מיני פרויקט במערכות חלונות תשפ"ב

מערכת לניהול שירות משלוחים

תוכן עניינים

מבוא

הפרויקט השנה עוסק בניהול מערכת מידע של שירות משלוחים המתבצעים באמצעות רחפנים, המשלוחים מתבצעים בין הלקוחות השונים של השירות. למשל משלוחים מחנויות ועסקים שונים לצרכנים השונים.

החברה מחזיקה תחנות-בסיס עבור תחזוקת וטעינת הרחפנים וכן מידע על לקוחות וחבילות שנשלחו באמצעות השירות.

החברה המפעילה את שירות המשלוחים רוצה לעקוב אחר ביצוע המשלוחים וכן לנהל את המעקב על תחזוקת הרחפנים. בהמשך הפרויקט יהיה מעקב רחפנים וחבילות.

הערה : **שימו לב שביצוע הפרויקט לפי דרישות מינימליות (<u>ללא בונוסים</u>) בלבד לא תאפשר לקבל ציון מעל 80.**

למערכת ישנם 2 ממשקים למשתמשים שונים:

- ממשק החברה המפעילה - ניהול ועדכון מידע על:
  - תחנות בסיס (לטעינת סוללות הרחפנים)
  - רחפנים
  - לקוחות
  - חבילות
- ממשק הלקוח: **(לבונוס)**
  - הרשמה
  - עדכון פרופיל (נתוני הלקוח)
  - הזנת חבילה חדשה למשלוח
  - מעקב חבילות שנשלחו ע"י הלקוח
  - מעקב חבילות שנשלחו אל הלקוח

<u>המטרה</u>: תרגול ויישום של

- עקרונות שפת #C
- תבניות עיצוב תוכנה:
  - ארכיטקטוניות (מודל השכבות, Contract Design)
  - יצרניות (Singleton, Simple Factory)
  - התנהגותיות (Observer, Iterator)
- יצירת ממשק משתמש באמצעות תשתית הפתוח המודרנית WPF
- עבודת צוות
- חשיבה תכנותית ולמידה עצמית

<u>הערה</u>: ההכוונה במסמך זה היא כללית בלבד, שכן חלק מן הדרישה היא להפעיל חשיבה עצמאית יצירתית. כמו כן, חלק מהקריטריונים בבחינה הסופית של הפרויקט יתבססו על נושאים תיכנותיים שתתבקשו להתמודד איתם במהלך פיתוח הפרויקט. חשוב להדגיש שציון מתחת ל-85 בקורס זה בדומה לכל קורס מעשי איננו משמעותי בעיני המעסיקים בתעשייה.

שימו לב שישנן אפשרויות שונות לצבירת בונוסים בפרויקט. בונוסים הינם תוספות ושיפורים שנעשה מעבר לידע והכלים הנלמדים במסגרת הקורס או הצעת תוספות שאופיינו מראש כתוספות "לבונוס".

<u>חובה לקרוא את כל פרק של מבנה הפרויקט ודרישות כלליות לפני תחילת העבודה על הפרויקט!</u>

<u>דגשים</u>:

- העבודה תתבצע אך ורק בזוגות (במקרים מיוחדים, ניתן לקבל אישור מהמרצה לעבודה יחידנית, אך ודאי לא בשלישיות). הזוגות יהיו קבועים לכל אורך הסמסטר (לא ניתן לעבור במהלך הסמסטר מזוג לזוג). לא ניתן להיות שותף עם חבר שרשום בקבוצה שונה. בכל אחד מן המקרים האלה הפרויקט ייפסל לשני השותפים.
- כל אחד מהשותפים לזוג חייב להיות שותף מלא בכל אחד מהתרגילים (שבוצעו) והשלבים של הפרויקט. (במידה ויתברר כי שותף אחד עשה שלב מסוים ואלו שותף אחר עשה את החלק האחר, הציון יהיה פרופורציונלי למספר השלבים שנעשו ע"י כל אחד מן השותפים, כמו כן - ציון כל שותף יוכפל במקדם הערכה של מידת ההשתתפות השותף בפרויקט שייקבע בזמן ההגנה על הפרויקט כציון הגנה).
- אמנם הרושם ואיכות חווית המשתמש של הממשק הגרפי בפרויקט אינו מהווה חלק מהשיקולים למתן ציון

הפרויקט , אך קחו בחשבון שבאופן טבעי ברמת תת-מודע הרושם שנעשה על הבוחן בחוויה הזו עשוי להשפיע על שיקול דעתו לטובה או לרעה. זאת אומרת - אל תשקיעו יותר מדי זמן ביצירת חווית משתמש איכותית במיוחד, אבל תשתדלו לא להביא משהו מכוער ומסורבל.

- נשקלת אפשרות שלאחר סוף הסמסטר, ייבחר פרויקט מצטיין מכל קבוצת הקורס ומבין אלו, ייבחרו הפרויקטים המצטיינים, והם יזכו את יוצריהם בפרס שייקבע ע"י ראש המחלקה. המדדים העיקרים למצוינות הם:
  - תכנון ותכנות נכונים
  - מקוריות ויצירתיות
  - למידה עצמית
  - עבור הצטיינות - גם לרמת חווית משתמש (UX) יש משמעות בהערכה

שימו לב שהקריטריונים לבחינת מצוינות הפרויקט הם אחידים לכל הקמפוסים ולכל הסטודנטים ללא קשר למה לימד(ה) או לא לימד(ה) אותם המרצה שלהם\ן.

תהליך ההגשה:

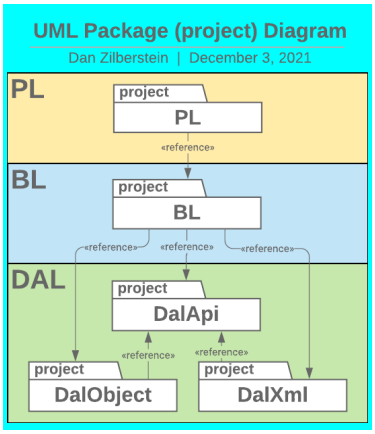
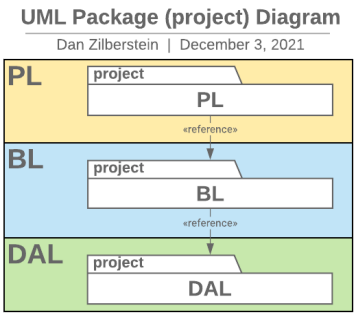
- הפרויקט מחולק ל**ארבעה שלבים** המבוססים זה על קודמו, השלב הראשון מתבסס על התרגילים 1 - 3.
- בסיום כל אחד מן השלבים חובה להגיש את הפרויקט בתיבת ההגשה המתאימה, לפי הנחיות הגשה.
- בדיקת הפרויקט תתבצע לאחר סיום הפרויקט כולו, אך תכולת ההגשות של השלבים יוכלו להעיד על רמת ומידת ההשקעה של הסטודנטים בפרויקט ויכולות להיבדק ע"י בוחן הפרויקט.
- בכל שלב מוגש, המערכת חייבת ליצור הרצה תקינה וביצוע פעולות בממשק משתמש לפי הנדרש בכל שלב
- עם הגשת השלב האחרון והמסכם תתבצע בדיקה יסודית של העבודה שתכלול גם הגנה (מעין בחינה בע"פ ע"י מרצה או בוחן חיצוני) של שני השותפים, כ"א בנפרד. המרצה\הבוחן שלכם גם יקבל את ההגנה וגם יבדוק את איכות הפרויקט לאחר מכן, ויתן ציון סופי של פרויקט.
- הציון הסופי על הפרויקט יתייחס לכל רכיביו, וכן - לאיכות ההצגה ולעמידה במועדי ההגשה.
- ההגנה על הפרויקט תתבצע בשיעור האחרון (בנוכחות או בזום -- השיטה תיקבע בהמשך) או במועד אחר שייקבע ע"י המרצה של הקבוצה ע"פ החלטתו ובלי קשר לקבוצות ומרצים אחרים, ובכל מקרה ההגנה תיקבע לכל המאוחר עד סוף מועדי א' של סמסטר זה.
- **שימו לב:** מי שהיה שותף מלא בפרויקט שלו לכל חלקיו - רוכש הבנה מעשית ומעמיקה בחומרים הנלמדים ובעצם מגיע למבחן הרבה יותר מוכן מאשר סטודנט שלקח קטעי קוד מחבריו, קיבל הנחיה מחבריו - תוך חוסר הבנה במה שהוא עושה.
- הגשת כל שלבי הפרויקט תתבצע בעזרת טאגים (tags) בדומה להגשות התרגילים בקורס, כאשר שם הטאג הינו: **PR01**, **PR02**, **PR03**, **PR04** בהתאם למספר השלב המוגש.
- בהגשה הסופית של הפרויקט יש לצרף מסמך המפרט תוספות (לבונוס) בפורמט DOCX או PDF, וכן קודי גישה (סיסמאות) במידה וישנם בממשק המשתמש.
- מרכיבי ציון הפרויקט:
  - עמידה בדרישות בסיס (80% מהציון)
  - בונוסים ע"פ בחירת הסטודנטים (20% מהציון)
- בדרישות הפרויקט כלולים "בונוסים" שהם הצעותינו לאפשרויות ההרחבה שאנחנו מציעים. אין שום צורך לבצע את כל הבונוסים על מנת להגיע לציון מירבי בקורס. הערכת רמת וכמות הבונוסים ומתן ציון עליהם בסמכות של מרצה קבוצתכם והבוחן.

נ.ב.: יש לוודא שתכני הפרויקט (כולל תמונות וכדומה) יתאימו לרוח ההלכה – גם לפי הדעות המחמירות ביותר.

מבנה הפרויקט ודרישות כלליות

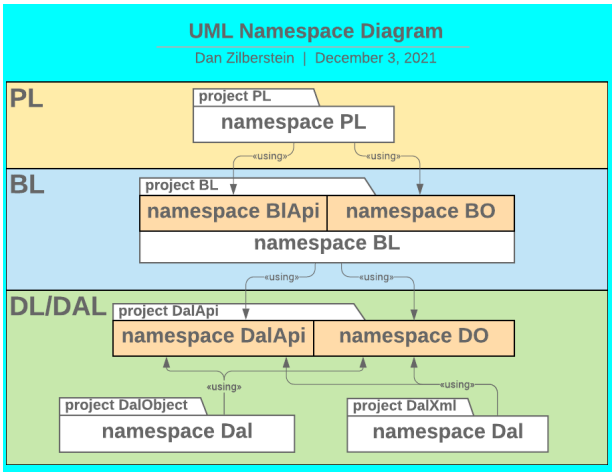
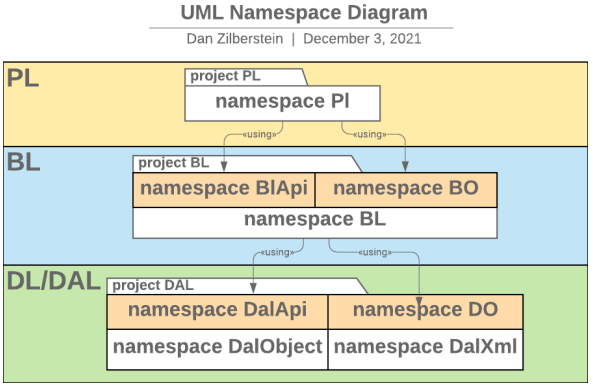
הפרויקט ייבנה באותו המאגר git ו-github ותחת אותו Solution שבו הסטודנטים עובדים מתחילת הסמסטר.

