### תיאור קצר של הפיצ'רים שבחרנו לממש בתרגיל הקודם:

* **פיצ'ר ראשון – הצגת מידע מהפייסבוק בצורה טבלאית עם יכולות מיון:**

פיצ'ר זה מאפשר למשתמש לבחור אוסף מידע מסוג כלשהו (תמונות, חברים, פוסטים, עמודים שאהב וכו'...) ולהציגו בצורה טבלאית עם יכולות מיון לפי ערך רצוי מבין הערכים המוצגים.  
כרגע מומשו רק האפשרויות הבאות:

* הצגת תמונות שהמשתמש העלה
* החברים של המשתמש
* העמודים שהמשתמש אהב
* כל הפוסטים בהם המשתמש מתוייג

אך אופן כתיבת הקוד נעשה כך שניתן בעתיד להוסיף אוספים אחרים של מידע בקלות רבה   
(לדוגמא: תמונות שהמשתש תוייג בהן ויש להן יותר מ20 לייקים)

\*פירוט נוסף תחת תבנית מס' 3 – Static Factory Class

מיקום בקוד: תיקיית DataTables וקובץ TabDataTables.cs.

* **פיצ'ר שני – ניתוח החברות של המשתמש עם אחד מחבריו לפי בחירה:**

פיצ'ר זה מאפשר למשתמש לבחור את אחד החברים שלו מרשימת החברים (שגם משתמשים באפליקציה)

לאחר מכן יוכל המשתמש להציג סטטיסטיקות כגון:

* כמה לייקים החבר עשה על תמונות של המשתמש
* רשימת כל התגובות של החבר על תמונות של המשתמש (בלחיצה כפולה תוצג התמונה עם פירוט נוסף על אותה התמונה)
* מהי התמונה האחרונה שהמשתמש והחבר תויגו בה יחדיו

בנוסף יוכל המשתמש למצוא את אוספי התמונות הבאים:

* תמונות שהמשתמש העלה והחבר מתוייג בהן (מקובץ לפי אלבום)
* תמונות שגם המשתמש וגם החבר מתוייגים בהן (מקובץ לפי שם המשתמש שהעלה את התמונה)
* תמונות של החבר שהמשתמש מתוייג בהן (מקובץ לפי אלבום)

\*\* בעת ניסיון להצגת התמונות המשתמש יכול לבחור באיזה אלבומים לחפש במקום לסרוק את כל האלבומים.

מיקום בקוד: קובץ TabFriendshipAnalyzer.cs (החלק הלוגי נמצע בקובץ FriendshipAnalyzer.cs).

### תבנית מס' 1 – Singleton

* סיבת הבחירה / שימוש בתבנית:

רצינו לייצר מחלקה שלמעשה תנהל ותהיה "השורש" של כל האפליקציה.

מטרת המחלקה היא לאכוף את חוקי האפליקציה כגון:

1. בכל רגע נתון באפליקציה קיים רק משתמש אחד מחובר.
2. כל מחלקה בתוך האפליקציה יכולה לקבל ממחלקה זו (באופן סטטי כמובן) את המשתמש המחובר.
3. החזקת הגדרות האפליקציה.
4. המחלקה תנהול את כל הThreads באפליקציה ותדאג לסגור אותם בעת הצורך.

* אופן המימוש:

בכדי שהמחלקה תהיה Singleton עלינו לוודא כי ניתן לייצר מופע אחד בלבד שלה ובכל קריאה נוספת למחלקה זו נקבל את המופע הקיים (אם קיים).

