Azrieli
College of Engineering
Jerusalem

עזריאלי
מכללה אקדמית להנדסה
ירושלים

הנדסת תוכנה

11. מבוא לתבניות תיכון

מה היום?

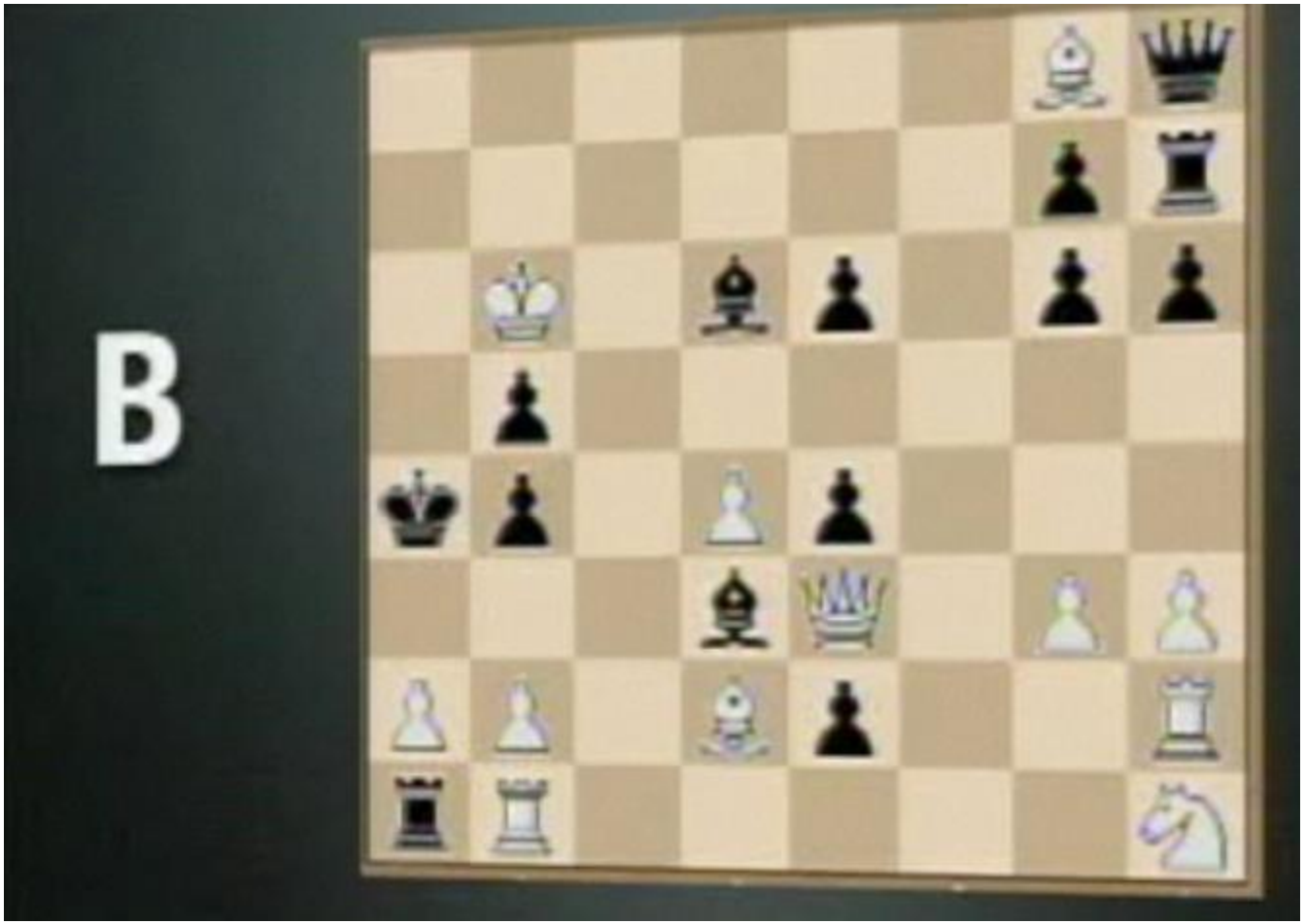
- מבוא לתבניות תיכון Design Patterns
  - הדגמה – כולל BDD ו- Refactoring
  - בהמשך: עקרונות תיכון מונחה עצמים
- שעה 3\תרגיל:
  - סקרי סבב (3 כולל הגשת סקר איכות התיכון)
- פרויקט
  - מעבר לסבב אחרון, סקר קוד + CI (פרטים בהגדרות התרגיל)
  - הצגה סופית, סקר

