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

* Find Your Best Match  
  פיצ'ר זה מאפשר למשתמש לחפש את השידוך הזוגי הטוב ביותר מבין רשימת חבריו בפייסבוק.  
  המשתמש יוכל לבחור קריטריונים על פיהם הוא רוצה לחפש שידוך (גיל ומגדר).  
  הפיצ'ר יציג למשתמש את חמשת השידוכים הטובים ביותר.  
  השידוכים האופציונאליים נקבעים לפי רמת הפעילות של חבריו בפעילויות שפרסם המשתמש, פוסטים - מעבר על כל הפוסטים של המשתמש ובדיקה מי מהחברים של המשתמש ביצע לייק או תגובה על הפוסט כל תגובה או לייק מזכה את החבר בניקוד, תמונות – מעבר על כל האלבומים של המשתמש והוצאת כל התמונות מכל אלבום ובדיקה האם קיים חבר שעשה לייק או תגובה לאחת התמונות, במידה וכן יעלה הניקוד של החבר, דפים – אנו לוקחים את כל הדפים שהמשתמש עשה להם לייק ובודקים את כל הדפים שהחבר עשה להם לייק, אם קיים דף שגם המשתמש עשה לו לייק וגם החבר עשה לו לייק הניקוד של חבר יעלה.  
  ככל שהחבר/ה יהיו מעורבים יותר בפעילויות שפרסם המשתמש כך דרגת הרלוונטיות לשידוך תהיה גבוהה יותר (בתנאי שהחבר/ה יעמדו בקריטריונים של המשתמש), לאחר קבלת התוצאות ניתן ללחוץ על העיגול התמונה של החבר שיצא וזה מוביל אותך ישר למסנג'ר בפייסבוק עם אותו חבר.

ניתן למצוא בקוד: MyBestMatch.cs

* Popularity  
  פיצ'ר זה מאפשר למשתמש לקבל משוב אודות פעילויותיו בפייסבוק בעזרת חישוב ושקלול רמת פעילותם של חבריו לפייסבוק בפעילויותיו של המשתמש.  
  הפיצ'ר יציג למשתמש בהתאם לבחירתו (פוסטים, תמונות, סרטונים), את הפעילות הטובה ביותר, הגרועה ביותר והממוצע הסטטיסטי.  
  רמת הפופולאריות של הפעילויות נקבעת לפי כמות הלייקים והתגובות שקיבלה כל פעילות.  
  ככל שהפעילות שפרסם המשתמש תקבל יותר לייקים ותגובות כך היא תהיה פופולאריות יותר ולהפך.

ניתן למצוא בקוד: myBestActivity.cs

### תבנית מס' 1 – Command

* סיבת הבחירה / שימוש בתבנית:

בפיצ'ר Popularity שלנו ישנו תפריט בחירה המוצג למשתמש שעל בסיסו מוצגים למשתמש נתונים, רצינו לשפר תפריט זה שיהיה reusabley, maintainability- אם נרצה בעתיד להוסיף עוד אופציות בחירה למשתמש נצטרך רק להוסיף שורת קוד אחת ופונקציה שעושה את הפעולה עצמה בלבד ללא שינוים של הקוד בכלל והכל יתבצע באופן אוטומטי, נוכל גם להחליף את השימוש של כל אחד מהאוביקטים ולהשתמש בכל אחד מהם בנפרד לשימושים שונים.

* אופן המימוש:

המימוש התבצע בצורה הבאה: בנינו מחלקה MenuItemCheckBox שמכילה בתוכה delegate שמחליף את הצורך ב ICommand בתור interface והמחלקה מכילה בתוכה checkbox ואת הפרטים הרלוונטים ליצירת checkbox עם אופצית בחירה על ידי הפוקנציה Selected().

