Diagrams

Chonnam National University
School of Electronics and Computer Engineering

Kyungbaek Kim

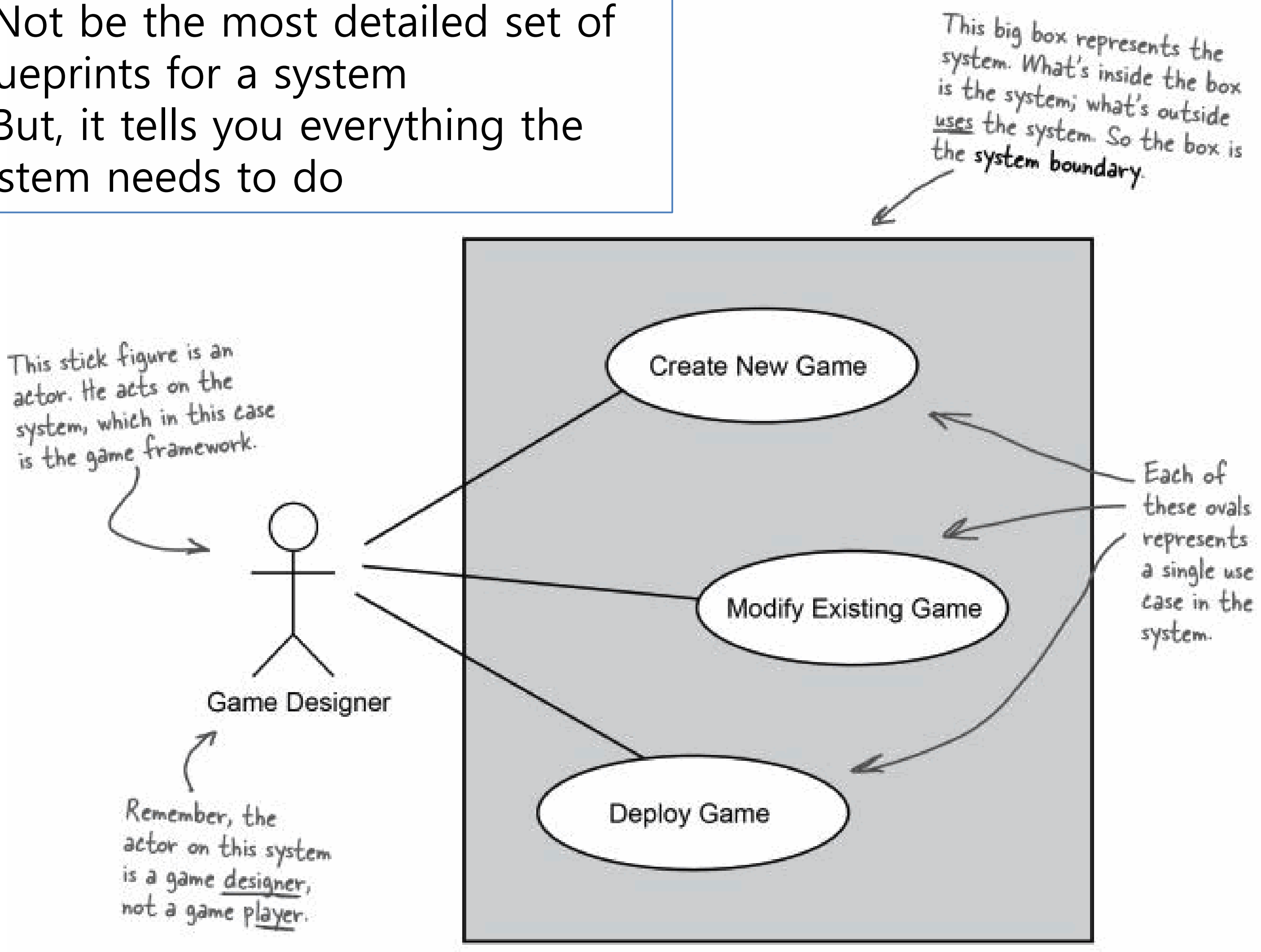
Contents

Usecase and Usecase Diagram

- Usecase
  - A case or way of using a module or method
- Usecase Diagram
  - A set of modules or methods of a system
  - The baseline of functional requirements of a system
  - Usually generated at the start up period of a project
  - Usually becomes a basis of other diagrams

Usecase Diagram

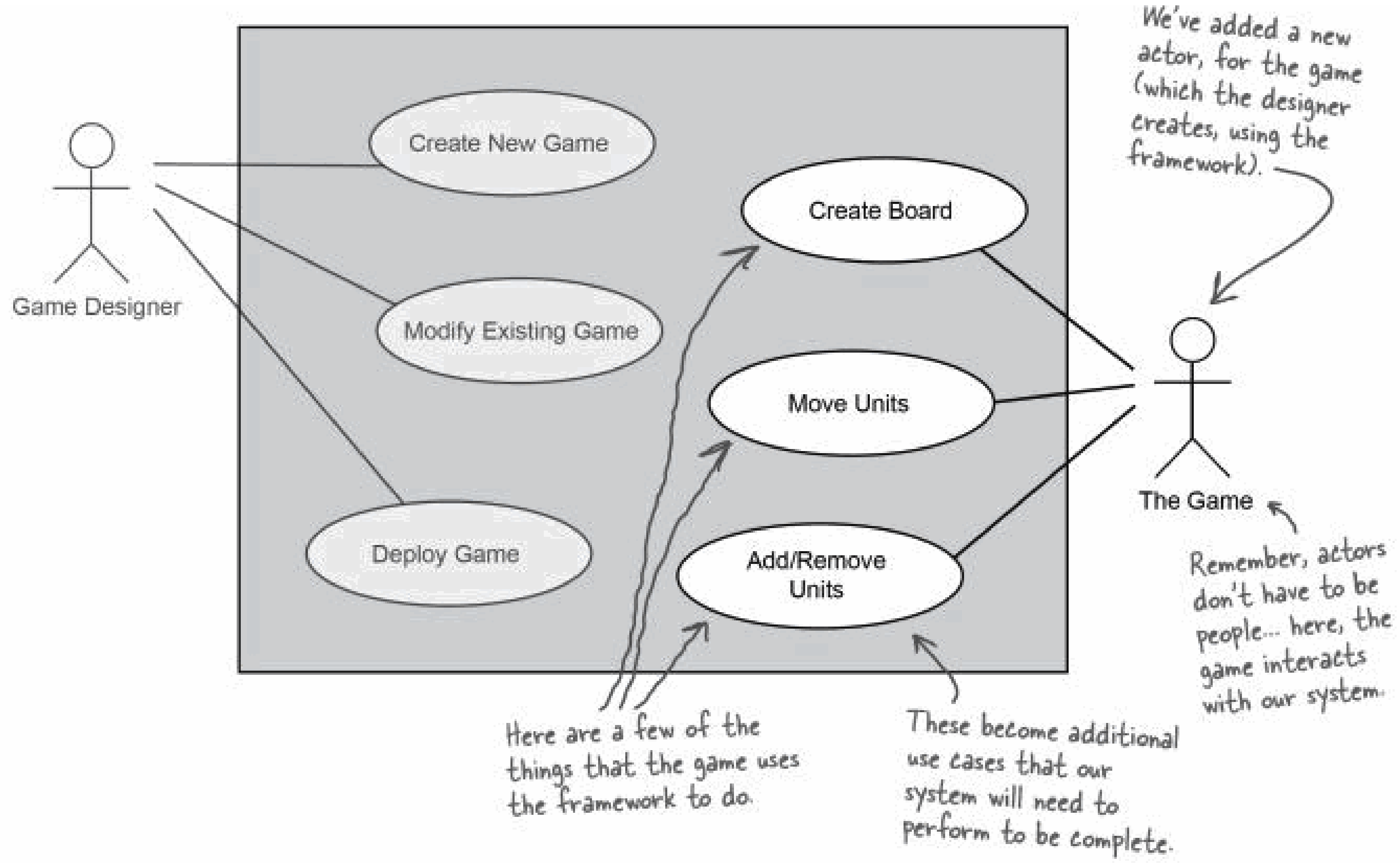
* **Blueprint of your system**
- Not be the most detailed set of blueprints for a system
- But, it tells you everything the system needs to do


This big box represents the system. What's inside the box is the system; what's outside uses the system. So the box is the system boundary.
This stick figure is an actor. He acts on the system, which in this case is the game framework.
Create New Game
Modify Existing Game
Deploy Game
Game Designer
Each of these ovals represents a single use case in the system.
Remember, the actor on this system is a game designer, not a game player.

Actor

- Actor locates outside of a system and interact with the system.
- Types of Actor
  - Users of a system
  - Other systems interacting with a system
- Naming of Actor
  - Focus on the **Role**

Actors are people, not always


Game Designer
Create New Game
Modify Existing Game
Deploy Game
Create Board
Move Units
Add/Remove Units
The Game
We've added a new actor, for the game (which the designer creates, using the framework).
Remember, actors don't have to be people... here, the game interacts with our system.
Here are a few of the things that the game uses the framework to do.
These become additional use cases that our system will need to perform to be complete.

