תבניות עיצוב

DESIGN PATTERNS

שם המגיש: זאב מינדלי

Zeev mindali mindalizeev@gmail.com זאב מינדלי

תקציר

מהי תבנית עיצוב ?

- Design patterns בא אלינו מעולם תכנות מונחה עצמים, השיטות השונות ב DP מתארות מחלקות ואובייקטים המתקשרים ביניהם על מנת לפתור בעיה כללית בהקשר נתון.

איך ה תבניות עיצוב נוצרו?

- מניסיון טוב ולא טוב שהצטבר לאורך זמן.
- החלו לראות שלבעיות דומות יש פיתרון טוב דומה.

אילו יתרונות מקנה השימוש ב תבניות עיצוב?

- קל להשתמש שוב בעיצובים מצליחים.
- קל להפריד בין מה שמשתנה לבין מה שלא.
- קל לתעד ולתחזק את הקוד.
- עוזרים להגיע לעיצוב "נכון" מהר יותר.

ה Design Pattern מתחלקים לשלוש מחלקות:

- Creational – מתעסקת עם יצירת האובייקטים.
- Behavioral – מאפיינת את הדרכים בהן מחלקות או אובייקטים מושפעים הדדית ומחלקים את האחריות ביניהם.
- Structural – מטפלת עם ההרכבה של מחלקות או אובייקטים.

Zeev mindali mindalizeev@gmail.com זאב מינדלי

הקדמה

"תכנון תוכנה בגישה מונחית עצמים הוא קשה, ותכנון לשימוש מחדש בגישה מונחית עצמים הוא קשה אף יותר".

למרות קיומן של שיטות שונות לתיכון וניתוח תוכנה, מתכנתים בלתי מנוסים נתקלים עדיין באפשרויות רבות עצומות רבות מלבחור מבינם, בבואם לתכנן תוכנה .לעומת מתכנת בלתי מנוסה, מתכנת מנוסה שנתקל בבעיה, ינסה לבדוק איזה אלמנטים בבעיה, דומים לבעיות אחרות שהוא פתר, ויישם כמה מהחוקים שמניסיונו פתרו את האלמנטים בצורה נכונה.

תבניות מאפשרות לקחת חלק מהניסיון, והחוקים שמומחים הסיקו ולנסחם בצורה כללית, כך שניתן יהיה לזהות את האלמנטים החוזרים בבעיות בהקשרים שונים. כמובן שאין תחליף לניסיון טוב, אולם כוחם של התבניות הוא בצבירת הניסיון, לידי צורה המאפשרת צבירת ניסיון בדרך מהירה בהרבה.

מהי תבנית

Zeev mindali mindalizeev@gmail.com זאב מינדלי

לפי הגדרה הלקוחה מעולם הבנייה, כל תבנית מתארת בעיה, אשר מתרחשת שוב ושוב בסביבתנו, ומכילה בתוכה את הפתרון לאותה הבעיה, באופן כזה שנוכל להשתמש בפתרון לבעיה זו פעמים רבות מבלי לבצע את אותו המשימה פעמיים .

הגדרה זו הינה נכונה גם לתכנות מונחה עצמים, הפתרונות שלנו מבוטאים ע"י אובייקטים וממשקים במקום קירות ורעפים, במטרה ליצור מכלול שלם לפתרון בעיה גדולה יותר.

באופן כללי לתבנית יש ארבעה תכונות עיקריות:

1. **שם התבנית** - תכונה זו היא הבסיסית ביותר, אומנם היא נשמעת כי תכונה זניחה שאין להתייחס אליה ברצינות, ההפך הוא הנכון, המייסדים הראשונים של ה DP טוענים כי אחד האלמנטים החשובים ביותר של התבנית היא מתן שם נכון ומדויק המתאר את התבנית, כך אנחנו יוצרים מילון מונחים המשמש אותנו בעת שיחה עם חברינו לעבודה ,ובעת בניית ארכיטקטורת המערכת. עם זאת הם מציינים כי מציאת שם מתאים לתבניות השונות הייתה משימה לא פשוטה שלקחה הרבה מאוד שעות מזמנם.
2. **תאור הבעיה** - אומר לנו מתי להשתמש בתבנית, כמו כן היא מתארת את הבעיה שאנו רוצים לפתור לפרטי פרטים, זה יכול לכלול גם בעיות תכון ספציפיות כמו איך לייצג אלגוריתמים כאובייקטים, זה יכול לכלול גם הסברים על מבנה מחלקות או אובייקטים וכמו כן זה יכול לכלול רשימה של תנאים שהבעיה חייבת לעבור על מנת שזה יהיה הגיוני להשתמש בתבנית זו לשם פתרון לבעיה.
3. **הפתרון** – הפתרון הינו תיאור האלמנטים השונים של התיכון, הקשרים ביניהם ,האחריות של כל אחד מהם, ושיתוף הפעולה ביניהם. הפתרון אינו מכיל תיכון או אימפלמנטציה ממשית ספציפית, אלא מתאר את הפתרון בצורה יותר אבסטרקטית כך שסידור האלמנטים השונים יובילו לפתרון הבעיה.
4. **ההשלכות** – התוצאות והמכיר של שימוש בתבנית זו, אלמנט זה הוא קריטי לצורכי השוואה בין תבניות ובהחלטה שלנו האם שווה לנו להשתמש באותה תבנית.

