בס"ד תשפ"ג סמסטר א'

מיני פרויקט במערכות חלונות

שלב 2

מטרות השלב

עבודה עם רשימות ואוספים כלליים

עבודה עם ממשקים

עבודה עם מחלקות\ממשקים גנריים

חריגות

יצירת שכבה לוגית בסיסית

הנחיות לביצוע התרגיל והגשתו

* 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 כל שלב.

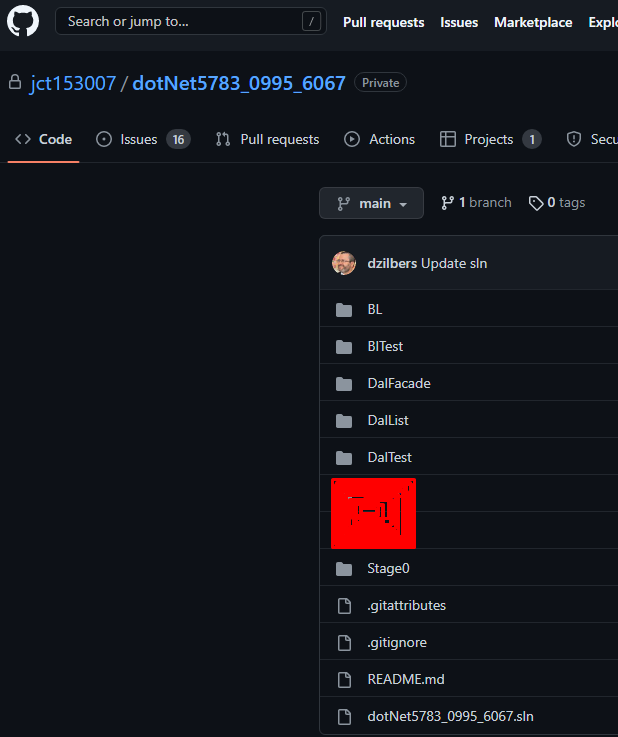
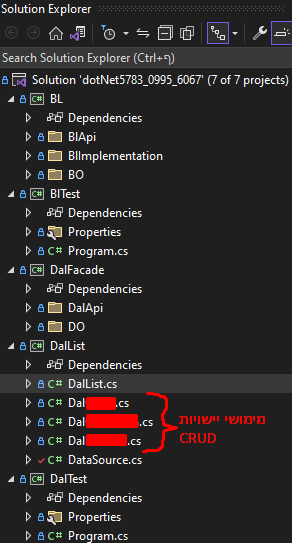
הקדמה

בשלב 1 התחלנו לבנות את שכבת הנתונים של הפרויקט וכתבנו תכנית ראשית לבדיקת השכבה.

בשלב 2:

1. **נעדכן את שכבת הנתונים** עם הכלים הנוספים שנלמדו בינתיים
2. נוסיף שכבה לוגית בסיסית
   1. נגדיר את ישויות שכבת הנתונים.
   2. נממש פעולות לעבודה מול שכבת הנתונים.
3. נבנה ממשק משתמש כקונסול טקסטואלי (CLI) לבדיקות

מבנה המאגר ב-github.com וה-Solution אחרי שלב 2:

שימו לב יש לגמור את החלק של שדרוג שכבת הנתונים עד ליום ראשון, למעט דברים קטנים (או גדולים ) שאינם ברורים. ביום ראשון נשתדל לענות על השאלות וגם להסביר את המשך שלב 2.

הגשת התרגיל כולו בס"ד שבועיים מהיום , בשבוע הראשון של כסלו.

המטלה –

## עדכון שכבת הנתונים

* 1. ב"חזית" שכבת הנתונים (פרויקט DalFacade) נוסיף תת-תיקיה בשם DalApi ובתוכה:
     1. נוסיף ממשק **גנרי** בשם ICrud עם פרמטר גנרי אחד (נניח שנקרא לו T)
     2. בממשק הגנרי החדש הנ"ל, נוסיף הצהרות למתודות CRUD הבאות:

Add,Delete,Update,Get

כאשר טיפוס הנתונים המתקבל או המוחזר הינו T (הפרמטר הגנרי).

