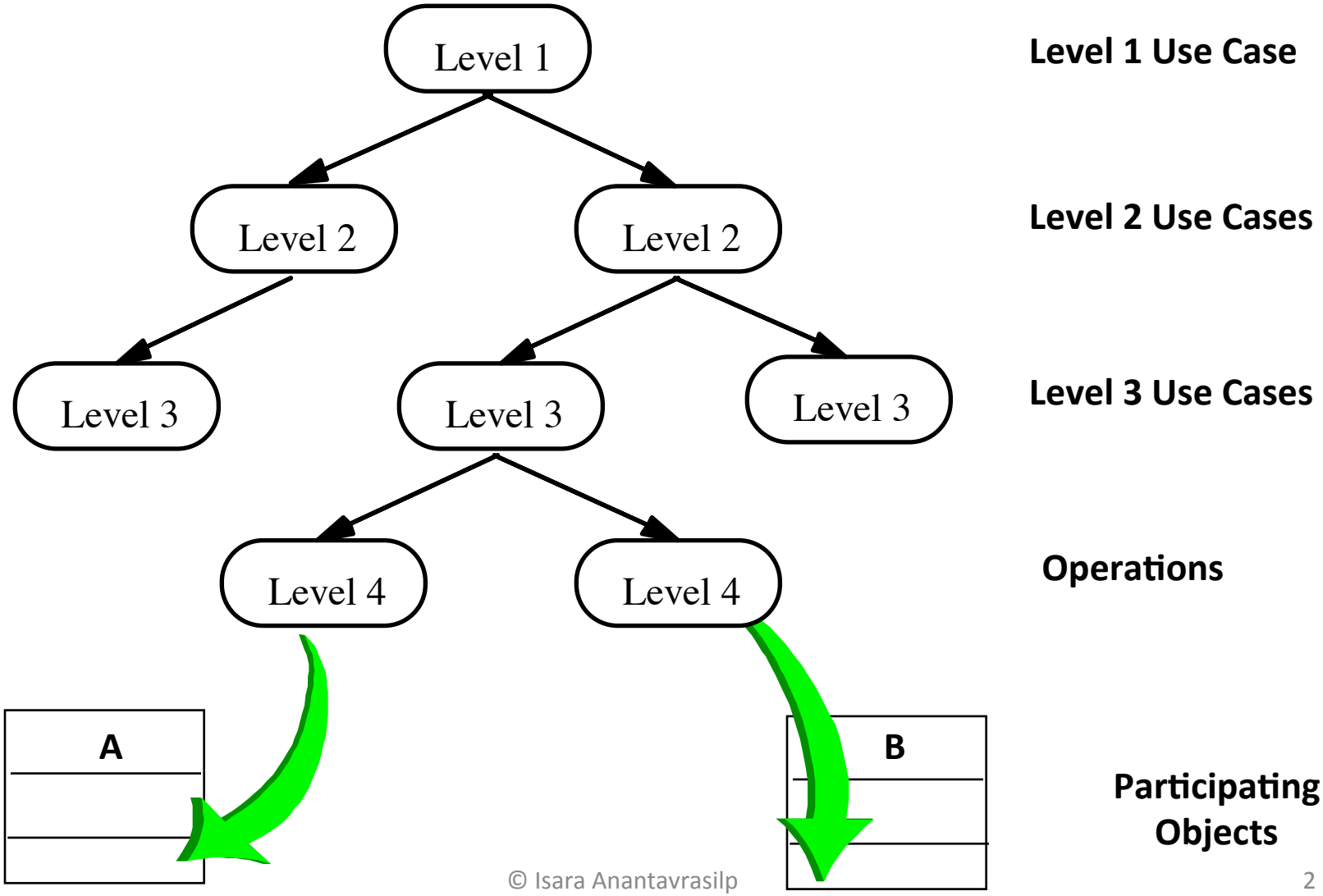
Object-Oriented Analysis and Design

Isara Anantavrasilp

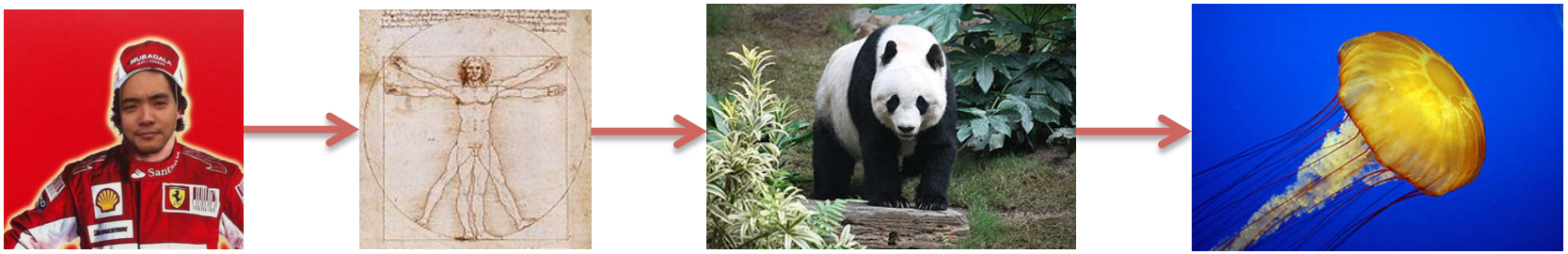
Lecture 7: Modeling with Classes

From Use Cases to Objects



Abstraction (Revisited)

- Abstraction: The classification of **phenomena** into concepts It is a process of deriving higher concepts or ideas from lower concepts or concrete objects
  - BiGGA-> Human -> Mammal -> Animal



- Abstraction allows us to ignore unessential details
- Ideas can be expressed by **models**

Activities during Object Modeling

Main goal: Find the important abstractions

- Steps during object modeling
  - Class identification
    - Based on the fundamental assumption that we can find abstractions
  - Find the attributes
  - Find the operations
  - Find the associations between classes
- Order of steps
  - Goal: get the desired abstractions
  - Order of steps secondary, only a heuristic
- What happens if we find the wrong abstractions?
  - We iterate and revise the model

Class Identification

- Approaches
  - Application domain approach
    - Ask application domain experts to identify relevant abstractions
  - Syntactic approach
    - Start with use cases
    - Analyze the text to identify the objects
    - Extract participating objects from flow of events
  - Design patterns approach
    - Use reusable design patterns
  - Component-based approach
    - Identify existing solution classes.

