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

1. **Contest:**

פיצ'ר זה מאפשר למשתמש לנהל הגרלות בקלות, כאשר עליו להכניס רק את תיאור התחרות ותמונה מתאימה (אופציונלי), מהו האינדיקציה שלפיה משתמשים נוספים ייכנסו לרשימת ההגרלה (לייק לפוסט / תגובה לפוסט / גם וגם) וכמות המנצחים בסיום ההגרלה. הפיצ'ר כותב אוטומטית פוסט ומדפיס אותו לקיר של המשתמש. לאחר מכן, המשתמש יכול לעקוב אחרי רשימת הנרשמים להגרלה על ידי לחיצה על כפתור בודד, שמעדכן אוטומטית את הרשימה לפי אינדיקציות ההשתתפות. כשהמשתמש מחליט שנגמרה ההגרלה, ורוצה לפרסם מנצח/ים – עליו ללחוץ על כפתור בחירת מנצח, והפיצ'ר אוטומטית מגריל כמות מנצחים ככמות שהמשתמש בחר מראש, ומציג למשתמש את המנצחים בחלון חדש. בנוסף, מאפשר מעקב נוח אחרי כל ההגרלות שמנוהלות על ידי אותו משתמש. כמו כן, ישנה אפשרות למחיקת הגרלה.

ניתן למצוא את הפיצ'ר הנ"ל בחלון הראשי של האפליקציה תחת כפתור “Contests menu”.

1. **Memories:**

פיצ'ר זה מאפשר למשתמש להציג תחת קורת גג אחת מגוון פעילויות שביצע באתר פייסבוק, עם אפשרות סינון לפי טווח תאריכים. על המשתמש לבחור אילו נושאים ירצה שיוצגו בפניו, ועליו לבחור תאריך התחלה וסיום (כאשר יש אופציית בחירה של יום בודד). כאשר ילחץ על כפתור ה "Fetch", כל המידע המבוקש יוצג בפניו.

ניתן למצוא את הפיצ'ר הנ"ל בחלון הראשי של האפליקציה תחת כפתור “Memories”.

### תבנית מס' 1 – [Factory Method]

* סיבת הבחירה / שימוש בתבנית:

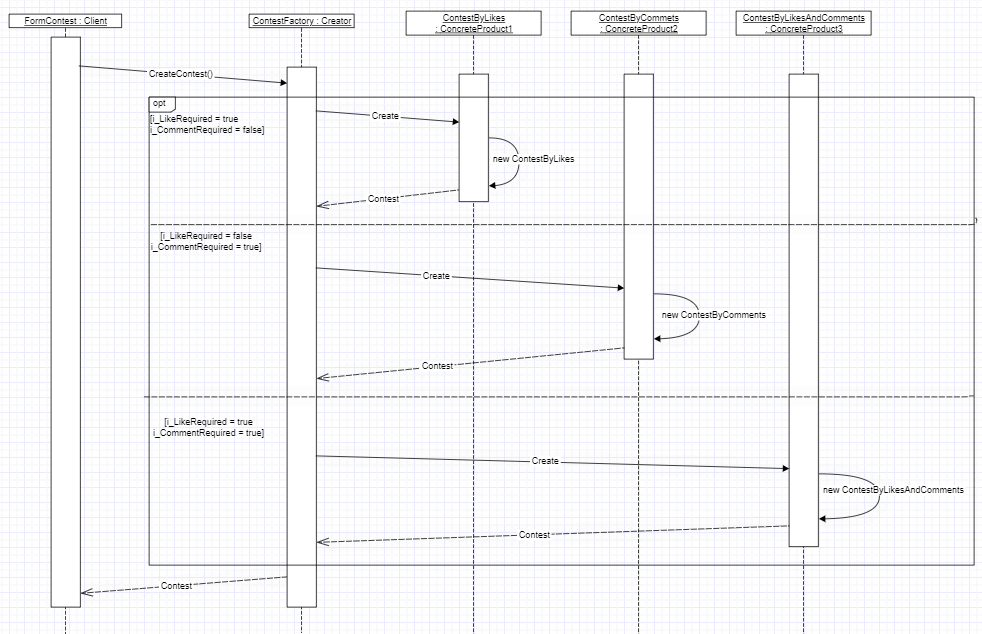
בפיצ'ר הראשון שמימשנו קיימים 3 סוגי תחרויות שונים. במקום לבצע את מלאכת היצירה של התחרויות השונות ב client, בחרנו להשתמש ב factory method ובכך מלאכת היצירה מתבצעת בצד הלוגי. בנוסף, אם בעתיד נרצה להוסיף סוגים שונים של תחרויות, במימוש הנוכחי הדבר יתאפשר בתחזוקה מינימלית בעזרת הורשה מהמחלקה האבסטרקטית Contest וללא שינוי בלוגיקה (בניגוד לקוד הקודם, שם היה צורך בשינוי הלוגיקה עצמה והוספת השדות המתאימים לפי התחרות החדשה שנרצה להוסיף).

