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

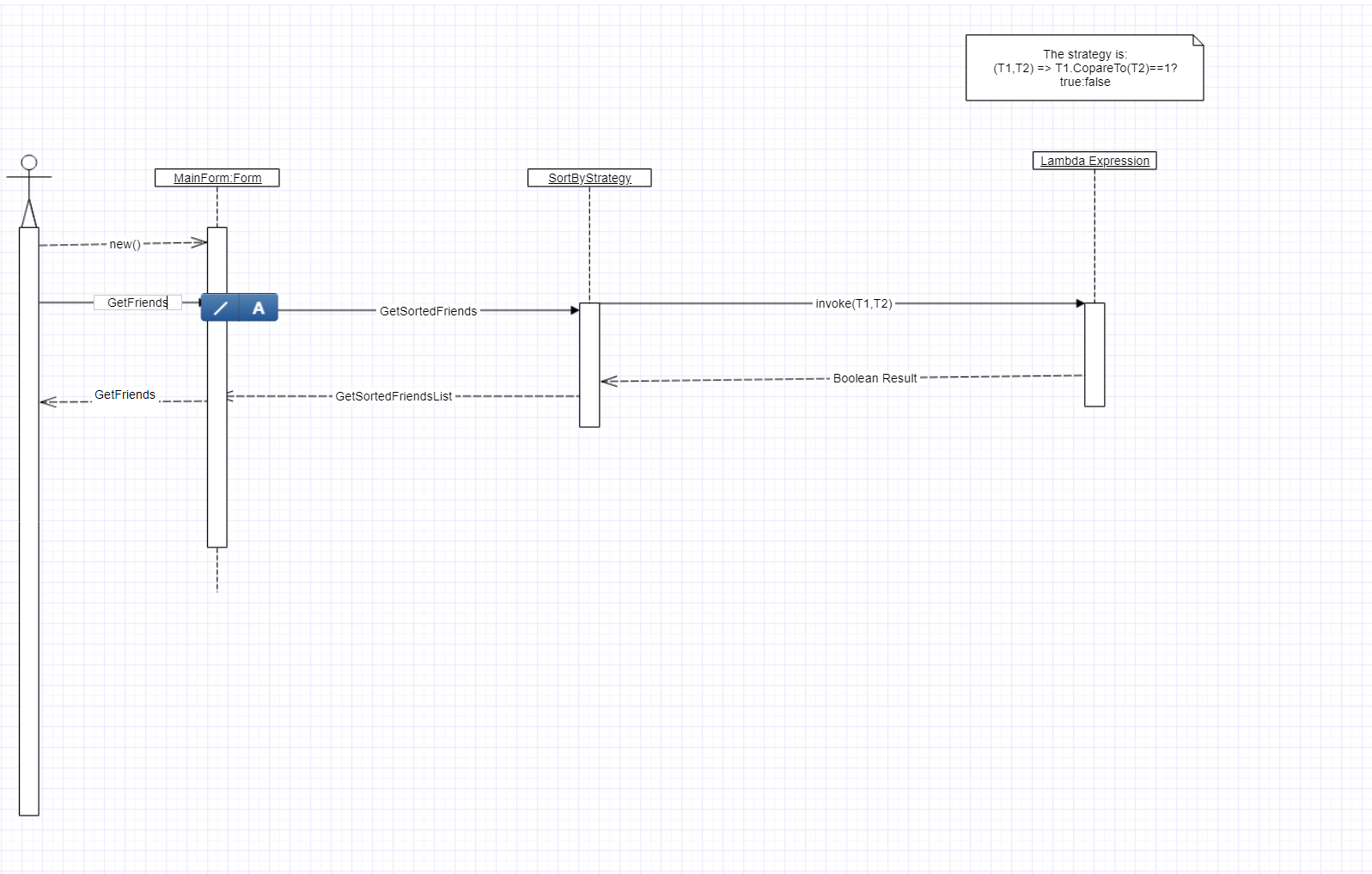
* בפיצר הראשון אפשרנו למשתמש לחפש אהבה לפי פרמטרים שנקבעו מראש, ע"י מאפיינים כמו דת, שפה וכו' שחשוב לו שלו ול"משודך" יהיה אותם במשותף.
* הפיצ'ר השני - הפיצ'ר מאפשר למשתמש ולחבריו לתכנן ולקבוע פעילות ספורטיבית חברתית, בהינתן תחזית מזג האוויר ל5 ימים הקרובים - כדי שיוכלו לבחור את היום המתאים ביותר לאותה פעילות.

