

# תכנות מונחה עצמים בשפת C++

אוהד ברזילי  
אוניברסיטת תל אביב

# תבניות תיכון (Design Patterns)

המצגת מבוססת על מצגות של פרופ' סיוון טולדו ופרופ' עמירם יהודאי על פי הספר:

Design Patterns: Elements of Reusable Object-Oriented Software  
By Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides

כל הזכויות שמורות למחברים

## מוטיבציה

- בחיי יום יום אנחנו מתארים דברים תוך שימוש בתבניות חוזרות:
- מכונית א' היא כמו מכונית ב', אבל יש לה 2 דלתות במקום 4.
- אני רוצה ארון כמו זה, אבל עם דלתות במקום מגרות.
- גם בפיתוח תוכנה, אנחנו יכולים להסביר כיצד לעשות משהו ע"י התייחסות לדברים שעשינו בעבר, ובצורה כזאת להקל על התקשורת עם עמיתים.
- נאחסן את המבנה בעץ בינרי, ונבצע חיפוש לרוחב.

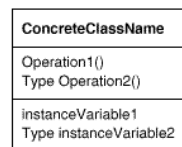
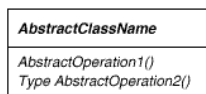
## הגדרות מהספרות

- "Design Patterns are recurring solutions to design problems you see over and over." [Alpert, Brown, Wof, 1998].
- "Design Patterns constitute a set of rules describing how to accomplish certain tasks in the realm of software development." [Pree, 1994].
- "A pattern address a recurring design problem that arises in specific design situations and presents a solution to it." [Buschmann et al, 1996].
- "Patterns identify and specify abstractions that are above the level of single classes and instances, or of components." [Gamma et al, 1993].

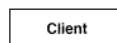
## מהי תבנית תיכון?

- תבנית תיכון היא פתרון מקובל לבעיית תיכון נפוצה בתכנות מונחה עצמים.
- תבנית תיכון מתארת כיצד לבנות מחלקות כדי לענות על הדרישה הנתונה.
- מספקת מבנה כללי שיש להשתמש בו כשממשים חלק מתכנית.
- לא מתארת את המבנה של כל המערכת.
- לא מתארת אלגוריתמים ספציפיים.
- מתמקדת בקשר בין מחלקות.
- מתארת נסיון מצטבר של מתכנתים, שניתן ללמוד ועוזר לתקשורת בין מהנדסי תוכנה.

## תרשימי מחלקות

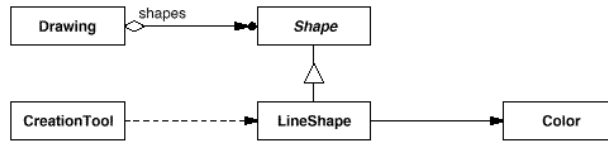


(a) Abstract and concrete classes



(b) Participant Client class (left) and implicit Client class (right)

## תרשימי מחלקות



(c) Class relationships

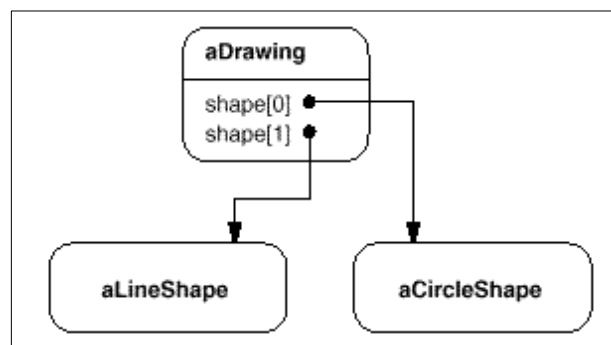


(d) Pseudocode annotation

תכנת מונחה עצמים בשפת C++  
אוניברסיטת תל אביב

7

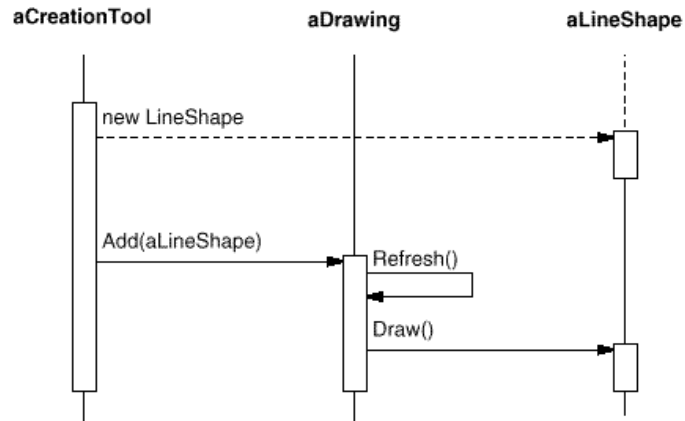
## תרשימי עצמים



תכנת מונחה עצמים בשפת C++  
אוניברסיטת תל אביב

8

## תרשים סדרתי (sequence diagram)



תכנות מונחה עצמים בשפת C++  
אוניברסיטת תל אביב

9

## מקורות

- המושג נעשה פופולרי בעקבות הספר שמכונה GoF (ראה בשקופית הבאה)
- הגישה אומצה מעבודותיו של כריסטופר אלכסנדר, מתחום ארכיטקטורה של מבנים. הוא כתב:

"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice"

תכנות מונחה עצמים בשפת C++  
אוניברסיטת תל אביב

10

# GoF

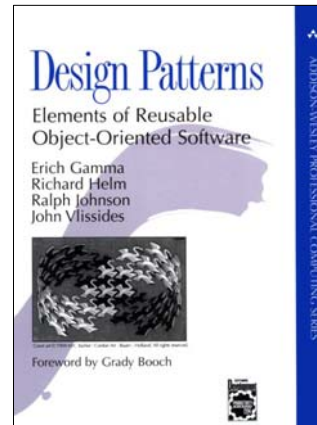
## Design Patterns: Elements of Reusable Object-Oriented Software

By Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.

Published by Addison Wesley Professional.

Series: Addison-Wesley Professional Computing Series.

ISBN: 0201633612; Published: Oct 31, 1994; Copyright 1995; Dimensions 7-3/8x9-1/4 ; Pages: 416; Edition: 1st.



# רקע

- אבחנותיו של אלכסנדר רלבנטיות גם למערכות תוכנה.
- במקום לדבר על קירות ודלתות, אנו עוסקים בעצמים ומנשקים.
- נעשה מאמץ לקטלג תבניות תיכון כלליות, וגם תבניות שמתאימות לתחומי יישום מוגדרים.
- הקריטריון לקבלת תבנית תיכון – נעשה בה שימוש במספר יישומים אמיתיים.
- סוגים נוספים של תבניות, לתיעוד דרכים טובות לפתור בעיות מסוגים שונים (כולל למשל ניהול כוח אדם). Best Practice.
- אנטי תבניות (Anti patterns) דברים שיש להימנע מהם.

## מרכיבים עיקריים של תבנית תיכון

ארבעה מרכיבים חיוניים:

- **שם התבנית.** שימוש בשם אחיד ומקובל מסייע לתקשורת בין מהנדסי תוכנה, מעלה את רמת ההפשטה.
- **הבעייה.** מתי ניתן להשתמש בתבנית? איזה בעייה היא אמורה לפתור? יכול לכלול מבנה עצמים שהוא סימפטום לבעייה, או אוסף תנאים שצריכים להתקיים.
- **הפתרון.** מתאר את מרכיבי הפתרון, היחסים ביניהם, אחריות כל אחד ושיתופי פעולה. תבנית לפתרון, ולא מימוש קונקרטי.
- **השלכות.** תוצאות וקשרי גומלין (trade-offs) של שימוש בתבנית. חשוב לצורך הערכה של חלופות תיכון והבנת העלות והתועלת של התבנית. השלכות על יעילות, גמישות, יכולת הרחבה ויבילות (portability).

## מושגים שקשורים לתבניות תיכון

- תבניות תיכון מטפלות ברמת ביניים בין שני סוגי התבניות הבאים (עפ"י Buschmann and al):
- תבניות ארכיטקטוניות: צורת ארגון בסיסית של מערכת תוכנה שלמה. מספקת אוסף תתי מערכות קבועות מראש, חלוקת אחריות, כללים והנחיות לארגון היחסים ביניהן.
  - תבניות קידוד, או idioms. תבנית ברמה נמוכה, ספציפית לשפת תכנות מסוימת. מתאר איך לממש היבט מסוים של רכיבים ויחסים ביניהם בעזרת המבנים ששפת התכנות מספקת.
- תבניות תיכון אמורות להיות בלתי תלויות בשפה, אבל התבניות של GoF (וגם אחרות) מתייחסות במיוחד לשפות מונחות עצמים (ופרט ב C++).

## תבניות תיכון | Frameworks

Design patterns focus more on reuse of recurring architectural design themes, while frameworks focus on detailed design ... and implementation." [Coplien, Schmidt, 1995]

- Framework (OO Software) הוא אוסף מחלקות קשורות זו לזו, שניתן להרחיב ו/או ליצור מופעים, כדי לממש יישום.
- תבנית תיכון מגדירה ארכיטקטורת תוכנה ניתנת לשימוש חוזר, שמספקת מבנה והתנהגות כלליים של משפחה של יישומים ממוחשבים.

## תבניות תיכון | Frameworks

- מספק מידע נוסף על שיתופי הפעולה בין המחלקות והשימוש בהן בתחום יישום מסוים.
- תבנית אינה יישום מלא שניתן להריץ, כי חסרים חלקים שיש להוסיף או להחליף כדי לקבל את הפונקציונליות הדרושה.
- תבניות תיכון יכולות לתאר את הקשרים בין מחלקות ב Framework
- Framework יכול להשתמש במספר תבניות תיכון



## סיווג תבניות

- הספר של GoF מציג 23 תבניות שמחולקות לשלוש משפחות לפי המטרה שלהן:
  - תבניות ליצירה Creational: נוגעות לתהליך היצירה של עצמים.
  - תבניות מבנה Structural: עוסקות בהרכבה של מחלקות ועצמים.
  - תבניות התנהגות Behavioral: מאפיינות את הדרכים בהן מחלקות ועצמים מתקשרים ומחלקים אחריות.
- קלסיפיקציה נוספת מתייחסת לתחום העיסוק של תבנית – מחלקות או עצמים.
- בנוסף, ניתן להתייחס לקבוצות של תבניות שמופיעים בדרך כלל ביחד, או כאלה שמהווים חלופות שונות לפתרון בעיות דומות.

