CLASSNOTES: DAY 12-2
Today's schedule:
- src
  - main
  - test
    - java
      - com.cydeo
        - tests
          - base
          - tests (rest of our test packages)
        - utilities
          - Driver
          - ConfigurationReader
          - BrowserUtils
        - pages
- configuration.properties
- pom.xml

#1- Page Object Model Design Pattern
#2- Synchronization: ExplicitWaits

- The reason we have created different packages, and different utility classes, different type of file (.properties) are
  - re-usability
  - to organize our code
  - less code
  - efficient
  - easy to maintain
  - to centralize
  - avoid hard coding

#1- configuration.properties : to centralize and avoid hard coding some of the important test data
#2- TestBase: we can centralize and re-use setup/teardown methods, some important variables and objects etc.
#3- ConfigurationReader : to be able to repeatedly and easily read from our configuration.properties file
#4- Driver : to be able to instantiate our object in less line of code and also be able to steadily pass the same driver instance around in our project
#5- BrowserUtils : to centralize and easy to re-use some of the general utility methods
  - utility methods that are not specific to one page and can be applied in different pages

---

Page Object Model Design Pattern (POM)
#1- WHAT IS Page Object Model Design Pattern?
  - Creating java class for each page of the web application.
  - All of the related web elements to current page will be stored to its own .java class.
  - We can also store related re-usable utility methods in its "page" class as well.
- We have centralized pretty much everything re-usable such as:
  - important test data: in our configuration.properties
  - utility methods in utilities packages under different classes
    - Driver
    - ConfigurationReader

    - BrowserUtils
- The only thing that we didn't centralize or create a structure around is LOCATING WEB ELEMENTS.
- PAGE OBJECT MODEL DESIGN PATTERN SOLVES THIS ISSUE WE CURRENTLY HAVE.

- How are we going to implement POM Design Pattern?
#1- Every time we create a java class for a new page of our application, we will initialize our driver instance and the object that class.
  - PageFactory.initElements(driver, this);
  - This method basically initializes the driver instance and the object of the class.
  - After this line we will be able to use THE OBJECT of the class, to reach the avaiable web elements.
#2- Instead of using .findElement method we will use @FindBy annotation
regular email format :
  sometext@domain.com
  sometext@domain.gov
  sometext@domain.edu
- When we implement POM Design pattern we solve StaleElementReferenceException by default.
- How it works?
  - Every time we want to use WebElement located using @FindBy annotation, it will be re-located in the line we are using it.
  - Basically it will automatically do a "fressness check" which will eliminate the StaleElementReferenceException