Zeev mindali mindalizeev@gmail.com זאב מינדלי

כמו כן ישנם כמה בעיות יסודיות עם ההגדרה של תבנית, אנשים יכולים לפרש את המושג תבנית בדרכים שונות, ישנו אלמנט של הסתכלות, זאת אומרת מה שלי ניראה כתבנית יכול להראות למישהו אחר כ מבנה נתונים בסיסי. בנוסף תבניות גם תלויות בסביבת הפיתוח שלנו, לדוגמא תבנית שתוארה לסביבת פיתוח C++ יכולה להשתנות חלקית או אפילו בצורה משמעותית לסביבת פיתוח של c# או לJAVA, לכן עלינו להיות אף יותר מדויקים עם ההגדרות שלנו.

כאשר אנו מדברים על תבניות אנו לא מדברים על מבנים כמו רשימה מקושרת, התבניות שאנו מתייחסים אליהם הם תבניות של אובייקטים מסוימים המתקשרים אחד עם השני באופן מסוים כך שנותן לנו פתרון לבעיה כוללת.

מאפייני התבנית

Zeev mindali mindalizeev@gmail.com זאב מינדלי

בכדי להבין את הבניות לעומק קודם צריך לדעת כמה מאפיינים פורמלים של תבניות וכמה אלמנטים אשר נעשה בהם שימוש חוזר בין תבנית לתבנית. פירוק התבניות לאלמנטים ,ותיאורם בפורמט אחיד, יוכל לעזור לנו בכך.

ניתן לחלק תבניות לרכיבים היסודיים הבאים: שם התבניות, לתבנית צריך להיות שם משמעותי .

זה מאלץ אותנו לייחס מילה יחידה או צירוף מילים לתבנית ולידע והמבנה שהיא מתארת. שמות טובים תורמים לאוצר המילים, ומקדמים יכולת דיון על הפשטות שונות.

תיאור ה התבנית נעשה בתבנית מסוימת, כך שכל תבנית שתוסבר

תחולק בצורה הבאה:

- **<u>שם וסיווג התבנית</u>**

השם וסיווג התבנית מציגים בפנינו תבנית מופשטת ברמה גבוהה יותר. השם צריך לומר משהו על הבעיה, פתרונה וההשלכות במילה או שניים. בתוך כך, מתאפשר לנו להעשיר את עולם המושגים כך שניתן לקרוא לתבנית באחידות ולזהותה בוודאות .

- **<u>מה התבנית עושה בפועל</u>**

משפט קצר שיענה על השאלות הבאות: מה התבנית עושה? מה תכליתה של התבנית ? לאיזה בעיות תכנון התבנית עונה במיוחד?

- **<u>מוטיבציה</u>**

איך התבנית פותרת ומתאימה לבעיה? הדגמה באמצעות תרחיש מסוים של פתרון בעיה.

- **<u>מתי כדאי</u>**

באיזה מקרים כדאי ליישם את התבנית? כיצד נזהה מצבים כאלו? מספר כללים לבחירת התבנית מתוך הקטלוג של התבניות.

Zeev mindali mindalizeev@gmail.com זאב מינדלי

- **<u>תרשימים</u>**

תיאור המחלקות בתבנית וייצוגן ע"י TMO או UML או Sequence diagram

- **<u>משתתפים</u>**

המחלקות והאובייקטים שבתבנית ותפקידם. יחסי גומלין איך בא לידי ביטוי שיתוף הפעולה בין המשתתפים כך שכל משתתף. ימלא את תפקידו? בנוסף תיאור יחסי הגומלין באמצעות

- **<u>השלכות</u>**

איך התבנית ממלאת את תכליתה? מהם יחסי-הגומלין והתוצאות של השימוש בתבנית ? איזה היבט של המערכת יכול להשתנות ללא תלות במערכת ?

- **<u>מימוש</u>**

על איזה מלכודות, רעיונות וטכניקות עליך לדעת ביישום את התבנית ?

- **<u>שימושים ידועים</u>**

מקרים בהם ידוע שימוש של התבנית .

- **<u>תבניות קשורות</u>**

איזה תבניות קשורות לתבנית זו? מהם ההבדלים המשמעותיים?