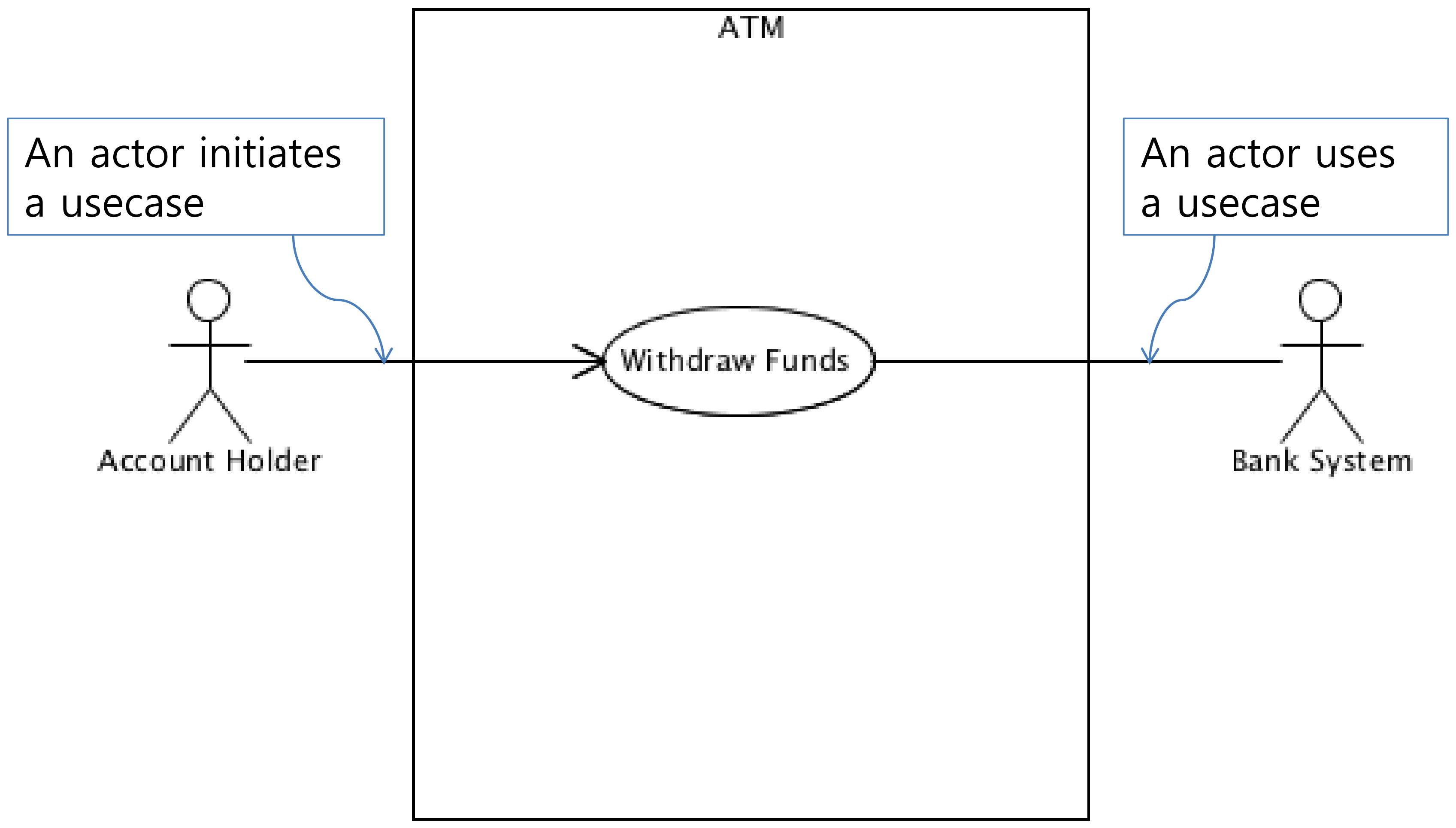
How to identify actors

- Use the following questions
  - Who use the functions of a system?
  - Which needs the resources of a system?
  - Who manages a system?
  - Which hardware is required?
  - Which other systems are required?
  - Which is interesting of the output of a system?

How to identify usecases

- Use the following questions
  - What is the main functionality of a system?
  - Which information is modified (store, remove, search…)?
  - Which events are requested from an actor to a system, or vise versa?
  - Which input/output is used by a system?

Communicates



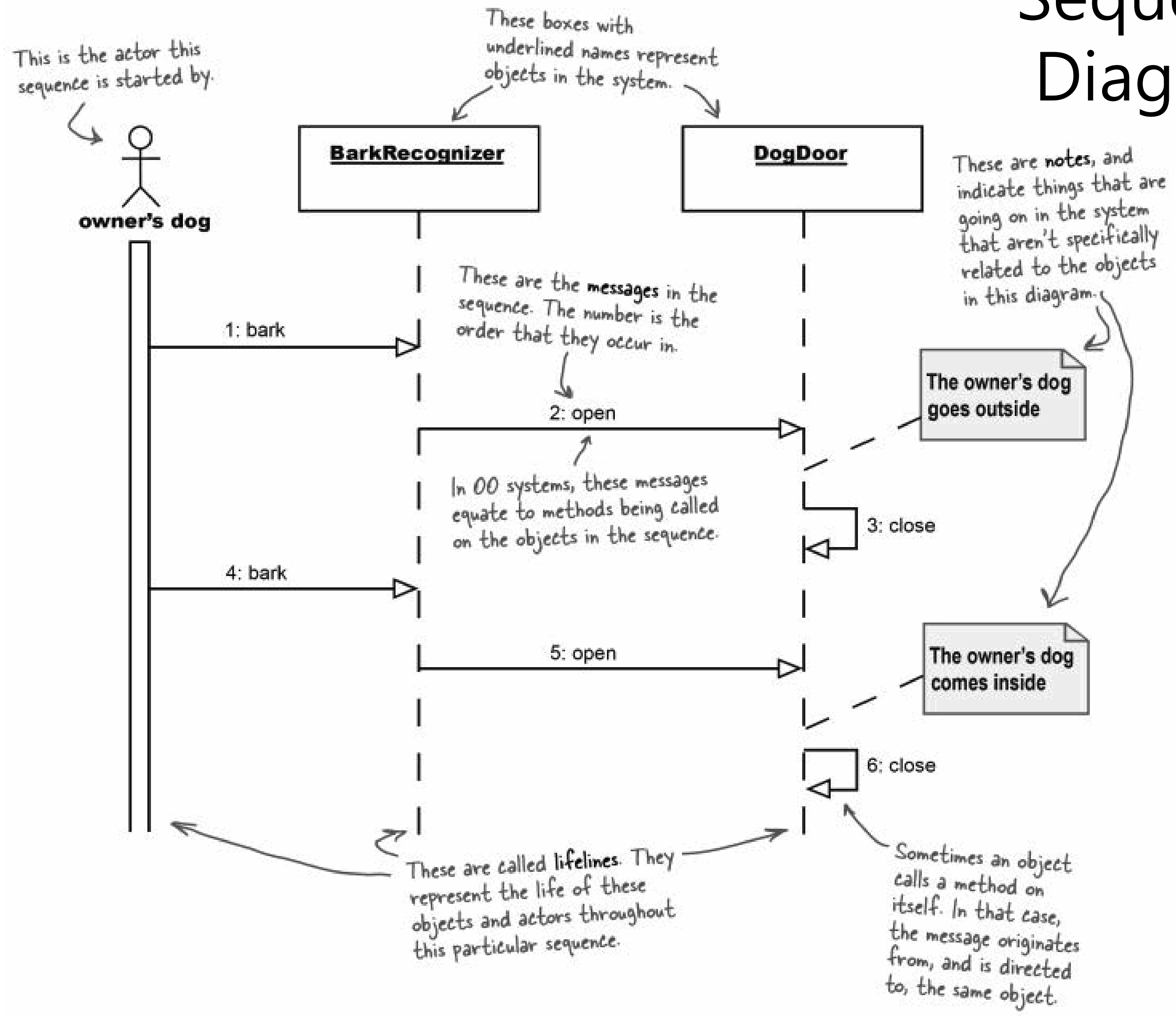
Sequence of drawing a usecase diagram

- Identify Actors
- Identify Usecases
  - Every usecase should interact at least one actor
  - Granularity of usecases should be similar
- Define Relationships
  - Between actors → generalization
  - Between actors and usecases → communicates
  - Between usecases → include, extend
- Factoring Usecases

Sequence Diagram

- A visual way to show the things that happen in particular interaction between an actor and your system
  - Focus on the timing sequence and the messages
  - Dynamic modeling
- Realization of a usecase diagram
  - Define operations and properties of objects of a system

Sequence Diagram


This is the actor this sequence is started by.
These boxes with underlined names represent objects in the system.
owner's dog
BarkRecognizer
DogDoor
These are notes, and indicate things that are going on in the system that aren't specifically related to the objects in this diagram.
These are the messages in the sequence. The number is the order that they occur in.
1: bark
2: open
In OO systems, these messages equate to methods being called on the objects in the sequence.
The owner's dog goes outside
3: close
4: bark
5: open
The owner's dog comes inside
6: close
These are called lifelines. They represent the life of these objects and actors throughout this particular sequence.
Sometimes an object calls a method on itself. In that case, the message originates from, and is directed to, the same object.

