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

* **Calendar:**  
  מטרת הפיצ'ר: ריכוז נתונים של תאריך בלוח שנה והצגה מסודרת למשתמש, עם יכולת ניהול נתונים.   
  יומן המציג:  
  > ימי הולדת  
  > אירועים  
  > רשימת משימות  
  לפי תאריך שנבחר על ידי המשתמש.  
  המשתמש בוחר תאריך מתוך לוח שנה שמוצג לו. כאשר הוא לוחץ על התאריך המבוקש מוצגים בפניו כל הנתונים שציינו. המשתמש יכול לנהל את רשימת המשימות שלו ע"י:  
  א. הוספת משימות חדשות  
  ב. סימון וי על משימות שבוצעו, סימון זה יוביל למחיקת המשימה מן הרשימה.    
  כל המשימות שהמשתמש מוסיף ליומן נשמרות באפליקציה. סגירת האפליקציה וכניסה חוזרת תציג את כל המשימות של המשתמש.  
  הקוד של הפיצ'ר נמצא במחלקות: Calendar.cs ו- CalendarForm.cs .
* **Style your Facebook page:**  
  מטרת הפיצ'ר: כיום חשיבות רבה לנראות של עמוד חברתי. מספר העוקבים של עמוד מושפע רבות מהנראות והעיצוב שלו. ככל שעמוד יהיה יותר מושך לעין, מעניין וייחודי כך הסיכוי שמספר העוקבים שלו יגדל.  
  הפיצ'ר כולל מספר הגדרות שמטרתן שיפור נראות העמוד ומתן פן ייחודי לבעל העמוד.  
  ההגדרות הן:  
  > שינוי צבע העמוד  
  > שינוי צבע הכפתורים בעמוד  
  > שינוי צבע שם המשתמש    
  > הוספת מוטו אישי   
  > בחירת תמונות מועדפות של המשתמש שמייצגות אותו באופן הטוב ביותר, התמונות יוצגו בעמוד של המשתמש כאשר כל כמה שניות תמונה מתחלפת. המשתמש יכול לשנות בכל עת את התמונות, להוסיף תמונות או להסיר תמונות.  
  כל ההגדרות הללו נשמרות באפליקציה. סגירת האפליקציה וכניסה חוזרת תציג את כל ההגדרות האחרונות שבוצעו.

הקוד של הפיצ'ר נמצא במחלקות: SwappingPhotosBox.cs , MainForm ו- ImageGalleryAlbum

### תבנית מס' 1 – [Observer]

* סיבת הבחירה / שימוש בתבנית:

ב-MainForm מוצגת רשימה של המשימות שעל המשתמש לבצע היום. על-מנת שהמידע ב-MainForm והמידע ב-CalendarForm יהיה מסונכרן, נרצה שבכל פעם שנוספת/מוסרת משימה מלוח השנה ב-calendarForm, בנוסף לכך שהוא יבצע עדכון ב-calendar הלוגי, הוא יפעיל את המתודה הרלוונטית ב-MainForm עבור הוספה/הסרה של משימות ל-listBoxTodayTasks. את הבדיקה האם מדובר בתאריך של היום מבצע ה-MainForm כחלק מהמתודה אותה ביקש לצרף לרשימת המאזינים של שני האירועים. זאת מאחר ובמידה ובעתיד רכיבים נוספים יירצו להתעדכן אודות הוספת/הסרת משימה, כל שעליהם יהיה לעשות הוא לצרף לרשימת המאזינים הרלוונטית את המתודה שיירצו שתופעל. למשל, אם בעתיד נרצה להציג את המשימות של השבוע הקרוב, בכל פעם שהמתודה תופעל בעקבות הוספת/הסרת משימה ל-listBox ב-CalendarForm, נבדוק האם התאריך של המשימה נמצא בטווח של השבוע הקרוב ואם כן- נוסיף/ נסיר.

* אופן המימוש:
* את הקוד הרלוונטי לתבנית ניתן למצוא במחלקות: CalendarForm ו-MainForm.
* מדובר במימוש שהוא method-oriented ולא object-oriented.

ה-CalendarForm מחזיק שני מופעים מסוג Action<Task> - האחד עבור משימה שנוספה ללוח השנה (TaskAdded) והשני עבור משימה שנמחקה מלוח השנה (TaskRemoved).

