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

* [הפיצר הראשון יוצר רשימה ומציג למשתמש, את חבריו בפייסבוק, החברים החדשים שלו, וחברים שהסירו אותו מרשימת החברות.]
* [הפיצר השני יוצר שני רשימות למשתמש לפי החיפוש שבחר בעצם כדי למצוא דייט מרשימת החברים הקיימים, בין אם הכי מתאים או מתאים אבל לא לגמרי]

### תבנית מס' 1 – [Observer Pattern]

* סיבת הבחירה / שימוש בתבנית:

[תיאור הסיבה / הצורך בשימוש בתבנית במערכת שלכם:]

באפליקציה שלנו, יש את פיצר החברים שמתואר לעיל. רצינו להוסיף רכיבים חדשים לאפליקציה שלנו, רכיב שעובד מאחורי הקלעים ומעדכן את רשימות החברים אונליין כאשר יש שינוי, כלומר אם נוספו\נמחקו חברים, המשתמש יכול לראות את השינוי אונליין כאשר הUI מתעדכן בהתאמה לשם כך.

הרכיב יקרא "Updater". "Updater" הינו רכיב שמתפקד כ"צופה" (subsriber/observable) הוא מקושר (יש לו רפרנס) לmainFormובאמצעות המטודה, הוא יכול לבדוק האם רשימת החברים היא אותו הרשימה שכרגע מעודכנת בשרת הפייסבוק. לשם ביצוע בפועל של הבדיקה ברקע, בנינו אובייקט נוסף בשם "SchedulerChecker" שהינו מתפקד כאובייקט היחוס (obsever/publisher) שהוא אחראי לפעול ברקע ובאינטרוולי זמן קבועים, האוביקט הזה מקושר ל Updater ומפעיל את המטודה שלנו ע"י קריאה ל notify().

* אופן המימוש:

[תיאור המימוש והיכן ניתן למצוא אותו בקוד]

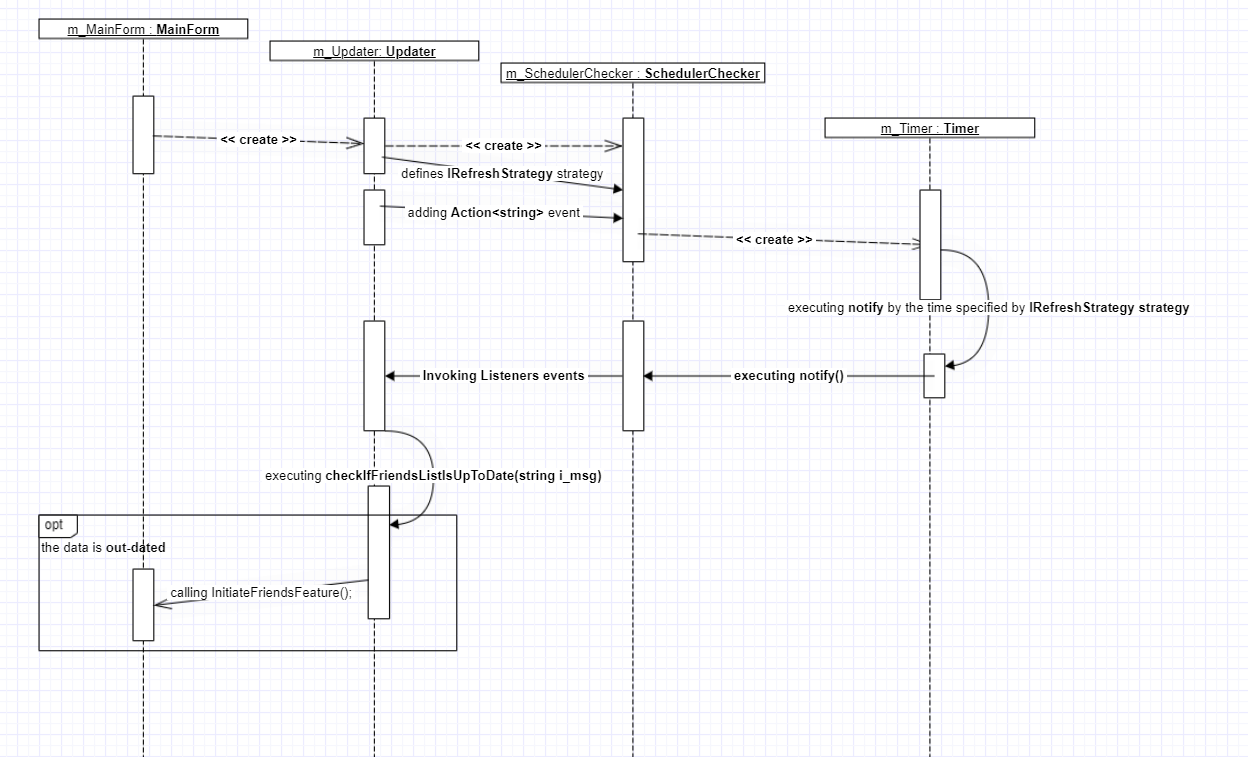
את רוב המימוש ניתן למצוא בקובץ "UpdatingComponents.cs" שם יש את כל המחלקות המדוברות. בחרנו לממש את תבנית הצופה בפיצרים של דוט נט – שימוש באירועים (events), בעצם המחלקה "Updater" מחזיקה את המטודה הרצויה שהיא Action<string> ואותה המטודה עוברת (הרשמה) לאובייקט היחוס שנוצר בבטנו של "Updater". אוביקט היחוס "SchedulerChecker" מחזיק רשימה של אירועים (רשימה של איוונטי אקשיינים) ובעמצעות המטודה notify הוא מעורר את המטודות שמקושרות לאירועים הללו.