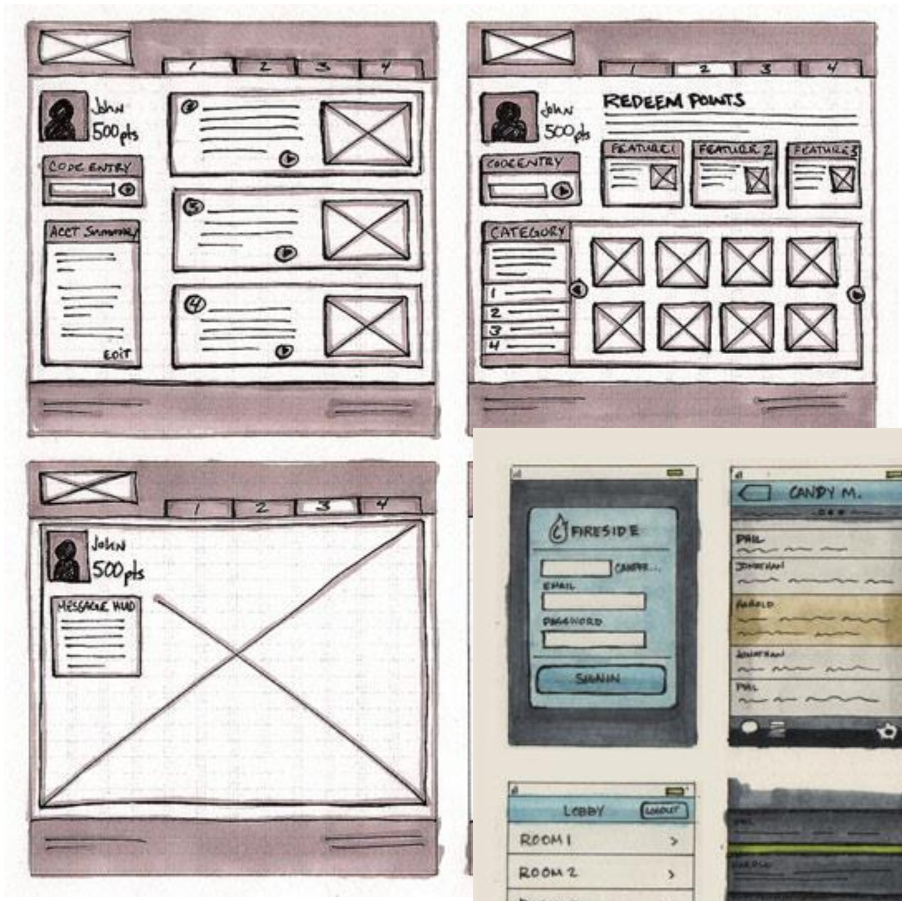
GUI Sketch

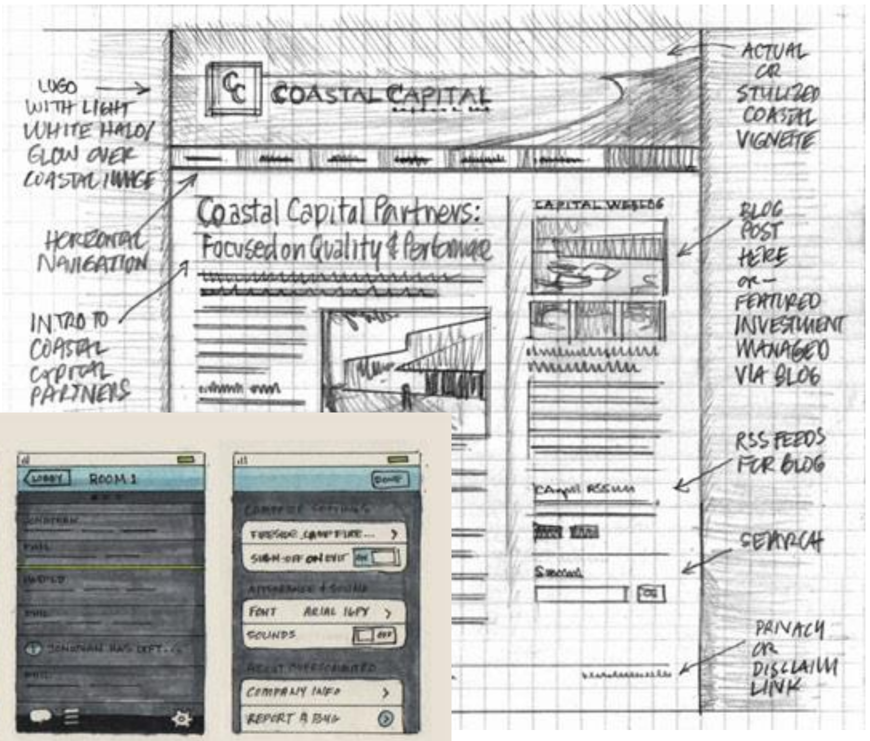
- Useful to implement a window applications
  - Swing based applications
  - Android/iPhone applications
- Identify the standard User Interface of a system

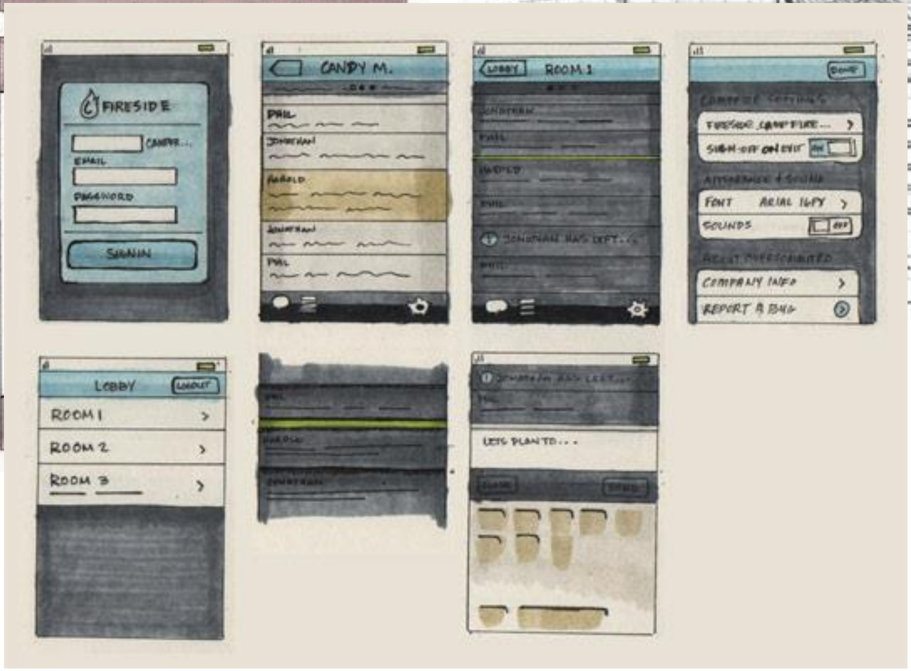
Notes of GUI sketch

- The standard User Interface
  - The types of components
  - The types of messages
- Required input parameters of a component
- Properties of each input parameter
  - Name : better to be standardized
  - Maximum/minimum length, types of input
  - Handling of errors
- Business process of given inputs