ה-access modifier של מופעים אלה הינו Public event שכן, נרצה לאפשר למאזינים הצטרפות והסרה לרשימת המאזינים אך לא נרצה לאפשר להם גישה להפעלת invoke.

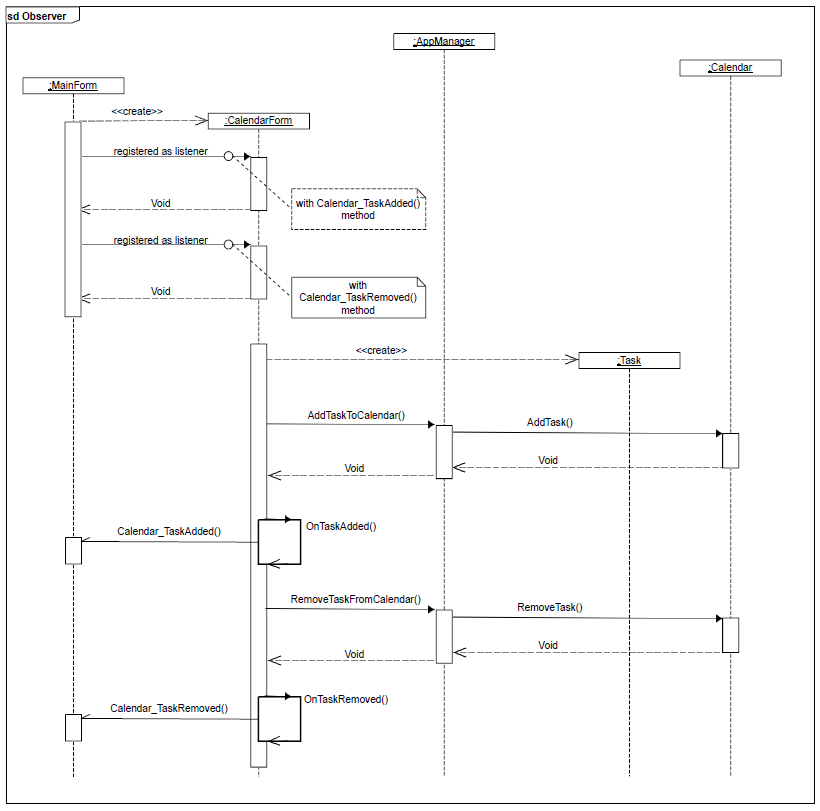
ה- MainFormנרשם כמאזין לשני אירועים אלה באמצעות += והמתודה הרלוונטית (לא כ-node מסוג Action).

* הוספת משימה - כאשר המשתמש מוסיף משימה ב-CalendarForm (באמצעות לחיצה על כפתור Add new task) – המשימה מתווספת ללוח השנה וכן, מתבצע TaskAdded.invoke המפעיל את המתודות השונות של המאזינים. במקרה שלנו, ה-MainForm. במתודה שמופעלת, המשימה עצמה מתקבלת כפרמטר ומתבצעת בדיקה האם המשימה שנוספה היא לתאריך של היום ואם כן, היא מתווספת ל-listBox המציג את המשימות של היום כאשר נשמר סדר המיון שנקבע להצגת הרשימה.
* הסרת משימה- כאשר המשתמש עובר לתאריך אחר בלוח השנה שב- CalendarForm או לחלופין סוגר את הטופס, רגע לפני כן, כל המשימות שהינן checked נמחקות ומתבצע TaskRemoved.invoke עבור כל משימה שנמחקת ומפעיל את המתודה הרלוונטית ב-MainForm. במתודה שמופעלת, המשימה עצמה מתקבלת כפרמטר ומתבצעת בדיקה האם המשימה שהוסרה הייתה בתאריך של היום ואם כן, היא מוסרת מה-listBox המציג את המשימות של היום.

