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

**Dating:**

**מציאת התאמה מתוך רשימת החברים של המשתמש באופן הבא:**

המשתמש ממלא את הנתונים במסך על פי הדרישות שלו למציאת קשר חברי או זוגי. לאחר מילוי הפרטים המשתמש לוחץ על הכפתור find me a match! וזה מחזיר לו את רשימת החברים שעומדים בקריטריונים שהוא סימן עם הפרטים הבאים: תמונת פרופיל, סטטוס , כתובת מגורים, מספר לייקים שהחבר עשה למשתמש על כל התמונות, חברים משותפים ורשימת צ'ק-אין משותפים.

הערות: חברים שאין לך עיר מגורים יוחזרו עבור כל עיר שהמשתמש יבחר.

סטטוס / עבודה / השכלה / לייקים - נתונים שלא ניתן לקבל מ API.

מחלקות שמעורבות בפיצ'ר:

DatingFeature

**Your luck:**

במסך יש 4 כפתורים:

Get My Sign Of zodiac – יוצג המזל של המשתמש על פי תאריך הלידה שלו. יוצג תיאור של המזל ותמונה.

Who is your Famous – יוצג תמנה של ופרטים על סלב שהמזל שלו זהה למזל של המשתמש.

Who has the same luck as your? – יוצג תמונה ושם של חבר מהרשימת חברים של המשתמש שהמזלות שלהם זהים.

Your soulmate – בהתאם למזל של המשתמש יוצג לו מזל שתואם לו עם הסבר ותמונה.

בנוסף, הרחבנו את הפיצ'ר על ידי כך שלמשתמש יש אופציה סינון חברים על ידי סוג מזל.

מחלקות שמעורבות בפיצ'ר:

YourLuckFeature

### תבנית מס' 1 – [Iterator]

* סיבת הבחירה / שימוש בתבנית:

השתמשנו בתבנית העיצוב Iterator Pattern , בנינו מחלקה בשם Checkins שיש בה אוסף של מיקומים שהמשתמש ביקר בהם.

נרצה לספק ל-Client בתוכנית שלנו (או לכל מחלקה אחרת), אוסף מיקומים זה מבלי לתת לו את האפשרות להוריד או להוסיף מיקומים לאוסף זה תוך שמירת חופש הבחירה למחלקה לבחור איזה מבנה נתונים ברצונה לממש את אוסף המיקומים שלה בעתיד, ובכך לא נצטרך לשנות את הקוד אצל ה Client.

* אופן המימוש:

בנינו ממשק בשם IIterator שמכיל את המתודות וה properties הבאים:

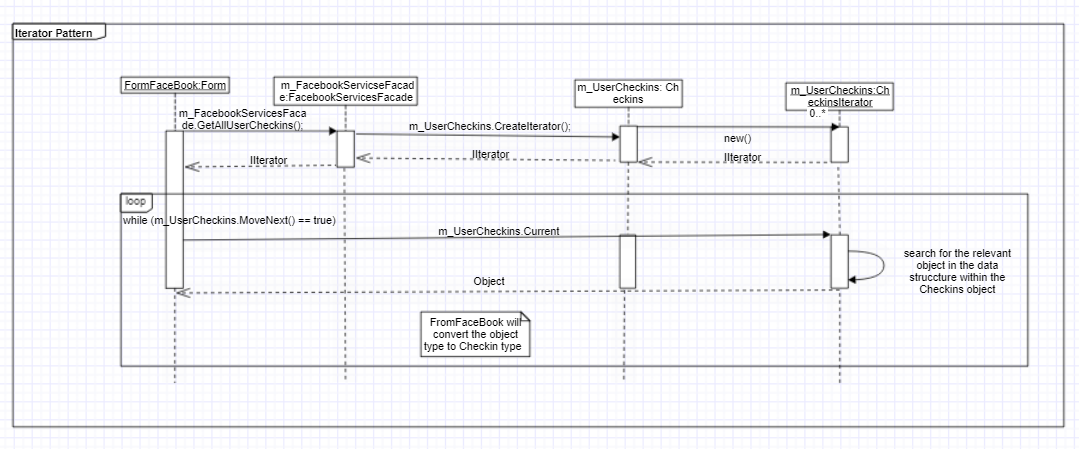
a. Current = החזר את האיבר בו אנו נמצאים כרגע במבנה

