### **Implicit wait vs explicit wait. Millal kumb?**

* 1. **Implicit Wait:** Globaalne seade. Käsib WebDriveril oodata kindlaksmääratud aja (nt 10 sekundit) *iga kord*, kui findElement elementi kohe ei leia. See on "rumal" ootamine, mis kehtib kõigile elementidele.
  2. **Explicit Wait (WebDriverWait):** Lokaalne, intelligentne ootamine. Ootab *kindlat tingimust* (nt elemendi nähtavust, klõpsatavust) konkreetses koodi kohas.
  3. **Millal:** Peaaegu alati tuleks eelistada **Explicit Wait**'i, kuna see on paindlikum ja ootab spetsiifilisi tingimusi (nt AJAXi laadimise järel). Implicit wait'i ei soovitata kasutada koos explicit wait'iga (vt allpool).

### **findElement vs findElements. Mis juhtub, kui elementi pole?**

* 1. findElement(By): Tagastab esimese leitud elemendi (kui WebElement). Kui elementi ei leita (ja implicit ootamine aegub), viskab see **NoSuchElementException** erindi.
  2. findElements(By): Tagastab *kõik* leitud elemendid nimekirjana (List<WebElement>). Kui ühtegi elementi ei leita, **viskab see erindi asemel tühja nimekirja** (List.size() == 0).

### **NoSuchElementException vs TimeoutException. Põhjuste vahe.**

* 1. **NoSuchElementException**: Viskab findElement, kui elementi ei leitud DOM-ist (isegi pärast implicit ooteaja möödumist).
  2. **TimeoutException**: Viskab WebDriverWait (explicit wait), kui määratud aeg (nt 10s) sai täis, aga oodatud *tingimus* (nt visibilityOfElementLocated) ei täitunud. Element võib DOM-is olemas olla (seega findElement leiaks selle), aga see pole näiteks nähtav.

### **StaleElementReferenceException. Põhjused ja tüüplahendused.**

* 1. **Põhjus:** Element, millele kood viitas, on "vananenud". Tavaliselt juhtub see siis, kui DOM on muutunud (nt leht on uuesti laetud või AJAX on osa lehest uuendanud). Muutuja mälus viitab elemendile, mida DOM-is enam ei eksisteeri.
  2. **Lahendus:** Kõige levinum lahendus on element **uuesti üles leida** (nt korrates findElement käsku) just enne sellega interakteerumist, sageli try-catch bloki sees või kasutades Page Object Model'i, mis otsib elemendi "laisalt" (lazy loading).

### **CSS vs XPath. Miks 80% juhtudest CSS on eelistatud?**

* 1. CSS-lokaatorid on enamasti eelistatud, sest nad on:
  2. **Kiiremad:** Eriti vanemates brauserites (nagu IE) parsis brauser CSS-i oluliselt kiiremini kui XPath-i.
  3. **Loetavamad:** CSS-i süntaks on sageli lühem ja kergemini mõistetav kui XPath-i oma.
  4. **Vähem "rabedad":** Kuigi XPath on võimsam (saab liikuda DOM-is üles, .., ja otsida teksti järgi), katab CSS enamiku vajadustest lihtsamalt.

### **Page Object Model. Põhiprintsiibid.**

* 1. Page Object Model (POM) on disainimuster, mille põhiprintsiibid on:
  2. **Vastutuse eraldamine:** Testiloogika (nt LoginTest.java) on lahus lehe ülesehitusest (nt LoginPage.java).
  3. **Üks klass lehe kohta:** Iga veebileht (või suur komponent) on esindatud eraldi klassiga.
  4. **Lokaatorid ja meetodid:** See klass sisaldab lehe elementide lokaatoreid (väljadena) ja meetodeid, mis nendega interakteeruvad (nt public void login(String user, String pass)).
  5. **Testid ei kasuta lokaatoreid:** Testid kutsuvad ainult leheobjekti meetodeid (nt loginPage.login(...)), mitte otse driver.findElement(...). See muudab testid loetavamaks ja hoolduse lihtsamaks.

### **Headless Chrome. Levinud piirangud ja vead.**

* 1. **Tuvastamine:** Mõned veebisaidid (eriti *bot*-kaitsega) suudavad *headless* režiimi tuvastada ja blokeerivad sisu.
  2. **Erinev käitumine:** Võib renderdada lehti erinevalt (nt vaikimisi aknasuurus on erinev) võrreldes tavalise (*headful*) Chrome'iga.
  3. **Piirangud:** Failide allalaadimine, hüpikakende haldamine või brauseri laienduste (extensions) kasutamine võib olla keerulisem või nõuda lisakonfiguratsiooni.

