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

* הפיצ'ר הראשון שהוספנו הוא חיפוש חברים לפי המזל שלו כפי שמוגדר בגלגל המזלות – Zodiac Sign Manager.

המשתמש ילחץ על הכפתור ויקבל בחזרה רשימה של כל החברים שלו בעלי אותו מזל כמוהו.

עבור מימוש הפיצ'ר יצרנו Form חדש ובו ראשית יופיע ההורוסקופ של המשתמש המחובר. בלחיצת כפתור של: Find my Zodiac match נוצרת רשימה של כל חברי הפייסבוק של המשתמש הנוכחי שמחובר לאפליקציה (אם טרם נוצרה בשלב אחר של ריצת האפליקציה). יצרנו מחלקה חדשה המבצעת חישוב עבור כל חבר פייסבוק של המשתמש המחובר מה המזל שלו כפי שמוגדר בגלגל המזלות. המחלקה תוסיף כל חבר כזה לתיבת ההדפסה והמשתמש יקבל רשימה של כל חברי הפייסבוק שלו בעלי אותו מזל כמו שלו.

* הפיצ'ר השני שהוספנו הוא חיפוש המעריץ הטוב ביותר- FindYourBestFan

המשתמש ילחץ על הכפתור ויקבל בחזרה את השם המלא והתמונה של החבר הפייסבוקי שלו שלחץ "Like" על הפוסטים / התמונות של המשתמש לפי בחירתו.

עבור מימוש הפיצ'ר יצרנו Form חדש ובו checkbox בו המשתמש יוכל לסנן על פי איזה פרמטר הוא רוצה לבחון את המעריץ הטוב ביותר (תמונות/פוסטים/שניהם) ובעת לחיצה על הכפתור "Find Your Best Fan" הוא יראה את התמונה ואת השם של המעריץ הגדול ביותר (זה שעשה את כמות ה"Likes" הגדולה ביותר).  
הלוגיקה של הפרמטרים עובדת בצורה זהה ב2 מחלקות שונות.

יצרנו מחלקה חדשה המבצעת את החישוב עבור כל חבר פייסבוק של המשתמש המחובר. המחלקה משתמשת בשירותי המחלקה FacebookWrapper שכן יש לבצע בדיקה עבור כל משתמש פייסבוקי.

בלחיצת כפתור אם סומן הפרמטר של הפוסטים - נבדוק עבור כל הפוסטים שהמשמש המחובר פרסם- מי החבר שעשה לפוסטים אלה הכי הרבה לייקים.   
אם סומן הפרמטר של התמונות - נבדוק עבור כל התמונות שהמשתמש המחובר פרסם - מי החבר שעשה לתמונות אלה הכי הרבה לייקים.  
אם שני הפרמטרים מסומנים – נפעיל את שתי המחלקות לבדיקת החבר הכי טוב ובfacade נחבר את התוצאות של שתי המחלקות ונכריע מי החבר הכי טוב מבין שתיהן.

### תבנית מס' 1 – Iterator

* **סיבת הבחירה / שימוש בתבנית:**

iterator הוא אובייקט שמאפשר לנו לעבור על איברי קבוצה נתונה, מבלי לדעת מה מבנה הנתונים בו מאוחסנים איברים הקבוצה. אובייקט ה-iterator מאפשר לנו מעבר על איברי הקולקציה, על ידי ממשק שאינו תלוי במימוש הפרטני של הקולקציה.

האיטרטור ממומש כך שאנו מאפשרים את אותן סט פעולות על כל סוגי מבני הנתונים.

בחרנו להשתמש בתבנית ה-Iterator כיוון שבאפליקציה שלנו יש מעבר על תמונות ואלבומי המשתמש. האלבומים מוחזקים במבנה נתונים של בו בכל איבר יש אוסף תמונות. התבנית מאפשרת לנו לעבור על התמונות מהאלבומים ללא צורך בידיעה מהו מבנה הנתונים והתחשבות בו בעת כתיבת הקוד לתכנית שלנו. כלומר, אם בעתיד נרצה לשנות את מבנה הנתונים עליו אנו עובדים נוכל לעשות זאת ללא שינוי בקוד.

* **אופן המימוש:**

יצרנו מחלקה חדשה ובה יצרנו את פעולת האיטרטור.

כאשר המשתמש בוחר לצפות במעריץ הטוב ביותר שלו לפי תמונות, מחלקת ה-client תפנה למחלקת PhotoIterator והוא ייצר עבורה איטרטור באמצעות פקודה ה-yiels return.