מקורות

- Design Patterns: Elements of Reusable Object-Oriented Software
- Freeman et. al. Head First Design Patterns
- Robert Martin, Clean Code
- Martin Fowler, Refactoring
- Frank Buschmann, Kevlin Henney, Douglas C. Schmidt "On Patterns and Pattern Languages"
- מספר שקפים מ- ESaaS, ברקלי

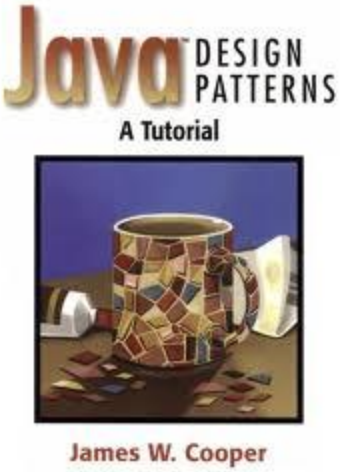
תבניות תיכון Design Patterns


A


B

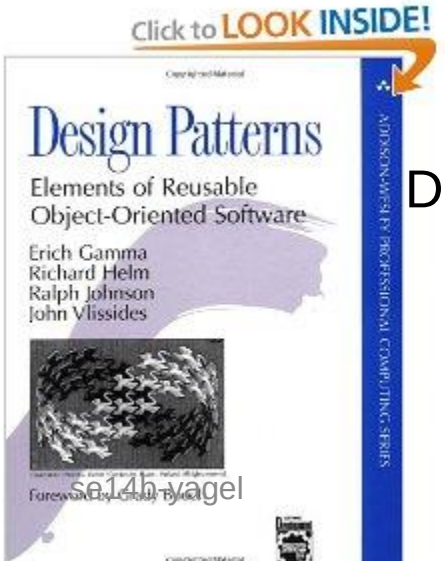

A
Samisch-Nimzowitsch 1923 -
zugzwang for White

תבניות תיכון (עיצוב?)



יישום העקרונות שראינו עד כה:

"design patterns are proven techniques used by experienced developers to tackle recurring design problems without resorting to first principles"