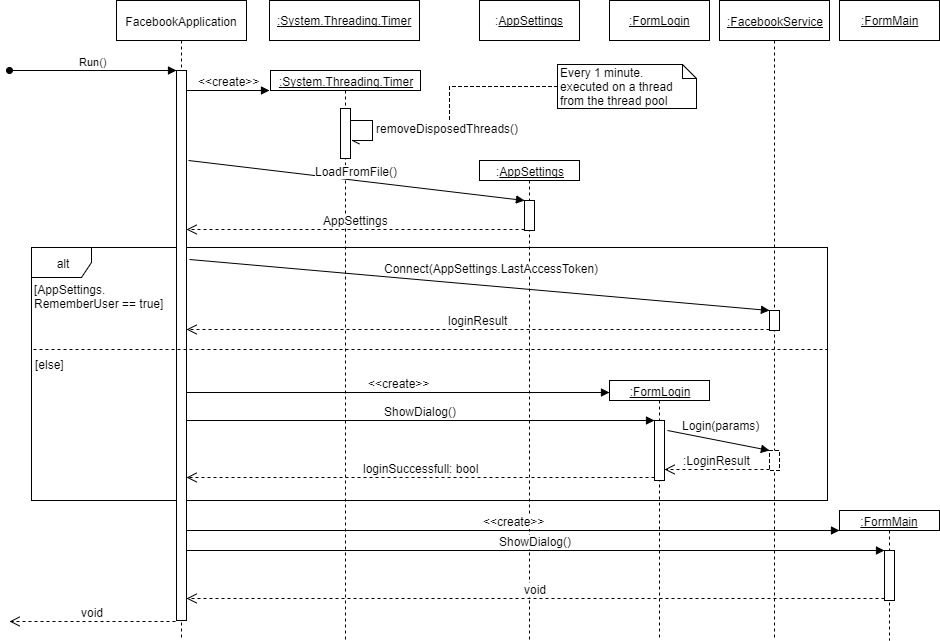
בחרנו לממש את הסינגלטון ע"י הגדרת המחלקה כמחלקה סטטית העונה על ההגדרה לעיל.

סיבות לבחירת אופן מימוש זה לפטרן singleton:

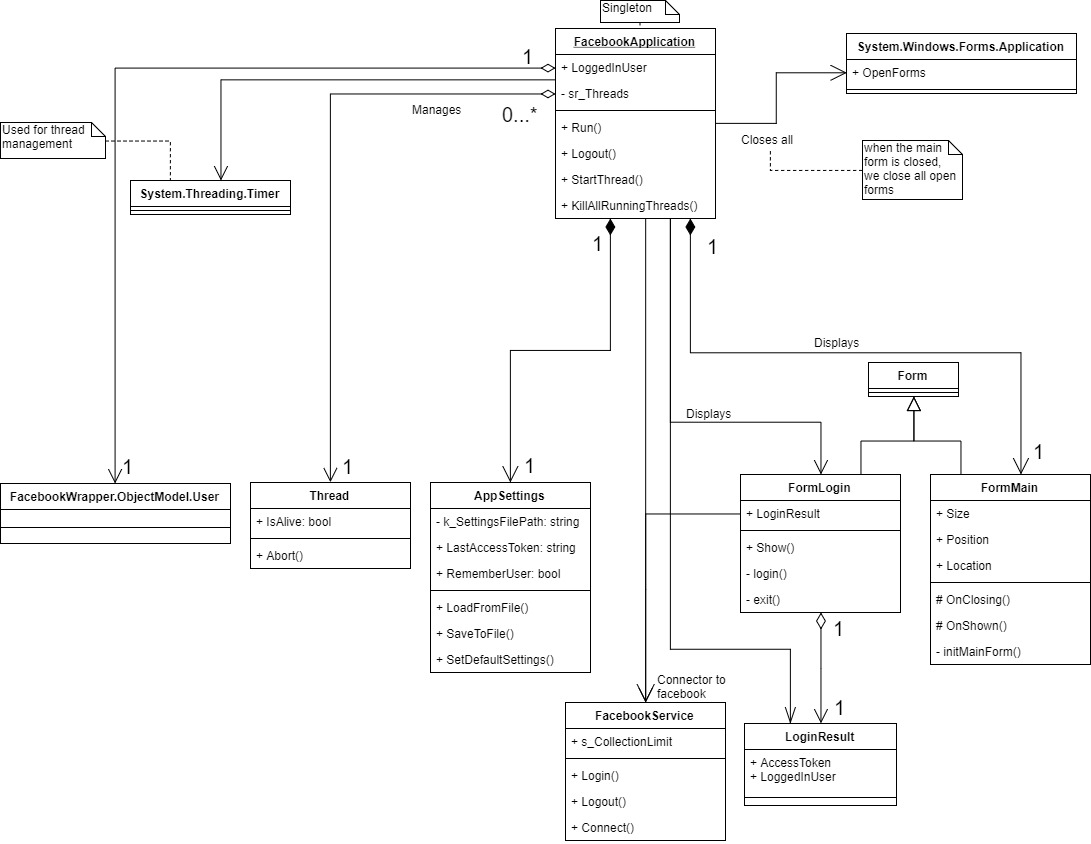
1. אין ברצוננו לרשת ממחלקה אחרת או לממש Interface.
2. מיד עם הפעלת האפליקציה (במתודה main) ישנה קריאה למתודה Run של מחלקה זו. המשך ריצת התוכנית מנוהלת מתוך המחלקה.   
   לכן, אין משמעות לlazy creation כיוון שאנחנו כן רוצים ליצור מופע של המחלקה עם כל הproperties שבה מיידית.
3. כאשר עובדים עם מחלקה סטטית המחלקה היא thread safe מבחינת היצירה שלה ללא צורך בשימוש ב double check lock או volatile.

