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

* **חייה וירטואלית (טמגוצ'י)** – פיצ'ר זה מוסיף למשתמש חוויה נוספת לפייסבוק שהיא בחירת חייה וירטואלית.

ניתן לבחור מבין מספר חיות וירטואליות. יש לבחור לחיה שם ולהאכיל אותה כדי שמדד השמחה שלה יישאר גבוה. במידה ומדד השמחה של החיה יהיה נמוך, תצא הודעה למשתמש שהוא צריך להאכיל את החיה. ניתן לשנות את החיה בכל שלב.

כמו כן, לכל משפחת חיות יש אוכל שמתאים לה המתחלף בעת בחירת החיה.

פרטי החיה שהמשתמש בחר נשמרים ומוצגים לאחר סגירה ופתיחה של האפליקציה.

* **טיימר** – פיצ'ר זה מספק למשתמש התראה על זמן השימוש באפליקציה. המשתמש יכול בכל שלב ללחוץ על האייקון של השעון כדי לדעת כמה זמן הוא משתמש באפליקציה ובנוסף להגדיר טיימר שיתריע למשתמש שמומלץ לעשות הפסקה לפי הזמן שנבחר מבין האפשרויות בטופס. בחרנו לעשות פיצ'ר זה כי שימוש ברשתות חברתיות גורם להתמכרות ורצינו ליצור מנגנון שימחיש למשתמש את זמן השימוש באפליקציה וייתן למשתמש התראה לקחת הפסקה, הוספנו מנגון נוסף של שעון עצר.

### תבנית מס' 1 – Strategy

* סיבת הבחירה / שימוש בתבנית:

רצינו לספק למשתמש חווית פייסבוק אינטראקטיבית ובכך לאפשר לו למיין את התמונות הנבחרות שלו לפי קריטריון ההשוואה - תאריך, מהחדש לישן (ברירת המחדל) ולהפך.

בנוסף, אפשרנו למשתמש למיין את הדפים להם סימן לייק לפי קריטריון ההשוואה - שם הדף לפי סדר הא"ב.

המטרה היא לבצע מספר אלגוריתמים (2 עבור מיון התמונות ו-2 עבור מיון הדפים שסומנו בלייק) באמצעות אובייקט אחד בזמן ריצה (אובייקט אחד עבור התמונות ואוקייבט נוסף עבור הדפים שסומנו בלייק).

התבנית Strategy מאפשרת בזמן ריצה לקבוע את התנהגות של האובייקט.

כמו כן התבנית מאפשרת לנו להפריד בין הלוגיקה של המיון למתמש של התבנית (שהוא FormMain).

ניתן להוסיף אסטרטגיות נוספות למיון כמו למשל כמות הלייקים לתמונה, כמות הלייקים לדף הפייסבוק מבלי לשנות את הקוד הקיים.

* אופן המימוש:

המימוש של האסטרטגיות והממשק המשותף נמצא בתיקיה – Strategy.

הממשק ISortStrategy המגדיר מתודה אחת שהיא Sort.

