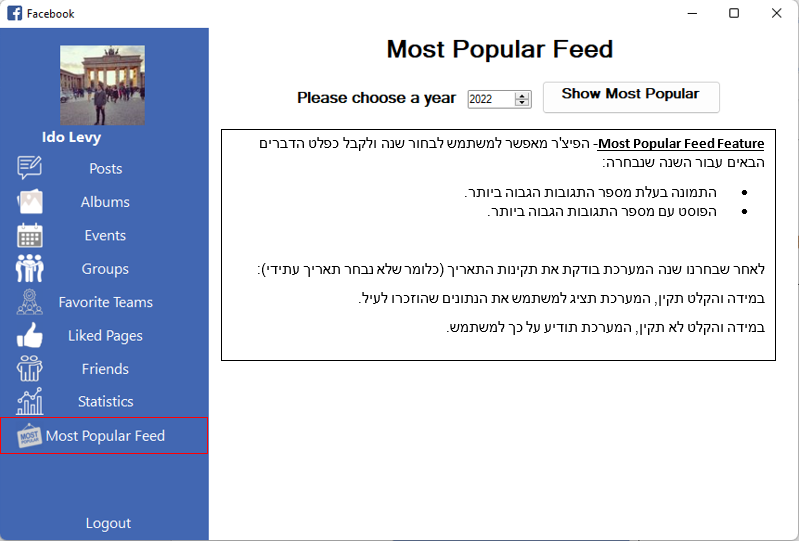
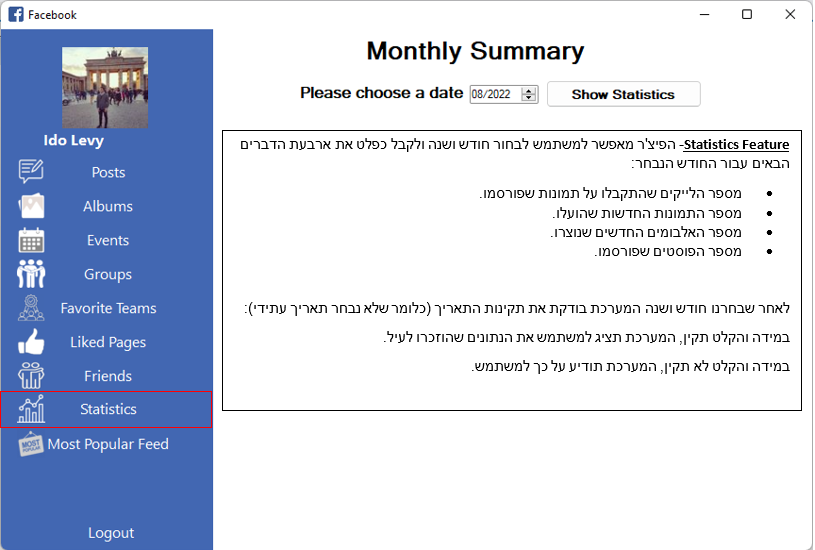
### תיאור קצר של הפיצ'רים שבחרנו לממש בתרגיל הקודם:





### תבנית מס' 1 – Strategy

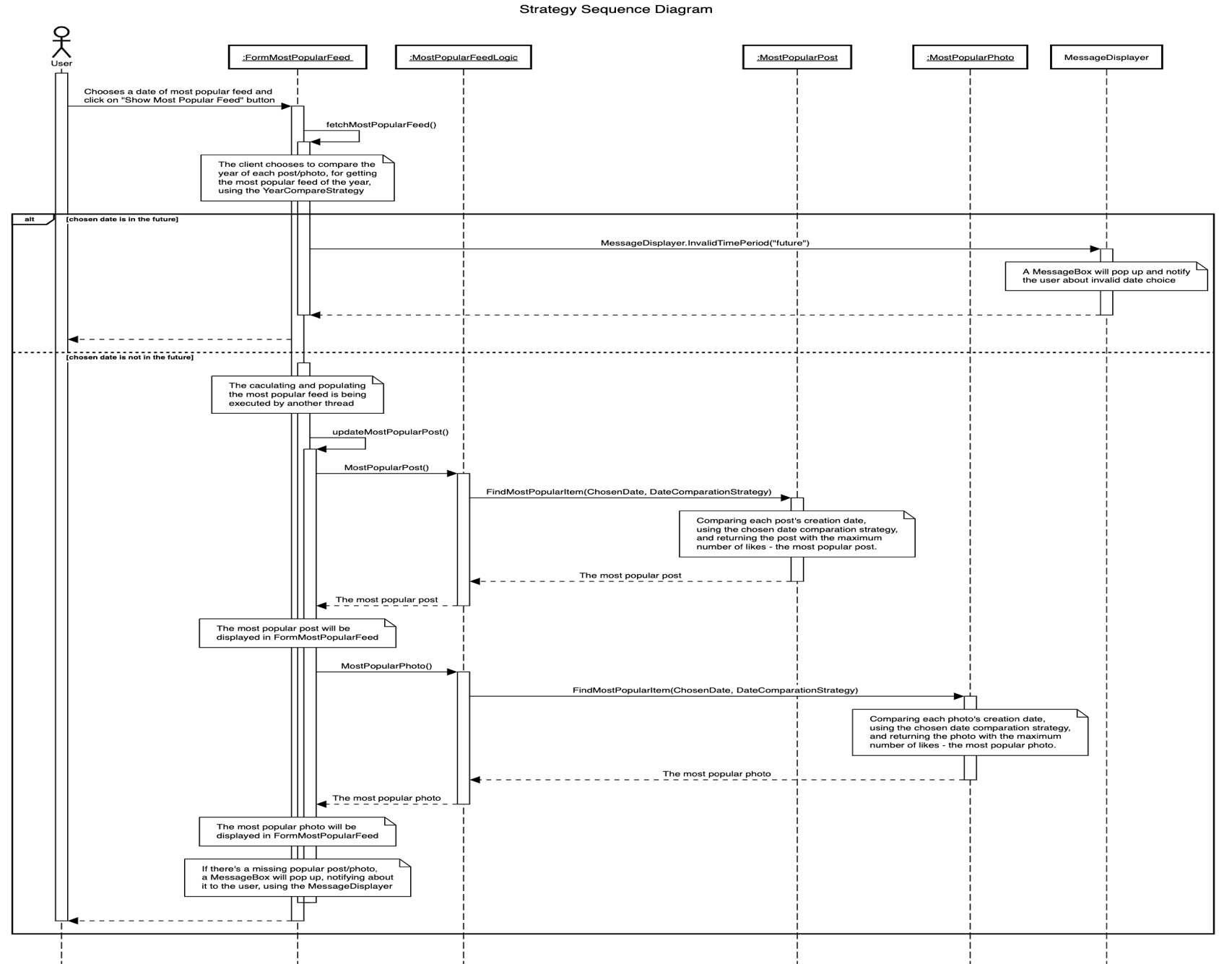
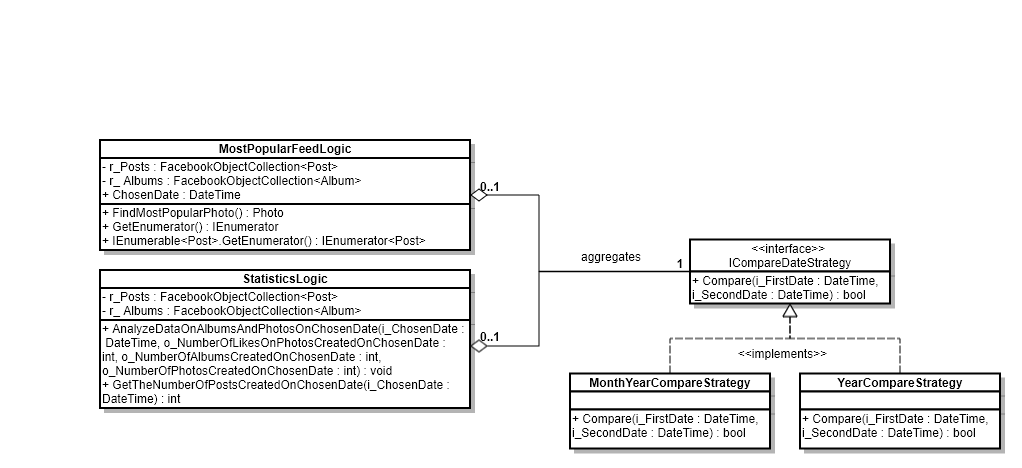
* **סיבת הבחירה / שימוש בתבנית:**

