

# OBJECT-ORIENTED MODELLING (OOM) - 32536

## MODULE 9A

**Advanced UML Modelling;  
Reusability; Patterns;  
System Architecture  
(POOD – Chapter 5 )**

"Copyright Practical OO Analysis,  
Thomson Publishing, 2005".

1

## Module Outline

- **Advanced topics with Class Diagrams**
  - Reusability; Granularity;
  - Patterns; Robustness;
- **Process-Components for:**
  - Reuse and System Architecture

"Copyright Practical OO Analysis,  
Thomson Publishing, 2005".

2

## Types of Reuse

# Code Design Analysis

```
public class Person {
    public void setPersonDtls() { };
    public void changePersonDtls() { };

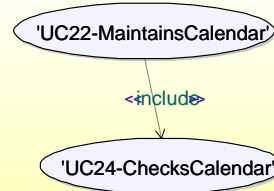
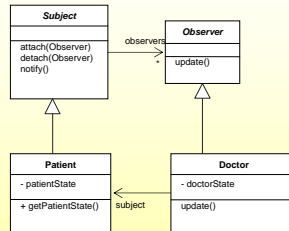
```

```
    private char Person_ID;
    private date DateOfBirth;
}
```

```
public class Patient extends Person {
    public Patient (); {
    }
    public void getPatientDtls (); { }
    public void calcPatientAge (); { }
    public void admitPatient (); { }

```

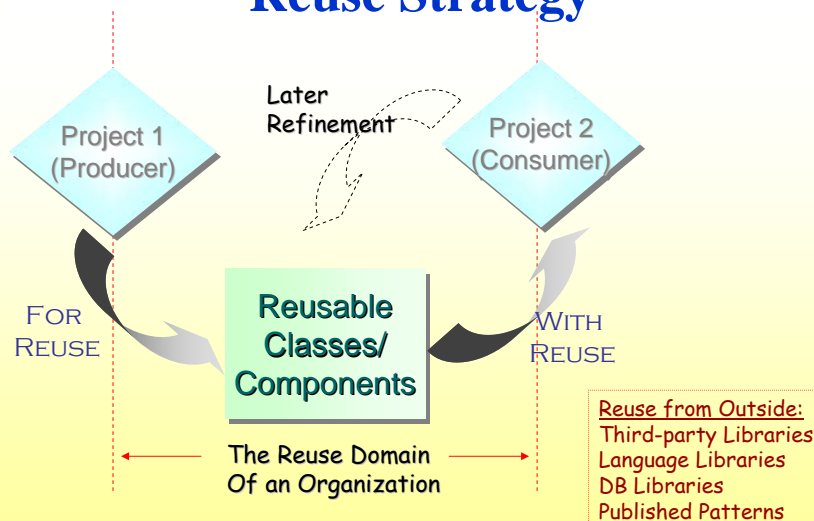
```
    private char PatientDtls;
    private bool Admitted;
}
```



"Copyright Practical OO Analysis,  
Thomson Publishing, 2005".

3

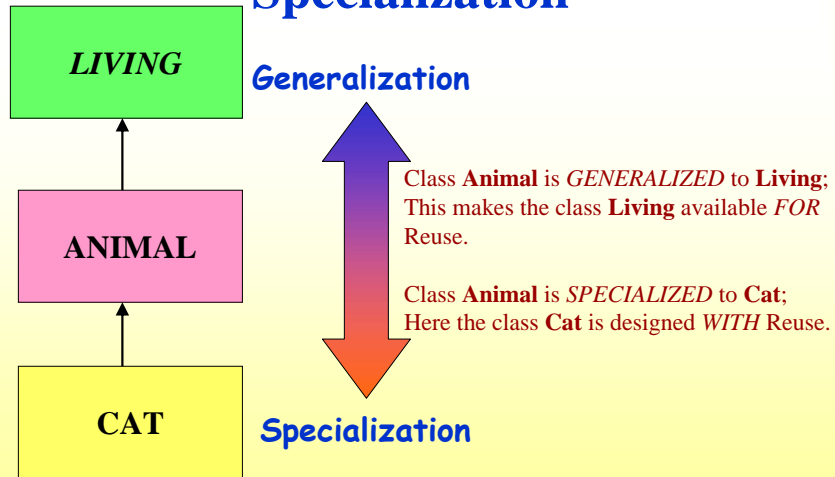
## “With” and “For” Reuse: Part of Reuse Strategy



"Copyright Practical OO Analysis,  
Thomson Publishing, 2005".

