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

•FrienDiscover - האופציה להכיר חברים חדשים דרך חברים קיימים:

המערכת תגריל חבר קיים של המשתמש הרשום, ומתוך רשימת החברים של החבר שהוגרל נגריל משתמש זר למשתמש הרשום.

במסך האפליקציה יופיעו פרטיו של המשתמש השני שהוגרל, הכוללים: תמונת פרופיל, גיל ואודות.

המשתמש הרשום יכול לבחור האם לבקר בפרופיל של המשתמש המוגרל השני במטרה להוסיפו כחבר או לבצע הגרלה מחדש לפגישת משתמש אחר.

ניתן למצוא פיצ'ר זה ב- FrienDiscoverForm.cs.

•סטטיסטיקה על אופן השימוש של המשתמש הרשום בפייסבוק:

המערכת מציעה שני סוגי סטטיסטיקות להצגה:

1. התפלגות המדינות בהם המשתמש ביצע checkin.

2. התפלגות תיוג החברים של המשתמש בפוסטים שלו.

המידע מוצג ע"י תרשימי עוגה שמראים את התפלגות הנתונים השונים.

בנוסף, ניתן להוסיף עוד סטטיסטיקות בעתיד, וזאת הודות לרכיב התכנות האחראי על מילוי הנתונים בתרשים העוגה - PieChartDataLoadingComponent.cs.

ניתן למצוא פיצ'ר זה ב- StatisticsForm.cs.

### תבנית מס' 1 – [Observer]

* סיבת הבחירה / שימוש בתבנית:

עקב טבע המידע שאיתו אנו עובדים (מתעדכן באופן תכוף), החלטנו להוסיף אפשרות רענון עבור הטופס הנוכחי המוצג ביישום.

הוספנו לממשק המשתמש כפתור רענון, המהווה מיידע (notifier) כאשר מספר טפסים (forms) במערכת מאזינים (listeners) לכפתור הרענון. לכן תבנית העיצוב המתאימה למצב זה היא תבנית העיצוב Observer.

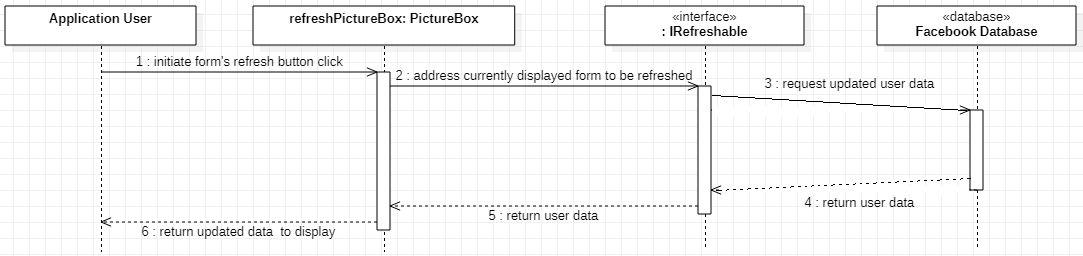
* אופן המימוש:

הרעיון בו השתמשנו הוא כפתור רענון יחיד (notifier), המודיע על לחיצה מהמשתמש עבור רענון הטופס הנוכחי (listener).

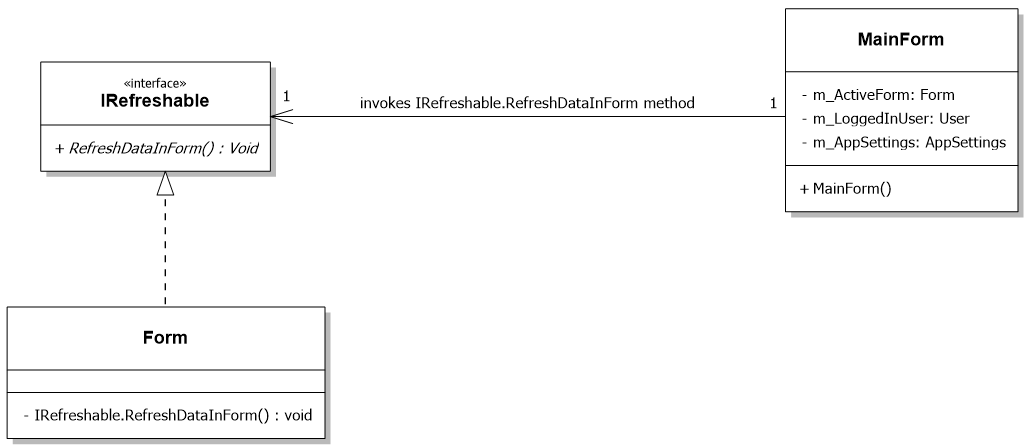
הגדרנו ממשק IRefreshable, שאותו יממשו הטפסים אשר ניתנים לרענון.

בכל לחיצה על כפתור הרענון, תתבצע הפעלה של מתודת הממשק בטופס המוצג, שמטרתה היא למשוך נתונים עדכניים משרתי פייסבוק ולהציגם בממשק המשתמש.

