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

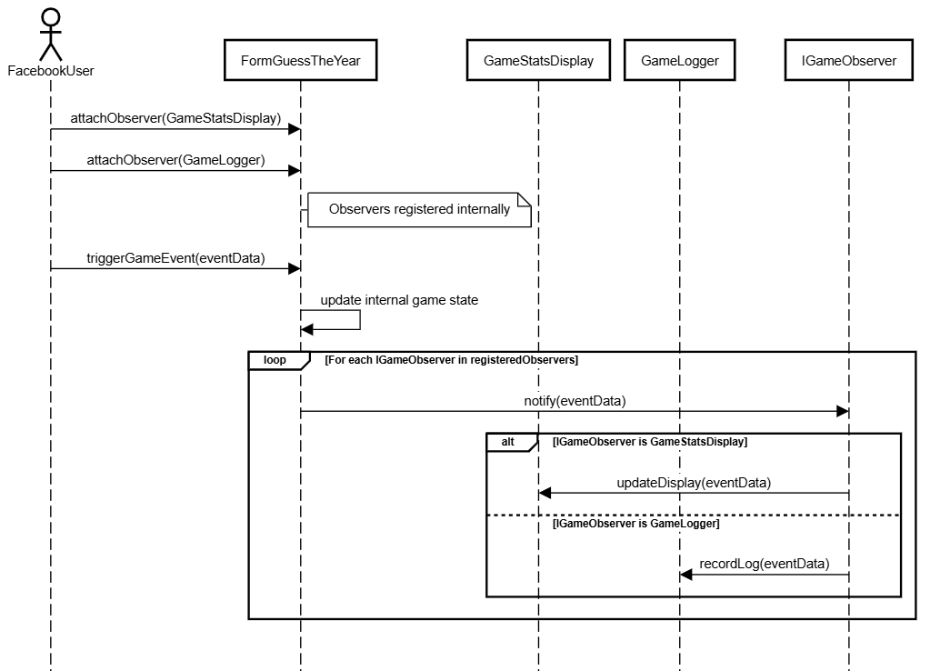
* Profile Analyzer
  + פיצ'ר שמנתח את פרופיל המשתמש ומציג מידע מקיף על הפעילות שלו בפייסבוק:
    - סטטיסטיקות כלליות (מספר לייקים, חברים, אירועים, פוסטים, סרטונים ותמונות).
    - התמונה שקיבלה הכי הרבה לייקים והתמונה שקיבלה הכי מעט.
    - רשימות חברים מסוננות (חברים שמדברים את אותה שפה, מאותה עיר, עם אותו יום הולדת).
* Guess The Year
  + משחק אינטראקטיבי שבו המשתמש צריך לנחש באיזו שנה צולמה תמונה מהאלבומים שלו:
    - המשחק מציג תמונה אקראית מהאלבומים של המשתמש.
    - מוצגות 4 אפשרויות בחירה של שנים שונות.
    - המשתמש צריך לבחור את השנה הנכונה שבה צולמה התמונה.
    - המשחק שומר מעקב אחר תשובות נכונות ושגויות.
    - בסיום מוצג סיכום של הביצועים.
    - רושם לוגים על התקדמות המשחק לקובץ טקסט GameLog.txt.

