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

* **ברכות יום הולדת מותאמות אישית לפי הגיל והמין של המשתמש** – הפיצ'ר עובר על רשימת החברים של המתשמש ובונה רשימה של החברים החוגגים יום הולדת, לאחר מכן עובר על הרשימה שנוצרה ובהתאם למין של המשתמש בודק האם קיימת ברכה מיוחדת לגילו (לדוגמה – עבור נקבה בת 12 קיימת ברכת בת מצווה) ומפרסם אותה בקיר שלו, במידה ולא קיימת ברכה מיוחדת מפרסם ברכת יום הולדת סטנדרטית.
* **סינון התמונות בעלות מספר התגובות הגבוהה ביותר עבור קהל יעד ספציפי** - הפיצ'ר עובר על רשימת התמונות ויוצר רשימת תמונות ממויינת על פי מספר התגובות לתמונה שהתקבלו ממשתמשים העומדים בדרישות הסינון של המשתמש (גיל, מין וסטטוס זוגי) ומחזיר את התמונות הראשונות (בעלות מספר התגובות הגבוה ביותר שהתקבלו מקהל היעד שהמתמש הגדיר), כמות התמונות שתוחזר גם היא מוגדרת על פי דרישת המשתמש (1, 3 או 5 תמונות).

### תבנית מס' 1 – Iterator

* סיבת הבחירה / שימוש בתבנית:

השתמשנו בתבנית Iterator מכיוון שיש לנו במחלקה של הלוגיקה (Wrapper) אוספים שונים של פריטים (לעיתים מפולטרים) עבורם נרצה לאפשר לקליינט לסרוק את הפריטים מבלי להכיר את מבנה הנתונים של האוסף כך שאם בעתיד נרצה להחליף את המבנה נוכל לעשות זאת.

בחלק מהמקומות בהם נעשה שימוש בתבנית יצרנו לא רק פילטר איטרטור אלא filter projection Itertator (כלומר החזרנו רק חלק מהשדות של האובייקט אותו סרקנו מתוך אובייקטים שעמדו בתנאי מסויים).

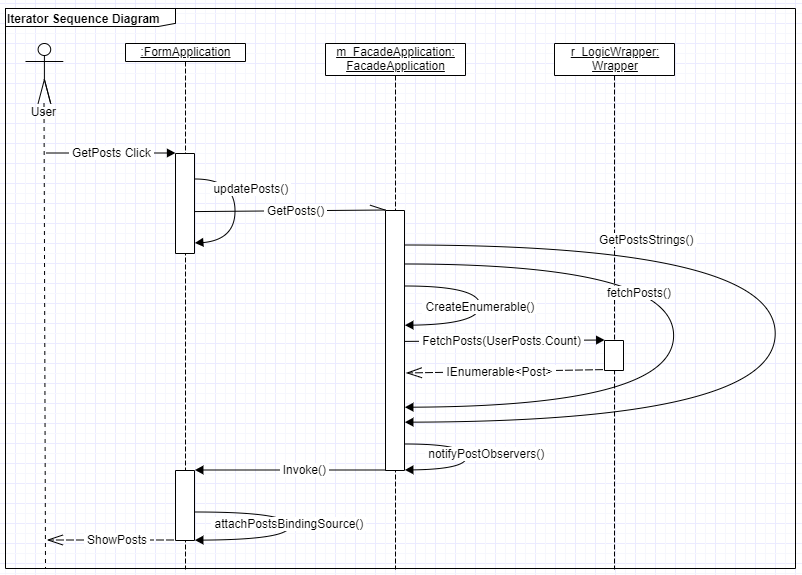
* אופן המימוש:

במחלקה שאחראית על הלוגיקה (Wrapper)במקום להחזיר את מבני הנתונים השונים בכל אחת מהמתודות המבקשות אותם (FetchPosts, FetchEvents, Etc.) ובכך לחשוף את מבני הנתונים למי שרוצה לסרוק ולעבור עליהם, החזרנו IEnumerable שבו יש רק את המטודה GetEnumerator.