* Sequence Diagram



* Class Diagram



פירוט מחלקות

* המחלקה MainForm מהווה את הטופס הראשי במערכת, בו נמצא כפתור הרענון (notifier).
* הממשק IRefreshable מכיל מתודה RefreshDataInForm, כך שכל טופס שירצה להיות ניתן לרענון יצטרך לממש ממשק זה.
* המחלקה Form מהווה שם כללי לכלל הטפסים הניתנים לרענון, המממשים את הממשק IRefreshable ומהווים מאזינים ללחיצה על כפתור הרענון. לאחר לחיצה על כפתור הרענון, תופעל מתודת הרענון של הטופס הנוכחי.

### תבנית מס' 2 – [Strategy]

* סיבת הבחירה / שימוש בתבנית:

כפי שרשום בתיאור הפיצ'רים, במערכת שלנו קיימת אפשרות לראות סטטיסטיקות שונות לגבי פרטים שונים אודות פעילות המשתמש בפייסבוק.  
כדי להציג סטטיסטיקה מסוימת, המערכת נדרשת לבצע חישוב לפי תנאי מסוים, למשל לבדוק התפלגות חברים המתויגים בפוסטים של המשתמש או לבדוק בכמה מדינות המשתמש ביצע checkin.  
העובדה, שכל סטטיסטיקה מופקת באמצעות חישוב של פרמטרים המתקבלים ע"י תנאי שונה, מעלה את הצורך בשינוי אופן ההתנהגות של הרכיב האחראי על הפקת נתוני הסטטיסטיקה ולכן השתמשנו בתבנית העיצוב Strategy.

* אופן המימוש:

במערכת שלנו, קיימת מחלקה האחראית על הפקת נתוני הסטטיסטיקה (PropertyCountCalculator) המכילה שדה שהוא מצביע למתודה מסוג Func הנקרא PropertyCountConditionStrategyMethod.

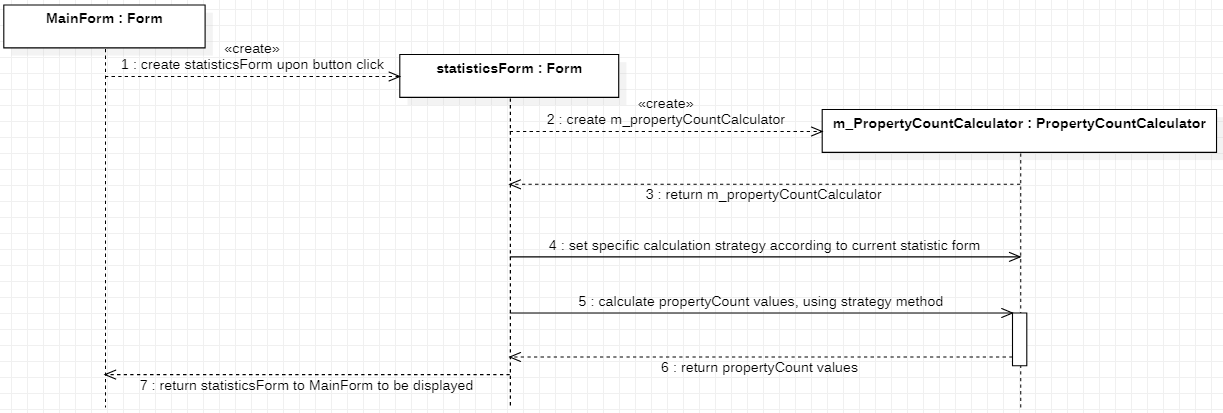
בעת הפקת נתוני הסטטיסטיקה, מופעלת המתודה CalculatePropertyCountValues המבצעת מניית פרמטרים מתאימים לפי תנאי של הסטטיסטיקה הרצויה, ומפעילה מתודה שמייצגת את התנאי, ששדה PropertyCountConditionStrategyMethod מצביע אליה.

בנוסף במערכת שלנו, קיים טופס סטטיסטיקה (StatisticsForm) האחראי על הצגת הסטטיסטיקות בממשק המשתמש.

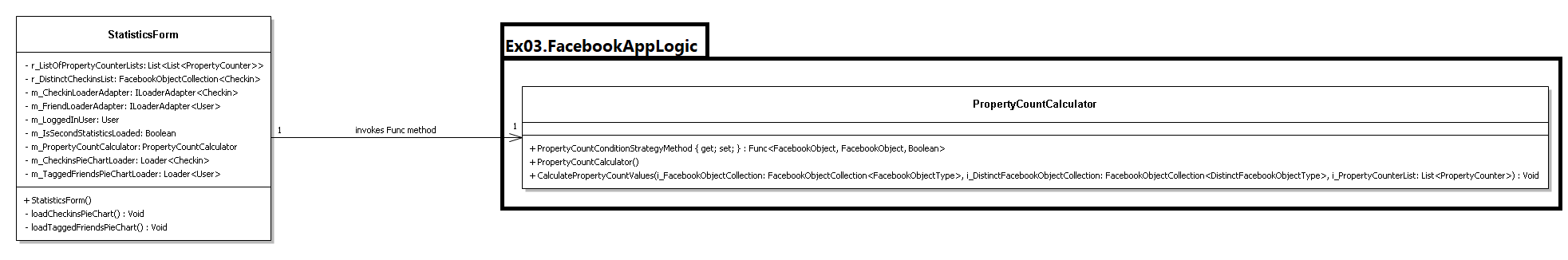
טופס זה משנה את תנאי הסטטיסטיקה לפי הסטטיסטיקה שהמשתמש בחר, וכך בעצם משנה את אופן ההתנהגות של הרכיב PropertyCountCalculator אשר מפיק נתוני סטטיסטיקה בהתאם לתנאי הנוכחי.

