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

* פיצ'ר תמונות אהובות:

בפיצ'ר זה, כאשר המשתמש רוצה לצפות באלבום כלשהו ששיך לו, האפליקציה בודקת ומציגה את של 3 התמונות המובילות (3 התמונות עם כמות הלייקים הכי גבוהה באלבום), ומתוך 3 תמונות אלה המשתמש יכול באמצעות לחיצת כפתור להפוך תמונה נבחרת לתמונת פרופיל.

* פיצ'ר בילויים (הערה – במטלה הראשונה הפי'צר רק הציג את 10 המקומות המובילים בהם חבריו ביצעו check-in):

האפליקציה מאפשר למשתמש לבחון היכן חבריו אוהבים לבלות, ובכך לקבל חוות דעת מהם היכן אולי כדאי לו לבקר. כעת, בנוסף לכך, ישנו לוח hangout שבו ניתן לפרסם הצעה לבילוי ולהצטרף להצעות קיימות בין החברים.

### תבנית מס' 1 – Strategy

* סיבת הבחירה / שימוש בתבנית:

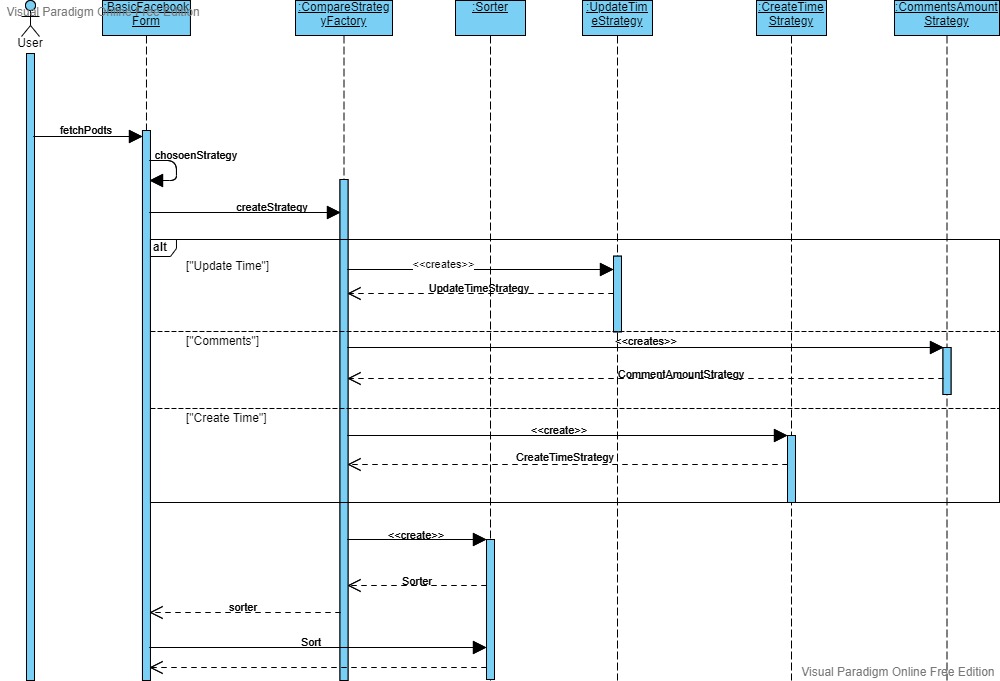
הוספתי אפשרות למיין את הפוסטים של המשתמש בשלושה אופנים שונים: זמן יצירה, זמן עדכון וכמות תגובות. תבנית strategy מאפשרת לעשות שימוש במחלקה הממיינת (sorter) באופן שבו ניתן לעשות בה שימוש חוזר, כיוון שהתנאי למיון אינו חלק מהמחלקה, אלא נמצא במחלקה אחרת (strategy). כך, אם בעתיד נרצה להוסיף סוגי מיונים שונים לתוכנית – יהיה אפשר פשוט לממש את הממשק ההשוואה ולספק אותו למחלקה הממיינת.