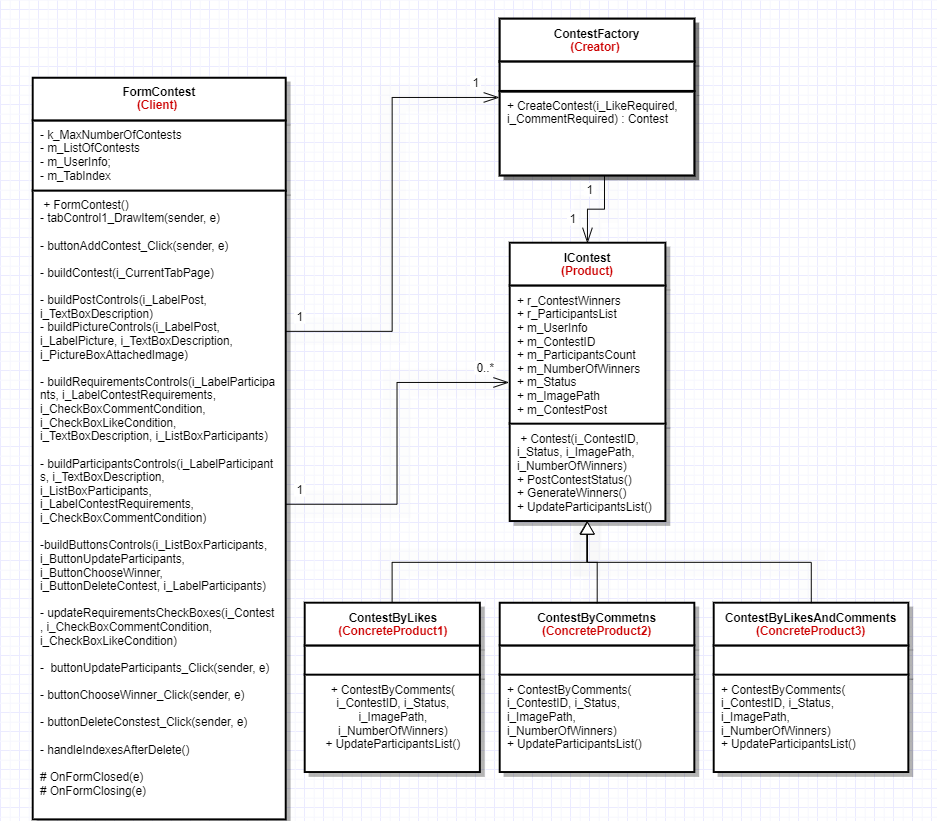
* אופן המימוש:

מימשנו את תבנית העיצוב הנ"ל על ידי יצירת 3 מחלקות שונות (ContestByComments, ContestByLikes, ContestByLikesAndComments) כאשר כל מחלקה מייצגת סוג תחרות אחר והן יורשות מהמחלקה האבסטרקטית Contest. במחלקה האבסטרקטית ממומש רוב הלוגיקה הנדרשת לתחרויות, ועוד מתודות אבסטרקטיות אשר ממומשות באופן שונה בכל אחת מהמחלקות היורשות.

בנוסף, במקום שהאחריות על יצירת התחרות תהיה אצל המשתמש, יצרנו מחלקה בשם ContestFactory, שבהתאם לפרמטרים שהיא מקבלת מחזירה את התחרות המתאימה.