### תבנית מס' 1 – [Strategy]

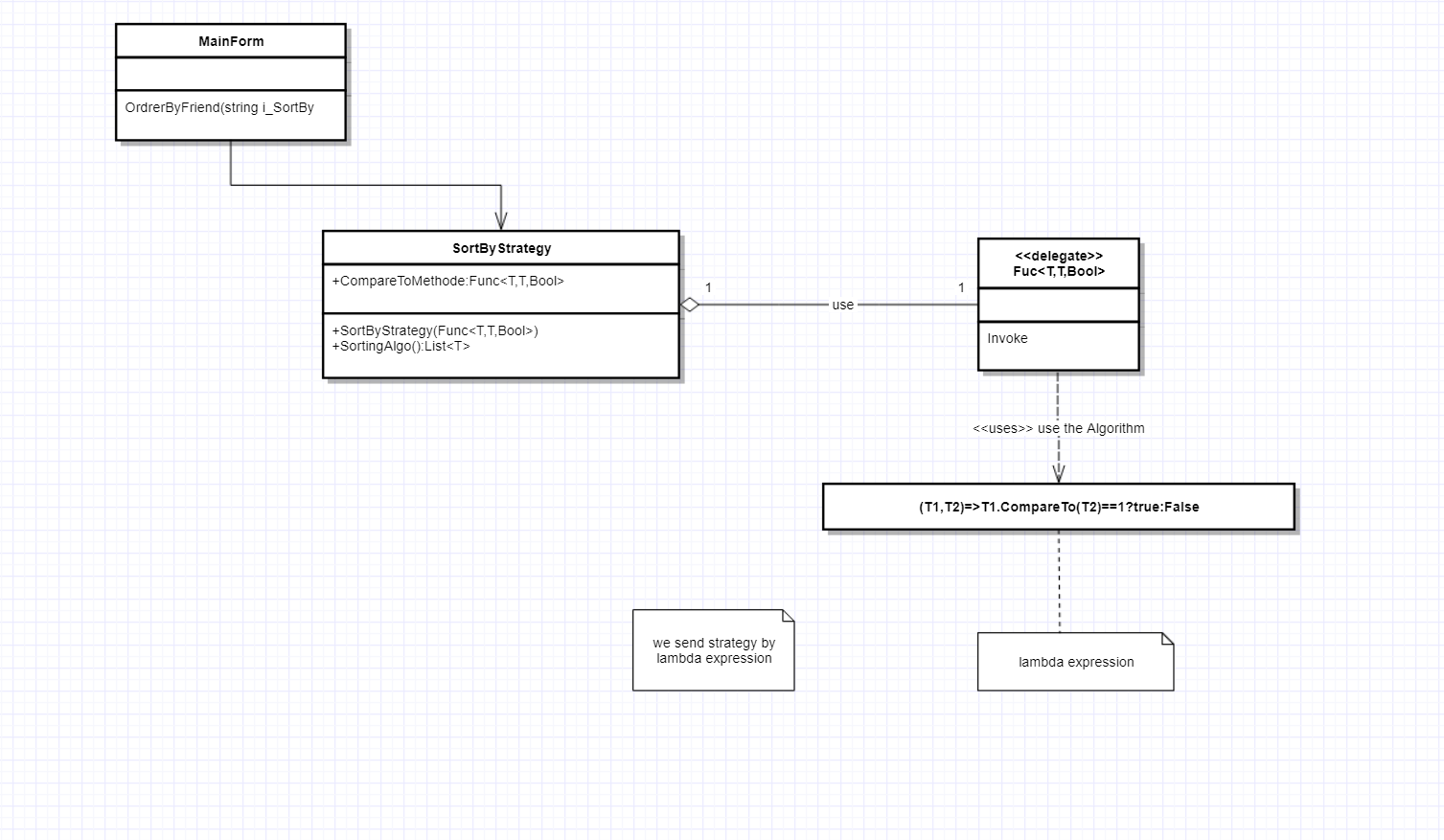
* סיבת הבחירה / שימוש בתבנית:

הסיבה בגללה בחרנו לממש Strategy Pattern במערכת שלנו נבעה מהצורך למיין את הרשימות אותן אנו מציגים למשתמש. במערכת שלנו כאשר לוחצים על כפתור FacebookMode יש אופציה להציג את רשימת החברים של המשתמש ועוד. לרשימה הנ"ל רצינו להוסיף אופציה למיון לפי קטגוריות משתנות. מאחר והמיון נעשה באותו אופן ורק אסטרטגיית המיון השתנה, בהתאם לבחירת המשתמש, בחרנו להשתמש ב-Strategy Pattern כך שבכל פעם שנרצה למיין רשימה, נבחר רק באסטרטגיה הרצויה ולפי בחירת המשתמש נמיין את הרשימה. כמו כן בעולם המקצועי יתכן ובעתיד נרצה להוסיף עוד פיצ'רים שמממשים מיון ושכפול קוד זה עוד יחזור על עצמו. כאן באה לידי ביטוי תבנית ה Strategy וזאת מפני שרצינו את היכולת להחליט בזמן ריצה ("בעתיד"), איזה אלגוריתם של הצגת תוצאה נרצה שיקרה, וזאת מבלי לשכפל קוד.

* אופן המימוש:
* את המימוש ביצענו באמצעות מחלקה בשם SortByStrategy כשאר המחלקה היא מסוג <T>, כלומר, מחלקה גנרית אליה נוכל לשלוח כל מיני טיפוסים בהתאם לבחירת המשתמש.
* המחלקה מחזיקה בתוכה member מסוג delegate Func אליו נשלח בכל פעם את האסטרטגיה הרצויה, כשאר התבנית היא <T, T, bool>.
* בתוך המחלקה קיימת מתודה Sort שמחזירה רשימה מסוג T כאשר בתוך המתודה אנו ממשים מיון באותה הצורה כאשר שאלת ההשוואה וההחלטה האם לבצע החלפה(מיון) היא לפי האסטרטגיה אותה קיבלנו מהמשתמש בתוך ה-delegate Func. אותה נפעיל באמצעות Invoke יחד עם שני פרמטרים T1 ו-T2 אותם נשלח להשוואה.
* Sequence Diagram



