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

* [הפיצ'ר הראשון שבחרנו לממש : **BEST FRIENDS** כאשר המשתמש בוחר בפיצ'ר זה הוא מקבל את רשימת החברים שלו ממויינים לפי מספר הלייקים ומספר ה comments שכל אחד מחבריו עשה לו (כמובן במקרה בו כן הייתה גישה לרשימת החברים ול "LikeBY"), כאשר לכל חבר יש ניקוד המתקבל באופן הבא: עבור like יקבל 2 נקודות ועבור comment 2.5 נקודות. לבסוף נמיין את רשימת החברים לפי שדה הניקוד כאשר הניקוד הגבוהה ביותר ידורג ראשון וכן הלאה.

בנוסף לאחר הצגת רשימת ה best friends ל list box יוכל המשתמש לבחור בכל אחד מהחברים הופיעים שם ולראות עבור כל אחד את כמות ה likes ה comments ואת הניקוד שקיבל (כמובן הכל בהנחה שפייסבוק API יאפשר גישה לנתונים אלה)]

* [הפיצ'ר השני שבחרנו לממש: **CATEGORY PHOTOS** בפיצ'ר הזה המשתמש יכול לבחור מבין 2 הקטגוריות המופיעות לו ב combo box (countries ו- date) עבור הבחירה ב country נחלץ מתוך שרת הפייסבוק את כל אחת מהמדינות בהן הועלתה כל אחת מהתמונות, ועבור הבחירה date נחלץ משרת הפייסבוק את כל התאריכים בהן הועלו כל אחת מהתמונות. יוצג למשתמש

ב list box המדינות או התאריכים (תלוי בבחירתו). בנוסף המשתמש יוכל לבחור כל אחד מבין רשימת המדינות/תאריכים ולפי בחירתו תוצג ב list box נוסף רשימת התמונות השייכות לאותה בחירה. וגם כאן יוכל המשתמש לבחור בתמונה מסויימת להצגת התמונה עצמה.

לדוגמה: המשתמש בוחר countries ---> מוצגות לו כל המדינת בהן הוא העלה תמונה ----> המשתמש בוחר ב Thailand -----> מוצגת לו רשימת התמונות שהועלו בתאילנד ----> המשתמש בוחר בתמונה מסויימת להצגתה ---> מוצגת לו התמונה ]

### תבנית מס' 1 – [Iterator]

* סיבת הבחירה / שימוש בתבנית:

[שימוש ב Iterator על פני שימוש במבנה נתונים ספציפי נותן יתרון של encapsulation בכך שאנו מסתירים את פירטי מבנה הנתונים מהקליינט.

בנוסף, אם בעתיד נרצה לשנות את מבנה הנתונים של האיברים המוחזקים במחלקה CountriesPhotoManager, הקליינט ימשיך לעבוד עם הנתונים באותו אופן מבלי לשנות את הקוד ובכך הקוד יהיה קל יותר לתחזוקה כיוון שיש הפרדה בין הקליינט לבין המבנה הפנימי של האובייקט.]

* אופן המימוש:

[את המימוש ניתן למצוא בקוד במחלקה CountriesPhotoManager שזוהי בעצם המחלקה שבה הקומפיילר יוצר מחלקה חדשה המממשת את IEnumerable ואת IEnumerator

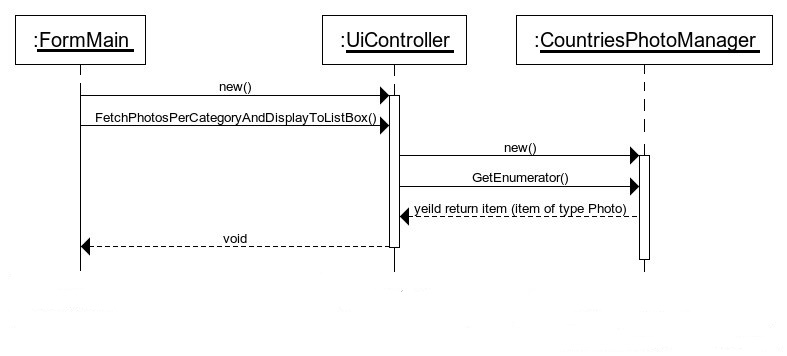
תיאור המימוש:

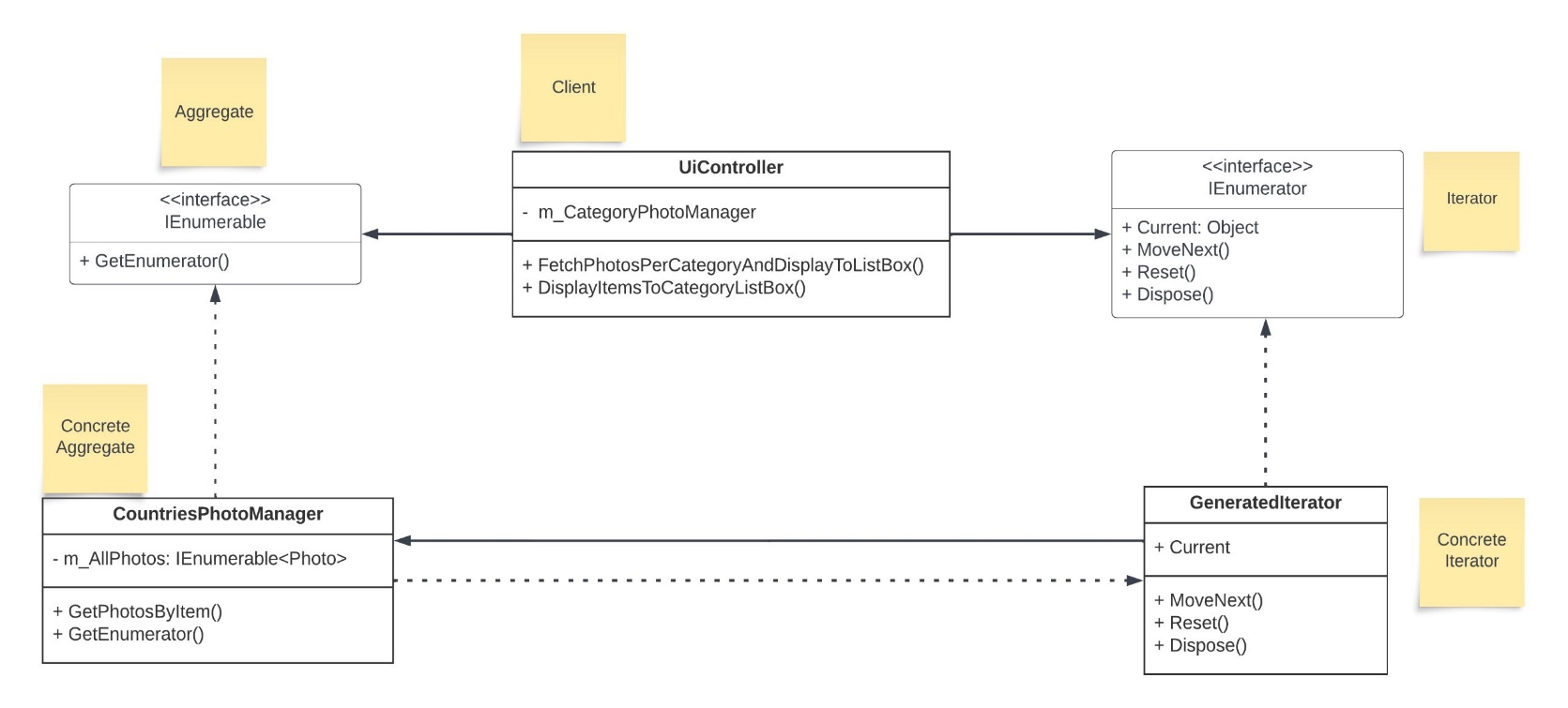
במטודה GetPhotosByItem השתמשנו בפיצר של C# 3 באמצעות שימוש ב foreach עם yield בפנים.

