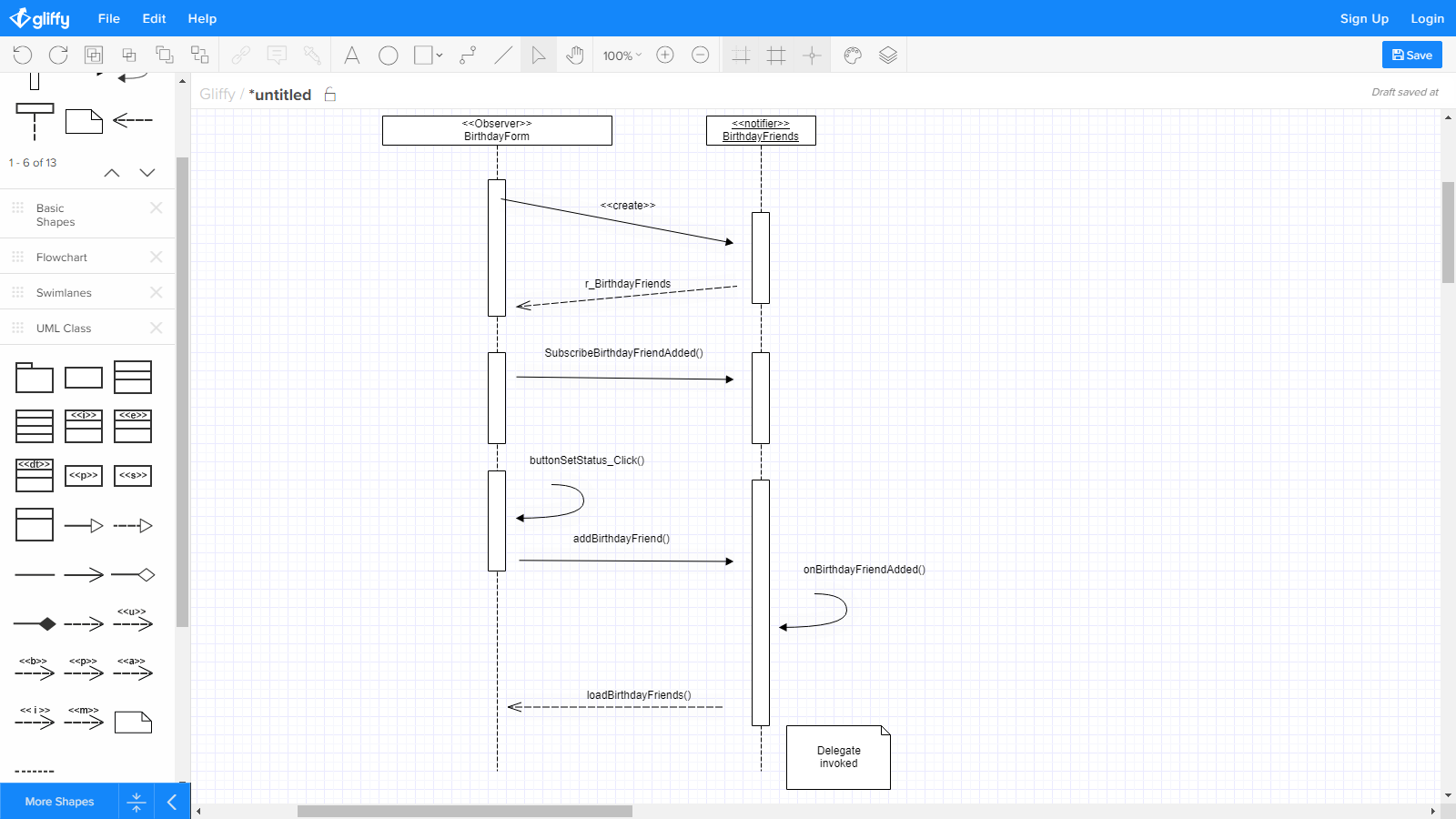
### תבנית מס' 1 – Observer

* סיבת הבחירה / שימוש בתבנית:  
  לפני השימוש בתבנית, המשתמש יראה את כל האנשים שחברים שלו בפייסבוק שיש להם יום הולדת בפיצ'ר היום הולדת, כולל אנשים שלא רלוונטים עליו.  
  אנו מעוניינים ליצור פיצ'ר שיראה רק את החברים "החשובים" שחוגגים יום הולדת על ידי כך שנאפשר גישה אל פאנל החברים שיש להם יום הולדת רק אם המשתמש לחץ בעבר על כפתור "פירסום פוסט יום הולדת" שבנינו.  
  בנוסף, הרכיבים יוכלו לדעת מתי המשתמש ביצע לחיצה על "פירסום פוסט יום הולדת" בצורה שאינה קושרת בין מחלקת המשתמש למחלקת החברים שנרשמו ע"י על ידי notifer.
* אופן המימוש:

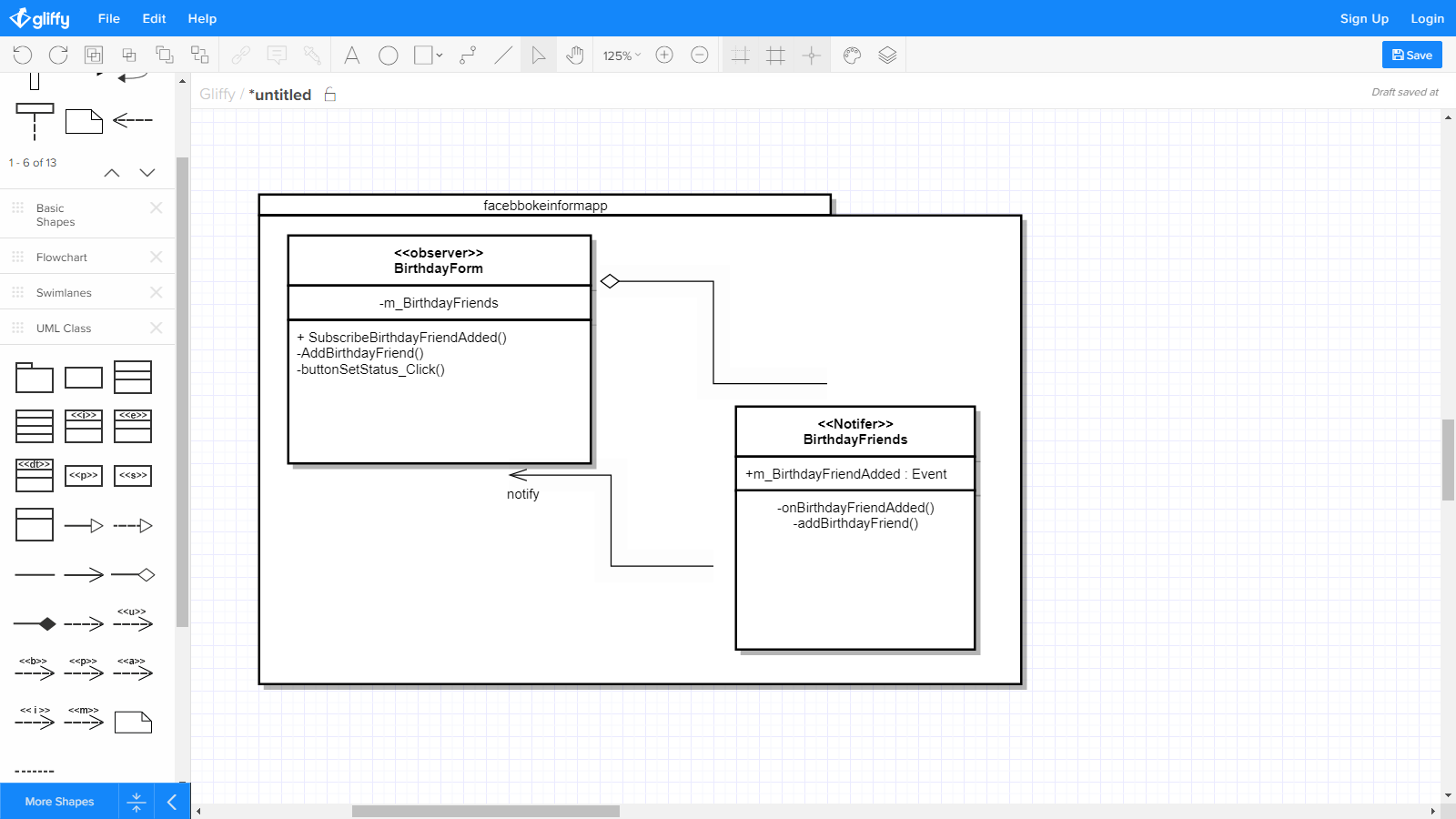
הגדרנו delegate במחלקת המודיע BirthdayFriends עם access modify – event ואליו נרשם כל מאזין שירצה לדעת על לחיצת כפתור "פרסום פוסט יום הולדת" לחבר שחוגג יום הולדת.