בפועל, Updater מחזיק אובייקט "Frinedlist" אשר נמשך מפייבסוק, ומשווה אותו עם הרשימה הנוכחית שנמצאת ב mainForm ומוצגת ב UI, אם אכן יש צורך בשינוי כלשהו, "Updater" קוראת למטודה מתוך mainForm שמאתחלת את כל הפיצר של החברים מהתחלה ובעצם כך הכל נהיה מעודכן ב UI עבור המשתמש – חוויה של real-time.

תיאור שלם של הרכיבים נמצא למטה בתיאור של התבנית השניה "Strategy pattern".

* Sequence Diagram

[sequence diagram שמציגה את התהליך שקשור לתבנית והאינטראקציה בין המחלקות]



* Class Diagram

[class diagram שמתארת את המחלקות שלכם שמעורבות בתבנית (תיאור מלא שכולל Properties ו- Methods) והיחסים ביניהם (תאור מלא עבור כל יחס כפי שלמדנו בכיתה). עבור כל מחלקה שלכם, כיתבו מי המקבילה שלה (אם יש כזו) בתבנית (Pattern) שבחרתם

### 

### תבנית מס' 2 – [Strategy pattern]

* סיבת הבחירה / שימוש בתבנית:

[תיאור הסיבה / הצורך בשימוש בתבנית במערכת שלכם]

בהתאם ובמהשך לרכיבים החדשים שתוארו בתבנית הקודמת שמלמעלה ^^, מתבקש עכשיו מימוש פעולת ה "SchedulerChecker". רכיב זה אמור לפעול ברקע (כמו תהליך אחורי) ובאינטרוולים של זמן קבועים לקרוא למטודה ששייכת ל "Updater" שהיא בעצם בודקת האם יש משהו חדש בשרת.

באובייקט שלנו מתבקשת השאלה ע"פ איזה זמן האובייקט צריך לבצע את זה? כל כמה זמן? אפשר לדוגמה, שזה יתקשר עם השרת ברקע כל 10 שניות, או כל 3 דקות או בעצם בעתיד נחליט בזמן אחר. אולי 10 שניות זה קצת אינטנסיבי, אכן זה דורש משאבים בעיקר של רשת מרובים יותר, לפיכך, קיימות אסטרטגיות רבות שאפשר לבחור מהן בזמן ריצה, ופה נכנסת תבנית האסטרטגיה במימוש שלנו.

