### 

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

**משחק מיקומי תמונות אלבומים:**

* הפיצ'ר מציג למשתמש ארבע תמונות Cover מתוך אלבומיו אשר קיים להם מיקום ותמונה. (משתמש ללא 4 אלבומי בעלי תמונה ומיקום, הפיצ'ר לא יהיה זמין עבורו).
* בעת לחיצה על אחת התמונות, יפתח טופס המשחק-
  + תמונת האלבום
  + 4 תשובות אפשריות נכונות
* בעת מענה על תשובה נכונה- מוצגת הודעה למשתמש, מספר הנקודות שצבר במשחק עולה והתמונה בטופס הראשי של האפליקציה מוחלפת לתמונת אלבום אחרת.
* בעת מענה על תשובה לא נכונה- מוצגת הודעה למשתמש. התמונה נשארת על הטופס הראשי של האפליקציה, כלומר המשתמש יכול לנסות לשחק שנית ולנחש שוב היכן צולמה התמונה.

**מציאת דייט:**

* הפיצ'ר מאפשר למשתמש למצוא דייט בעזרת הפייסבוק עם אדם שאינו זר לחלוטין.
* על המשתמש לבחור את חתך הנתונים: מין וטווח גילאים.

לאחר לחיצה על כפתור Find Match יוצגו למשתמש כל החברים שלו אשר:

* + במין המבוקש
  + בטווח הגילאים המבוקש
  + המצב המשפחתי שלו אינו מוגדר או מוגדר כרווק

במידה ונמצאה התאמה, המשתמש יוכל לבחור מתוך רשימת המתאימים/ות את החבר שהוא מעוניין לצפות בנתוניו.

### תבנית מס' 1 – Adapter

* **סיבת הבחירה / שימוש בתבנית:**

במערכת השתמשנו ב-Adapter Pattern כדי להתאים לצרכינו את השימוש באובייקט Post.

האובייקט Post מכיל נתונים שונים כמו Message/ Description /Name/PictureUrl וכו'.

במערכת שלנו, אנו מציגים רק Name, Date, Message, ו-Picture.

שמנו לב, כי חלק מהפוסטים מכילים Description, חלק Message וחלק רק Name.

בנוסף, לחלק מהפוסטים יש PictureUrl ולחלקם אין.

לכן, בחרנו להשתמש בPattern שמאפשר לנו להשתמש באופן מותאם באובייקט Post ולבחור בעזרת תנאים לוגיים איזה פרטים להציג בעת לחיצה על כל אחד מהפוסטים.

לדוגמא-

1. כאשר משתמש ילחץ על פוסט שלא קיימת לא תמונה.

תטען התמונה הדיפולטית הבאה:

1. כאשר משתמש ילחץ על פוסט שלא קיים לו Message, יוצג Description ובמידה וגם Description לא קיים, יוצג ה-Name.

* **אופן המימוש:**

ניתן למצוא את הקוד במחלקה AdapterPost.

