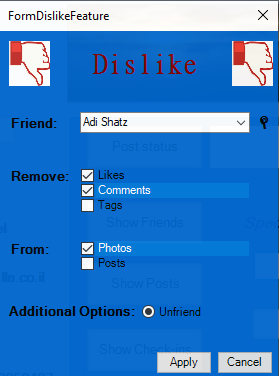
### תיאור קצר של הפיצ'רים שבחרנו לממש בתרגיל הקודם:

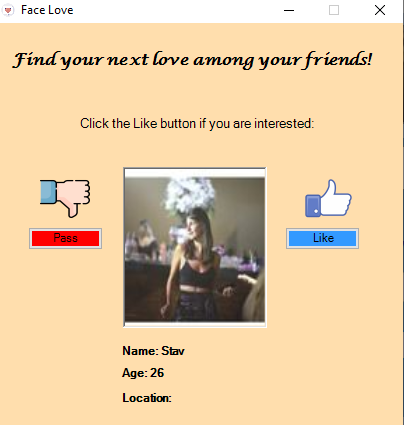
**feature FaceDislike**   
פיצ׳ר שמאפשר למשתמש לאתר משתמש מרשימת החברים שלו ולבצע כל אחת מן הפעולות הבאות:

* ביטול לייקים על תמונות\פוסטים של החבר.
* מחיקת תגובות על תמונות\פוסטים של החבר.
* הסרת תיוגים מתמונות\פוסטים משותפים עם החבר.
* אופציה לביטול החברות עם אותו המשתמש

**FaceLove feature**

פי'צר שנותן למשתמש פלטפורמת הכרויות עם חבריו לפסייבוק.

המשתמש בוחר את המין בו הוא מעוניין ואת טווח הגילאים, הפיצ'ר מציג לו אחד אחרי השני את רשימת החברים העונים על הקריטריונים שבחר, והמשתמש יכול ללחוץ "like" או “pass” כאשר לחיצה על “like” נותנת למשתמש אפשרות לשלוח הודעה אישית לחבר שאהב.



### תבנית מס' 1 Strategy -

* **סיבת הבחירה / שימוש בתבנית:**

מימשנו את פיצ׳רFaceLove כך שלמשתמש יש אפשרות לסנן את החברים שהוא מעוניין להכיר על ידי טווח גילאים ומין.

חשבנו שאם בעתיד נרצה לתת למשתמש אופציות אחרות לסינון החברים, למשל: חברים הגרים באותו איזור מגורים שלי, חברים הנולדו באותו החודש בו אני נולדתי ועוד', נרצה לבצע זאת במינימום קוד ותחזוקה ובכך הפיצ'ר שלנו יהיה בנו בצורה הנדסית כך שהוא ניתן לשינויים מהבחינה של אופן סינון החברים הרצויים.

**התבנית Strategy פותרת בעיה זו, ולכן בחרנו להשתמש בה:**

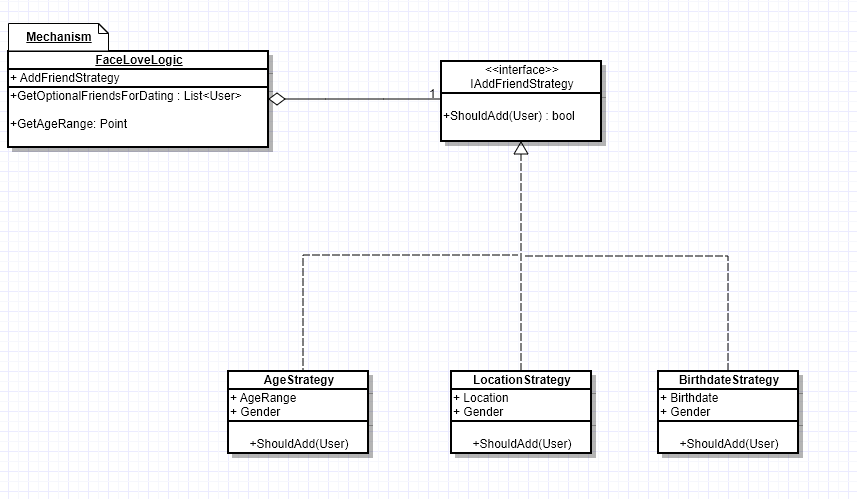
התבנית Strategy אומרת, במקום לממש את הרכיבים שתרצה אולי להחליף בעתיד במחלקת הbase (במקרה שלנו ב (FaceLoveLogic נגדיר בסיס פולימורפי IAddFriendStrategy ובו נגדיר את המתודות/החלקים שנרצה להחליף בעתיד.

