

# ***Pertemuan 10***

## ***Class Diagram***

# *Class Diagram*

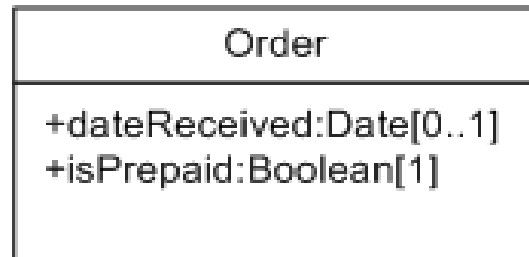
Class Diagram mendeskripsikan jenis-jenis objek dalam sistem dan berbagai macam hubungan statis yang terdapat diantara mereka. Class diagram juga menunjukkan properti dan operasi sebuah class dan batasan-batasan yang terdapat dalam hubungan-hubungan objek tersebut. UML menggunakan istilah fitur sebagai istilah umum yang meliputi properti dan operasi sebuah class.

# *Properti, Atribut dan Asosiasi*

## *(lanjutan)*

Asosiasi merupakan sebuah garis solid antara dua class, ditarik dari class sumber ke class target. Nama properti bergerak sampai tujuan akhir sebuah asosiasi bersama dengan multiplicity. Tujuan akhir sebuah asosiasi menghubungkan dengan class yang merupakan jenis properti.

Properti dalam susunan atribut dapat digambarkan sebagai berikut:



# *Multiplicity*

Multiplicity merupakan indikasi tentang berapa banyak objek yang akan mengisi properti. Multiplicity yang sering digunakan adalah:

- 1 (contoh: satu pesanan hanya bisa untuk seorang pelanggan)
- 0..1 (contoh: pelanggan perusahaan dapat memiliki seorang sales rep)
- \* (contoh: tidak ada jumlah maksimal / tidak terbatas berapa jumlah pesanan yang dapat dibuat oleh pelanggan)

# *Generalisasi*

Contoh dari gambar class sebelumnya yang merupakan generalisasi melibatkan pelanggan perorangan dan pelanggan perusahaan. Keduanya mempunyai persamaan dan perbedaan. Persamaan tersebut dapat dimasukkan kedalam class pelanggan umum (supertype) dengan pelanggan perorangan dan pelanggan perusahaan sebagai subtype.

Dengan menggunakan perspektif perangkat lunak, interpretasi tersebut sudah termasuk: pelanggan perusahaan merupakan subclass dari pelanggan. Dalam object oriented subclass mewarisi semua fitur superclass dan dapat melakukan semua metode superclass.

# System Analysis and Design with UML

## 1. System Analysis

1. Business Process Identification
  - **Use Case Diagram**
2. Business Process Modeling
  - **Activity Diagram** or Business Process Modeling Notation (BPMN)
3. Business Process Realization
  - **Sequence Diagram** (Buat untuk setiap use case dengan menggunakan pola **Boundary-Control-Entity**)

## 1. System Design

1. Program Design
  1. **Class Diagram** (Gabungkan **Boundary-Control-Entity** Class dan susun story dari sistem yang dibangun)
  2. **Package Diagram** (Gabungan class yang sesuai, boleh menggunakan pola B-C-E)
  3. **Deployment Diagram** (arsitektur software dari sistem yang dibangun)
2. **User Interface Design** (Buat UI design dari **Boundary Class**)
3. **Entity-Relationship Model** (Buat ER diagram dari **Entity Class**)

# Class Diagram Elements

1. Classes
2. Attributes
3. Operations
4. Relationships

# Classes

- **Templates** for creating instances or objects
- All objects of a class have same structure and behavior, but contain **different attributes**
  1. **Concrete**: used to create actual objects
  2. **Abstract**: extended to create other classes



# Attributes

- Units of information relevant to the **description** of the class
- Only **attributes important** to the task should be included
- Attributes should be **primitive types** (integers, strings, doubles, date, time, Boolean, etc.)