4

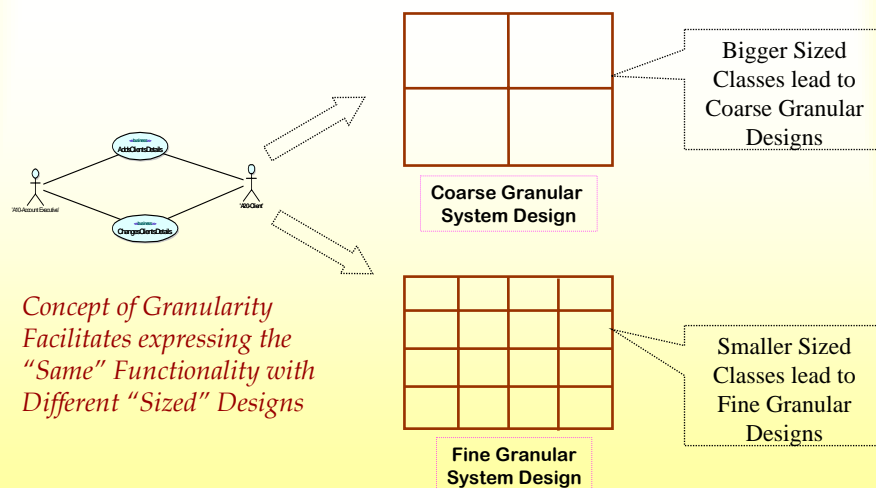
## Generalization versus Specialization



"Copyright Practical OO Analysis, Thomson Publishing, 2005".

5

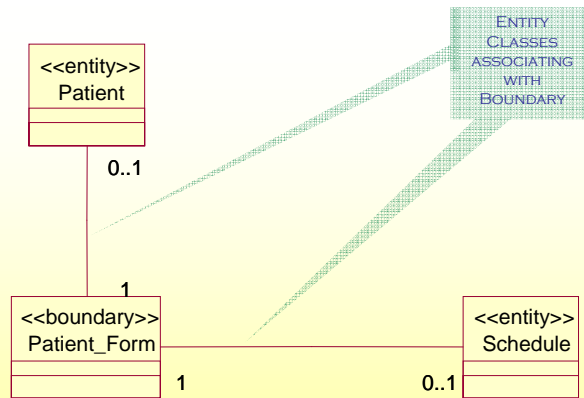
## Granularity of Object-oriented Designs



"Copyright Practical OO Analysis, Thomson Publishing, 2005".

6

# Identifying Lack of Robustness



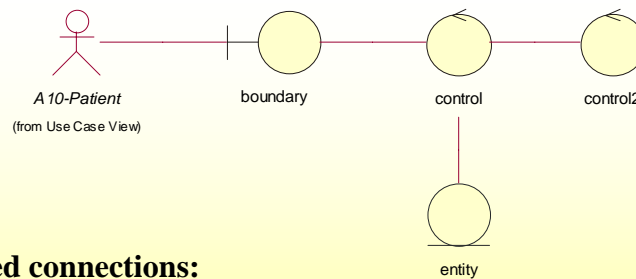
"Copyright Practical OO Analysis, Thomson Publishing, 2005".

7

# Rules of Robustness

Based on "Use Case Driven Object Modelling with UML"

Doug Rosenberg with Kendall Scott (1999, 2001)



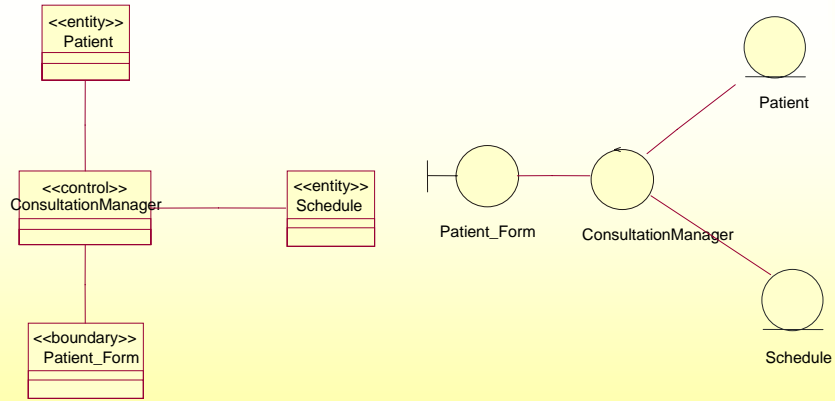
## Allowed connections:

- actor - boundary
- boundary - controller
- controller - entity
- controller - controller

"Copyright Practical OO Analysis, Thomson Publishing, 2005".

