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

**הסבר על הפיצ'רים:**

**פיצ'ר 1:** מיון אלבומים לפי מספר תמונות באלבום.

הסבר: יצרנו מערך של אלבומים שגודלו כגודל מספר האלבומים שיש למשתמש.

לאחר מכן רצנו על האלבומים של המשתמש והוספנו למערך את האלבום בעל מספר התמונות הגדול ביותר. מימשנו את המיון בעזרת אלגוריתם bubble sort, בהנחה שכמות האלבומים הוא מספר שחסום על ידי 1000, ולכן זמן הריצה לא נפגע.

בעתיד השאיפה למיין גם לפי מספר Likes או מספר comments.

את הפיצ'ר אפשר למצוא לאחר שהשתמש לוחץ על כפתור Albums ולאחר מכן על Sort By, האלבומים יופיעו ממוינים לפי כמות התמונות בכל אלבום.

בקוד הפיצ'ר נמצא תחת AlbumsOps בתוך פונקציית SortLabel\_Click().

**פיצ'ר 2:** משחק הניחושים.

הסבר: משחק- נחש את הגיל של החברים שלך.

כאשר המשתמש לוחץ על כפתור Games, מופיע לו משחק הניחושים.

המערכת מגרילה חבר ראשון, ומופיעה שאלה – מה הגיל של החבר \_\_? (הערה: באפליקציה השאלה- מה הגיל של גיא רונן? התשובה 45)

על המשתמש לענות ולאחר מכן מופיע לו האם צדק או טעה, לאחר מכן מוגרל חבר נוסף, ככה המשתמש יכול לאסוף נקודות עד שנגמר המשחק.

את הפיצ'ר ניתן למצוא תחת כפתור Games. ובקוד ניתן למצוא את הפיצ'ר תחת GamesOps.

**פיצ'ר 3:** רשימת ספורט.  
זהו פיצ'ר שהוספנו למטלה מספר 2 על מנת שנוכל לממש ולהשתמש בתבניות הנדרשות.  
יצרנו פיצ'ר שמטרתו לעזור למשתמש לנהל את המטלות בספורטיוביות שהוא רוצה לבצע בעתיד הקרוב. המשתמש יכול להוסיף פעילות לרשימה, להוסיף לה תאריך יעד, לסמן שהפעילות נעשתה. המידע של המשתמש נשמר גם במעבר בין הדפים השונים. במידה והמשתמש מבצע פעילות ומסמן על כך, קופצת התראה המעודדות אותו להמשיך!

