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

בתרגיל הקודם מימשנו את הפיצ'רים:

NavigateToEvent

FilterPosts

MostLikedPhoto

MostAttendedEvent

MostLikedPost

כעת הרחבנו את המערכת שוב והוספנו אפשרות לשנות צבע של תמונה במגוון סוגי צבעים מיוחדים ובנוסף הוספנו ערכת thug life להוסיף על תמונה קיימת וניתן לפרסם את התמונה בפייסבוק.

### תבנית מס' 1 – Decorator

* סיבת הבחירה / שימוש בתבנית:

כאשר הסתכלנו על tab albums וחשבנו על הרחבה עתידית של הtab כאשר נרצה לעשות מניפולציות עם התמונות. כשנרצה לקשט אותם בכל מיני קישוטים מגניבים כמו לשנות צבע של תמונה או להוסיף לתמונה פריטים שונים, במצב זה נתקלנו בבעיה.

הבעיה היא שבמקרה כזה נרצה שיהיה פולימורפיזם בין המקשט למקושט וכך נימנע גם בין השאר משכפול קוד. נרצה שיהיה כאן כמו multi proxy כמה שכלולים ביחד.

נרצה שהמקשטים יחזיקו עדיין את הגרעין ויוסיפו את השכלולים שלהם. לצורך כך השתמשנו בdecorator.

* אופן המימוש:

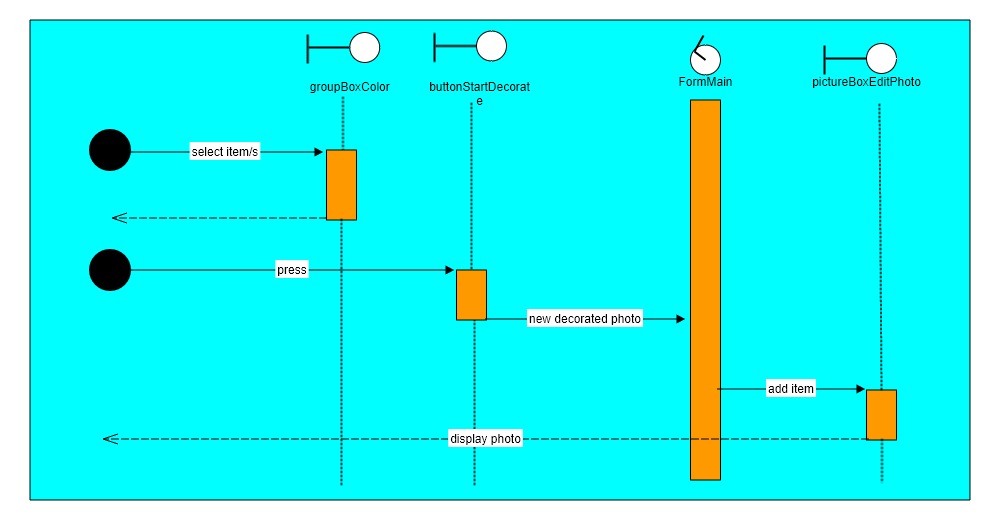
בנינו את המחלקה האבסטרקטית PhotoDecorator שהיא בעצם מהווה את הגרעין. היא תחזיק בקומפוזיציה רפרנס לPhotoComponent אותו אנו הולכים לקשט.

מPhotoDecorator יירשו 2 מחלקות ColorChangerDecorator ו- ThugLifeDecorator. כעת 2 המחלקות הללו מכילות את הגרעין של קישוט התמונה. כלומר פתרנו את בעיית שכפול הקוד בין 2 המחלקות הללו בהקשר של טיפול בגרעין של התייחסות לPhotoComponent.

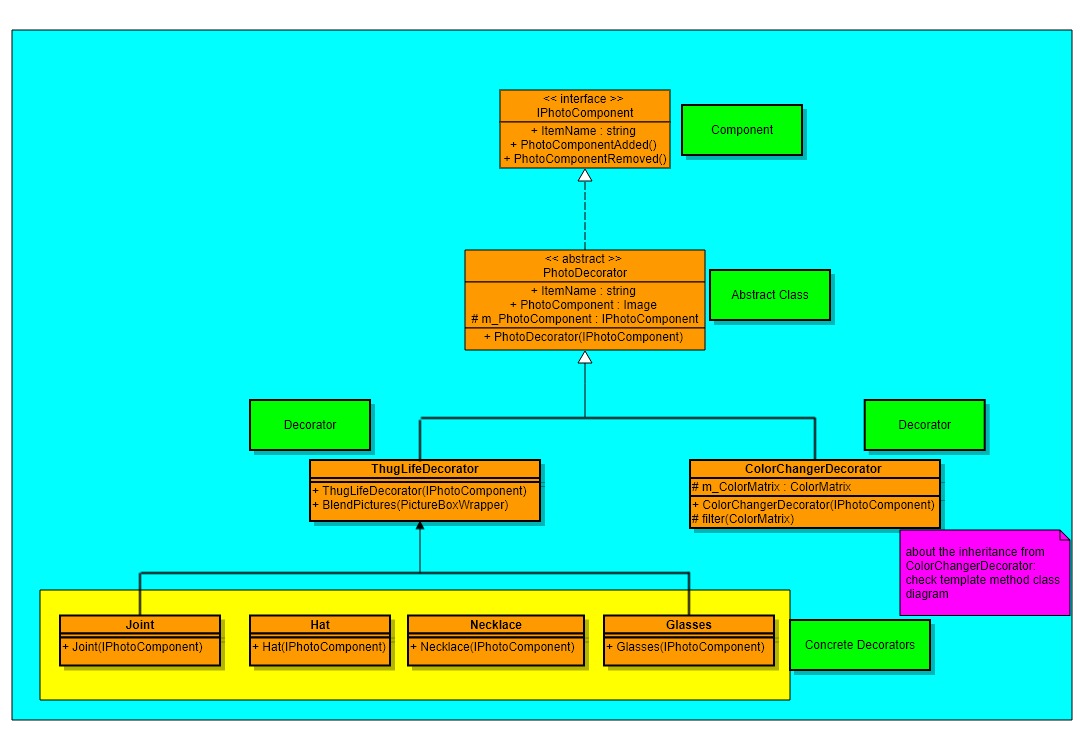
בנוסף אם בעתיד נרצה להוסיף מקשטים חדשים אז נוכל לעשות זאת בקלות.