בחלקים שונים בתוכנית, נדרש לבצע השוואה בין 2 תאריכים שונים, על מנת לקבוע האם אובייקט מסוים עונה על קריטריונים הנחוצים לתפקוד תקין של המערכת.  
למשל, כאשר אנו מחפשים את הפוסט והתמונה הפופולאריים ביותר בשנה מסוימת, נדרש לבצע השוואה בין השנה שנבחרה לתאריך היווצרות האובייקט המבוקש.  
מכך עולה הבעיה הבאה: כי אם בעתיד נרצה למצוא את הפוסט והתמונה הפופולאריים ביותר בחודש ובשנה מסוימים, נדרש לשנות את **אסטרטגיית** ההשוואה.  
הפתרון: המשתמש יבחר באיזו אסטרטגיית השוואה הוא ירצה להשתמש, ובכך להשיג את מבוקשו, מבלי לשנות את מנוע המערכת אשר תפקידה למצוא את האובייקטים המבוקשים.

* **אופן המימוש:**

הממשק ICompareDateStrategy משמש כאסטרטגיית השוואה שיש לממשה.  
ניתן לראות מימוש ממשק זה במחלקות הבאות:   
YearDateCompareStrategy, MonthYearDateCompareStrategy.  
ניתן לראות שימוש בממשק במחלקות הבאות:

MostPopularFeedLogic, StatisticsLogic

* **Sequence Diagram**
* **Class Diagram**
* Context -> MostPopularFeedLogic, StatisticsLogic
* Strategy -> ICompareDateStrategy
* ConcreteStrategy -> MonthYearCompareStrategy, YearCompareStrategy

### תבנית מס' 2 – Observer

* **סיבת הבחירה / שימוש בתבנית:**

בעת התפתחויות מסוימות באפליקציה, נרצה להודיע למספר גורמים על שינויים הרלוונטיים עבורם, כמו למשל עבור כפתורי תפריט הצד באפליקציה, אשר אינם ניתנים לשימוש עד אשר המשתמש התחבר למערכת, ויוצאים משימוש כאשר המשתמש התנתק מהמערכת.

על כן, על הגוף האחראי על חיבור המשתמש למערכת האפליקציה (FacebookAccountManager) להודיע על התחברות/התנתקות המשתמש מהמערכת לכל הכפתורים הנ״ל.