* אופן המימוש:

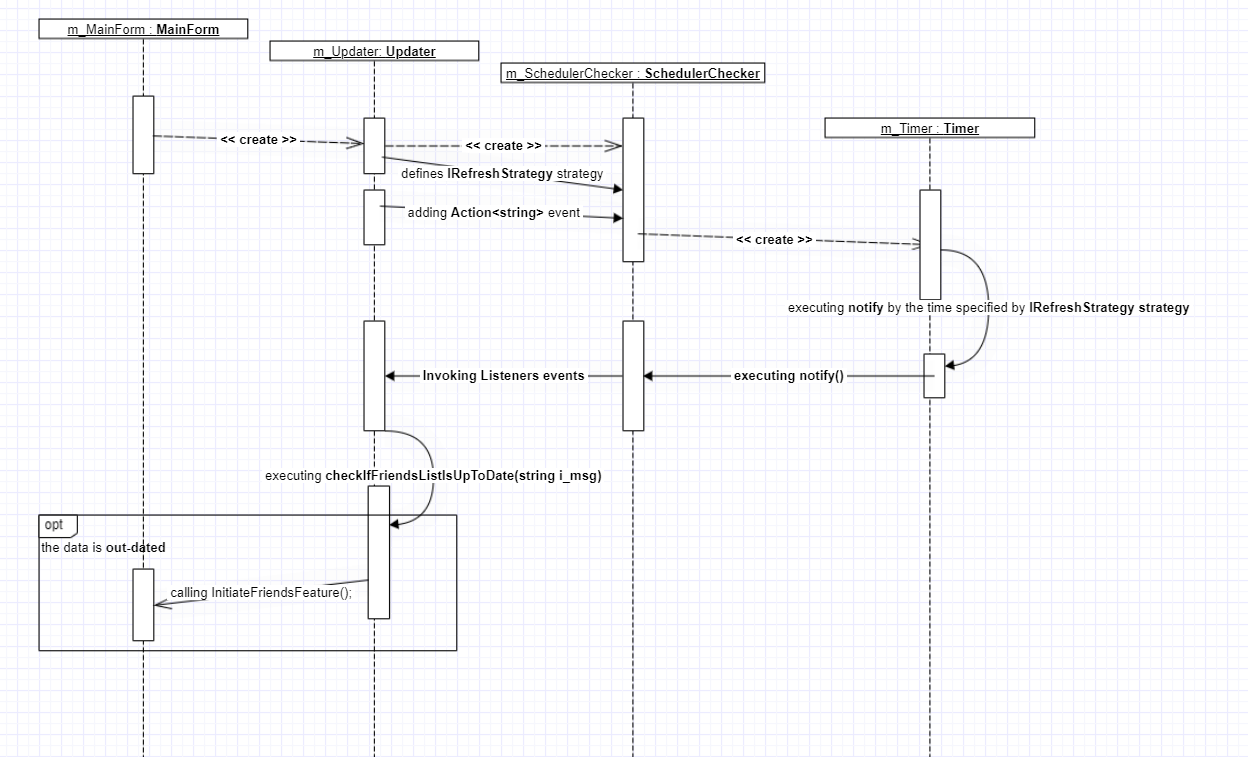
[תיאור המימוש והיכן ניתן למצוא אותו בקוד]

את רוב המימוש ניתן למצוא בקובץ "UpdatingComponents.cs" שם יש את כל המחלקות המדוברות. קודם כל, בחרנו להשתמש במחלקה "Timer" – היא בצעם פיצר נחמד בדוט נט, מחלקה שפועלת כמו תהליך אחורי, ובעצם מבצעת מטודה באינטרוולים קבועים שנקבעים בזמן האיתחול של האובייקט עצמו. אופן מימוש של תבנית האסטרטגיה: יש את הממשק "IRefreshStrategy" שהיא בעצם מגדירה איך אסטרטגיה קונקרטית אמורה להיות. מחלקה שכזו נמצאת (באמצעות קוצפוזיציה) בתוך המחלקה "SchedulerChecker ".

לבסוף המחלקות הקונקרטיות של האסטרטגיות הן: "QuickRefreshStrategy" שמגדירה אינטרוולים של 10 שניות (מהיר ורציף) ויש את המחלקה "SlowRefreshStrategy" שמגדירה אינטרוולים של 3 דקות (איטי), לבסוף המתכנת בוחר בזמן האיתחול של המחלקה "Timer" באיזה אסטרטגיה הוא בוחר.