הבעיה הנוספת שנוצרה כאן היא שכעת יהיה שכפול קוד בין המקשטים של ColorChangerDecorator ושכפול קוד בין המקשטים של ThugLifeDecorator. פתרון לכך מוצג בתבנית עיצוב מס' 2 template method.

* Sequence Diagram



* Class Diagram



### תבנית מס' 2 – Template Method

* סיבת הבחירה / שימוש בתבנית:

בהמשך ישיר לבעיה של תבנית עיצוב מס' 1 Decorator. לאחר שפתרנו את בעיית שכפול הקוד בהתייחסות לגרעין, כעת נרצה לפתור את בעיית שכפול הקוד בין המקשטים של ColorChangerDecorator ובין המקשטים של ThugLifeDecorator.

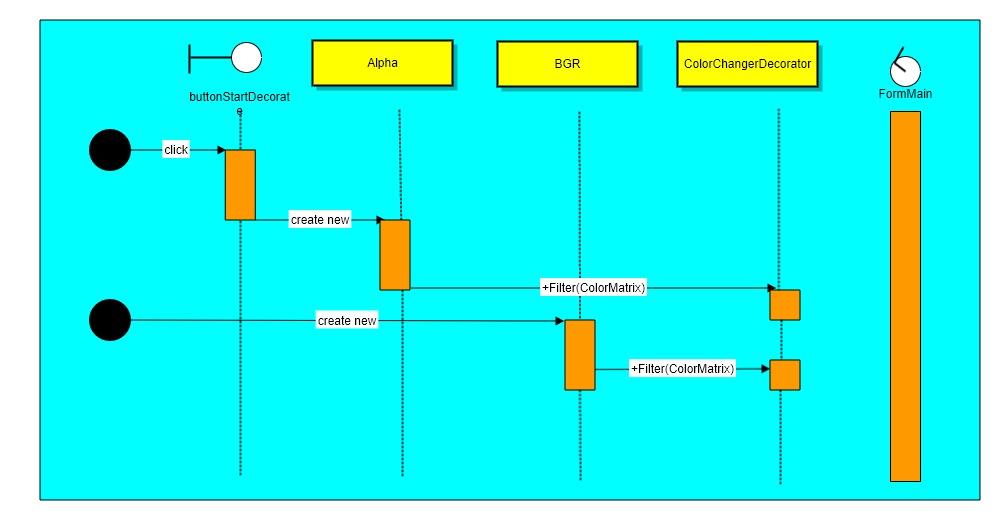
הבעיה שנוצרה כאן ישר הובילה אותנו ל2 פתרונות אפשריים template method או strategy. במקרה הזה בחרנו להשתמש בtemplate method.

* אופן המימוש:

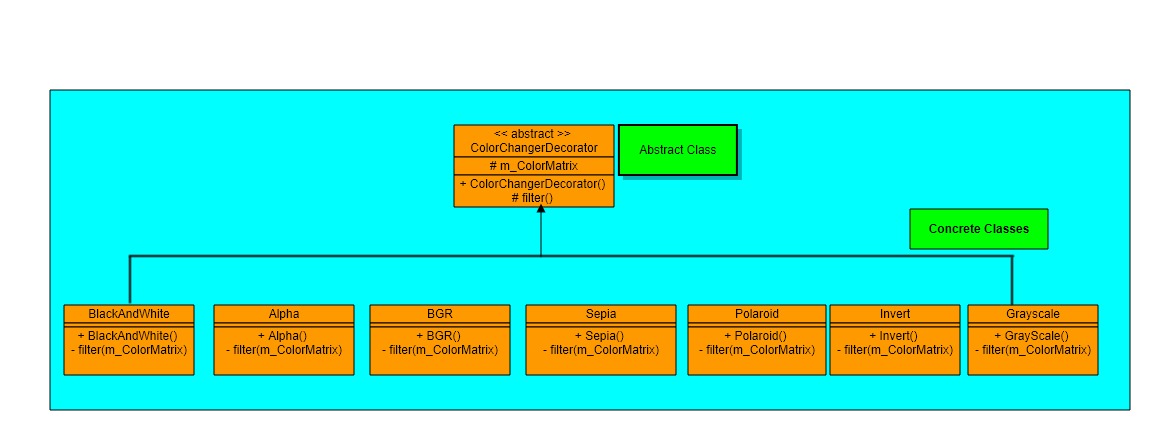
יצרנו את המחלקות BGR,Sepia,Polaroid,Alpha,BlackAndWhite,GrayScale,Invert שירשו מColorChangerDecorator ובכך יהיו פולימורפיים אליו.

