

OBJECT-ORIENTED MODELLING (OOM) - 32536

MODULE 4

UML – Advanced Class Diagrams
(POOA – Chapter 6;
POOD – Chapter 3)

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

1

Module Outline

- Philosophy of Class modelling
- Major Ingredients of Class Diagram
- Relationships in Class Diagram:
 - Inheritance, Association & Aggregation
- Multiplicities in Class Diagram

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

2

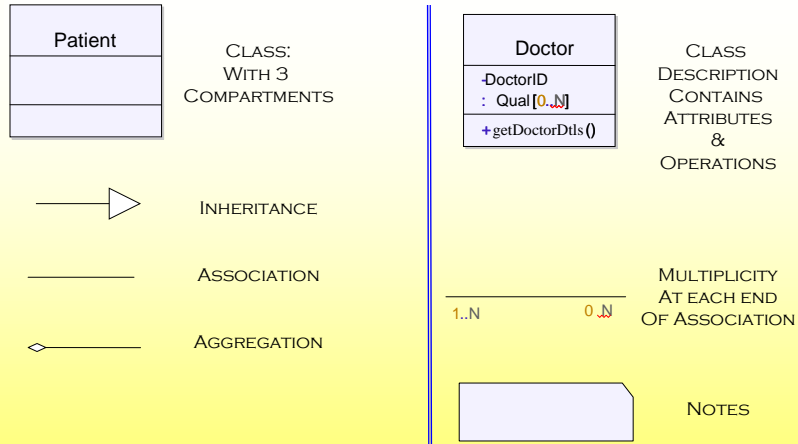
What makes a good Diagram?

- Can you understand the intention of the diagram? What is it trying to explain?
- Identify the audience
- Find the right level of abstraction
- Often useful to have the same concepts presented in different ways - details switched on or off
- It comes down to dependencies
- Diagrams can be linked and notes added

Some Useful Approaches in Class modelling

- Responsibility identification
- Service identification
- Relationship modelling
- Collaborations analysis
- Contract specification
- Event modelling
- Rule modelling

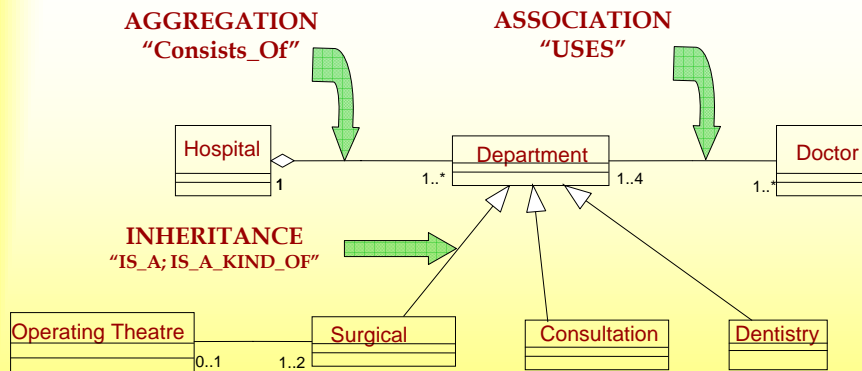
Major Ingredients of a Class Diagram



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

5

Relationships on a class diagrams



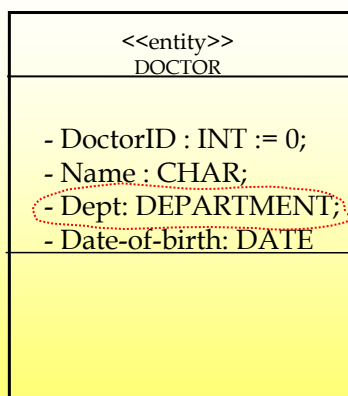
"Copyright Practical OO Analysis,
Practical OO Design, Thomson
Publishing, 2005"

6

Inheritance, Association, Aggregation, Multiplicity & Notes

- Inheritance: The lower level classes are kind of higher level classes.
- Association: A class using another class in some way.
- Aggregation: A special kind of association. A closer relationship between classes.
- Multiplicity: An object of one class relates to how many objects of the other class?
- Notes: A free formatted description.