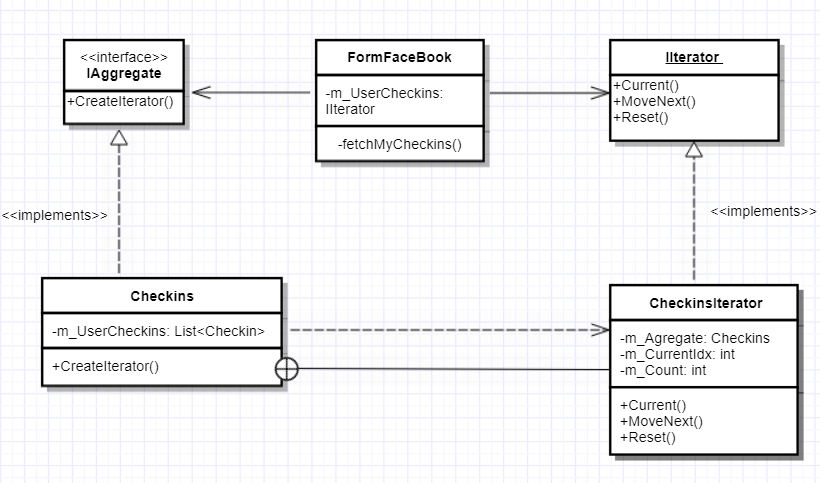
b. MoveNext = התקדם לאיבר הבא במבנה הנתונים

1. Reset = חזור לתחילת מבנה הנתונים

וממשק IAggregate שמכיל את המתודה () CreateIterator. שתפקידה להחזיר ל Client אובייקט מסוג CheckinsIterator , אשר יורש מIIterator וממומש בתור Nested Class בתוך Checkins, וממש את המתודות לסריקה המבנה הנתונים.

אנחנו שולחים איטראטור שמכיל את אוסף ה Checkins מבלי לחשוף ל Clientאת מבנה הנתונים של האוסף.

* Sequence Diagram
* Class Diagram



### תבנית מס' 2 – [Strategy]

* סיבת הבחירה / שימוש בתבנית:

בחרנו בתבנית עיצוב Strategy על מנת ליצור סינון של החברים על פי המזל שלהם בפיצ'ר YourLuck. המשתמש בוחר מתוך ה listView שמוצג לו את המזל שאיתו הוא רוצה לסנן את החברים שלו , והוא מקבל חזרה רשימה של החברים בעלי המזל שהוא בחר.

בעזרת שימוש בתבנית עיצוב זו יצרנו סינון לחברים , ואם בעתיד נרצה לשנות או להחליף את הדרך בה אנחנו מסננים חברים או להוסיף שיטות סינון נוספות נוכל לממשק את הinterface IFriendsBy ולהציג רשימת חברים על פי הסינון שיצרנו.

* אופן המימוש:
* יצרנו ממשק בשם IFriendsBy בעל חתימה של מתודה אחת: bool

friendsBy(string zodiacSign); . המתודה מקבלת string ומחזירה true/false.

* יצרנו 12 מחלקות – מחלקה עבור מזל שמממשות את הממשק IFriendsBy:

FriendsByAquarius – מסנן את החברים על פי המזל "Aquarius"

FriendsByAries – מסנן את החברים על פי המזל "Aries"

FriendsByCancer – מסנן את החברים על פי המזל "Cancer"

FriendsByCapricorn – מסנן את החברים על פי המזל "Capricorn"

FriendsByGemini – מסנן את החברים על פי המזל "Gemini"

FriendsByLeo – מסנן את החברים על פי המזל "Leo"

FriendsByLibra – מסנן את החברים על פי המזל "Libra"

