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

1. **פיצ'ר מרכזי**: פיצ'ר חיפוש מסלול בין אירוע שעתיד להתרחש ובין כתובת שהמשתמש מזין. המשתמש בוחר אירוע מתוך רשימת האירועים שלהם עשה Attending, וע"י יצירת URL אנו פונים לשרתי Google Maps ומביאים את המסלול האופטימלי. ממומש במחלקה FormMain.
2. **פיצ'ר מרכזי**: פיצ'ר הצגת פוסטים ע"י פילטור לפי שנה או לפי סוג הפוסט או לפי מיון עולה/יורד של כמות התגובות על הפוסטים. ממומש במחלקה FormMain.
3. **פיצ'ר משני**: יצירת דף בעל שש תמונות שנבחרות באופן רנדומלי מכלל התמונות שנלקחות מכלל האלבומים של המשתמש. ממומש במחלקה FormMain.
4. **פיצ'ר משני**: צנזור פוסטים לפי מילה שהמשתמש מזין. המשתמש מזין מילה שאותה רוצה לצנזר וע"י מעבר על גבי כל הטקסטים הדינאמיים באפליקציה המילה מצונזרת ומוחלפת בכוכביות כאורך המילה שנבחרה. ממומש במחלקה FormMain.
5. **פיצ'ר משני**: פתיחת חיפוש בYoutube של אחד משמות העמודים מקטגוריית מוזיקה שלהם המשתמש עשה לייק. לאחר לקיחת רשימת העמודים הנ"ל בוחרים את אחד משמות העמודים (אחד האמנים) באופן רנדומלי ומבצעים שאילתה לשרתים של Youtube על מנת להציג את תוצאות החיפוש בתצורת סרטונים. ממומש במחלקה FormMain ומשתמש במחלקה YouTubeVideo.
6. **פיצ'ר משני מתרגיל 2**: המשתמש בוחר כיצד ייראה עמוד ה-Pages באפליקציה, בכך שהוא בוחר מאיזה אובייקט יורכבו הרשימות, בפרט ListBox או ComboBox. על מנת למממש את הפיצ'ר הזה באופן נוח ו-maintainabily, השתמשנו בתבניות Adapter ו-Factory Method.

### תבנית מס' 1 – Iterator

* סיבת הבחירה / שימוש בתבנית:

אנחנו רוצים לאפשר ל-FormMain (ה-client) לסרוק את אוסף הפוסטים של המשתמש המחובר לאפליקציה ב-O(n) ולא לחשוף באיזה מבנה נתונים מוחזקים הפוסטים. בנוסף, כחלק מפיצ'ר פילטור הפוסטים לפי שנה או לפי סוג הפוסט, כאשר המשתמש בוחר בפילטר מסוים, נרצה גישה רק לפוסטים הרלוונטים שעוברים את הסינון כש- FormMain יסרוק אותם ולא נרצה לשכפל קוד כשנעשה זאת.