מיקום בקוד: קובץ FacebookApplication.cs

* Sequence Diagram



* Class Diagram



### תבנית מס' 2 – Class Proxy

* סיבת הבחירה / שימוש בתבנית:

כחלק מפונקציונליות המערכת רצינו להוסיף לPictureBox הקיים (legacy component)

פונקציונאליות חדשה.  
המטרה – לייצר PictureBox שגדל כאשר העכבר מעליו ושיש לו תמונת "Loading" שתופיע עד שהתמונה נטענת מהשרת.

מכיוון שניתן לממש את הפונקציונליות הנ"ל ע"י מתן מימוש למתודות קיימות בpictureBox ללא חשיפת שינויים לוגיים לclient בחרנו לעשות שימוש בפטרן Class Proxy.

הסיבה לכך שבחרנו לעשות שימוש בהורשה ולא קומפוזיציה היא שאנחנו רוצים שהclient יוכל להשתמש בgrowingPictureBoxProxy כPictureBox ללא צורך לגשת לpropety של המחלקה.

* אופן המימוש:

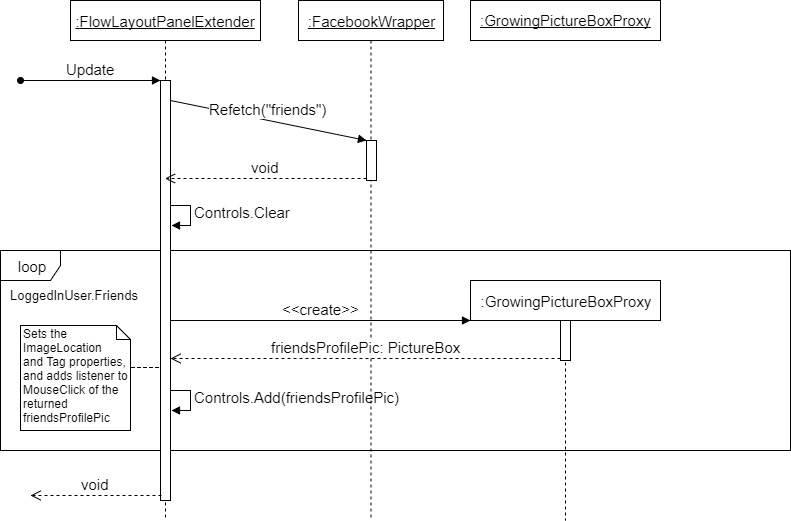
בכדי לממש את הproxy ירשנו מהמחלקה PictureBox והוספנו מימוש (override) למתודות OnMouseEnter/OnMouseLeave.

בנוסף, הוספנו מספר הגדרות לctor בכדי שיהיו ערכי ברירת מחדל כמו תמונת loading, גודל ברירת מחדל וsizeMode .