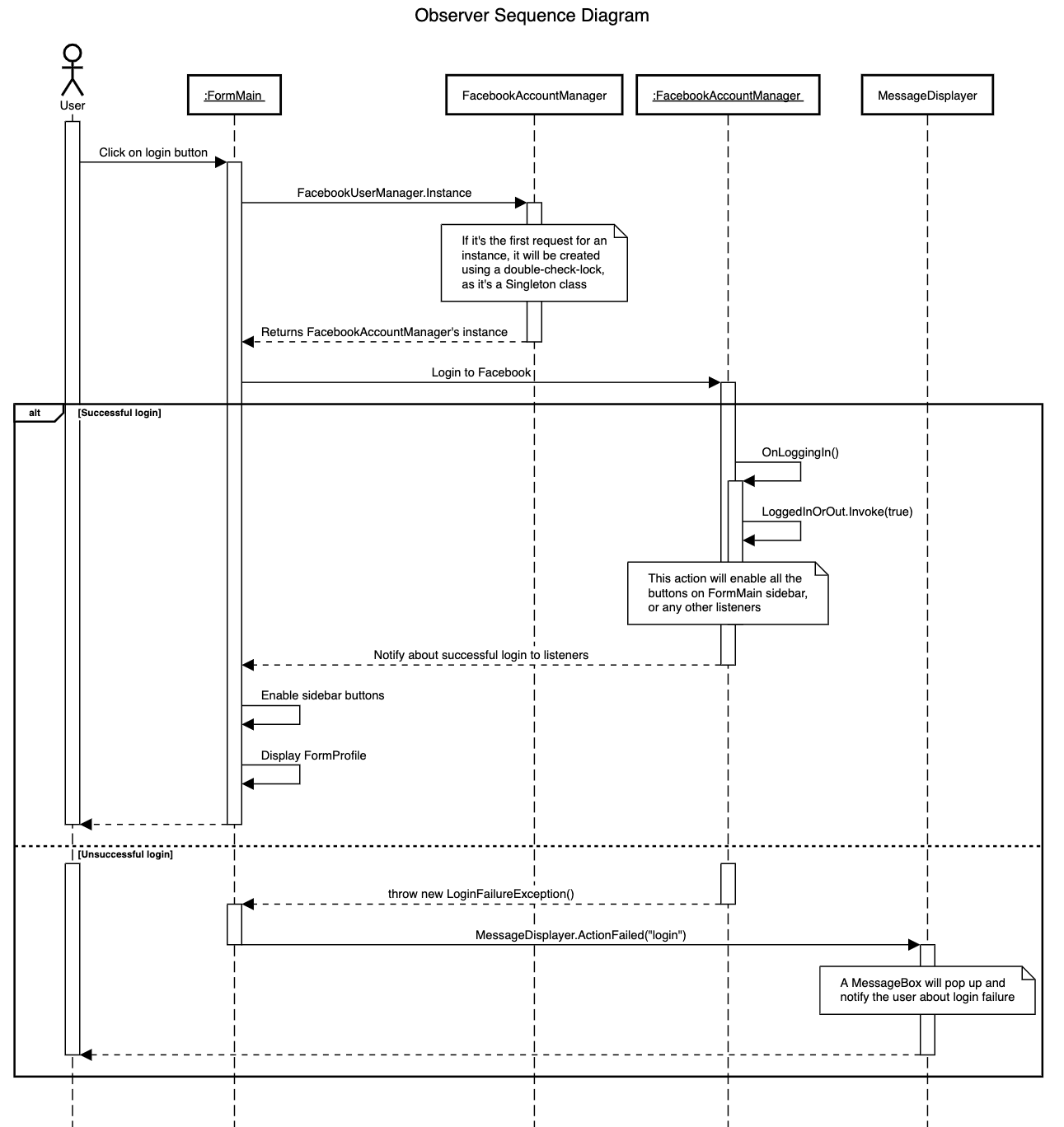
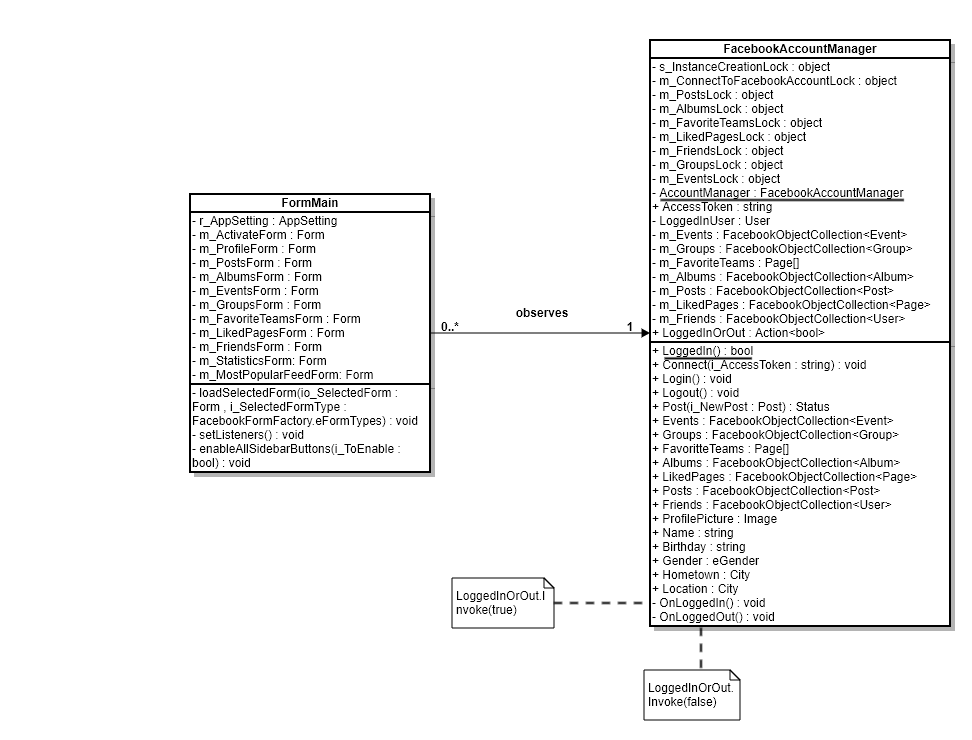
הבעיה העולה מכך: אם בעתיד נרצה שמספר גופים נוספים יעודכנו על התחברות/התנתקות המשתמש מהמערכת, נצטרך לעדכן את הקוד בגוף האחראי על התחברות והתנתקות מהמערכת, על אף שלא מתווספת אף לוגיקה חדשה.