8

## Incorporating Robustness in Design

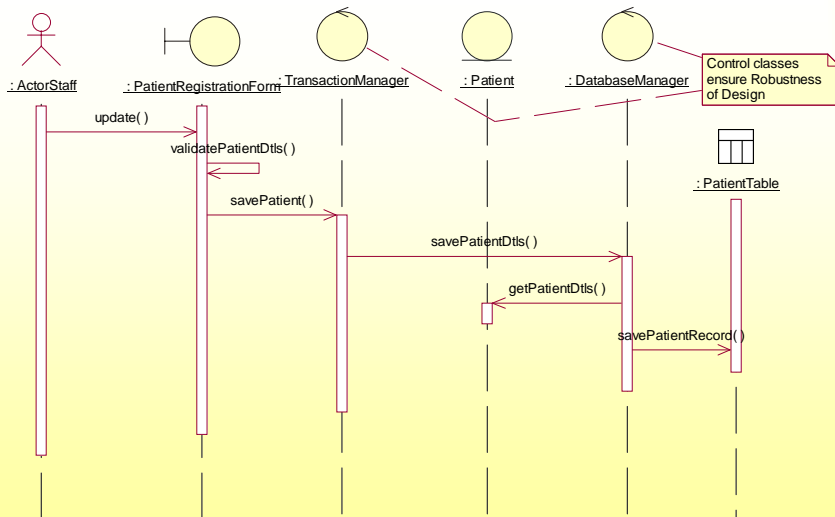


PATIENT, DOCTOR AND CONSULTATIONMANAGER WITH  
 <<CONTROL>>, <<BOUNDARY>> AND <<ENTITY>> CLASSES;

"Copyright Practical OO Analysis,  
 Thomson Publishing, 2005".

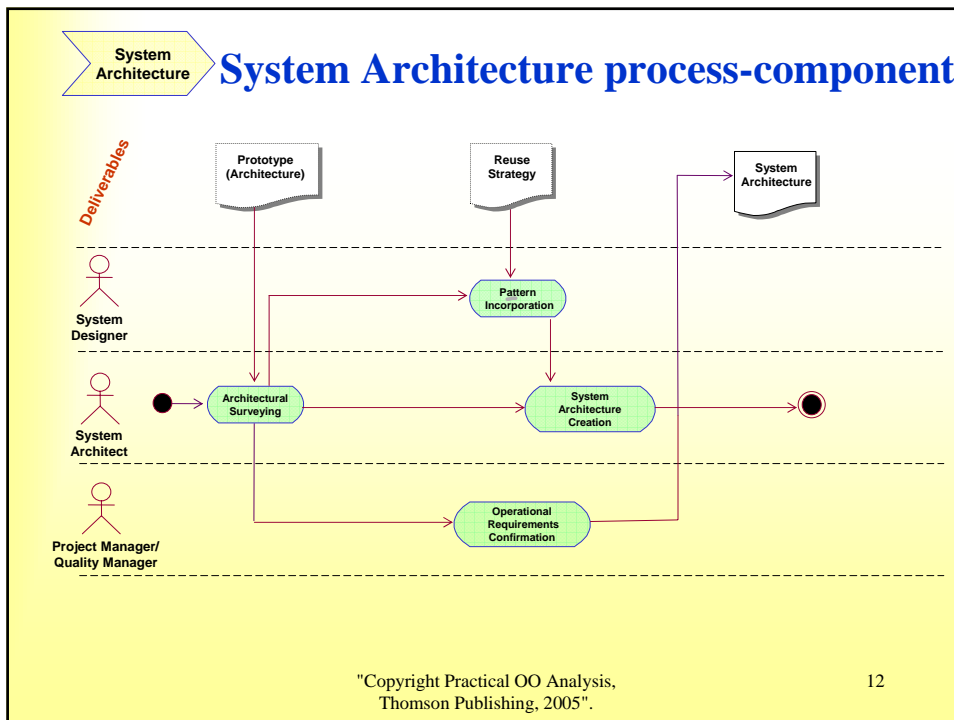
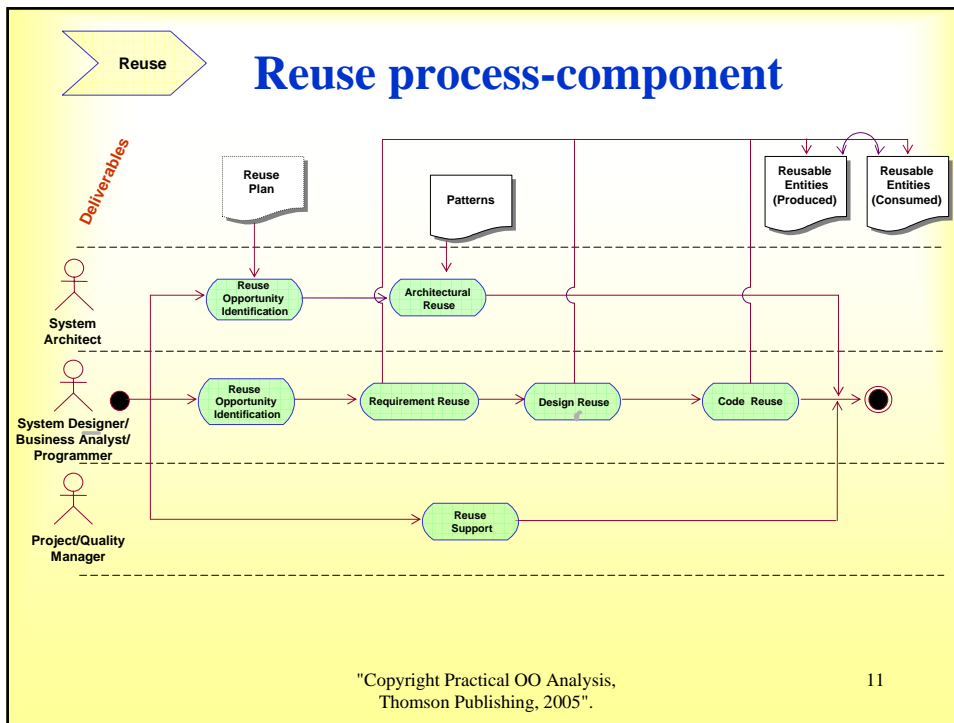
9

