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

* קבלת רשימה של חברים שאהבו את אותו סרט\ספר\תוכנית טלוויזיה\זמר שאתה אוהב.

תחילה בוחרים קטגוריה מבין 4 הקטגוריות הבאות: סרטים, ספרים, תוכניות טלוויזיה ומוזיקה.

מתקבלת רשימה של כל הפריטים בקטגוריה שעשית להם לייק, למשל בקטגוריית סרטים מתקבלת רשימה של כל הסרטים שאהבת ולאחר בחירת סרט מהרשימה מתקבלת רשימת החברים שעשו לייק לאותו סרט*.*

* רשימה של כל החברים הרווקים עם הפרדה ל2 רשימות – בנים ובנות

ואפשרות לראות רשימת אירועים של חבר שנבחר.

### תבנית מס' 1 – Builder

* סיבת הבחירה / שימוש בתבנית:

הסיבה בגינה בחרנו להשתמש בתבנית זו היא מהיותו של האוביקט Reminder אוביקט מורכב הבנוי ממספר חלקים. שאיפתנו הייתה להפריד את הקוד של יצירת ה-Reminder מה-client בכדי שקוד ה-client יהיה "נקי" יותר ויותר תחזוקתי, הווי אומר, ששינוי במבנה של ה-reminder לא יגרור שינויים נרחבים בקוד ה-client. בנוסף, ע"י ההפרדה הנ"ל, נוכל להשתמש באותו תהליך יצירה כדי ליצור סוגים שונים של –Reminder, בא לאמר שכעת בעודנו משתמשים ב-API של Facebook כדי ליצור את ה-Reminder, בעתיד נוכל להשתמש ב-API של Twitter למשל, כדי ליצור Reminder – מבלי לשנות את קוד ה-client (השינויים יבוצעו ב-builder עצמו, ואף יתרה מכך, מהיותו של ה-builder כ-interface, כל מימוש קונקרטי של builder זה יכול לשמש אותנו ב-client מה שמאפשר לנו גמישות רבה יותר – רוצה לאמר, נוכל להחליף בין API שונים בלי לשנות את קוד ה-client עבור היצירה של ה-Reminder).

* אופן המימוש:

המחלקות אותן יצרנו כדי לתמוך בתבנית זו הן:

1. **Reminder:**

האוביקט המורכב עצמו, אשר את תהליך יצירתו רצינו להפריד מקוד ה-client. אובייקט זה מכיל בתוכו את האובייקט – ReminderMessege (ההודעה עצמה – ראה סעיף ב'), רשימה של string המהווה קישור לאירועים של המשתמש, ואת המייל של המשתמש.

1. **ReminderMessege:**

אוביקט המכיל פרטים של ההודעה עצמה, כגון גוף ההודעה ונושאו.

