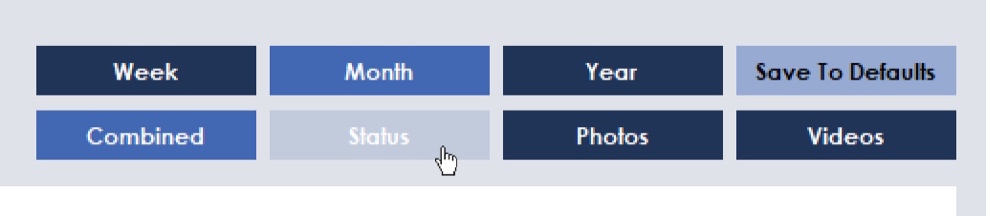
### תיאור קצר של הפיצ'רים שבחרנו לממש בתרגיל הקודם:

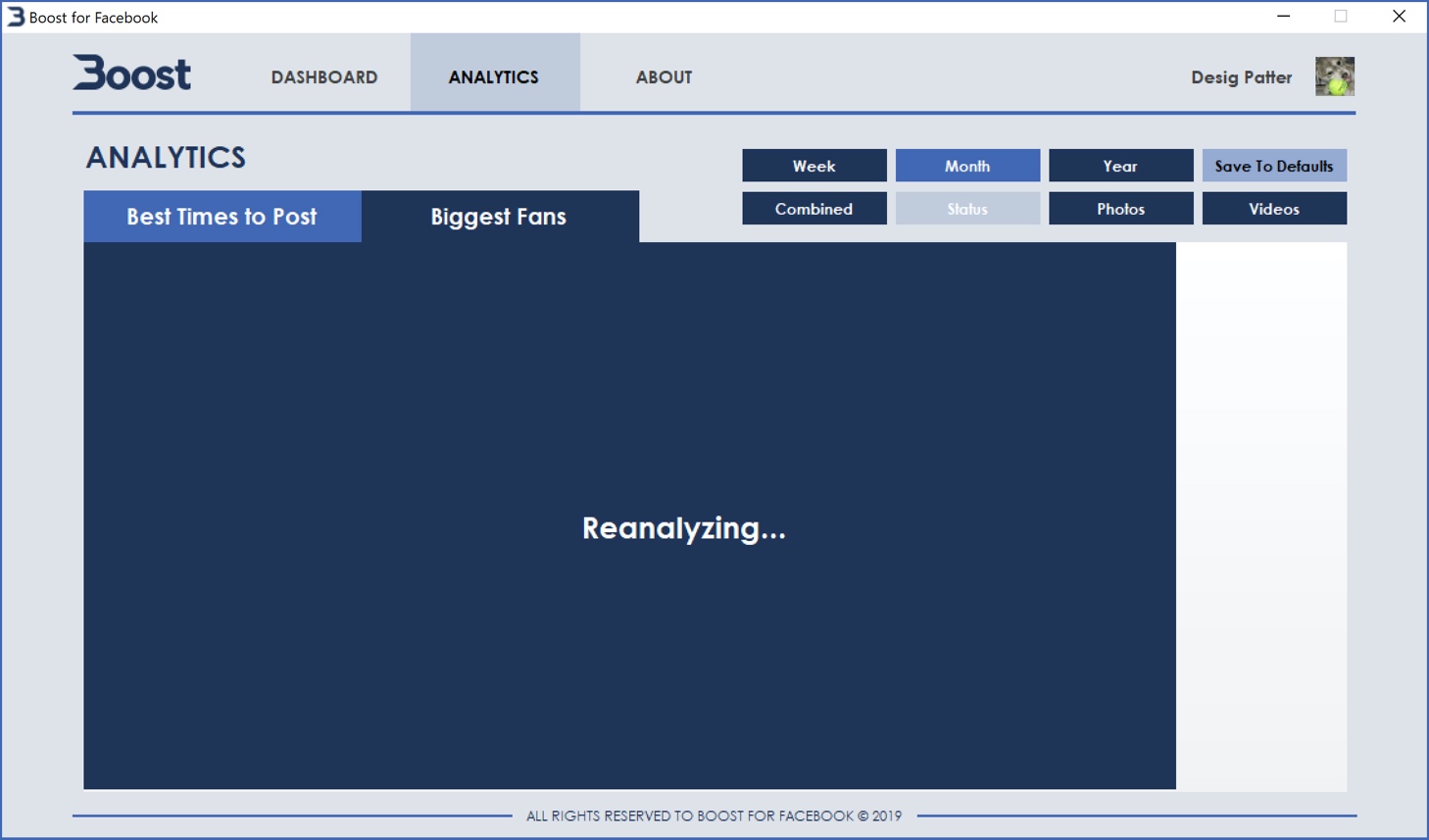
* - **Best Times to Post**מציאת הזמן הכי טוב לפרסם פוסט בחתך של תמונות, סטטוסים ווידאו ולפי קבוע זמן. בנוסף לכך, הצגה של מפת חום אשר מציגה את הזמן הטוב ביותר על גבי המפה לפי חתך כולל של כל הפרמטרים המוזכרים לעיל.
* **Biggest Fans** - מציאת החברים אשר עושים הכי הרבה לייקים לפוסטים של המשתמש, אותם אנו מגדירים כ״מעריצים הגדולים ביותר״ לפי חתך כולל של פוסטים.

