Software Testing
Course's Code: CSE 453
Automated Tool support for Testing
[Chapter 5]

Chapter 5

What is a test tool

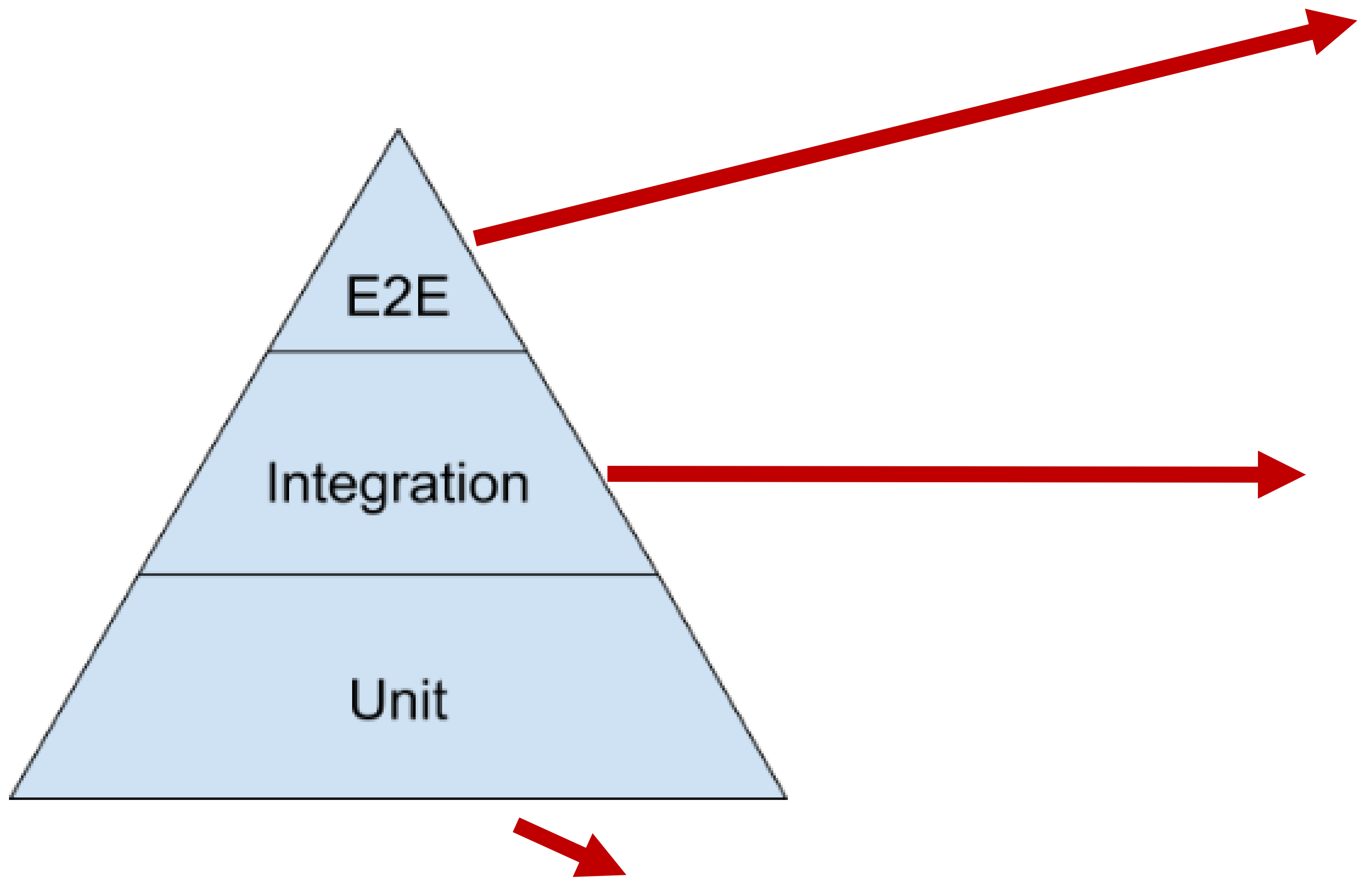
- **A test tool is a software product that supports one or more test activities**, such as planning and control, specification, building initial files and data, test execution and test analysis.

- **Types of Test Tool**
  - Test Management Tools
  - Requirements Management Tool
  - Configuration Management Tool
  - **Test Execution Tool -> Testing with code [In this course]**
  - Test Design Tool
  - Record (or capture playback) tools -> Codeless Testing

Consideration during Automated Testing

- ❑ **Developing software to test the software is called test automation/automated testing**
- ❑ Clear and reasonable expectation should be established in order to know what can and what can not be accomplished with automated testing
- ❑ There should be a clear understanding of the requirements that should be met in order to achieve successful automated testing
- ❑ Select a tool that allows the implementation of automated testing in a way that conforms to the specified long-term testing strategy

Automated Tool Support towards Different levels of testing pyramid



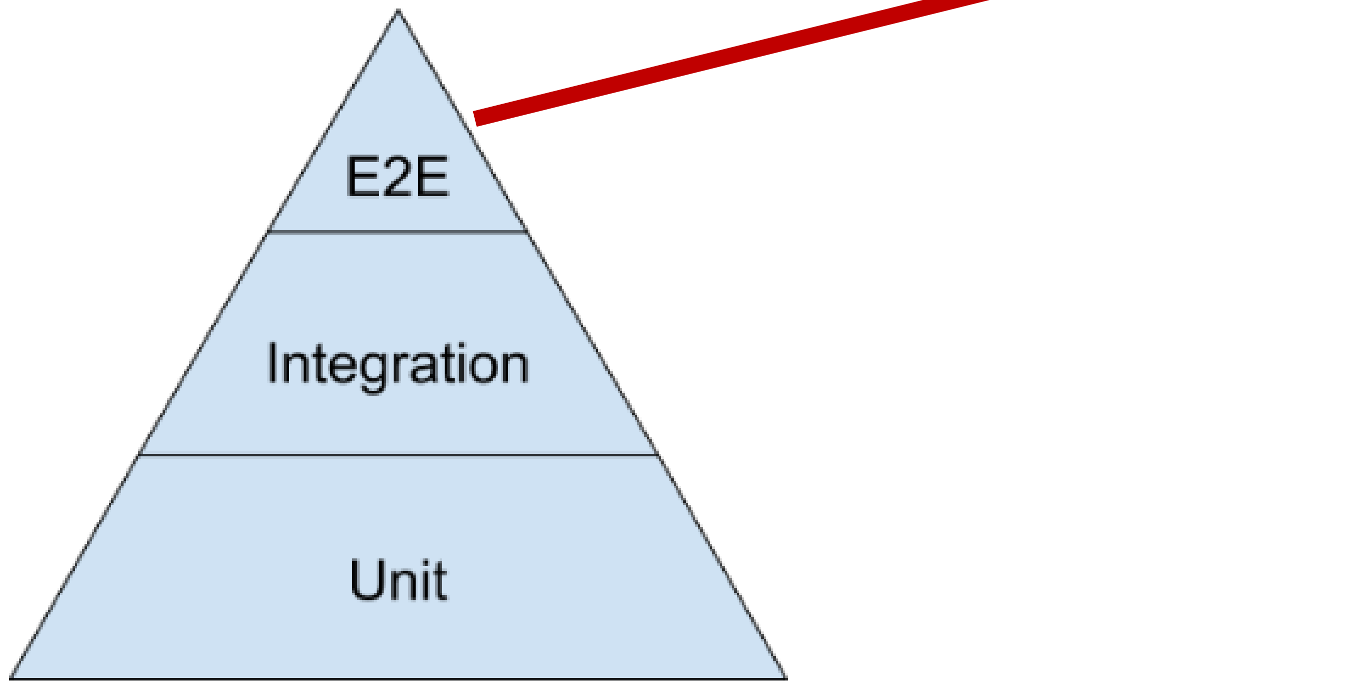
- **Selenium**, Cypress, Playwright, Appium

- **Postman** for API testing tool manually,
- Rest Assured for automated API testing tool, [This also follow BDD]
- Pact for Consumer Driven Contract testing tool,
- **Cucumber** follows BDD [Behavior Driven Development]
- **Mockito** to test interfaces between different components of a software
- WireMocks for API testing

- **Junit**, TestNG, **Mockito** for Java language
- Jest for java script
- Nunit for .net languages
- pytest for python

- **This is not the Complete List**
- **More tools exist**
- **Go for a search**

End to End Test using Selenium (Web Testing)



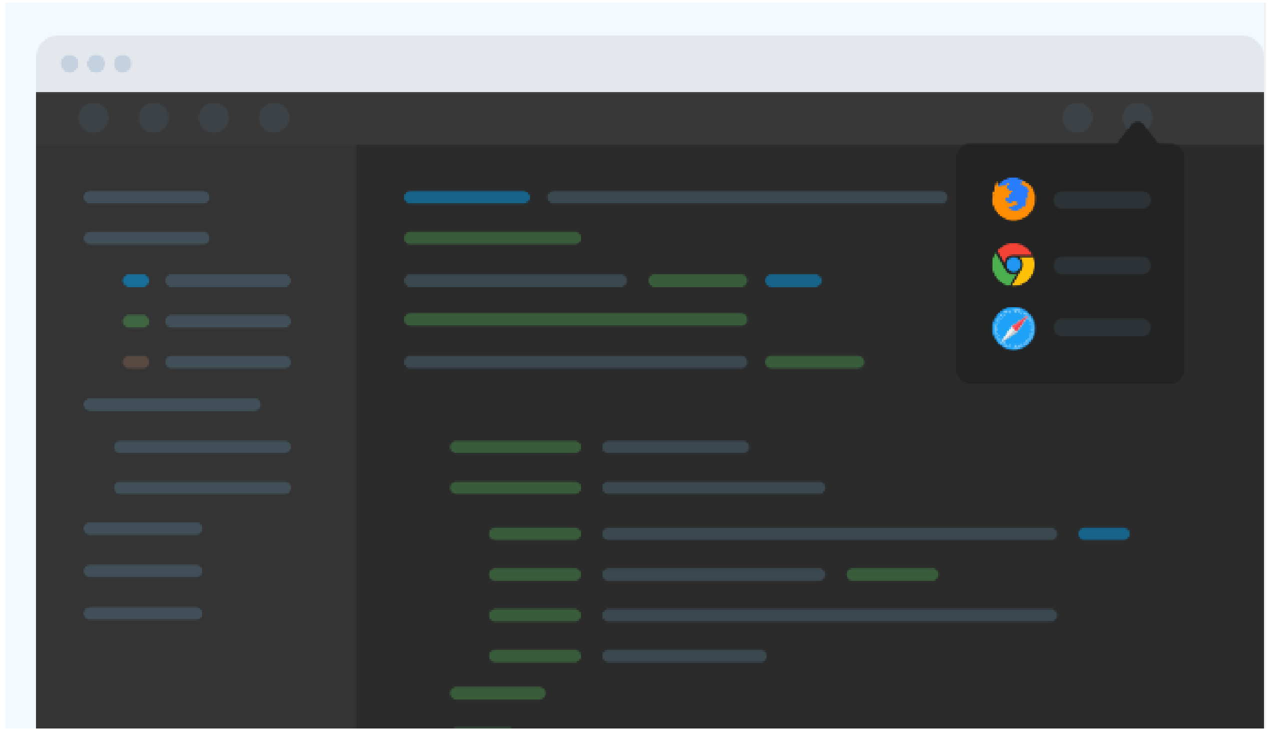
- **Selenium**,
- Cypress,
- Playwright,
- Appium

What is web application testing?

- When a website is deployed, anything can happen**: broken links**, **difficult navigation**, **web security**, and **many other potential risks**. A careful testing process is key to reducing these risks and maintaining web quality.

- Web app testing, or web testing, is a software testing practice that helps ensure the quality and functionalities of the app according to the requirements.

- Web testing must detect all underlying issues, such as *functional discrepancies*, *security breaches*, *integration problems*, *environmental issues*, or *traffic stress* before it is delivered.



Web Application Testing Techniques

Web App Functionality Testing

One of the most common tests for web apps. Functionality testing checks if the initial build works as per its design. It often covers **link testing**, **form validation**, **cookie testing**, **HTML and CSS validation**, and **database connection checkup**.

Web App Interface Testing

Interface testing examines how the web interface responds to emulated interruptions, as well as its compatibility and interaction between different servers. Three key areas to focus on are the **application server**, **web server**, and **database server.**

Web Application Testing Techniques

Web App Compatibility Testing

Compatibility testing checks whether or not the web design is compatible with a variety of browsers and devices. This includes **browser and OS compatibility testing**, along with **mobile browsing and printing options testing**..

Web App Performance Testing

Performance testing is load testing for web apps. Besides the **tests on traffic load**, **stress tests** and **scalability tests** are also crucial to the web performance, especially when it is potentially released to a large audience.

How to Effectively Test a Website Application?

- ❑ **Rigorously carry out cross-browser compatibility testing**
- ❑ **Define and select key parameters for usability tests**
- ❑ **Execute performance tests under various conditions**
- ❑ **Apply tests to all elements, third-party, and extensions of the web app**
- ❑ **Ensure load tests are incrementally performed**
- ❑ **Incorporate exploratory testing into the software development lifecycle**
- ❑ **Involve the development team throughout the testing process**

Introduction to Selenium

- ❑ Primarily **Selenium** is for **automating web applications for testing purposes**. The principle use is **front-end(UI) testing of websites.**
  - ➢ Boring **web-based administration** tasks can (and should) also be automated as well.
  - ➢ This is one kind of **end-to-end testing tool**
- ❑ Selenium **automates web browsers**. It is most famous for enabling rapid, repeatable web-app testing, which allows developers to ship new releases faster and with confidence.
- ❑ Selenium **is an umbrella project for a range of tools** and libraries that enable and support **the automation of web browsers**
- ❑ This is the official website of Selenium : **https://www.selenium.dev/**
- ❑ Selenium is an Open Source project

Introduction to Selenium

- ❑ Selenium supports several programming languages:

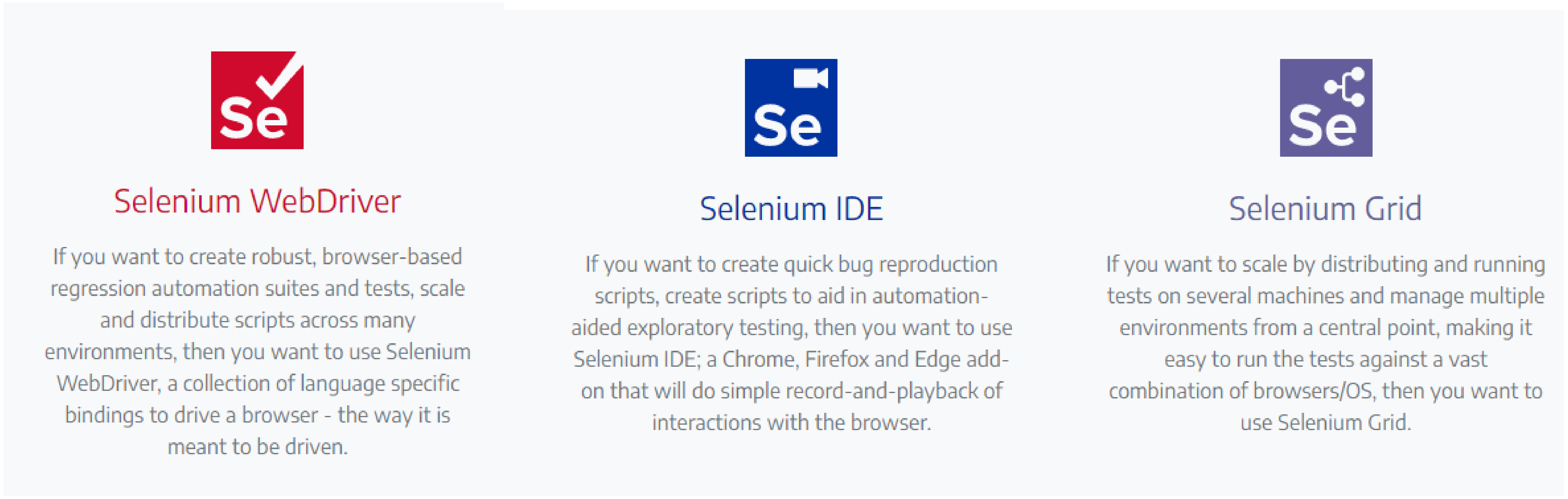
C# Ruby Java Python JavaScript

- ❑ Supports different OS: Windows, Mac, Linux

- ❑ Supports different browsers: Edge, Chrome, Opera, Firefox, Safari, Internet Explorer

Introduction to Selenium

- ❑ Three kinds of products are come under Selenium project



Selenium WebDriver

If you want to create robust, browser-based regression automation suites and tests, scale and distribute scripts across many environments, then you want to use Selenium WebDriver, a collection of language specific bindings to drive a browser - the way it is meant to be driven.

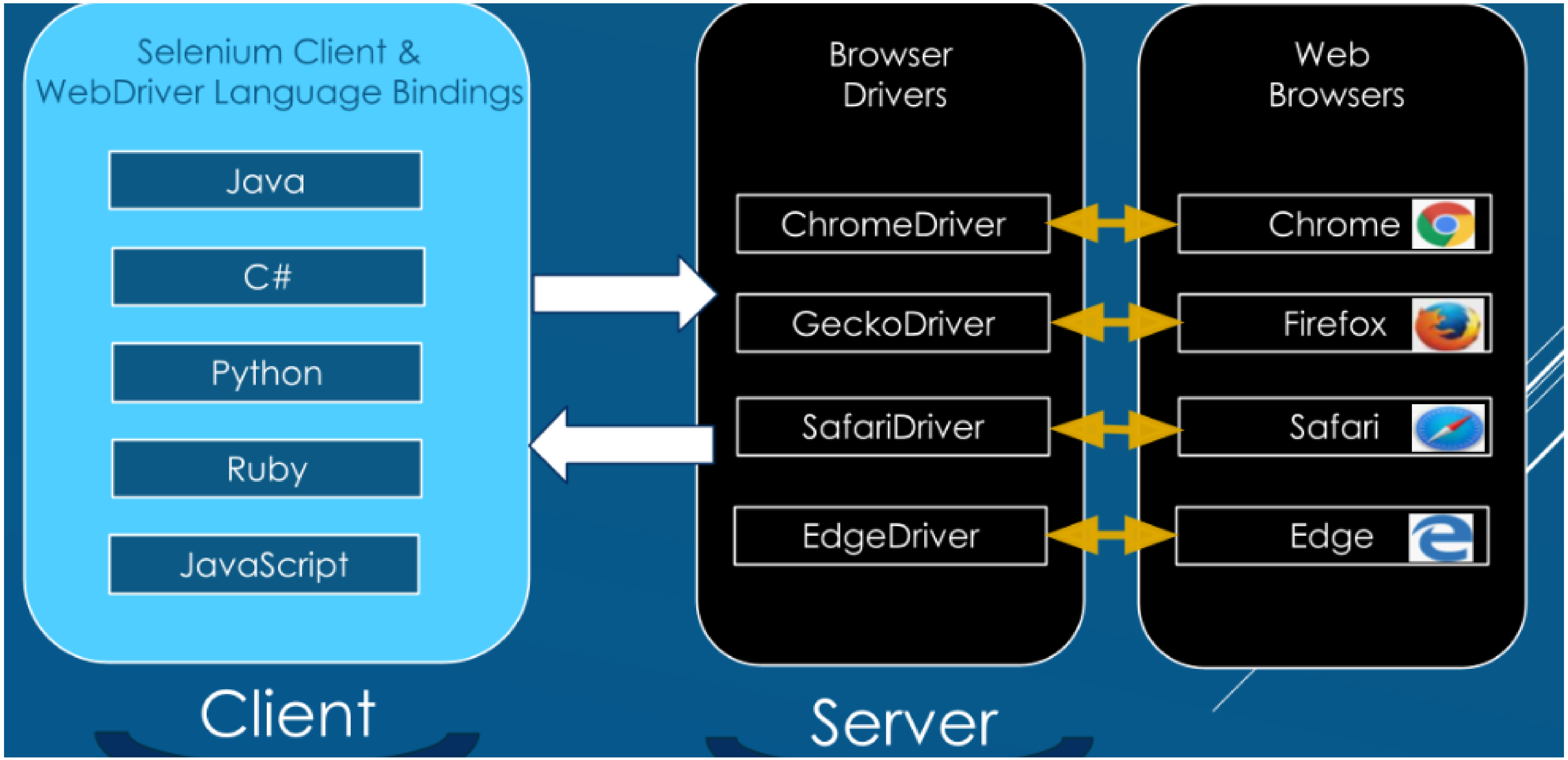
Selenium IDE

If you want to create quick bug reproduction scripts, create scripts to aid in automation-aided exploratory testing, then you want to use Selenium IDE; a Chrome, Firefox and Edge add-on that will do simple record-and-playback of interactions with the browser.

