המתנה

קוד אוטומציה חייב לכלול המתנות. הסיבה היא פשוטה - קוד אוטומציה, שהוא קוד חיצוני ואינו חלק מהקוד של המערכת הנבדקת, תמיד ירוץ בקצב שונה מהמערכת אותה הוא בודק. ליתר דיוק, קוד אוטומציה כמעט תמיד ירוץ **מהר יותר** מהמערכת אותה הוא בודק. לכן קוד אוטומציה חייב לדעת להמתין. להמתין לפעולות שתסתיימנה להתבצע במערכת הנבדקת. דוגמא טריביאלית לכך ניתן לראות בתהליך הכניסה למערכת - Login. לאחר הזנת שם משתמש וסיסמא, על האוטומציה להמתין לסיום תהליך הכניסה למערכת ועליית מסך הבית. דוגמא נוספת - לאחר הזנת כתובת URL האוטומציה צריכה להמתין עד שדף ה- Web יסיים להיטען לפני שיהיה אפשר לבצע פעולות. בנוסף, מערכות Web מודרניות כוללות פעולות אסינכרוניות שמתרחשות גם לאחר סיום טעינת דף ה-Web (מנגנון AJAX).

כמעט לאחר כל פעולה באוטומציה קיים צורך להמתין למערכת הנבדקת. לכן חשוב מאוד להכיר את סוגי ההמתנה, להבין אותם לעומק על מנת שנהיה מסוגלים לבחור את מנגנון ההמתנה המתאים לכל מצב בו אנו נדרשים לטפל.

במדריך זה נדון בסוגים שונים של המתנות ב- Selenium Driver. התוכן הכתוב בפוסט זה רלבנטי גם עבור גרסה 3.141.59 וגם עבור גרסא 4 (שבעת כתיבת שורות אלה הינה 4 beta)

המתנות ב- Selenium מתחלקות ל- 4 חלקים:

- המתנה מובנית (שאינה בשליטתנו)
- Implicit Wait
- Explicit Wait
- Fluent Wait

כאשר ללא ספק, Explicit Wait הוא המנגנון המרכזי של Selenium, ובו נעסוק במרבית הפוסט.

בחלק האחרון של הפוסט נדון במקרים בהם סוגי ההמתנה של Selenium אינם מתאימים, ונציג סוג המתנה נוסף, שאינו חלק ממנגנון Selenium אותו אנו מקבלים בזמן ההתקנה.

המתנה מובנית שאינה בשליטתנו

תהליך האוטומציה, בגדול מאוד מאוד, פועל בצורה הבאה- קוד האוטומציה שלנו מפעיל את כלי האוטומציה (נניח ChromeDriver או GeckoDriver) והוא בתורו מפעיל את המערכת הנבדקת. לאחר ביצוע הפעולה המבוקשת על ידי כלי האוטומציה, מוחזרת השליטה לקוד האוטומציה על מנת לבצע את הפעולה הבאה בתור.

נסתכל כדוגמא על פעולת Get:

```
1    driver.get("http://www.google.com");
```

WebDriver לא יחזיר את השליטה לקוד האוטומציה עד לסיום הטעינה של הדף. פעולת המתנה זו נועדה לבצע סנכרון בסיסי, אשר אינו בשליטתנו. חשוב לציין בשלב זה **שעצם העובדה שהדף נטען לא אומר שכל הרכיבים בו סיימו להיטען**. זהו מנגנון ה- AJAX שציינו קודם, והוא מחייב אותנו לזהירות מרובה בעת פיתוח תסריטים אוטומטיים.

מבין פעולות נוספות שכוללות המתנה מובנית שאינה בשליטתנו ניתן להביא כדוגמא את פעולת click ופעולת sendKeys. פעולות אלה נחשבות פעולות סינכרוניות- WebDriver מבטיח למפתח האוטומציה שאין צורך לבנות מנגנון שמבטיח שהפעולות תתבצענה, ואכן Selenium מסונכרן עם המערכת הנבדקת ומוודא שהפעולה הסתיימה לפני שהקוד חוזר בחזרה לשליטת האוטומציה לבצע את הפקודה הבאה בתור.

Thread.sleep()

אולי זה המקום הנכון להתחיל. על פניו, מאוד הגיוני. קוד האוטומציה צריך להמתין למערכת הנבדקת, אז נכתוב המתנה מפורשת. אם לדוגמא ספרנו עשרות פעמים ואנחנו משוכנעים שתהליך ה- Login לא יארך מעבר ל-2 שניות, אם נבנה המתנה של 4 שניות - בטוח נהיה מכוסים.

הוא גם קיים ב- Selenium עצמו בצורה הבאה:

```
1    driver.manage().timeouts().wait(3000);
```

כמעט ולא קיים פרויקט שבחנתי בשנים האחרונות, שאין בו שימוש מסיבי ב- Thread.sleep. **זו טעות קלאסית, והיא אחד הגורמים המרכזיים לטסטים שאינם יציבים**.

