שלב 4 א' – מימוש תבניות עיצוב בשכבת DAL

1. **הפיכת מחלקת DalList ל singleton**

**נזכיר שמטרת תבנית העיצוב Singleton** היא ליצור מחלקה שאפשר יהיה ליצור ממנה מופע (אובייקט) אחד לכל היותר. לכל היותר הכוונה שאם לא נעשה שימוש במחלקה זו אז לא נרצה ליצור אפילו אובייקט אחד ממנה. לכן לא נשתמש במחלקה סטטית כי ממנה תמיד מיוצר מופע אחד בלבד, ולפעמים לא נרצה שום מופע.

**על מנת להפוך את מחלקת DalList לסינגלטון, בצעו את ההוראות הבאות על מחלקת DalList בפרויקט DalList:**

1. הגדר את המחלקה כ **sealed**
2. החלף את הרשאת המחלקה **DalList** מ public ל **internal**
3. הוסף למחלקה תכונה **ציבורית סטטית** בשם **Instance** לקריאה בלבד (עם get בלבד), עם אתחול לאובייקט של המחלקה עצמה.
4. הוסיפו למחלקה בנאי **פרטי** (אם כבר היה קיים בנאי אז שנו את הרשאתו **לפרטית**).

שימו לב - בשלב זה נקבל שגיאת קומפילציה בתוכנית הראשית ב **DalTes**t מכיוון ששינינו את הרשאת המחלקה ל internal

לאחר שנהפוך את כל השכבה לפקטורי, תיפתר השגיאה הזו.

1. **הגדרת מחלקת יצרן (Simple Factory) עבור שכבת הנתונים**

**נזכיר שמטרת** **תבנית העיצוב Simple Factory** היא לאפשר לשכבה עליונה לייצר אובייקטים משכבה תחתונה אשר מממשים ממשק מוסכם, **מבלי להכיר את המחלקה המממשת** .השכבה התחתונה תגדיר "מפעל לייצור אובייקטים" (**Factory**), והייצור יעשה דרך מתודה (**Get**) שתחזיר עצם מאותחל מטיפוס הממשק.

**אצלנו בפרויקט הנבנה ע"פ מודל השכבות:**

* השכבה העליונה היא **BL** (שאותה נבנה בהמשך שלב זה) והשכבה התחתונה היא **DAL**
* **ובהמשך,** השכבה העליונה תהיה **PL**  והשכבה התחתונה שלה היא **BL**

כלומר את מחלקת היצרן (Factory) **נגדיר גם בשכבה DAL וגם בשכבה BL**, כי שתיהן משמשות כ "שכבה תחתונה".

**נרצה שמפעל הייצור (Factory) במתודה Get, יקרא בזמן ריצה את שם המחלקה שאותה הוא צריך לייצר בשכבת הנתונים מקובץ קונפיגורציה. מ2 סיבות:**

* כדי שיוחלט בזמן ריצה עם איזה מימוש של שכבת נתונים בחרנו לעבוד - עם רשימות בזכרון (List) או עם קבצי XML. ולא יהיה צורך לקמפל את שכבת הנתונים (DalFacade) כל פעם שרוצים להחליף מחלקה מממשת.
* בעיה טכנית - לא ניתן להגדיר רפרנס מעגלי. כרגע יש כבר רפרנס מכיוון DalList לכיוון DalFacade ולא ניתן להגדיר רפרנס גם בכיוון ההפוך.

**לצורך כך, בפרק הנוכחי:**

* נגדיר קובץ קונפיגורציה בשם dal-config.xml שבו נציין את המחלקה שנרצה לעבוד איתה.
* נגדיר מחלקה בשם Config שיודעת לקרוא את הקובץ בזמן ריצה ולקבל את שם המחלקה
* נגדיר מחלקה בשם Factory שתיעזר במחלקת Config כדי לטעון ולאתחל את המחלקה הנכונה שאיתה נרצה לעבוד.