Selenium Grid

If you want to scale by distributing and running tests on several machines and manage multiple environments from a central point, making it easy to run the tests against a vast combination of browsers/OS, then you want to use Selenium Grid.

Architecture of Selenium 4



Architecture of Selenium 4

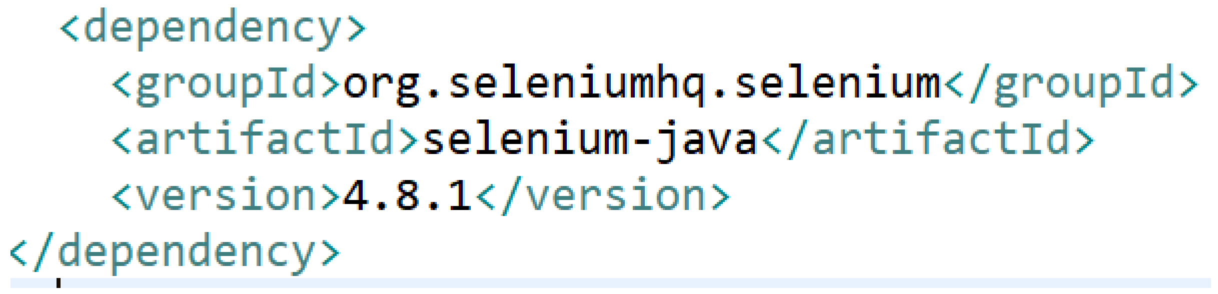
- **The first component has 2 parts combined into one — Selenium Client is a separate part and WebDriver Language Bindings is a different part.**
  - Selenium is an API that has commands for automating our browser.
  - WebDriver talks to the web browser through a browser driver. All the languages have their own bindings. Bindings mean the same commands written for Java is also written for C#, Python, Ruby, and JavaScript.
- **The second component is Browser Drivers, and we see it has 2 functions.**
  - The first function receives a request from Selenium Client and WebDriver Language Bindings, then passes that request to the browser. A driver is also known as a proxy, which has the responsibility for controlling the browser.
  - The second function is to return a response from the browser back to the Selenium Client and WebDriver Language Bindings. All the drivers use a W3C WebDriver Protocol, and most of them are created by the browser vendors.
- **The third component is Web Browsers.**
- This is where all of the Selenium commands are performed.
- Here's the process. The browser receives a performance request and sends back a response to the driver.

How to Set Up Selenium 4

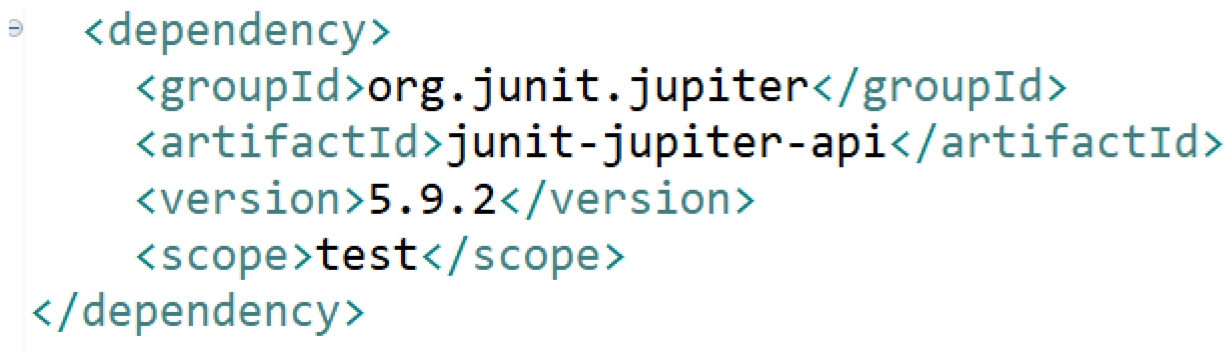
- We need to create a maven project. In this course, We will use
  - Selenium-java : Programming Language to write test scripts
  - Eclipse: IDE to compile and run scripts
  - WebDriverManager: Automated driver management and other helper features for Selenium WebDriver in Java
  - Junit5: Execution of codes in Test Cases

Maven Dependency for Selenium 4 webdriver

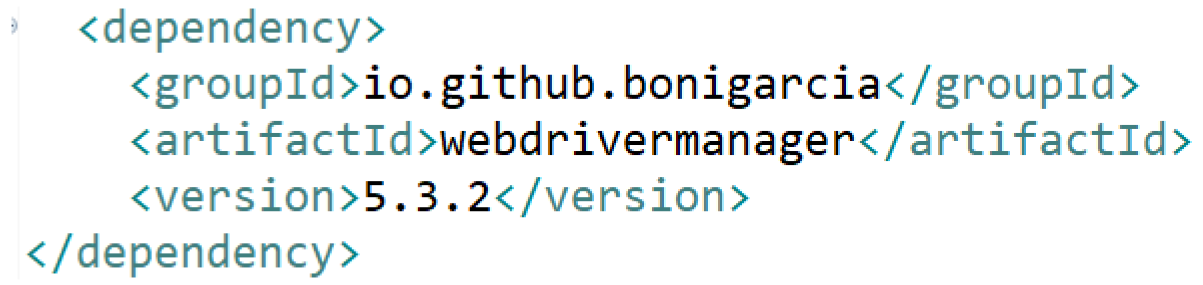
- We will add the Maven Dependencies for Selenium-Java, Web Driver Manager and Junit5



```xml
 <dependency>
   <groupId>org.seleniumhq.selenium</groupId>
   <artifactId>selenium-java</artifactId>
   <version>4.8.1</version>
</dependency>
```



```xml
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <version>5.9.2</version>
    <scope>test</scope>
</dependency>
```



```xml
  <dependency>
    <groupId>io.github.bonigarcia</groupId>
    <artifactId>webdrivermanager</artifactId>
    <version>5.3.2</version>
</dependency>
```

1st Test using Selenium

- Create a JUNIT class under the package of src/test/java folder
- This time check BeforeAll, Before Each, AfterEach method
- Step 1: Create an object of WebDriver class

```
WebDriver driver;
```

- Step 2: Under @BeforeAll, setup the web driver manager. Let's assume, we want to automate Chrome Browser.

```
WebDriverManager.chromedriver().setup();
```

- Step 3: The session will be started. This step will be implemented under @BeforeEach

```
driver=new ChromeDriver();
```

1st Test using Selenium

- Step 4: Take actions on Browser. In this example, we are navigating to a web page. This step also can be implemented under @BeforeEach

```
driver.get("https://www.selenium.dev/selenium/web/web-form.html");
```

- Step 4: In this step, we will also do one more task that is to maximize the window.

```
driver.manage().window().maximize();
```

  - ❑ We can also automate full screen and custom size window of the page

- Step 5: Request browser information. There are a bunch of types of information about the browser you can request, including window handles, browser size / position, cookies, alerts, etc. This step will be implemented under @Test.

```
String title = driver.getTitle();
assertEquals("Web form", title);
```

- In this example, we will test the title of the web page

1st Test using Selenium

- Step 6: Establish Waiting Strategy. Read more about Waiting strategies. This step can also be implemented under @Test.

```
driver.manage().timeouts().implicitlyWait(Duration.ofMillis(500));
```

The **Implicit Wait in Selenium** is used to tell the web driver to wait for a certain amount of time before it throws a "No Such Element Exception". The default setting is 0. Once we set the time, the web driver will wait for the element for that time before throwing an exception.

- Step 7: Find an Element. The majority of commands in most Selenium sessions are element related, and you can't interact with one without first finding an element. This step can also be implemented under @Test.

```
WebElement textBox = driver.findElement(By.name("my-text"));
WebElement submitButton = driver.findElement(By.cssSelector("button"));
```

- In this example, we will find one textbox named as my-text and submit buttion.

1st Test using Selenium

- Step 8: Take action on element. There are only a handful of actions to take on an element, but you will use them frequently. This step can also be implemented under @Test.

```
textBox.sendKeys("Selenium");
submitButton.click();
```

- In this example, “Selenium” is written inside the text box and submit button will be clicked.

- Step 9: Request element information. Elements store a lot of information that can be requested.

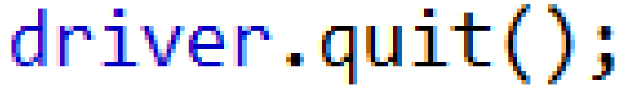
```
WebElement message = driver.findElement(By.id("message"));
String value = message.getText();
assertEquals("Received!", value);
```

- In this example, a web element is found out by id(“message”), then information of that element is requested using getText(). Next the value of the text is tested.

- Step 10: End the session. This ends the driver process, which by default closes the browser as well. No more commands can be sent to this driver instance.

1st Test using Selenium

- **Step 10:** End the session. This ends the driver process, which by default closes the browser as well. No more commands can be sent to this driver instance. This step can be implemented under @AfterEach.

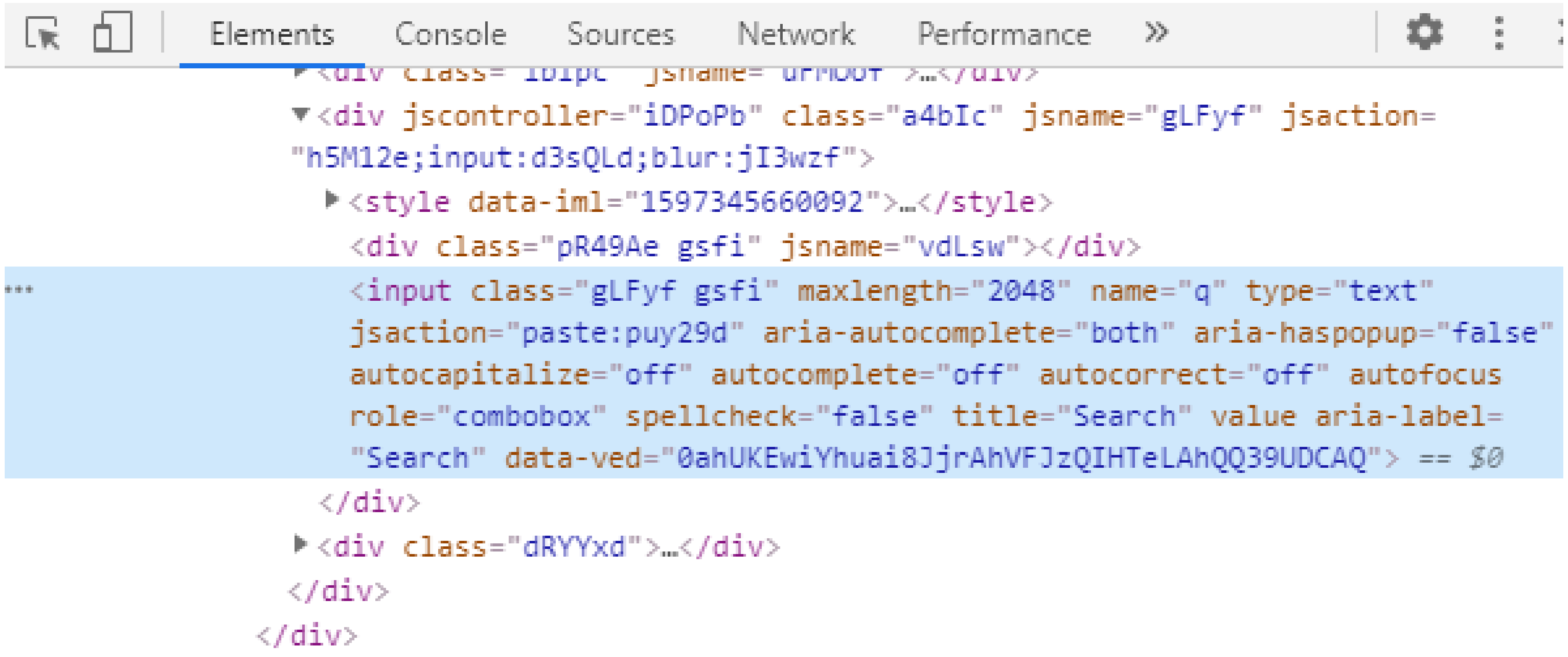


```
driver.quit();
```

- **Step 11:** Run the class as a junit. You can see chrome browser will be started automatically and do all the tasks that you have specified in the program

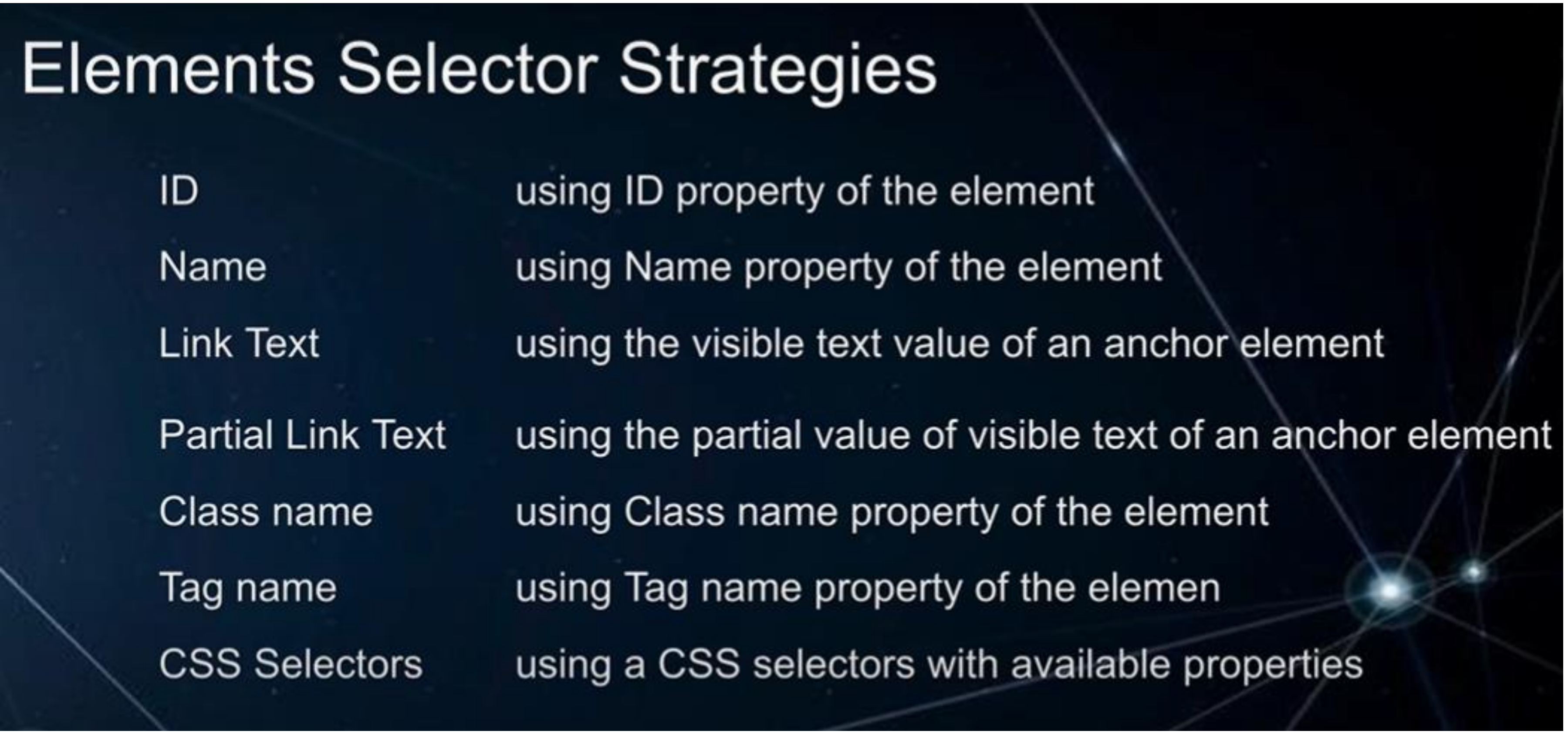
Locators in Selenium

- ➢ Locators can be termed as an address that identifies an web element uniquely on a web page
- ➢ These are the HTML properties of a web element



```
    ▶<div class="ibipc" jsname="urMOOf">…</div>
    ▼<div jscontroller="iDPoPb" class="a4bIc" jsname="gLFyf" jsaction=
    "h5M12e;input:d3sQLd;blur:jI3wzf">
      ▶<style data-iml="1597345660092">…</style>
       <div class="pR49Ae gsfi" jsname="vdLsw"></div>
...    <input class="gLFyf gsfi" maxlength="2048" name="q" type="text"
       jsaction="paste:puy29d" aria-autocomplete="both" aria-haspopup="false"
       autocapitalize="off" autocomplete="off" autocorrect="off" autofocus
       role="combobox" spellcheck="false" title="Search" value aria-label=
       "Search" data-ved="0ahUKEwiYhuai8JjrAhVFJzQIHTeLAhQQ39UDCAQ"> == $0
     </div>
    ▶<div class="dRYYxd">…</div>
    </div>
  </div>
```

Locators in Selenium



Elements Selector Strategies

ID	using ID property of the element
Name	using Name property of the element
Link Text	using the visible text value of an anchor element
Partial Link Text	using the partial value of visible text of an anchor element
Class name	using Class name property of the element
Tag name	using Tag name property of the elemen
CSS Selectors	using a CSS selectors with available properties

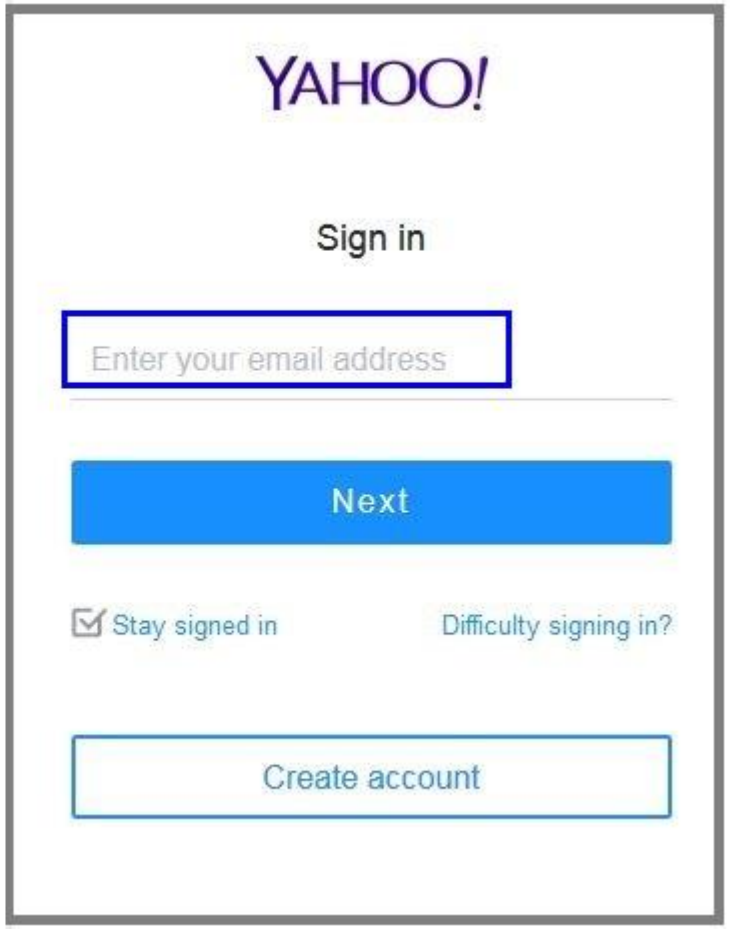
- ➢ There are different ranges of web elements like radio button, text box, id etc.
- ➢ Identifying these element is a tricky approach
- ➢ Selenium uses locators to interact with the web elements on a web page
- ➢ The locator is a "property-value" pair – For example "id" is the property and "email" is the value

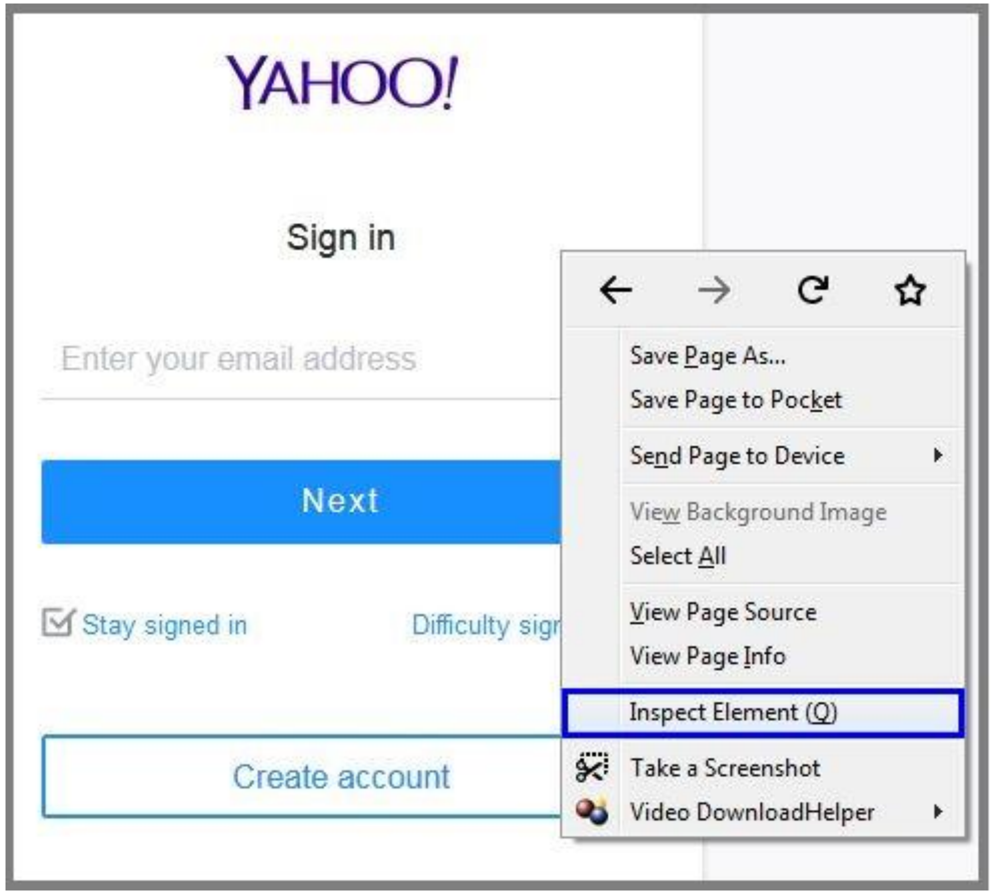
ID Locator in Selenium

- Ids’ are the most preferred way to locate elements on a web page, as each id is supposed to be unique which makes ids faster and more reliable way to locate elements on a page.
- With ID Locator strategy, the first element with the id attribute value matching the location will be returned.
- A **NoSuchElementException** will be raised, if no element has a matching id attribute.