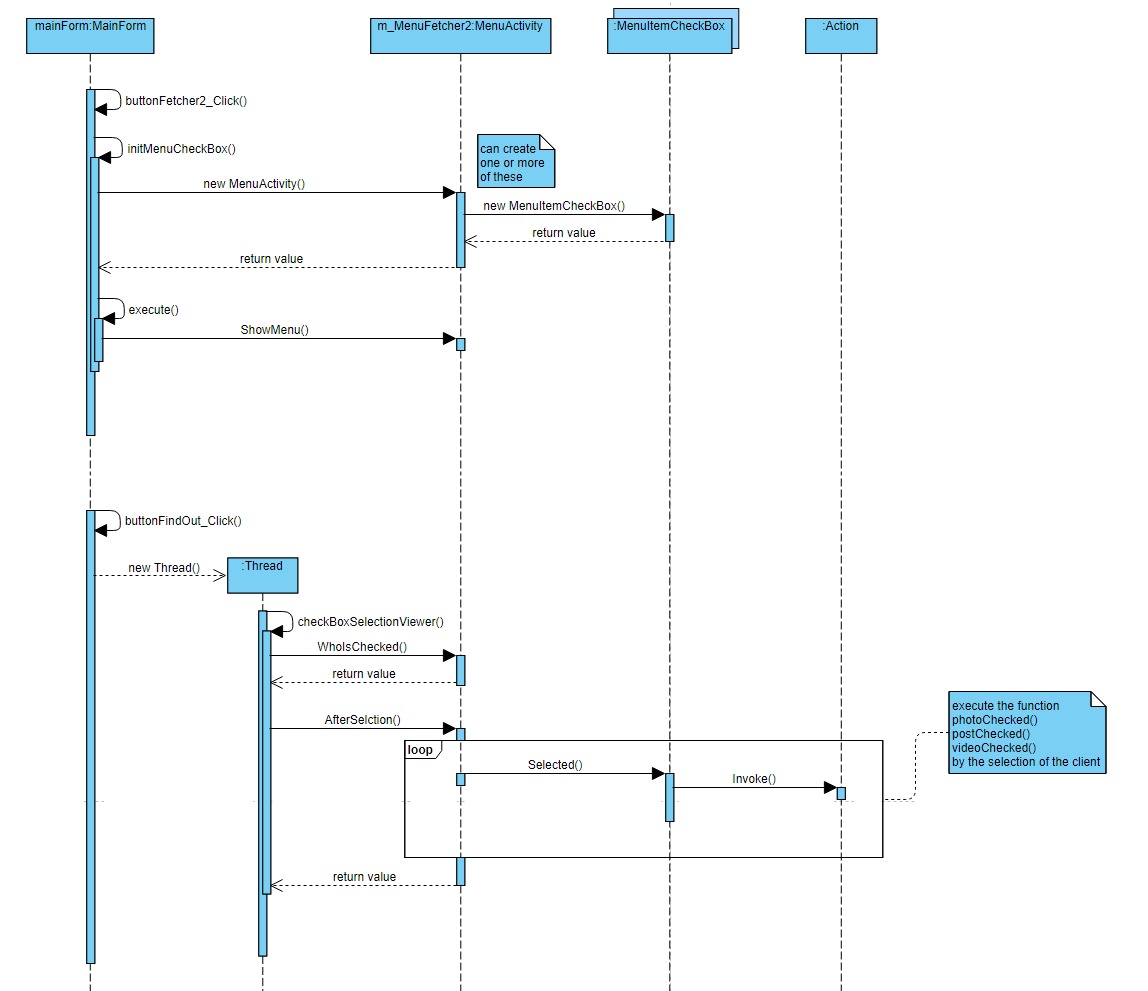
מחלקה נוספת שבינינו MenuActivity שיורשת LIST של MenuItemCheckBox ומממשת 3 מטודות: להציג למשתמש את מה שהוא יצר – showMenu(), לבדוק איזה checkbox הוא בחר מתוך מה שנוצר – WhoIsChecked(), ולבצע את הפעולה הנדרשת לאחר סימון ה checkbox – AfterSelction()ב O(1) בגישה ישירה לפקד שסומן.

