שלב 4 ב – בניית שכבת BL והפרדה בין השכבות

1. בניית שכבת הנתונים BL

השכבה הלוגית, המכונה BL = Business Layer, מנהלת את כל הלוגיקה של המערכת שאנחנו בונים. כל הכללים של המערכת ממומשים בה. השכבה תתקשר כלפי מטה עם שכבת נתונים (DAL) על מנת לאחזר את הנתונים ולעדכנם. כמו כן השכבה תתקשר עם שכבת התצוגה על מנת לעבד את הקלט ולספק נתונים של קלט עבור ממשק המשתמש.

השכבה הלוגית תהיה אחראית על:

* הגדרת טיפוסים של ישויות לוגיות לבקשת האפליקציה (ע"י ממשק המשתמש)
* הלוגיקה שמאחורי נתונים:
  + בדיקת תקינות והתאמת נתוני הקלט שהגיעו מהתצוגה
  + בדיקת התאמת נתוני קלט ע"פ המידע בשכבת הנתונים
  + שלמות והתאמה של כל הנתונים המאוחסנים בשכבת נתונים
  + ביצוע חישובים והפקת תוצאות מהנתונים היבשים שהגיעו מהשכבה שמתחתיה.

בשלבים הבאים, נוסיף 2 פרויקטים ל Solution:

* פרויקט בשם BL מסוג Class Library שיהווה את שכבת הנתונים (BL = Business Layer) ויכיל:
  + BO - נגדיר את חוזה הנתונים של השכבה הלוגית - הגדרת הישויות הלוגיות והתכונות (properties) שלהם
  + BlApi - נגדיר את חוזה השירות של השכבה הלוגית - הגדרת הממשקים (כנגד כל ישות לוגית ראשית), שיכילו מתודות לצורך גישה לכל ישות לוגית
  + נממש את ממשקי הגישה לישויות הלוגיות (מימוש מתודות הממשקים עבור כל סוג של ישות לוגית ראשית במחלקות מתאימות)
* פרויקט בשם BlTest מסוג Console Application שיהווה תוכנית בדיקה ידנית (manual test program) של השכבה הלוגית

1. הוספת פרויקט BL מסוג Class Library

* הוסיפו ל **Solution** פרויקט חדש בשם **BL** מסוג **Class Library**.

לאחר שייפתח פרויקט חדש תחת ה solution, לחצו לחיצה כפולה על שם הפרויקט ב-Solution Explorer.

ייפתח לעריכה קובץ **לניהול המאפיינים של הפרויקט,** הוסיפו לקובץ את השורות הבאות (המודגשות בצהוב) בסוף האלמנט **PropertyGroup**:

*<Project Sdk="Microsoft.NET.Sdk">*

*<PropertyGroup>*

*<TargetFramework>net7.0</TargetFramework>*

*<ImplicitUsings>enable</ImplicitUsings>*

*<Nullable>enable</Nullable>*

**<BaseOutputPath>$(SolutionDir)\bin\</BaseOutputPath>**

**<OutputPath>$(SolutionDir)\bin\</OutputPath>**

**<AppendTargetFrameworkToOutputPath>false</AppendTargetFrameworkToOutputPath>**

**<AppendRuntimeIdentifierToOutputPath>false</AppendRuntimeIdentifierToOutputPath>**

**<UseCommonOutputDirectory>true</UseCommonOutputDirectory>**

**<RootNamespace />**

*</PropertyGroup>*

*</Project>*

בצעו שמירה (**Save**) של הקובץ

* **הוסיפו reference** מפרויקט BL לפרויקטים DalFacade ו DalList  
  ודאו ב**קובץ המאפיינים של הפרויקט** שהטקסט שלו התעדכן לאחר הוספת הרפרנס:

*<ItemGroup>*

*<ProjectReference Include="..\DalFacade\DalFacade.csproj"/>*

*<ProjectReference Include="..\DalList\DalList.csproj"/>*

*</ItemGroup>*

תזכורת: ה reference לפרויקט DalList הוא רפרנס מאולץ.שנועד רק כדי לגרום לפרויקטים הללו להיבנות באופן אוטומטי כאשר בונים את פרויקט BL, וקיומם באותה ספריית הרצה.