### תבנית מס' 1 – Observer

* סיבת הבחירה / שימוש בתבנית:

כחלק מחוויית המשתמש, כאשר בוחרים אלמנט חדש לאנליזה:

במקרה זה, מופיעה אנימציה אשר מציגה למשתמש שנעשית אנליזה מחדש. אנימציה זו איננה חסרת משמעות:

* בשלב הראשון, ״אנימציית הכניסה״ ההתחלתית מופיעה בגלישה/צמיחה למסך עד שחלונה יגיע לגודל חלון האב שלו.
* בשלב השני, ״אנימציית היציאה״ תחל רק אם התקבלה התרעה שהאנליזה החדשה הסתיימה.

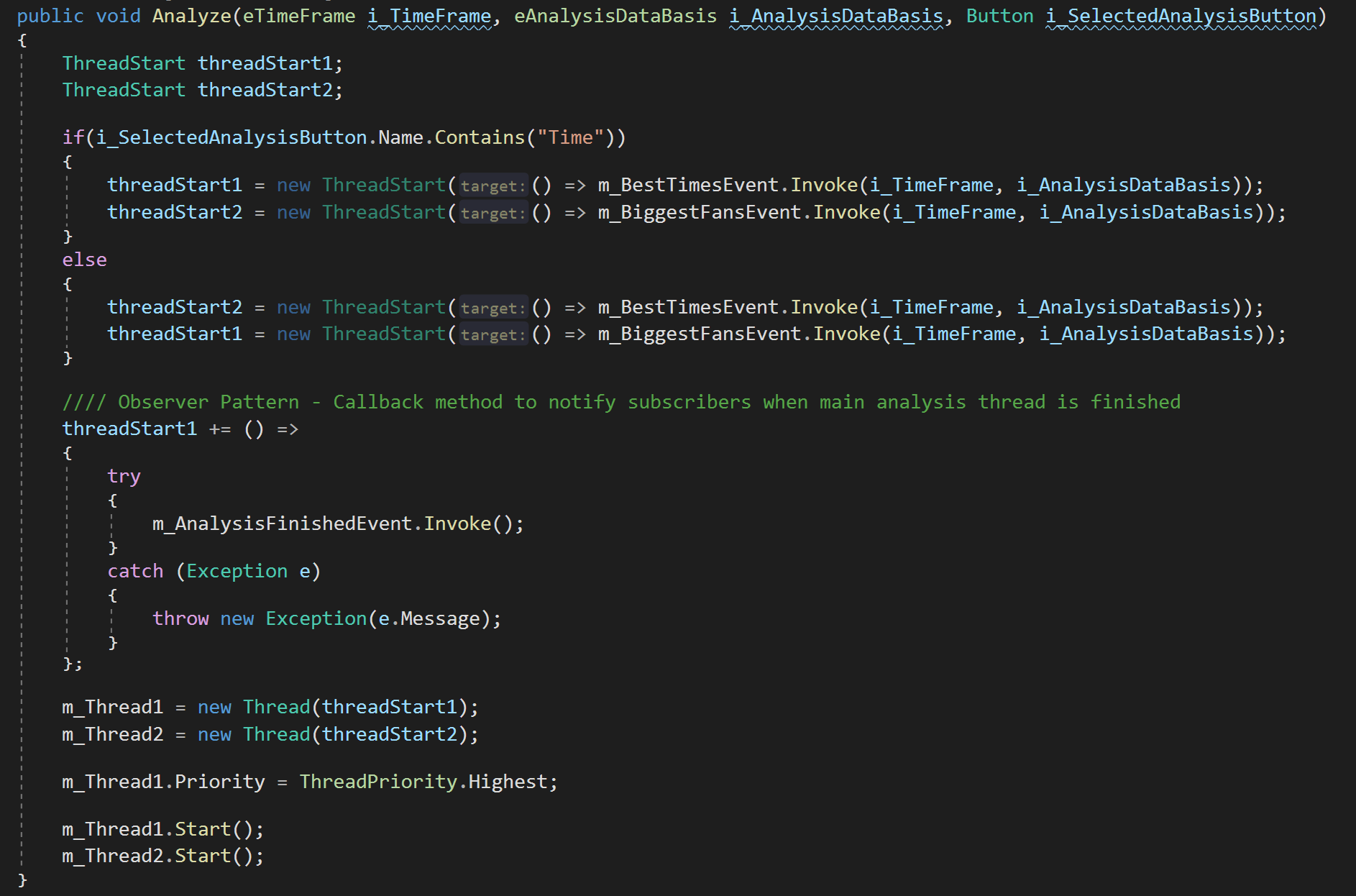
לפי ההסבר הנ״ל, החבירה בתבנית Observer ברורה.