Zeev mindali mindalizeev@gmail.com זאב מינדלי

תבניות ידועות שהוגדרו

אוסף התבניות מסווגות ל- 3 מחלקות ששמם מצביע על מטרתם. הסיווג יכול לעזור לנו למצוא תבניות דומות.

שמות הקבוצות הן:

- **תבניות יצירה**)**Creational patterns**(- עוסקות בתהליך יצירת אובייקט.
- **תבניות התנהגותיות**)**Behavioral patterns**(- מתארות את הדרכים בהם המחלקות או האובייקטים פועלים זה על זה, ואיך מתחלקת האחריות ביניהם ,
- **תבניות מבניות**)**Structural patterns**(- עוסקות בהרכב המחלקות והאובייקטים.

להלן טבלה מקובלת המחלקת התבניות לפי קבוצת השיוך שלהם ולפי ה Scope של התבנית)מחלקה או אובייקט(

	מטרה			
מרחב		יצירה	מבני	התנהגותי
	מחלקה	Factory Method	Adapter(c)	Interpreter Template Method
	אוביקט	Abstract Factory Builder Prototype Singleton	Adapter(o) Bridge Composite Decorator Façade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

טבלה 1.1)שיוך תבניות לקבוצות(

Zeev mindali mindalizeev@gmail.com זאב מינדלי

תבניות מסוג מחלקה מגדירות תבניות אשר עוסקות בקשרים בין מחלקות ובמחלקות יורשות שלהם .לעומת זאת תבניות מסוג אובייקט מייצגות תבניות אשר עוסקת בקשרים בין אובייקטים ,תבניות דינמיות שמשתנות תוך כדי זמן ריצה.

רשימת תבניות - תיאור כללי

Abstract factory - מספק ממשק ליצירת משפחות של אובייקטים דומים ,מבלי לציין את המחלקה הממשית שלהן.

Adapter - ממיר ממשק של מחלקה לממשק אחר ,מולו מצפה הלקוח לעבוד, תבנית זו מאפשרת למחלקות שלא יכלו לתקשר ביניהם את האפשרות לתקשר.

Bridge - מפריד בין הפשטה למימוש ,כך שיתכן ותיווצר שונות רבה ביניהם.

Builder - מפריד בין בניית אובייקט מורכב ובין ייצוגו ,כך שניתן לנצל את אותו תהליך בנייה לבניית מספר ייצוגים.

Chain of responsibility - מונע צימוד בין שולח הבקשה ובין מקבלה ,ובכך מאפשר ליותר מאובייקט אחד לטפל בבקשה .האובייקטים המטפלים נשמרים בשרשרת ,והבקשה מועברת לאורך השרשרת עד למציאת אובייקט שיטפל בה.

Command - עוטף בקשה כאובייקט ובכך מאפשר לקבוע מי יטפל בבקשה עפ"י פרמטרים או מידע אחר ,ואף מאפשר תמיכה בפעולות בלתי ישימות.

Composite - מאפשר הרכבת אובייקטים למבנה של עץ ע"מ לייצג היררכי ובנוסף עבודה אחידה עם אובייקטים בודדים או הרכבה שלהם.

Decorator - מחבר באופן דינאמי אחריות נוספת לאובייקט, מהווה אלטרנטיבה טובה ונוחה להורשה.

Interpreter - בהינתן שפה ,מגדיר את ייצוג התחביר שלה.

Flyweight - משתמש בשיתוף ע"מ לתמוך ביעילות בכמות גדולה של אובייקטים.

Factory Method - מגדיר ממשק ליצירת אובייקט ,אבל מניח למחלקות יורשות לקבוע מאילו .

מחלקות ייוצרו האובייקטים. נותן למחלקה לדחות את יצירת האובייקטים למחלקות יורשות.

Facade - מספק ממשק אחיד לקבוצה של ממשקים בתת מערכת.

Observer - מגדיר יחס של אחד לרבים בין אובייקטים ,כך ששינוי במצב האובייקט ישודר לאובייקטים התלויים בו ,והם יתעדכנו בהתאם.

Mediator - מגדיר אובייקט המסתיר כיצד קבוצה של אובייקטים מתקשרת, תורם לצימוד חלש ע"י כך שהאובייקטים לא פונים במפורש אחד לשני.

Memento - מבלי לפגועה בעיקרון ה- Encapsulation, מאפשר שמירת המצב הפנימי של האובייקט ,ע"מ שניתן יהיה לשחזרו בעתיד.

Singleton - מוודא כי נוצר מופע אחד בדיוק למחלקה ,ומספק נקודת גישה גלובלית לאובייקט.

Prototype - מציין את סוג האובייקטים אותם יש ליצור בהתאם לאובייקט קיים המהווה אב טיפוס ,ויוצר אובייקטים ע"י שכפול אב הטיפוס.