## Effect Of Robustness the Process of Saving Patient's Registration Details

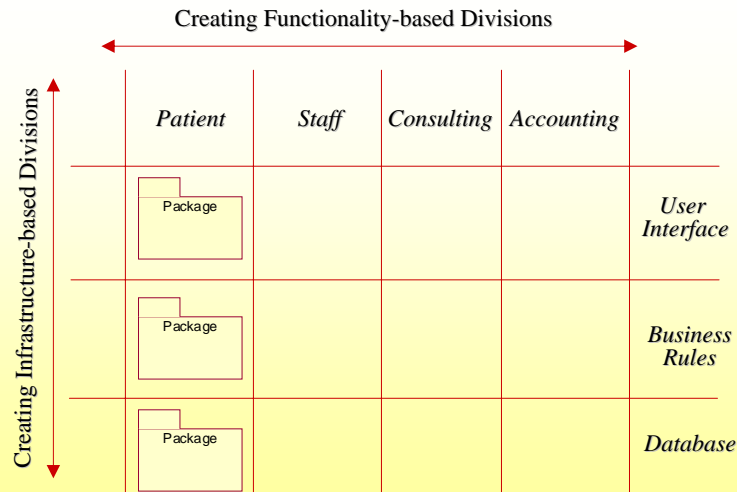


"Copyright Practical OO Analysis,  
 Thomson Publishing, 2005".

10



## Basic System Architecture Considerations



A Hospital Management System

"Copyright Practical OO Analysis,  
Thomson Publishing, 2005".

13

## Technique: Revision of inheritance hierarchies

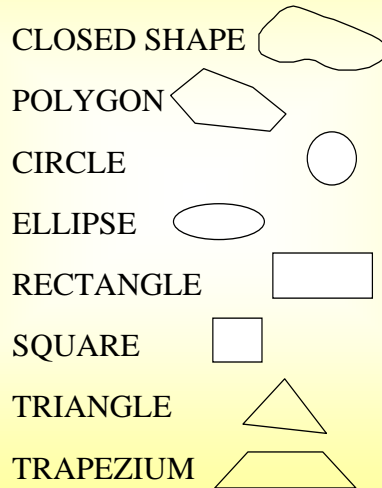
- Avoid mixing and implementation inheritance
- Avoid excessive hierarchy depths
- Possible fragmentation
- Problem if inheriting unwanted features
- Placing of services wrt level in hierarchy
- Use of single vs. multiple inheritance
- Inherit or buy?
- Conceptual model underlying inheritance hierarchy?

