### 

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

* יצירת קורות חיים באמצעות מידע המשתמש מפרופיל הפייסבוק, אפשרות לעריכתן ושמירתן על מחשב המשתמש.

אופציה נוספת, היא שליחת קורות החיים למייל הנתון בפרופיל המשתמש(כתובת הניתנת לשינוי).

* פרסום פוסט בקיר המשתמש, בתאריך ובשעה כלשהי בעתיד, לבחירתו

### תבנית מס' 1 – [Observer]

* סיבת הבחירה / שימוש בתבנית:

בממשק winforms, אשר מממשק בין המשתמש לבין האפליקציה

ייתכנו מספר רב של שדות, ביניהם : שדות טקסט (textbox, textList ועוד), שדות תמונה (pictureBox) ועוד רבים אחרים אשר קצרה היריעה מכדי להרחיב.

באפליקציה אשר יש צורך לאפשר ניתוק משתמש אחד בכדי לאפשר חיבור שני, יש צורך לנקות את שדות אלה, בכדי שהאפליקציה תפעל כראוי.

משמע, בעת יציאת משתמש מן האפליקציה, יש לעבור בין השדות השונים, ולנקותם.

בדרך זו, נצטרך לעבור שדה שדה ולבצע זאת. אופן ביצוע זה אינו תחזוקתי ויכול לגרום לקושי במימוש.

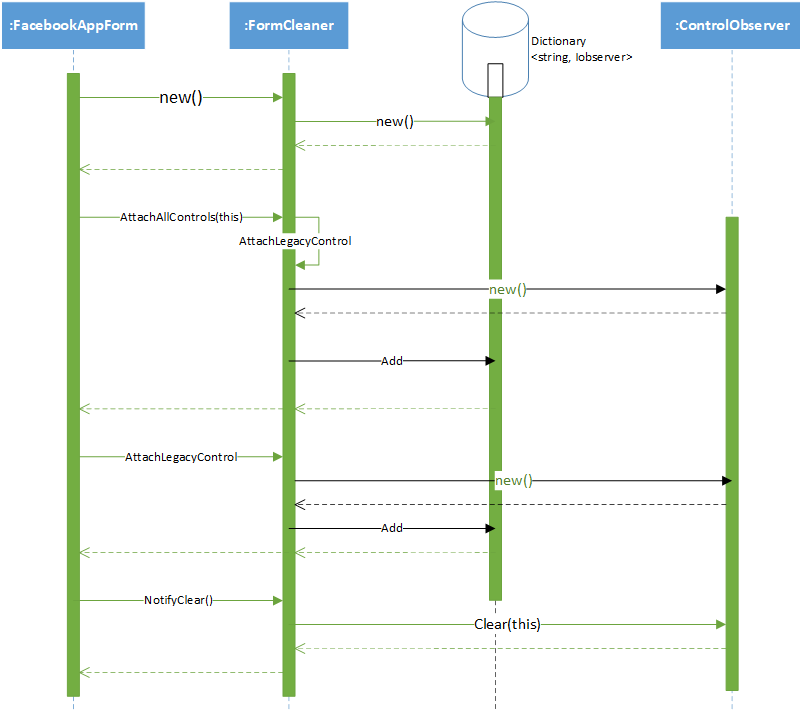
לכן בחרנו לממש תבנית Observer לניקיון כלל השדות בטופס, כאשר האחרונים הם מאזינים. כאשר הטופס יזהה שהמשתמש יוצא, הוא יקרא לכלל השדות, לרוקן את תוכנם.

* אופן המימוש:

כלל המימוש נמצא במחלקה CleanerObject וזאת בכדי שכל המימוש יהיה בקובץ אחד, באופן portably, כך שאפשר לצרפו לפרוייקטים אחרים בעתיד במהירות.