### **Miks implicit + explicit koos teevad üleliigseid viivitusi?**

* 1. Nende kooskasutamine tekitab **ettenägematuid ja pikki ooteaegu**. Kui WebDriverWait (explicit, nt 10s) hakkab elementi otsima, teeb see *polling*-tsükleid (nt iga 500ms tagant). Kui element kohe ei leidu, käivitub findElement (mida WebDriverWait sisemiselt kasutab) ja *implicit* ootamine (nt 10s). See tähendab, et *iga* explicit ootamise katse võib omakorda oodata 10 sekundit, muutes maksimaalse ooteaja 10s asemel palju pikemaks.

### **Kuidas jätta Chrome avatuks pärast testi (detach)?**

* 1. Selleks tuleb ChromeOptions klassile lisada eksperimentaalne seade "detach" väärtusega true.
  2. **Java:** options.setExperimentalOption("detach", true);
  3. **Python:** options.add\_experimental\_option("detach", True)

### **Java: == vs .equals() Stringide võrdlemisel. Millal true?**

* 1. **==**: Võrdleb **viiteid** (objekti mäluaadresse). See on true ainult siis, kui kaks muutujat osutavad *täpselt samale* objektile mälus.
  2. **.equals()**: Võrdleb **sisu** (kas tähemärgid on identsed).
  3. Stringide puhul == annab true peamiselt siis, kui võrreldakse kompileerimise ajal teadaolevaid literaale (mis paigutatakse "string pool"-i), nt String a = "test"; String b = "test"; (siin a == b). Kuid new String("test") == new String("test") on alati false.

### **Java: List vs ArrayList. Miks väljad/argumendid tüübiga List?**

* 1. See on "kodeeri liidese, mitte implementatsiooni vastu" printsiip.
  2. **List** on liides (interface), mis defineerib, *mida* nimekiri tegema peab (nt add, get, size).
  3. **ArrayList** on konkreetne klass (implementatsioon), mis *kuidas* seda teeb (dünaamilise massiiviga).
  4. Kasutades väljadel ja meetodi argumentidel List tüüpi, on kood **paindlikum**. Kui otsustad tulevikus ArrayList-i asemel kasutada näiteks LinkedList-i (millel on teistsugused jõudlusomadused), pead muutma ainult objekti loomise kohta (nt new LinkedList<>()), aga mitte ühtegi meetodit, mis seda nimekirja kasutab.

### **JUnit: millal assertEquals vs assertTrue(condition)?**

* 1. **assertEquals(expected, actual)**: Kasuta, kui võrdled **kahte väärtust** (nt stringid, numbrid, objektid). See annab palju parema ja informatiivsema veateate (nt "expected: <'Login Successful'> but was: <'Invalid Password'>").
  2. **assertTrue(condition)**: Kasuta, kui kontrollid tingimust, mille tulemus on **tõeväärtus** (boolean), nt assertTrue(list.isEmpty()) või assertTrue(element.isDisplayed()).

### **WebDriverWait: visibilityOfElementLocated vs presenceOfElementLocated. Vahe.**

* 1. **presenceOfElementLocated(By)**: Ootab, kuni element on **DOM-is olemas**. Element *ei pea* olema nähtav (nt see võib olla peidetud display: none või width: 0).
  2. **visibilityOfElementLocated(By)**: Ootab, kuni element on **DOM-is olemas *JA* nähtav** (st selle kõrgus ja laius on suuremad kui 0 ning see pole CSS-iga peidetud).

### **Maven Surefire: kuidas käivitada üks testiklass või -meetod CLI-st?**

* 1. Selleks kasutatakse -Dtest parameetrit:
  2. **Üks testiklass:** mvn test -Dtest=MinuTestiKlass
  3. **Üks testimeetod:** mvn test -Dtest=MinuTestiKlass#minuTestiMeetod

### **Lokaatoreid püsivamaks: millal kasutada data-\* atribuute ja miks?**

* 1. data-\* atribuute (nt data-testid="login-button") tuleks kasutada lokaatoritena, et muuta testid **stabiilsemaks** ja vastupidavamaks koodimuutustele.
  2. **Miks?** Arendajad muudavad sageli id-sid ja class-e (nt CSS-i refaktoorimisel või kujunduse muutmisel), mis lõhub testid. data-\* atribuudid on spetsiaalselt mõeldud testimise ja JavaScripti haakimispunktideks, mida *ei* muudeta koos stiilidega. Nende kasutamine eraldab testimise loogika rakenduse visuaalsest implementatsioonist.