1. הוספת פרויקט BlTest מסוג Console Application

* הוסיפו ל **Solution** פרויקט חדש בשם **BlTest** מסוג **Console Application**.

לאחר שייפתח פרויקט חדש תחת ה solution, לחצו לחיצה כפולה על שם הפרויקט ב-Solution Explorer.

ייפתח לעריכה קובץ **לניהול המאפיינים של הפרויקט,** הוסיפו לקובץ את השורות הבאות (המודגשות בצהוב) בסוף האלמנט **PropertyGroup**:

*<Project Sdk="Microsoft.NET.Sdk">*

*<PropertyGroup>*

*<OutputType>Exe</OutputType>*

*<TargetFramework>net7.0</TargetFramework>*

*<ImplicitUsings>enable</ImplicitUsings>*

*<Nullable>enable</Nullable>*

**<BaseOutputPath>$(SolutionDir)\bin\</BaseOutputPath>**

**<OutputPath>$(SolutionDir)\bin\</OutputPath>**

**<AppendTargetFrameworkToOutputPath>false</AppendTargetFrameworkToOutputPath>**

**<AppendRuntimeIdentifierToOutputPath>false</AppendRuntimeIdentifierToOutputPath>**

**<UseCommonOutputDirectory>true</UseCommonOutputDirectory>**

*</PropertyGroup>*

*</Project>*

בצעו שמירה (**Save**) של הקובץ

* **הוסיפו reference** מפרויקט BlTest לפרויקט BL  
  ודאו ב**קובץ המאפיינים של הפרויקט** שהטקסט שלו התעדכן לאחר הוספת הרפרנס:

*<ItemGroup>*

*<ProjectReference Include="..\BL\BL.csproj"/>*

*</ItemGroup>*

* כעת סמנו את פרויקט BlTest להיות הפרוייקט ההתחלתי של ה solution   
  (set as startup project)

1. יצירת 3 תתי ספריות תחת פרויקט BL

**בשלבים 1-3, כשעיצבנו את הארכיטקטורה של שכבת הנתונים (DAL), הפרדנו את החוזים (DO/DalApi) והמימוש שלהם לפרויקטים נפרדים**. את החוזים הגדרנו בפרויקט DalFacade ואת המימוש שלהם הגדרנו בפרויקט DalList.

**לעומת זאת, כעת בשלב 4,** **הארכיטקטורה של השכבה הלוגית** (BL), נגדיר את החוזים **(BO/BlApi)** והמימוש באותו פרויקט בשם BL.

**לצורך כך, הוסיפו לפרויקט BL את 3 תתי-הספריות הבאות:**

1. תת-ספריה בשם **BO** - עבור הישויות לוגיות
2. תת-תיקייה בשם **BlApi** - עבור ממשקי השכבה הלוגית
3. תת-תיקייה בשם **BlImplementation** - עבור **המימוש** של מתודות הממשק של השכבה הלוגית

**תזכורת ליצירת תת-ספרייה:** עמדו עם עכבר ימני על פרויקט **BL**   
ואז לחצו על **Add → New Folder** וקראו לספריה בשמה

1. הוספת קובץ אנומרציות לספריה BO

מלבד קבצי הגדרת הישויות הלוגיות, בספריה זו גם נשמור גם את קובץ האנומרציות (enum) הנחוצות בהגדרת הישויות הלוגיות.

1. הוסיפו לספריה BO "מחלקה" בשם Enums.cs.
2. רוקנו את הקובץ כך שתישאר בו רק השורה: **namespace BO;**

שימו לב - הגדרת האנומרציות בשכבה BL נפרדת מהאנומרציות בשכבת DAL, גם אם חלקן יהיו זהות. בהמשך פרק זה, כאשר תגדירו את הישויות הלוגיות עצמן, תמלאו את הקובץ בהגדרות של enums על פי הצורך.