Code Example 3.1: In Doctor class for Association Relationship with Department



```
class Doctor {  
    public:  
  
    ...  
    private: □  
        INT *PatientID;  
        CHAR *name;  
        DEPARTMENT *dept;  
        DATE Date-of-birth;  
};
```

Code Example 3.2: In both Department and Doctor classes for bi-directional Association Relationship

```
class Department {
private:
    INT *DeptID;
    CHAR *DeptName;
public:
    DOCTOR *aDoctor;
    aDoctor.getDocID();

+getDeptName() {
    -code to get DeptName --};
};
```

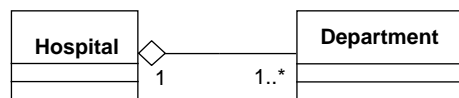
```
class Doctor {
public:
    DEPARTMENT *aDept;
    aDept.getDeptName();
+getDocID(){---
    -code to get DocID --};
private:
    INT *DoctorID;
    CHAR *Name;
    CHAR Address;
    DATE Date-Joined;
};
```

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

9

“By Reference” Aggregation in Design

- In Implementing “By Reference”, only a reference to the Doctor object is used. Therefore, it is possible for other Department objects to also share (point to) the same Doctor.
- *Note: Association is also implemented “By Reference”*



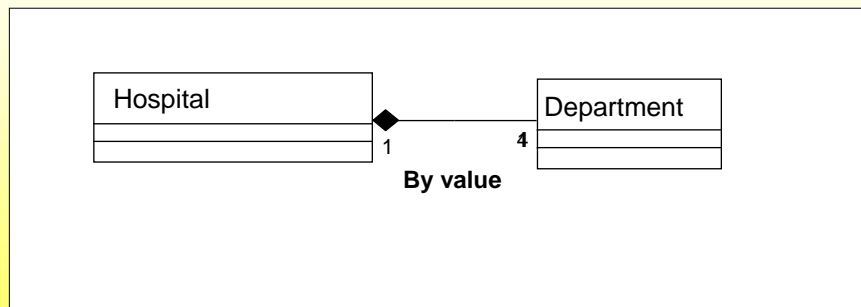
By reference

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

10

“By Value” Aggregation in Design

- In Implementing “By Value”, a copy of the aggregate parts (e.g. Room) is made. The “contained” rooms cannot be shared by other Departments.



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

11

Code Example 3.5: for Inheritance Relationship

```
public class Department {
    public void setDeptDtls() { };
    public void changeDeptDtls() { };

    private char DeptID;
    private double Capacity;
}

-----
public class Surgical extends Department {
    public Surgical (); {
    }
    public void operate(); { }
    public void bookOperation (); { }
    public void releaseTheatre(); { }

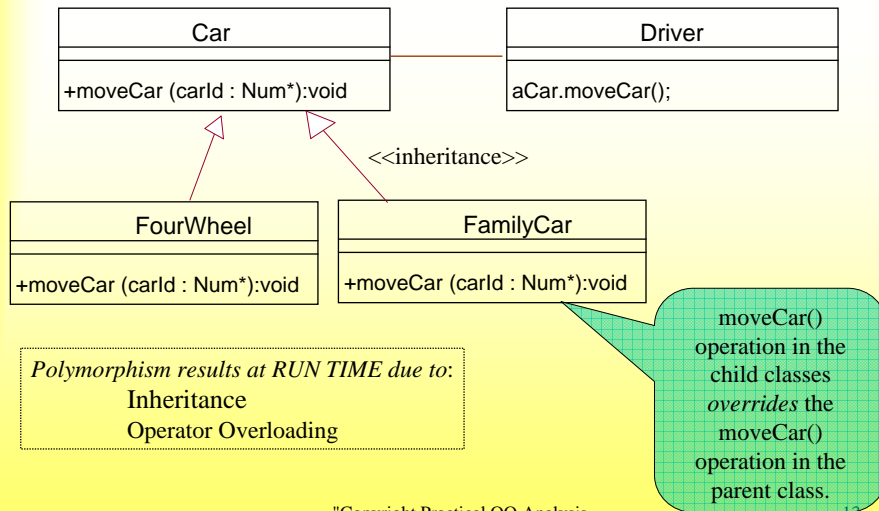
    private char Name;
    private int NoOfRoom;
}

-----
```

12

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

Polymorphism (through Operation Overriding)



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

13

Code Example 3.6: for Polymorphism

```

Class Driver:
  Car myCar;
  myCar := new FourWheel;
  myCar.moveCar();

  myCar := new FamilyCar;
  myCar.moveCar();
  
```

This pseudo code indicates that in when the driver object issues a call to moveCar, whichever Object has been instantiated (either FourWheel or FamilCar) will move.

