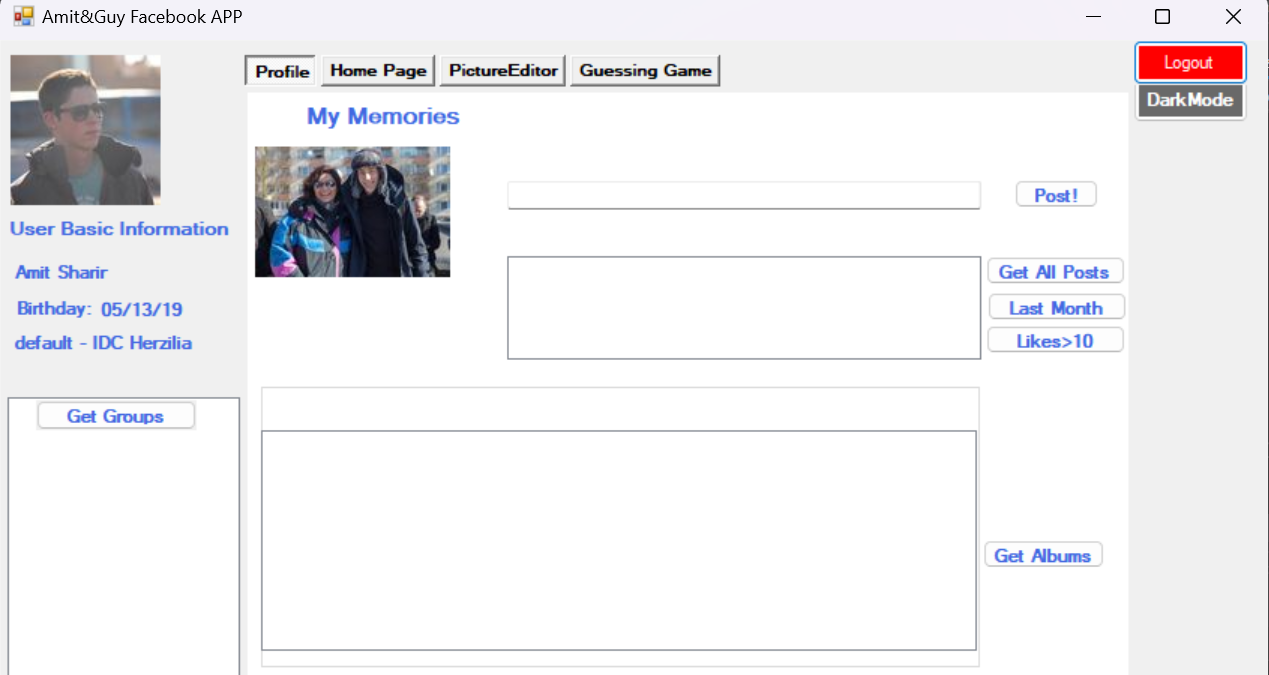
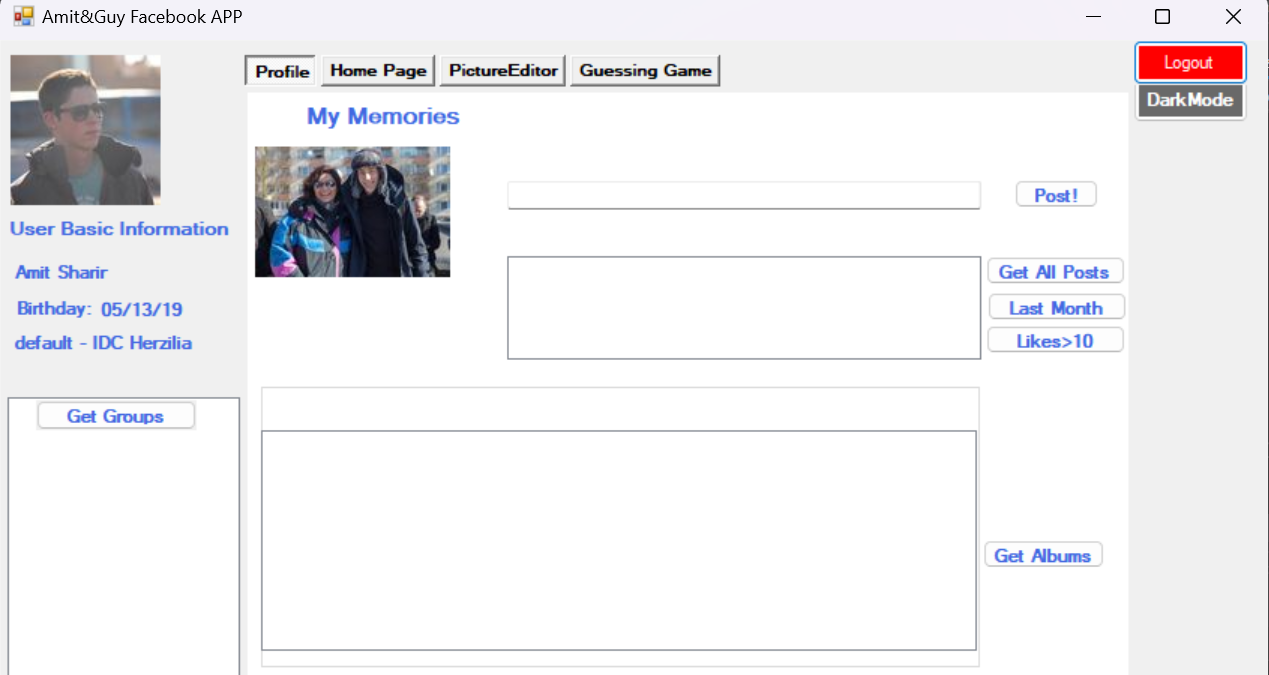