המימוש של המחלקה MenuActivity מתבצע ב mainForm ברגע שלוחצים על הכפתור של הפיצ'ר השני נקראת המטודה InitMenuCheckBox() שבה מגדירים את המחלקה MenuActivity ולפיה נוצרים באופן דינמי ה checkbox עם המטודות הרלוונטיות לכל סימון(כאשר המשתמש סימן את פקד זה), לאחר מכן מופעלת המטודה Execute() שמפעילה את המטודה showMenu() של MenuActivity בפעולה זו כל המאפינים של אותו checkbox מתקבלים בו ונוספים לPANEL שנשלח במטודה זו.

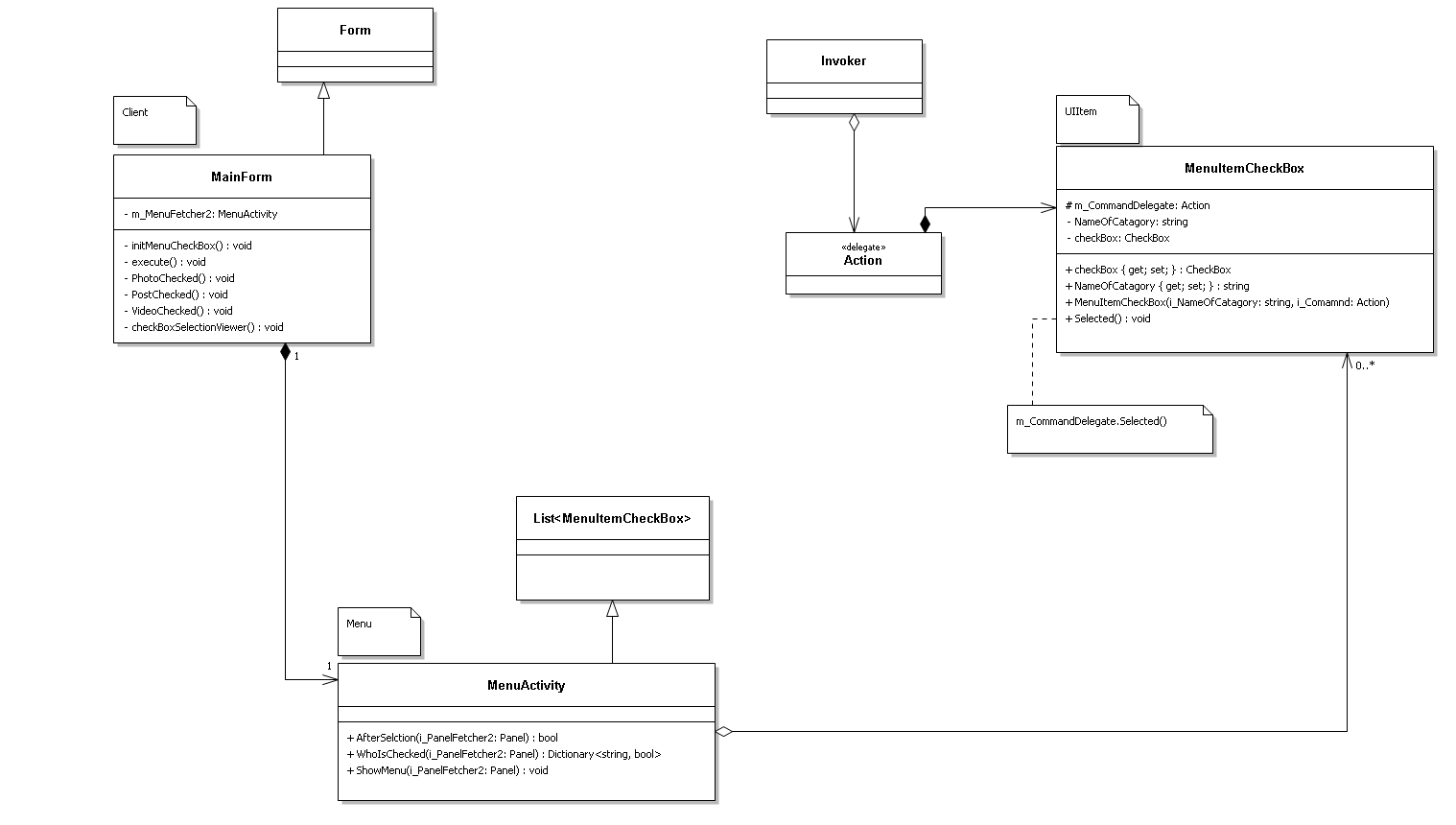
לאחר לחיצה על כפתור ה FIND אנו בודקים איזה checkboxs המשתמש סימן אותם כ ture באמצעות המטודה WhoIsChecked() של המחלקה MenuActivity ומפעילים את AfterSelction() על מי שנבחר ובכך מגיעים למטודה Selected() של MenuItemCheckBox והפעולה הנדרשת מתבצעת.