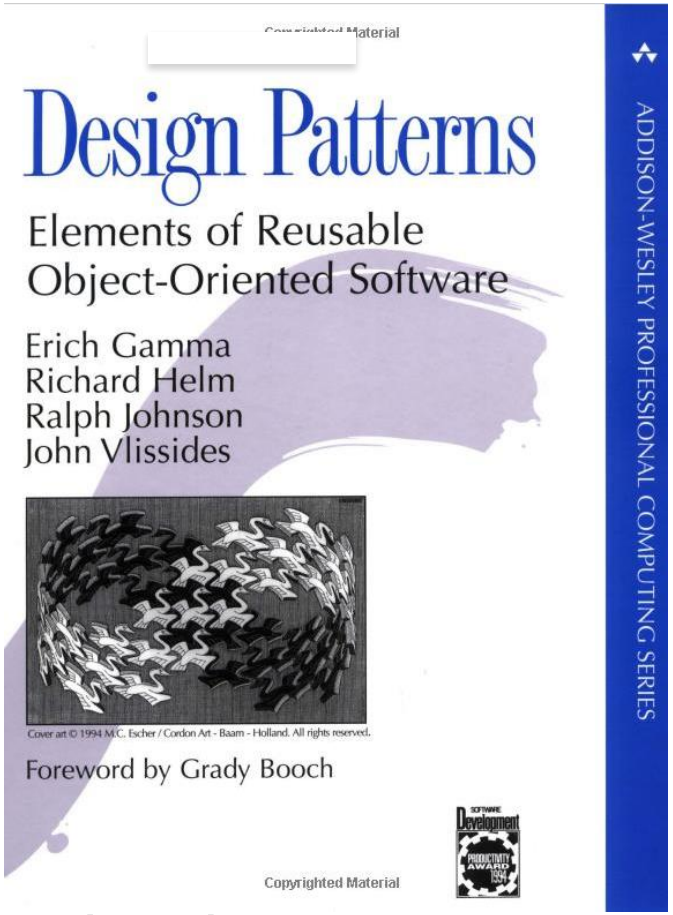


Design Patterns: Elements of Reusable Object-Oriented Software

Gamma, Helm, Johnson and Vlissides, 1995

The Gang of Four (GoF)

- 23 *structural* design patterns
- description of communicating objects & classes
  - captures common (and successful) solution to a *category* of related problem instances
  - can be customized to solve a specific (new) problem in that category
- Pattern ≠
  - individual classes or libraries (list, hash, ...)
  - full design—more like a blueprint for a design



The GoF Pattern Zoo

1. Factory
2. ***Abstract factory***
3. Builder
4. Prototype
5. ***Singleton/Null obj***
6. ***Adapter***
7. ***Composite***
8. ***Proxy***
9. Bridge
10. Flyweight
11. ***Façade***
12. ***Decorator***

Creation

Behavioral

Structural

13. ***Observer***
14. Mediator
15. Chain of responsibility
16. Command
17. Interpreter
18. ***Iterator***
19. Memento (memoization)
20. State
21. ***Strategy***
22. ***Template***
23. Visitor

Meta-Patterns

Separate out the things that change from those that stay the same

1. Program to an Interface, not Implementation
2. Prefer composition & delegation over Inheritance
   - delegation is about interface sharing, inheritance is about implementation sharing

Antipattern

- Code that looks like it should probably follow some design pattern, but doesn't
- Often result of accumulated *technical debt*
- Symptoms:
  - Viscosity (easier to do hack than Right Thing)
  - Immobility (can't DRY out functionality)
  - Needless repetition (comes from immobility)
  - Needless complexity from generality

SOLID OOP principles
(Robert C. Martin, co-author of Agile Manifesto)

Motivation: minimize cost of change

- Single Responsibility principle
- Open/Closed principle
- Liskov substitution principle
- Injection of dependencies
  - traditionally, Interface Segregation principle
- Demeter principle