**הערה**: המתודה המחזירה רשימת אובייקטים של ישות תחזיר סוג <IEnumerable<T

* + 1. נוסיף ממשקים עבור כל אחת מישויות הנתונים:
       1. הרשאת הממשקים הינה public
       2. הממשקים מרחיבים את הממשק ICrud, כאשר בארגומנט הגנרי נשתמש בטיפוס ישות הנתונים המתאימה:
       3. שם כל ממשק הינו **I***DoEntity* (עבור כל ישות יש להחליף את המילה *DoEntity* בשם הישות) - נשתמש בסוג ישות הנתונים המתאימה

**לדוגמה**:

**using DO;**

**namespace DalApi;**

**public interface I***DoEntity* **: ICrud<***DoEntity***> { ... }**

* + 1. אם בממשק של ישות מסוימת יש צורך במתודות נוספות, נוסיף אותן לממשק,אך בשום פנים אין להוסיף הצהרות של מתודות CRUD בסיסיות (הן כבר מוגדרות ב-ICrud),   
       **כלומר** - יתכן שרוב או כל הממשקים יהיו "ריקים" (**המשמעות**: הם מגדירים ממשק CRUD עבור ישות **מסוימת**)
    2. נוסיף ממשק בשם IDal ובו נגדיר תכונות (properties) עבור כל ישות:
       1. שם התכונה יהיה כשם הישות
       2. טיפוס התכונה יהיה כשם הממשק המתאים כנ"ל
       3. בתכונה תהיה רק מתודה get (זאת אומרת - כל התכונות הן לקריאה בלבד)
    3. נוסיף בתת-תיקיה DalApi מחלקה חדשה בשם **Exceptions** על מנת להגדיר חריגות מתאימות לפי הכללים שנלמדו בקורס
       1. בתוך הקובץ נמחק את המחלקה Exceptions כליל
       2. נוסיף מחלקות עבור חריגות מתאימות שנרצה לזרוק משכבת הנתונים
       3. החריגות אמורות להיות כלליות לפי סוג בעיה - אין לעשות חריגות מיוחדות לכל ישות בנפרד.  
          כלומר תהיינה חריגות כלליות לכל הישויות. מומלצות בשכבת הנתונים שתי החריגות הבאות (כי בעצם אלו הדברים היחידים שנבדקים בלוגיקת השכבה):
          1. חריגה עבור ישות שלא נמצאה או מזהה חסר (עבור עדכון, מחיקה או בקשה)
          2. חריגה של כפילות מזהה (עבור הוספה של אובייקט עם מזהה שכבר קיים)
       4. ניתן להוסיף עוד חריגות לפי הצורך בתנאי שהצורך ינומק.
  1. במודול **DataSource**:
     1. **נהפוך** את כל המערכים לאוספים של מערך דינמי - <>List
     2. נבטל את כל איברי (שדות\תכונות) מעקב האינדקסים של המערכים ("מצייני המקום הפנוי הראשון") שהשתמשתם בהם בשלב 1
     3. נעדכן את קוד האתחול בהתאם
  2. מימוש שכבת הנתונים **DalList**:
     1. עבור כל המחלקות למימוש CRUD שכתבתם בשלב 1:
        1. נוסיף יבוא של מרחב השמות DalApi :

**using DalApi;**

* + - 1. נשנה את הרשאת המחלקה ל-internal ונוסיף הצהרה על מימוש ממשק הישות המתאימה כפי שהוגדר ב-DalApi, למשל:

using DalApi;

using DO;

namespace Dal;