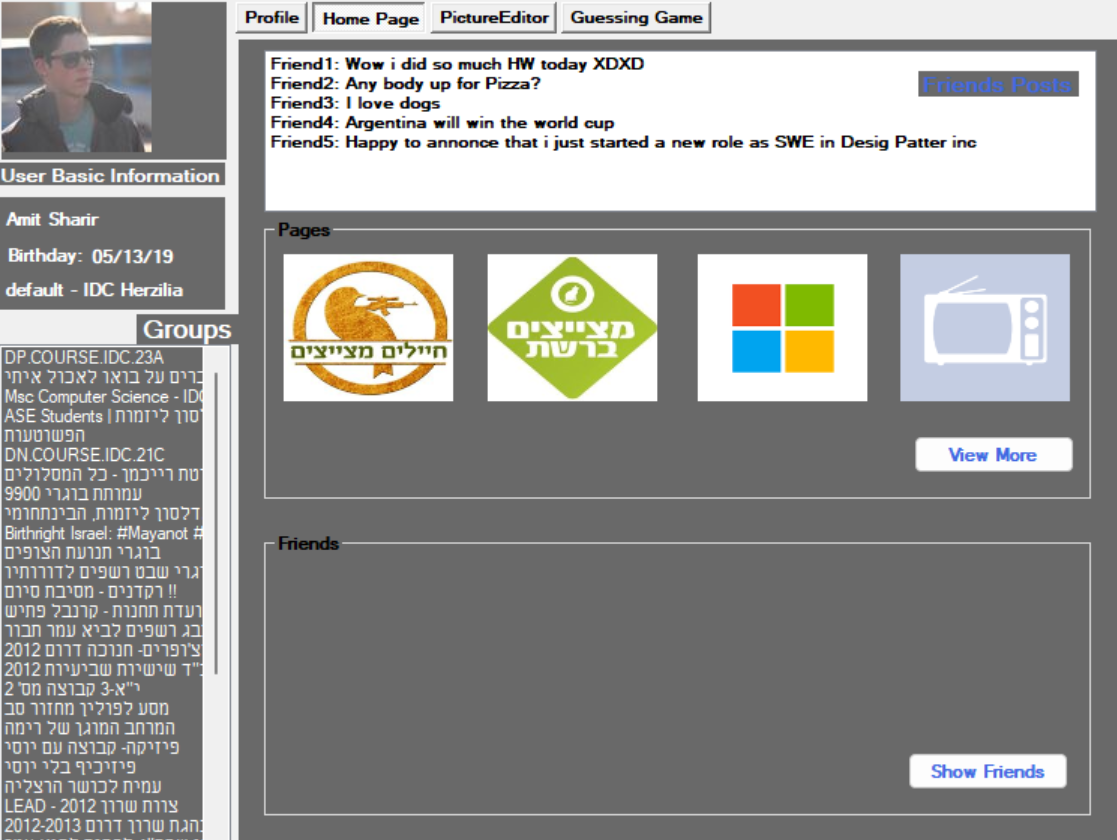
### תיאור קצר של הפיצ'רים שבחרנו לממש בתרגיל הקודם:

* **Toonify picture -** חלון זה מאפשר למשתמש להמיר את תמונת הפרופיל שלו באמצעות AI לתמונה שהיא Toonified, כלומר הפיצ'ר מאפשר למשתמש להמיר את תמונת הפרופיל שלו לתמונה המדמה דמות מצוירת והיא מציגה לו את שתי התמונות במקביל אחת ליד השנייה. לצורך מימוש הפיצ'ר הזה השתמשנו [בAPI של חברת Deep AI](https://deepai.org/machine-learning-model/toonify).
* **Guessing Game** - חלון זה מאפשר למשתמש לשחק משחק ניחושים אינטראקטיבי וכיפי המבוסס על חודש הלידה שלו. המערכת מקבלת את חודש הלידה של המשתמש מפרופיל הפייסבוק שלו ומגרילה למסך 4 אנשים מפורסמים אשר נולדו באותו חודש של המשתמש. המשתמש צריך לנחש איזה מהמפורסמים נולדו בחודש שלו - במידה וצדק נוספת לו נקודה ובמידה וטעה יורדת לו נקודה. רשימת המפורסמים אשר מוצגת למשתמש במשחק מתקבלת מData Base אינטרנטי שביצענו עליו Parsing (ניתן לראות את הData Base תחת תיקיית הResources של הפרויקט).
* **LeaderBoard** – הינו פיצ'ר שהוספנו במסגרת מטלה #2. הפיצ'ר מאפשר למשתמש לראות את טבלת הניקוד לאורך כל הזמנים של כלל המשתתפים שהתחברו דרך מחשב המשתמש ושיחקו בGuessing Game. מבחינת עיצוב המשחק והמערכת, התצוגה שמתאפשרת באמצעות הפיצ'ר החדש מוסיפה "Gamification" ומשפרת את חווית המשתמש בכך שהופכת את המחשק לתחרותי ואינטרקטיבי יותר. המערכת שומרת את תוצאות המשחק בין המשתמשים וההפעלות השונות על ידי שימוש בData Base ייעודי שיצרנו (אשר גם נמצא בתיקיית הresources), כך גם כאשר יוצאים מהמערכת ונכנסים מחדש (מריצים מחדש את התוכנית), המערכת תמיד שומרת את התוצאות של כלל המשתמשים.
* **FilteredPosts** – הינו פיצ'ר חדש שהוספנו במסגרת המטלה הנוכחית לטובת המימוש של Design Pattern Strategy עליו עוד נפרט בהמשך. הפיצ'ר מאפשר למשתמש לבחור איזה סוג של פוסטים הוא רוצה לראות במסך הבית שלו. נכון להיום קיימות 3 אופציות סינון שהוספנו, הצגה של כל הפוסטים שהיו אי פעם, הצגה של פוסטים מהחודש האחרון בלבד והצגה של של פוסטים שזכו ליותר מ10 לייקים. בזכות הפיצ'ר החדש חווית המשתמש משתפרת שכן הוא זוכה לקבל לוגיקה משופרת ששירותי פייסבוק הקלאסיים אינם מציעים לו. המשתמש יכול כעת לקבל מידע מעובד על פי בחירה.