ניתן למצוא בקוד: MenuActivity.cs, MenuItemCheckBox.cs, MainForm.cs

* Sequence Diagram



* Class Diagram



### תבנית מס' 2 – Strategy

* סיבת הבחירה / שימוש בתבנית:

בחרנו להשתמש ב pattern Strategy בגלל שקיים אצלנו בקוד בפיצ'ר Find Your Best Match חישוב של הניקוד שיש לכל חבר, נוצר לנו LIST של חברים שצריך להחזיר אותו ממוין לMainForm או לכל דורש אחר. הרחבנו את השימוש בפיצ'ר והוספנו אופצית בחירה למשתמש, באמצעות אופציה זו המשתמש יכול לבחור האם לקבל את 5 ההתאמות הכי טובות שלו או לקבל את 5 ההתאמות הכי גרועות שלו. על מנת לאפשר זאת ביצענו את תבנית זאת, שבעקבות שימושה אנחנו יכולים לחסוך בשיכפול קוד, ולגרום לתחזוקתיות טובה יותר. בנוסף לכך נוכל גם להשתמש במיון על LIST כזה או אחר בעוד מקומות בקוד בעקבות הגדרתו כ <T>(reuse) כמובן שנצטרך לתת ערכים ברי השוואה על מנת שזה יסתדר.

