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

* SortingFriends: המשתמש מקבל את רשימת חבריו ויכול למיין אותם לפי סוג המיון אותו בוחר. כאשר המשתמש בוחר חבר מסוים, מוצגות תכונות רלוונטיות עבור אותו סוג מיון
* FindBestFriend: המשתמש מחפש את החבר הכי טוב לו (ע"פ מתן משקולת לכל תכונה) שיש לו יום הולדת ב4 חודשים הקרובים ומציג את פרטיו. לאחר מציתא החבר הכי טוב, ניתן ליצור לחבר מסיבת הפתעה
* נציין כי בחלק מן התבניות לא ציינו בדיאגרמות עצמן מה כל מחלקה מייצגת על פי הGang of four, מכיוון שחלק מן התבניות מומשו באמצעות פיצ'רים שקיימים ב c# ולכן שמות התפקידים לא תואמים את הדיאגמות הללו.

### תבנית מס' 1 – Strategy

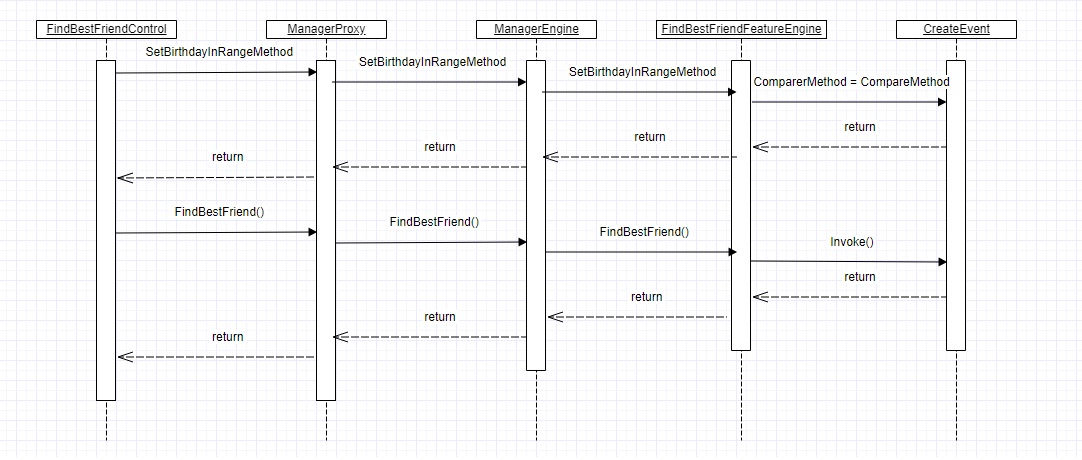
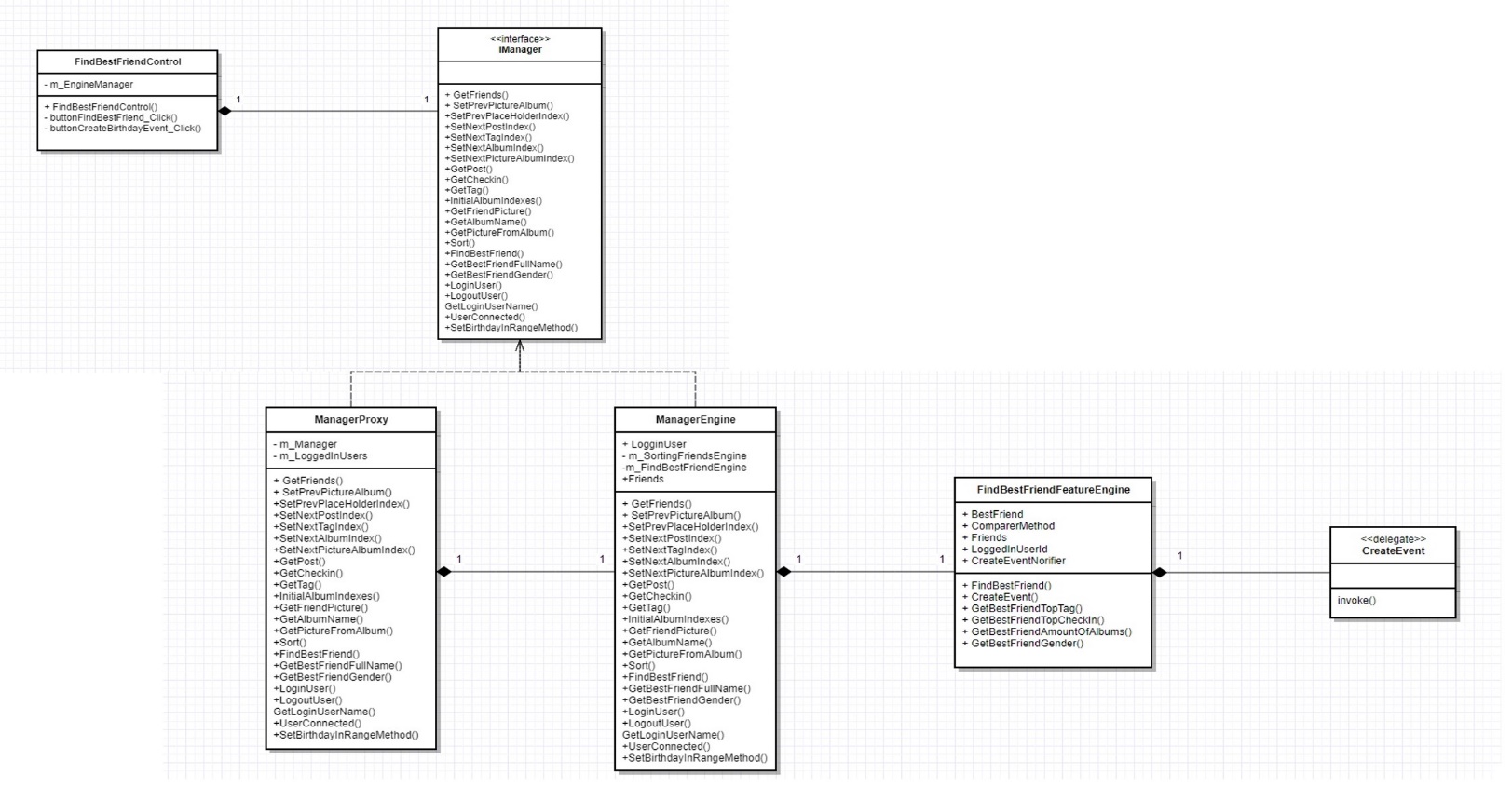
* סיבת הבחירה / שימוש בתבנית:

באפליקציה שלנו מחלקת FindBestFriendFeatureEngine מהווה את התבנית Strategy.

בחרנו בתבנית זאת על מנת לאפשר קוד גמיש אשר נוכל להשתמש בו שוב ללא צורך בשינוי קוד (reuse). הבעיה שאחנו באים לפתור היא שאם בעתיד ירצה המשתמש להשתמש במחלקת FindBestFriendFeatureEngine וירצה להגדיר את מספר הימים שבו ירצה לחפש את החבר הכי טוב שיש לו יום הולדת לצרכיו, המימוש יאפשר למשתמש להגדיר את אלגוריתם ההשוואה לפיו ירצה לעבוד. השתמשנו בתבנית זאת שהמשתמש יוכל לספק למחלקת FindBestFriendFeatureEngine את מתודת ההשוואה למציאת החברים בטווח תאריכים מסויים.

* אופן המימוש:

FindBestFriendFeatureEngine – זוהי מחלקה אשר מטרתה למצוא את החבר הכי טוב של המשתמש. במחלקה זאת אנו מספקים למשתמש במחלקה זאת לבחור את מתודת ההשוואה של מציאת החברים שיש להם יום הולדת בטווח זמנים מסויים. מימשנו את התבנית במחלקה זאת ע"י Func Property אשר קיים ב c#. המשתמש במחלקה יכול לבצע השמה לפונקציית השוואה לפי בחירתו. בנוסף, על מנת להימנע מפוטנציאל לבאגים בכך שהמשתמש לא יספק מתודת השוואה, סיפקנו בעצמנו מתודת ברירת מחדל.