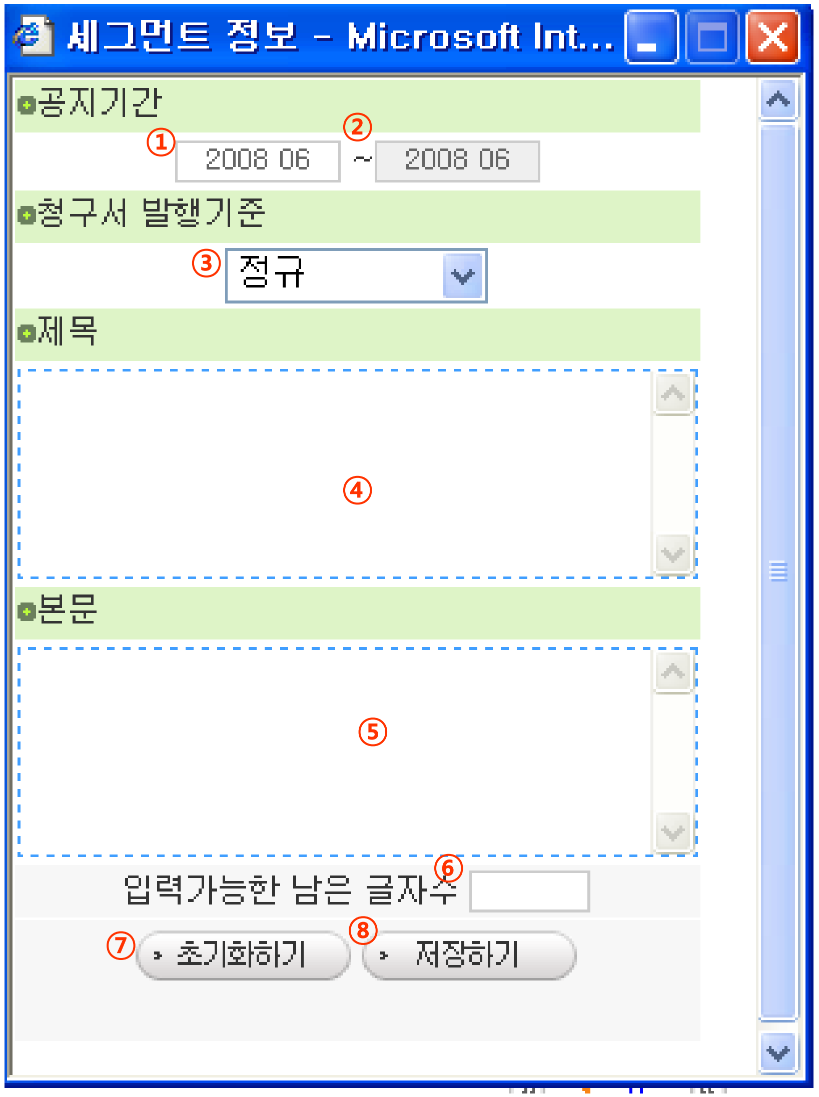
Example of GUI sketch




LOGO WITH LIGHT WHITE HALO/ GLOW OVER COASTAL IMAGE
COASTAL CAPITAL
ACTUAL OR STYLIZED COASTAL VIGNETTE
HORIZONTAL NAVIGATION
INTRO TO COASTAL CAPITAL PARTNERS
Coastal Capital Partners: Focused on Quality & Performance
CAPITAL WEBLOG
BLOG POST HERE OR FEATURED INVESTMENT MANAGED VIA BLOG
RSS FEEDS FOR BLOG
SEARCH
PRIVACY OR DISCLAIM LINK



화면ID	MMSW006	화면 명	공지사항. 입력.수정

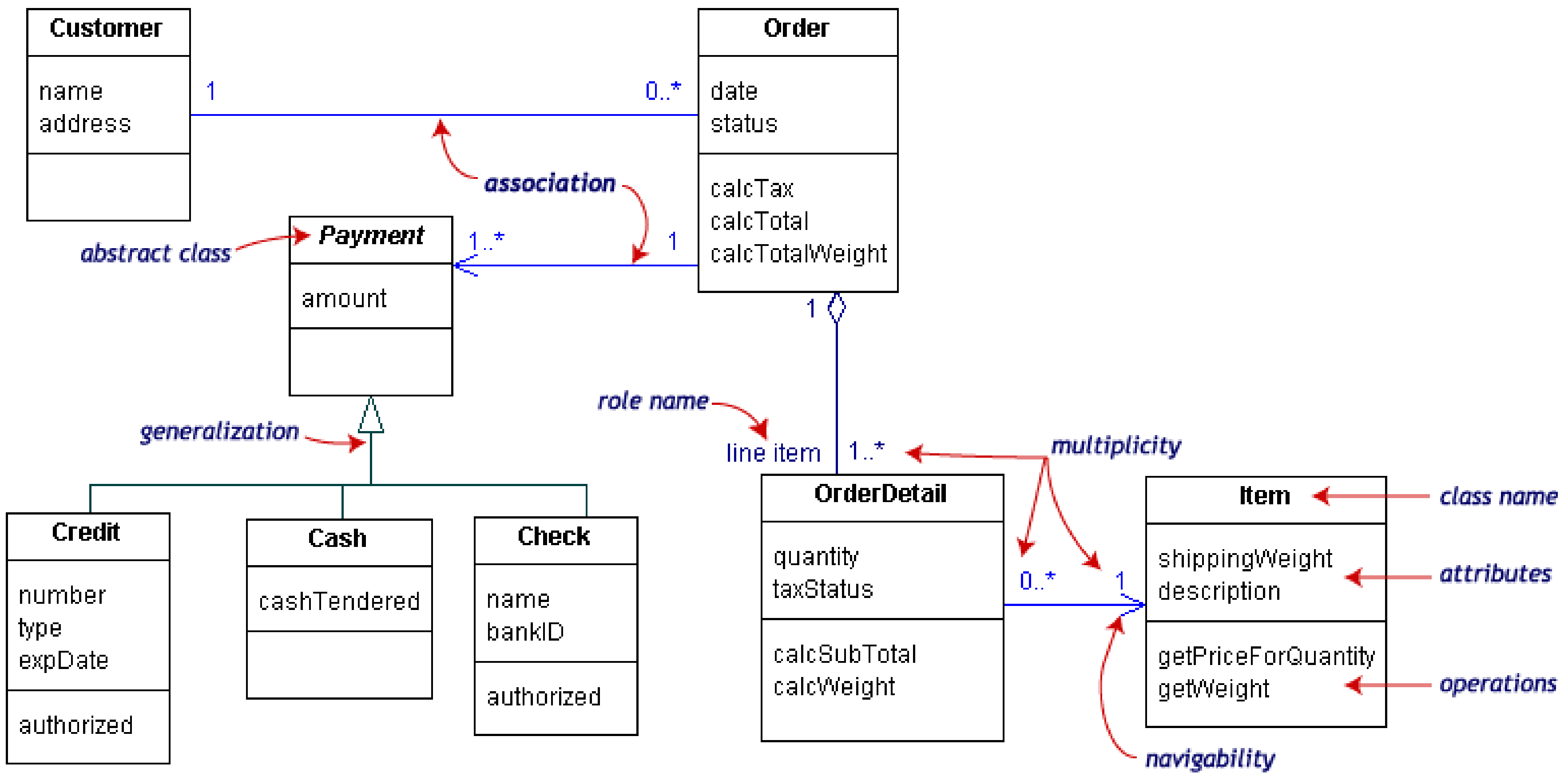
화면 layout	설 명
	1.화면설명 공지사항 입력.수정 2.주요화면 설명 1)공지기간-시작월부터 2)공지기간 -종료월까지 3)청구서 발행기준-정규/반송.재발행 4)제목-공지사항 제목 5)본문-공지사항내용 6)입력가능한 남은 글자수-본문에 입력 가능한 글자수 7)초기화-다시 원래대로 빈칸으로. 8) 저장하기- 저장하기

From : 화면 정의서 **LG텔레콤 Mobile 청구서**

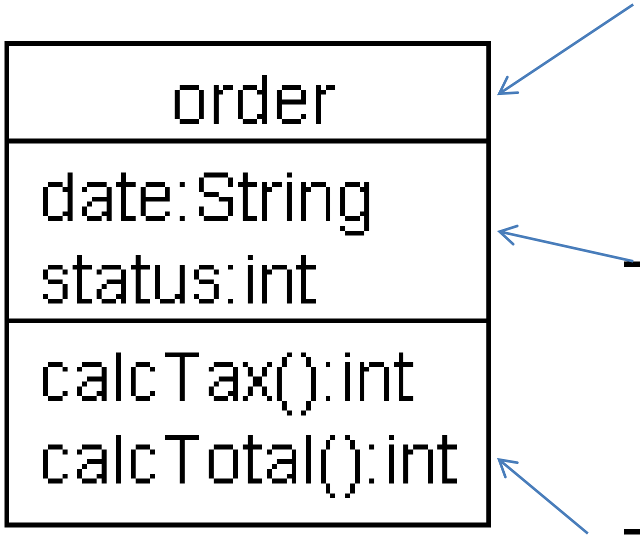
Class Diagram

- A type of static structure diagram
- Describes the structure of a system
  - Classes
  - Attributes
  - Operations (or methods)
  - Relationships among the classes
- Frequently used by Object-Oriented Design

Example of Class Diagram



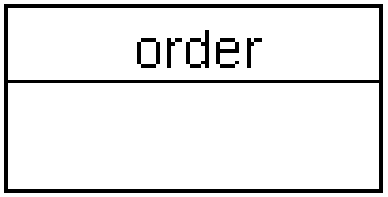
A simple class



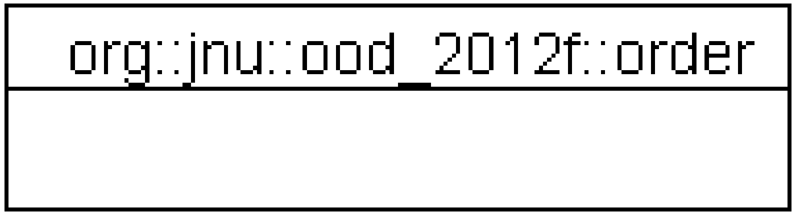
- A class with three sections
  - Upper part
    - The name of class
    - Mandatory
  - Middle part
    - The attributes
    - Optional
  - Lower part
    - The methods or operations
    - Optional

