סיכום שיעור תשיעי:

1. **Files I/O**

Java IO משמש אותנו ע"מ לקרוא ולכתוב מ/אל קבצים (קבצי טקסט, PDF, תמונות וכו').
ישנם שני דרכים של streams המשמשים לעבודה עם קבצים:

- **InputStream** – The InputStream is used to read data from a source.
- **OutputStream** – The OutputStream is used for writing data to a destination.

https://docs.oracle.com/javase/tutorial/essential/io/index.html

בשיעור ראינו דוגמא של קריאה/כתיבה מ.אל קובץ טקסט (.txt):

- קריאה:

```
public classMain {
   public String read() {
         try(BufferedReader br = new BufferedReader(new FileReader("file.txt"))) {

   StringBuilder sb = new StringBuilder();

   String line = br.readLine();

   while (line != null) {

      sb.append(line);

      line = br.readLine();

   }

   String everything = sb.toString();

return everything;

}

}
```

- כתיבה:

```
public classMain {
   public String read() {

try {

   File logFile=new File(C://...file.txt);

   BufferedWriter writer = new BufferedWriter(new FileWriter(logFile));

String string = “knva”;

   writer.write (string);

   writer.close();

} catch(Exception e) {

   e.printStackTrace();

}

}
```

- <u>שימוש באוטומציה</u> – כפי שדיברנו בשיעור, לפעמים נרצה לבצע בדיקות שונות ללא שינוי קוד, ולכן נרצה לקרוא את הקונפיגורציה מתוך קובץ מסוים, כדי שהקוד יעבוד בצורה דינמית.
- ראינו דוגמא של קריאת קונפיגורציה מתוך קובץ XML (ראו דוגמא מטה)

- בדוגמא מטה נראה איך ניתן לקרוא מתוך הקובץ את סוג הדפדפן ולהריץ את התכנית בהתאם:

```
<?xml version="1.0" encoding="utf-8" ?>
<config>
  <browserType>Chrome</browserType>
</config>
```

```
@BeforeClass
public void test01() throws Exception{
        String browserType = getData("BrowserType");
        if(browserType.equals("Chrome")){
        driver = new ChromeDriver();
        }else if(browserType.equals("IE")){
        driver = new InternetExplorerDriver();
        }
        driver.manage().window().maximize();
        driver.get("http://www....");
}
public static String getData (String keyName) throws Exception{
                File fXmlFile = new File("c://.../test.xml");
                DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
                DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
                Document doc = dBuilder.parse(fXmlFile);
                doc.getDocumentElement().normalize();
                return doc.getElementsByTagName(keyName).item(0).getTextContent();
        }
```

-

2. **יעילות טסטים**

עד היום עבדנו מתוך מחלקה אחת.
ישנן מספר דרכים ע"מ לעבוד בצורה יעילה יותר כך שהקוד שלנו:
* יהיה קל לתחזוקה.
* יכיל פחות כפילות.
* יהיה יותר קריא.

- קבועים (Constants):

ב Java יצירת קבועים תהיה ע"י הפיכתם ל public static final כך שניתן יהיה לגשת אליהם מכל מקום, ושלא יהיה ניתן לשנות את ערכם בזמן ריצה.
לדוגמא נייצר מחלקה בשם Constants ונקרא לה מכל מקום בטסט (ממחלקת Test):
https://docs.oracle.com/javase/tutorial/java/javaOO/classvars.html

```
class Constants{

public static final String MY_ID = “id”;

}

class Test{

@Test

public void test01(){

driver.findElement(By.id(Constants.MY_ID)).click();

        }

}
```

- **POM (page object model)**

הרעיון הכללי הינו להפוך כל דף אינטרנט למחלקה בפרויקט הכוללת את כל מה שרלוונטי לאותו דף האינטרנט.

- **Page Object Model** is a design pattern to create **Object Repository** for web UI elements.
- Under this model, for each web page in the application, there should be corresponding page class.
- This Page class will find the WebElements of that web page and also contains Page methods which perform operations on those WebElements.
- Name of these methods should be given as per the task they are performing, i.e., if a loader is waiting for the payment gateway to appear, POM method name can be waitForPaymentScreenDisplay().

```
class LoginPage{
private static WebElement element = null;

public static webElement pressLogin(WebDriver driver){
element = driver.findElement(By.id("userName"));
return element;
            }
}
```

```
class Main{
public static WebDriver driver;

@Test
public void test01(){
LoginPage.pressLogin(driver).click();
            }
}
```

- **Page Factory**

דרך נוספת לעבודה הינה שימוש במחלקת PageFactory אשר משמש כמעין הארכה ל POM כאשר נממש את הפונקציה שנקראת initElements כל האלמנטים באותו הדף יאותחלו.
https://github.com/SeleniumHQ/selenium/wiki/PageFactory

```
public class Main{
static LoginPage loginPage ;
// To initialize elements.
@BrforeTest
Public static void initialize(){
loginPage = PageFactory.initElements(driver, LoginPage.class);
                }
@Test
public void test01(){
                loginPage.loginButton.clickLogin();
                }
}
```

```
public class LoginPage{
@FindBy(how = How.ID, using = "login")
private WebElement loginButton;

public LoginPage(WebDriver driver){ // we need to create a constructor
this.driver = driver;
                }
public void clickLogin(){
loginButton.click();
                }
}
```

תזכורת ממה שעשינו בכיתה:

1. Initialize page factory
2. Call page factory method
3. Use find by annotation
4. Creating a constructor
5. Using an item

```
public class TestEfficiency {
    private static WebDriver driver;

    @BeforeClass
    public static void beforeMyClass() {
        System.setProperty("webdriver.chrome.driver", "C:\\ \\
        driver = new ChromeDriver();

        driver.manage().window().maximize();
        driver.get("https://dgotlieb.github.io/WebCalculator");

        POM_and_Factory pomAndFactory = PageFactory.initElements(driver, POM_and_Fac-
tory.class);
        POM_and_Factory.pressFive(driver).click();

    }
```

1

2

```
public class POM_and_Factory {
    private static WebElement element = null;
    private final WebDriver driver;

    @FindBy(how = How.ID, using = "equal")
    private static WebElement calculateButton;

    public POM_and_Factory(WebDriver driver) { // we need to create a constructor
        this.driver = driver;
    }

    static void clickCalculate() {
        calculateButton.click();
    }

    static WebElement pressFive(WebDriver driver) {
        element = driver.findElement(By.id("five"));
        return element;
    }
}
```

3

4

5