למה?

א. כי ההנחה שזמן ההמתנה, שהספיק מעל ומעבר למחשב בו פותח תסריט האוטומציה הוא מספק למחשב או שרת שבו האוטומציה תרוץ באופן קבוע היא הנחיה שגויה ולרוב מתבררת כלא נכונה. זה מקור ראשון לטסט flaky - לפעמים עובד ולפעמים לא.

ב. הטסט שמפותח היום סביר להניח שירוץ באופן קבוע במשך חודשים ושנים. ההנחה שזמני התגובה של המערכת היום הם אותם זמני תגובה של המערכת בעתיד היא הנחה שגויה. ככל שמערכת גדלה, שינויים בזמני התגובה שלה הם יותר נפוצים.

ג. כי זמני תגובה מספיקים לכל המצבים, או "מעל ומעבר" לנדרש עשויים להאריך את הטסט ללא צורך. אם נניח לצורך הדוגמא שזמן התגובה הממוצע של המערכת הוא 1 שניה, אבל על מנת להתחשב בסעיפים א' וב' נבצע המתנה של 4 שניות, "כדי להיות בטוחים", אז הפעולה תתארכנה שלא לצורך בממוצע בעוד 3 שניות.

אז איך זה קורה? למה השימוש בו כל כך נפוץ?

א. חוסר ידע, בעיקר אצל מפתחי אוטומציה צעירים, אבל לא רק.

ב. לפעמים מפתה כחלק מתיקון תסריט flaky שפשוט מסרב להסתדר להוסיף "sleep קטן" והכל יהיה בסדר.

ג. לפעמים במנגנונים הקיימים של המתנה אנחנו לא מצליחים לייצר את ההמתנה שאנחנו צריכים (זה יכול לקרות, אבל זה אמור להיות אירוע נדיר).

ד. קל יותר לבצע sleep מאשר לחפש את מנגנון ההמתנה הנכון.

פוסט זה בא לטפל בסעיפים א' - ג'.

לסיכום, שימוש ב sleep צריך להיות המוצא האחרון, לאחר שכל שאר המנגנונים המוכרים נכשלו, לאחר שחקרתן ובדקתן והתייעצתן בפורומים. שימוש ב- Thread.sleep הוא מה שקוראים לו בעגת המתכנתים Code Smell. זה אומר שאם נבחן את הטסט והקוד שנכתב סביבו (page objects, utilities וכדומה) נגלה עוד קטעי קוד שכתובים בצורה בעייתית ועשויים לגרום לתקלות ותסריטים לא יציבים.

המתנה מרומזת - Implicit Wait

המתנה מרומזת, או בשמה Implicit Wait היא **הנחיה כללית** לדגום את דף ה- Web, או ליתר דיוק את ה- DOM, כל פעם שמחפשים אלמנט על הדף ובכל פעולה שמבצעים עליה. פעולה Implicit Wait נועדה לתת זמן למערכת הנבדקת להסתנכרן לקוד האוטומציה, מבלי לבקש זאת בצורה מפורשת כל פעם שישנה סכנה שהאוטומציה לא תהיה מסונכרנת לקוד המערכת.

ברירת המחדל ב- Selenium היא שאין Implicit Wait (או במילים אחרות ערך המשתנה implicitWait הוא 0).

כלומר, יש לבקש באופן מפורש להפעיל את המנגנון במידה ומעוניינים להשתמש בו. הפעלת המנגנון מתבצעת לדוגמא כך:

```
1    driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
```

נדגיש, **הגדרת Implicit Wait תגרום ל- WebDriver לדגום את ה- DOM ולהמתין בכל חיפוש אלמנט את הזמן שהוגדר עד למציאתו.** כמובן שאם האלמנט יימצא לפני הזמן שהוגדר, הפעולה תתבצע וקוד האוטומציה לא ימתין את מלוא הזמן שהוגדר לו ב- implicitWait.

Implicit Wait מצד אחד מעלה את יציבות התסריטים, במקומות בהם אין סנכרון טוב מספיק בין קוד האוטומציה למערכת הנבדקת. המתנה אוטומטית בכל פעולה מעלה את היציבות ותחסוך נפילות.
יחד עם זאת, שימוש ב- Implicit Wait עלול להאריך מאוד את זמן הריצה, לעתים בצורה בלתי סבירה. דמיינו תסריט בעל 20 פעולות אשר זמן ה- Implicit Wait שלו הוא 30 שניות. במקרה הגרוע התסריט עלול לארוך עד 600 שניות (30 שניות המתנה לפעולה) - כלומר 10 דקות. מצבים כאלה עשויים לקרות ככל שקוד האוטומציה הופך למורכב יותר. כמובן שאין חובה שתסריט ייקח את הזמן המקסימלי שהוא עשוי לקחת. כאשר Selenium יזהה את האלמנט הוא יחזיר את השליטה לכלי האוטומציה לביצוע הפעולה גם אם לא עבר הזמן המצויין. יחד עם זאת, סטטיסטית, שימוש implicitWait כמעט תמיד יגרום לבדיקות איטיות יותר בצורה משמעותית. בנוסף, מצבים קלאסיים בהם מתרחשת המתנה ארוכה מהנדרש בדרך כלל קורים במצבים בהם מבוצע שילוב בין Implicit Wait ל- Explicit Wait. מכיוון ש- implicitWait הוא הגדרה גלובלית, יש להזהר מתי ואיך משתמשים בה.