1. **:IReminderBuilder**

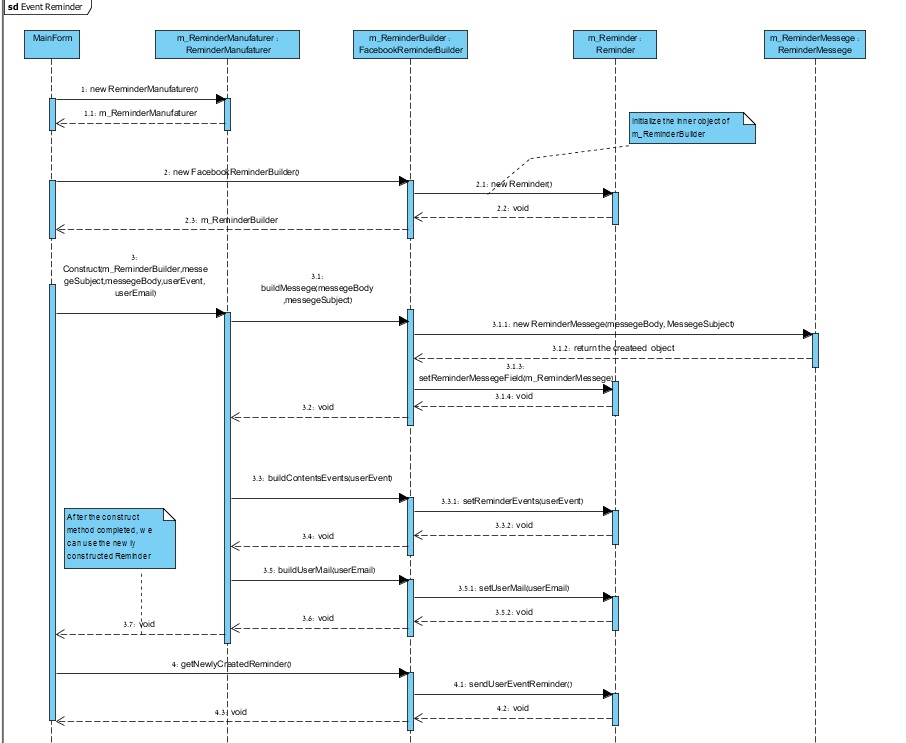
זהו ממשק ה-builder, תפקידו הוא לציין לסוגי ה-builder השונים, מה עליהם לתמוך כדי להיות חלק מתפקיד היצירה. הווי אומר שכאשר נרצה ליצור Reminder builder עבור הממשק של Facebook, עליו יהיה לממש מתודות אלו. נבחין – "היצרן" (סעיף ה') לא מודע לאיזה סוג של –builder הוא משתמש, עליו לדעת כי ה-builder שבו הוא ישתמש יממש את המתודות הרלוונטיות ליצירת האוביקט המורכב.

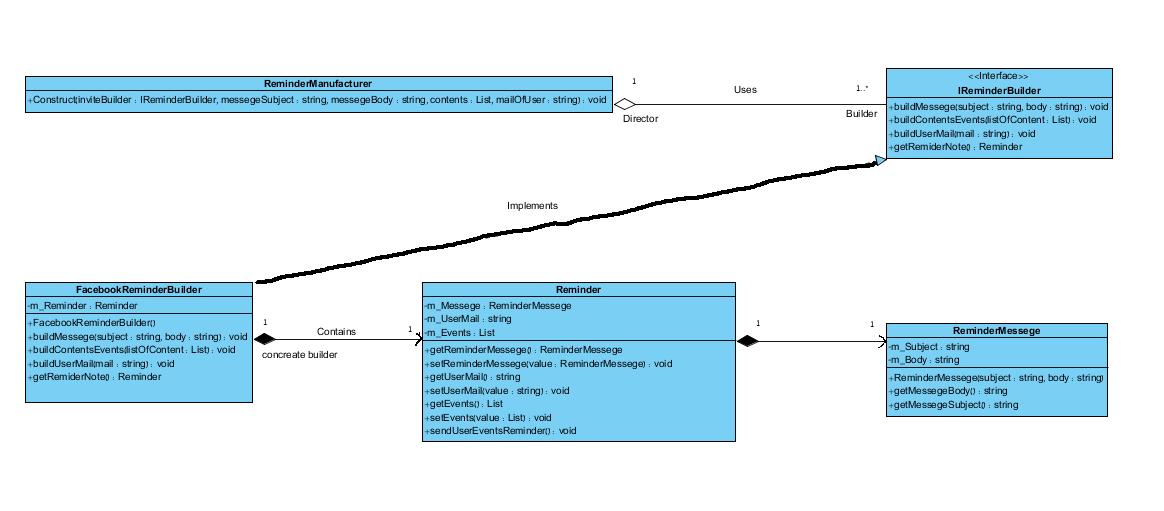
1. **FacebookReminderBuilder:**

זהו builder קונקרטי, כלומר, builder זה משתמש ב-API של Facebook וממש את הנדרש ממנו לפי הממשק של IReminderBuilder. בעזרת builder זה, "היצרן" יכול להשתמש בו כדי ליצור Reminder מסויים. הסיבה להפרדה בין ה-builder עצמו (שלמעשה הוא interface) לבין סוגי ה-builder השונים, היא לאפשר למערכת להיות pluggable בהיבט שבו אם נרצה לשנות את ההתמשקות מ-facebook ל-twitter למשל, אז חלק זה של המערכת, כבר יהיה מוכן (נצטרך כמובן להכין builder יעודי עבור ה-twitter עצמו).