* **DarkMode** – הינו פיצ'ר חדש שהוספנו במסגרת המטלה הנוכחית לטובת המימוש של Observer עליו עוד נפרט בהמשך. הפיצ'ר מאפשר למשתמש לשנות את הרקע של כלל החלונות של המערכת שלנו לגוון כהה. בשנים האחרונות ניתן לראות שימוש הולך וגובר של Dark mode בקרב אפליקציות. לפי [מחקר](https://medium.com/geekculture/4-reasons-to-adopt-the-dark-mode-in-your-app-b235632c550c) שראינו, 83% מהאנשים ציינו כי הם משתמשים בפיצ'ר זה במערכות המחשב שלהם, בנוסף קיימת ציפייה בקרב 65% מהאוכלוסיה שפיצ'ר זה יהיה קיים במערכות מחשב. זאת ועוד, פיצ'ר זה יכול גם לסייע לאנשים בעלי לקות ראייה ולשפר את הAccessabilty של המערכת שלנו. לאור כל זאת, הפיצ'ר החדש משפר את חווית המשתמש.





### תבנית מס' 1 – [Stratagy]

* סיבת הבחירה / שימוש בתבנית:

כחלק מהפיצ'ר FilteredPost החדש שהוספנו במסגרת מטלה זו, אפשרנו לקליינט לבחור איזה פוסטים הוא רוצה לראות מתוך כלל הפוסטים שהיו לו עד כה בפייסבוק. כפי שציינו לעיל, קיימות מספר לוגיקות הצגה (הצגה של כל הפוסטים שהיו אי פעם, הצגה של פוסטים מהחודש האחרון בלבד והצגה של של פוסטים שזכו ליותר מ10 לייקים), אשר המשתמש יכול לבחור מביניהם ולפי כך יוצגו לו הפוסטים הרלוונטיים. בתור מפתחי המערכת אנו שואפים להכניס עוד אפשרויות סינון במהלך המשך הפיתוח של המערכת.

בחרנו למממש את הפיצ'ר החדש לפי Design Pattern Startagy כדי שאם בעתיד נרצה להוסיף לוגיקות סינון חדשות, נוכל לעשות זאת בקלות באופן כזה שכלל השינויים יהיו רק במחלקה ספיציפית. באופן הזה אנו משפרים את התחזוקיות של המערכת, נוכל להימנע משכפול קוד ובמילים אחרות אנו מאפשרים הוספת פיצ'רים בקלות בעתיד עם מינימום שינויים במחלקות Legacy. עוד יתרון לשימוש בStratagy הוא שניתן להוסיף\לשנות אסטרגיות סינון חדשות במהלך הריצה של התוכנית ולהשתמש בהם בזמן אמת.

בין היתר, מימוש של Design Pattern זה מאפשר Loose Coupling בין המחלקות שמממשות את הלוגיקה ואלה שמשתמשות בה ובכך משפר את הקוד שלנו מבחינת גמישות והופך את הקוד ליותר Maintainable. אלגורתמי הFilter שהוספנו הם Encapsulated ומופרדים למחלקות אחרות, מה שיאפשר לנו בעתיד לתקן ולשנות את הלוגיקה שלהם במידה וידרש, וגם יקל עלינו להשתמש באותם אלגורתמים במקומות אחרים במידה ונצטרך (Reusability).

* אופן המימוש:

1. לטובת המימוש יצרנו מספר מחלקות אשר יושבות בחלק הלוגי של המערכת.
2. יצרנו ממשק חדש בשם IFilterStratagy אם מתודה בשם Select.
3. יצרנו מחלקה חדשה בשם FilterStratagy אשר מחזיקה בקומפוזציה מופע של IFilterStratagy .
4. יצרנו 3 מחלקות אשר מממשות את הממקש IFilterStratagy, כל אחת עם לוגיקת סינון אחרת.
   * ShowAllFilter
   * MoreThanTenLikesFilter
   * LastMonthFilter
5. אנו יוצרים מופע חדש של אובייקט מסוג FilterStratagy בFormMain כאשר את סוג הסינון עצמו שהוא מחזיק בקומפוזיציה אנו בוחרים בעזרת בחירת המשתמש.
6. אנו שולחים את הFilterStrategy שיצרנו למתודת fetchFilteredPosts אשר לפי האסטרטגיה הנבחרת מציגה את רשימת הפוסטים המסוננים.
7. חשוב לציין שMoreThanTenLikesFilter וLastMonthFilter מממשות את פונקציית Select באמצעות LINQ כפי שלמדנו בכיתה.
8. לLINQ יתרונות רבים כמו לדוגמא: שיפורי ביצועיים ביחס לגישות המשתמשות בלולאות וריקורסיה, קריאות משופרת (כפי שדיברנו בשיעור 13 בכיתה, עקב סדר הפעולות שרשום משמאל לימין),קוד יותר Maintainable ועוד.

