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

* **Memories game:**

**פיצ'ר המאפשר למשתמש לשחק במשחק הזיכרון ברמות קושי שונות, כאשר הקלפים במשחק נמשכים באופן אקראי מאלבומי התמונות של המשתמש. הפיצ'ר נועד לספק למשתמש חווית משחק מהנה המשלבת זיכרונות נוסטלגיים מהעבר.**

* **Grouper:**

**פיצ'ר המאפשר למשתמש ליצור רשימות חברים (friend lists) חדשות על בסיס מכנה משותף שלו איתם, מתוך רשימת קטגוריות אפשריות קבועות מראש:**

1. **עובדים \ עבדו יחד.**
2. **לומדים \ למדו יחד.**
3. **נמצאים באותו סטטוס רומנטי.**
4. **אוהדים את אותה קבוצה \ קבוצות.**
5. **גרים באותה העיר.**
6. **בני אותו שכבת גיל.**

**הפיצ'ר יציג למשתמש את רשימת חבריו המתאימים לקטגוריה הנבחרה, וכל מה שנותר יהיה למשתמש לעשות הוא לבחור מתוך הרשימה מי מהחברים הוא רוצה לצרף ל"friend list" וללחוץ על כפתור יצירת הרשימה, ורשימה בעלת שם מתאים תיווצר עבורו.**

### תבנית מס' 1 – Observer

* **סיבת הבחירה / שימוש בתבנית:**

השתמשנו בתבנית **Observer** במחלקה ‘***FaceBookUserManager’***, דרכה (בין היתר) מתחבר ומתנתק המשתמש מהמערכת. במהלך מימוש הפרויקט נתקלנו בצורך לעשות פעולות מסוימות בתפריט הראשי כאשר המשתמש מתחבר או מתנתק מהאפליקציה, כמו להציג \ להסתיר את נתוני המשתמש או להחליף את הenable value של כפתורי התפריט הראשי. רצינו שהמידע על כך שהמשתמש התחבר או התנתק מהמערכת יעבור לטופס הראשי באופן "שקוף", ובצורה שאם בעתיד נרצה שפעולות נוספות יקרו בזמן התחברות והתנתקות מהמערכת, נוכל לעשות זאת ביתר קלות.

בגלל הסיבות הנ"ל בחרנו לממש את תהליך הנוטיפיקציה בעזרת תבנית **‘Observer’** שכן תבנית זו פותרת בדיוק את הבעיות ההנדסיות שציינו.