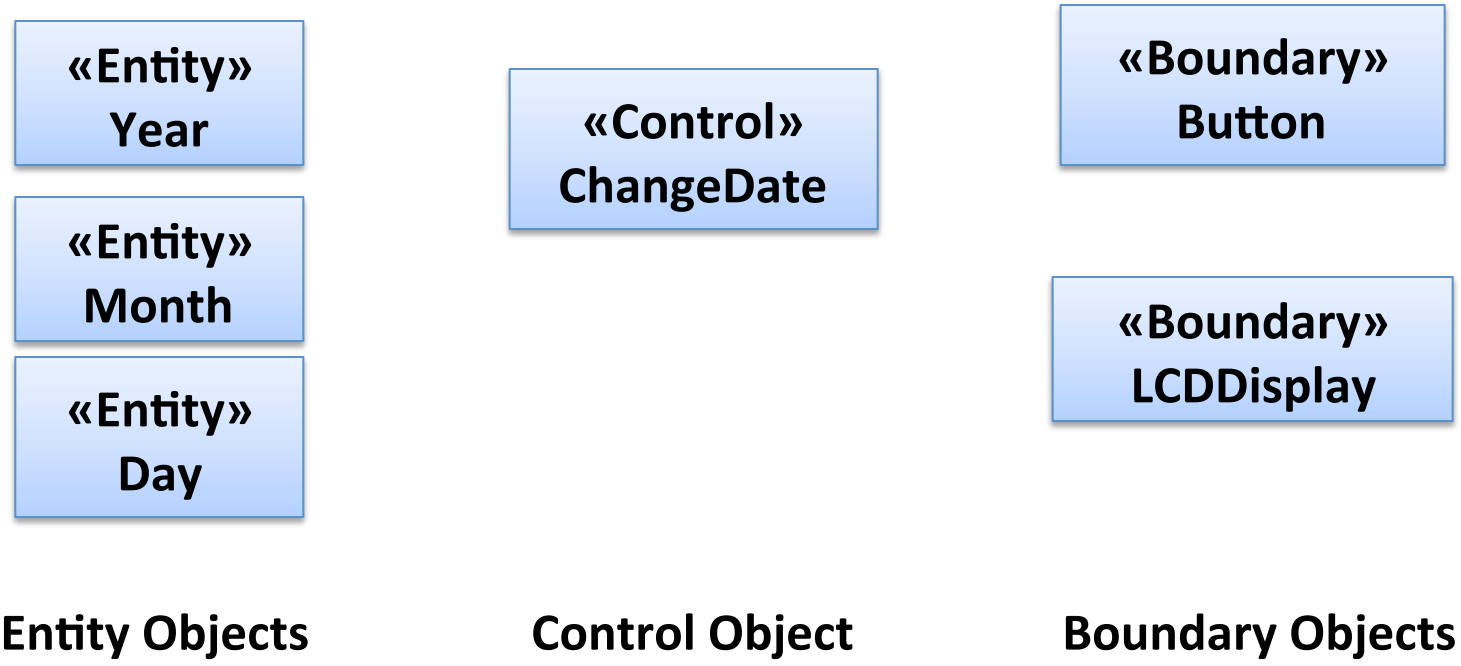
Four Different Types of Objects

- Entity Objects
  - Represent the persistent information tracked by the system (Application domain objects, also called "Business objects")
- Boundary Objects
  - Represent the interaction between the user and the system
- Control Objects
  - Represent the control tasks performed by the system
- Data Access Objects
  - Used to retrieve data from and send data to the DB

Example: Modeling A Watch

Object Types in UML

- We can use the **stereotype** mechanism to distinguish the 3 types of objects
- Stereotype is denoted by **guillemets « »** or **angle quotes**