Proxy - מייצר תחליף לאובייקט ,דרכו ניתן לשלוט באובייקט.

Visitor - מייצג פעולה על חלקי אובייקט .ניתן לשנות את הפעולה בלא לשנות את המחלקות עליהן מבוססים חלקי האובייקט.

Template Method - מאפשר שמירת שלד של אלגוריתם כאשר חלק מצעדי האלגוריתם נותרים לביצוע ע"י מחלקות יורשות.

State - מאפשר לאובייקט לשנות את התנהגותו בהתאם לשינוי מצב)האובייקט יופיע כשייך למחלקה אחרת.(**Strategy** - מגדיר קבוצה של אלגוריתמים ,עוטף את כולם ומאפשר להם לתקשר ביניהם.

תבניות התנהגותיות

כאשר מפתחים מערכת גדולה ישנו קושי בניתוח אינטראקציה בין אובייקטים, כמו בארגון בעולם האמיתי, עולה הצורך להגדיר סמכויות ולאפיין את אופי התקשורת בין הישויות שבה. תבניות התנהגותיות, עוסקות בבעיות אלו, בדומה לארגון, ע"י הגדרת הסמכויות הנדרשת והתקשורת בין הישויות. ירושה והרכבה עוזרים לתבניות התנהגותיות בהגדרת היחסים בין המחלקות .

בירושה משתמשים לדוגמא ב Template method-ע"י הגדרת אלגוריתמים מופשטים המסתמכים על פעולות מופשטות. מחלקה יורשת מגדירה את פעולות אלו ע"י הגדרת הפעולות האלו. אולם בהרכבה משתמשים יותר מירושה בתבניות התנהגותיות. חלק מהתבניות ההתנהגותיות, מתארות איך קבוצה של אובייקטים שווי עוצמה, מתקשרים בכדי לבצע מטלה, שאובייקט פשוט אינו יכול לבצע לבד.

התבניות Mediator, Chain of responsibility, Observer, Strategy, Command, Visitor : מתארות דרכים להגשים מטרות אלו. בפרק זה נתאר בפירוט רב את התבניות: Strategy, Visitor,Observer.

Zeev mindali mindalizeev@gmail.com זאב מינדלי

Zeev mindali mindalizeev@gmail.com זאב מינדלי

Visitor

מה התבנית

ייצוג הפעולות אותן יש לבצע על האלמנטים המרכיבים אובייקט .Visitor מאפשר להגדיר פעולות חדשות ,מבלי לשנות את המחלקות של האלמנטים עליהן הן פועלות.

מתי כדאי?

- האובייקט מכיל מחלקות רבות עם ממשקים שונים ,ורוצים לבצע פעולות התלויות במחלקות המממשיות .
- כאשר יש צורך להפעיל פעולות שונות ובלתי תלויות על האלמנטים המרכיבים את האובייקט ,ורוצים להימנע מ"לכלוך" המחלקות עם פעולות אלו Visitor מאפשר לשמור על פעולות דומות יחדיו ,ע"י הגדרתם במחלקה אחת. כאשר מבנה האובייקט משותף למספר אפליקציות ,ניתן להשתמש ב-Visitor בכדי להכניס את הפעולות רק במקום בו הן נדרשות.
- המחלקות המרכיבות את האובייקט אינן משתנות אולם מתוספות להן פעולות חדשות באופן תדיר.

משתתפים

Visitor - מגדיר את הפעולה Visit עבור כל מחלקה ממשית של Element הקיימת במבנה האובייקט. שם וחתימת הפעולות מזהה את המחלקה השולחת את דרישת ה Visit ל Visitor. דבר המאפשר ל Visitor לקבוע את המחלקה הממשית של האלמנטים עליהם עובדים.

ConcreteVisitor - מממש את הפעולות המוגדרות ע"י Visitor כל פעולה מממשת חלק מהאלגוריתם המוגדר לקבוצת האובייקטים במבנה האובייקט .ConcreteVisitor נותן את ההקשר לאלגוריתם ושומר את המצב שלו) state(. מצב זה זה שומר ,לרוב ,את סך התוצאות שהתקבלו במעבר על האובייקט.

Element - מגדיר את הפעולה Accept המקבלת Visitor כפרמטר

Zeev mindali mindalizeev@gmail.com זאב מינדלי

ConcretElement - מממש את הפעולה Accept **ObjectStructure -** יכול למנות את האלמנטים שלו, יכול לתת ממשק המאפשר ל – Visitor לגשת לאלמנטים שלו, יכול להיות ~~Collection או Composite~~ כלשהו.

