



# הנדסת תוכנה

## 11. מבוא לתבניות תיכון

# מה היום?

- מבוא לתבניות תיכון Design Patterns
  - הדגמה – כולל BDD ו- Refactoring
  - HW5 Refactoring?
  - בהמשך: עקרונות תיכון מונחה עצמים
- עבודה על תרגילים 5/6
- שעה 3\תרגיל:
  - סקרי סבב
- פרויקט
  - סבבים

# 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



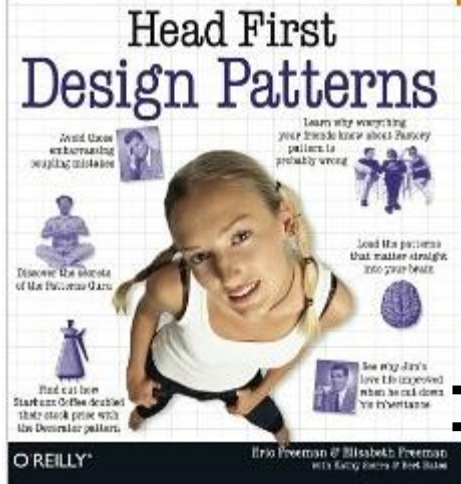
B



A

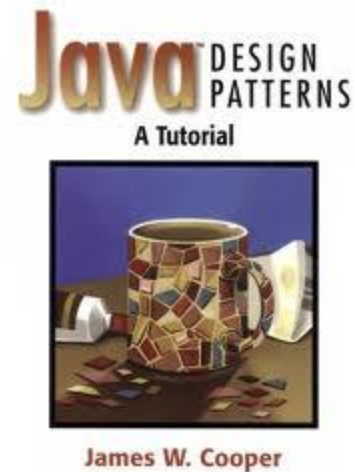


Samisch-Nimzowitsch 1923 -  
zugzwang for White

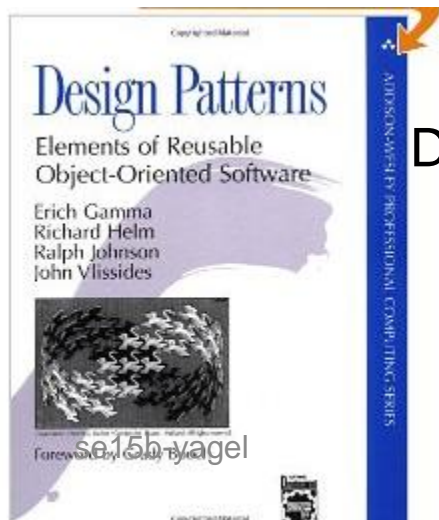


# תבניות תיכון (עיצוב?)

יישום עקרונות שראינו עד כה:



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



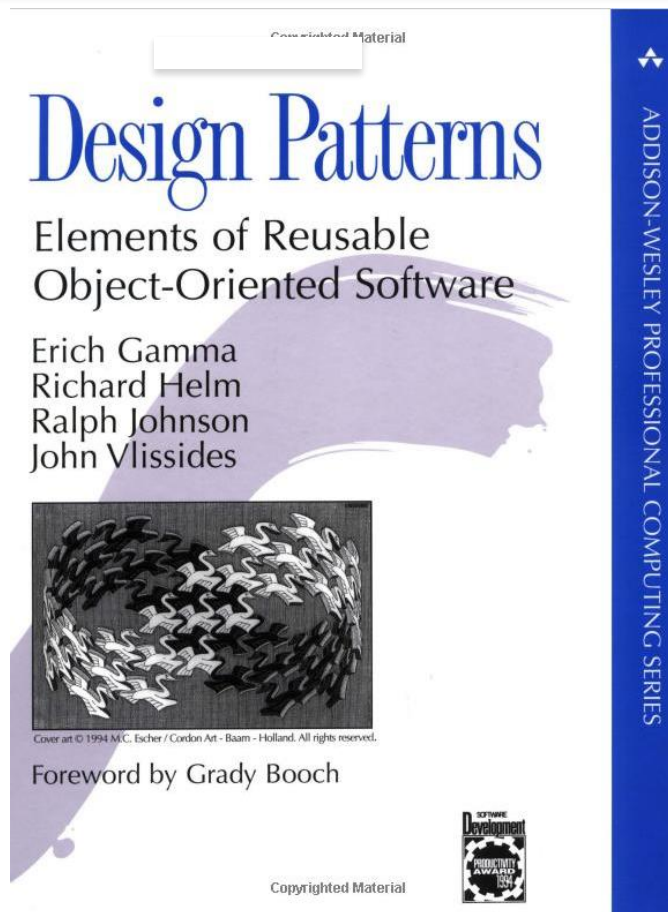
Design Patterns: Elements of Reusable Object-Oriented Software