* **אופן המימוש:**

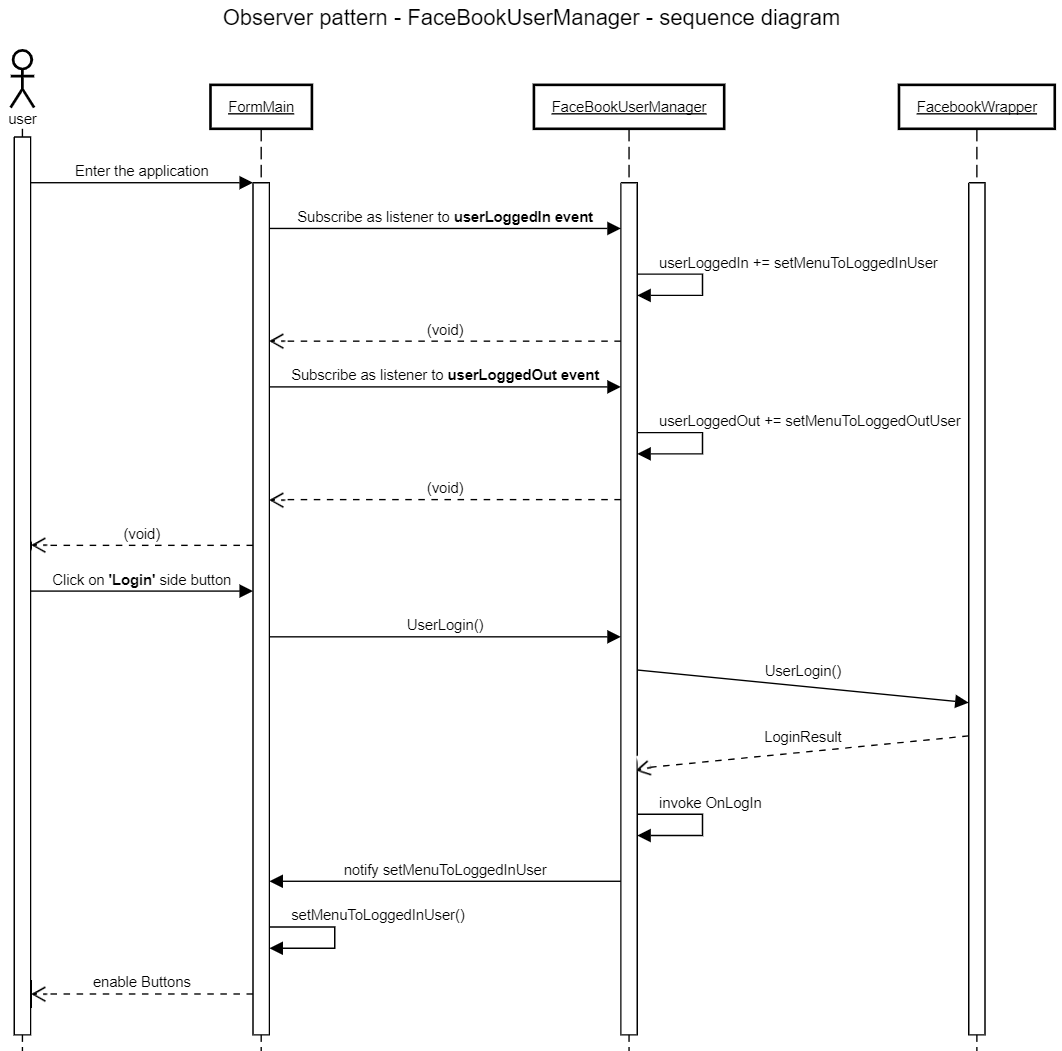
המחלקה ‘***FaceBookUserManager’*** מממשת את תבנית **Observer** כך:

* המחלקה ‘***FaceBookUserManager’*** מחזיקה Actions בשם ***‘UserLoggedIn’*** ו***’UserLoggedOut’***, המחזיקים רפרנס לכל הפעולות שרוצות "להאזין" לרגע בו משתמש מתחבר למערכת, או מתנתק ממנה.
* בטופס הראשי מתבצעת הרשמה של הפעולה ‘***setMainMenuToLoggedInUser’*** כמאזין לפעולת ההתחברות למערכת, ושל הפעולה ‘***setMainMenuToLoggedOutUser’*** כמאזין לפעולת ההתנתקות מהמערכת.
* ברגע שמשתמש מתחבר למערכת, מופעלת המטודה ***’OnLogin’*** המפעילה את כל הפעולות שמאזינות לאירוע ההתחברות. באופן זהה מופעלות הפעולות הרשומות להתנתקות ברגע שהמשתמש מתנתק מהמערכת.

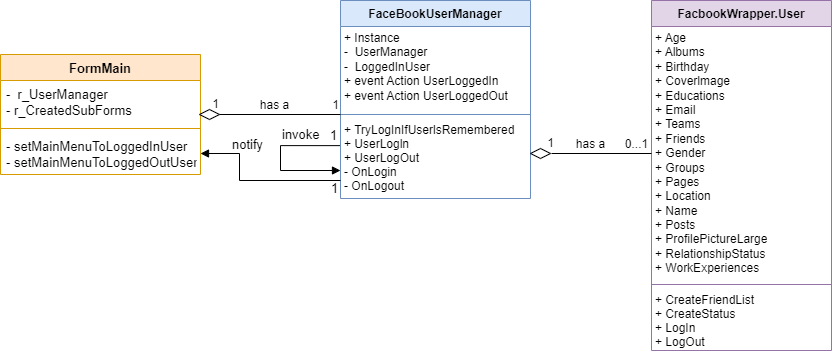
**אם כך המחלקה ‘FaceBookUserManager’ משמשת כNotifier, ו’FormMain’ משמשת כObserver בתבנית.**

