בס"ד תשפ"ג סמסטר א'

**מיני פרויקט במערכות חלונות**

***שלב 4***

# **מטרות התרגיל**

מימוש ארכיטקטורת (מודל) שלושת השכבות כפי שנלמד בקורס.

שימוש בתבניות עיצוב (Design Patterns) של סינגלטון (Singleton) ויצרן פשוט (Simple Factory).

# **הנחיות לביצוע התרגיל והגשתו**

* העבודה תתבצע בזוגות בלבד
* חובה להשתמש בכלי לניהול גרסאות git ובאתר github.org
* חובה לבצע את השלב באותו המאגר ובאותו ה-Solution כמו שלב 0 (מקדים)
* חובה להגיש במודל קישור על פי הנחיות ההגשה
* חובה להקפיד על פורמט זה על מנת למנוע מצב של אי קבלת ציון על תרגיל מסוים
* חובה לקרוא את התיאור הכללי של הפרויקט לפני תחילת העבודה על השלב

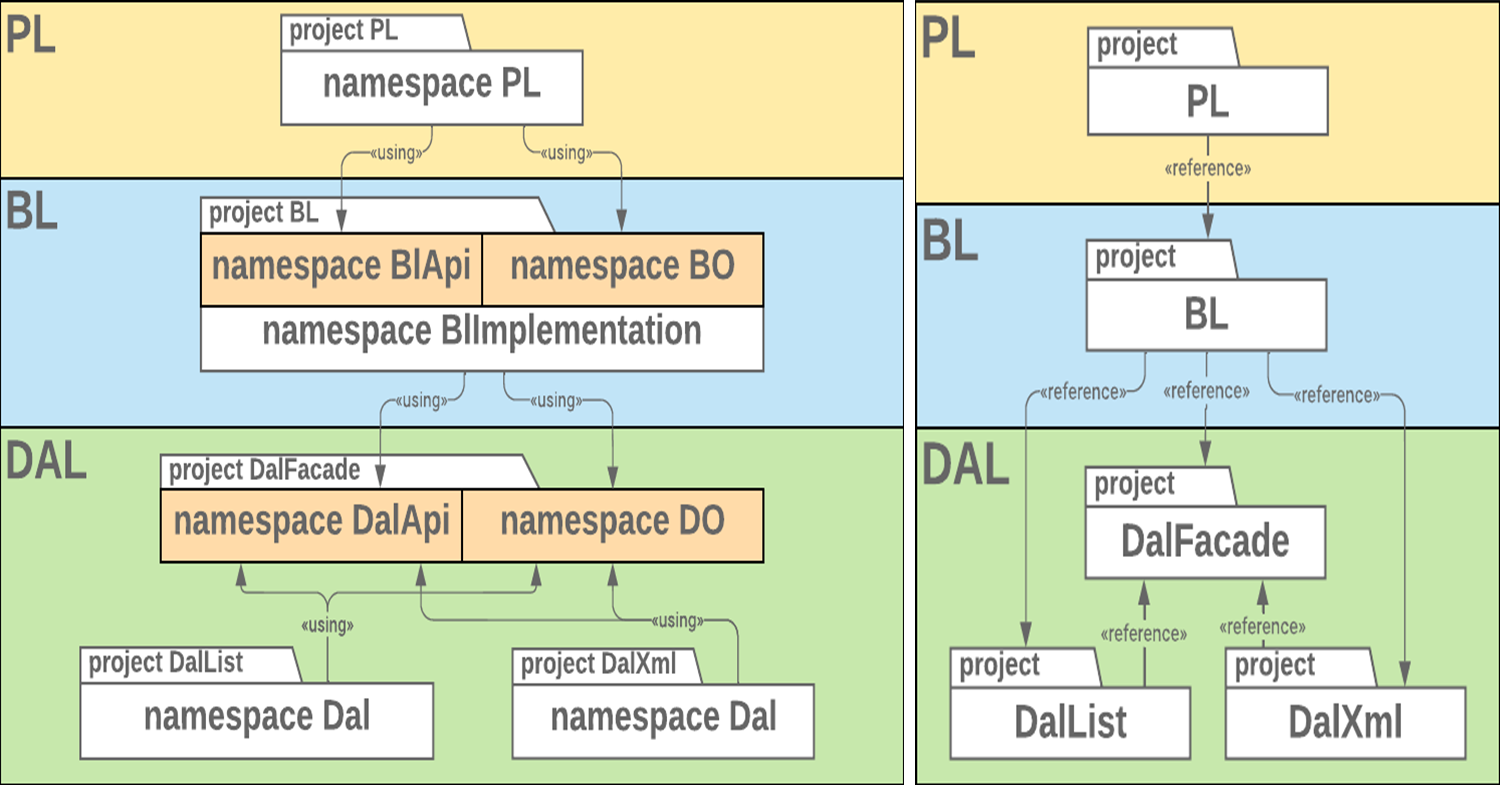
# **הקדמה**

בשלבים 1 עד 3 התחלנו לבנות את שכבות הפרויקט.

כתבנו תוכניות ראשיות לבדיקת השכבה הלוגית ושכבת הנתונים

והתחלנו ליצור ממשק משתמש גרפי. **בשלב 4:**

1. **נעדכן את כל השכבות** עם תבניות עיצוב שנלמדו בינתיים:
   1. הפיכת מחלקות מתאימות לסינגלטון (Singleton)
   2. יצירת מחלקות יצרן (Factory) פשוט עבור שכבת הנתונים והשכבה הלוגית, ועדכון השכבות שמעליהן בהתאמה.
2. בשכבת נתונים נוסיף שליטה בקונפיגורציה ע"י קובץ xml ייעודי
   1. מטרה: לנתק תלות מעגלית בין פרויקטים DalFacade ו-DalList