פרויקטים ב-VS ייפתחו ע"פ התבניות הבאות (פרויקטים של קונסולים של תרגיל 1 ותרגיל 2 אינם מיוצגים - יש לקבוע את מקומם ותלויותם\הפניותם בהתאם).

תבנית של מודל השכבות כדלקמן (להלן - מבנה 2) **לבונוס**	תבנית של מודל השכבות כדלקמן (להלן - מבנה 1)
	

הפרויקט שיוגדר להרצה ב-Solution - הינו פרויקט PL.

כל חלקי השכבות יוגדרו במרחב שמות (namespace) מתאים כדלקמן. בכל מבנה שתבחרו - יש לבצע את השינויים בהתאם לארכיטקטורה המוצגת (לשנות מרחבי שמות ולהעביר את המודולים בהתאם, אם נדרש). תרשימי מרחבי השמות:

תרשים מרחבי השמות במבנה 2 **לבונוס**	תרשים מרחבי השמות במבנה 1
	

שימו לב: מרחבי שמות של ישויות מאבדות את ההיררכיה של היותן מתחת ל-IDAL\IBL כפי שהיה בתרגילים, מרחבי השמות הכוללות את ממשק, פקטורי וכו' משתנים מ-IDAL\IBL ל-DalApi\BIApi בהתאם (שמות הממשקים עצמם לא משתנים). שינויי שמות כדאי לבצע דרך refactoring על מנת לחסוך הרבה עבודה.

שימו לב: הנחיות ודוגמאות שב-OSF אינם מעודכנות ואינם מדויקות עבור בניית מודל השכבות (Multi-Tier Architecture), שימוש בתבניות עיצוב (Design Patterns) ובחריגות. פרויקט שהמבנה שלו יהיה ע"פ הדוגמאות מ-OSF יפסיד הרבה מאוד נקודות מהציון הסופי.

דרישות כתיבת קוד כלליות:

- **חובה להקפיד על העקרונות SOLID תוך פיתוח הפרויקט (למעט ISP) - בכל השכבות**
- **חובה להקפיד על עקרונות DRY (אל תחזור על עצמך - Don't Repeat Yourself) ו-KISS (תשאיר את זה פשוט, טמבל - Keep It Simple, Stupid) בכתיבת הפרויקט**
- כל טיפוס יכלול תיעוד ע"י ///
- כל הגדרת מתודה (לא כולל מימושים של מתודות ממשק) תכלול תיעוד ע"י ///
- כל תכונה (property) תכלול תיעוד ע"י ///
- כל התיעודים הנ"ל חייבים לכלול כן הסבר קצר ב-Summary וכן הסברים קצרים על הפרמטרים, ערך מוחזר וכו' - כל דבר לפי התבנית שלו
- בכל פעולה לא טריוויאלית (ורק לא טריוויאלית) - תכלול תיעוד פנימי ע"י הערות בהתאם
- יש להשתמש בפרגמה #region על מנת להקל ניווט בתוך הקוד
- חובה לעצב את הקוד בהתאם לדרישות:
  - הזחות
  - ריווח שורות וריווח בתוך שורה
  - מוסכמות שמות - CamelCase\camelCase לפי הדרישות של מיקרוסופט עבור C#
  - שמות משמעותיים לפי הגישה ש הקוד צריך לתעד את עצמו

רענון דרישות בעניין שימוש וטיפול בחריגות:

- כל שכבה (DAL ו-BL) תזרוק חריגה במקרה של תקלה או חוסר התאמה לוגית כלשהיא
- אסור בהחלט לזרוק חריגה מטיפוס Exception!
- כל שכבה תגדיר טיפוסים עבור החריגות שהיא זורקת במסגרת מרחב שמות מתאים של אותה השכבה (ב-DO או ב-BO) כך שהן תוכרנה בשכבה שמעליה
  - החריגות תכלולנה שדות נוספים ובנאים לפי הצורך
- כל שכבה תתפוס את החריגות מהשכבה שמתחתיה ותטפל בהן בהתאם - ייתכן שע"י זריקת חריגה משלה (במקרה של BL)
- שם טיפוס החריגה הוא המתאר את התקלה באופן כללי - שדות והטקסט שניתן בחריגה יכלול פרטים נוספים על התקלה אך אינם המזהה של סוג התקלה - אין ליצור חריגות בנפרד לישויות שונות אלא לפי סוג של תקלה!
- בשכבת הממשק (PL) תטופלנה כל החריגות בהתאם לממשק המשתמש ולא תקרוס התוכנית בשום מצב בגלל חריגה שלא טופלה!
- בשכבת PL טיפול בחריגות יהיה בהתאם לפונקציונליות הנדרשת - אסור להקפיץ חלון הודעה שרק ידפיס את תוכן החריגה או את הודעת החריגה - בחלון הודעה הקופץ ה-PL צריך לתאר את הבעיה בצורה המובנת למשתמש (לא למתכנת) על בסיס תוכן (נתונים) באובייקט החריגה

דרישות כלליות של שכבת תצוגה PL

- אסור לבצע כל לוגיקה של הפרויקט למעט בדיקת תקינות קלט בסיסית (פורמט, תווים חוקיים וכו')
- בכל פעולה המבוצעת מ-GUI אסור לשלוח יותר מבקשה אחת ל-BL (אסור להפעיל יותר מפעולה אחת של BL)
- חובה להשתמש בכל מה שנלמד על WPF בקורס לפי הסילבוס:
  - פקדים בסיסיים
  - פקדים מרובי נתונים
  - תסדירים
  - שימוש במשאבים (Resources)
  - קישור בין תכונות תצוגה שונות וקישור של תכונות תצוגה לנתונים בקוד האחורי - בכמה דרכים עם עדכון חד כיווני ודו-כיווני
  - שימוש ב-DataContext בצורה היררכית - עבור קישור לנתונים
  - כדאי להשתמש ~~שימוש~~ ב-ObservableCollection
- **אפשריות בונוס:**

- שימוש פונקציונליות נוספת של WPF שאיננה חובה בסילבוס של הקורס, למשל:
  - טריגרים (trigger - שלושה סוגים - טריגר תכונה, טריגר נתונים, וטריגר אירוע)
  - תבניות עיצוב (DataTemplate) בכללי ולתבניות אלמנט בפקד מרובה נתונים בפרט
  - המרות בקישור נתונים (IConverter
  - סגנון (Style), ערכות נושא (Theme)
  - תבניות פקד (ControlTemplate)
  - טרנספורמציות של פקדים (Transform)
  - תכונות מצורפות (attached properties)
  - גרפיקה (drawing, shapes)
  - פקודות והתנהגויות (commands and behaviors)
  - ארכיטקטורה MVVM
  - ועוד ועוד ועוד - לפי הזמן והרצון של הסטודנטים
- שימוש בישויות תצוגה PO וב-data binding מלא דרך ישויות אלה:
  - יצירת ישויות PO מתאימות - אסור להוסיף כל קוד לישויות של BO
  - בישויות PO - עבור אוספי אובייקטים ניתן (וכדאי) להשתמש ב-ObservableCollection
- כל בקשה ל-BL תתבצע בפועל רקע (Background Worker), כאשר מתקבלת תוצאה מבקשה כזו - יעודכן מידע באובייקטים של BO\PO השמורים באובייקט המחלקה של החלון המתאים
- לא יהיה שיתוף מידע בין החלונות - העברת המידע תתבצע בעזרת העברת ארגומנטים עבור הפרמטרים של בנאי החלון, כולל אובייקט של BL, או העברת פונקציה לבנאי בפרמטר מסוג דלגט

דרישות כלליות של שכבה לוגית BL

- **רק הטיפוסים המוגדרים במרחבי השמות BlApi (מחלקות IBL ו-BIFactory) ו-BO (ישויות לוגיות) יהיו עם הרשאת public**
- כל הבקשות שעשויות להגיע משכבת PL לשכבת BL חייבות להיות מוגדרות כפעולות בממשק השכבה IBL.
- **כל הטיפוסים המוגדרים במרחב השמות BL יהיו עם הרשאת ברירת מחדל (ז"א ללא הרשאה כלל שזה אומר - internal)**
- **כל הישויות של BO**:
  - יהיו PDS חלקי (PDS - Passive Data Structure):
    - **לא יכללו כל קוד** למעט העמסת פעולת ToString לצורכי debug
    - יכולים לכלול אוספים גנריים (IEnumerable<>)
    - יכולים לכלול תכונות מטיפוס של ישות BO אחרות
    - יכולים לרשת מישות BO אחרת (יש להיזהר מאוד עם שימוש בירושה ב-BO!)
  - חייבות להכיל את כל המידע הנדרש עבור בקשת PL אפשרית (פונקציה של IBL), עבור פונקציונליות המיוצגת בחלון מתאים של PL
- **מחלקת BIFactory** הינה חלק מתבנית עיצוב Simple Factory
  - תהיה **מחלקה סטטית**
  - תכיל פעולה GetBl
    - ללא פרמטרים
    - טיפוס ערך מוחזר IBL
    - מחזירה אובייקט מטיפוס שמימש את הממשק IBL בהתאם (אובייקט של BL)
- הפעולות המוגדרות ב**ממשק IBL**:
  - יכולות לקבל פרמטרים מטיפוסים:
    - של דוט-נט
    - של BO
    - של דלגטים בהתאם
- על טיפוס הדלגט להיות מוגדר ב-BO או להיות אחד הטיפוסים של דלגט

המוגדרים בדוט-נט
- יכולות להחזיר ערך מטיפוס:
  - של דוט-נט
  - של BO
  - אוסף גנרי של איברים מטיפוסים כנ"ל (IEnumerable<T>) - **אסור להגדיר סוג ערך מוחזר אוסף מסוג ספציפי (למשל List)**
- **מחלקת BL** (המממשת את הממשק IBL) ומחלקות עזר:
  - תהיה ללא הרשאה (=internal)
  - תשתמש בתבנית עיצוב Singleton
  - כל השדות (אם ישנם) יהיו בהרשאה private
  - אם יש צורך בשיתוף מידע עם מחלקות נוספות בשכבה - יש להגדיר תכונות (properties) בהתאם, עם הרשאה internal
  - מתודות עזר תהינה בהרשאות internal או private בהתאם לשימוש בהן
  - כל מחלקות עזר (אם ישנן) תהיינה ללא הרשאה (=internal)
    - דרישות לגבי שדות, תכונות ומתודות כנ"ל עבור מחלקה BL
  - **אסור** להפוך או ליצור אוספים כלליים (IEnumerable) לאובייקטים של אוספים ממשיים (מערך או List) למעט רשימת מצב הרחפנים הנשמרת בשכבה ולמעט מקרים של "לית ברירה"
  - חובה להשתמש בביטויי למבדה בכל מקום מתאים
  - בגרסה הסופית אין להשתמש בלולאת foreach במקום שניתן להשתמש ב-LINQ
  - חובה להשתמש ב-LINQtoObject בכל מקום מתאים - ניתן להשתמש בפורמט שאילתות ו-\או בפורמט זימון מתודות הרחבה (רצוי בשרשור)
  - ב-LINQtoObject חובה להשתמש לפחות פעם אחת ב:
    - let
    - select new
    - קיבוץ (grouping)
    - מיון

במימוש של BL ניתן וכדאי לבנות מחלקות עזר מתאימות, כמו כן ליצור ישויות פנימיות (internal) שעוטפות את ישויות BO לצורך תיאור והצמדה של פעולה\ות שמתבצעת\ות ע"י ישות מסוימת למחלקה שעוטפת את הישות.