באפליקציה שלנו המאזין הינו BirthdayForm אשר רוצה לדעת מתי נלחץ "פירסום פוסט יום הולדת" לחבר וזו על מנת לאפשר את הגישה אל הפנל אשר מציג את חברי היום הולדת הספציפים.

* Sequence Diagram



* Class Diagram



### תבנית מס' 2 – Iterator

* סיבת הבחירה / שימוש בתבנית:

במחלקת BirthdayFriends אנו מחזיקים סט של חברים שיש להם יום הולדת, מכיוון שמהותה של מחלקה זו היא לחבר בין המשתמש לבין החברים שיש להם יום הולדת, נרצה לתת את האפשרות למעבר על מבנה נתונים זה. אנו לא רוצים להגביל את מימוש מחלקה זו למבנה נתונים מסויים לכן ממשנו איטרטור אשר יתן אפשרות לעבור על חברים שיש להם יום הולדת ללא ידיעה באופן מימוש מבנה הנתונים.

* אופן המימוש:

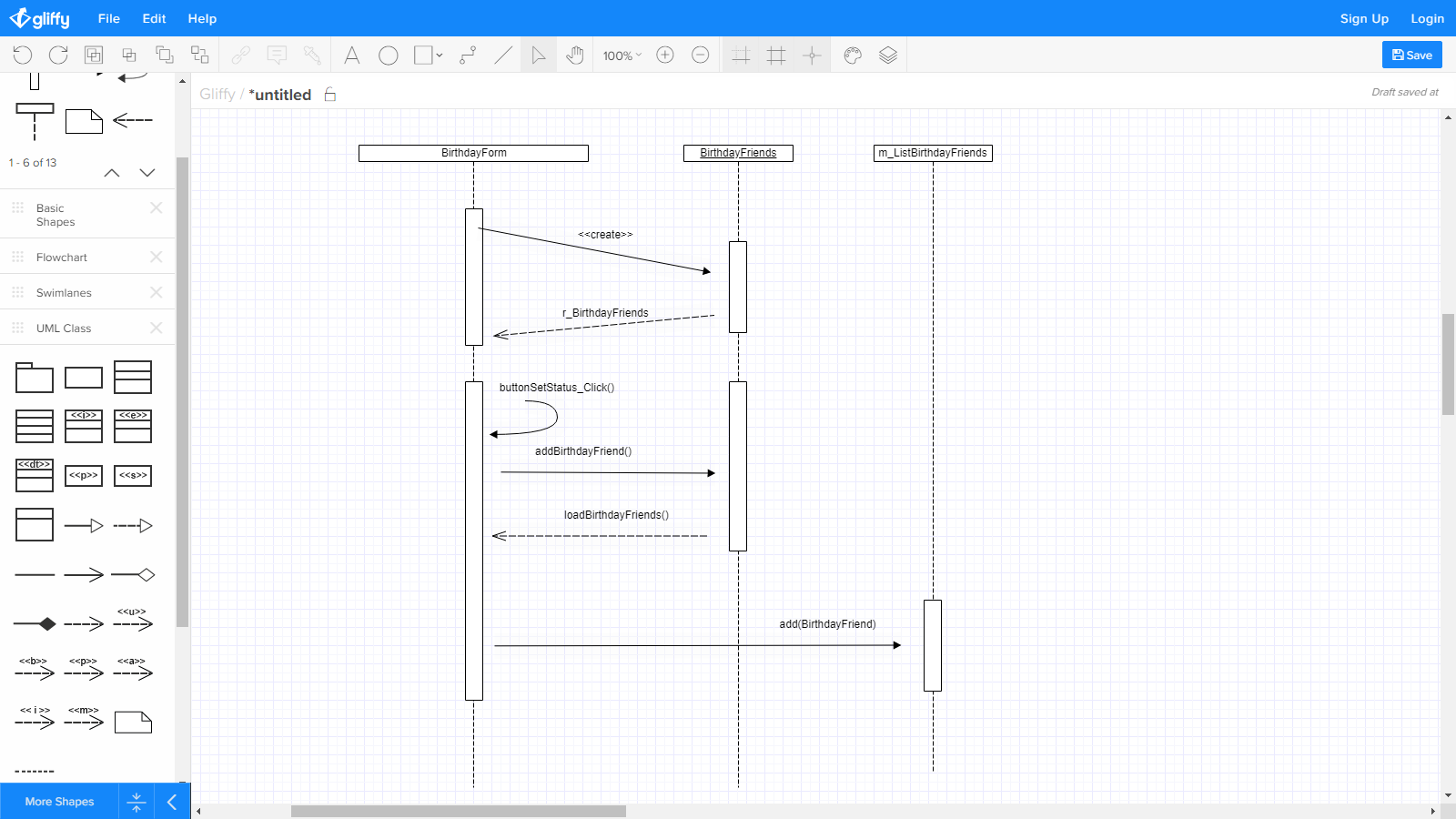
ממשנו איטרטור על המחלקה BirthdayFriends אשר עובר על מבנה הנתונים המחזיק את החברים שיש להם יום הולדת ע"י מימוש הממשק IEnumerable ומימוש המתודה GetEnumerator.