* Sequence Diagram

[sequence diagram שמציגה את התהליך שקשור לתבנית והאינטראקציה בין המחלקות]

* Class Diagram

[class diagram שמתארת את המחלקות שלכם שמעורבות בתבנית (תיאור מלא שכולל Properties ו- Methods) והיחסים ביניהם (תאור מלא עבור כל יחס כפי שלמדנו בכיתה). עבור כל מחלקה שלכם, כיתבו מי המקבילה שלה (אם יש כזו) בתבנית (Pattern) שבחרתם

### תבנית מס' 2 – [Iterator]

* סיבת הבחירה / שימוש בתבנית:

במסגרת פיצ'ר הGuessing Game שקיימת במערכת שלנו, אנו עובדים על Data Base שמכיל פרטים על אנשים מפורסמים ואנו נדרשים לבצע Parsing שלו מקובץ טקסט למבנה נתונים שקל ונוח לעבוד איתו. בחרנו להשתמש בתבנית זו כדי שיהיה קל להשתמש (במקרה שלנו לעבור\לבצע איטרציה על רשימת המפורסמים) במידע שיושב בData Base במסגרת מחלקת GuessingGame - אשר מנהלת את הלוגיקה של המשחק שיצרנו. המימוש מאפשר למחלקת GuessingGame לסרוק את המידע בצורה נוחה ללא התחשבות במבנה הנתונים הספיציפי שבו מוחזק המידע של האנשים המפורסמים (שקיבלנו באמצעות Parsing של הData Base). אופן מימוש זה מאפשר לנו לבצע בצורה קלה שינויים בData Base עצמו ובאופן בו אנו מחזיקים את המידע שיושב בו מבחינת מבנה הנתונים. אם בעתיד נרצה לשנות את הדאטא בייס ו\או לשנות את מבנה הנתונים ששומר את המידע שלו נוכל לעשות זאת בקלות ללא שינויים בקוד שיחשב כComponent Legacy שיושב תחת מחלקת הGuessingGame.כלומר אם בעתיד נרצה לעבוד עם קולקציה אבסטרקטית אחרת נוכל לעשות זאת בקלות– הקוד יותר Maintainable.

בנוסף לכך, מימוש Iterator Design Pattern שעשינו מאפשר לנו לבצע פעולות LINQ, אשר מאפשרות לנו בקלות ובנוחות לבצע פעולות לוגיות על מבנה הנתונים (וגם משפרות את הקריאות של הקוד שלנו).

חשוב לציין, שאם בעתיד נרצה לשנות את המשחק, לשלוף מידע אחר מהData Base ולהציג משחק אחר, נוכל לבצע את השינויים בקלות יחסית על ידי שינויים שנבצע במחלקה האינומרבילית שהוספנו כלומר המערכת יותר Flexible.

* אופן המימוש:

1. ראשית יצרנו מחלקה בשם FamousPerson שמטרתה להחזיק את פרטי המידע שאנו רוצים להציג על כל אדם מפורסם.
2. שנית יצרנו את מחלקת EnumerableFamousPersons אשר מממשת את מחלקת IEnumerable<T>.
3. נשים לב כי במקרה שלנו, האיבר הגנרי <T> של IEnumerable הוא FamousPerson.
4. בנוסף למטודות רגילות שהוספנו למחלקה ( Init, AddFamousPerson) הוספנו למחלקה שתי מטודות אשר הן בעצם המימוש של IEnumerable – GetEnumerator אשר מחזיר IEnumerator<FamousPerson> וגם מטודה IEnumerable.GetEnumerator() אשר נדרשת לטובת מימוש מלא של הממשק, IEnumerable במקרה שלנו היא מבצעת קריאה לפונקציה הראשונה שהזכרנו (GetEnumerator).
5. חשוב לציין שבפונקציית GetEnumerator() עשינו שימוש במתודלוגיית yield return אשר נלמדה בכיתה.
6. מתודלוגייה זו מאפשרת לנו לייצר iterator בקלות שכן yield return מחזיר לנו אובייקט מסוג IEnumerator<T> והוא כבר מממומש כחלק מC#.
7. קיים גם שיפור ביצועים מסוים בכך שבאופן הזה אנו לא מחזירים את כל הקולקציה במידה ונדרש רק איבר מסוים (בכל פעם נחזיר איבר אחד).
8. אנו עושים שימוש בעובדה ש EnumerableFamousPersons מממש את IEnumerable<FamousPerson> במחלקת GuessingGame. כפי שניתן לראות שם אנו מבצעים שימוש בLINQ בתוך מטודת chooseRandomCelebrityBornInMonth אשר באמצעותה אנו שולפים רנדומלית אנשים מפורסמים מהקולקציה.
9. הסיבה שאנו יכולים להשתמש בLINQ על הקולקציה הוא בזכות זה שמימשנו את הממשק IEnumarable.
10. לLINQ יתרונות רבים כמו שציינו בסעיף 8 תחת ההסבר על הStratagy.

* Sequence Diagram

[sequence diagram שמציגה את התהליך שקשור לתבנית והאינטראקציה בין המחלקות]

* Class Diagram

[class diagram שמתארת את המחלקות שלכם שמעורבות בתבנית (תיאור מלא שכולל Properties ו- Methods) והיחסים ביניהם (תאור מלא עבור כל יחס כפי שלמדנו בכיתה). עבור כל מחלקה שלכם, כיתבו מי המקבילה שלה (אם יש כזו) בתבנית (Pattern) שבחרתם

### תבנית מס' 3 – [Observer]

* סיבת הבחירה / שימוש בתבנית:

בחרנו להשתמש בתבנית זו על מנת לעדכן את כל הפקדים באפליקציה במצב של רצון מצד המשתמש להציג את האפליקציה במצב “Dark-Mode”, מימוש בצורה זו מאפשר לObserver לקבל הודעות על אירועים שקרו, ולSubject להודיע על שינויים שקרו אצלו, זאת מבלי ליצור יחס תלות בין מחלקות. מחלקת Dark-Mode מודיעה כאשר יש דרישה של המשתמש להציג את האפליקציה במצב כהה, היא אינה יודעת למי היא מודיעה ומה מתבצע עם ההודעה. מחלקת FormMain מקבלת את ההודעה בצורה אוטומטית (מכיוון שהיא רשומה לאירוע m\_ReportClickedDelegates) ומבצעת פעולות באופן עצמאי בהתאם להודעה. מימוש זה מאפשר לנו להודיע בעתיד על בקשה מצד המשתמש להציג את הפקדים בצורה כהה, לכל גורם המתעניין במידע, ומאפשר לכל גורם מתעניין לעשות עם המידע פעולות אשר מוגדרות באופן פרטני עבורו.

כלומר יצרנו אי- תלות בין המחלקות של הobservers ולבין מחלקת הsubject אשר מודיעה על שינויים במצבה- במקרה שלנו DarkMode, FormMain לא תלויות אחת בשנייה, מחלקת FormMain יכולה לקבל הודעות מDarkMode מבלי לדעת איך בפועל אירועים/שינויים נוצרו ומומשו, בנוסף היא יכולה גם להירשם כמאזינה למחלקות אחרות (ויכולה לעשות עם המידע מגוון פעולות). מהצד השני לDarkMode יש אפשרות להודיע על אירועים או שינויים למחלקות אחרות אשר מתעניינות בידע.

בנוסף, מבחינת גמישות, התבנית מאפשרת לנו להוסיף ולהוריד בצורה דינמית מאזינים, דבר זה מאפשר לקוד שלנו להתרחב בקלות ולהוסיף עוד מחלקות בעתיד שיוכלות להירשם כמאזינות (extensible). כלומר כל מחלקה אחרת יכולה להרשם להודעות של המחלקה, ולהשתמש בהודעה כרוצנה מבלי לשנות את הקוד של DarkMode.