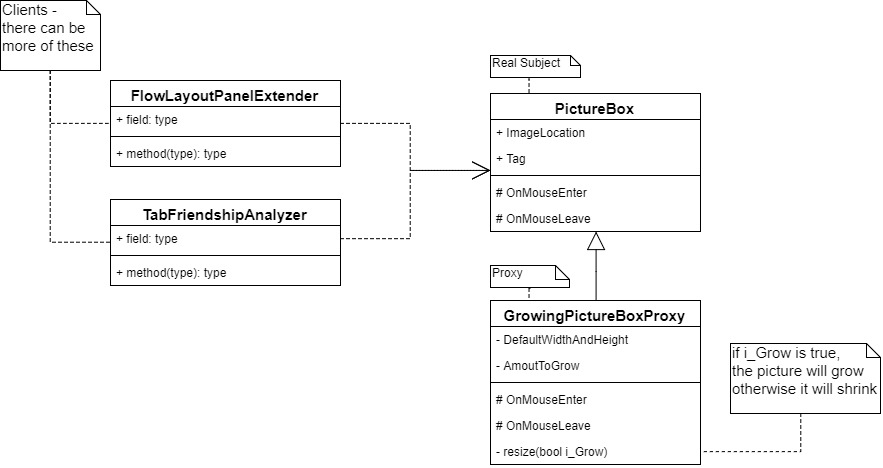
כל השינויים שבוצעו "שקופים למשתמש", ואינם משנים את האופן שבו עובדים עם הפקד.

מיקום בקוד: קובץ ControlsAndProxies. GrowingPictureBoxProxy.cs

* Sequence Diagram



* Class Diagram



### תבנית מס' 3 – Static Factory Class

* סיבת הבחירה / שימוש בתבנית:

לטובת מימוש הפיצ'ר הראשון (הצגת אוספי מידע בצורה טבלאית) רצינו שהUI יוכל לעבוד באופן פולימורפי עם אובייקטים שונים הנמצאים בחלק הלוגי של האפליקציה.

האובייקטים השונים הם מחלקות המחזיקות בקומפוזיציה אובייקט מסוג DataTable שמכיל את כל המידע הנמצא באוסף.

בכדי שניתן יהיה להוסיף מחלקות חדשות מהסוג המפורט לעיל גם בעתיד מבלי לשנות את הUI , בחרנו לממש Static Factory Class שיהיה זה שייצור אובייקטים קונקרטיים ויחזיר ל UI אובייקטים מסוג האבא האבסטרקטי.

במצב הנ"ל הUI לא צריך להכיר את המימושים הקונקרטיים והוא לא זה שיוצר אותם.

* אופן המימוש:

ראשית יצרנו מחלקה אבסטרקטית FacebookDataTable שכל אובייקט חדש שעושה שימוש בDataTable בכדי לייצג אוסף כלשהו יירש ממנה. מבחינת הUI העבודה היא מול מחלקה זו.

עבור כל תצוגה של אוסף מידע יצרנו מחלקה קונקרטית המממשת לפי הצורך את המתודות האבסטרקטיות של FacebookDataTable ומוסיפה במידה הצורך פונקציונליות לוגית נוספת.

בכדי ליצור אובייקטים חדשים יצרנו את הFactory כמחלקה סטטית, ולמחלקה זו יצרנו פונקציה בשם CreateTable , אשר מקבלת כפרמטר את הcontext על פיו המחלקה תדע איזה אובייקט קונקרטי ליצור ולהחזיר.   
את הcontext בחרנו לממש בעזרת enum המייצג את כל המחלקות שמומשו.

