Design Patterns

Akın Kaldıroğlu
akin@javaturk.org

February 24th 2015

www.javaturk.org

Akın Kaldıroğlu?

- He is from Ayvalık; an Eagean child :)
- An İTÜ graduate, 1990.
- He has lived in US to get graduate study in CE and SWE and worked as developer in different companies.
- He lives on giving consultancy and training on Java and SWE.
- Keeps a blog at **www.javaturk.org**.
- Music, philosophy and his kids are among what he loves most.
- **akin@javaturk.org, @kaldiroglu**

www.javaturk.org

Seminars

- As part of my social responsibility I keep giving in class or online seminars on different topics:
  - Clean Code
  - Design Patterns
  - Top-10 Reasons That Your Java Code is not Object-Oriented
  - *Java 8 and Functional Programming*
  - *JVM and Its Tuning*
  - *What Does Being a Programmer Mean?*

www.JavaTurk.org

Agenda

- Nature of Software
- Principles of Systems
- Design Patterns
- A Case Study: Proxy Pattern

www.javaturk.org

It isn't that they can't see the solution.
It is that they can't see the problem.

G. K. Chesterton,
The Point of a Pin in The Scandal of Father Brown

www.JavaTurk.org

Nature of Software

- The most essential properties of software is complexity and change.
  - It is highly complex and infinitely changeable.

The most radical possible solution for constructing software is not to construct it at all.

F. Brooks in No Silver Bullet

www.javaturk.org

Software is Complex - I

- Software has been claimed to be more complex comparing to many other engineering products or scientific works:

- Abstract-conceptual, intellectual, has no physical limits,

- Invisible, hard to think about and comprehend,

- Mostly unique, many factors that make a piece of software different from other pieces.

