המרכז האקדמי לב

מחלקה למדעי מחשב, הנדסת תכנה והנדסת מערכות תקשורת

קורס 153007

מיני פרויקט במערכת חלונות

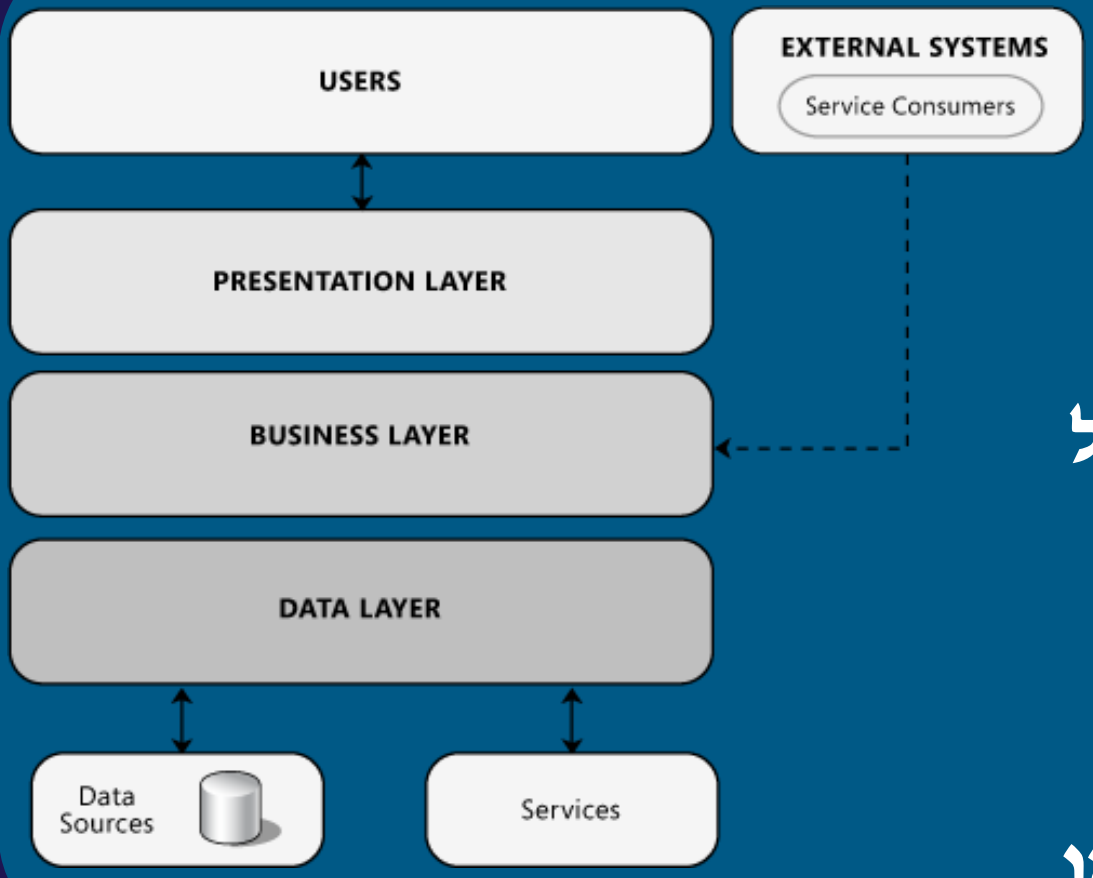
מצגת הקורס

מודל השכבות, תבניות עיצוב

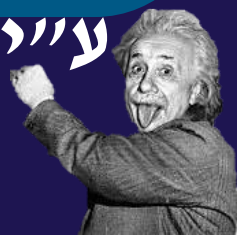
תשפ״ג סמסטר א׳

דן זילברשטיין תשפ״ג 2022/23

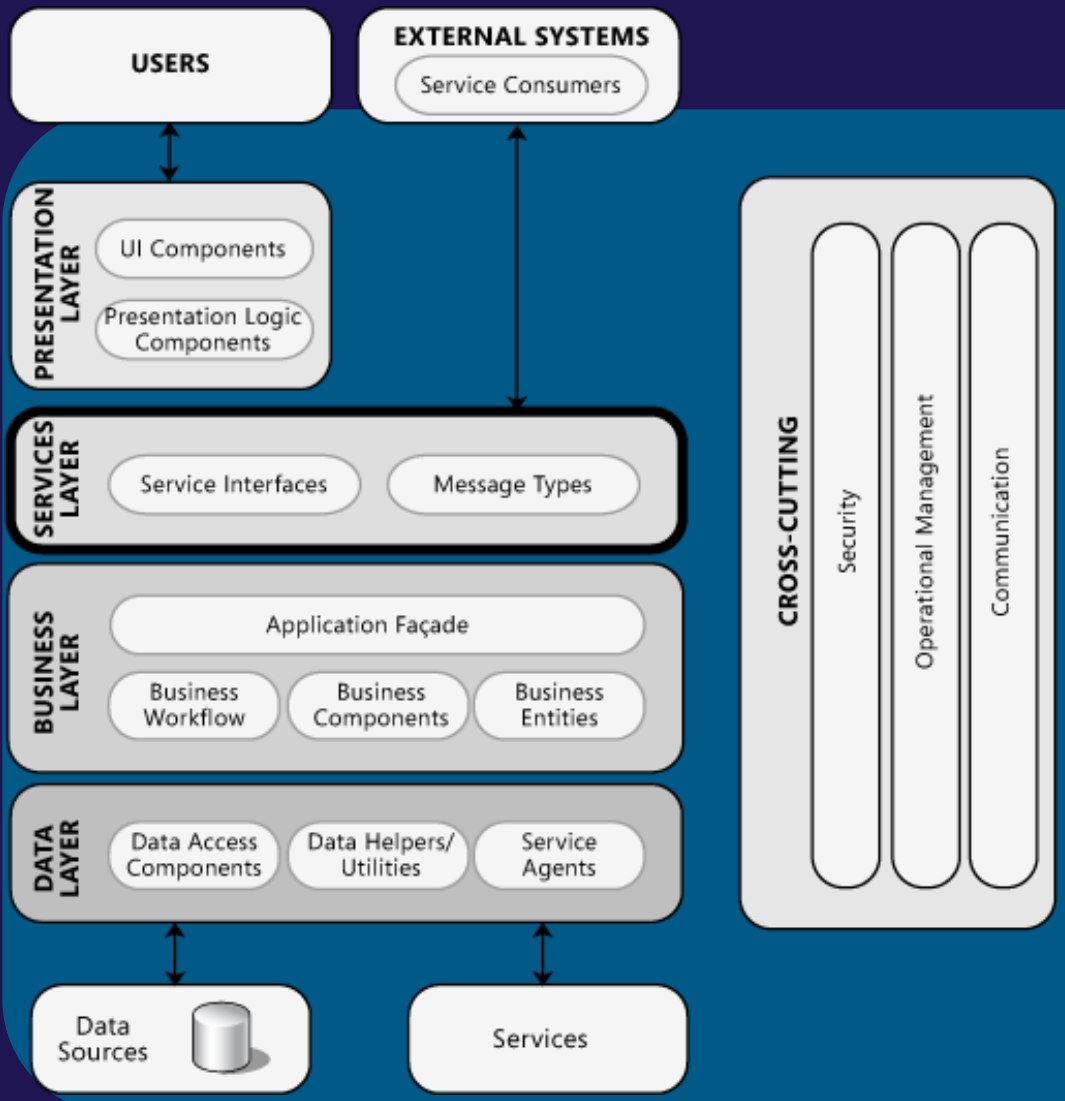
מודל שלושת השכבות של מיקרוסופט



- UI/PL – אחראי על ממשק ואינטראקציה עם משתמש
- BL – אחראי על הלוגיקה העיקרית של המערכת
- DAL – אחראי על גישה לנתונים – מקומית או חיצונית או ע"י שירותים