### תבנית מס' 1 – Observer

* סיבת הבחירה / שימוש בתבנית:
  + בחרנו להשתמש בתבנית Observerמשום שהיא מאפשרת עדכון אוטומטי של מספר רכיבים שונים במערכת בכל פעם שמתרחש שינוי במצב המשחק.
  + ריכוז האחריות בעדכון הצופים - התבנית מאפשרת לרכז את הלוגיקה לעדכון רכיבי ממשק המשתמש (UI) ורישום הנתונים במקום אחד (GuessTheYearGame) במקום לקרוא ידנית לכל עדכון בנפרד.
  + תמיכה במערכת גמישה ומדרגית - ניתן להוסיף ולנהל מספר בלתי מוגבל של "צופים" (Observers) מבלי לשנות את הלוגיקה המרכזית של המשחק.
  + הפחתת תלות הדדית בין מחלקות - התצוגה (FormGuessTheYear) אינה תלויה ישירות במחלקת המשחק (GuessTheYearGame) אלא מתעדכנת אוטומטית דרך הצופים.
  + תמיכה בכמה סוגים של תגובות לשינוי - לדוגמה, יש צופה (GameStatsDisplay) שמעדכן את ה-Labels במסך, ויש צופה אחר (GameLogger) שמתעד את השינויים בקובץ לוג (GameLog.txt).
  + שיפור ביצועים והימנעות מקוד מיותר - במקום לבצע קריאות ישירות ועדכונים רבים בנפרד, GuessTheYearGame פשוט קורא לnotifyObservers() וכל הצופים מתעדכנים יחד.
* אופן המימוש:
  + ממשק IGameObserver :
    - מגדיר את המתודה

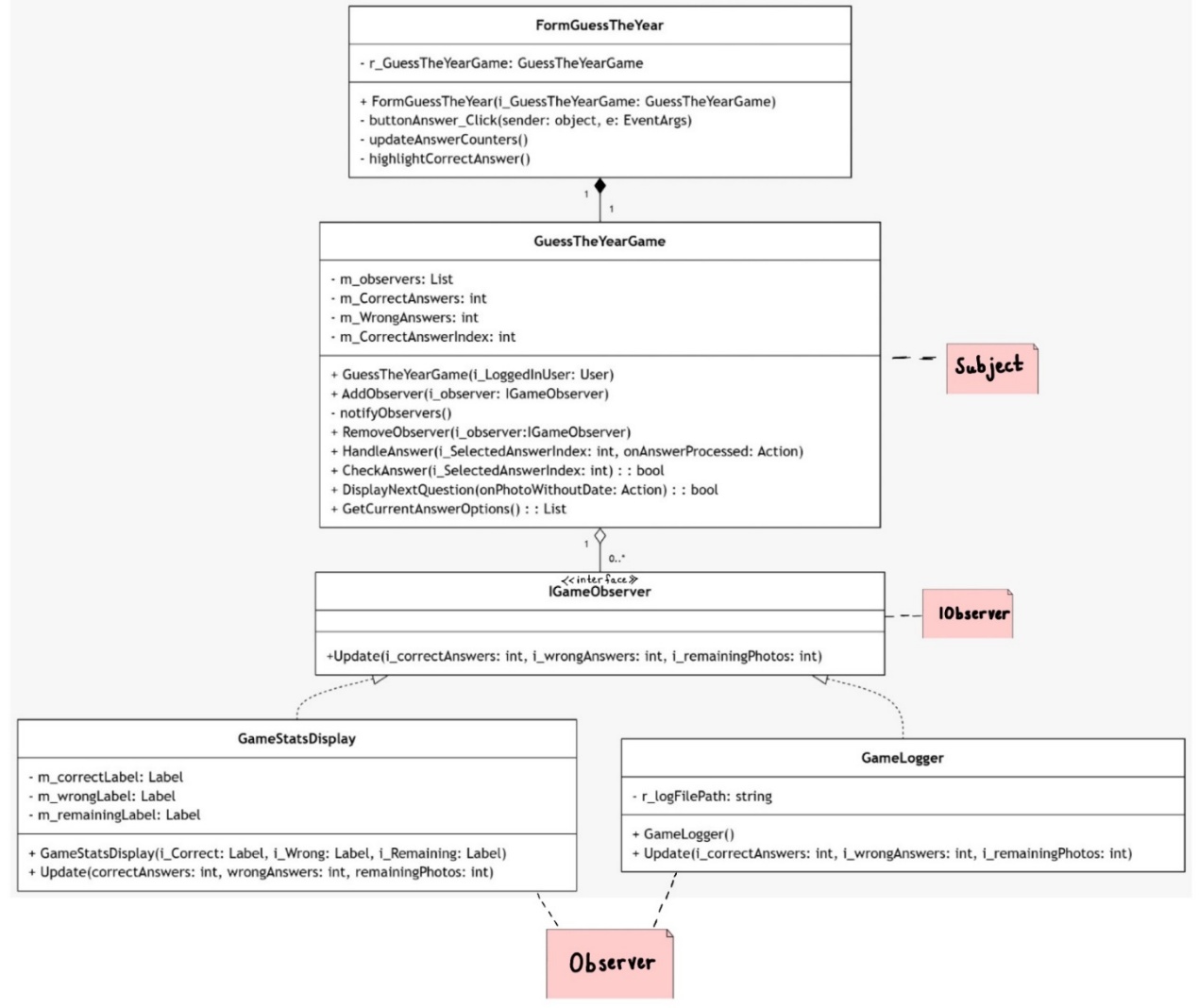
Update(correctAnswers, wrongAnswers, remainingPhotos).