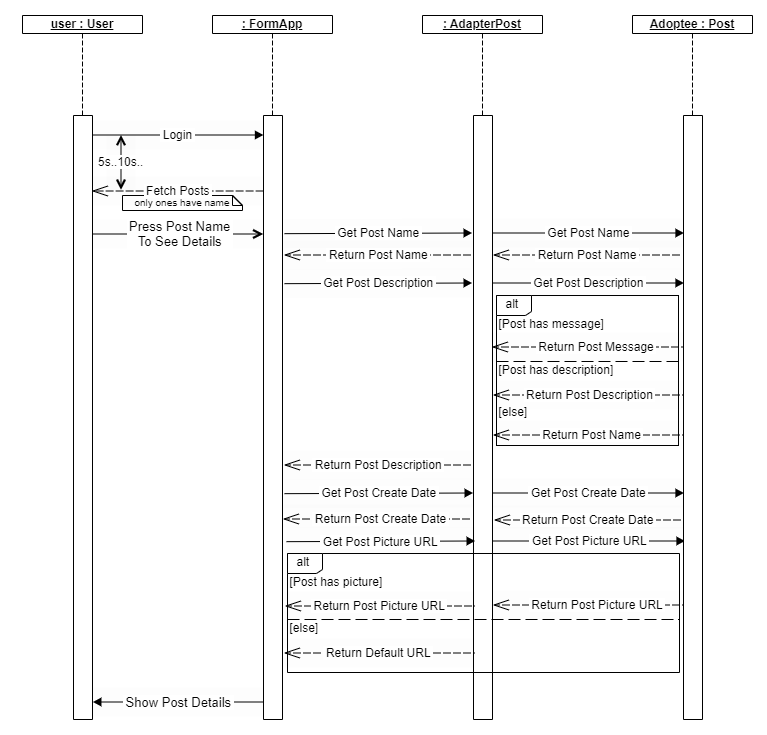
בממשק המשתמש כאשר אנו טוענים את הפוסטים של המשתמש לרשימת הפוסטים באפליקצייה listBoxPosts, אנו למעשה יוצרים אובייקטים מסוג AdapterPost שעוטפים בתוכם את ה-Postים המקוריים.

כלומר ListBoxPosts למעשה מכיל אוסף של אובייקטים מסוג AdapterPost, שכל אחד מהם עוטף בקומפוזיציה Post.

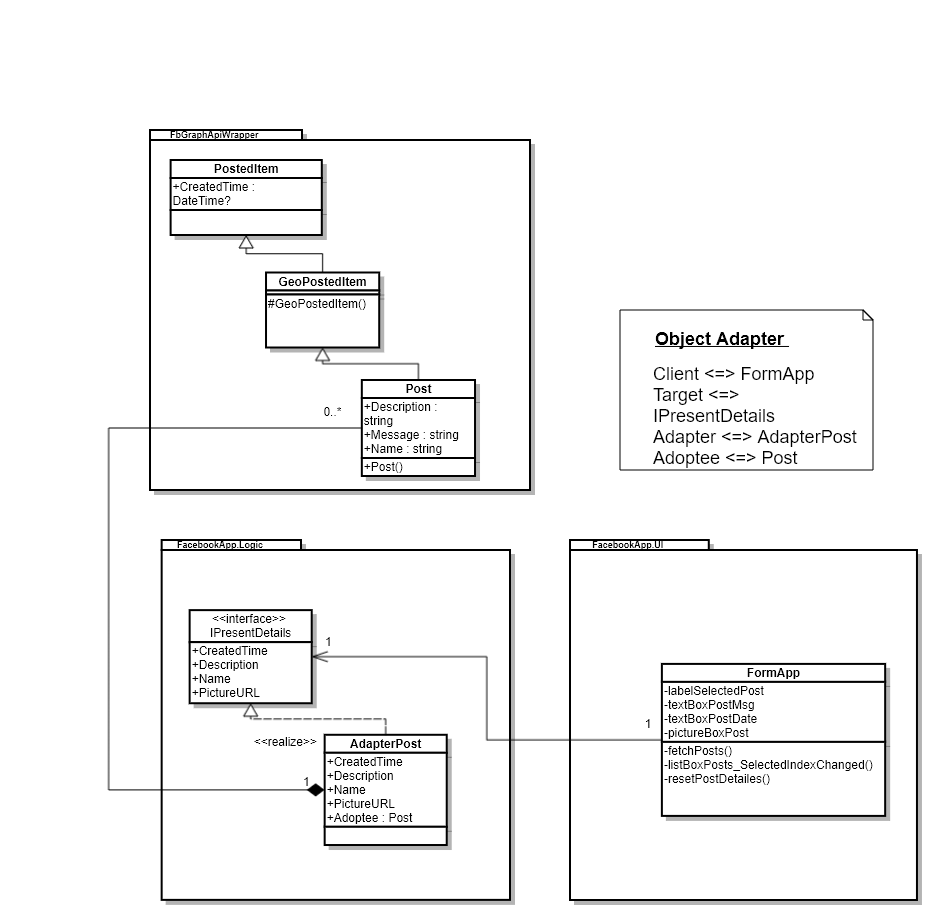
כעת, בעת לחיצה על פוסט מסויים מתוך רשימת הפוסטים מתבצעת אחת מהפעולות הבאות:

1. אם זו הפעם הראשונה – נטענים הנתונים הרלוונטים ה'מותאים' ל-Data Members של הAdapterPost על פי התנאים הלוגיים בתוך ה-Properties ולאחר מכן מוצגים למשתמש.
2. אחרת, (אם זו לא הפעם הראשונה) כל פנייה Propertyל תחזיר את הData Member המתאים שכבר מאותחל בנתונים הרלוונטים של הפוסט המבוקש.

* **Sequence Diagram**



* **Class Diagram**

****

### תבנית מס' 2 – FAÇADE

* **סיבת הבחירה / שימוש בתבנית:**

בחרנו לממש את ה-Façade Pattern בפיצ'ר של "משחק מיקומי תמונות".

הסיבה שבגינה בחרנו לממש Pattern זה, היא כיוון שהמשחק מכיל לוגיקה רבה ומורכבת ורצינו ליצור הפרדה מוחלטת בין השכבה הלוגית לממשק המשתמש של המשחק.

בעתיד, ניתן יהיה בקלות רבה יותר להתאים ממשק משתמש חדש למשחק זה.

הדבר היחידי שיהיה צורך בו, הוא לפנות לאותן מתודות וProperties שמחזירות מהFacade את הנתונים הרלוונטים לתצוגה ולאתחול המשחק.

* **אופן המימוש:**

המחלקה הסטטית FacadePictureGame מכילה DataMembers אשר זהים לרכיבים המוצגים בממשק המשתמש.

למשל: בטופס המשחק FormPictureGame קיימים ארבעה כפתורים עם תשובות אופציונאליות

ובאותו אופן ב FacadePictureGame קיימות ארבע תשובות אפשריות.

בנוסף, המחלקה מכילה מתודות ליצירת משחק, לאתחול משחק חדש וכו'.