האיטרטור שנוצר יעזור ל-client לעבור על התמונות של המשתמש באמצעות פקודת ה-move next שקיימת לו ולראות מיהם החברים של המשתמש המחובר שעשו לו לייק.

* Sequence Diagram

[sequence diagram שמציגה את התהליך שקשור לתבנית והאינטראקציה בין המחלקות]

* Class Diagram

[class diagram שמתארת את המחלקות שלכם שמעורבות בתבנית (תיאור מלא שכולל Properties ו- Methods) והיחסים ביניהם (תאור מלא עבור כל יחס כפי שלמדנו בכיתה). עבור כל מחלקה שלכם, כיתבו מי המקבילה שלה (אם יש כזו) בתבנית (Pattern) שבחרתם

### תבנית מס' 2 – Strategy

* **סיבת הבחירה / שימוש בתבנית:**

תבנית זו נותנת פתרון הנדסי למימוש מנגנון שניתן לשנות את ההתנהגות שלו. התבנית הזו מאפשרת לנו לייצר קומפוזיציה בין המחלקה שמריצה את האלגוריתם המלא לבין המחלקה שמריצה התנהגות מסוימת של האלגוריתם, כך שאנו מחליטים בזמן ריצה מה ההתנהגות של האלגוריתם.

התבנית הזו מאפשרת לאלגוריתם שלנו להשתנות באופן עצמאי בין משתמשים שונים- בהתאם להגדרה שלנו, סוג הנתונים, רצון המשתמש ועוד..

תבנית זו מאפשרת לנו לבצע שינוי בהתנהגות מנגנון כלשהו מבלי לשנות את המנגנון ומבלי להשתמש בהורשה.

באפליקציה שלנו אנו מאפשרים למשתמש לנו לבחור כיצד הוא רוצה לראות את המעריץ הטוב ביותר שלו- כלומר אנו רוצים לאפשר לו לשנות את התנהגות המערכת, לפי בחירתו. בחירה אחת תניב תוצאה שונה מבחירה אחרת.

אנו מעוניינים לאפשר בחירה של מימוש בזמן ריצה כך שהתכנית שלנו תשתנה באופן עצמאי בין משתמשים שונים או ריצות שונות של התכנית. בנוסף, המימוש הנ"ל מאפשר לנו לשנות את המנגנון של התכנית ללא שימוש בהורשה כלל.

* **אופן המימוש:**

[תיאור המימוש והיכן ניתן למצוא אותו בקוד]

* Sequence Diagram

[sequence diagram שמציגה את התהליך שקשור לתבנית והאינטראקציה בין המחלקות]

* Class Diagram

[class diagram שמתארת את המחלקות שלכם שמעורבות בתבנית (תיאור מלא שכולל Properties ו- Methods) והיחסים ביניהם (תאור מלא עבור כל יחס כפי שלמדנו בכיתה). עבור כל מחלקה שלכם, כיתבו מי המקבילה שלה (אם יש כזו) בתבנית (Pattern) שבחרתם

### תבנית מס' 3 – [שם התבנית]

* **סיבת הבחירה / שימוש בתבנית:**

תבנית command היא תבנית שמייצגת פקודות. היא נועדה לעטוף את כל המידע הרלוונטי להרצת מתודות מסוימות (המופע ממנו קוראים לה, שם המתודה, הפרמטרים שנשלחים אליה ועוד) בתוך אובייקט יחיד.

אנו מגדירים ממשק בשם ICommand (ומכאן שם התבנית) וכל המתודות שלנו מוגדרות בתוך מחלקות (מחלקה לכל מתודה) כך שמחלקות הללו מממשות את ICommand. המחלקות הללו הינן מחלקות פנימיות בתוך ה-client.

* אופן המימוש:

[תיאור המימוש והיכן ניתן למצוא אותו בקוד]

* Sequence Diagram

[sequence diagram שמציגה את התהליך שקשור לתבנית והאינטראקציה בין המחלקות]

* Class Diagram

[class diagram שמתארת את המחלקות שלכם שמעורבות בתבנית (תיאור מלא שכולל Properties ו- Methods) והיחסים ביניהם (תאור מלא עבור כל יחס כפי שלמדנו בכיתה). עבור כל מחלקה שלכם, כיתבו מי המקבילה שלה (אם יש כזו) בתבנית (Pattern) שבחרתם