השלכות

- **יתרונות**
  1. הוספת אופרציות חדשות בקלות
  2. חלוקה למחיצות, ניתן להפריד התנהגויות שונות באופן מוחלט אחד מהשניה ע"י שימוש ב Visitor נפרדים.
- **חסרונות**
  1. חדש ConcreteElemtnt חדש קשה לביצוע, כל ConcreteElement הוספת מצריך אופרציה אבסטרקטית בתוך Visitor ואימפלמנטציה מתאימה בכל ConcreteVisitor . מחלקה מסוג
  2. הממשק של ConcreteElement חייב להיות חזק די כדי לתת ל visitors לעשות את עבודתם.

מימוש

ניתן למצוא מימוש של התבנית בשפות וסביבות פיתוח שונות באתר הבא:

http://www.dofactory.com/Patterns/PatternVisitor.aspx

Strategy

מה התבנית

הגדרת משפחה של אלגוריתמים ,עטיפת כל אחד מהם והפיכתם לברי החלפה Strategy מאפשר לאלגוריתמים להשתנות ,בלא קשר ללקוח המשתמש בהם.

<u>מוטיבציה</u>

לא נצטרך לקשור מחלקות לאלגוריתמים ספציפיים, דבר שיסרבל את התוכנית, אם נרצה להוריד ולהוסיף אלגוריתמים, או אם נרצה להחליף אלגוריתם קיים. לדוגמא, נרצה להחליף בזמנים שונים, אלגוריתמים שמפענחים טקסט לשורות, מכיוון שידוע לנו למשל, שפענוח טקסט בשיחה במסוף עם אדם אחר יעיל יותר מאלגוריתם המפענח טקסט של מסמך. אם נרצה להוסיף ולהוריד אלגוריתם אחר לתוכנית, נצטרך לציין את שמו מפורשות במחלקות המשתמשות בו. יתר על כן, השימוש יקשה עלינו יותר אם נרצה להחליף אלגוריתמים בזמן ריצה. לסיכום, שינוי האלגוריתם ידרוש שינוי חודרני אצל המשתמש, כך שהוא הופך להיות מודע לו.

<u>מתי כדאי?</u>

- הרבה מחלקות קשורות שונות רק בהתנהגותן ,Strategy מאפשר לקנפג מחלקה לעבודה עם מספר רב של התנהגויות
- האלגוריתם משתמש במידע שלא צריך להיות חשוף ללקוח.
- המחלקה מגדירה הרבה התנהגויות המופיעות כסדרה של תנאים בפעולות .במקום הרבה תנאים ,פשוט ניתן להעביר כל קטע קוד ל Strategy המתאים.

משתתפים Strategy - מגדיר ממשק משותף לכל

האלגוריתמים הנתמכים.

ConcreteStrategy - מספק מימוש של האלגוריתמים אחדים. בהתאם לממשק Strategy

Context - לקוח של ConcreteStrategy בזמן ריצה ,ההתייחסות היא לאובייקט מסוג. Strategy

מימוש

ניתן למצוא מימוש של התבנית בשפות וסביבות פיתוח שונות באתר הבא:

http://www.dofactory.com/Patterns/PatternStrategy.aspx

יחסי גומלין Strategy ו- Context מממשים ביחד את האלגוריתם הנבחר. Context מעביר ל-Strategy את המידע הדרוש. Context מעביר את הבקשות מהמשתמש ל-Strategy, כך משמש Context מעין מתווך שרק המשתמש ניגש אליו. המשתמש בד"כ יוצר את Concrete Strategy ומעבירו ל-Context, כך שהמשתמש יכול לבחור Concrete Strategy מתוך משפחת ה- Strategy.

Zeev mindali mindalizeev@gmail.com זאב מינדלי

Observer

מה התבנית

הגדרת קשר יחיד-רבים בין אובייקטים כך שכשאובייקט אחד משנה מצב, כל האובייקטים התלויים בו מקבלים הודעה על כך ומעדכנים את מצב.

<u>מוטיבציה</u>

נרצה לשמור על עקביות בין אובייקטים קשורים, כגון מידע וצורותיו הויזואליות כפי שמתואר

<u>מתי כדאי?</u>

- כאשר להפשטה יש שני היבטים, אחד תלוי בשני עצמים נפרדים מקלים על שינוי ושימוש חוזר.
- כאשר שינוי באובייקט אחד דורש שינוי במספר לא ידוע של עצמים.
- כאשר אובייקט יכול להודיע לעצמים אחרים בלי להניח משהו עליהם.

<u>משתתפים</u>

Subject - שומר רשימה של observers בעל פעולות של הוספה והסרת Observers וכמו כן פעולת Notify העוברת על רשימת ה Observers ומיידעת

Zeev mindali mindalizeev@gmail.com זאב מינדלי

אותם על שינוי מסויים.

Observer – מייצא פעולה פשוטה ש עדכון.

Concrete Subject – כאשר Notify שומר מצב קורא ל Subject יורש מ

המצב משתנה.