במקרה של בקשות הוספת מופע חדש של ישות או עדכון של מופע קיים של ישות - שכבת BL היא אחראית בלעדית על בדיקת תקינות נתונים מבחינת ערך ספציפי ומבחינת תיאום הערכים עם ערכי מופעים אחרים של הישות ומידע במופעים של ישויות אחרות.

בכל מקרה של תקלה או טעות בטיפול בבקשה מ-PL, תיזרק ממימוש השכבה חריגה מתאימה, שבמקרה הצורך תוגדר ב-BO. אם התקלה קרתה בעקבות תקלה (חריגה) בשכבת DAL - מימוש שכבת BL יזרוק חריגה חדשה כאשר החריגה המקורית תיכלל כחריגה פנימית (InnerException) בחריגה החדשה - ע"י הפעלת בנאי מתאים של החריגה הנזרקת.

***שימו לב** שציון הפרויקט תלוי במידה רבה מתכולת ופונקציונליות השכבה הזו! השכבה הזו **אינה** בשום פנים ואופן מתווך פשוט להפעלת פעולות של שכבת DAL! וזוהי בין היתר המשמעות של הפרדת ישויות DO ו-BO - הישויות ב-BO אינן זהות בתוכנם לישויות DO - למעט מקרים כאשר הדמיון ביניהם מתבקש בגלל הדרישות שבאות מכיוון שכבת ה-PL.*

דרישות כלליות של שכבת נתונים DAL

- רק הטיפוסים המוגדרים במרחבי השמות DalApi (מחלקות IDal ו-DalFactory) ו-DO (ישויות נתונים) יהיו עם הרשאת public
- כל הבקשות שעשויות להגיע משכבת BL לשכבת DAL חייבות להיות מוגדרות כפעולות בממשק השכבה IDal
- כל יתר הטיפוסים המוגדרים במרחב השמות Dal יהיו עם הרשאת ברירת מחדל (ז"א ללא הרשאה כלל שזה

אומר - internal)

- **ישויות ה-DO**:
  - יהיו PDS (לא יכללו כל קוד למעט העמסת פעולת ToString לצורכי debug) - ע"י struct בלבד
  - יכללו תכונות מטיפוסים שהם ValueType, string או enum **בלבד**
    - זה כולל גם (לדוגמה): DateTime, TimeSpan, כמו כן סוגי enum שאולי תגדירו גם ב-DO
  - לא יכללו תכונות מטיפוס שאינו טיפוס דוט-נט (גם לא טיפוס DO אחר, למעט enum)
  - לא יכללו שום סוג של אוסף (גם לא יכללו מערכים)
- **מחלקת DalFactory** הינה חלק מתבנית עיצוב Simple Factory
  - תהיה **מחלקה סטטית**
  - תכיל פעולה GetDal
    - בסיסית (מבנה 1):
      - פרמטר אחד מסוג מחרוזת
      - טיפוס ערך מוחזר IDal
      - מחזירה אובייקט מטיפוס שמימש את הממשק IDal בהתאם (אובייקט של DalObject או של DalXml) ע"פ המחרוזת שהתקבל בפרמטר
    - **לבונוס (מבנה 2):**
      - ללא פרמטרים
      - טיפוס ערך מוחזר IDal
      - מחזירה אובייקט מטיפוס שמימש את הממשק IDal בהתאם (אובייקט של DalObject או של DalXml) ע"פ התצורה בקובץ dal-config.xml
      - תשתמש ב-reflection על מנת לטעון את האסמבלי המתאים וליצור את האובייקט המוחזר
      - ראו בנספח המלצות נוספות (חלקיות) איך לבצע את הבונוס הזה
- הפעולות המוגדרות **בממשק IDal**:
  - יכולות לקבל פרמטרים מטיפוסים של דוט-נט, של DO, ושל דלגטים בהתאם.
  - יכולות להחזיר ערכים מטיפוסים של דוט-נט, של DO, ואוספים של DO (רק ע"י IEnumerable גנרי)
- הפעולות בממשק IDal ייבנו לפי הישויות של DO ובהתאם לסכימה של CRUD
  - סכימה CRUD או Create-Request-Update-Delete - פעולות הוספה, בקשה (אובייקט בודד, אוסף כולו, או אוסף לפי תנאי סינון), עדכון, מחיקה)
  - תנאי סינון יכול להיות ע"י פעולה נפרדת או ע"י פעולה שמועברת כארגומנט לפרמטר מטיפוס פרדיקט
  - יתכן שחלק מהישויות לא זקוקות לחלק מפעולות CRUD לפי לוגיקת המערכת שהסטודנטים בונים - אין צורך לכתוב פונקציות שלא צריך אותן
- כל מחלקה המממשת את הממשק IDal (מחלקות DalObject ו-DalXml) חייבת לממש תבנית עיצוב Singleton
- במימושים של הממשק IDal אסור לבצע כל חישוב או בדיקה לוגית למעט בדיקת שלמות הנתונים (למשל בדיקת הימצאות אובייקט שנדרש למוחקו או לעדכנו, או אי הימצאות אובייקט שנדרש להוסיפו)

- **במימושים של הממשק IDal** באופן כללי:
  - שכבת DAL אחראית רק לגישה לנתונים, לכן השכבה לא תבצע שום בדיקות נתונים למעט המצאות או אי הימצאות של מופע הישות - בהתאם לפעולה - ע"פ מזהה או מפתח ייחודי של הישות. השכבה תזרוק חריגה מתאימה במקרה של תקלה כלשהי
  - **לבונוס**: במקרה של מחיקה אין למחוק את הישויות אלא יש לסמן ישות שלא פעילה (בישויות הנתונים המתאימות יש להוסיף שדה בהתאם) - זאת מכיוון שישויות אחרות עשויות להכיל מזהה של הישות הזו.
  - בהוספה של ישות עם "**מזהה טבעי**"[1] (**שאיננו** אוטומטי - למשל מספר רץ) יש לוודא שישות עם

[1] מזהה טבעי - למשל מספר תעודת זהות עבור לקוח, מספר סריאלי עבור ציוד כלשהו, מספר סניף\תחנה הנקבע ע"י החברה, וכדומה

מזהה זהה לא קיימת באחסון הנתונים (או קיימת אך לא פעילה - **לפי הבונוס של המחיקה כנ"ל**)
- במחיקה או עדכון הנתונים יש לוודא שישות עם המזהה הנדרש קיימת (ופעילה - **לפי הבונוס של המחיקה כנ"ל**)
- על מנת לנהל מספר מזהה ייחודי עבור ישות מסוימת שאין לה שדה מזהה טבעי ייחודי יש להשתמש במספר רץ המנוהל בישות של קונפיגורציה (מחלקה Config) שאינה חשופה לשכבת BL (זאת אומרת - אינה חלק מ-DO)
- אפשר שפונקציית הוספת ישות עם מספר רץ כנ"ל תחזיר את המספר המזהה הזה
- פונקציות הוספה ועדכון כללי יקבלו בפרמטר את אובייקט הישות המתאימה
- ייתכנו מתודות עדכון ייחודיות (שינוי מצב בדרך כלל) אשר יקבלו בפרמטרים את מזהה מופע הישות וערך העדכון או "הנחיית" עדכון בדרך אחרת המתאימה לכם
- עבור מתודות המחזירות רשימות:
  - סוג ערך מוחזר יהיה IEnumberable<> המתאים
  - אין להשתמש בלולאת foreach במקום שניתן להשתמש ב-LINQ
  - עבור רשומות מסוננות יש להגדיר פרמטר מסוג פרדיקט מתאים - שכבת ה-BL תגדיר את תנאי הסינון (כי זוהי לוגיקה)
  - **לבונוס של המחיקה כנ"ל**: אין לכלול אובייקטים לא פעילים בתשובה לבקשות רשימת\אוסף אובייקטים

- במימוש בזיכרון שנקרא **DalObject**:
  - מחלקת DataSource
    - מחלקה סטטית ללא הרשאה (internal) שמהווה מקום אחסון נתונים עבור מימוש DAL בזיכרון (DalObject)
    - נמצאת באותו פרויקט ומרחב שמות יחד עם DalObject
    - המחלקה כוללת את האיברים הבאים (כולם בהרשאה internal):
      - רשימות (סטטיות) כל הישויות (List<>) של DO
      - מחלקה פנימית Config עם משתני תצורה של האפליקציה ומשתני שדות פתח אוטומטיים ("מספרים רצים")
      - פעולת אתחול Initialize שתפקידה לבנות נתונים ראשוניים שיהיו מוכנים תמיד עם תחילת ההרצה של האפליקציה - כל הנתונים חייבים להיות מתואמים ותקינים מכל הבחינות
  - ל-DalObject אסור להחזיר הפניה לרשימו ב-DataSource בשום מצב
    - אפשר ליצור העתק בפשטות ע"י החזרת

```
return DataSource.SomeEntities.Select(item=>item);
```

או

```
return from item in DataSource.SomeEntities
       select item;
```

- במימוש בקבצי XML שנקרא **DalXml**:
  - לא יישמרו הנתונים בזיכרון אלא בתוך פעולות בלבד
  - מספרים רצים ומידע נוסף על קונפיגורציה ינוהלו בעזרת קובץ XML של קונפיגורציה המתעדכן אוטומטית בכל עדכון (יצירה של מספר רץ חדש)
  - כל פעולה תבצע קריאת הנתונים הרלוונטיים מקבצי XML בתחילתה
  - כל פעולה המבקשת לעדכן נתונים תבצע כתיבה לקבצים המתאימים לפני סיומה

נ.ב. ייתכן שבזמן ההצגה של הפרויקט תתבקשו להריץ את המערכת יותר מפעם אחת בו זמנית - ועדיין העבודה ביניהם צריכה להיות מתואמת!

תיאור שלבי הפרויקט בקצרה

- **שלב 1** (כחצי שבוע)
  השלמת המערכת על פי מודל השכבות ע"פ אחד המבנים לעיל כולל שימוש בתבניות עיצוב בהתאם.
  יש לוודא הגדרת פרויקטים מתאימים בסביבת העבודה.

- **שלב 2** (כשבועיים עד שבועיים וחצי)
  השלמת ממשק גרפי מלא תוך שימוש בטכניקות מתקדמות של WPF.

- **שלב 3** (כשבוע)
  בניית מימוש חדש של IDal עבור שכבת נתונים המאחסן את כל המידע בקבצי XML.

- **שלב 4** (כשבוע)
  נבנה סימולטור של רחפנים תוך שימוש בתהליכונים

שלב ראשון – מימוש מודל השכבות מלא

כללי

יש לממש באופן מלא את ארכיטקטורת (מודל) שלושת השכבות כפי שנלמד בקורס. בשלב הזה יש לבחור את הארכיטקטורה בין מבנה 1 לבין מבנה 2 לבונוס.