**internal** class Dal*DoEntity* **: I***DoEntity*

{

…

כאשר *DoEntity* היא ישות נתונים שהוגדרה ב-DO

* + - 1. נעדכן את חתימות המתודות הקיימות בהתאם לממשקים הנ"ל
      2. נעדכן את מימושי המתודות בהתאם לשינוים שביצענו בין מערכים לרשימות כנ"ל
      3. נחליף את זריקות החריגה הכללית שהוספתם בשלב 1 בחריגות מתוך המודול Exceptions שיצרתם בסעיף הקודם.
    1. נוסיף מחלקה עם הרשאה public בשם DalList שתממש את הממשק IDal.

sealed public class DalList : IDal

נממש בה את התכונות (properties) שהוגדרו בממשק IDal כך שיחזירו אובייקטים מטיפוס שמממש את הממשק המתאים לכל ישות נתונים.  
נ.ב. יש להשתמש באתחול מקוצר של התכונה בעזרת האופרטור "<=, למשל:

public I*DoEntity* *DoEntity* => new Dal*DoEntity*();

* 1. **בתוכנית ראשית של בדיקות שכבת הנתונים** שכתבתם בשלב 1:
     1. נשנה את סוגי כל המשתנים המקבלים רשימות אובייקטים ממערך ל-<>IEnumerable המתאים לפי כל ישות וישות.
     2. כדי לבדוק את השינויים ( ICrud) שנעשו על שכבת הנתונים, יש לשנות את הגישה אל המתודות בצורה הבאה:
  + יש ליצור ולשמור אובייקט של DalList במשתנה מקומי או בשדה סטטי של מחלקת Program (הטיפוס של משתנה\שדה יהיה IDal)
  + הפניה לישויות המתאימות תהיה דרך התכונות של אובייקט שהוגדר לעיל

## השכבה הלוגית - תיאור כללי

השכבה לוגית מנהלת את כל הלוגיקה של המערכת שאנחנו בונים. כל הכללים של המערכת ממומשים בה. השכבה תתקשר כלפי מטה עם שכבת הנתונים 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.

בבניית השכבה הלוגית נבצע את הצעדים הבאים:

* + נוסיף שני פרויקטים - אחד עבור שכבה לוגית ושני עבור תוכנית בדיקות של שכבה לוגית
  + נוסיף בפרויקט השכבה הלוגית תת-תיקיות עבור:
    1. ישויות לוגיות bo
    2. ממשק שכבה לוגית - ממשקים I…
    3. מימוש שכבה לוגית -
  + נגדיר ישויות לוגיות (כולל חריגות מותאמות של שכבה לוגית) – כמו שרשום בתיאור בעמוד 8 ו 9
  + נוסיף ממשקים של שכבה לוגית
  + נממש לוגיקה בסיסית של הפרויקט
  + נטפל בחריגות המתקבלות משכבת הנתונים (נתפוס אותן) ונזרוק חריגות חלופיות (של שכבה לוגית) איפה שנדרש

## לא לשכוח לכתוב תעוד כמו שצריך. זה מקובל וחשוב מאוד להתרגל

## הכנת התיקיות והפרויקטים לשלב הנוכחי - הנחיות מפורטות:

* 1. תוסיפו פרויקט חדש BL מסוג Class Library:
     1. עשו לחיצה כפולה על שם הפרויקט ב-Solution Explorer (ייפתח לעריכה קובץ ניהול הפרויקט עם סיומת csproj) - יש להוסיף לו את השורות הבאות (המודגשות) בסוף האלמנט PropertyGroup:

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

*<PropertyGroup>*

*<TargetFramework>net6.0</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 />**