את המתודה GetEnumerator ממשנו באמצעות foreach על הסט ושימוש בפיצ'ר

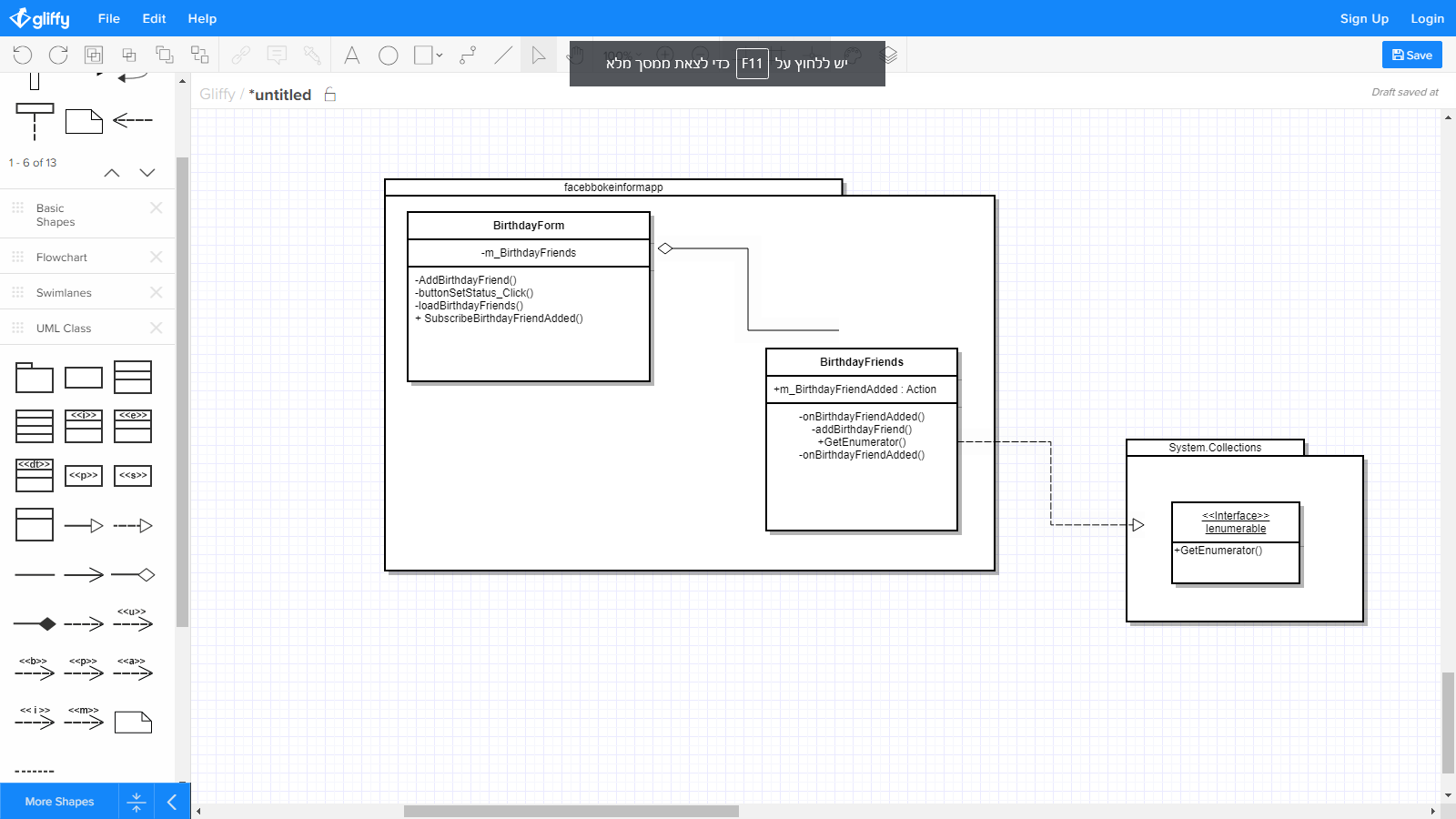
yield return.

