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

* [תיאור קצר של הפיצ'ר הראשון]
* [תיאור קצר של הפיצ'ר הראשון]

### תבנית מס' 1 – Observer

* סיבת הבחירה / שימוש בתבנית:

בחרנו להשתמש בתבנית  **Observer** משום שהיא **מאפשרת עדכון אוטומטי של מספר רכיבים שונים במערכת בכל פעם שמתרחש שינוי במצב המשחק**.

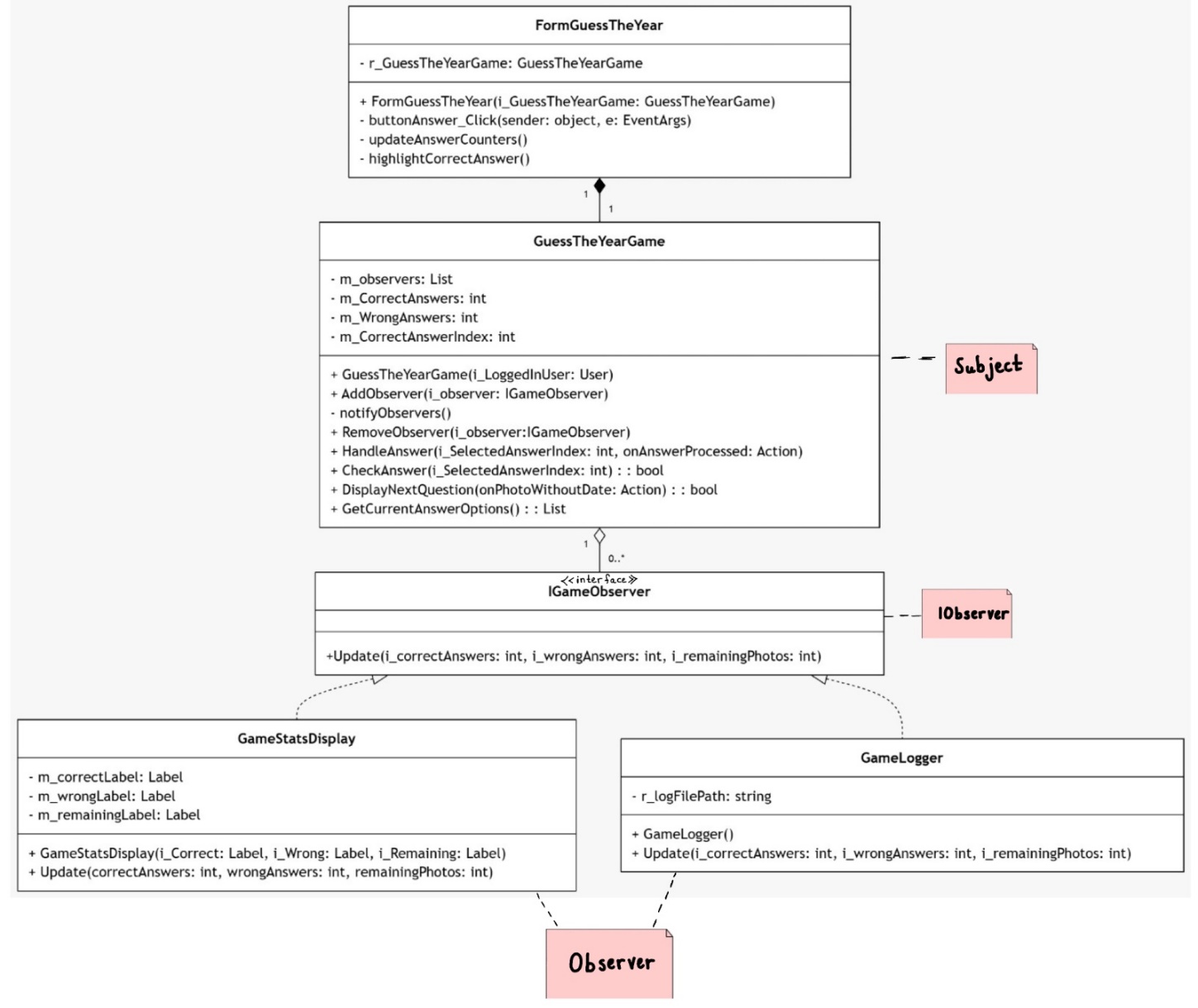
* **ריכוז האחריות בעדכון הצופים** – התבנית מאפשרת לרכז את הלוגיקה לעדכון רכיבי ממשק המשתמש (UI) ורישום הנתונים במקום אחד (GuessTheYearGame), במקום לקרוא ידנית לכל עדכון בנפרד.
* **תמיכה במערכת גמישה ומדרגית** – ניתן להוסיף ולנהל מספר בלתי מוגבל של "צופים" (Observers) מבלי לשנות את הלוגיקה המרכזית של המשחק.
* **הפחתת תלות הדדית בין מחלקות** – התצוגה (FormGuessTheYear) אינה תלויה ישירות במחלקת המשחק (GuessTheYearGame), אלא מתעדכנת אוטומטית דרך הצופים.
* **תמיכה בכמה סוגים של תגובות לשינוי** – לדוגמה, יש צופה (GameStatsDisplay) שמעדכן את ה-Labels במסך, ויש צופה אחר (GameLogger) שמתעד את השינויים בקובץ לוג (GameLog.txt).
* **שיפור ביצועים והימנעות מקוד מיותר** – במקום לבצע קריאות ישירות ועדכונים רבים בנפרד, GuessTheYearGame פשוט קורא **ל- notifyObservers()**, וכל הצופים מתעדכנים יחד.
* אופן המימוש:
  + ממשק IGameObserver :
    - מגדיר את המתודה

Update(correctAnswers, wrongAnswers, remainingPhotos).