* Class Diagram

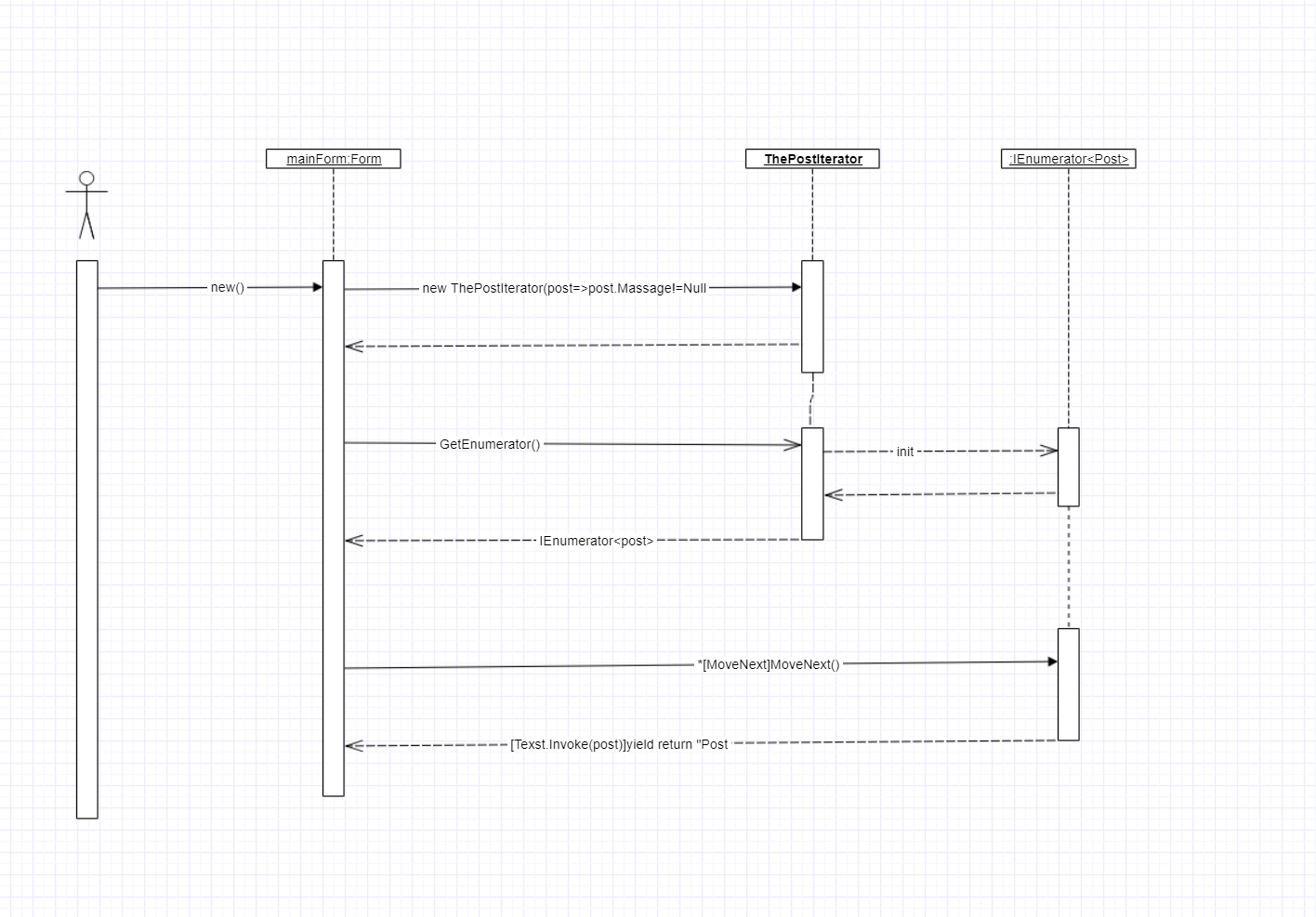


### תבנית מס' 2 – [Iterator]

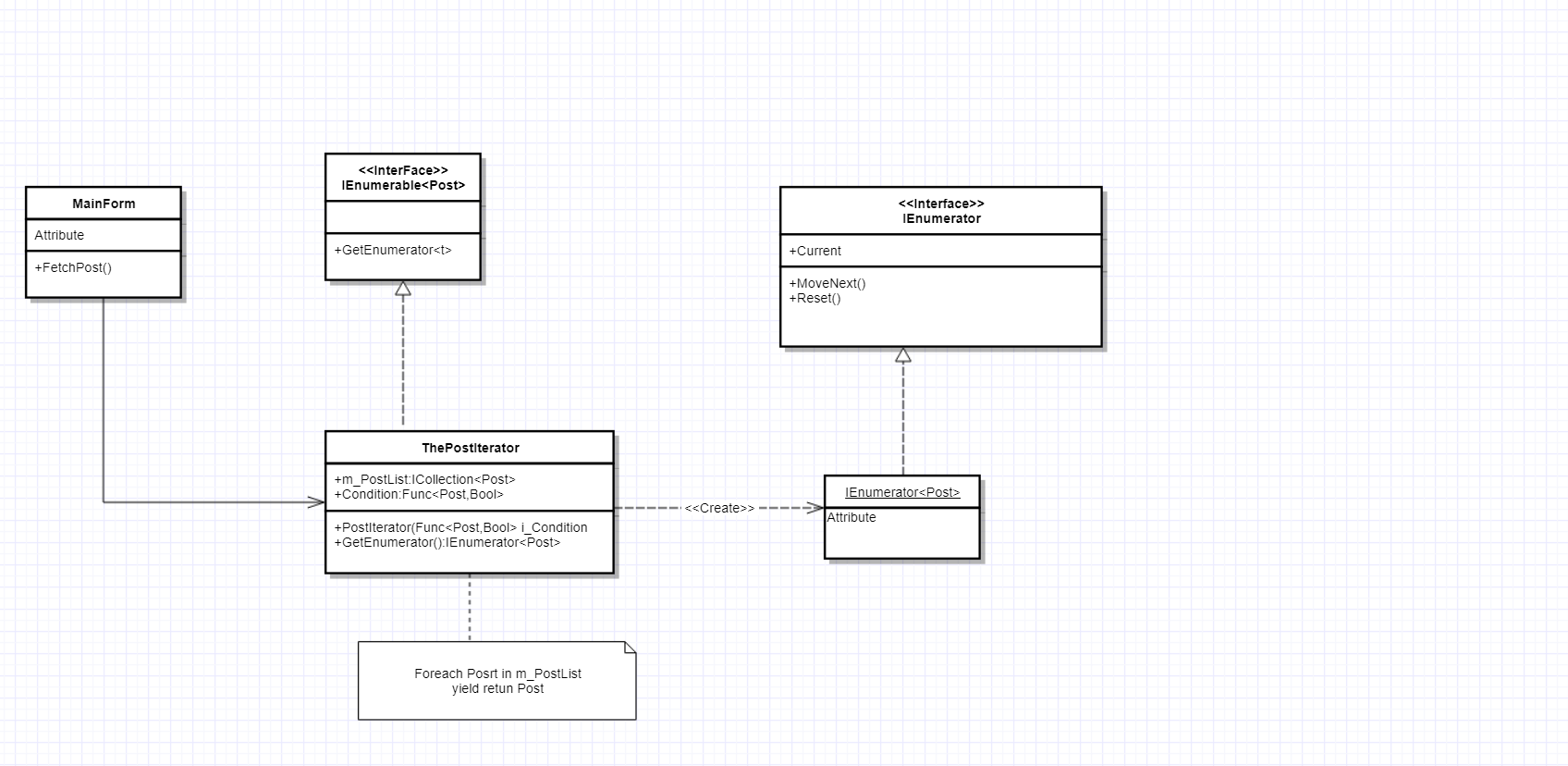
* סיבת הבחירה / שימוש בתבנית:

הסיבה בגללה בחרנו לממש Pattern Iterator במערכת שלנו נבעה מהצורך לקבל את כל ה- Posts של המשתמש שבהם יש messege שאינו ריק (ולאחר מכן להציגם) ובנוסף רצינו שתהייה לנו את האפשרות לעבור על ה- Posts ללא תלות במימוש מבנה הנתונים שלהם.

