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

* [Most liked picture in a specific album- זהו פיצ'ר אשר מראה למשתמש מהי התמונה בעלת הכי הרבה לייקים לפי אלבום ספציפי שהוא בוחר, ובנוסף מראה למשתמש מתי נוצרה התמונה.]
* [- Search information of event זהו פיצ'ר אשר מאפשר למשתמש ע"י בחירת event ולחיצה על הכפתור המתאים, חיפוש מהיר בgoogle אודות הevent, בנוסף מוצגים למשתמש תאריך ההתחלה, תאריך הסיום, ומידע אודות הevent.]
* הוספנו פיצ'ר נוסף- LikedPages- אשר מציג למשתמש ב**Two way data binding** מידע אודות הדפים אותם הוא אוהב.

### תבנית מס' 1 – [Strategy]

* סיבת הבחירה / שימוש בתבנית:

הרצון לאפשר למשתמש להחליט בזמן ריצה על אופן הצגת הסטטוסים (לפי שם, לפי תאריך פרסום הסטטוס, לפי מספר הלייקים וכו').

בנוסף, רצינו לאפשר להוסיף בעתיד בקלות שיטות מיון נוספות כמו למשל לפי שם בסדר יורד, או לפי תאריך עדכון הסטטוס וכו'.

בנוסף, אופן מימוש ה-pattern הוביל אותנו לשימוש ב-pattern נוסף- Factory Method, יצרנו מחלקה נוספת – StatusSorterFactory, מחלקה זו מייצרת עבורנו שיטת מיון קונקרטית.

* אופן המימוש:

יצרנו מחלקה אבסטרקטית **StatusSorter(ה-Strategy)** ,אשר מחזיקה כ-protected ב-listbox, ובנוסף יש לה מתודה אבסטרקטית Sort.

**ממחלקה זו יורשות 3 מחלקות נוספות- StatusCreatedSorter, StatusNameSorter, StatusLikedBySorter (שלושת ה-Concrete Strategies).**

שלושת המחלקות מממשות באופן קונקרטי את המתודה Sort, ואם בעתיד נרצה להוסיף שיטת מיון חדשה, פשוט ניצור מחלקה חדשה שתירש מStatusSorter, וכמובן נעדכן את הFactory שלנו שיוצר את האובייקט הקונקרטי.

בנוסף, על מנת להתאים את ה-pattern שבחרנו לצרכינו המלאים השתמשנו בעוד pattern נוסף:

נתקלנו בבעיה- על מנת לאפשר למשתמש לבחור את שיטת המיון השתמשנו בListBox, אשר אל ה-ListBox הכנסנו אובייקטים מסוג Enum.

הEnum שלנו (eSortTypes אשר נמצא במחלקה האבסטרקטית StatusSorter) נועד להבדיל בין שיטות המיון וההצגה השונות.

למעשה הבעיה היא שה-default ToString של כל מחלקה מסוג Enum למעשה

מחזירה את השם של הEnum, כך שלדוגמא שיטת מיון מסוג CreatedTime הוצגה

למשתמש בדיוק כך במקום Created Time, ובנוסף לא ניתן לדרוס מתודה זו כשמשתמשים ב- Enum.

ע"מ לפתור את הבעיה יצרנו מחלקה חדשה – EnumWrapper, אשר מחזיקה

כ-Properties גם Enum וגם Name מסוג string.

כך למעשה פתרנו את הבעיה שנוצרה בעזרת Adapter, ולמעשה הוספנו לListBox משתנה מסוג EnumWrapper במקום משתנה מסוג Enum.

לבסוף, ע"מ לשמור על הקוד שלנו מיינטנבילי, השתמשנו בטכנולוגית c#- יצרנו מחלקה אשר נקראת Description ויורשת מ-Attribute, כך שמעל כל משתנה