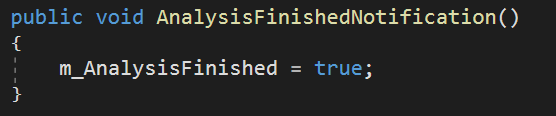
* אופן המימוש:

בחרנו לממש את התבנית בעזרת Delegates ו-Events כאשר המחלקה בה נמצאת מתודת האנליזה (AnalyticsViewModel) מחזיקה ב-Delegate הנקרא: AnalysisFinishedEventHandler() ו-Event הנקרא m\_AnalysisFinishedEvent

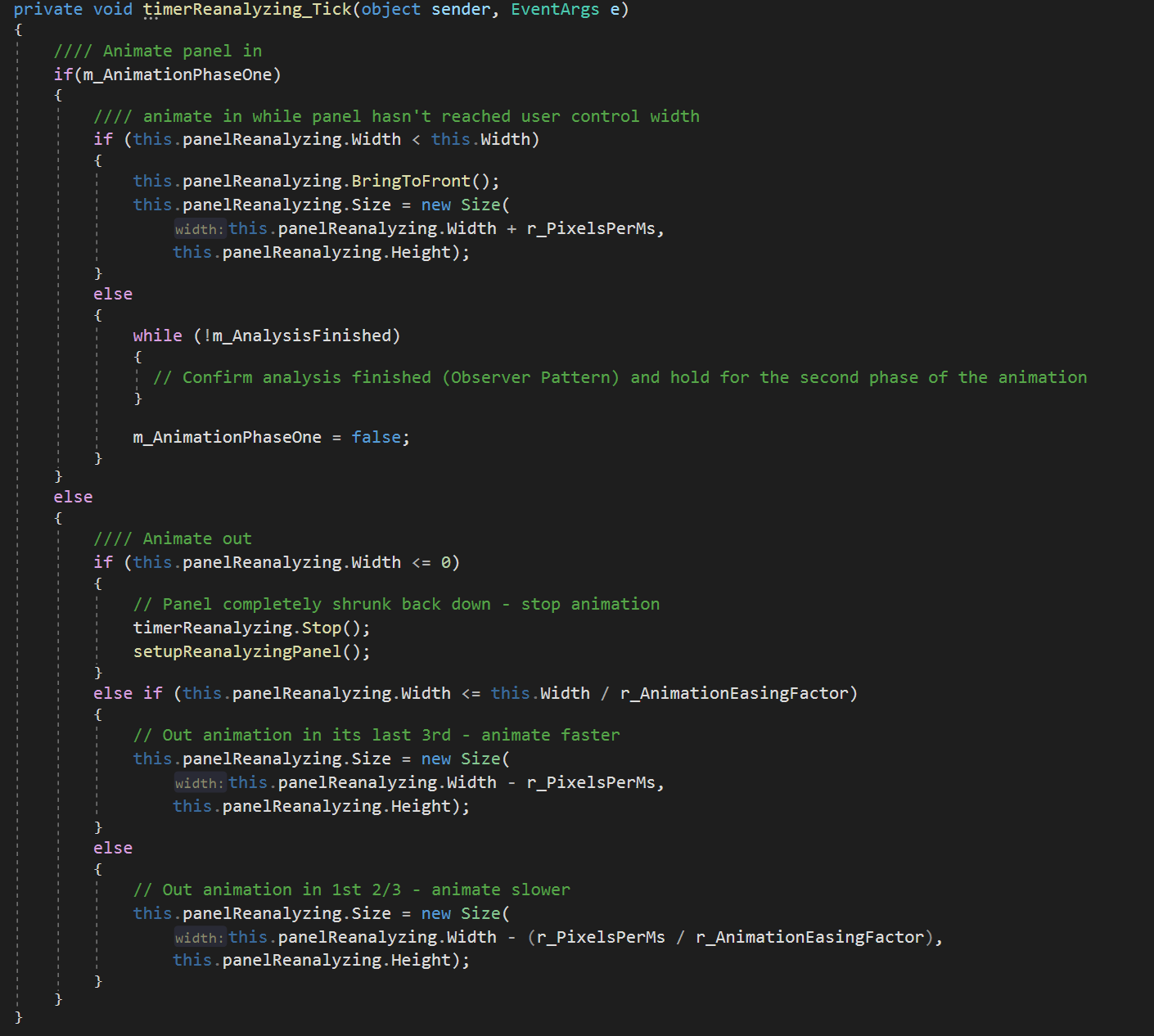
כמתודת האנליזה, Analyze, מפעילה את האנליזות של שני הפיצ׳רים שלנו שלנו (BestTimesToPost ו-BiggestFans) בשני ת׳רדים נפרדים, כאשר הת׳רד הראשון (בעל עדיפות גבוהה יותר) עושה אנליזה לפיצ׳ר שחלונו כרגע מוצג למשתמש ובהתאמה האנליזה השנייה נעשית בעדיפות נמוכה יותר ברקע.