1. בניית הישויות הלוגיות

הישויות הלוגיות יעברו בין שכבת התצוגה העליונה לבין השכבה הלוגית שמתחתיה. הם יעברו ב 2 הכיוונים. ישות לוגיות מוגדרת כך שהיא תארוז מידע עבור הצרכים של שכבת התצוגה (החלונות השונים).

תחילה נגדיר **ישויות לוגיות ראשיות**:

**ישות לוגית ראשית** שנגדיר בשכבה הלוגית מבטאת ישות מרכזית בתצוגה.

* **לחלק** מהישויות הלוגיות הראשיות יש **ישות נתונים ש"מקבילה"** לה. והיא תכיל בגדול את אותם התכונות **עם תוספות של תכונות על פי הצורך**.  
  לדוגמא: **ישות לוגית ראשית Student**, עבור מסך בתצוגה שמטפל בפעולות שניתן לעשות על סטודנט בודד
* **לחלק** מהישויות הלוגיות הראשיות, **אין ישות נתונים "מקבילה"**, היא מכילה מידע שנאסף מבסיס נתונים מכמה ישויות נתונים והיא נחוצה לצורך התצוגה.  
  לדוגמא: **ישות לוגית ראשית GradeSheet**, עבור מסך בתצוגה שמציג גיליון ציונים לסטודנט, ע"פ נתוני הסטודנט בשילוב עם נתוני הקורסים שהוא רשום אליהם.

לאחר מכן, **נגדיר ישויות עזר לוגיות** לצרכי התצוגה. למשל ישות מצומצמת עבור הצגה של ישות לוגית ראשית ברשימה של ישויות עזר לוגיות. **יש יותר ישויות לוגיות מישויות נתונים. הישויות הלוגיות הן תוצר של הסקת מסקנות מהקשרים שבין ישויות הנתונים ושאילתות על המידע הכולל בבסיס נתונים.**

**השכבה הלוגית לא שומרת אצלה שום נתונים ולא צריכה להגן עליהם במעבר בין השכבות.** כלומר, המופעים של הישויות הלוגיות בשכבה זו לא נשמרים בשום בסיס נתונים אלא מגיעים מוכנים משכבת התצוגה כפרמטרים של מתודות. בתוך המתודות מרכיבים מהם אובייקטים מסוג ישויות נתונים. ואותם מעבירים כלפי מטה לשכבת הנתונים. לכן, הישויות הלוגיות יוגדרו כ reference types מסוג **class**.

כללים להגדרת הישויות:

* כל ישות תוגדר במחלקה נפרדת, בקובץ נפרד, תחת ספריית BO.
* שם המחלקה כשם הישות ומתחיל באות גדולה.
* הרשאת גישה public
* כל התכונות יהיו properties ציבוריים (public) ולא שדות
* יש לחשוב לגבי כל תכונה האם היא צריכה להיות nullable או לא.
* תכונה שהיא מסוג enum יש להגדיר עבורה enum מתאים בקובץ האנומרציות (בסעיף הבא)
* תכונה שצריכה להיות לקריאה בלבד, תוגדר עם init.
* אסור להוסיף למחלקה שום פונקציות, למעט דריסה של ToString. (מימוש Tostring יתבצע ע"י reflaction – יוסבר בהמשך השלב).

דוגמא אפשרית להגדרת הישויות ניתן לראות בתיאור הכללי של הפרויקט.

1. בניית הממשקים של השכבה הלוגית

**מטרת הממשקים של השכבה הלוגית היא לרכז ולהצהיר על הפעולות/הפונקציונליות שהשכבה הלוגית מאפשרת עבור שכבת התצוגה שמעליה. הממשקים שנגדיר:**

* **ממשק לוגי ראשי** בשם **IBl** שירכז את הגישה **לתתי-ממשקים לוגיים**
* **תתי-ממשקים לוגיים**. תת-ממשק **עבור כל ישות לוגית** ראשית, אשר יכלול את המתודות הקשורות לאותה ישות לוגית ראשית.

