**#1) Inheritance**

Inheritance is the mechanism in which a child class or a subclass gets the properties of the base class or parent class so that we can easily reuse the methods of the base class in a subclass. Inheritance is accomplished by the word “extends”.

**#2) Encapsulation**

Encapsulation is the process of wrapping up code and data together in a single unit. It is used to hide the data of a class from another class. Encapsulation can be achieved when you declare all variables as private and a public method in a class to get the values of the variable.

**#3) Polymorphism**

Polymorphism is best explained by the verb “One Interface Multiple Method”. It allows us to perform a task in multiple ways. It is a combination of Method Overriding (Static Polymorphism) and Method Overloading (Dynamic Polymorphism).

**Implementation Of OOPS Concepts**

As we know the basic definition of the OOPS principles, we will now focus on how these concepts are implemented in your Selenium Testing and what are the potential examples of OOPS in Selenium.

**#1) Inheritance**

In a typical Page Object Model, we create a Base Class where we initialize WebDriver interface, Data Source, Excel Reader, Property File or Config File, WebDriver waits and so on. We extend the Base Class in our Test Class and Utility Class.

We do it by using the extends keyword and achieve the Inheritance. This facilitates the reusability of the class and we don’t have to write the same initialization code over and over again. This not only promotes reusability but it also shortens the code that improves the Time and Space Complexity of your suite.

**#2) Encapsulation**

In a POM Project, we know that we create a separate class for every page. All these classes are the best examples of Encapsulation where we keep the data of a class separated from the other class.

In these POM Classes, we declare the data members using @FindBy and initialize them using a constructor with initElement() to utilize them in the test methods.

**#3) Polymorphism**

Polymorphism relies on the concept of one interface supporting multiple methods. As we know, WebDriver is an interface that supports multiple methods of different browsers like ChromeDriver(), IEDriver(), SafariDriver() and FirefoxDriver().

Hence the following line of codes will work to instantiate the browsers in two different ways.

**Using WebDriver Interface**

**WebDriver driver = new ChromeDriver();**

Here, we are using WebDriver Interface to instantiate the Chrome Browser.

This WebDriver interface supports all the methods that are there in the ChromeDriver class and the same can be rewritten for Firefox browsers which support methods of the FirefoxDriver class.

**WebDriver driver = new FirefoxDriver();**

**Without Using WebDriver Interface**

**ChromeDriver driver = new ChromeDriver()**;

Here, we are creating a reference for the ChromeDriver class and it will only support the methods that are there in the ChromeDriver class.

**Method Overloading In Selenium**

Methods Overloading is a process of using the two methods in the same class with the same name and different parameters.

**For Example,**

Testing(int a, char b)  
Testing(char b)

Now in Selenium, we all use Implicit Wait to make the page wait for some specified time interval.

This is the best example of Method Overloading as we can provide different Timestamp or TimeUnit like SECONDS, MINUTES, etc.

**Method Overriding In Selenium**

Method overriding is a process where a method in the child class has the same name and the same parameters as that of the method in its base class.

When we talk about the child class and the base class, we assume that both the classes are in an “is-a” relationship which is nothing but the inheritance.

**Examples of Method Overriding**

In the WebDriver interface, we use two different methods for navigating or accessing any website i.e. **driver.get() and driver.navigate().to().**

These two methods are examples of Method Overriding.

**Enlisted below is the basic difference between the navigate() and get() method and this is frequently asked in Selenium Interviews.**

* get() method will wait till the page is completely loaded in the browser while navigate() would not.
* navigate() method essentially returns a Navigate Interface which allows a user to traverse back, forward, or refresh pages as you would do in an actual browser window, while this functionality is not possible with the get() method

public class RemoteWebDriver implements WebDriver, JavascriptExecutor,

FindsById, FindsByClassName, FindsByLinkText, FindsByName,

FindsByCssSelector, FindsByTagName, FindsByXPath,

HasInputDevices, HasCapabilities, Interactive, TakesScreenshot {

...

public void get(String url) {

execute(DriverCommand.GET, ImmutableMap.of("url", url));

...

}

As you can see the RemoteDriver is overrridng the get() method it gets from the WebDriver interface

public interface WebDriver extends SearchContext {

....

void get(String url);

....

}

It does not matter if you are inheriting from an interface or from a class, as long as you are changing a method that you inherit from parent, your are overidding it

1) Inheritance concept

There is a baseclass in my framework which consists of commonly used method such as initialization of browser, implicit wait etc.

In every testcase, I want to initialize my browser so, in order to use that method in my TC I need to inherit base class