* + - כל מחלקה שמקשיבה לשינויים במשחק חייבת לממש את הממשק הזה.
    - נמצאת בתיקיית Observers.
  + מחלקת GameStatsDisplay (צופה שמעדכן את ה-UI ):
    - שומרת הפניות לתוויות (Labels) במסך המשחק.
    - כאשר המשחק מתעדכן, הוא מעדכן את ה-Labels אוטומטית דרך Update().
    - נמצאת בתיקיית Observers.
  + מחלקת GameLogger(צופה ששומר מידע בקובץ GameLog.txt):
    - רושם לוגים על התקדמות המשחק לקובץ טקסט.
    - שומר את מספר התשובות הנכונות, השגויות והתמונות שנותרו.
    - נמצאת בתיקיית Observers.
  + מחלקת GuessTheYearGame (הנושא - Subject)
    - מחזיקה רשימה של כל ה) Observers -צופים).
    - ספקת פונקציות AddObserver() ו notifyObservers() –
    - בכל פעם שמתקבלת תשובה או שמתחילה שאלה חדשה, כל הצופים מתעדכנים אוטומטית. בכל פעם שמתקבלת תשובה או שמתחילה שאלה חדשה, כל הצופים מתעדכנים אוטומטית.
    - נמצאת בתיקיית Logic.
  + חיבור הצופים למשחק (FormGuessTheYear):
    - בעת יצירת המשחק, אנו מוסיפים את הצופים
    - עת, בכל שינוי במשחק, גם ה-UI וגם קובץ הלוגים מתעדכנים אוטומטית.
    - נמצאת בתיקיית UI.
* Sequence Diagram