## סיווג תבניות

|       |        | Purpose   |  |  |
|-------|--------|---|--|--|
|       |        | Creational  | Structural   | Behavioral   |
| Scope | Class  | <a href="#">Factory Method (107)</a>  | <a href="#">Adapter (139)</a>  | <a href="#">Interpreter (243)</a><br><a href="#">Template Method (325)</a>   |
|       | Object | <a href="#">Abstract Factory (87)</a><br><a href="#">Builder (97)</a><br><a href="#">Prototype (117)</a><br><a href="#">Singleton (127)</a> | <a href="#">Adapter (139)</a><br><a href="#">Bridge (151)</a><br><a href="#">Composite (163)</a><br><a href="#">Decorator (175)</a><br><a href="#">Facade (185)</a><br><a href="#">Proxy (207)</a> | <a href="#">Chain of Responsibility (223)</a><br><a href="#">Command (233)</a><br><a href="#">Iterator (257)</a><br><a href="#">Mediator (273)</a><br><a href="#">Memento (283)</a><br><a href="#">Flyweight (195)</a><br><a href="#">Observer (293)</a><br><a href="#">State (305)</a><br><a href="#">Strategy (315)</a><br><a href="#">Visitor (331)</a> |

## תבניות יצירה

- Abstract Factory – יצירה של משפחות של מחלקות ללא ציון המחלקה הקונקרטית
  - לדוגמא: חלון ופס גלילה של יצרן מסוים
- Builder – הפרדת תהליך יצירה של עצם מורכב מהייצוג שלו, כך שניתן לחזור על אותו תהליך יצירה עבור עצמים ממחלקות שונות
  - לדוגמא: המרת מסמך מפורמט אחד למשנהו

## תבניות יצירה

- Factory Method – הגדרת ממשק ליצירת עצמים תוך מתן אפשרות למחלקות נגזרת לקבוע את טיפוס העצם הנוצר
  - לדוגמא: היררכיה מקבילה של יישום ומסמך שהוא יוצר
- Prototype – יצירת עצמים חדשים ע"י שיבוט עצם קיים
  - לדוגמא: כפתור היוצר עצמים מוסיקליים

## תבניות יצירה

- Singleton – יצירת מחלקה עם מופע יחיד
- לדוגמא: Abstract Factory או שרון מערכת

## תבניות מבנה

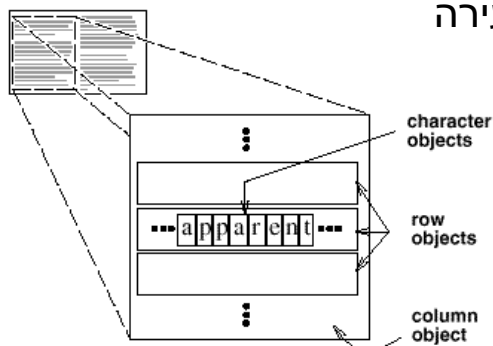
- Adapter – התאמת ממשק של מחלקה לממשק שלו מצפים הלקוחות שלה
- לדוגמא: מחלקה להצגת טקסט המעוניינת להיות גם תת מחלקה של Shape כדי להיות מוצגת על המסך וגם להשתמש בלוגיקה המוגדרת ב TextView
- Bridge – מבטל את התלות בין ממשק ובין מימוש, המאפשר לכל אחד מהם מתפתח בצורה עצמאית
- פיתוח היררכיה של מימושי חלונות במקביל לפיתוח היררכיה של סוגי חלונות

## תבניות מבנה

- Composite – הצגת עצמים במבנה עץ ומתן ממשק אחיד לעצמים פשוטים ולעצמים מורכבים
  - לדוגמא: עצמים גרפיים פרימיטיביים כגון: קווים, מלבנים ומלל ומורכבים כגון: תמונה
- Proxy – יוצר פונדקאי או שומר מקום לעצם כדי לפקח על הגישה אליו
  - לדוגמא: תמונות "כבדות" במסמך, מצביעים חכמים

## תבניות מבנה

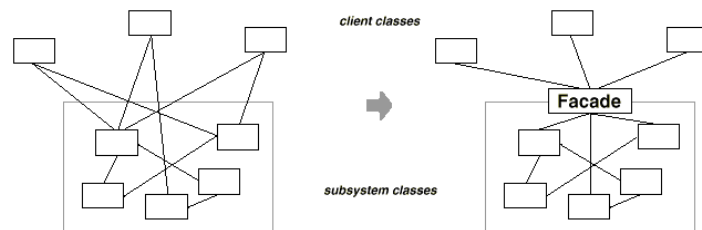
- Flyweight – מאפשר שיתוף יעיל של עצמים מרובים בגרעיניות זעירה
  - לדוגמא: עצם לכל תו