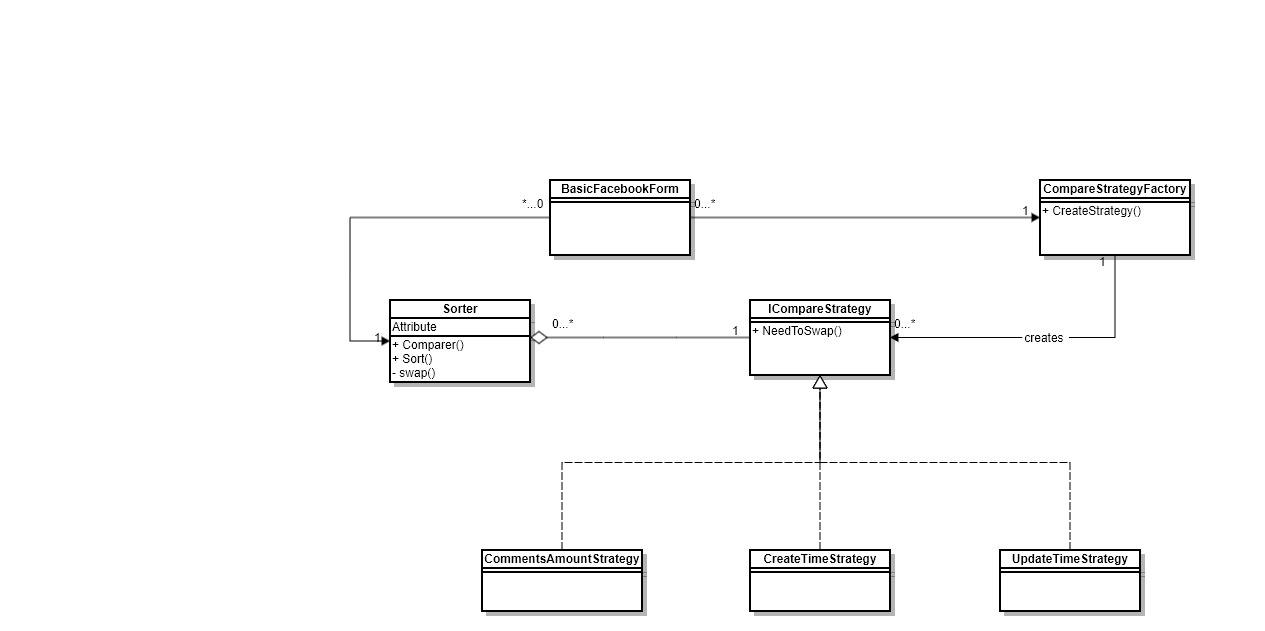
* אופן המימוש:

עבור מימוש תבנית זו, יצרתי מחלקה Sorter (בתפקיד ה-context), אשר עושה שימוש במחלקה המממשת את הממשק ICompareStrategy (בתפקיד ה-Strategy). את ממשק זה, מממשות 3 מחלקות – CommentsAmountStrategy, CreateTimeStrategy, UpdateTimeStrategy (בתפקיד המחלקות הקונקרטיות של ה-Strategy). בנוסף, ישנה מחלקה (CompareStrategyFactory) עם factory method שיוצרת אסטרטגיה לפי בקשת המשתמש. כלל מחלקות אלה ממומשות בשכבה הלוגית של האפלקיציה, וניתן לראות את השימוש בהן בשכבת ה-UI במתודה fetchPosts.

המתמש מזמן כיצד הוא רוצה למיין את הפוסטים, ולפי כך נוצרת האסטרטגיה במחלקת ה"מפעל" ומובאת למחלקה הממיינת.