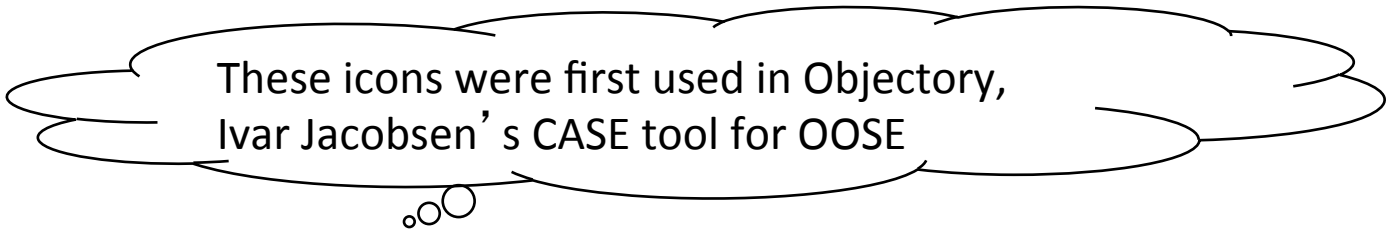


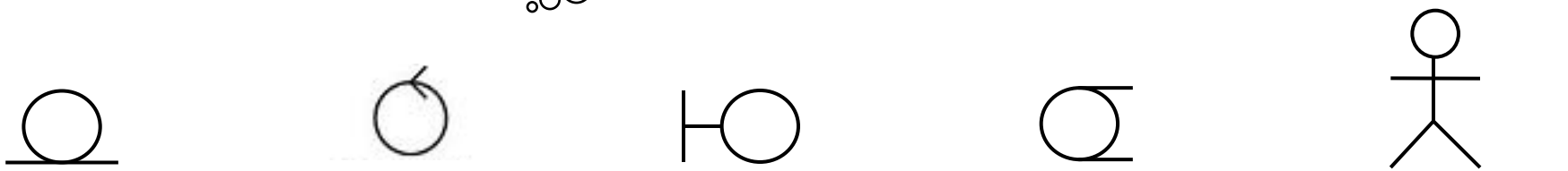
UML is an Extensible Language

- Stereotypes allow you to extend the vocabulary of the UML so that you can create new model elements, derived from existing ones
- Examples:
  - Stereotypes can also be used to classify method behavior such as «constructor», «getter» or «setter»
  - To indicate the interface of a subsystem or system, we use the stereotype «interface»
- Stereotypes can also be represented with icons and graphics:
  - This can increase the readability of UML diagrams.

Icons for Stereotypes

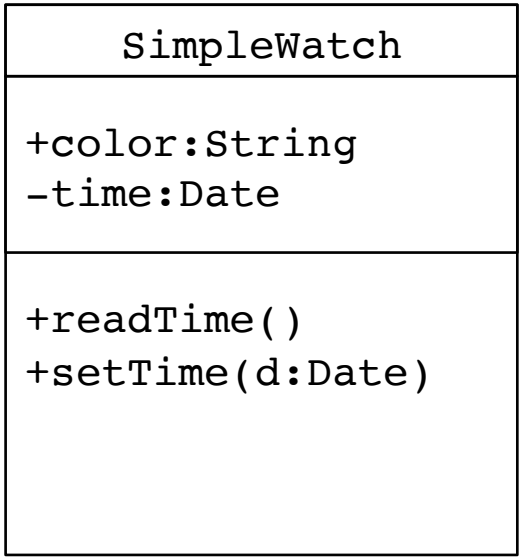
- One can use **icons** for stereotypes
  - When the stereotype is applied to a UML model element, the icon is displayed beside or above the name





Extended Class Notation

- **Visibility**: Denotes whether other objects can access particular attributes or methods
  - "-" sign denotes invisibility
  - "+" sign denotes visibility
- **Method signature**: Shows the information needed to invoke a method of a class
  - Method visibility
  - Method name
  - Method parameters



The 3 UML "Amigos"

Ivar Jacobson

Invented use cases, sequence diagrams and collaboration diagrams to model software controlled telephone switches at Ericsson (1967)

James Rumbaugh

Developed the OMT Notation (Object Modeling Technique)