בעת אתחול הטופס מאותחל האובייקט[לא יודע איך לקרוא לו] , המחזיק במבנה נתונים (מילון) את כל המאזינים לו. אנו צירפנו את כל השדות מן הControl שאנו מעוניינים שיאזינו לו, בעת אתחול הטופס. בעת יציאת המשתמש מופעלת הפונקציה resetUI, אשר בעבר ביצענו ניקוי השדות זו אחר זו וכעת אנו רק קוראים לObserver , בכדי שיודיע לשדות הרלוונטים להתנקות.

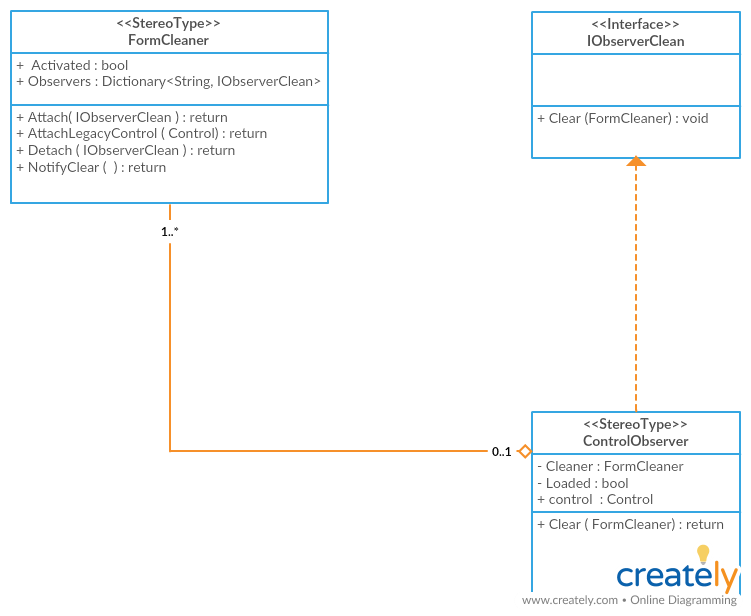
* Sequence Diagram



מתבצע כמה פעמים עד ניקוי מלא של הפקדים

בסוף התוכנית מיידע שצריך לנקות את הפקדים

* Class Diagram



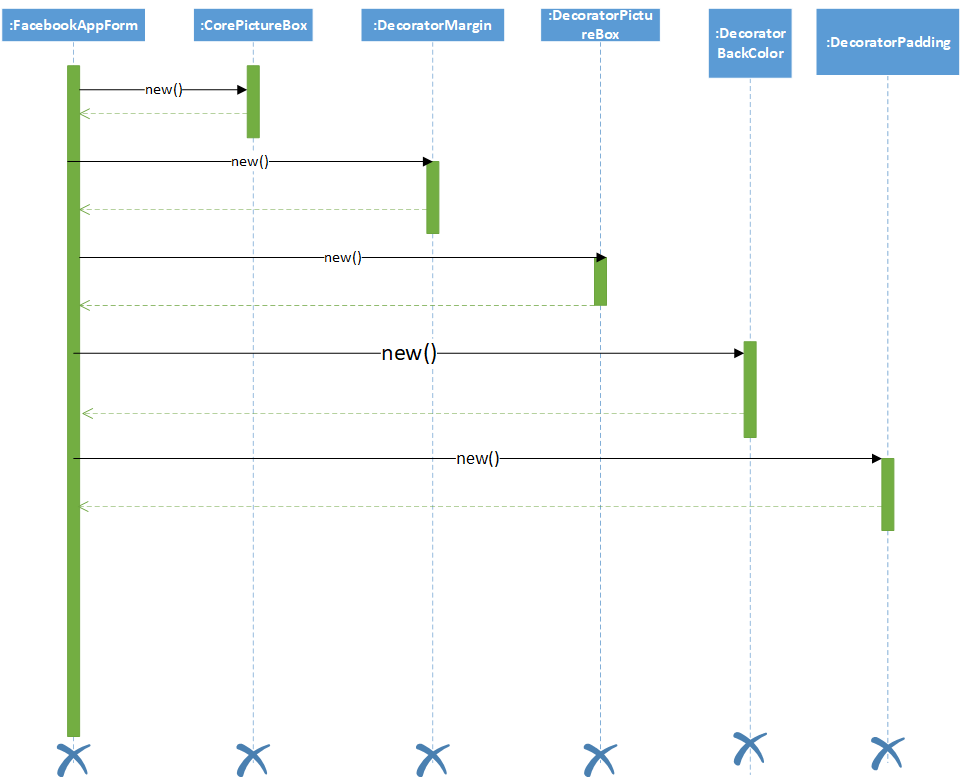
### תבנית מס' 2 – [Decorator]