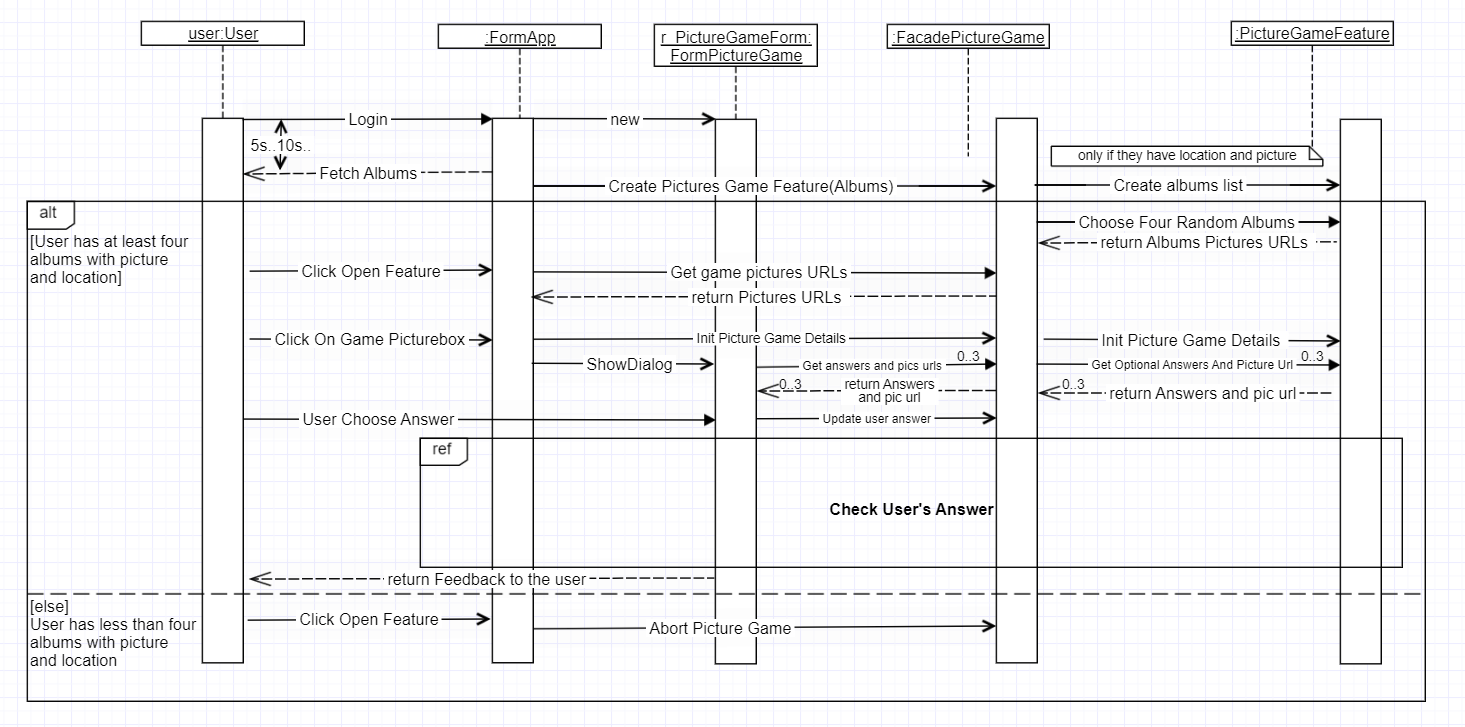
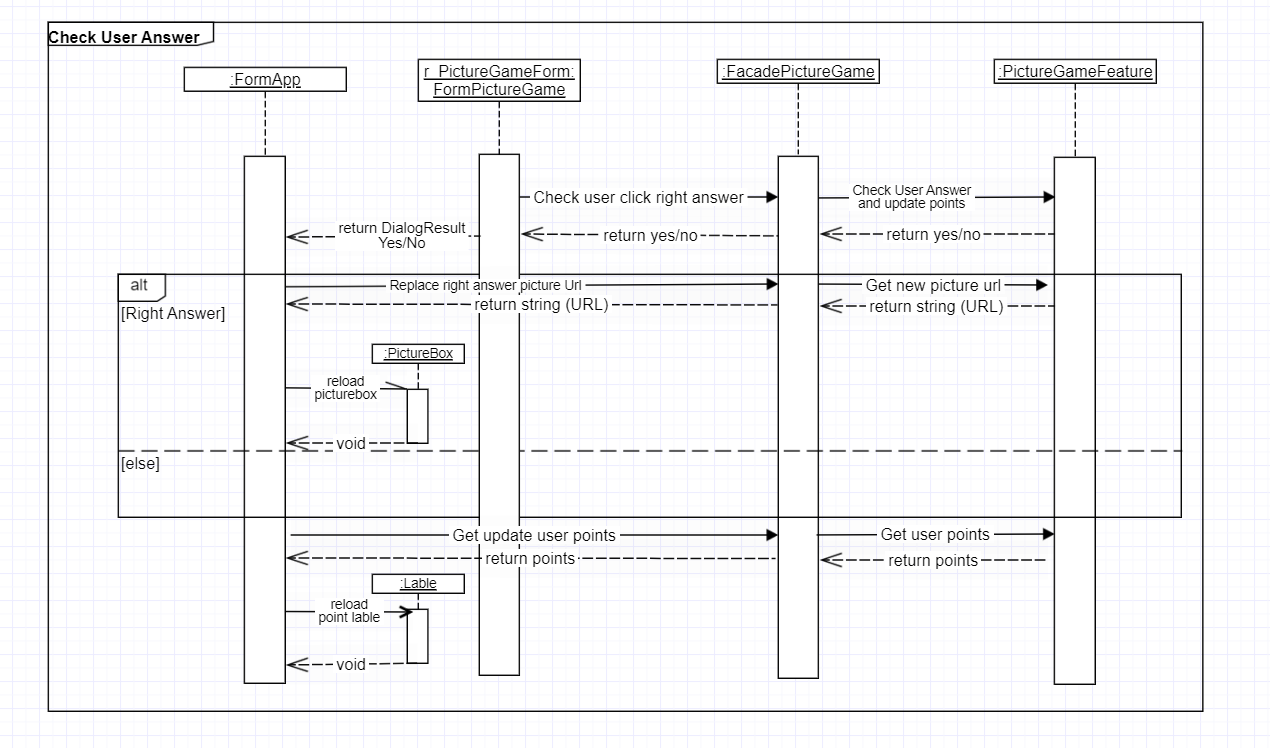
מתודות אלו חשופות לממשק המשתמש ומפעילות את סט המתודות הנדרשות מהשכבה הלוגית של המשחק.

