-aarch64.tar.gz	1.85 MB	Apr 3
 geckodriver-v0.33.0-macos.tar.gz	2.05 MB	Apr 3
 geckodriver-v0.33.0-win-aarch64.zip	1.47 MB	Apr 3
 geckodriver-v0.33.0-win32.zip	1.54 MB	Apr 3
 geckodriver-v0.33.0-win64.zip	1.59 MB	Apr 3
 Source code (zip)		Apr 3
 Source code (tar.gz)		Apr 3

```
public class TestJavaSelenium {  
    public static void main(String[] args) {  
        WebDriver driver;  
  
        System.setProperty("webdriver.gecko.driver", "resources\\geckodriver.exe");  
        driver= new FirefoxDriver();  
        driver.get("http://www.youtube.com");  
  
        try {  
            Thread.sleep(5000);  
        } catch (InterruptedException e) {  
            // TODO Auto-generated catch block  
            e.printStackTrace();  
        }  
  
        driver.quit();  
    }  
}
```

1. `WebDriver driver;`: Se declara una "variable" llamada `driver` que nos permitirá controlar un navegador web.
2. `System.setProperty("webdriver.gecko.driver", "resources\\geckodriver.exe");`: Configura el programa para usar el "controlador" (geckodriver) de Firefox.
3. `driver = new FirefoxDriver();`: Abre un nuevo navegador Firefox y lo conecta con nuestra variable `driver`.

4. `driver.get("http://www.youtube.com");`: Va a la página web de YouTube.
5. `try { Thread.sleep(5000); } catch (InterruptedException e) { e.printStackTrace(); }`: Espera (no hace nada) durante 5 segundos. Esta parte a menudo se utiliza para asegurarse de que la página se cargue completamente antes de realizar más acciones.
6. `driver.quit();`: Cierra el navegador Firefox.

En resumen, este código automatiza la apertura de Firefox, va a YouTube, espera 5 segundos, y luego cierra el navegador. Es como decirle a un robot: "Abre Firefox, ve a YouTube, espera un poco, y luego cierra Firefox".