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

14

Operations

Can also have:

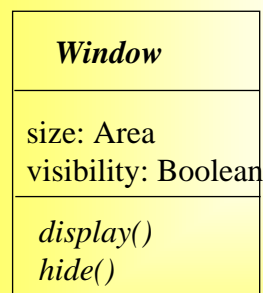
- isQuery (attribute)
- visibility
- scope
- parameters
- return type
- concurrency semantics
- other properties

Various Forms

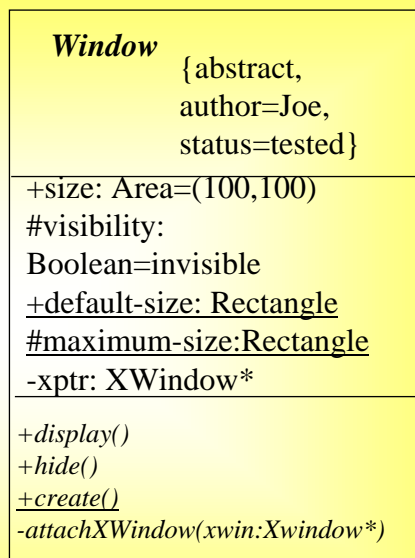
suppressed



analysis level



implementation level



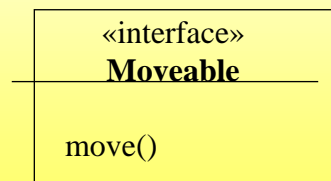
+ optional names for lists

UML Interfaces

- show some or all of the operations in a class (“some” here means that a class can have several interfaces as in Java)
- specify a service
- each interface represents a role that object plays (Booch *et al.*, 1999, p161)
- operations shown are all visible outside class
- may be realized by a class or a component
- define a contract

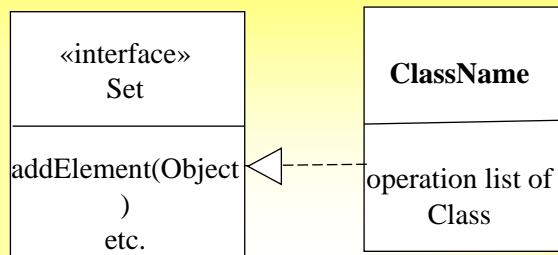
Interfaces (continued)

- has no attributes, associations (i.e. no state) or methods (i.e. no internal structure)
- has no direct instances
- programming focussed, not useful for specification (Cheesman and Daniels, 2001)

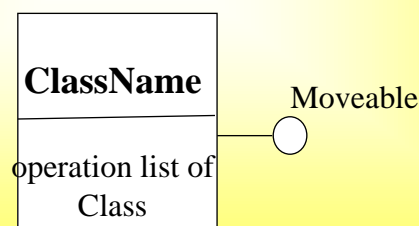


Interface is formally equivalent to an abstract class with no attributes and methods and only abstract operations
(OMG, 2001, p3-51)