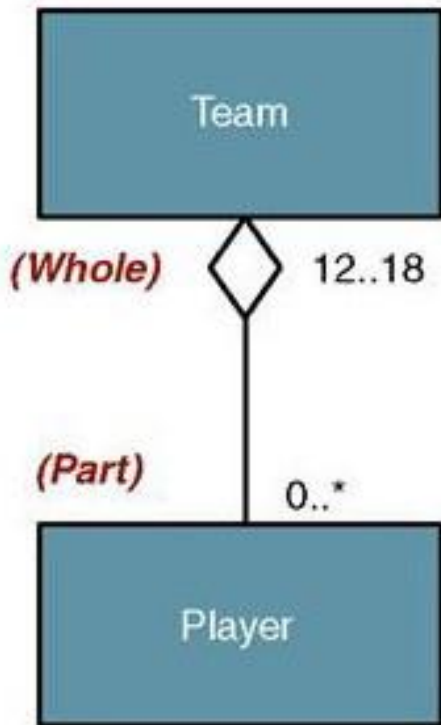
# Operations (Methods)

- Defines the **behavior** of the class
- **Action** that instances/objects can take
- Focus on relevant problem-specific operations (at this point)

# Relationships

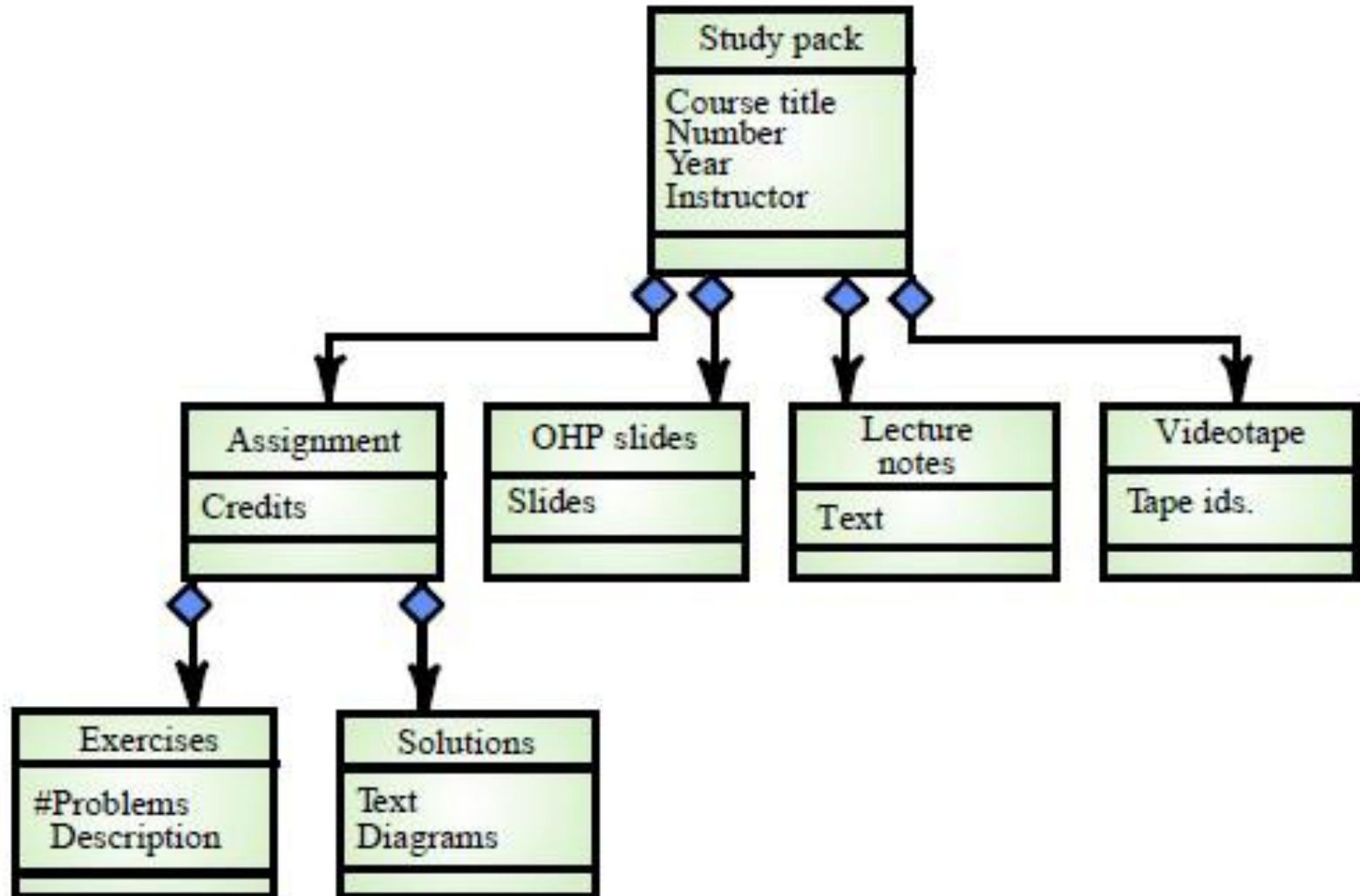
- **Generalization**
  - “Is-A” relationship
  - Enables inheritance of attributes & oper's
  - Subclasses and superclasses
  - Principle of substitutability
    - Subclass be substituted for superclass
- **Aggregation**
  - “Has-A” relationship
  - Relates parts to wholes
  - Uses decomposition
- **Association**
  - Relationships that don't fit “Is-A” or “Has-A”
  - Often a weaker form of “Has-A”
  - Miscellaneous relationships between classes
  - Example:
    - Patient schedules an appointment
    - So the appointment has a patient
    - This is weak

