תיכנות יישומי בעזרת Design Patternsסמסטר ג' תשע"ח, תרגיל 3גיא רונן ©

מגישים: זוהר קליין, 204456818 , יהונתן שער 201582962

במטלה 1 בחרנו לממש את הפיצ'רים הבאים:

* **User Categorization** - פיצ'ר זה מאפשר למשתמש האפליקציה לראות את החברים שלו בפייסבוק על פי כמות חברים בעלי סטטוס מערכת יחסים של נשואים/רווקים ובנוסף רשימת חברים ממויינים ע"י מין של גברים/נשים.
* **Filter Images** - פיצ'ר זה מאפשר למשתמש האפליקציה לבחור מתוך מגוון אפשרויות פילטר לתמונת פרופיל, כגון:
* Original.
* Grayscale.
* Vintage.
* Transparency.
* Negative.

במטלה 2 בחרנו לממש את הפיצ'רים הבאים:

* **Manage Liked Pages**- פיצ'ר זה מאפשר למשתמש האפליקציה לנהל את הדפים שהוא סימן ב"אהבתי" תחת קטגוריות/תיקיות שונות ואף ליצור תיקיות בתוך תיקיות אשר מכילות את הדפים שהוא אהב.

בנוסף, המשתמש יכול לראות מידע אודות הדפים שהוא אהב.

* **Events Handling**- פיצ'ר זה מאפשר למשתמש האפליקציה לנהל את האירועים שיש לו בפרופיל הפייסבוק האישי שלו בכך שהוא יכול לשנות את שמם, לראות מידע אודות האירוע ואת הזמן בו הוא מתרחש.

חשוב לומר כי השינויים אודות האירועים יבוצעו אך ורק באפליקציית DiscoverMe שיצרנו ולא באפליקציית Facebook.

### תבנית מס' 1 – [Iterator]

* תבנית העיצוב Iterator היא תבנית עיצוב אשר מטרתה לספק לנו דרך לגשת לאלמנטים של אובייקטים מקובצים ללא חשיפת הייצוג הבסיסי שלהם.
* סיבת הבחירה בתבנית העיצוב Iterator הינה בכדי לבצע מעבר יעיל על אובייקטים שונים.

בפרויקט שלנו היה חשוב לעשות זאת בכדי לאפשר ל"שחקן" (UserCategorization) אפשרות לעבור על האובייקטים של רשימת החברים שהגבלנו אותה לפי צרכיו (בהמשך נרחיב). בנוסף, השחקן יבצע מעבר על הרשימה ללא שום ידע מהו מבנה הנתונים של אותה רשימה. דבר זה יבטיח לנו Reusable של הקוד בעתיד.

* בתוכנית שלנו מימשנו 2 מחלקות אשר נותנות מענה למימוש תבנית העיצוב:
* **DiscoverMeUser:**

מחלקה בעלת Properties מסויימים, מתוך כל ה Properties שיש למשתמש פייסבוק, שאותם ורק אותם נחשוף למי שישתמש באיטרטור הזה.

ה Properties הם:

- FirstName

- LastName

- Gender

- RelationshipStatus

- Birthday

- PictureNormalURL

כאשר מחלקה כלשהי תרצה להשתמש בשירות של Iterator בעזרת foreach, היא תגלה שרק ה Properties הנ"ל חשופים ולא תהיה לה גישה למידע נוסף אודות החברים.