העברנו את הinjection points מחוץ למחלקת הbase ובכך נוכל להחליף בקלות את האסטרטגייה הקיימת במינימום שינויים במחלקת הbase , נצטרך רק לשלוח לה אסטרטגייה חדשה שאותה היא מחזיקה בקומפוזיצייה.

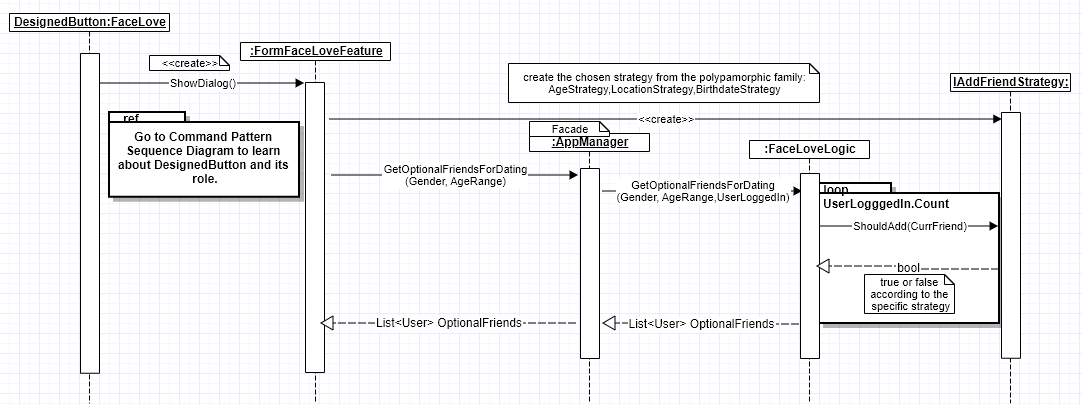
* **אופן המימוש**:
* מימוש IAddFriendStrategy Interface (בתפקיד ה**strategy**) ובה פונקציה בוליאנית ShouldAdd.
* 3 מחלקות: LocationStrategy, AgeStrategy, BirthdateStrategy, המממשות את IStrategy , כך שכל אחת מהמחלקות מייצגת אסטרגיה לבחירת חבר ומממשת את ShouldAdd בהתאמה לאופן האסטרטגיה.
* המחלקת הלוגית FaceLoveLogic (בתפקיד ה**mechanism**) מחזיקה ברפרנס את האסטרגיה IAddFriendStrategy הרצוייה שנוצרה על ידי טופס הפיצ'ר FormFaceLove ( בתפקיד ה **client**).

כעת כאשר נחליט בעתיד לשנות את אופן סינון החברים בפיצ'ר שלנו, מה שעלינו לעשות זה לממש אסטרטגיה חדשה המממשת את IAddFriendStrategy בצורה שנבחר ,לתת ל FaceLoveLogic (המכניזם) את האסטרטגיה שבחרנו, וכך נוכל לקבל רשימת חברים לפי כל פילטר שנבחר לממש במינימום כתיבה ותחזוקה של הקוד הקיים, מה שמאפשר גמישות של הפיצ'ר לשינויים ומגוון את הפעולות שאפשר לבצע בו.

**Class Diagram:**



**Sequence Diagram:**



### תבנית מס' 2 – Command

* **סיבת הבחירה / שימוש בתבנית:**

במסך הראשי של האפליקציה חילקנו ל-2 תפריטים (פאנלים) המאפשרים למשתמש לבצע את הפעולות הבאות:

* הצגת נתונים על חשבון הפייסבוק (רשימת חברים, רשימת פוסטים, רשימת צ'ק-אינים).
* שימוש בפיצ'ר המיוחדים שלנו (faceLove, Dislike, ChangeCover).

אם בעתיד נרצה להוסיף תפריטים חדשים או להרחיב את התפריטים הקיימים בפעולות נוספות נידרש לשנות את הקוד בטופס המציג את התפריטים הנ''ל מה שיגרור לתחזוקה רבה עבור כל הוספה/שינוי שכזה, בנוסף אם נרצה לבצע reuse לתפריט שבנינו עבור שימושים אחרים לא נוכל לבצע זאת במימוש שביצענו עד למטלה זו, כל אלה יחד מעלים בעיה הנדסית באפליקציה שלנו.