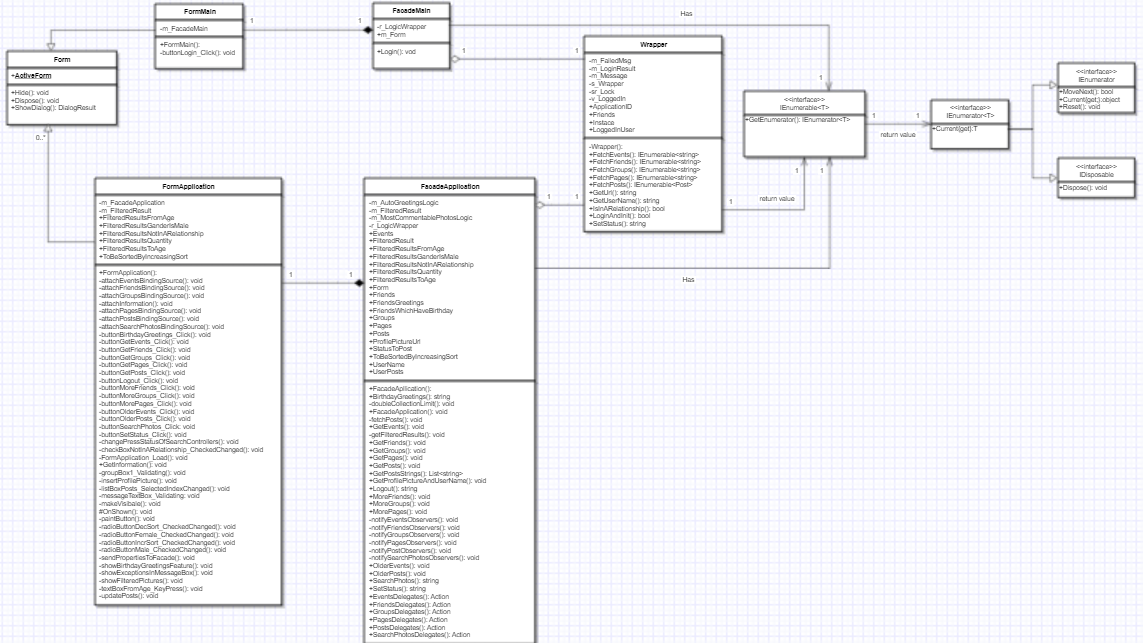
המשתמש מקבל את IEnumerable ומפעיל את המטודה GetEnumerator שפועלת מאחורי הקלעים כאשר מופעלת לולאת foreach על האוסף המוחזר והוא מקבל את הטיפוס IEnumerator החושף למשתמש 5 מטודות : Current, MoveNext, Reset, Dispose, Current.

כאשר עברנו על הקולקציות השונות ב Wrapper יצרנו IEnumerable שמהווה filter projection Itertator באמצעות הפיצ'ר yield return שבשפה.

* Sequence Diagram



* Class Diagram



|  |  |
| --- | --- |
| Our Implementation | Pattern |
| * IEnumerable<T> * IEnumerable * IEnumerator * IEnumerator<T> * IDesposeable | Iterator |
| * FormApplication * FormMain | Client |
| Wrapper | Aggregate |
| * FetchPosts * FetchEvents * FetchPages * FetchGroups * FetchFriends | ConcreteIterator |
| * FacadeApplication * FacadeMain | ConcreteAggregate |

### תבנית מס' 2 – Observer

* סיבת הבחירה / שימוש בתבנית:

רצינו להגדיר תלות של אחד לרבים בין אובייקטים הנמצאים במחלקה FormApplication ומציגים לנו את המידע באמצעות DataBinding שמחבר בינם לבין אובייקטים ב FacadeApplication המחזיקים את המידע הרצוי, כך שכאשר אובייקט אחד (הנמצא FacadeApplication) משנה את מצבו, כל התלויים בו (הנמצאים ב FormApplication) מיודעים על השינוי ומתעדכנים אוטומטית.

השימוש בתבנית מסייע בשינוי ושימוש חוזר.

שינוי של אובייקט אחד גורר שינוי באובייקט אחר, ולא ידוע מראש כמה אובייקטים יש לשנות.