נרחיב בהרצאה נפרדת

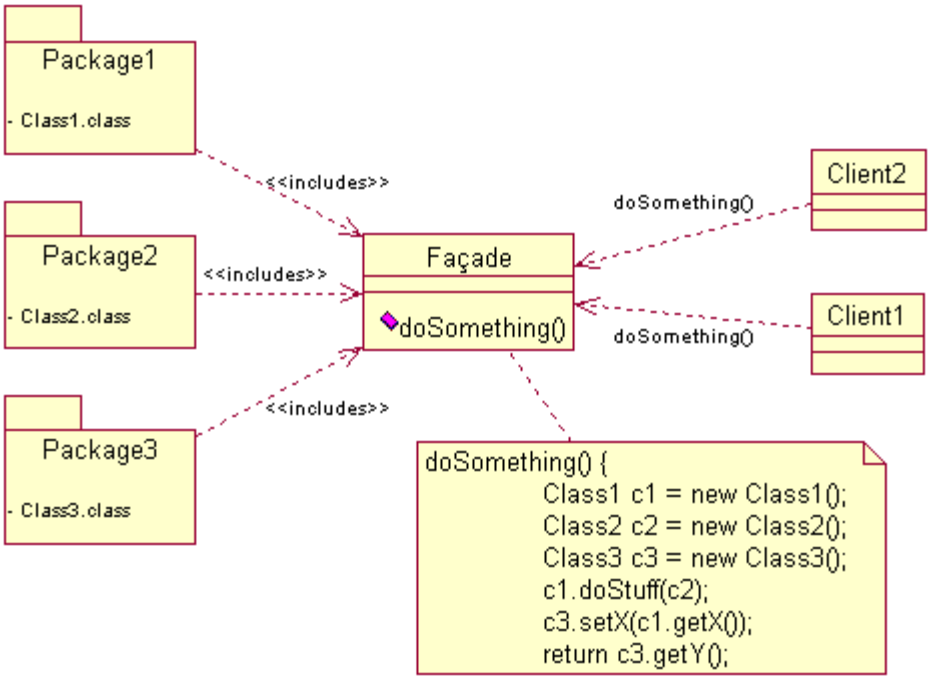
Refactoring & Design Patterns

Methods within a class	Relationships among classes
Code smells	Design smells
Many catalogs of code smells & refactorings	Many catalogs of design smells & design patterns
Some refactorings are superfluous in Ruby	Some design patterns are superfluous in Ruby
Metrics: ABC & Cyclomatic Complexity	Metrics: Lack of Cohesion of Methods (LCOM)
Refactor by extracting methods and moving around code within a class	Refactor by extracting classes and moving code between classes
SOFA: methods are **S**hort, do **O**ne thing, have **F**ew arguments, single level of **A**bstraction	SOLID: **S**ingle responsibility per class, **O**pen/closed principle, **L**iskov substitutability, **I**njection of dependencies, **D**emeter principle

מי מהמשפטים הבאים שקרי?

1. תוכנה שמשתמשת בהרבה תבניות תיכון אינה בהכרח טובה יותר
2. תוכנה בעלת תיכון טוב יכולה להמשיך להתפתח עד שמגיעים למצב שתבניות הופכות לאנטי-תבניות
3. ניסיון להחיל תבניות תיכון מוקדם מדי יכול להיות גרוע כמו החלתן מאוחר מדי
4. רוב תבניות התיכון מיועדות לשפות תוכנה מסוימות