הטופס StatisticsForm מבצע השמה של מתודת בדיקה ע"י שימוש בביטוי lambda וכך אנו מממשים strategy method.

* Sequence Diagram



* Class Diagram



פירוט מחלקות

* המחלקה StatisticsForm אחראית על הצגת הסטטיסטיקות בממשק המשתמש.
* המחלקה PropertyCountCalculator אחראית על הפקת נתוני סטטיסטיקה לפי מתודת תנאי שהוזנה לה לפני ביצוע החישוב.

### תבנית מס' 3 – [Visitor]

* סיבת הבחירה / שימוש בתבנית:

כפי שנאמר בתיאור הפיצ'רים, המערכת שלנו מציעה למשתמש להכיר חברים חדשים דרך חברים קיימים.  
לשם כך, קיים רכיב לוגי במערכת (FrienDiscoverLogicComponent) הבוחר חבר חדש מתוך רשימת החברים של חבר קיים של המשתמש הרשום.

החלטנו להפוך את הרכיב ל- reusableיותר ולנתק אותו מהקונטקסט של בחירת משתמשים באפליקציית פייסבוק לכדי בעיה כללית יותר - בחירת אובייקט מתוך מאגר אובייקטים על פי צורך יוזם הבקשה, ובכך הפכנו את הרכיב לכללי יותר.

לשם כך, החלטנו לממש את תבנית העיצוב Visitor.

בנוסף, אם בעתיד נרצה לממש מערכת אחרת שצריכה לבחור אובייקט מתוך מאגר אובייקטים על פי צורך יוזם הבקשה, נוכל ישירות להשתמש במימוש שכתבנו עבור אפליקציית הפייסבוק שכן הוא כללי וניתן לשימוש חוזר.

* אופן המימוש:  
  יצרנו ממשק הנקרא ISelectable<T>, שאותו כל מחלקה, שרוצה להשתמש ברכיב בחירת אובייקט ממאגר אובייקטים, צריכה לממש.  
  ממשק זה מכיל שני properties:  
  1. SelectableObjectsList: רשימת אובייקטים מסוג T, מדרגה ראשונה, הקשורים ליוזם הבקשה.   
  במערכת שלנו, מדובר על המשתמש הרשום ועל רשימת חבריו.

2. SelectedObjectLists: רשימת רשימות אובייקטים מסוג T, מדרגה שנייה, המהווה את מרחב האובייקטים הקשורים לרשימת האובייקטים מהדרגה הראשונה (SelectableObjectsList).

במערכת שלנו, מדובר על רשימת רשימות החברים של החברים של המשתמש הקיים.

הרכיב "שמבקר" (visitor) במערכת שלנו, היא המחלקה Selector<T> אשר מהווה מימוש פתרון עבור הבעיה הכללית שתוארה בסעיף הקודם.

מחלקה זו מכילה מתודה ציבורית בשם PickFirstDegreeAndSecondDegreeObjects המקבלת אובייקט מסוג ISelectable<T> ומחזירה by ref שני אובייקטים מסוג T המהווים את אובייקט הראשון שנבחר ואת האובייקט השני שנבחר.

מתודה זו מבצעת 2 בדיקות על האובייקטים במהלך פעולתה:

1. בדיקה האם האובייקט השני הוא יוזם הבקשה.

במערכת שלנו, יוזם הבקשה הוא המשתמש הרשום והאובייקט השני הוא חבר חדש פוטנציאלי. לא נרצה שהחבר החדש שיבחר הוא המשתמש הרשום עצמו.

1. בדיקה האם האובייקט השני מקושר ליוזם הבקשה.

במערכת שלנו, לא נרצה להציע למשתמש חבר חדש שהוא בדיעבד חבר קיים של המשתמש הרשום, לכן נבדוק שלא קיים קשר חברות ביניהם.

על מנת לשמור על כלליות המחלקה, לאור התנאים המשתנים בתסריטי השימוש השונים של המחלקה Selector<T>, החלטנו להשתמש בתבנית העיצוב Strategy במחלקת Selector<T>, ובכך אפשר להזין תנאי בדיקה ייחודים עבור כל מקרי השימוש במחלקה.

לשם מימוש תבנית העיצוב Strategy, המחלקה Selector<T> מכילה שני שדות מסוג Func<T, bool>.

### תבנית מס' 3 – [Visitor]

* Sequence Diagram
* Class Diagram

פירוט מחלקות