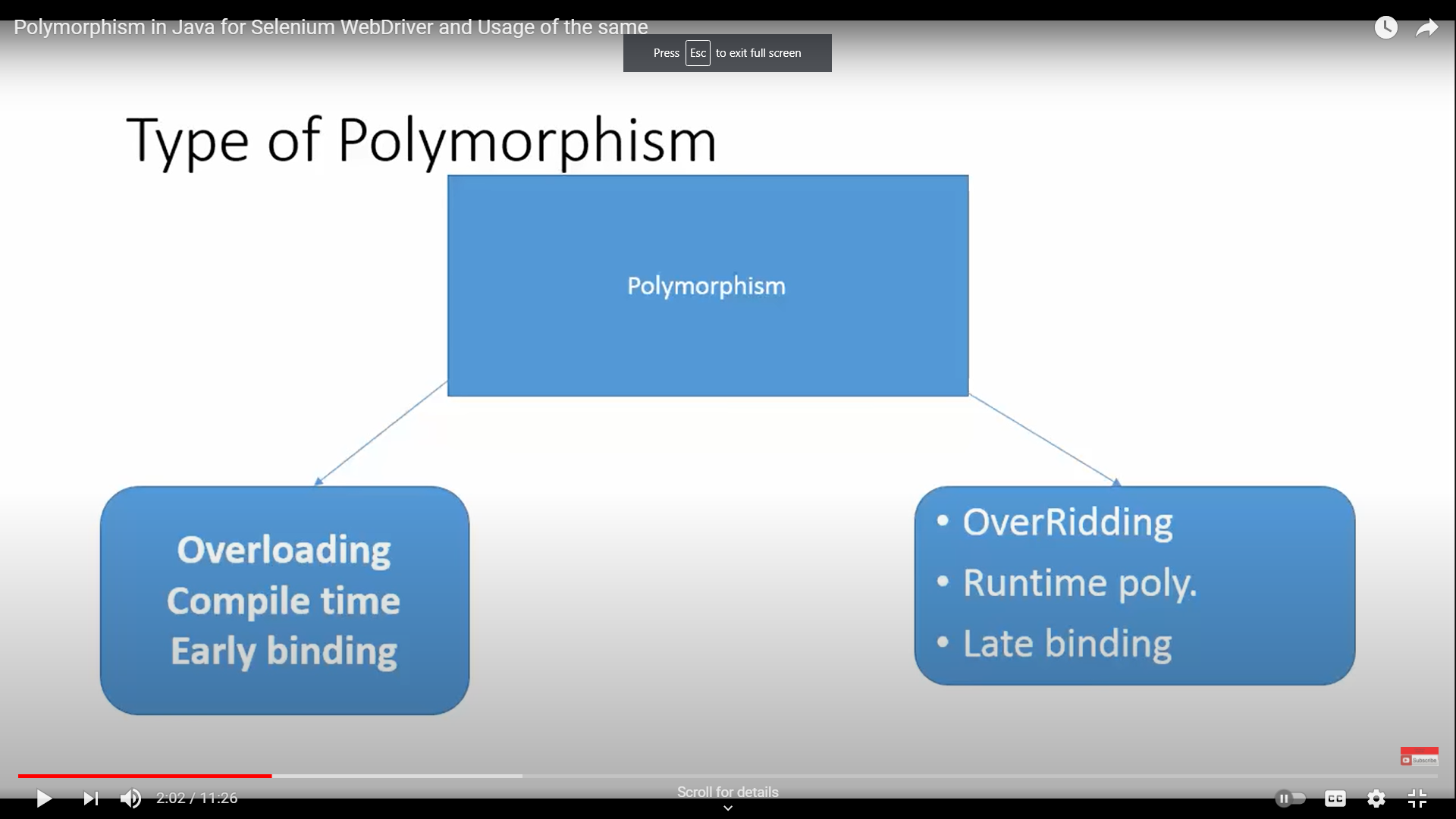
2)Encapsulation

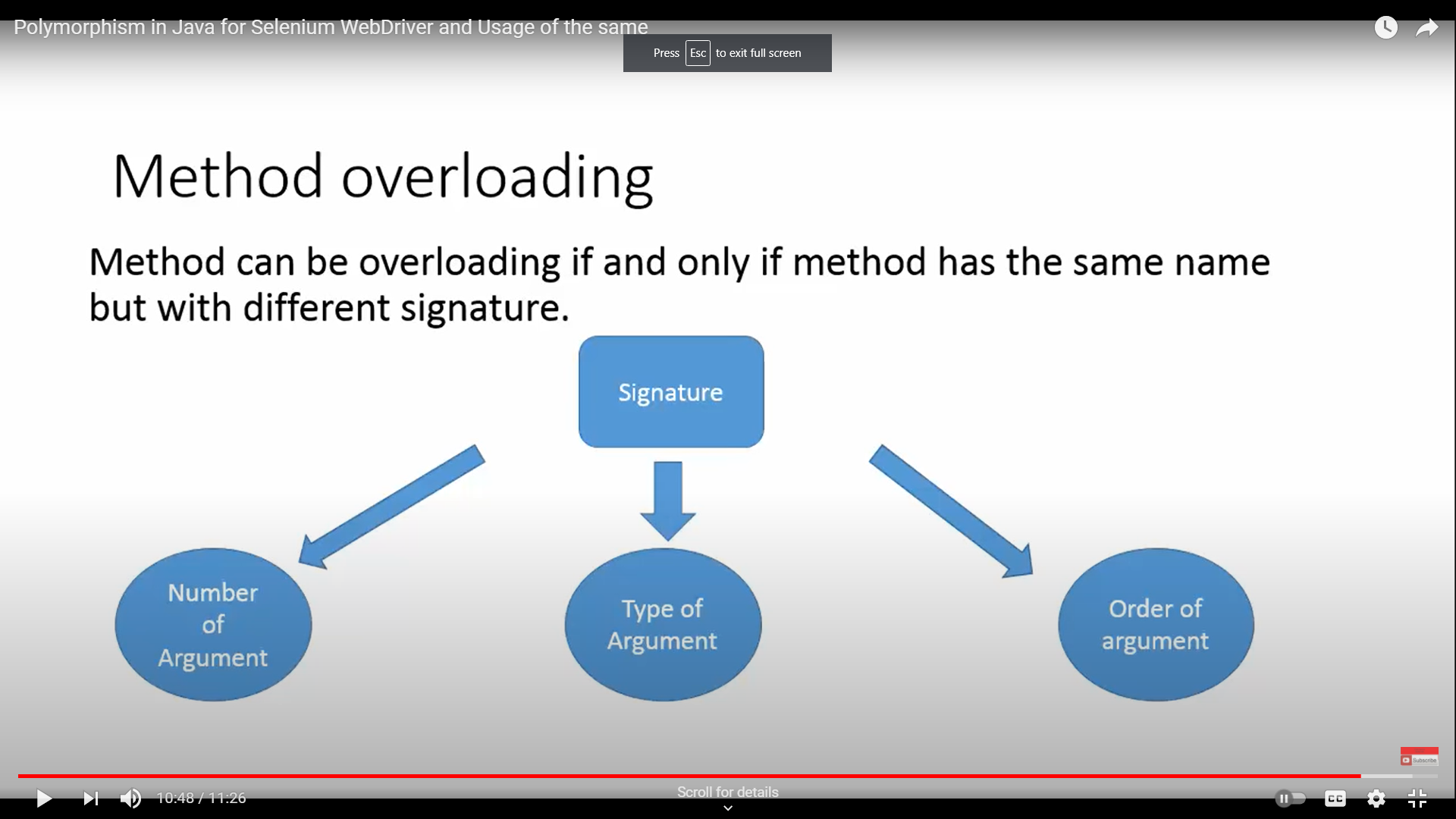
Defining private varaibles/locators and accessing them through public methods (this is encapsulation)