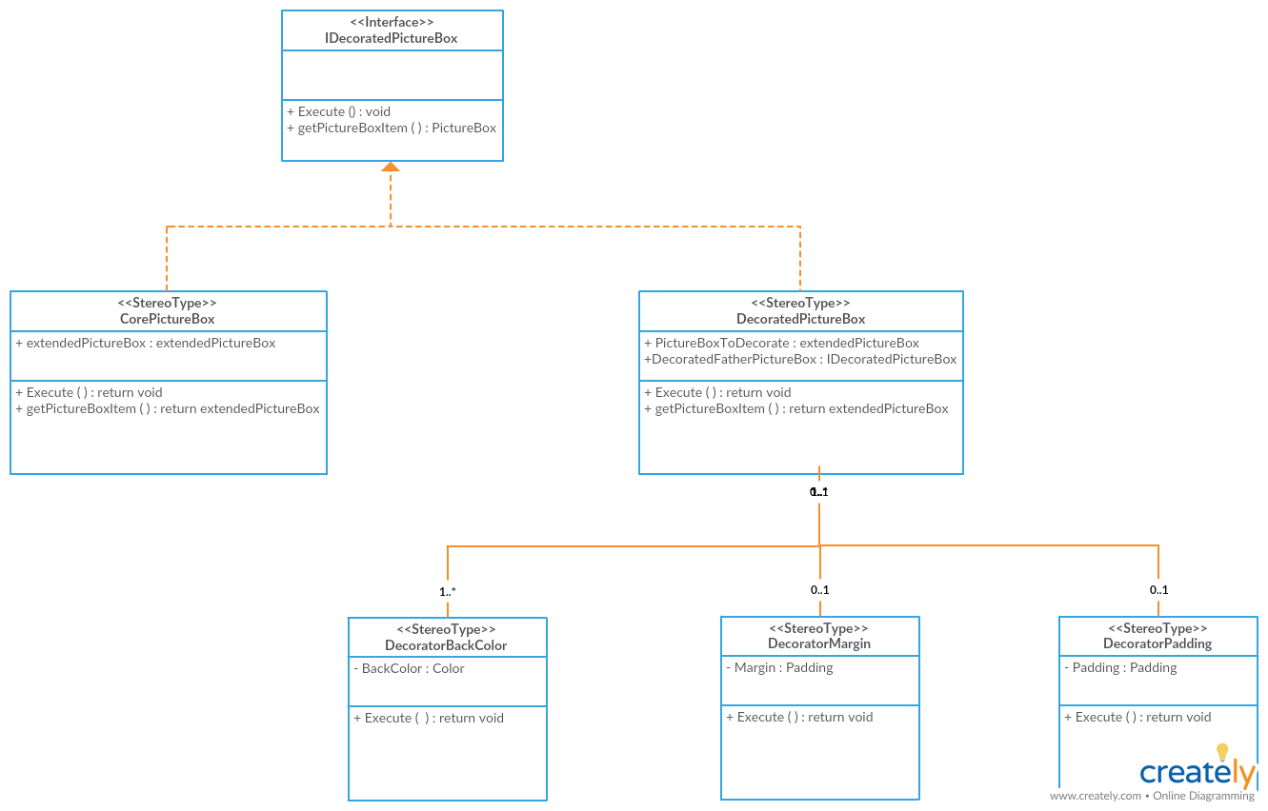
* סיבת הבחירה / שימוש בתבנית:

באפליקציה השתמשנו בהרחבה לטיפוס pictureBox, ורצינו לממש אותו באופן שנוכל להשפיע על תכונותיו ועל סדר החלתן. כמו כן, רצינו לממש אפשרות להרחבת תכונותיו בעתיד, לכן בחרנו בתבנית Decorator.