המתנה מפורשת - Explicit Wait

<u>המקרה הנפוץ</u>

להלן דוגמא להמתנה ב- Selenium לאלמנט שיהיה לחיץ:

```
1    WebDriverWait wait = new WebDriverWait(driver, 30);
2    WebElement someElement =
3    wait.until(ExpectedConditions.elementToBeClickable(By.id("Some Id")));
```

נסביר את הקוד:
שורה 1: הגדרת אובייקט חדש מסוג WebDriverWait, עם זמן המתנה מקסימלי של 30 שניות.
שורה 2-3: הפעלת מתודת until. מתודה זו מבצעת את פעולת ההמתנה בפועל. היא מחזירה במקרה זה אובייקט מסוג WebElement שעומד בתנאי שנשלח אליה במחלקת ExpectedConditions.

כחלק מ- WebDriver מוגדרים סוגים שונים של המתנה. ישנם למעלה מ- 20 סוגים של המתנות, שאמורים לכסות את רוב המקרים הנפוצים. לחלק מההמתנות נדרש לספק אובייקט By (כמו בדוגמא הנ"ל) ובחלק מסופק אובייקט WebElement. דוגמאות:

המתנה לאלמנט שיהיה לחיץ	`elementToBeClickable`
המתנה לאלמנט שיהיה נראה על המסך	`visibilityOf`
המתנה לאלמנט שייבחר	`elementToBeSelected`

את רשימת ההמתנות המלאה ניתן למצוא בתיעוד Javadoc של אובייקט ExpectedConditions כאן.

למעשה, התחבירים של סוגי ההמתנות השונים המופיע במקרה הנפוץ והדוגמאות המופיעות כאן ובקישור המצורף הם מקרים פרטיים של ארבעה סוגים עיקריים. על מנת להקל על המשך קריאת הפוסט, רגע לפני שנציג את ארבעת הסוגים, נעשה השלמה זריזה למושג ב- Object Oriented Programming שנקרא **Anonymous Class**.

Anonymous Class

מחלקה אנונימית היא מחלקה שאינה כתובה בקובץ java משלה כמו מחלקות אחרות, אלא היא מחלקה שמוגדרת כחלק ממתודה או מחלקה אחרת. לא ניתן ליצור מופעים רבים שלה אלא מופע חד פעמי. במסגרת המופע החד פעמי נממש את המתודות בהם נרצה להשתמש במסגרת המופע שנוצר.
עיקר השימוש ב- Anonymous Class מופיע כמימוש של interface.
למה זה טוב? בעיקר כאשר כל מימוש של interface שונה ממימוש אחר.
על מנת להמחיש את הרעיון, נשתמש ב- interface של Selenium שנקרא ExpectedCondition:

```
1    public interface ExpectedCondition<T> extends Function<WebDriver, T> {
2    }
```

נשים לב שה - interface הזה הינו interface ריק, ומלבד קיבוע ה- Function לצורכי WebDriver המתודות שלו הן מתודות אותן הוא מקבל בירושה מה- interface אבא שלו - Function:

```
1     public interface Function<F, T> extends java.util.function.Function<F, T> {
2        T apply(@NullableDecl F var1);
3
4        boolean equals(@NullableDecl Object var1);
```

```
5       }
```

בעצם המימוש של ExpectedCondition נועד לקבע את Function לצורכי Selenium (שימו לב להגדרה של WebDriver כחלק מ- Function).

איך ייראה מימוש לדוגמא?

```
1     Condition cond = new ExpectedCondition<WebElement>() {
2             public WebElement apply(WebDriver driver) {
3                 try {
4                     return driver.findElement(locator);
5                 }
6             }
7         };
8     }
```

נסביר את הקוד:

שורה 1: לתוך משתנה מסוג Cond (שהוא interface בעצמו) נייצר אובייקט חדש מסוג ExpectedCondition.

מכיוון ש- ExpectedCondition הוא interface ולא יכול להיות מאותחל ללא מימוש המתודות עליהן הוא מצהיר, שורות 2-6 מממשות את ה- interface

שורות 2-6: מימוש ה- interface כחלק מהגדרת המשתנה cond. במקרה שלנו - ביצוע findElement ללוקייטור מסויים.

לסיכום, מחלקת ExpectedCondition נועדה להיות מחלקה אנונימית שתממש את מתודת apply, אשר תהווה את תנאי ההמתנה למתודת until של מחלקת WebDriverWait. איך הכל משתלב? נראה זאת בהמשך.