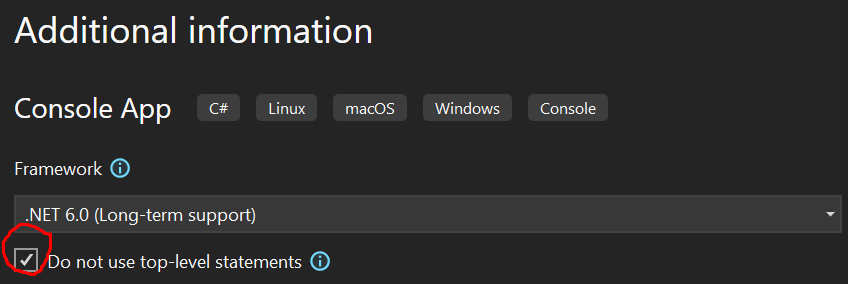
*</PropertyGroup>*

*</Project>*

* + 1. בתפריט ההקשר של הפרויקט (לחיצה על כפתור ימני כאשר הסמן מעל שם הפרויקט) בחרו מאפיינים (Properties) וודאו שהשדה Default namespace ריק (תחת כותרת Application>General, כתוצאה מהשורה המודגשת האחרונה לעיל), ובשדה Base output path (תחת כותרת Build>Output, כתוצאה מהשורה המודגשת הראשונה לעיל) מופיע:

**$(SolutionDir)\bin\**

* + 1. **נ.ב.** יתר ההוספות שעשינו לעיל לא משתקפות בממשק המאפיינים של פרויקט ולכן עשינו אותם ע"י עריכה ישירה בתוך הקובץ csproj
    2. בתפריט ההקשר של הפרויקט בחרו הוספת הפניות פרויקט (...Project Reference), ובו סמנו את הפרויקטים DalFacade ו-DalList (לחצו OK לאחר מכן)
    3. בתפריט ההקשר של הפרויקט בחרו …Build Dependencies>Project Dependencies וודאו שהסימון דומה לסימון ההפניות כנ"ל.

* 1. תוסיפו פרויקט BlTest
     1. תוסיפו ב-Solution פרויקט חדש מסוג Console Application
     2. בחלון הבא יש להקליד את שם הפרויקט BlTest
     3. בחלון הבא יש לסמן "Do not use top-level statements":
     4. עשו לחיצה כפולה על שם הפרויקט ב-Solution Explorer (ייפתח לעריכה קובץ ניהול הפרויקט עם סיומת csproj) - יש להוסיף לו את השורות הבאות (המודגשות) בסוף האלמנט PropertyGroup:

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

*<PropertyGroup>*

*<OutputType>Exe</OutputType>*

*<TargetFramework>net6.0</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>*

* + 1. בתפריט ההקשר של הפרויקט בחרו מאפיינים (Properties) ותוודאו שבשדה Base output path (תחת כותרת Build>Output, כתוצאה מהשורה המודגשת הראשונה לעיל) מופיע:

**$(SolutionDir)\bin\**

* + 1. **נ.ב.** יתר ההוספות שעשינו לעיל לא משתקפות בממשק המאפיינים של פרויקט ולכן עשינו אותם ע"י עריכה ישירה בתוך הקובץ csproj
    2. בתפריט ההקשר של הפרויקט בחרו הוספת הפניות פרויקט (...Project Reference), ובו סמנו את הפרויקט BL
    3. בתפריט ההקשר של הפרויקט בחרו …Build Dependencies>Project Dependencies וודאו שהסימון דומה לסימון ההפניות כנ"ל (מסומן פרויקט BL בלבד)

## בניית ארכיטקטורה של השכבה לוגית - הנחיות מפורטות:

נוסיף שלוש תת-תיקיות בפרויקט BL:

1. תת-תיקייה עבור ישויות לוגיות BO
2. תת-תיקייה עבור ממשקי שכבה לוגית BlApi
3. תת-תיקייה עבור מימוש של שכבה לוגית BlImplementation

