### 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 – Visitor

* סיבת הבחירה / שימוש בתבנית:

לטובת שדרוג חווית המשתמש בכל הקשור לתצוגה טבלאית של נתוני המשתמש, כאשר המשתמש לוחץ לחיצה כפולה על שורה בטבלה בחרנו להציג טופס עם יותר פרטים אודות אותו אובייקט.

כל אובייקט מיוצג בעזרת הצגת טופס על המסך.  
בכדי לשמור על הפרדה בין לוגיקה לUI רצינו שהאובייקטים הלוגיים, שהם מימושים שונים של המחלקה האבסטרקטית FacebookDataTable, לא יכירו את האופן בו הטופס מוצג אלא יחזיקו רכיב (אותו ניתן להחליף) שעושה זאת.

הרכיב הנ"ל הוא למעשה הvisitor.  
במימוש הנוכחי הוא לא עושה שינויים אצל האובייקט שמחזיק אותו, כפי שלמדנו שvisitor אמור לעשות, אך אם בעתיד נרצה לאפשר למשתמש לעדכן פרטים בשרת הפייסבוק באמצעותו נוכל לעשות זאת בקלות ע"י החלפת המימוש של הvisitor בלבד.

* אופן המימוש:

כל טבלה אותה ניתן להציג בTabDataTables למעשה ממומשת ע"י מחלקה כלשהי שיורשת מהמחלקה האבסטרקטית FacebookDataTable.   
בכדי שהvisitor ידע לעבוד עם מחלקות אלה מבלי להכיר כל אחת באופן קונקרטי (הימנעות מ tight coupling), מימשנו ממשק בשם IDisplayableObjectHolder, אשר מגדיר שלכל הממשים שלו יש property בשם ObjectToDisplay מסוג object.

הvisitor במימוש שלנו הוא FacebookObjectDisplayer, תפקידו לייצר ולהציג את הטופס המתאים בעזרת Factory Method שמייצר את הטופס המתאים ע"פ הcontext שנשלח אליו.   
כאמור, רכיב זה אינו מכיר את המחלקה FacebookDataTable ויורשיה, אלא את הממשק IDisplayableObjectHolder בלבד.

מכיוון שהמחלקה האבסטרקטית FacebookDataTable מממשת את הממשק הנ"ל ומחזיקה אובייקט מסוג FacebookObjectDisplayer, כך גם כל היורשים שלה.

לבסוף מימשנו מתודה בFacbookDataTable בשם DisplaySelectedObject, שהיא המתודה שהclient מכיר. כל תפקיד המתודה הנ"ל הוא להפעיל את המתודה של הvisitor שמייצרת ומציגה את הטופס.

מיקום בקוד:

1. מחלקת הvisitor :   
   הקובץ FacebookObjectDisplayer.cs
2. הממשק שהvisitor מכיר:

הקובץ IDisplayableObjectHolder.cs

1. המחלקות אשר מממשות את הממשק, מחזיקות ועושות שימוש בvisitor:

המחלקה DataTables.FacebookDataTable.cs וכל יורשיה הנמצאים באותה התיקיה

* Sequence Diagram



* Class Diagram



### תבנית מס' 2 – Strategy Method

* סיבת הבחירה / שימוש בתבנית:

במספר מקומות שונים באפליקציה נתקלנו בצורך לסנן תמונות מסוימות מתוך אוסף של תמונות כלשהו, כאשר שיטת הסינון שונה בכל מקום.

בכדי להימנע משכפול קוד, ובכדי לאפשר הוספה של אופציות סינון נוספות בעתיד בקלות, החלטנו לעשות שימוש בstrategy method.

בחירה זו תאפשר לנו לממש פעם אחת מתודה, אשר מקבלת אוסף תמונות כלשהו ואת אסטרטגיית המיון (בצורה של Func<Photo, bool> ) ומחזירה אוסף תמונות "מסונן".

* אופן המימוש:

בחרנו לייצר את המחלקה FacebookObjectCollectionUtils אשר נממש בה extension methods עבור אובייקטים שונים מסוג FacebookObjectCollection<T>

שני המימושים הרלוונטיים לתבנית הנ"ל הם:

1. FilterAlbumColection
2. FilterPhotoColection

שתי המתודות הנ"ל הן Extension Methods לאוספים של אלבומים ותמונות (בהתאמה) והן מקבלות בתור פרמטר נוסף את אסטרטגיית המיון (ה-Strategy Method)

השימוש בStrategy Method באופן הנ"ל חוסך את הצורך לממש ממשק המגדיר את אופן המיון ומאפשר לשלוח את אסטרטגיית המיון כlambda expression כאשר האסטרטגיה פשוטה.

מיקום בקוד:

FacebookObjectCollectionUtils.cs – המחלקה בה ממומשות המתודות שמקבלות כפרמטר את הStrategy Method

FriendshipAnalyzer.cs – המחלקה אשר שולחת את הStrategy Method כפרמטר למתודות הנ"ל.

מיקום בתוך המחלקה:

1. המתודה GetPhotosTaggedTogether – שורה 61
2. המתודה GetPhotosFromAlbumsUserIsTaggedIn – שורה 68

* Sequence Diagram



* Class Diagram





### תבנית מס' 3 – Observer

* סיבת הבחירה / שימוש בתבנית:

כחלק מאופן המימוש של הפיצ'ר, בו אנו מציגים מידע מהפייסבוק בצורה טבלאית, רצינו שטעינת השורות לטבלה תיעשה באופן אסינכרוני.

כתוצאה מכך עלה הצורך להודיע למשתמש ברגע שהטבלה מסיימת להיטען.

לכן, בחרנו לעשות שימוש בתבנית observer.

* אופן המימוש:

יצרנו ממשק, IRowsPopulatedObserver, המגדיר את המתודה AllRowsUpdated, אותה כל מחלקה המעוניינת "להאזין" צריכה לממש.

הרכיב הלוגי, FacebookDataTable, האחראי על ביצוע פעולת עדכון השורות מחזיק רשימה של מאזינים וכן מממש שתי מתודות המאפשרות להוסיף או להסיר מאזין מהרשימה.

כאשר פעולות טעינת השורות לטבלה מסתיימת, מופעלת המתודה notifyAllRowsPopulatedObservers אשר מודיעה לכל מאזין ברשימת המאזינים להריץ אתAllRowsUpdated שלהם.

\*\* כמובן שניתן היה לממש את התכנית בעזרת Action, וכך גם עשינו במקומות אחרים בקוד, אך בחרנו לעשות זאת כדי לתרגל עבודה עם Interface (וגם כדי לפשט את הדיאגרמות 😊)

מיקום בקוד:

הפטרן ממומש בקבצים הבאים:

1. DataTables.FacebookDataTable.cs – שורה 66 ומטה
2. DataTables.IRowsPopulatedObserver.cs - הממשק
3. Forms.Tabs.TabDataTables.cs – observer אשר מממש את הממשק ונרשם כ"מאזין"

* Sequence Diagram



* Class Diagram