1. **ReminderManufacturer:**

זהו היצרן. היצרן מקבל את הדרישות מהלקוח, את התבנית של ה-builder איתו אנו רוצים להשתמש (במקרה שלנו ה-FacebookBuilder) ואחראי על תהליך יצירת Reminder. למעשה היצרן הנ"ל מקשר בין הלקוח לבין היוצר, כך שהלקוח מדבר עם היצרן, היצרן מעביר את הדרישות ליוצר, ומחזיר תשובה ללקוח.

* Sequence Diagram:
* Class Diagram



המקביל ל-builder הוא ה-interface בשם IReminderBuilder כאשר המימוש הקונקרטי שלו (כי למדנו שניתן להחליף –builder במהלך החיים של הקוד) הוא, כרגע, FacebookReminderBuilder. ה-Compositor האחראי ליצירת והפעלת ה-builder, ע"י קבלת הוראות מהמשתמש, הוא ה-ReminderManufaturer.

### תבנית מס' 2 – Adapter

* סיבת הבחירה / שימוש בתבנית:

הסיבה בגינה בחרנו להשתמש בתבנית זו הינה כדי לגשר על הממשק שמוצע ע"י רכיב ה-Legecy ה- User לבין הממשק שאותו ה-client צריך עבור האוביקט הממש את הממשק ISmartPostStatus. אוביקט זה, עתיד להשתמש במתודה PostStatus של ה-User אך ממשקו שונה מהממשק של User. רוצה לאמר, הממשק של האוביקט בא לבצע PostStatus ובנוסף לגבות את ה-Post בקובץ אצל המחשב של המשתמש (קובץ log). עקב חוסר ההתאמה בין ממשק ה-User לממשק שאותו אנו צריכים, הווי אומר – ה-User לא מאפשר שמירת Post בקובץ log, עלינו לבצע אדפטציה לממשק זה, ועל כן נשתמש ב-UserAdapter שמכיל בקומפוזיציה את ה-User המקורי ומתאים אותו לממשק אותו אנו רוצים (כאמור ממשק ה-ISmartPostStatus).

* אופן המימוש:

המחלקות אותן יצרנו / איתן השתמשנו כדי לתמוך בתבנית זו הן:

1. **ISmartPostStatus:**

זהו ה-interface אותו אנו רוצים לעבוד עבור ביצוע של –PostStatus. למעשה, מחלקה זו מהווה את ה-Target עבור הלקוח. ה-Adapter שנבנה (ראה סעיף ב') יממש את ה-interface הנ"ל תוך התממשקות מול מחלקת User של FacebookWrapper (האדפטציה עצמה).