- **Example**: Now, let’s understand the working of ID locator with an example. Here we will launch Google Chrome and navigate to yahoo.com. Here, We will try to locate the email text box using ID Locator.

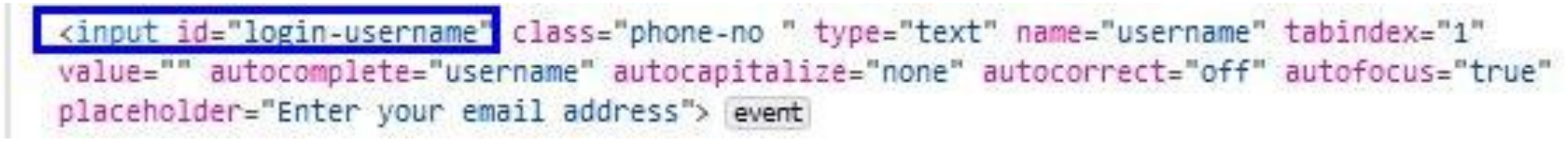
ID Locator in Selenium





- Use the same in your Selenium test script as well:
- driver.findElement(By.id("login-username")).sendKeys("seleniumtesting@yahoo.com"); //id locator for text box

- Right-click on the above web element to inspect the element.



```
<input id="login-username" class="phone-no " type="text" name="username" tabindex="1"
value="" autocomplete="username" autocapitalize="none" autocorrect="off" autofocus="true"
placeholder="Enter your email address"> event
```

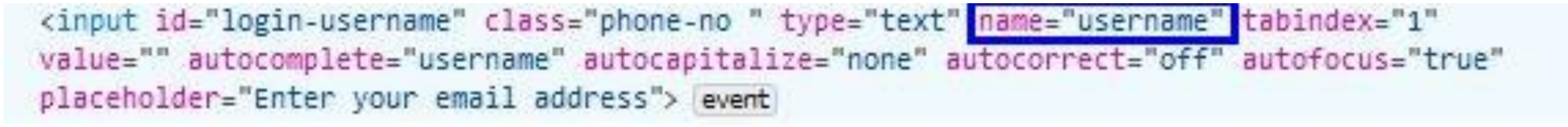
Name Locator in Selenium

- This is also the most efficient way to locate an element with a name attribute. With this strategy, the first element with the value name attribute will be returned.

- A **NoSuchElementException** will be raised, if no element has a matching name attribute.

- **Example:** Now, let's understand the working of the name locator with an example. In the below image you can see, the name locator with a value called username. The difference is that you should use a name instead of id.

Name Locator in Selenium



```
<input id="login-username" class="phone-no " type="text" name="username" tabindex="1"
value="" autocomplete="username" autocapitalize="none" autocorrect="off" autofocus="true"
placeholder="Enter your email address"> event
```

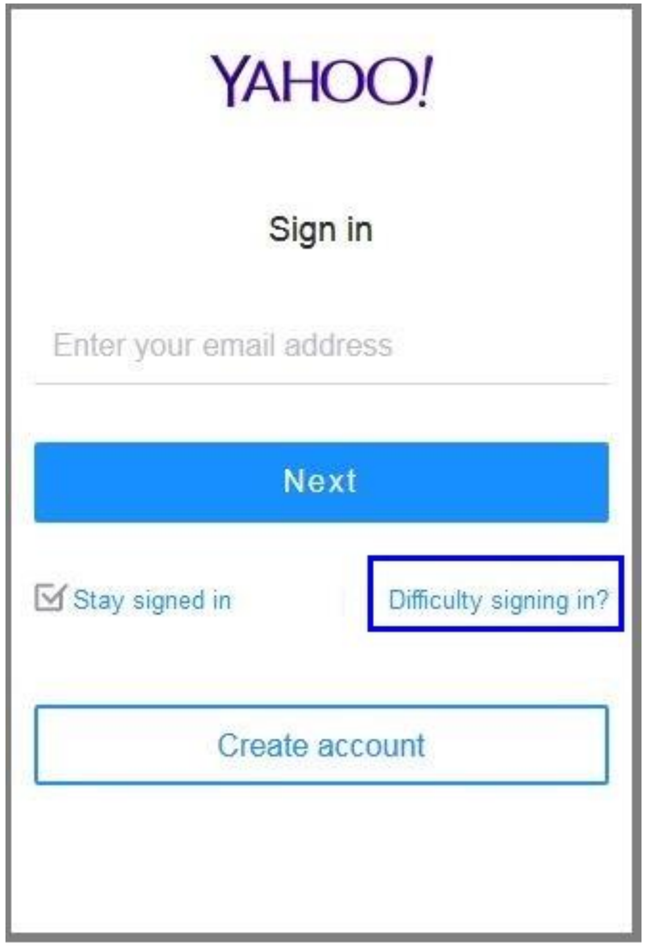
Use the same in your Selenium test script as well:

driver.findElement(By.name("username")).sendKeys("seleniumtesting@yahoo.com");
//name locator for text box

The tagName and className locators are similar to ID and name locators. They are used to select the elements having the html tag or css class.

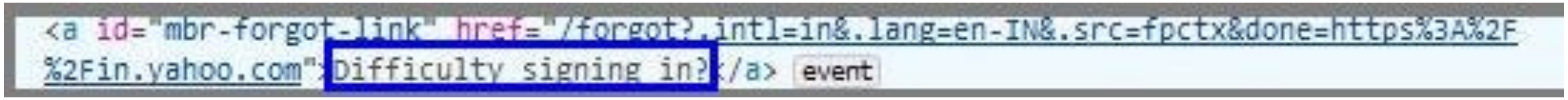
linkText Locator in Selenium

- Selenium linkText locator is used for identifying the hyperlinks on a web page. It can be identified with the help of an anchor tag "a". In order to create the hyperlinks on a web page, you can use anchor tags.
- **Example**: Now, let's understand the working of the linkText locator with an example. Suppose you want to locate 'Difficulty signing in?' link as shown below.



linkText Locator in Selenium

- Right-click on *"Difficulty signing in?"* link for inspecting-you can notice that it starts with an anchor tag, which doesn't have any id and name attributes. In those cases, you can use **linkText** locator.

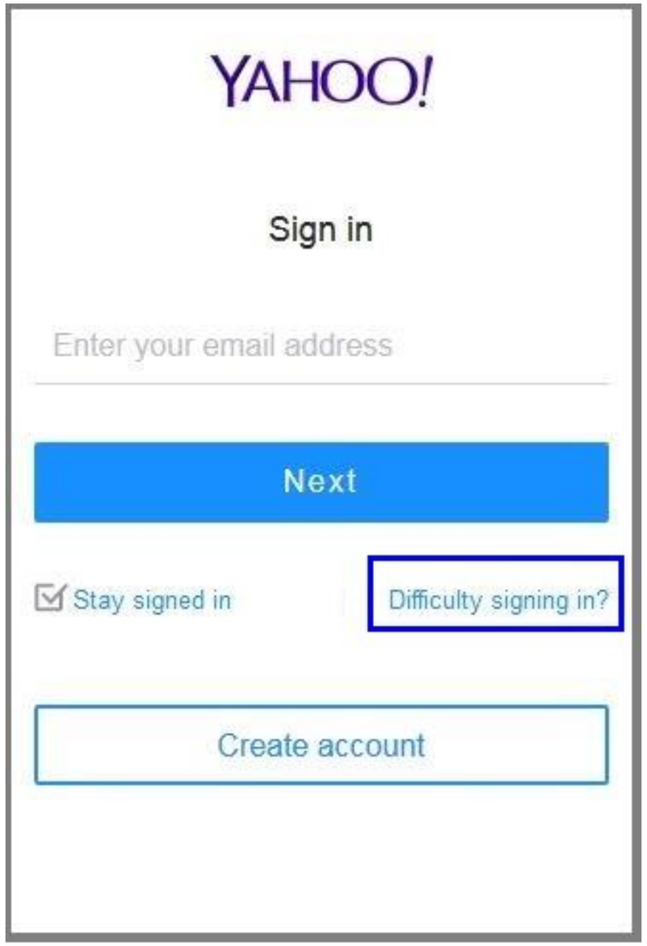


```
<a id="mbr-forgot-link" href="/forgot?.intl=in&.lang=en-IN&.src=fpctx&done=https%3A%2F%2Fin.yahoo.com">Difficulty signing in?</a> event
```

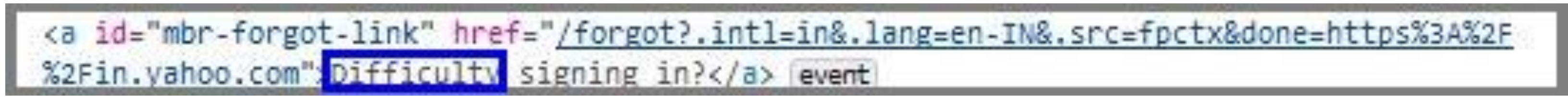
```
driver.findElement(By.linkText("Difficulty signing in?")).click(); //linkText locator for links
```

partialLinkText Locator in Selenium

- In some situations, we may need to find links by a portion of the text in a linkText element. In those situations, we use Partial Link Text to locate elements.
- **Example:** Now, let’s understand the working of PartiallinkText locator with an example. Suppose you want to locate *‘Difficulty signing in?’* link as shown below.



- Right click on *“Difficulty signing in?”* link for inspecting. Now, instead of pasting full text we will give just *Difficulty*.



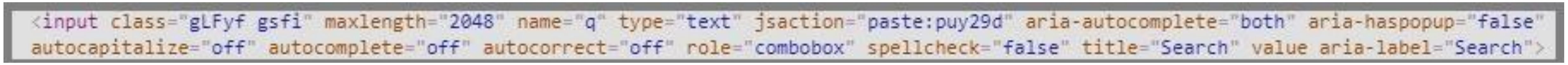
```
<a id="mbr-forgot-link" href="/forgot?.intl=in&.lang=en-IN&.src=fpctx&done=https%3A%2F
%2Fin.yahoo.com">Difficulty signing in?</a> event
```

driver.findElement(By.partiallinkText("Difficulty")).click();
//partiallinkText locator for links

XPath Locator in Selenium

- XPath also called as XML Path is a language to query XML documents. **XPath** is an important strategy to locate elements in selenium.

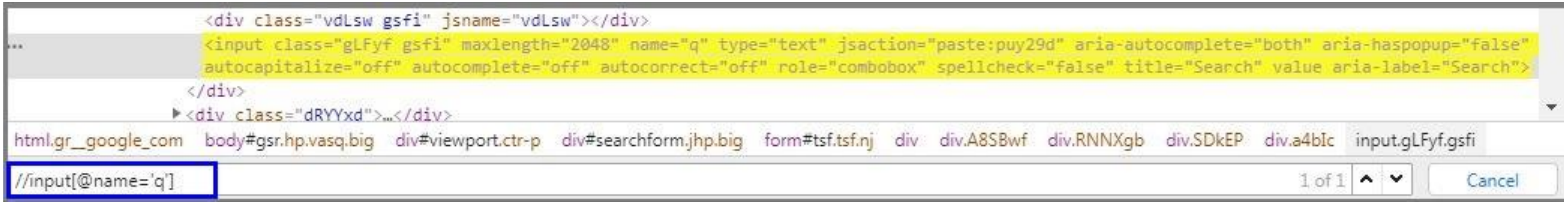
- **Example:** Now, let’s understand the working of XPath locator with an example. Suppose you want to locate the search box using XPath of **google.com**. On inspecting the web element you can see an input tag and attributes like class and name. To construct XPath we make use of tag names and attributes to locate the search bar.



```
<input class="gLFyf gsfi" maxlength="2048" name="q" type="text" jsaction="paste:puy29d" aria-autocomplete="both" aria-haspopup="false"
autocapitalize="off" autocomplete="off" autocorrect="off" role="combobox" spellcheck="false" title="Search" value aria-label="Search">
```

XPath Locator in Selenium

- Click the Elements tab and press Ctrl + F to open a search box in Chrome's developer's tool. In the above image, you can see an input tag.
- Now, I will start with **//**input, where //input implies a tag name. Here we make use of *name* attribute and pass 'q' in single quotes as its value.
- This will give the below XPath expression:
    - **//input[@name='q']**



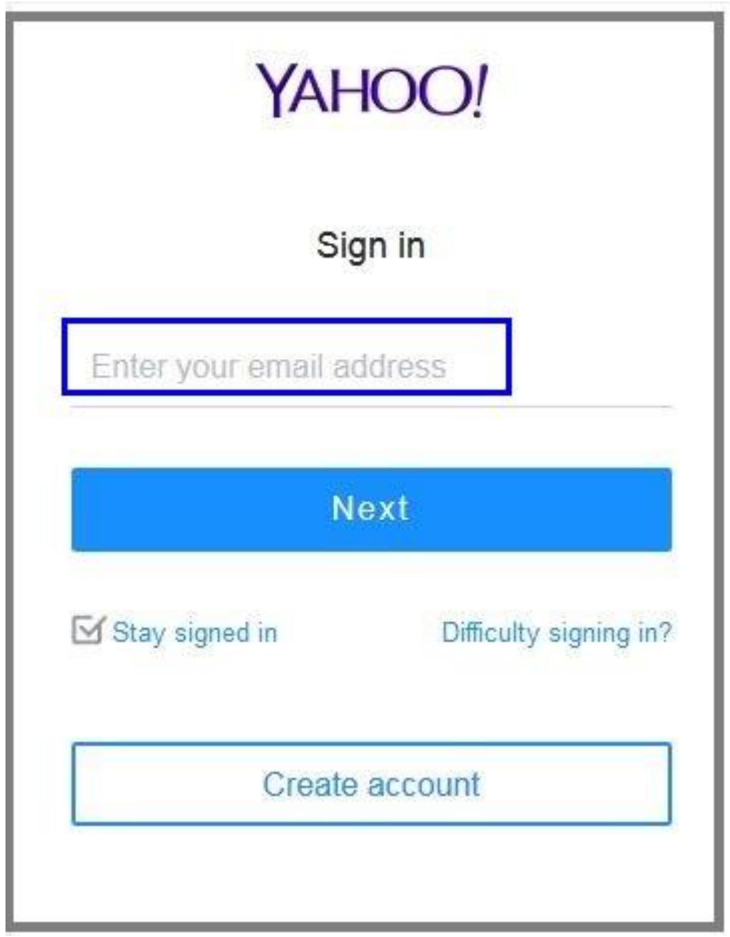
```
<div class="vdLsw gsfi" jsname="vdLsw"></div>
<input class="gLFyf gsfi" maxlength="2048" name="q" type="text" jsaction="paste:puy29d" aria-autocomplete="both" aria-haspopup="false" autocapitalize="off" autocomplete="off" autocorrect="off" role="combobox" spellcheck="false" title="Search" value aria-label="Search">
</div>
<div class="dRYYxd">…</div>
```

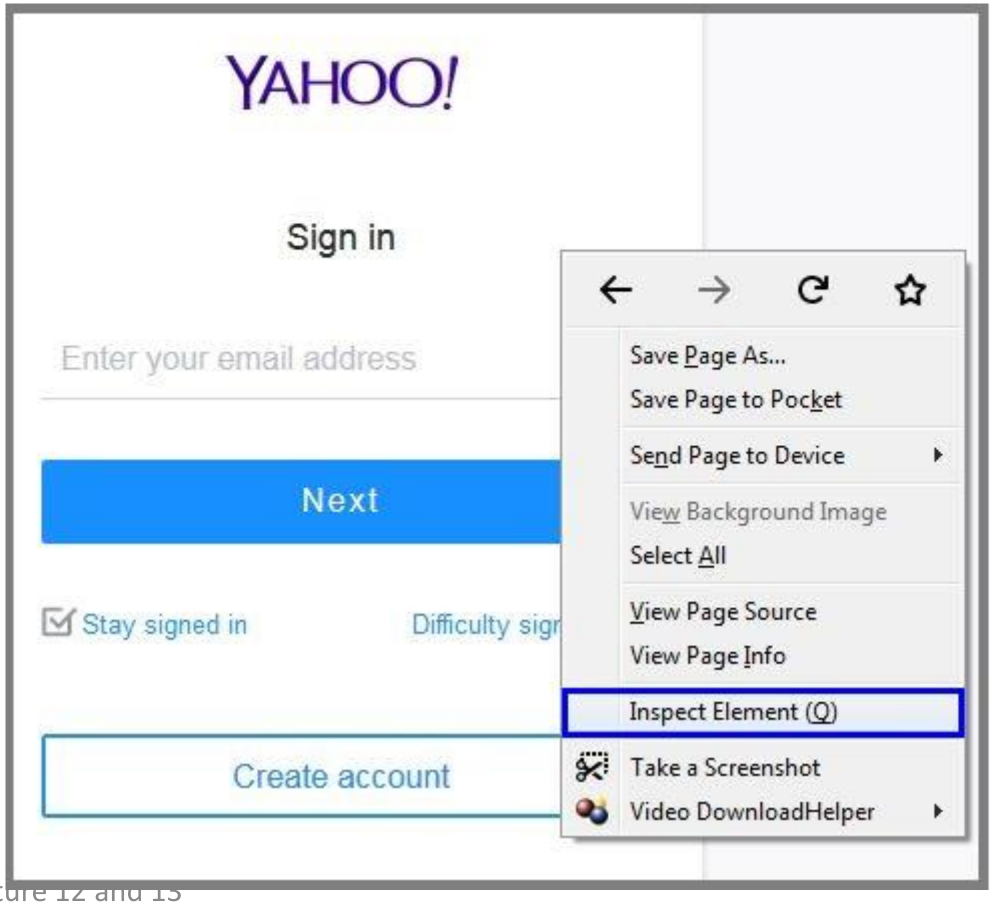
html.gr__google_com body#gsr.hp.vasq.big div#viewport.ctr-p div#searchform.jhp.big form#tsf.tsf.nj div div.A8SBwf div.RNNXgb div.SDkEP div.a4bIc input.gLFyf.gsfi
//input[@name='q']
1 of 1
Cancel

driver.findElement(By.xpath("//input[@name='q']")).sendKeys("Selenium Webdriver"); //xpath for search box