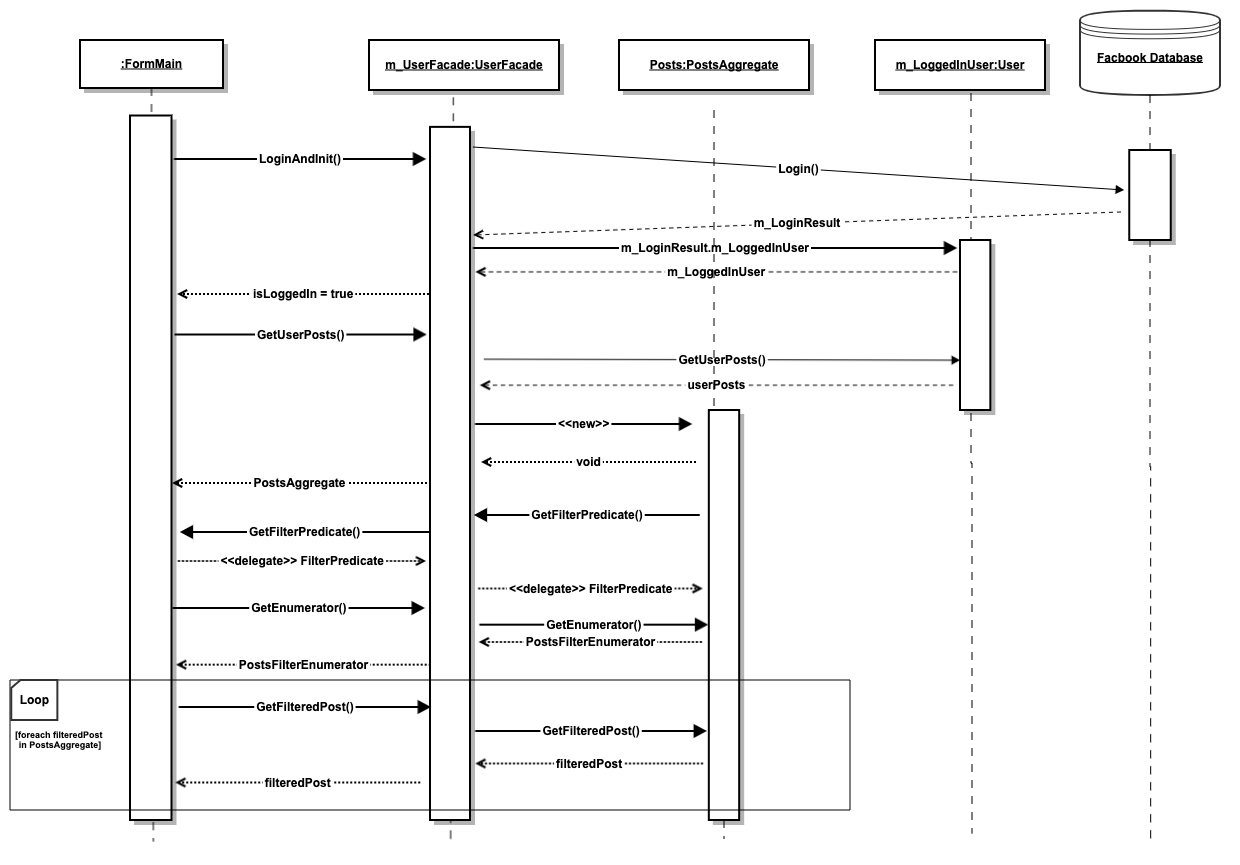
באופן זה, אם בעתיד נרצה להחליף את מבנה הנתונים בו מוחזק הפוסטים, נוכל לעשות זאת בקלות וזה לא יגרור שינוי במקומות אחרים בקוד חוץ מב-PostsAggregate.

* אופן המימוש:

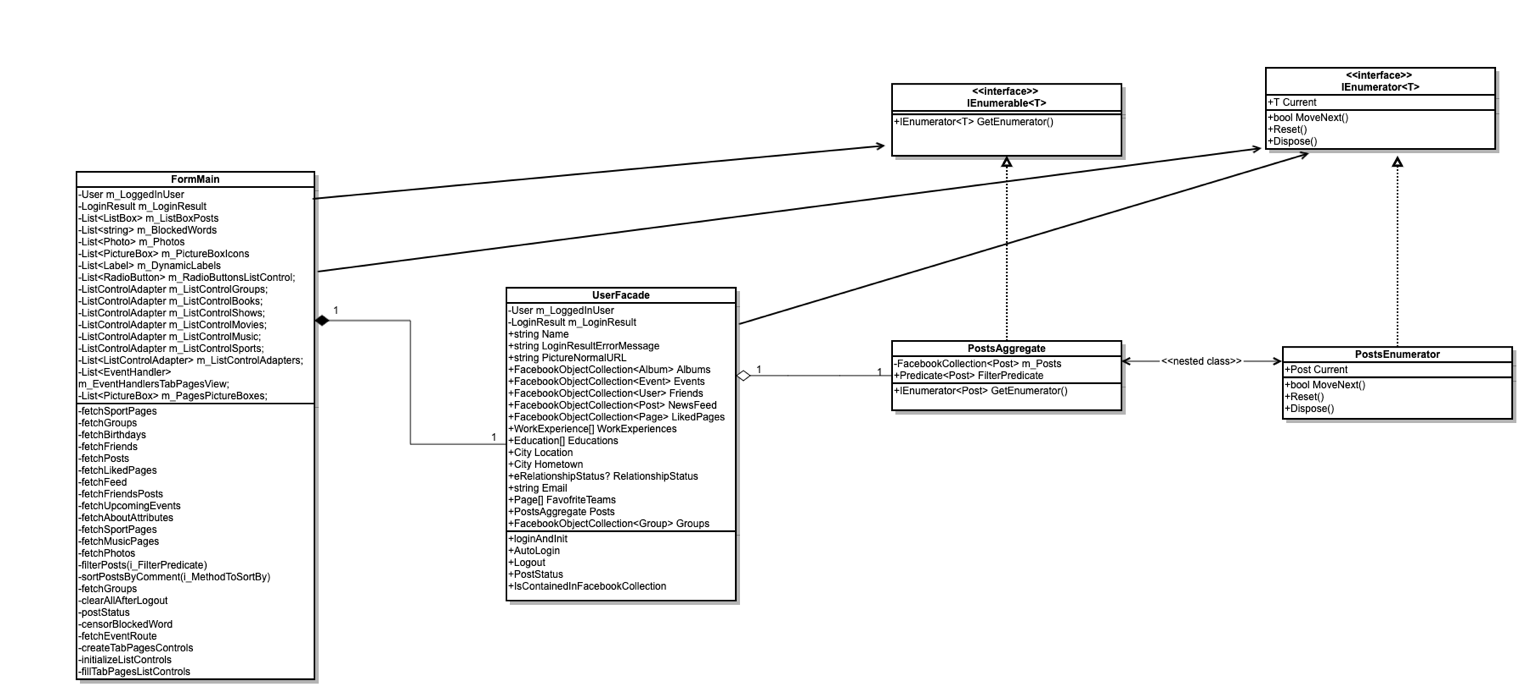
מימשנו מחלקה בשם PostsAggregate שהיא משמשת כ-Aggregate בתבנית ומממשת את הממשק IEnumerable. המחלקה מחזיקה אוסף של פוסטים כשדה ו-property של Predicate שעוטף את המתודה לפיה נפלטר את הפוסטים (מקבלת post ומחזירה bool) ואותו נקבל מה-client. המתודה GetEnumerator תחזיר filter Iterator בעזרת שימוש ב- foreach על אוסף הפוסטים שמפעיל על כל פוסט את ה-Predicate ומחזיר רק את הפוסטים שעוברים את הסינון בעזרת שימוש ב-yield return.