1. הגדרת תתי הממשקים עבור הישויות הלוגיות הראשיות

כל הממשקים הלוגיים יוגדרו תחת ספריית BlApi  
עבור כל ישות לוגית **ראשית** יוגדר interface נפרד ששמו כשם הישות בתוספת I בהתחלה.  
הרשאת הגישה public.  
בכל ממשק יוגדרו מתודות המתאימות לפונקציונאליות שהוא מאפשר.  
המתודות לא חייבות להיות זהות בין ישות לישות.  
אין ממשק מתודות מוסכם כמו שהיה בשכבת DAL.

דוגמא אפשרית להגדרת הממשקים ניתן לראות בתיאור הכללי של הפרויקט.

1. הגדרת ממשק לוגי ראשי בשם IBl

**המטרה**: הוספת ממשק לוגי ראשי בשם **IBl** שירכז את כל ממשקי השכבה הלוגית (תתי הממשקים)  
הוסיפו לספריית BlApi ממשק בשם IBl. הרשאת גישה public.  
הגדירו בו תכונה ציבורית, עם get בלבד, עבור כל אחד מתתי הממשקים שהגדרתם בסעיף הקודם.

1. **הגדרת מחלקת** Tools **עבור מתודות עזר לשכבה הלוגית**

נרצה לייצר **קובץ עבור מתודות עזר** שונות, שאולי נצטרך בהמשך.  
לצורך כך הוסיפו לספרית BO מחלקה סטטית בשם **Tools בהרשאת internal**, בקובץ נפרד.  
לדוגמא, ניתן להוסיף במחלקה זו פונקציות המרה מטיפוסי DO לטיפוסי BO ולהיפך.

**מתודה שיש לכלול במחלקה זו - מתודת הרחבה סטטית בשם ToStringProperty אשר תחקור** כל טיפוס T בזמן ריצה בעזרת **Reflection ו**תייצר **מחרוזת** מכל התכונות שלו **שנחקרו** בזמן ריצה.

המתודה תזומן מתוך כל **העמסה** של **ToString** בכל **ישות BO**:

**public override string ToString() => this.ToStringProperty();**

מכיוון **שישויות BO אינן שטוחות** ויכולות להכיל תכונות שהם אוסף, אז המתודה צריכה לדאוג **להיכנס לעומק התכונות הללו** ולהחזיר מחרוזת שכוללת את איברי הרשימה כחלק מערכי התכונות של הישות.

בהמשך, כל פעם שתרגישו צורך בהוספת מתודת עזר, הוסיפו אותן כאן.

1. מימוש השכבה הלוגית

כללים למימוש נכון של מתודות תתי הממשקים:

1. **המלצה חמה** - אל תוסיפו בראש קובץ המימוש using namespace BO/DO, על מנת שתאולצו לציין את מרחב השמות DO/DO לפני כל שם של טיפוס ישות לוגית או ישות נתונים. במתודות הללו עושים שימוש ב 3 סוגי הישויות והם בעלי שמות זהים ולכן זה יכול מאוד לבלבל. כך יקל עליכם להבדיל בין ישות לוגית לישות נתונים. למשל: להבדיל בין BO.Student לבין DO.Student. לפעמים, תוך כדי העבודה מרחב השמות נוסף אוטומטית, זה יכול לקרות כמה פעמים, המלצה חמה - תמיד להסיר אותו.
2. **השכבה הלוגית לא שומרת אצלה נתונים של ישויות BO ולכן לא צריכה להגן עליהם במעבר בין השכבות.** כלומר, **המופעים המקומיים** של הישויות לוגיות שתיצרו בתוך המתודות שממשות את תתי-הממשקים בשכבה זו לא נשמרים בשום בסיס נתונים, אלא מחושבים מקומית בתוך המתודות ואז מועברים לטיפול לשכבה שמעל או לשכבה שמתחת. לכן, הישויות הלוגיות הוגדרו כ reference types מסוג **class**.
3. כל מימוש יחזיק שדה פרטי מסוג IDal שדרכו יפנו המתודות לממשקי ה DAL השונים ויפעילו מתודות מתאימות
4. המתודות יפנו לשכבה התחתונה דרך ממשקי DAL, יקבלו ממנה אובייקטים של DO ויהפכו אותם לאובייקטים של BO שאותם יחזירו כערך חוזר של המתודה, כלפי מעלה לשכבת התצוגה PL.
5. ובכיוון ההפוך, המתודות יקבלו כפרמטרים אוביקטים של BO מהשכבה העליונה PL, יהפכו אותם לאובייקטים של DO שאותם ישלחו דרך ממשקי ה DAL כדי לתשאל/לעדכן את בסיס הנתונים.
6. המתודות יכולות להחזיר ערכים מטיפוסים של דוט-נט ושל BO וכן אוספים של BO
7. במימוש המתודות יעשה שימוש, בתחביר LINQ בעזרת שתי האפשרויות שלמדנו למימוש: מתודות הרחבה ותחביר שאילתות linqToObject  
   חובה להשתמש ב 2 האפשרויות הנ"ל עם לפחות 2 מופעים מכל סוג!
8. במימוש המתודות, אסור להשתמש בלולאת foreach במקום שבו ניתן להשתמש בתחביר שאילתות ו-\או במתודות הרחבה (רצוי בשרשור)
9. חובה להשתמש בביטויי למבדה במקומות המתאימים.
10. בשכבת BL אסור להחזיק אובייקטים או אוספי אובייקטים של ישויות מכל סוג (ישויות נתונים או ישויות לוגיות אפילו לא ישויות BO) אלא רק פעולות!! האובייקטים נשמרים ב שכבת ה DAL בסופו של דבר.
11. המתודות יעטפו קריאות ל DAL בבלוק try/catch ויתפסו **חריגות DO**. חריגות DO ישלחו **כחריגות פנימיות בתוך חריגות BO חדשות** שיזרקו כלפי מעלה לתצוגה.
12. הגדרת מחלקה מממשת עבור כל תת ממשק לוגי

עבור כל תת ממשק שהגדרתם בסעיפים הקודמים, הגדירו בספריית BlImplementation מחלקה שתממש את המתודות שלו.  
שמות המחלקות יהיו כשמות הישויות, בתוספת Implementation  
הרשאת גישה **internal**.  
הגדירו את המחלקה כיורשת של הממשק המתאים  
הוסיפו למחלקה שדה private מסוג IDal, בשם \_dal, דרכו תוכלו לזמן את המתודות הנדרשות משכבת DAL: private DalApi.IDal \_dal = DalApi.Factory.Get;  
כעת ממשו את המתודות של הממשק לפי הכללים שצוינו.

דוגמא אפשרית למימושים ניתן לראות בתיאור הכללי של הפרויקט.

1. הגדרת מחלקת Bl שתממש את הממשק IBl

תחת ספריית BlImplementation, צרו מחלקה בשם **Bl**, בהרשאת **internal** שתירש מהממשק הלוגי הראשי, **IBl**.  
ממשו את התכונות של הממשק כך שיחזירו מופעים של מחלקות המממשות שהגדרתם בסעיף הקודם.  
לדוגמא: public IStudent Student => new StudentImplementation();

1. הגדרת מחלקת Factory עבור שכבת BL

כזכור, בתחילת שלב זה הגדרנו מחלקת **Factory** עבור שכבת הנתונים DAL. כעת, נרצה להגדיר מחלקה דומה עבור השכבה הלוגית BL.

נרצה ששכבת התצוגה, אשר היא מעל השכבה הלוגית, תוכל **לייצר אובייקטים מסוג Bl מבלי להכיר את המחלקה המממשת BlImplementation** אלא רק את **הממשק המוסכם IBl**.

**המחלקת Factory שהגדרנו בשכבת נתונים DAL**, קראה את שם המחלקה המממשת שעליה לאתחל **מקובץ קונפיגורציה.**

**לעומת זאת, בשכבה הלוגית BL, ניצור מחלקת Factory פשוטה יותר** שמאתחלת את המחלקה המממשת **במפורש דרך הקוד**. כלומר, השכבה שמעל לא תכיר את המחלקה המממשת, אך השכבה שמתחת תצטרך להתקמפל מחדש במידה והמחלקה המממשת תשתנה.