* + - כל מחלקה שמקשיבה לשינויים במשחק חייבת לממש את הממשק הזה.
  + מחלקת GameStatsDisplay (צופה שמעדכן את ה-UI ) :
    - שומרת הפניות לתוויות (Labels) במסך המשחק.
    - כאשר המשחק מתעדכן, הוא **מעדכן את ה-Labels אוטומטית** דרך Update().
  + מחלקת  **GameLogger**(צופה ששומר מידע בקובץ GameLog.txt:
    - רושם לוגים על התקדמות המשחק לקובץ טקסט.
    - שומר את מספר התשובות הנכונות, השגויות והתמונות שנותרו.
  + **מחלקת GuessTheYearGame (הנושא-Subject)**
    - מחזיקה רשימה של כל ה) Observers -צופים).
    - ספקת פונקציות AddObserver() ו notifyObservers() –
    - בכל פעם שמתקבלת תשובה או שמתחילה שאלה חדשה, **כל הצופים מתעדכנים אוטומטית**. בכל פעם שמתקבלת תשובה או שמתחילה שאלה חדשה, **כל הצופים מתעדכנים אוטומטית**.
  + חיבור הצופים למשחק (FormGuessTheYear):
    - בעת יצירת המשחק, אנו מוסיפים את הצופים
    - עת, בכל שינוי במשחק, **גם ה-UI וגם קובץ הלוגים מתעדכנים אוטומטית**.
* Sequence Diagram

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

* Class Diagram



### תבנית מס' 2 – [שם התבנית]

* סיבת הבחירה / שימוש בתבנית:

בחרנו להשתמש **בתבנית Strategy** במערכת שלנו מכיוון שהיא מספקת **פתרון גמיש ומודולרי** לסינון רשימת החברים לפי קריטריונים שונים.

במערכת שלנו יש צורך **בסינון דינמי של רשימת חברים** לפי קריטריונים שונים, כמו:

* + **שפות משותפות** (CommonLanguagesStrategy)
  + **עיר מולדת משותפת** (SameHometownStrategy)
  + **תאריך לידה זהה** (SameBirthdayStrategy)
  + **חברים שעשו לייק לתמונות** (PhotoLikesStrategy)

**היתרונות המרכזיים של Strategy במערכת שלנו:**

* **מונע כפילויות בקוד** – במקום לכתוב לוגיקת סינון נפרדת בכל מקום, יש לנו ממשק אחיד לכל האסטרטגיות, והקוד **מרוכז ומסודר** במקום אחד.
* **מאפשר החלפה דינמית של קריטריוני סינון** – במקום להשתמש ב-if-else מסובכים, ניתן **להחליף את האסטרטגיה בזמן ריצה** מבלי לשנות קוד קיים.
* **שומר על עקרון ה-SOLID** – במיוחד **עקרון הפתיחות לסגירה (Open/Closed Principle)** – אם רוצים להוסיף **סוג חדש של סינון**, אין צורך לשנות קוד קיים, אלא רק להוסיף אסטרטגיה חדשה.
* **משפר את הביצועים** – אין צורך **ליצור מחדש את מחלקת ניהול הסינון בכל פעם**, אלא רק מחליפים את האסטרטגיה הנוכחית.
* אופן המימוש:

המערכת שלנו מאפשרת למשתמש **לסנן חברים** לפי קריטריונים שונים.  
כדי לעשות זאת, חילקנו את הקוד למבנה הבא:

1. **ממשק IFriendFilterStrategy** – מגדיר את הפעולה שכל סוג של סינון חייב לממש.
   * לדוגמה: כל אסטרטגיית סינון צריכה להחליט **האם לכלול חבר ברשימה או לא**.
2. **מימושים שונים של אסטרטגיות סינון** – כל שיטת סינון מיושמת כמחלקה נפרדת, למשל:
   * CommonLanguagesStrategy – מחזירה חברים עם שפה משותפת.
   * SameHometownStrategy – מחזירה חברים מאותה עיר מולדת.
   * SameBirthdayStrategy – מחזירה חברים עם תאריך לידה זהה.
   * PhotoLikesStrategy – מחזירה חברים שעשו לייק לתמונות.
3. **מחלקת FriendFilterContext** – מנהלת את אסטרטגיית הסינון.
   * מקבלת **אסטרטגיה ראשונית** (למשל, סינון לפי שפה משותפת).
   * מאפשרת **להחליף אסטרטגיה בזמן ריצה** כדי לשנות את קריטריון הסינון.
   * מבצעת את **הסינון בפועל**, תוך שימוש בלולאה שעוברת על החברים ובודקת אם הם מתאימים לקריטריון.
4. **מחלקת FriendAnalyzer** – משתמשת ב-FriendFilterContext כדי להפעיל את הסינון בפועל.
   * יש לה פונקציות שמבצעות **שינוי אסטרטגיה** לפני הרצת הסינון.
   * משתמשת ב-FriendFilterContext יחיד, כך שהמערכת **לא יוצרת מחדש את האובייקט כל פעם**, אלא רק מחליפה אסטרטגיה.

* 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:

[תיאור הסיבה / הצורך בשימוש בתבנית במערכת שלכם]

* 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