ניתן למצוא את המימוש של תבנית העיצוב הזאת במחלקות הבאות:

* הקריאה ל ContestFactory מתרחשת ב FormContest במתודה buttonAddContest\_Click (שורה 64).
* ContestFactory.
* Contest.
* ContestByComments.
* ContestByLikes.
* ContestByLikesAndComments.
* Sequence Diagram
* Class Diagram



### תבנית מס' 2 – [Facade]

* סיבת הבחירה / שימוש בתבנית:

משתמש בסיסי בפייסבוק יכול לבצע פעולות מגוונות – לראות את הפרטים האישיים שלו, אלבומים, רשימת חברים ועוד.

בכדי להשיג את הנתונים הללו יש לגשת למספר מתודות במחלקת User ולחשב פונקציונליות בהתאם לבקשה.

בכדי לפשט עבור ה client את הפעולות האלו יצרנו facade אשר מטרת המתודות שבתוכו הם פישוט הפעולות – בפעולה אחת אצל ה client מוחזרת הרשימה המבוקשת, ללא צורך בהבנה של המתודות במחלקה User.

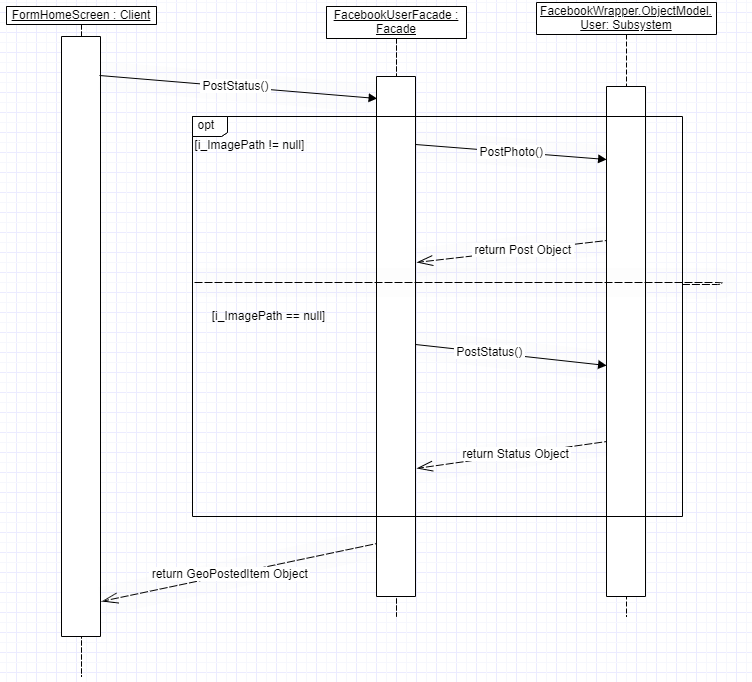
* אופן המימוש:

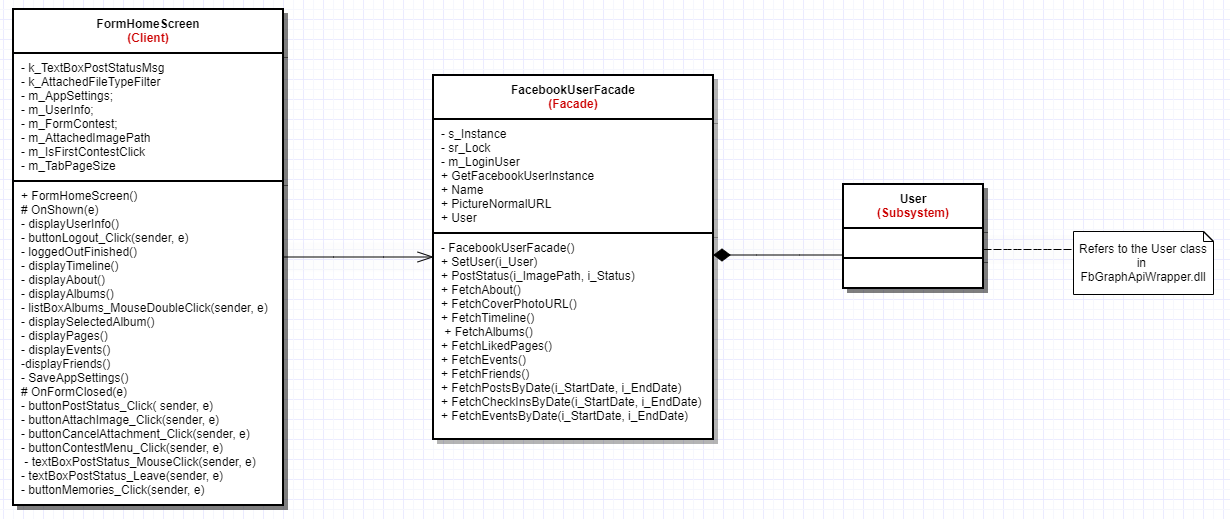
יצרנו מחלקה בשם FacebookUserFacadeאשר מחזיקה בקומפוזיציה אובייקט מסוג User.

