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

הפיצ'ר הראשון שהוספתי הוא חיפוש המעריץ הגדולים ביותר שלך.

פיצ'ר זה נועד למוצא את שלושת המעריצים הגדולים ביותר של המשתמש, כלומר: שלושת האנשים אשר הכי פעילים בחשבון הפייסבוק של המשתמש. הן מבחינת כמות הלייקים והן מבחינת התגובות בפוסטים ובתמונות שהעלה המשתמש. קיימים 4 פרמטרים עליהם מבוסס מדד ההערצה: לייק על תמונה, תגובה על תמונה, לייק על פוסט ותגובה על פוסט. בררת המחדל היא: ניקוד של 1 עבור לייק או תגובה לתמונות ו-0 עבור לייק תגובה לפוסט. ניתן לשנות את בררת המחדר ולתת לכל פרמטר ערך בטווח בין 0-100. ניקוד זה מהווה את מידת החשיבות אשר המשתמש מייחס לכל אחד מהמאפיינים הללו, כלל שהניקוד למאפיין גבוה יותר ביחס לניקוד של המאפיינים האחרים כך הוא מקנה ערך גבוה יותר במדד הערצה.  
  
עבור מימוש פיצ'ר זה יצרתי FORM חדש ובו גישה למחלקה בחלק הלוגי של הקוד בשם BestFan. אשר שם מחושב עבור כל משתמש שהגיב ו/או עשה לייק ולבסוף מחזיר את 3 המעריצים הגדולים ביותר של אותו היוזר כאשר ניתנת גם האופציה להמשיך ולשאול על מעריצים נוספים לפני הסדר.

הפיצ'ר השני שהוספתי הוא המראה(The Mirror) – הצגת תחומי העיינין של המשתמש באמצעות מילות מפתח.

פיצ'ר זה נועד להציג בפני המשתמש את תחומי העניין אשר פייסבוק מאתר עבורו בהתבסס על מילות מפתח המצויות בפוסטים ובשמות התמונות שלו. בפני המשתמש ניתנת האפשרות להחליט מה הוא מספר האזכורים אשר החל ממנו יוצג התחום כתחום העניין של המשתמש. ככול שהדירוג שהזין המשתמש עבור מדד הרגישות לתחומי העניין שלו גבוהה יותר כך המסקנות יהיו יותר מדויקות יותר עבור תחומי העניין שלו.  
  
עבור מימוש זה יצרתי FORM חדש ובו גישה למחלקה בחלק הלוגי בשם PersonalityAnalyzer אשר מחלקה זו מהווה את החלק של זיהוי תחום העניין והסקת המסקנות אשר מתממשקת עם DATABASE אשר יצרתי בשם WordsDatabase הכולל מילון עם מילות מפתח אשר ממפות לסט של מילים שמצביעות על עניין בתחום מסוים משמעויות עבור מילים אלו (עבור מילה מפתח אחת יכול להיות כמה תחומי עניין/ תפוסי התנהגות) .