* Sequence Diagram
* Class Diagram

### תבנית מס' 2 – Observer

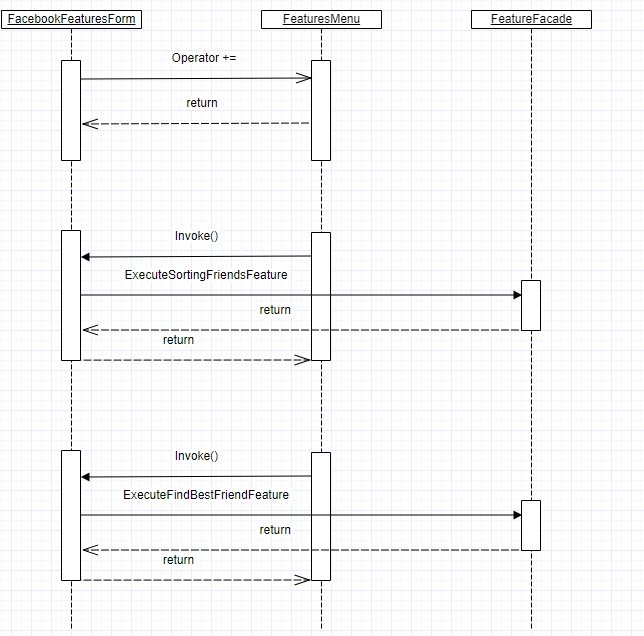
* סיבת הבחירה / שימוש בתבנית:

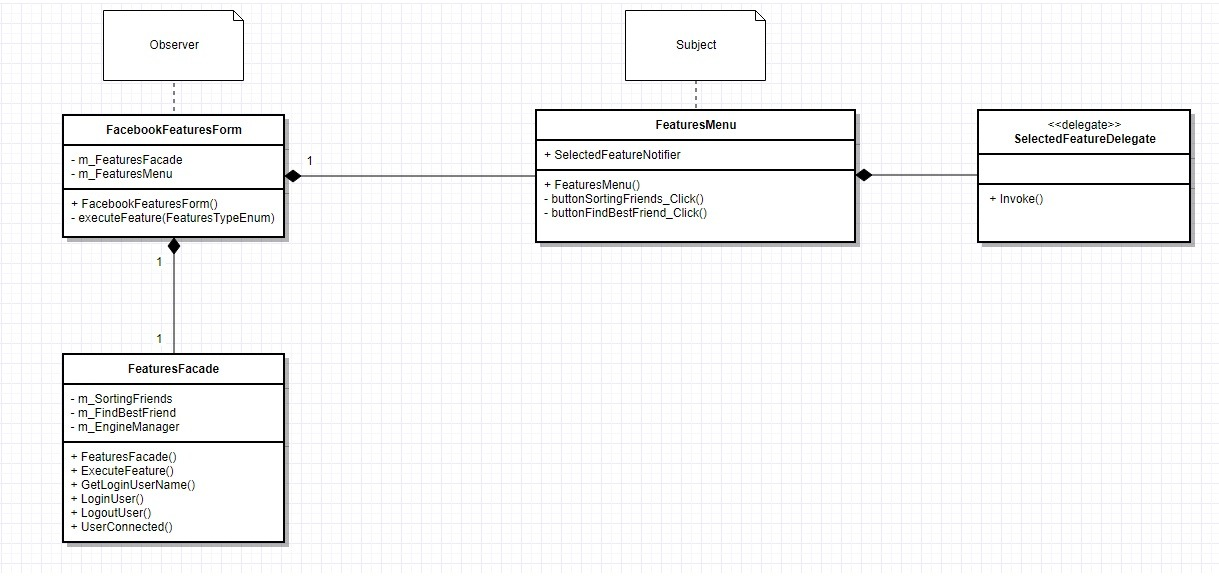
באפליקציה שלנו מחלקת FeaturesMenu מהווה בתבנית זאת את Subject ואילו מחלקת FacebookFeaturesForm את Observer. השתמשנו בתבנית זאת מהסיבה שבמידה וקיים שינוי בstate של המחלקה ומחלקות שונות המתעניינות בשינוי הstate ונרשמו כמאזינות, אז באופן אסינכרוני, המחלקה תודיע לכל המאזינים שלה על השינוי. תבנית זאת מטרתה לפתור את הבעיה שאם בעתיד נרצה להוסיף מאזין נוסף לאירוע המבוקש כלומר המאזין ירצה לדעת כאשר יש שינוי ב state מסויים, המאזין החדש יוכל להצטרף מבלי צורך לשנות את הקוד המחלקה, כלומר כל observer רושם את עצמו אצל ה subject וברגע שהוא משתנה הוא יודיע לכל המאזינים (באופן אסינכרוני). השימוש בתבנית זאת אצלנו באפליקציה מתבצעת בכך שה FacebookFeaturesForm מאזין ל FeaturesMenu. ברגע שכפתור מסויים נלחץ ב FeaturesMenu הוא מודיע ל FacebookFeaturesFormשידע איזה פאנל עליו להציג כרגע.