מתודת ה-Invoke של ה-Event שהוזכר קודם לכן, מחוברת לסופו של הת׳רד הראשון (כסוג של מתודת Callback) ובכך תפעיל את כל המתודות הרשומות לאירוע בסופה של האנליזה.



תחת AnalyticsView, המכיל את חלון האנימציה reanalyzingOverlay (מופע של ReanalyzingView), אנו רושמים לאירוע m\_AnalysisFinishedEvent את המתודה האחראית להודעה לאנימציה על סיום האנליזה.

המתודה הנ״ל, משנה ערך בוליאני במחלקת האנימציה ReanalyzingView במהלך האנימציה כפי שהוסבר בסעיף הקודם, לאחר שזה קורה, השלה השני של האנימציה יחל ואחרת, האנימציה תישאר בשלב הראשון.



מיקום התבנית בקוד:

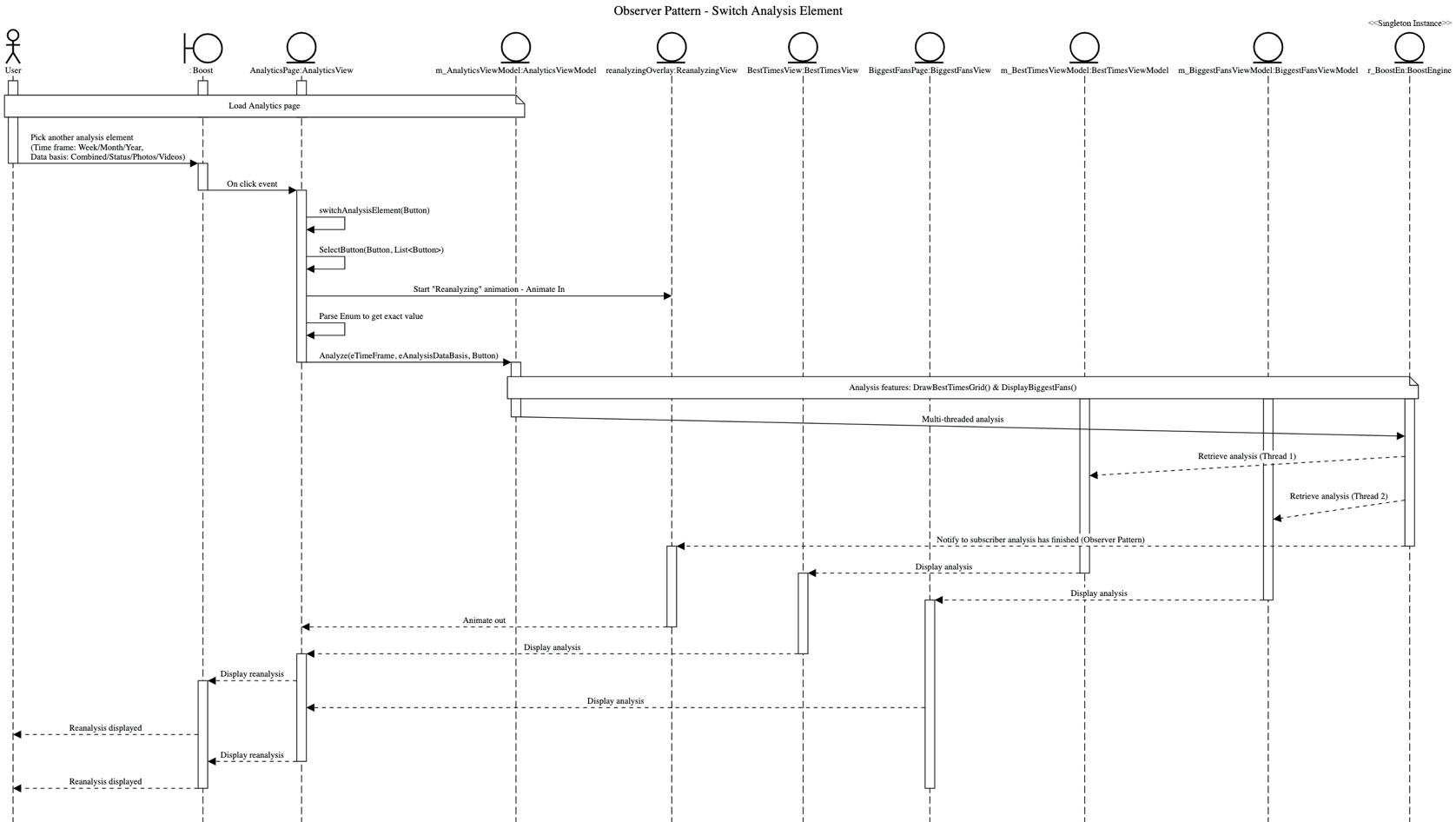
Observervable Subject & Notifier:

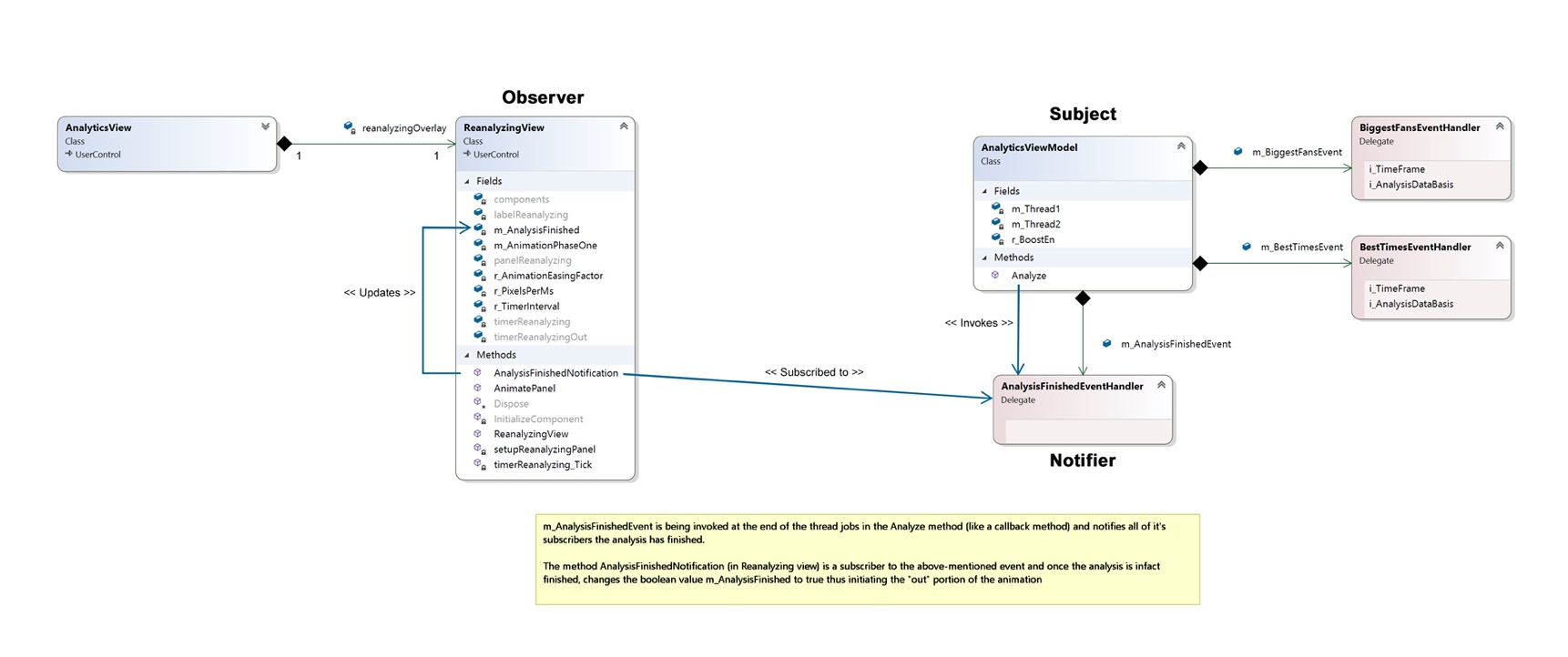
ViewModels🡺AnalyticsViewModel

Observer:

View🡺ReanalyzingView

* Sequence Diagram



* Class Diagram

### תבנית מס' 2 – Decorator

* סיבת הבחירה / שימוש בתבנית:

בחרנו בתבנית עיצוב זו לצורך מיון של ערכי ה-Value ב-Dictionaries בהסתכלות עתידית אל מקרים בהם נרצה להוסיף סוג נוסף של מיון, או מיון בתוך המיון הנוכחי, לאנליזות שלנו.