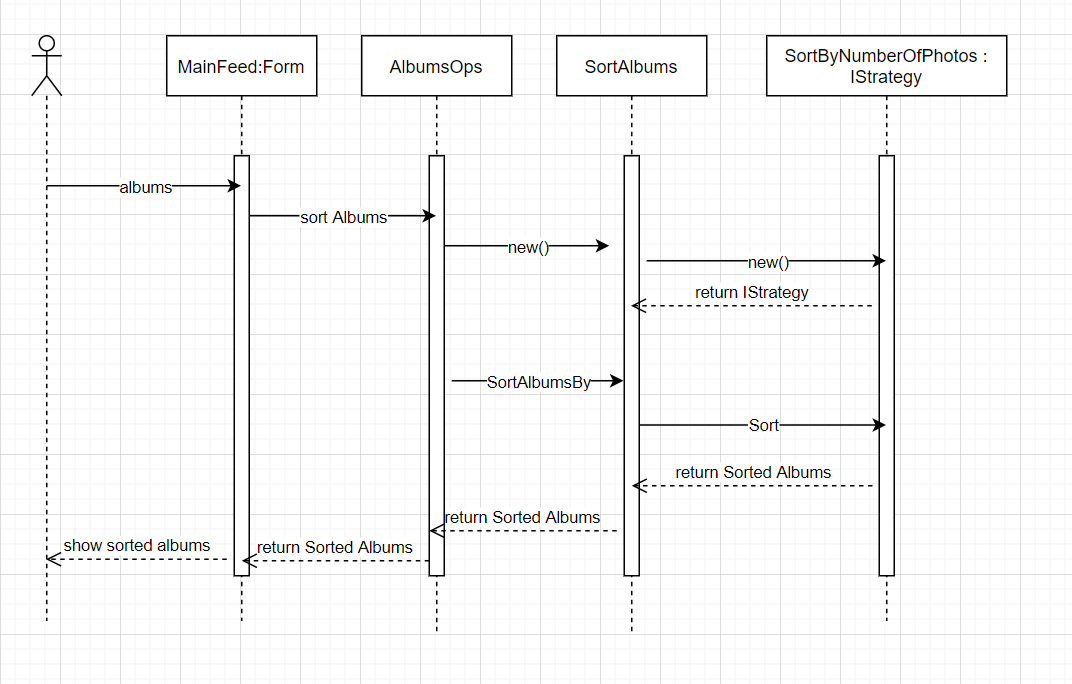
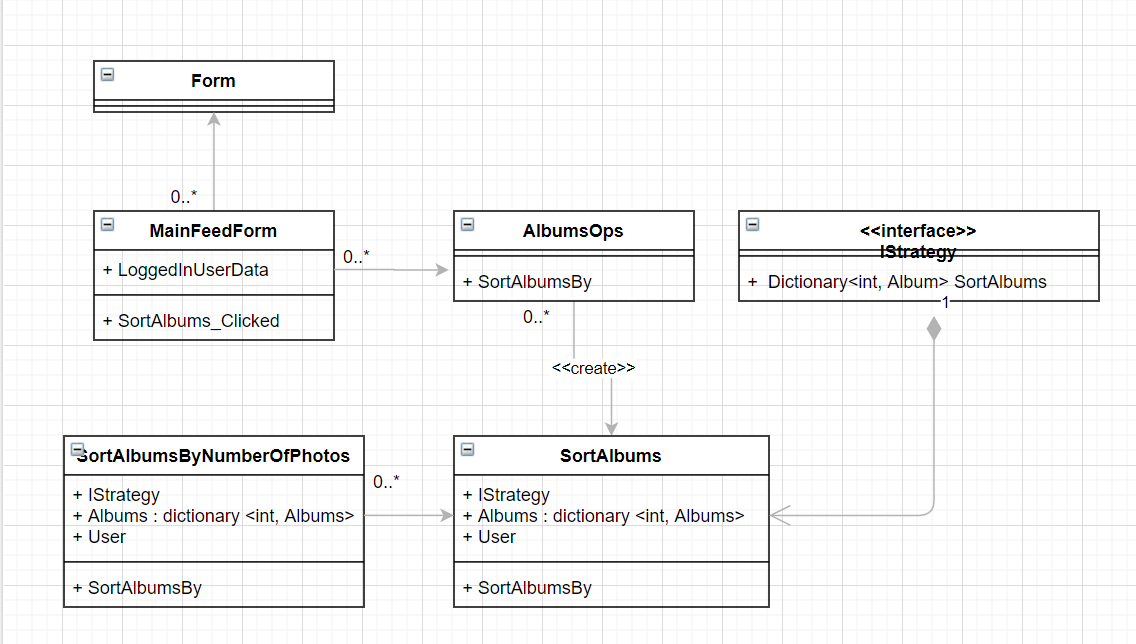
### תבנית מס' 1 – Strategy-

* **סיבת הבחירה / שימוש בתבנית:**

Strategy מציע פתרון למימוש המיון במידה ואם נרצה בעתמיד לשנות את סוג המיון ממיוון על פי מספר תמונות למיון אחר נוכל לעשות זאת מבלי לשנות את הקוד. בעצם, הclient יוכל להוסיף מימושי מיון שונים שיבנו על הפתרון שאנחנו מציגים. לא יהיה צורך לשנות קוד או לשכפל קוד אלא רק לשנות את סוג האסטרטגיה של המיון.   
באפליקציה שלנו ישנו פיצ'ר שממיין אלבומים לפי מספר התמונות בכל אלבום.

* **אופן המימוש:**

1. יצרנו מחלקה מסוג interface שנקראת IStrategy, בתוך מחלקה זו יש את המתודה שאחראית על ההתנהגות של המיון עצמו.
2. יצרנו מחלקה שנקראית SortAlbums שבתוכה ישנה מתודה שנקראית sortAlbumsBy ושם בעצם מחליטים את סוג המיון שיעשה על ידי המחלקה. המתודה קוראת לפונקציה SortAlbum
3. מחלקת SortAlbumsByNumOfPhotos – הינה דוגמא למימוש מיון כאשר המיון נעשה לפי מספר התמונות בכל אלבום. מחלקה זו יורשת מIStrategy ומממשת את המתודה SortAlbums.
4. האלבומים שאותם אנחנו ממיינים נשמרים בDictionary<int, Album> , שנרצה לסדר אותם לפי המיון נבצע foreach על האלבומים ונראה למשתמש את האלבומים בצורה ממויינת.

* **Sequence Diagram**
* **Class Diagram**

### תבנית מס' 2 –Iterator-

•**סיבת הבחירה / שימוש בתבנית:**

רצינו לתת למשתמש את האפשרות לדפדף על כל פעילויות הספורט שהמערכת הגדירה שעליו לעשות, ( בלי קשר לפיצר שמוממש בתרגיל הקודם ).

