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

* [**Smart post:** פיצ'ר זה מאפשר למשתמש לפרסם הודעה לחתך מסוים של החברים שלו. החתך נקבע לפי מספר פילטרים שהמערכת מאפשרת כגון: בחירה לפי מין, בחירה לפי מצב משפחתי (רווק, נשוי וגרוש), בחירה לפי גיל (מעל/מתחת לגיל 18), בחירה לפי עיר מגורים. על המשתמש לבחור באיזה מבין הפילטרים הוא רוצה להשתמש (אחד או יותר). לאחר שהמערכת תבצע את הפילטור היא תציג למשתמש את החברים שנבחרו. ברגע שהמשתמש יבחר לפרסם את הפוסט, המערכת תתייג את אותם חברים בפוסט.   
  את הפיצ'ר ניתן למצוא ב DesktopFacebook module תחת התיקייה CustomFeatures תחת התיקייה SmartFilter
* **Friendship match scale:** פיצ'ר זה מאפשר למשתמש לבחור את אחד מחבריו בפייסבוק המופיעים ברשימה ולחשב את מידת התאמתם ממספר בחינות: פרטים אישיים(עיר מגורים,סטטוס זוגי, תפקיד מקצועי אשר בו הם עוסקים בעבודתם הנוכחית), סגנון מוזיקה אהוב ומקומות בילוי מועדפים. המערכת מחשבת את אחוז ההתאמה של חברו עם המשתמש המחובר לאפליקציה ומציגה אותו.  
  את הפיצ'ר ניתן למצוא ב DesktopFacebook module תחת התיקייה CustomFeatures תחת התיקייה FriendshipMatchScale.

**\*תוספת לפיצ'ר בגרסה הנוכחית:** כעת ניתן לראות את התפלגות האחוזים לפי קטגוריות.

### תבנית מס' 1 – Strategy

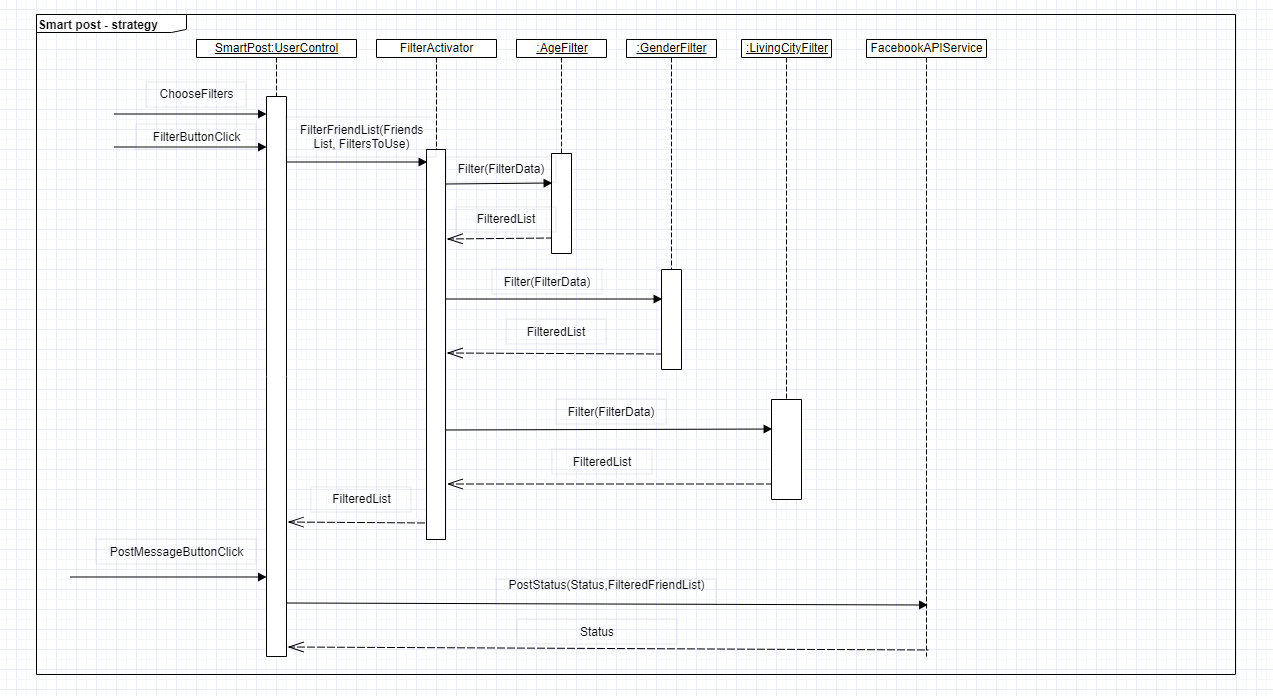
* סיבת הבחירה / שימוש בתבנית:

הסיבה שבחרנו להשתמש ב strategy במקרה זה היא שיש לנו כמה סוגי filter שונים. כל אחד מהם מחזיק אלגוריתם פילטור שונה. מטרתנו היתה שהמחלקה שמפעילה את כל המנגנון תעבוד בצורה דינאמית ותוכל להשתמש באלגוריתמי פילטור שונה ע"פ דרישת המשתמש בזמן הריצה. כמובן שאופן הפעלת ה filter זהה אך ה filter שמופעלים בכל רגע נתון יכולים להיות שונים.

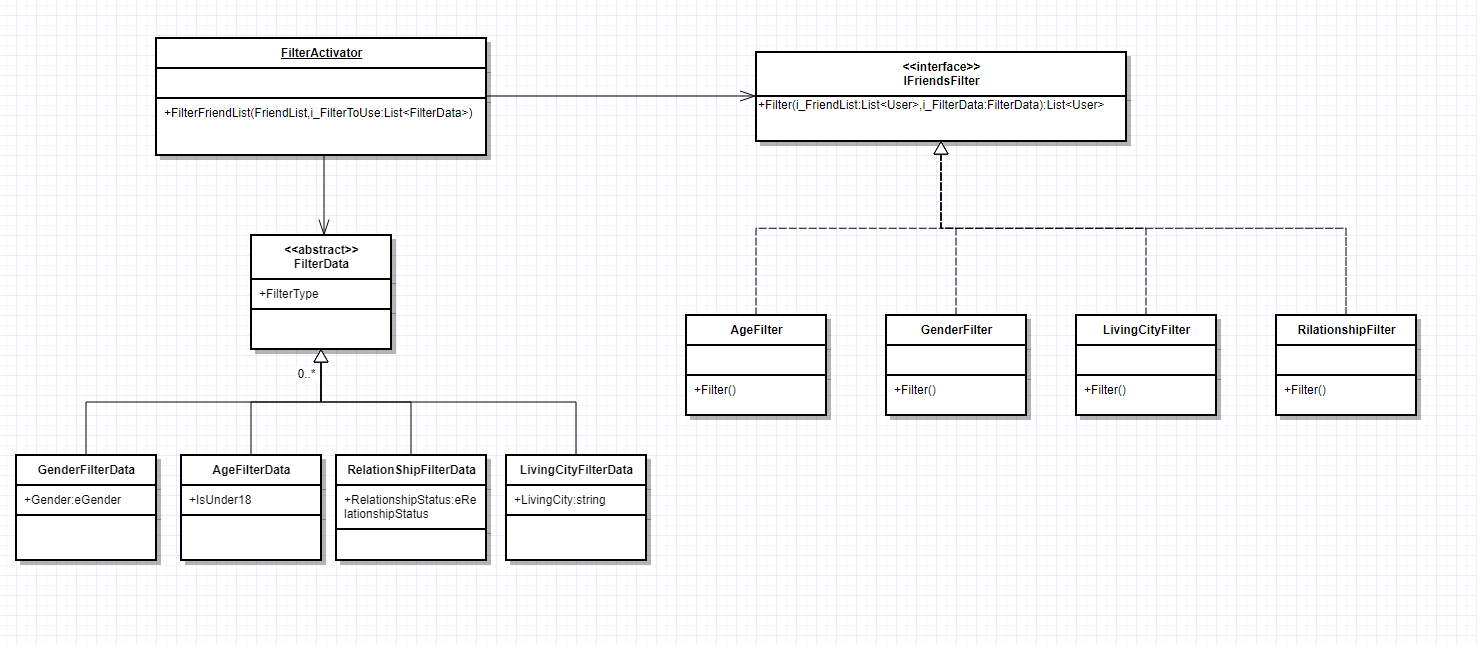
* אופן המימוש:

יצרנו interface בשם IFriendsFilter. ה interface משמש עברנו בתור ה strategy ומכיל את המתודה של האלגוריתם. יצרנו חמישה filter שונים שכל אחד מהם מממש אלגוריתם פילטור שונה. לבסוף יצרנו מחלקה שאחראית להפעלת ה filter הללו ולהחזיר תוצאה בהתאם לאלגוריתם שנבחר. את ה pattern ניתן למצוא ב DesktopFacebook module תחת התיקייה CustomFeatures תחת התיקייה SmartFilter

* Sequence Diagram



* Class Diagram



המחלקה IFriendsFilter מייצגת את ה strategy

המחלקות LivingCityFilter, RelationshipFilter, GenderFilter, AgeFilter מייצגות את ה concrete strategy.

המחלקה FilterActivator מייצגת את ה context.

### תבנית מס' 2 – Observer

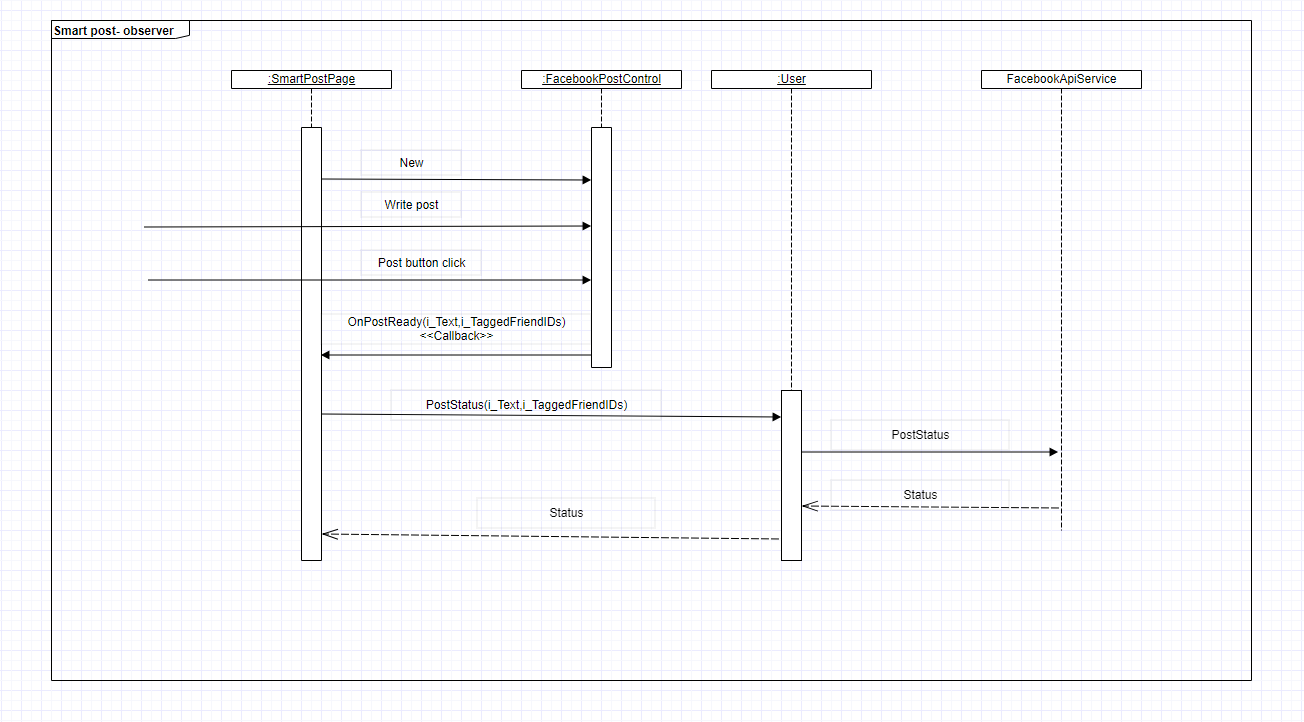
* סיבת הבחירה / שימוש בתבנית:

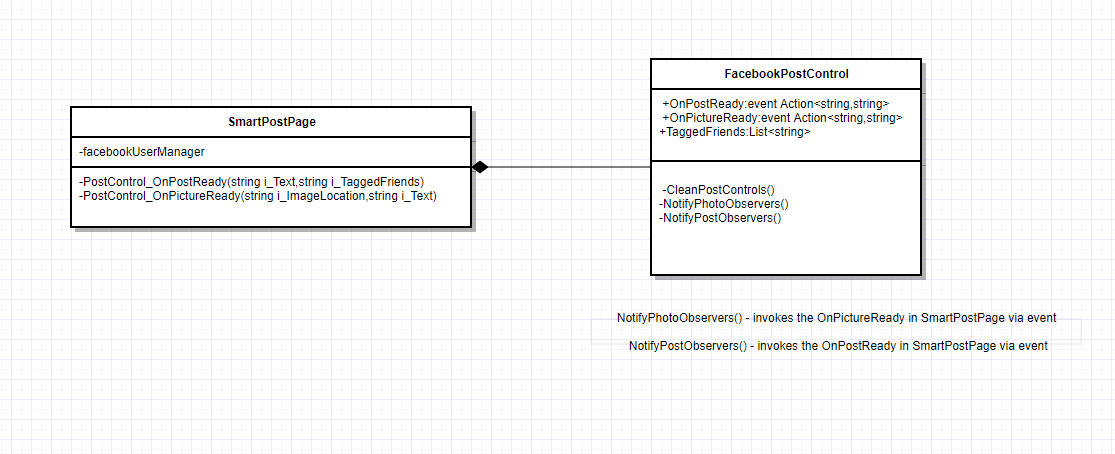
בחרנו להשתמש ב pattern הזה מכיוון שהמסכי ה UI שלנו מורכבים מ layout ראשי ובתוכו נמצאים UserControls שמאגדים בתוכם כמה פקדים ולוגיקת UI מסויימת. ה UserContols מבצעים פעולות שבסופם יש להודיע ל layout הראשי שמכיל אותם על מנת שיוכל להשתנות בהתאם או להפעיל לוגיקה מסויימת. לכן observer היה ה pattern הנכון לשימוש.

* אופן המימוש:

SmartPostPage מכיל בתוכו רכיב UI שהוא UserControl בשם FacebookPostControl שמאגד בתוכו את כל הפקדים הנדרשים לפרסום Post או photo ב Facebook. כמו כן FacebookPostContol מכיל בתוכו שני events אחד בשם OnPostReady והשני בשם OnPictureReady. ה events האלה מהווים אינדיקציה כאשר ה UserControl שלנו מוכן לפרסם ב Facebook. מכיוון שה UserControl שלנו לא מחזיק מופע של ה client של Facebook, היה עלינו לגרום ל Layout שמכיל את ה UserControl להוות Observer ל UserControl על מנת שיוכל לדעת מתי אפשר לפרסם פוסט בפייסבוק. ה layout אכן מכיל facebook client. ה layout שלנו היא המחלקה SmartPostPage. את ה pattern ניתן למצוא במחלקות SmartPostPage ו FacebookPostControl. שנמצאות ב DesktopFacebool module.