## בניית ישויות לוגיות - הנחיות מפורטות:

1. **כל** ההגדרות של ישויות לוגיות ומודולים נוספים המפורטים בהמשך בפרק הזה נוספים **בתת-תיקיה BO** של השכבה הלוגית (בפרוייקט BL)
2. **עבור כל ישות לוגית** המתוארת בתיאור כללי של הפרויקט, נוסיף **מחלקה** (במודול נפרד)
   1. המודול אמור להיווצר אוטומטית עם namespace BO בעת הוספת המחלקה
   2. שם המחלקה יהיה זהה לשם הישות
   3. הרשאת המחלקה public
   4. נגדיר את כל הנתונים כתכונות (properties):
      * תכונה אוטומטית
      * הרשאה public
      * ללא אתחול בהצהרה
   5. אין להוסיף מתודות למחלקה מלבד דריסת ToString
   6. אסור להוסיף כל לוגיקה\קוד ביצוע למעט דריסה של ToString, זאת אומרת שאסור להוסיף: בנאים מכל סוג, הורס, מתודות כלשהן
   7. לדוגמה:

**namespace BO;**

**public class** *BoEntity*

**{**

**public ... { set; get; }**

**...**

**}**

כאשר *BoEntity* הינו שם של ישות לוגית, למשל "**Student**"

1. ניצור קובץ נוסף (מודול נפרד) ששמו Enums.cs ובו: לא משתמשים באינם שהוגדר בדל אלא מגדירים חדש בביאל
   1. עבור האנומרציות שמוגדרות עם ישויות הנתונים, נגדיר גם האנומרציות (enum) לוגיות בצורה זהה לאנומרציות ביושיות הנתונים  
      **נ.ב.** לא נוכל (**אסור**) להשתמש בטיפוסים של ישויות הנתונים עבור ישויות לוגיות
   2. נוסיף במודול גם אנומרציות נוספות כפי שנדרש בישויות הלוגית
2. נוסיף קובץ (מודול נפרד) עבור הגדרת חריגות של השכבה הלוגית
   1. בהתאם לסוגי התקלות בשכבה לוגית כפי שיתואר בהמשך
   2. החריגות מוגדרות ע"פ הכללים המתוארים במסמך החריגות בחומרי הקורס

## 

## בניית ממשקים של השכבה הלוגית

מטרת הממשקים של השכבה הלוגית היא לרכז ולהצהיר על הפעולות שהשכבה חושפת. בשכבה נוסיף ממשק ראשי שירכז את הגישה לממשקים הספציפיים, אשר יכללו את המתודות הקשורות ליישות לוגית ראשית מסוימת.

1. **את כל** הממשקים נוסיף **בתת-תיקיה BlApi**
2. עבור כל ההגדרות חובה להוסיף תיעוד "חיצוני" בעזרת הערה מפורמטת "///"
   1. לפני כל ממשק
   2. לפני כל תכונה
   3. לפני כל מתודה
3. נוסיף ממשקים עבור כל **ישות לוגית ראשית** (ראו בתיאור כללי של הפרויקט) בעמוד 11
   1. הרשאה public
   2. שם ממשק בפורמט I*BoEntity*, כאשר *BoEntity* הינו שם ישות לוגית ראשית (למשל "**IStudent**")
   3. נוסיף הצהרות של מתודות לפי הצורך כפי שמופיע בתיאור כללי של הפרויקט
   4. מתודות הוספה או עדכון כולל של ישות יקבלו פרמטר עם מופע של ישות מתאימה
   5. מתודות פעולות ייחודיות (למשל שינוי סטטוס של ישות) יקבלו פרמטר(ים) לפי צורך ע"פ ראותכם
   6. כל הפרמטרים יהיו **by-value** (אסור להגדיר פרמטרים עם ref\out\in)
   7. מתודות תחזירנה מופע של ישות מתאימה או אוסף מופעי של ישות מתאימה או ערך מתאים לפי הצורך  
      **נ.ב.** במתודות שאמורות להחזיר רשימה כלשהי, סוג הערך המוחזר יהיה אוסף כללי גנרי: <IEnumerable<*BoEntity* בהתאם (למשל **<IEnumerable<Student**)
   8. לדוגמה:

**using BO;**

**namespace BlApi;**

**public interface I***BoEntity*

**{**

**public ... Method1(...);**

**...**

**}**

1. נוסיף ממשק ראשי בשם IBl שירכז את כל הממשקים של השכבה
   1. הרשאה public
   2. נגדיר תכונות עבור כל ישות לוגית ראשית (ראו בתיאור כללי של הפרויקט)
      1. הרשאה public
      2. טיפוס - ממשק ישות לוגית ראשית מתאימה כנ"ל
      3. שם - כשם ישות לוגית ראשית *BoEntity* (למשל "**Student**")
      4. לקריאה בלבד (רק עם get)
      5. ללא אתחול בהצהרה

**לדוגמה**:

**public interface IBl**

**{**

**public I***BoEntity**BoEntity* **{ get; }**

**...**

**}**

## 

## בניית הלוגיקה (מימוש) של שכבה לוגית - הנחיות מפורטות:

**הסבר כללי**:

השכבה הלוגית לא שומרת אצלה שום נתונים, כלומר המופעים של ישויות הנתונים והישויות לוגיות בשכבה זו לא נשמרים בצורה ישירה, אלא מחושבים בעת הצורך בתוך המתודות.

המתודות של השכבה הלוגית מופעלות בעקבות בקשות של (זימון מ-) שכבת התצוגה. אם מדובר בבקשת נתונים לתצוגה בממשק משתמש - המתודות יצרו מופעים לוגיים על בסיס הנתונים שיוּבאו משכבת הנתונים ע"י בקשות לשכבת הנתונים. ואם מדובר בבקשות הוספה או עדכון הנתונים - שכבת התצוגה תעביר מופע ישות לוגית מתאימה בארגומנטים של הבקשה (המתודה המזומנת מתוך שכבת התצוגה).

**לדוגמא**:

נניח שאנחנו בונים אפליקציה של מזת"ל המנהלת הרשמת סטודנטים לקורסים. בשכבת הנתונים אנו שומרים נתוני סטודנטים, קורסים, ו"קורסים עבור סטודנט". אזי בשכבה הלוגית יצרנו מחלקה שיש בה אפשרות לשמור פרטי סטודנט ורשימת כל הקורסים שלו. כאן במימוש השכבה:

1. כאשר נתבקש לכך על ידי ממשק המשתמש להציג את הסטודנט יחד עם רשימת הקורסים שלו אזי נייצר מופע של ישות כזו בשכבה הלוגית ונחזיר אותה לשכבת התצוגה.
2. כאשר נתבקש להוסיף סטודנט חדש (יחד עם הקורסים שנרצה לרשום אותו מראש), שכבת התצוגה תיצור מופע של ישות הלוגית הזו של סטודנט ותעביר אותה לשכבה לוגית בארגומנט של הבקשה, והמתודה של השכבה תיצור ותבצע הוספות של מופעי ישויות הנתונים המתאימות ע"י בקשות לשכבת הנתונים - הוספת מופע ישות נתונים של סטודנט והוספה של מופעים מתאימים של ישות "קורס עבור סטודנט".

**הנחיות מימוש מפורטות**:

1. כל המחלקות של מימוש השכבה הלוגית תהיינה **בתת-תיקייה BlImplementation**
2. עבור כל ישות לוגית ראשית (שעבורה קיים ב-BlApi ממשק *IBoEntity*) נוסיף מחלקה בשם Bl*BoEntity*
   1. נוסיף יבוא של מרחב השמות BlApi :

**using BlApi;**

* 1. נשנה את הרשאת המחלקה ל-internal ונוסיף הצהרה על מימוש ממשק הישות הלוגית הראשית המתאימה כפי שהוגדר ב-BlApi, למשל:

using BlApi;

namespace BlImplementation;