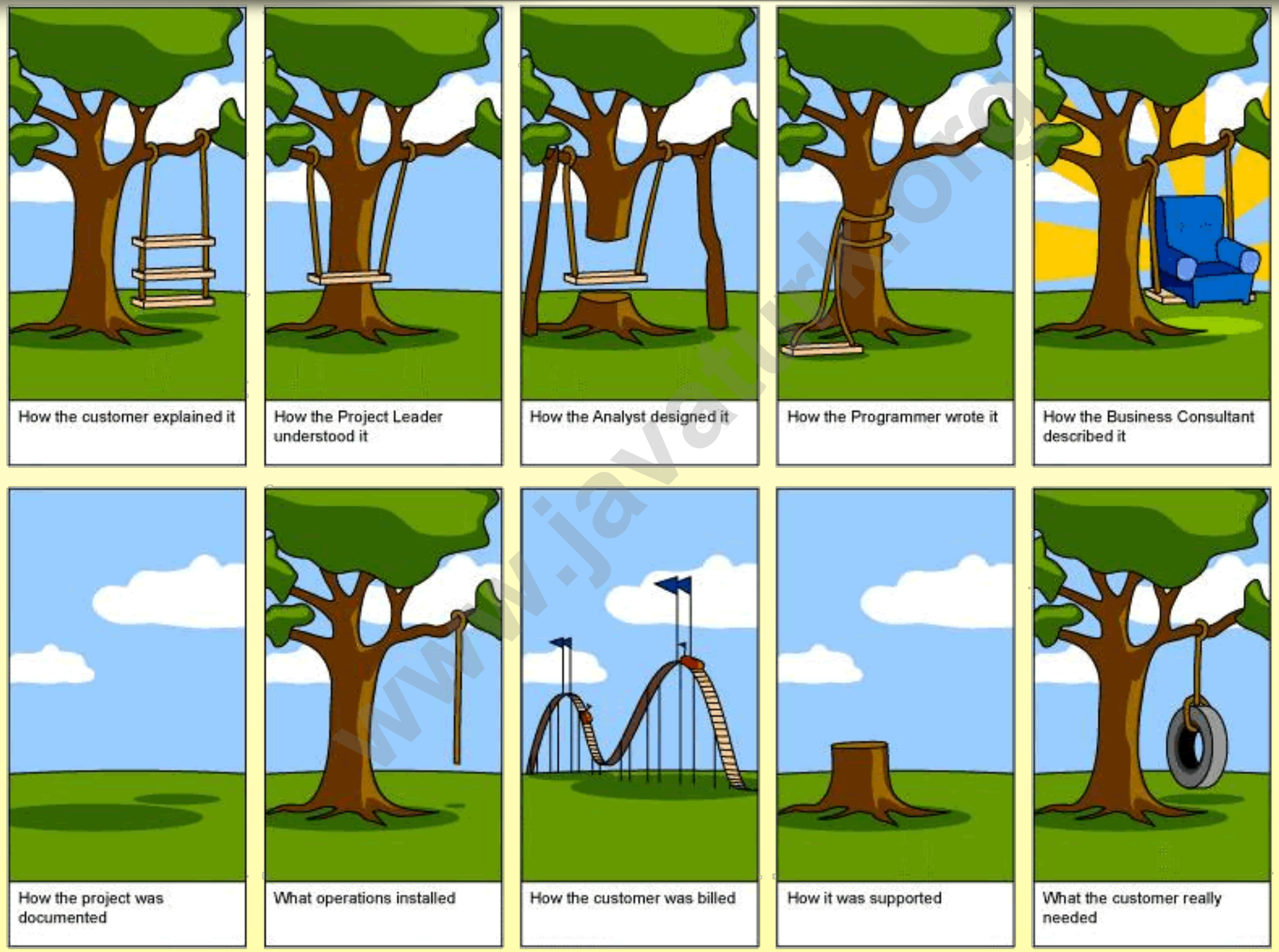
www.JavaTurk.org

Software is Complex - II

- Software systems have too many number of states,
  - Impossible to define, design and test them exhaustively,
  - Most probably the easiest part is coding.
- Pieces of software system are all unique otherwise we combine similar parts into a single unit
- So having a “bigger” software can only be achieved adding new, unique parts not just enlarging the existing ones!