נדון עכשיו בארבעת סוגי המימוש ExpectedCondition שמומשו על ידי Selenium.

ExpectedCondition <WebElement>

מנגנון זה מגדיר פעולות המתנה שקשורות לאובייקט WebElement. בדוגמא שלפנינו - המתנה לאלמנט שיופיע על המסך. במידה והאלמנט מופיע, האלמנט יוחזר. במידה והוא לא מופיע, יוחזר הערך null וזה יהיה הסימן למתודת until של מחלקת WebDriverWait שהתנאי לא מומש, ויש לשוב ולהפעיל את מתודת apply עד

לעמידה בתנאי או פקיעת הזמן המירבי להמתנה. המתודה הבאה היא דוגמא למקרה כזה, ונלקחה מתוך מחלקת ExpectedConditions של Selenium:

```
1     public static ExpectedCondition<WebElement> visibilityOfElementLocated(final
2     By locator) {
3        return new ExpectedCondition<WebElement>() {
4            public WebElement apply(WebDriver driver) {
5                try {
6                    return
7     ExpectedConditions.elementIsVisible(driver.findElement(locator));
8                } catch (StaleElementReferenceException var3) {
9                    return null;
10               }
11           }
12
13           public String toString() {
14               return "visibility of element located by " + locator;
15           }
16       };
17    }
```

נסביר את הקוד:

שורה 1: המתודה מקבלת locator ומחזירה אובייקט מסוג ExpectedCondition (שכאמור הוא interface שימומש כ- anonymous class וישמש את מתודת until בקביעה האם התנאי מומש).

שורה 3: יצירת אובייקט חדש של ExpectedCondition ובהמשך מימוש מתודת apply, שהיא המתודה המרכזית של ה- interface.

שורה 4: מימוש מתודת apply הנדרשת על ידי ExpectedCondition

שורה 7: הפעלת מתודה פנימית שבודקת האם האלמנט מופיע. במידה וכן, המתודה תחזיר את האלמנט. במידה ולא, יחזור הערך null אשר יסמן למתודת wait של מחלקת WebDriverWait שיש להמשיך ולדגום את הפעולה לפי הפרמטרים שהוגדרו - זמן מקסימלי ומרווח בין דגימות.

שורות 8-9: במידה ובזמן בדיקת האלמנט נפלנו על ה - Exception הספציפי שהוגדר, יהיה זה סימן כאילו לא נמצא האלמנט על הדף ועלינו להמשיך ולבצע את מנגנון ההמתנה כפי שהוגדר. אם לא היו מוגדרות שורות 8-9 תהליך ההמתנה היה נפסק מכיוון שהתרחש Exception.

ExpectedCondition <WebDriver>

מנגנון זה מגדיר פעולות רלבנטיות ל- WebDriver. בדוגמא שלפנינו, מדובר על המתנה למעבר ל- frame חדש. מכיוון שנושא ניהול ה- frames ב - Selenium הינו בטיפולו של WebDriver ולא של WebElement, הגדרת ה - ExpectedCondition במקרה הזה תהיה על ידי WebDriver. במקרה שלנו, במידה וה- frame זמין וניתן לבצע מעבר אליו, יוחזר ה- WebDriver לאחר ביצוע תהליך המעבר ל- frame. במידה ולא, יוחזר הערך null וזה יהיה הסימן למתודת until של מחלקת WebDriverWait שהתנאי לא מומש, ויש לשוב ולהפעיל את מתודת apply עד לעמידה בתנאי או פקיעת הזמן המירבי להמתנה. גם המתודה הזו, כמו קודמתה, היא נלקחה מתוך מחלקת ExpectedConditions של Selenium:

```
1     public static ExpectedCondition<WebDriver>
2     frameToBeAvailableAndSwitchToIt(final String frameLocator) {
3        return new ExpectedCondition<WebDriver>() {
4            public WebDriver apply(WebDriver driver) {
5                try {
6                    return driver.switchTo().frame(frameLocator);
7                } catch (NoSuchFrameException var3) {
8                    return null;
9                }
10           }
11
12           public String toString() {
13               return "frame to be available: " + frameLocator;
14           }
15       };
16    }
```

נסביר את הקוד:

שורות 1-2: המתודה מקבלת String שמייצגת את הלוקייטור של ה- frame ומחזירה אובייקט מסוג ExpectedCondition (שכאמור הוא interface שימומש כ- anonymous class).

שורה 3: יצירת אובייקט חדש של ExpectedCondition ובהמשך מימוש מתודת apply, שהיא המתודה המרכזית של ה- interface.

שורה 6: הפעלת מתודה פנימית שמנסה לעבור ל- frame המבוקש. במידה והצלחנו לבצע את המעבר, המתודה תחזיר את ה- WebDriver שמצביע ל- frame הרלבנטי. במידה ולא, יחזור הערך null אשר יסמן למתודת wait של מחלקת WebDriverWait שיש להמשיך ולדגום את הפעולה לפי הפרמטרים שהוגדרו - זמן מקסימלי ומרווח בין דגימות.