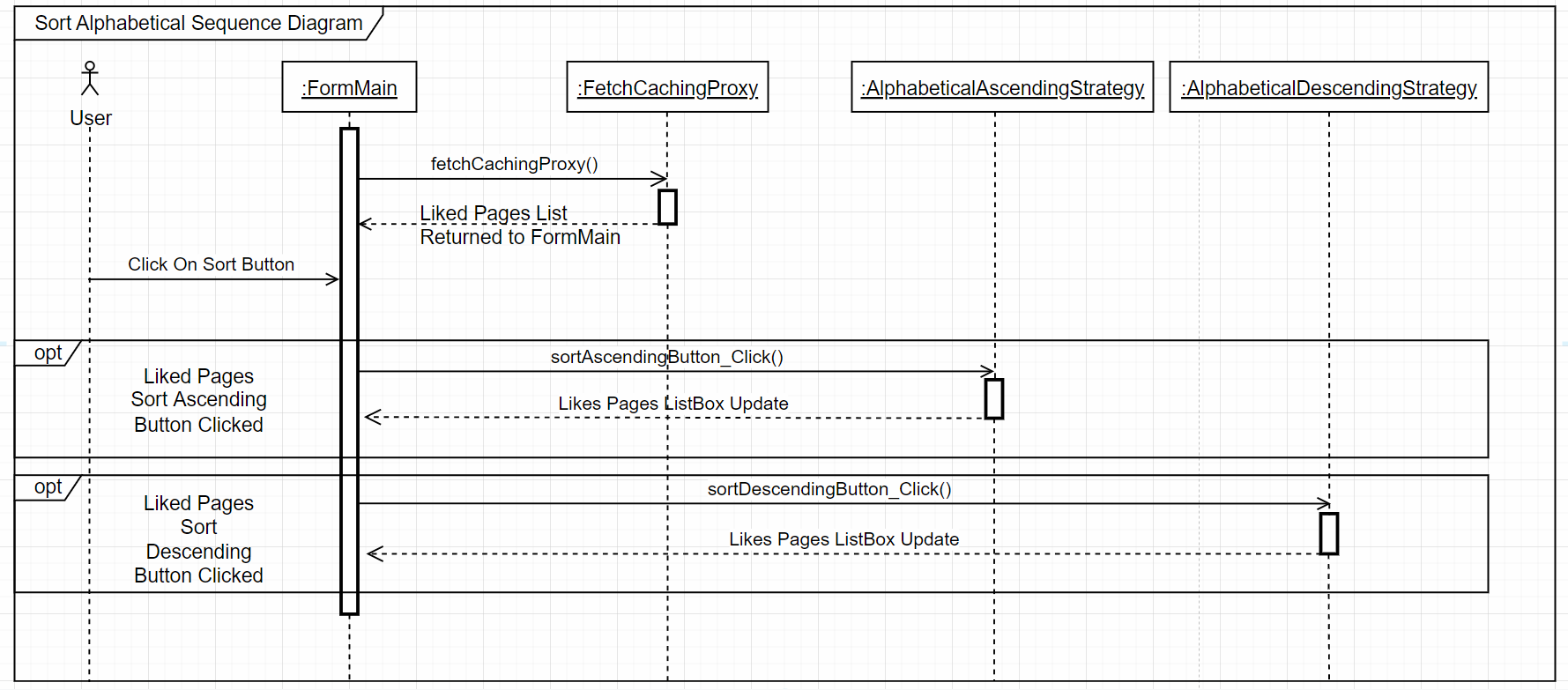
ישנן 4 מחלקות המממשות את הממשק:

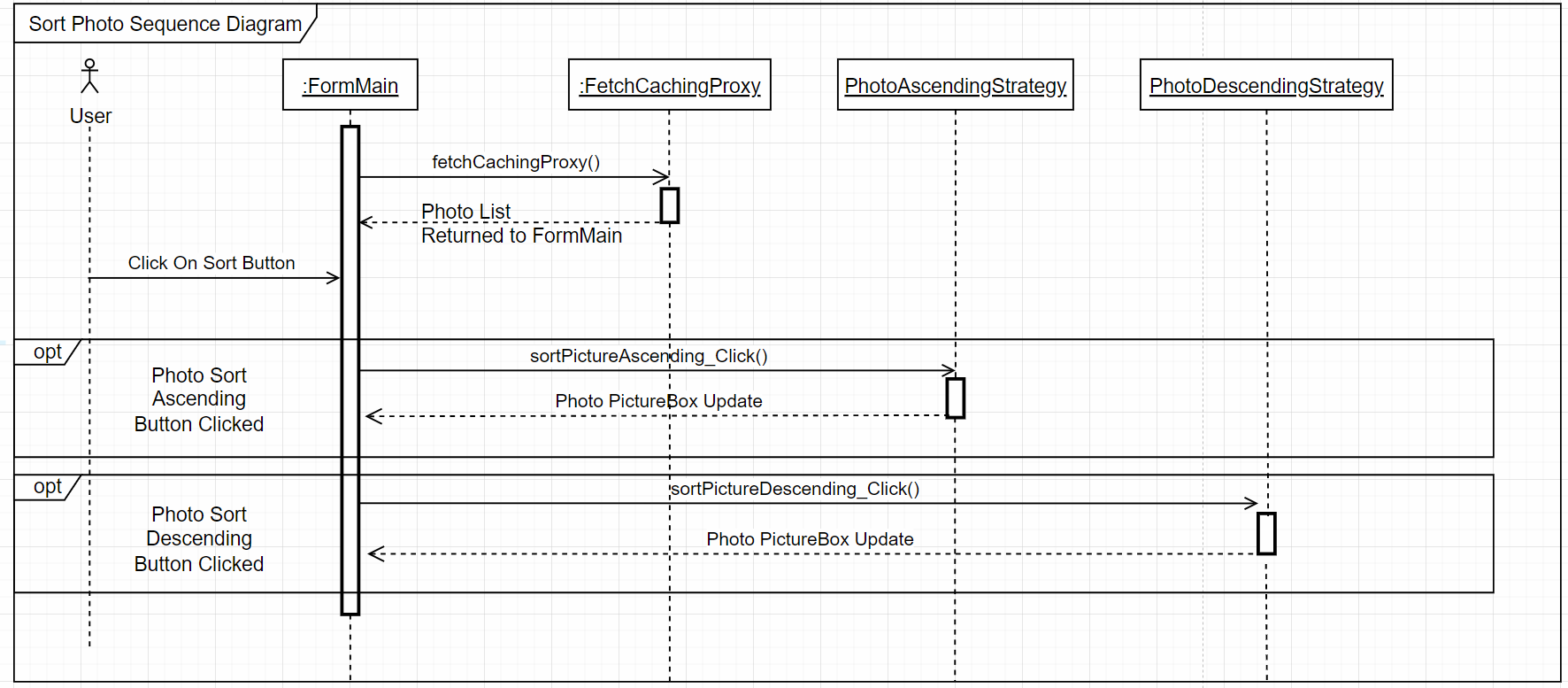
* + AlphabeticalAscendingStrategy
  + AlphabeticalDescendingStrategy
  + PhotoAscendingStrategy
  + PhotoDescendingStrategy