בעקבות כך, הקומפיילר דואג לייצר לנו מחלקות שיממשו את IEnumerable ואת IEnumerator

המטודות העיקריות מהממשקים הנ"ל הם: MoveNext, Reset, Current שבאמצעותן ניתן לעבור על איברי מבנה הנתונים באמצעות foreach מבלי לדעת את סוג מבנה הנתונים.

חשוב לציין כי הוספנו תנאי אשר מחליט את הItem הבא בכל פעם שיקראו למטודה MoveNext]

* Sequence Diagra
* Class Diagram



### תבנית מס' 2 – [Strategy]

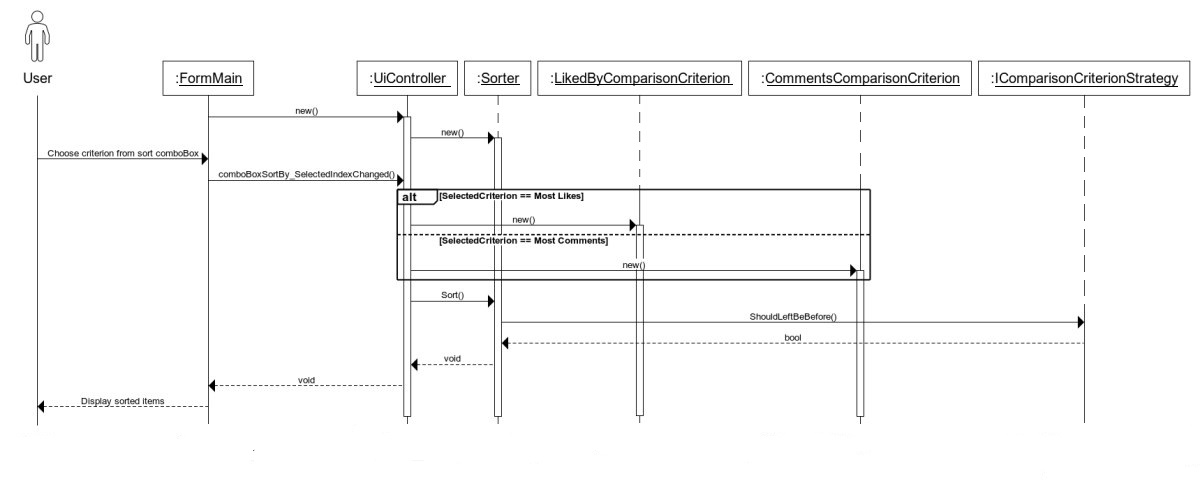
* **סיבת הבחירה / שימוש בתבנית:**

[עבור פיצ'ר שיכולות להיות לו כמה אפשרויות שונות של אלגוריתם (לפי בחירת המשתמש), למשל בפיצ'ר מיון התמונות לפי קריטריון אותו בחר המשתמש מבין הקריטריונים המוצעים לו, עבור כל קריטריון מיון יש לוגיקת אלגוריתם (אסטרטגיה) שונה, לכן נרצה להפריד בין לוגיקת האלגוריתם לבין ה context שמשתמש באלגוריתם ובכך הקוד יהיה נקי יותר וקל יותר לתחזוקה. כך שאם בעתיד נרצה להוסיף (או להסיר או לשפר) קריטריון נוסף למיון התמונות ולו אלגוריתם שונה (אסטרטגיה חדשה) כל שנצטרך לעשות זה ליצור עבורו מחלקה המממשת את המתודה אותה מממשים כל אחת מהאסטרטגיות האחרות, שגם הם נמצאות במחלקות משלהם, לפי הלוגיקה שלו. ולפי בחירת המשתמש עבור קריטריון המיון תופעל האסטרטגיה המתאימה, ללא כל צורך לשנות את מחלקת ה context. כלומר ניתן להציג אלגוריתמים חדשים עבור אופן המיון מבלי לשנות את הקוד הקיים.]