In Page object model, we are declaring locators as priviate and sending them to public methods,

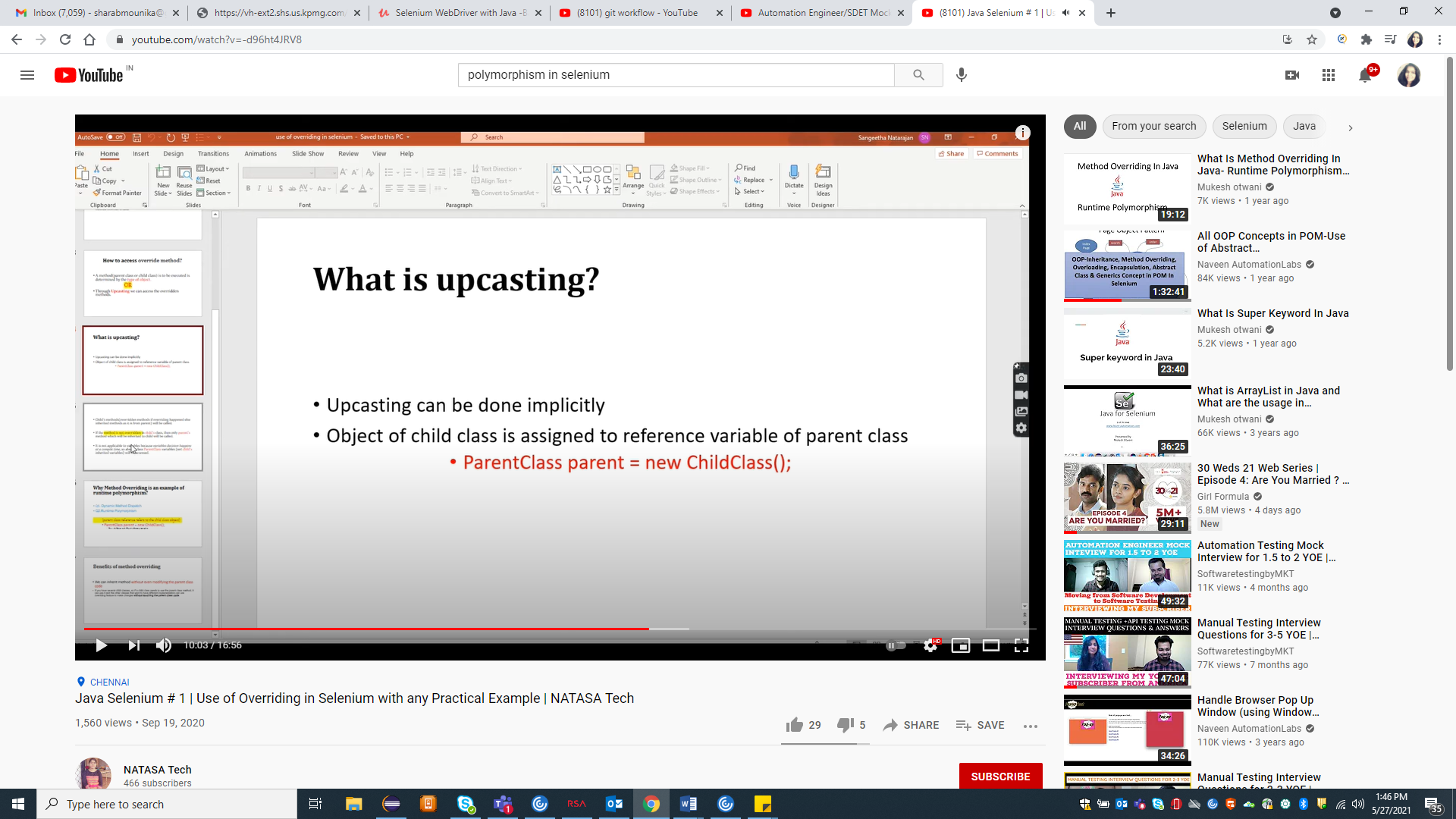
In this way other classes cannot directly access objects or locators of the class directly they need to get call public methods if they want to make use of private methods