דוגמא Facade

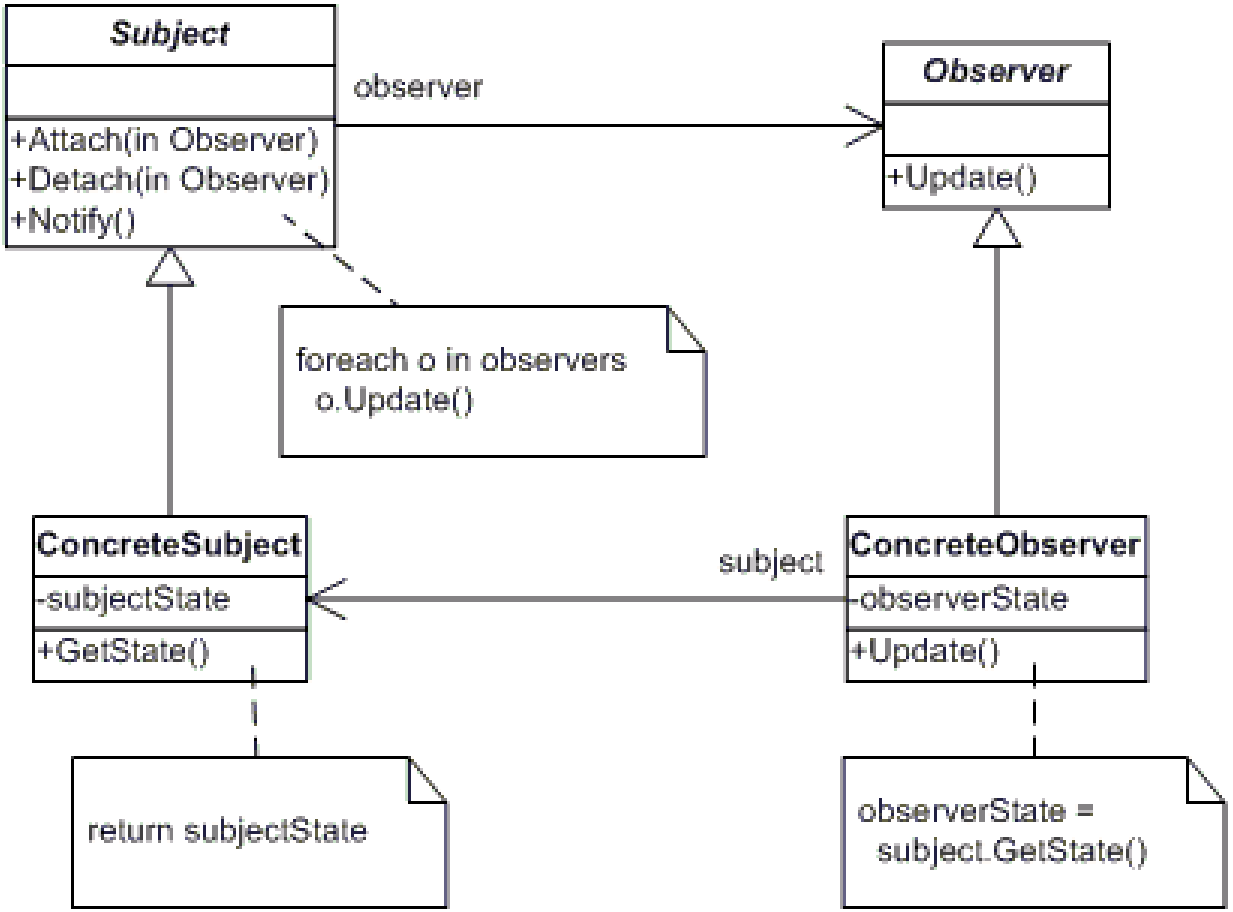


דוגמא לתבנית: Observer

- הגדרת תלות one-to-many בין אוביקטים שתלויים במצב של אוביקט אחר, דרך ממשק
  - תבנית המאפשרת ששינוי במצב של אובייקט (Subject), יופץ לכל המעוניינים (Observers) ללא קשר למספרם וסוגם
  - Don’t call us, we’ll call you...
- אלו עקרונות מעורבים כאן?
  - OCP, DIP, ...
- שמות נוספים:
  - Publish-Subscribe, Event Notification
- כיום בתוך שפות התוכנה: Java Listener, .Net Event (RX) – כמו תבניות נוספות

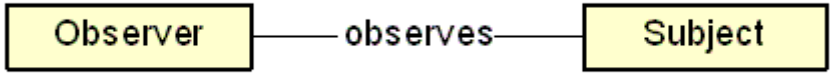
Observer

Example



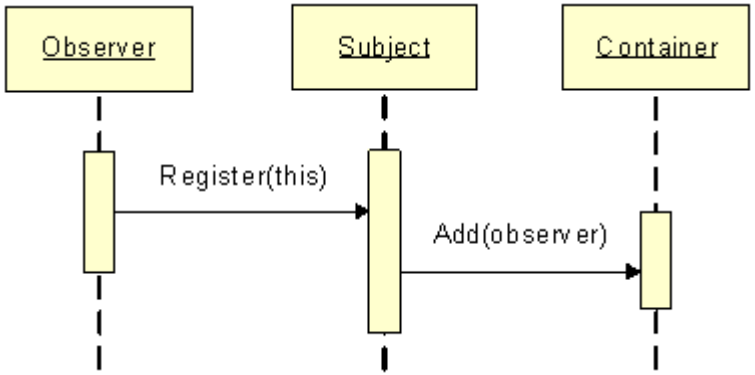
Observer

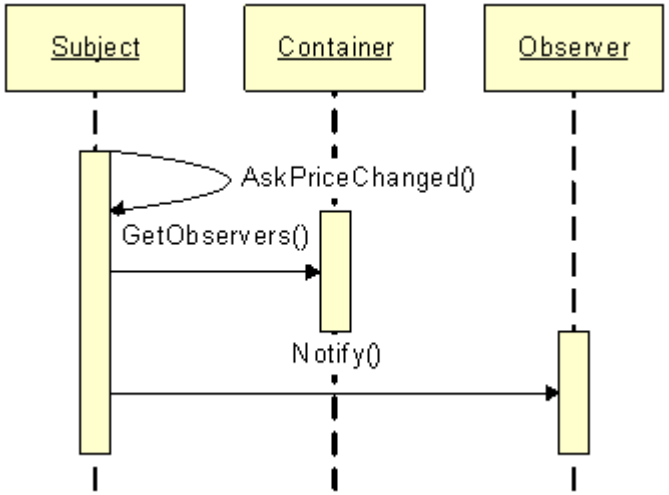
- קשר לוגי:

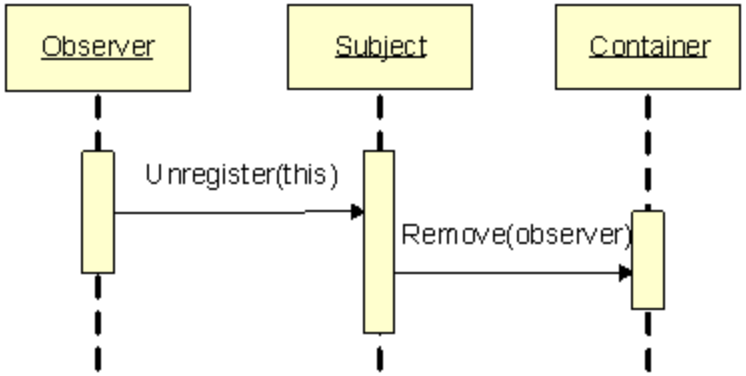


Observer

- תרשימי רצף:







למי לדעתך מבין הבאים תבניות תיכון הכי יכולות לתרום?

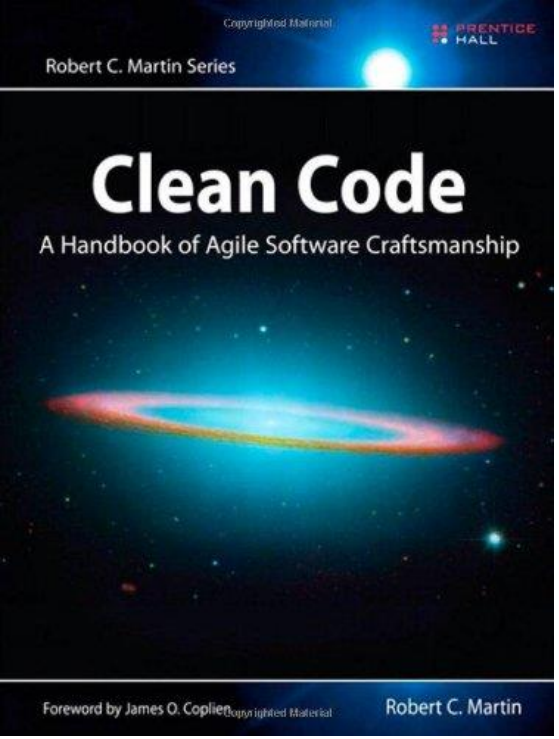
1. לחובבים או מהנדסי תוכנה מתחילים כדי שידעו איך לפתור בעיות נפוצות
2. למהנדסים עם מעט ניסיון שכבר הרגישו קשיים ויכולים להעריך פתרון מוכן
3. למהנדסים מנוסים שיכולים להתאים תבנית ידועה לבעיה מסוימת
4. למומחים שמבינים את ההקשר הרחב והאם מתאימה תבנית מסוימת לבעיה הנידונה

Refactor to a Design Pattern

- Null Object Pattern
- Refactoring: Replace Conditional with Polymorphism
  - Achieving, e.g., Tell Don't ask, delete code - "Use the source (control), Luke"
- Code: https://github.com/jce-il/JobSite-NullObjectPatternDemo-
  - (Java BDD: Jnario / Xtend, Mocks: Mockito)
  - Ruby Example (~7min. / repo)

בפעם הבאה

- עקרונות: תיכון מונחה עצמים
- בהמשך:
- חווית משתמש
- עוד כלים – שילוב מתמשך ועוד



לסיכום

- תבניות תיכון
  - לכידת תובנות
  - ככלי תקשורת בין מפתחים
- הקשר לבדיקות, תיכון מתמשך, Refactoring, ...
- => Clean Code (e.g., Cheat Sheet)