1. **UserAdapter:**

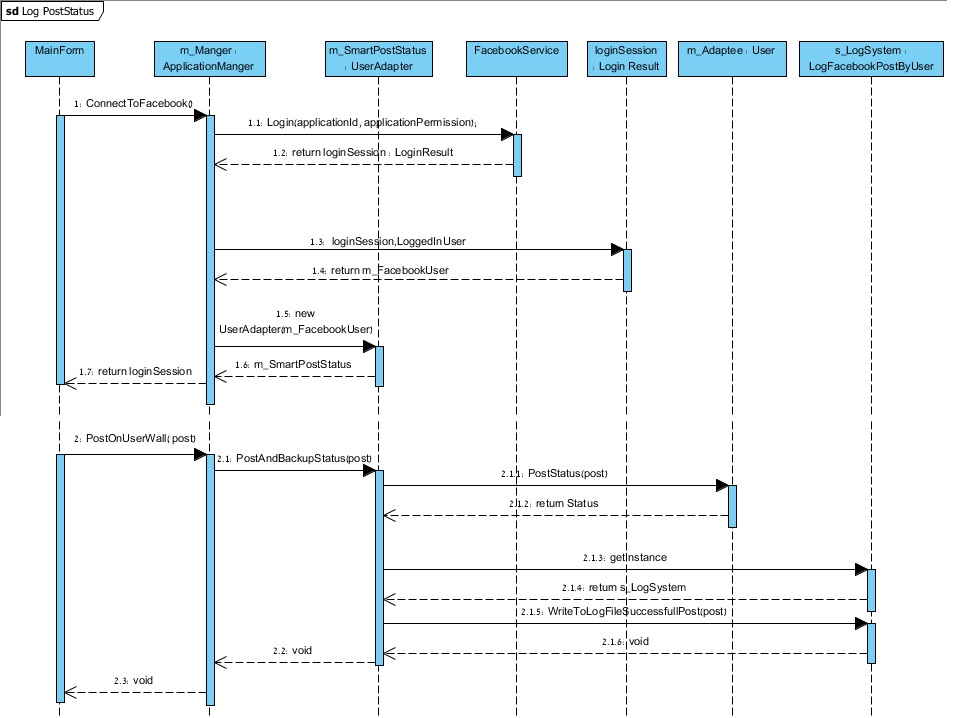
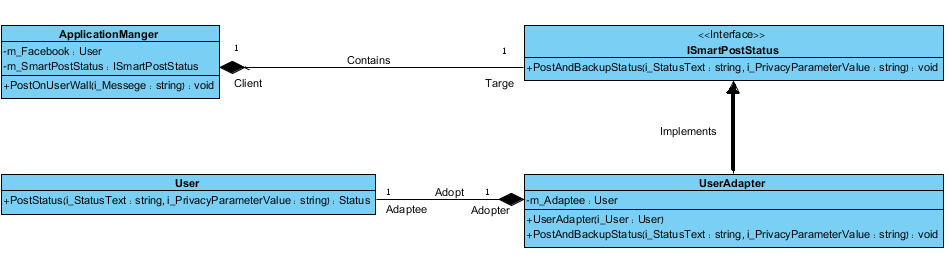
זהו ה-Adapter עצמו, כפי שמרמז שמו. מחלקה זו מחזיקה בתוכה את האוביקט User ומממשת את הממשק ISmartPostStatus. המחלקה תשתמש במתודות הקיימות ב-User ובנוסף תבצע התאמה לעבודה מול קובץ ה-Log שאותו אנו רוצים לעדכן בכל פעם שמתבצע PostStatus.

1. **LogFacebookPostByUser:**

מחלקה זו נועדה לצורך עדכון קובץ ה-Log, בכל פעם שמתבצע PostStatus. אנא הבחן כי המחלקה משמשת את ה-UserAdapter כחלק ממענה על הממשק שהלקוח מצפה לעבוד מולו (ISmartPostStatus). הווי אומר שמחלקה זו תאפשר ל-Adapter לענות לדרישת הלקוח. הבחן בנוסף כי מחלקה זו ממומשת כסינגלטון (הסבר נרחב בחלק המסביר על תבנית 3).

1. **User:**

זוהי מחלקה פנימית של FacebookWrapper והיא המחלקה שעבורה נבצא אדפטציה, כלומר היא ה-Adaptee במימוש התבנית הנ"ל.

* :Sequence Diagram
* Class Diagram:

המקביל אל ה-client הוא ה-ApplicationManger אשר מצפה לעבוד עם רכיב המממש את הממשק ISmartPostStatus. המטרה שלנו, כלומר ה-Targer הוא כאמור רכיב ISmartPostStatus, ברשותנו יש רכיב legacy והוא ה-User, הווי אומר, ה-User הוא ה-Adaptee. האדיפטציה מתבצעת ע"י רכיב ה-UserAdapter שהוא כאמור ה-Adapter והוא מגשר על הפער בין מה שה-client מצפה לעבוד איתו (ISmartPostStatus) לבין הרכיב הקיים. (אבחנה – הוכנסו לדיאגרמה רק המתודות הרלוונטיות לתהליך כדי לשמור על הניקיון של הדיאגרמה).