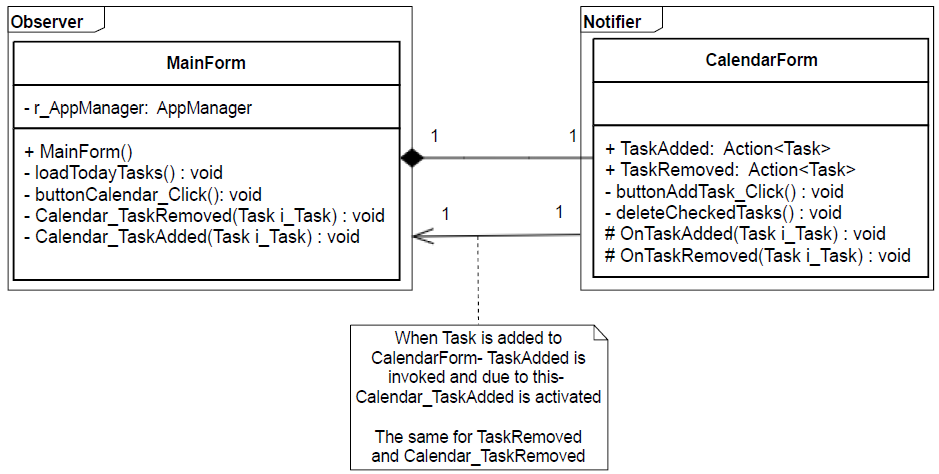
Notifier- CalendarForm

Observer\ listener - MainForm

* Sequence Diagram



* Class Diagram



### תבנית מס' 2 – [Iterator]

* סיבת הבחירה / שימוש בתבנית:

בחרנו לממש את pattern iterator על מנת להבטיח שהאופן בו ה-client סורק את האיברים ב-Aggregate יהיה באופן מנוהל. כלומר, שה-client יוכל לבצע סריקה על אוסף האיברים ב- Aggregate מבלי לדעת באיזה מבנה נתונים האיברים נשמרים.   
היתרון במימוש זה הוא שאם בעתיד נחליט לשנות את מבנה הנתונים בו האיברים נשמרים, ה-client לא יהיה מודע לשינוי במבנה הנתונים והוא לא יצטרך לבצע שינוי בקוד שלו בעקבות זאת.

* אופן המימוש:

1. Client - AppManager
2. Aggregate - IEnumerable<T>
3. Concrete Aggregate – Calendar
4. Iterator - IEnumerator<T>
5. Concrete Iterator – IEnumerator<Task>

בחרנו לממש את configurable filter iterator שמשתמש ב-strategy expression. כאשר מחלקת AppManager מתפקדת כ-client ומחלקת Calendar מתפקדת כ-Aggregate. האיברים אותם ה-client מעוניין לסרוק הם רשימת ה-tasks הרלוונטיים לאותו היום במחלקת Calendar.   
מחלקת Calendar מממשת את הממשק IEnumerable<T> כאשר T הוא אובייקט מסוג Task.  
ישנן שתי שתי פונקציות של GetEnumerator: המתודה GetEnumerator של IEnumerable ממומשת בצורה מפורשת, ואילו המתודה השנייה ממומשת בעזרת yield return בצורה לא מפורשת (המילה yield גורמת לקומפיילר לייצר מאחורי הקלעים את המחלקה עם חמשת הפונקציות שלמדנו).  
במחלקה AppManager רצנו באמצעות foreach על אוסף המשימות היומיות ב-Calendar, כאשר בפועל מתקיים הקוד הבא :

using (IEnumerator<Task> iterator = m\_Calendar.GetEnumerator())

{

while (iterator.MoveNext())

{

tasks.Add(iterator.Current);