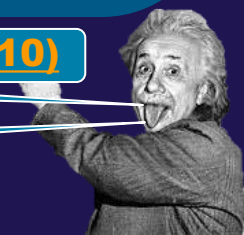


הוספת שכבת שרות



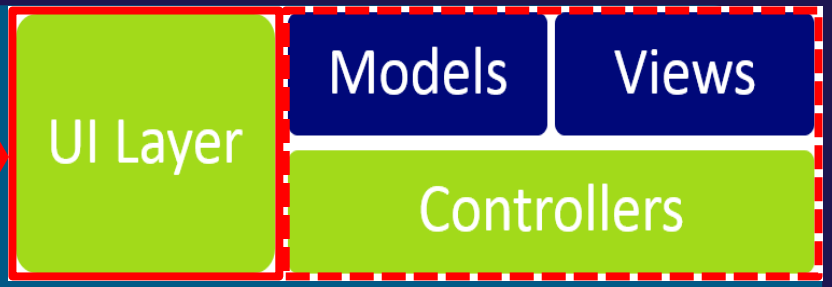
- התרשים מאפשר מבט יותר עמוק
- שכבת שרות – הרבה פעמים שכבת תקשורת

https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ee658109(v=pandp.10)



http://www.tonymarston.net/php-mysql/3-tier-architecture.html

תמונת 3TA ליישומים עם ממשק משתמש



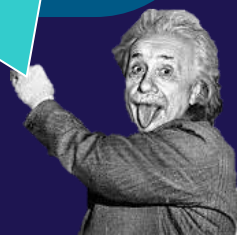
Layer = Tier

הרבה UI/PL Layers

אפליקציה: Business Logic

ממשק אחיד - DAL

הרבה Data Layers



דוגמה – תחזית מזג האוויר

- שכבת תצוגה (ממשק משתמש) – PL

```
public void ShowWeather(int day)
{ Weather w =  bl.getWeather(DateTime.Now.Day - day);  myShow(w); }
```

- שכבת לוגיקה – BL

```
public Weather getWeather(int day)
{  float x = dal.getTemperature(day);
   float y = dal.getWindDirection(day);
   // ....
   Weather w = someComputation(x,y,...);
   return w;
}
```

חוזה הנתונים של **BO**
כולל **Weather**

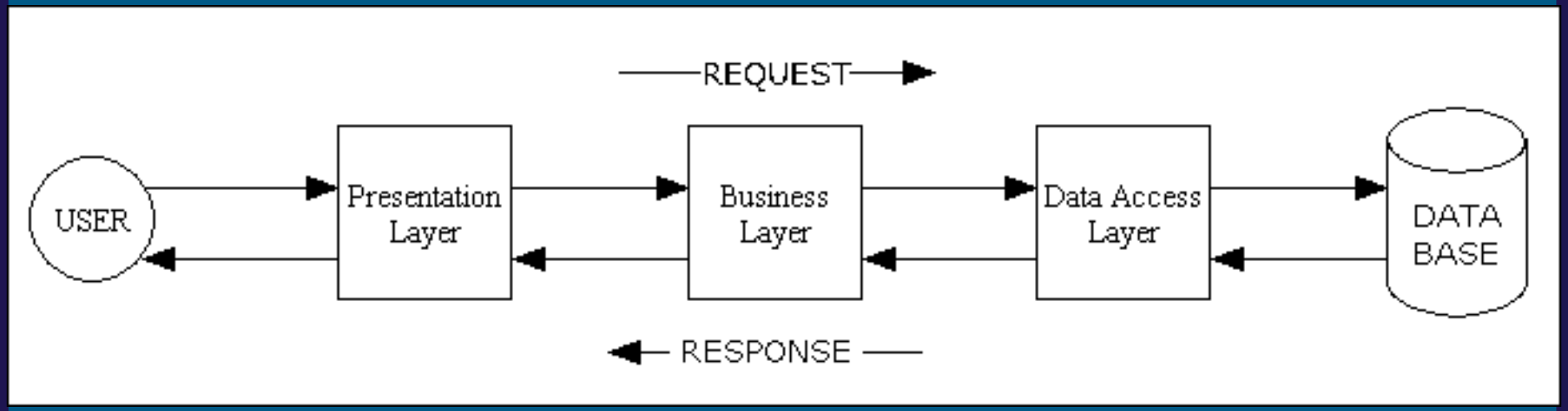
- שכבת גישה לנתונים – DAL

```
public float getWindDirection(int day) { // ... }
public float getTemperature(int day)   { // ... }
```