   2. ניצור קובץ קונפיגורציה של שכבת נתונים
   3. ניצור מחלקת עזר לטעינת הקונפיגורציה מהקובץ
   4. נממש את יצרן (Factory) שכבת הנתונים ע"י שימוש בנתוני קונפיגורציה

מבנה המאגר ב-github.com וה-Solution אחרי שלב 4 זהה לזה שאחרי שלב 3.   
היחסים בין הפרויקטים של Solution ובין מרחבי השמות מיוצגים בתרשימים הבאים:  
****

# **המטלה**

## הפיכת מחלקת DalList לסינגלטון

1. **נחליף** את הרשאת המחלקה ל-**internal**
2. **נוסיף** מאפיין "**sealed**" לפני "**class**" בהגדרת המחלקה (על מנת למנוע ירושה ממנה)
3. **נוסיף תכונה סטטית** עם הרשאה **public** עם **get בלבד** (לקריאה בלבד) ועם **אתחול** לאובייקט של המחלקה עצמה:

**public static IDal Instance { get; } = new DalList();**

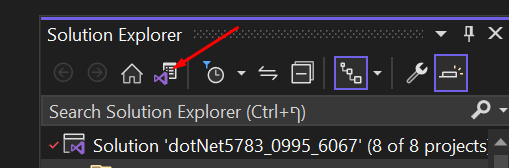
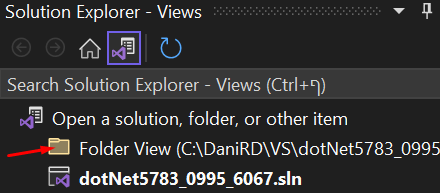
1. נהפוך את הרשאת בנאי מופע המחלקה ל-**private**
2. לבונוס - ניתן לעשות סינגלטון שיהיה Thread Safe ועם Lazy Initialization מירבי - אבל רק אם אתם יודעים להסביר איך עשיתם ולמה זה Thread Safe ו-Lazy Initialization

## יצירת קונפיגורציה עבור שכבת DAL

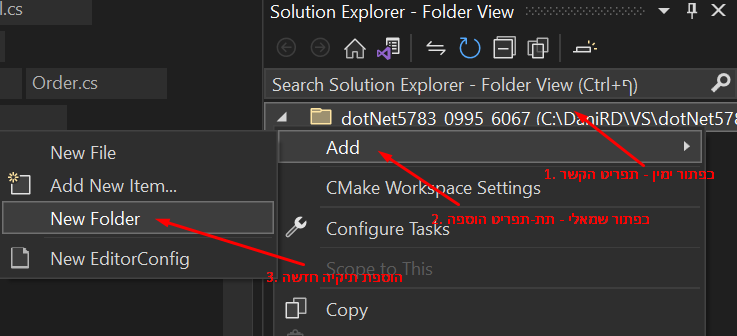
1. ניצור קובץ קונפיגורציה ב-Solution

נעשה את העבודה בתצוגת תיקיות בויזואל סטודיו כדלקמן:

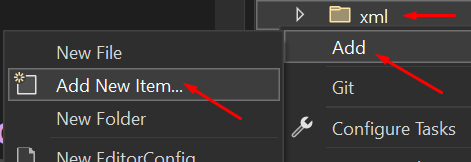
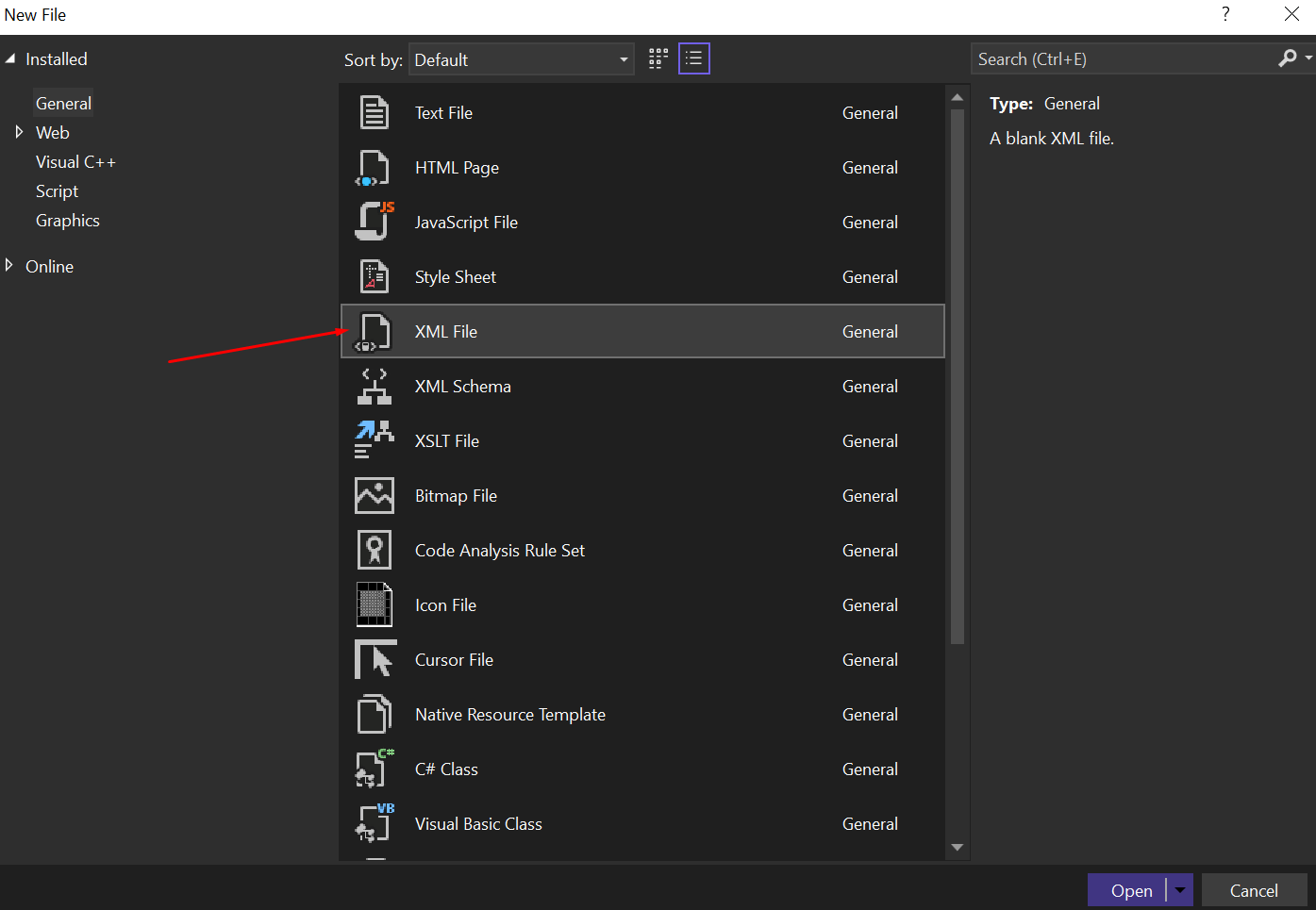
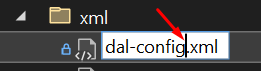
* 1. בסרגל של Solution Explorer נלחץ על סמל  של "Switch between solutions and available views" ולאחר מכן נעבור לתצוגת תיקיות (Folder View) ע"י לחיצה כפולה:

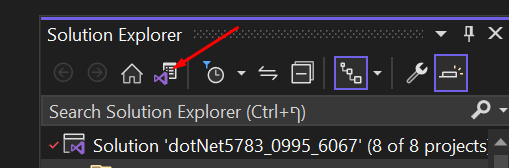
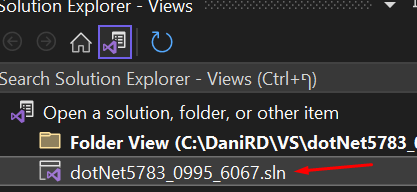
* 1. בתיקיה הראשית (של ה-Solution) נוסיף תת-תיקיה חדשה בשם xml:

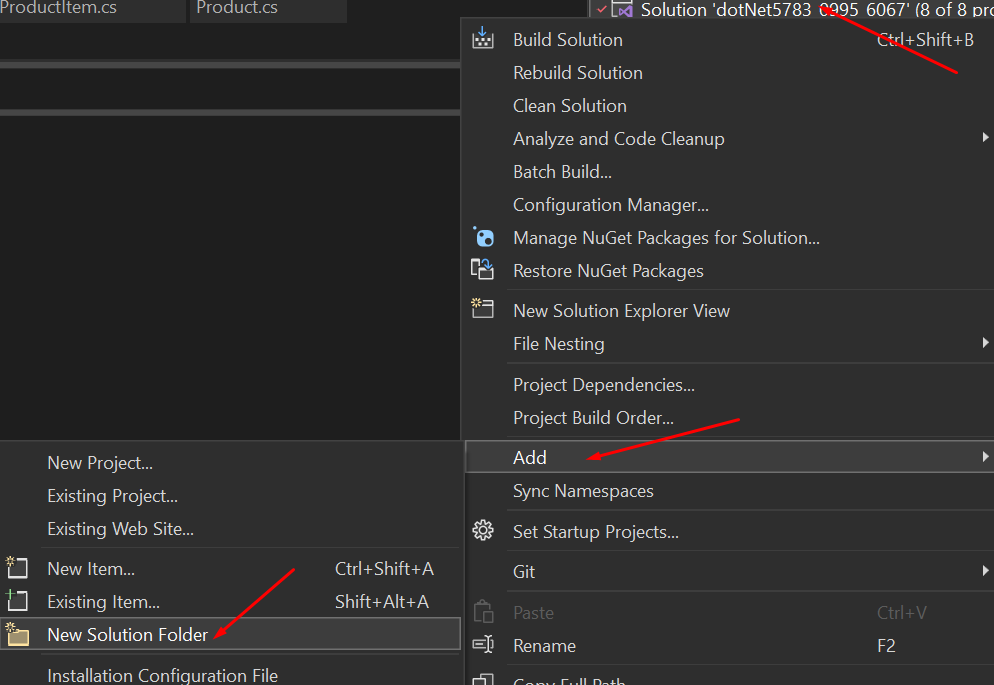
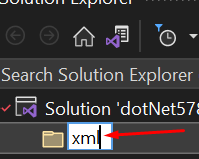
* 1. בתת-התיקיה xml שיצרנו, נוסיף אלמנט חדש (...Add New Item) מסוג קובץ XML, ונשנה את שמו ל-**dal-config.xml**:

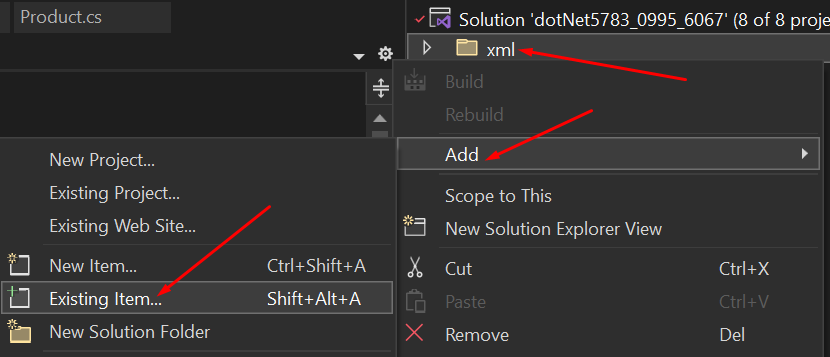
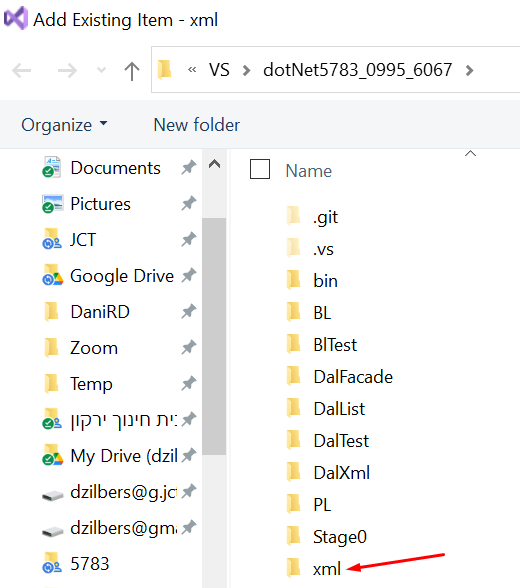
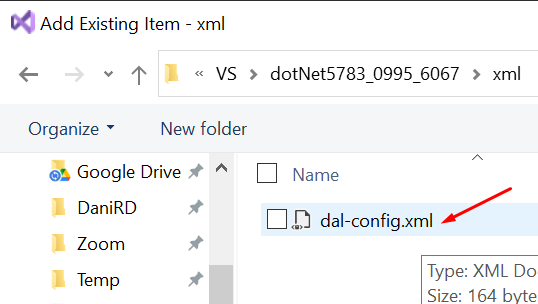
1. נוסיף ב-Solution גישה לקובץ שיצרנו
   1. נחזור לתצוגת Solution הרגילה בויזואל סטודיו - בדומה לסעיף **a.1.** לעיל, ע"י לחיצה כפולה על הקובץ עם סיומת "**sln.**":

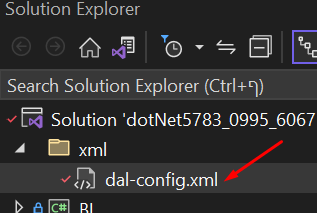
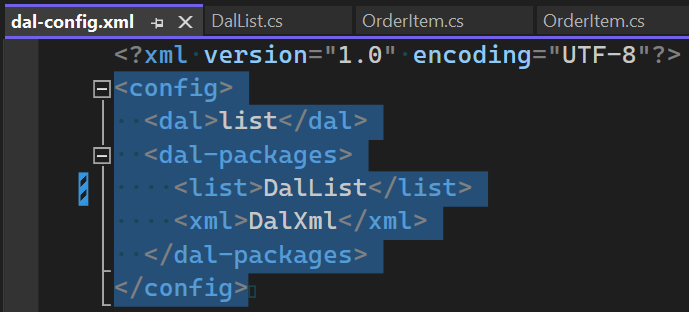
* 1. נוסיף ב-Solution תיקייה מדומה (New Solution Folder) ונשנה את שמה ל-**xml**:

* 1. בתיקיה המדומה xml הזו נוסיף אלמנט קיים (Add Existing Item) - קישור לקובץ dal-config.xml שיצרנו בסעיף **c.1.** לעיל:

1. ניצור את קונפיגורציית שכבת DAL בקובץ xml
   1. בויזואל סטודיו נפתח את הקובץ dal-config.xml ונוסיף לא את התוכן הבא:

לנוחיותכם על מנת לעשות העתק\הדבק:

**<config>**

**<dal>list</dal>**

**<dal-packages>**

**<list>DalList</list>**

**<xml>DalXml</xml>**

**</dal-packages>**

**</config>**

* 1. הסבר קצר:
     + תגיות <שם> ו-<שם/> מייצגות תחילה וסיום (בהתאם) של רכיב (אלמנט) של מידע
       - יש אלמנטים פשוטים - שהתוכן שלהם הינו טקסט המייצג ערך מחרוזת
       - יש אלמנטים מורכבים - שהתוכן שלהם הוא רצף של אלמנטים מקוננים
     + אלמנט ראשי **config** הינו **אלמנט שורש** של קובץ ה-xml הזה
     + אלמנט **dal** מכיל את שם חבילה של מימוש שכבת DAL שאנחנו הולכים להשתמש
     + אלמנט **dal-packages** מכיל רשימה (ע"י אלמנטים מקוננים בו) של חבילות מימוש DAL אפשריות בפרויקט שלנו (קיימות ועתידיות):
       - שם תת-אלמנט - שם חבילה כפי שקבענו אותה בקובץ הזה ואנחנו יכולים לבחור בה בעזרת הערך של האלמנט **dal** הנ"ל
       - תוכן (ערך) של תת-אלמנט הינו שם הפרויקט המכיל מימוש מסוים של שכבת DAL (שהו גם שם קובץ ה-"**dll.**" הבינארי שלו) - למשל **DalList** שבנינו בשלב 1 ועדכננו בשלבים הבאים אחריו, ו-**DalXml** שנבנה בשלב 6 של הפרויקט

## הוספת יצרן פשוט (Simple Factory) לשכבת הנתונים (שמשתמש בקונפיגורציית השכבה)

כל העבודה בפרק הזה מתבצעת בתת-תיקיה **DalApi** של פרויקט **DalFacade**.

1. נוסיף מחלקה סטטית **DalConfig** כדלקמן:

**namespace DalApi;**

**using System.Xml.Linq;**

**static class DalConfig**

**{**

**internal static string? s\_dalName;**

**internal static Dictionary<string, string> s\_dalPackages;**

**static DalConfig()**

**{**

**XElement dalConfig = XElement.Load(@"xml\dal-config.xml")**

**?? throw new DalConfigException("dal-config.xml file is not found");**

**s\_dalName = dalConfig?.Element("dal")?.Value**

**?? throw new DalConfigException("<dal> element is missing");**

**var packages = dalConfig?.Element("dal-packages")?.Elements()**

**?? throw new DalConfigException("<dal-packages> element is missing");**

**s\_dalPackages = packages.ToDictionary(p => "" + p.Name, p => p.Value);**

**}**

**}**

**[Serializable]**

**public class DalConfigException : Exception**

**{**

**public DalConfigException(string msg) : base(msg) { }**

**public DalConfigException(string msg, Exception ex) : base(msg, ex) { }**

**}**

הסבר קצר:

* 1. מרחב שמות **System.Xml.Linq** כולל כלים לטיפול בקבצי xml וב-DOM XML.
  2. מתודה **XElement.Load** טוענת קובץ xml, מבצעת ניתוח שלו (parsing) ובונה עץ DOM (של Document Object Model) - עץ אובייקטים מטיפוס **XElement** ומחזירה את האובייקט של אלמנט השורש של xml.
  3. אובייקט מטיפוס **XElement** מכיל מידע מנותח על אלמנט של xml.
  4. מתודה **("Element("dal** מביאה אובייקט של אלמנט עם שם **"dal"** מתוך העץ של DOM, כאשר תכונה **Value** מביאה את הערך של האלמנט (אם נסתכל באלמנט בקובץ: **<dal>list</dal>**: - מחרוזת "**list**").
  5. זימון של **()Element("dal-packages")?.Elements** מחזיר אוסף של כל האלמנטים שהם "בניו" של אלמנט בשם **"dal-packages"** בעץ של DOM, זאת אומרת כל תת-האלמנטים של אלמנט **<dal-packages>**.
  6. מתודה **(ToDictionary(p => "" + p.Name, p => p.Value** בונה ומחזירה טבלה מגובבת [hash table] ("מילון") ששם תת-האלמנט הוא מפתח גיבוב, וערך האלמנט הינו הערך הַמֻּצְמָד למפתח הגיבוב, ומהקובץ שלנו נקבל (בעזרת רישום בפורמט דמוי json שהוא יותר אינטואיטיבי מ-xml להצגת אובייקטים):