**עבודה אסינכרונית עם Threads**

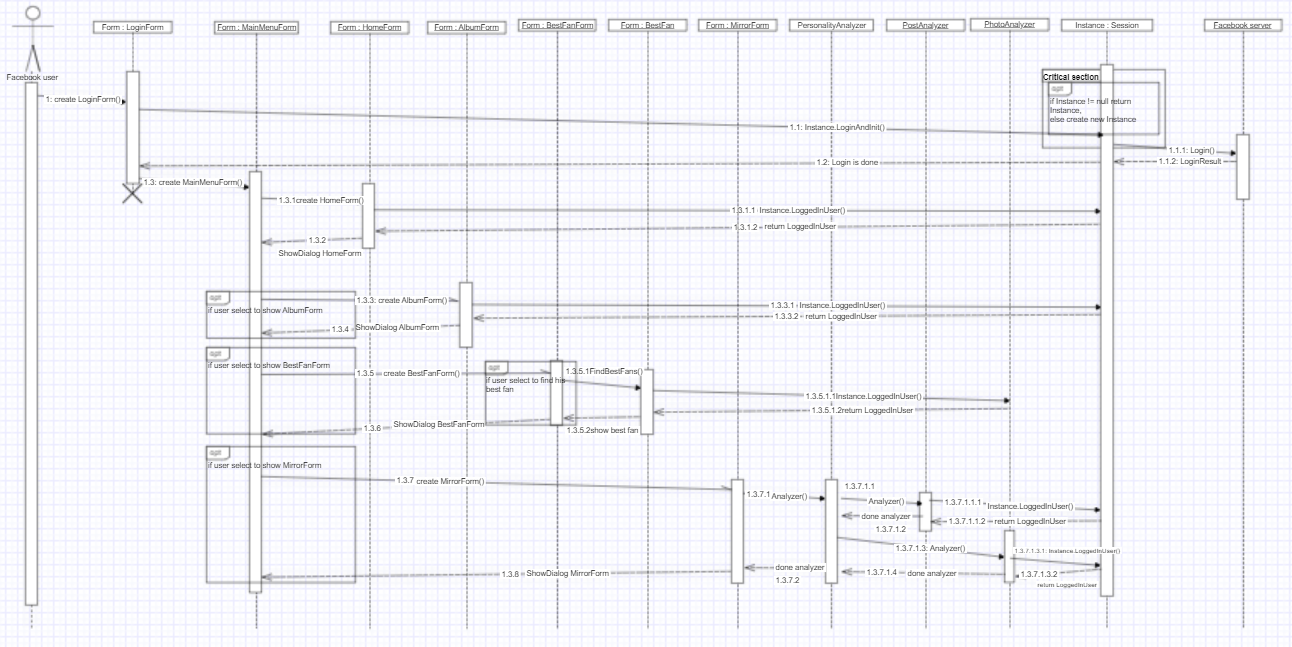
השתמשתי בתכנות א-סינכרוני על מנת לאפשר UI טוב יותר כיוון שפעולות מול שרתי פייסבוק לוקחות זמן רב מרגע הבקשה מול שרתי פייסבוק ועד שהוא מגיע חזרה לתוכנה שלי ובכך עשויה להוות עיקוב וחווית משתמש פגומה.  
בעזרת עבודה א-סינכרונית אנו ניגש באופן מקבילי לשרתי פייסבוק ובזמן זה נאפשר למשתמש להמשיך את עיסוקו וברגע שבקשות אלו נענות וחוזרות אלינו הם יוצגו למשתמש.  
השתמשתי ב- Threads במקומות הבאים במחלקתHomeForm :