* Sequence Diagram



* Class Diagram

המחלה FacebookPostControl משמשת בתור ה subject.  
המחלקה SmartPosPage משמשת בתור ה observer.

### תבנית מס' 3 - Iterator

* סיבת הבחירה / שימוש בתבנית:

בחרנו להשתמש בתבנית זו מפני שרכיב הcalculator שלנו מכיל רכיבי חישוב שתפקידם לחשב את אחוז ההתאמה על פי קטגוריה מסוימת (musicMatchCategory,CheckinsMatchCategory,PrivateDetailsMatchCategory) . רכיבים אלו מחשבים את אחוז ההתאמה ומכילים שני DataMemebers שאותם היינו רוצים להציג בui: MatchValue(ערך אחוז ההתאמה) ובCategoryName (הקטגוריה שבה הושג אחוז ההתאמה בין החבר ליוזר המחובר למערכת). לכן, איננו רוצים להעביר לUi את כל האובייקט הנ"ל אלא רק חלק ממנו, ובנוסף איננו רוצים לאפשר לו לערוך את אחוז ההתאמה ושתהיה לו גישה לאחד מהאובייקטים המוצגים לעיל. נרצה שהוא יוכל לעבור על כלל הקטגוריות מבלי יכולת לשנותן או שתהיה לו גישה למבנה הנתונים שבו האובייקטים הללו מאוכלסים. מכאן הצורך להשתמש בתבנית ITERATOR.

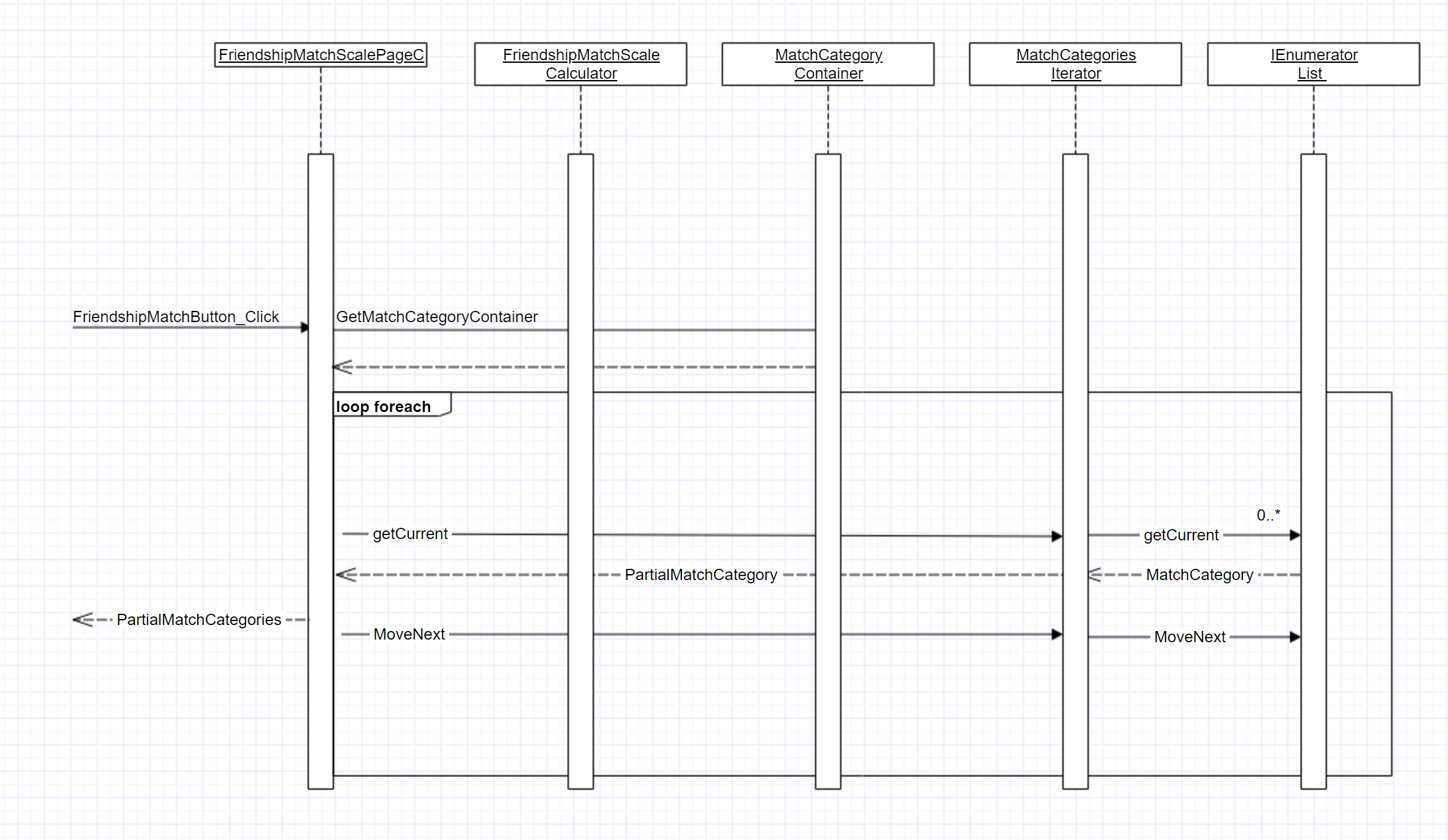
* אופן המימוש:

תחילה, יצרנו class שנקרא: MatchCategoryContainer (CustomFeatures-FriendshipMatchScale) אשר מכיל בתוכו רשימה מסוג MatchCategory -רשימה המכילה בתוכה את כלל האובייקטים אשר תפקידם הוא לחשב אחוז התאמה לפי קטגוריה מסוימת. (האובייקטים הללו יורשים מMatchCategory). מחלקה זו יורשת מIENUMERABLE ולכן בה נממש את GetEnumerator . בנוסף יצרנו מחלקה PRIVATE - MatchCategoriesIterator שתממש את הITERATOR עבור המחלקה MatchCategoryContainer. מחלקה זו יורשת מIenumerator ובכך היה עלינו לממש את כל הפונקציות הדרושות (current,getNext,reset וכו'..).

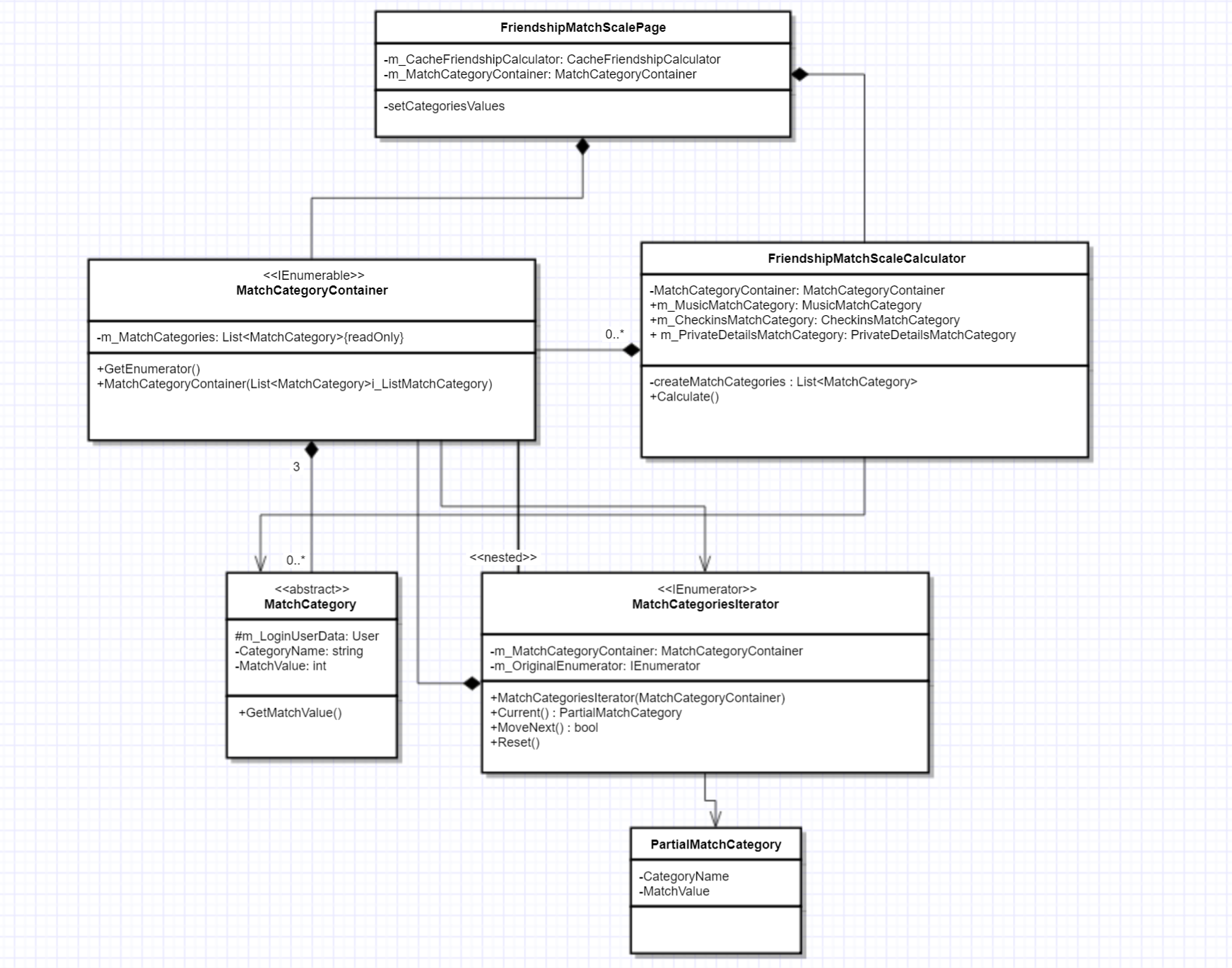
במימוש הפונקציות הללו, השתמשנו בITERATOR המקורי של הList אותו אנו מחזיקים בCONTAINER אך על מנת שלא נחזיר את כל האובייקט , יצרנו מחלקה : partialMatchCategory אשר מכילה אך ורק את הDataMembers הרלוונטים לUi. במימוש Current הפעלנו את הCurrent של האיטרטור המקורי של הlist ויצרנו ממנו אובייקט partailMatchCategory שאותו החזרנו לui. בכך אפשרנו לClient לעבור על כלל סוגי החישובים ולהציג את שם הקטגוריה ואחוז ההתאמה בלבד, מבלי לדעת מהו מבנה הנתונים ומבלי יכולת לבצע שינוי בהם.

* Sequence Diagram

D



* Class Diagram



\*Aggregate – מיוצג על ידי MatchCategoriesContainer – מממש את Ienumerable . ( כלומר על פי הpattern הוא מממש את Iaggregate.

\*ConcreteIterator – מיוצג על ידי MatchCategoriesIterator אשר מממש את Ienumerator (כלומר על פי הPattern את IIterator)

\*Client – מי שמבצע בעצם את השימוש בITERATOR על ידי שימוש בForEach : מיוצג על ידי FriendshipMatchScalePage .