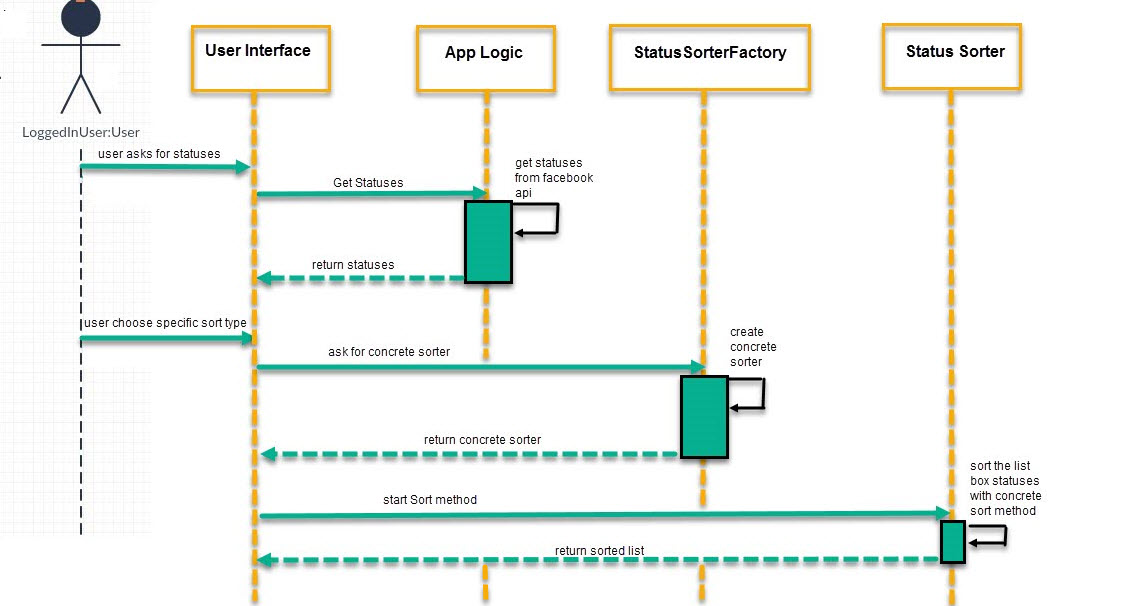
ב-Enum שלנו הוספנו Attribute שמתאר את שמו כפי שנרצה שיוצג למשתמש.

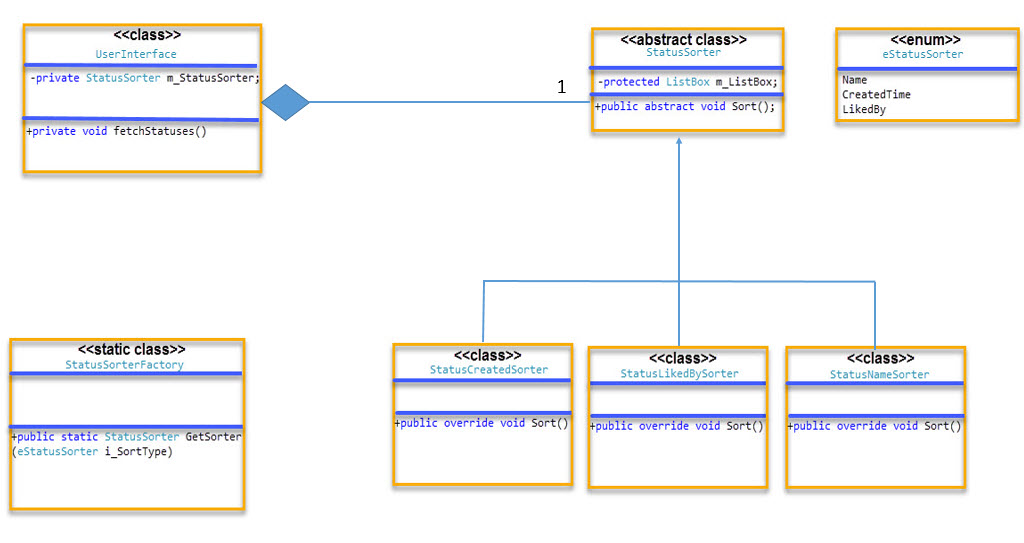
בנוסף יצרנו מחלקה נוספת- EnumExtensions אשר בה מתודה סטטית אחת-

GetName אשר יודעת בעזרת Reflection לגשת לAttribute שלנו ולהגיע אל

ה- string שאנו רוצים שיוצג למשתמש.

בנוסף מתודה זו מומשה בעזרת טכנולוגית Extension Method כך שניתן להפעילה על אובייקט מסוג Enum וכך לשמור על מבנה קוד Object Oriented.