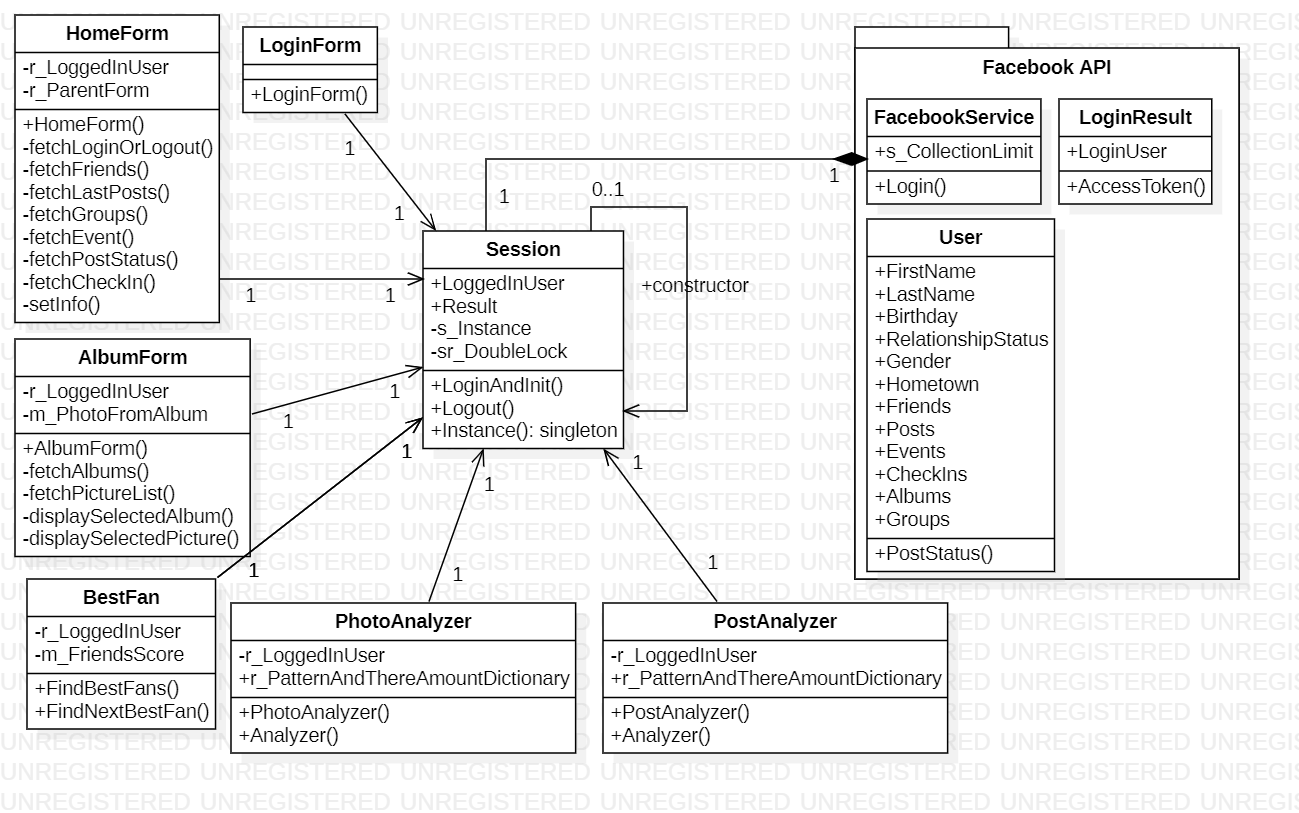
* בעת לחיצה על כפתור "Show Friends" יופעל פעולת thread שתריץ את המתודה הבאה: "fetchFriends"
* בעת לחיצה על כפתור "Post" יופעל פעולת thread שתריץ את המתודה הבאה: fetchPostStatus""
* בעת לחיצה על כפתור "Show Event" יופעל פעולת thread שתריץ את המתודה הבאה: fetchEvent""
* בעת לחיצה על כפתור "Show Last Posts" יופעל פעולת thread שתריץ את המתודה הבאה : fetchLastPosts""
* בעת לחיצה על כפתור "Show Groups" יופעל פעולת thread שתריץ את המתודה הבאה: fetchGroups ""
* בעת לחיצה על כפתור "Show Check In" יופעל פעולת thread שתריץ את המתודה הבאה: fetchCheckIn""

**עבודה עם Data Binding**

בחרתי להשתמש ב- One way Data Binding כאשר האובייקט שבתוך תקיית ה- DataSources הוא של ה-User ששייך למחלקת FacebookWrapper ((FacebookWrapper.ObjectModel.User.  
מי שמשתמש בשיטה זו היא המתודה  *שממוקם* במחלקת HomeForm *אשר נמצע בחלק של ה-UI בקוד.  
שימוש זה מאפשר עבודה פשוט אשר מעדכנת את הנתונים בצורה קלה מהירה עם מעט קוד ובכך משפר את קריאות הקוד בכתיבה מהירה יותר אשר ברורה כיוון שהיא משתמשת במוסכמות השפה.*

### **תבנית מס' 1 –** Singleton

* סיבת הבחירה / שימוש בתבנית:  
  תבנית זו היא ממשפחת ה- Creational Patterns ומיועדת לפתור את הבעיה של שימוש במופע יחיד של אובייקט אשר ניתנת האפשרות לתמוך ב- thread-safe זאת על מנת שמחלקות שונות אשר צריכות לגשת לאותו אובייקט יכירו אותו ולא העתק שלו. בעזרת מופע יחיד זה אנו ניצר קוד בטוח נוטל בעיות של ריבוי מופעים או עותקים של אותו אובייקט ובכך אנו נמנע באגים עתידים אשר עשויים להתרחש.  
  במקרה זה האובייקט אשר אנו מעוניינים שיתקיים לו מופע יחיד הוא User.
* אופן המימוש:  
  מימוש זה יצרתי את הSingelton- בעזרת double check lock זה בעת הגישה ל- instance של המחלקה.  
  שימוש זה מתבצע במחלקת Session אשמח מתקיים Ex02.Logic כאשר תתקיים פנייה למחלקת Session יתרחש אחד משני המקרים הבאים :  
  1) ה- instance היחיד קיים ונחזיר את האובייקט הקיים.  
  2) ה- instance היחיד אינו קיים ולכן ניצור אובייקט חדש ונחזיר אותו.  
  נשים לב שמימוש זה הוא fully-lazy.
* Sequence Diagram
* Class Diagram