שכבת הנתונים DAL

- ההנחיות מתייחסות לארכיטקטורה של מבנה 1
  - כל מי שבחר מבנה 2 לבונוס - עליו להבין ולהסיק את השינויים המתאימים בהנחיות

מי שמתקשה - יש להישאר במבנה 1 ולא לחפש ולשאול אחרים איך לעשות ואיזה שינויים לעשות

- יש לשנות את מרחב השמות IDal ל-DalApi
- יש לשנות את מרחב השמות IDAL.DO ל-DO
- יש ליצור מחלקה סטטית DalFactory (הרשאה public)
  - במחלקה תהיה מתודה סטטית GetDal:
    - הרשאה public
    - פרמטר מסוג מחרוזת
    - טיפוס ערך מוחזר IDal
    - בהתאם למחרוזת, המתודה תחזיר מופע של DalObject *או של DalXml (בהמשך)*, אחרת תיזרק חריגה מתאימה (שתוסף לחריגות של DO) או להשתמש בחריגה המתאימה ביותר של דוט-נט (מ-System בלבד)
- יש למחוק את הרשאת המחלקה DalObject (זאת אומרת להפוך אותה ל-internal)
- יש להפוך את המחלקה DalObject לסינגלטון (Singleton) כאשר התכונה Instance תהיה עם הרשאה internal
  - לבונוס - לעשות סינגלטון Thread Safe ועם Lazy Initialization מירבי - אבל רק אם אתם יודעים להסביר למה מה שעשיתם הינו Thread Safe ו-Lazy Initialization

שכבת לוגית BL

- יש לשנות את מרחב השמות IBL ל-BlApi
- יש לשנות את מרחב השמות IBL.BO ל-BO
- יש ליצור מחלקה סטטית BlFactory (הרשאה public)
  - במחלקה תהיה מתודה סטטית GetBl:
    - הרשאה public
    - ללא פרמטרים
    - טיפוס ערך מוחזר IBL
    - המתודה תחזיר מופע של BL
- יש להפוך את המחלקה BL לסינגלטון (Singleton) כאשר התכונה Instance תהיה עם הרשאה internal
  - לבונוס - לעשות סינגלטון Thread Safe ועם Lazy Initialization מירבי - אבל רק אם אתם יודעים להסביר למה מה שעשיתם הינו Thread Safe ו-Lazy Initialization
- יש לעדכן את אתחול אובייקט של IDal כך שישתמש ב-DalFactory על מנת לקבל את המימוש המתאים של IDal

שכבת התצוגה PL

- יש לעדכן את אתחול אובייקט של IBL כך שישתמש ב-BlFactory על מנת לקבל את המימוש המתאים של IBL

שלב שני – השלמת שכבת התצוגה

המערכת הבסיסית תספק ממשק שיאפשר את ניהול מרכז המידע של החברה המספקת את שירות המשלוחים.
לבונוס, יש לספק אפשרות לשני סוגים של משתמשים, עובדי החברה המנהלים מידע במערך השליטה של החברה ובנוסף לקוחות שרוצים להשתמש בשירות המשלוחים

נ.ב. עיצוב ממשק המשתמש יכול להתבסס על טכנולוגיות שונות (חלונות, עמודים (Page), חלון יחיד עם לשוניות או "מדפי הרחבה", וכל עיצוב אחר שתבחרו). לשון ההנחיות יתייחס לטכנולוגית חלונות בלבד - תתאימו את העבודה לשכם בהתאם לטכנולוגיה שבחרתם.

להבדיל מתרגיל 3, **חובה** להשתמש ב-Data Binding על מנת לסנכרן את התצוגה עם הנתונים - יש להשתדל למעט עד כמה שאפשר את כמות הקוד האחורי למינימום הנדרש.

התכנית תיפתח במסך ראשי המפנה לאפשרויות השונות. כאשר יהיו לפחות המסכים הבאים:

- חלון כניסה ראשי - הפניה לתצוגות השונות
- בונוס :
  - אפשרויות כניסה
    - מנהל / עובד בחברה
    - לקוח קיים
    - רישום למערכת

ממשק ניהול מרכז המידע של החברה:

- תצוגת רחפנים (נעשה בתרגיל 3)
  - יש לשנות את המימוש לפי הדרישות לעיל
  - יש להשלים במקרה הצורך
  - בחלון רשימת רחפנים יש להוסיף מצב תצוגה "מקובצת" ע"פ מצב הרחפן ((יש לאפשר רענון - כפתור רענון?)
  - יש לאפשר פתיחת חלון חבילה המועברת ע"י הרחפן (אם ישנה כזו)
  - **שימו לב** שבתרגיל 3 מומלץ להוסיף שדה מטיפוס תאריך-זמן עבור כניסה לטעינה בישות נתונים "טעינת רחפן" - מי שלא עשה את זה - עכשיו הזמן לתיקון
- תחנות הבסיס:
  - הצגת רשימת התחנות
    - יש לאפשר מצב תצוגה רגילה של רשימה
    - יש לאפשר מצב תצוגה של רשימה "מקובצת" לפי הימצאות עמדות טעינה פנויות, או לפי כמויות עמדות טעינה פנויות (יש לאפשר רענון - כפתור רענון?)
  - תצוגת פרטים של תחנה (חלון אחד לכל המצבים עם התאמה בדומה לחלון רחפן בתרגיל 3)
    - הוספה, עדכון ~~ומחיקה~~ של תחנה
    - יש לאפשר פתיחת חלון רחפן מתוך רשימת הרחפנים בטעינה
- תצוגת לקוחות :
  - הצגת רשימת הלקוחות
  - תצוגת פרטי לקוח (חלון אחד לכל המצבים עם התאמה בדומה לחלון רחפן בתרגיל 3)
    - הוספה, עדכון ~~ומחיקה~~ של לקוח
    - יש לאפשר פתיחת חלון חבילה מתוך רשימות החבילות של הלקוח (רשימות החבילות הנשלחות והחבילות המתקבלות ע"י הלקוח)
- תצוגת חבילות
  - הצגת רשימת החבילות
    - יש לאפשר מצב תצוגה רגילה של רשימה
    - יש לאפשר מצב תצוגה של רשימה "מקובצת" לפי שולח או לפי מקבל (Grouping)
    - יש לאפשר סינון רשימה לפי קריטריון או קריטריונים מתאימים (טווח תאריכים, מצב חבילה, וכדומה)

- תצוגת פרטי חבילה (חלון אחד לכל המצבים עם התאמה בדומה לחלון רחפן בתרגיל 3)
  - הוספה, עדכון ומחיקה של חבילה
  - יש לאפשר פתיחת חלונות לקוח ורחפן בהתאם עבור הרחפן המוביל את החבילה (רק אם החבילה שוייכה אך עוד לא סופקה), עבור השולח ועובר המקבל של החבילה
  - יש לאפשר מתן אישור לאיסוף ולהספקת החבילה
  - מחיקת חבילה תתאפשר רק אם עוד לא שויכה לרחפן

- **תצוגת פרטי ישות תהיה ע"פ ישות של BO (רחפן, תחנה, לקוח, חבילה)**
- **תצוגת רשימות תהיה ע"פ ישות לרשימה מתאימה של BO (רחפן לרשימה, וכו')**
- **עליכם להשלים את הישויות הלוגיות וישויות הנתונים כולל כל התכונות הפנימיות - ע"פ הנדרש בממשק המשתמש שאתם מפתחים**

- **בונוס ממשק לקוח:**
  - תינתן אפשרות להירשם ולהיכנס למערכת כלקוח.
  - תתאפשר הוספת שליחת חבילה
  - תתאפשר הצגת חבילות של הלקוח בלבד
  - תתאפשר אישור איסוף ואישור קבלת חבילה
  - ועוד

- **בונוס שיפור ארכיטקטורה של שכבת תצוגה PL:**
  - המטרה - DataBinding מירבי
  - יצירת מחלקות מודל עבור כל אחד מהחלונות (DroneModel, וכו') במרחב שמות Model
  - יצירת ישויות תצוגה (PO) בהתאם לצרכי התצוגה כולל תוספת קוד שנדרש ל-DataBinding מלא במרחב שמות Model
  - קישור DataContext של חלונות מתאימים למודל שלהם

שלב שלישי - שמירת נתונים בקבצים

הוספת פרויקט DalXml (בארכיטקטורה המורחבת - מבנה 2) או מחלקה DalXml ומחלקות עזר שלה בפרויקט DAL (במרחב שמות DalXml) (בארכיטקטורה הפשוטה יותר - מבנה 1) - המחלקה תוגדר כסינגלטון ותממש את הממשק IDal באמצעות Linq-To-XML ובעזרת Xml Serializer.

יש להכין קבצי xml שיחליפו את האוספים שיצרנו כבר (כלומר יחליפו את מקור הנתונים) - קובץ אחד לכל אחת מהישויות של DO, אשר יכתבו בפורמט המתאים למבנה היישות אותה הוא מייצג. כדאי להוסיף תת-תיקיה בשם Data בתיקיית ה-solution, עבור מי שפתח אותה עבור ארכיטקטורה מהמורחבת - שימו לב שזו תיקייה שנמצאת רמה מעל תיקיית bin.

הערה : בהגדרות ignore, מוגדר שתיקיית bin המכילה קבצי ריצה לא תעלה למאגר גיט, ולכן אם נשים בתיקייה זו קבצי מידע, הם לא יסונכרנו.

המודולים הקשורים ל-DalXml יוגדרו במרחב שמות DAL או DL בדומה ל-DalObject. מי שהשתמש כבר בארכיטקטורה המורחבת - אנא לשים לב להתאמות האפשריות שתידרשנה בקובץ dal-config.xml עם שמות המחלקה, הפרויקט ומרחב השמות לא יהיו זהים לברירת המחדל.

המחלקה (סינגלטון) בשם DalXml מממשת ממשק IDal כפי שכבר הוזכר..

על הפרויקט חלות כל הדרישות הכלליות והטכניות כמו ב-DalObject.

ניתן ליצור מחלקות עזר לעבודה עם קבצי xml שבין היתר תהיינה אחראיות על כל האיתחולים, אפשרות שמירה וטעינה של קובץ וכן אפשרות תשאול ע"י שאילתת Linq. לאחר מכן יש לממש את כל המתודות של הממשק של ה-IDal. לגבי עבודה עם קבצי XML מקומיים, עיין בנספח.

בשכבה הזו עליך להחזיק נתוני קונפיגורציה רלוונטיים בתוך קובץ config.xml (אפשר להוסיף אלמנטים בהתאם לקובץ הקיים.
כגון :

- ה"מספרים הרצים" הנדרשים לכמה מהישויות. הנתונים שיתווספו לקובץ הם מספר רץ עבור המפתח הייחודי של חבילה ושל ישויות נוספות עבור מי שהוסיף בהם מזהה שהוא מספר רץ.
- נתוני צריכת הסוללה

מבחינת דרישות השכבה:

- לפחות קובץ אחד של אחת הישויות של DO, יש להשתמש ב- linq to xml עבור כל פעולות ההוספה עדכון ומחיקה ושליפה (להשתמש באובייקטים של XElement)
- חובה לטעון את הנתונים בתחילת של כל פעולת השכבה ולשמור אותם בקובץ בסוף כל פעולת הוספה\עדכון\מחיקה על מנת לאפשר עבודה מקבילה של מספר הרצות היישום
- לא תהיה שמירת נתונים בין בקשה לבקשה בשכבה הזו - כל האובייקטים והאוספים יווצרו מקומית בתוך הפעולות, בסיום הפעולות יגיעו בכח עצמם למאסף הזבל (GC).
- במימוש של שאר הישויות ניתן להשתמש ב-Serialize, כלומר לשמור את זה ל-XML באמצעות xmlSerialize

שלב רביעי - סימולטור רחפנים, תהליכונים

תיאור כללי של הסימולטור

בשלב הזה של המיני-פרויקט נדרש לממש סימולטור של רחפן שיפעל בשכבת BL. התפקיד של הסימולטור הוא להפעיל את הרחפן בצורה אוטומטית לבחירת חבילה לביצוע משלוח, ביצוע משלוח וחוזר חלילה - עד שאין מספיק סוללה לבצע שום משלוח. בשלב הזה הרחפן יגיע לתחנת בסיס הקרובה שיש שם עמדות פנויות, יעגון לביצוע הטענה עד שהסוללה מגיעה ל-100%. לאחר מכן הרחפן יחזור לבצע את השליחויות.