* Sequence Diagram
* Class Diagram



### תבנית מס' 2 – [Observer]

* סיבת הבחירה / שימוש בתבנית:

האפליקציה שלנו בנויה כך שיש הכלה בין הUserInterface ו- AppLogic.

מטרת השימוש בobserver היא הרצון לעדכן את תמונת הפרופיל באפליקציה ע"י שימוש בlisteners – במקרה הזה תמונת הפרופיל הראשית בעמוד הראשון של האפליקציה.

השימוש מתבצע כך שברגע שמחלקת הלוגיקה מקבלת את הכתובת לטעינת תמונת הפרופיל, היא מייד "מודיעה" על כך לכל המעוניין- במקרה הזה לUserInterface לחלק ספציפי של טעינת תמונת הפרופיל.

בחרנו ב-pattern זה בעיקר בשביל היכולת בעתיד בקלות להוסיף listeners נוספים אשר מחלקת הלוגיקה תוכל לעדכן אותם שהיא השיגה את כתובת תמונת הפרופיל וכך תתבצע טעינת תמונת פרופיל למקומות נוספים באפליקציה בעת הצורך.

* אופן המימוש:

הוספנו למחלקת ה-**AppLogic(ה-Notifier)** event מסוג Action אשר מחזיק כתובות למתודות שמקבלות משתנה מסוג string כפרמטר – כתובת הטעינה של תמונת הפרופיל, השם של ה-event הוא ReportOnLoadingProfilePicture.

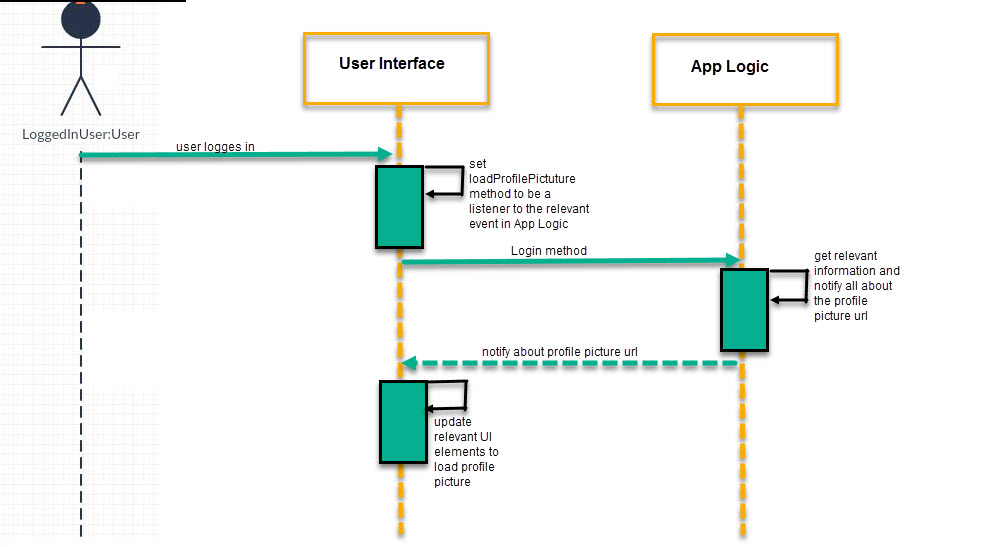
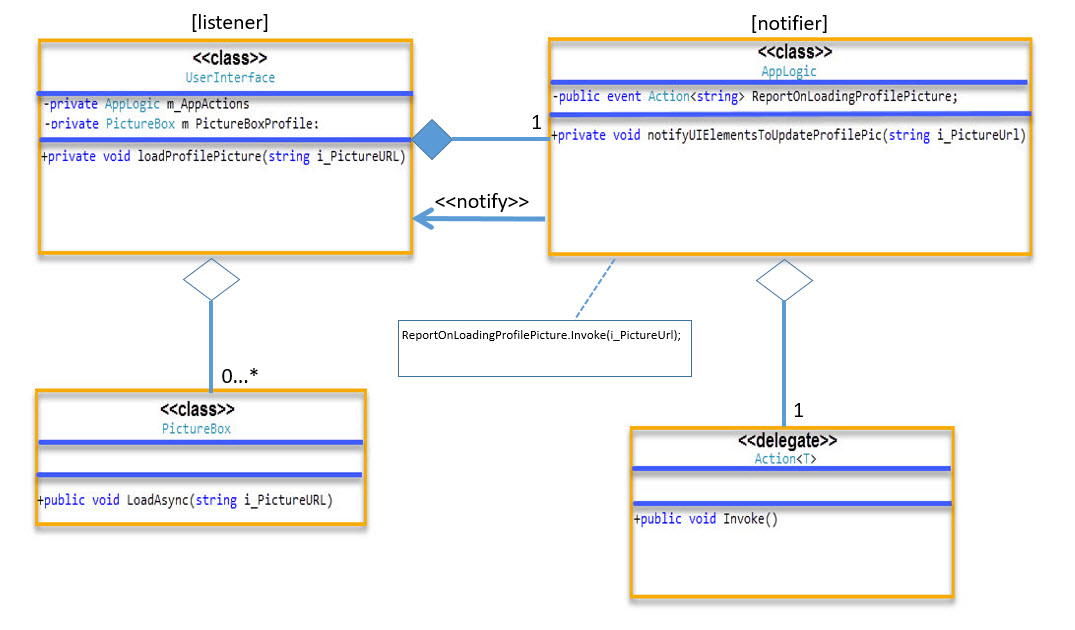
בנוסף יצרנו מתודה חדשה במחלקת ה-AppLogic אשר תפקידה הוא ליידע את כל המעוניין ברגע שמחלקת הלוגיקה השיגה את כתובת הלינק לטעינת תמונת הפרופיל- notifyUIElementsToUpdateProfilePic.