ה-client, FormMain, מחזיק כשדה אובייקט מסוג UserFacade במשתנה m\_UserFacade שמחזיק כ-property אובייקט PostsAggregate במשתנה Posts +שבו יוחזקו כל הפוסטים של המשתמש לאחר שקיבל אותם משרתי פייסבוק. כאשר נקבל את הפוסטים של המשתמש מפייסבוק, הם ייכנסו ל-Aggregate שנמצא ב-UserFacade ונשתמש בו במתודה fetchPosts שבה נסרוק את הפוסטים בפעם הראשונה ונכניס את כולם לListBoxes המתאימים ללא סינון (כאשר FilterPredicate לא מוגדר, כלומר הוא null, לא מתבצע סינון וכל הפוסטים מוחזרים ב-Iterator) בעזרת קריאה למתודה GetEnumerator כדי לקבל את ה-Iterator ואז קריאה למתודות MoveNext ושימוש ב-property Current של Iterator עבור כל ListBox רלוונטי בדף הבית ולבסוף קריאה ל-Dispose על ה-Iterator, ובכך נסרוק את אוסף הפוסטים ללא גישה למבנה הנתונים הקונקרטי שבו הם מוחזקים. בנוסף, במתודה sortPostsByComment, נסרוק את אוסף הפוסטים בעזרת ה-Iterator עם שימוש ב-foreach (שמשתמש מאחורי הקלעים בMoveNext, Current, Dispose) כאשר המשתמש יבחר לסנן את הפוסטים בסדר עולה\יורד. במתודה filterPosts נקבל כפרמטר את ה-Predicate לפי הבחירה של המשתמש ונגדיר אותו ב-FilterPredicate ב- PostsAggregate, ואז בעזרת ה-Iterator, נסרוק את אוסף הפוסטים המסוננים ונקבל רק את אלה שעברו את הפילטר.

* Sequence Diagram



* Class Diagram



**Client – FormMain (uses UserFacade in order to use the iterator)**

**Aggregate – IEnumerable<T>**

**ConcreteAggregate - PostsAggregate**

**Iterator – IEnumerator<T>**

**ConcerteIterator – PostsEnumerator (created on the fly using yield return)**

**StaticFactoryClass – ListControlFactory**

### תבנית מס' 2 – Strategy

* סיבת הבחירה / שימוש בתבנית:

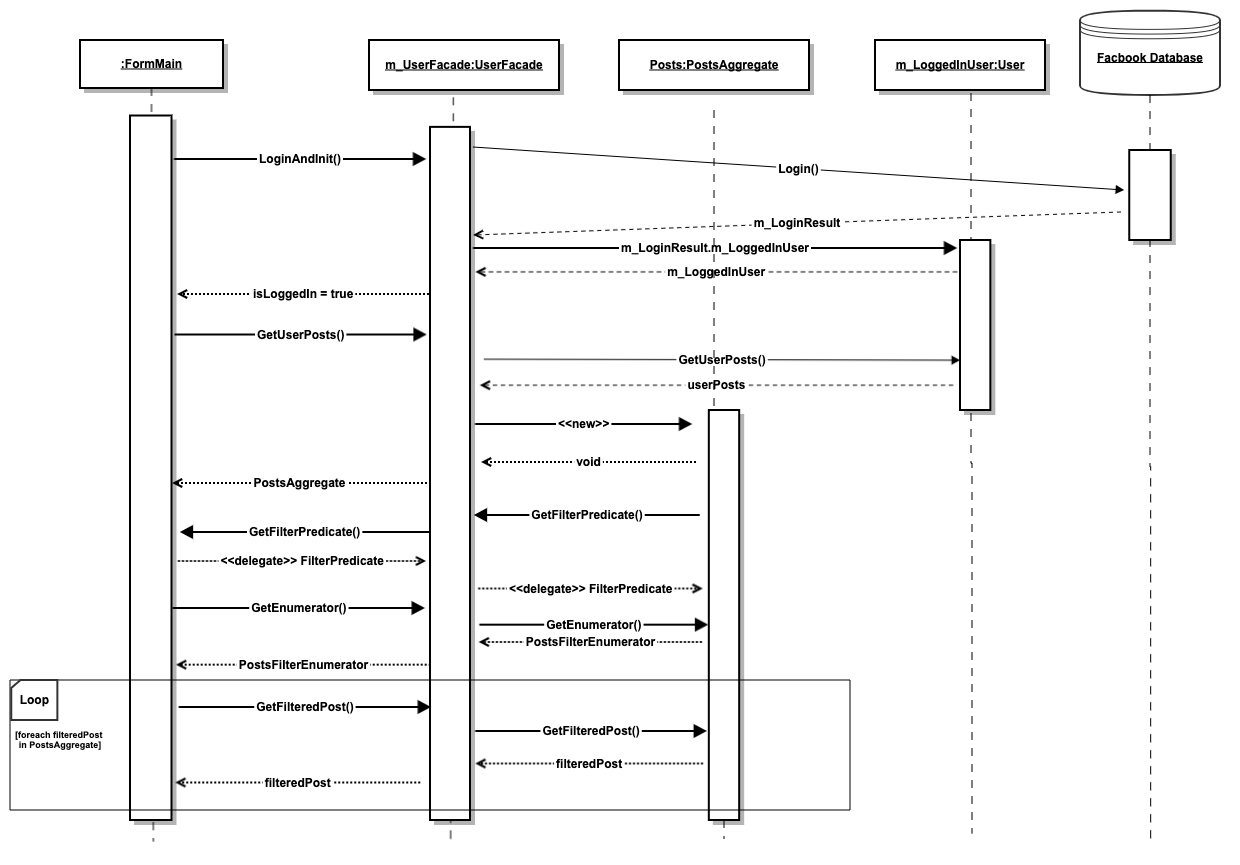
כחלק מפיצ'ר פילטור הפוסטים לפי שנה או לפי סוג הפוסט, כאשר המשתמש בוחר בפילטר מסוים, נרצה גישה רק לפוסטים הרלוונטים שעוברים את הסינון כש- FormMain יסרוק אותם ולא נרצה לשכפל קוד כשנעשה זאת. נשים לב שאם נעשה את זה בכמה מתודות שונות יהיה הרבה שכפול קוד וזה לא יהיה תחזוקתי, בכל פעם שנרצה להוסיף אפשרות סינון חדשה נצטרך למממש מתודה חדשה שתממש את הסינון ותחזור על הרבה קוד. נרצה מנגנון maintainabily שאליו נזריק את ההתנהגות הפשוטה והמתחלפת של מתודת הסינון בזמן ריצה ורק במקום המתאים, מבלי לשנות את הקוד או לשכפל אותו עבור כל סינון. באופן זה, אם בעתיד נרצה להוסיף לאפליקציה אפשרויות סינון נוספות, נוכל רק להגדיר את הסינון עצמו בשורת קוד ולהעביר אותו למנגנון ולא נצטרך לשכתב את כל המסביב.

* אופן המימוש:

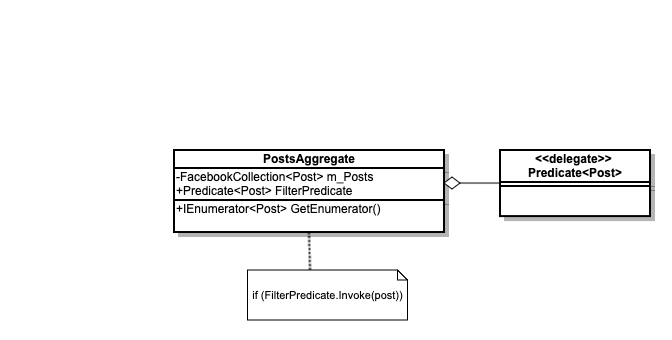
החלק של ההתנהגות המתחלפת במנגנון, כלומר ה-Injection Point , הוא בדרך כלל פשוט וקצר, ולכן במקום להשתמש בממשק עם מתודה אחת ולעשות איתו קומפוזיציה, החלטנו להשתמש