הסיבה לפיה לא בחרנו להשתמש בתבנית Proxy היא המקרה בו נרצה לעשות סינון לסינון קיים בצורה תחזוקתית.

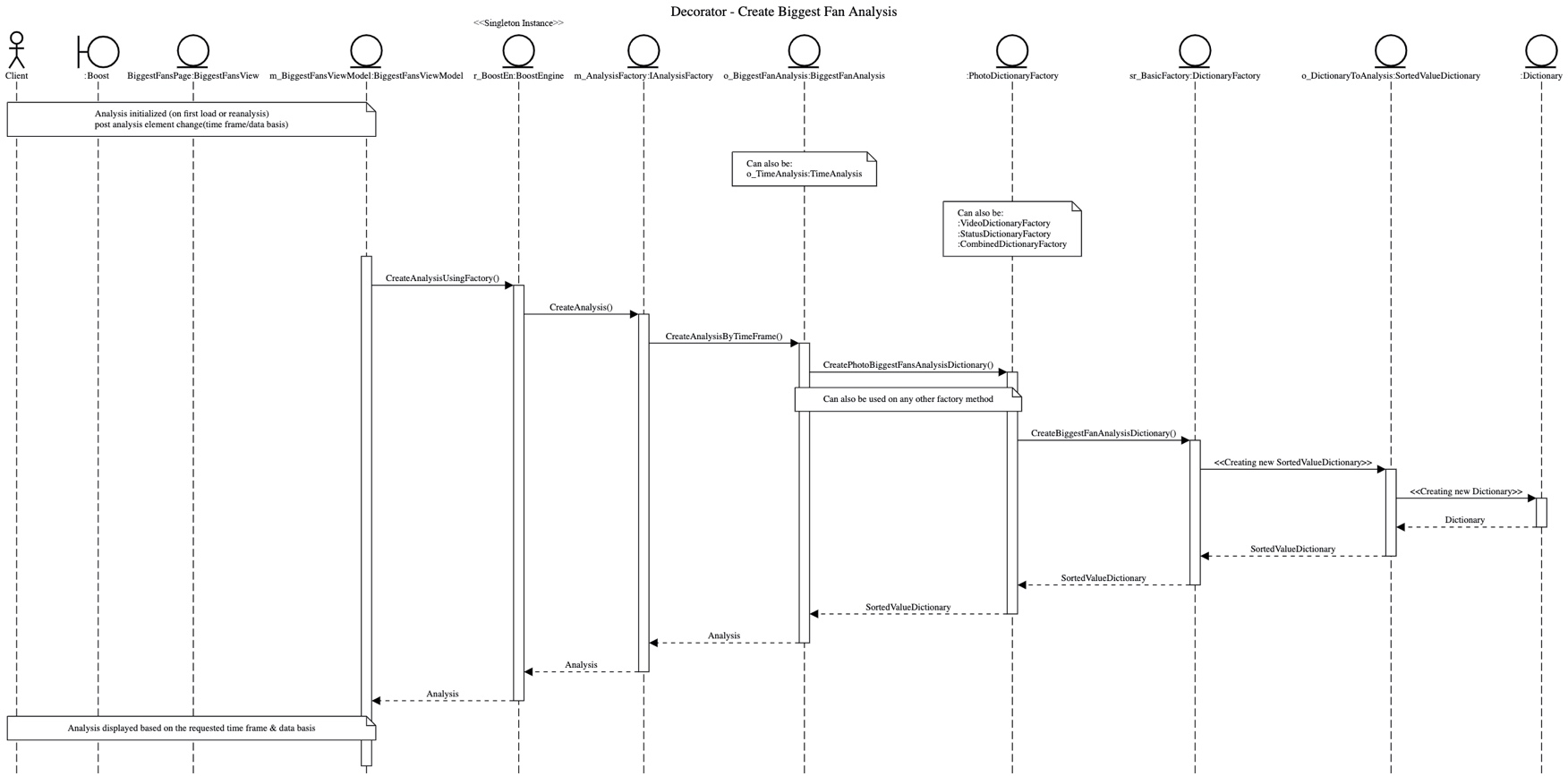
* אופן המימוש:

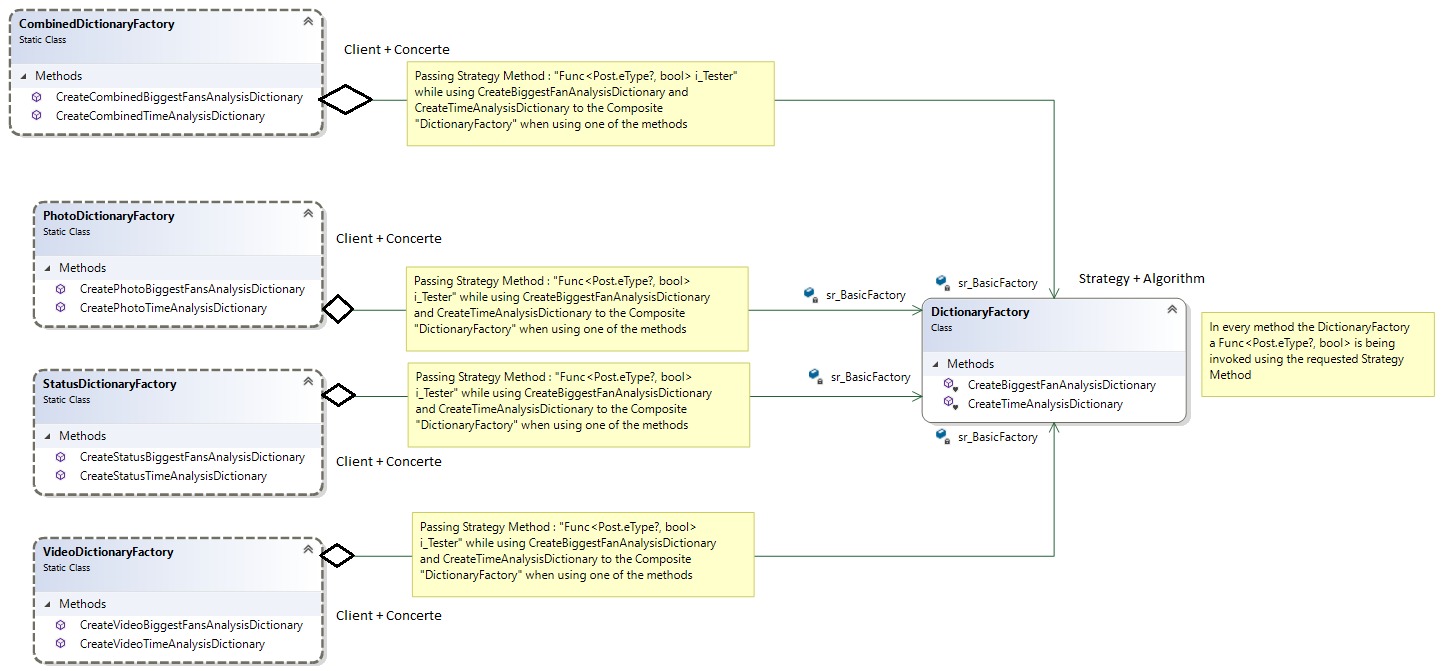
התבצע מימוש מהאינטרפייס IDictionary וקומפוזיציה אצל המחלקה המממשת (DictionaryDecorator), מחלקו זו מממשת את Dictionary בצורה גנרית. בנוסף, יצרנוConcrete Decorator בשםSortedValueDictionary אשר מממש את המיון לפי ערכי ה Value בהם חפצנו.

מיקום התבנית בקוד:

Model🡺Design Patterns🡺Decorator

* Sequence Diagram



* Class Diagram

### תבנית מס' 3 – Strategy

* סיבת הבחירה / שימוש בתבנית:

במהלך בניית ה- Factory Method נתקלנו בבעיה של פונטציאל שכפול קוד וחוסר תחזוקה עתידית של הקוד. על מנת להתגבר על כך מימשנו בעזרת הפיצ'ר מהשפה שנקרא בשיעור Strategy Method מימוש גנרי של המיון הדרוש לכל אחת מFactories המשתמשות במתודות CreateBiggestFanAnalysisDictionary או CreateTimeAnalysisDictionary .

סיבה נוספת היא שימוש חוזר עתידי של הקוד אם נרצה לקבל Dictionary הממויין בצורה אחרת.