"Copyright Practical OO Analysis,  
Thomson Publishing, 2005".

14

## EXAMPLE

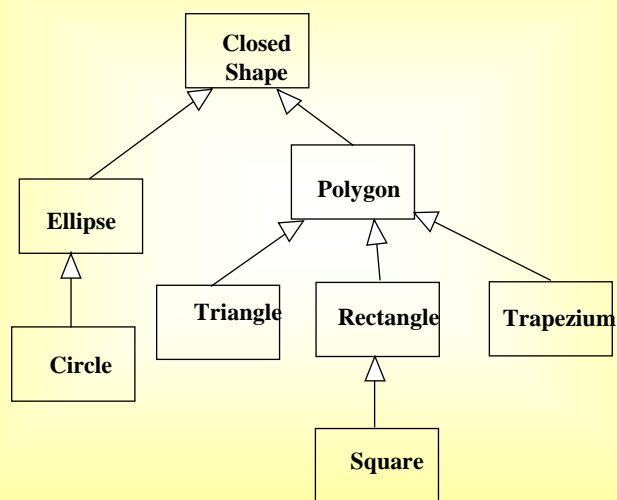
Consider inheritance hierarchy :



"Copyright Practical OO Analysis,  
Thomson Publishing, 2005".

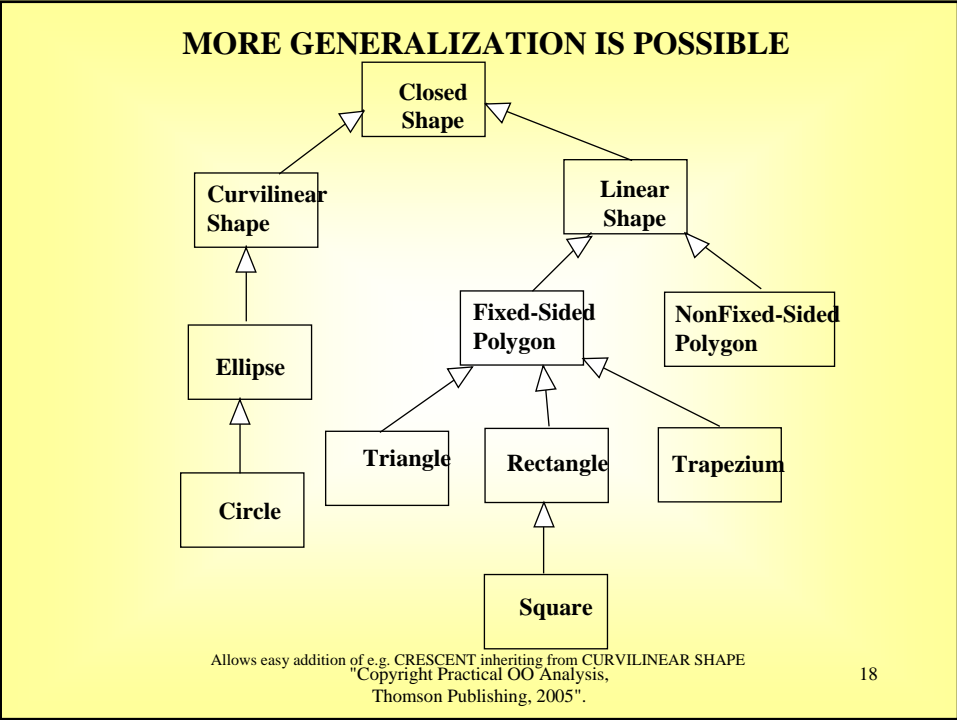
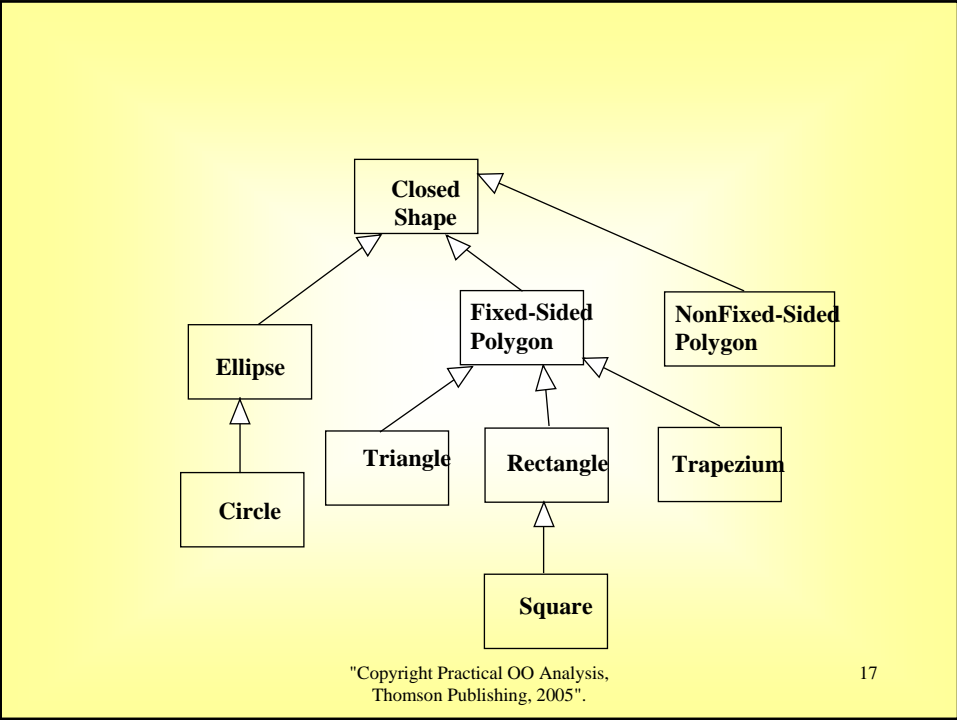
15

