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

* **פיצ'ר ראשון- טיימר ליום הולדת הבא:**

כאשר המשתמש יבצע התחברות דרך האפליקציה יופיע שעון עצר שסופר ימים, שעות, דקות ושניות **עד** ליום הולדת הבא. השעון מתעדכן באופן אוטומטי כל שניה.

* **פיצ'ר שני- טיימר לאירוע:**

לאחר התחברות המשתמש תופיע רשימת האירועים. כאשר נשים את הסמן על אחד מהאירועים יופיע כלי תיאור (ToolTip) שבו מוצג הזמן שעבר **מאז** האירוע או הזמן שנותר **עד** מועד האירוע. הזמן מוצג בפורמט של ימים, שעות, דקות ושניות.

### תבנית מס' 1 – Strategy

* סיבת הבחירה / שימוש בתבנית:

בתוכנית שלנו יש מחלקה CollectionAdapter<T> שמחזיקה קולקציה בקומפוזיציה וממירה אותה למערך של מחרוזת.

ההמרה של כל פריט בקולקציה למחרוזת יכולה לקרות בעזרת הפעלת המתודה ToString ,

אך במקרה זה אנו מגבילים את השימוש של CollectionAdapter<T> רק על קולקציות עם איברים שמומשה להם המתודה ToString.

כדי להמנע משכפול קוד במקרה של המרת קולקציה לקולקציות מחרוזות באמצעות אסטרטגיית המרה שתיהיה רלוונטית תמיד, כדי שהקוד יהיה תחזוקתי יותר, וכדי שנוכל לאפשר שינוי של אסטרטגיית ההמרה של איברים, בחרנו בתבנית Strategy.

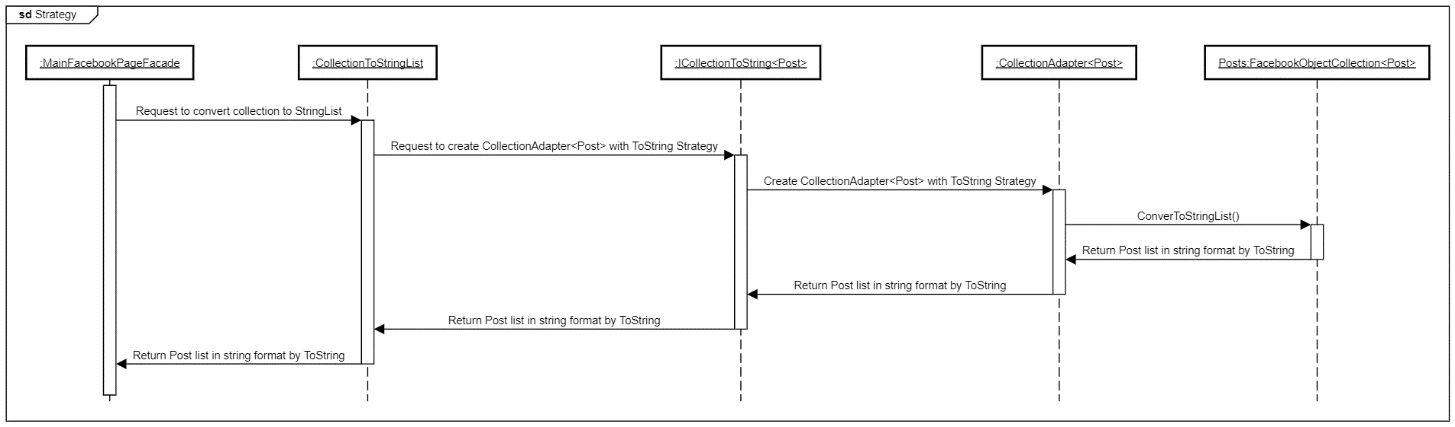
* אופן המימוש:

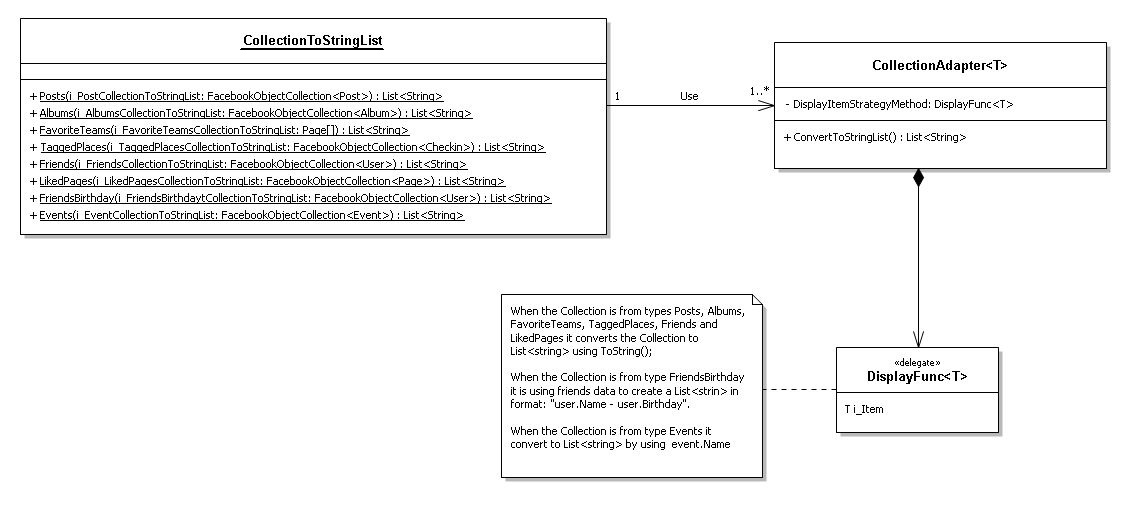
המחלקה CollectionAdapter<T> מחזיקה בקומפוזיציה, בנוסף לקולקציה שהיא ממירה, את אסטרטגיית ההמרה.

אסטרטגיית ההמרה היא delegate מסוג DisplayFunc<T> שמקבל T, כך ש T זה סוג האיבר בקולקציה, ומחזירה מחרוזת שאליה נמיר את האיבר מהקולקציה.

כמו כן המתודה ConvertToStringList (שנמצאת ב CollectionAdapter<T> ) רצה על הקולקציה שרוצים להמיר, מפעילה על כל איבר בקולקציה את אסטרטגיית ההמרה(שמוחזקת בקומפוזיציה בCollectionAdapter<T>) ומוסיפה את המחרוזת שהתקבלה למערך של מחרוזות שמוחזר בסיום המתודה.

בפועל, המחלקה הסטטית CollectionToStringList שאחראית לעשות את ההמרות של קולקציות למערך מחרוזות,בכל אחת מהמתודות שלה, יוצרת CollectionAdapter<T> עם T שונה, ונותנת לו (בקומפוזיציה) בעזרת למדה אקספיישן את אסטרטגיית ההמרה הרלוונטית.

* Sequence Diagram
* Class Diagram



* CollectionAdapter<T>- מקבילה ל- Context.
* DisplayFunc<T>-היא Delegate ומקביל ל- Strategy.
* השתמשנו ב- Delegate ולא ב- Interface כי יש צורך להחזיק בקומפוזיציה מטודה אחת, לכן ConcreteStrategy לא מופעים בדיאגרמה.

### תבנית מס' 2 – Observer

* סיבת הבחירה / שימוש בתבנית:

בתכנית שלנו יש רכיבים בתוך MainFacebookPageFacade אשר מתעדכנים (הנתונים מ- User LoggedInUser). נרצה כאשר הם מתעדכנים שיודיעו לכל מי שרוצה להתעניין שהתבצע שינוי (בלי להכיר את המתעניין). המחלקה FormFacebookPage היא המתעניינת בשינויים ורק כאשר יתבצע שינוי היא תציג את העדכון.