## תבניות מבנה

■ Facade – מתן ממשק אחיד לאוסף ממשקים בתוך תת מערכת. יוצר ממשק ברמת הרכיב ומפשט את השימוש בתת המערכת

■ לדוגמא: קומפיילר



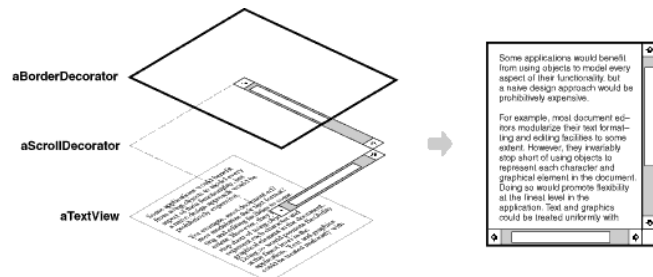
תכנות מונחה עצמים בשפת C++  
אוניברסיטת תל אביב

25

## תבניות מבנה

■ Decorator – הוספה דינאמית של תכונות שלא על ידי ירושה

■ לדוגמא: מסמך עם מסגרת ופס גלילה



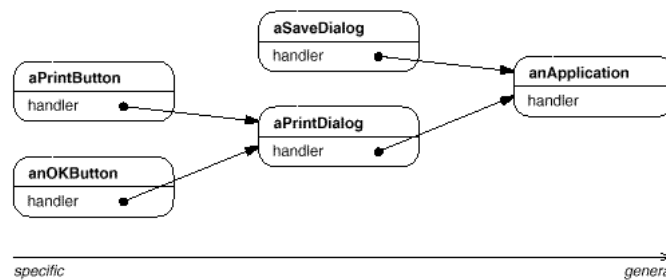
תכנות מונחה עצמים בשפת C++  
אוניברסיטת תל אביב

26

## תבניות התנהגות

- Chain of Responsibility - מניעת צימוד בין שולח ההודעה והנמען שלה ע"י מתן כמה נמענים אפשריים לטיפול בהודעה. הנמענים הפוטנציאליים ישורשרו זה לזה וההודעה תחלחל ביניהם עד לטיפול

■ לדוגמא: בקשת Help עבור כפתור



27

## תבניות התנהגות

- Command – הכמסה של בקשה כעצם. לקוח עשוי לטפל במשתנה הודעה (מתודה!) לאגור בקשות, לסדר אותן בתור, ולבצע פעולת UNDO
- לדוגמא: בתפריט כללי אין את הלוגיקה לשליחת הודעה מסוימת

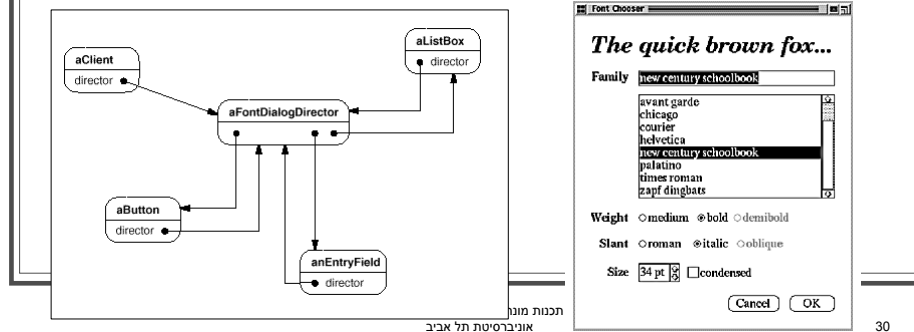
- Interpreter – בהינתן שפה נגדיר ייצוג של הדקדוק שלה בד בבד עם מפרש השפה
- לדוגמא: שפת ביטויים רגולרים

## תבניות התנהגות

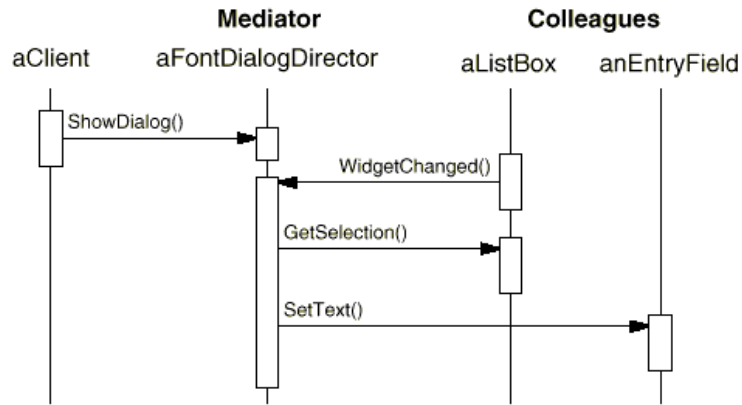
- Iterator – מתן דרך לגשת לאברי עצם מוכל בצורה סדרתית ללא חשיפת הייצוג של העצם המוכל
  - לדוגמא: איטרטור רשימה, עץ וכו'
- Memento – מאפשר לעצם לייצא את המצב שלו לצורך שימוש עתידי ללא פגיעה בעקרון ההכמסה
  - לדוגמא: פעולת UNDO