CSS Selectors in Selenium

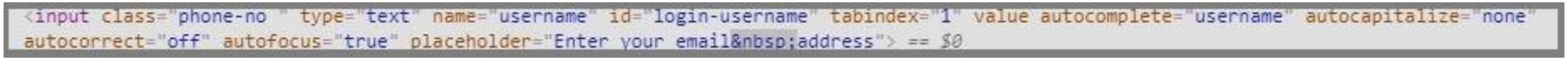
- CSS Selector is the combination of an element selector and a selector value which identifies web element within a web page. Like Xpath, CSS Selector is also used to locate web elements having no ID, class or Name.
- **Example:** Now, let's understand the working of CSS Selector with an example. Here we will launch Google Chrome and navigate to *yahoo.com*. Here, I will try to inspect the *email text box* using **CSS Selector.**





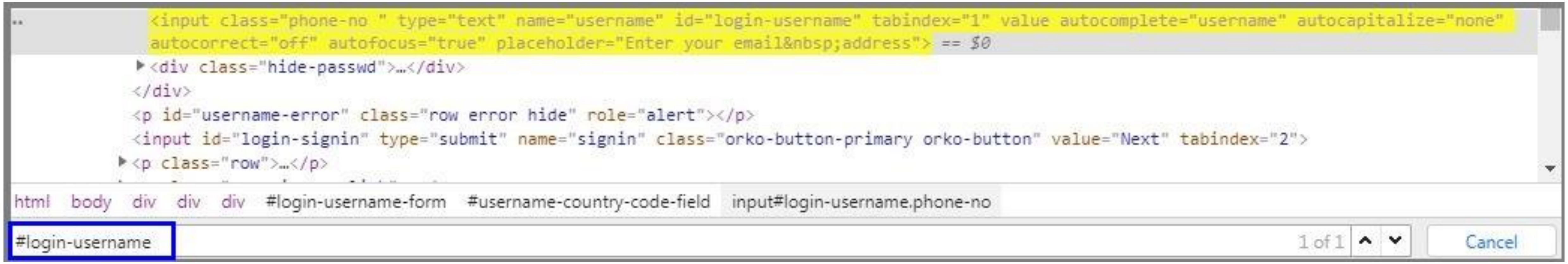
CSS Selectors in Selenium

- Once inspecting the element, we can see the image as below



```
<input class="phone-no " type="text" name="username" id="login-username" tabindex="1" value autocomplete="username" autocapitalize="none"
autocorrect="off" autofocus="true" placeholder="Enter your email&nbsp;address"> == $0
```

- Click Elements tab and press Ctrl + F to open a search box in Chrome's developer's tool. In this case, CSS Selector will start with **#** and on giving the value of *id* attribute as *login-username*, the element gets highlighted.



```
<input class="phone-no " type="text" name="username" id="login-username" tabindex="1" value autocomplete="username" autocapitalize="none"
autocorrect="off" autofocus="true" placeholder="Enter your email&nbsp;address"> == $0
  ▶<div class="hide-passwd">…</div>
</div>
<p id="username-error" class="row error hide" role="alert"></p>
<input id="login-signin" type="submit" name="signin" class="orko-button-primary orko-button" value="Next" tabindex="2">
▶<p class="row">…</p>
```

html body div div div #login-username-form #username-country-code-field input#login-username.phone-no

#login-username 1 of 1 Cancel

driver.findElement(By.cssSelector("#login-username")).sendKeys("seleniumtesting@yahoo.com");

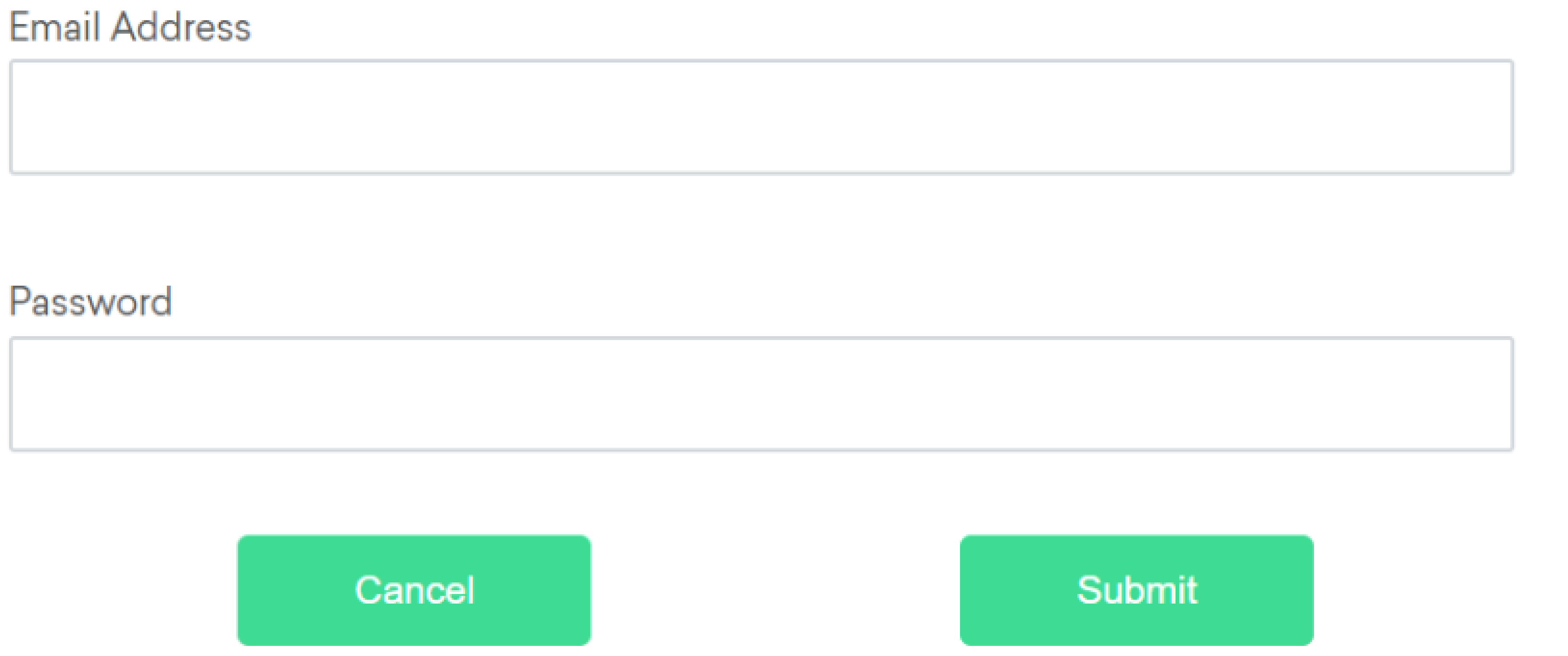
CSS Selectors in Selenium

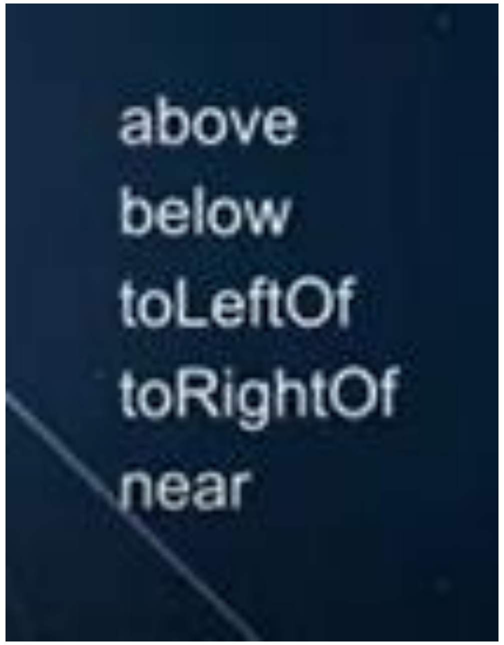
Other Selectors		By.cssSelector
By.className("register")	=>	".register"
By.tagName("table")	=>	"table"
By.id("unique_id")	=>	"#unique_id"
By.name("login")	=>	"[name=login]"
By.xpath("//body/nav")	=>	"body > nav"
By.xpath("//body//nav")	=>	"body nav"

Selenium 4: Relative Locators

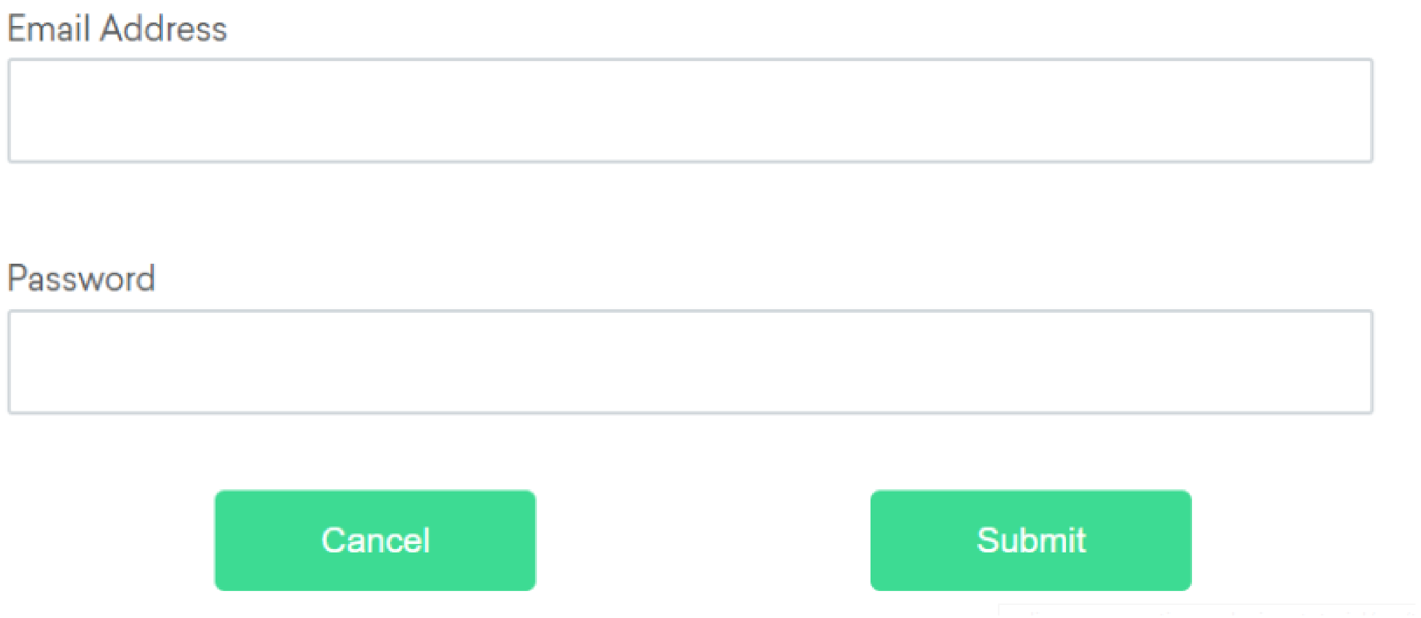
https://www.selenium.dev/documentation/webdriver/elements/locators/

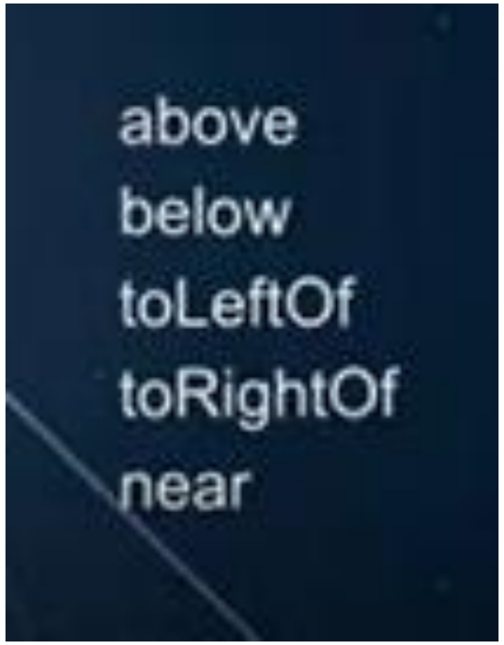
These locators are helpful when it is not easy to construct a locator for the desired element, but easy to describe spatially where the element is in relation to an element that does have an easily constructed locator.




above
below
toLeftOf
toRightOf
near

Selenium 4: Relative Locators




above
below
toLeftOf
toRightOf
near

Above
If the email text field element is not easily identifiable for some reason, but the password text field element is, we can locate the text field element using the fact that it is an "input" element "above" the password element.

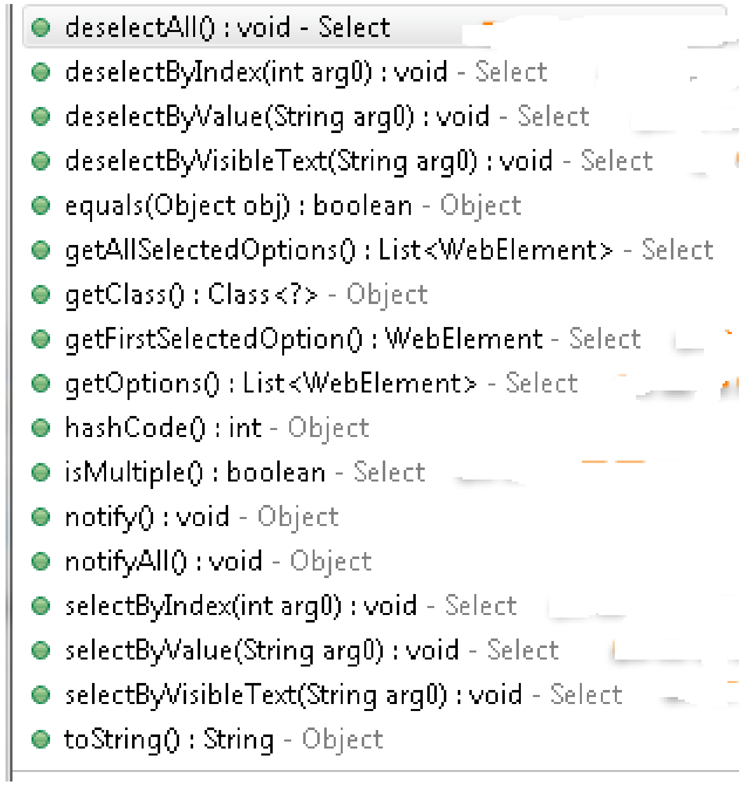
```
By emailLocator = RelativeLocator.with(By.tagName("input")).above(By.id("password"));
```

Select Class in Selenium – How to handle Dropdown

- Select class is a WebDriver class provides the implementation of the HTML Select tag
- Syntax: Select oselect=new Select(findElementBy.id("id"));

  oselect.

https://www.selenium.dev/documentation/webdriver/elements/select_lists/



```
deselectAll() : void - Select
deselectByIndex(int arg0) : void - Select
deselectByValue(String arg0) : void - Select
deselectByVisibleText(String arg0) : void - Select
equals(Object obj) : boolean - Object
getAllSelectedOptions() : List<WebElement> - Select
getClass() : Class<?> - Object
getFirstSelectedOption() : WebElement - Select
getOptions() : List<WebElement> - Select
hashCode() : int - Object
isMultiple() : boolean - Select
notify() : void - Object
notifyAll() : void - Object
selectByIndex(int arg0) : void - Select
selectByValue(String arg0) : void - Select
selectByVisibleText(String arg0) : void - Select
toString() : String - Object
```

Select Class in Selenium – How to handle Dropdown

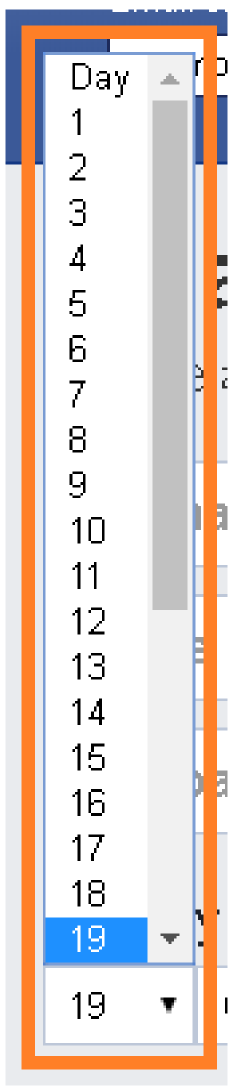
- The select class has several methods to select or de-select dropdown's values -
  - **selectByVisibleText("parameter")**
  - **deselectByVisibleText("parameter")**
  - **selectByIndex(index)**
  - **deselectByIndex(index)**
  - **selectByValue("value")**
  - **delectByValue("value")**

- The select class provides one method to check whether drop-down allows a user to select multiple values -
  - **isMultiple() -** returns **true** if dropdown allows **multiple** selection, otherwise **false**.

- To deselect multiple selected values - **deselectAll()**

Select Class in Selenium – How to handle Dropdown

- **parameter -** represents the value of dropdown.
- **index -** represents the index of the dropdown's value. It starts with **0**, i.e., the index of the first value of dropdown is **zero**.
- **value -** represents the value of "**value**" property.
    - **How to access?**
        - First, identify the dropdown field on the webpage.
        - Then we can select or unselect values from it.



Select Class in Selenium – How to handle Dropdown

•**Code** -

selectByVisibleText() -

```
WebElement daytext = driver.findElement(By.id("day"));//identifing the 'day'
dropdown available on facebook login page.
Select select = new Select(daytext);//instantiating the dropdown elements as Select
class object.
select.selectByVisibleText("10");//selecting value from dropdown.
```

•**Explanation** -
"**day**" is the drop-down element having day values from (**1-30/31**).
•"**select**" is the object of **the Select** class that refers to "**day**" dropdown's values.
•We are selecting **"10" as the text parameter.**

Select Class in Selenium – How to handle Dropdown

selectByIndex() -

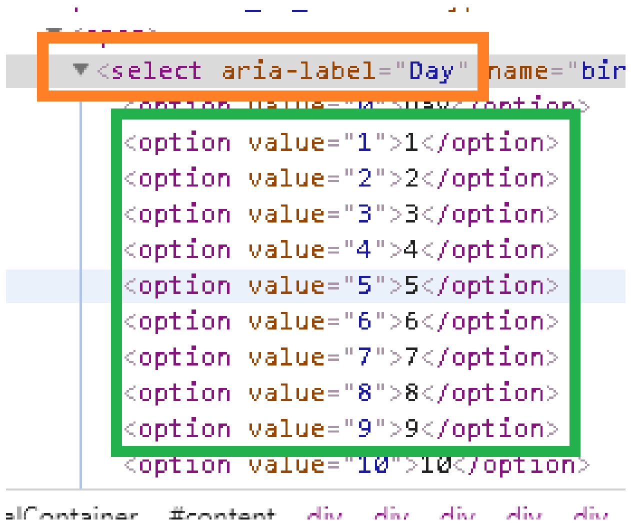
```
WebElement dayIndex = driver.findElement(By.id("day"));//identifing the 'day'
dropdown available on facebook login page.
Select select = new Select(dayIndex);//instantiating the dropdown elements as
Select class object. select.selectByIndex(11);//selecting value from dropdown.
```

•**Explanation** -
"**day**" is the drop-down element having day values from (**1-30/31**).
•**"select"** is the object of **the Select** class that refers to "**day**" dropdown's values.
•We are selecting "11" as the index value. It will select "11" as the date in the drop-down.