  - Relationships among pieces of software are not linear.

www.JavaTurk.org

Hard To Comprehend



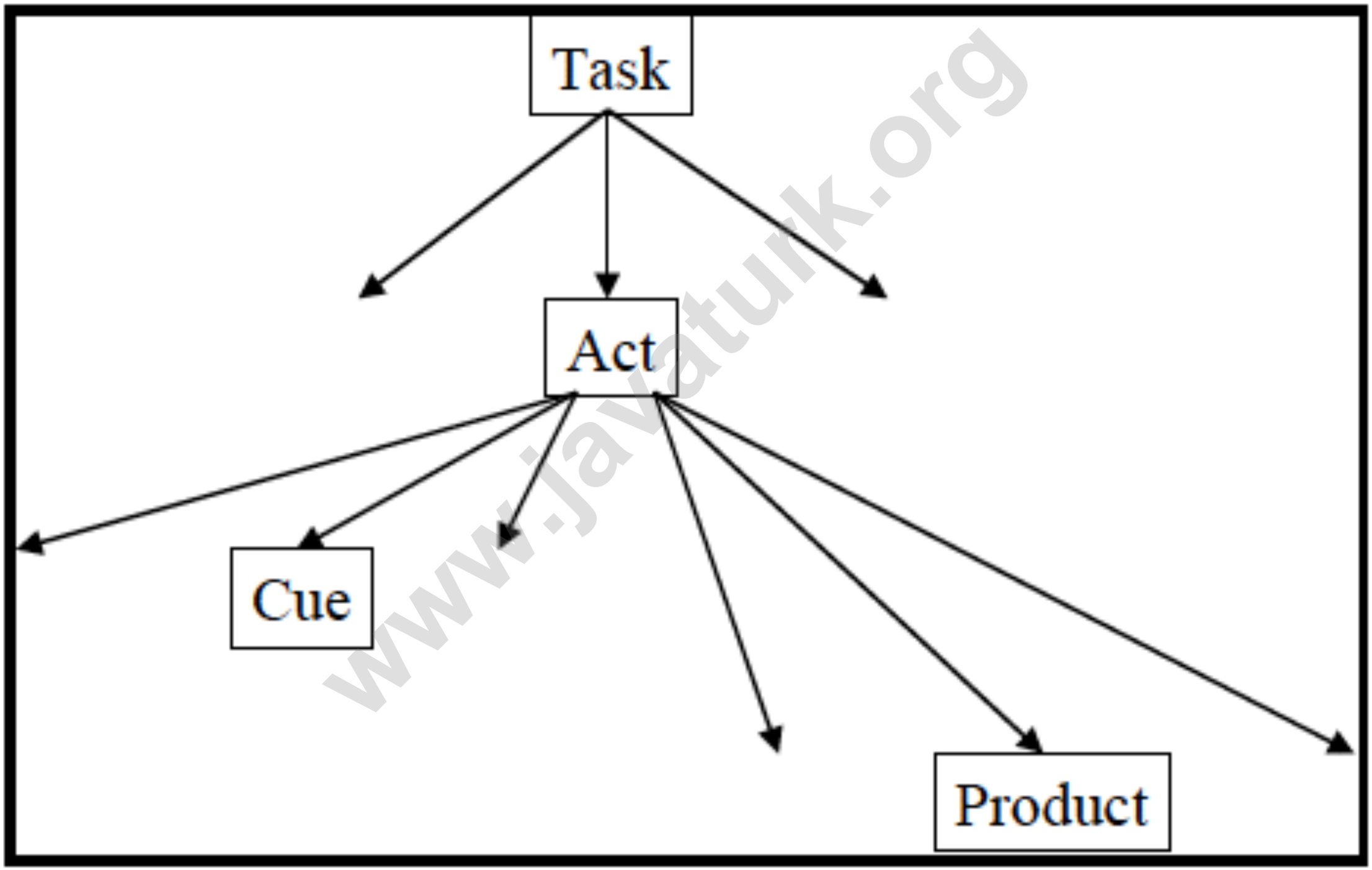
www.JavaTurk.org

Wood's Task Complexity

- In a paper titled "*Task Complexity: Definition of The Construct*", 1986, R. E. Wood claimed that there are three factors that determine the complexity of a task:
  - products of the task
  - required acts for the task
  - information cues for the task