מחלקת BirthdayForm משתמשת באיטרטור של המחלקה BirthdayFriends על מנת למלא אובייקט UI אשר מציג את רשימת החברים שיש להם יום הולדת.

* Sequence Diagram



* Class Diagram



### תבנית מס' 3 – Strategy

* סיבת הבחירה / שימוש בתבנית:

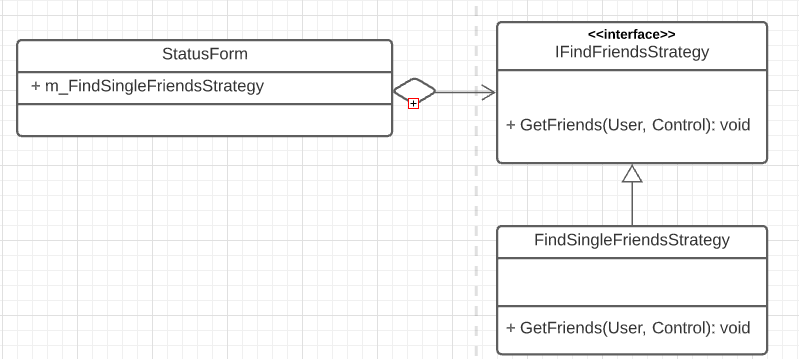
בתוך המחלקה DatingForm קיימת מתודה לסינון החברים על פי סטטוס היחסים שלהם. החשיבה היא שבעתיד נרצה להוסיף מנועי סינון וחיפוש נוספים ולכן יהיה נכון לממש זאת באמצעות strategy.   
בכדי למנוע את הצורך בשינוי הלוגיקה יצרנו בסיס פולימורפי IFindFriendStrategy ובו הגדרנו מתודה שהיא אבסטרקטית המשמשת אותנו כ- injection point שיהיה בעצם abstract method ב- abstract creature נפרד.  
כלומר, זה ישמש אותנו כמן "חור" שניתן "להזרקה" בעתיד, להחלפה בעתיד, קרי בעתיד יהיה אפשר לשנות את התוצאה ואת ההתנהגות של המנגנון מבלי לשנות את הקוד שלו ובלי לממש אותו מחדש, כלומר בלי לשכפל את הקוד שלו.

תתקיים קומפוזיציה בין המנגנון לבין הבסיס הפולימורפי, שאותו אם נרצה נוכל לשנות בעתיד.  
המתודה אחראית על קבלת שני פרמטרים כפאקטורים חיצוניים ובעזרתם למצוא את החבר הרנדומלי בהתאם לקטוגריה שנבחרה ע"י המשתמש.

* אופן המימוש:

המחלקה האבסטרקטית IFindFriendStrategy יש לה מתודה .GetMyRandomFriend   
אשר מקבלת שני פרמטרים כפאקטורים חיצוניים ובעזרתם מוצאת את החבר המתאים.  
החשיבה היא שכל מתודה שתצטרך בצע חיפוש בחברים של המשתמש תחזיק מופע של הstrategy הרלוונטי אליה.

* Class Diagram



* Sequence Diagram