1. בחירת הרחפן למשלוח מתבצעת ע"י המתודה שכתבתם ב-BL בתרגיל 2, להזכירכם:
   a. בבחירת חבילה למשלוח, הסימולטור יבחר את המשלוח בעדיפות הגבוהה ביותר (קודם כל) ובמשקל הכבד ביותר (באותה רמת העדיפות) שהוא יכול לבצע.
   b. יכולת לבצע משלוח משמעותה שהסוללה תספיק להגיע לשולח ולאסוף את המשלוח, להוביל את המשלוח ליעד, ואז להגיע לתחנה הקרובה ביותר להטענה.

2. בעת השלמת המשלוח תיבדק אפשרות לבצע משלוח נוסף. אם אין אפשרות כזו - יטוס הרחפן לתחנת בסיס הקרובה ביותר עם עמדות פנויות.

3. ***בעת חיפוש אחר תחנת בסיס לטעינה ייתכן מצב שבתחנה הקרובה אין עמדות טעינה פנויות ואין מספיק סוללה להגיע לתחנה אחרת שיש בה עמדות פנויות. על הסטודנטים יש להציע ולממש התנהגות הרחפן במקרה כזה.***

4. לאחר השלמת הטעינה הרחפן ינסה לחזור לבצע שליחויות. אם בהשלמת הטעינה אין משלוחים שהרחפן יכול לבצע - הרחפן יבדוק אפשרות משלוח מידי פעם (לפי שעון [טיימר] צעד השהיה בסימולטור - ראו בהמשך) תוך כדי הישארות במיקום התחנה שבה נטען הרחפן.

5. **<u>המלצה</u>**: לאחר השלמת החבילה ולאחר השלמת הטעינה יעבור הרחפן למצב פנוי במשך טיימר צעד השהיה, מה שיאפשר תחושה נוחה מבחינת חוויית משתמש (ויזואלית) ועשוי להקל על אלגוריתמיקת הסימולטור ועל דיבוג.

6. בכל שלב - של תנועה והטענה - יעודכן מצב הסוללה מחזורית לפי טיימר צעד השהיה בסימולטור.

7. גודל טיימר צעד ההשהיה יוגדר כקבוע במחלקת הסימולטור מסוג שלם, ביחידות של מילישניות (msec).

8. מהירות תנועת הרחפן תוגדר כקבוע במחלקת הסימולטור מסוג דצימלי (double), ביחידות של ק"מ לשניה (הכן - על מנת ליהנות מסימולציה מהירה - הרחפנים שלכם יהיו כנראה על-קוליים 😉).

9. קצב טעינה הסוללה של הרחפנים (ב-% לדקה, או לשניה - לבחירתכם) מסוג double, כמו כן צריכת סוללה ב-% לק"מ מסוג double עובר כל מצב תנועה (ללא מטען, חבילה קלה\בינונית\כבדה) הגדרתם בתרגילים 1 ו-2 והשתמשתם בתרגיל 3 ובשלבים הקודמים של הפרויקט. שימו לב על היחידות והטיפוסים כפי שמופיע בסעיף הזה ותעדכנו את הקוד הקיים במידת הצורך. ההנחיה הזו לא תחזור בהנחיות המפורטות בהמשך אך היא עדיין בגדר חובה.

10. המרחק שיופיע בישות "חבילה ברחפן" יבטא מרחק מהמיקום הנוכחי של הרחפן עד ליעד התנועה הנוכחית.
    a. לאחר בחירת חבילה - עד לשולח החבילה.
    b. לאחר איסוף חבילה - עד למקבל החבילה.

11. מיקום הרחפן יעודכן בהגעת הרחפן לכל נקודה - השולח, המקבל, תחנת בסיס - בעת איסוף ואספקת חבילה, הגעה לתחנת בסיס.
    a. **לבונוס**: עדכון מיקום הרחפן יתבצע גם בתנועה במחזוריות של טיימר צעד השהיה

12. בשלבים הבאים יתבצע עדכון תצוגה בכל החלונות הרלוונטיים הפתוחים של כל הנתונים המוצגים הרלוונטיים:
    a. כל עדכון סוללה יגרום עדכון מצב סוללה בחלונות רשימת הרחפנים ורחפן (מתאים).
    b. **לבונוס (חלק מהבונוס בסעיף a.11.)**: עדכון מיקום הרחפן יתבצע גם בתנועה במחזוריות של טיימר צעד השהיה - בחלונות רשימת רחפנים ורחפן

c. בחירת חבילה למשלוח - עדכון תצוגה בחלונות רשימת רחפנים, רחפן, רשימת חבילות, חבילה (מתאימים)
d. איסוף חבילה - עדכון תצוגה בחלונות רחפן, רשימת חבילות, חבילה
e. אספקת חבילה (למקבל) - עדכון תצוגה בחלונות רשימת רחפנים, רחפן, רשימת חבילות, חבילה, לקוח (השולח והמקבל - ברשימת החבילות שנשלחו\יועדו ללקוח - החבילה תעבור לרשימה מתאימה של חבילות שכבר הגיעו ליעדן).
f. בחירת תחנת טעינה (סטטוס הרחפן ועמדת טעינה בתחנה תיתפס כבר בשלב הזה) - חלון רשימת רחפנים, רחפן, רשימת תחנות, תחנה.
g. תחילת טעינה (הגעה לתחנת בסיס לצורך טעינה) - רשימת רחפנים, רחפן
h. סיום טעינה (הרחפן יישאר בתחנה עד לבחירת חבילה למשלוח), אך עמדת הטעינה תשוחרר - בחלונות רשימת רחפנים, רחפן, רשימת תחנות, תחנה.

13. הסימולטור ירוץ בתהליכון נפרד **בעזרת פועל רקע** (BackgroundWorker) שיופעל מחלון הרחפן - עבור אותו הרחפן.
a. **לבונוס**: תתאפשר הפעלת סימולטור רחפן עבור מספר רחפנים במקביל.

14. **שימו לב: כפי שנלמד בשיעורים - התהליכון הראשי (תהליכון UI) הינו התהליכון היחיד שיכול לעדכן נתונים ב-PL ולעדכן תצוגה.**

15. *מכיוון שתהיה גישה ועדכון נתונים במקביל משניים או יותר תהליכונים - הנתונים הם נתונים משותפים עבור קטעי קוד מקביליים ויש סכנת יצירת חוסר תיאום בנתונים באמצע פעולות - מה שעלול לגרום שיבוש נתונים, שיבוש תצוגה, ואף קריסת האפליקציה עם חריגות שעוד לא הכרתם עד כה (בעיה שנקראת "בעיית קטע קריטי"). קטעי קוד אלה נקראים "קטע קריטי" כפי שתלמדו בקורס עקרונות מערכות הפעלה בהמשך, וחלה חובה להגן על כניסה לקטע קריטי וניהול תור המבקשים להיכנס לקטע קריטי. למזלכם, הכלים הקיימים בשפה C# פשוטים לשימוש - ראו בהמשך בהנחיות מפורטות. על משמעות הפנימית ותפקוד הכלים האלה תלמדו לעומק בקורס עקרונות מערכות הפעלה.*

שכבת התצוגה - הפעלת סימולטור

בחלון הרחפן (במצב עדכון) יש להוסיף כפתור למעבר למצב אוטומטי (הפעלת סימולטור). אפשר לתת בכפתור כיתוב "Automatic" או "Simulator" או בעברית למי שהממשק שלו בעברית, או משהו אחר שיהיה ברור מטרת הכפתור. אפשר גם להשתמש בכל סוג פקד אחר המתאים לעיצובכם להפעלת הסימולטור.
בלחיצה על הכפתור ייבנה פועל רקע (BackgroundWorker) מתאים, שיכלול עדכוני ביניים (progress) ואפשרות עצירה (cancellation), כאשר בדלגט הראשי שלו (DoWork) תופעל בקשת הפעלת סימולטור בשכבת BL. לאחר מכן פועל הרקע יופעל.
נ.ב. עבור הסימולטור ב-BL יש להעביר מתודות עדכון ביניים ועצירה אשר יבצעו את עדכון נתוני שכבת PL ועדכון תצוגה בהתאם (רצוי דרך קישור נתונים - Data Binding, כמובן).

לאחר הפעלת פועל הרקע יש להסתיר את כל כפתורי הפעולה האחרים ובמקום הכפתור (או תחליפו) הנ"ל יופיע כפתור (או תחליפו) עם כיתוב "Manual" או "Regular" וכדומה, בהתאם - שאפשר להבין ממנו שמדובר בחזרה למצב ידני \ עצירת הסימולטור.

בלחיצה על כפתור חזרה למצב ידני יסומן לסימולטור שהוא צריך לעצור (סימון cancellation של פועל הרקע). חזרה בפועל למצב ידני תתבצע רק לאחר שהסימולטור יסתיים. ניתן "להפעיל" פקד דינמי "מסתובב" או דומהו בהפעלת הכפתור ועד לעצירת הסימולטור. בחזרה למצב ידני, תחזור התצוגה לקדמותה כפי שהייתה לפני הפעלת הסימולטור.

בסגירת החלון של הרחפן שנמצא במצב אוטומטי יש לסמן לסימולטור להסתיים, למנוע את סגירתה החלון - וכאשר הסימולטור יסתיים - יש לסגור את החלון מיד. בהמתנה לסיום הסימולטור ניתן "להפעיל" פקד דינמי "מסתובב" או דומהו כנ"ל.

לבונוס (חלק מהבונוס בסעיף a.11. בתיאור הכללי של הסימולטור): תתאפשר פתיחת מספר חלונות רחפן (עבור רחפנים שונים) במקביל, ויהיה אפשר להעביר בכמה מהם או בכולם (ע"י לחיצות בכל חלון כזה) כך שיהיו במצב אוטומטי במקביל.

באתר הקורס מופיע סרטון הדגמה (אל תלינו על העיצוב - אנו בטוחים שעיצובכם יהיה הרבה יותר מרשים) ליחצו לצפייה.