# Agregasi



- Agregasi – sebuah relasi yang menyatakan bahwa satu kelas “utuh (*whole*)” yang lebih besar memuat satu atau lebih kelas “bagian (*part*)” yang lebih kecil. Sebaliknya, kelas “bagian” adalah bagian dari kelas “utuh”
- Dalam UML 2.0 notasi agregasi sudah tidak dipakai lagi

# Contoh Agregasi



# Contoh Agregasi

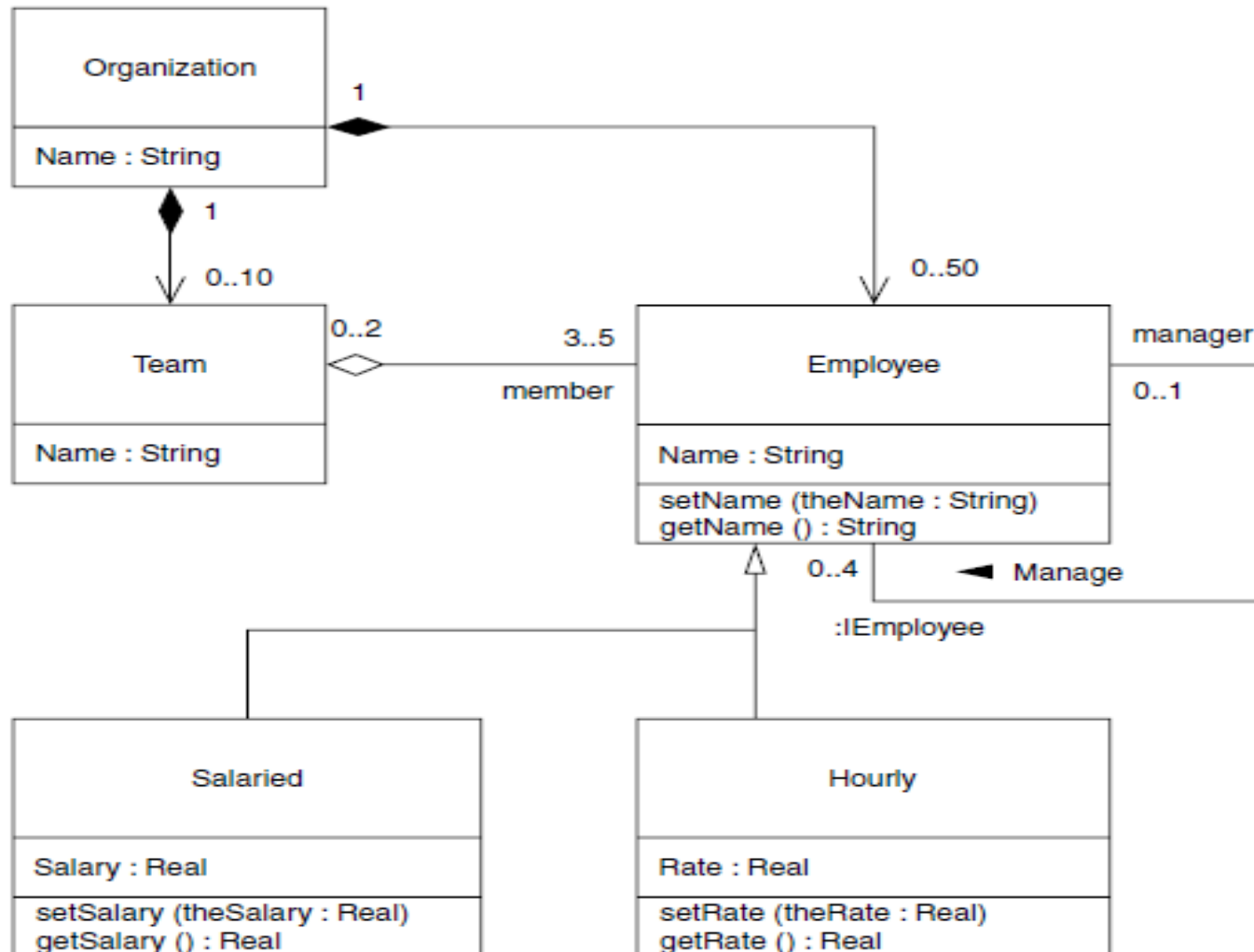
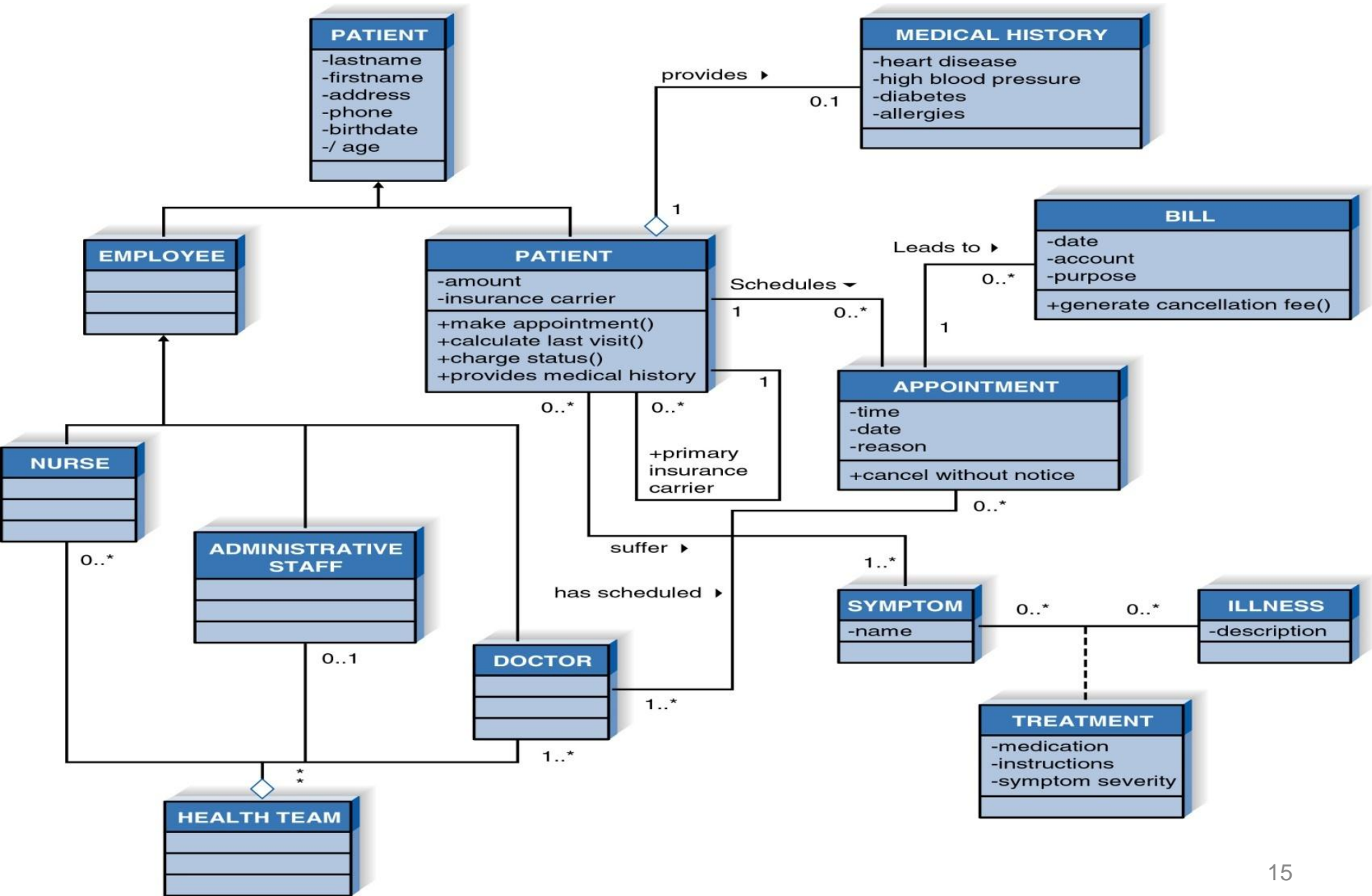


FIGURE 3.10. Aggregations, compositions, and generalizations between classes.