Select Class in Selenium – How to handle Dropdown

selectByValue() -

Here, I inspected "**day**" drop-down using developer tool. In the image you can see **options** available for the dropdown. You can see "**value**" property and its **value** beside <**option**> tag. In the below code, we will use this **value**.



```
▼<select aria-label="Day" name="bir
    <option value="1">1</option>
    <option value="2">2</option>
    <option value="3">3</option>
    <option value="4">4</option>
    <option value="5">5</option>
    <option value="6">6</option>
    <option value="7">7</option>
    <option value="8">8</option>
    <option value="9">9</option>
    <option value="10">10</option>
```

WebElement dayValue = driver.findElement(By.id("day")); //identifing the 'day' dropdown available on facebook login page.
Select select = new Select(dayValue); //instantiating the dropdown elements as Select class object.
select.selectByValue("11"); //selecting value from dropdown

- **Explanation -**
  "**day**" is the drop-down element having day values from (**1-30/31**).
- "**select**" is the object of **the Select** class that refers to "**day**" dropdown's values.
- We are selecting **"11" as the value.** It will select "**11**" as the date in the drop-down..

Select Class in Selenium – How to handle Dropdown

getOptions() -

This method helps to get all the options belonging to a select tag. It takes no parameters and returns a list of web elements

```
Select oselect= driver.findElement(By.id("day")); //identifing the 'day' dropdown available on facebook
List<WebElement> elementcount= oselect.getOptions();
System.out.println(elementcount.size());
```

Select Class in Selenium – How to handle Dropdown

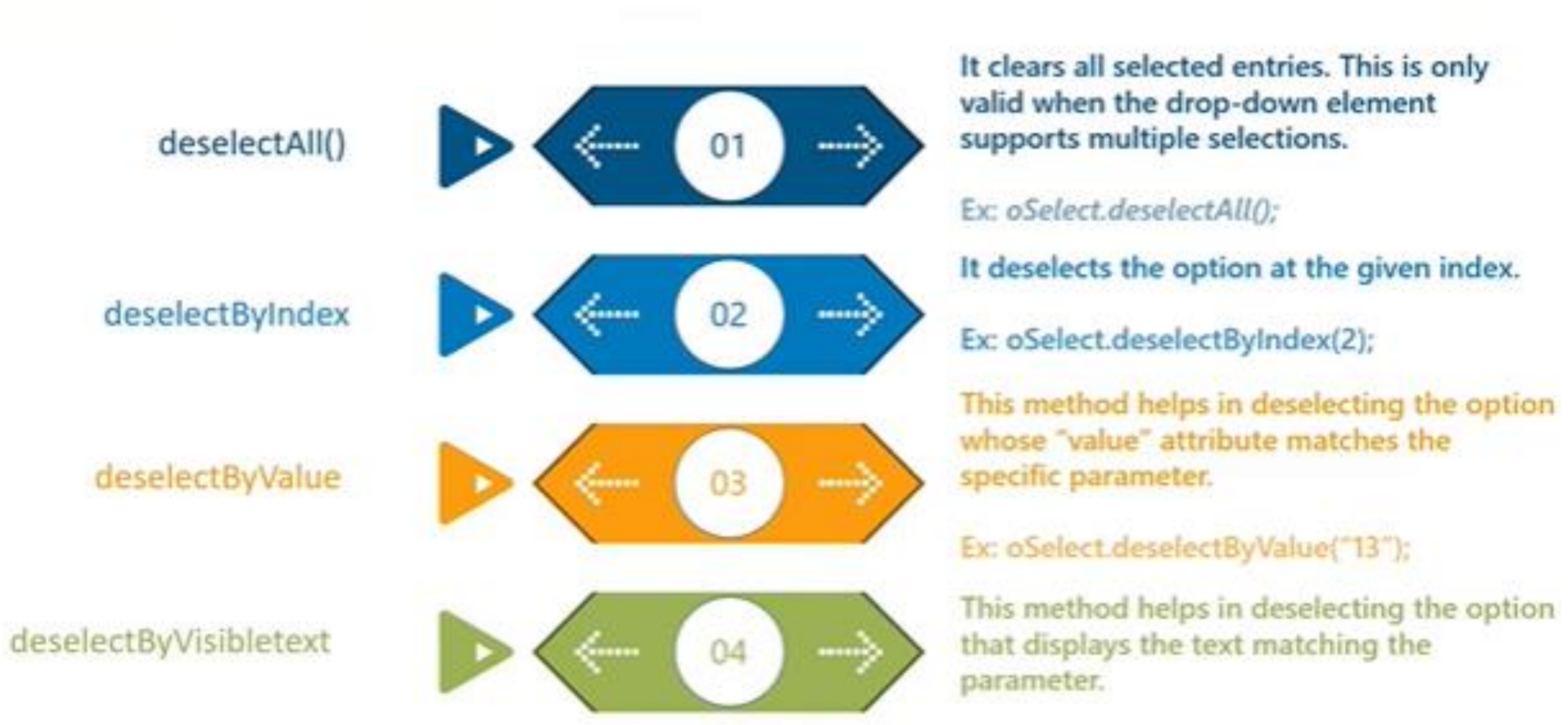
- Multiple Select attribute is a Boolean expression. When it is present, it specifies that multiple options can be selected at once

isMultiple() -

- This method tells whether the select element support multiple select options at the same time or not. It takes no parameters and returns Boolean value

oSelect.isMultiple() -

Select Class in Selenium – How to handle Dropdown



deselectAll()

It clears all selected entries. This is only valid when the drop-down element supports multiple selections.

Ex: *oSelect.deselectAll();*

deselectByIndex

It deselects the option at the given index.

Ex: oSelect.deselectByIndex(2);

deselectByValue

This method helps in deselecting the option whose "value" attribute matches the specific parameter.

Ex: oSelect.deselectByValue("13");

deselectByVisibletext

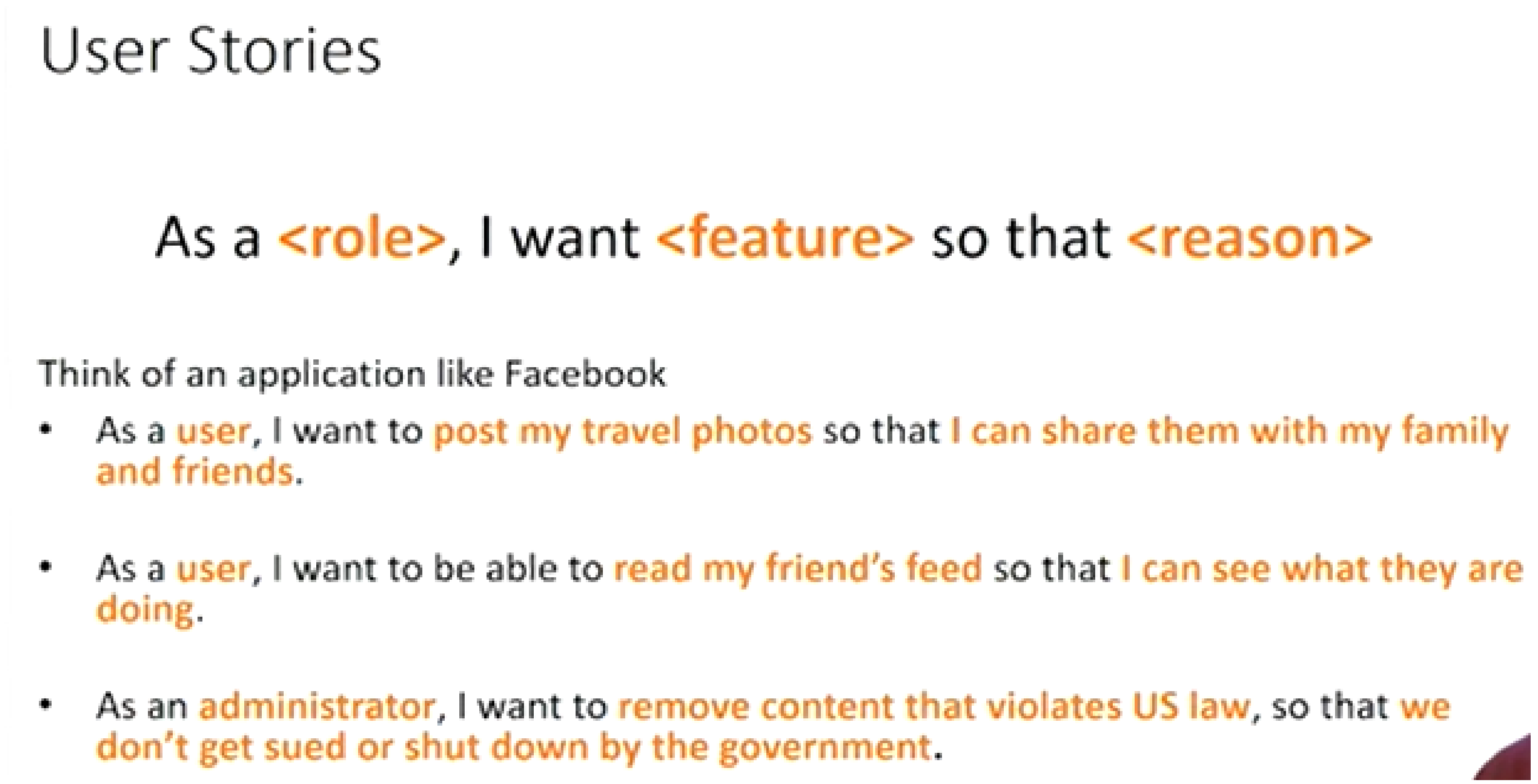
This method helps in deselecting the option that displays the text matching the parameter.

Chapter 5

Chapter 5. Automated Tool support for Testing

User Stories and Behaviour Driven Development (BDD)

- **Requirements need to be clear, precise, and unambiguous**
    - **To make that we need to develop test cases for each requirement**



User Stories

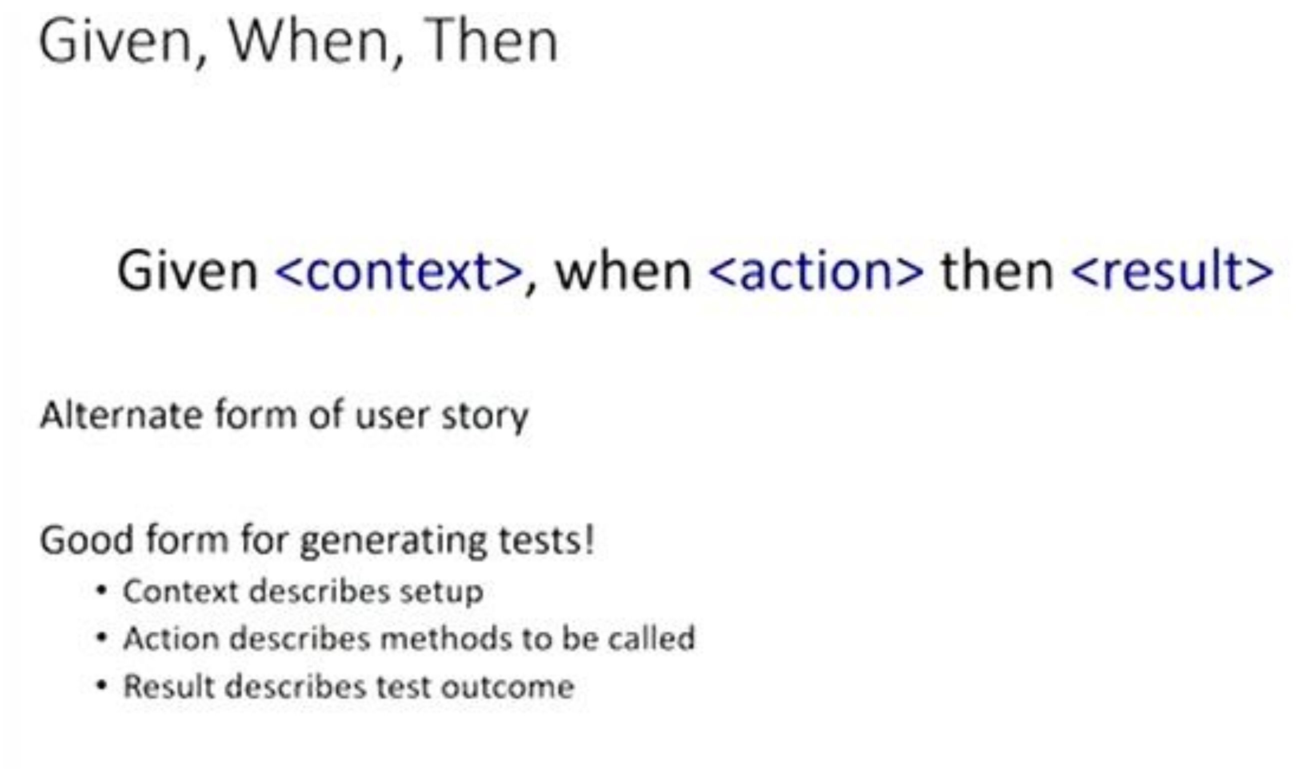
As a <role>, I want <feature> so that <reason>

Think of an application like Facebook

- As a user, I want to post my travel photos so that I can share them with my family and friends.
- As a user, I want to be able to read my friend's feed so that I can see what they are doing.
- As an administrator, I want to remove content that violates US law, so that we don't get sued or shut down by the government.

What is Behaviour Driven Development (BDD)?

- BDD is defined as a process for software teams to work in a collaborative fashion and to bridge the gap between business and the technical people with respect to requirements, misunderstandings, and assumptions that might impact the delivery piece.



Given, When, Then

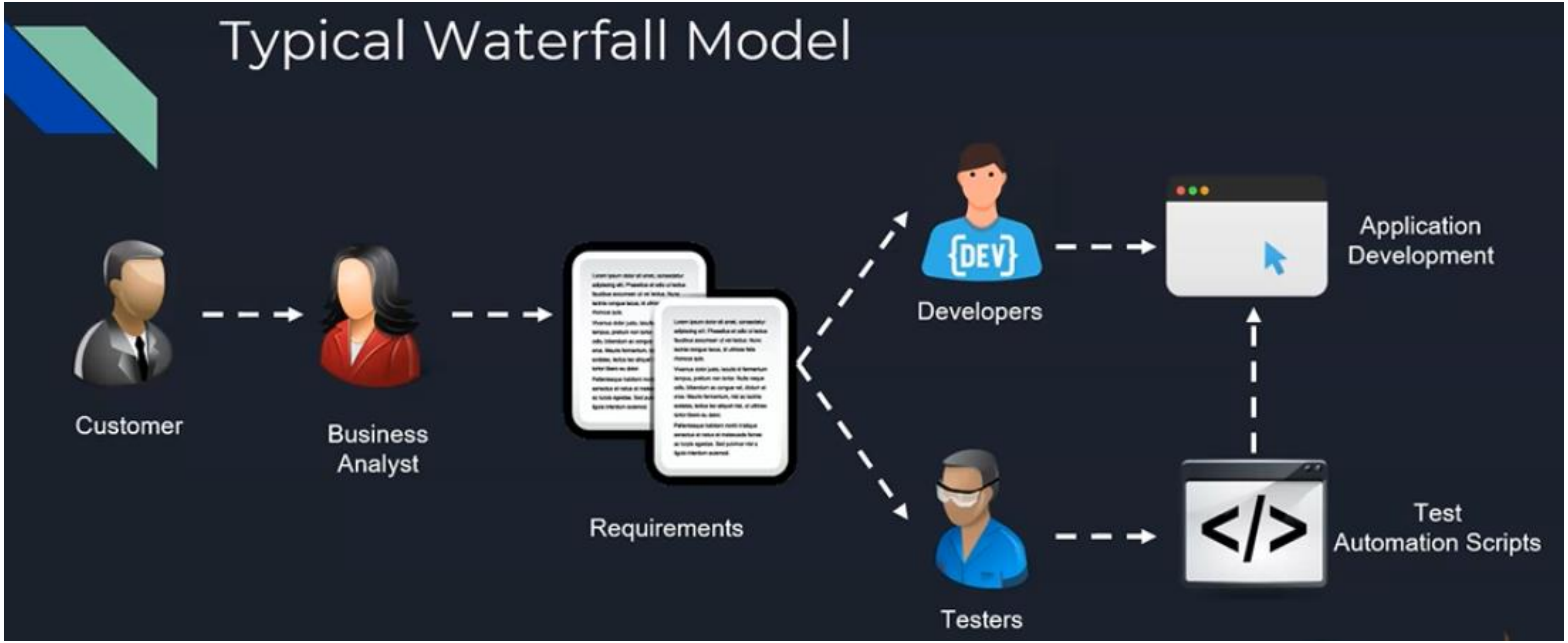
Given <context>, when <action> then <result>

Alternate form of user story

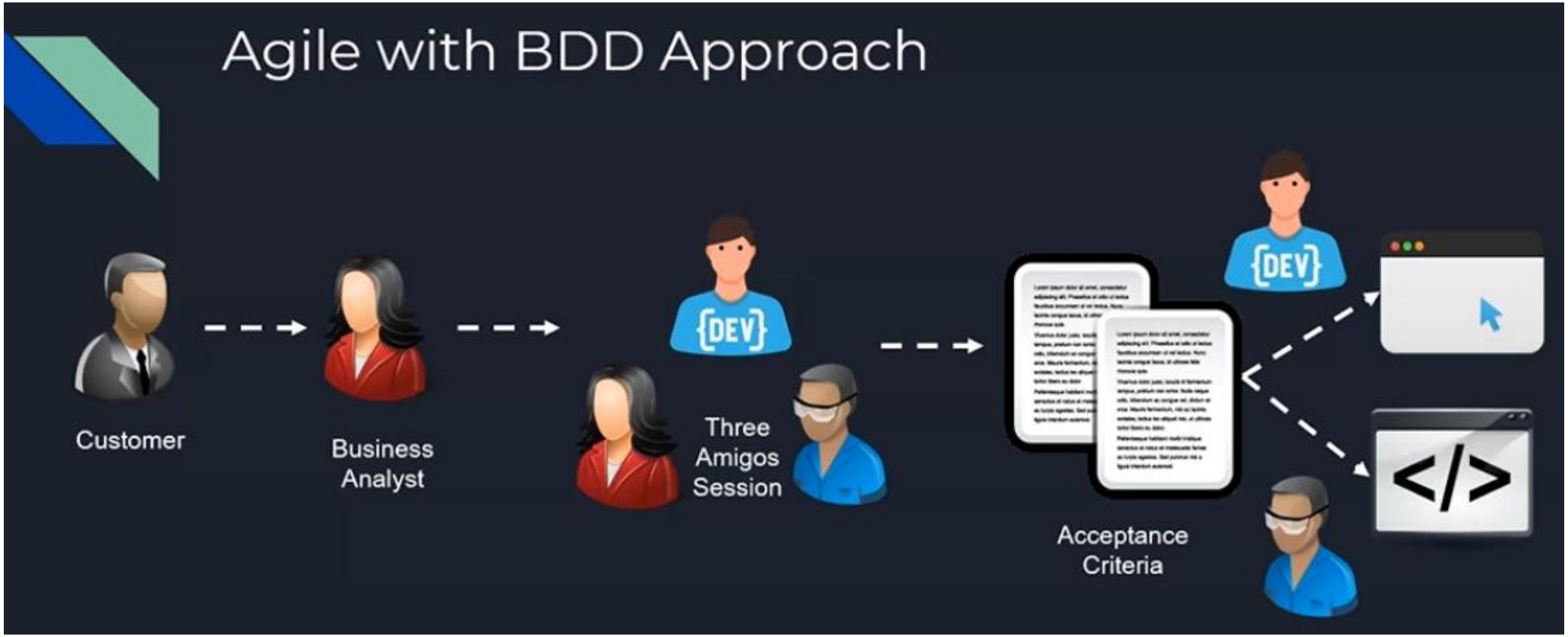
Good form for generating tests!