FriendsByPisces – מסנן את החברים על פי המזל "Pisces"

FriendsBySagittarius – מסנן את החברים על פי המזל "Sagittarius"

FriendsByScorpio – מסנן את החברים על פי המזל "Scorpio"

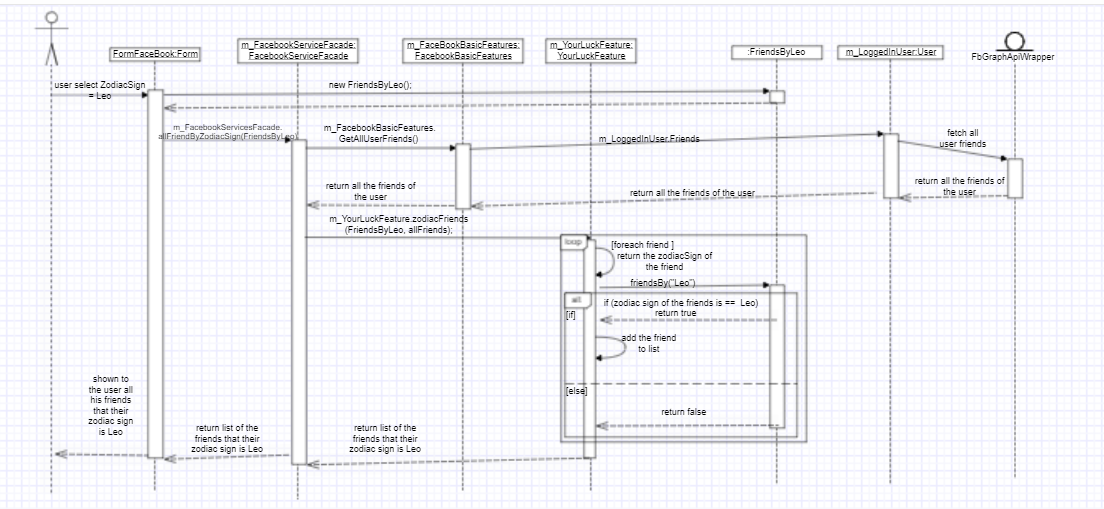
FriendsByTaurus – מסנן את החברים על פי המזל "Taurus"

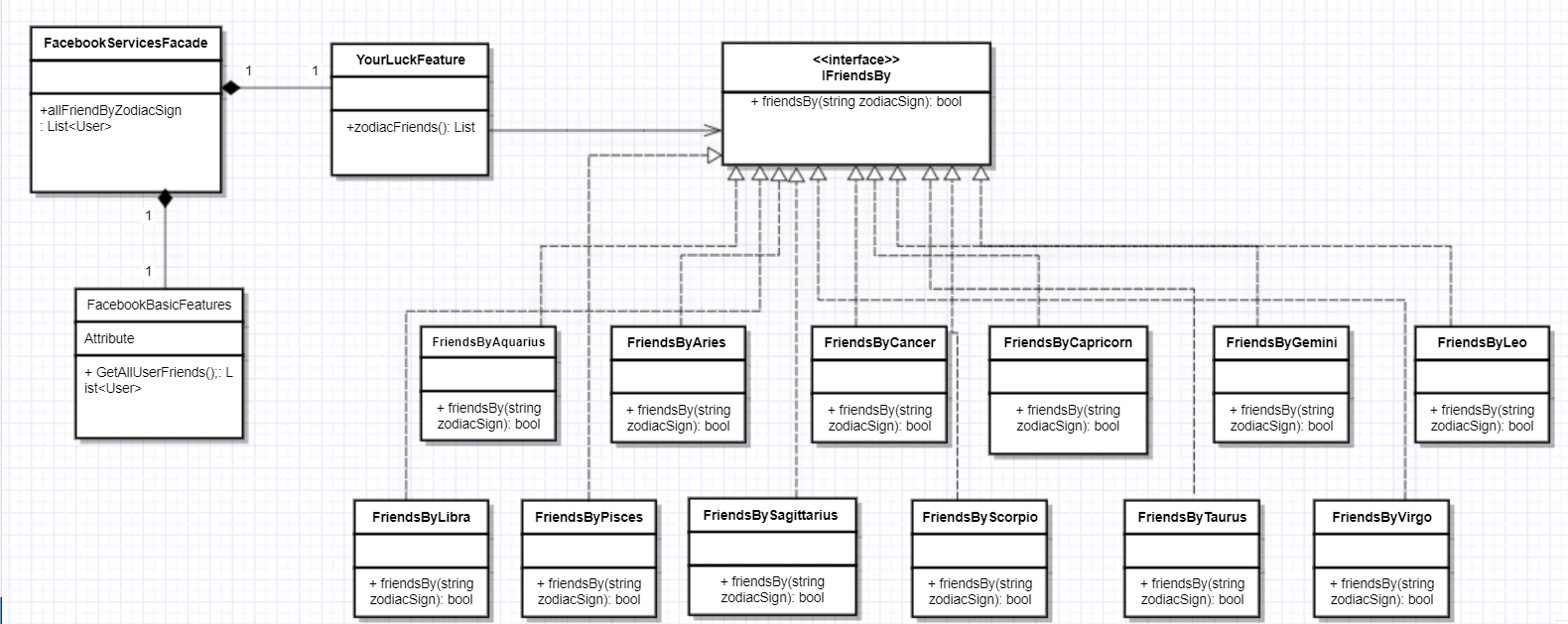
FriendsByVirgo – מסנן את החברים על פי המזל "Virgo"