שורות 7-8: במידה ובזמן בדיקת ה- frame נפלנו על ה - Exception הספציפי שהוגדר, יהיה זה סימן כאילו לא הצלחנו לבצע את החלפת ה- frame ועלינו להמשיך ולבצע את מנגנון ההמתנה כפי שהוגדר.

ExpectedCondition <Boolean>

מנגנון זה מגדיר המתנה למצב שמתרחש על גבי המסך / אלמנט / driver. הערך המוחזר ממנו הוא מסוג בוליאני - true / false. בדוגמא שלפנינו, נמתין לטקסט מסויים שיופיע על ה- WebElement שאנחנו משתמשים. במידה וכן, יוחזר הערך true. במידה ולא, יוחזר הערך null וזה יהיה הסימן למתודת until של מחלקת WebDriverWait שהתנאי לא מומש, ויש לשוב ולהפעיל את מתודת apply עד לעמידה בתנאי או פקיעת הזמן המירבי להמתנה. גם המתודה הזו, כמו קודמותיה, היא נלקחה מתוך מחלקת ExpectedConditions של Selenium:

```
1     public static ExpectedCondition<Boolean> textToBePresentInElement(final
2     WebElement element, final String text) {
3        return new ExpectedCondition<Boolean>() {
4            public Boolean apply(WebDriver driver) {
5                try {
6                    String elementText = element.getText();
7                    return elementText.contains(text);
8                } catch (StaleElementReferenceException var3) {
9                    return null;
10               }
11           }
12
13           public String toString() {
14               return String.format("text ('%s') to be present in element %s",
```

```
15    text, element);
16          }
17       };
18    }
```

נסביר את הקוד:

שורות 1-2: המתודה מקבלת אלמנט לבדוק עליו ו- String שמייצג את הטקסט שאמור להופיע עליו, ומחזירה אובייקט מסוג ExpectedCondition (שכאמור הוא interface שימומש כ- anonymous class).

שורה 3: יצירת אובייקט חדש של ExpectedCondition ובהמשך מימוש מתודת apply, שהיא המתודה המרכזית של ה- interface.

שורות 6-7: הפעלת מתודה פנימית שבודקת האם הטקסט מופיע על האלמנט. במידה וכן, נחזיר את הערך true. במידה ולא, יחזור הערך null אשר יסמן למתודת wait של מחלקת WebDriverWait שיש להמשיך ולדגום את הפעולה לפי הפרמטרים שהוגדרו - זמן מקסימלי ומרווח בין דגימות.

שורות 8-9: במידה ובזמן בדיקת המצאות הטקסט על האלמנט נפלנו על ה - Exception הספציפי שהוגדר, יהיה זה סימן כאילו לא הצלחנו לבצע את החלפת ה- frame ועלינו להמשיך ולבצע את מנגנון ההמתנה כפי שהוגדר.

ExpectedCondition <Alert>

מנגנון זה מגדיר המתנה למצבים הקשורים ב- alert. הערך המוחזר ממנו הוא alert. בדוגמא שלפנינו, נמתין להופעת alert על המסך. במידה ויופיע ה- alert, הוא יוחזר. במידה ולא, יוחזר הערך null וזה יהיה הסימן למתודת wait של מחלקת WebDriverWait שהתנאי לא מומש, ויש לשוב ולהפעיל את מתודת apply עד לעמידה בתנאי או פקיעת הזמן המירבי להמתנה. גם מתודה הזו, כמו קודמותיה, היא נלקחה מתוך מחלקת ExpectedConditions של Selenium:

```
1     public static ExpectedCondition<Alert> alertIsPresent() {
2        return new ExpectedCondition<Alert>() {
3            public Alert apply(WebDriver driver) {
4                try {
5                    return driver.switchTo().alert();
6                } catch (NoAlertPresentException var3) {
7                    return null;
8                }
```

```
9           }
10
11          public String toString() {
12              return "alert to be present";
13          }
14      };
15  }
```

נסביר את הקוד:

שורה 1: המתודה מחזירה אובייקט מסוג ExpectedCondition (שכאמור הוא interface שימומש כ-anonymous class).

שורה 2: יצירת אובייקט חדש של ExpectedCondition ובהמשך מימוש מתודת apply, שהיא המתודה המרכזית של ה- interface.

שורות 3-5: הפעלת מתודה פנימית שבודקת האם מופיעה alert על המסך. במידה וכן, נחזיר את ה- Alert. במידה ולא, יחזור הערך null אשר יסמן למתודת wait של מחלקת WebDriverWait שיש להמשיך ולדגום את הפעולה לפי הפרמטרים שהוגדרו - זמן מקסימלי ומרווח בין דגימות.