ה-Context הוא FormMain והוא מכיל (קומפוזיציה) מופע אחד מסוג ISortStrategy<List<Page>> עבור מיון הדפים שסומנו בלייק ומופע נוסף מסוג ISortStrategy<List<Photo>> עבור מיון התמונות.

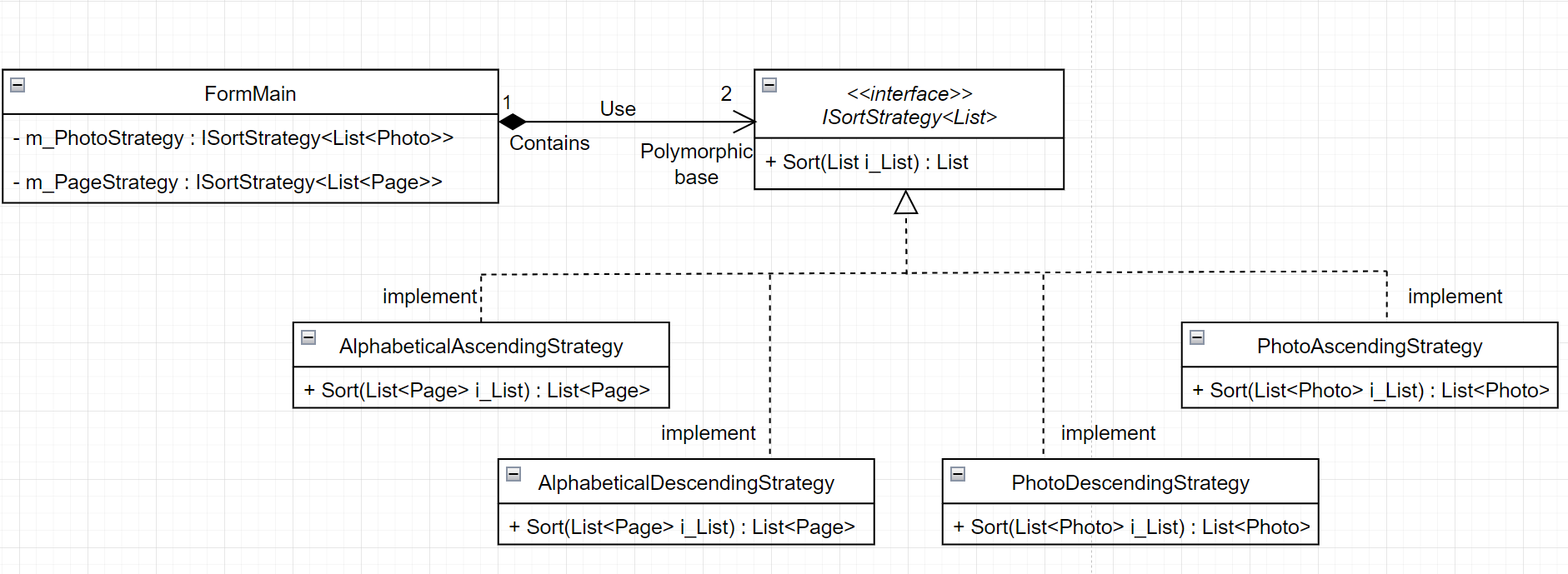
המיון מתבצע בזמן ריצה ע"י לחיצת הכפתור ומפעיל את את האסטרטגיה המבוקשת.