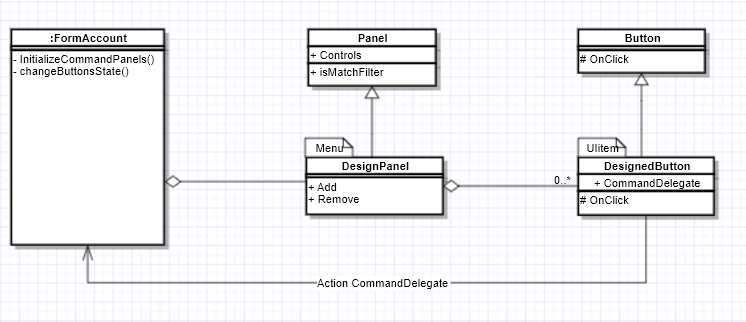
**התבנית Command פותרת בעיה זו, ולכן בחרנו להשתמש בה:**

השימוש בתבנית זו תורמת לתחזוק התפריט והרחבת הפעולות בו (לא נדרש לזכור היכן צריך לשנות את הקוד ונוכל להוסיף אופציות נוספות לתפריט בשורת קוד בודדת).

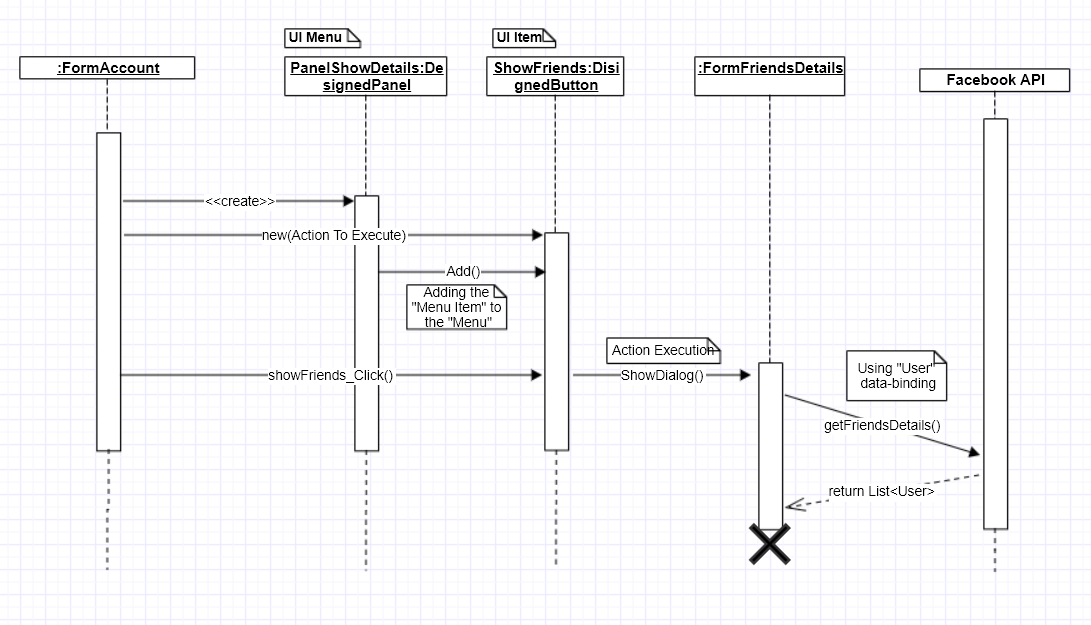
בנוסף, המעבר למימוש תפריט בעזרת מחלקות (UI ITEM וMenu) תורמים למעבר התפריט לתצורת object oriented שזו כמובן הצורה שבה אנחנו מעוניינים כמתכנתי C#.

כמו כן, קיבלנו הפרדה בין השכבה הלוגית לממשק המשתמש, כך שאחד לא תלוי בשני ואם נרצה להחליף את ממשק המשתמש/לוגיקה נוכל לבצע זאת ובכך קיבלנו reuse לתפריט כמו שרצינו.