שורות 6-7: במידה ובזמן בדיקת המצאות הטקסט על האלמנט נפלנו על ה - Exception הספציפי שהוגדר, יהיה זה סימן כאילו לא הצלחנו לעבור ל- Alert ועלינו להמשיך ולבצע את מנגנון ההמתנה כפי שהוגדר.

ExpectedCondition - המקרה הכללי

כפי שכבר ראינו, אובייקט ExpectedCondition הוא interface על בסיס מוגדר (נניח WebElement או WebDriver) שמכיל מתודה מרכזית הרלבנטית לנו: apply. למעשה, מכיוון ש- ExpectedCondition הוא template (כלומר יכול לממש כל מחלקה. בדוגמאות הקודמות ראינו מימוש ל- WebElement, WebDriver, Alert ו- Boolean) אנחנו יכולים להגדיר בעצמנו את הפעולות שתתבצענה על ידי ExpectedCondition ולהגדיר המתנה בעצמנו. בדוגמא הבאה נרצה לבצע חיפוש בגוגל ולוודא ששורת החיפוש שלנו תופיע ב- title של דף התוצאות. ההמתנה תחזיר את ה- String אותו הזמנו בשורות החיפוש, והתווסף ל- title הקבוע של Google. מחלקת ה- ExpectedCondition שלנו תיראה כך:

```
1     public static ExpectedCondition<String> addedTitleIs(final String title) {
2        return new ExpectedCondition<String>() {
3            private String currentTitle = "";
```

```
4
5           public String apply(WebDriver driver) {
6               this.currentTitle = driver.getTitle();
7               if (!title.contains(this.currentTitle))
8                   return null;
9               return this.currentTitle.substring(0, currentTitle.indexOf('-'));
10          }
11
12          public String toString() {
13              return String.format("title to be \"%s\". Current title: \"%s\"",
14                      title, this.currentTitle);
15          }
16      };
17  }
```

נסביר את הקוד:

שורה 1: מכיוון שפונקצית until של WebDriverWait מצפה לקבל אובייקט מסוג ExpectedCondtion, כאשר בזמן ההמתנה תופעל מתודת apply, מתודת ההמתנה שלנו צריכה להחזיר ExpectedCondition. אנחנו בחרנו להשתמש ב- String כערך המוחזר, ולכן מימשנו אותו ב- Generics (וזה הזמן למי שלא מכירה מה זה Java Generics לרשום בצד ולסמן לעצמה את הנושא הבא שכדאי ללמוד ולשלוט בו).

שורות 5-10: בדומה לדוגמאות שראינו עד כה, גם כאן - מימוש של מתודת apply של מחלקת ExpectedCondition. המתודה מקבלת את ה- String של ה- title הבסיסי שאליו יצורף הרכיב הנוסף. במקרה שלנו, title שיתקבל כפרמטר למתודה הוא "Google Search - " (בלי שורת החיפש שהזנו בדף החיפוש).

המתנה שוטפת - Fluent Wait

מחלקת Fluent Wait היא מחלקה מקבילה ל- WebDriverמאפשרת להגדיר גם את הפעולה שיש להמתין לה, גם את הזמן המקסימלי להמתנה וגם את מרווחי הבדיקה בין הפעולות (polling interval). למעשה, FluentWait ו- WebDriverWait הם בעלי יכולות זהות. WebDriverWait הוא גרסה ספציפית ומכוונת של FluentWait - הגדרות בסיס אשר מוגדרות ב- WebDriverWait באופן אוטומטי, אינן אוטומטיות ב- FluentWait ועל כן

FluentWait הוא מנגנון גמיש יותר מאשר WebDriverWait. להלן דוגמא למימוש המתנה על ידי מתודת FluentWait שממתינה במשך 30 שניות להופעה של אלמנט עם id בשם foo. המתודה תדגום את הדף כל 5 שניות, ובמידה ויתקבל Exception מסוג NoSuchElementException, המתודה תתעלם ותתייחס ל-Exception כאילו עדיין לא התקיים התנאי ויש להמשיך ולדגום.

```
1     // Waiting 30 seconds for an element to be present on the page,
2     checking
3     // for its presence once every 5 seconds.
4     Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)
5       .withTimeout(Duration.ofSeconds(30))
6       .pollingEvery(Duration.ofSeconds(5))
7       .ignoring(NoSuchElementException.class);
8
9     WebElement foo = wait.until(new Function<WebDriver, WebElement>() {
10      public WebElement apply(WebDriver driver) {
11        return driver.findElement(By.id("foo"));
12      }
13    });
```

נסביר את הקוד:

שורה 4: נגדיר משתנה מסוג Wait (משתנה שהוא interface). לתוך המשתנה נגדיר אובייקט חדש מסוג FluentWait

שורה 5: הגדרה של הזמן המקסימלי להמתנה.