חוזה הנתונים של
DO כולל **float**

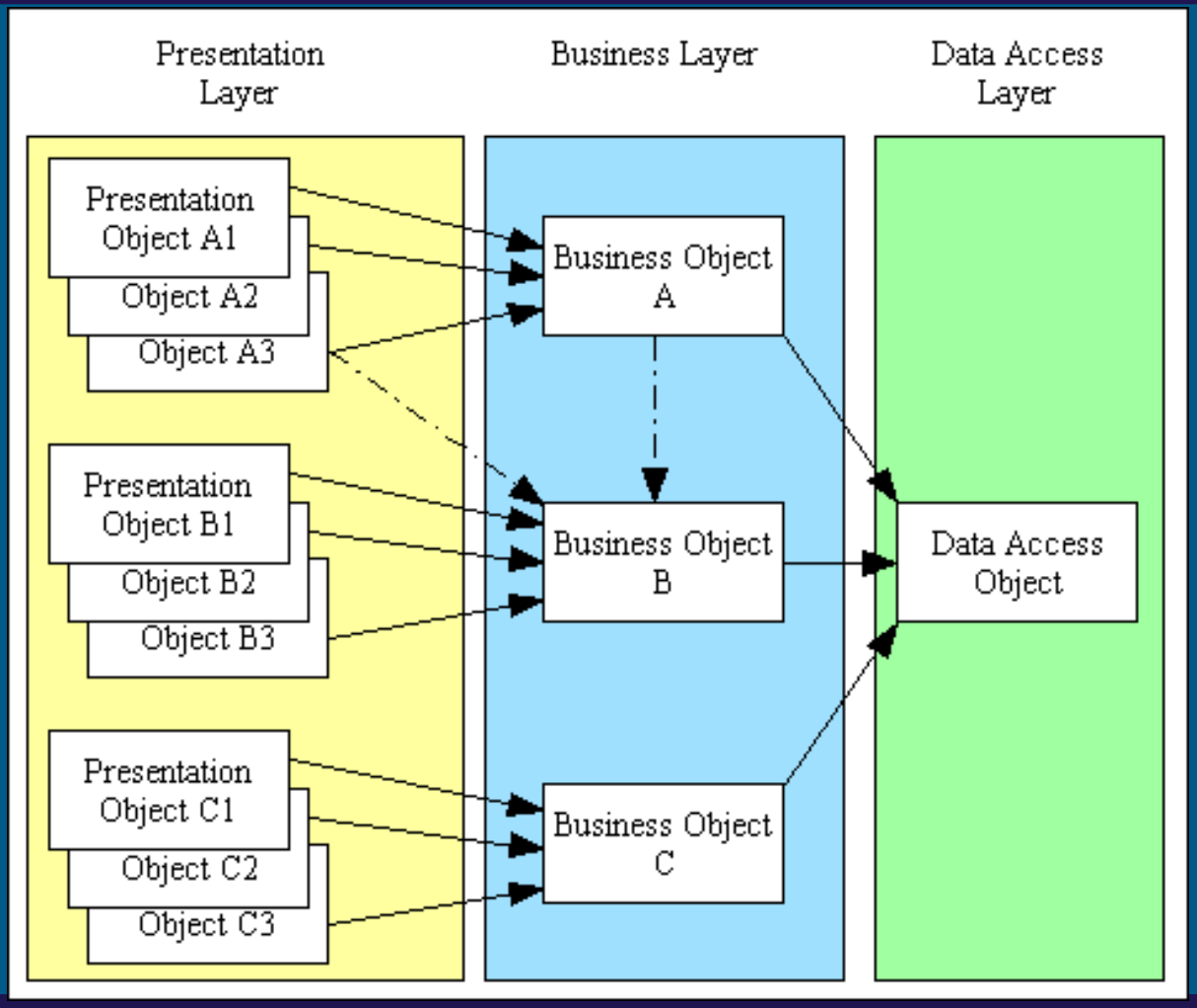
חוזה נתונים בשתי השכבות כולל סוג **int**...

שלושת השכבות בפעולה



http://www.tonymarston.net/php-mysql/3-tier-architecture.html

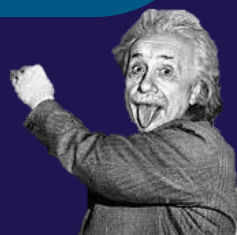
שלושת השכבות – הנתונים



http://www.tonymarston.net/php-mysql/3-tier-architecture.html

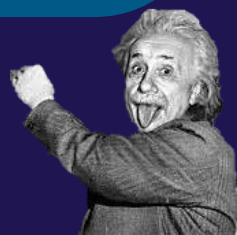
תכנון הארכיטקטורה שלכם

- איזה שכבות אתם צריכים?
- במערכות מבוזרות – איפה כל שכבה תמוקם?
- תחליטו האם כדאי למזג שכבות מסוימות
- מה הכללים בעבודה בין השכבות?
- מה הדרישות לפונקציונליות כוללת (cross-cutting)
- תגדירו אינטרפייסים בין השכבות (פונקציות ונתונים)
- מדיניות הפריסה
- פרוטוקולים של תקשורת



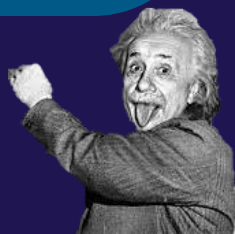
אינטרפייסים בין שכבות

- נשתמש בשלוש שכבות: PL, BL, DAL
- כל שכבה "מכירה" רק את השכבה שמתחתיה
- ממשק בין שכבות – אינטרפייס (Application Program Interface – API)
  - חוזה שרות (Service Contract) – פונקציות, פרמטרים, ערכים מוחזרים (IBl, IDal)
  - חוזה נתונים (Data Contract) – מבני הנתונים המועברים ע"י חוזה שרות (BO, DO)



מימוש שכבות בויז'ואל סטודיו

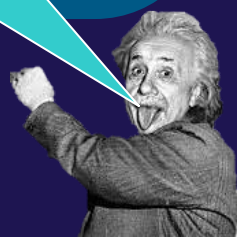
- פרויקט ממשק משתמש (קונסול\גרפי\אחר)
- פרויקטי ספריית מחלקות לשכבות לוגיקה, חוזה נתונים ונתונים (BL, DalFacade, DalList, DalXml - בהמשך)
  - לא נפתח פרויקט נפרד לממשק של BL
- קשרים בין הפרויקטים:
  - DAL תלוי ב-DalFacade
  - BL תלוי ב-DalFacade , וב-DalList
  - PL תלוי ב-BL