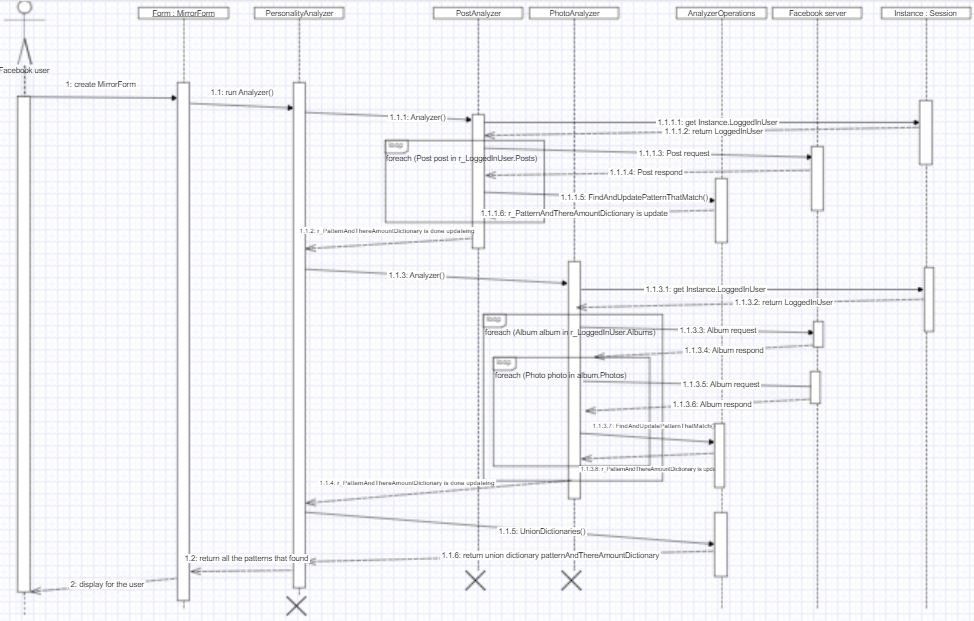
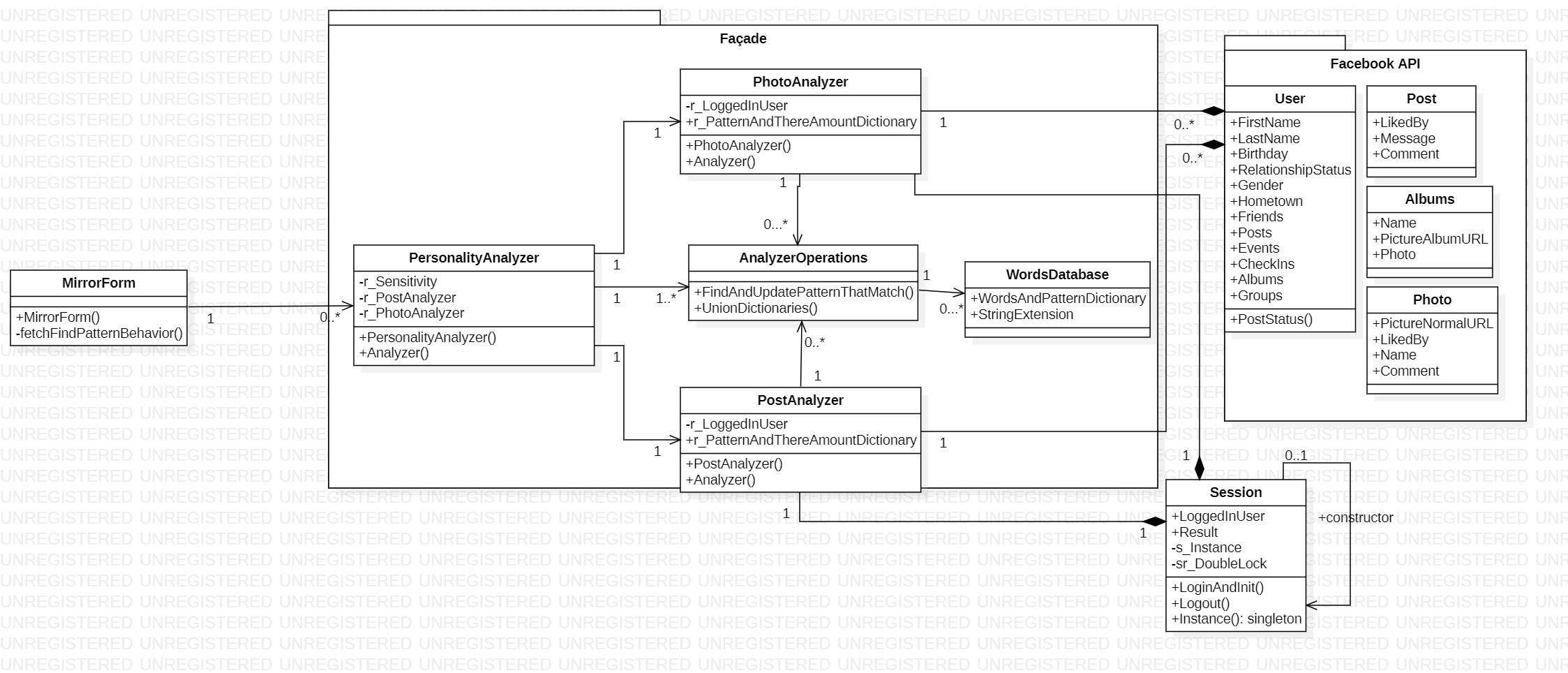