## תבניות התנהגות

- Mediator – הגדרת עצם המכמיס את הצורה שבה עצמים מתקשרים. המתווך מקטין את התלות בין העצמים ע"י מניעת תקשורת מפורשת ביניהם



## תבניות התנהגות



תכנות מונחה עצמים בשפת C++  
אוניברסיטת תל אביב

31

## תבניות התנהגות

- **Observer** – הגדרת תלות של יחיד-רבים בין עצמים כך שלאחר שינוי ביחיד כל צופיו מיוודעים ומעודכנים בצורה אוטומטית
  - לדוגמא: MVC, שרון מערכת ושרון תצוגה
- **State** – מאפשר לעצם לשנות את התנהגותו כתוצאה משינוי במצבו הפנימי. העצם יראה כאילו הוא החליף מחלקה
  - לדוגמא: TCPConnection

תכנות מונחה עצמים בשפת C++  
אוניברסיטת תל אביב

32



## תבניות התנהגות

- Strategy – הגדרה של משפחה של אלגוריתמים, הכמסה שלהם, ומתן אפשרות להחלפה ביניהם
  - לדוגמא: שבירת שורות ארוכות במסמך הנוצר מזרם
- Template Method - הגדרה של אלגוריתם תוך דחיית כמה צעדים למחלקות נגזרת. מחלקות נגזרת יכולות להגדיר מחדש מקטעים ללא צורך להגדיר מחדש את כל האלגוריתם
  - לדוגמא: פתיחת מסמך ביישום ללא תלות בסוג המסמך או היישום בפועל

## תבניות התנהגות

- Visitor – מייצג פעולה שיש לבצע על עצמים במבנה נתון. התבנית מאפשר להגדיר פעולות חדשות ללא צורך בשינוי המחלקות של העצמים במבנה
  - לדוגמא: עצי גזירה של שפות

# תבניות תיכון נבחרות

(חלק ראשון)

## Model View Controller

- גישה לבניית ממשקים גראפיים שהוצעה בסמולטוק 80, ואומצה גם ע"י מפתחים בשפות וסביבות אחרות.
- יישום בנוי משלושה סוגים של עצמים: מודל Model: עצם של היישום.
- מראה View: הצגה של המודל על המסך.
- בקר Controller: מגדיר את הדרך שבה הממשק מגיב לקלט של המשתמש.
- גישה זאת עדיפה על טיפול בשלושת המרכיבים האלה ביחד.
- MVC משפר את הגמישות ואת השימוש החוזר.

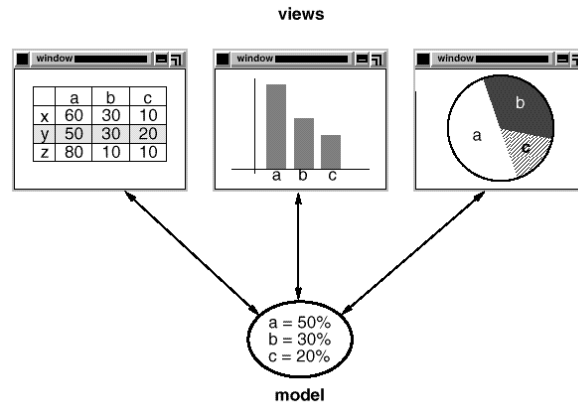
## מודל לעומת מראה

- נניח בינתיים שאין בקר.
- למודל יכולים להיות מספר מראות.
- רוצים ליצור הפרדה בין המראה למודל באמצעות פרוטוקול:
  - המראה שולח ראשית בקשה להתחבר למודל (להירשם כמנוי).
  - כאשר המצב של המודל משתנה, הוא שולח לכל המנויים הודעה על השינוי.
  - כל מראה מעדכן את עצמו.
- נשתמש בתבנית Observer (יש לה גם שימושים נוספים)

## תבנית התיכון Observer

- מטרה: להגדיר תלות של אחד לרבים בין עצמים, כך שכאשר האחד משנה את מצבו, העצמים התלויים מקבלים הודעה ומתעדכנים אוטומטית. (תבנית התנהגות).
- מוטיבציה: לשמור על עקביות בין עצמים קשורים, בלי לגרום לצימוד חזק מדי.
- ישימות:
- כאשר להפשטה יש שני הבטים, אחד תלוי בשני. עצמים נפרדים מקלים על שינוי ושימוש חוזר.
- כאשר שינוי בעצם אחד דורש שינוי במספר לא ידוע של עצמים.
- כאשר עצם יכול להודיע לעצמים אחרים בלי להניח משהו עליהם.