**{**

**"list": "DalList",**

**"xml": "DalXml"**

**}**

1. נוסיף מחלקה סטטית **Factory** (מחלקת יצרן של שכבת DAL) כדלקמן:

**namespace DalApi;**

**using System.Reflection;**

**using static DalApi.DalConfig;**

**public static class Factory**

**{**

**public static IDal? Get()**

**{**

**string dalType = s\_dalName**

**?? throw new DalConfigException($"DAL name is not extracted from the configuration");**

**string dal = s\_dalPackages[dalType]**

**?? throw new DalConfigException($"Package for {dalType} is not found in packages list");**

**try**

**{**

**Assembly.Load(dal ?? throw new DalConfigException($"Package {dal} is null"));**

**}**

**catch (Exception)**

**{**

**throw new DalConfigException("Failed to load {dal}.dll package");**

**}**

**Type? type = Type.GetType($"Dal.{dal}, {dal}")**

**?? throw new DalConfigException($"Class Dal.{dal} was not found in {dal}.dll");**

**return type.GetProperty("Instance", BindingFlags.Public | BindingFlags.Static)?**

**.GetValue(null) as IDal**

**?? throw new DalConfigException($"Class {dal} is not singleton or Instance property not found");**

**}**

**}**

הסבר קצר:

* 1. יִבּוּא סטטי של מחלקה **using static DalApi.DalConfig** מאפשר שימוש במתודות סטטיות של מחלקה כאילו היו פונקציות גלובליות
  2. פניה דרך אינדקס למשתנה מסוג טבלה מגובבת ("מילון") **[s\_dalPackages[dalType** מביא ערך תואם למפתח שבאינדקס
  3. מתודה **Assembly.Load** טוענת מודול (קובץ) **dll.\*** המכיל ספריית מחלקות (במקרה שלנו - **DalList.dll**)
  4. זימון **("{Type.GetType($"Dal.{dal}, {dal** מביא reflection (זאת אומרת מטא-דטא) של הטיפוס **Dal.DalList** ממוֹדוּל **DalList.dll** (במקרה שלנו)
  5. הזימון **type.GetProperty** מחזיר מטא דטא של תכונה **"Instance"** שחייבת להיות סטטית ועם הרשאה public
  6. הזימון של **(GetValue(null** על המטא דטא של התכונה **"Instance"** מחזיר ערך של התכונה הסטטית הזו (שזהו אמור להיות המופע [היחיד] של מחלקת הסינגלטון **DalList** שלנו)

1. לבונוס: שכלול הפונקציונליות על מנת לאפשר ב-**dalconfig.xml** להוסיף ל"חבילות" אטריבוטים של שם מרחב השמות **=namespace** ושם מחלקה **=class** ובכך לאפשר שהשמות יהיו שונים מהקביעות של ברירת מחדל שעשינו עד כה בפרויקט (IDal ושם המחלקה זהה לשם ה"חבילה")

## השלמת עדכונים בשכבות BL ו-PL

### **בשכבת לוגית BL**

1. בתת-תיקיה **BlApi** נוסיף מחלקה סטטית **Factory**
   1. הרשאה public
   2. במחלקה נוסיף מתודה סטטית **Get**:
      * הרשאה public
      * ללא פרמטרים
      * טיפוס ערך מוחזר **IBL**
      * המתודה תחזיר מופע של מחלקה **Bl** (מתת תיקייה **BlImplementation**)
2. בכל קוד המימוש של השכבה:
   1. נמחוק **;using Dal**
   2. נחליף את יצירת מופע של מחלקה **DalList** לקבלתו ממחלקת היצרן: **()DalApi.Factory.Get**

### 

### **שכבת התצוגה PL**

1. בכל קוד המימוש של השכבה:
   1. נמחוק **;using BlImplementation**
   2. נחליף את יצירת האובייקט של **Bl** לקבלתו ממחלקת היצרן: **()BlApi.Factory.Get**

**בהצלחה רבה!**