הקבלה ל- Singleton Pattern:

Client: HomeForm, AlbumForm, LoginForm,BestFan, PhotoAnalyzer, PostAnalyzer  
Creator: public sealed class Session  
Product: Instance

### **תבנית מס' 2 –** Facade

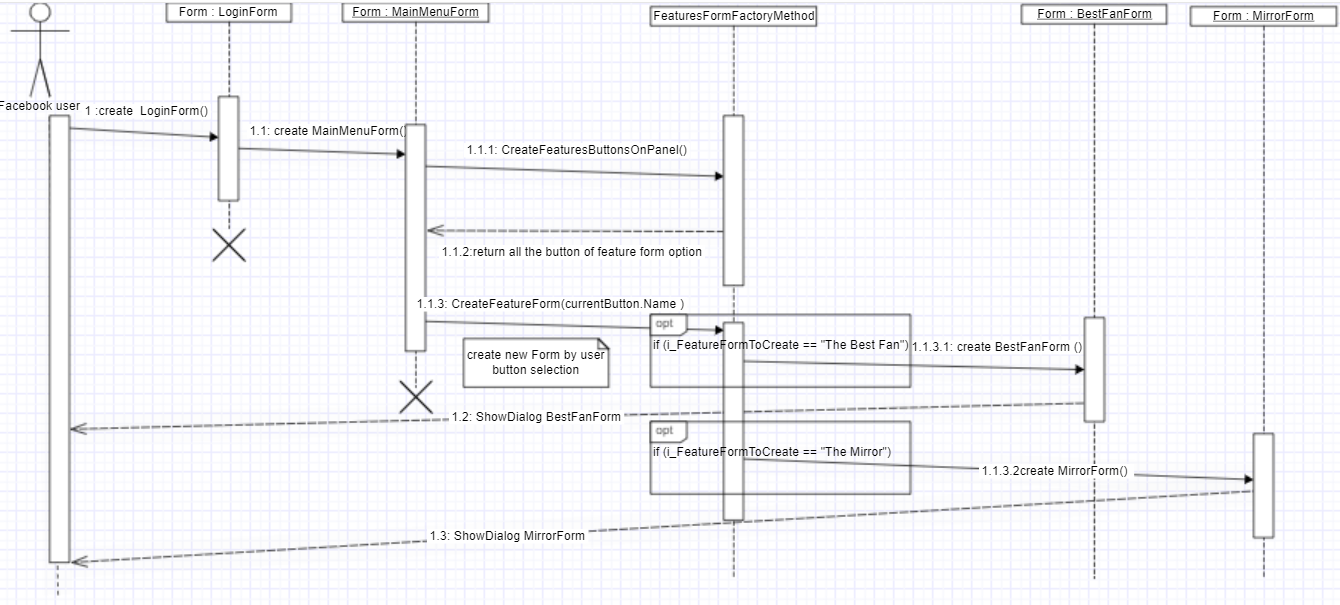
* סיבת הבחירה / שימוש בתבנית:  
  תבנית זו היא ממשפחת ה- Structural Patterns ומיועדת לאפשר בנייה של ממשק פשוט שרק הוא אחראי על הפעלת תתי מערכות מורכבות המרכיבות אותו, זאת תוך הפחתת תלוי בין כלל תתי המערכות המרכיבות אותו.  
  שימוש בתבנית זו קורה בפיצ'ר "The mirror" אשר מבצע מעבר על כל הפוסטים והתמונות של המשתמש ומנתח את דפוס ההתנהגות של המשתמש.  
  פיצ'ר זה הוא מורכב אשר אנו נרצה לאפשר גישה פשוטה לממשק שלו ובנוסף ברצונינו להפחית כמה שניתן את התלות בין תתי המערכות של פיצ'ר זה ונאפשר גישה ישירה רק ליוצר ה-facade עצמו. כך למעשה אנו מאפשרים להסתיר חלק מהמערכות של הפיצ'ר ומאפשרים גישה סדורה עם ממש אחיד.  
  נשים לב שתתי המערכות בעתיד עשויות לגדול וכך אנו יוצרים אי תלות בין תתי המערכות של הפיצ'ר ורק ה- facade אחראי לטפל בתתי מערכות חדשות.
* אופן המימוש:  
  במימוש זה יצרתי את תתי המערכות הבאות אשר ה- facade (בשם: PersonalityAnalyzer) מארגן ומנהל.  
  1) **PhotoAnalyzer** – אחראית על ניתוח תפוסי ההתנהגות מהתמונות.  
  2) **PostAnalyzer** – אחראית על ניתוח תפוסי ההתנהגות מהפוסטים.  
  3) **AnalyzerOperations** – מחלקה שאחראית על פעולות שונות של ניתוח התנהגות(כגון: ניתוח תפסי התנהגות על פי מחרוזות חוזרות, איחוד מילונים[קיים על מנת להפחית תלות בין תתי מערכות]).  
  4) **WordsDatabase** – מחלקה שמחזיקה במילון הקשרים בין ערך של מילה לערך תפוסי התנהגות.  
  מחלקת ה- PersonalityAnalyzer מסתירה בעזרת הממשק את תתי המערכות שלה ולכן ה-UI ייעזר רק בה על מנת להשתמש בפיצ'ר זה.  
  קוד זה ניתן למצוא בחלק ה- Ex02.Logic תחת תיקייה בשם facade.