Class Name

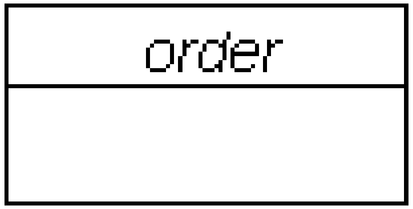
- Every class has an unique name
- Distinct to the other classes
- Simple name → using only class name
- Path name → including package name
- Abstract class → use italic font



Simple Name



Path Name



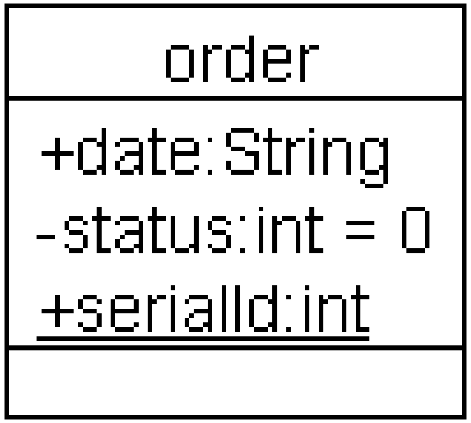
Abstract

Attribute

- Represented with nouns
- Format

Visibility Name : Type = Default_Value

  - Visibility
    - + : public
    - - : private
    - # : protected
    - underline : static

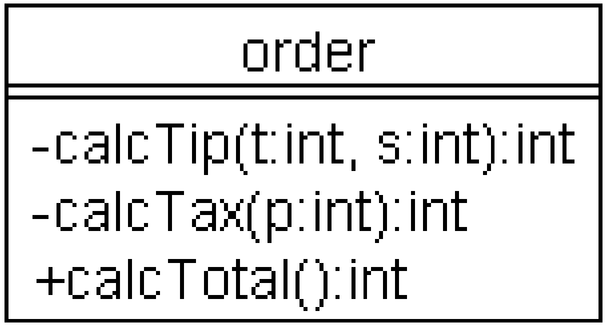


Operations

- Represented with verbs
- Format

Visibility Name (Parameter-List) : Return-Type-Expression

  - Parameter-List
    - Use tuples as (Parameter Name : Parameter Type)



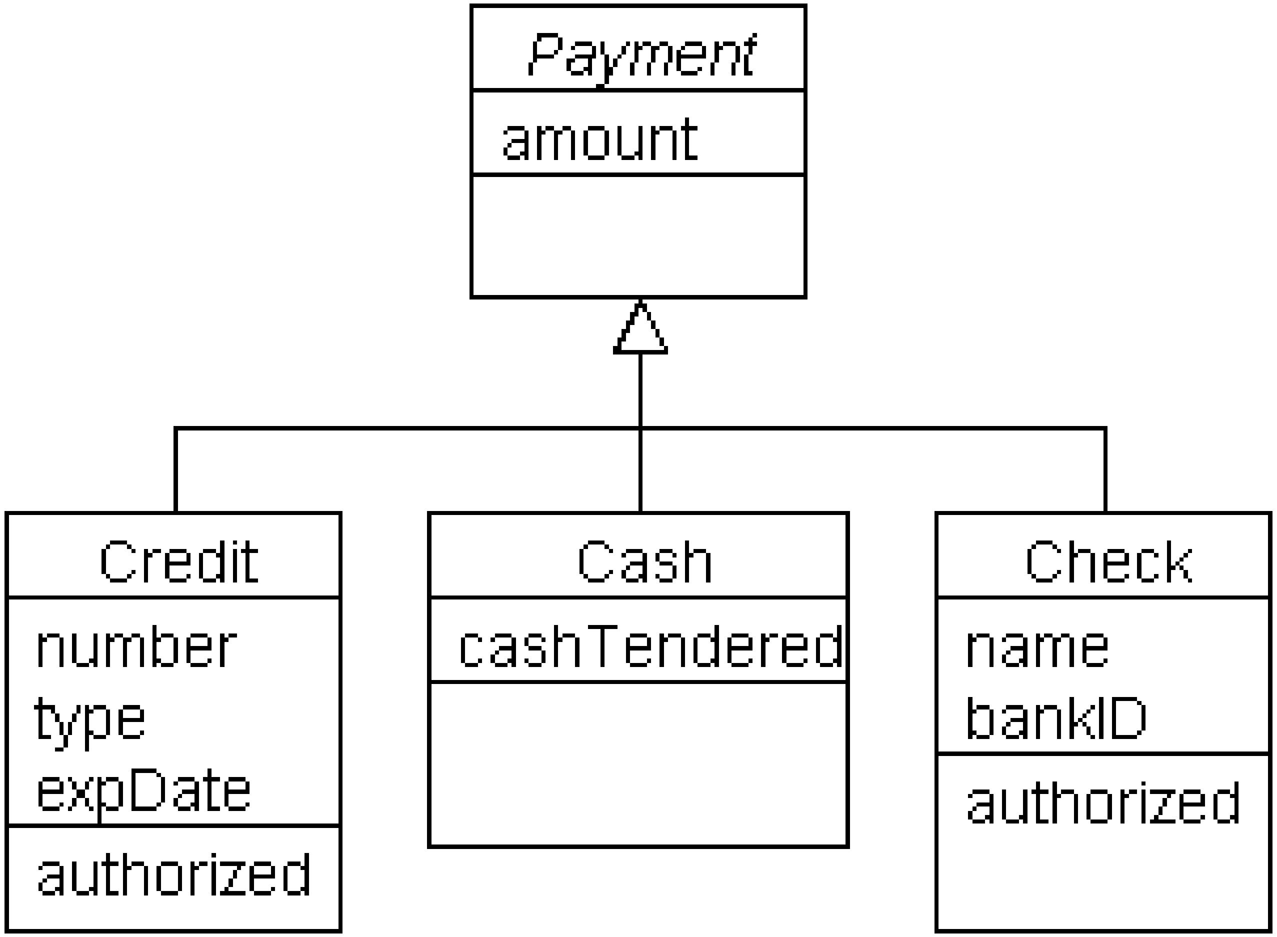
Relationships

- Logical or physical connections between classes
- Types of relationships
  - Generalization
  - Realization
  - Association
  - Aggregation
  - Composition
  - Dependency

Generalization

- “is a” relationship
  - e.g., A human is a mammal. A mammal is an animal.
- Two related classes
  - Subclass : a specialized form of superclass
  - Superclass : generalization of subclass
- Inheritance in Object-Oriented Language