ב-Strategy Method, כלומר העברת delegate של מתודה שתמממש את ההתנהגות המתחלפת של אפשרות הסינון. המנגנון שלנו נמצא במחלקה PostsAggregate שיצרנו כחלק מתבנית Iterator. היא מחזיקה Predicate של מתודת הסינון, ובמתודה GetEnumerator היא סורקת את אוסף הפוסטים ומחזירה רק את הפוסטים שעברו את הסינון של ה-Predicate בעזרת שימוש ב-yield return בתוך foreach. כלומר מימשנו filter Iterator שבעזרתו ניתן לסרוק רק את הפוסטים שעברו סינון וניתן בקלות לשנות בו את אופן הסינון. המתודה filterPosts שב-FormMain, תקבל מהמשתמש את אופן הסינון, שיתקבל כ-Predicate שמצביע למתודה. כל אחד מה-click event handlers של הכפתורים המתאימים בטופס (שורות 807 – 923 בקוד) קורא ל-filterPosts עם המתודה המתאימה לכפתור שממומשת עם Lambda Expression. filterPosts תשים ב-property FilterPredicate של m\_PostsAggregate את ה-Predicate שקיבלה, ותשתמש ב-filter Iterator שלו כדי לסרוק עם foreach את אוסף הפוסטים המסוננים, כאשר בכל איטרציה היא תקבל רק פוסט שעבר את התנהגות הסינון שהוזרקה למנגנון, כנדרש.