* Sequence Diagram



* Class Diagram

### תבנית מס' 2 – Iterator

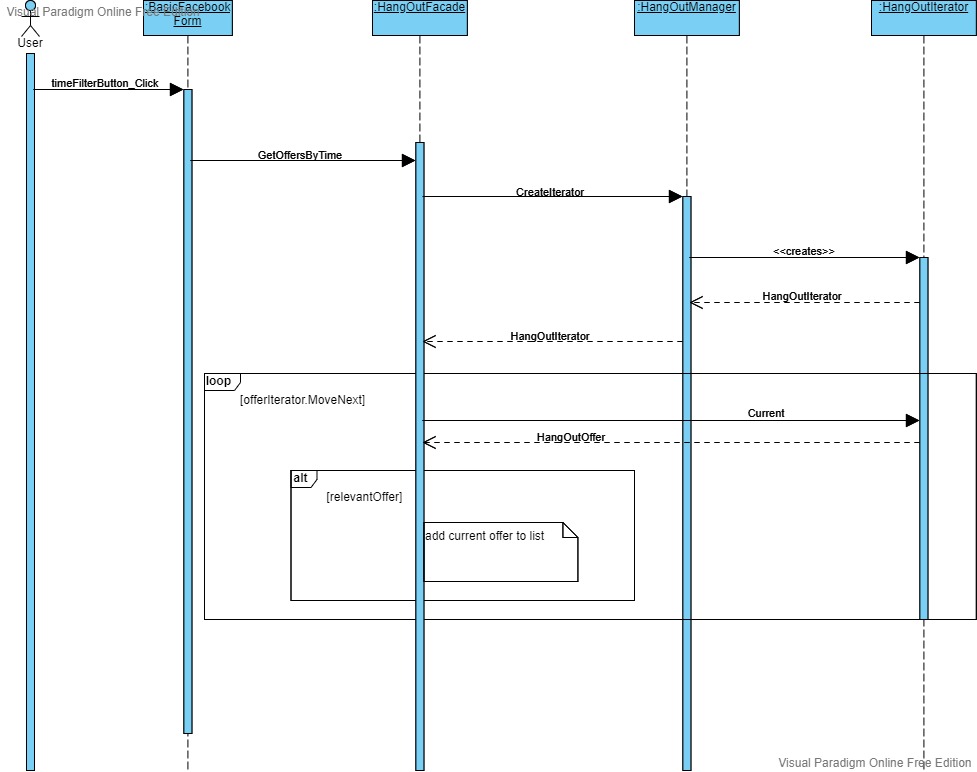
* סיבת הבחירה / שימוש בתבנית:

המחלקה HangOutManager מחזיקה מבנה נתונים שבו כלל הצעות הבילויים של המערכת. לכן, רציתי לספק למשתמש את האופציה לעבור על כלל ההצעות ששמורות במערכות, מבלי להכיר את מבנה נתונים בו הכל שמור, על מנת לבצע אופרציות נוספות על בסיס המידע ששמור במערכת אשר HangOutManager לא בהכרח מציע.