* אופן המימוש:

כלל המימוש במחלקה extendedPictureBox וזאת בכדי להראות את השוני בין אפשרות הירושה לאפשרות מימוש שינויים באובייקט בעזרת תבנית Decorator.

* Sequence Diagram
* Class Diagram



### תבנית מס' 3 – [Command]

* סיבת הבחירה / שימוש בתבנית:

בחרנו להשתמש בתבנית זו כיוון שרצינו לעשות הפרדה בין רכיבי ה-UI של הפונקציות (הקשורות ביצירת קורות חיים)שהוספנו למערכת לבין הפעולות שהמערכת מבצעת בעקבות הפעלתם.

כך בעתיד נוכל להחליף רכיב UI מבלי לשנות הלוגיקה שעומדת מאחוריו, להחליף את הלוגיקה עבור רכיב קיים וכן גם להוסיף רכיבים נוספים ולוגיקה נוספת תוך שינוי מינימלי בקוד הקיים.  
אופן המימוש:

יצרנו מחלקה אבסטרקטית FormFacebookAppCommandאשר מממשת את הממשק הדוט נטי ICommandExecutor. בה שדה בשם Client אשר יחזיק Reference למחלקה העוטפת של יורשי AppFormCommand.

יצרנו שלוש מחלקות מקוננות ב-FacebookAppForm,

אשר יורשות מ- FormFacebookAppCommand:

CreateResumeCommand 1.

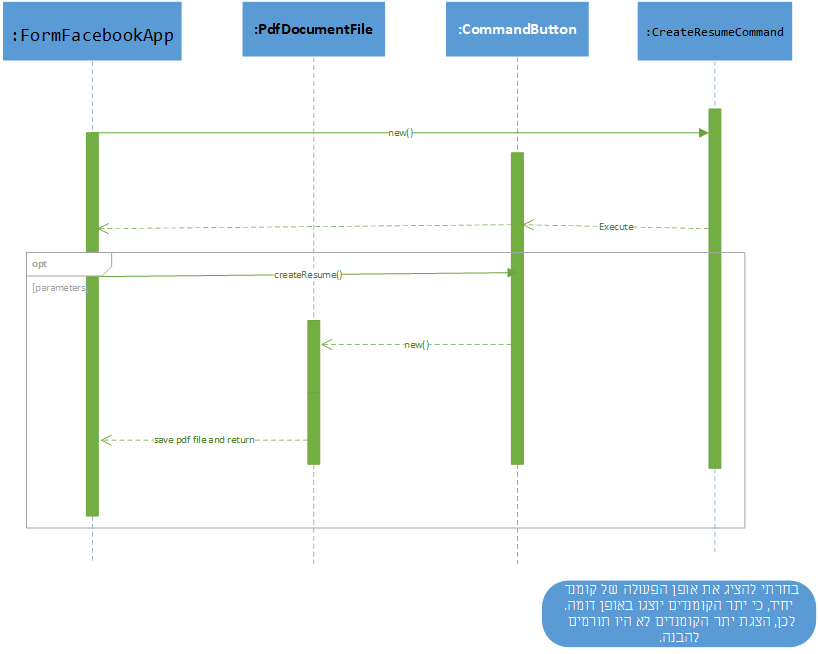
FetchResumeInformationCommand 2.

3. SendResumeToMailCommand

כל אחת מהמחלקות מממשת את Execute באופן קונקרטי תוך שימוש במחלקה PdfDocumentFile שמבצעת בפועל את הפונקציות הנ"ל.

על מנת ליצור פקד חדש מסוג Button אשר עובד בתבנית Command יצרנו את המחלקה CommandButton אשר יורשת מ-Button של WinForms ובה שדה בשם Command מסוג ICommandExecutor וכן דריסה של מתודת OnClick של Button כך שזו תקרא למתודת Execute של Command.

* Sequence Diagram



* Class Diagram

