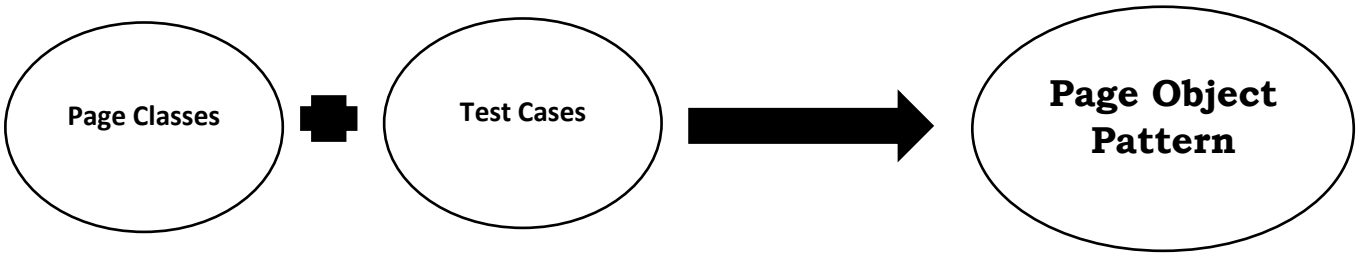
## POM(Page Object Model) NEDİR?

Web UI(user interface) elementleri için oluşturan, test otomasyonunda yaygın olarak kullanılan bir tasarım modelidir/desenidir(pattern).




✓Her bir web sayfasının bir page class'ı olmalı ve burada o sayfaya ait tüm WebElement'ler tanımlanmalıdır.

Page class örneği

```
public class AmazonHomePageV1 {

    4 usages
    private WebDriver ldriver;

    1 usage
    public AmazonHomePageV1(WebDriver driver) {
        this.ldriver = driver;

        PageFactory.initElements(ldriver, page: this);
    }


    1 usage
    @FindBy(id = "twotabsearchtextbox")
    WebElement txtSearch;

    1 usage
    @FindBy(id = "nav-link-accountList")
    WebElement accountAndList;

    1 usage
    @FindBy(id = "nav_prefetch_yourorders")
    WebElement orders;


    public void searchFor(String key) { txtSearch.sendKeys( ...keysToSend: key + Key

    1 usage
    public void navigateToOrders(){
        Actions actions = new Actions(ldriver);
        actions.moveToElement(accountAndList)
                .perform();

        WebDriverWait wait = new WebDriverWait(ldriver, Duration.ofSeconds(10));
        wait.until(ExpectedConditions.visibilityOf(orders)).click();

    }
}
```

WebElementler

Metodlar

Test Case örneği

```
public class Day02_C05_OPTIONALPageClassV2 {


    @Test
    public void test(){



        WebDriverManager.chromedriver().setup();
        WebDriver driver = new ChromeDriver();
        driver.manage().window().maximize();
        driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));

        driver.get("https://www.amazon.com.tr/");


        // Driver objemiz static degil ise bu driver objemizi page class imizda kullanmak adina constructor kullaniriz
        AmazonHomePageV1 homePage = new AmazonHomePageV1(driver);

        //homePage.searchFor("modem");

        homePage.navigateToOrders();


    }
}
```

## POM NEDEN GEREKLİDİR?

Akla gelmesi gereken iki kavram: maintenance + reusable

✓ Kod daha okunabilir ve anlaşılabilir olur.

✓ Büyüyen test paketinin bakımı daha kolaydır.

✓ Yeniden kullanılabilir(objects, classes, methods, data)

✓ Debuggingin daha kolay olması için bağımsız test senaryoları oluşturmamızı sağlar.

✓ Test execution süresini hızlandırır.

✓ Daha iyi, sürdürülebilir, daha hızlı ve anlaşılır olmasını sağlamak için tercih edilir.

## PAGE FACTORY CLASS?

✓Selenium WebDriver tarafından Page Object Modelini uygulamak için kullanılan bir class'tır. Page Class kullanımını daha basit ve kolay hale getirmek için kullanılabilir.

✓PageFactory Class'tan initElements() methodunu kullanarak page class objelerini ilk değeri atamamız gerekli.

✓ daha sonra aklımıza hemen

@FindBy (Tek kriterle arar)

@FindBys(İki kriterle arar(iki kriter de şart and operatörü gibi))

ve @FindAll (En az 1 kriter ister or operatörü gibi)

anotasyonları gelmeli.

```
public class SauceDemoPage {

    public SauceDemoPage() {
        PageFactory.initElements(Driver.getDriver(), page: this);
    }

    @FindBy(xpath ="//input[@id='user-name']")
    public WebElement username;

    @FindBy(xpath ="//input[@id='password']")
    public WebElement password;

    @FindBy(xpath ="//input[@id='login-button']")
    public WebElement loginButton;
```

## DRIVER CLASS ve SINGLETON ?

✓ tüm sınıflarda paylaşılan tek bir statik driver döndürür.