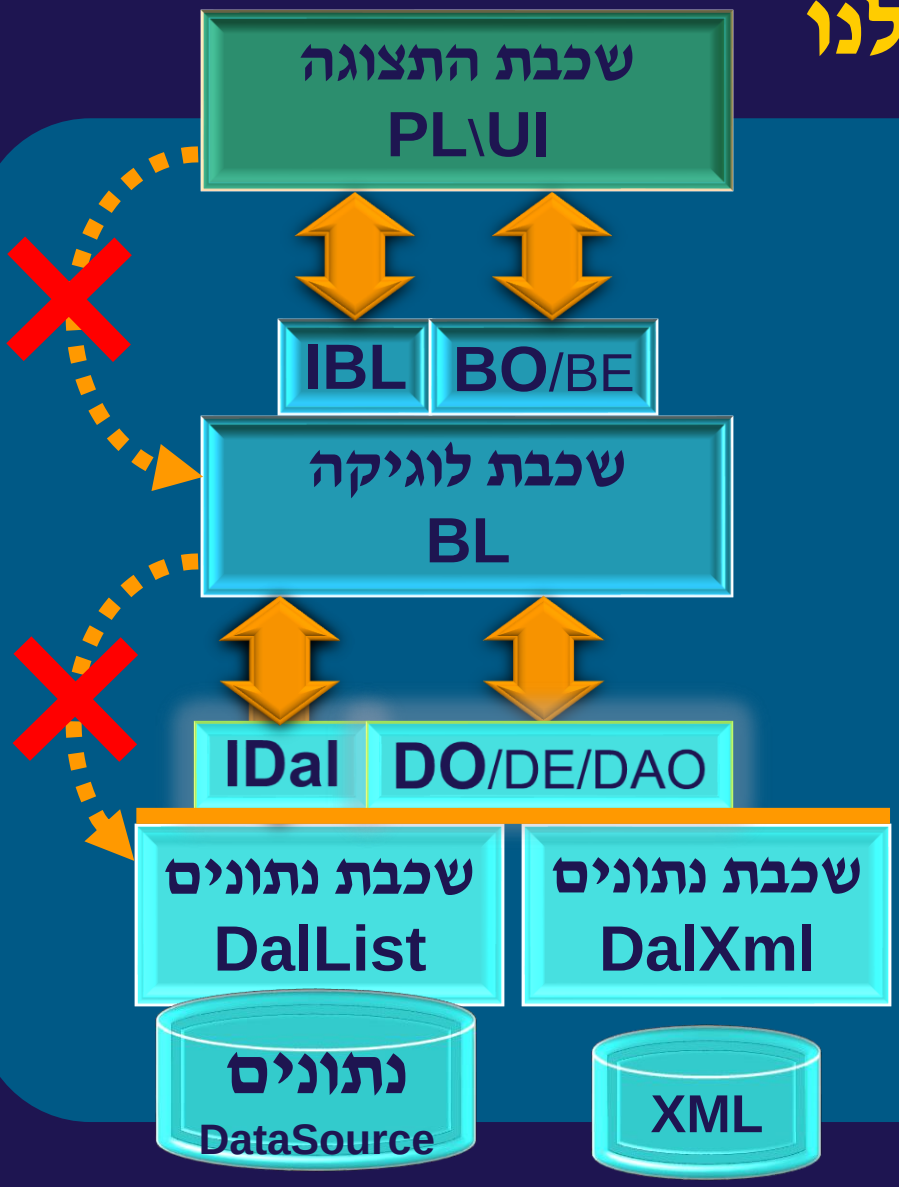
העברת נתונים בין-DAL ל-BL

- בשכבת DAL מוחזקים נתונים או חלק מהנתונים בזיכרון
- אסור לשכבה להעביר דרך API הפניות למופעי ישויות ולאוספים הנמצאים בה – שמא יקלקלו אותם "בחוץ"
- אסור לשכבה DAL לשמור מופעי הישויות שהתקבלו דרך API מ-BL בנתונים שלה – שמא המופעים ישמשו BL לצרכים נוספים

- דרך 1: שיבוט אובייקטים של ישויות DO
- דרך 2: שימוש ב-struct או ב-record עבור ישויות DO



תמונת 3TA בפרוייקט שלנו



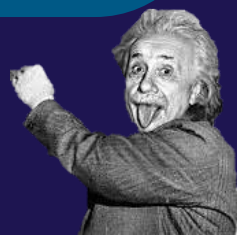
- פיתחנו שכבת נתונים DalList
  - עם חוזה נתונים DO
  - ובדקנו אותה
- הוספנו שכבה לוגית BL
  - עם חוזה נתונים BO
  - ובדקנו אותה
- הוספנו DalApi/BlApi (חוזי שירות)
- הוספנו ממשק משתמש PL
- נשפר הפרדה (עוד מעט)
- ובסוף נוסיף שכבת נתונים DalXml

Oops

כל מימושי שכבת נתונים ממשים את החוזים
של שכבת הנתונים שב-DalFacade

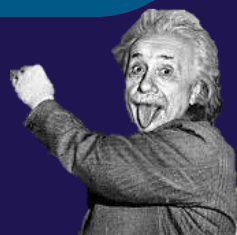
שיפור הפרדה בין שכבות

- צריך להבטיח שלמות נתונים בשכבות שמחזיקים נתונים
  - תבנית עיצוב סינגלטון
    Singleton Design Pattern
- צריך להעלים את החיבור הישיר בין המימושים
  - תבנית עיצוב יצרן
    Simple Factory Design Pattern



למה תבנית פיתוח Singleton?

- קודם כל אנחנו נרצה לדאוג שלא ייווצר יותר ממופע בודד של שכבת DAL
- לזה מיועדת תבנית פיתוח של סינגלטון
- מטרה – להבטיח שיהיה מופע אחד ויחיד של המחלקה בלבד
- ב-#C לפעמים משתמשים לצורך זה במחלקה סטטית
- אך פעמים רבות זה לא עונה על הצרכים (ירושה ממחלקה אחרת ועוד)

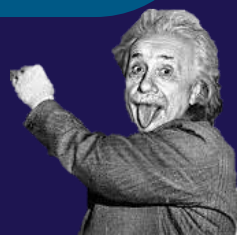


תבנית פיתוח Singleton – הגדרה

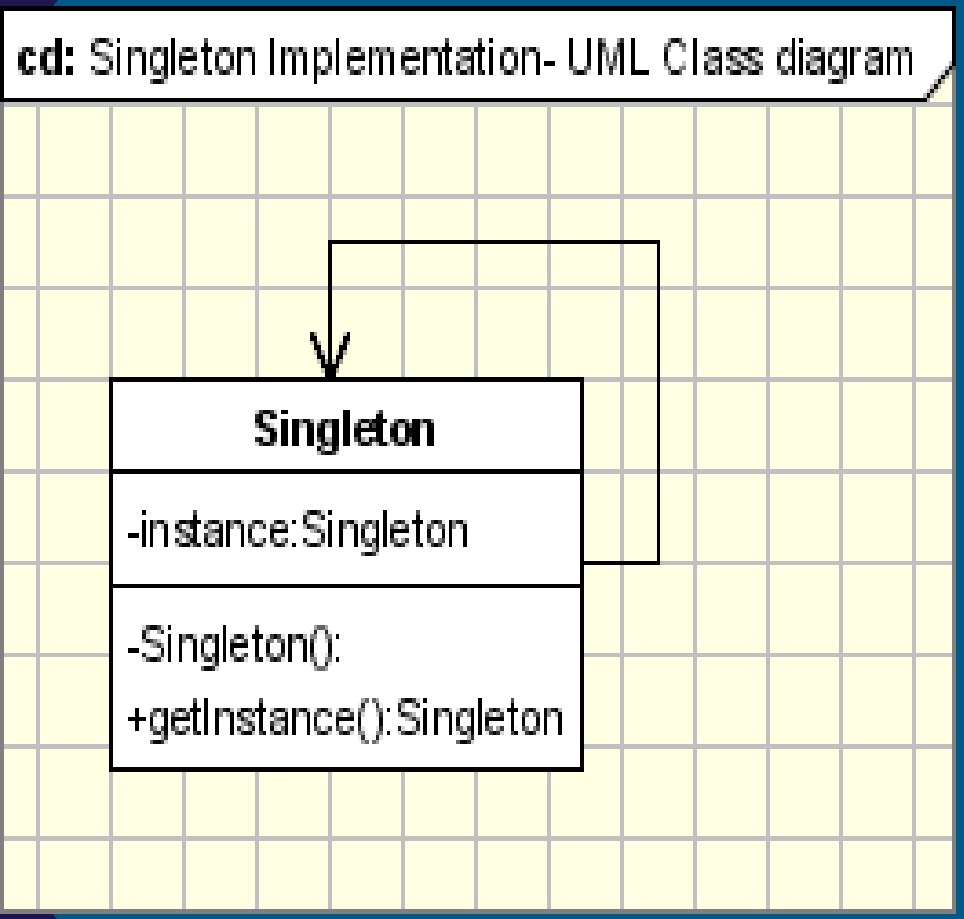
- מוטיבציה

כנ"ל – דרישה להגבלה של כמות מופעים למופע בודד. למשל מחלקה שנדרש לתאום פעולות בתוכנית או לניהול משאבים