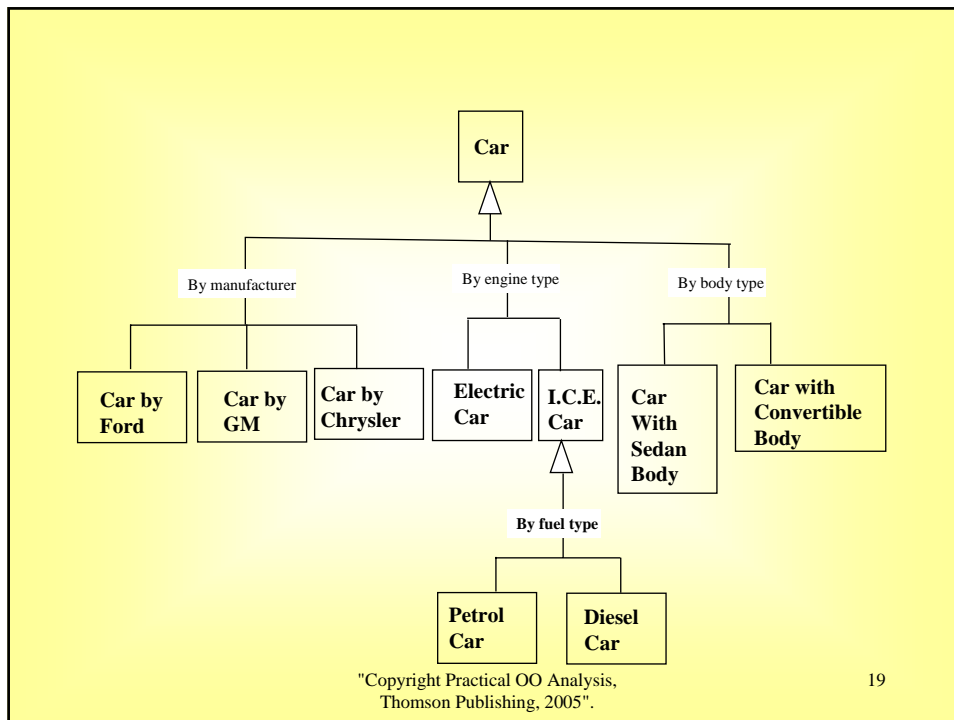
## FIRST ATTEMPT



"Copyright Practical OO Analysis,  
Thomson Publishing, 2005".