הכנות בשכבת נתונים

לצורך מניעת שיבוש נתונים עקב בעיית קטע קריטי, בשכבת התצוגה - במימושים ב-DalObject וב-DalXml יש לעשות את התוספות הבאות:

- בתחילת **כל** מודול (קובץ C#) המכיל מימוש מתודות IDal, יש להוסיף את ההוראה הבאה:

```
using System.Runtime.CompilerServices;
```

- לפני כל הגדרת **כל** מתודה המממשת מתודה מהממשק IDal (זאת אומרת - לפני כל מתודה עם הרשאה public), יש להוסיף את האטריבוט הבא. הוספת האטריבוט ימנע ביצוע יותר ממתודה אחת באובייקט המחלקה (DalObject או DalXml) בו זמנית:

```
[MethodImpl(MethodImplOptions.Synchronized)]
public Customer GetCustomer(int customerId)
```

הכנות בשכבה לוגית

לצורך מניעת שיבוש נתונים עקב בעיית קטע קריטי, בשכבת התצוגה - במימושים ב-BL יש לעשות את התוספות הבאות:

- בתחילת **כל** מודול (קובץ C#) המכיל מימוש מתודות IBL, יש להוסיף את ההוראה הבאה:

```
using System.Runtime.CompilerServices;
```

- לפני כל הגדרת **כל** מתודה המממשת מתודה מהממשק IBL (זאת אומרת - לפני כל מתודה עם הרשאה public), יש להוסיף את האטריבוט הבא. הוספת האטריבוט ימנע ביצוע יותר ממתודה אחת באובייקט המלקה (DalObject או DalXml) בו זמנית:

```
[MethodImpl(MethodImplOptions.Synchronized)]
public Customer GetCustomer(int customerId)
```

- בכל מקום המבצע סדרת פעולות מתואמת ע"י פניות ישירות או עקיפות לשכבת נתונים (למשל - הוספת "טעינת-רחפן" והורדת עמדות טעינה פנויות בתחנה), יש לעטוף את הקטע הקריטי (קטע הקוד הכולל את סדרת הפעולות) ע"י בלוק נעילה לאובייקט שכבת הנתונים (נניח שבמשתנה\שדה\תכונה בשם dal מוחזקת הפניה לאובייקט שכבת הנתונים):

```
lock (dal)
{
    var baseStation = findClosestBaseStation(drone, charge: true);
    dal.AddDroneCharge(droneId, baseStation.Id);
    dal.BaseStationDroneIn(baseStation.Id);
    drone.Status = DroneStatuses.Maintenance;
    double distance = drone.Distance(baseStation);
    drone.Location = baseStation.Location;
    drone.Battery = Max(0, drone.Battery - distance * batteryUsages[DRONE_FREE]);
    drone.DeliveryId = null;
}
```

- שימו לב שגם שאילתא (מושהית - IEnumerable<>) המתבצעת בו במקום (למשל בעזרת FirstOrDefault או כל פונקציה מבצעת אחרת), גם כשזו פעולה אחת - צריך לעטוף כנ"ל:

```
int? nextParcel(DroneForList drone)
{
    lock (dal)
    {
        return dal.GetParcels(p => p?.Scheduled == null
                                  && (WeightCategories)(p?.Weight) <= drone.MaxWeight
                                  && requiredBattery((int)p?.Id) < drone.Battery)
                    .OrderByDescending(p => p?.Priority)
                    .ThenByDescending(p => p?.Weight)
                    .FirstOrDefault()?.Id;
    }
}
```

סימולטור בשכבה הלוגית

בממשק IBL יש להוסיף פונקציית להפעלת סימולטור אשר תקבל את מספר הרחפן, דלגט עבור פעולת עדכון תצוגה מטיפוס `Action` ודלגט עבור בדיקת עצירה מטיפוס `Func<bool>`. הפונקציה לא תחזיר שום ערך.

במימוש הפונקציה ב-BL יזומן בנאי של הסימולטור (ראו בהמשך). לבנאי יועברו בארגומנטים אובייקט ה-BL כמו שהוא וכל הפרמטרים שקיבלה הפונקציה משכבת PL. העברת אובייקט ה-BL הוא על מנת לאפשר לסימולטור להשתמש במתודות עזר שכבר כתבתם ב-BL. הרשאת מתודות כאלה, אם בסימולטור יהיה לזמן אותן, תשונה להרשאת אסמבלי\פרויקט (internal).

יש לבנות מחלקת סימולטור. כדלקמן:

- בתחילת מודול הסימולטור תידרשו כנראה להוראות הבאות לפחות:

```
using System;
using BO;
using System.Threading;
using static BL.BL;
using System.Linq;
```

- הרשאת המחלקה - internal (ז"א אין צורך לכתוב הרשאה).
- בנאי הסימולטור יקבל בפרמטרים אובייקט מטיפוס BL, כמו כן אותם הפרמטרים שקיבל פונקציית הפעלת הסימולטור כנ"ל.
- מכיוון שבסימולטור תצטרכו גישה לשכבת הנתונים, תשנו את ההרשאה של השדה dal במחלקה BL ל-internal, כך שתוכלו לגשת לשכבת הנתונים בלי לבקש את האובייקט מחדש מ-DalFactory (ובכך תימנע טעות אפשרית של פניה למימושים שונים של שכבת הנתונים ב-BL ובסימולטור).
- במחלקה יוגדרו שדות קבועים עבור מהירות הרחפן וטיימר צעד השהייה כפי שהוסבר בתיאור הכללי לעיל.
  - המלצה לטיימר צעד השהייה - 500 או 1000 (חצי שניה או שניה)
  - ערך המהירות תלוי בגודל האזור שבו בחרתם למקם את התחנות ואת הלקוחות.
- בבנאי תיכתב לולאה שתנאי העצירה שלה הוא כאשר פונקציית בדיקת עצירה מסמנת שצריך לעצור את הסימולטור.
- בתוך גוף הלולאה תוסף טיפול לפי מצב הרחפן וע"פ האלגוריתמיקה המתוארת בתיאור הכללי לעיל.
- בתחילת כל שלב מתאים בלולאה (תנועה או המתנה בטעינה או במצב פנוי) תתבצע השהייה בעזרת:

```
Thread.Sleep(DELAY);
```

- שימו לב שאין לכלול את ההשהייה בשום בלוק של **lock** (ראו בהמשך)
- כל קטע קוד המבצע סדרה של עדכונים ב-BL ו-\או מפעיל פונקציות BL - חובה לעטוף בבלוק חסימת קטע קריטי:

```
lock (bl) { ... }
```

- כל קטע קריטי כנ"ל הכולל גם פניות ישירות לשכבת נתונים - חובה לעטוף גם בבלוק חסימת קטע קריטי כפול:

```
lock (bl) lock (dal) { ... }
```

נ.ב. כל תוספת טכנולוגית וחזותית לפי בחירתכם ויציתיותכם שלא כלולה בחומרי הקורס אך תואמת את מטרות הקורס - תתקבל בברכה לבונוס ותקבל בין 1 ל-5 נק' לציון. כל תוספת כזו שמחוץ למטרות הקורס תשפר את סיכוייכם שהפרויקט שלכם ייבחר לתחרות הפרויקטים המצטיינים ואף יוכל לזכות בה.

בהצלחה רבה לכולם!

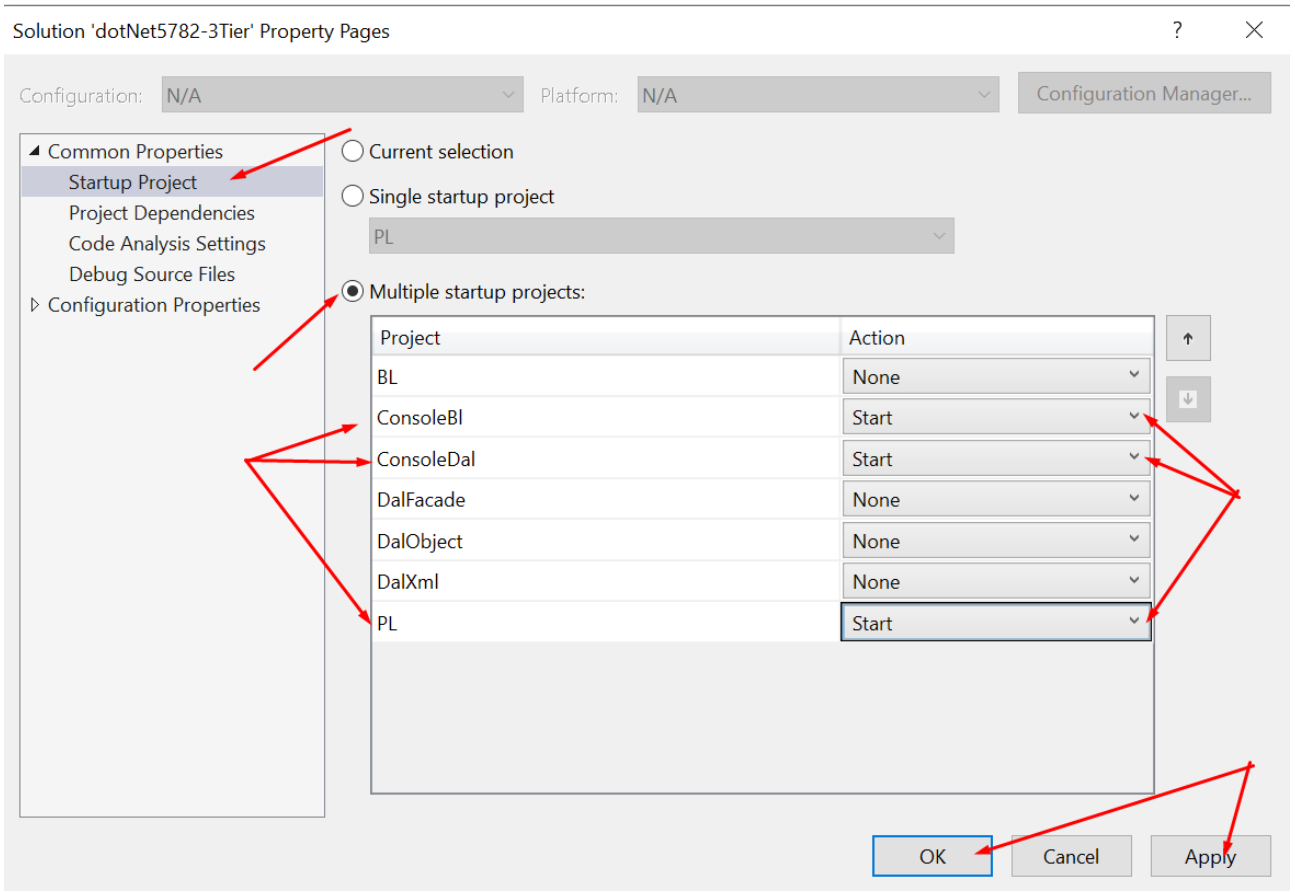
נספח א' - תצורת ה-Solution עם הפרויקטים ומבנה מתקדם של ארכיטקטורה

תיקון לשם פרויקט

על מנת למנוע בלבול, בארכיטקטורה של "מבנה 2" המתקדמת מומלץ לתת שם פרוייקט DalFacade במקום DalApi. המשך הנספח מסתמך על שם הפרויקט המתוקן.
לחלק מפרויקטים אחרים גם יש שם קצת שונה במידע שבנספח - אך קל לנחש למה הכוונה.