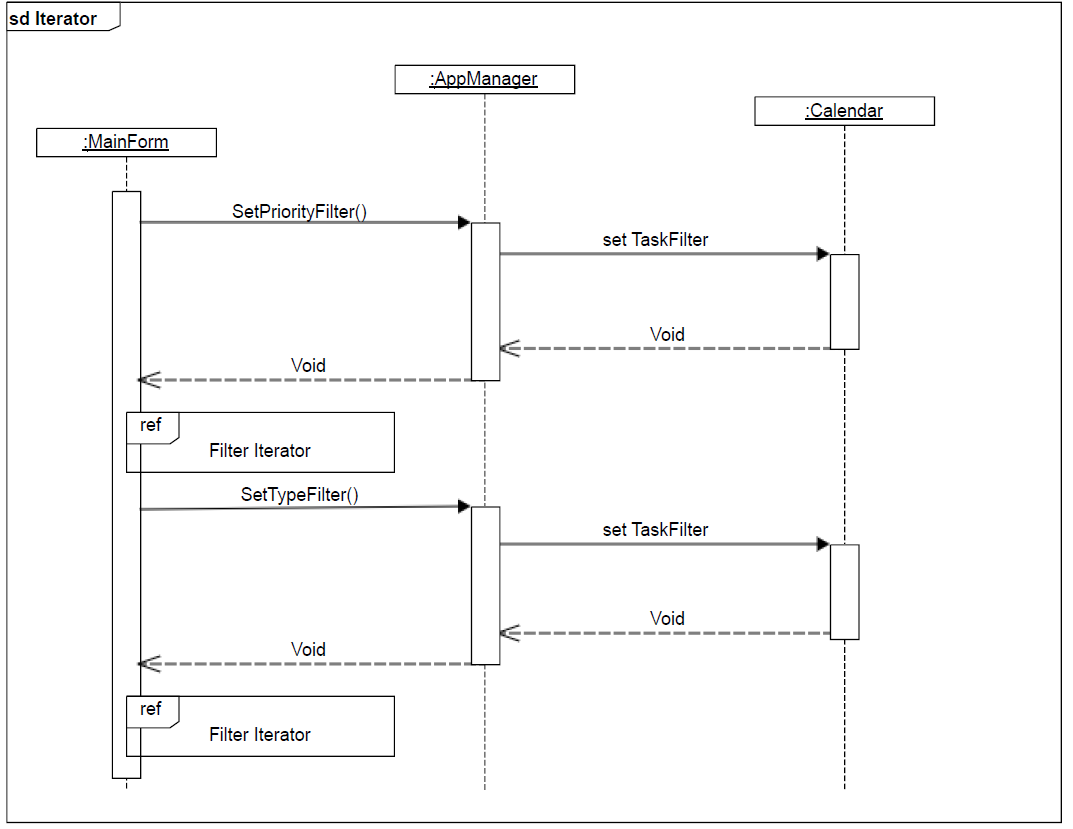
}

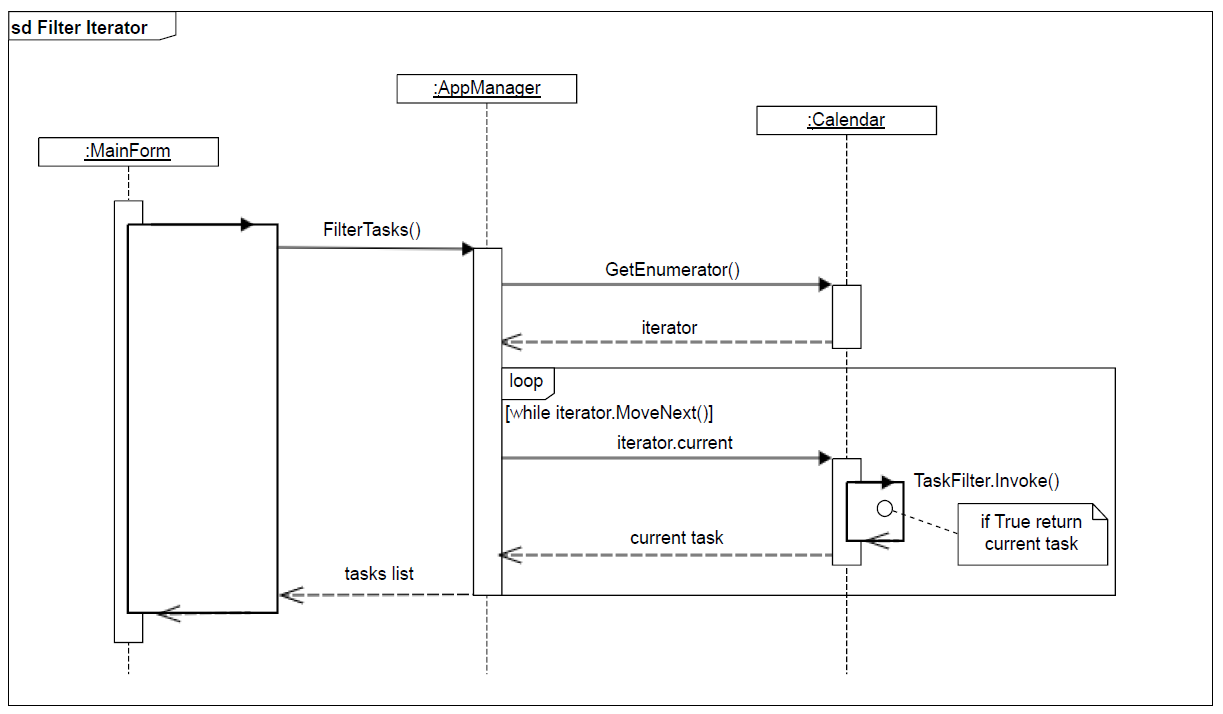
}

במחלקת Calendar המתודה GetEnumerator() ממומשת כך, שרק אם ה-task עבר את ה-filter, כלומר TaskFilter.Invoke החזיר True, אז ה-task יישלח ל-AppManager.   
שינוי והשמה של ה-filter נעשה באמצעות strategy expression

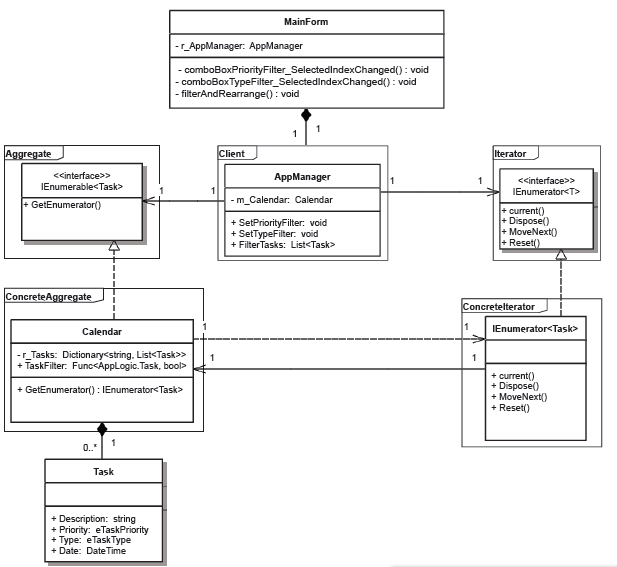
ניתן למצוא את המימוש במחלקה Calendar.cs , AppManager.cs

* Sequence Diagram





* Class Diagram



### תבנית מס' 3 – [Strategy]

* סיבת הבחירה / שימוש בתבנית:

ב-MainForm מוצגות למשתמש המשימות היומיות שעליו לבצע.

* אסטרטגיית מיון - כל משתמש עובד באופן אחר ומעוניין בדרך שונה להצגת המשימות– לאחד נוח שהמשימות מוצגות באופן מסודר ע"פ אורך מהקטן לגדול, לאחר מהגדול לקטן וכו'.

כלומר, זיהינו במנגנון המיון נקודה שייתכן ונרצה להחליף בעתיד (injection point). על-מנת לאפשר את החלפת אסטרטגיית המיון בעתיד, השתמשנו ב-pattern Strategy כאשר המחלקה Sorter מחזיקה מופע של Func ובכל פעם ממיינת לפי המתודה שמוחזקת שם. כלומר, כל המימוש נשאר זהה אך נקודה זו בלבד ניתנת להחלפה. בכדי להמחיש זאת, הראנו דוגמה ל-2 מיונים אפשריים.

כחלק ממימוש pattern Iterator, השתמשנו בשנית בתבנית Strategy כמתואר כדלהלן ( ההסבר המורחב והדיאגרמה נמצאים תחת pattern Iterator)

* אסטרטגיית סינון - ייתכן ולמשתמש זמן קצר לביצוע המשימות ולכן הוא מעוניין שיוצגו לו רק המשימות שנמצאות בעדיפות גבוהה. או לחלופין, המשתמש נמצא בעבודה ולכן המשימות שעליו לבצע בבית לא רלוונטיות עבורו כעת.