* כאשר היוזר בוחר במזל איתו הוא רוצה לסנן את רשימת החברים שלו , מתבצעת קריאה למתודה allFriendByZodiacSign(IFriendsBy i\_FriendsBy) שנמצאת ב Facade. מתוך הפונקציה הזאת מתבצעות 2 קריאות – הקריאה הראשונה לפונקציה שמחזירה את כל החברים של המשתמש שהתחבר לאפליקציה , הקריאה השנייה היא לפונקציה שבעזרתה מתבצע סינון החברים -

zodiacFriends(IFriendsBy i\_FriendsBy,List<User> i\_allFriends) - על ידי קריאה למתודה friendsBy מתוך פונקציה זו מתבצע סינון החברים ומוחזרת לUI רשימת החברים בעלי המזל הנבחר.

* Sequence Diagram
* הדיאגרמה מתארת אסטרטגיית מיון אחת – על ידי מזל Leo . אך שאר סוגי המיון מתבצעים באופן זהה – למעט שינוי האסטרטגיה.



* Class Diagram

### תבנית מס' 3 – [Observer]

* סיבת הבחירה / שימוש בתבנית:

בחרנו בתבנית עיצוב Observer על מנת ליידע את כל המחלקות הרלוונטיות בעת התחברות יוזר חדש לאפליקציה שלנו.

כאשר יוזר מבצע התחברות לאפליקציה , אנחנו ניידע את המחלקות מי היוזר החדש שהתחבר ובעזרת זה היא תוכל לקנפג את כל הנתונים באפליקציה בהתאם ליוזר שהתחבר.

בעזרת תבנית זו והממשק שיצרנו אנחנו מונעים את ההיכרות הציקלית בין כל אחת מהמחלקות המאזינות ליוזר, אין תלות קיום של היוזר בכל אחת מהמחלקות .

וכעת בעתיד אם נרצה ליידע מחלקות נוספות על התחברות יוזר למערכת , נוכל להוסיף אותם לרשימת ה"מאזינים", כמו כן ניתן יהיה להסיר "מאזינים" מתוך הרשימה.