תצורת ה-Solution והפרויקטים

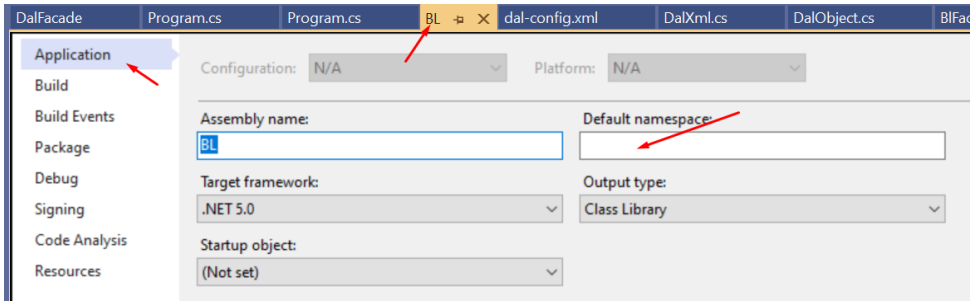
שימו לב על פרויקט ההרצה של ה-Solution שלכם מסומנים להרצה כל הפרויקטים של הרצה:

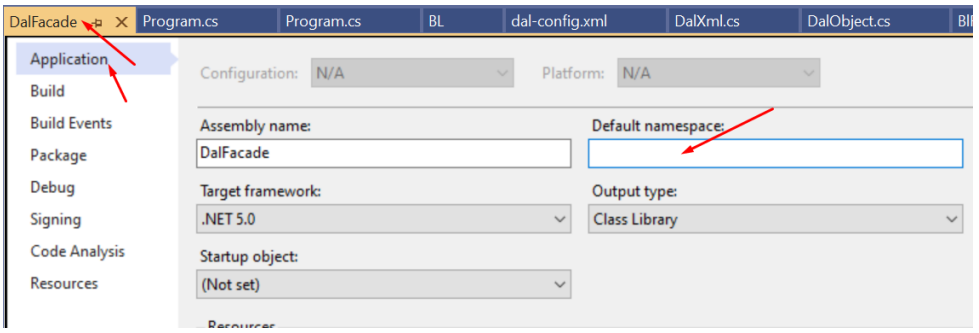


תצורת הפרויקטים

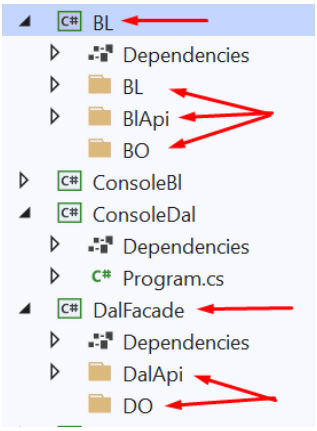
בפרויקטים של BL ושל DalFacade כדאי לארגן את תיקיות הפרויקט כך שמרחבי השמות של כל המודולים (החדשים) שתוסיפו בתוך התיקיות יקבלו אוטומטית את שם מרחב השמות המתאים. כך לא תצטרכו לעדכן את מרחבי השמות כל פעם.

1. יש לבטל את מרחב שמות ברירת המחדל במאפייני הפרויקט:

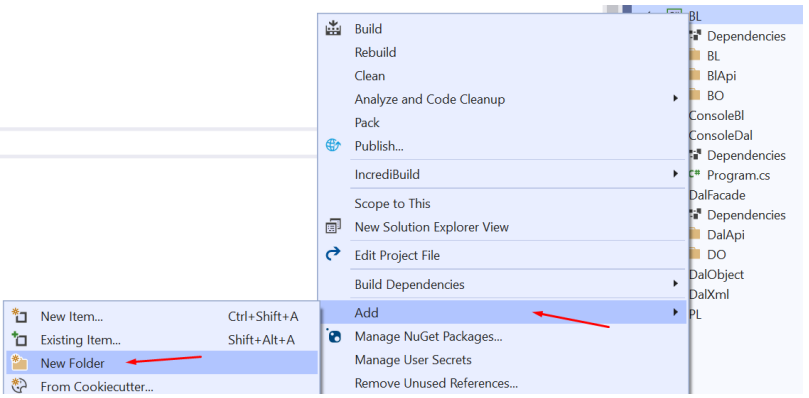




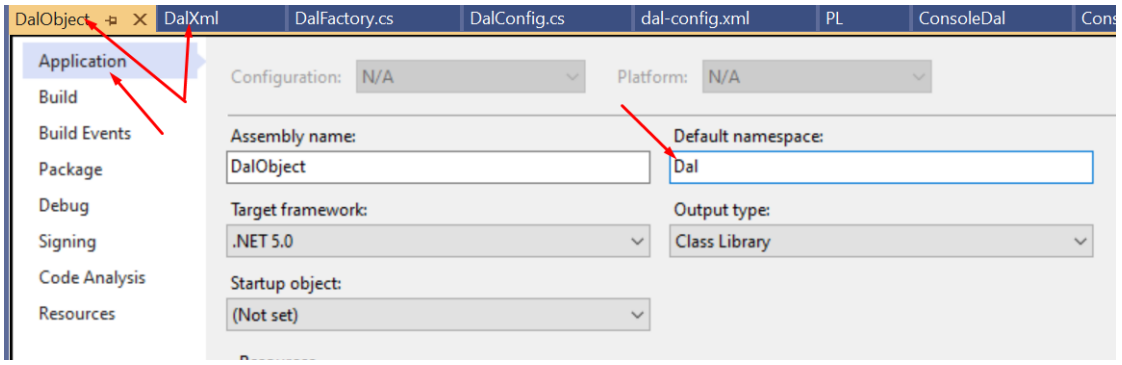
2. יש לחלק את הפרויקטים לתת-תיקיות



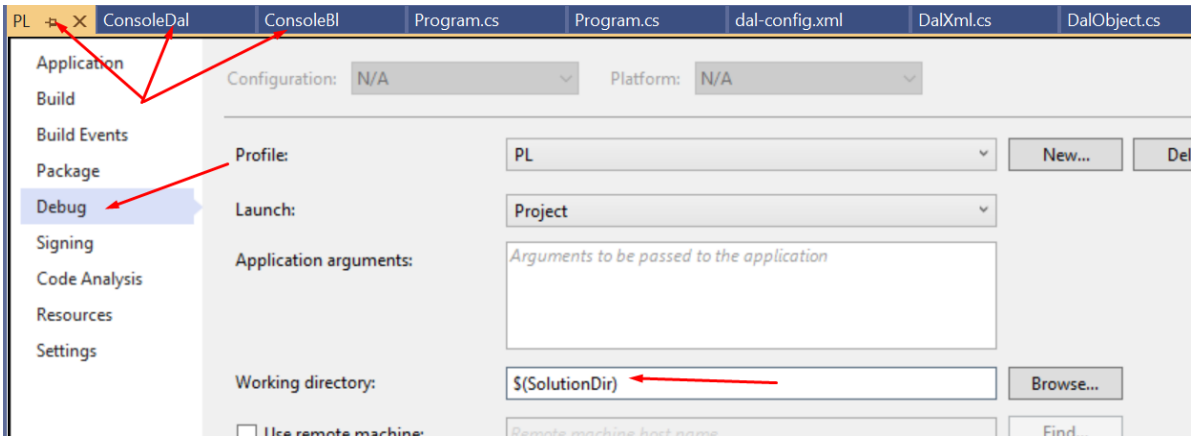
הפעולה מתבצעת ע"י הוספת תיקיה בפרויקט:



3. עבור הארכיטקטורה המתקדמת (מבנה 2), בכל הפרויקטים המממשים את הממשק IDal (פרויקטי **DalObject** ו-**DalXml**) יש להגדיר את מרחב השמות ברירת המחדל להיות **Dal**:



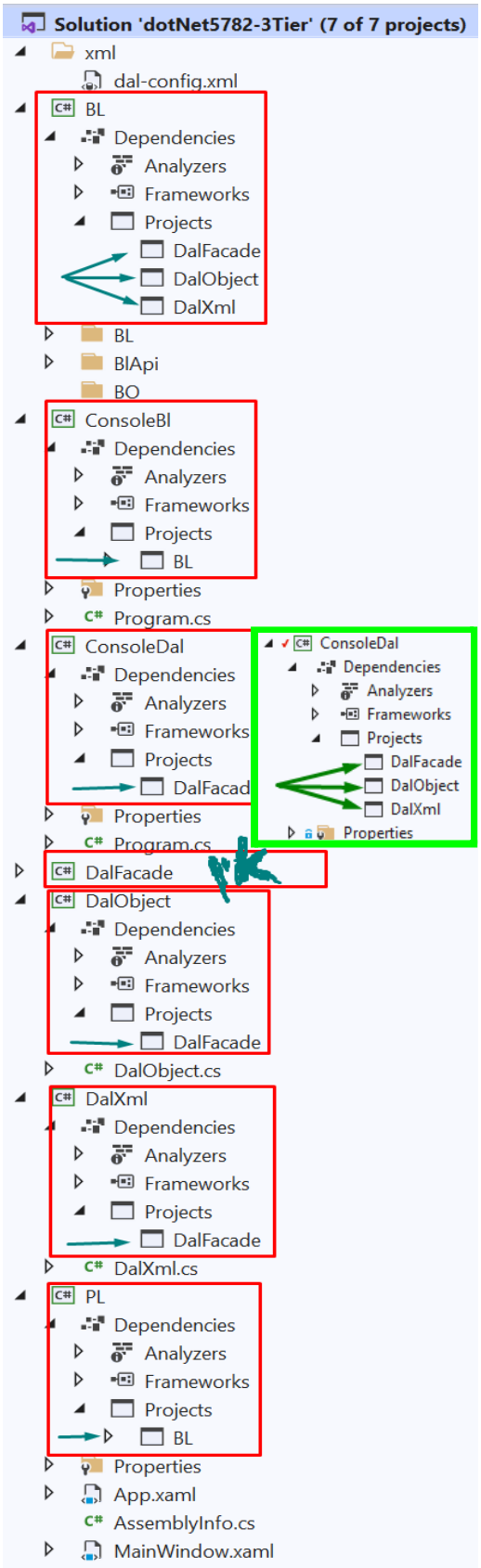
4. בכל הפרויקטים של הרצה (פרויקטי קונסול ו-WPF) יש להגדיר את תיקיית ההרצה במאפייני הפרויקט לתיקיה הראשית של ה-Solution שלכם. על מנת לעשות זאת יש לפתוח את מאפייני הפרויקט (עבור כל אחד מהפרוייקטים כנ"ל), לעבור למאפייני Debug של הפרויקט ולרשום בשדה Working Directory: את ההגדרה הבאה: $(SolutionDir), כדלקמן:



ההגדרה הזו תעזור לכם גם בשלב השלישי של הפרויקט בגישה לתיקיה עם קבצי ה-xml וגם בארכיטקטורה המתקדמת (מבנה 2).