* אופן המימוש:
* את המימוש ביצענו באמצעות מחלקה בשם FetchPostsIterator שיורשת מ- IEnumerable<Post> כאשר המחלקה מחזיקה בתוכה private members מסוג ICollection<Post> שבהפעלת ה-constractor תחזיק בתוכה את רשימת כל ה- Posts של המשתמש (ה- Posts עם message ריק ועם message שאינו ריק) ובנוסף delegate Func מסוג Func<Post, bool> בשם Condition כאשר תפקידו יהיה לשמור את הטסט לפיו נחזיר ב- yield return את ה- Post , במקרה של הקוד שלנו, כאשר נצהיר על new PostIterator (במחלקה MainForm) נשלח בתור פרמטר ל-constractor את שורת הקוד הבאה:
* post => post.Message != null. וכשאר נעבור ב-Foreach על האובייקט שיצרנו (מסוג PostIterator בתוך במחלקה MainForm ) התכנית תקרא מאחורי הקלעים למתודה GetEnumerator (שנמצאת במחלקה PostIterator) שמחזירה IEnumerator<Post> (מחזירה איטרטור).
* בתוך המתודה GetEnumerator נעבור על כל ה- Posts של המשתמש (אותם שמרנו ב-private member) וכל Post שעובר את ה-Test נחזיר באמצעות yield return.
* באופן הזה אנו מאפשרים יצירת איטרטור מסוג PostIterator יחד עם אפשרות סינון (Test) שבקוד שלנו בחרנו לסנן את ה- Posts שה-message שלהם ריק.
* Sequence Diagram



* Class Diagram



### תבנית מס' 3 – [Observer]

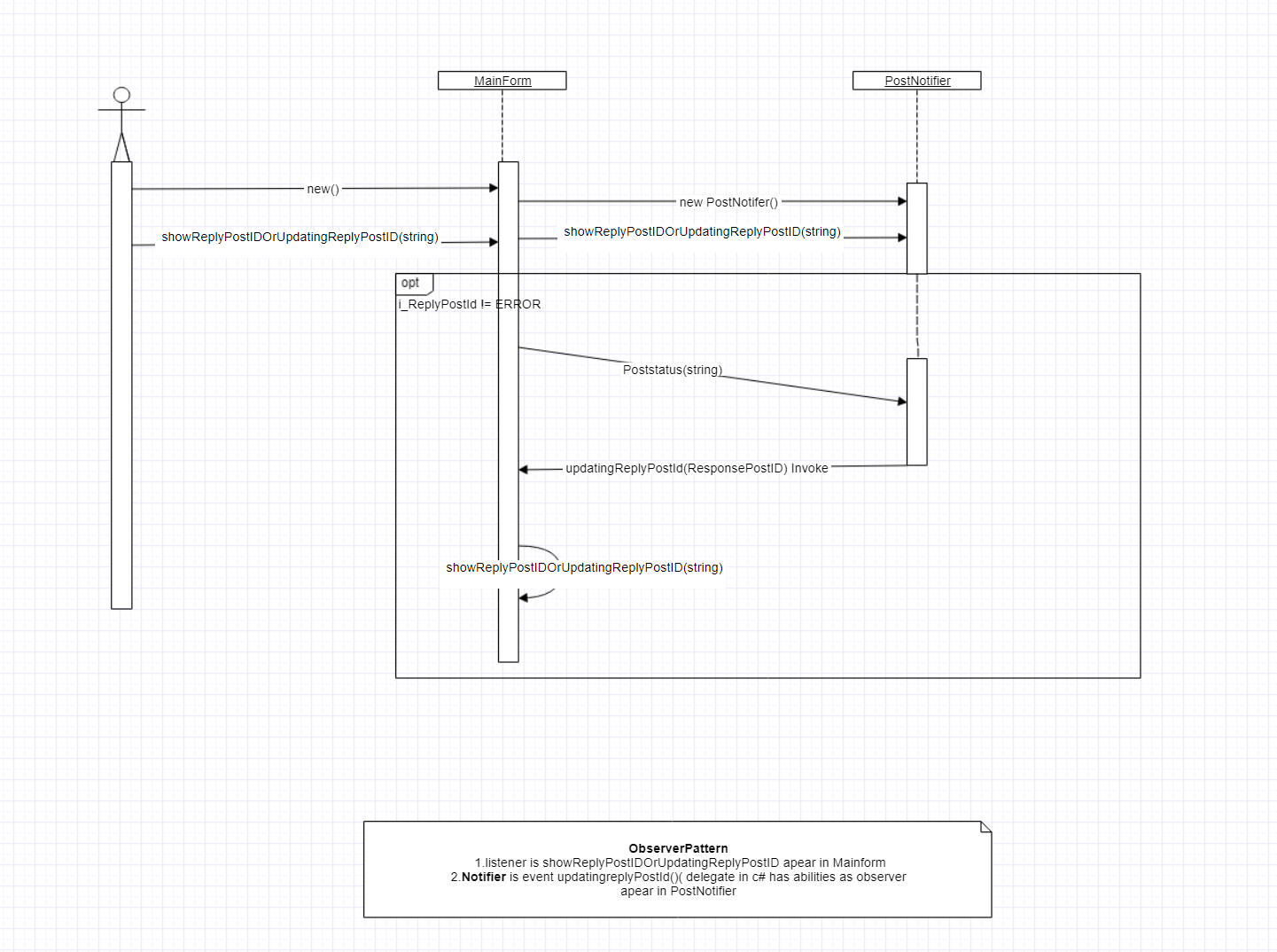
* סיבת הבחירה / שימוש בתבנית:

הסיבה בגללה בחרנו לממש Observer Pattern במערכת שלנו נבעה מהצורך להודיע על תשובת ה-Post שהתקבלה מפרסום ה-Post בFacebook , משכבת הלוגיקה לשכבת ממשק המשתמש, תשובה זו מעידה על פרסום Post בהצלחה מכיוון שלא רצינו שיהיה קשר ציקלי בין שתי הישויות השתמשנו בתבנית observer שמאפשרת בעצם loose coupling. , לכן זהו חיווי רצויי למשתמש.

* אופן המימוש:
* את המימוש ביצענו כך: במחלקת PostNotifier הוספנו את הרכיב הבא:
* public event Action<string> UpdatingReplyPostID , כאשר רכיב זה ב-C# שמממש delegate בכתיב קצר, והוא משמש כ- **Notifier**, כלומר פונקציות יכולות להירשם אליו ובכך להאזין לכל אירוע שהוא מפרסם, כאשר פונקציית PostStatus(string) מופעלת, נשלח בקשה לפרסום Post חדש ב- facebook, אם תהליך זה מסתיים בהצלחה חוזרת הודעה שמעידה על הצלחה עם ה PostID שמסמל ערך ייחודי לפרסום, ערך מוחזר זה מועבר לפונקציה הבאה:onUpdateReplyPostID(string) אשר בה נעשה Invoke לכל המאזינים של ה-event, ובכך נוצר אירוע שמפורסם לכל המאזינים. אם התהליך של פרסום ה-Post נכשל בפונקציית PostStatus(string) נזרקת שגיאה וחוזר ערך Error בתור PostId.
* מחלקת MainForm מייצרת מופע של PostNotifierבתהליך האתחול שלה,

הפונקציה הבאה:

* showReplyPostIDOrUpdatingReplyPostID(string)
* נרשמת ל- event שתואר לעיל במופע מטיפוס PostNotifier,פונקציה זו משמשת כ-**Lisetner** עבור ה- event שתואר לעיל.
* כאשר אירוע נוצר בתהליך לעיל, פונקציה הנ"ל מקבלת קריאה להתבצע, ובביצועה מציגה לממשק המשתמש את הערך המוחזר מפרסום ה-Post בתוספת הודעה מתאימה שמתארת את ערך זה.
* Sequence Diagram



* Class Diagram