למשל: המתודה CreatePicturesGameFeature() של ה Facade חשופה למחלקות בממשק המשתמש, ובפועל מריצה מספר מתודות מהשכבה הלוגית של המשחק.

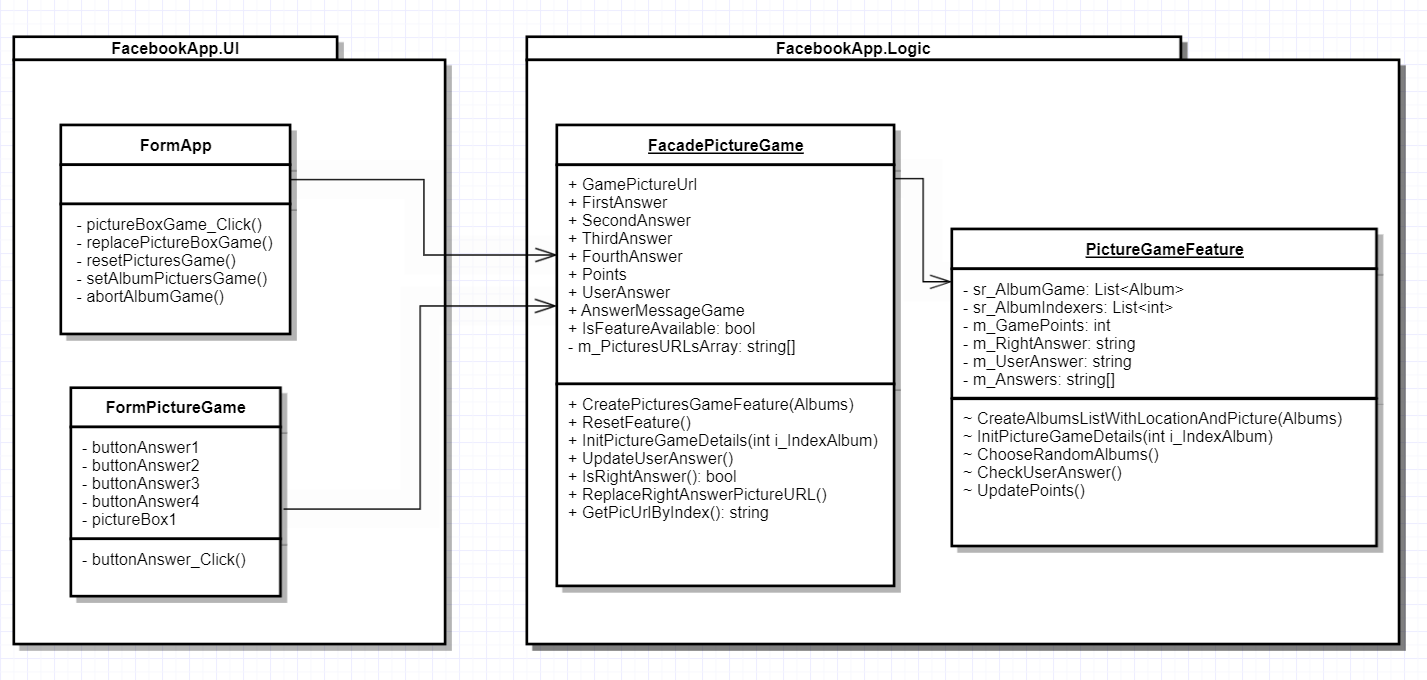
מחלקת הFacade שלנו מבצעת קריאות ללוגיקה של המשחק הממומשת במחלקה הסטטית PictureGameFeature ומחזירה לממשק המשתמש (הטופס המוצג FormPictureGame) את הנתונים הרלוונטים להצגה.