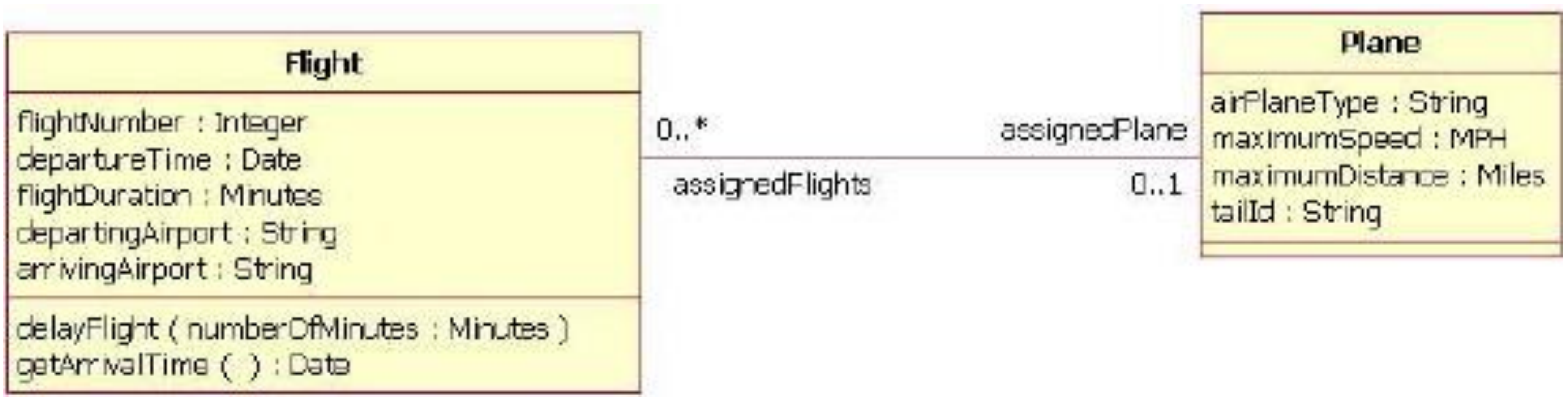
Drawing of Generalization



Association

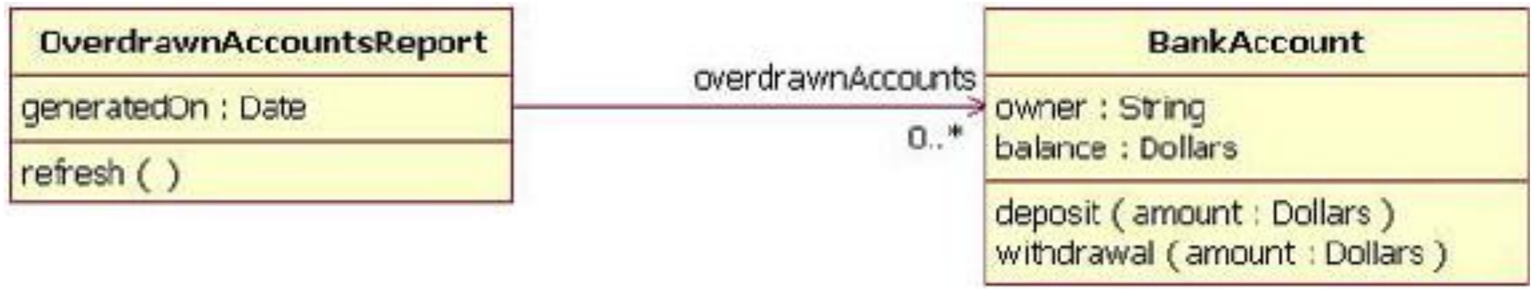
- Represents a family of links
- Relationship between instances
- Binary associations are normally represented as a line
  - An association can be **named**
  - The ends of an association can be annotated with **Role names, Ownership indicators, Multiplicity, Visibility** and others
- Types, in the aspect of ***navigability,*** that is, the ability of sending a query
  - Bidirectional Association
  - Unidirectional Association

Bidirectional Association



- Two classes know each other
- In the example
  - A Plane instance can be assigned to 0 or many Flight instances
  - A Flight instance can be assigned to 0 or 1 Plane instance

Unidirectional Association



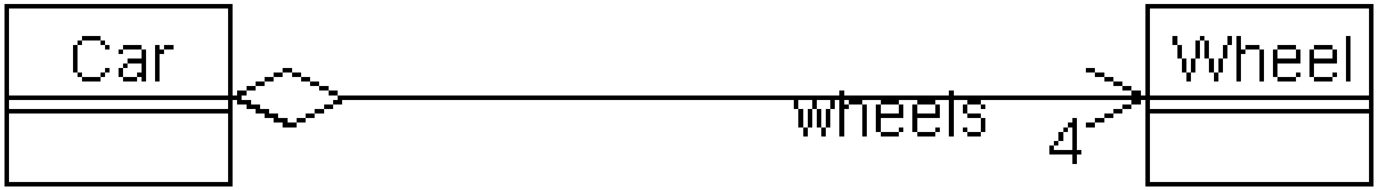
- Only one of two classes knows the relationship
- In the example
  - An OverdrawnAccountReport instance can be assigned to 0 or many BankAccount instances
  - BankAccount instance does not know the relationship

Multiplicity

- Potential Multiplicity Values
  - 0..1 : Zero or one
  - 1 : Only one
  - 0..* : Zero or many
  - * : Zero or many
  - 1..* : One orr many
  - 3 : Only three
  - 0..4 : Zero to four

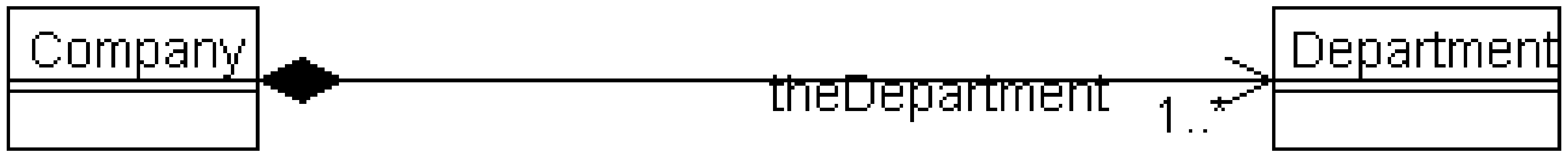
Aggregation

- Relationship between whole and part
- "has a" relationship
  - e.g., a car has four wheels
- Whole and parts are independent to each other
  - Have different lifetime

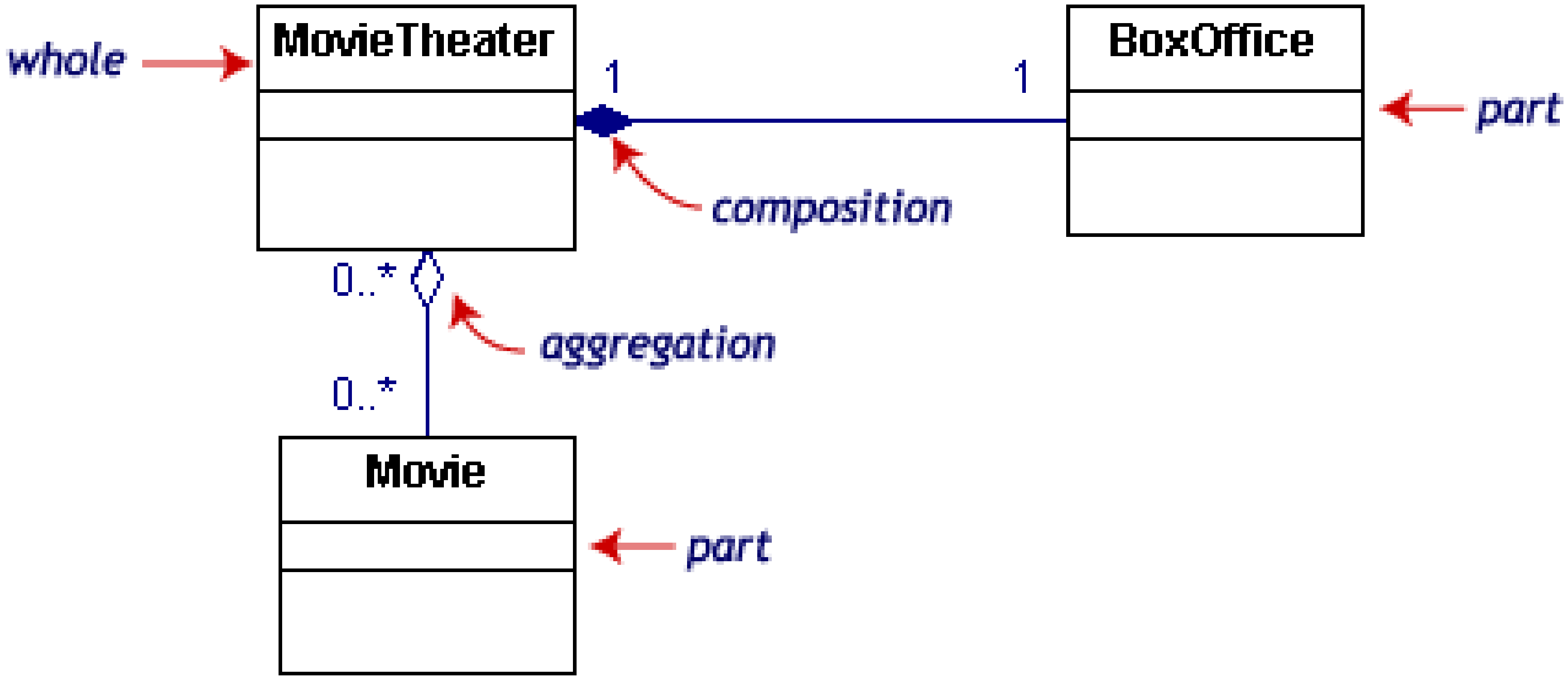


Composition

- Same concept to Aggregation
- Except one thing
  - Whole and parts are dependent to each other
  - Have the same lifetime