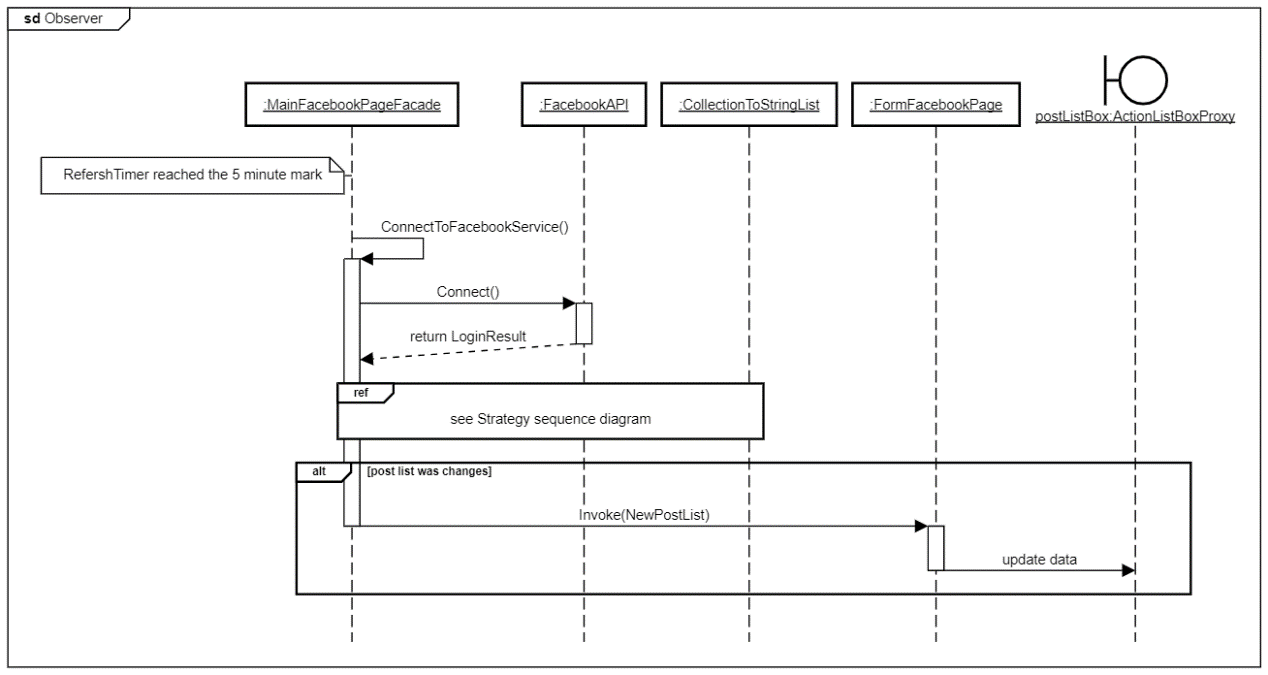
* אופן המימוש:

המחלקה MainFacebookPageFacade מחזיקה event, שהוא delegate שמקבל רשימה של מחרוזות, עבור כל הקולקציות מ- User LoggedInUser שהשתמשנו בהם. כאשר מתבצע שינוי מופעלת מטודה המודיעה (notifier) לכל מי שמאזין ל- event ע"י invoke. בתוך ה- invokeנשלח העדכון (רשימה של מחרוזות).

העדכון מתבצע באופן יזום ע"י RefreshTimer הנמצא ב- MainFacebookPageFacade שמבצע אותו כל חמש דקות.

המחלקה FormFacebookPage (המחלקה המאזינה) רושמת לכל event את המטודה שתופעל כאשר הוא יודיע על שינוי. המטודות המאזינות יקבלו את הרשימות של המחרוזות ויעדכנו את ה- ListBoxesשנמצאים ב- UI. חשוב לציין שכל מטודה מעדכנת ListBox אחר.

* Sequence Diagram



* Class Diagram



* MainFacebookPageFacade- מקבילה ל-Notifier.
* FormFacebookPage- מקבילה ל- Listner.
* בגלל השימוש ב- delegetes לא היה צורך להשתמש ב- Interface או מחלקות נוספות.