כדי לעשות זאת עלינו לעבור על כל האובייקטים מסוג SportActivity שקיימים עבור המשתמש הנוכחי.

הבחירה בשימוש בIterator היא ברורה מכיוון וpattern הזה מאפשר לעבור על אוסף של פריטים, סורק את האוסף ומבצע פעולות על האוסף.

מבנה הנתונים שאנו משתמשים בו הוא רשימה המכילה אובייקט מסוג SportList.

הוספנו את התבנית iterator כיוון שאם בעתיד נרצה לשנות את מבנה הנתונים, נרצה לאפשר ל-client לסרוק את אוסף האיברים ולהמשיך לבצע פעולות על האוסף בידיוק באותו האופן בו עשה עד עכשיו ולא לשנות את הקוד בclient.

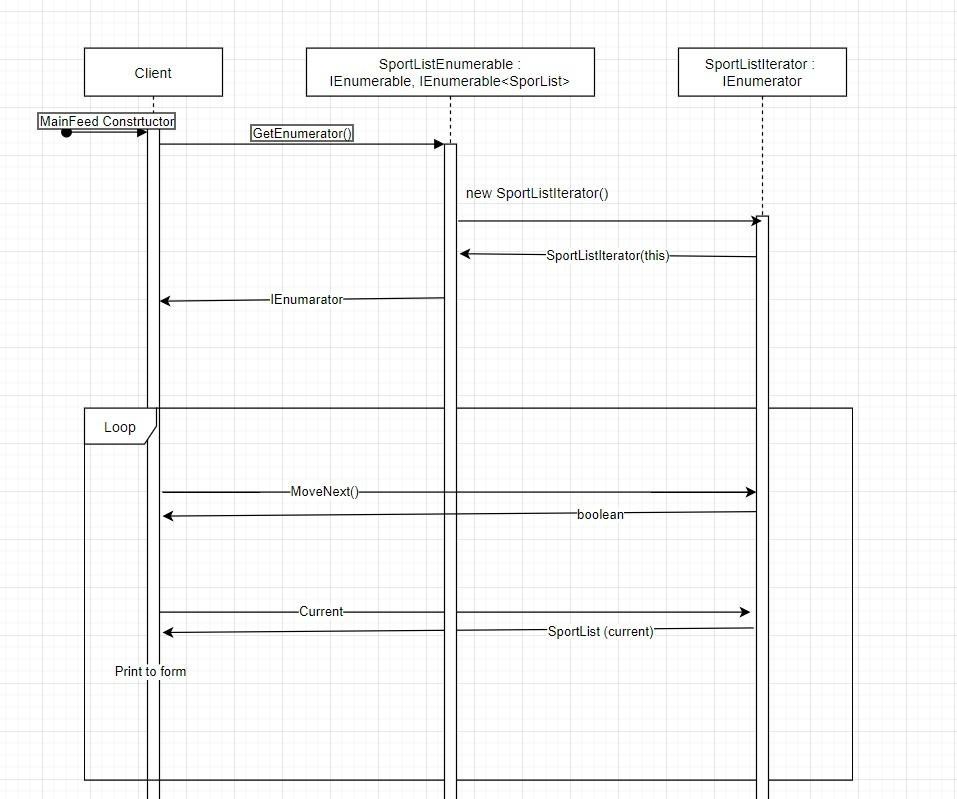
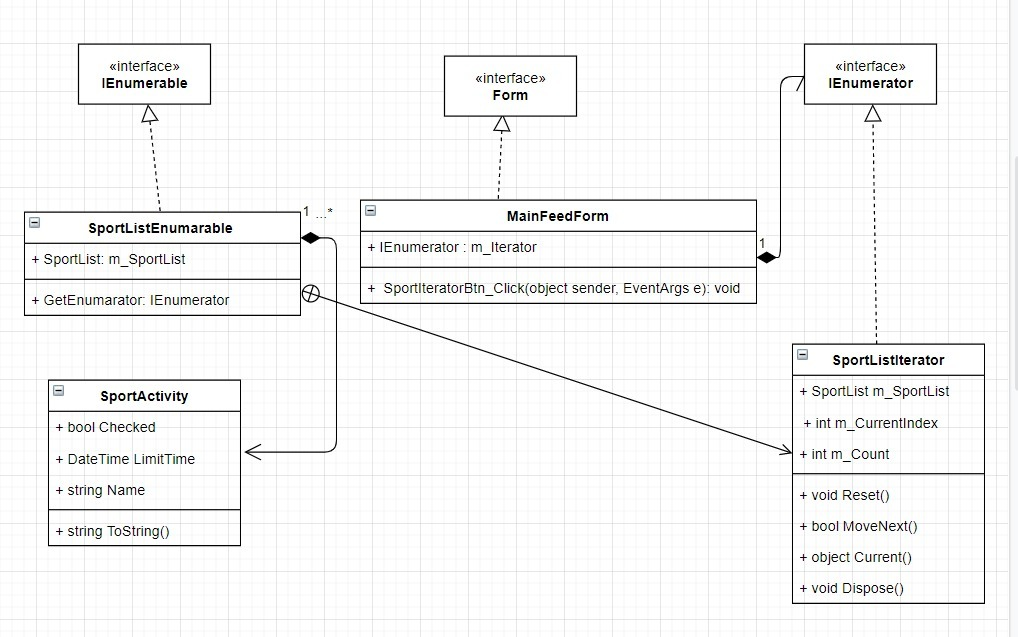
לכן, השתמשנו ב-iterator מהצד של ה-client. כרגע המימוש של מבנה הנתונים הוא רשימה אבל ה-client כלל לא מודע למבנה הנתונים ולכן לא יצטרך לשנות את הקוד.