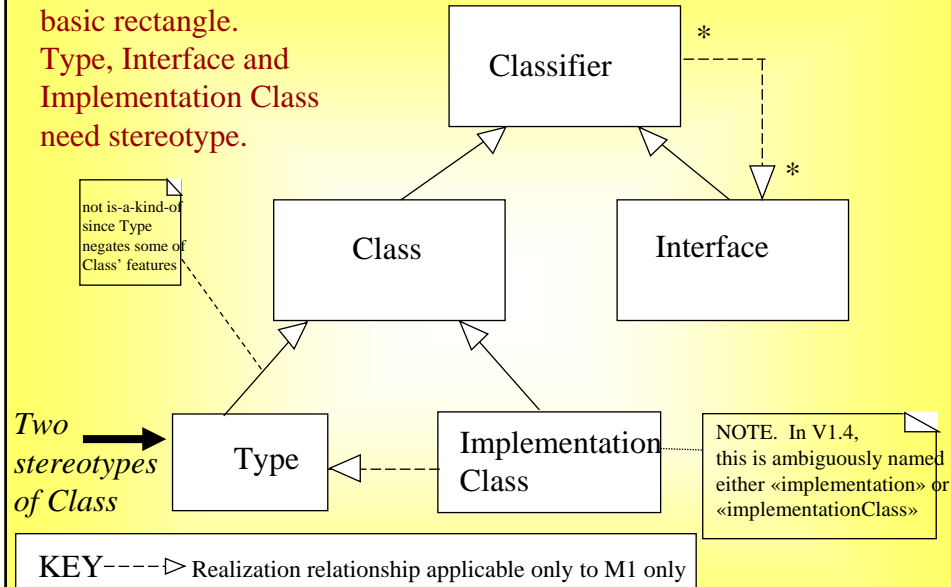
A class realizes an interface



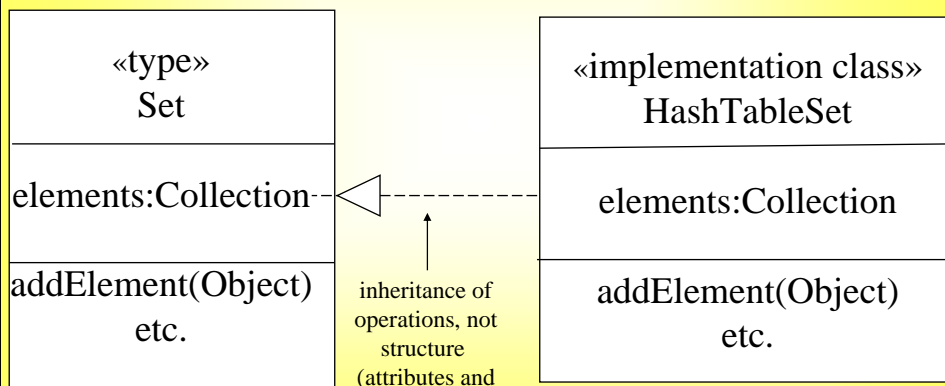
or



Classifier and Class have basic rectangle.
 Type, Interface and Implementation Class need stereotype.

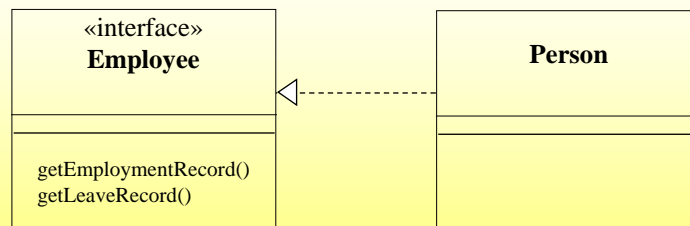


UML Classes can be stereotyped as Types or Implementation class



Roles

Interface represents a role being played e.g.



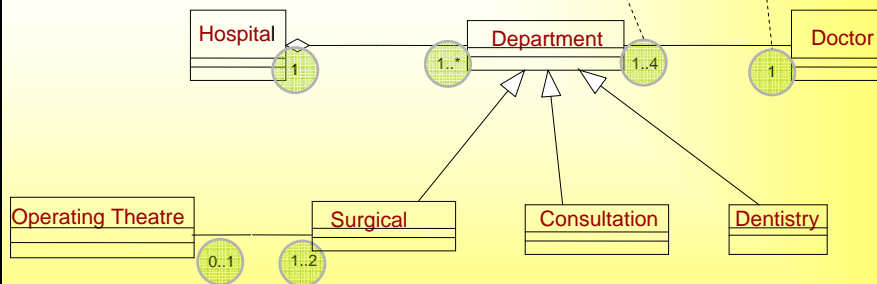
Sub-Module

Multiplicities

Provide Business Rules;
Invaluable in DB Design.

Multiplicities on class Diagrams

Multiplicities:
An Object of one Class Relates
to How Many Objects of the
Other Class?



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

25

Multiplicities on class diagrams

- Multiplicities encompass Business Rules
 - They are for Objects belonging to Classes
 - They are useful for Database Design
- Multiplicity states:
 - For two Classes in an Association/Aggregation relationship
 - “An Object of one Class deals with How many Objects of Another Class”
- A Car object associates with Four Wheels
 - Then multiplicity will be 1 for Car and 4 for Wheel
- **Note:** Multiplicities are meaningless for Inheritance relationship