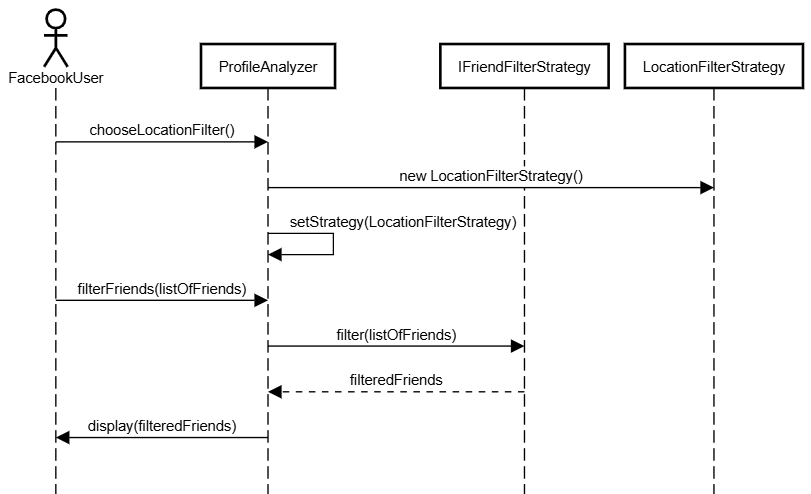


* Class Diagram



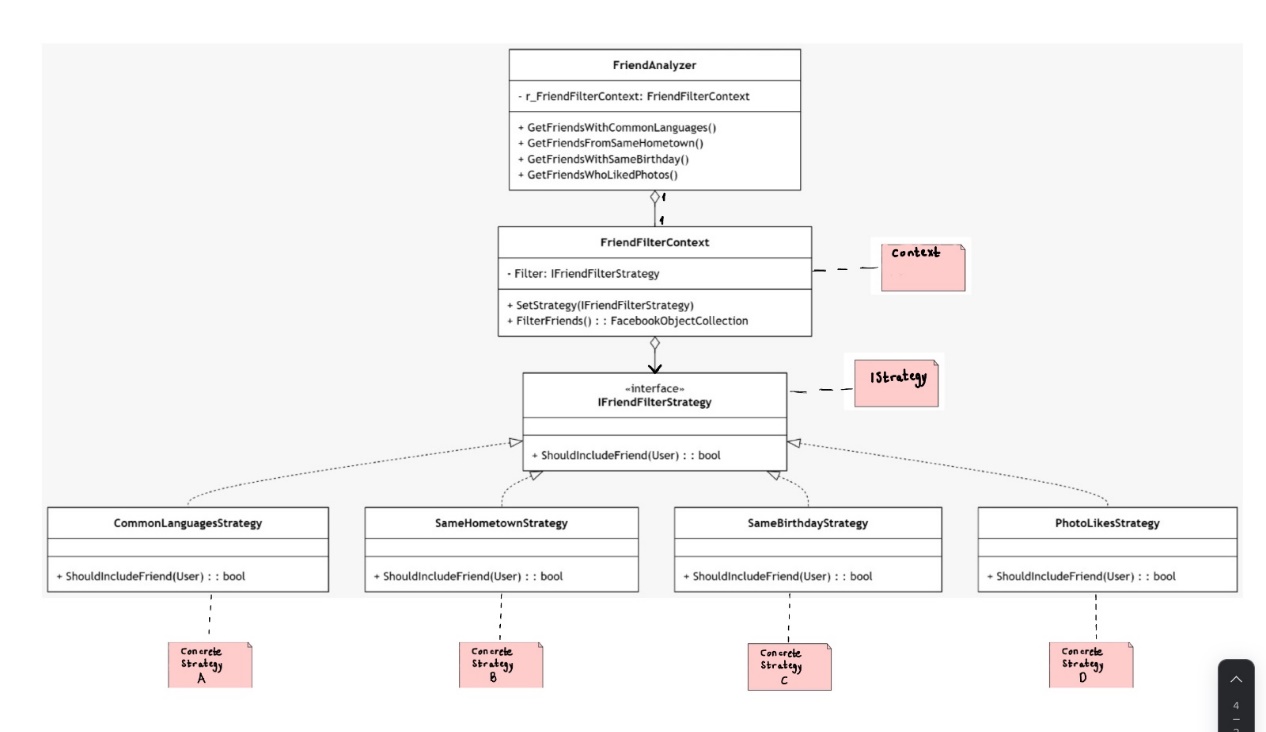
### תבנית מס' 2 – Strategy

* סיבת הבחירה / שימוש בתבנית:
  + בחרנו להשתמש בתבנית Strategy במערכת שלנו מכיוון שהיא מספקת פתרון גמיש ומודולרי לסינון רשימת החברים לפי קריטריונים שונים.
  + במערכת שלנו יש צורך בסינון דינמי של רשימת חברים לפי קריטריונים שונים, כמו:
    - שפות משותפות (CommonLanguagesStrategy)
    - עיר מולדת משותפת (SameHometownStrategy)
    - תאריך לידה זהה (SameBirthdayStrategy)
    - חברים שעשו לייק לתמונות (PhotoLikesStrategy)
  + היתרונות המרכזיים של Strategy במערכת שלנו:
    - מונע כפילויות בקוד - במקום לכתוב לוגיקת סינון נפרדת בכל מקום, יש לנו ממשק אחיד לכל האסטרטגיות, והקוד מרוכז ומסודר במקום אחד.
    - מאפשר החלפה דינמית של קריטריוני סינון - במקום להשתמש ב -if else מסובכים, ניתן להחליף את האסטרטגיה בזמן ריצה מבלי לשנות קוד קיים.
    - שומר על עקרון ה – SOLID במיוחד עקרון הפתיחות לסגירה – (Open/Closed Principle) אם רוצים להוסיף סוג חדש של סינון, אין צורך לשנות קוד קיים, אלא רק להוסיף אסטרטגיה חדשה.
    - משפר את הביצועים - אין צורך ליצור מחדש את מחלקת ניהול הסינון בכל פעם, אלא רק מחליפים את האסטרטגיה הנוכחית.