ConcreteSubject – ממש את פעולת העדכון.

1. תמיכה בהודעות הפצה לנמענים רבים. להודעה אין נמען מיוחד שאליו מיועדת ההודעה. ההודעה מתקבלת אצל כל האובייקטים שביקשו זאת, ולכן מספר האובייקטים יכול להשתנות בזמן ריצה.

2. הודעות בלתי צפויות, העלולות לגרום לעדכונים רבים ומיותרים של האובייקטים התלויים. יתר על כן, האובייקטים התלויים אינם מודעים למחיר העדכון האסינכרוני שלהם .כאשר יש עדכונים רבים, קשה לעקוב אחר מקורם, מכיוון ש subject - אינו יודע מי מבקש לעדכנו.

מימוש

ניתן למצוא מימושים בשפות שונות של תבנית ה Observer ב - http://www.dofactory.com/patterns/patternobserver.aspx#_self1

תבניות קשורות

1. Mediator - כאשר ישנם הרבה מערכות עדכונים ה ChangeManager יכול לשמש Observer . ל Subject ככלי מקשר בין ה
2. SingleTon – אפשר לעשות את מנהל השינויים כ Singleton כך שיהיה יחיד וייחודי.

Zeev mindali mindalizeev@gmail.com זאב מינדלי

תבניות מבניות

תבניות מבניות) Structural patterns(הם תבניות שמתרכזות בעיקר ביצירה של מחלקות ואובייקטים כל שייצרו מבנים יותר גדולים, תבניות מבניות משתמשים בהורשה על מנת ליצור ממשקים לדוגמא

Adapter

מהי התבנית עושה בפועל

ממיר ממשק של מחלקה לממשק אחר שהמשתמש מצפה לו ה Adapter גורם למחלקות שלא יכלו לעבוד אחד עם השני עקב ממשקים לא תואמים את האפשרות לעבוד. ה Adapter ידוע גם בתור Wrapper.

מוטיבציה

לעתים,מחלקה שהוגדרה גנרית במטרה שיעשה בה שימוש במקרים רבים, אינו ניתן לשימוש במקרה מסוים רק מכיוון שהממשק שלו לא עונה על דרישות הממשק הספציפיות שהאפליקציה דורשת.

ניקח לדוגמה drawing editor שבו המשתמש יכול לצייר ולסדר אלמנטים גרפיים)קוים ,פוליגונים, טקסט וכו'(וליצור ציורים ודיאגרמות .האבסטרקציה הבסיסית של ה editor היא אובייקט גרפי, בעל צורה ניתנת לשינוי ואשר יכול לצייר את עצמו. הממשק לאובייקט הגרפי מוגדר על ידי מחלקה אבסטרקטית שנקרא "Shape" ה editor מגדיר subclass של Shape לכל סוג של אלמנט גרפי LineShape :עבור קו PolygonShape ,עבור פוליגון, וכן הלאה.

ה subclass ים עבור אלמנטים גיאומטריים בסיסיים כמו קו ופוליגון הם קלים למימוש, מכיוון שהאפשרויות לצייר אותם ולערוך אותם הן מוגבלות מטבען. אבל ה subclass שמציג טקסט

Zeev mindali mindalizeev@gmail.com זאב מינדלי

, ,TextShapeהוא מסובך יותר למימוש באופן משמעותי ,מכיוון שאפילו עריכת טקסט בסיסית דורשת ניהול buffer ועדכון מתוחכם של המסך. במקרה זה נשמח כמובן לקבל מחלקה גנרית מוכנה עבור טקסט שכבר נכתב על ידי מישהו ומבצע פונקציונליות של הצגת ועריכת טקסט. נניח שקיים כזה ונקרא לו לצורך העניין .TextView היינו רוצים להשתמש בו ישירות בתור TextShape אבל בדוגמא שלנו מי שתכנן אותו לא חשב על Shapes באותו רגע, ולכן הוא אינו תואם למחלקה Shape כרגע .

איך בכל זאת נוכל להשתמש ב מחלקות מוכנות כ TextView באפליקציות שמצפות למחלקה עם ממשק שונה לגמרי ?

הפתרון:

הצעה לפתרון **גרוע**: נשנה את TextView כך שיהיה מסוג Shape , אם אין לנו את קוד המקור שבו TextView נכתב ?יותר מכך, אם בעוד חודש נצטרך אותו לשימוש באפליקציה אחרת לא נרצה לשנות אותו שוב

פתרון **טוב**: נוכל להגדיר את TextShape כך שהוא *יתאים את עצמו* לשימוש ב מחלקה המוכנה TextView זאת נוכל לעשות בשתי דרכים:

1.נבצע שימוש בהורשה מרובה ונירש את הממשק שלו מ Shape ואת המימוש שלו מ TextView.