- דרישות:
  - להבטיח שיהיה מופע אחד ויחיד
  - גישה גלובלית לקבלת המופע הנ"ל

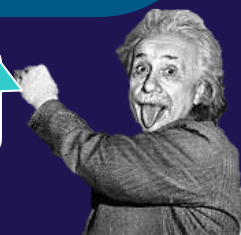


תבנית פיתוח Singleton – מימוש



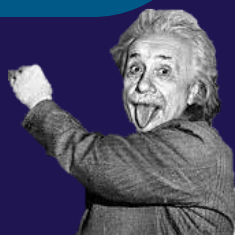
- מימוש: פונקציה שיוצרת את האובייקט ומחזירה אותו בפעם הראשונה ומחזירה את אותו האובייקט בפעמים הבאות
- פרטים:
  - בנאי פרטי
  - שדה פרטי סטטי שמכיל\מפנה\מצביע לאובייקט
  - פונקציה ציבורית כנ"ל

ובדרך כלל לא מאשרים ירושה מסינגלטון



תבנית פיתוח Singleton בסיסית ב-C#

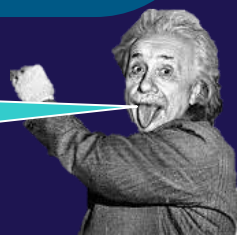
```
public sealed class MySingleton
{
    static readonly MySingleton instance = new MySingleton();

    // Explicit static constructor to ensure instance initialization
    // is done just before first usage
    static MySingleton() {}

    MySingleton() {} // default => private

    // The public Instance property to use
    public static MySingleton Instance { get => instance; }

    // Implementation specific data members and methods...
}
```



תבנית פיתוח Singleton קצרה יותר ב-C#

```
public sealed class MySingleton
{
   public static MySingleton Instance { get; }
                                        // = new MySingleton();

   // Explicit static constructor to ensure instance initialization
   // is done just before first usage
   static MySingleton() => Instance = new MySingleton(); // !!!!!
   MySingleton() {} // default => private

   // Implementation specific data members and methods...
}
```

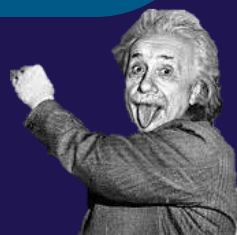
יש באג בדוט-נט ☹



תבנית פיתוח Singleton בסיסית – חסרונות

- האתחול מתבצע באתחול המחלקה
- אם מעבירים את האתחול לתוך הפונקציה – יש צורך באבטחת תהליכונים (נעילה\lock\mutex)
- אתחול המחלקה עדיין לא מספיק עצל (ראו בשקף הבא) (lazy instantiation)
- ראו דוגמה במסמך ב-moodle
  - מאפשר הגדרה פשוטה עם שתי דרישות בלבד מהמחלקה שלכם: sealed ובנאי פרטי (private)

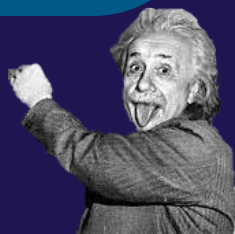
לא חובה



לא חובה

תבנית פיתוח Singleton – עצל לחלוטין

```
public sealed class MySingleton
{
    class Nested {
        static Nested() {}
        internal static readonly MySingleton s_instance
                                                = new MySingleton();
    }

    static MySingleton() {}
    MySingleton() {}
    public static MySingleton Instance { get => Nested.s_instance; }

    // Implementation specific data members and methods...
}
```



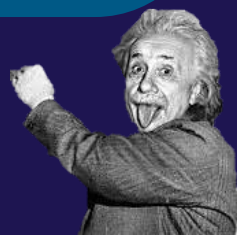
ניתוק תלות ישירה בין שכבות

- ניסיון – שימוש ישיר ויצירת אובייקט מתאים
  - בעיה – מה קורה כשרוצים להחליף שכבה?
  - שינוי קוד בשכבה שמעל...
- פתרון:

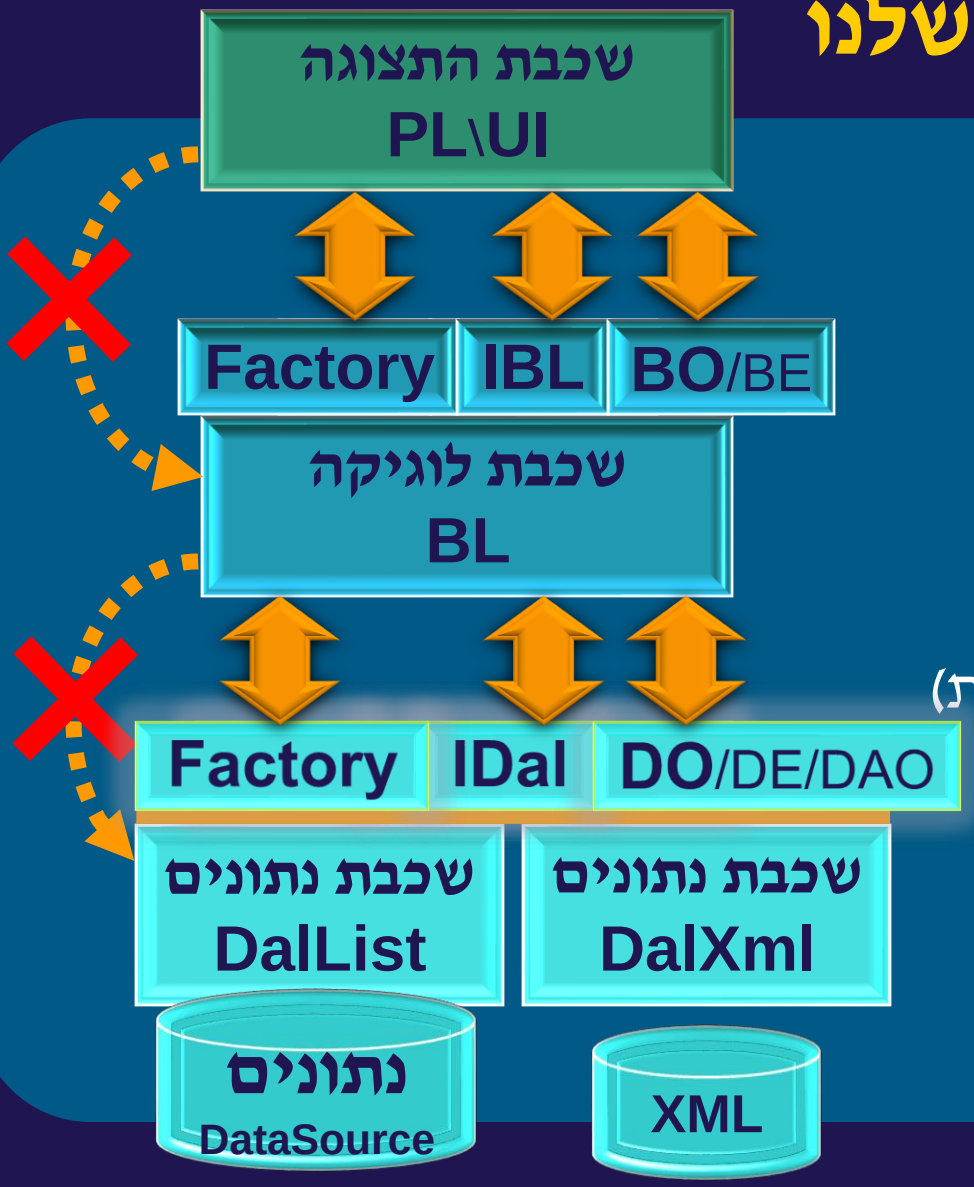
a. יצירת ממשק (interface) שכולל חוזה שרות: IBl, IDal