* אופן המימוש:

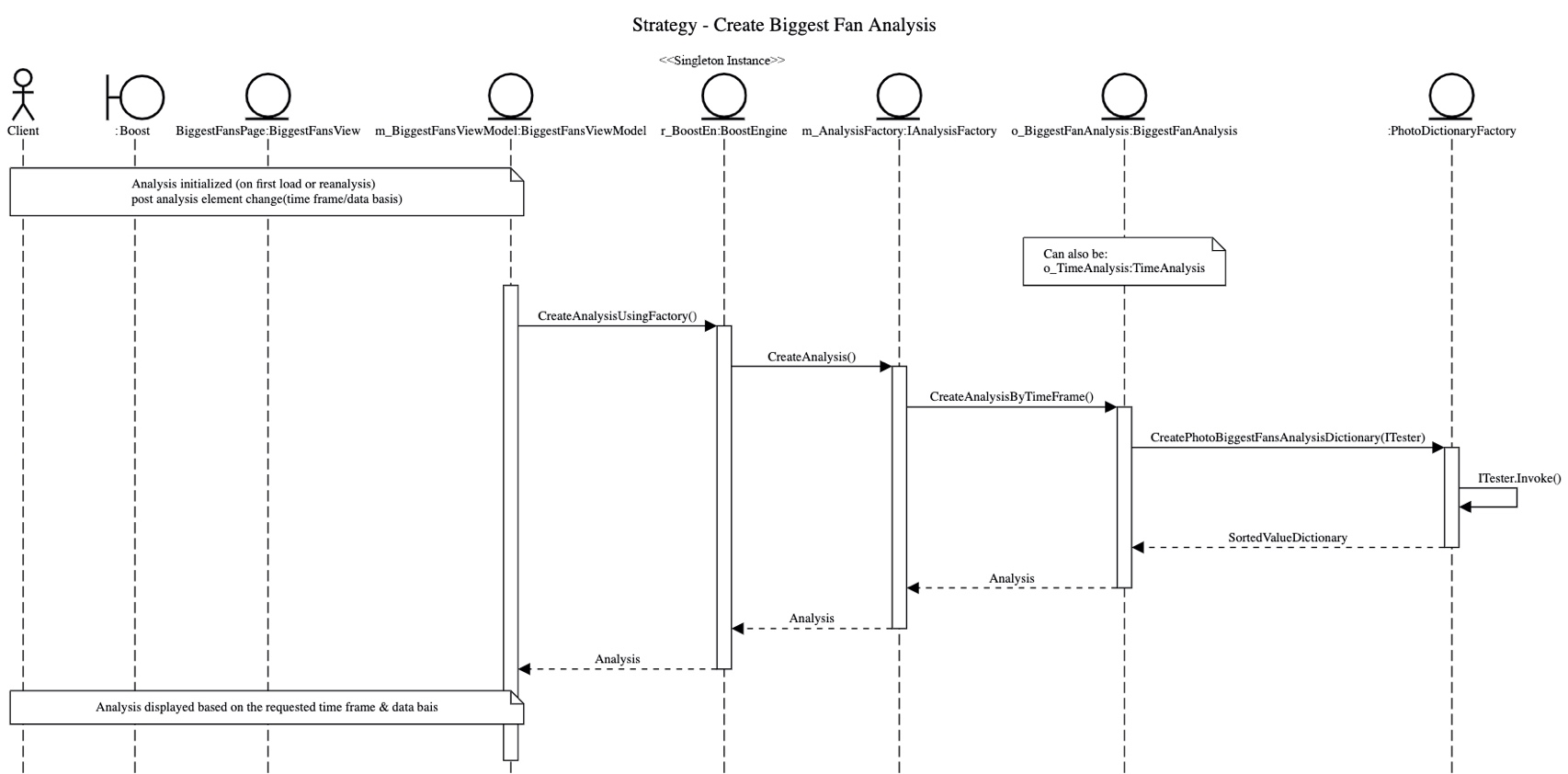
המימוש מבוצע על ידי שימוש במתודה המקבלת פרמטר מסוג Func<Post.eType?, bool> בתוכו מבוצע Invoke לאותה המתודה בה רצינו להשתמש הקריאה לאותה המתודה התבצעה על ידי שימוש בלמדה אקספרישן המקצר את תהליך בניית המתודה כיוון שיש לה רק שימוש אחד והוא יהיה רק בתוך הקריאה למתודה המבוקשת.

מיקום התבנית בקוד:

Model🡺Design Patterns🡺MethodFactory🡺 DictionaryFactory.cs

והקריאה לאותו מתודה מתבצעת בשאר הFactories באותה התיקייה

* Sequence Diagram



* Class Diagram