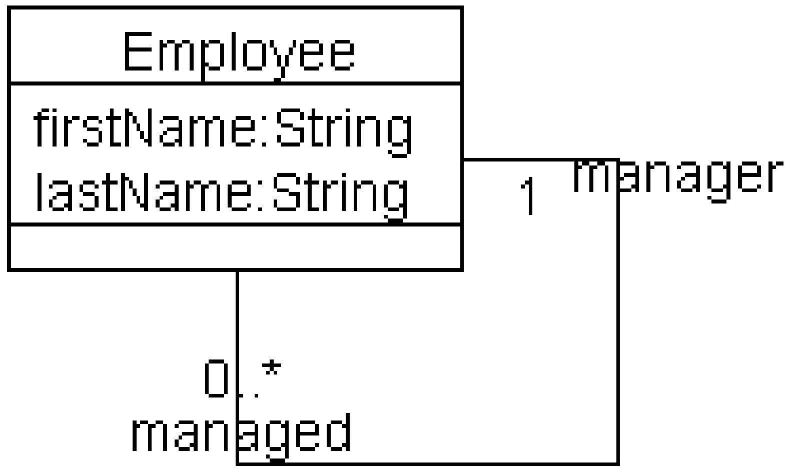


Example of Aggregation and Composition



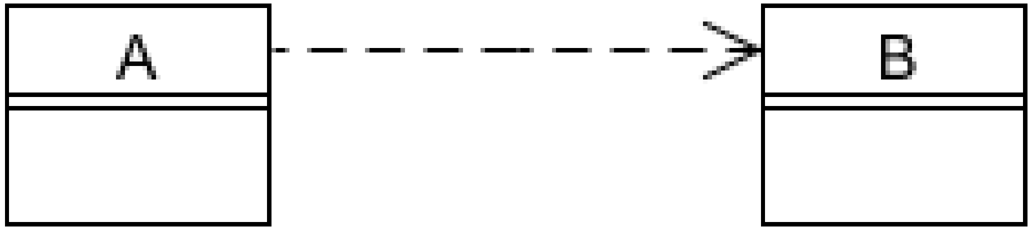
Reflexive association

- One class can be associated with itself



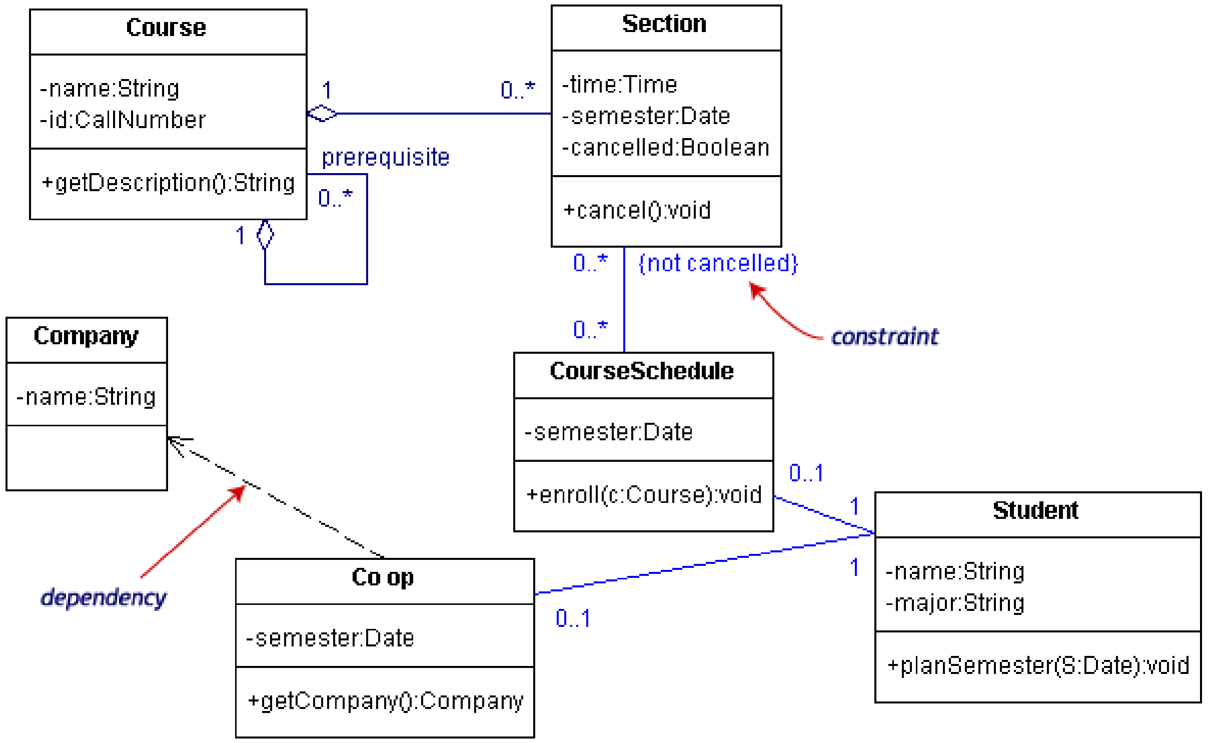
Dependency

- Weaker form of relationship
- Indicates that one class depends on another
- "using" relationship
  - B is used for a method parameter of A
  - B is used for a local parameter of A



Constraint

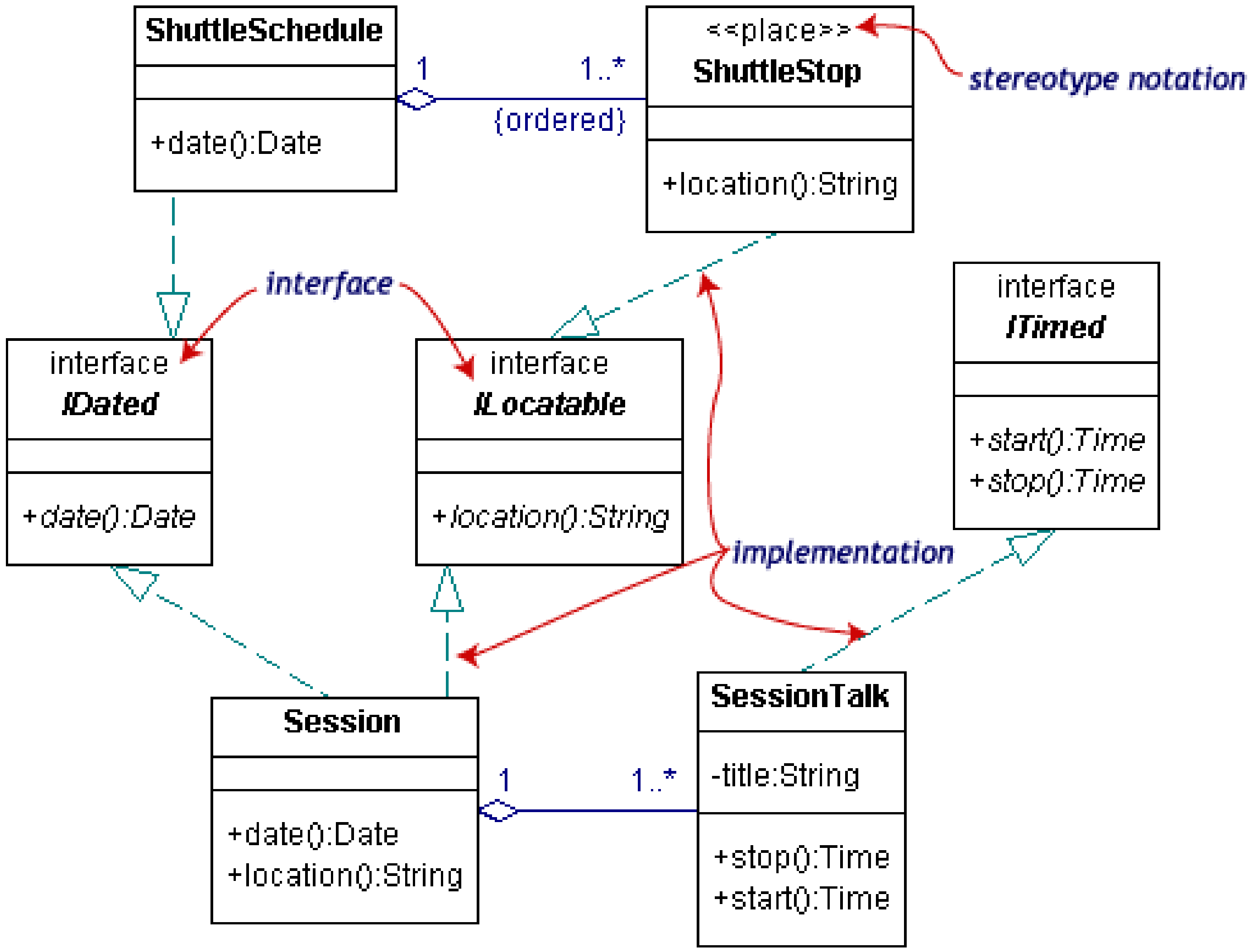
- Indicate the implementation condition
- Used with “ { } ”