2. נבצע הרכבה (composition) של אובייקט TextView בתוך אובייקט TextShape ונממש את TextShape במונחים של הממשק של TextView כלומר ע"י קריאות לפעולות של האובייקט TextView שנמצא בתוכו.

לעתים קרובות ה adapter גם אחראי לפונקציונליות שהמחלקה המותאמת (adapted) אינה מספקת .הדיאגרמה מראה כיצד זה מתבצע. למשתמש ישנה אפשרות לגרור (drag) צורה שצייר למיקום חדש TextView .אינו מתוכנן לכך כי הוא יודע רק לטפל בטקסט עצמו .לכן TextShapeמוסיף את הפונקציונליות הזו על ידי מימוש של פעולת

()CreateManipulatorשמייצרת מופע של ה subclass המתאים של Manipulator אשר יודע לבצע פעולת גרירה .

Manipulator הוא abstract class עבור אובייקטים שיודעים כיצד לשנות צורה של אלמנט גרפי בהתאם לקלט מהמשתמש, כמו למשל, גרירה למיקום חדש. לכל צורה יש subclass מתאים לה של .Manipulator כך הוספנו פונקציונליות ש TextView לא מספק אך ה editor דורש אותה .

<u>מתי כדאי?</u>

- כאשר רוצים להשתמש במחלקה שהממשק שלה לא מתאים לזה הנדרש.
- כאשר רוצים ליצור מחלקות לשימוש חוזר אשר משתפות פעולה עם מחלקות לא שייכות או לא ידועות,כלומר לא בהכרח בעלות ממשק מתאים. בלבד (יש צורך בשימוש במספר מחלקות יורשות ,אבל לא יעיל לרשת כ"א מהן ע"מ Object Adapter(
- יכול להתאים את עצמו לממשק של מחלקת האב Adapter .אובייקט Adapter. ליצור עבורן

<u>משתתפים</u>Target - מגדיר את הממשק בו

משתמש הלקוח.

Client - משתף פעולה עם אובייקטים המקיימים את הממשקTarget

Adaptee - מגדיר ממשק קיים לו יש לבצע התאמה.

target. לזה של Adaptee מתאים את הממשק של**Adapter -**

Zeev mindali mindalizeev@gmail.com זאב מינדלי

Composite

מהי התבנית עושה בפועל

להרכיב אובייקטים למבני עץ שמייצגים את ההיררכיה של היחס חלק-שלם .לאפשר ללקוחות לטפל באובייקטים בודדים ובהרכבות באופן אחיד .)תבנית מבנה(.

דוגמא :עצמים גרפיים.

מוטיבציה

בעולם התוכנה ניתן לתאר הרבה תהליכים או נתונים בתור עץ, תבנית זו מאפשרת לנו לממש מבנה של עץ בצורה חכמה ונוחה לשימוש.

מתי כדאי?

כאשר רוצים לממש יחס חלק - שלם. כאשר רוצים לאפשר ללקוחות להתעלם מהבדלים בין אובייקטים מורכבים לאובייקטים בודדים.

משתתפים

Component - מגדיר את הממשק עבור אובייקטים בהרכבה.מממש את התנהגות ברירת המחדל עבור הפעולות המשותפות לכל המחלקות.מגדיר ממשק לגישה וניהול תתי העצים.

אופציונאלי (מגדיר ממשק לגישה לאב ,וממש אותו) אם יש צורך בכך **Leaf** - מייצג אובייקט מסוג עלה בהרכבה. מגדיר את ההתנהגות של אובייקטים פרימיטיביים בהרכבה.

Composition **-** מגדיר את ההתנהגות של אובייקטים בעלי ילדים. שומר את הילדים וממשמש את הפעולות הנגזרות מכך.

Client - מפעיל את האובייקטים בהרכבה תוך שימוש בממשק של Composition

יחסי גומלין

לקוחות משתמשים במחלקת COMPONENT על מנת לתקשר עם אובייקטים מסוג Composite. אם הנמען הוא עלה) Leaf(אז מעבדים את הבקשה באופן ישיר, אם נמען הוא מסוג Composite אז הבקשה בדרך כלל עוברת לילדיו. ישנה אפשרות לעשות פעולות מסויימות לפני ואחרי העברת בקשות לילדים מסוג Component.

השלכות

- יכולה להיווצר בעיה של רקורסיה כשקליינט מצפה לאובייקט פרימיטיבי הוא גם יכול לקבל אובייקט מסוג Composite.
- מקל על הוספת קומפוננטות חדשות, החיסרון בכל שקשה להגביל את הקומפוננטות של Composite. לפעמים נרצה ש Composite יכיל רק קומפוננטות מסויימות.