* Sequence Diagram





* Class Diagram



### תבנית מס' 2 – Iterator

* סיבת הבחירה / שימוש בתבנית:

החלטנו להשתמש בIterator כאשר אנו עוברים על הפוסטים באפליקציה.  
המצב הקודם היה כך שהמשתמש ידע שהוא עובד עם רשימה ,וכדי לקבל את הפוסט מהרשימה היה צריך לגשת לרשימת פוסטים דרך האינדקס, וכדי להתקדם לפוסט הבא היה צריך לקדם/להחזיר אחורה את האינדקס.

רצינו לבצע פתרון כללי יותר שיוכל להתאים למבנה נתונים אחר שלא בהכרח פועל עם גישה ישירה (אינדקס), וגם נוכל לעבור על הנתונים בלי שה Client ידע איך זה פועל כן כדי שיהיה ניתן לשנות בעתיד וגם כדי לשמור על המשתמש מלעשות טעויות וליצור באגים ללא צורך.

בנוסף הפתרון משפר את הקריאות וההבנה של הקוד.

בעקבות הפתרון שלנו מצאנו שיכולנו למחוק את הproperty PostIndex שנמצא בתוך מחלקת FacebookPosts שכבר לא היה בו צורך.

* אופן המימוש:

כדי לממש את הIterator השתמשנו בכלים שבשפה C# כדי שנוכל לעשות איטרציה על הפוסטים (foreach, moveNext,Current,Dispose,Reset)

השתמשנו ב IEnumerator,IEnumerable.

יש לנו מחלקה בשם FacebookPosts שבה יש property כרשימה של כל הפוסטים( הפוסטים מסוג PostAdapterWinForms שהשתמשנו במטלה 2).

מחלקה זו היא מחלקה האגרציה –בתוכה אוסף כל הפוסטים, היא ממשת את IEnumerable<PostAdapterWinForms> , GetEnumerator() שמחזירה את הIterator.

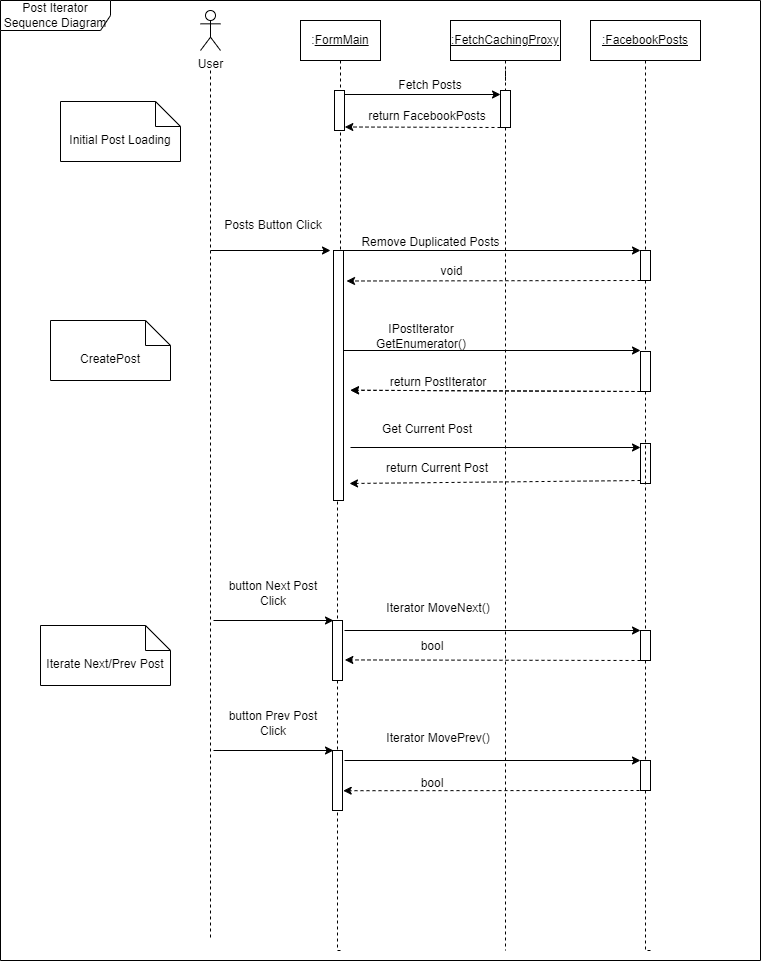
בתוך המחלקה FacebookPosts יש לנו מחלקה מקוננת PostIterator private, שהוא הIterator שלנו הוא ממש את הממשק IPostIterator שהוא הרחבה של IEnumerable< PostAdapterWinForms> ובנוסף יש לו מתודה של MovePrev() שרצינו להוסיף לIterator.