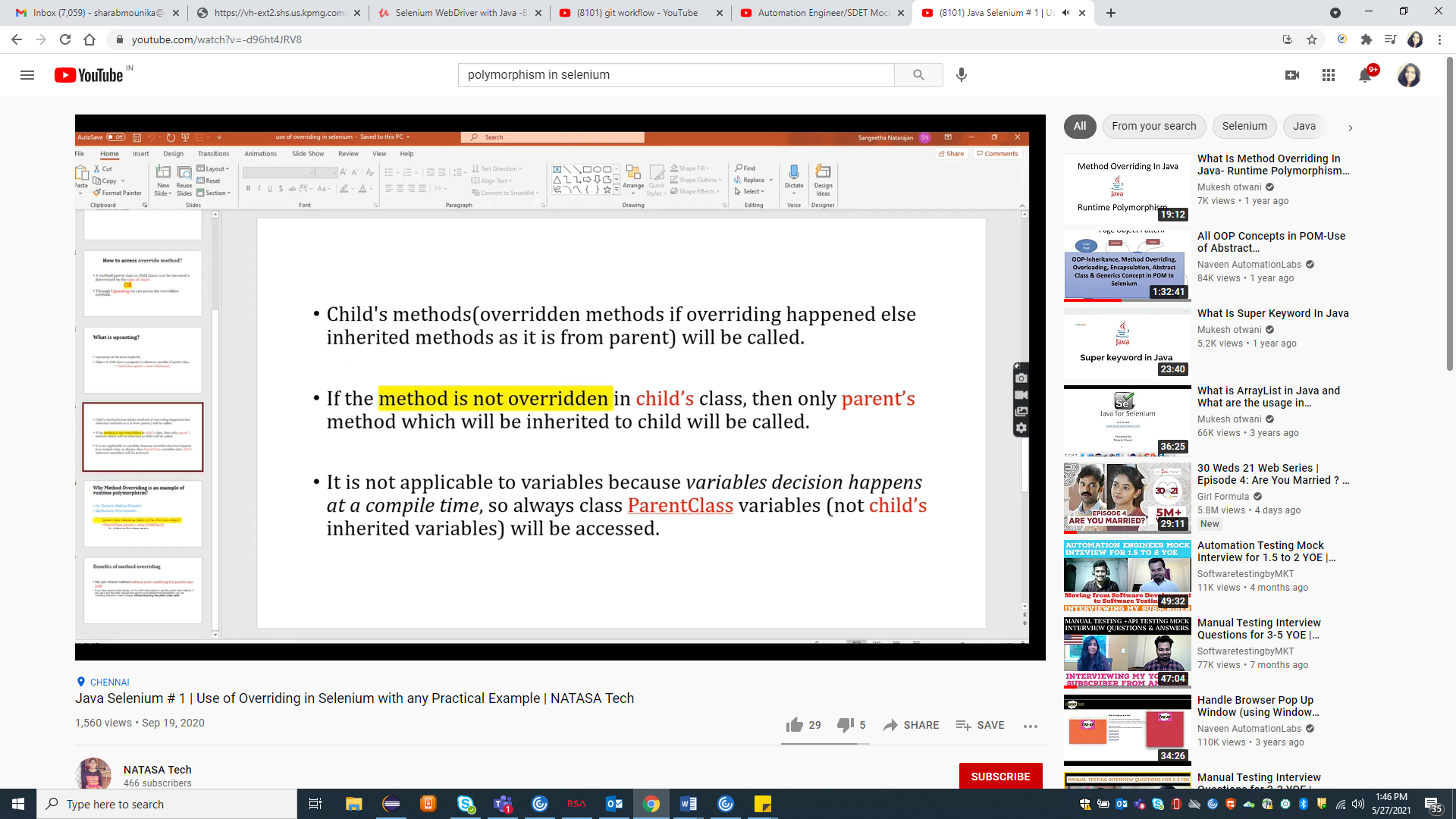
3) Polymorphism

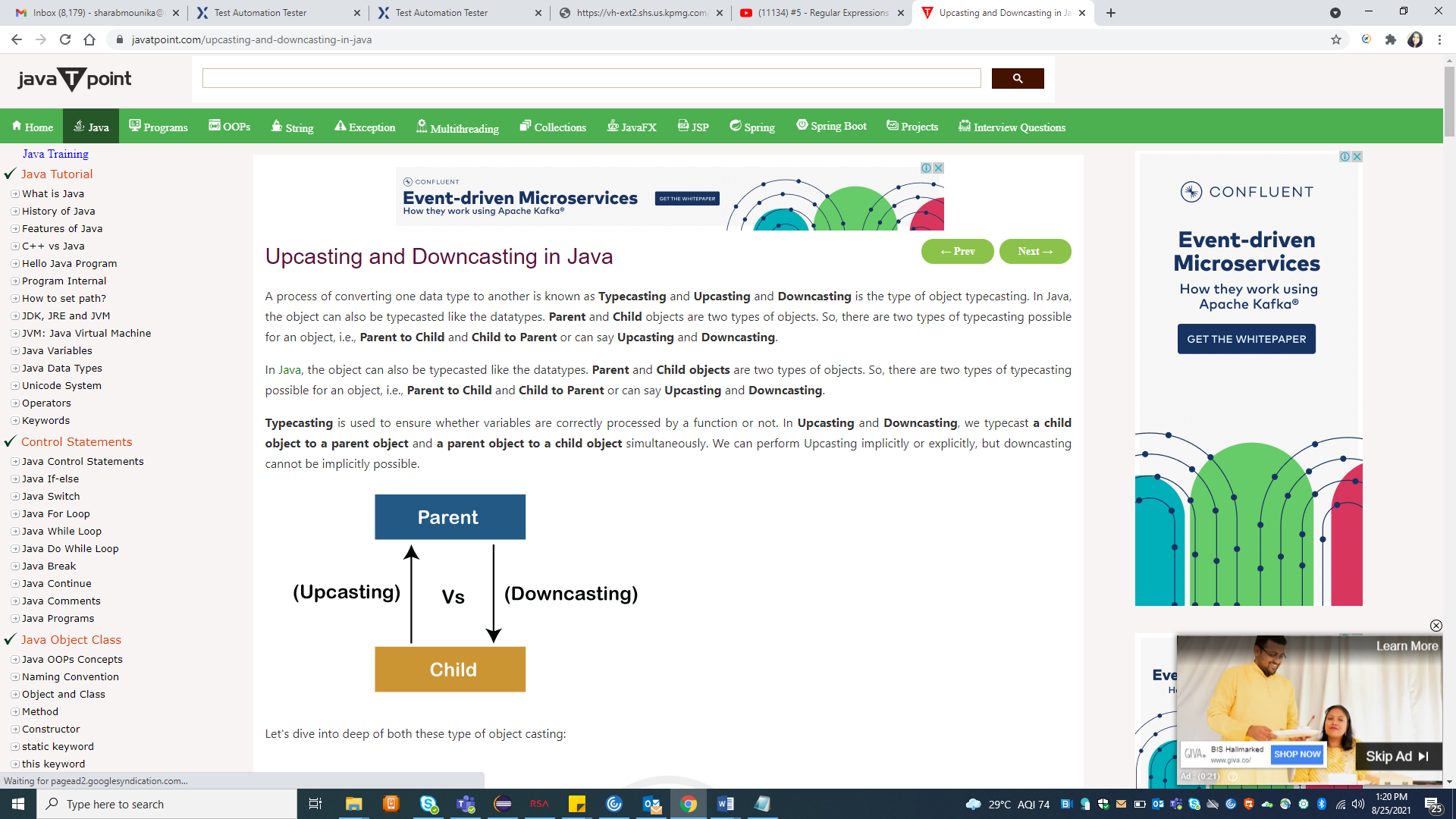


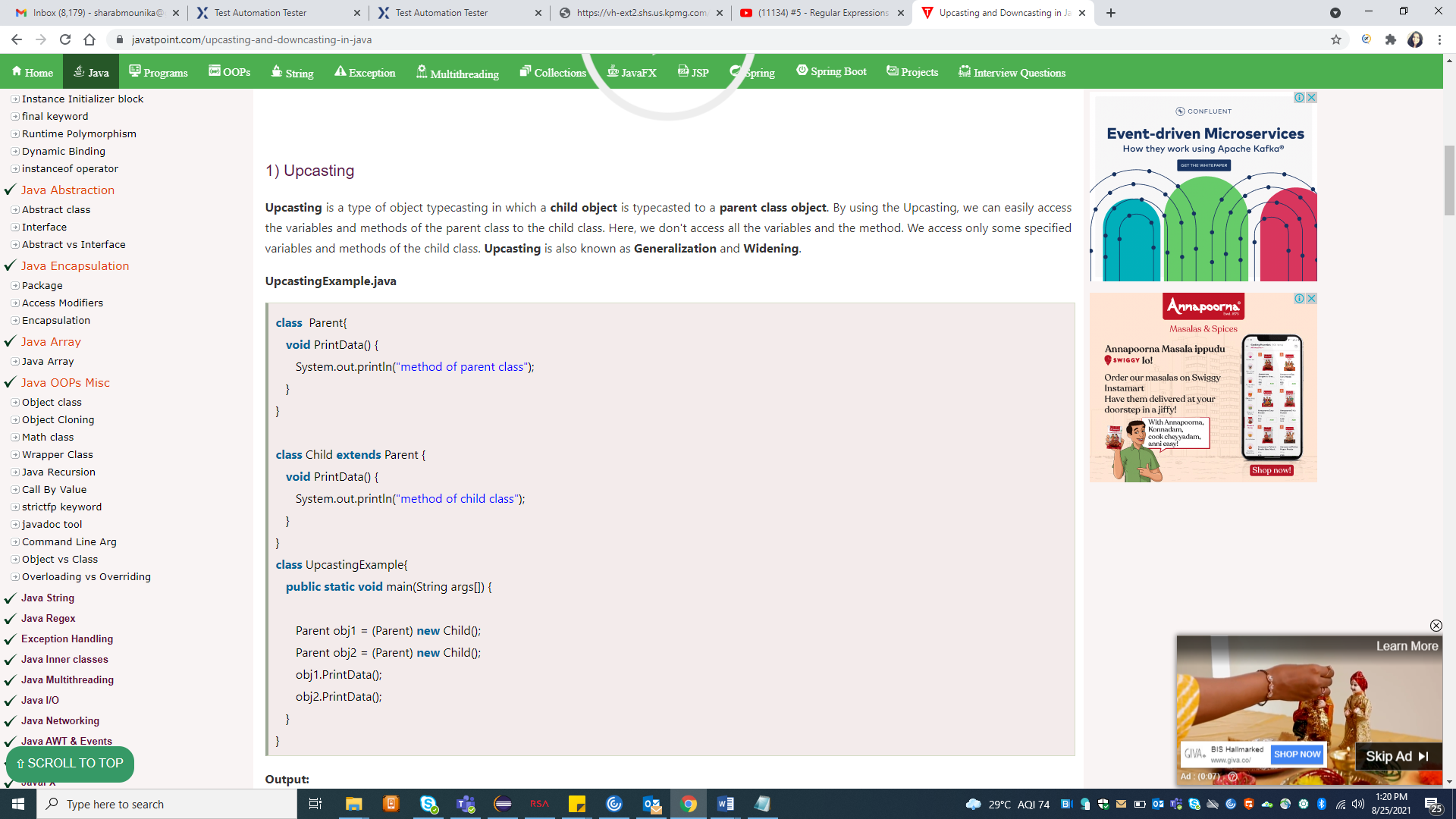


Upcasting









2) Downcasting

**Upcasting** is another type of object typecasting. In Upcasting, we assign a parent class reference object to the child class. In Java, we cannot assign a parent class reference object to the child class, but if we perform downcasting, we will not get any compile-time error. However, when we run it, it throws the **"ClassCastException"**. Now the point is if downcasting is not possible in Java, then why is it allowed by the compiler? In Java, some scenarios allow us to perform downcasting. Here, the subclass object is referred by the parent class.

Below is an example of downcasting in which both the valid and the invalid scenarios are explained:

**DowncastingExample.java**

//Parent class

