When things go awry: Exceptions in Selenium

Has it happened to you that you went to a restaurant to have a nice dinner and when it's time to pay the bill, you realize that you have forgotten your wallet at home? This will be called Wallet Not Found exception in real life. Similarly, there are instances in Selenium when you intend your code to do something, but it does not do as expected. They are called exceptions.

When an exception occurs, your program flow disrupts, and it can halt your execution. A programmer should always handle checked exceptions. To handle exceptions, you should understand what this exception is and how to handle/resolve it.

I will talk about some common exceptions in WebDriver and will discuss their resolutions.

StaleElementReferenceException:

When the element that you are trying to perform an operation on is no longer present on the page or DOM has been refreshed then this exception is thrown. A real time scenario is when you click on submit button with invalid username and password. And then again when you click on submit button with valid username and password then you will get *StaleElementReferenceException.*

For example:

```
WebElement username = driver.findElement(By.id("username"));
                    WebElement pwd = driver.findElement(By.id("password"));
                    WebElement submitBtn = driver.findElement(By.id("submit"));

                    username.sendKeys("invalidUserName");
                    pwd.sendKeys("invalidPassword");
                    submitBtn.click();

                    username.sendKeys("validUserName");
                    pwd.sendKeys("validPassword");
                    submitBtn.click();
```

Resolution: This exception can be resolved by re allocating the element to reference variable. For example, in order to avoid exception you will have to

reallocate username, password and submit button again before performing scenario with valid username, password.

```
username = driver.findElement(By.id("username"));

pwd = driver.findElement(By.id("password"));

submitBtn = driver.findElement(By.id("submit"));

 username.sendKeys("validUserName");
pwd.sendKeys("validPassword");
submitBtn.click();
```

Another standard way of handling this exception is using Page Factory pattern. If you use Page Factory pattern in your framework then this exception will not occur. The reason is Page Factory does lazy initialization of Web Element. It searches for element every time when you perform an operation on it. So, every time you want to click or type on an element, it reallocates it.

P.S: Do not use @CacheLookup annotation with your element in Page Factory pattern because @CacheLookup annotation caches your element and does not search it again.

NoSuchElementException:

This is also one of the most common exception in WebDriver. This exception class is the subclass of *NotFoundException* class. The reason is your element is no accessible to your code. This occurs because you have given an incorrect locator, element is inside the frame or the element is not loaded.

Resolution:

- First, check if element is indeed on the page.
- Then check if there is any spelling mistake or syntax error in your locator.
- If element is inside frame, then you need to switch to frame to access this element.
- If element is not loaded due to synchronization issue, then you will have to give some wait time to handle this exception. Refer my article on waits here.

ElementNotVisibleException:

This exception occurs when an element is present in the DOM but it is not visible on the web page. This exception class is subclass of *ElementNotInteractableException* class. It could be that either element is hidden, or it not properly loaded. So, when WebDriver tries to perform operation on such element then this exception is thrown.

Resolution:

Try to give some wait time before performing operation on the element so that it becomes visible on the web page. Use visibilityOf(WebElement element) method of WebDriverWait class to ensure visibility of the element. If it is hidden, then with the help of indexing try to find locator that is displayed on the page.

ElementNotSelectableException:

This exception is subclass of *InvalidElementStateException* class. This occurs because an element is present in the DOM but can not be selected on the web page. This could be because the element is either disabled or already in selected state.

Resolution:

- Give some wait time to make element selectable.
- Use *driver.isEnabled()* method to check if element is disabled. If its disabled do not perform operation to avoid exception.
- Use *driver.isSelected()* method to check if element is already selected. If its selected do not perform operation to avoid exception.

Element is not clickable at point (xx, xx). Other element would receive the click:

I have mentioned this WebDriver exception here because I have faced it in real time. And you also might have faced it. This occurs when your element is

obfuscated by some other div or images. And due to this it does not receive click properly.

Resolution:

- Use click from the action class because it clicks in the middle of the element.

```
Actions action = new Actions(driver);
action.moveToElement(element).click().perform();
```

- Use click from JavaScriptExecutor class.

InvalidSelectorException:

This exception is subclass of *NoSuchElementException* class. This exception occurs when there is syntax error in your Xpath or CSS Selector. Or when you have used By.id(), By.name(), By.className() etc but passing Xpath or CSS Selector.

Resolution:

This is one of the simplest exceptions to resolve. You just need to check your locator and methods and make sure they are correct.

NoSuchSessionException:

This exception is subclass of WebDriverException class. This occurs when your browser is crashes, your driver.exe crashes or you execute a piece of code after driver.quit().

Resolution:

- Always use stable version of browser and driver.exe
- Perform driver.quit() only in after or teardown methods.

Compound class names not permitted:

This is part of WebDriverException. I have mentioned it here because you might be frequently getting it if you are using class name as your locator. This occurs when you have spaces in your class name.

Resolution:

Replace spaces in your class name with . and this exception will go away.

NoAlertPresentException:

This exception is subclass of *NotFoundException* class. This occurs when you are trying to accept, dismiss an alert or read text from an alert that is not yet displayed.

Resolution:

You can use a method to check whether Alert has opened.

```
public boolean isAlertPresent() {
	try {
	driver.switchTo().alert();
	return true;
	}catch(NoAlertPresentException e) {
		return false;
	}

    }
```

If this method returns true you can accept, dismiss or read text from the alert. If this method returns false, then you can give a wait time to let alert display.

There are huge number of exceptions in Selenium. You can find a complete list from official documentation here. I have given an overview of common exceptions and their resolutions. That's it for now. I will be back with some more knowledge sharing articles. Happy reading! 😊