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  $\neq$ 
  - 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*

- **S**ingle Responsibility principle
- **O**pen/Closed principle
- **L**iskov substitution principle
- **I**njection of dependencies
  - traditionally, **I**nterface Segregation principle
- **D**emeter principle
  - traditionally, **D**ependency Inversion principle

נרחיב בהרצאה נפרדת?

# Refactoring & Design Patterns

## Methods within a class

Code smells

Many catalogs of code smells & refactorings

Some refactorings are superfluous in Ruby

Metrics: ABC & Cyclomatic Complexity

Refactor by extracting methods and moving around code within a class

SOFA: methods are **S**hort, do **O**ne thing, have **F**ew arguments, single level of **A**bstract

## Relationships among classes

Design smells

Many catalogs of design smells & design patterns

Some design patterns are superfluous in Ruby

Metrics: Lack of Cohesion of Methods (LCOM)

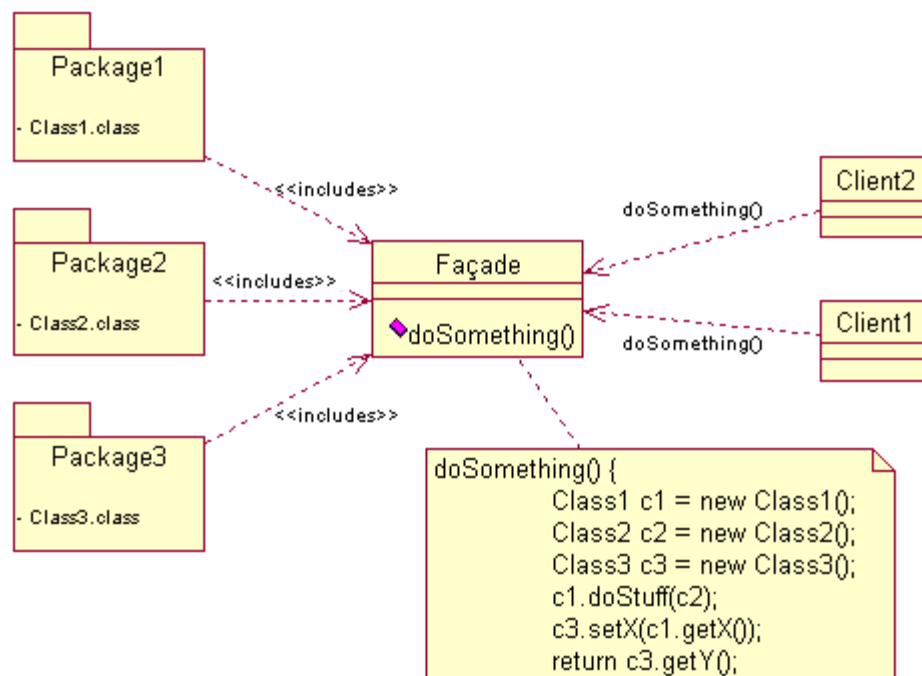
Refactor by extracting classes and moving code between classes