* אופן המימוש:
  + ממשק IFriendFilterStrategy – מגדיר את הפעולה שכל סוג של סינון חייב לממש.
    - לדוגמה: כל אסטרטגיית סינון צריכה להחליט האם לכלול חבר ברשימה או לא.
  + מימושים שונים של אסטרטגיות סינון – כל שיטת סינון מיושמת כמחלקה נפרדת, למשל:
    - CommonLanguagesStrategy מחזירה חברים עם שפה משותפת.
    - SameHometownStrategy מחזירה חברים מאותה עיר מולדת.
    - SameBirthdayStrategy מחזירה חברים עם תאריך לידה זהה.
    - PhotoLikesStrategy מחזירה חברים שעשו לייק לתמונות.
    - נמצאת בתיקיית Strategies.
  + מחלקת FriendFilterContext – מנהלת את אסטרטגיית הסינון.
    - מקבלת אסטרטגיה ראשונית (למשל, סינון לפי שפה משותפת).
    - מאפשרת להחליף אסטרטגיה בזמן ריצה כדי לשנות את קריטריון הסינון.
    - מבצעת את הסינון בפועל, תוך שימוש בלולאה שעוברת על החברים ובודקת אם הם מתאימים לקריטריון.
    - נמצאת בתיקיית Strategies.
  + מחלקת FriendAnalyzer – משתמשת בFriendFilterContext כדי להפעיל את הסינון בפועל.
    - יש לה פונקציות שמבצעות שינוי אסטרטגיה לפני הרצת הסינון.
    - משתמשת בFriendFilterContext יחיד, כך שהמערכת לא יוצרת מחדש את האובייקט כל פעם, אלא רק מחליפה אסטרטגיה.
    - נמצאת בתיקיית Logic.
* Sequence Diagram