בנוסף, המחלקה מממשת פונקציות המקלות על המשתמש לקבל את המידע הרצוי, והשימוש המרכזי בו מתרחש ב FormHomeScreen.

ניתן למצוא את המימוש של תבנית העיצוב הזאת במחלקות הבאות:

* מאותחל ב FormLoginScreen לאחר התחברות מוצלחת של המשתמש במתודה SetUserInfoWrapper.
* FormHomeScreen
* Contest
* FormContest
* FormMemoriesFetch
* Sequence Diagram



* Class Diagram

### תבנית מס' 3 – [Singleton]

* סיבת הבחירה / שימוש בתבנית:

כאשר User מתחבר למערכת, אנו רוצים לוודא שהוא אכן המשתמש היחיד שמחובר בו זמנית לאפליקציה. לכן, בעזרת שימוש בתבנית העיצוב הנ"ל, נוכל להבטיח שה client לא יחזיק (בטעות או שלא) מספר אובייקטים מסוג FacebookUserFacade (אשר מכיל בתוכו את המשתמש המחובר כעת).

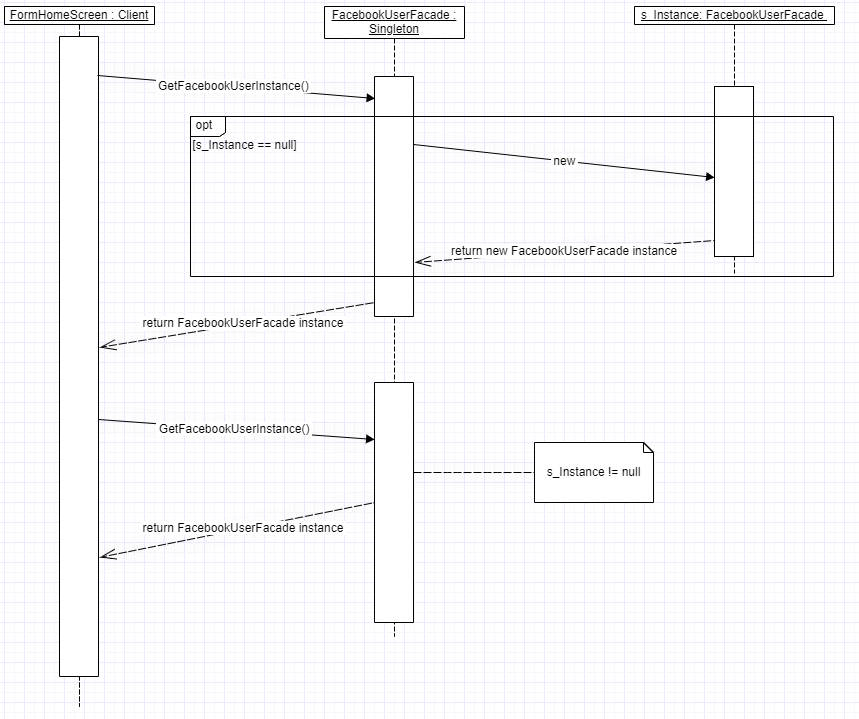
בנוסף, כאשר משתמש כלשהו מחובר לאפליקציה, נרצה אפשרות לגשת אליו מכל מקום באפליקציה בכדי למשוך פרטים רלוונטים בהתאם ל Form הנוכחי. Singleton מאפשר בדיוק את זה.