✓ Framework'deki tüm class'larda aynı driver objesini kullanmamıza izin verecektir

**✓ Singleton Pattern:**

Tekli kullanım, yalnızca örneği olabilecek şekilde geliştirilir, Singleton Driver kullanıcıyı WebDriver'ın gerekli olduğu tüm örnekler için aynı objeyi kullanmaya zorlayacaktır işte tam da bu yüzden Singleton Pattern bir classın farklı class'larda obje oluşturularak kullanımını engellemek için kullanılır.

Driver class'ındaki getDriver() method'unun obje oluşturularak kullanılmasını engellenmektedir.

## SINGLETON CLASS OLUŞTURMA?

✓Class'ta oluşturulan constructor private olmalı böylece farklı classlarda obje oluşturulamaz.

✓Class'da statik bir referans variable olmalı; her yerden kullanılabilir hale getirmek için static gereklidir.

✓ Class'ın bir kez başlatılıp başlatılmadığını kontrol etmesi gereken, class'ın objesi olarak return type static bir method declare edilmelidir.

## TESTNG NEDİR?

✓JUnit'ten sonra çıkmış olan bir test frameworküdür.

✓Açık kaynaklıkdır, yalnızca JAVA ile çalışır ve JDK 7 veya üstünü gerektirir.

✓ Kolay açıklama, gruplama, sıralama ve parametrelendirme, bağımlı test, paralel testler vs. yardımıyla daha güçlü ve esnek test senaryoları sağlayarak eski frameworkün sınırlamalarını ortadan kaldırır.

## TESTNG'NİN JUNIT'E GÖRE AVANTAJLARI NELERDİR?

✓ Uygulama için HTML raporları üretir.

✓ TestNG'de, JUnit'te bulunan @beforeclass ve @afterclass gibi bir kısıtlama yoktur.

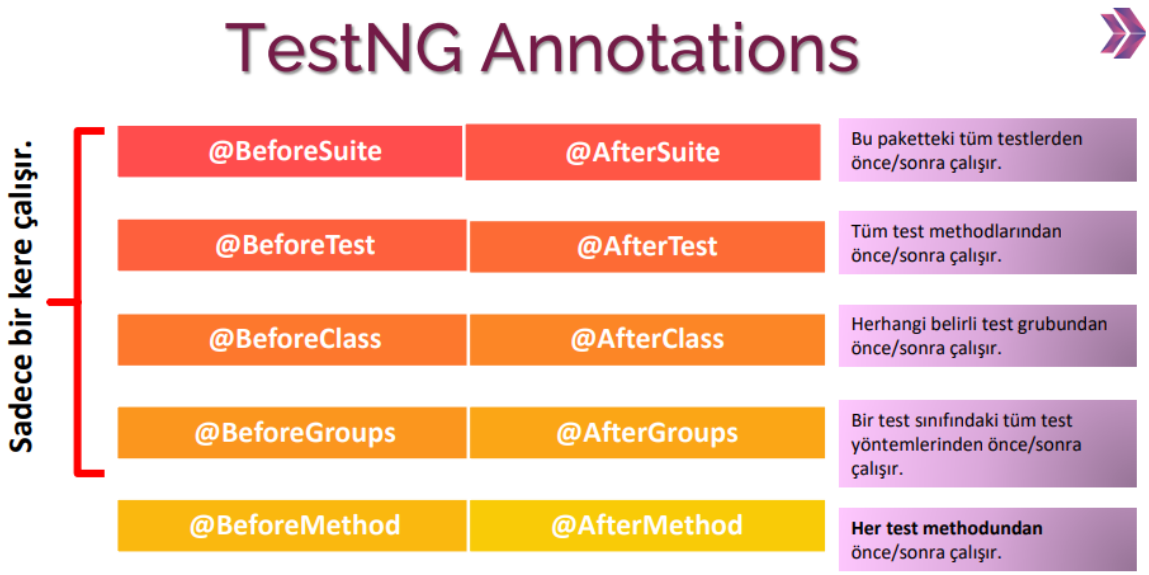
✓ TestNG, JUnit'te mümkün olmayan test senaryolarını kolayca gruplamanızı sağlar.

✓ TestNG, @Before/After suite, @Before/AfterTest ve Before/AfterGroup gibi üç ek düzeyi destekler.

✓ TestNG herhangi bir sınıfı genişletmez. TestNG Framework, her bir test senaryosunun diğer test senaryolarından bağımsız olduğu test senaryolarını tanımlamanıza olanak tanır.