תלויות בין הפרויקטים

על מנת שכל פרויקט יוכל לגשת לתוכן של פרויקט אחר, ועל מנת לאפשר לתוכנת בניית האפליקציה של מיקרוסופט (MSBuild) ליצור את קבצי האסמבלי בתיקיות יחד עם קבצי ההרצה, וגם על מנת לאפשר את טעינת האסמבלי הנדרש בארכיטקטורה המתקדמת (מבנה 2), יש לארגן את התלויות בין הפרויקטים בצורה נכונה. להלן תמונת ההגדרות של התלויות:



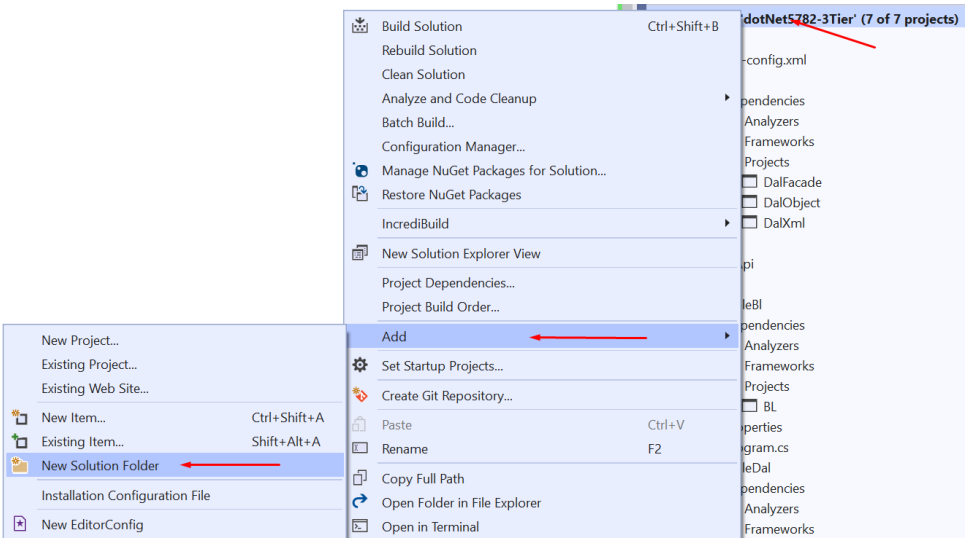
תצורת ארכיטקטורה מתקדמת (מבנה 2)

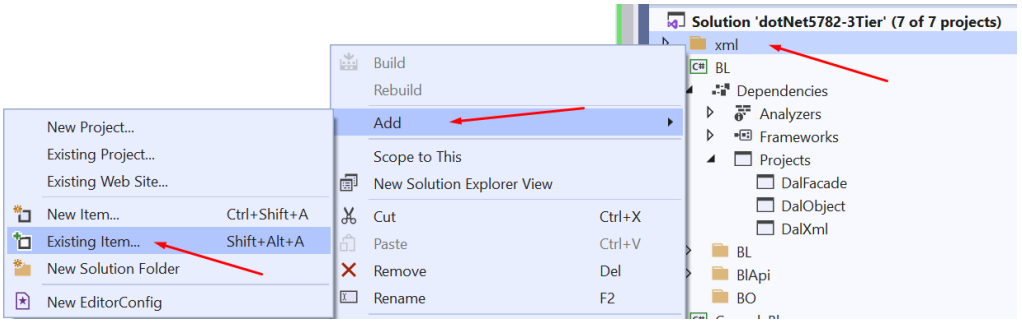
יש ליצור תת-תיקיה xml בתיקית ה-Solution בסייר קבצים ובה ליצור קובץ dal-config.xml עם התוכן הבא:

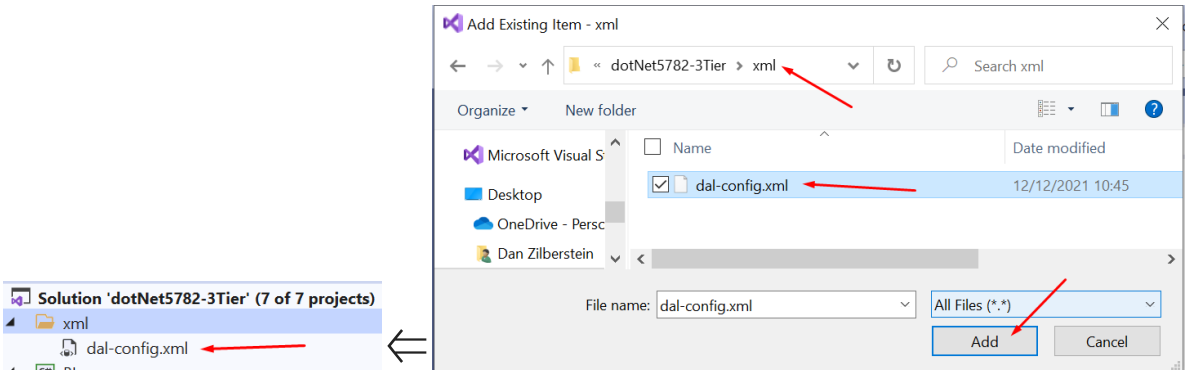
```
ConsoleDal | ConsoleBl | Program.cs | Program.cs | dal-config.xml
<?xml version="1.0" encoding="UTF-8"?>
<config>
  <dal>list</dal>
  <dal-packages>
    <list>DalObject</list>
    <xml>DalXml</xml>
  </dal-packages>
</config>
```

על שני הסטודנטים השותפים לפרויקט שמממשים את הארכיטקטורה המתקדמת חובה להבין את המבנה ואת התוכן של הקובץ.

לאחר מכן יש ליצור תיקיית Solution בשם xml ולהוסיף אליה את הקובץ הנ"ל שבניתם כלדקמן:







לאחר מכן בפרויקט DalFacade במרחב שמות (בתת-תיקיה) DalApi יש ליצור שתי מודולים של מחלקה: DalConfig ו-DalFactory כדלקמן:

```
DalConfig.cs | dal-config.xml | PL | ConsoleDal | ConsoleBl | Program.cs | Program.cs | DalXml.cs | DalObject.cs | BlFactory.cs | IBL.cs
DalFacade | DalApi.DalConfig
1   using System;
2   using System.Collections.Generic;
3   using System.Xml.Linq;
4   using System.Linq;
5
6   namespace DalApi
7   {
8       class DalConfig
9       {
10          internal static string DalName;
11          internal static Dictionary<string, string> DalPackages;
12          static DalConfig()
13          {
14              XElement dalConfig = XElement.Load(@"xml\dal-config.xml");
15              DalName = dalConfig.Element("dal").Value;
16              DalPackages = (from pkg in dalConfig.Element("dal-packages").Elements()
17                             select pkg
18                            ).ToDictionary(p => "" + p.Name, p => p.Value);
19          }
20      }
21      public class DalConfigException : Exception
22      {
23          public DalConfigException(string msg) : base(msg) { }
24          public DalConfigException(string msg, Exception ex) : base(msg, ex) { }
25      }
26  }
```

```
DalFactory.cs | DalConfig.cs | dal-config.xml | PL | ConsoleDal | ConsoleBl | Program.cs | Program.cs | DalXml.cs | DalObject.cs | BlFactory.cs | IBL.cs | MainWindow.xaml.cs | BL.cs
DalFacade | DalApi.DalFactory | GetDal()
1   using System;
2   using System.Reflection;
3
4   namespace DalApi
5   {
6       public class DalFactory
7       {
8           public static IDal GetDal()
9           {
10              string dalType = DalConfig.DalName;
11              string dalPkg = DalConfig.DalPackages[dalType];
12              if (dalPkg == null) throw new DalConfigException($"Package {dalType} is not found in packages list in dal-config.xml");
13
14              try { Assembly.Load(dalPkg); }
15              catch (Exception) { throw new DalConfigException("Failed to load the dal-config.xml file"); }
16
17              Type type = Type.GetType($"Dal.{dalPkg}, {dalPkg}");
18              if (type == null) throw new DalConfigException($"Class {dalPkg} was not found in the {dalPkg}.dll");
19
20              IDal dal = (IDal)type.GetProperty("Instance",
21                      BindingFlags.Public | BindingFlags.Static).GetValue(null);
22              if (dal == null) throw new DalConfigException($"Class {dalPkg} is not a singleton or wrong propertry name for Instance");
23
24              return dal;
25          }
26      }
27  }
```

כמובן שעל מנת לזכות בבונוס על שני הסטודנטים השותפים לפרויקט להבין את כל מה שמופיע בשני המודולים, לדעת להסביר כל שורה וסיבה של כל דבר במודולים האלה. כמו כן, מומלץ לשכלל את המודולים בהוספת תמיכה ב**אטריבוטים** של אלמנטים ברשימת החבילות על מנת לתת גמישות לשם מרחב השמות ולשם המחלקה המממשת את הממשק IDal בפרויקטים המתאימים. למשל (ללא ההרחבה הזו הבונוס יהיה חלקי בלבד):

```
<list class="MyDal" namespace="Dal">DalObject</list>
```

כמובן לא תהיינה הנחיות או עזרה מצד המרצים לתוספת הזו!

הבהרות

שלב 1 - השלמת ארכיטקטורה, כולל נספח א'

1. תיקון בנספח: ל-ConsoleDal יש תלוית\הפניות גם ל-DalFacade וגם ל-DalObject ו-DalXml

שלב 2 - השלמת ממשק משתמש

1. **פעולות מחיקה** בממשק משתמש ובפעולות שכבות:
   a. אם לא נעשה בונוס של מחיקה ע"י סימון "פעיל \ לא פעיל" - ניתן למחוק רק:
      - חבילה, אם עוד לא שויכה
      - טעינת-רחפן (בשכבת נתונים)
   b. מי שעושה בונוס של מחיקה ע"י "פעיל \ לא פעיל" יש לאפשר גם:
      - מחיקת תחנה: מרגע המחיקה (סימון כ"לא פעיל") לא ניתן לשלוח אליה רחפנים להיטען, אך רחפנים שכבר נשלחו - ישתמשו בתחנה עד לשחרור מטעינה
      - מחיקת רחפן: מרגע המחיקה (סימון כ"לא פעיל") לא ניתן לשלוח לטעינה ולא ניתן לשלוח להעביר חבילה, אך פעולה נוכחית תושלם במלואה
      - מחיקת לקוח: מרגע המחיקה (סימון כ"לא פעיל") לא יוכל לבקש שליחת חבילה חדשה ולא יוכלו לשלוח אליו חבילות חדשות
2. **עיצוב ממשק משתמש**, כפי שכבר כתוב במפורש בהנחיות - מבחינת כמות חלונות וארגון המידע וכדומה - ניתן לשיקול דעת של הסטודנטים, למשל יש שביקשו להציג את הרשימות בעזרת חלון יחיד וניתן לברור איזו רשימה מוצגת, וכדומה.
3. **אישור איסוף ואספקת חבילה**:
   a. בתרגיל 3 התבקשתם לעשות בחלון הרחפן
   b. בהנחיות כאן מופיע גם בהנחיות חלון חבילה. יש להפעיל היגיון בריא תוך שמירה על עקרון DRY. לכן ניתן לעשות אחד מהשניים:
      - להשאיר את הכפתורים (או הפעולות) בחלון הרחפן המוביל חבילה

      <u>או</u>
      - להעביר את הפעולות לחלון חבילה
   c. יחד עם זאת, מי שעושה בונוס של ממשק לקוח: יש לאפשר גם ללקוח לאשר שאספו ממנו חבילה (כמובן אם יש כמה חבילות הממתינות לאיסוף - אז יש "לשייך" את הפעולה לחבילה ספציפית) או שהגיעה אליו חבילה (כמובן אם יש כמה חבילות **<u>בדרך</u>** אליו [שכבר נאספו] - אז יש "לשייך" את הפעולה לחבילה ספציפית)

שלב 4 - סימולטור

1. נוספה בהנחיות התייחסות לסגירת חלון הרחפן כאשר הוא נמצא במצב אוטומטי.