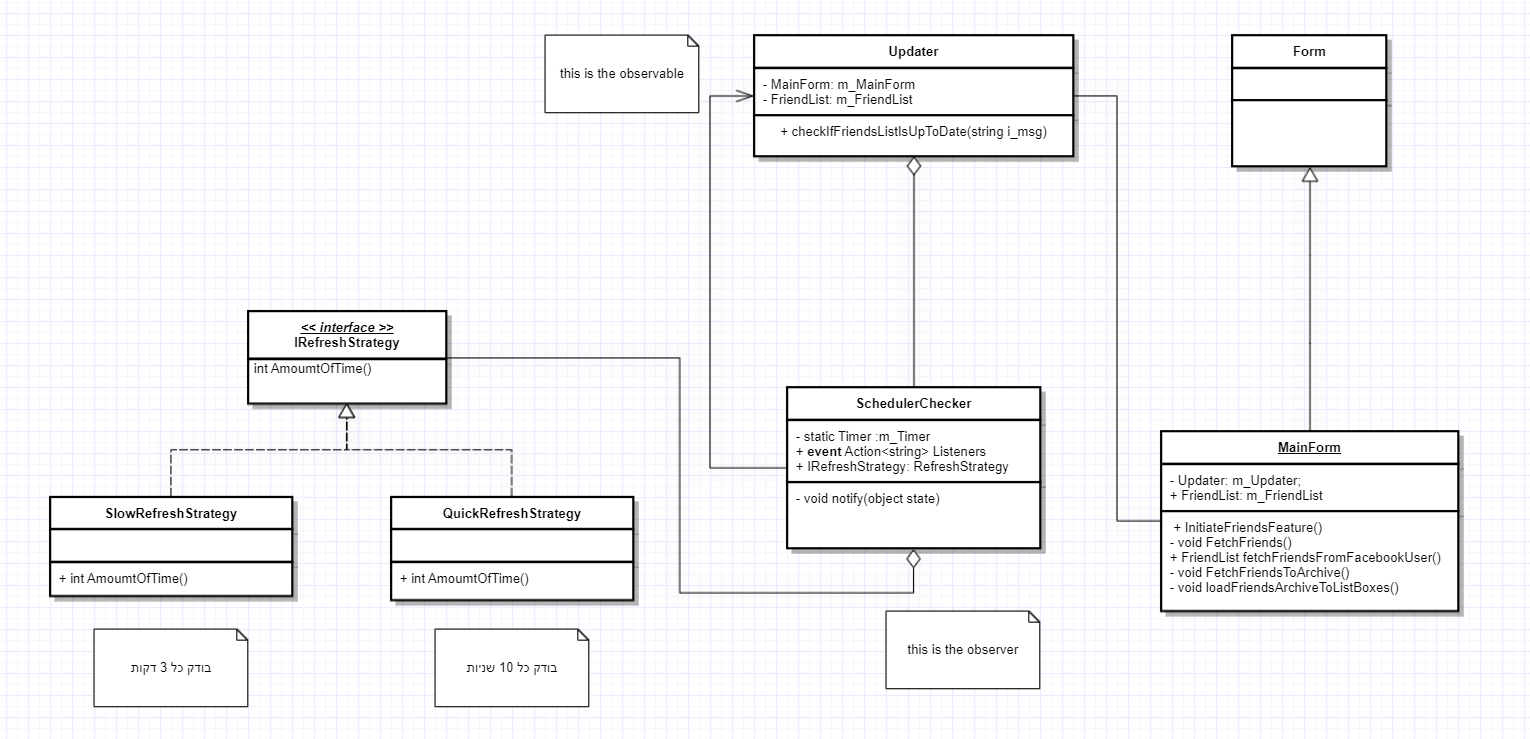
* Sequence Diagram

[sequence diagram שמציגה את התהליך שקשור לתבנית והאינטראקציה בין המחלקות]



* Class Diagram

[class diagram שמתארת את המחלקות שלכם שמעורבות בתבנית (תיאור מלא שכולל Properties ו- Methods) והיחסים ביניהם (תאור מלא עבור כל יחס כפי שלמדנו בכיתה). עבור כל מחלקה שלכם, כיתבו מי המקבילה שלה (אם יש כזו) בתבנית (Pattern) שבחרתם