בנוסף, במחלקת ה-**UserInterface(ה-Listener),** יצרנו מתודה- loadProfilePicture.

המתודה הזאת נרשמת להאזין לevent של ה-AppLogic, וברגע שהיא מיודעת על קבלת הלינק לתמונת הפרופיל היא טוענת את תמונת הפרופיל לאפליקציה.

אם בעתיד נרצה לטעון את תמונת הפרופיל לחלקים נוספים באפליקציה- פשוט נירשם כ"מאזינים" בעזרת מתודות חדשות בחלקים נוספים באפליקציה.

את ה-pattern מימשנו בעזרת שימוש בטכנולוגיות c# שנועדו בדיוק למקרים הללו- מימשנו delegates ספציפים אשר רלוונטים לצרכינו- במקרה הזה delegate מסוג Action.

* Sequence Diagram
* Class Diagram

### תבנית מס' 3 – [Iterator]

* סיבת הבחירה / שימוש בתבנית:

השתמשנו ב-pattern זה משום שרצינו להסתיר מפני המשתמש את מבנה הנתונים המשמש להחזקת האלבומים של ה-User.

בנוסף רצינו לממש זאת בצורה שנוכל לגשת למבנה הנתונים ולData שהוא מחזיק ברציפות ב-O(n) מבלי לשנות את הקוד בUserInterface, כך שגם אם בעתיד נשנה את מבנה הנתונים, נוכל בעצמנו לשנות את לוגיקת המתודת ה-MoveNext של

ה-Enumerator, במחלקה עצמה המחזיקה את מבנה הנתונים ולא לשנות קוד בשכבת ממשק המשתמש, ובכך לשמור על דרישותינו הנ"ל.