•**אופן המימוש:**

oישנה מחלקה שנקראית SportActivity שהיא מייצגת את האובייקט שאנו רוצים לשמור באוסף שלנו. כלומר אוסף הנתונים מכיל אובייקטים מסוג זה שיש להם 3 שדות, string שמכיל את שם הפעילות, תאריך סף לביצוע הפעילות ודגל שבודק האם הפעילות הספורטיבית בוצעה או לא

oיש מחלקה שנקראית SportListEnumrable שמממשת את IEnumerable ואת IEnumerable<SportList>. היא זו שמחזיקה בתוכה את מבנה הנתונים- במימוש הנוכחי היא מחזיקה רשימה של SportActivity. בנוסף, מממשת את המטודות GetEnumerator של IEnumerable.

oבתוך המחלקה SportListEnumrable יש מחלקה private nested שנקראית SportListIterator שמממשת את IEnumerator ולכן מממשת 3 מתודות, MoveNext, Current ו-Reset.

* **Sequence Diagram**
* **Class Diagram**

### תבנית מס' 3 – Template Method-

• **סיבת הבחירה / שימוש בתבנית:**

בתקופת בחירות סוערות זו, רצינו לאפשר למשתמשי הפייסבוק לשלוח הודעות ברודקסט לכל האנשים בפייסבוק בעלי דעות פוליטיות זהות דומות לשלהם.

כלומר, אנשים בעלי דעות פוליטיות ימניות יוכלו לשלוח לבעלי דעות ימניות/ מרכז - והפו (תמיד נשלח למרכז).

החלטנו להשתמש בtemplate method pattern מכיוון שאם בעתיד נרצה להוסיף פיצר שמאפשר שליחה לכולם אך באופן שונה(ימין שמאל ומרכז, רק שמאל או רק ימין), נוכל לעשות זאת מבלי לשנות קוד קיים אלא רק לממש עוד מחלקה בעלת תכונות זהות למחלקות הקיימות.