www.javaturk.org

Wood's Task Complexity



www.JavaTurk.org

Wood's Approach to Software

- If we apply Wood's formulation to software:
  - Products are the outputs of the software
  - Acts are the program units such as classes and methods in the software
  - Information cues are the data tokens that are processed in the program units i.e. the acts.

www.JavaTurk.org

Types of Complexities

- **Component Complexity = *f*(**# of distinct acts, # of information cues) = *f*(complexity of the acts + complexity of information cues)

- **Coordinative Complexity = *f*(**relationships among the acts)

- So, what do these two mean to you?

www.JavaTurk.org

Cohesion

- Component complexity is determined by how the acts and data/inf. are together => togetherness
  - Togetherness means being focused on one thing while excluding anything else.
  - Easier comprehension!
- **High-cohesion**

www.JavaTurk.org

Coupling

- Coordinative complexity is a measure of how an act is related to others => relatedness
  - Low relatedness means lower amount of information about others
  - Easier comprehension!
- **Low-coupling**

www.JavaTurk.org

Know Yourself and Don't Gossip!

- **Highly-cohesive** and **lowly-coupled** systems tend to know more about themselves and less about others.
  - Objects should know only themselves but everything about themselves!
  - It should not allow any spread of their responsibilities to other objects.
  - Objects should know as less as possible about others! Objects don't gossip!
- More cohesion means less coupling!
- **Highly-cohesive** and **lowly-coupled** systems make comprehension easier.

www.JavaTurk.org

Software Constantly Changes

- Maintenance is mostly changing the software
- In other engineering products, what is changed is either broken or worn out parts
- In software new requirements are the main driver behind the change

www.JavaTurk.org

Design for Change

- So highly-cohesive and lowly-coupled systems can be changed more easily than otherwise,
  - Change requires comprehension.
- Since maintenance costs more than development, don't design only for development.
- **Design for change**

www.JavaTurk.org

High and Low Cohesion

Low Cohesion

SystemServices
makeEmployee makeDepartment login logout deleteEmployee deleteDepartment retrieveEmpByName retrieveDeptByID

High coupling!

High Cohesion

LoginService
login logout

EmployeeService
makeEmployee deleteEmployee retrieveEmpByName

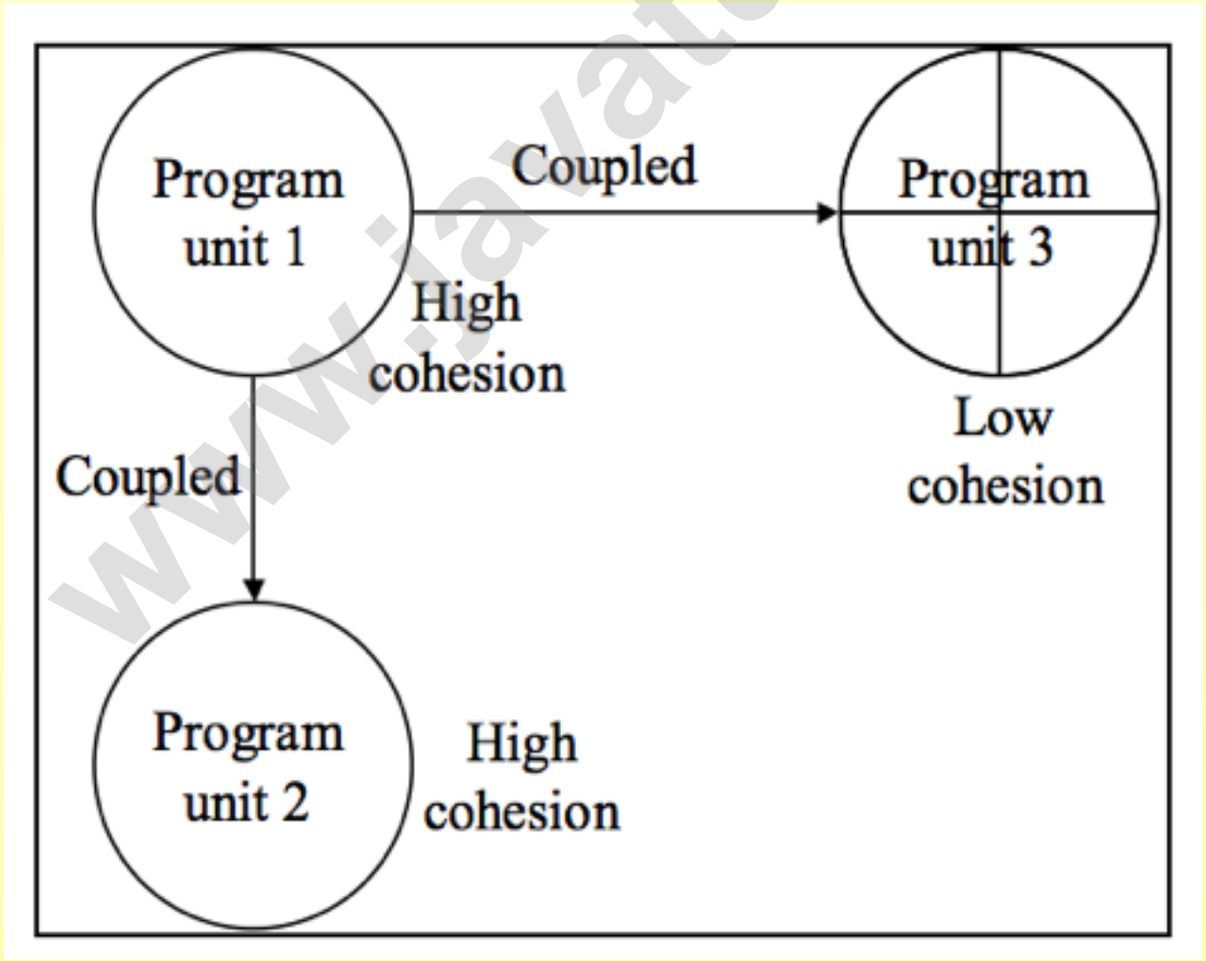
DepartmentService
makeDepartment deleteDepartment retrieveDeptByID