ה-Facade שמימשנו הוא אטום. לא ניתן לפנות לשכבה הלוגית PictureGameFeature מממשק המשתמש אלא רק דרך FacadePictureGame.

* **Sequence Diagram**



* **Class Diagram**



### 

### תבנית מס' 3 – ABSTRACT FACTORY

* **סיבת הבחירה / שימוש בתבנית:**

באפליקצייה שלנו ישנם טפסים שונים המוצגים למשתמש:

טופס אלבום – המכיל כותרת, תמונה, מיקום ותאריך.

טופס חבר (משתמש) – המכיל שם, תמונה, מין, גיל ומקום מגורים.

טפסים אלו, זהים מבחינת כמות ה-Labels ומיקום התמונה ולכן, על מנת למנוע שיכפול קוד ויצירת טפסים מרובים בממשק המשתמש בחרנו לממש Abstract Factory Pattern שמייצר את הטופס המבוקש להצגה בכל פעם ומציג אותו למשתמש.

בעתיד, ואם נרצה להציג טפסים נוספים, ניתן יהיה בקלות רבה (בהוספת שורות קוד בודדות) לבצע זאת.

* **אופן המימוש:**

המחלקה FormDetails הינה מחלקה שאנו מייצרים ממנה אובייקט באפליקצייה הנקרא: r\_FormDetails .

זהו למעשה טופס Template לנתונים המוצגים באפליקציה שלנו.

מחלקה זו מחזיקה כDataMember:

* IFormDetails שהוא Interface הממומש ע"י המחלקות IFormFriend ו-IFormAlbums.

למעשה, כל אובייקט שנרצה להציג על גבי הטופס יממש Interface זה.

* Details שהיא מחלקה אבסטרקטית אותה יורשים FriendDetails ו- AlbumDetails, אלו הנתונים הרלוונטים להצגה עבור אוייבקטים מסוג Album/Friend בהתאמה.

כאשר משתמש בטופס הראשי לוחץ על פתיחת טופס חבר (Friend) לדוגמא:

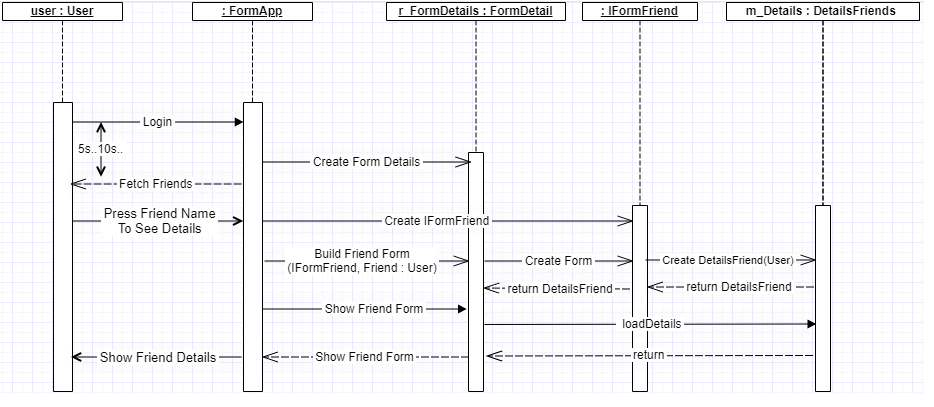
1. מתבצעת הקריאה לr\_FormDetails.BuildForm עם המחלקה הרלוונטית שמממשת את הInterface IFormDeatils ועם האוייבקט שנרצה להציג את נתוניו.