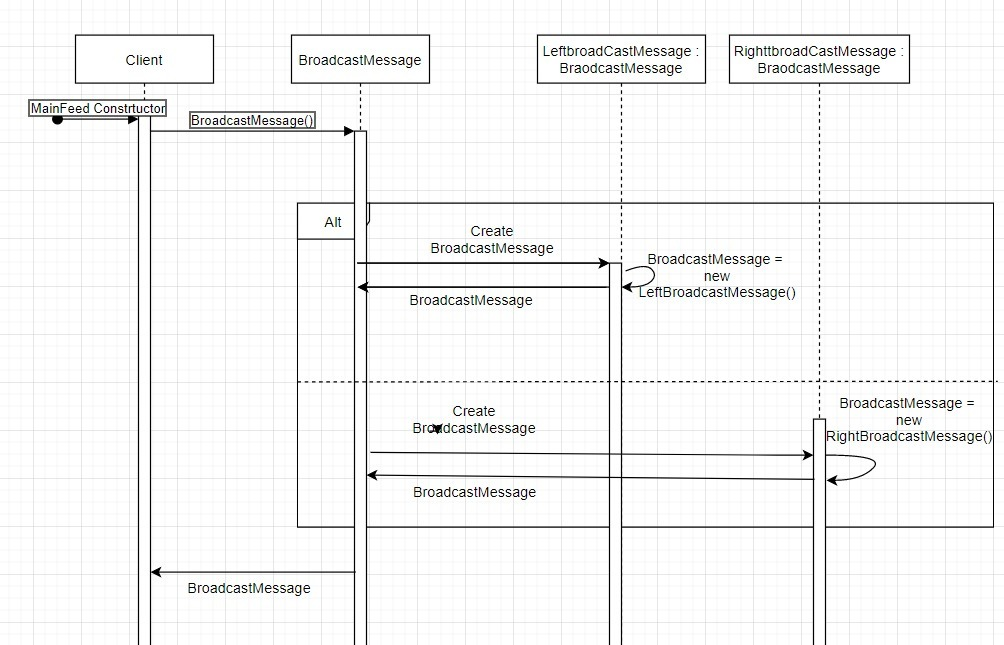
•**אופן המימוש:**

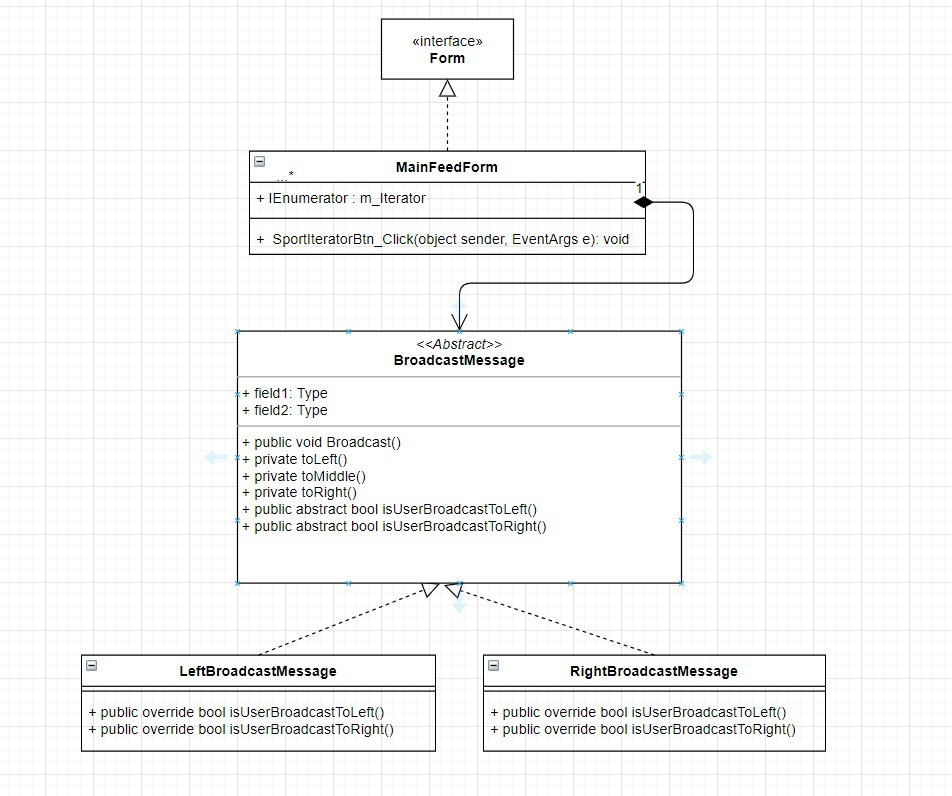
oישנה מחלקה אבסטרקטית שנקראית BroadcastMessage אשר ברשותה מספר מטודות.

האחת, BroadCast: אינה מטודה אבסרקטית, המטודה הזו למעשה עושה את עיקר העבודה ובכל יישום של המחלקה תמיד המטודה הזאת תהיה בשימוש. בנוסף 3 מטודות פרקטיות שלמעשה שולחות את ההודעות לפלח הנכון באוכלוסיה על פי השקפה פוליטית אשר תמיד יהיה זהה בכל מימוש מהיותו תהליך טכני. בנוסף, 2 מטודות אבסטרקטיות שמטרתן לאפשר לממש של מחלקת BroadcastMessage להחליט למי להוא רוצה לשלוח. המטודות הללו הן מטודות שמחזירות true/false ועל הממש חובת מימוש - למעשה נותן אפשרות לאפיין את המחלקה הספציפית בזמן המימוש.

oהמחלקה LeftBroadcastMessage אשר מממשת את BroadcastMessage וגם את 2 המטודות האבסטרקיות שלה ובכף מאפיינת את עצמה כמחלקה ששולחת רק לשמאל ולא לימין. (והפוך לגביי RightBroadcastMessage

* **Sequence Diagram**



* **Class Diagram**