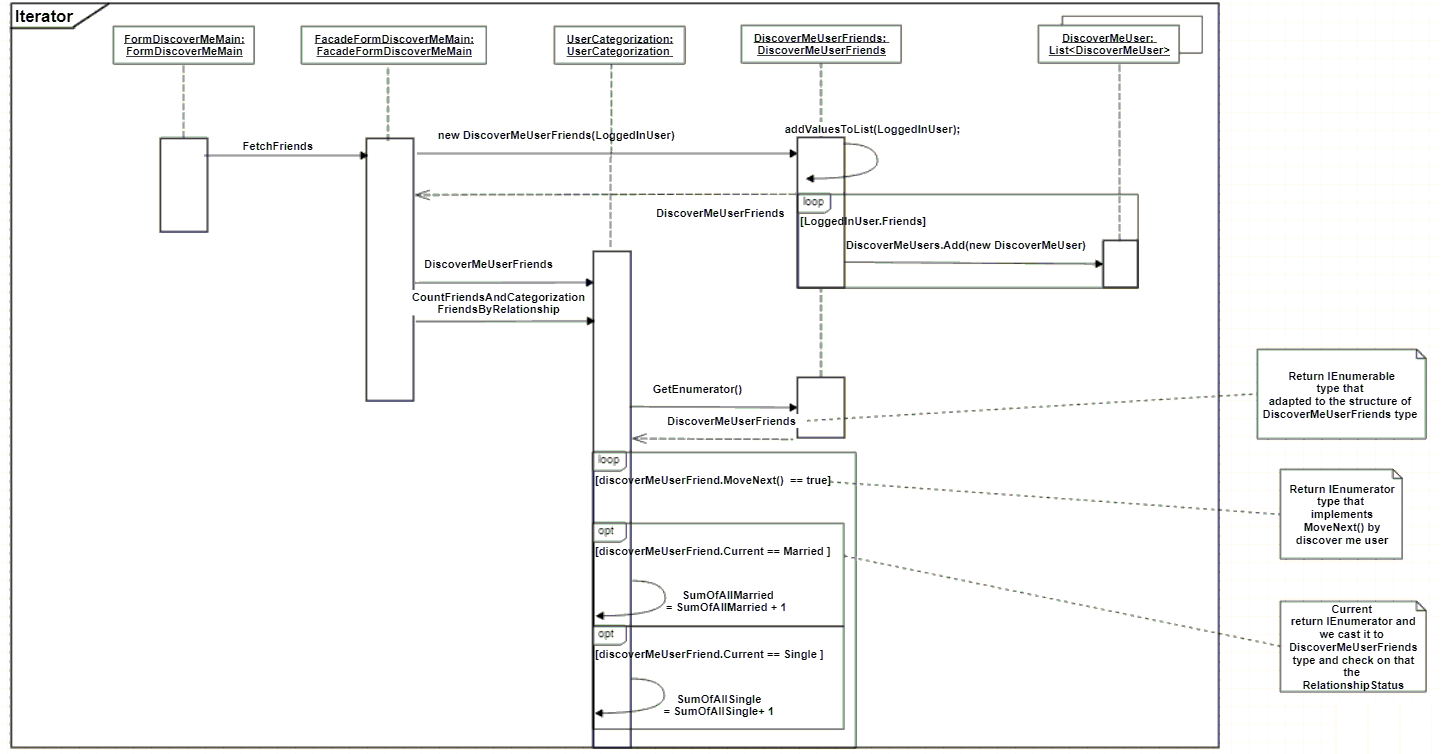
* **DiscoverMeUserFriends:**

זו מחלקת ה ConcreteAggregate של התבנית עיצוב (כבר יש לנו IAggregate בעזרת הממשק ה C# שנקרא IEnumerable).

מחלקה זו יוצרת רשימה (List) מסוג DiscoverMeUser ומוסיפה אליה את כל ה Properties המתאימים של החברים.

לבסוף, המחלקה מספקת איטרטור למי שיחפוץ להשתמש בה ובכך יהיה ניתן "לרוץ" על הרשימה ולשלוף נתונים.

* **Sequence Diagram:**



* **:Class Diagram**

### 

### תבנית מס' 2 – [Chain Of Responsibility]

* תבנית העיצוב Chain Of Responsibility היא תבנית עיצוב אשר מטרתה היא מניעת קשר בין השולח של הבקשה למקבל שלה על ידי כך שיותר מאובייקט אחד יוכל לטפל בבקשה.
* סיבת הבחירה בתבנית העיצוב Chain Of Responsibility הינה בכדי לאפשר ל Client לשלוח בקשה לשרשרת כלשהי והיא זו שתעביר את הבקשה חוליה – חוליה עד שתגיע ליעד הרצוי. זאת במקום שה Client יצטרך להכיר את כל מחלקות המערכת כדי לנהל את הבקשה שלו.