b. יצירת מחלקות ייצרן Factory (ב-BlApi וב-DalApi)

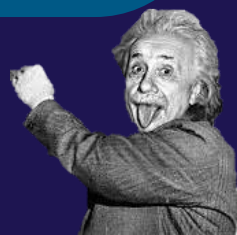
c. הפיכת פונקציית ייצור לסטטית לחיסכון ביצירת מופע של מחלקת ייצור (וגם ללא סינגלטון)



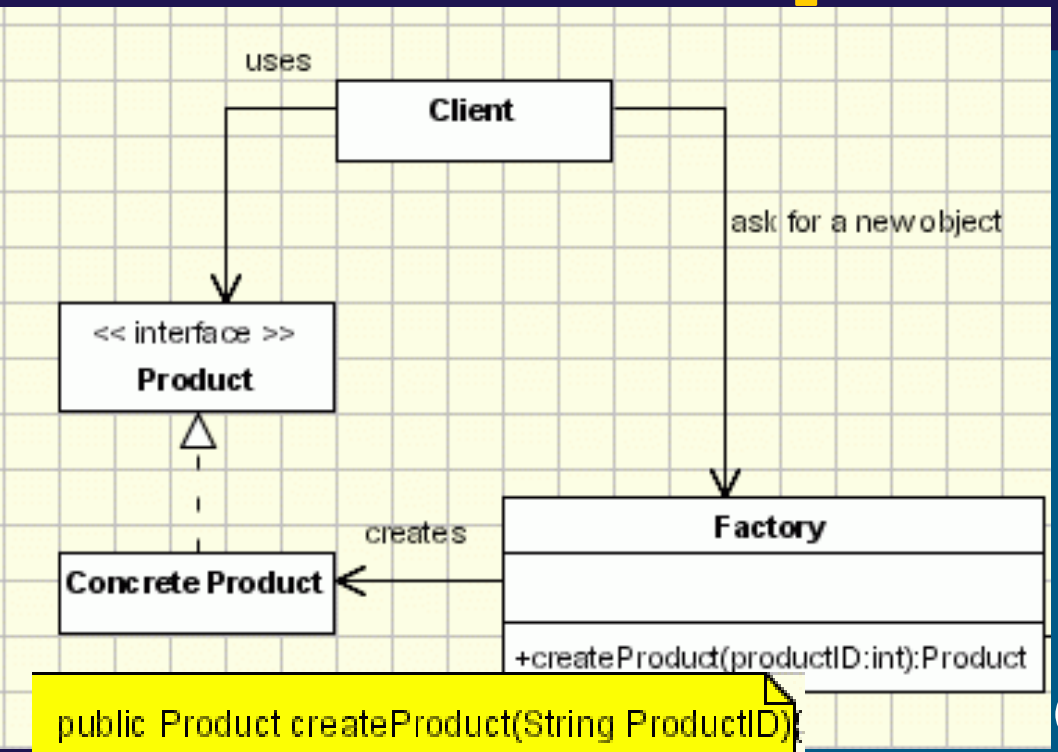
תמונת 3TA בפרוייקט שלנו



- פיתחנו שכבת נתונים DalList
  - עם חוזה נתונים DO
  - ובדקנו אותה
- הוספנו שכבה לוגית BL
  - עם חוזה נתונים BO
  - ובדקנו אותה
- הוספנו DalApi/BlApi (חוזי שירות)
- הוספנו ממשק משתמש PL
- **נשפר הפרדה (עכשיו)**



תבנית פיתוח Simple Factory

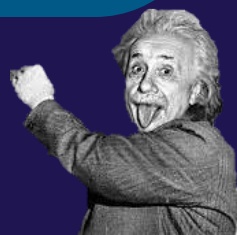



```
public Product createProduct(String ProductID){
    if (id==ID1)
        return new OneProduct();
    if (id==ID2)
        return AnotherProduct;
    ... // so on for the other Ids
    return null;
}
```

- מוטיבציה

  ניתוק השרות מהלקוחות שלו, ובמיוחד ניתוק לוגיקת ייצור המופעים של השרות

- דרישות
  - יצירת ממשק השרות (אינטרפייס, חוזה שרות)
  - יצירת מחלקת ביניים (factory) האחראית הבלעדית על יצירת המופעים



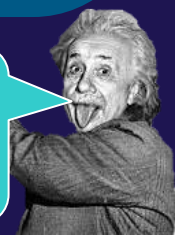
תבנית פיתוח Factory ב-C#

- נעשה מחלקה סטטית (אין צורך בסינגלטון)
- הפונקציה בדרך כלל תקבל פרמטר: enum, מספר או מחרוזת