* Class Diagram



### תבנית מס' 3 – Iterator

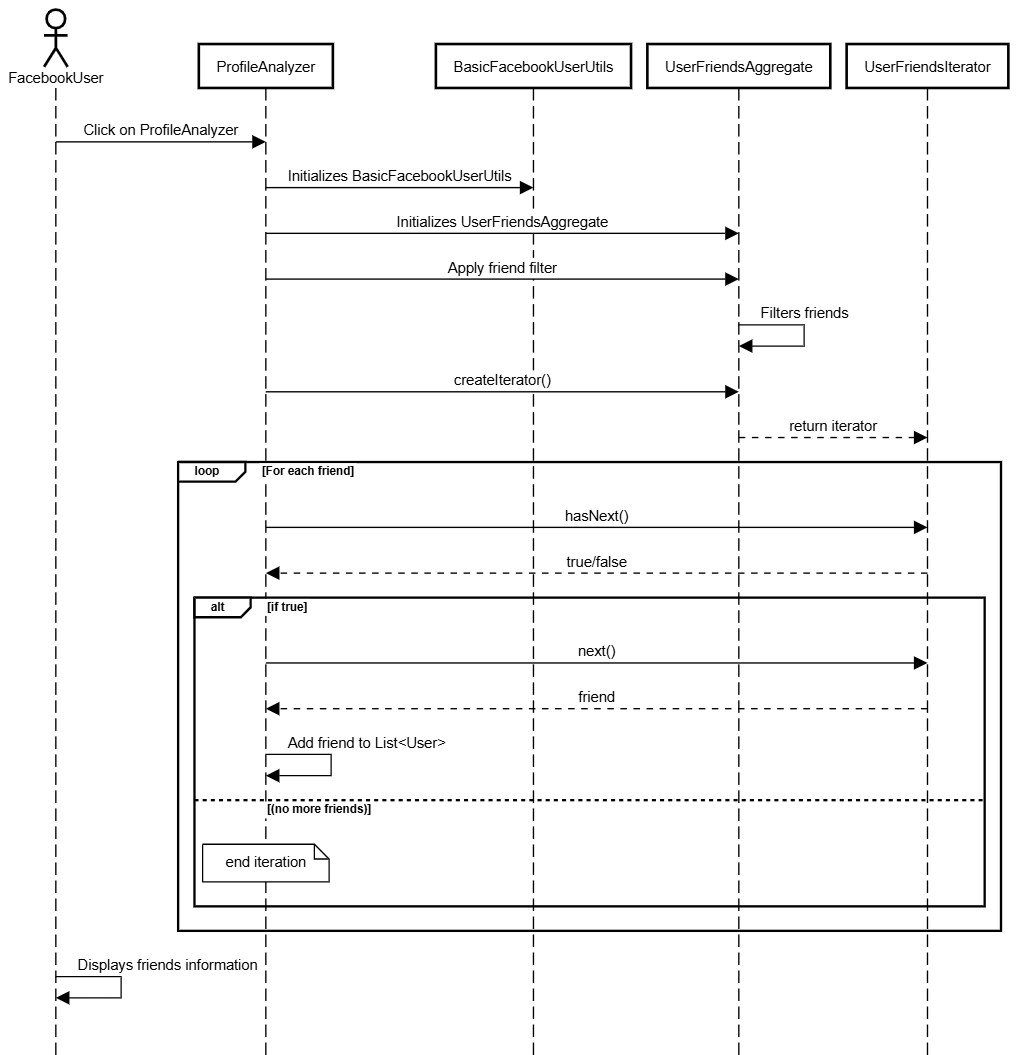
* סיבת הבחירה / שימוש בתבנית:
  + במערכת שלנו, אנו עובדים עם אוספים גדולים ומורכבים של נתונים, כמו:
    - רשימת החברים של המשתמש בפייסבוק.
    - רשימת כל התמונות מכל האלבומים של המשתמש.
  + כדי לאפשר ללקוח (Client) לסרוק את הנתונים בצורה פשוטה ויעילה, בחרנו להשתמש בתבנית העיצוב .Iterator
  + תבנית זו מספקת מנגנון מעבר סטנדרטי ואחיד על כל האלמנטים באוסף בזמן ריצה של O(n) כלומר, מעבר ליניארי על הנתונים ללא צורך בחשיפת המבנה הפנימי של האוסף.
  + היתרונות המרכזיים של Iterator במערכת שלנו:
    - גישה אחידה לכל סוגי הנתונים - הלקוח יכול להשתמש באותו מנגנון (foreach) לכל אוסף, בלי תלות במבנה הפנימי.
    - סריקה ליניארית O(n) – - כל מעבר על האוסף מתבצע באופן רציף וללא קפיצות בזיכרון, מה שמשפר ביצועים ויעילות.
    - הפרדת האחריות - מחלקות הנתונים (האוספים) אינן אחראיות לניהול המעבר על האלמנטים שלהן.
    - תמיכה בסריקה ללא שינוי באוסף - מאפשר מעבר על הנתונים מבלי לשנות את המבנה שלהם.
    - שיפור הקריאות והתחזוקה - הלקוח יכול לסרוק את הנתונים עם foreach במקום להשתמש בלולאות מסובכות עם אינדקסים.
* אופן המימוש:
  + במערכת שלנו, תבניתIterator יושמה תוך שימוש במנגנונים המובנים של C# IEnumerable<T>) ו(IEnumerator<T> - האלמנטים באוסף, מבלי לחשוף את המבנה הפנימי שלהם.
  + מחלקות לניהול הנתונים (ConcreteAggregate)
    - FacebookFriendsAggregate.cs מנהלת את רשימת החברים של המשתמש.
    - FacebookPhotosAggregate.cs מנהלת את רשימת כל התמונות מכל האלבומים.
    - מחלקות אלו מממשות את הממשק IEnumerable<T>, . מה שמאפשר מעבר באמצעות foreachללא צורך בניהול אינדקסים ידני.  
      הן מחזירות Enumerator אוטומטי, מה שמאפשר מעבר על הנתונים לפי הצורך.
    - נמצאת בתיקיית Iterators.