**class** Parent {

    String name;

    // A method which prints the data of the parent class

**void** showMessage()

    {

        System.out.println("Parent method is called");

    }

}

// Child class

**class** Child **extends** Parent {

**int** age;

    // Performing overriding

    @Override

**void** showMessage()

    {

        System.out.println("Child method is called");

    }

}

**public** **class** Downcasting{

**public** **static** **void** main(String[] args)

    {

        Parent p = **new** Child();

        p.name = "Shubham";

        // Performing Downcasting Implicitly

        //Child c = new Parent(); // it gives compile-time error

        // Performing Downcasting Explicitly

        Child c = (Child)p;

        c.age = 18;

        System.out.println(c.name);

        System.out.println(c.age);

        c.showMessage();

    }

}

Exceptions:

1)Filenotfound exception - when you have given wrong path

2)Null pointer exception- when object or varaiable doesn't have life,

eg:you have created a driver object say "Public WebDriver driver" in your test but didn't inherited base class in which browser driver is invoked so in this case even though you have created driver variable but it doen't have any life

eg:you have created a driver object say "Public WebDriver driver" but didn't assigned driver object which has life

3)Invalid state exception ( when you are trying to perform some actions when element is disable)

4)NoSuchElement Exception - when xpath is incorrect

5)ArrayIndex out of bound Exception -

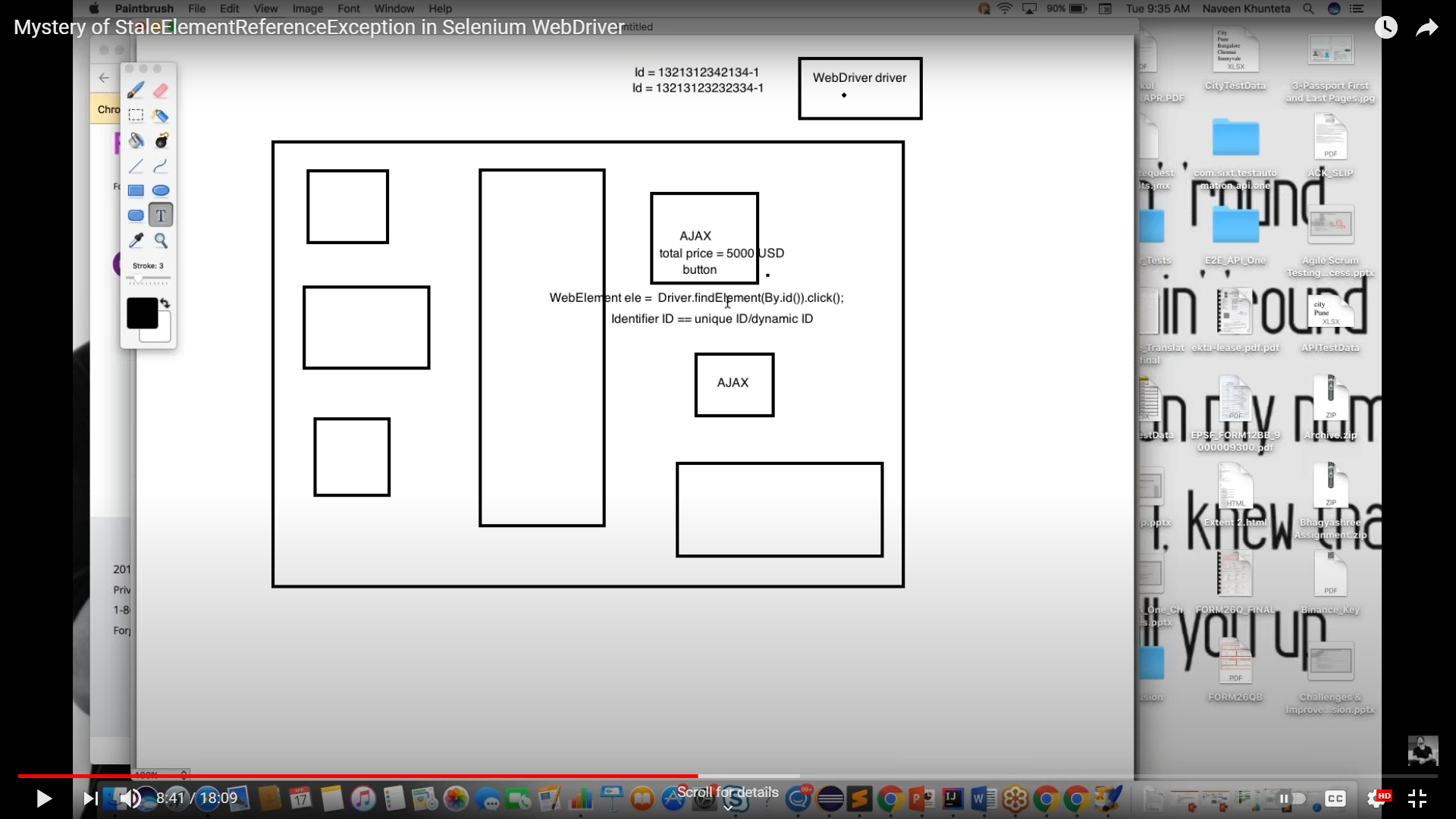
while creating you have given int[] a = new int[2] means it can store 2 elements and you are trying to access a[2], array index starts with 0 that means it will have only a[0] and a[1]