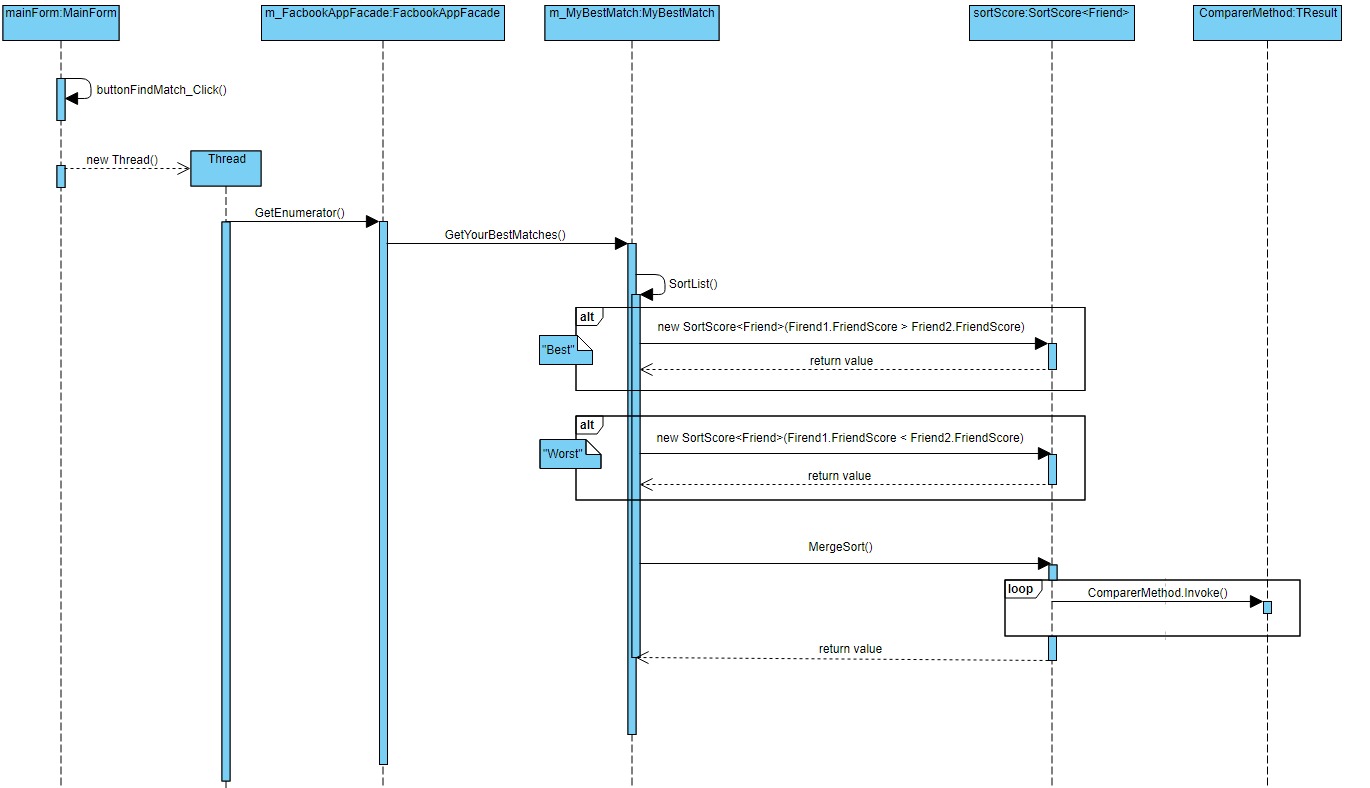
* אופן המימוש:

את המימוש עשינו בצורה הבאה בנינו מחלקה בשם SortScore<T> שמחזיקה delegate Func אשר מקבל שתי T ומחזיר Bool, במחלקה זו בנינו מתודה mergeSort אשר ממינת לנו את המערך ב o(nlogn), בפונקצית המיון הגדרנו את האסטרטגיה שלנו בעזרת invoke