  + שימוש באיטרטור (IEnumerator<T>) למעבר על הנתונים &Iterator): (ConcreteIterator
    - במקום לממש מחלקת Iterator ידנית, אנו משתמשים במנגנון האיטרציה של #C (IEnumerator<T>)שמספק את כל הפונקציות הנדרשות לניהול האיטרציה, כמו:
      * MoveNext() מעבר לאלמנט הבא.
      * Current קבלת האלמנט הנוכחי.
      * Reset() חזרה לתחילת האוסף (אם נדרש).
      * האיטרטור נוצר באופן אוטומטי על ידי GetEnumerator() כאשר מבוצעת לולאת foreach ולכן אין צורך לקרוא לפונקציות Next() או IsDone() באופן ידני.
  + מחלקות שמעבדות את הנתונים (Client)
    - FriendAnalyzer.cs מבצע ניתוחים על רשימת החברים.
    - PhotoAnalyzer.cs מבצע ניתוחים על התמונות.
    - מחלקות אלו משתמשות באובייקטי האוסף והאיטרטור כדי לבצע ניתוח חכם של הנתונים.
    - FriendAnalyzer משתמש באיטרטור כדי לעבור על רשימת החברים ולנתח אותם.
    - PhotoAnalyzer סופר את מספר התמונות, מוצא את התמונה עם הכי הרבה לייקים ומבצע פעולות נוספות על הנתונים.
    - נמצאת בתיקיית Logic.

טבלת התאמה בין הדיאגרמה לקוד

|  |  |  |
| --- | --- | --- |
| רכיב בדיאגרמה | תפקיד | המחלקה במערכת |
| Aggregate | ממשק כללי לאוספים עם יצירת איטרטור | IEnumerable<T> (מובנה ב-C#) |
| ConcreteAggregate | מנהל את הנתונים בפועל ומחזיר איטרטור | FacebookFriendsAggregate, FacebookPhotosAggregate |
| Iterator | מגדיר ממשק אחיד למעבר על הנתונים | IEnumerator<T> של C# |
| ConcreteIterator | מממש את הלוגיקה של האיטרציה | IEnumerator<T> של C# (אין צורך במימוש ידני) |
| Client | משתמש באיטרטור כדי לעבור על הנתונים | FriendAnalyzer, PhotoAnalyzer |

המימוש שלנו עומד בכל הדרישות של תבנית Iterator אך תוך שימוש במנגנונים המובנים של C# מה שהופך את הקוד לנקי, יעיל וקל לתחזוקה.

* Sequence Diagram





* Class Diagram