* אופן המימוש:

כמו שרשמנו למעלה, מחלקת FeaturesMenu מהווה בתבנית זאת את Subject. במחלקה זאת השתמשנו ב event אשר קיים ב c# אשר מספק לנו מבחוץ attach ו dettach ע"י אופרטור += ו -=. כאשר נלחץ כפתור מסויים מתבצע invoke ל event ונשלח לכל המאזינים איזה כפתור נלחץ. FacebookFeaturesForm מכיל בתוכו את מחלקת FeaturesMenu ומספק מתודה לביצוע ל FeaturesMenu באמצעות אופרטור +=, אשר תתבצע כאשר המשתמש ילחץ על מסויים בתוך FeaturesMenu.

* Sequence Diagram



* Class Diagram

### תבנית מס' 3 – Iterator

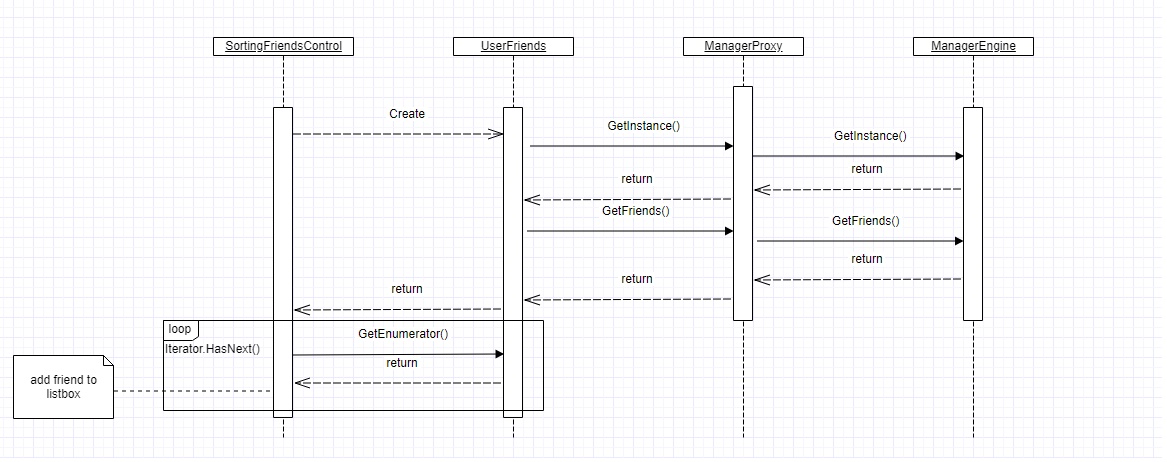
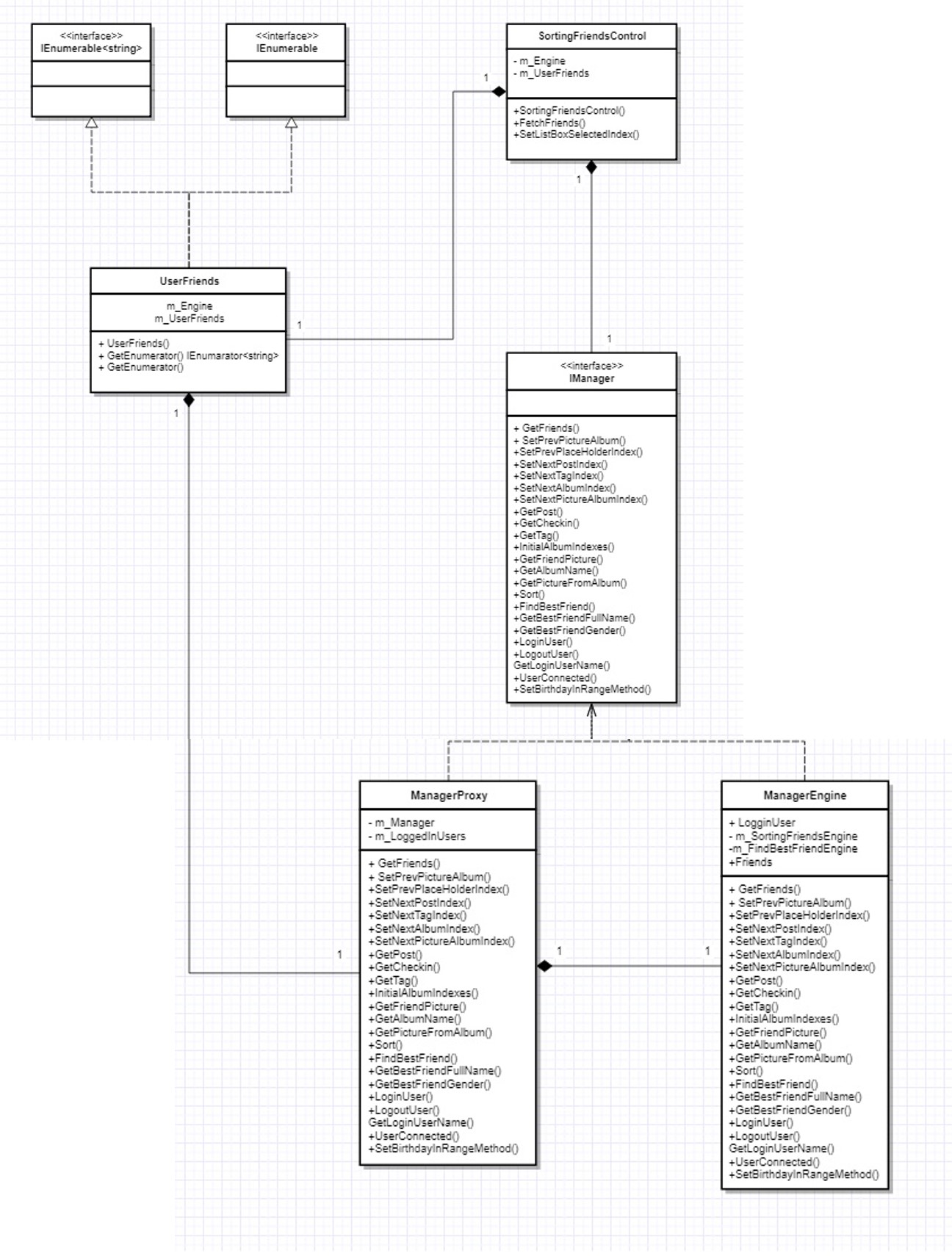
* סיבת הבחירה / שימוש בתבנית:

באפליקציה שלנו מימשנו את תבנית זאת על מחלקת UserFriends. תבנית זאת באה לפתור את הבעיה בכך שניתן לגשת ולעבור על איברי קולקציה במחלקה מסויימת מבלי להכיר את מימוש הקולקציה כלומר אם בעתיד נרצה להחליף את מבנה הנתונים שבו ישמרו החברים של המשתמש, הפעולה תהיה שקופה לכל המשתמשים בה. בחרנו בתבנית זאת על מנת שנוכל לספק למשתמש את המעבר על חבריו מבלי היכרות עם מבנה הנתונים ומבלי ליצור תלות בקוד המחלקה. השתמשנו בתבנית זאת על מחלקת UserFriends אשר מספקת למשתמש במחלקה לעבור על שמות כל חבריו מבלי להכיר את אופן המימוש שבו נשמרו החברים והמחלקה המשתמשת בה היא מחלקת SortingFriendsControl.

* אופן המימוש:

מימשנו את את תבנית זאת על מחלקת UserFriends בכך שהוא ממש את ה interfaces הבאים: IEnumerable<T>, IEnumerable. את המתודה GetEnumerator() של

IEnumerable<T> מימשנו באמצעות yield return שנלמד בכיתה. לכן באופן כזה אפשרנו למשתמש במחלקה לרוץ על איברי המחלקה ע"י לולאת forecah.

* Sequence Diagram
* Class Diagram