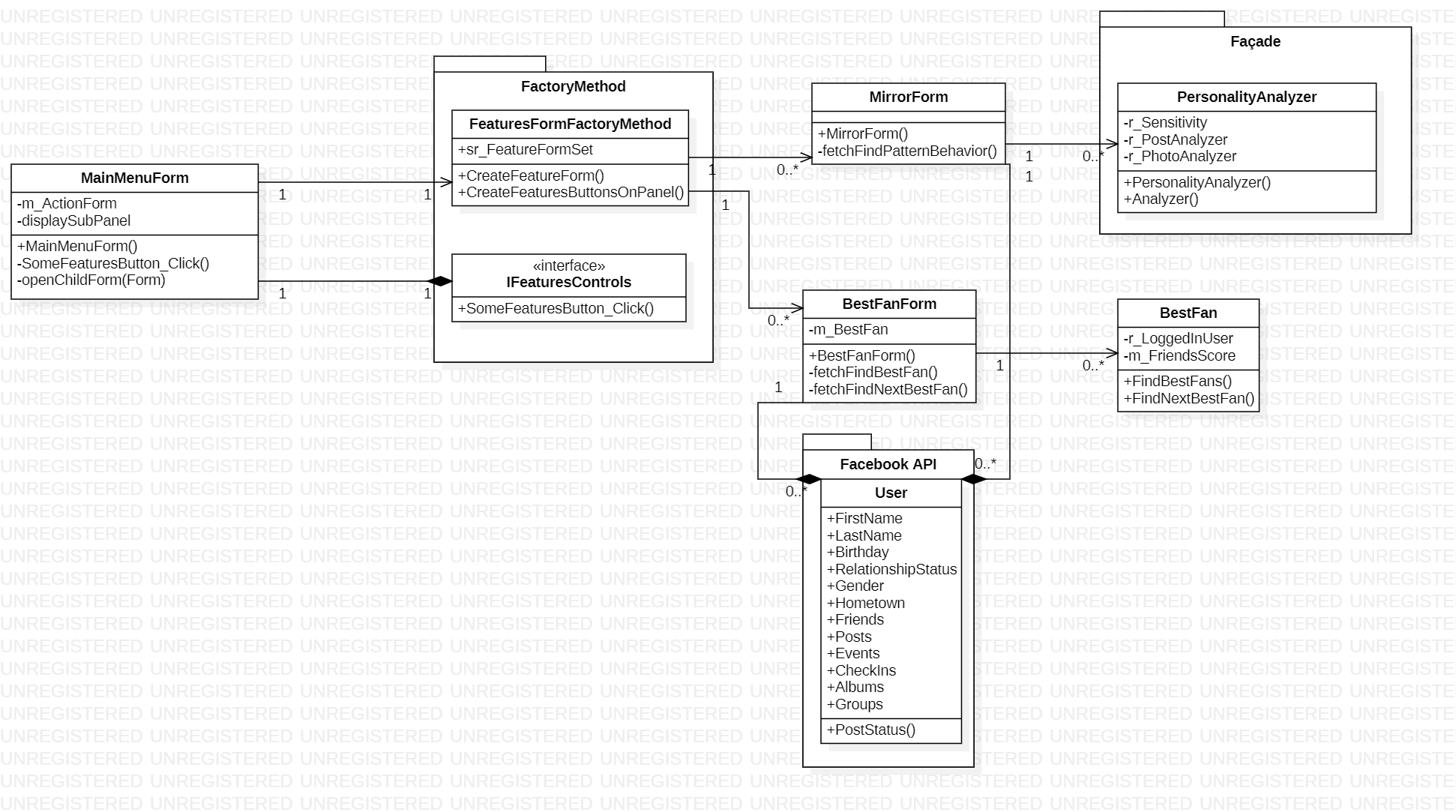
* Sequence Diagram  
  
* Class Diagram  
  

הקבלה ל- Facade Pattern:

Client: MirrorForm  
Façade: public class PersonalityAnalyzer  
Sub- system classes: PostAnalyzer, AnalyzerOperations, PhotoAnalyzer, WordsDatabase

### **תבנית מס' 3 –** Factory Method

* סיבת הבחירה / שימוש בתבנית:  
  תבנית זו היא ממשפחת ה- Creational Patterns מיועדת לאפשר בנייה של פיצ'רים חדשים בעזרת שינויים מינוריים בקוד, מאפשרת מודלריות כאשר ממשק המשתמש אינו מכיר אילו פיצ'רים קיימים עבורו, ופשוט תיעזר מחלקת FeaturesFormFactoryMethod אשר תייצר עבורו את הכפתורים של פיצ'רים אלו. לאחר מכן בעת לחיצה על כפתור הפיצ'ר מחלקה זאת תעזור ותבין איזה כפתור של איזה פיצ'ר נבחר, זאת בלי שהFORM שמחזיק את הכפתור של ה- FeatureForm יכיר את אותו הכפתור.  
  בנוסף, נשים לב שכל FeaturesForm הם חלק ממשפחת של FORM ומרחיבים את אותה המחלקה.
* אופן המימוש:  