**internal** class Bl*BoEntity* **: I***BoEntity*

{

…

כאשר *BoEntity* היא ישות לוגית ראשית מתאימה שהוגדרה ב-BO

* 1. נוסיף שדה פרטי Dal מטיפוס IDal שיאותחל לאובייקט מטיפוס DalList
  2. אסור להגדיר שדות או תכונות נוספות
  3. נממש את כל המתודות שהוגדרו בממשק המתאים:
     1. על מנת למנוע טעויות, אסור לעשות ייבוא (using) ל-BO ול-DO (אסור **לשניהם**)
     2. עבודה עם ישויות נתונים תיעשה בצורה מפורשת: DO.*DoEntity* (למשל **DO.Student**)
     3. עבודה עם ישויות לוגיות תיעשה באופן מלא: BO.*BoEntity* (למשל **BO.Student**)
     4. יצירת המופעים של ישויות מכל סוג תתבצענה באתחול מהיר
     5. אסור לבצע כל קלט או פלט
     6. המתודה תבצע את כל הבדיקות הלוגיות של תקינות הנתונים, והתאמה בין הנתונים, והתאמה של הפעולות שהתבקשו לנתונים הקיימים בשכבת הנתונים
     7. המתודה תבצע את כל החישובים הנדרשים על מנת לבצע את הלוגיקה הנדרשת
     8. אם המתודה מבצעת בקשת נתונים, המתודה:
        + תבקש את כל הנתונים הנדרשים משכבת הנתונים ע"י זימון מתודת IDal מתאימה
        + תיצור מופע ישות לוגית מתאים ותאכלס אותו בנתונים (עדיף ע"י אתחול מהיר) ע"פ הנתונים שהתקבלו משכבת הנתונים או חושבו בקוד של המתודה
        + תחזיר את מופע הישות הלוגית שנוצר
     9. אם המתודה מבצעת הוספה או עדכון, המתודה:
        + תשתמש בנתונים שהתקבלו בפרמטרים של הבקשה (בדרך כלל מופע ישות לוגית ראשית מתאימה)
        + תבקש את מופעי ישויות הנתונים **הנדרשים ללוגיקה של המתודה** משכבת הנתונים ע"י זימון מתודת IDal מתאימה
        + תבקש את מופעי ישויות הנתונים **הנדרשים לעדכון** משכבת הנתונים ע"י זימון מתודת IDal מתאימה, תעדכן אותם בהתאם ותשלח בקשת עדכון לשכבת הנתונים
        + תיצור מופעי ישויות הנתונים **הנדרשים להוספה** עם אתחול מהיר של הנתונים (על בסיס הנתונים שהתקבלו בפרמטרים, משכבת הנתונים או חושבו ע"י המתודה), ותשלח בקשות הוספה לשכבת הנתונים
        + תיצור מופע ישות לוגית מתאים ותאכלס אותו בנתונים (עדיף ע"י אתחול מהיר) ע"פ הנתונים שהתקבלו משכבת הנתונים או חושבו בקוד של המתודה
        + תחזיר את מופע הישות הלוגית שנוצר
     10. לאחר השלמת ביצוע המתודה כל הנתונים בשכבת הנתונים חייבים להיות שלמים ותואמים, ובאחריות המתודה לדאוג לכך ולוודא את זה
     11. **חריגות**:
         + חובה לתפוס את החריגות של שכבת הנתונים ולטפל בהן בהתאם לדרישות הלוגיקה
         + אם נדרשת זריקת חריגה לכיוון שכבת התצוגה - תיזרק חריגה חילופית מתאימה שתכלול את חריגת שכבת הנתונים כחריגה פנימית  
           **נ.ב.** יש לוודא שבחריגה מתאימה שהוגדרה ב-BO נוסף בנאי שמקבל פרמטר עבור חריגה פנימית (inner exception)
         + עבור מצבים של תקלה בתוך שכבה לוגית תיזרקנה חריגות מתאימות של שכבה לוגית שהוגדרו ב-BO
         + החריגות לא יהיו ספציפיות לסוג של יישות לוגית אלא לסוג של תקלה (למשל - ערך נתון לא תקין, סדר תאריכים לא נכון, וכדומה)
  4. מותר להוסיף מתודות עזר עם הרשאה private לפי הצורך