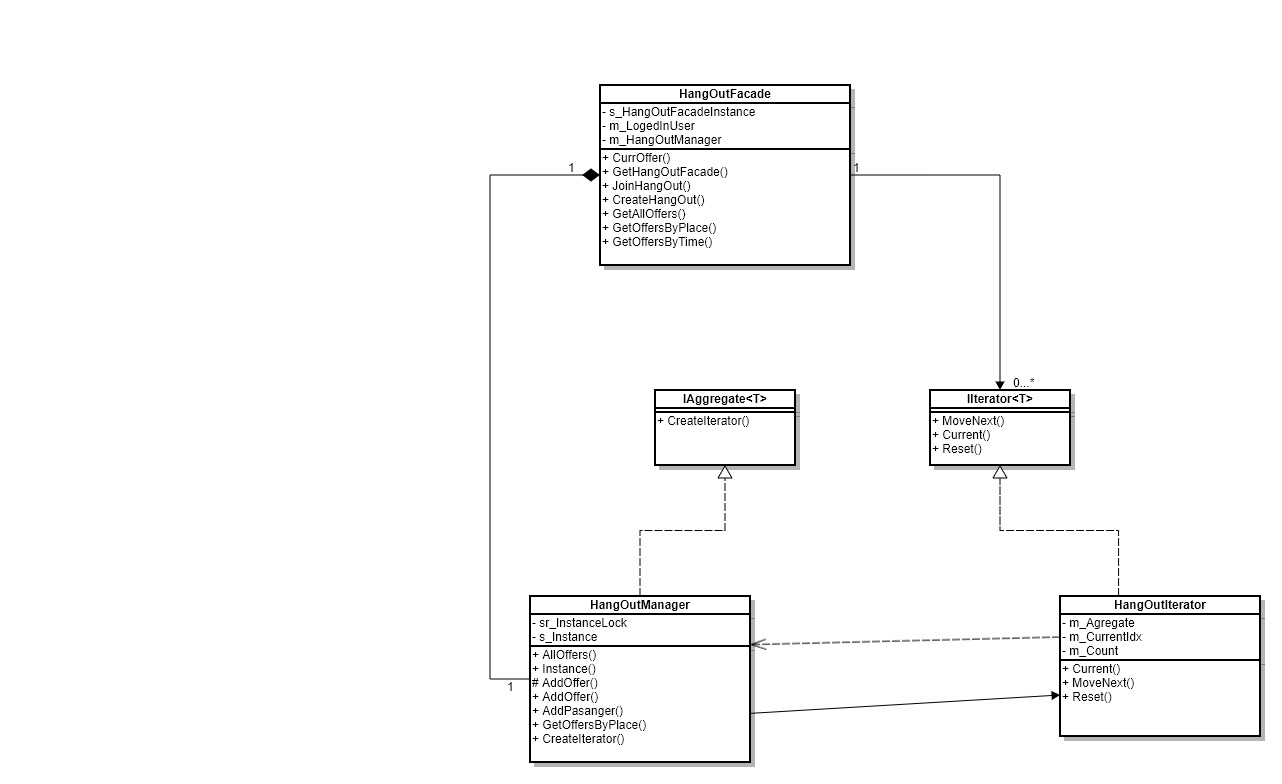
* אופן המימוש:

המחלקה HangOutManager מממשת ממשק IAggregate<T> ובו מתודה CreateIterator<T>, כאשר T=HangOutOffer. בנוסף, במחלקה זו יש מחלקה מקוננת HangOutIterator המממשת ממשק IIterator<T> עם אותו T, ובו 3 מתודות: MoveNext, Current, Reset. ניתן למצוא את המחלקות והממשקים תחת השכבה הלוגית.

* Sequence Diagram



* Class Diagram



1

### תבנית מס' 3 – Observer

* סיבת הבחירה / שימוש בתבנית:

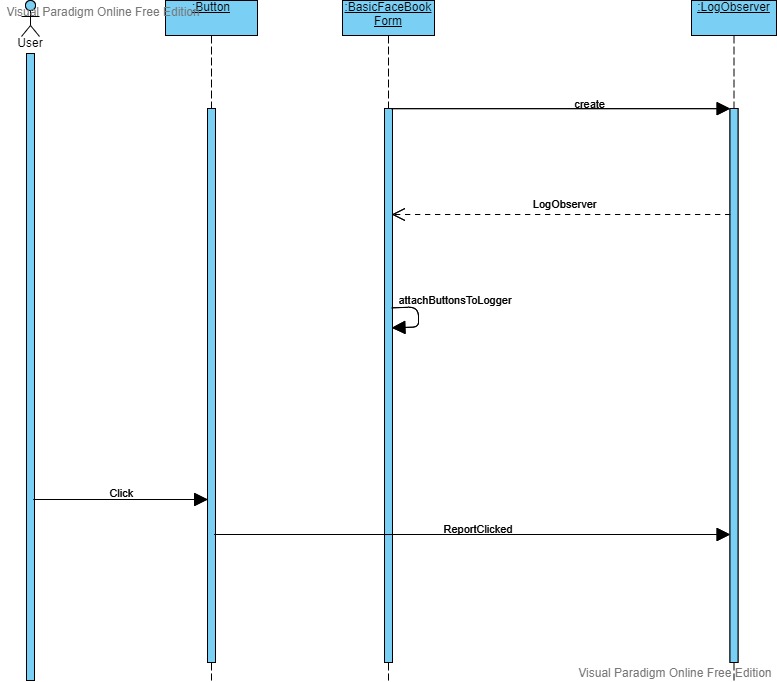
קבצי לוג (Log) הם כלי חיוני בפיתוח ותחזוקת אפליקציות. בחרתי לממש Crash Log כדי לעקוב ולתעד אחר הפעולות באפליקציה אל מול המשתמש. כך, במידה ותהיה בעיה באפליקציה, יהיה אפשר לקבל מידע ולעקוב אחר הפעולות שבוצעו באפליקציה בכדי לנסות לגרום שתרחיש שכזה לא יקרה שנית.