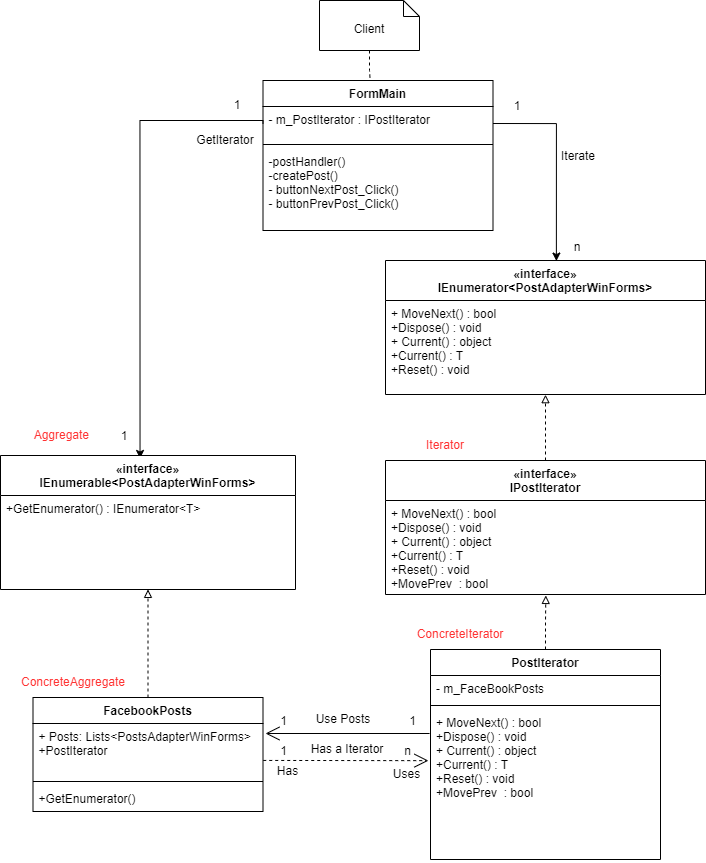
כדי שהמתשמש בFormMain יוכל להשתמש בIterator יש לו משתנה מסוג IPostIterator שמקבל את הIterator במתודה postsHandler ע"י ה- GetEnumerator של FacebookPosts.

כדי לקבל את הפוסט הנוכחי עם DataBinding במתודת createPost משתמשנו בCurrent.

כדי להתקדם ע"י לחיצת כפתור נשתמש בMoveNext(), באותו אופן MovePrev().

Sequence Diagram



* Class Diagram

### תבנית מס' 3 – Observer

* סיבת הבחירה / שימוש בתבנית:

באפליקציה ישנם שני רכיבים המציגים הודעות למשתמש. הרכיב הראשון הוא התראה על זמן השימוש באפליקציה, הרכיב השני הוא התראה על כך שצריך להאכיל את החייה הוירטואלית.

כדי לשפר את חווית המשתמש באפליקציה, רצינו לרכז את ההתראות במקום אחד ולכן בחרנו לרכז אותן תחת חלון שיפתח בלחיצה על הכפתור 'Notifications'.

למשל, במידה וישנה התראה על שימוש בזמן האפליקציה, בנוסף להתראה בחלון קופץ , ההתראה תתווסף לחלון של ה-Notifications.

השתמשנו בתבנית זו כדי להמחיש מנגנון של הפצת התראה, ומנגנון של ההאזנה להודעות.

כמו כן, השימוש בתבנית זו מאפשר לנו להרחיב בעתיד את הפונקציונליות של האפליקציה ולהאזין להתראות אחרות, בנוסף מכיוון שהשתמשנו בdelegates נוכל להוסיף מאזינים נוספים כגון Email Listener/ DM Listener ועוד..