**לצורך כך, בפרויקט BL, הגדירו "מפעל לייצור אובייקטים":**

1. **תחת ספריית BlApi צרו מחלקה public סטטית בשם Factory.**
2. **הגדירו בתוכה פונקציה public סטטית בשם Get שתחזיר מופע מטיפוס המממש את הממשק IBl. (בפועל יוחזר מופע של מחלקת Bl)**
3. **הגדרת חריגות עבור השכבה הלוגית**

**שכבת BL תתפוס חריגות שיזרקו משכבת DAL , תייצר חריגות מתאימות שיוגדרו בשכבת BL ותזרוק אותן כלפי שכבת התצוגה שמעליה.  
החריגה של שכבת BL תכלול בתוכה את החריגה שהתקבלה משכבת DAL כחריגה פנימית.**

**יש להגדיר תחת ספריית BO מחלקות חריגה מקבילות למחלקות שהוגדרו בשכבת DO (לדוג' – מזהה לא קיים).  
בנוסף, יש להגדיר חריגות עבור תקלות פנימיות של שכבת BL (לדוג' – קלט לא תקין).**

**תחת ספריית BO, צרו קובץ ("מחלקה") בשם Exceptions.cs  
מחקו את כל תוכן המחלקה, פרט להגדרת ה namespace.  
בקובץ זה הוסיפו מחלקות עבור כל סוגי החריגות הנדרשות.  
הגדרת המחלקות:**

* מכיוון שהחריגות שתגדירו כעת שייכות לשכבת ה BL, **שם הטיפוס החדש של החריגה יתחיל ב- BL**
* טיפוס החריגה החדש יירש ממחלקת Exception
* מעל כל חריגה נוסיף את האטריביוט [Serializable]
* לחריגה יוגדר בנאי שמקבל כפרמטר את הודעת השגיאה ושולח אותה לבנאי של מחלקת האב.
* **בנוסף, יוגדר בנאי שמקבל שני פרמטרים: הודעת שגיאה וחריגה פנימית. הבנאי ישלח את שני הפרמטרים לבנאי של מחלקת האב.**

דוגמא:

namespace DO;

[Serializable]

public class BlDoesNotExistException : Exception

{

public BlDoesNotExistException(string? message) : base(message) { }

public BlDoesNotExistException(string message, Exception innerException)

: base(message, innerException) { }

}

1. **בניית תכנית בדיקות – BlTest**

**בפרוייקט BlTest שיצרנו, נבנה תוכנית בדיקות במבנה דומה לתוכנית הבדיקות של שכבת DAL.**

**התוכנית תבקש קלט מהמשתמש, תפעיל פונקציות BL מתאימות ותדפיס את הפלט למסך.**

**שימו לב שבנוסף לאפשרויות התפריט שהיו בשכבת DAL, כאן יש לאפשר שימוש בכל הפונקציות שהגדרתם בממשקים של שכבה זו.**

**כדי "לחסוך" בשכבה זו אתחול של רשימות הנתונים, ניתן להוסיף reference לפרוייקט DalTest וכך לזמן מהתוכנית הראשית את פונקציית האתחול שהגדרנו בשכבת DAL.**

**בנוסף, התוכנית הראשית תגדיר שדה עבור הממשק הראשי של השכבה הלוגית: IBl.  
static readonly BlApi.IBl s\_bl = BlApi.Factory.Get();**

**דרך שדה זה תוכלו להפעיל את המתודות השונות שהגדרתם ומימשתם בשלב זה.**

**פירוט הדרישות לתוכנית הבדיקות:**

**אין צורך לאפשר פעולות CRUD.**

**רק ביצוע הזמנה:**

**קליטת ת.ז. של לקוח (או 0 ללקוח מזדמן)**

**אח"כ קליטת המוצרים להזמנה: לכל מוצר מזהה וכמות להזמנה.**

**לאחר כל קליטת מוצר יודפסו המבצעים שהתקבלו למוצר (אם יש) והסכום הסופי לתשלום להזמנה עד עכשיו.**

**אח"כ מאפשרים הוספת מוצר נוסף או סיום ההזמנה.**

**בסיום ההזמנה מאפשרים ביצוע הזמנה נוספת או יציאה.**