## תבנית התיכון Observer - מוטיבציה

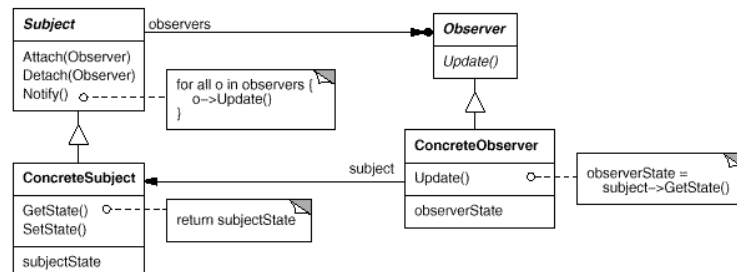


תכנות מונחה עצמים בשפת C++  
אוניברסיטת תל אביב

39

## תבנית התיכון Observer – מבנה

■ נשתמש בתרשים המחלקות:



תכנות מונחה עצמים בשפת C++  
אוניברסיטת תל אביב

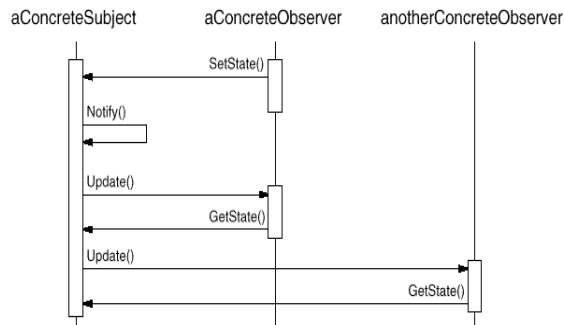
40

## תבנית התיכון Observer - משתתפים

- Subject – שומר רשימה של Observers. מייצא פעולות להוספה והשמטה של Observers. כולל פעולת notify ששולחת הודעת עדכון לכל ה Observers.
- Observer – מייצא פעולה מופשטת של עדכון.
- ConcreteSubject – יורש מ Subject, שומר מצב, קורא ל notify כאשר המצב משתנה.
- ConcreteObserver – יורש מ Observer, מחזיק התייחסות ל ConcreteSubject, מממש את פעולת העדכון.
- הערה (לגבי כל תבניות התיכון): שמות המשתתפים בתבנית מתארים את תפקידי המחלקות בתבנית. שמות המחלקות במערכת יהיו בדרך כלל שונים, ויבטאו את התפקיד הכללי יותר של המחלקה.

## תבנית התיכון Observer שיתוף פעולה

- נשתמש ב sequence diagram



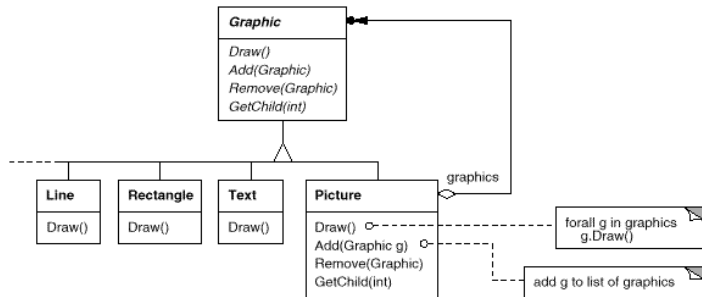
## Model View Controller – המשך

- מראות (Views) יכולים להיות מקוננים.
- לדוגמא, לוח בקרה מכיל כפתורים.
- מראה מורכב מתנהג בדיוק כמו עצם מראה רגיל.
- נשתמש בתבנית התיכון Composite (שימושית בהרבה הקשרים נוספים).

## תבנית התיכון Composite

- מטרה: להרכיב עצמים למבני עץ שמייצגים את ההיררכיה של היחס חלק-שלם. לאפשר ללקוחות לטפל בעצמים בודדים ובהרכבות באופן אחיד. (תבנית מבנה).
- דוגמא: עצמים גרפיים.
- ישימות:
  - כאשר רוצים לממש יחס חלק-שלם.
  - כאשר רוצים לאפשר ללקוחות להתעלם מהבדלים בין עצמים מורכבים לעצמים בודדים.