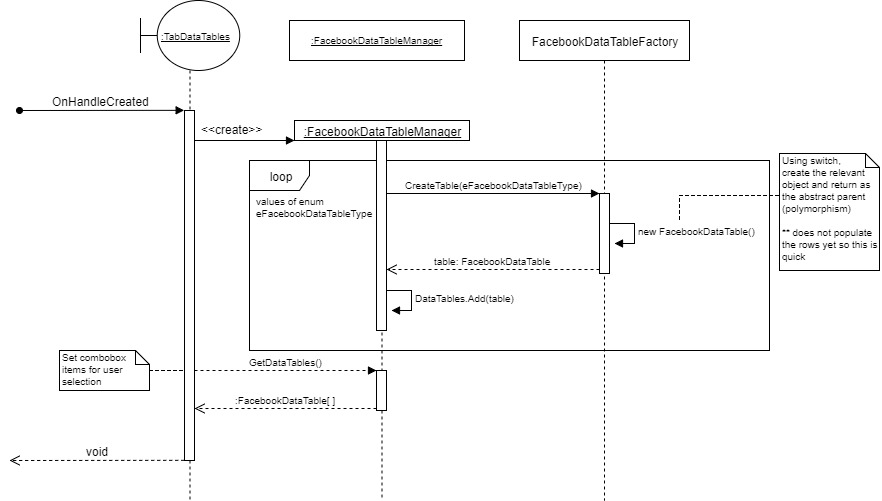
בסופו של דבר הUI (שהוא הclient במקרה זה) עובד מול הFactory ומול המחלקה האבסטרקטית בלבד כאשר הוא מכיר את הenum בכדי לשלוח את הcontext המתאים.

איך ניתן להוסיף מימוש חדש של FacebookDataTable בעתיד?

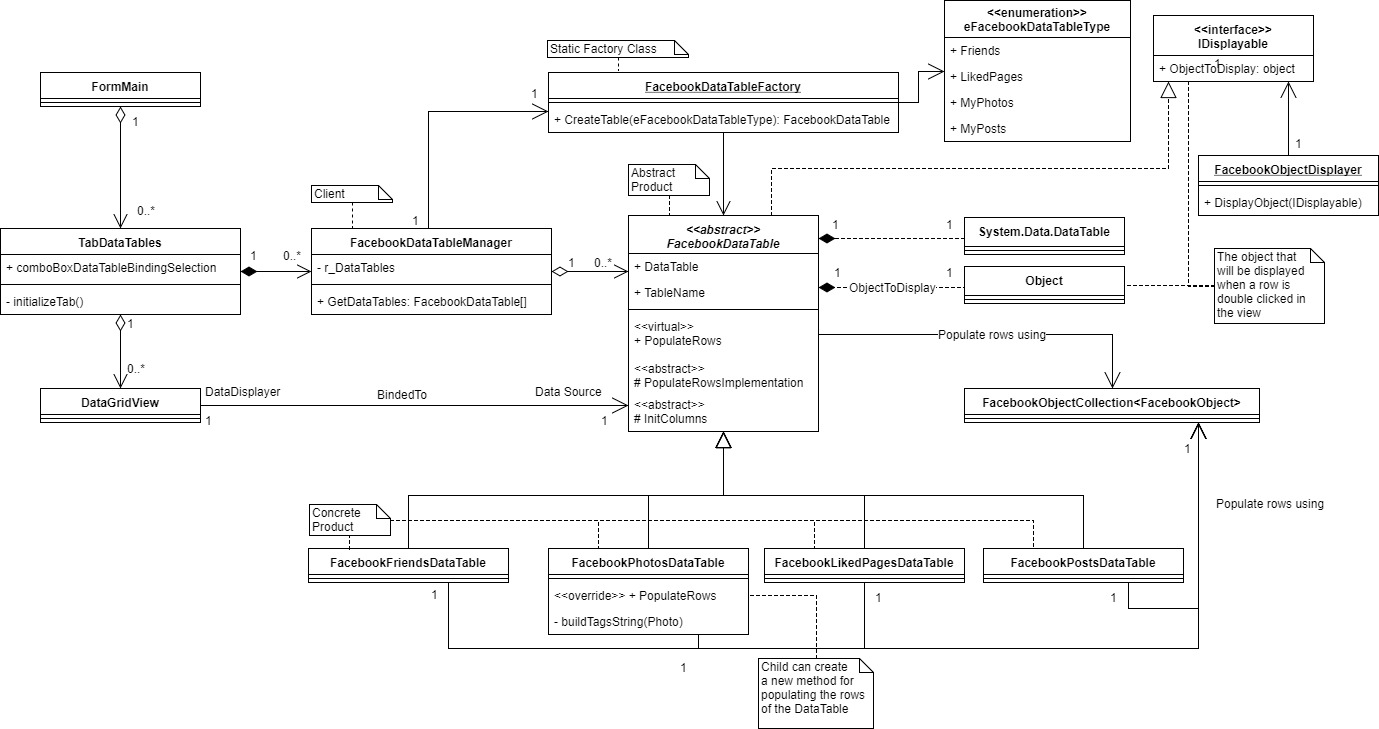
1. יוצרים מחלקה היורשת מFacebookDataTable
2. מממשים את המתודות האבסטרקטיות בכדי להגדיר את הDataTable המוחזר בקומפוזיציה וכן אופן מילוי שורות הDataTable כאשר מקבלים אוסף לתצוגה.
3. מוסיפים לenum מזהה למחלקה החדשה
4. מוסיפים לFactory עוד case שבו מחזירים את המחלקה החדשה שנוצרה