* אופן המימוש:
* יצרנו ממשק בשם ILoggedInUserObservers בעל חתימה של מתודה אחת: void ReportUserLoggedIn(User i\_LoggedInUser); - תפקידה של מתודה זו היא לבצע את העדכון במחלקה הרלוונטית (ה-observer)

-יצרנו מחלקה בשם FaceBookUser שמכילה רפרנס לUser (של FacebookWrapper ) ורשימת "מאזינים " , מכילה בתוכה 4 מתודות:

AttachObserver(ILoggedInUserObservers i\_UserObservers) - – הוספת "מאזין" לרשימת ה"מאזינים"

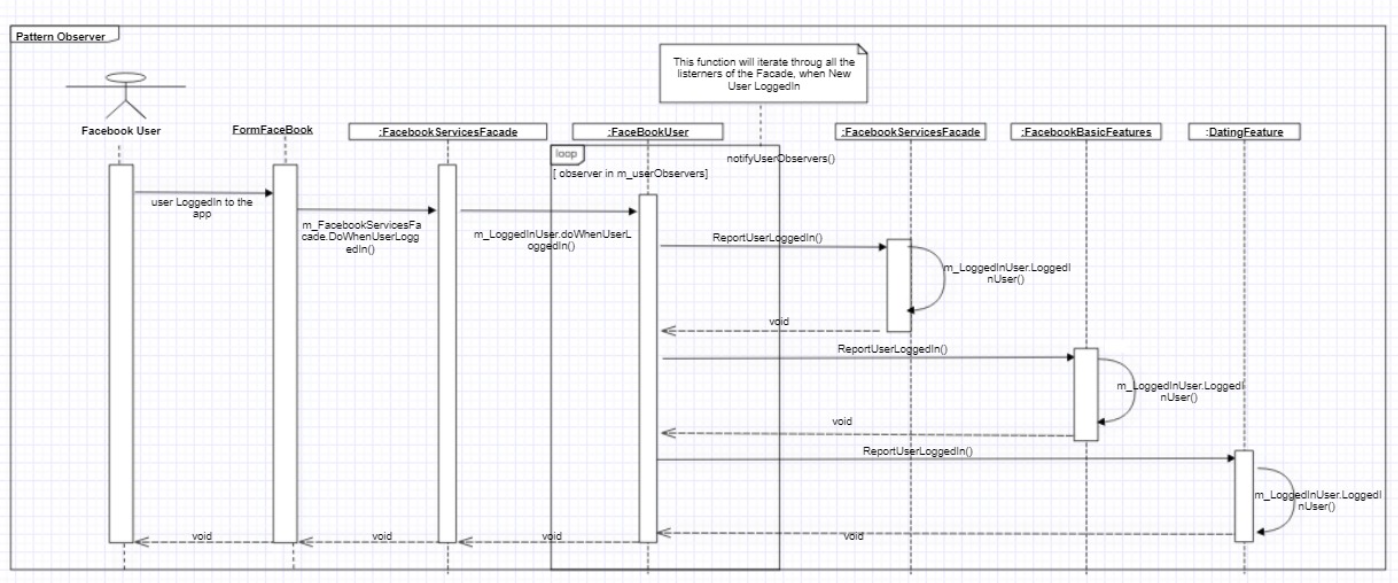
DetachObserver(ILoggedInUserObservers i\_UserObservers) - – הסרת "מאזין" מרשימת ה"מאזינים"

doWhenUserLoggedIn(User i\_LoggedInUser) - – פונקציה אשר תיקרא כאשר יוזר התחבר למערכת . היא מבצעת את עדכון היוזר ויידוע של המחלקות המאזינות על היוזר שהתחבר.

notifyUserObservers() - – מיידעת את המאזינים שיוזר התחבר לאפליקציה.

3 המחלקות ה"מאזינות" (ה-observers) להתחברות יוזר לאפליקציה הן: FacebookBasicFeatures , FacebookBasicFeatures , DatingFeature. 3 מחלקות אלה מממשות את הממשק ILoggedInUserObservers ואת המתודה ReportUserLoggedIn, ומחזיקות רפרנס ל FaceBookUser().

כאשר יוזר יבצע התחברות למערכת , תתבצע קריאה למתודה DoWhenUserLoggedIn(m\_LoggedInUser) שאחראית לבצע יידוע לכל המאזינים שיוזר התחבר למערכת ואתחול.

* Sequence Diagram
* Class Diagram