מימוש

ניתן למצוא מימוש של התבנית בשפות וסביבות פיתוח שונות באתר :

הבא

http://www.dofactory.com/Patterns/PatternComposite.aspx

תבניות יצירה

תבניות יצירה, לפי שמם עוסקות ביצירת אובייקטים ובאתחולם, הם עוזרים למערכת להיות עצמאית מבחינת יצירה, הרכבה, וייצוג האובייקטים שלה. תבנית יצירה מחלקתית תשתמש בהורשה על מנת ליצור מופע של המחלקה אותה היא מייצגת ולעומת זאת תבנית יצירה מסוג אובייקט תעשה שימוש ב Delegation על מנת ליצור מופעים של אובייקטים. בפרק זה נדגים שתי סוגים של תבניות יצירה **Singleton** ו **AbStractFactory**

Singleton

מהי התבנית עושה בפועל

מוודא שיהיה מופע אחד לאותו אובייקט הממשמש את התבנית וכמו כן מספק גישה גלובלית לאובייקט

מוטיבציה

זה חשוב שתהיה את האפשרות להגביל אובייקטים למופע חיי אחד, לדוגמא כתבנו תוכנה קליינט שמתחבר לשרת מסוים, אנו רוצים לוודא שבמהלך הקישוריות של התוכנה לשרת לעולם לא ייווצר connection נוסף) בעיה נפוצה בקרב תכנות עם Connections(. אם נגדיר את ה connection כ Singleton הדבר יבטיח לנו שלא יהיה לנו Connection זומבי)שלא נהרס או שנוצר בטעות ולא משתמשים בו.(

מתי כדאי?

- כאשר אנו רוצים שיהיה בדיוק מופע אחד של האובייקט, ושיהיה לו גישה גלובלית ידועה **משתתפים**

- Singleton מגדיר אופרציית instance שמקנה למשתמשים גישה ל מופע היחיד של האובייקט

- אחראי ל יצירה ולשימור המופע שלו.

שיתוף פעולה

משתמשים ניגשים למופע של Singleton אך ורק דרך אופרציית instance של Singleton.

מימוש

ניתן למצוא מימושים שונים לתבנית באתר הבא:

http://www.dofactory.com/Patterns/PatternSingleton.aspx

Abstract factory

מהי התבנית עושה בפועל

לספק ממשק ליצירת משפחות של אובייקטים שקרובים זה לזה, או קשורים באופן מסוים, או תלויים זה בזה, מבלי לפרט את ה מחלקה הממשית שלהם.

מוטיבציה

ניקח לדוגמא אפליקציה מסוימת שתומכת במספר אפשרויות של ממשק גרפי כמו למשל Presentation manager ו Motif. הסטנדרטים השונים של הממשקים מביאים לכך שכפתורים ,חלונות, פסי הגלילה וכו' נראים ומתנהגים אחרת בכל אחד מהם .לא נרצה לכתוב עותקים של האפליקציה לכל אפשרות גרפית שקיים בעולם.

כדי להיות פורטביליים לסוגים השונים של הממשקים הגרפיים הקיימים ושל אלה שיתווספו ,האפליקציה לא תפרט בקוד אילו אובייקטים בדיוק היא תיצור עבור כפתור, פס גלילה וכו' עבור כל סטנדרט מסוים .אם היא הייתה עושה כן זה היה מקשה לשנות את הקוד עבור כל סטנדרט מכיוון שהיינו צריכים לעבור בכל פעם על כל הקוד ולמצוא את כל המקומות שאובייקטים אלה מופיעים.

מתי כדאי?

- כאשר אנו רוצים שהמערכת תהיה חופשית מבחינת ייצור, ייצוג והרכבת האובייקטים שלה.
- כאשר משפחות של אובייקטים שקשורים אחד לשני מעוצבים לעבוד ביחד
- אנחנו רוצים לספק ספריה למשתמש ולא לחשוף את האימפלמנטציה אלא רק את הממשקים

משתתפים **AbstractFactory** - מגדיר ממשק עבור פעולות שיוצרות אובייקטים אבסטרקטיים של.products

ConcreteFactory - מממשים את הפעולות של abstract factory כדי ליצור אובייקטים מוחשיים של product.

Abstract - מגדירים ממשק לאובייקט מסוג מסויים של Products

Products

Concrete Product - המתאים con crete factory שיווצר על ידי ה product מגדירים אובייקט מוחשי של Abstract Product. מממש את הממשק של

Client - Abstract Product. ו Abstract Factory משתמש רק בממשק שהוגדר על ידי המחלקות

<u>יחסי גומלין</u>

בדרך כלל נוצר מופע אחד של ConcreteFactory בזמן ריצה. הוא יוצר אובייקטים מסוג productבעלי מימוש מסוים. כדי ליצור products אחרים נצטרך ConcreteFactory אחר .

ה AbstractFactory דוחה את שלב יצירת האובייקטים ל concrete classes שיורשים ממנו.