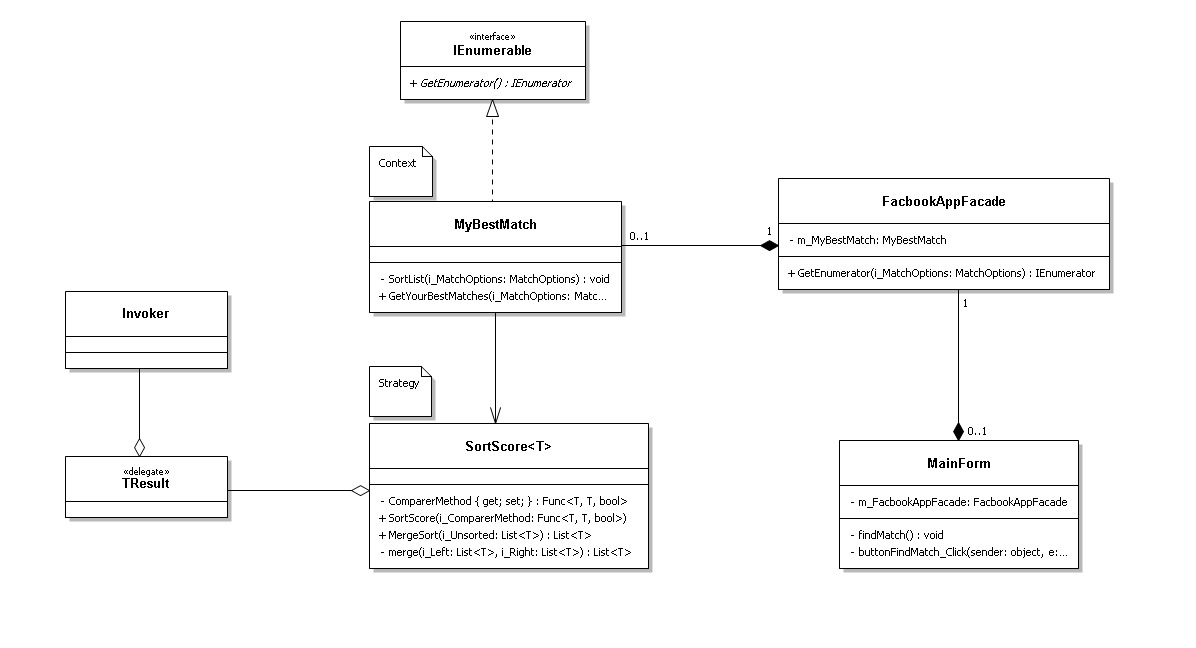
שבודק לפי איזה אסטרטגיה לפעול האם למיין בסדר עולה או למיין בסדר יורד על פי בחירת המשתמש. את המימוש של מחלקה זו עשינו במחלקה MyBestMatch שלאחר כל החישובים נותרנו עם LIST של חברים לא ממוין שמכיל את כל החברים של המשתמש ואת הניקוד שצברו במהלך הבדיקה, בפונקציה SortList() בודקים מה המשתמש בחר ב MAINFORM על ידי לקיחת המאפינים והשמתם במחלקה MatchOption ולפי זה ממינים את המערך, באמצעות יצירה של אוביקט מסוג SortScore שהוא במקרה הזה FRIEND, ומגדירים לו את שיטת הפעולה הרלוונטית לפי הביקוש של המשתמש, את התוצאה שומרים ב LIST והמשתמש לוקח אותה באמצעות ITERATOR שאותו נסביר בהמשך.

מיקום בקוד: SortScore.cs, MyBestMatch.cs

* Sequence Diagram



* Class Diagram



### תבנית מס' 3 – Iterator

* סיבת הבחירה / שימוש בתבנית:

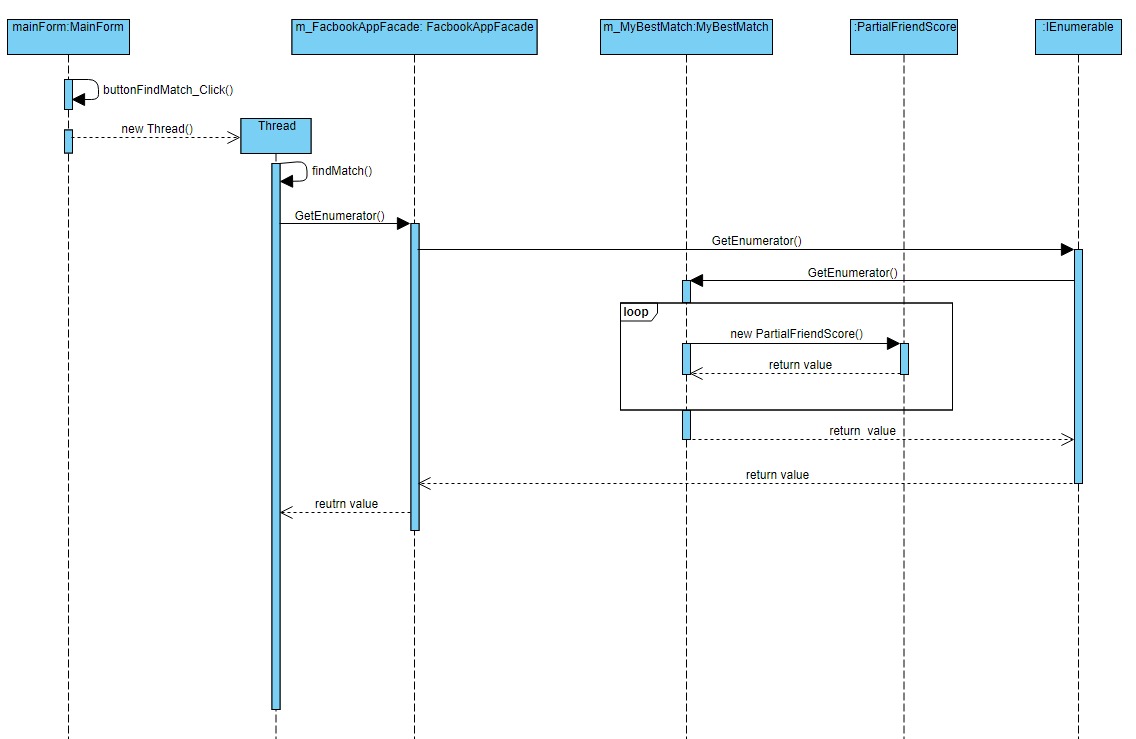
בחרנו להשתמש ב Pattern Iterator על הפיצ'ר הראשון - Find Your Best Match אשר מחזיר List<Friends> שמכיל רשימה של חברים פוטנצילים לשידוך, עם רשימה זאת המשתמש יכול לבחור לעשות כל מיני דברים כשקשורים להצגתם, או שמירתם בבסיס נתונים כזה או אחר. רצינו לא לחשוף את מבנה הנתונים list למשתמש, אך לתת לו לעבור על התוצאה כדי שיוכל לבצע פעולות הצגה או שמירה שגם עליהם ביצענו הגבלה, ונתנו למשתמש מחלקה שנותנת לו למשוך חלק מהמאפיינים של האוביקט המוחזר ולכן מימשנו את תבנית זאת.