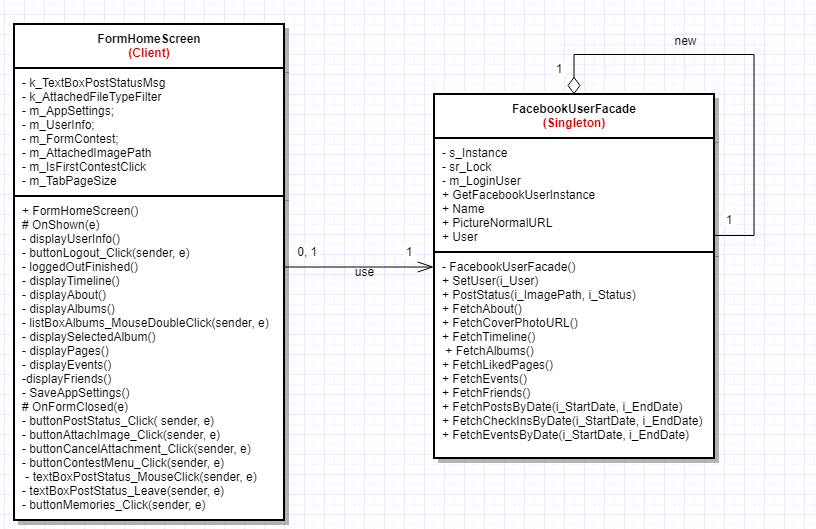
* אופן המימוש:

ה Singleton שלנו מגן על יצירה נוספת של FacebookUserFacade.

במחלקה של FacebookUserFacade הסתרנו את ה Constructor והשתמשנו ב double-check locking כדי לוודא שהוא אכן המשתמש היחיד שנוצר. במידה ונוצר משתמש בעבר, מתודת יצירת האובייקט תחזיר את האובייקט שכבר נוצר, ולא אחד חדש.

ניתן למצוא את המימוש של תבנית העיצוב הזאת במחלקה FacebookUserFacade, תחת המתודה GetFacebookUserInstance.

* Sequence Diagram
* Class Diagram



**עבודה אסינכרונית**

עשינו שימוש בקוד בתכנות אסינכרוני בשני מקומות שונים:

* במחלקה FormLoginScreen, במתודה OnShown: ללא עבודה אסינכרונית, בזמן ניסיון ההתחברות בעזרת ה Access Token, הפקדים בטופס ההתחברות (FormLoginScreen) לא עלו כראוי והטופס היה "קפוא", שכן הם חיכו לסיום המתודה שמטפלת בהתחברות. יצרנו Thread חדש שמפעיל את המתודה שמטפלת בהתחברות המשתמש בעזרת ה Access Token, כאשר במקביל ה Thread הראשי מסיים את טעינת הדף, ואז מחכה לסיום ה Thread הנוסף.

כעת, כאשר יישמנו את השימוש בעבודה אסינכרונית, טופס ההתחברות עולה כראוי ובמקביל ה Thread המשני מטפל בהתחברות המשתמש.

* במחלקה FormHomeScreen, במתודה fetchUserInfo: בטופס הבית לאחר ההתחברות נטען מידע אודות המשתמש (מידע בסיסי, אלבומי תמונות, רשימת חברים ועוד...). בכדי שה Thread הראשי יסיים לטעון את הטופס במלואו ללא צורך בטעינת המידע אודות המשתמש, יצרנו מספר Threads, שמטרת כל אחד מהם היא לטעון במקביל מידע אחר אודות המשתמש.

**עבודה עם Data Binding**

* בטופס הבית לאחר ההתחברות, בלחיצה כפולה על אלבום מתוך רשימת האלבומים של המשתמש – נפתח טופס חדש שבו יש שימוש ב Data Binding בכדי להציג את תמונות האלבום הנבחר.