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

26

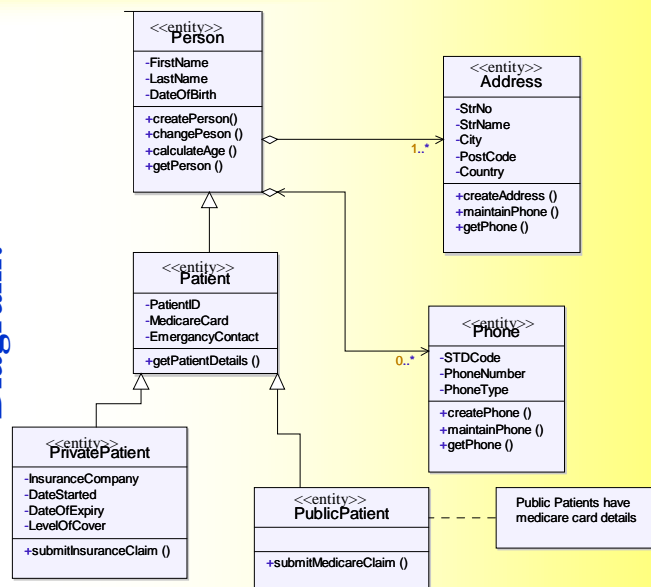
Sub-module

Class diagrams for hospital management system

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

27

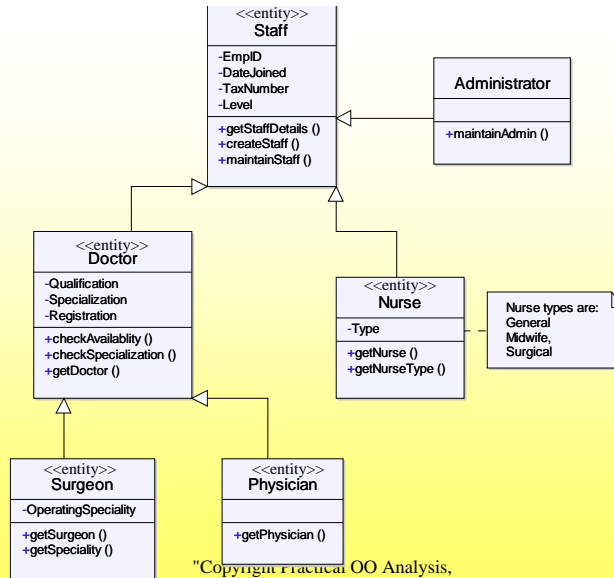
“Patient Details “ Class Diagram:



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

28

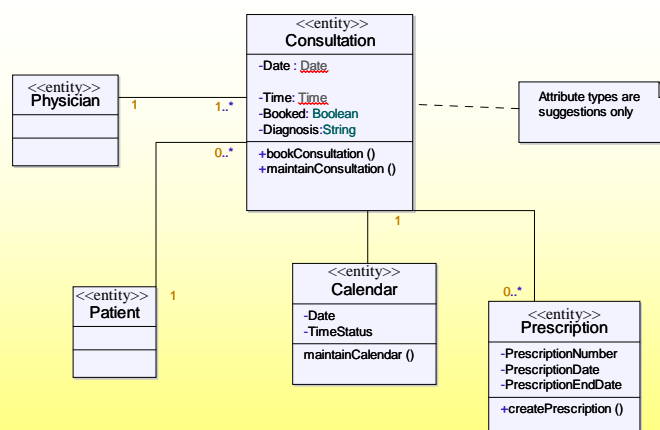
“Staff Details “ Class Diagram



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

29

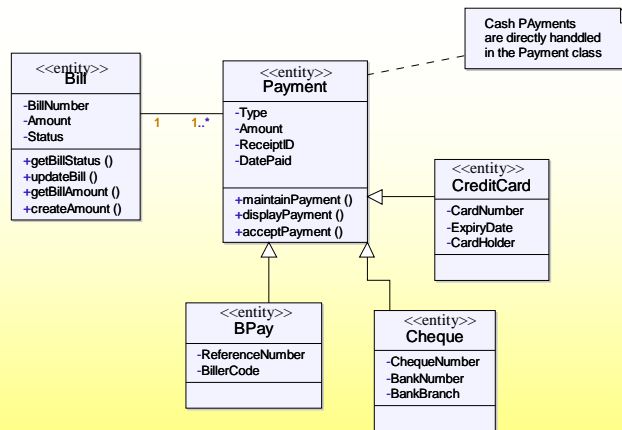
“Consultation Details “ Class Diagram



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

30

“Accounting “ Class Diagram



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

31

Conclusions

- **Class Diagrams show**
 - ✓ Classes
 - ✓ Attributes of Classes
 - ✓ Operations of Classes
 - ✓ Relationships among the Classes.
- **Multiplicities are Important, as they show Business Rules**
- **Stereotypes and Notes add value to the Diagrams**

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

32

WORKBOOK



EXERCISES

For The Class Room!!

Workbook Exercises

1. Inspect the list of classes documented in the last lecture.
2. Incorporate association, inheritance and aggregation.
3. Add multiplicity to either end of the association relationship.
4. Create/add an inheritance hierarchy including an Abstract class.
5. Spot the possibility of creating polymorphic design.
6. Be sure to add explanatory notes when needed.



PROJECT WORK : **(DURING TUTORIALS IN THE LAB);**

Follow the Project Work Requirements given at the End of
the Chapter;
Discuss Project Work with Team Members and Tutor;
Carry Out the Project Work