מימשנו Strategy Method גם פעם נוספת במחלקה PostsSorter במתודת המנגנון Sort שלה קוראת המתודה sortPostsByComment במחלקה FormMain כחלק מפיצ'ר מיון הפוסטים של המשתמש לפי סדר עולה\יורד של תגובות על מנת להזריק למנגנון המיון את סדר המיון ללא שכפול קוד. בדיאגרמות נתאר רק את המימוש הראשון של התבנית.

* Sequence Diagram



* Class Diagram



**Context – PostsAggregate**

**Strategy – Predicate<Post>**

### תבנית מס' 3 – Observer

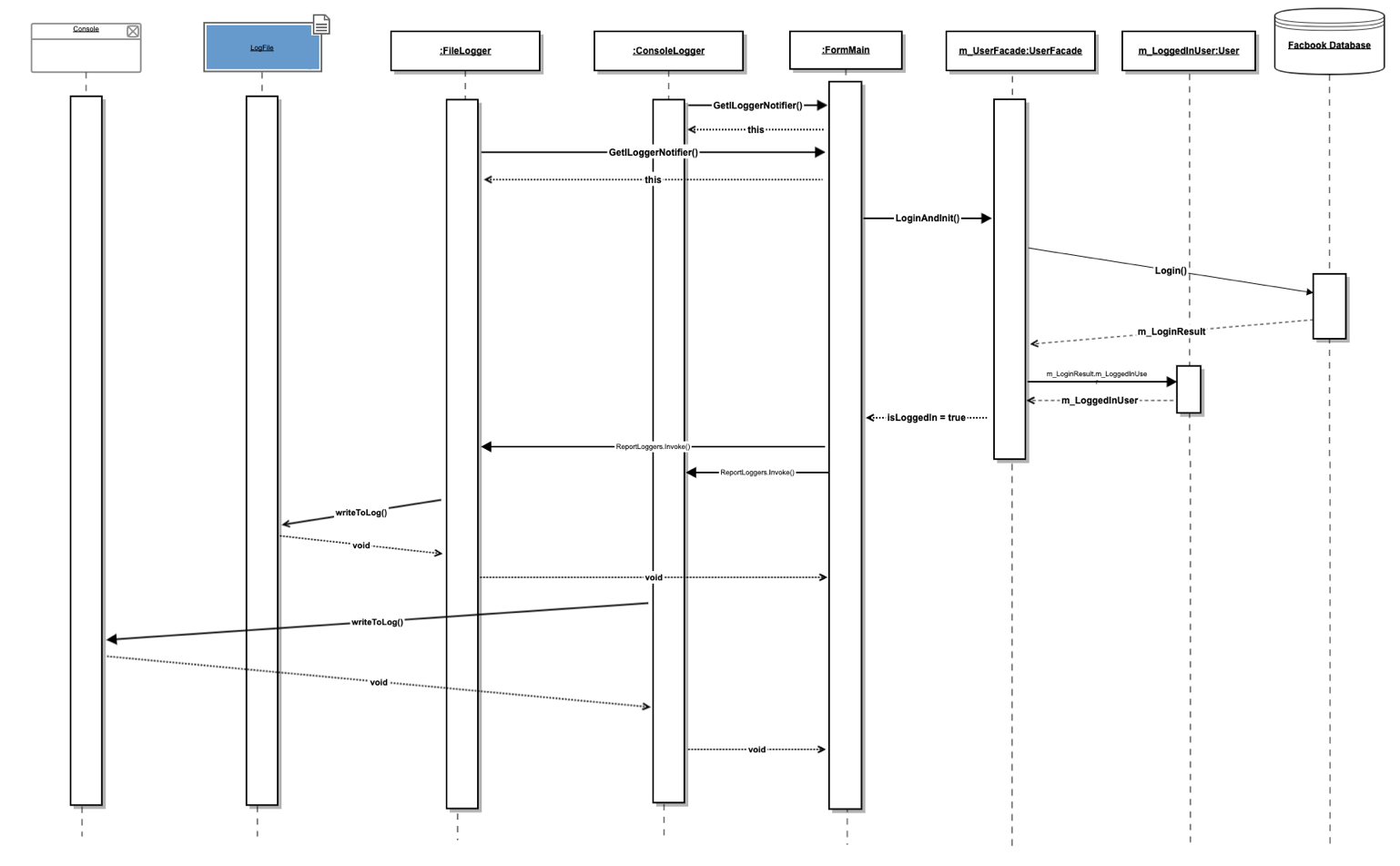
* סיבת הבחירה / שימוש בתבנית:

מימשנו מערכת של Logger שכותבת ל-Console או לקובץ טקסט כל דבר שהמשתמש עושה על מנת לעקוב אחר תחזוקת המערכת. כלומר הרכיב FormMain צריך להודיע לכל ה-Loggers הרלוונטים כאשר מתבצעת פעולה שדורשת תיעוד. כלומר FormMain רוצה להפעיל מתודה של רכיבים אחרים שיקשיבו לו, מבלי להכיר את הרכיבים הקונקרטים שמקשיבים, כלומר להמנע מהכרות ציקלית בין FormMain ל-Loggers. נרצה פתרון תחזוקתי שלא דורש שכפול קוד וכך אם בעתיד נממש Loggers נוספים במערכת, שמתעדים בצורה אחרת למשל, לא נצטרך לשנות את הקוד הקיים, אלא רק להוסיף את הרכיב החדש לרשימת המאזינים.

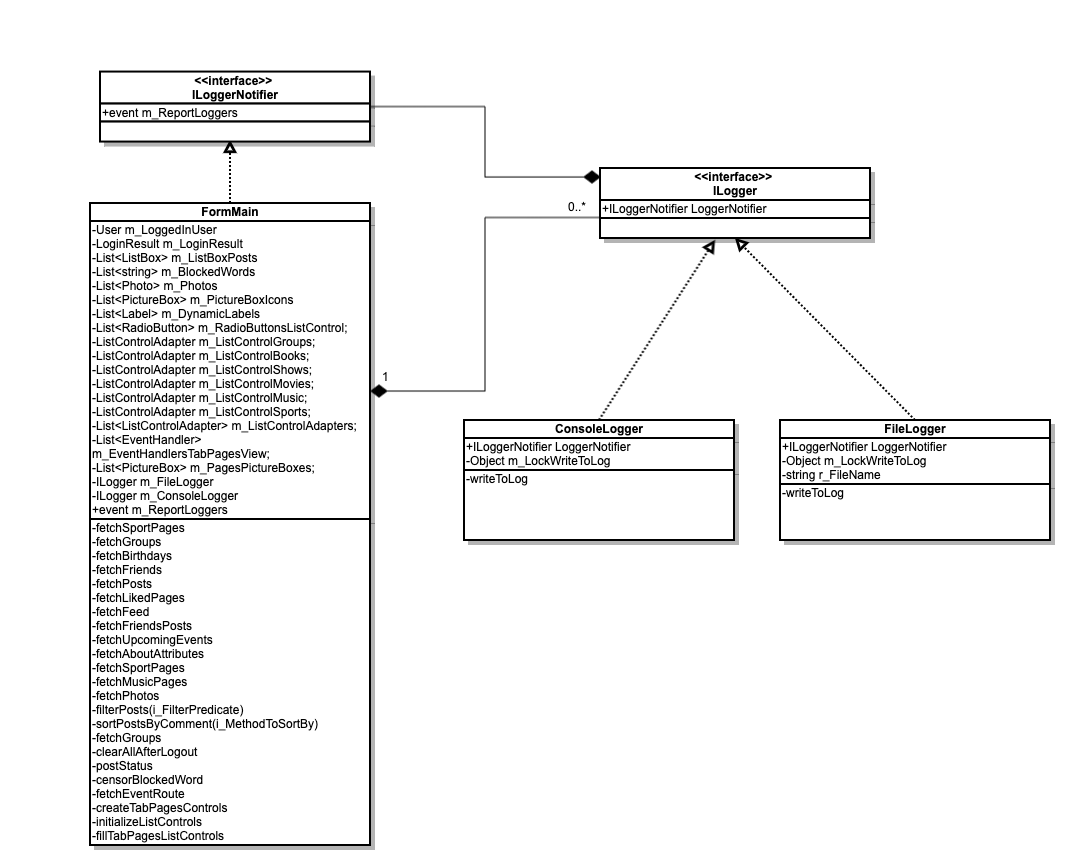
* אופן המימוש:

מימשנו את Pattern Observer בעזרת delegate ובשימוש בפיצ'רים של C# and .NET. בנינו ממשק חדש בשם ILoggerNotifier ש-FormMain מממש ממנו, בממשק יש הכרזה אחת של event Action<string> m\_ReportLoggers שישמש כמעטפת למתודות השונות של המאזינים שיירשמו לאירוע. המילה event גורמת לכך שכל Logger יוכל להוסיף מתודה שלו או למחוק מתודה שלו מהאירוע (גם אם הוא null), אבל כל דבר אחר לא חשוף לו. בנינו ממשק נוסף בשם ILogger אותו מממשות כרגע 2 מחלקות: ConsoleLogger, FileLogger. הממשק מכריז על property מסוג ILoggerNotifier על מנת של-loggers תהיה אפשרות להרשם לאירוע. בשני ה-loggers יש מתודה private void writeToLog(string i\_Message) שמתעדת, כל אחת בדרכה, את הפעולות שמתבצעות באפליקציה בזמן ריצה (אחת כותבת ל-Console והשניה לקובץ טקסט חיצוני). שניהם מקבלים ב-constructor משתנה של ILoggerNotifier ונרשמים לאירוע m\_ReportLoggers שלו עם המתודה writeToLog שלהם. כעת FormMain יודיע לכל המאזינים על כל פעולה שהתבצעה במערכת בעזרת הפעלת האירוע עם הודעה מתאימה. למשל במתודה loginAndInit אם המשתמש מתחבר בהצלחה לאפליקציה, האירוע יופעל וכך יופעלו כל המתודות שנרשמו בעזרת שורת הקוד m\_ReportLoggers?.Invoke("User logged in.");. בעזרת סימן השאלה לא נצטרך לבדוק אם האירוע null.

* Sequence Diagram



* Class Diagram



**ISubject - ILoggerNotifier**

**ConcreteSubject – FormMain**

**IObserver - ILogger**

**Concrete Observer – Consoler Logger, FileLogger**