6)StaleElementReference Exception: stale means not fresh/ decayed or old

You performed an action bcoz of which page changes/reloads and DOM is rebuilt then previously found elements become stale

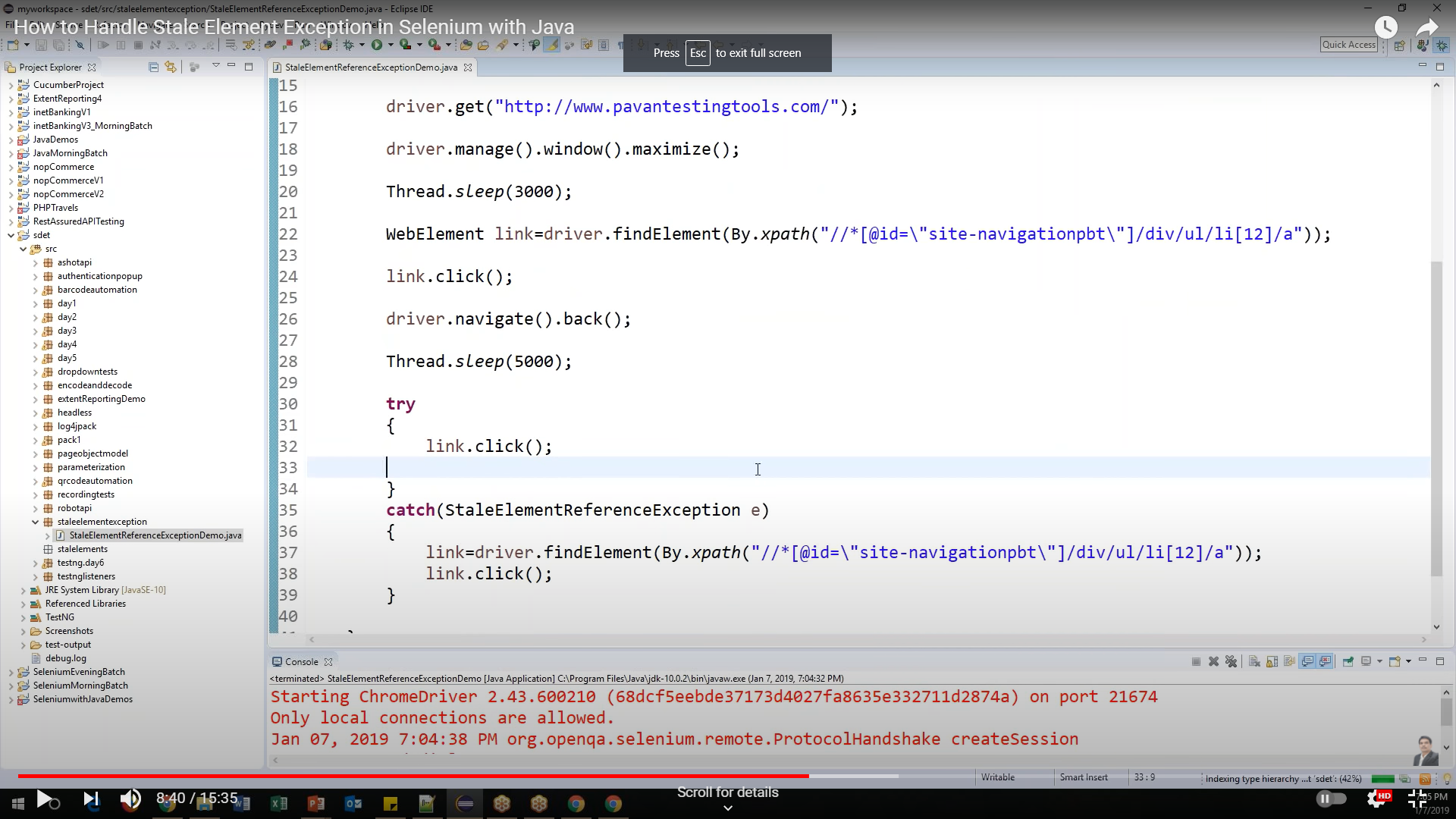
AJAX Components: when you refresh a webpage, only some components will be refreshed not the entire page e.g. when you refresh flight booking page only price components get changes rest remains same