שורה 6: הגדרה של מרווח הבדיקה בין פעולה לפעולה

שורה 7: התעלמות מ- Exception מסויימים (במקרה שלנו - אלמנט שלא נמצא על הדף)

שורה 9: הפעלת מנגנון ההמתנה. יש לשים לב שהפעלת מנגנון ההמתנה מבוצע לא על ידי מחלקת ExpectedCondition אלא על ידי מחלקת האב ממנו היא יורשת - Function.

שורות 10-11: הגדרה של פעולת ההמתנה - הגדרה למה ממתינים. במקרה הזה - למציאת אלמנט על הדף.

מימוש פרטי של המתנה

עד שלב זה דנו בהרחבה במנגנונים השונים של Selenium וביכולת שלהם לתת מענה למצבים שונים של המתנה בתהליך הבדיקות האוטומטיות. ראינו סוגים שונים של המתנה, חלקם בהתייחסות ל- WebDriver, חלק בהתייחסות ל- WebElement, ואף ראינו איך ניתן לבצע ייחוס משלנו למחלקת ExpectedCondition של

Selenium, מימוש משלנו למחלקת Function וכן שימוש הן במנגנון הבסיס - WebDriverWait והן במנגנון הכללי יותר של המתנה - FluentWait. מנגנוני ההמתנה שצויינו עד כה, יחד עם מנגנון Implicit Wait מכסים בצורה טובה את רוב המקרים בהם נתקל במהלך הבדיקות האוטומטיות.

אך משהו עדיין חסר בהשלמת התמונה.

נבחן את המקרה הבא: נניח ואנחנו מחכים שבזמן טעינת הדף יופיע אלמנט על המסך. במידה והוא לא עולה, חלק מתהליך ההמתנה כולל ביצוע רענון (refresh) לדף ובדיקה חוזרת האם האלמנט מופיע על המסך.

השוני העיקרי בין מצב זה למצבים בהם התמודדנו עד כה הוא **הצורך לבצע פעולה אקטיבית כאשר פונקציית apply מחזירה את הערך null.** מתודת until תפעיל שוב ושוב את ExpectedCondition עד שהתוצאה שתחזור אינה null, או עד שאחד התנאים להמתנה (זמן המתנה, זריקת Exception) לא יתקיימו, אך ללא אפשרות להגדיר פעולה שתתבצע כאשר התוצאה המוחזרת היא שלילית.

אם כן, עלינו לבנות מנגנון שיאפשר לנו להחליט על פעולה נדרשת שתתבצע כאשר בדיקת התנאי נכשלת.

המנגנון שנגדיר מזכיר מאוד את המנגנון של Selenium. הוא יכלול הגדרה של interface ומימוש שלו כ-anonymous class בדיוק כמו המנגנון של Selenium. מנגנון זה הופיע לראשונה בספר Selenium WebDriver - From Foundations To Framework.

```
1   public interface Attemptable {
2      void attempt() throws Throwable;
3      void onAttemptFail();
4   }
```

שתי מתודות מוגדרות ב- interface הזה:

attempt - פעולת הניסיון לבחינת קיום התנאי

onAttemptFail - מה עושים כאשר נכשלים.

נבחן את הקוד שמפעיל את המנגנון:

```
1     public class Retry {
2        private final long interval;
3        private final TimeUnit unit;
4        private long count;
5
6       private WebDriver driver;
```

```
7
8       public Retry(WebDriver driver, int count, int interval, TimeUnit unit) {
9           this.count = count;
10          this.interval = interval;
11          this.unit = unit;
12          this.driver = driver;
13      }
14
15
16      public void attempt(Attemptable attemptable) {
17          for (int i = 0; i < count; i++) {
18              try {
19                  attemptable.attempt();
20                  return;
21              } catch (Throwable e) {
22                 if (i == count - 1) {
23                      throw new IllegalStateException(e);
24                  }
25                 attemptable.onAttemptFail();
26             }
27              try {
28                  unit.sleep(interval);
29              } catch (InterruptedException e) {
30                  throw new IllegalStateException(e);
31              }
32          }
33      }
34  }
```

שורות 2-4: interval הוא זמן המנוחה בין הבדיקות. Count הוא כמות הפעמים שנבדוק.

שורות 7-13: הבנאי של המחלקה (Constructor) - מאתחל פרמטרים

שורה 16: המתודה attempt מקבל אובייקט מסוג Attemptable. כלומר יש פה חלוקה מאוד ברורה בין המחלקה Retry לבין Attemptable. המחלקה - Retry מתייחסת למנגנון הריצה- כמה רצים, מתי רצים, והפעלת מנגנון ההמתנה ומנגנון הכשלון, בעוד Attemptable מגדיר את הקריטריון להצלחה ומה עושים עם כשלון.

שורה 17: ננסה לבצע את הפעולה כמות הפעמים שהוגדר בתחילת המחלקה.

שורה 18-20: ננסה לבצע את הפעולה. אם נצליח, נחזור וזהו סימן להצלחה.