- Context describes setup
- Action describes methods to be called
- Result describes test outcome

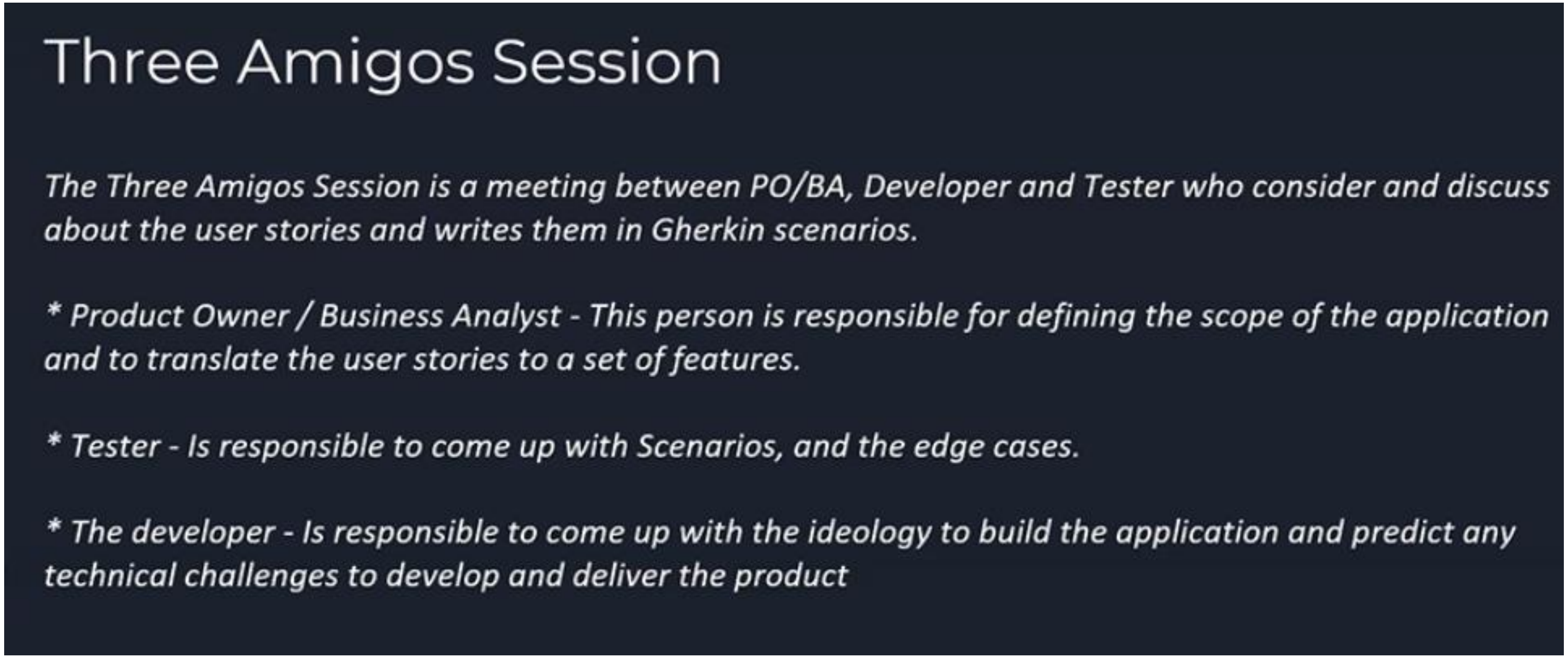
Typical Waterfall Model



Agile with BDD Approach



Three Amigoes Session



Three Amigos Session

The Three Amigos Session is a meeting between PO/BA, Developer and Tester who consider and discuss about the user stories and writes them in Gherkin scenarios.

** Product Owner / Business Analyst - This person is responsible for defining the scope of the application and to translate the user stories to a set of features.*

** Tester - Is responsible to come up with Scenarios, and the edge cases.*

** The developer - Is responsible to come up with the ideology to build the application and predict any technical challenges to develop and deliver the product*

Introduction to Cucumber

- **Cucumber is a testing tool for Behavior-Driven Development approach.**

It defines application's behavior using simple English and defined by a language called Gherkin.

- **Gherkin is a special language that is designed for Behavior-Driven Development approach and it is used by many organizations extensively.**

It helps to easily read and understand the automated acceptance tests.

Just to recollect, while we were discussing the Three Amigos Session, we learned that the outcome of the session would be the acceptance criteria, isn't it?

- **These acceptance criteria are nothing but the Scenarios, and it is written in this Gherkin language.**

Introduction to Gherkin

- Gherkin is a set of grammar rules that makes plain text structured enough for Cucumber to understand.

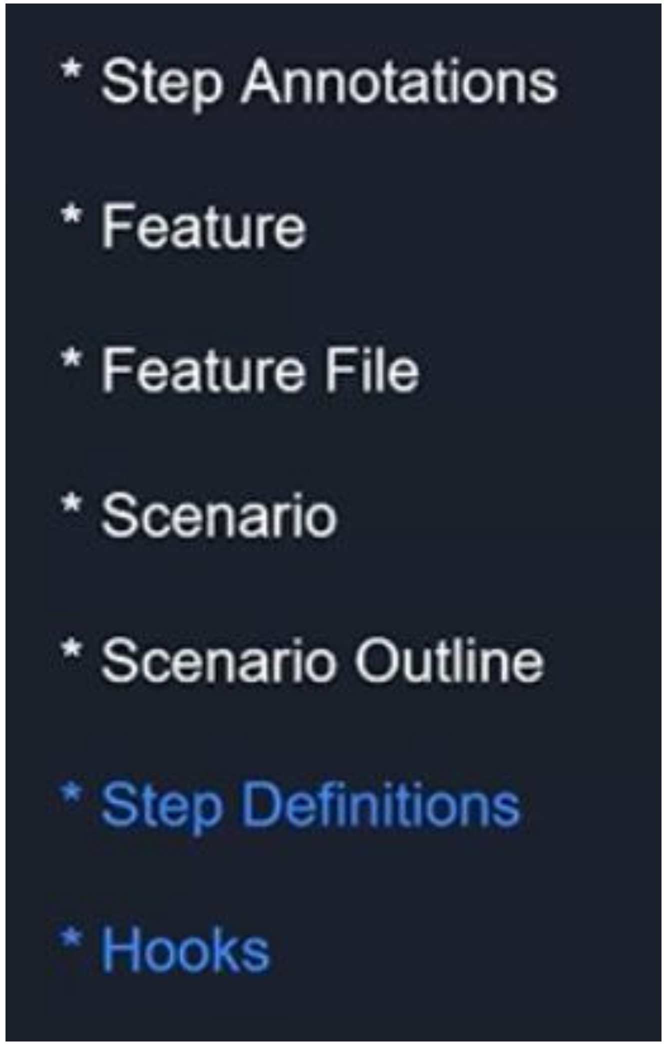
- Moreover, Gherkin is an English text language that helps the Cucumber tool to interpret and execute the automated test scripts, which are nothing but other acceptance tests.

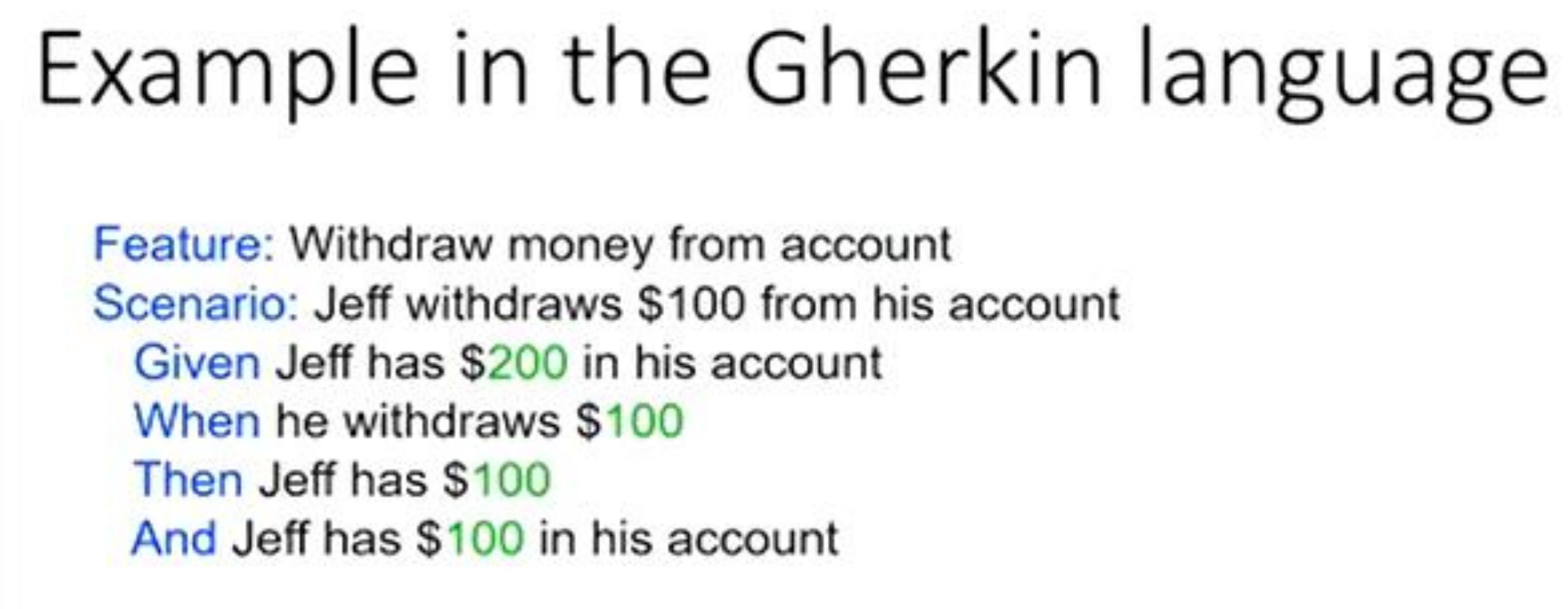
- Apart from this, Gherkin serves multiple purposes. These three purposes, which are given here, are the most common ones.

- First thing, it addresses the unambiguous executable specifications; and of course, it helps to automate the tests that are related to our application, using Cucumber.

- And also, it documents how the system actually behaves as well, using the concepts of living documentation.

Cucumber and Gherkin


* Step Annotations
* Feature
* Feature File
* Scenario
* Scenario Outline
* Step Definitions
* Hooks


Example in the Gherkin language

```
Feature: Withdraw money from account
Scenario: Jeff withdraws $100 from his account
  Given Jeff has $200 in his account
  When he withdraws $100
  Then Jeff has $100
  And Jeff has $100 in his account
```

There are totally 7 key concepts that we should understand when it comes to Cucumber and Gherkin.

•The first 5 that you're seeing here, the Step Annotations, Feature, Feature File, Scenario and Scenario Outline, are specific to Gherkin.

•The Step Definitions and Hooks are something that are related to the test automation framework that we are going to use along with Cucumber.

Installation of Cucumber

Installation:

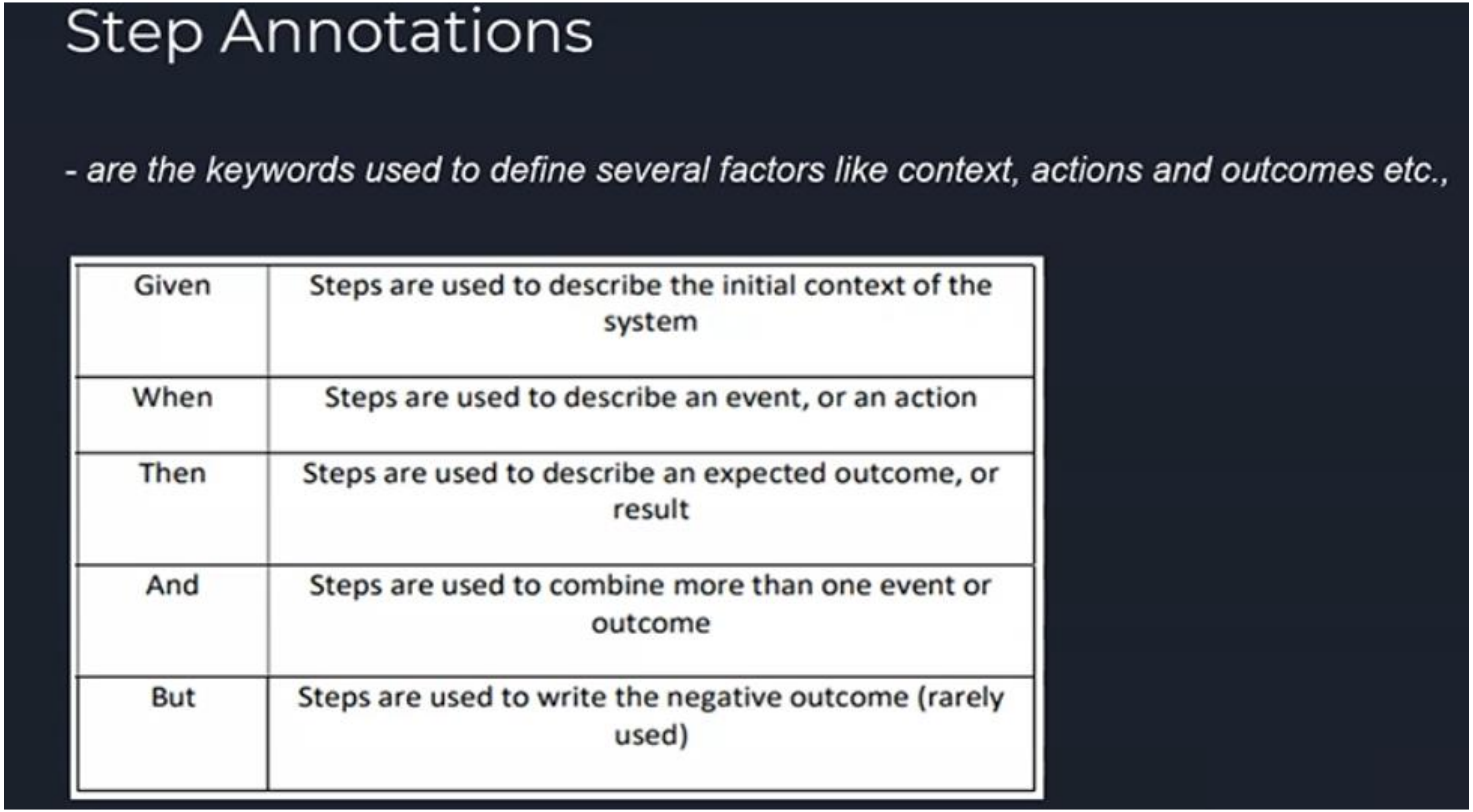
1. Creates a maven project in Eclipse.

2. Install cucumber plug-ins in eclipse

3. Add following dependencies in maven project:
Cucumber related files: Cucumber-core, Cucumber Java, junit-jupiter-engine, Cucumber-junit, cucumber-picocontainer, extentreports-cucumber7-adapter, cucumber-junit-platform-engine

https://cucumber.io/docs/guides/10-minute-tutorial/?lang=java

Step Annotations

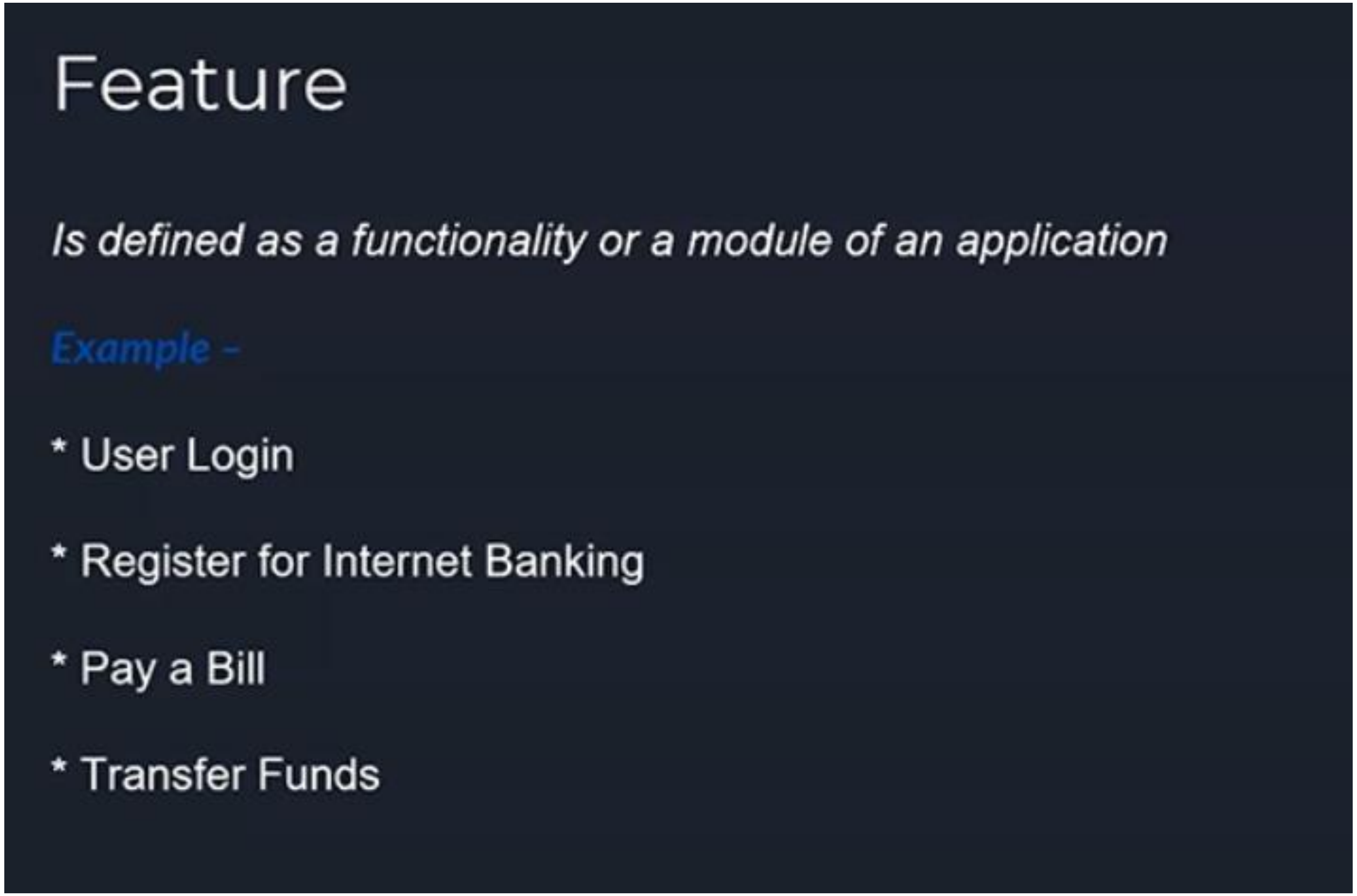


Step Annotations

- are the keywords used to define several factors like context, actions and outcomes etc.,

Given	Steps are used to describe the initial context of the system
When	Steps are used to describe an event, or an action
Then	Steps are used to describe an expected outcome, or result
And	Steps are used to combine more than one event or outcome
But	Steps are used to write the negative outcome (rarely used)