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

# Facade αλγορ



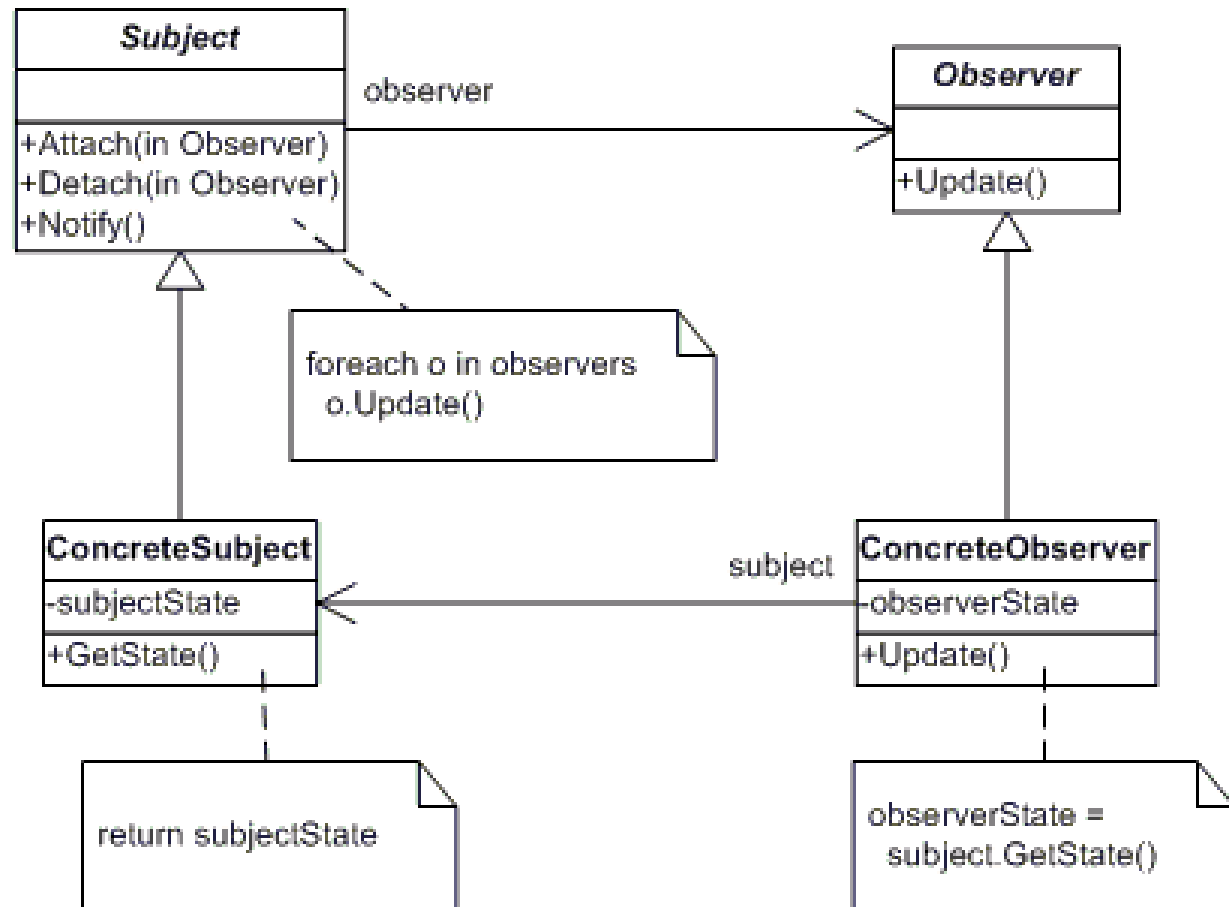


# דוגמא לתבנית: Observer

- הגדרת תלות one-to-many בין אובייקטים שתלויים במצב של אובייקט 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?
  - DIP, OCP, ...
- שמות נוספים:
  - Publish-Subscribe, Event Notification
- כיום בתוך שפות התוכנה: Net Event, Java Listener, (RX) – כמו תבניות נוספות