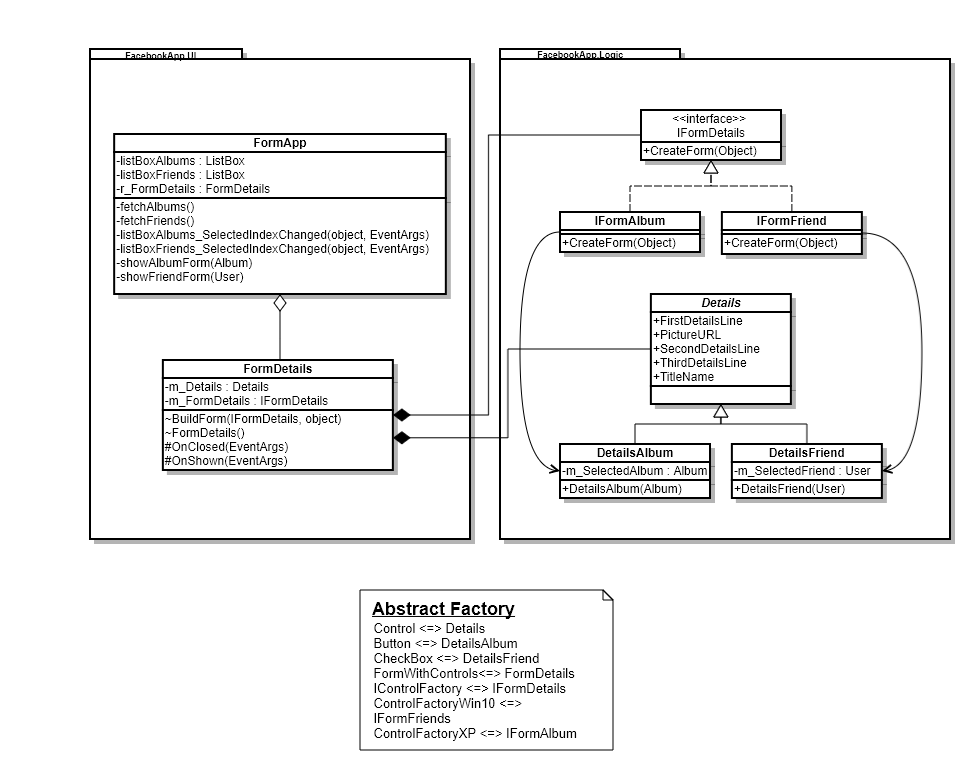
כלומר, בדוגמא זו נשלח IFormFriend ואובייקט User i\_Friend.

1. המתודה BuildForm תציב בDataMember m\_IFormDetails את המחלקה המממשת את הinterface (IFormFriend/IFormAlbum) ותקרא למתודה של הInterface הנקראת CreateForm תוך שליחת האוייבקט (User/Album) כObject.
2. המתודה CreateForm של כל אחת מהמחלקות המממשות את interface IFormDetails תייצר אובייקטDetails המתאים לUser/Album ותחזיר אותו לטופס.
3. אחרי קבלת האובייקט מסוג Details עם הנתונים לתוך הDataMember m\_Details ובעת הצגת הטופס r\_FormDetails.ShowDialog נלקחים הנתונים מm\_Details לתוך הLables בטופס הTemplate ומוצגים למשתמש.

לדוגמא: this.labelLine1.Text = m\_Details.FirstDetailsLine

* **Sequence Diagram**

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



* **Class Diagram**