# תבנית התיכון Composite דוגמא

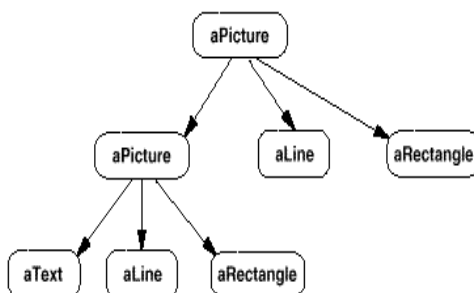


תכנות מונחה עצמים בשפת C++  
אוניברסיטת תל אביב

45

# תבנית התיכון Composite - דוגמא

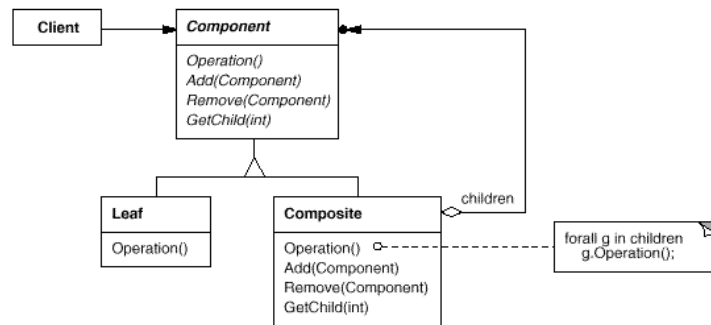
■ דוגמא למבנה של ציור



תכנות מונחה עצמים בשפת C++  
אוניברסיטת תל אביב

46

## תבנית התיכון Composite - מבנה



תכנות מונחה עצמים בשפת C++  
אוניברסיטת תל אביב

47

## תבנית התיכון Composite משתתפים

- **Component** – מגדיר ממשק לעצמים בהרכבה. מיישם ברירת מחדל לפעולות המשותפות. מגדיר ממשק לגישה לרכיבים הילדים.
- **Leaf** – (אחדים) – יורש מ **Component**. מייצג רכיבי עלה. מיישם התנהגות לעצמים הפרימיטיביים.
- **Composite** (אחדים?) - יורש מ **Component**. מייצג רכיבים מורכבים. מיישם פעולות על הילדים ע"י איטרציה על רשימת הרכיבים הכלולה בו.
- **Client** – לקוח של **Component**, מבצע פעולות על עצמים רק דרך ממשק ה **Component**.

תכנות מונחה עצמים בשפת C++  
אוניברסיטת תל אביב

48



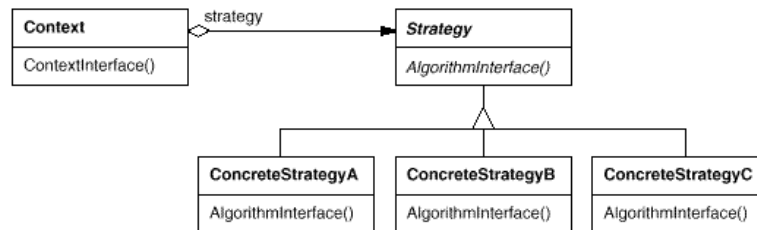
## המשך – Model View Controller

- הבקר משמש להכמסה של מנגנון התגובה: איך מראה מגיב לקלט של המשתמש.
- מאפשר שנויים בצורת התגובה בלי לשנות את המראה.
- לדוגמא, החלפת כפתורים לפקודות בתפריט.
- שינוי כזה אפשרי אפילו בזמן ריצה.
- היחס בין מראה לבקר הוא דוגמא לתבנית התיכון Strategy

## תבנית התיכון Strategy

- מטרה: לספק הכמסה למשפחה של אלגוריתמים ולעשותם ברי החלפה. לאפשר לאלגוריתמים להשתנות באופן בלתי תלוי בלקוחות. (תבנית התנהגות).
- מוטיבציה:
  - הסרת האלגוריתם מפשטת את הלקוחות.
  - מדיניות שונה בזמנים שונים.
- ישימות:
  - כאשר אוסף מחלקות שונות זו מזו רק בהתנהגות.
  - יש צורך בחלופות (למשל שוני בזמן/זכרון).
  - האלגוריתם צריך להסתיר נתונים מורכבים.
  - להוציא התנהגות מותנית מהמחלקה

## תבנית התיכון Strategy - מבנה



תכנות מונחה עצמים בשפת C++  
אוניברסיטת תל אביב

51

## תבנית התיכון Strategy – משתתפים

- Strategy – מגדיר מנשק משותף לכל האלגוריתמים הנתמכים.
- ConcreteStrategy (אחדים) – מספק מימוש של האלגוריתם בהתאם למנשק Strategy
- Context – לקוח של Strategy. בזמן ריצה, ההתייחסות היא לעצם מטיפוס של אחד מ ConcreteStrategy.

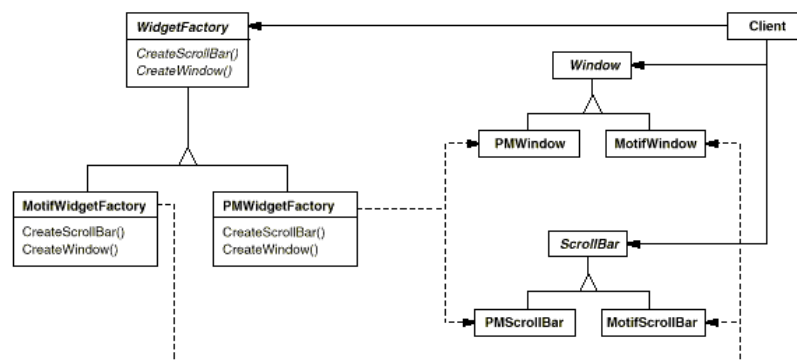
תכנות מונחה עצמים בשפת C++  
אוניברסיטת תל אביב

52

## תבנית התיכון Abstract Factory

- מטרה: לספק ממשק ליצירת משפחה של עצמים קשורים, בלי לקבוע את המחלקות המוחשיות.
- דוגמא: יישום מבוסס חלונות שצריך לרוץ על מספר מערכות חלונות תוך תלות מינימלית במערכת החלונות.
- כללית – יש מספר מוצרים ומספר גירסאות (או משפחות) כל מוצר זמין בכל המשפחות. המערכת משתמשת במוצרים ממשפחה אחת, אך התלות במשפחה צריכה להיות מזערית