בפרויקט שלנו היה חשוב לעשות זאת בכדי לאפשר ל"שחקן" (FormDiscoverMeMain) לשלוח אובייקט מסויים אשר עובר בשרשרת של חוליות עד שמגיע למחלקה אשר אליה האובייקט רצה להגיע ובכך לא צריך לשלוח את האובייקט למיקום ספציפי.

כך, כאשר משתמש האפליקציה DiscoverMe בחר לראות דבר מסויים, המחלקה FormDiscoverMeMain שלחה את בקשתו לשרשרת, הבקשה רצה במורד החוליות של השרשרת עד שהגיעה למקום שאליו המשתמש ביקש להגיע והחזירה את התוצאה הרצויה.

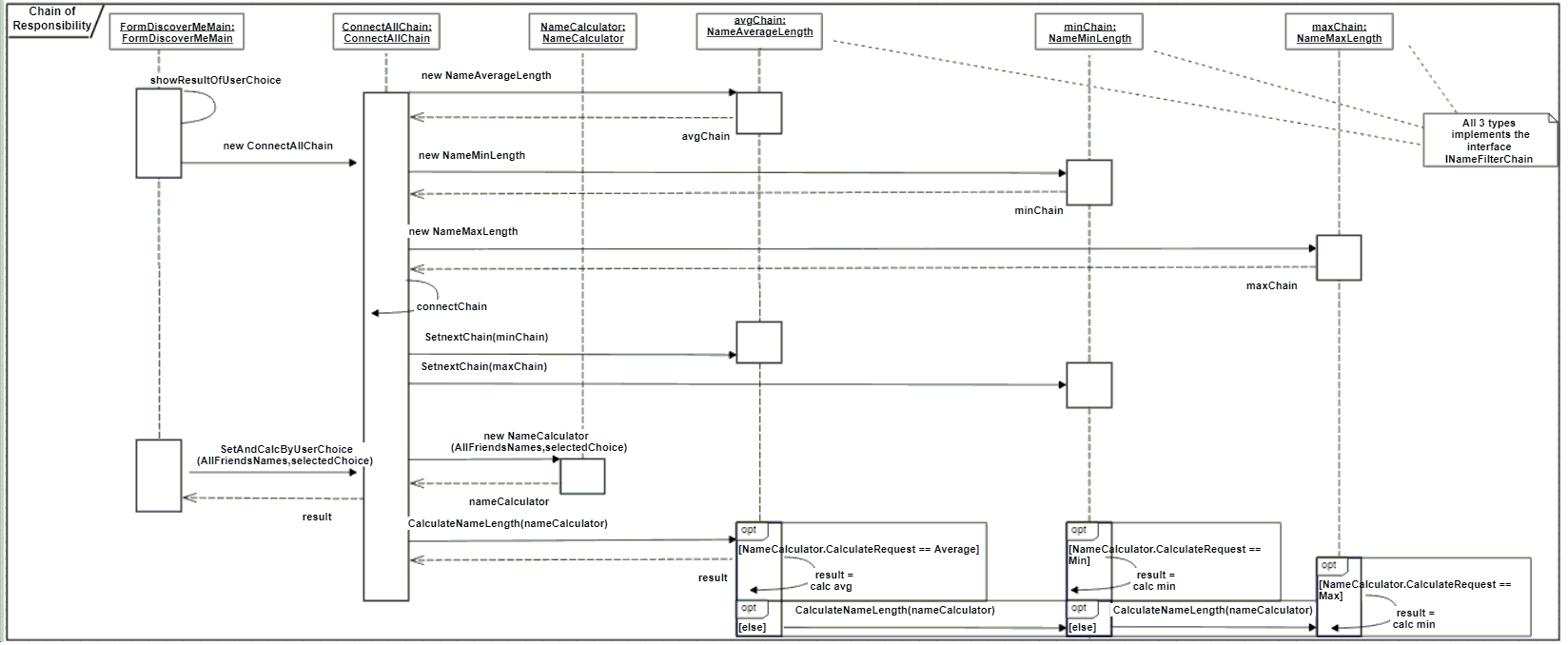
* בתוכנית שלנו מימשנו ממשק בשם INameFilterChain שאותו כל חוליה (מחלקה) בשרשרת מממשת. המחלקות הן:
* NameAverageLength
* NameMaxLength
* NameMinLength

כל אחת מהמחלקות מממשות את INameFilterChain לפי האלגוריתם שלה.