Feature



Feature

Is defined as a functionality or a module of an application

Example –

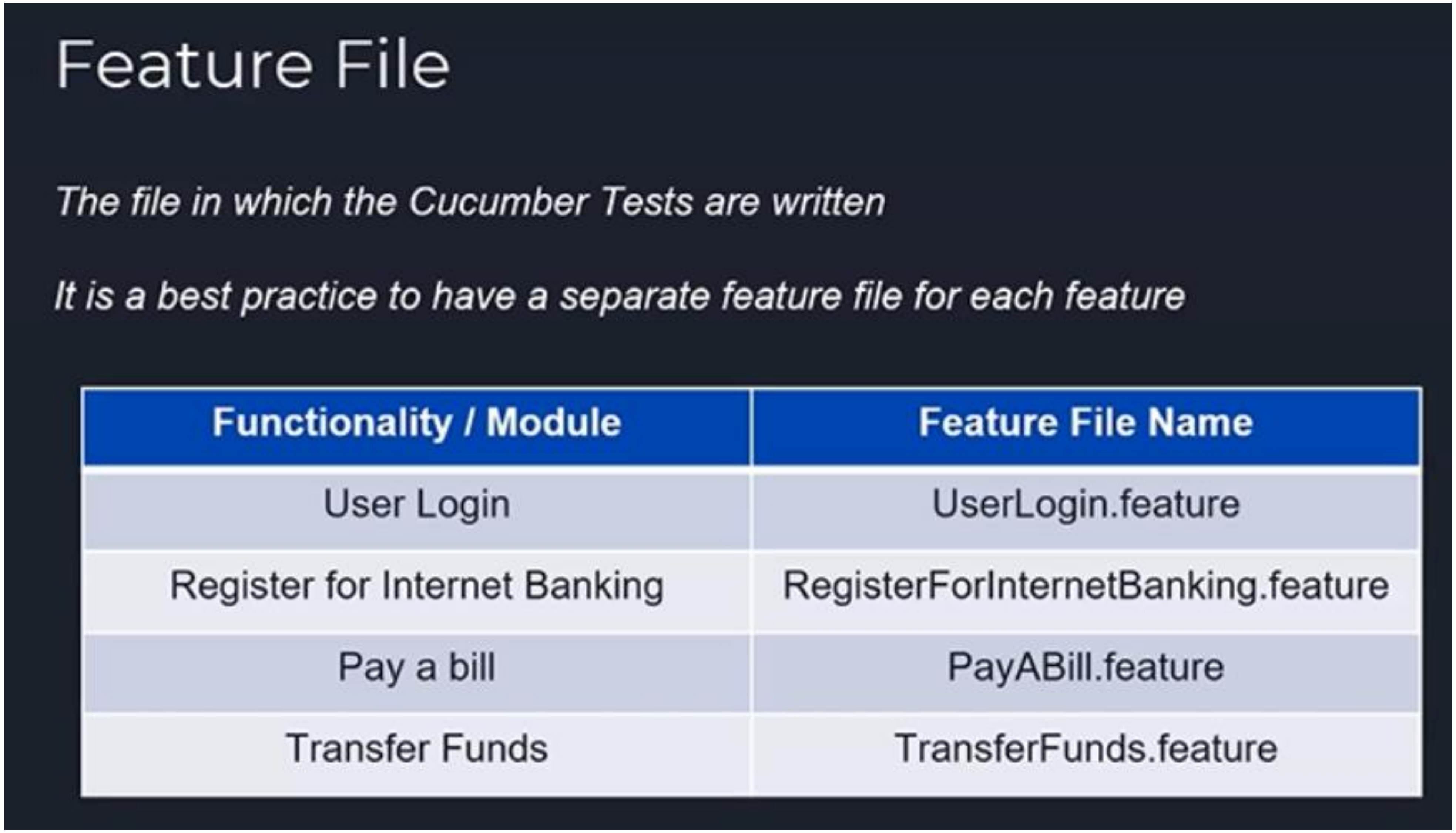
* User Login

* Register for Internet Banking

* Pay a Bill

* Transfer Funds

Feature File



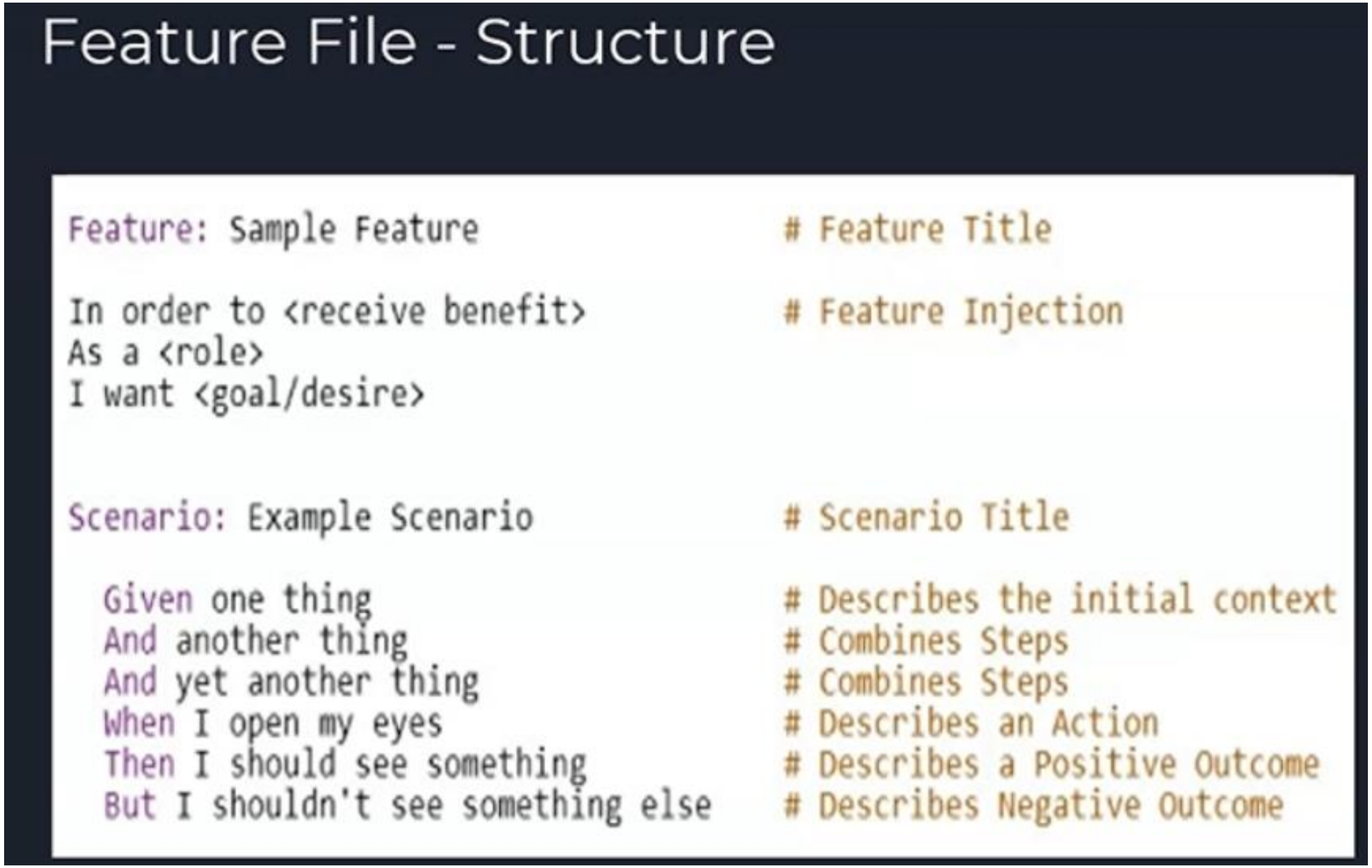
Feature File

The file in which the Cucumber Tests are written

It is a best practice to have a separate feature file for each feature

Functionality / Module	Feature File Name
User Login	UserLogin.feature
Register for Internet Banking	RegisterForInternetBanking.feature
Pay a bill	PayABill.feature
Transfer Funds	TransferFunds.feature

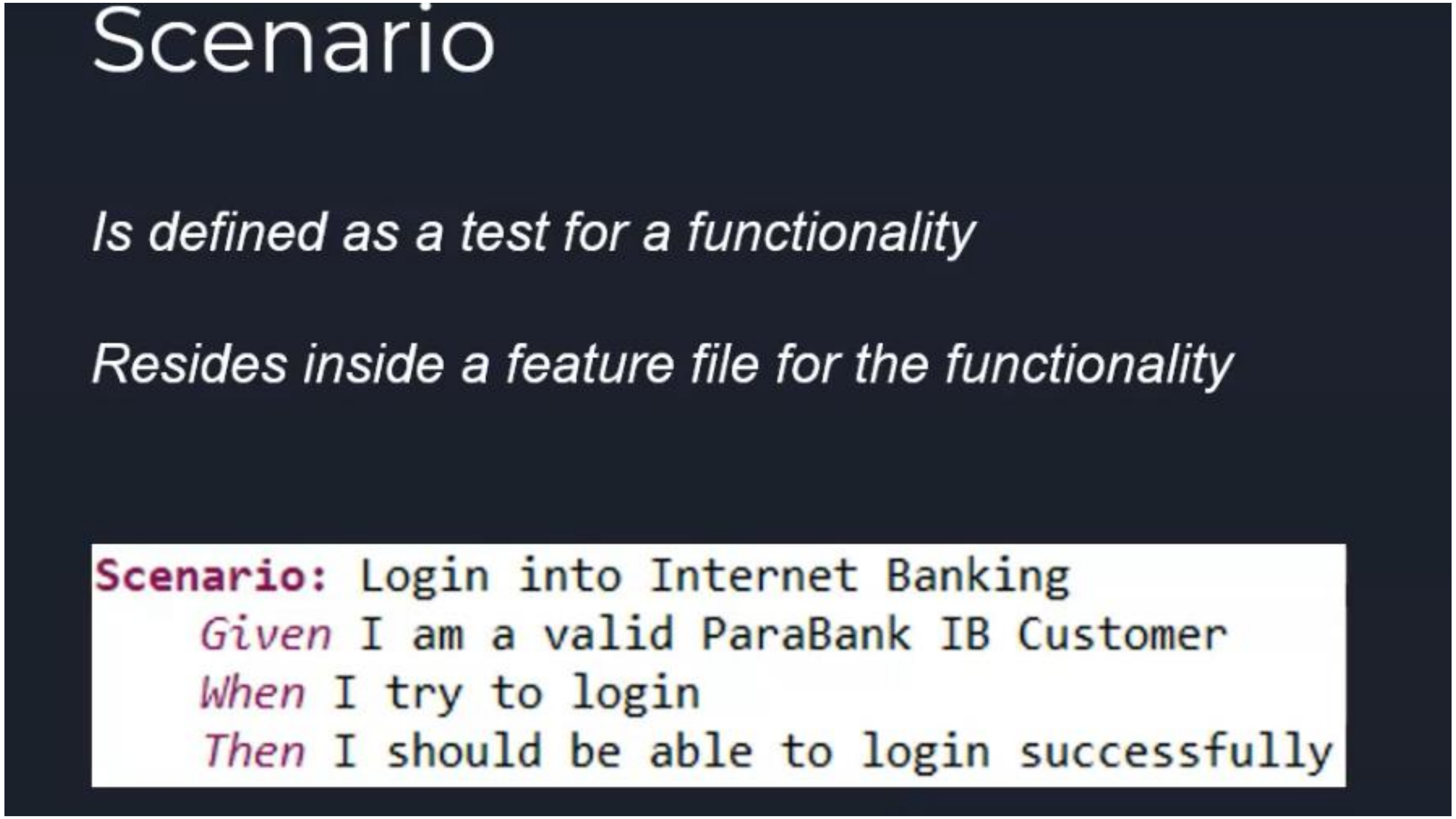
Feature File Structure



Feature File - Structure

```
Feature: Sample Feature                          # Feature Title

In order to <receive benefit>                    # Feature Injection
As a <role>
I want <goal/desire>


Scenario: Example Scenario                       # Scenario Title

  Given one thing                                # Describes the initial context
  And another thing                              # Combines Steps
  And yet another thing                          # Combines Steps
  When I open my eyes                            # Describes an Action
  Then I should see something                    # Describes a Positive Outcome
  But I shouldn't see something else             # Describes Negative Outcome
```

Scenario



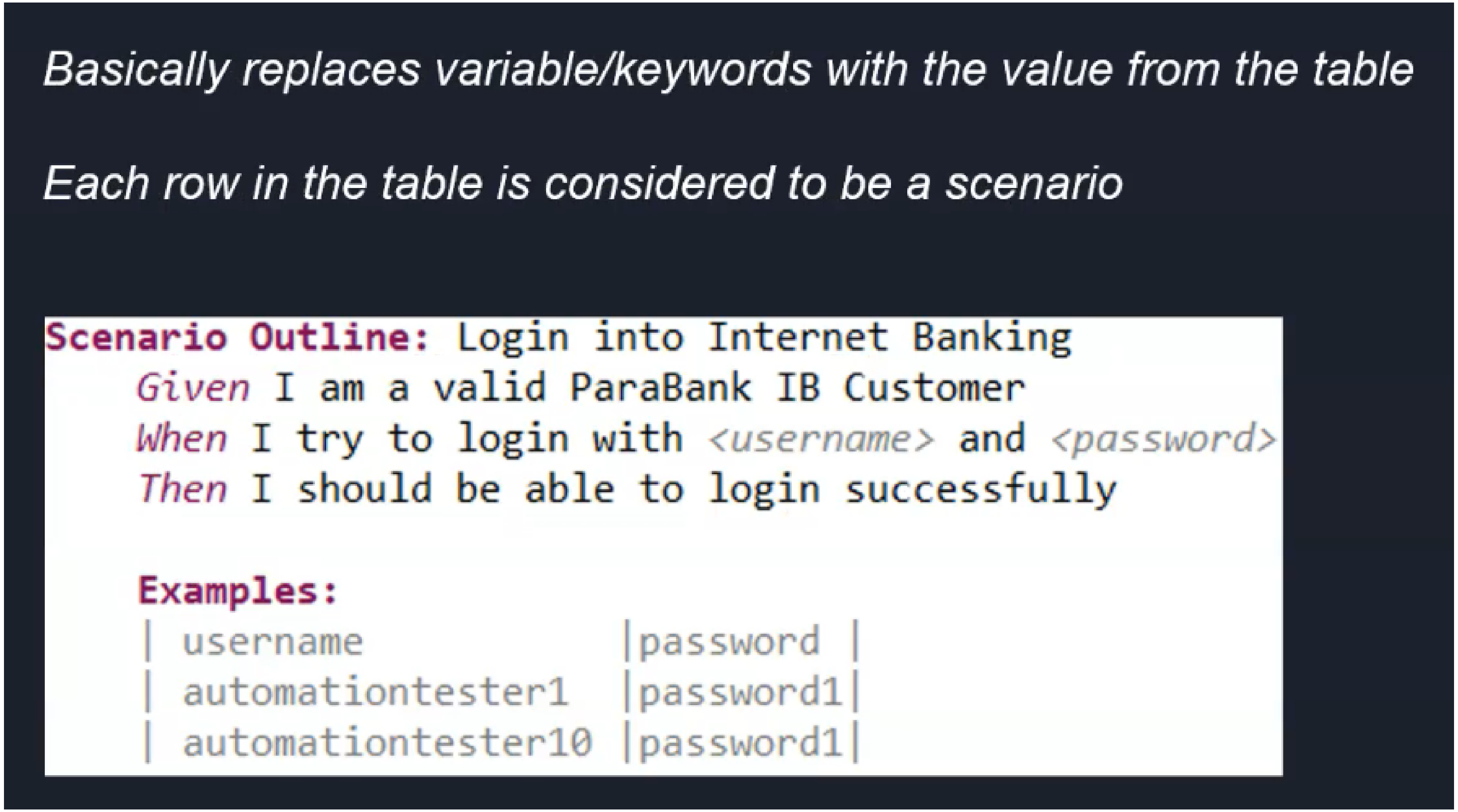
Scenario

Is defined as a test for a functionality

Resides inside a feature file for the functionality

```
Scenario: Login into Internet Banking
    Given I am a valid ParaBank IB Customer
    When I try to login
    Then I should be able to login successfully
```

Scenario Outline

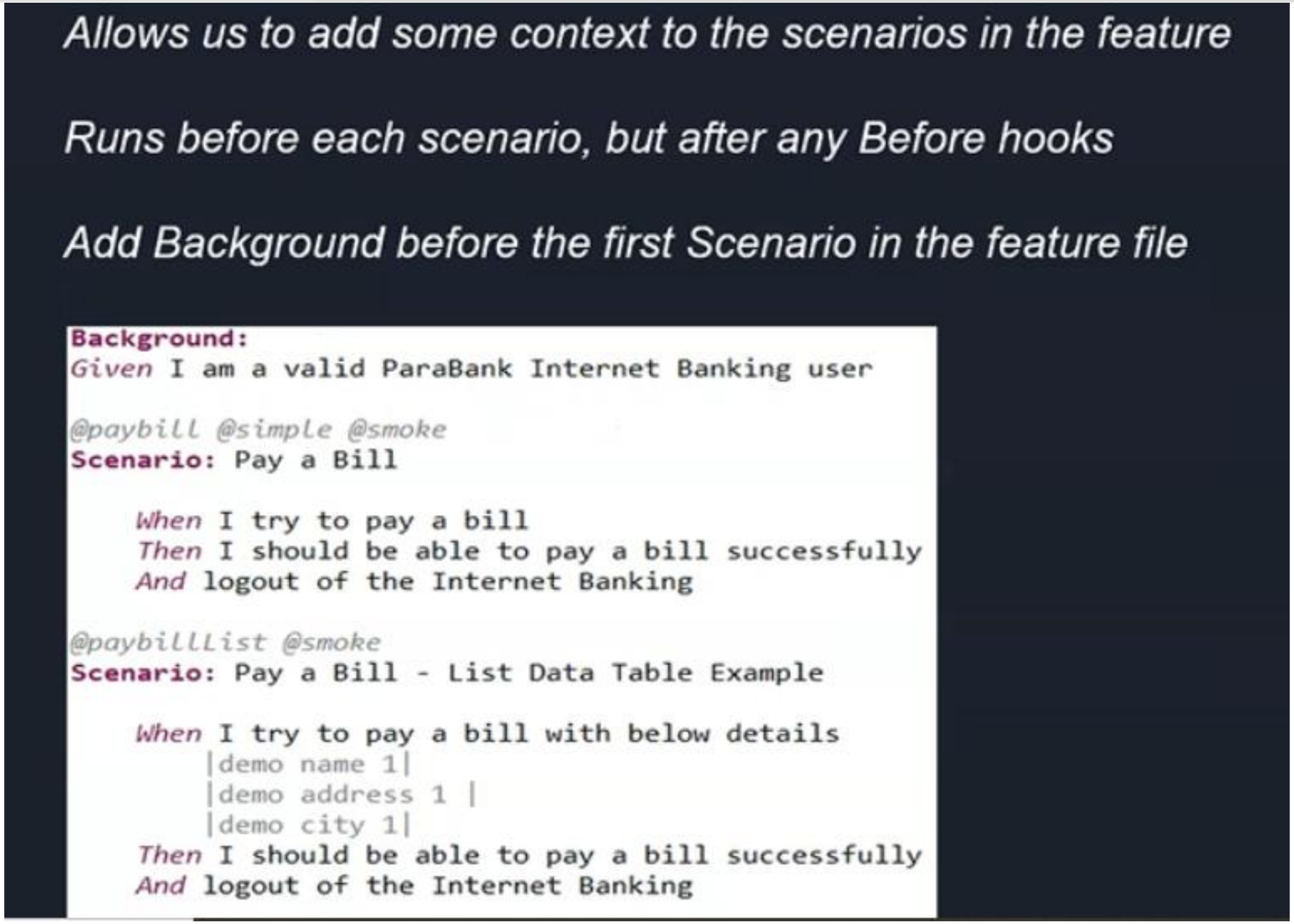


Basically replaces variable/keywords with the value from the table

Each row in the table is considered to be a scenario

```
Scenario Outline: Login into Internet Banking
    Given I am a valid ParaBank IB Customer
    When I try to login with <username> and <password>
    Then I should be able to login successfully

    Examples:
    | username           |password |
    | automationtester1  |password1|
    | automationtester10 |password1|
```

Background



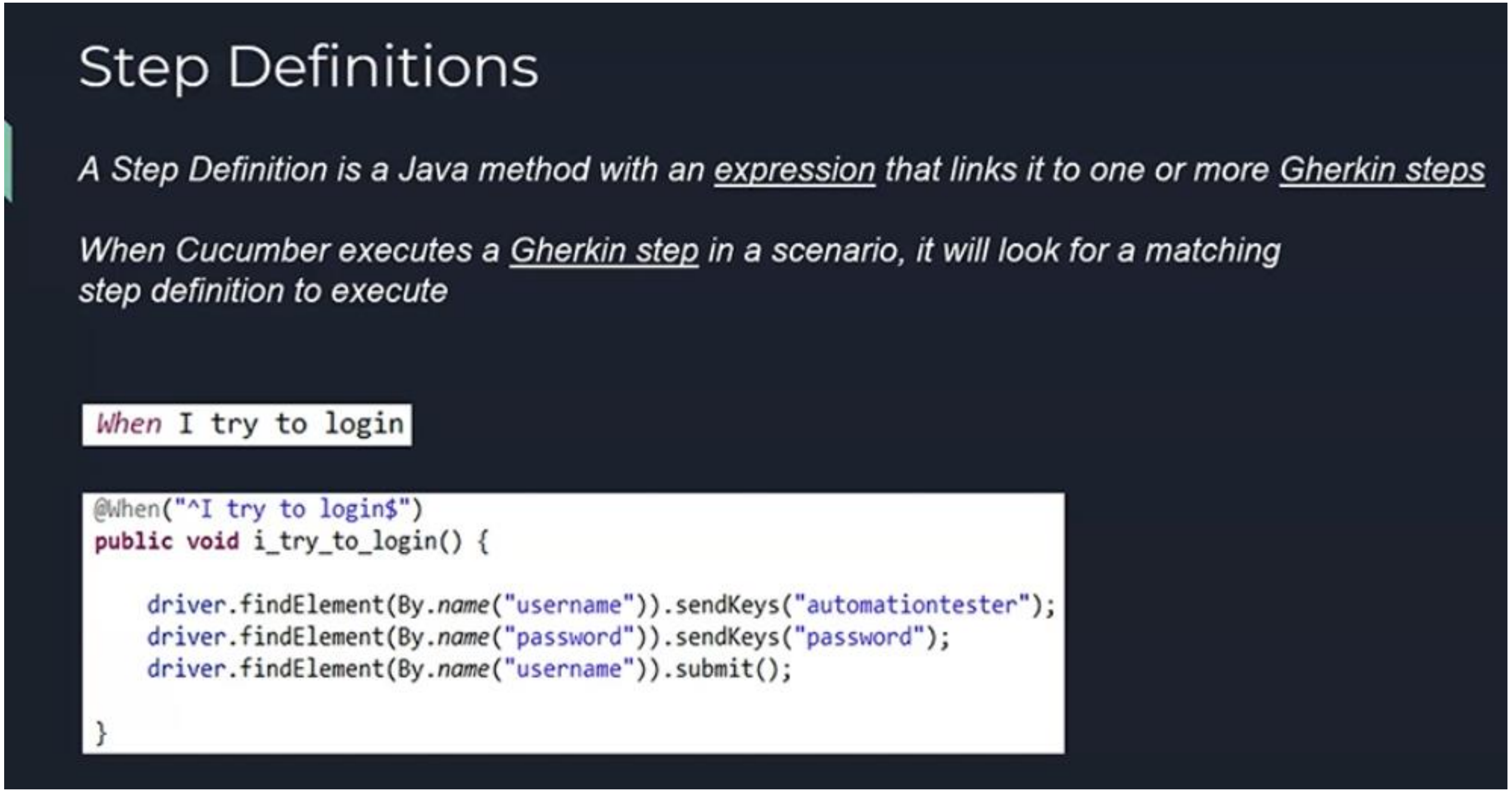
Allows us to add some context to the scenarios in the feature