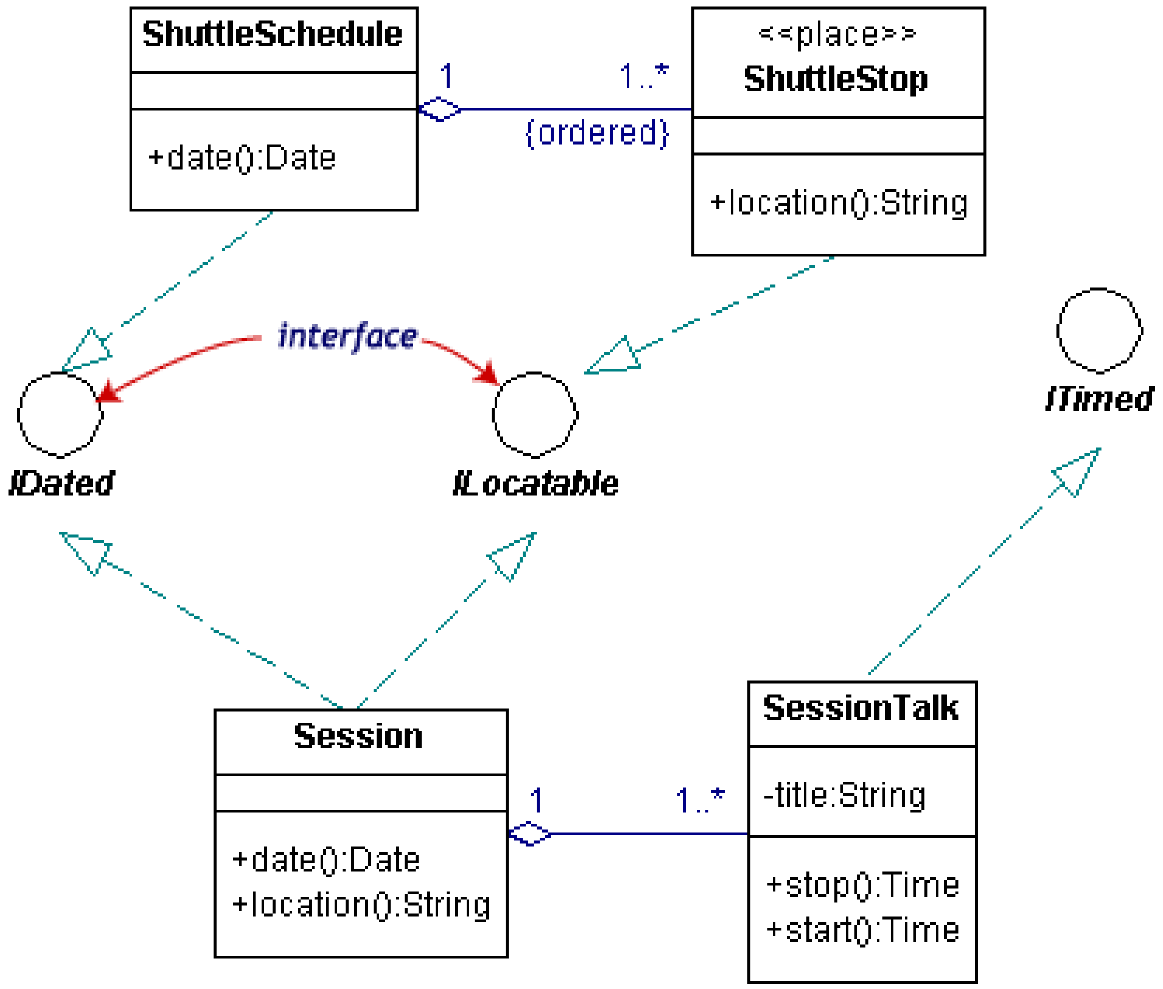
Realization

- Implement relationship
- Two model elements
  - Client : realizes (implements or executes) the behavior of a model element
  - Supplier : specifies the behavior of a model element
- Interface in Object Oriented Language
  - Allow loose coupling between components
  - Provide better flexibility to softwares

Drawing of Realization

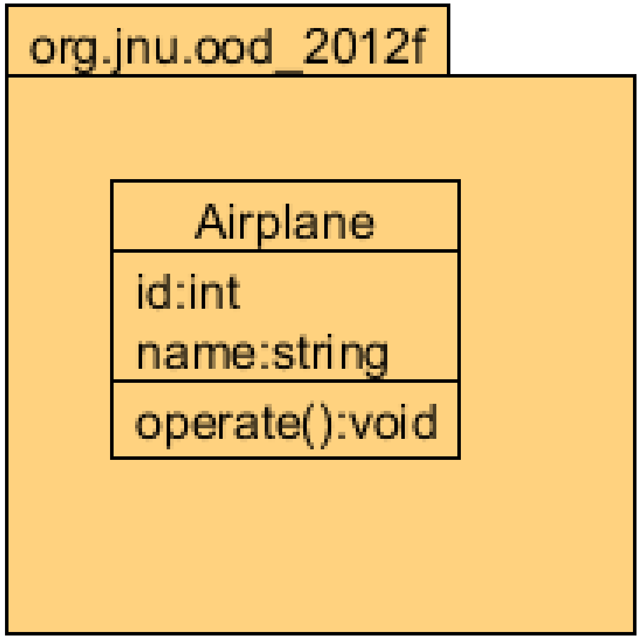


Circle Representation of Realization

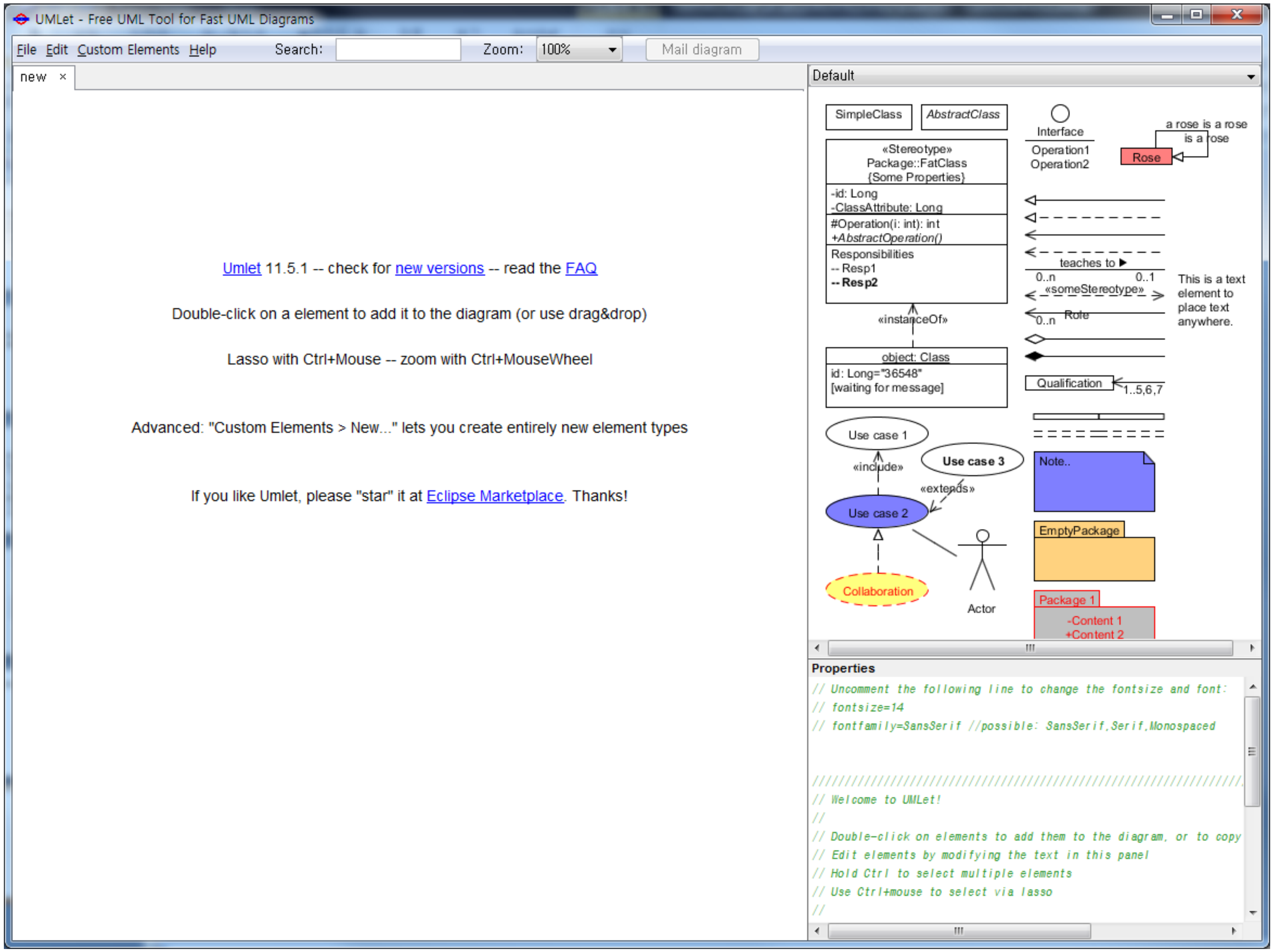


packages

- Class diagram may include the packages
- Each package has the distinct name space.



UMLet



Umlet 11.5.1 -- check for new versions -- read the FAQ

Double-click on a element to add it to the diagram (or use drag&drop)

Lasso with Ctrl+Mouse -- zoom with Ctrl+MouseWheel

Advanced: "Custom Elements > New..." lets you create entirely new element types

If you like Umlet, please "star" it at Eclipse Marketplace. Thanks!

```
// Uncomment the following line to change the fontsize and font:
// fontsize=14
// fontfamily=SansSerif //possible: SansSerif,Serif,Monospaced


//////////////////////////////////////////////////////////////
// Welcome to UMLet!
//
// Double-click on elements to add them to the diagram, or to copy
// Edit elements by modifying the text in this panel
// Hold Ctrl to select multiple elements
// Use Ctrl+mouse to select via lasso
//
```