1. יצירת קובץ קונפיגורציה לקביעת סוג המימוש, בשם dal-config.xml

**המטרה**: הוספת ספריה **xml**, ובה קובץ קונפיגורציה שדרכו נוכל לקבוע בכל ריצה **באיזה מהפרויקטים נרצה להשתמש כדי לגשת לבסיס הנתונים.**

1. **פתחו את תיקיית ה solution (מקש ימני -> Open Folder in File Explorer)**
2. **צרו תיקיה חדשה בשם xml**
3. תחת תיקיה זו, צרו קובץ טקסט חדש בשם **dal-config.xml**. (שימו לב למחוק מסיומת שם הקובץ את ".txt")
4. לאחר מכן, בויז'ואל סטודיו, עמדו עם עכבר ימני על ה **solution** ואז: **Add → New Solution Folder**. קראו לספריה בשם **xml** (באותיות קטנות)
5. לחצו במקש ימני על הספריה xml, בחרו **Add → Existing Item**
6. בחרו להוסיף את הקובץ הקיים שזה עתה יצרתם: xml/**dal-config.xml**
7. **פתחו את הקובץ דרך הויז'ואל סטודיו** והעתיקו את הטקסט הבא להיות תוכן הקובץ עצמו:

<?xml version="1.0" encoding="utf-8"?>

<config>

<dal>list</dal>

<dal-packages>

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

<xml>DalXml</xml>

</dal-packages>

</config>

1. בצעו שמירה (Save) של הקובץ

**הסבר לגבי תוכן הקובץ dal-config.xml:**

* האלמנט הפנימי הראשון נקרא <dal> ובתוכו נכתוב שם של אלמנט אחר שקיים בקובץ, אותו האלמנט שמכיל את שם הפרויקט שבו נרצה להשתמש בריצה הנוכחית כדי לגשת לבסיס הנתונים.
* האלמנט הפנימי השני נקרא <dal-packages> והוא מכיל רשימה של חבילות (packages) אפשריות של סוגי בסיסי נתונים. בסיס נתונים מסוג list ובסיס נתונים מסוג קבצי xml (שיהיו קיימים בהמשך).

למשל, בטקסט הקודם האלמנט <dal> מכיל את המילה list, כדי לציין שבתוך האלמנט שנקרא <list> נמצא שם הפרויקט שאיתו נרצה לעבוד בריצה הנוכחית (**DalList).**

1. הוספת מחלקת Config לקריאת קובץ הקונפיגורציה

**בפרויקט DalFacade,** הוסיפו תחת ספריה **DalApi** מחלקה חדשה בשם **Config**. עם הקוד המוכן הבא: (הסבר הקוד מייד אח"כ)

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 הקוד של המחלקה Config:**

* מרחב שמות System.Xml.Linq כולל כלים לטיפול בקבצי xml וב-DOM XML.
* 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 קובץ ה xml.
* אובייקט מטיפוס **XElement** מכיל מידע מנותח על אלמנט כלשהו של xml.
* מתודה **Element("dal")** מביאה אובייקט של אלמנט עם שם **"dal"** מתוך העץ של DOM, והתכונה **Value** מביאה את הערך שבתוך האלמנט. כלומר:
  + אם האלמנט dal בקובץ נראה כך: **<dal>list</dal>**, אזי המחרוזת שתחזור לתוך s\_dalName תהיה **list**
  + ואם האלמנט dal בקובץ נראה כך: **<dal>xml</dal>**, אזי המחרוזת שתחזור לתוך s\_dalName תהיה **xml**
* 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>**. כלומר:
  + <list>DalList</list>
  + <xml>DalXml</xml>
* מתודה **ToDictionary(p => "" + p.Name, p => p.Item2)** בונה ומחזירה "מילון" שבו המפתחות הם שמות התגיות של המימושים והערכים הם האובייקטים המכילים את 3 הפרמטרים: package,namespace,class שנצטרך לצורך יצירה דינאמית של מופע מהמחלקה

1. הוספת מחלקת לטעינת מחלקת המימוש שנקבעה בקובץ הקונפיגורציה Factory

מחלקת הייצור (**Factory**) תהיה **ציבורית וסטטית** והיא תהיה האחראית הבלעדית ליצירת האובייקטים שמממשים את הממשק.  
המחלקה תהיה **ציבורית** על מנת שניתן יהיה לגשת אליה מהשכבה שמעליה.  
המחלקה תהיה **סטטית** על מנת ל"הכריח" את המתודות שלה להיות סטטיות - כדי לחסוך במופעים.