Added inheritance to E/R Modeling, 1991

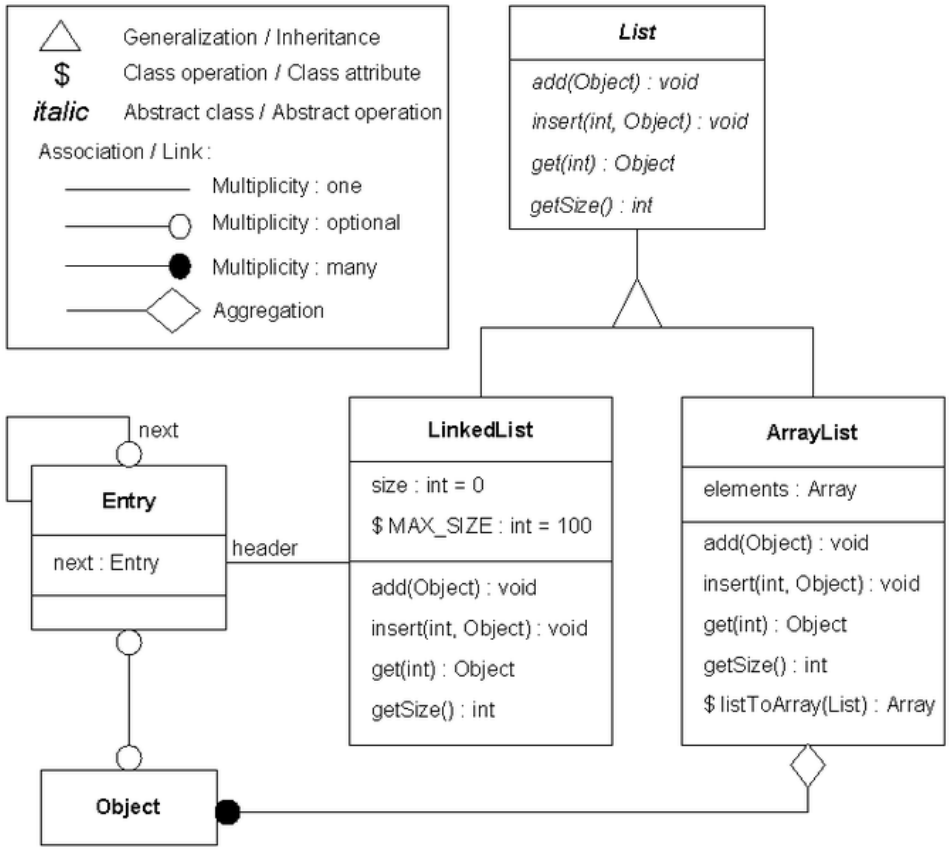
Grady Booch

One of the first OO modelers

Developed the Booch Method ("Objects as clouds", 1991)