Lower coupling!

www.javaturk.org

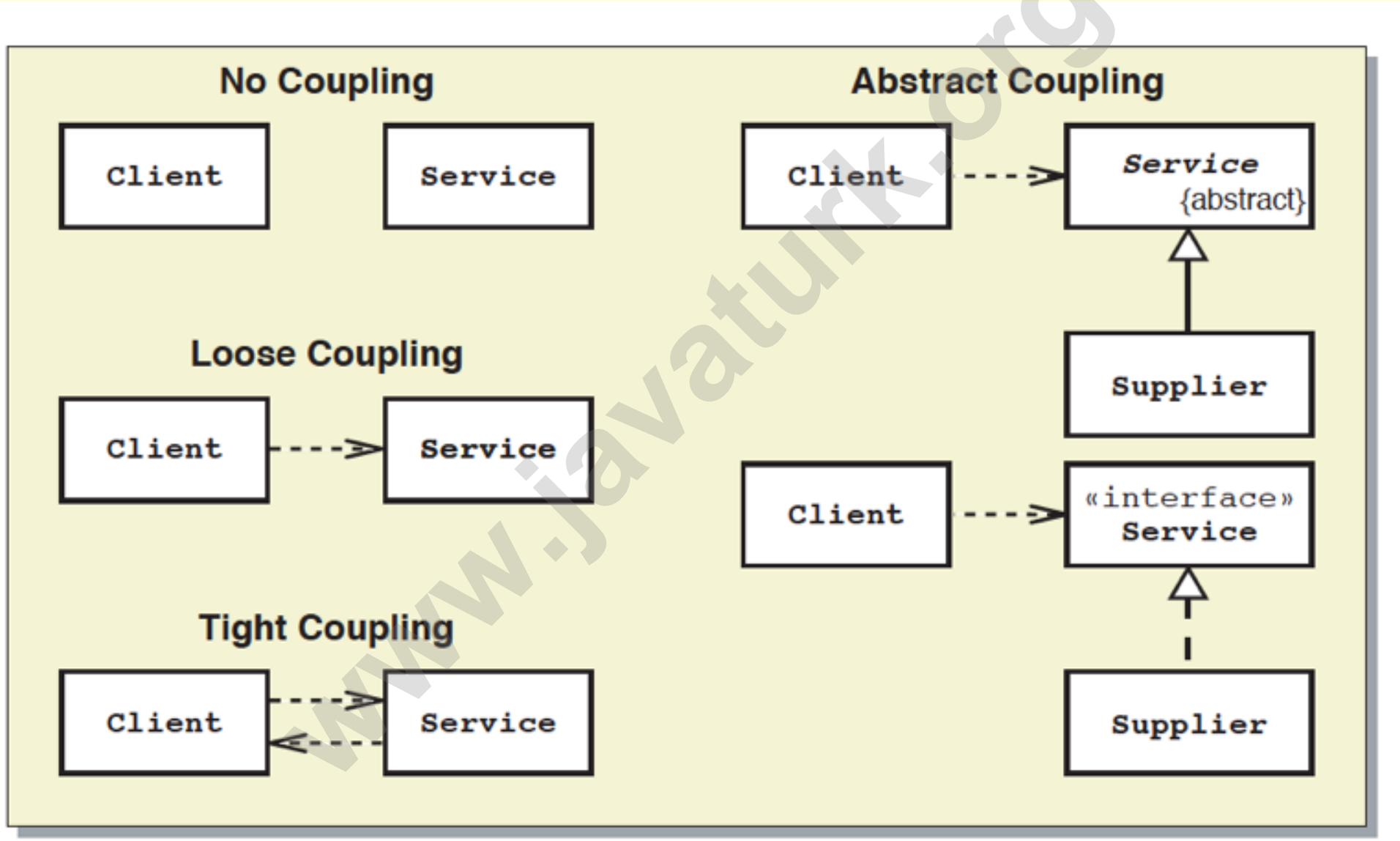
Cohesion & Coupling

- No infinitive cohesion nor zero-coupling!
- Quality of coupling is important.



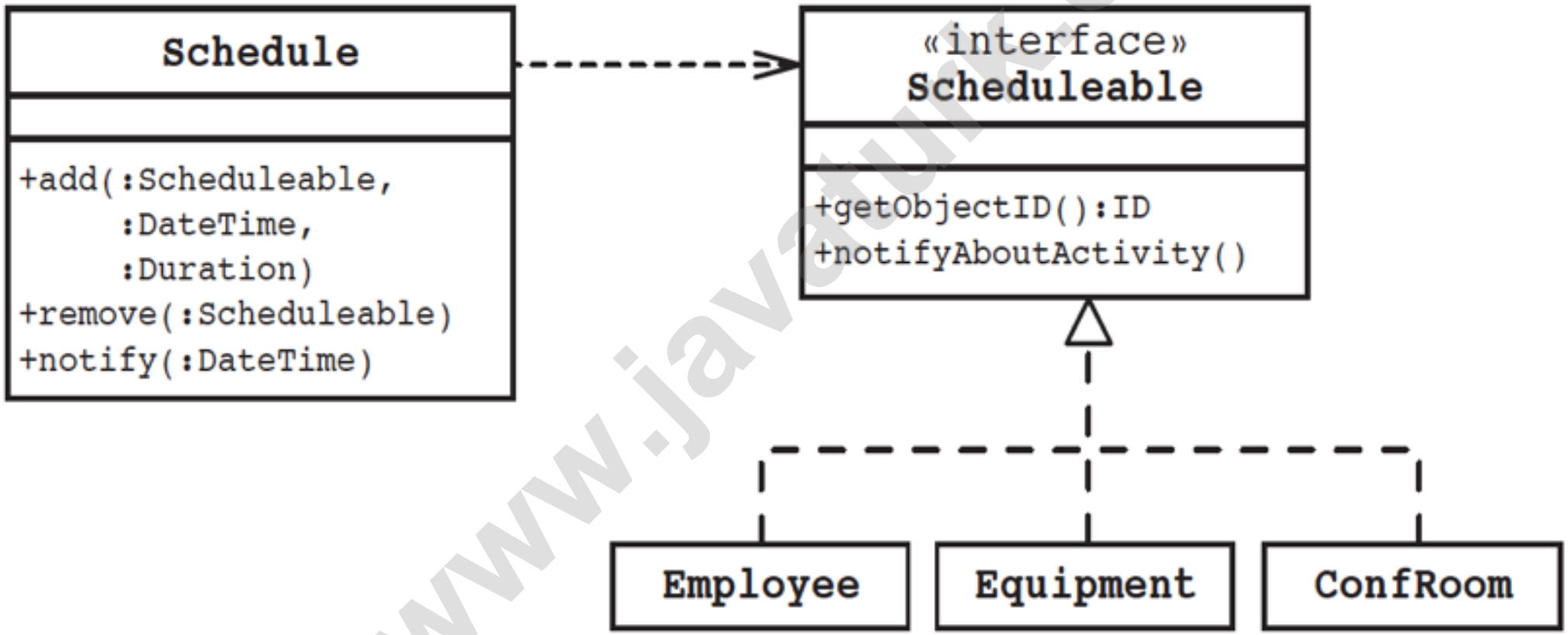
www.javaturk.org

Kinds of Coupling



www.javaturk.org

Abstract Coupling



www.javaturk.org

Lower Coupling

- Lower coupling can only be achieved through interface coupling (or message coupling).
  - **Program to an interface, not an implementation**
- Reason of existence of a class can only be its responsibilities
  - Responsibilities can only be revealed through interface
- Objects only know about their responsibilities
  - **Responsibility-driven design**

www.JavaTurk.org

What is Design Pattern? I

- Design patterns, through object-oriented principles, let us
  - find responsibilities correctly,
  - distribute those responsibilities among the objects by paying attention to cohesion, coupling and change,
- Design patterns solves frequently seen software design problems in terms of highly-cohesive and lowly-coupling objects.