* אופן המימוש:

על מנת לממש את ה Pattern על הפיצ'ר Find Your Best Match אשר מחזירה רשימה של החברים שהכי מתאים למשתמש על פי חישוב מסוים. המחלקה MyBestMatch יורשת מ IEnumerable – מחלקה זאת בעצם עושה את כל החישוב הרלוונטי כאשר יוצרים אותה, ומייצרת List<Friend> שמכיל באופן ממוין את רשימת החברים שהכי מתאים לשידוך למשתמש, בעקבות הורשה זאת אנו חייבים לממש את המתודה GetEnumerator(), במתודה זאת, אנו עוברים על גדולו של הLIST ועושים TEST שבסופו של דברים יחזיר למשתמש רק מספר מצומצם של חברים שמתאימים ולא את כולם, את מספר זה מי שמשתמש במחלקה זו יכול להגדיר, לאחר TEST זה באמצעות yield return נוצרת המחלקה PartialFriendScore שמאפשרת למי שמשתמש במחלקה MyBestMatch לקבל אך ורק את השם המלא של החבר ואת תמונת הפרופיל שלו. על מנת להמשיך ולקיים את התבנית FACADE שאותו עשינו בעבר, את המימוש למשתמש אנחנו מבצעים דרך המחלקה של FAÇADE בפונקציה GetEnumerator() אשר תפקידה הוא לגרום ליצירה של אוביקט מסוג MyBestMatch ובכך לגרום לחישוב הנתונים הרלוונטים ושמירתם במבנה נתונים (List) באותו מחלקה שהופך לנו ל IEnumerator אותו אנו שולחים ל MainForm שם אנחנו מקבלים את ה IEnumerator בפונקציה findMatch() ושם בודקים האם קיימת תוצאה כל שהיא ואם כן עוברים לפונקציה updatePictureBoxAndLabelOfMatches() שעוברת על מבנה הנתונים ומציגה למשתמש את הפרטים הרלוונטים בעזרת המחלקה PartialFriendScore

מיקום בקוד: MyBestMatch.cs, PartialFriendScore.cs, MainForm.cs, FacbookAppFacade.cs

* Sequence Diagram



* Class Diagram