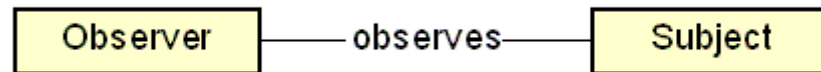
# Observer

## Example



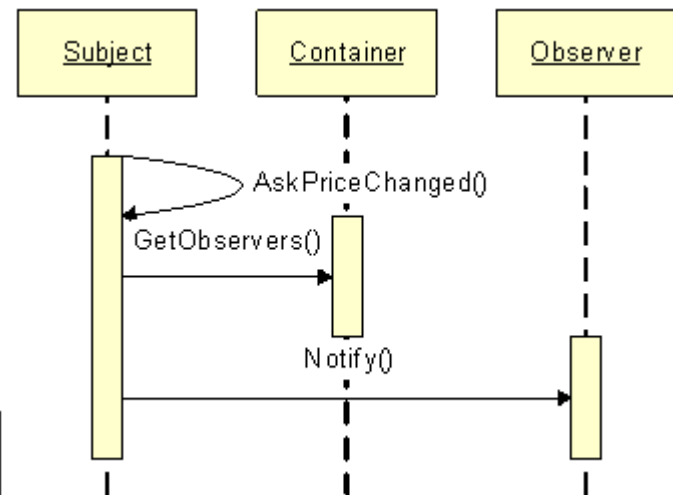
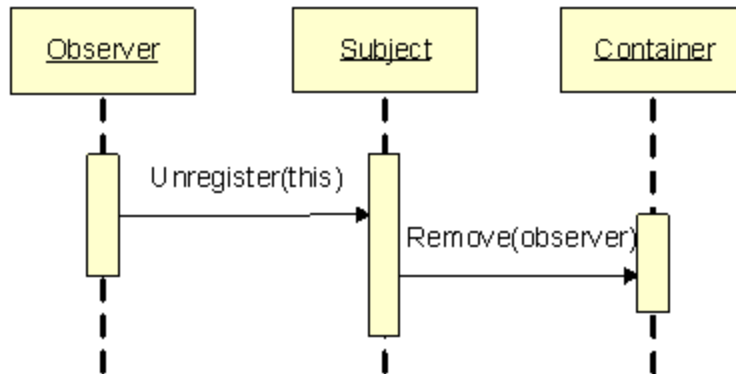
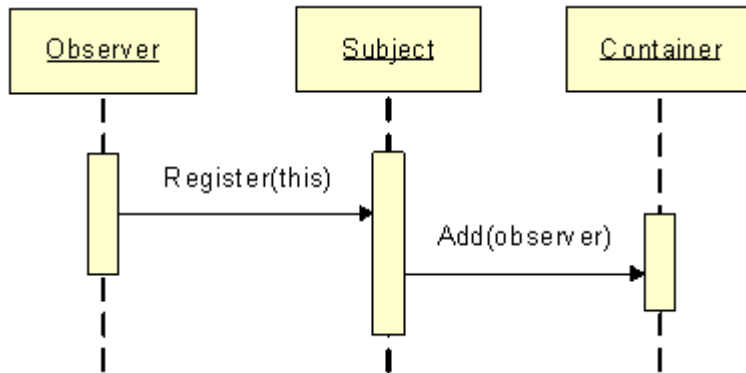
# Observer

• קשר לוגי:



# Observer

## • תרשימי רצף:



# GoF Patterns in Android

- Coursera: [Pattern-Oriented Software Architectures: Programming Mobile Services for Android Handheld Systems](#)  
part of a [track](#) (\*/\*more)
  - [PubSub-Slides1](#), [Slides2](#)
- Other lecture: [slides](#) (\*/\*), [video](#)

# 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
- כלים מתקדמים – סקרי קוד, גיט 2





## בפעם הבאה – מח"ר

- סיכום הקורס

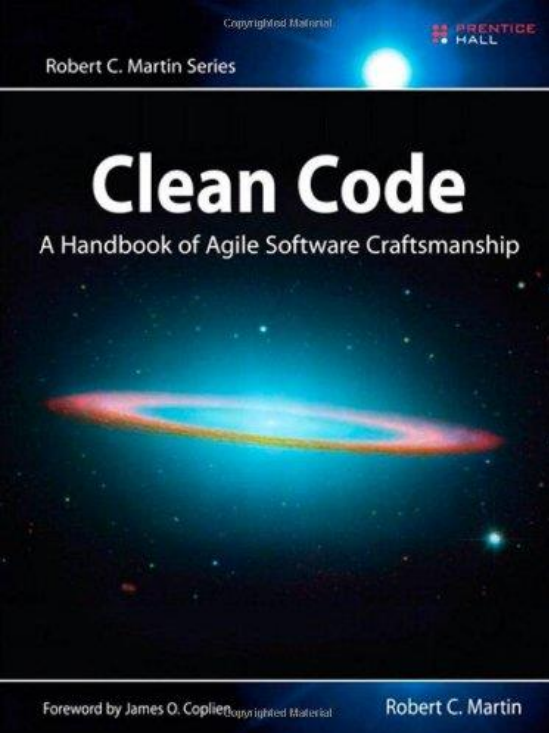
– [סקב](#) קורס, פרויקט, שיטות, תהליכים וכלים

– מבנה הבחינה

- מצגות סיום (הנחיות במסמך סבבים):

– הצגה צוותית: מוצר + רטרוספקטיבה (CI)

– כל אחד מספר על חלקו (משימה אישית) ויכול להישאל על כך



# לסיכום

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