### תבנית מס' 3 – [Template Method]

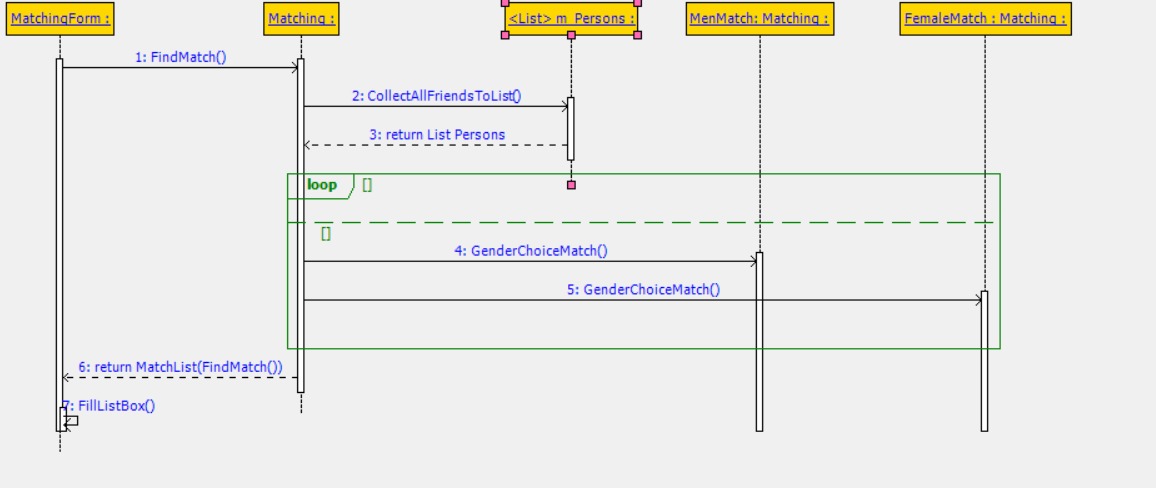
* סיבת הבחירה / שימוש בתבנית:

הסיבה שבה בחרנו להשתמש בתבנית זו היא משום שהיא נוחה וקלה לשימוש וגם מתאימה ביותר לסיטוציה שתתואר בהמשך, מה גם שזה שימושי עבורנו וחוסך מאיתנו כפל קוד מיותר, לכן השתמשנו בתבנית זו כדי ליצור שלד מוכן ואחר כך להשתמש באלגוריתם עצמו ולעשות בו שינויים קטנים\גדולים שאנו נצטרך לעשות כמו למשל: "גבר" חייב להיות רווק ואישה - לא חייבת להיות רווקה, יכולה להיות בערכת יחסים מסובכת (הרשנו סיטואציה מוזרה זו רק להמחשת המימוש היעיל תבניתית באפליקציה שלנו), השימוש בתבנית מאד דומה לדוגמא שהייתה לנו בשיעור עם SORTING מקטן לגדול וההפך במקום להעתיק את כל הקוד היה שלד מוכן ורק בוצעו שינויים קטנים באלגוריתם וכך נעשה גם בתבנית הזו.

* אופן המימוש:

את המימוש ניתן למצוא קודם כל בMATCHING FORM לאחר מכן אחרי בחירה של מאפיינים שהמשתמש בחר ולחיצה לחיפוש ניתן לראות את המימוש במחלקה MATCHING ששם נמצא השלד שלנו ולאחר מכן נשלך המימוש בהתאם לבחירת המשתמש בין אם זה גבר אז מחלקה של MENMATCH ואם הייתה בחירה אישה אז מחלקה של FEMALEMATCH

* Sequence Diagram



* Class Diagram

[class diagram שמתארת את המחלקות שלכם שמעורבות בתבנית (תיאור מלא שכולל Properties ו- Methods) והיחסים ביניהם (תאור מלא עבור כל יחס כפי שלמדנו בכיתה). עבור כל מחלקה שלכם, כיתבו מי המקבילה שלה (אם יש כזו) בתבנית (Pattern) שבחרתם