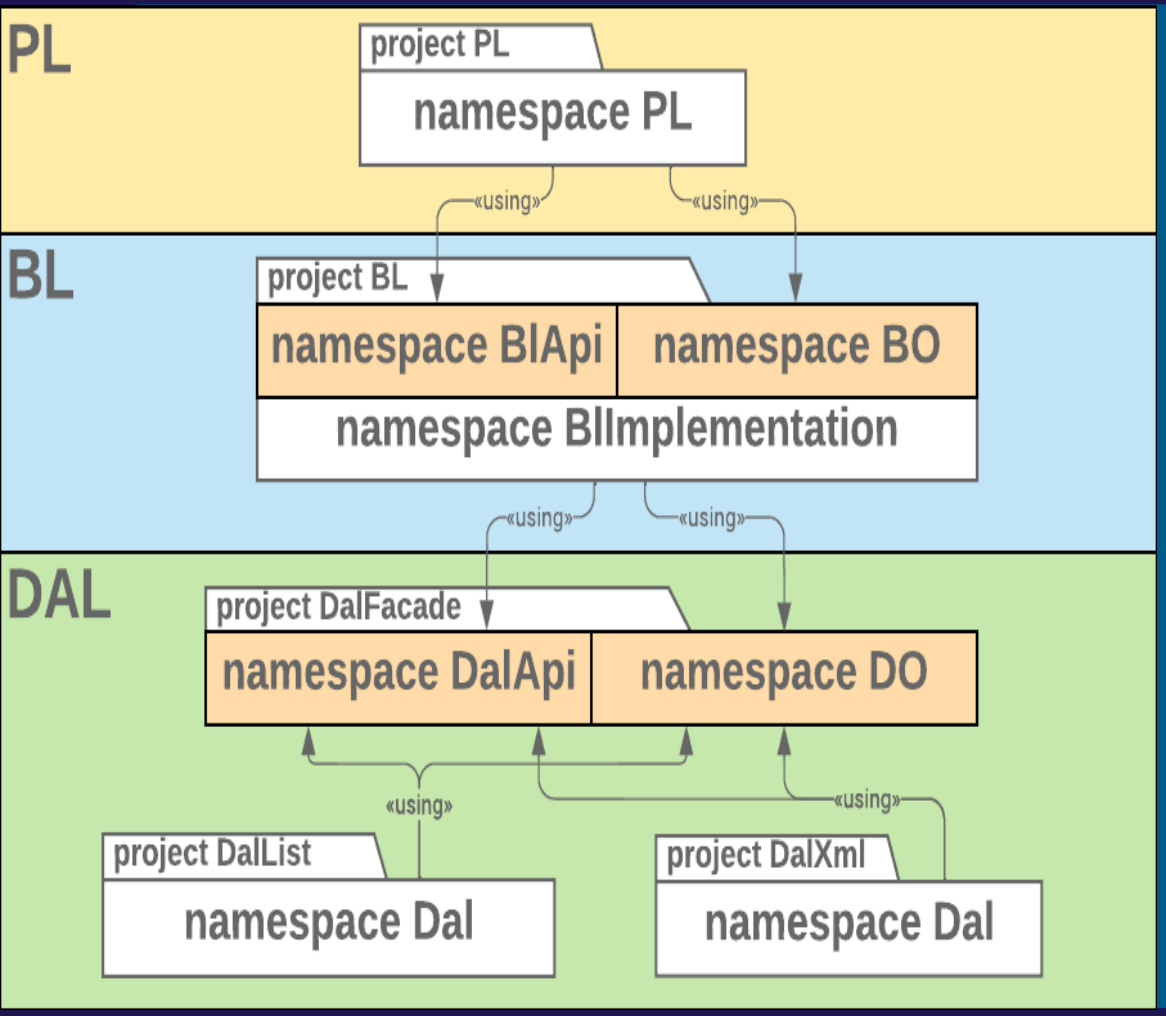
```
public static class Factory
{
   public static IDal Get(string type) => type switch
   {
      "list" => DalList.Instance;
      // "xml" => DalXml.Instance;
      _ => null;
   }
}
```

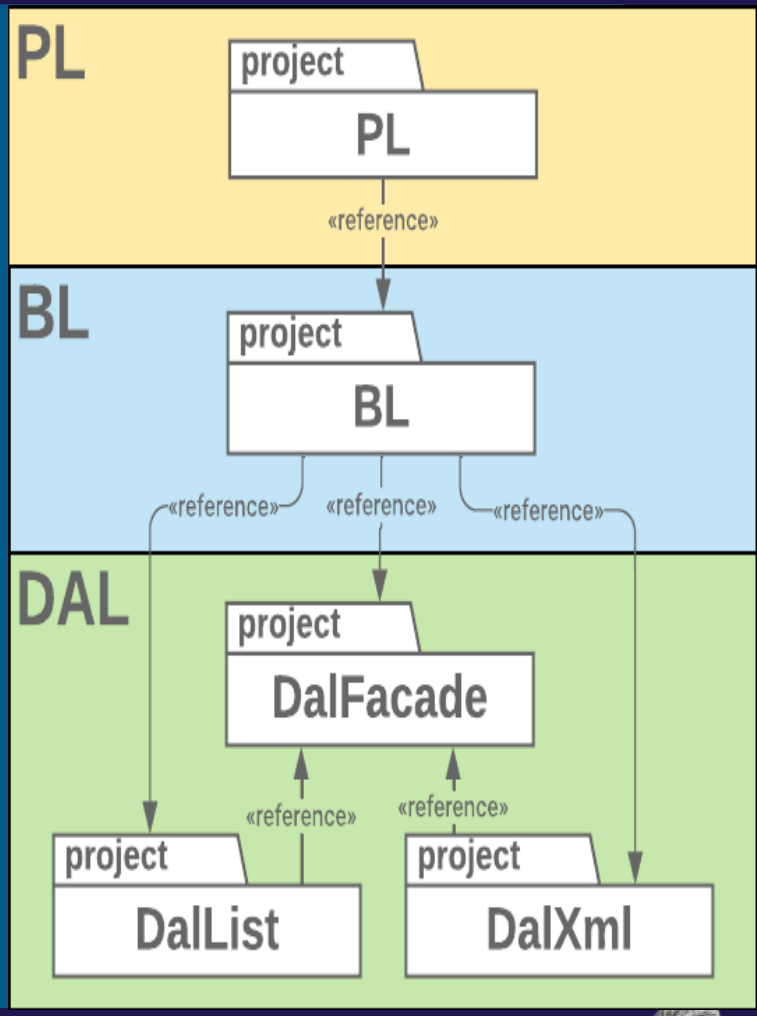
- יש בעיה של תלות מעגלית DalList ⇔ DalFacade
  - פתרון 1: DIP – שכלול תבנית פקטורי ע"י יצירת ממשק
  - פתרון 2: ניתוק הקשר בעזרת שימוש בקובץ קונפיגורציה