כלומר, אופן סינון הפריטים להצגה מתוך רשימת המשימות היומיות מהווה נקודה שייתכן ונרצה להחליף בעתיד. על-מנת לאפשר את החלפת אסטרטגיית הסינון בעתיד, השתמשנו ב-pattern Strategy כאשר המחלקה Calendar מחזיקה מופע של Func ובשילוב עם pattern Iterator מחזירה רק משימות המקיימות את תכונת הסינון המוגדרת כעת. בכדי להמחיש זאת, הראנו דוגמה ל-2 קטגוריות סינון אפשריות ולכל אחת 3 אופציות מתוכן ניתן לבחור.

* אופן המימוש:
* את הקוד הרלוונטי לתבנית ניתן למצוא במחלקות: Sorter, AppManger ו-MainForm.
* מדובר במימוש שהוא method-oriented ולא object-oriented. זאת מאחר ומדובר במתודה אחת פשוטה ולכן אין צורך ב-interface עבור כך.

מחלקת Sorter מחזיקה מופע של Func<int, int, bool> המהווה את פונק' ההשוואה בה תשתמש המחלקה בעת ביצוע המיון (comparerMethod).

מחלקה זו מקבלת ב-constructor מתודת השוואה ומשימה אותה לתוך comparerMethod, זאת ע"מ שלא יווצר מצב שלמחלקה אין מתודת השוואה לעבוד איתה.

מחלקת AppManger המהווה facade במערכת, מחזיקה במופע של Sorter הנקרא TaskSorter עבור מיון המשימות היומיות המוצגות ב-MainForm ובעת יצירת המופע היא שולחת מתודת השוואה עבור מיון בסדר עולה כ-default.

כאשר היוזר מעוניין לשנות את סדר הופעת המשימות ב-MainForm, הוא לוחץ על אופן המיון הרצוי וכתוצאה מכך ה-MainForm משים ל-comparerMethod של TaskSorter את מתודת המיון הרצויה ומיד לאחר מכן מבצע מיון על רשימת המשימות היומיות שהוא מחזיק ומציג את הרשימה לאחר המיון.

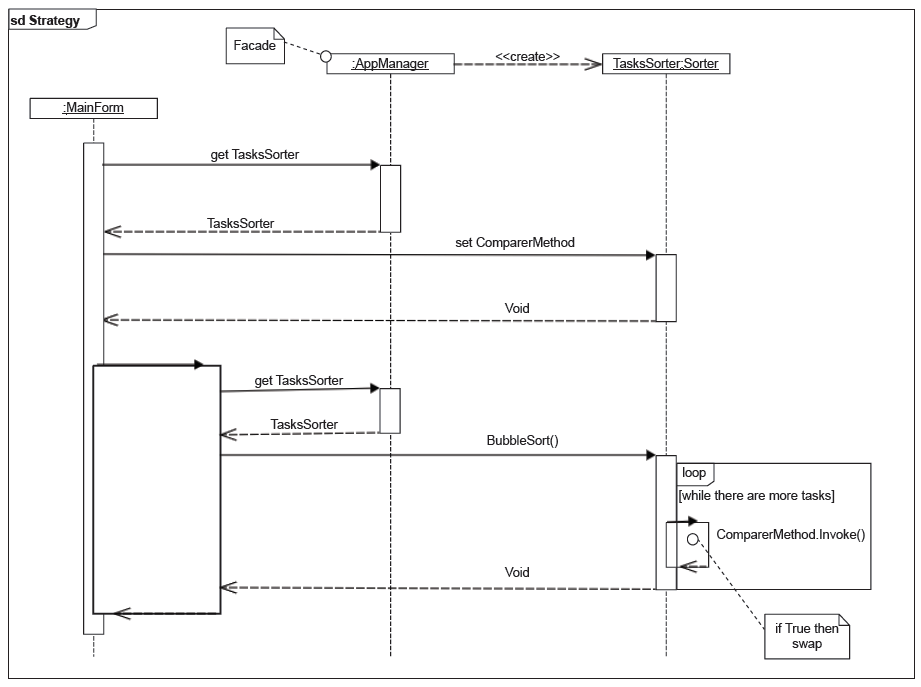
את ההשמות ל-comparerMethod ביצענו באמצעות lambda expression בשל היתרון שכך המתודה נמצאת רק במקום ובהקשר הרלוונטי לה.

Context – Sorter

Strategy- Method strategy - ComparerMethod

Client - MainForm

* Sequence Diagram



* Class Diagram