מחלקת הייצור תספק מתודה (**Get**) מייצרת **סטטית וציבורית** שתקבל פרמטר שבעזרתו היא תדע איזה טיפוס של אובייקט לייצר. ואם לא קיבלה פרמטר תייצר אובייקט ברירת מחדל שהיא מחליטה עליו. המתודה תהיה **ציבורית** על מנת שניתן יהיה לגשת אליה מהשכבה שמעליה - לייצר אובייקטים דרכה

**בפרויקט DalFacade,** הוסיפו תחת ספריה **DalApi** מחלקה חדשה בשם **Factory**. עם הקוד המוכן הבא: (הסבר הקוד מייד אח"כ)

public static class Factory

{

public static IDal Get

{

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 in dal-config.xml");

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

catch (Exception ex) { throw new DalConfigException($"Failed to load {dal}.dll package", ex); }

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

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

return type.GetProperty("Instance", BindingFlags.Public | BindingFlags.Static)?.GetValue(null) as IDal ??

throw new DalConfigException($"Class {dal} is not a singleton or wrong property name for Instance");

}

}

}

**הסבר הקוד של המחלקה Factory:**

המחלקה מכילה מתודה בשם **Get שמייצרת/מחזירה אתחול של המחלקה המתאימה DalList או DalXml,** ע"פ מה שכתוב בקובץ הקונפיגורציה **dal-config.xml** שאותו היא יודעת לקרוא בעזרת מחלקת **Config**.

**המתודה Get:**

* יִבּוּא סטטי של מחלקה **using static DalApi.DalConfig** מאפשר שימוש במתודות סטטיות של מחלקה כאילו היו גלובליות
* ניגש לשדה s\_dalName של מחלקת Config שהכנו ונקבל ממנו את שם האלמנט שמכיל את שם המחלקה שנרצה לממש
  + כלומר dalType יכיל בסופו של דבר את המחרוזת "xml" או "list".
* ניגש למילון של מחלקת Config שהכנו בסעיף הקודם ונפנה דרך אינדקס מסוג מחרוזת, **s\_dalPackages[dalType]** שיחזיר לנו לתוך המשתנה **dal** את שם המחלקה של המימוש
* מתודה **Assembly.Load** תטען את המודול (קובץ) **dll.\*** הנבחר
* הזימון הבא:

Type.GetType($"Dal.{dal}, {dal}")

מחזיר **reflection**  (מטא-דטא) של הטיפוס **Dal.DalList** ממוֹדוּל **DalList.dll** או של הטיפוס **Dal.DalXml** ממוֹדוּל **DalXml.dll**

* הזימון **type.GetProperty** מחזיר מטא דטא של התכונה **"Instance"** של המחלקה DalXml/DalList שחייבת להיות סטטית ועם הרשאה public
* 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/DalXml** שלנו)

1. עדכון DalTest לעבוד עם שכבת DAL המעודכנת

במחלקת Initialization, בפונקציה Initialize – הסירו מהגדרת הפונקציה את הפרמטר. האתחול של s\_dal יתבצע ע"י זימון ה Factory כך: s\_dal = DalApi.Factory.Get;

גם במחלקת Program, עדכנו את אתחול המופע של IDal באותה צורה וכן את זימון הפונקציה Initialize שלא צריך לקבל פרמטר.

כעת ניתן להסיר בפרויקט DalTest את ה reference לפרויקט DalList.

יש לוודא שהפרויקט עובר קומפילציה ולאחר מכן להחזיר את ה reference לצורך אילוץ סדר הקומפילציה של הפרויקטים (כמו שהסברנו לגבי פרויקט BL)

בצורה כזו, כאשר נוסיף בהמשך מימוש של DalXML לא נצטרך לשנות דבר בפרויקט זה.