ניתן למצוא את מימוש התבנית בקוד במחלקה ‘***FaceBookUserManager’*** שבפרויקט.‘***FacebookAppEngine’*** ואת השימוש בה במחלקה ***‘FormMain’*** שבפרויקט ***‘FacebookAppWinformsUI’***.

* **Sequence Diagram:**



* **Class Diagram:**



### תבנית מס' 2 – Strategy

* **סיבת הבחירה / שימוש בתבנית:**

השתמשנו בתבנית במחלקה ‘***FormGrouper’*** – שם ממומש הפיצ'ר השני שלנו. נזכיר שבמסגרת הפיצ'ר מתאפשר למשתמש ליצור רשימות חברים (friend lists) חדשות על בסיס מכנה משותף שלו איתם, מתוך רשימת קטגוריות אפשריות קבועות מראש. במסגרת הפיצ'ר היינו צריכים לבצע פעולת פילטור על החברים של המשתמש, כל פעם עם "אסטרטגיית פילטור" שונה - בהתאם לבחירת המשתמש.

הבחנו בכך שבדרך המימוש הקודמת בוצע שיכפול קוד של לוגיקת הסינון (מעבר על כל החברים, והחזרת החברים שעברו סינון), וכי ההבדל היחיד בין הקטגוריות הוא תנאי הפילטור (איזה חברים יכנסו לרשימה החדשה). בנוסף רצינו להפוך את המערכת ליותר אקסטנדבילית, כך שאם בעתיד נרצה להוסיף עוד תנאי סינון, נוכל לעשות זאת בקלות. בגלל הסיבות הנ"ל בחרנו לממש את תהליך הפילטור בעזרת תבנית **‘Strategy’** שכן תבנית זו פותרת בדיוק את הבעיות ההנדסיות שציינו.

* **אופן המימוש:**

המחלקה ‘***FormGrouper’*** משתמשת בתבנית **Strategy** באופן הבא:

* יצרנו **Interface** בשם ‘***IFriendsFilterStrategy’*** ובו יש מטודה בוליאנית בשם ***DoesFriendMatchFilter*** שתחזיק את תנאי הפילטור.
* יצרנו 6 מחלקות שונות, אחת לכל קטגוריה בפיצ'ר, המממשות את הממשק הנ"ל. כל אחת מהמחלקות מחזיקה "אסטרטגיית פילטור" שונה, בהתאם לקטגוריה.
* יצרנו מחלקה בשם ‘***FriendsFilterer’*** בה יש Property מסוג ‘***IFriendsFilterStrategy’*** המחזיק את אסטרטגיית הסינון הרצויה.
* ‘***FormGrouper’*** מחזיק מופע של ‘***FriendsFilterer’*** דרכו הוא מבצע סינון על החברים לפי האסטרטגיה שהמשתמש בחר. ברגע שהמשתמש בוחר את אופציית הסינון, הטופס "מזריק" את האסטרטגיה המתאימה – כלומר, מחלקה מתאימה המממשת את הממשק, למופע של ‘***FormGrouper’,*** שם מתבצע הסינון.

**אם כך חלוקת התפקידים בתבנית היא:**

* **המחלקה ‘*IFriendsFilterStrategy’* משמשת כStrategy.**
* **המחלקות ‘*StatusFilterStrategy’,* ‘*CityFilterStrategy’,* ‘*AgeFilterStrategy’* ‘*JobFilterStrategy’,* ‘*TeamFilterStrategy’,* ‘*StudyFilterStrategy’* משמשות כConcreteStrategies.**
* **המחלקה ‘*FriendsFilterer’* משמשת כContext.**
* **המחלקה ‘*FormGrouper’*  היא הClient.**

ניתן למצוא את מימוש התבנית בקוד במחלקות ‘***IFriendsFilterStrategy’,*** ‘***FriendsFilterer’*** ובכלהמחלקותבתיקיה ***‘FriendsFilterStrategies’*** שבפרויקט.‘***FacebookAppEngine’***את השימוש בתבנית אפשר למצוא במחלקה ***‘FormGrouper’*** שבפרויקט ***‘FacebookAppWinformsUI’***.