הפתרון: יכולת רישום לעדכונים מהגוף האחראי על התחברות וניתוק המשתמש מהמערכת באופן יזום ע״י כל אובייקט אחר במערכת, ללא עירוב הגוף המעדכן. פתרון זה מאפשר למערכת להמשיך לתפקד כרגיל, ללא צורך בעדכון הקוד שלא בהכרח נגיש לנו.

* **אופן המימוש:**

הוספת Event בשם LoggingInOrOut ב-FacebookAccountManager, המאפשר לכל אובייקט במערכת אשר נדרש להתעדכן בעת התחברות וניתוק המשתמש מהמערכת להירשם ולהתעדכן על כך בזמן אמת.

המחלקה המשתמשת בכך היא FormMain.

* **Sequence Diagram**
* **Class Diagram**
* Subject -> FacebookAccountManager
* Observer -> FormMain

### תבנית מס' 3 – Template Method

* **סיבת הבחירה / שימוש בתבנית:**

במערכת ישנו פיצ׳ר המציג את הפוסט והתמונה הכי פופולאריים בשנה מסוימת, לפי בחירת המשתמש.

על כן, המשתמש פונה אל מערכת האפליקציה ומבקש את האובייקטים הנ״ל כנדרש.

הבעיה העולה מכך: מלבד ללוגיקת מציאת האובייקטים הפופולאריים, שאר התשתית זהה לחלוטין, שכן שתיהן כוללות השוואה לפי תאריך מסוים, באסטרטגיית השוואה מסוימת, ומחזיקות מבנה נתונים של האובייקטים המבוקשים.

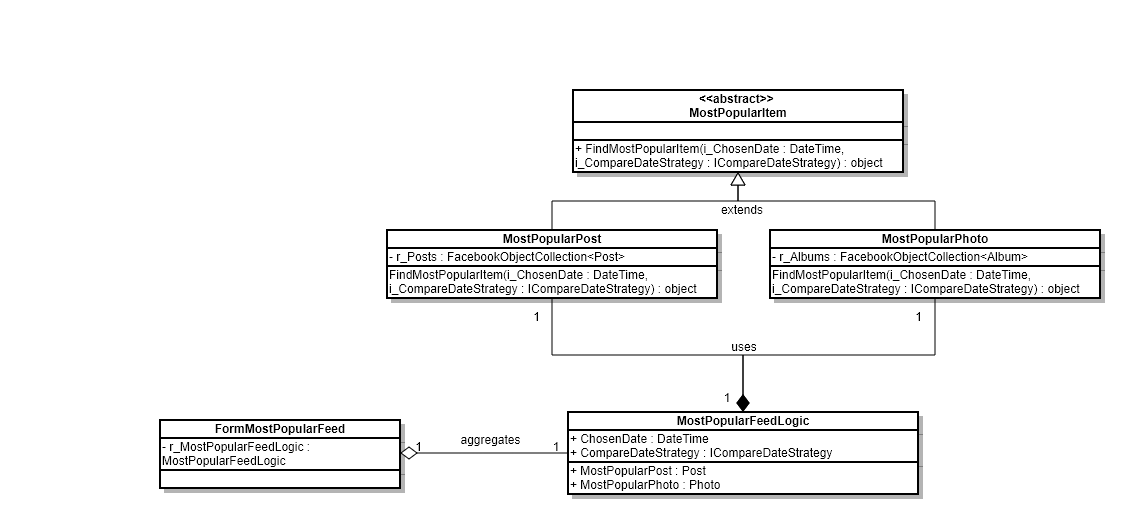
הפתרון: יצירת מחלקה אבסטרקטית, אשר מכיל מתודה אבסטרקטית בשם GetMostPopularItem, אשר מימושו משתנה בין כל מחלקה היורשת ממנו, האחראית על מציאת האובייקט הפופולארי ביותר בתחומה, ובכך נמנע שכפול קוד אשר אינו משתנה בין אופן מציאת האובייקט הפופולארי ביותר.

* **אופן המימוש:**

המחלקה MostPopularItem הינה המחלקה האבסטרקטית, המכילה את המתודה המדוברת, והמחלקות MostPopularPost ו-MostPopularPhoto יורשות ממנה ומממשות את המתודה GetMostPopularItem בהתאם לצרכן.

ניתן לראות שימוש במחלקות הנ״ל במחלקה MostPopularFeedLogic.

* Diagram, schematic

  Description automatically generated**Sequence Diagram**
* **Class Diagram**
* Abstract class -> MostPopularItem
* Concrete class -> MostPopularPost, MostPopularPhoto