* אופן המימוש:

1. ראשית מימשנו מחלקה בשם DarkMode שמהווה את מחלקת הSubject בתבנית.
2. למחלקה אובייקט "כפתור", m\_DarkModeButton שממומש ידנית ולא דרך המימוש האוטומטי של WinForms.
3. בנוסף למחלקה יש אירוע (event) שנקרה m\_ReportClickedDelegates שלמעשה ניתן להרשם אליו בצורה חופשית לכל מי שרוצה להפוך למאזין להודעות.
4. ישנו שדה בוליאני שמתאר את המצב של המערכת "כהה" או "בהיר" (m\_Enabled). ועוד שדה string ששומר על סטטוס המערכת (m\_Status).
5. לכפתור יש event handler כך שבעת לחיצה על הכפתור, הכפתור מפעיל מתודה שמבצעת שינויים בנראות הכפתור (changeEnabledStatus)ומתודה שאחראית על פעולה ההודעה לכל המאזינים(notifyClickObservers).
6. מימשנו את פעולת ההודעה(בתוך מטודה notifyClickObservers) ע"י null check באמצעות "?" ושימוש בinvoke. ה"?" מאפשר שימוש ב m\_ReportClickedDelegates רק במידה וקיים, אם לא הוא יוצר חדש.
7. במחלקת FormMain יש מופע של מחלקת DarkMode (החלון המרכזי מוסיף את הכפתור לרשימת הcontrols שלו). ובנוסף מתבצעת הרשמה לm\_ReportClickedChange ושיוכה למתודה "DisplayModeChange" שתופעל בעת הודעת שינוי מצב של מחלקת "DarkMode".

\*הערה לבודק – החלק הבא נכתב כנספח למטלה

החלק הבא הוא תוספת לdesign pattern שציינו לעיל לטובת נוחות שימוש ומתן אפשרות למשתמש לחזור אחורה ( המעבר מdark mode חזרה לתצוגה רגילה). מכיוון שכבר פירטנו על 3 design patterns בחרנו שלא לפרט על זאת.

כחלק מהתהליך של הdark mode מימשנו גם תבנית Command אשר נמצאת במחלקת DarkDisplayCommand, לא נפרט עליה בפני עצמה אך המוטיבציה לכך הייתה להפוך את בקשת המשתמש לאובייקט עצמי שמכיל את כל המידע על הבקשה. הטרנספורמציה הזו מאפשרת להעביר בקשות כארגומנט במתודה, אם בעתיד נרצהModes נוספים, נוכל ביתר קלות להוסיף אותם ואת המימוש שלהם, מבלי לשנות את המימוש של המצב הכהה והמצב המקורי. במקרה הפרטי שלנו design pattern זה מסייע לנו לחזור למצב הקודם של תצוגת המשתמש.

לצורך מימוש תבנית עיצוב זו, מימשנו מחלקה גם בשם ControlProperties ששומרת לכל control את שדה צבע הרקע והכתב ההתחלתיים שלו, זאת על מנת שכשהמשתמש ירצה לבטל את המצב הכהה הוא יחזור לתצוגה המקורית. באופן מימוש זה ניתן בעתיד להוסיף שדות נוספים שהמערכת שומרת את המצב ההתחלתי שלהם, כך שבעת שינוי נוכל להחזיר גם אותם לצורה המקורית ללא מאמץ נוסף.

* Sequence Diagram

[sequence diagram שמציגה את התהליך שקשור לתבנית והאינטראקציה בין המחלקות]

* Class Diagram

[class diagram שמתארת את המחלקות שלכם שמעורבות בתבנית (תיאור מלא שכולל Properties ו- Methods) והיחסים ביניהם (תאור מלא עבור כל יחס כפי שלמדנו בכיתה). עבור כל מחלקה שלכם, כיתבו מי המקבילה שלה (אם יש כזו) בתבנית (Pattern) שבחרתם