* **אופן המימוש:**
* יצירת מחלקה DesignedButton שיורשת מButton ומחזיקה מצביע לפונקציה שמבצעת את הפעולה כנדרש – המשחק בתפקיד ה- **UI Item**.
* יצירת מחלקה DesignPanel שיורשת מPanel ומחזיקה רשימה של DesignedButtons המשחק בתפקיד ה- **Menu**.
* המסך הראשי (UserAcountForm) מחזיק רפרנס DesignedPanel, ומוסיף לו את ה DesignedButtons- בעת היצירה.
* המשתמש יוכל לבצע את הפעולה שירצה בלחיצה על הכפתור המתאים ע״י הפעלת הפונקציה.

**Class Diagram:**

**Sequence Diagram:**



### תבנית מס' 3 – Observer

* **סיבת הבחירה / שימוש בתבנית:**

ראשית נציין, שהוספנו יכולת חדשה למערכת- שינוי תמונת הרקע של המשתמש.

כאשר המשתמש ירצה לשנות את התמונה שלו (המוצגת בטופס הראשי-FormAccount)),יפתח טופס חדש (FormChangeCoverPhoto), שנותן אפשרות להגריל תמונות אקראיות מתוך אלבומי המשתמש, ולבחור אחת מהן . התמונה שתבחר תהיה תמונת הרקע החדשה (לא בפייסבוק, אלא באפליקציה המקומית שלנו), ותעודכן באופן אוטומטי בטופס הראשי.

אם בעתיד, נרצה להוסיף טופס או רכיב אחר נוסף,שיעשה שימוש בתמונת הרקע של העדכנית של המשתמש, כיצד נדאג לכך שאם המשתמש שינה את תמונת הרקע שלו דרך האפליקציה, גם הטופס/רכיב הנוסף המשתמש בתמונת הרקע של המשתמש יוכל ל"הקשיב" לעדכון וזאת מבלי שהטופס שמבצע את שינוי תמונת הרקע יכיר את הרכיב החדש . ובכך נמנע מהיכרות מיותרת (tightly coupled) של רכיבים באפליקציה שלנו, וכל מי שירצה "להאזין" לשינוי תמונת הרקע יוכל לבצע זאת ביוזמתו.

**התבנית observer פותרת בעיה זו, ולכן בחרנו להשתמש בה.**

התבנית פותרת את הבעיה בכך שהיא מאפשרת להגדיר מודיע (טופס שינוי התמונה) ומאזין. (כל טופס שירצה לעדכן את תמונת הרקע שבו בהתאם לבחירת המשתמש- הטופס הראשי במקרה שלנו)

באופן זה, המודיע לא צריך להכיר את המאזין, ובכך יכולים להיות לו מאזינים שונים לאירוע אשר יכולים להוסיף/להוריד את עצמם ביוזמתם.

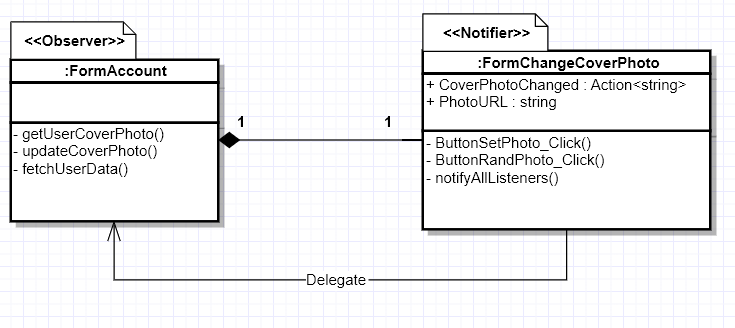
כעת לא נצטרך לשנות את קוד המודיע, ויוזמת ההאזנה עוברת לידי מי שמעוניין להאזין לאירוע.

* **אופן המימוש:**
* מחלקת FormChangeCoverPicture בתפקיד ה**Notifier**-
* מחלקת FormAccount בתפקיד ה-**Listener**

נגדיר delegate במחלקת המודיע עם access modify- event(אירוע),ו"נרשם" אליו במחלקת המאזין- כלומר המאזין מממש פונקציה שתפעל כשהמודיע מחליט להודיע.

במקרה שלנו, טופס שינוי התמונה מודיע לטופס הראשי כאשר המשתמש בחר להחליף את התמונה ומבצע Invoke לרשימת המצביעים למתודות שנוספו לdelegate הנ''ל עם כתובת הurl של התמונה החדשה. המאזין במקרה הFormAccount מופעלת המתודה שנשלחה ותמונת הרקע מתעדכנת בטופס הראשי בהתאם לכתובת הurl החדשה שנבחרה.

**Class Diagram:**



**Sequence Diagram:**