תבניות Factory Method ו-Abstract Factory
לא בקורס שלנו

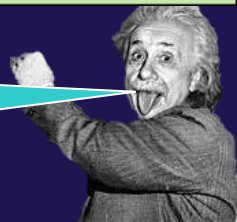


מבנה הפרויקטים ומרחבי השמות



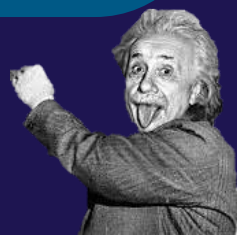


הרשאת **public** רק למחלקות של חוזים



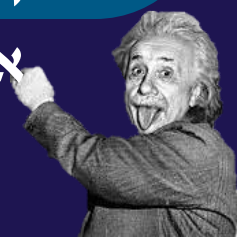
סיכום מודל השכבות: BL

- שכבת BL איננה מרובת מימושים – **היא בעצם הלב של המערכת** – מומלץ להחזיק את כל חלקיה בפרויקט אחד
- על מנת להפריד בין חוזי שרות ונתונים ומימוש השרות מומלץ לחק אותם בין מרחבי שמות שונים:
  - מרחב השמות BlApi יכיל את הממשק IBl ואת מחלקת היצירה Factory (עם הרשאות public)
  - מרחב השמות BO יכיל את הגדרות כל היישויות של חוזה הנתונים של השכבה (עם הרשאות public)
  - מרחב השמות BlImplementation יכיל את המימוש של השכבה כולל כל המחלקות הפנימיות הדרושות למימוש הפונקציונליות של השכבה (ללא הרשאות – ז"א internal)



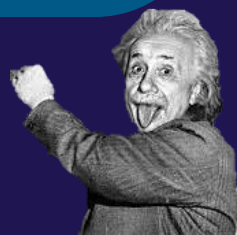
סיכום מודל השכבות: DAL

- שכבת DAL עשויה להכיל מספר מימושים – מומלץ להחזיק את כל חלקיה בפרוייקטים נפרדים
  - פרויקט DalFacade יכיל מרחבי שמות DalApi עם חוזה שרות (ממשק) IDal ומחלקת הייצור Factory, כמו כן מרחב שמות DO שיכיל את כל הגדרות היישויות של חוזה נתונים של השכבה (כל ההגדרות public)
  - פרוייקטים נפרדים לכל מימוש – כולם עם מרחב שמות DAL (ניתן להגדיר אותו כמרחב שמות ברירת מחדל של כל אחד מהפרויקטים)
    - DalList, DalXml, וכו'
  - בפרוייקטים של מימושים – הכול יוגדר ללא הרשאות (=internal)
  - פרוייקט DalList יכיל מחלקה DataSource שתחזיק רשימות כל הישויות של DO ותאתחל אותם עם מספר אלמנטים על מנת לא למלא את הנתונים מחדש בכל הרצה



שיפור: Factory של Dal

- לפי מה שהוצג קודם, Factory מייצרת בעיה של תלות מעגלית בין DalFacade ומימושי של Dal אשר מקושרים ל-DalApi (circular reference)
- נשפר את מחלקת הייצור על מנת לנתק את הקשר המעגלי
- נדרוש מכל פרויקטיי מימוש:
  - לכולם מרחב שמות ברירת מחדל בשם Dal
  - המחלקה הראשית שמממשת את IDal תהיה באותו שם כשם הפרוייקט
- כל הפרויקטים במערכת יגדירו תיקיה אחידה ברמה של Solution עבור קבצי ריצה שלהם (למשל ..\bin)

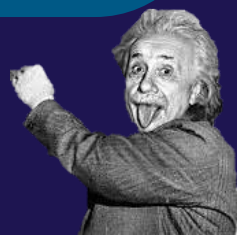


שיפור: Factory של Dal

- בתיקיית ההרצה הנ"ל ניצור גם קובץ בשם dal-config.xml
  - ע"י יצירת קובץ מסוג XML

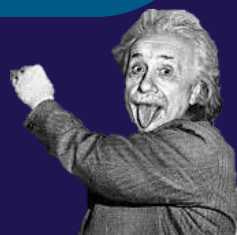
```
<config>
  <dal>list</dal>
  <dal-packages>
    <list>DalObject</list>
    <xml>DalXml</xml>
  </dal-packages>
</config>
```

- האלמנט dal-packages יחיל רשימת מימושים קיימים
- האלמנט dal יכיל את שם המימוש להרצה



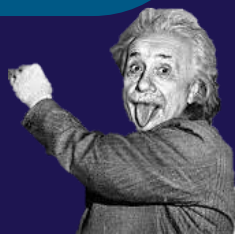
DalConfig.cs – dal-config.xml טעינת

```
namespace DalApi;
using System.Xml.Linq;

public class DalConfigException : Exception
{
   public DalConfigException(string msg) : base(msg) { }
   public DalConfigException(string msg, Exception in):base(message, in) { }
}

// continue in the next slide ...
```

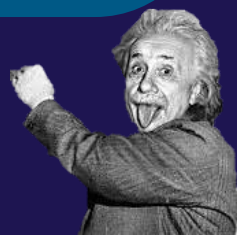
DalConfig.cs – config.xml טעינת

```
// ... continue from the previous slide
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("Failed to load dal-config.xml");
      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);
   }
}
```



Factory.cs– DAL ייצרן

```
namespace DalApi;
using System.Reflection;
using static DalApi.DalConfig
public static class Factory {
   public static IDal? Get {
      string dalType = s_dalName ??
         throw new DalConfigException($"DAL name extraction failed");
      string dalPkg = DalConfig.DalPackages[dalType] ??
         throw new DalConfigException($"Package {dalType} bad info");
      try { Assembly.Load(dalPkg); } catch (Exception) {throw new DalConfig…;}
      Type type = Type.GetType($"Dal.{dalPkg}, {dalPkg}") ??
         throw new DalConfigException(...);
      return type.GetProperty("Instance",
                                 BindingFlags.Public | BindingFlags.Static)
                 .GetValue(null) as IDal ?? throw new DalConfigException(...);
   }
}
```

לא חובה

בונוס עבור dal-config

- בפרויקט – לשכלל את DalConfig ואת Factory על מנת שיוכלו לקלוט קובץ dal-config.xml עם מידע נוסף ומפורט שמאפשר לנתק את אחידות שמות של פרוייקט\חבילה ושם מחלקת המימוש:

```
<config>
  <dal>list</dal>
  <dal-packages>
    <list class="DalObject">DalList</list>
    <xml namespace="DalXml">DalXml</xml>
  </dal-packages>
</config>
```

ב- DalObject מרחב שמות Dal וב- DalXml זהו גם שם המחלקה

לא חובה

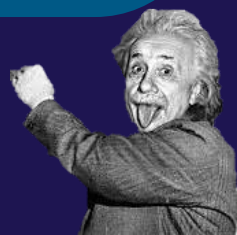
הרחבה בנושא using statement

- למשל מחיקת פקד ב-WPF מתבצע ע"י: Dispose()

```
using ( MyClass me = new MyClass(...) ) { ...... }
```

- המחלקה חייבת לממש ממשק IDisposable
- הקוד הנ"ל בפועל מבצע את הקוד הבא:

```
MyClass me = null;
try
{
   me = new MyClass(...);
   ......
}
finally
{
   if (me != null) me.Dispose();
}
```



שלב 4

- השלמת מבנה של ארכיטקטורה של מודל השכבות
- עבודה עם קובץ קונפיגורציה מול שכבת נתונים
- בתוך אותו "פתרון" עם תרגילים קודמים
- לפי ההנחיות
- חובה, עבודה באותם הזוגות