# Example Class Diagram



# More on Attributes

- **Derived attributes**
  - /age, for example can be calculated from birth date and current date
- **Visibility of attributes**
  - **+Public**: not hidden from any object
  - **#Protected**: hidden from all but immediate subclasses
  - **-Private**: hidden from all other classes
- **Default is private**









# More on Operations

- **Constructor**: creates object
- **Query**: see class state
- **Update**: change attribute values
- Operations can also be public, protected, or private
  - Default for operations is public

# More on Relationships

- A primary purpose of class diagrams is to show relationships, or *associations*, between classes
- Class can be related to itself (role)
  - Use a "+" sign to show it's a role and not the name of a relationship

# Relationship Multiplicity

Exactly one	 <pre>graph LR; Dept[Dept] --- 1  Boss[Boss]</pre>
Zero or more	 <pre>graph LR; Employee[Employee] --- 0..*  Child[Child]</pre>
One or more	 <pre>graph LR; Boss[Boss] --- 1..*  Employee[Employee]</pre>
Zero or one	 <pre>graph LR; Employee[Employee] --- 0..1  Spouse[Spouse]</pre>
Specified range	 <pre>graph LR; Employee[Employee] --- 2..4  Vacation[Vacation]</pre>
Disjoint ranges	 <pre>graph LR; Employee[Employee] --- 1..3, 5  Committee[Committee]</pre>

# Class

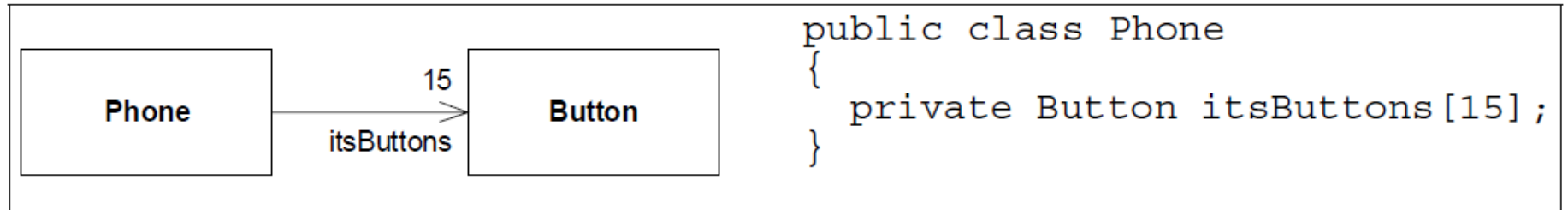
**Dialler**

```
public class Dialler  
{  
}
```

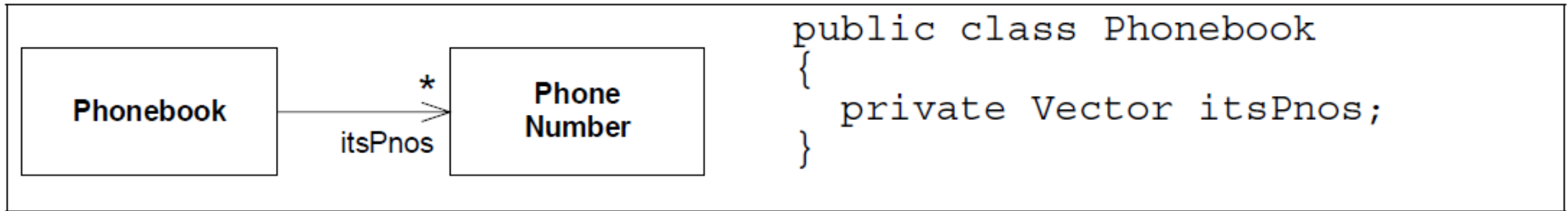
# Class with Attribute and Method

<table><tr><th>Dialler</th></tr><tr><td>- digits : Vector - nDigits : int</td></tr><tr><td>+ digit(n : int) # recordDigit(n : int) : boolean</td></tr></table>	Dialler	- digits : Vector - nDigits : int	+ digit(n : int) # recordDigit(n : int) : boolean	<pre>public class Dialler {     private Vector digits;     int nDigits;     public void digit(int n);     protected boolean recordDigit(int n); }</pre>
Dialler				
- digits : Vector - nDigits : int				
+ digit(n : int) # recordDigit(n : int) : boolean				