www.JavaTurk.org

What is Design Pattern? II

- Christopher Alexander says, "*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*“

- In this sense, a design pattern is an abstract and reusable solution.

www.JavaTurk.org

Repeating Desing Problems - I

- How to create objects?
  - How to create complex objects?
- How to control the access to an object?
- How to specify multiple ways of performing a task interchangeably?
- How to give commands to objects? How to implements do/redo/undo structures?
- How to implement event mechanisms among objects?

www.JavaTurk.org

Repeating Desing Problems - II

- How to give different abilities to objects during their lifetimes?
- How to manages complex states of an object?
- How to save the state of an object and retrieve it later?
- How to handle many objects as bundle?
- How to apply an operation to many objects?

www.JavaTurk.org

Constituents of Desing Patterns

Item	Description
Name	All patterns have a unique name that identifies them.
Intent	The purpose of the pattern.
Problem/Motivation	The problem that the pattern is trying to solve.
Solution/Structure	How the pattern provides a solution to the problem in the context in which it shows up.
Participants and collaborators	The entities involved in the pattern.
Consequences	The consequences of using the pattern. Investigates the forces at play in the pattern.
Implementation	How the pattern can be implemented. Note: Implementations are just concrete manifestations of the pattern and should not be construed as the pattern itself.
Generic Structure	A standard diagram that shows a typical structure for the pattern.
Applicability	What are the situations in which the design pattern can be applied?
Sample code	Code fragments that illustrate the pattern in a object-oriented language
Known uses	Examples of the pattern found in real systems
Related patterns	What design patterns are closely related to this one? What are the important differences?

www.JavaTurk.org

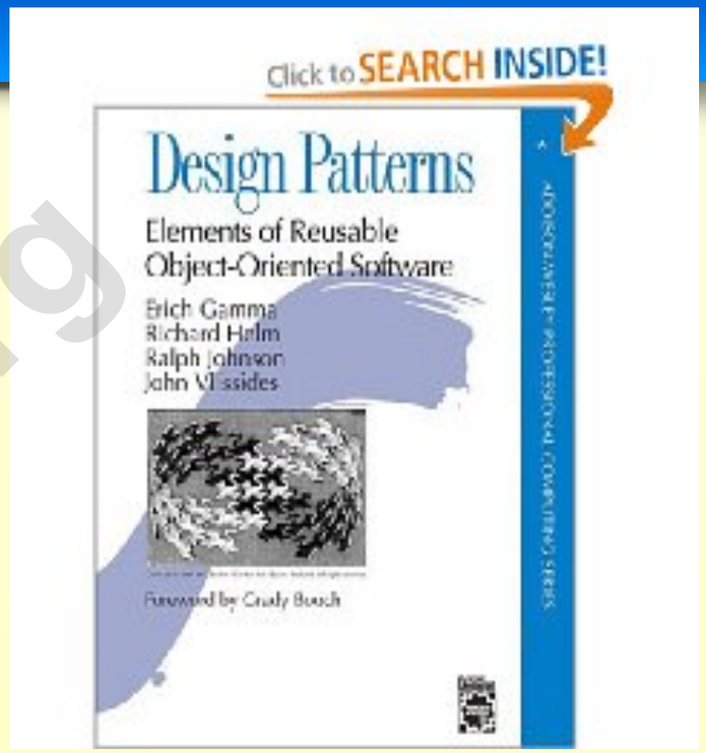
Why Design Patterns?

- Reusable designs,
- Formal and common language,
- Power to abstract away details while focusing on higher objectives.
- Design patterns give us a methodology to avoid potential pitfalls of procedural programming!

www.JavaTurk.org

Initial Works

- Christopher Alexander
- Trygve Reenskaug, Smalltalk'la MVC (Model-View-Controller)
- Kent Beck ve Ward Cunningham
- 1994 - Gang of Four (GoF) - Gamma, Helm, Johnson, and Vlissides
- 1996 - Buschmann, Meunier, Rohnert, Sommerlad, Stal