מיקום בקוד: התיקייה DataTables, כאשר ה Static Factory Class נמצא ב FacebookDataTableFactory.cs

* Sequence Diagram



* Class Diagram



### שימוש בעבודה אסינכרונית:

כחלק מהעבודה עם הsingleton ובכדי לאפשר ניהול וסגירה של כל הthread-ים שרצים בעת סגירת החלון הראשי, יצרנו מתודה בFacebookAppliction ("השורש") שדואגת ליצור ולהריץ thread ים חדשים.

דוגמאות למקומות בקוד בהם עשינו שימוש במתודה זו (וכוצאה מכך עבדנו עם thread נפרד)

1. בפיצ'ר של טבלאות המידע, כאשר אוסף המידע נטען לשורות הטבלא, הדבר נעשה באופן אסינכרוני כדי לא לפגוע בחוויות המשתמש בזמן טעינת השורות (פעולה שעלולה לקחת זמן רב)  
   מבחינת המשתמש זה יקרה כאשר הוא ילחץ על כפתור Fetch Data תחת data tables tab. הוספנו מתחת לטבלא את כמות השורות שנטענו מתוך סך השורות שיש לטעון כדי שהמשתמש יהיה מודע לסטטוס.  
   מיקום בקוד: FacebookDataTable.cs שורה 56
2. כאשר המשתמש לוחץ לחיצה כפולה על אובייקט של תמונה (יכול להיות דרך שורה בטבלא של תמונות או רכיב תמונה תחת friendship analyzer tab ) יוצג חלון עם מידע מפורט על התמונה, בחלון זה טעינת רשימת כל הלייקים והתגובות של המשתמשים נעשית באופן אסינכרוני.  
   גם במקרה זה בחרנו לעבוד בצורה אסינכרונית בכדי לא לפגוע בחווית המשתמש, בזמן שהאפליקציה מביאה נתונים מהשרת של פייסבוק.  
   מיקום בקוד: FormPhotoDetails.cs שורות 34-35
3. כאשר סוגרים את החלון הראשי של האפליקציה, דאגנו להריץ את המתודה הסטטית FacebookApplication.KillAllRunningThreads() שדואגת לכך שלא יהיו thread ים שמונעים מהאפליקציה להיסגר.

### עבודה עם DataBinding:

בכדי לשייך את כל המידע בטופס או פקד מסוים לאובייקט שנמשך מהפייסבוק עשינו שימוש בData Binding.

דוגמאות לחלק מהמקומות בקוד בהם עשינו שימוש בData Binding:

1. הטופס FormPostDetails שמציג מידע מפורט אודות פוסט מסוים שנבחר.
2. הטופס FormFriendDetails שמציג מידע מפורט אודות חבר מסוים שנבחר.
3. הפקד listBoxLikedPages שמציג את רשימת העמוד שהמשתמש עשה להם לייק.   
   פקד זה נמצא ב Forms.Tabs.TabAboutMe.cs