16





Inspired by Christopher Alexander's "pattern language" for buildings: *a solution to a problem in a context.*

GOF - 23 patterns in 3 categories:

- creational patterns
- structural patterns
- behavioural patterns

## Patterns should be:

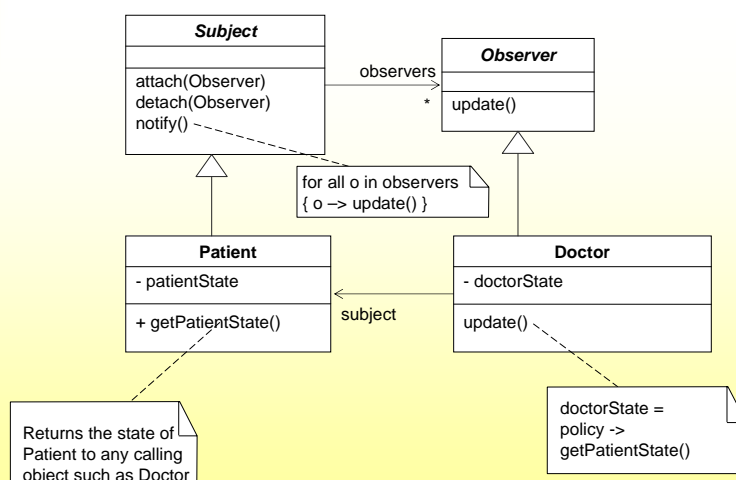
- Smart - not obvious to a beginner
- Generic - domain independent
- Well-proven - not dreamed up but gleaned from practice
- Simple
- Reusable
- Object-oriented

(Eriksson and Penker, 1998)

"Copyright Practical OO Analysis,  
Thomson Publishing, 2005".

21

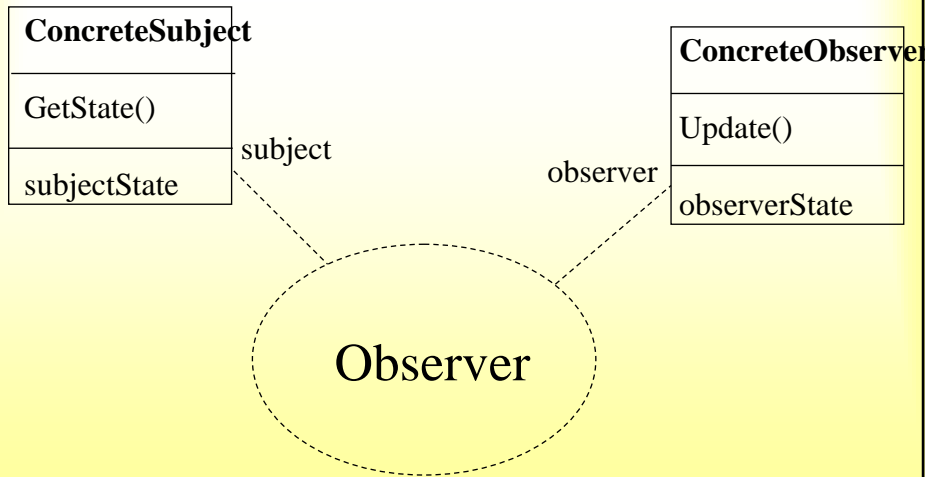
## Using a Design Pattern in HMS (Based on the Observer Pattern by Gamma et. al's work)



"Copyright Practical OO Analysis,  
Thomson Publishing, 2005".

22

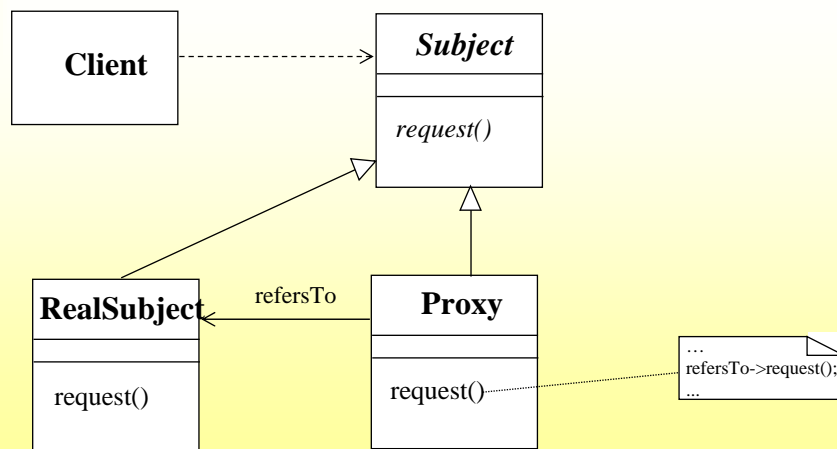
## Using pattern notation



"Copyright Practical OO Analysis,  
Thomson Publishing, 2005".