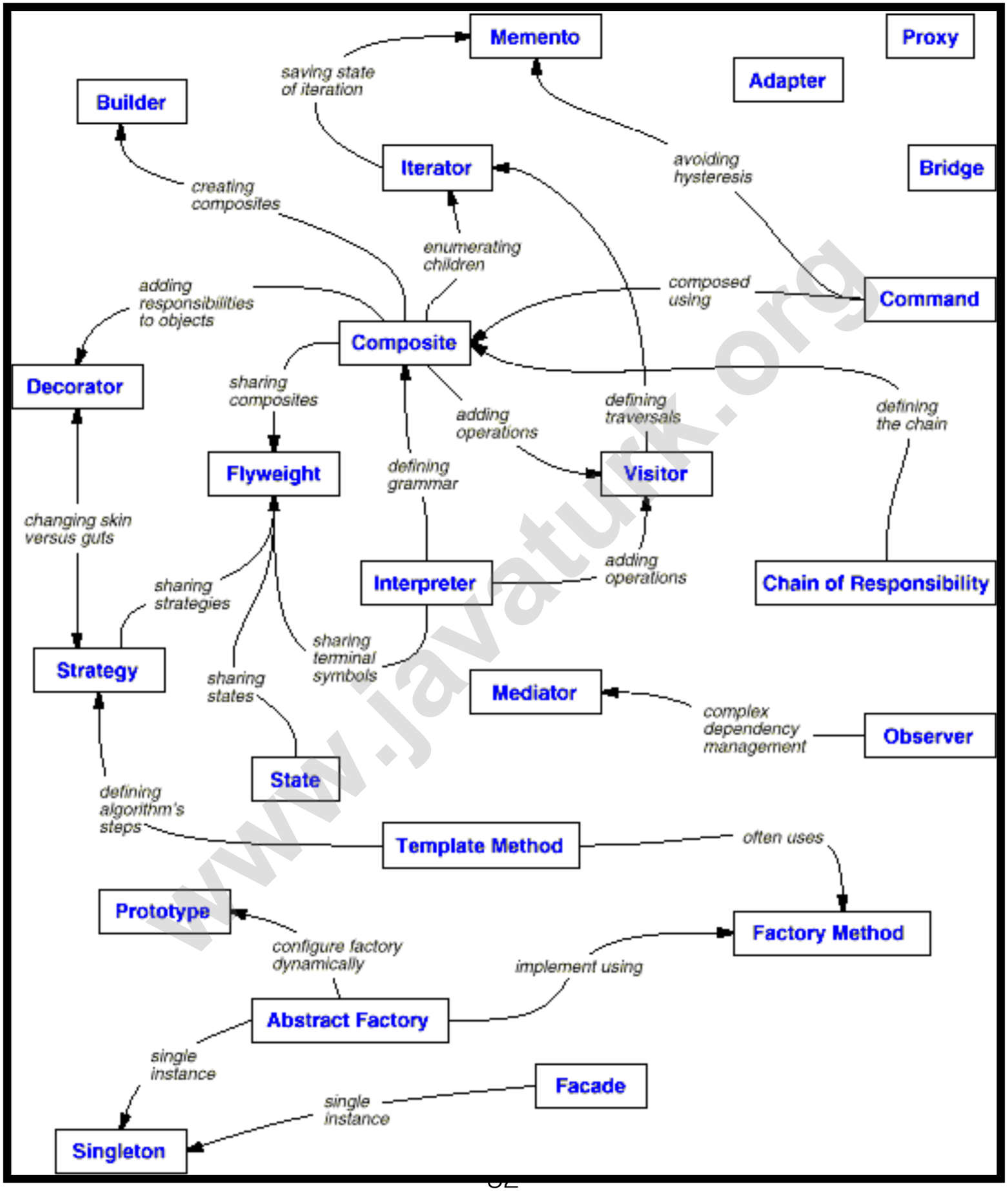


www.JavaTurk.org

GoF's Design Pattern Catalog

- GoF has 23 patterns in 3 different categories in their book:
  - Creational
  - Structural
  - Behavioral
- GoF is hard to understand,
- Examples given in C++,
- First two chapters is a very good summary of OO principles.

www.javaturk.org



www.javatechturk.org

Resources

- F. Brooks, Mythical-Man Month & No Silver Bullet
- Shalloway, Trott, Design-Patterns Explained
- E. Freeman et al., Head-First Design Patterns
- M. Page-Jones, Fundamentals of Object-Oriented Design in UML
- Özcan Acar, Java Tasarım Şablonları ve Yazılım Mimarileri

www.javaturk.org

Case Study: Proxy Pattern

- Let's solve following problem using GoF's proxy pattern:
  - In democracies, citizens have the right to reach the people that govern themselves,
  - PM as the highest governor has the responsibility to listen to citizens for their requests and has the right to respond as he wishes.
- But it is not practical and secure to allow 75 M citizens to reach PM.
- How can we manage this situation without violating the democracy and security of the PM?