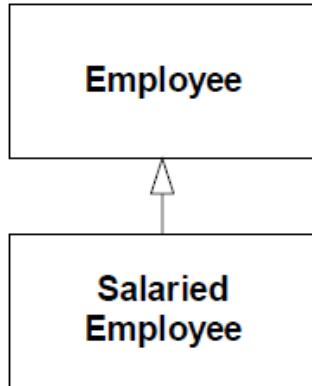
# Class Association



# Multiplicity



# Inheritance

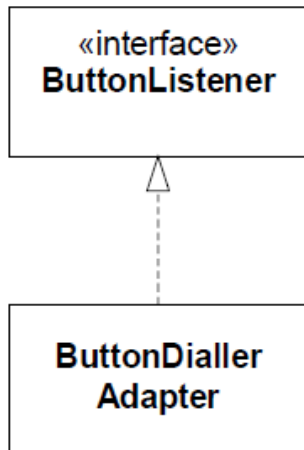


```
public class Employee
{
    ...
}

public class SalariedEmployee extends Employee
{
    ...
}
```



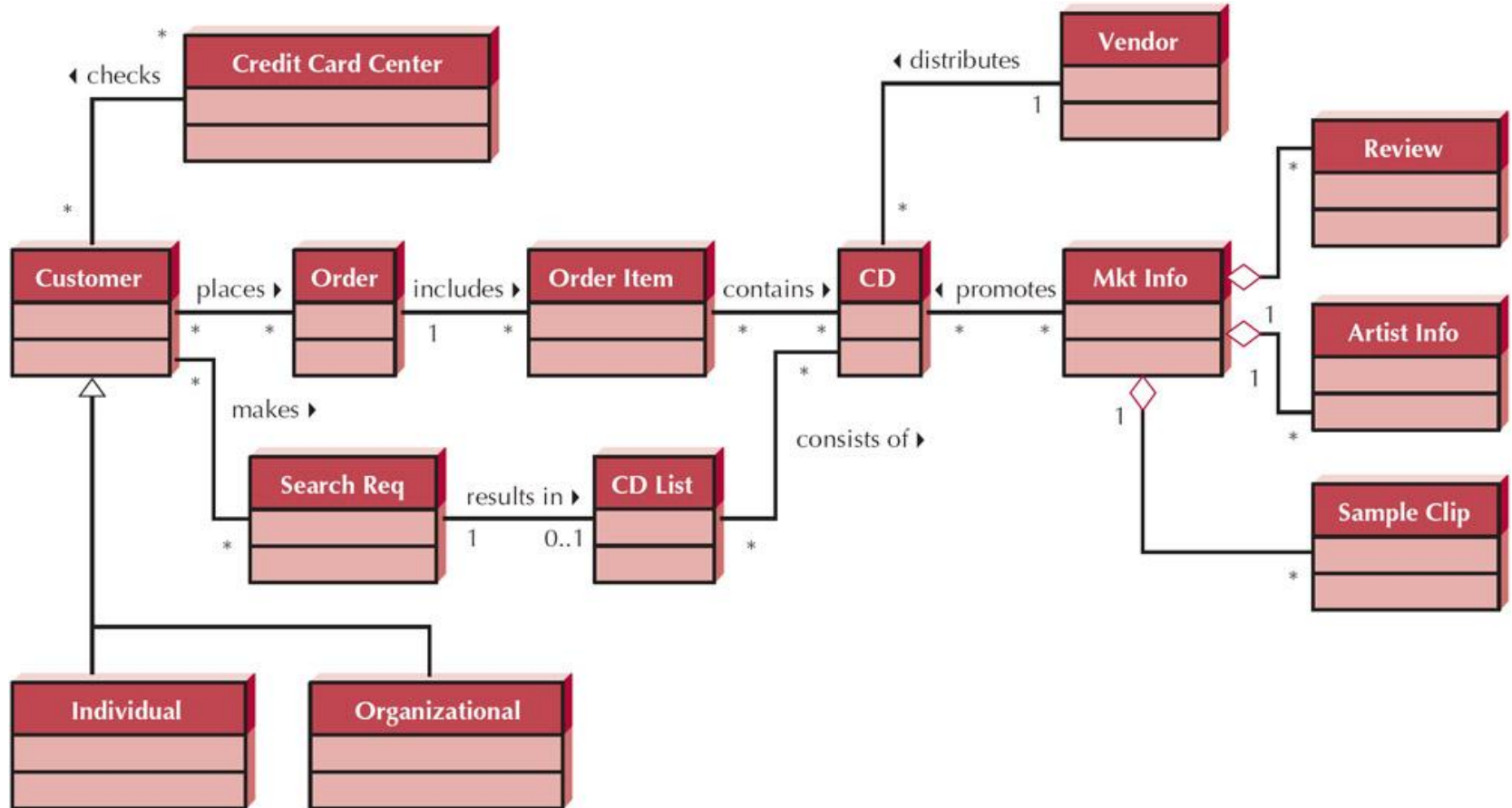
# Interface



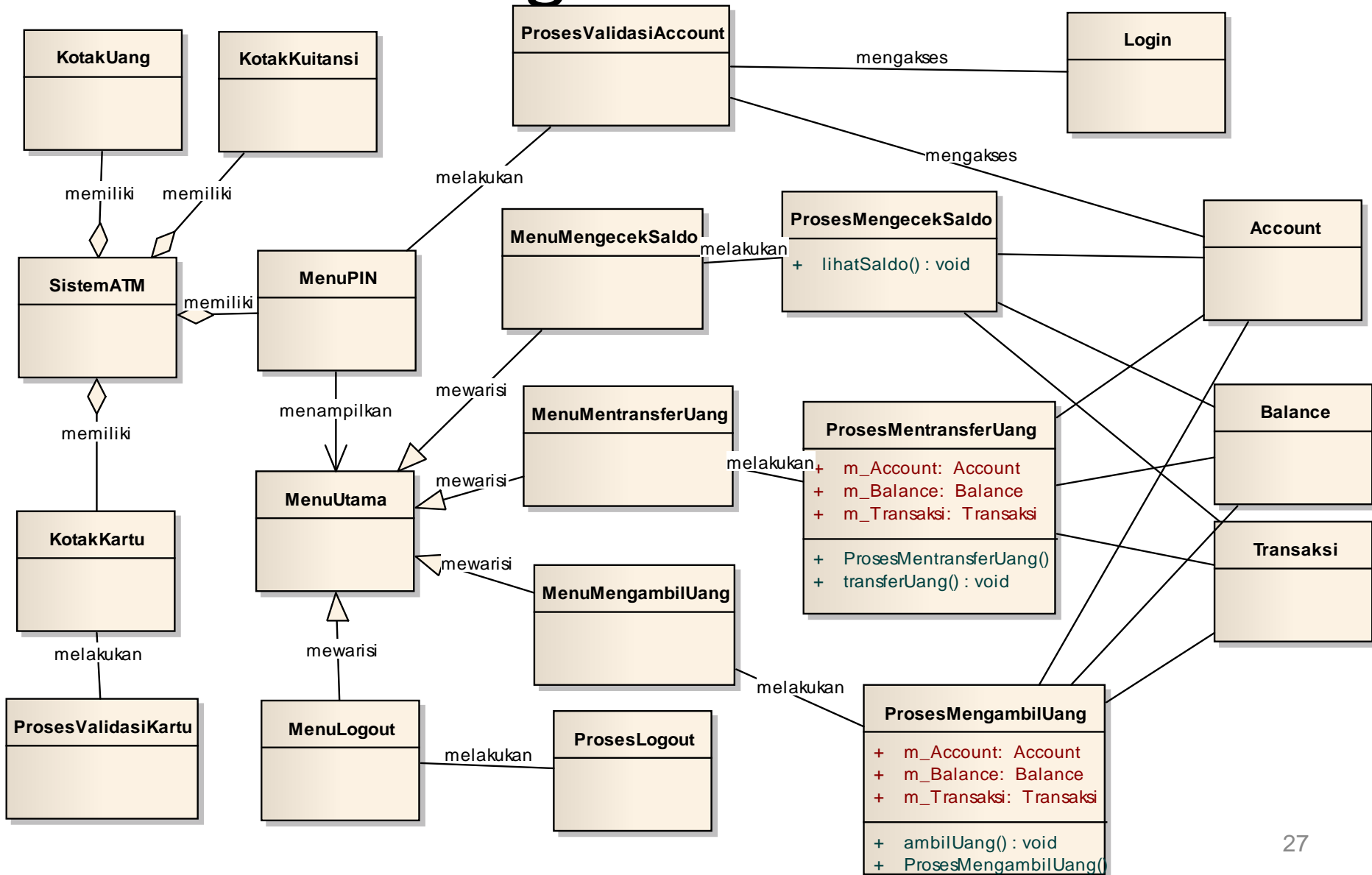
```
interface ButtonListener
{
    ...
}

public class ButtonDiallerAdapter
    implements ButtonListener
{
    ...
}
```