### תבנית מס' 3 – Decotator

* סיבת הבחירה / שימוש בתבנית:

במערכת שלנו יצרנו שני סוגים של ListBoxes ייחודיים. אחד מהבהב כאשר ה- ListBox מתעדכן והשני משמיע צליל כאשר ה- ListBox מתעדכן. בסוג הראשון השתמשנו בכל ה- ListBoxes חוץ מה- ListBox של הפוסטים.

ב- ListBoxשל הפוסטים רצינו שבעת עדכון ה- ListBox יבצע את שתי הפעולות (הבהוב והשמעת צליל).

כל ListBox מיוחד (מהבהב/ מצלצל) הוא סוג של פרוקסי(של ListBoxes) עם תוספות- הבהוב, השמעת קול ואולי עוד סוגי ListBoxes שנרצה להוסיף בעתיד.

הצורך לקומבינציה של שני סוגי ListBoxes ואולי עוד קומבינציות בעתיד, ללא שכפול קוד וכן הצורך להפריד בין המימושים של כל ListBox מיוחד ולא לרשום אותם תחת אותה המחלקה, הביא אותנו להשתמש בתבנית Decorator.

* אופן המימוש:

יצרנו Interface בשם IDecoratedListBox שמשמש כבסיס הפולימורפי. את ה- Interface ממשות המחלקות ActionListBoxProxy ו- ActionListBoxDecorator וממשות מטודה בשם operation.

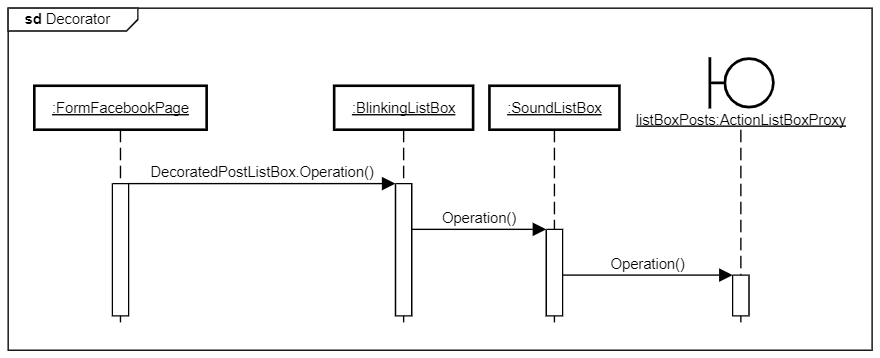
ActionListBoxProxy היא מחלקה שבנוסף למימוש ה- Interface יורשת מ- ListBox ומאפשרת עבודה עם ListBoxes תחת הבסיס הפולימורפי של ה- Interface.

ActionListBoxDecorator בנוסף למימוש ה- Interface, מחזיק אותו בקומפוזיצה.

באופן הזה כל ListBox מיוחד יורש מ- ActionListBoxDecorator, יחד עם הירושה ה- ListBoxes המיוחדים גם מחזיקים בקומפוזיציה, גם בעלי הבסיס הפולימורפי של ה- Interface וגם ממשות את operation שבה מפעילות את התכונה המיוחדת שלהם.

ActionListBoxDecorator מקבל ב- Constractor אובייקט מסוג IDecoratedListBox. בזכות זאת ניתן ליצור אובייקטים בעלי תכונות מרובות ע"י שרשור של ListBoxes ב- Constractor.

* Sequence Diagram



* Class Diagram



* FormFacebookPage- מקבילה ל- Client.
* IDecoratedListBox- מקבילה ל- IDecotated.
* ActionListBoxProxy- מקבילה ל- CoreDecorated.
* ActionListBoxDecorator- מקבילה ל- Decorator.
* BlinkingListBox- מקבילה ל- Decorator1.
* SoundListBox- מקבילה ל- Decorator2.