### תבנית מס' 3 – Singleton

* סיבת הבחירה / שימוש בתבנית:

הסיבה בגינה בחרנו להשתמש בתבנית זו נעוצה ממהותו של ה-

LogFacebookPostByUser. אוביקט זה נועד לגשת ולשמור בקובץ Log שאותו אנו יוצרים את כל פעולות ה-PostStatus אותן המשתמש מבצע. נבחין כי הקובץ Log אמור להיווצר לאורך כל התהליך רק פעם אחת בלבד. בכל התייחסות נוספת לקובץ זה, נרצה להשתמש בקובץ שכבר נוצר ולא ליצור אחד חדש. לשם המחשה, נניח ואוביקט זה לא היה סינגלטון, אזי, יתכן וכשנבצע מספר פעולות PostSatus ניצור מספר קבצי log וזה בהחלט לא כוונתנו (אנו רוצים רק קובץ אחד כזה לאורך כל התהליך).

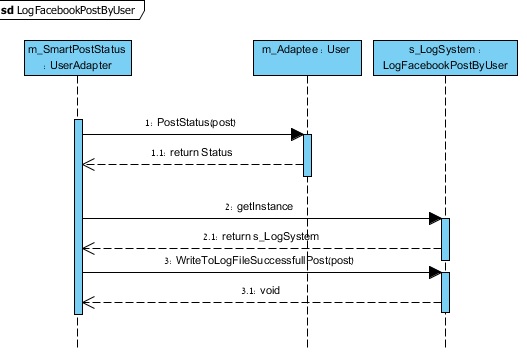
* אופן המימוש:

המחלקות אותן יצרנו / איתן השתמשנו כדי לתמוך בתבנית זו הן:

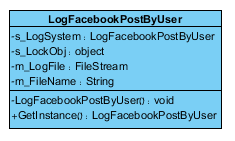
1. **LogFacebookPostByUser:**

אובייקט זה הוא הסינלטון בו אנו משתמשים. האוביקט משתמש ב-Private constructor כדי למנוע יצירה לא מבוקרת של האוביקט, רוצה לאמר, היצירה היחידה לאוביקט קורת רק כאשר מנסים לגשת לאוביקט פעם ראשונה. בכל פעם אחרת אנו נקבל את האוביקט שכבר נוצר. נבחין כי סינגלטון זה משתמש בשיטת ה-Double checking כדי למנוע נעילה סתמית של האובייקט כשאנו מגנים על האוביקט מבעיות סנכרון ב-Thread שונים.

* Sequence Diagram:



* איפה התיאור של התהליך שמבצע הסינגלטון במתאר את הסינגלטוניות שלו, כשמבקשים ממנו מופע בפעם הראשונה? הדיאגרמה אמורה לתאר את השימוש בתבנית, וזה לגמרי לא קורה בדיאגרמה הזו. (-4)
* מדוע קו החיים של המופע האחד מתחיל מהתחלה ולא רק מהרגע שהוא נוצר?. (-4)
* Class Diagram:



• לא מתואר בדיאגרמה מי משתמש בסינגלטון. חלק ניכר מהצידוק בשימוש בסינגלטון הוא ריבוי הרכיבים הדורשים את שירותיו והרצון לאפשר להם גישה נוחה ויזומה אליו.(-4)

o אין אינדיקציה לעובדה שיש למחלקה קשר סטטי עם עצמה (4-)

• דיאגרמת המחלקות לא מלאה (-4)

Threads-

נמצא במסך של הfeeds ונועד להביא את כל הרכיבים במסך ב"מקביל"

במסך יש רכיבים רבים שהבאה שלהם מהאתר לוקחת זמן לכן כל thread אחראי על חלק אחר : הפוסט עצמו, תמונה של מפרסם הפוסט ופרטים על הפוסט

Data Binding-

במסך my zone יש למעלה אופציה להכנס ל about שנותן מידע על היוזר שהתחבר באמצעות Data Binding