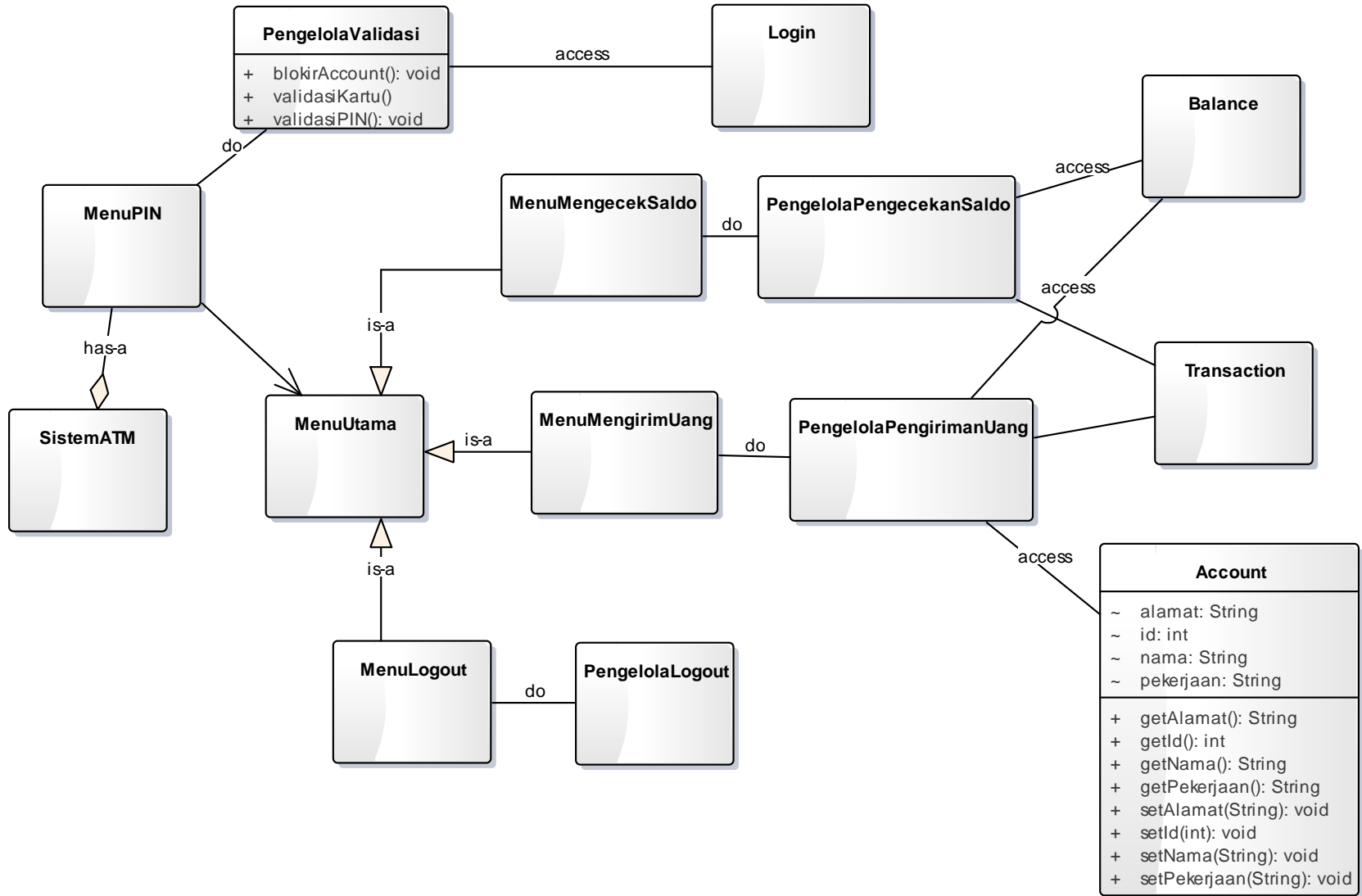
# Class Diagram: Internet Order Project



# Class Diagram: Sistem ATM



# Class Diagram – Case Study



# **DATA MODEL (ERD)**

# Data Model Sistem ATM