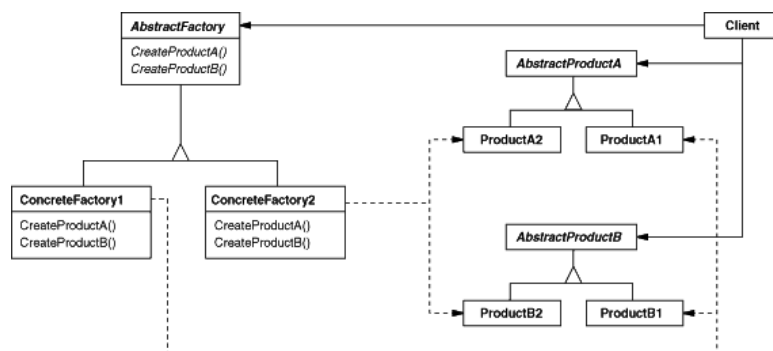
## תבנית תיכון Abstract Factory - דוגמא



## תבנית תיכון Abstract Factory - ישימות

- כאשר מערכת צריכה להיות תלויה באופן שבו מוצרים נוצרים, מורכבים ומיוצגים.
- כאשר המערכת צריכה להיות מקונפגת לאחת מבין מספר משפחות של מוצרים.
- כאשר משפחה של מוצרים מתוכננת לעבוד ביחד, ועלינו לאכוף זאת.
- כאשר רוצים לספק ספריה של מוצרים, ולגלות רק את המנשקים.

## תבנית תיכון Abstract Factory - מבנה



## תבנית תיכון Abstract Factory משתתפים

- AbstractFactory
  - מגדיר שרותים מופשטים ליצירת כל נמוצרים המופשטים.
- ConcreteFactory (N גרסאות)
  - יורש מ AbstractFactory ומיישם את השרותים ליצירת מוצרים מוחשיים.
- AbstractProduct (M גרסאות)
  - מגדיר מנשק לטיפוס של מוצר.
- ConcreteProduct (N\*M גרסאות)
  - יורש מ AbstractProduct ומיישם את הממשק.
  - מיועד להווצר ע"י ConcreteFactory מסוים.

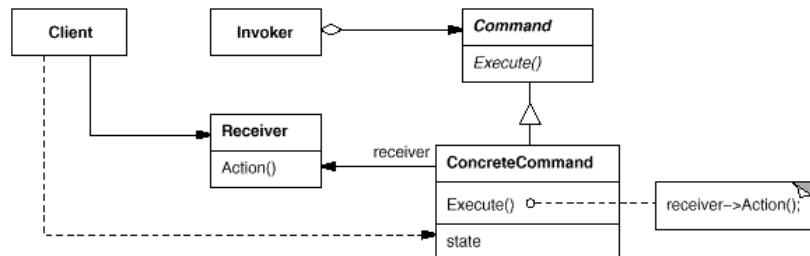
## תבנית תיכון Abstract Factory משתתפים (המשך)

- Client - לקוח של AbstractFactory ושל מספר AbstractProduct. שתי אפשרויות:
  - משתמש ב AbstractProduct אחד בלבד, וכדי להחליפו יש לשנות מקום אחד בקוד ולקמפל.
  - תומך ביותר מ AbstractProduct אחד. משתמש בפיצול (משפט switch) במקום אחד לכל היותר ביצירת עצם בית החרושת.

## תבנית התיכון Command

- מטרה: להגדיר ממשק לצורך ביצוע פעולה. (תבנית התנהגות).
- ישימות:
  - פרמטריזציה של עצמים על פי הפעולה שיש לבצע. תחליף מונחה עצמים למנגנון של callback.
  - לאפיין, לצבור ולבצע בקשות בזמנים שונים, אולי בתהליך נפרד.
  - תמיכה ב redo ו undo
  - לשמור רשימת פעולות לביצוע כאשר מחלימים מקריסת מערכת.
  - לבנות מערכת סביב פעולות ברמה גבוהה הבנויות מפעולות פרימיטיביות (טרנסקציות).

## תבנית התיכון Command – מבנה



## תבנית התיכון Command – משתתפים

- Command – מגדיר ממשק לביצוע פעולה.
- ConcreteCommand – יוצר קשירה בין עצם Receiver לבין פעולה. מממש שרות execute ע"י הפעלת פעולה או פעולות של ה Receiver.
- Client – יוצר עצם ConcreteCommand וקובע את ה Receiver שלו.
- Invoker – מבקש מה Command לבצע את הפקודה המתבקשת.
- Receiver – יודע כיצד לבצע את הפעולות הדרושות לביצוע פקודה מבוקשת. כל מחלקה יכולה להוות Receiver.

## סיכום תבניות תיכון

- פתרון מקובל לבעיית תיכון נפוצה בתכנות מונחה עצמים.
- מבנה כללי שיש להשתמש בו כשממשים חלק מתכנית.
- נסיון מצטבר שניתן ללמוד ועוזר לתקשורת בין מהנדסי תוכנה.
- ראינו חלק קטן מהתבניות מהספר של GoF.
- יישום לדוגמא בשפת תכנות נתונה (למשל ג'אווה).
- בעשור האחרון הוצעו כמה מאות תבניות, לא כולן חשובות באותה מידה.
- השימוש בתבניות חדר גם למושגים אחרים בהנדסת תוכנה, וגם בתחומים אחרים.
- תבניות ואנטי תבניות.