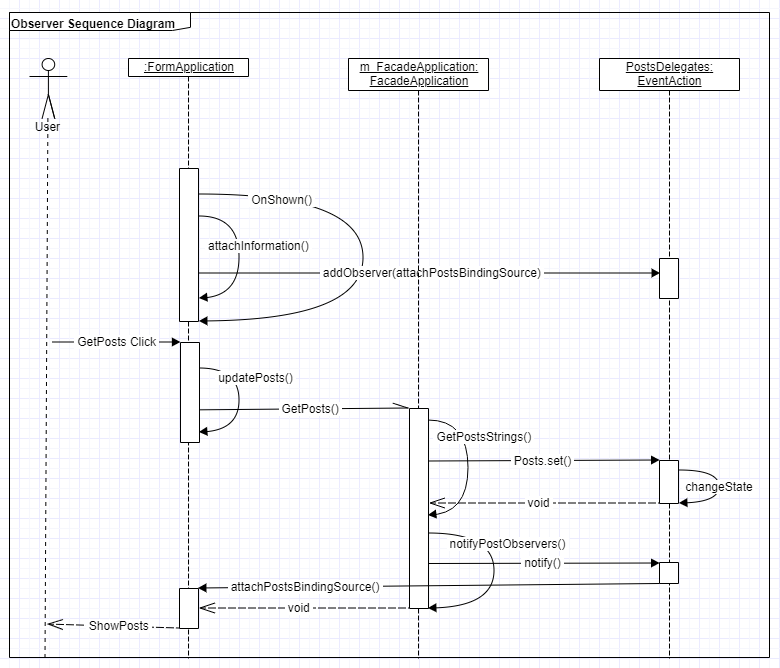
אובייקט צריך להודיע לאובייקטים אחרים מבלי להניח הנחות לגבי מה הם אותם אובייקטים, כלומר רוצים למנוע צימוד חזק.

* אופן המימוש:

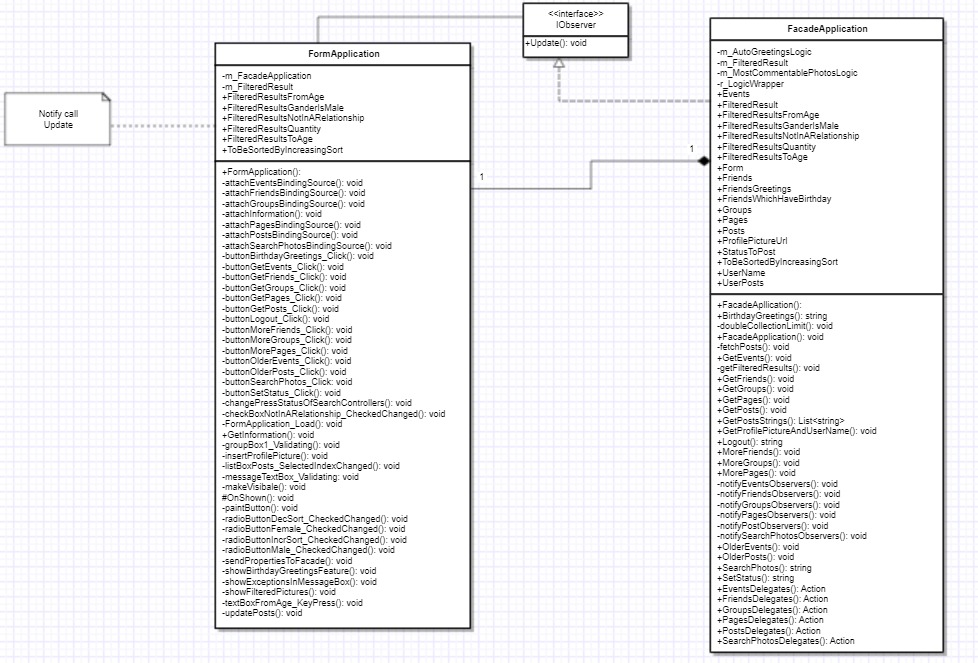
במחלקה FaçadeApplication יצרנו Properties מסוג Event Action המהווים Delegates לכל אחד מהאירועים אשר נרצה לעקוב אחריהם ועבור כל אחד מהם יצרנו מתודה האחראית על עדכון הצופים שלו (notifyPostObservers, notifyEventsObservers, Etc.) ובעצם מפעילה את המתודה שכל צופה התאים לאירוע.