* אופן המימוש:

בחרנו לממש את הdesign pattern שאנו מנצלים הפונקציונאליות של C# כך שאנו משתמשים בGenericDelegates and Events.

המחלקות PetForm ו-AlarmForm שאחראיות להציג הודעות למשתמש, ממשות ממשק בשם INotifier ממשק זה מחייב אותם להוסיף Event Action<string> ומתודה בשם Notify ( בחרנו ליצור ממשק לכל המחלקות שצריכות לדווח, במחשבה שיהיו עוד בעתיד).

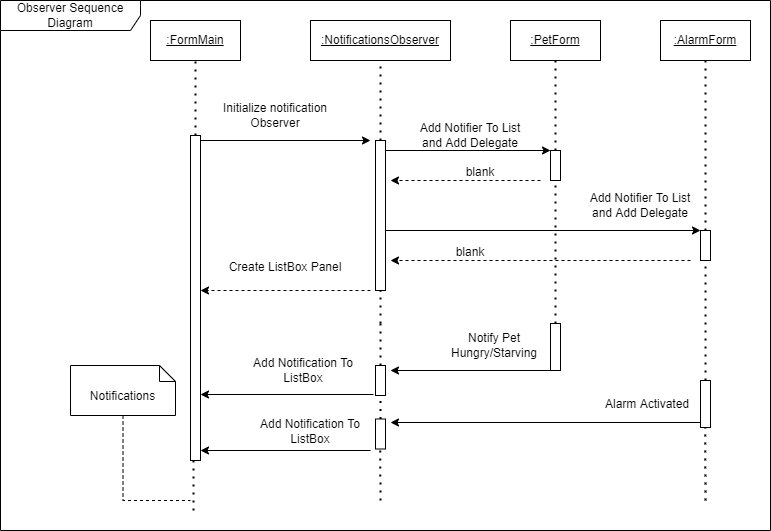
שני מחלקות אלה משמשות כNotifiers שלנו.

המחלקה המאזינה שלנו היא מחלקת NotificationsObserver היא מחזיקה רשימה של INotifiers ולכל Notifier היא מוסיפה delegate שמוסיף את ההתראה לרשימת ההתראות שלנו.

בכל מחלקה כאשר יש חלון קופץ הוספנו גם מימוש של Notify, ששולח התראה לחלון הNotifications שלנו במסך FormMain.

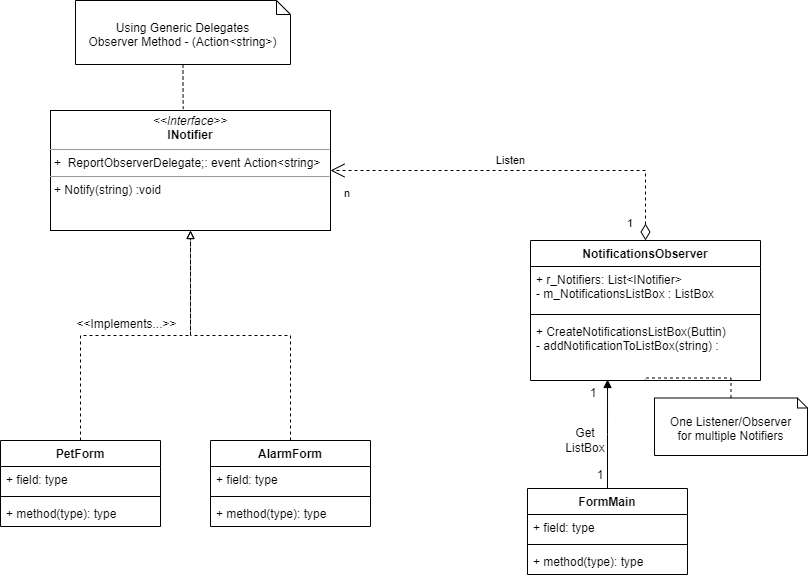
ברגע שנוצרת התראה שיש לצרף ל-Notifications, (בעקבות זמן השימוש באפליקציה או שהחיה הוירטואלית רעבה) מופעלת המתודה שרשומה ל-Delegate המעדכנת את ה-NotificationForm שהוא ה-Listener.

המחלקות נמצאות בתיקיית Observer בנוסף ל FormMain.cs, PetForm.cs,AlarmForm.cs.

* Sequence Diagram

הערה :

החצים שמפעילים את ההתראות עם הDelegate מופעלים רק לכיוון אחד ולכן הן פונים שמאלה ולא מקווקווים.

* Class Diagram