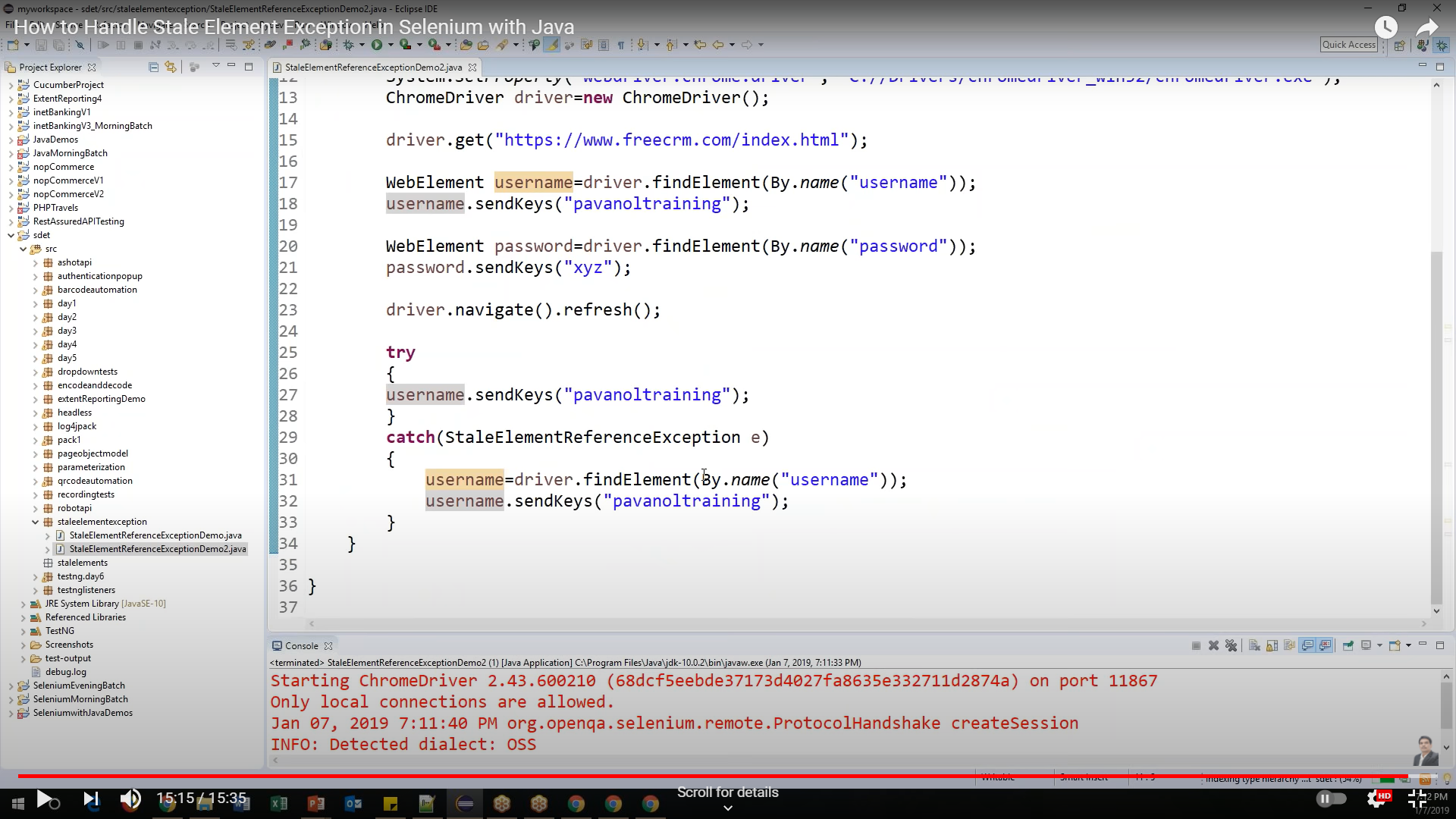
When a part of page or element which is ajax component gets refreshed then ID tagged to that element will be expired because of that we will throw StaleElementReference Exception, even though element is visible or present on dom



Solution:

1. First find the action which is making your element stale and write that in try and re-identify the element again and write that in catch block





**7)Element is not clickable at point exception:**

Element always try to click the element on middle of the element

Each browser has its own implementation so, sometimes click operation throws element is not clickable at specific coordinates. It mostly occurred in chrome and safari browser

* Chrome does not calculate the exact location of element
* Chrome always click in the middle of Element.
* Sometimes you will get this exception due to Sync issue also

**Solutions:**

1)Try writing different xpaths

2)try using mouse over and click

3)Try to click using Y coordinates

WebElement elementToClick = driver.findElement(By.xpath("Your xpath"));

// Scroll the browser to the element's Y position

((JavascriptExecutor)driver).executeScript("window.scrollTo(0,"+elementToClick.getLocation().y+")");

// Click the element

 elementToClick.click();

4) Try to click using X coordinates

WebElement elementToClick = driver.findElement(By.xpath("Your xpath"));

// Scroll the browser to the element's X position

((JavascriptExecutor)driver).executeScript("window.scrollTo(0,"+elementToClick.getLocation().x+")");

// Click the element

elementToClick.click();

Errors :

Assertion errors

parameters error - when method is expecting 3 but when you are sending only 2

Synchronization:

Selenium speed is way faster that application speed, so for making the webdriver and page loading (obejcts) to be at the same level, then only objects will be visible and won’t throw, no such element exceptions

[Alpha Testing](https://www.geeksforgeeks.org/?p=294073) is a type of software testing performed to identify bugs before releasing the product to real users or to the public. Alpha Testing is one of the user acceptance testing.

[Beta Testing](https://www.geeksforgeeks.org/?p=294134) is performed by real users of the software application in a real environment. Beta testing is one of the type of User Acceptance Testing.