ואת המחלקות Hat,Necklace,Joint שירשו מThugLifeDecorator וכך גם הם יהיו פולימורפיים אליו. ע"י כך פתרנו למעשה את בעיית שכפול הקוד בין כל קבוצה של מקשטים. ואם נרצה בעתיד להוסיף מקשט נוסף באותה קבוצה נצטרך לשנות זאת רק במקום אחד כלומר הדבר יהיה מיינטיינבילי וכך הפוטנציאל לבאגים גם יהיה קטן יותר.

* Sequence Diagram



* Class Diagram



### תבנית מס' 3 – Strategy

* סיבת הבחירה / שימוש בתבנית:

כאשר הסתכלנו על הפיצ'ר filter posts (סינון לפי keyword בpost)חשבנו על צורך עתידי שאם בעתיד נרצה להציג למשתמש אופציה לסנן לפי keyword של propery אחר למשל date (אולי נרצה להציג את כל הפוסטים בתאריך מסויים). במקרה כזה נרצה להציג למשתמש בחירה עבור איזה אסטרטגיית סינון הוא מעוניין להשתמש בלי לשכפל קוד, ושלא יהיו שינויים בקליינט לצורך כך.

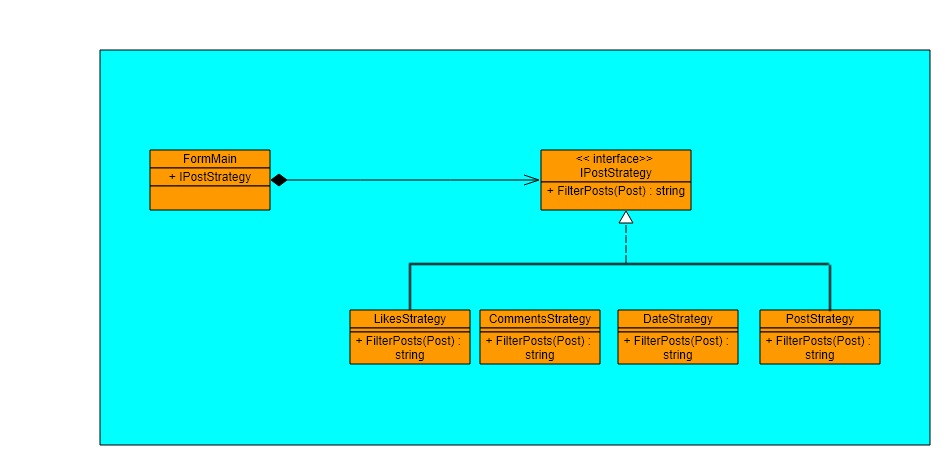
* אופן המימוש:

צורך זה הוביל אותנו להשתמש בstrategy. יצרנו את IPostStrategy שיכיל את המתודה FilterPosts. בנוסף יצרנו comboBoxPosts שבעזרת reflection מציג את כל הproperties של IPosts (שהשתמשנו בו בתבנית העיצוב adapter במטלה הקודמת והוא מציג בתוכו את הproperties אותם נרצה לראות ב dataGridViewPosts).

כעת יצרנו את PostStrategy,DateStrategy,LikesStrategy,CommentsStrategy שיממשו את הinterface – IpostStrategy ויציגו את הבחירה משלהם לפי מה נרצה לסנן את הפוסטים המוצגים. בנוסף, הוספנו property בקליינט FilterPostStrategy שכאשר המשתמש ילחץ על חיפוש בפוסטים נבדוק איזה שם מוצג בcomboBoxPosts וכך נאתחל את אסטרטגיית החיפוש. הבעיה שנוצרה כאן היא שבcomboBox יש string והיינו זקוקים למתודה שתהפוך string ל IPostStrategy דבר שהיה הופך את הקוד למאוד לא maintainable מאחר וזוהי מתודה בקליינט. לכן, יצרנו את המחלקה PostStrategyFactory שתיצור בעזרת factory method אסטרטגיית חיפוש חדשה.

כעת הקוד reusable,maintainable ואם בעתיד נרצה אסטרטגיית חיפוש אחרת לא נצטרך לשנות את הקוד אצל הלקוח.

* Sequence Diagram



* Class Diagram