23

## Example. Proxy pattern (structural)



"Copyright Practical OO Analysis,  
Thomson Publishing, 2005".

24

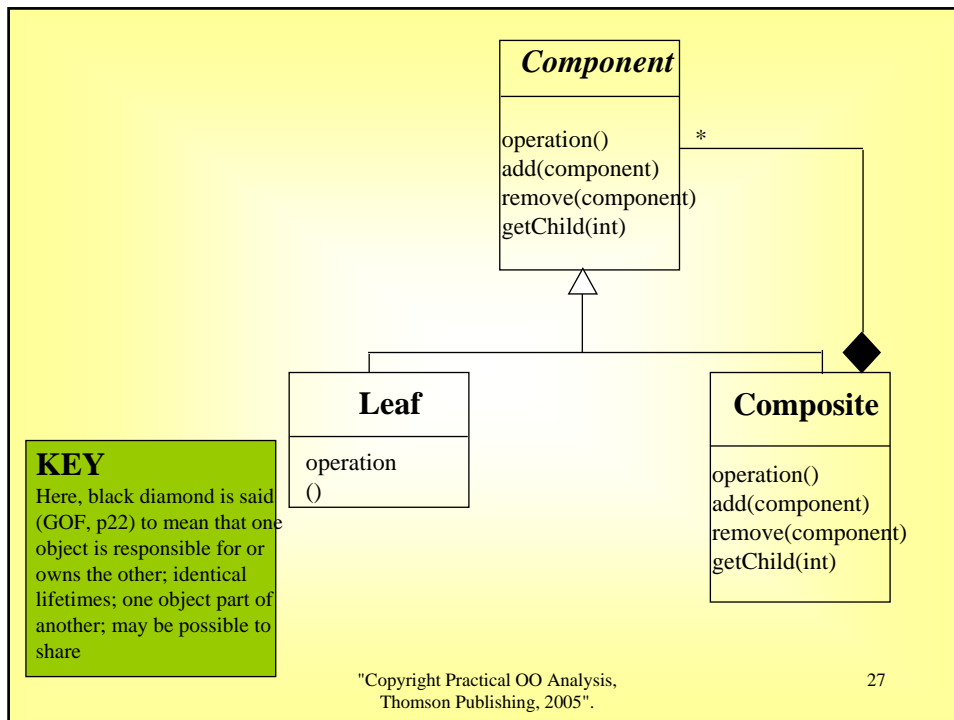
Proxy especially useful when Proxy class is cheap and RealSubject is expensive e.g. needs to be fetched from a database.

If RealSubject exists, Proxy refers to it; else it instantiates it and then refers the request.

## **Example: Composite Pattern**

- Creates tree structure for whole-part hierarchy. Clients are now able to treat both the whole and the part uniformly.
- - one of a group of Structural Patterns

(GOF, p8)



## Consequences

- Composition structure is recursive. When client expects a primitive object and gets a composite object, all is OK (but note not the reverse).

[But note that if client invokes add (component) and current component is leaf, it won't work, so this would appear to be implementation inheritance in which Leaf negates some inherited operations]

- makes client simple - can treat composites and primitives identically
- easy to add new kinds of components
- can make design overly general since there are no constraints on the *types* of object that can be plugged into the hierarchy

## Conclusions

- Discussed the important concepts of Reusability and Granularity in OO Modelling
- Discussed Patterns in Design

## WORKBOOK



## EXERCISES

*For The Class Room!!*

## Workbook Exercises

1. Draw a class diagram and explain 'for' and 'with' reuse in it.
2. Discuss how a pattern can be incorporated in design.
3. Demonstrate your understanding of granularity by drawing two class diagrams – one for granularity and another coarse granular.
4. Which are the most important activities in Reuse and System Architecture process component.
5. Identify two crucial Entity objects whose state change is important from both a functional and design viewpoint.
6. Draw the corresponding State Machine Diagrams for these two objects showing the start state, the various transitions, and one or more end states.
7. Comment the diagram with notes for understandability.
8. Repeat steps 5 to 7 for one <<boundary>>, <<control>> and <<table>> object each.