* **אופן המימוש:**

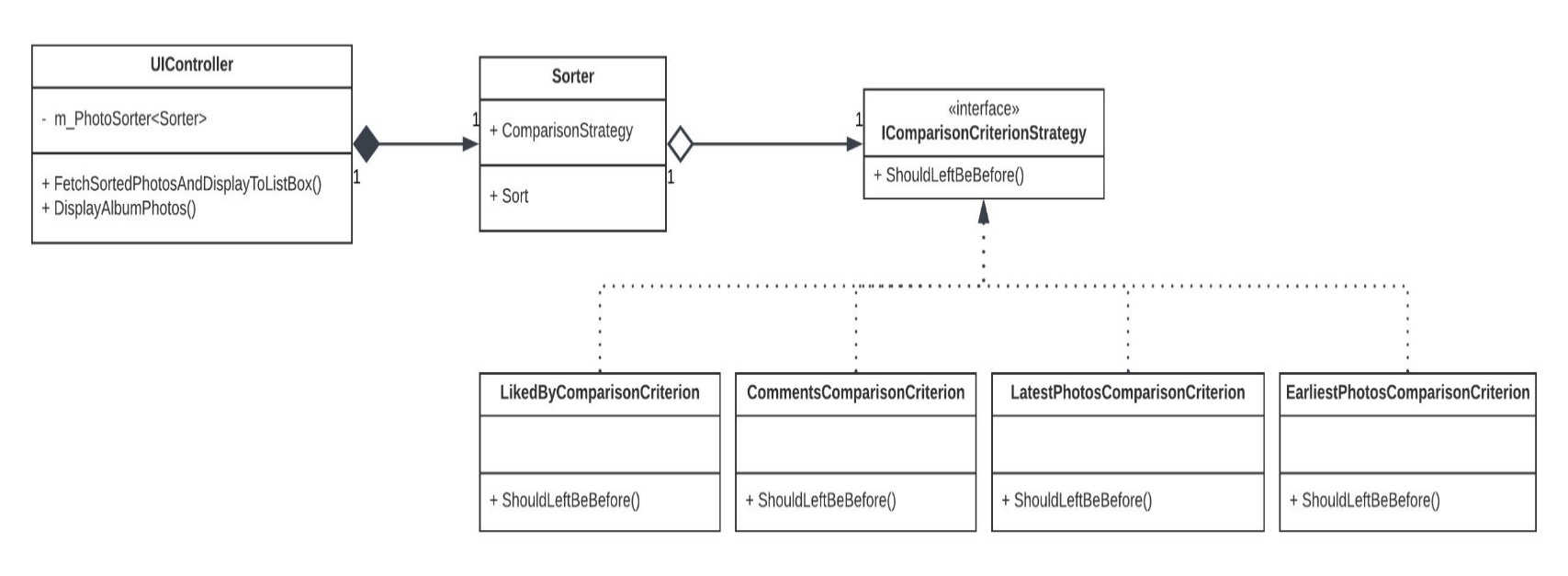
[עבור הפיצ'ר המציג את אלבומי התמונות- עבור הצגת האלבומים הוספנו combo box ובו המשתמש יכול לבחור כיצד למיין את התמונות המתקבלות מכל אלבום, האפשרויות הן: לפי כמות הלייקים, לפי כמות התגובות, לפי התמונה החדשה ביותר לישנה או להפך. לכל אחד מקריטריוני המיון הנ"ל לוגיקת אלגוריתם שונה ( אסטרטגיה שונה) לכן עבור כל אחד מהם יצרנו מחלקה נפרדת (concrete strategy) והן LikedByComparisonCriterion, CommentComparisonCriterion, LatestPhotosComparisonCriterion, EarliestPhotosComparisonCriterion המממשות את המתודה בממשק IComparisonCriterionStrategy