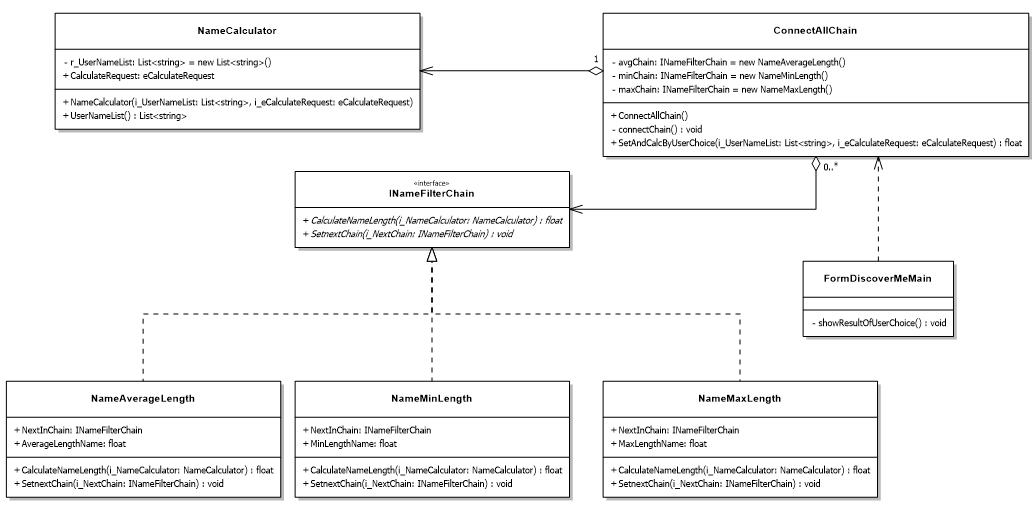
לאחר מכן, מימשנו את המחלקה NameCalculator שתפקידה ליצור את הבקשה שה Client ביקש לבצע.

לבסוף, יצרנו את המחלקה ConnectAllChain האחראית לחיבור ויצירת כל החוליות בשרשרת ושליחת הבקשה לחוליה הראשונה שהיא זו שתעביר את הבקשה לחוליה השניה וכן הלאה עד שתגיע ליעד הרצוי ע"י כך שתתבצע בדיקה בכל חוליה האם הבקשה זהה לתפקיד של החוליה שאליה היא הגיעה.

* **:Sequence Diagram**



* **Class Diagram:**



### תבנית מס' 3 – [Strategy]

* תבנית העיצוב Strategy היא תבנית עיצוב המגדירה משפחה של אלגוריתמים, מכמסת כל אחד מהם, ומאפשרת להחליף את השימוש בהם. ניתן לשנות את האלגוריתם באופן עצמאי בלי תלות במשתמשים במחלקה.
* סיבת הבחירה בתבנית העיצוב Strategy הינה בכדי לקנפג מחלקה לעבודה עם מספר רב של התנהגויות.