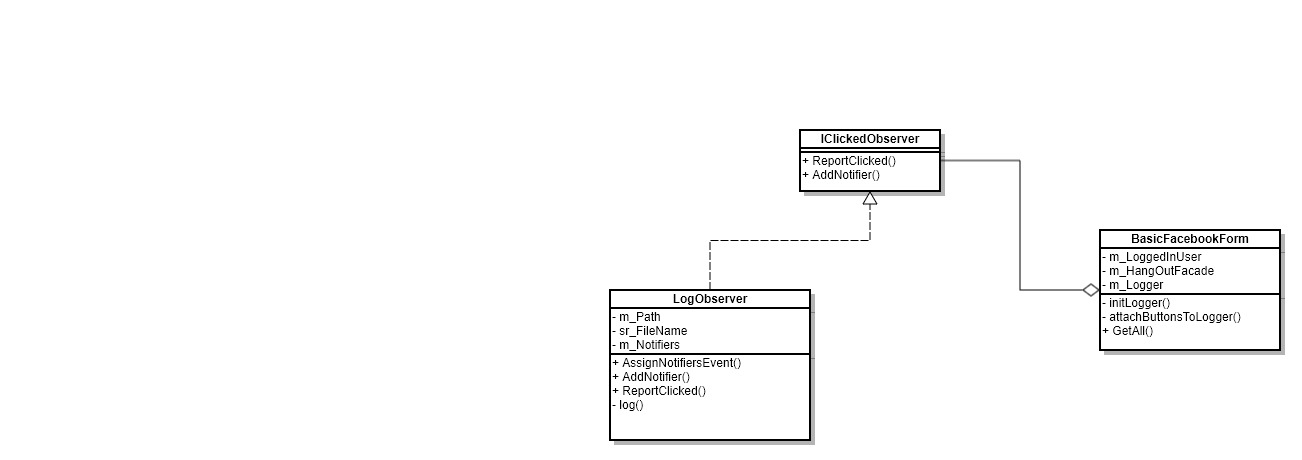
הLog יסייע באיתור ופתרון באגים וכך יתרום לתחזוקתיות של האפליקציה.

בחרנו בתבנית Observer מכיוון שהיא נותנת מענה למצב שבו נרצה שאובייקט מסוים יודיע לצופים עליו על שינויים המתרחשים בו.

* אופן המימוש:

מימשתי מחלקה קונקרטית בשם LogObserver שמממשת את הממשק IClickObserver, והיא תהיה ה-observer. בה הגדרתי רשימת כפתורים (Notifiers) שבכל הפעלה של האירוע click עליהם, הם ירשמו ל-Logger.txt.. מה שיאפשר מעקב ובדיקה של אירועים שגרמו לבאג. כשהמשתמש מתחבר למערכת, ונוצר הטופס הראשי של האפליקציה (BasicFacebookFitures), נוצר גם המופע של המחלקה, ומאותחלת גם רשימת הכפתורים אליהם מאזינים, ובמקביל גם ההרשמה לאירוע הלחיצה. כך, הכפתורים אינם "מודיעים" ישירות ללוג שהם נלחצו, אלא "מודיעים" לכלל מי שמאזין לאירוע זה, ובפרט ללוג..

* Sequence Diagram
* Class Diagram



0...\*

1