ברגע שהמשתמש בוחר את הקטגוריה לפיה הוא רוצה למיין נוצרת אסטרטגיה מהסוג שנבחר (new()) של האובייקט המייצג את הממשק וכך במחלקת ה context (Sorter) אופן המיון תתבצע לפי הלוגיקה המתאימה. ]

* Sequence Diagram

\*הערה: לצורך קריאות הדיאגרמה לעיל הצגנו רק שנים מאסטרטגיות, שכן שאר שאסטרטגיות יבואו לידי ביטוי בדיאגרמה באותו אופן כמו אלה שמוצגות.

* Class Diagram



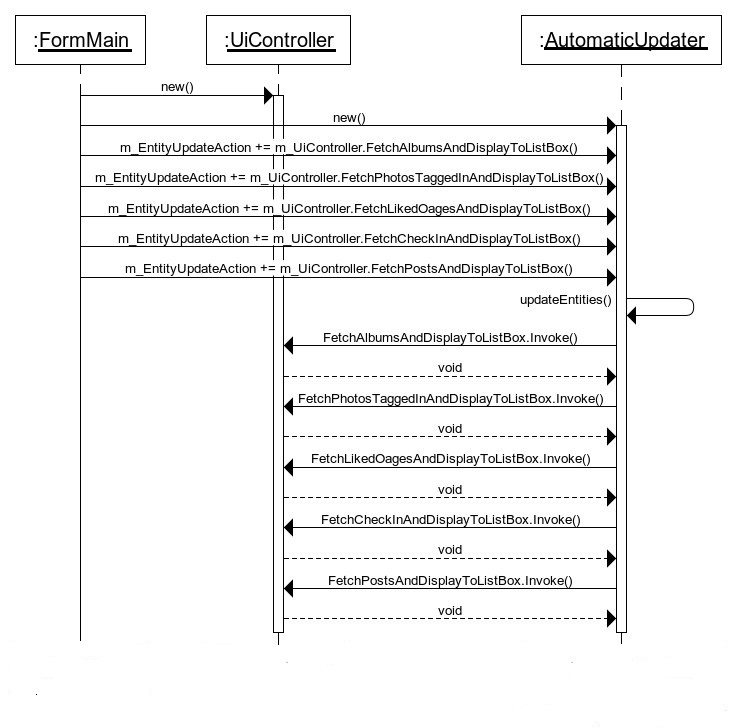
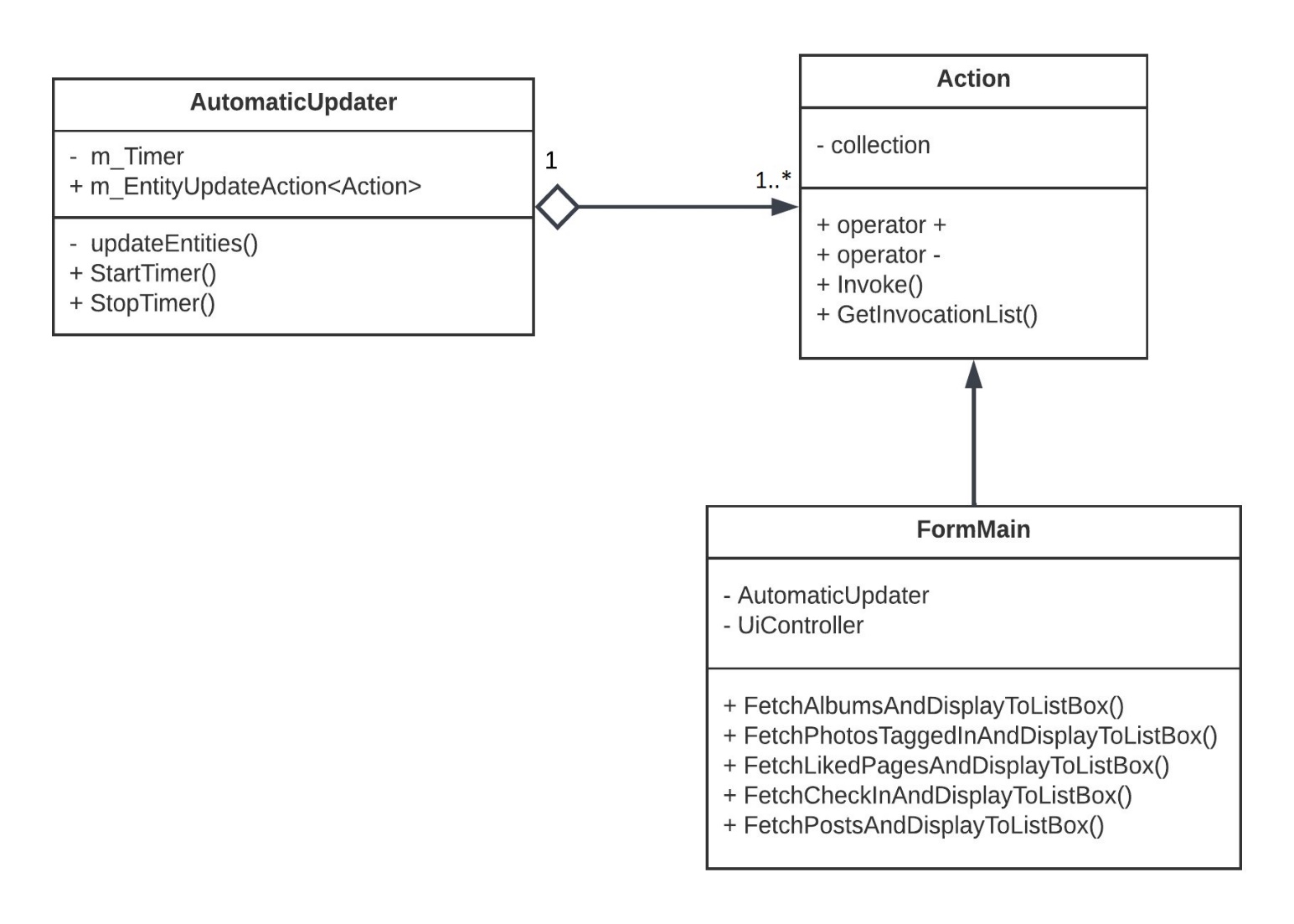
### תבנית מס' 3 – [Observer]