שורות 21-26: במידה ולא הצלחנו, נבדוק כמה פעמים כבר רץ הניסיון. אם אנחנו במסגרת הגבולות - נבצע את המתודה onAttemptFail שמבצעת פעולה / אתחול

שורות 27-31: ביצוע המתנה

בדוגמא הבאה נבנה מנגנון המתנה שינסה 10 פעמים עם המתנה של שניה בין פעולה לפעולה (סה"כ 10 שניות) שאלמנט יופיע על המסך. במידה והוא אינו מופיע, דף ה- Web ירוענן:

```
1     Retry retry = new Retry(driver, 10, 1, TimeUnit.SECONDS);
2     retry.attempt(new Attemptable() {
3        @Override
4        public void attempt() throws Throwable {
5            driver.findElement(By.id("Some ID"));
6        }
7
8        @Override
9        public void onAttemptFail() {
10           driver.navigate().refresh();
11       }
12    });
```

מתי נשתמש בכל סוג המתנה:

סוג המתנה	מתי

ImplicitWait	נעדיף שלא להשתמש בו באופן גורף. נשתמש בו כאשר מפתחים תסריט שמראש נראה כלא יציב, ולא נרצה על כל צעד להפעיל מנגנוני ExplicitWait. נפעיל אותו בתחילת הטסט ונכבה בסופו.
המתנות של Selenium	השיטה המועדפת במירב המצבים.
מימוש פרטי של ExpectedCondition	כאשר אין המתנה של Selenium שמתאימה בדיוק למה שאנחנו מחפשים.
FluentWait	כאשר רוצים להחליט באופן מפורש את כלל התנאים והפרמטרים של ההמתנה.
Retry	כאשר רוצים לשלוט בתהליך - מה קורה כאשר ההמתנה נכשלת.

המתנה כחלק מתשתית בדיקות אוטומטיות

להלן מס' המלצות למימוש המתנה כחלק מתשתית הבדיקות:

א. כדאי לקבוע את הפרמטרים של ההמתנה - זמן ההמתנה המירבי ומרווח הדגימה, כפרמטרים בקובץ properties ולא hard coded במנגנון ההמתנה. כך במקום אחד נוכל לבצע שינוי ולהגדיל / להקטין את הפרמטרים בבת אחת בהתאם לשינויים חיצוניים כמו שינוי ביציבות המערכת, שינוי ברוחב הפס וכו'. במידה ואנו זקוקים להמתנות שונות - נניח 30 שניות באופן רגיל ולפעמים 1 דקה, נוכל לבצע זאת על ידי פניה יחסית למשתנה הזמן. לדוגמא:

```
int timeout = 30;
int pollintInterval = 10;

// Wait for 30 seconds with 10 seconds polling interval
FluentWait wait1 = new WebDriverWait(driver,
```

```
timeout).pollingEvery(Duration.ofSeconds(pollintInterval));

// Wait for 60 seconds with 5 seconds polling interval
FluentWait wait2 = new WebDriverWait(driver, timeout *
2).pollingEvery(Duration.ofSeconds(pollintInterval /2));
```

ב. כדאי לכתוב מחלקת utilities שתקצר את כתיבת ההמתנות השונות. לדוגמא:

```
1       public final class Waits {
2          private Waits() {
3          }
4
5
6          public static Alert alertIsPresent(WebDriver driver,int timeOut ,By
7       by) {
8                return new WebDriverWait(driver,
9       timeOut).until(ExpectedConditions.alertIsPresent());
10         }
11
12
13         public static WebElement elementToBeClickable(WebDriver driver,int
14      timeOut ,By locator) {
15               return new WebDriverWait(driver,
16      timeOut).until(ExpectedConditions.elementToBeClickable(locator));
17         }
18      }
```

ואז מימוש לדוגמא יהיה:

```
1     WebElement element = Waits.elementToBeClickable(driver, 30, By.id("Some
2     ID"));
```

סיכום

הכרה של סוגי ההמתנה השונים ב- Selenium תאפשר לנו גם להיות מודעים לסיכונים שבתהליך הבדיקות, וגם ביכולות שלנו להתמודד עם פערים בתזמונים בין קוד האוטומציה למערכת הנבדקת. פרויקט אוטומציה שבנוי בצורה טובה יכיל:

- 95 אחוז המתנות נפוצות של Selenium (מבוססות על מחלקה ExpectedConditions)
- 3 אחוז המתנה מובנית אישית של ExpectedCondition
- 2 אחוז של מחלקת Retry
- Implicit wait לטסטים ספציפיים שעשויים להיות לא יציבים.

מקורות מידע:

https://www.selenium.dev/documentation/webdriver/waits/

https://www.lambdatest.com/blog/expected-conditions-in-selenium-examples/

https://www.testim.io/blog/how-to-wait-for-a-page-to-load-in-selenium/

http://makeseleniumeasy.com/2020/04/29/fluentwait-vs-webdriverwait-in-selenium-webdriver/

https://www.amazon.com/Selenium-WebDriver-Foundations-Yujun-Liang-ebook/dp/B01N9D0HMG