1. מותר להוסיף מחלקות עזר סטטיות או רגילות עם הרשאה internal לפי הצורך
2. נוסיף **מחלקה Bl**
   1. הרשאה public
   2. מממשת ממשק IBl.

sealed public class Bl : IBl

* 1. מממשת בתוכה את התכונות (properties) שהוגדרו בממשק IDal כך שיחזירו אובייקטים מטיפוס שמממש את הממשק המתאים לכל ישות לוגית ראשית.  
     **נ.ב.** יש להשתמש באתחול מקוצר של התכונה בעזרת האופרטור "<=, למשל:

public I*BoEntity* *BoEntity* => new Bl*BoEntity*();

## תוכנית ראשית לבדיקת השכבה הלוגית

בפרויקט BlTest שיצרנו, נבנה תכנית ראשית (קונסול) חדשה לבדיקות ידניות של הפונקציונליות של השכבה הלוגית. לתוכנית **אסור לכלול שום לוגיקה** **למעט קלט, בקשות משכבה לוגית ופלט**, למעט בדיקות תקינות פורמט קלט (כגון כשצריך מספר אבל הקלידו תווים שאין ספרות). התוכנית החדשה תהיה דומה מבחינת המבנה לזו שבנינו לבדיקות שכבת הנתונים, אך תטפל בדיקות הבקשות מהשכבה הלוגית.

המתודה Main בתוכנית תבצע לולאה שתדפיס בקונסול את התפריט הראשית למשתמש עם האפשרויות ע"פ ישויות לוגיות ראשיות ובנוסף אפשרות יציאה מהתוכנית. עבור כל ישות לוגית ראשית ניצור בתוכנית הראשית מתודה נפרדת לשל ישות לוגית ראשית שתציג תת-מפריט. בכל תת-תפריט יהיו כל האפשרויות לביצוע הפעולות הוגדרו בממשק ישות לוגית ראשית מתאימה ובנוסף אפשרות יציאה מתת-תפריט לתפריט הראשית.

התוכנית תיצור קודם כל מופע של מחלקה BL שתשמור אותו במשתנה או בשדה מטיפוס IBl.

**חובה** לאפשר בתכנית להפעיל את **כל הפעולות** של ממשקי השכבה הלוגית.

עבור בדיקת כל פעולה בממשק ישות לוגית ראשית (זאת אומרת, עבור כל אפשרות בתת-תפריט של הישות):

* נקלוט את כל הנתונים הרלוונטיים הדרושים לזימון הפעולה
* נכין את הנתונים ואת המופעים הנדרשים לפעולה
* נזמן את הפעולה (דרך תכונה מתאימה של המשתנה המכיל הפניה לאובייקט של BL כנ"ל)
* נדפיס את תוצאות הפעולה, בהתאם לסוג ערך מוחזר
* בהדפסות:
  + אובייקטים של ישויות נתונים יודפסו בעזרת מתודת ToString ישירות  
    **נ.ב.** **אסור לזמן את המתודה ToString בצורה מפורשת!**
  + אוסף יודפס בעזרת לולאת foreach ובתוכה הדפסת אובייקטים כנ"ל נתפוס את החריגות הצפויות (עבור הפעולה המסוימת) של שכבת BL
* נתפוס את החריגות הצפויות (עבור הפעולה המסוימת) של שכבת BL  
  **נ.ב.** לא נתפוס שום חריגות אחרות - על מנת לקבל אפשרות לדבג במקרה ומתרחשות
* עבור כל חריגה שנתפסה נדפיס את המידע על החריגה: שם החריגה, הודעה שבחריגה, מידע על חריגה פנימית (אם יש)
* **לבונוס**: לצורך המרות של מספרים ותאריכים מהקלט - להשתמש בפונקציה TryParse

**בהצלחה רבה!**