* **Sequence Diagram:**

Diagram

Description automatically generated

* **Class Diagram:**

Graphical user interface

Description automatically generated

### תבנית מס' 3 – Decorator

* **סיבת הבחירה / שימוש בתבנית:**

השתמשנו בתבנית במחלקות ‘***PictureBoxBordered’*** ***ו’PictureBoxNamed’***.

בנקודות רבות בפרויקט נתקלנו בצורך להציג תמונות ממוסגרות, או תמונות עם שמות. לשם כך מימשנו שתי מחלקות מתאימות היורשות מPictureBox, ומוסיפות לו לוגיקה מתאימה. בנוסף נתקלנו בצורך להציג חלק מהתמונות באופן ממוסגר וגם עם שם (למשל אלבומי התמונות).

לשם כך מימשנו בהתחלה מחלקה בשם **PictureBoxNamedAndBordered,** השילבה בין 2 התכונות. אולם, שמנו לב שמחלקה זו אינה אקסטנדבילית, כלומר, אם נרצה בעתיד ליצור תמונה מסוג נוסף ולשלב אותה עם האחרות (אחת מהן או שתיהן יחד) לא נוכל לעשות זאת בצורה נוחה, ונצטרך ליצור מחלקה נוספת עבור כל קומבינציה. בנוסף ראינו כי בשיטה זו מתבצעים הרבה שכפולי קוד.

בגלל הסיבות הנ"ל החלטנו לממש את המחלקות כפי שמציעה תבנית **Decorator**, שכן תבנית זו פותרת בדיוק את הבעיות ההנדסיות שציינו.

* **אופן המימוש:**

המחלקות ‘***PictureBoxBordered’*** *ו****’PictureBoxNamed’*** מממשות את התבנית **Decorator** באופן הבא:

* יצרנו מחלקה בשם ‘***PictureBoxDecorator’*** שיורשת מPictureBox ומחזיקה מופע של PictureBox. בנוסף המחלקה מממשת לוגיקה המשותפת לכל הפקדים מהסוג של תמונה מקוננת שנרצה ליצור בעתיד.
* שתי המחלקות ‘***PictureBoxBordered’*** *ו****’PictureBoxNamed’*** יורשות מPictureBoxDecorator, ולכן כוללות את הלוגיקה המשותפת, ונותר להן רק להוסיף לוגיקה מינימלית המבדילה אותן אחת מהשנייה.
* כאשר נרצה ליצור תמונה עם שם ומסגרת (ואולי עוד דברים בעתיד), נוכל אתה ליצור אחת ביתר קלות באופן משורשר, לדוגמא:

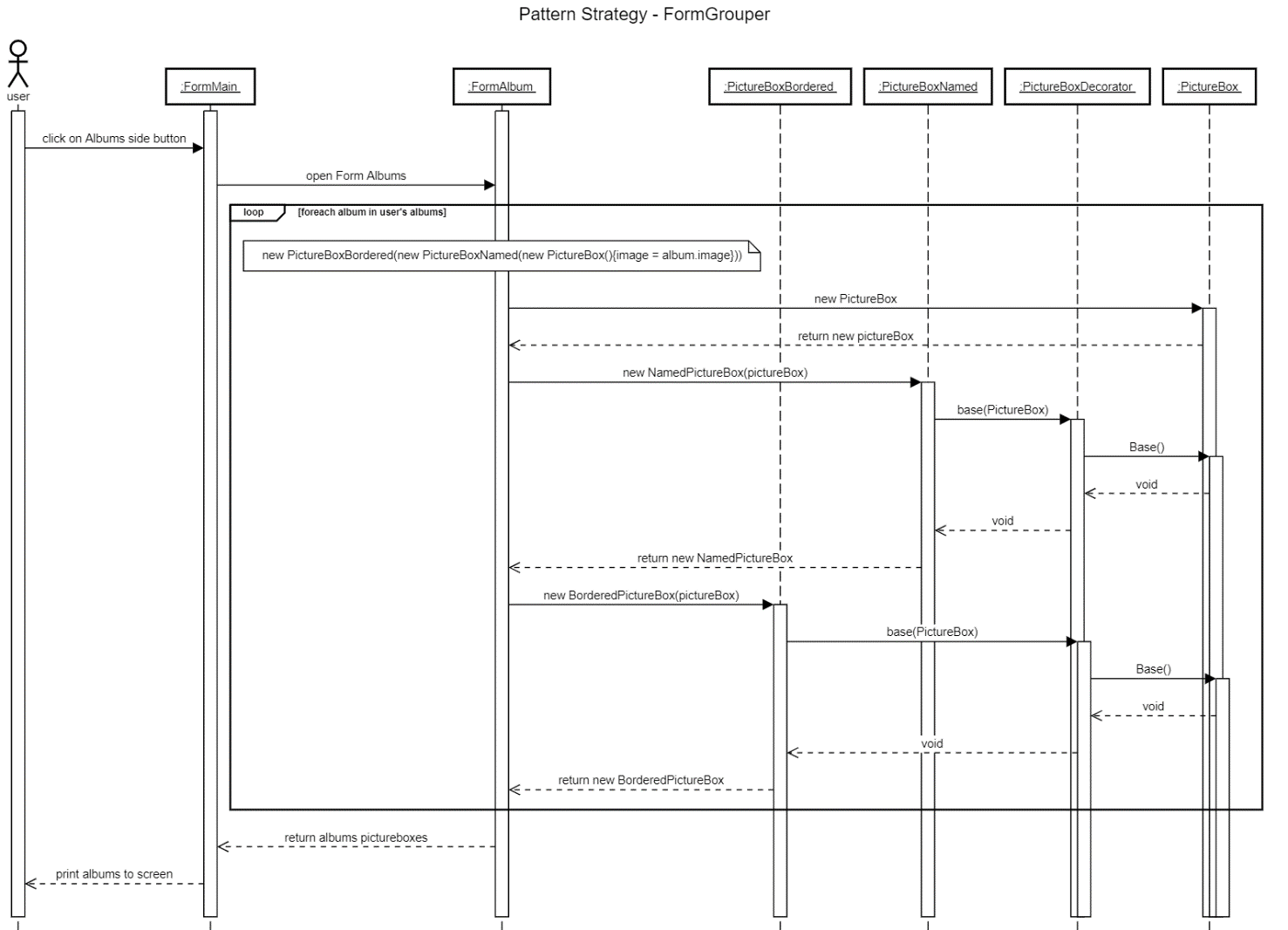
Picture = new PictureBoxBorder(new PictureBoxNamed(new PictureBox()));

**אם כך חלוקת התפקידים בתבנית היא:**

* **המחלקה ‘*PictureBoxDecorator’* משמשת כDecorator.**
* **המחלקות *’PictureBoxNamed’ ,* ‘*PictureBoxBordered’* משמשות כConcreteDecorators.**
* **המחלקה ‘*PictureBox’* משמשת כDecorated.**
* **הטפסים השונים הם הClient.**

ניתן למצוא את מימוש התבנית בקוד במחלקות ‘***PictureBoxDecorator’, ’PictureBoxNamed’ ,* ‘*PictureBoxBordered’*** שבפרויקט ***’FacebookAppWinformsUI*** ואת השימוש בתבנית אפשר למצוא בטפסי האלבומים, הקבוצות והאלבומים באותו הפרוייקט.

* **Sequence Diagram:**



* **Class Diagram:**

### 