במחלקה FormApplication כאשר הטופס מוצג ישנה קריאה למתודה attachInformation אשר בין היתר מבצעת את החיבור בין הצופים (האובייקטים המציגים את המידע בטופס) לבין האובייקטים הנצפים (האובייקטים המחזיקים את המידע ב FaçadeApplication) כאשר כל צופה מחבר את המתודה האחראית על עדכון המידע בטופס כמתודה שיש להפעיל כאשר יש שינוי באובייקט הנצפה (attachPostsBindingSource, attachEventsBindingSource, Etc.)

* Sequence Diagram



* Class Diagram



|  |  |
| --- | --- |
| Our Implementation | Pattern |
| FacadeApplication | Observer |
| FormApplication | Subject |
| event Action | IObserver |

### תבנית מס' 3 – Strategy

* סיבת הבחירה / שימוש בתבנית:

רצינו להגדיר משפחה של אלגוריתמי מיון (Increasing / Decreasing Sort), ולכמס כל אחד מהם כדי לאפשר להחליף את השימוש בהם. ניתן לשנות את האלגוריתם באופן עצמאי בלי תלות במשתמשים במחלקה.

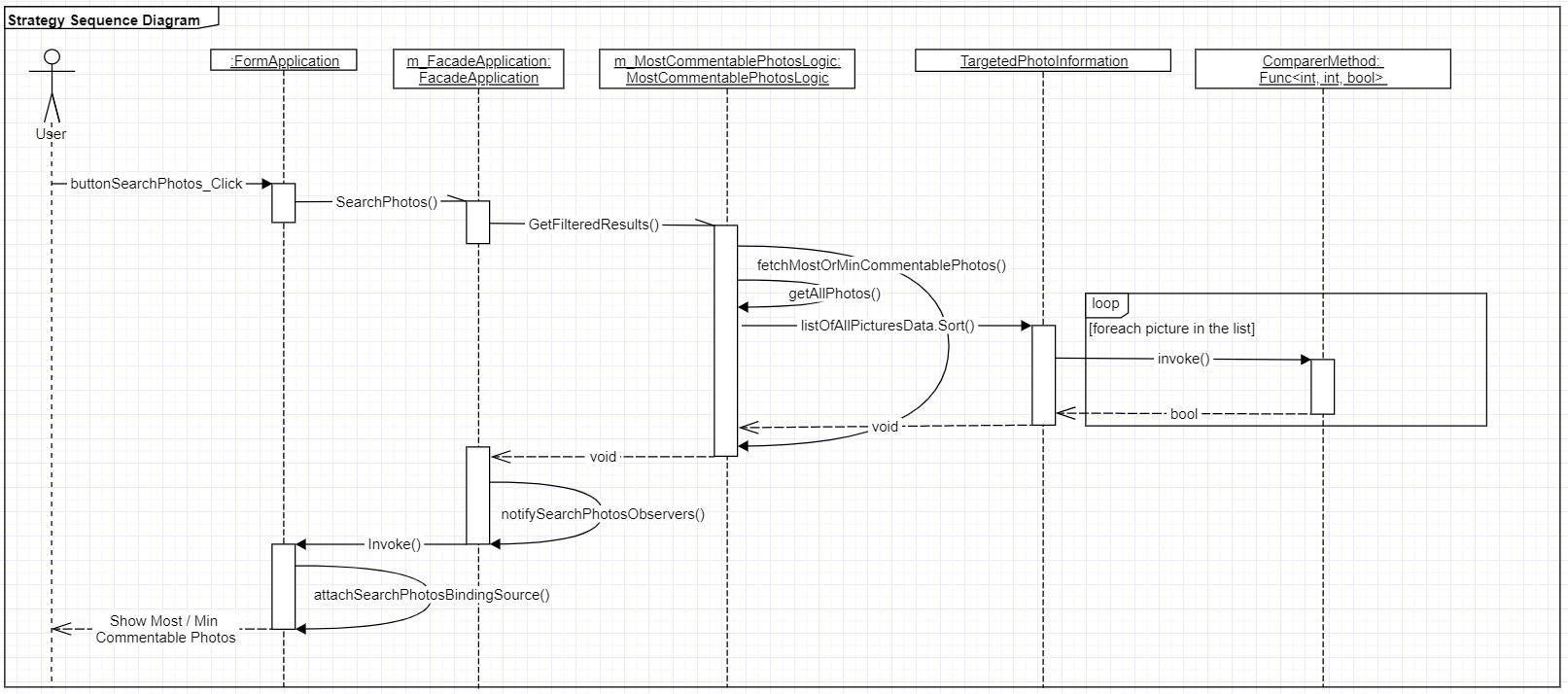
במקרה שלנו קיים צורך בווריאציה של האלגוריתם (בדיקה שונה עבור כל סוג מיון) התבנית "חסכה" לנו שימוש בשתי מחלקות קשורות השונות רק בהתנהגותן (אחת עבור כל מיון) ובכך מנענו שכפול קוד. Strategy מאפשר לקנפג מחלקה לעבודה עם מספר רב של התנהגויות.