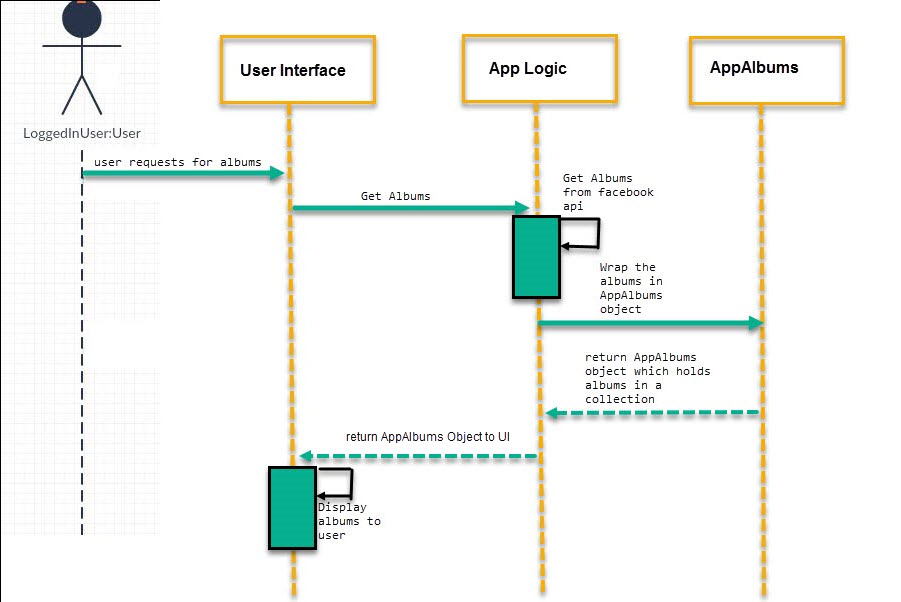
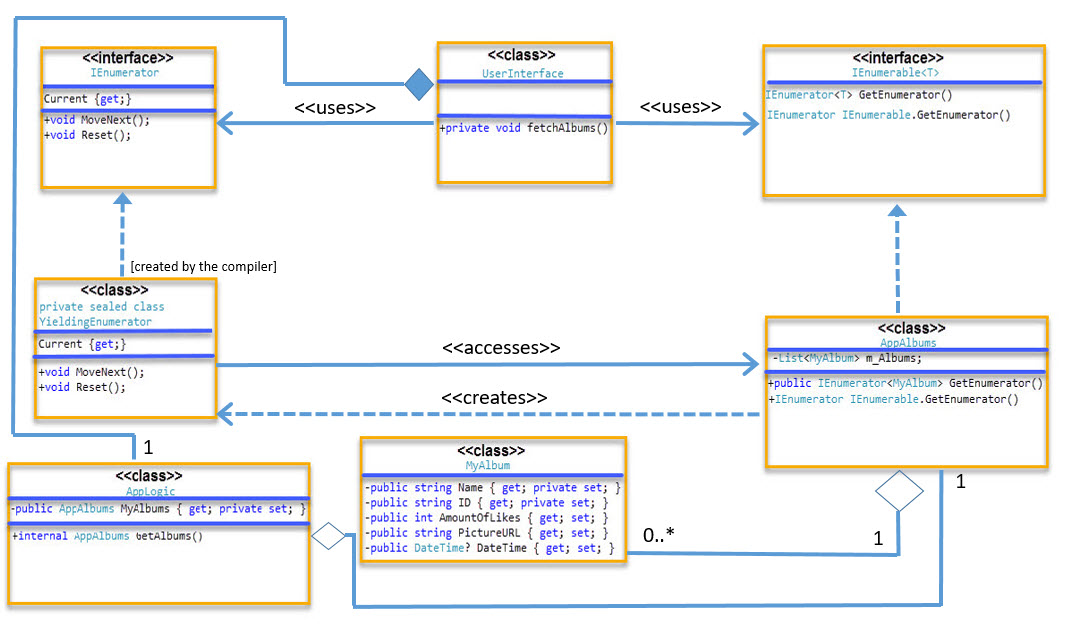
* אופן המימוש:

יצרנו מחלקה חדשה- **AppAlbums(ה-Concrete Aggregate),** אשר מחזיקה מבנה נתונים שמחזיק Data מסוג MyAlbum, במקרה זה מבנה הנתונים עצמו הוא מסוג List, אך כאמור אם בעתיד נרצה נוכל לשנות אותו לכל מבנה נתונים אחר שנחפוץ להשתמש בו.

מחלקה זו ממשת את ה-interface שבעזרתו אנו מממשים את היכולת לבצע איטרציות- **IEnumerator(ה-Aggregate),** והשתמשנו בטכנולוגיות c# על מנת לממש את המתודה GetEnumerator- **השתמשנו בפיצ'ר yield return אשר מסמנת לקומפיילר לבנות "מאחורי הקלעים" את מחלקת ה-iterator(ה-Concrete Iterator)**, אשר מממש את **IEnumerator(ה-Iterator).**

בנוסף- על מנת לשמור על קריאות הקוד כך שנוכל בשורה אחת בלבד להוסיף נתונים אל ה-ListBox, מימשנו את המתודה ToArray בעזרת Method Extension, אשר יודעת להפוך אובייקט מסוג IEnumarable למערך.

מימשנו מתודה זו במחלקה הנקראת CollectionExtensions, והמתודה מומשה על מנת שנוכל להשתמש במתודה AddRange של ה-ListBox.

* Sequence Diagram
* Class Diagram