Runs before each scenario, but after any Before hooks

Add Background before the first Scenario in the feature file

```
Background:
Given I am a valid ParaBank Internet Banking user

@paybill @simple @smoke
Scenario: Pay a Bill

    When I try to pay a bill
    Then I should be able to pay a bill successfully
    And logout of the Internet Banking

@paybillList @smoke
Scenario: Pay a Bill - List Data Table Example

    When I try to pay a bill with below details
         |demo name 1|
         |demo address 1 |
         |demo city 1|
    Then I should be able to pay a bill successfully
    And logout of the Internet Banking
```

Step Definitions



Step Definitions

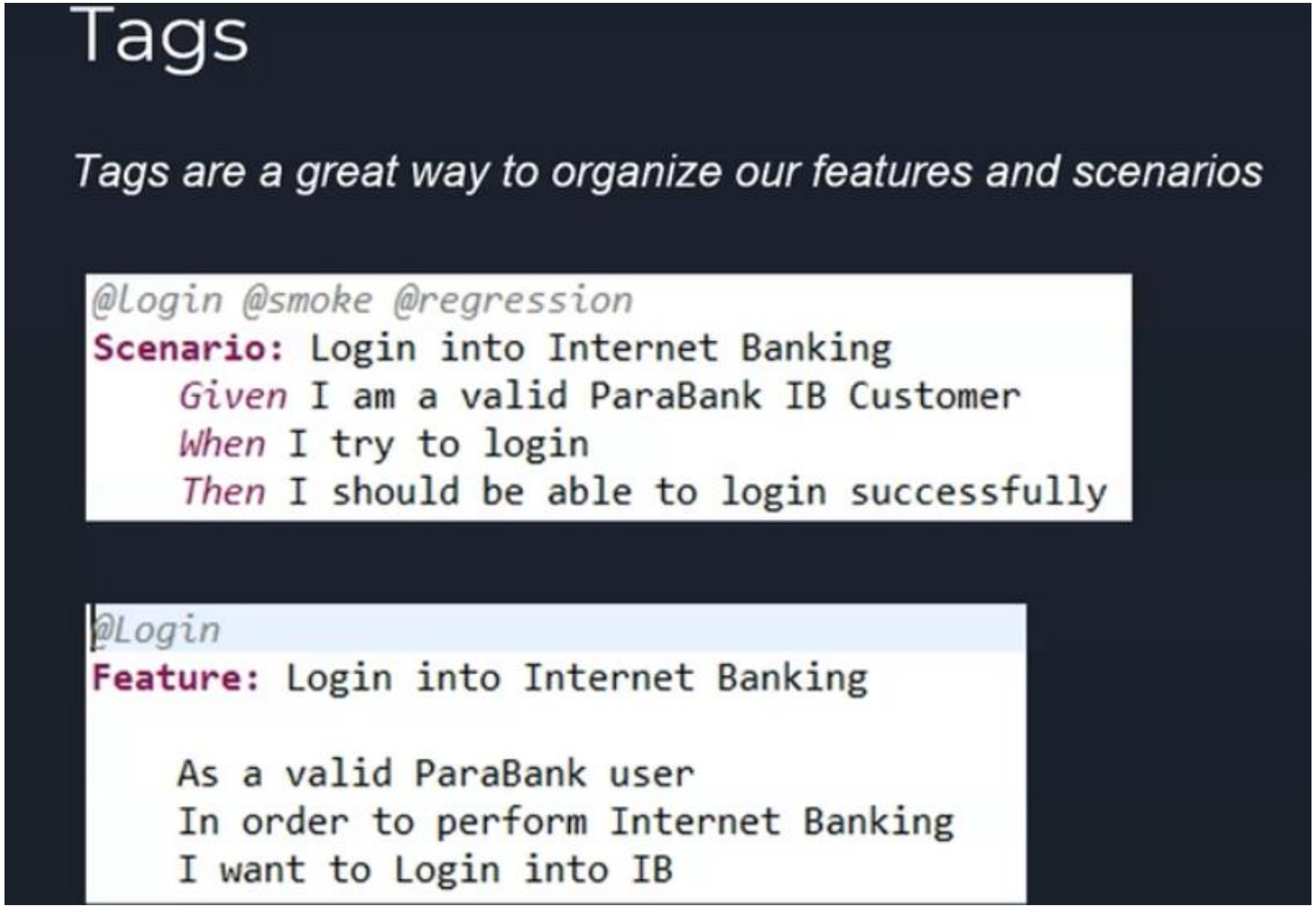
A Step Definition is a Java method with an expression that links it to one or more Gherkin steps

When Cucumber executes a Gherkin step in a scenario, it will look for a matching step definition to execute

```
When I try to login
```

```
@When("^I try to login$")
public void i_try_to_login() {

    driver.findElement(By.name("username")).sendKeys("automationtester");
    driver.findElement(By.name("password")).sendKeys("password");
    driver.findElement(By.name("username")).submit();

}
```

Tags



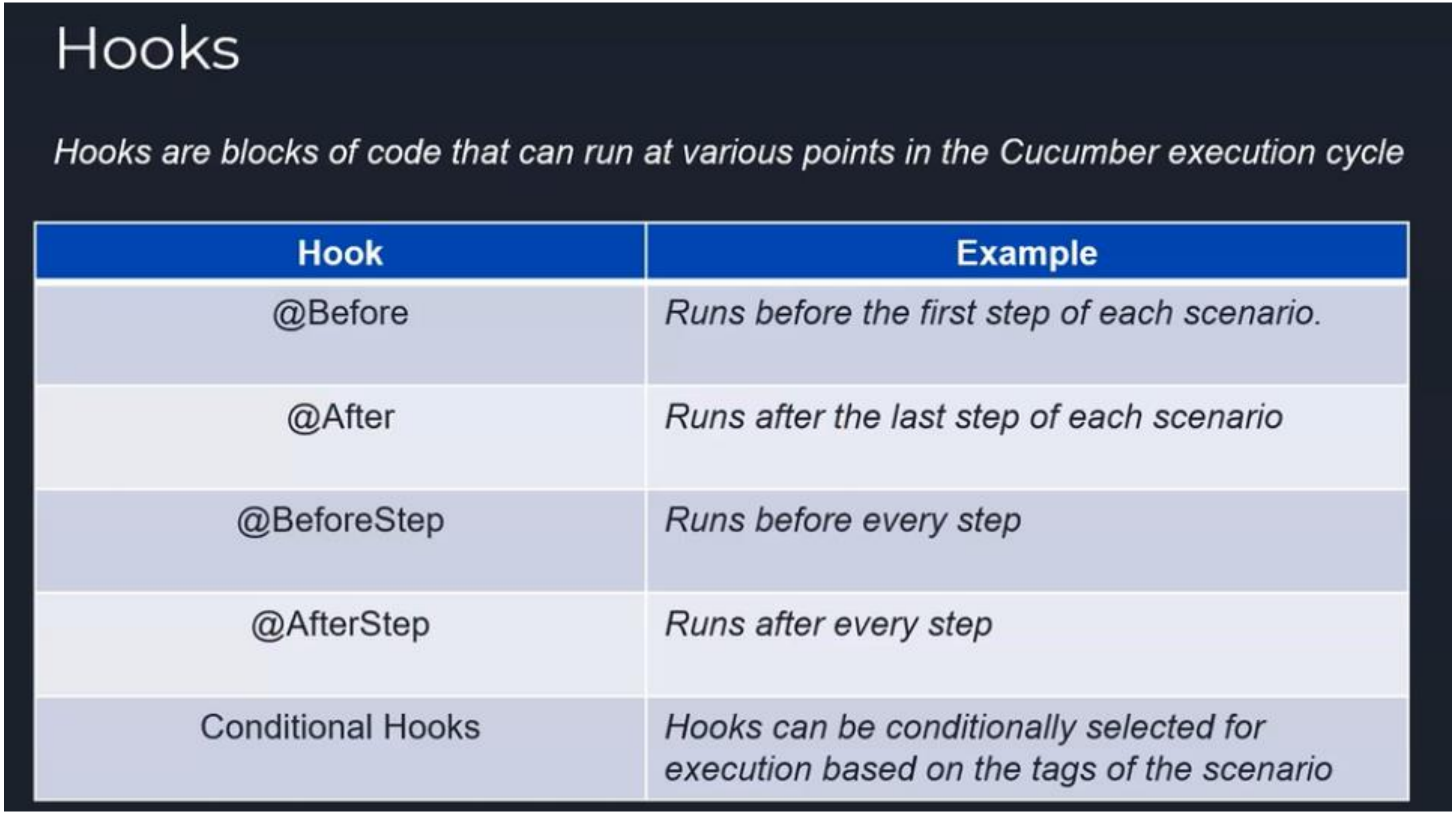
Tags

Tags are a great way to organize our features and scenarios

```
@Login @smoke @regression
Scenario: Login into Internet Banking
    Given I am a valid ParaBank IB Customer
    When I try to login
    Then I should be able to login successfully
```

```
@Login
Feature: Login into Internet Banking

    As a valid ParaBank user
    In order to perform Internet Banking
    I want to Login into IB
```

Hooks



Hooks

Hooks are blocks of code that can run at various points in the Cucumber execution cycle

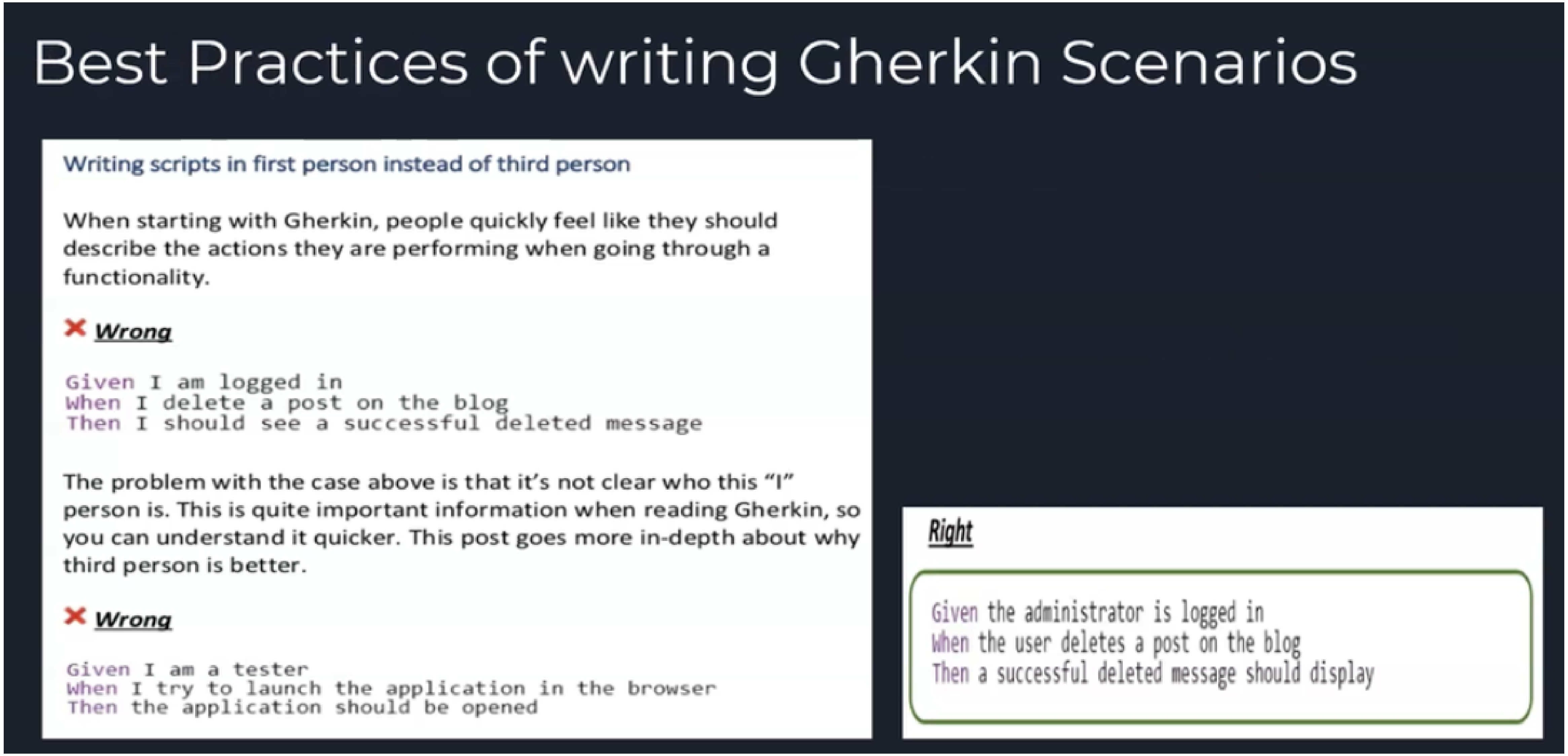
Hook	Example
@Before	*Runs before the first step of each scenario.*
@After	*Runs after the last step of each scenario*
@BeforeStep	*Runs before every step*
@AfterStep	*Runs after every step*
Conditional Hooks	*Hooks can be conditionally selected for execution based on the tags of the scenario*

3 Golden Rules

3 Golden Rules for Gherkin

- We will see about 3 golden rules that we need to follow religiously while writing the Gherkin.
- First, in a given Scenario, there should be only one Given, one When and one Then steps — not more than one Given or When or Then should be present as part of the Gherkin file.
- The next one is the maximum usage of two And steps are allowed
- Make sure your Scenario has maximum of five steps and not more than that.
- In case, if it is exceeding, try to split it into two different Scenarios.

Best Practices of Writing Gherkin Syntaxes



Best Practices of writing Gherkin Scenarios

Writing scripts in first person instead of third person

When starting with Gherkin, people quickly feel like they should describe the actions they are performing when going through a functionality.

✗ ***Wrong***

```
Given I am logged in
When I delete a post on the blog
Then I should see a successful deleted message
```

The problem with the case above is that it's not clear who this "I" person is. This is quite important information when reading Gherkin, so you can understand it quicker. This post goes more in-depth about why third person is better.

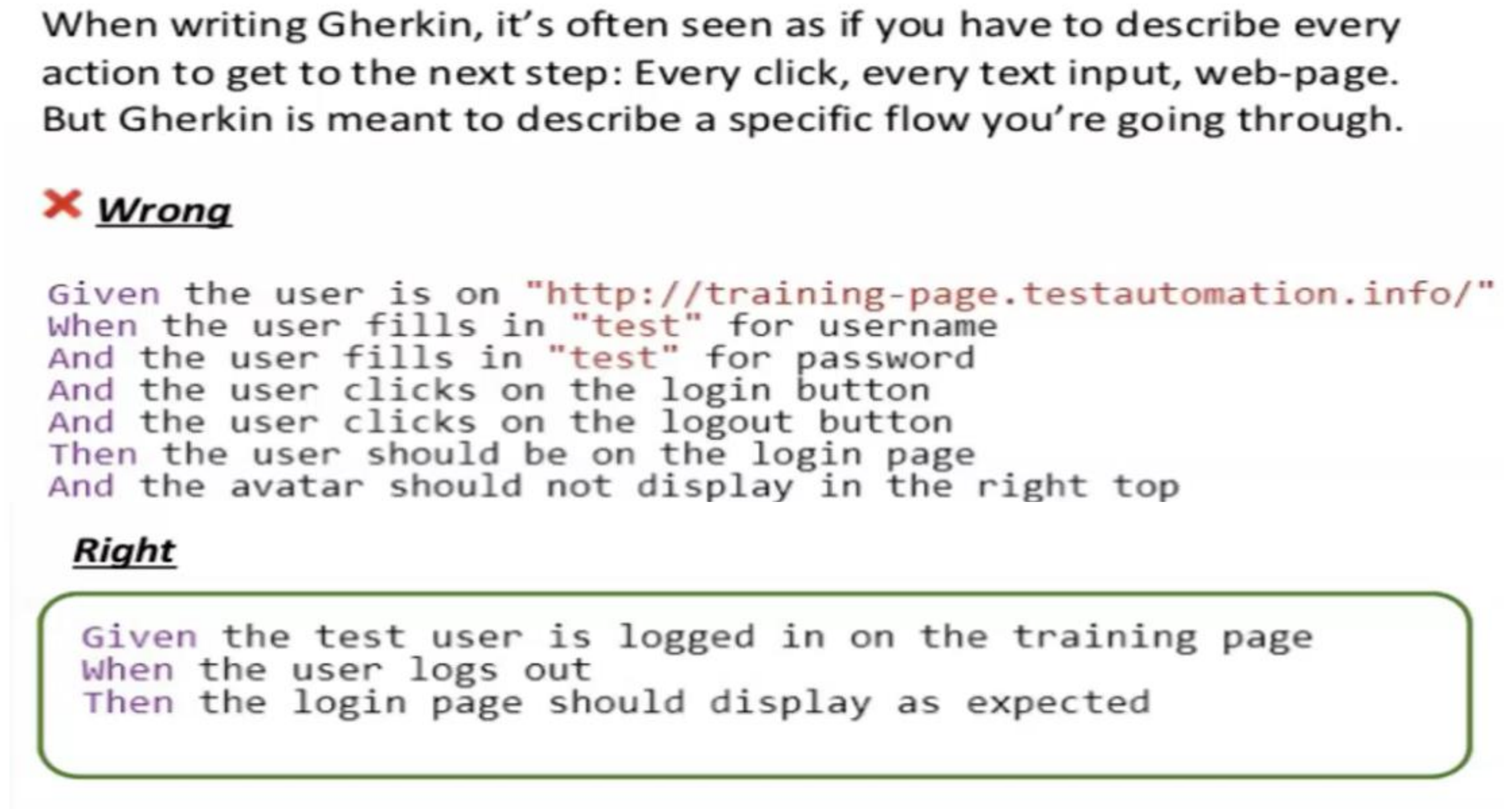
✗ ***Wrong***

```
Given I am a tester
When I try to launch the application in the browser
Then the application should be opened
```

Right

```
Given the administrator is logged in
When the user deletes a post on the blog
Then a successful deleted message should display
```

Best Practices of Writing Gherkin Syntaxes



When writing Gherkin, it's often seen as if you have to describe every action to get to the next step: Every click, every text input, web-page. But Gherkin is meant to describe a specific flow you're going through.

✗ ***Wrong***

```
Given the user is on "http://training-page.testautomation.info/"
When the user fills in "test" for username
And the user fills in "test" for password
And the user clicks on the login button
And the user clicks on the logout button
Then the user should be on the login page
And the avatar should not display in the right top
```

Right

```
Given the test user is logged in on the training page
When the user logs out
Then the login page should display as expected
```