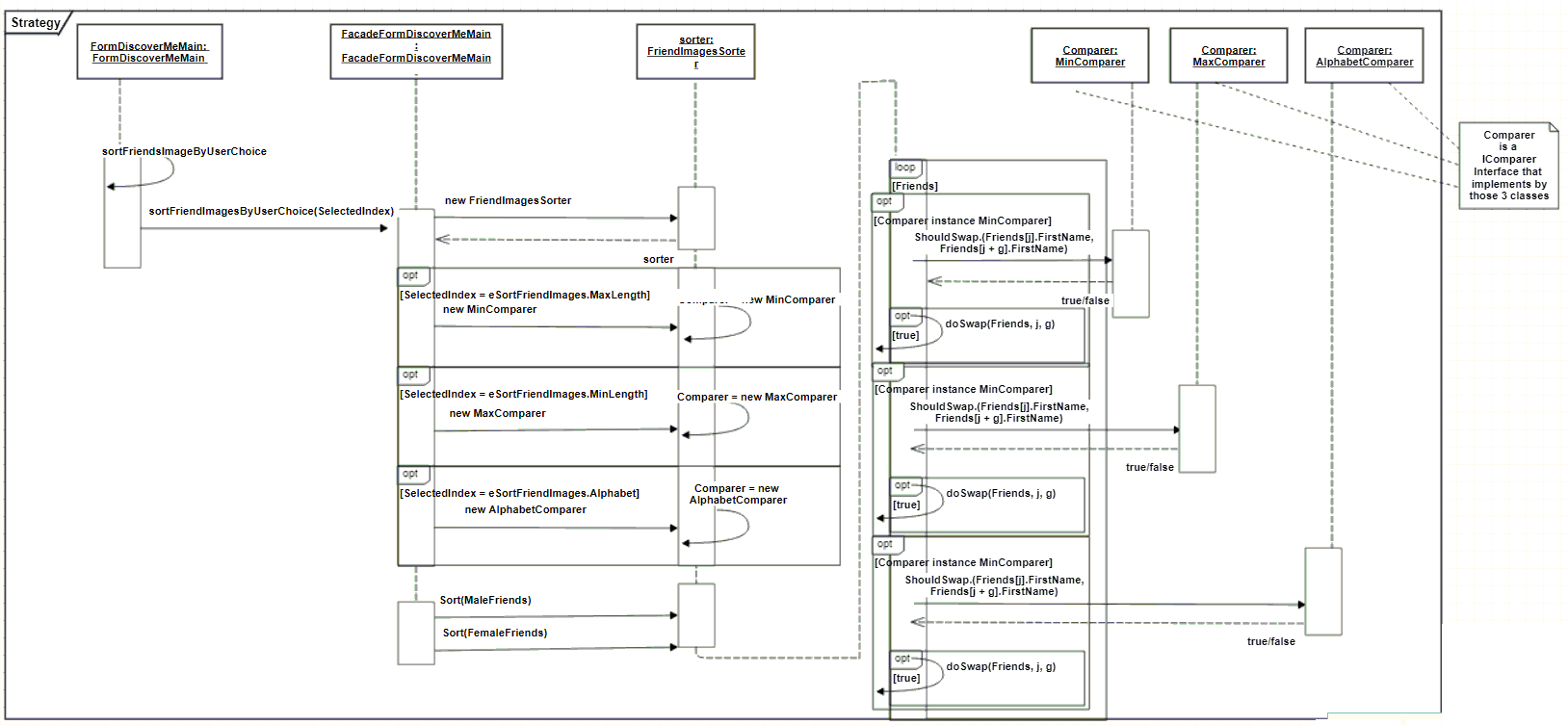
בפרויקט שלנו היה חשוב לעשות זאת בכדי שה"שחקן" (FacadeFormDiscoverMeMain) יוכל לבקש מיון מחדש של רשימת החברים לפי מגדר של ה Client ללא צורך לעבוד עם מספר מחלקות שונות – אלא רק עם מחלקה אחת בעלת אלגוריתם כללי שבו יהיה ה Injection Point שיקבע האם יש לבצע שינוי במיון או לא (נסביר בהרחבה בהמשך).

* בתוכנית שלנו מימשנו ממשק בשם IComparer המכיל מטודה אחת הנקראית ShouldSwap. ישנן 3 מחלקות המממשות את הממשק IComparer:
* MaxComparer
* MinComparer
* AlphabetComparer

כאשר כל אחת מהמחלקות הנ"ל מממשות את הממשק IComparer הן צריכות גם לממש את המטודה ShouldSwap המחזירה True או False כדי לדעת האם יש לבצע החלפה בין אובייקטים כדי למיין את הרשימת חברים.

לבסוף, יצרנו את המחלקה FriendImagesSorter שבה יש את המטודה Sort בעלת אלגוריתם כללי ובתוך האלגוריתם יש את ה Injection Point שקובע לאיזו מחלקה יש לגשת (MaxComparer,MinComparer,AlphabetComparer) כדי לבדוק האם יש לבצע מיון.

* **:Sequence Diagram**



* **Class Diagram:**