* אופן המימוש:

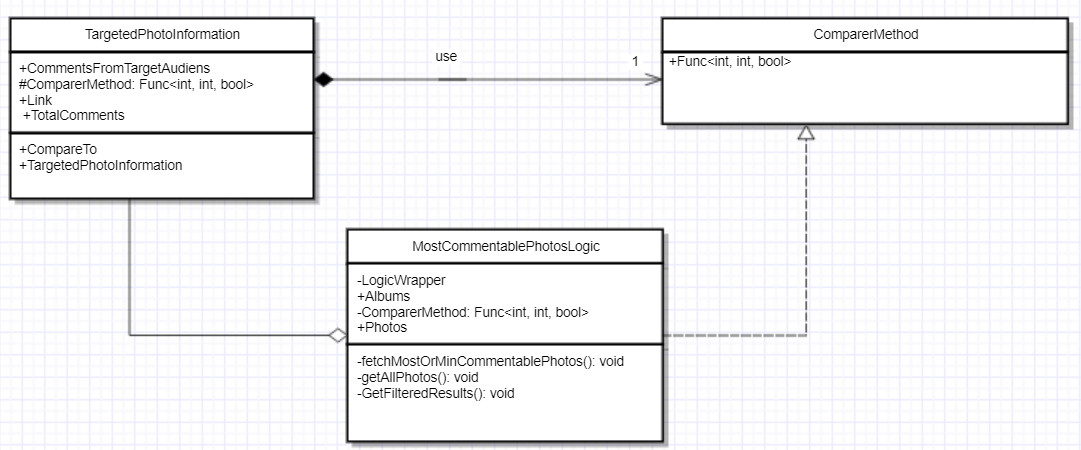
יצרנו Property בשם ComparerMethod מסוג Func<int, int, bool> במחלקות MostCommentablePhotosLogic ו- TargetedPhotoInformation.

המתודה GetFilteredResults שנמצאת במחלקה MostCommentablePhotosLogic יוצרת את הפונקציה של המיון בהתאם לתנאי שנקבע ע"י הערך שהיא מקבלת במשתנה v\_ToBeSortedByIncreasingSort (שמייצג את בחירת המשתמש בטופס FormApplication) ולאחר מכן לקראת סיום המתודה fetchMostOrMinCommentablePhotos (שהמתודה GetFilteredResults הפעילה) מתבצע מיון של התוצאות בהתאם לפונקציה שהמתודה GetFilteredResults הגדירה, המיון מתבצע במחלקה TargetedPhotoInformation כאשר מתבצעת השוואה בין שני אובייקטים של המחלקה במתודה CompareTo שמפעילה את המתודה ComparerMethod על שני האובייקטים וקובעת האם להחליף ביניהם בהתאם לתוצאה.

* Sequence Diagram



* Class Diagram



|  |  |
| --- | --- |
| Our Implementation | Pattern |
| ComparerMethod | Strategy |
| TargetedPhotoInformation | Context |
| MostCommentablePhotosLogic | ConcreteStrategy |