✓ Belirli bir grubun test senaryolarını çalıştırmanıza izin verir. 'Smoke' and 'Regression' iki grup testlerin olduğunu düşünülürse; yalnızca 'Regression' çalıştırılmak istenirse bu sadece TestNG Framework'de mümkündür.

✓ Test senaryolarının paralel yürütülmesi, yani birden fazla test senaryosunun çalıştırılması yalnızca TestNG Framework'de mümkündür, yine Cross Browser Test'e imkan tanır.




## Annotations Helper Attributes?

### Priority

- Test annotationlara öncelik verilmesine yardımcı olurken, varsayılan öncelik 0 ile başlar ve testler artan sırada yürütülür.

- Test annotationda herhangi bir priority attribute yoksa priority sıfır kabul edilir.

- TestNG (default) olarak @Test annotationları alfabetik sıraya göre execute eder.

```
@Test(priority = 1)
void test1() { System.out.println("Test1 run"); }
```

**dependsOnMethods**

- İkinci test methodu birinci test methoduna bağımlı olmak istediğinde bu attribute kullanılır.

- Birinci test methodu başarısız olursa, birinci test methodundaki bağımlı method, yani ikinci test methodu çalışmayacaktır.

- Bir parametrede bir method veya birden çok method iletilebilir.

```
@Test
public void loginTest() {
    System.out.println("Login Tested Successfully");
}

@Test
public void homepageTest() {
    System.out.println("Home Page Tested Successfully");
}

@Test(dependsOnMethods = {"loginTest", "homepageTest"})
public void smokeTest() {
    System.out.println("Smoke Tests were done successfully");
}
```

**Enabled**

- Belirli test yöntemini mevcut suitte/class'ta çalıştırmak isteyip istemediğimizi belirlememize yardımcı olur. Gereksinimin/fonksiyonun sık sık değişmesi gibi bazı nedenlerden dolayı bazen birkaç test yapmak istemiyoruz ve o belirli fonksiyon için mevcut çalışmayı bozmak istemiyoruz.Bu durumlarda, bu özelliği @Test(enabled = false) olarak ayarlayarak söz konusu testi görmezden gelebilir/devre dışı bırakabiliriz.

- Aynı @Ignore annotation gibidir, o test case skip edilir yani görmezden gelinir.

```
@Test ( enabled = false)
public void test04() { System.out.println("test04"); }
```

**timeOut**

- Test için bir zaman aşımı değeri belirlemeye yardımcı olur (genellikle milisaniye olarak kullanılır). Test belirtilen zaman aşımı değerinden daha fazlasını alırsa, test başarısız olarak işaretlenir. - Yöntemin makul bir süre içinde geri döndüğünden emin olmak için bu zaman aşımını bir performans testi yapmak için kullanabiliriz.

```java
@Test(timeOut = 500)
public void timeTestPositive() throws InterruptedException {
    Thread.sleep( millis: 400);
    System.out.println("Time test method PASSED");

}
```

**description**

- Testle ilgili bilgileri açıklayan @Test annotationa eklenen ve genelde tek bir dizeden oluşan açıklamadır.

```java
@Test(description="This is testcase1")
public void testcase1()
{
    System.out.println("HR");
}
```

**Groups**

- Aynı işlevselliğe ait farklı test senaryolarını gruplamak için kullanılır.

```java
Unwabu
@Test(groups = {"smoke", "regression"})
public void loginTest(){

}
```

**TESTNG ASSERTIONS?**

**HARD ASSERTIONS**

Bir assert ifadesi başarısız olduğunda test execution'ı durdurur!!!

**SOFT ASSERTIONS**

Bir assertion failed olursa
test execution durmaz!!

**XML?** (Extensible Markup Language)

Hem insanlar hem de bilgisayar sistemleri tarafından kolayca okunabilen belgeler oluşturmak için kullanılan bir işaretleme dilidir.

**TESTNG.XML DOSYASININ AVATAJLARI?**

• Test yöntemlerinin paralel yürütülmesini sağlar.

• Bir test yönteminin başka bir test yöntemine bağımlılığına izin verir.

• Test yöntemlerimize öncelik vermemize yardımcı olur.

• Test yöntemlerinin test grupları halinde gruplandırılmasını sağlar.

• @Parameters annotation kullanarak test senaryolarının parametreleştirilmesini destekler.

• @DataProvider annotation kullanarak veriye dayalı testlerde yardımcı olur.

• Beklenen sonuçların gerçek sonuçlarla doğrulanmasına yardımcı olan farklı türde iddialara sahiptir.

• Test özetimizi daha iyi ve net bir şekilde anlamak için farklı HTML raporları, kapsam raporları vb. vardır.