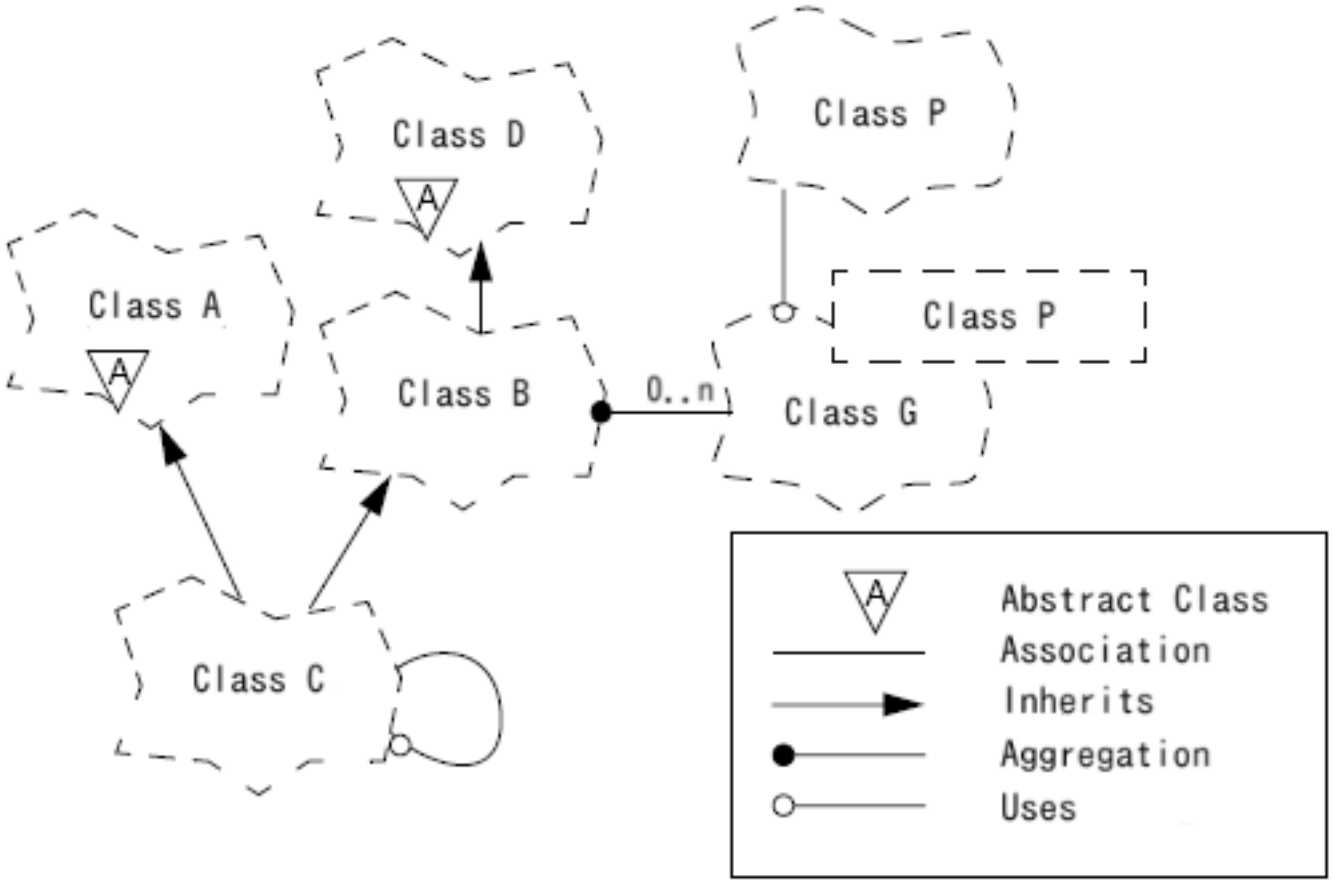
Images: http://en.wikipedia.org

OMT Notation
(Object Modeling Technique)



OMT is no longer an active language. However, it is still important to know about it , because it is the notation used in the book “Design Patterns”.

Booch Notation



Why do we distinguish 3 Object Types?

- Having three types of object leads to models that are more resistant (“resilient”) to change
  - The interface of a system changes more likely than the control
  - The way the system is controlled changes more likely than entities in the application domain
- Object types originated in Smalltalk:
  - Model, View, Controller (MVC)
    - Model <-> Entity Object
    - View <-> Boundary Object
    - Controller <-> Control Object

Smalltalk

- Smalltalk was developed at Xerox Parc in the 1970s by
  - Adele Goldberg
  - Alan Kay
- Smalltak was used to prototype the WIMP (windows, icons, menus, pointers) interface, the cornerstone for today's graphical user interfaces
- Objective-C is also designed based on Smalltalk

Images: http://en.wikipedia.org

How do we find Objects?

- Pick a use case and look at the flow of events
- Do a textual analysis (noun-verb analysis)
  - Nouns are candidates for objects/classes
  - Verbs are candidates for operations
  - This is also called **Abbott's Technique**
- After objects/classes are found, identify their types
  - Identify **real world entities** that the system needs to keep track of
    Identify **real world procedures** that the system needs to keep track of
    Identify **interface artifacts**