* סיבת הבחירה / שימוש בתבנית:

[עבור הפיצר שבו אנו מעדכנים את הנתונים מהשרת של פייסבוק כל זמן מסויים, בחרנו להשתמש בתבנית הנ"ל כדי שאם בעתיד נרצה להוסיף עוד פעולות שיבוצעו מאחורי הקלעים כל זמן מסויים, (כמו עדכון של נתונים חדשים) נוכל לבצע זאת בקלות ע"י הוספת Action חדש לDelegate של המחלקה המנהלת את הטיימר. כלומר, הקוד יהיה קל יותר לתחזוקה. כמו כן בדרך זו רק מי שנרשם לקבל עדכון על אירוע מסויים שקרה, במקרה זה הזמן שבחר המשתמש לעדכון מידע באפליקציה, הוא ורק הוא יקבל את העדכון ויפעל בהתאם ולא מטודות אחרות שאינן מעוניינות בכך.]

* אופן המימוש:

[יצרנו מחלקה חדשה AutomaticUpdateer שמטרתה לנהל את הפעולות שרוצות להרשם לעדכונים האוטומטים. כלומר, ניתן להוסיף פעולה באמצעות הDelegate שנמצא במחלקה בשם m\_EntityUpdateAction ולאחר X זמן יתבצעו הפעולות בחוטים נפרדים ע"מ שהמערכת לא תתקע בזמן פעולות העדכון]

* Sequence Diagram
* Class Diagram