  שימוש זה מתבצע במחלקת FeaturesFormFactoryMethod אשמח מתקיים בתיקיית FactoryMethod בחלק Ex02.DesktopGUI.  
  יצרתי 2 מחלקות הבאות:  
  1) IFeaturesControls - מטרתה לאפשר פולימורפיזם עבור המתודה CreateFeaturesButtonsOnPanel .  
  2) FeaturesFormFactoryMethod – בה ממומשת המתודה CreateFeatureForm אשר מקבלת מחרוזת ומחזירה את סוג ה- FORM שאותו בחר המשתמש לצור ולפתוח זאת מתוך מאגר של FeaturesForm האפשריים שממומש במחלקה זו.   
  בנוסף קיימת המתודה CreateFeaturesButtonsOnPanel אשר מטרתה ליצור בFORM הראשי רצף של כפתורים אחד אחר השני בהתאם לכמות הפיצ'רים הקיימים ובכך לפשט את ממשק המשתמש מלהכיר את כלל הכפתורים של הפיצ'רים.  
  בקוד עצמו ניתן לראות את הקריאות למתודות אלה ממחלקת MainMenuForm מהמטודות הבאות: SomeFeaturesButton\_Click , MainMenuForm (contractor)
* Sequence Diagram
* Class Diagram



הקבלה ל- Factory Method Pattern:

Client: MainMenuForm  
Creator: public class FeaturesFormFactoryMethod  
Product: BestFanForm, MirrorForm