www.JavaTurk.org

Finding Responsibilites

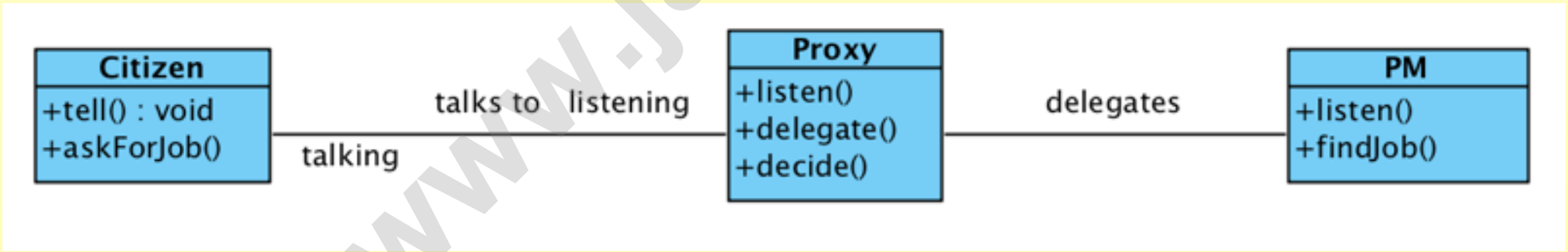
- Responsibilites:
  - PM
    - listening
    - finding job
  - Citizen
    - telling
    - asking for a job

- But main problem:
  - Citizen directly reaches the PM!

www.javaturk.org

Cohesion and Coupling

- How about a delegate between citizens and PM?
  - This decouples PM from citizens but
  - Violates the principle of the democracy!

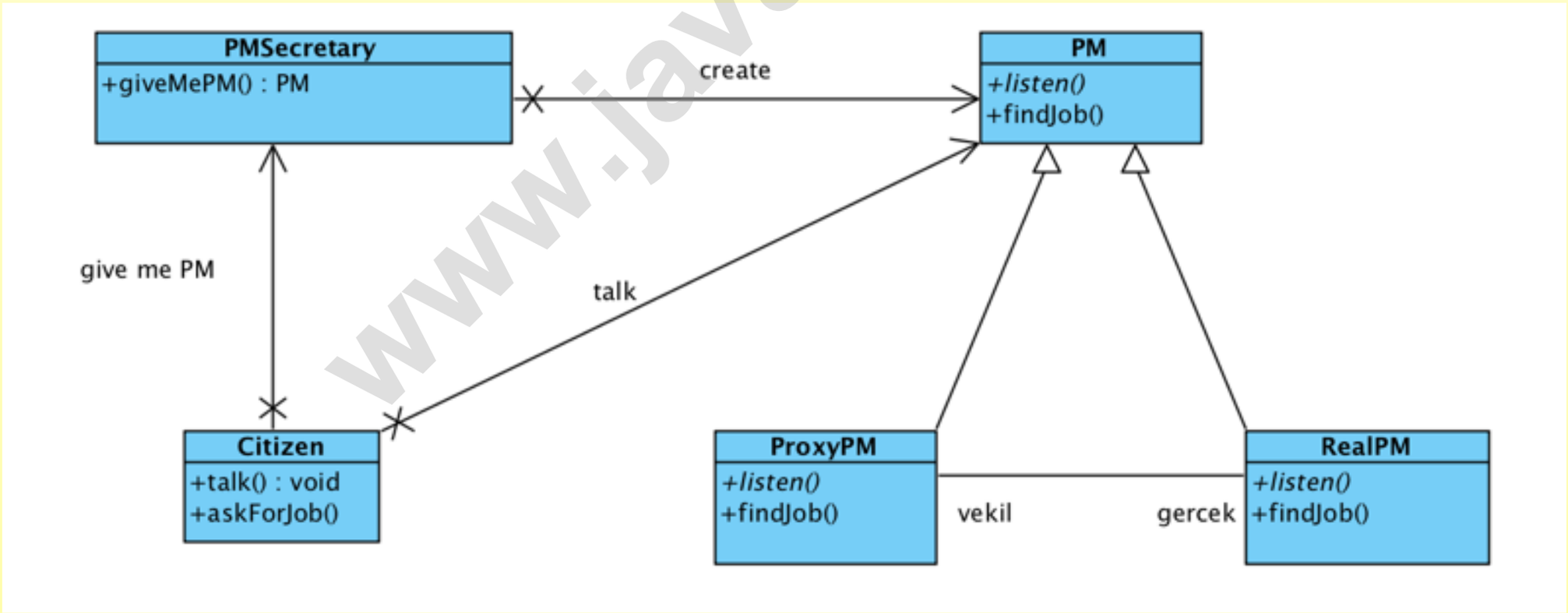


www.javaturk.org

Proxy Pattern

- We should both isolate PM and at the same time do this transparently to its clients,

- When we decouple PM from citizens through an interface:



www.javaturk.org

Other Kinds of Proxies

- Whenever you want to control the access to an object you can use proxy pattern:
- Security proxies in networks,
- Remote (distributed) objects (RMI, Corba, web services, etc.),
- Lazily-loading big or expensive objects

www.javaturk.org

Thanks for listening.

This presentation can be reach at
www.javaturk.org

www.javaturk.org