Example for using the Technique

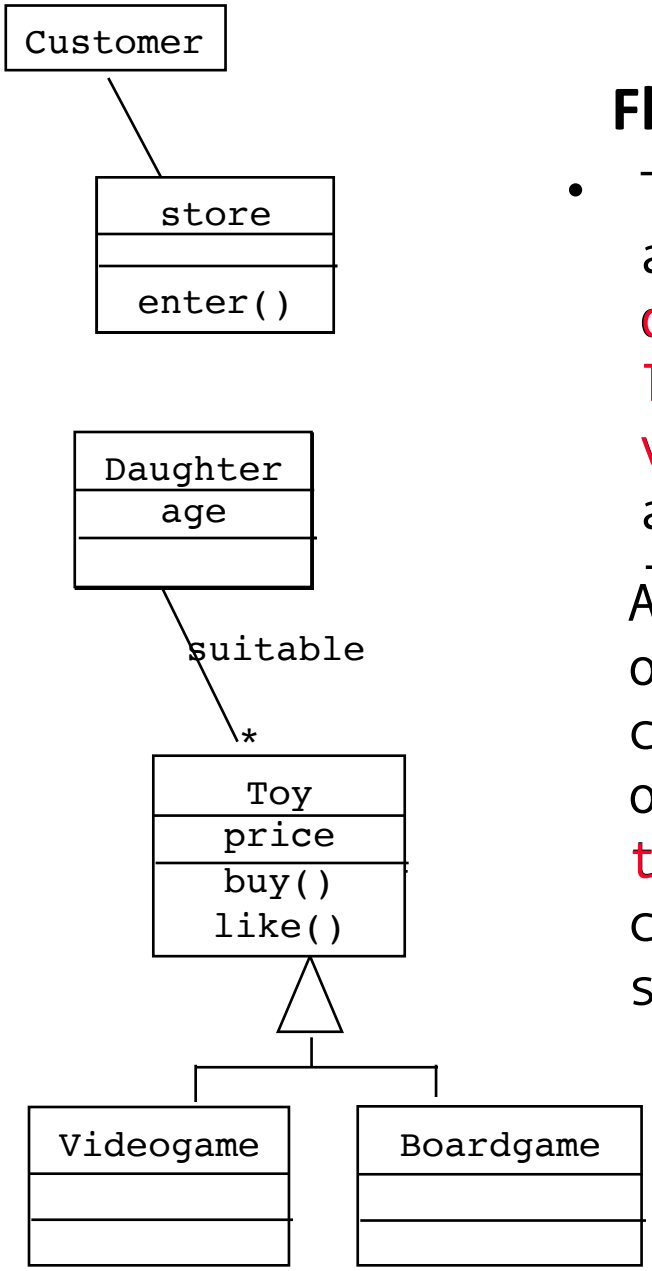
Flow of Events:

- The customer enters the store to buy a toy.
- It has to be a toy that his daughter likes and it must cost less than 50 Euro.
- He tries a videogame, which uses a data glove and a head-mounted display. He likes it.
- An assistant helps him.
- The suitability of the game depends on the age of the child.
- His daughter is only 3 years old.
- The assistant recommends another type of toy, namely the boardgame “Monopoly".

Textual Analysis (Abbot's Technique)

Example	Part of speech	UML model component
"Monopoly"	Proper noun	object
Toy	Improper noun	class
Buy, recommend	Doing verb	operation
is-a	being verb	inheritance
has an	having verb	aggregation
must be	modal verb	constraint
dangerous	adjective	attribute
enter	transitive verb	operation
depends on	intransitive verb	Constraint, class, association

Generating a Class Diagram from Flow of Events



Flow of events:

- The customer enters the store to buy a toy. It has to be a toy that his daughter likes and it must cost less than 50 Euro. He tries a videogame, which uses a data glove and a head-mounted display. He likes it.

An assistant helps him. The suitability of the game depends on the age of the child. His daughter is only 3 years old. The assistant recommends another type of toy, namely a boardgame. The customer buy the game and leaves the store

Ways to find Objects

- Syntactical investigation with Abbot's technique:
  - Flow of events in use cases
  - Problem statement
- Use other knowledge sources:
  - **Application knowledge**: End users and experts know the abstractions of the application domain
  - **Solution knowledge**: Abstractions in the solution domain
  - **General world knowledge**: Your generic knowledge and intuition

Order of Activities for Object Identification

1. Formulate a few scenarios with help from an end user or application domain expert
2. Extract the use cases from the scenarios, with the help of an application domain expert
3. Then proceed in parallel with the following:
   - Analyse the flow of events in each use case using Abbot's textual analysis technique
   - Generate the UML class diagram.

Steps in Generating Class Diagrams

1. Class identification (textual analysis, domain expert)
2. Identification of attributes and operations (sometimes before the classes are found!)
3. Identification of associations between classes
4. Identification of multiplicities
5. Identification of roles
6. Identification of inheritance

Who uses Class Diagrams?

- Purpose of class diagrams
  - The description of the static properties of a system
- The main users of class diagrams:
  - The application domain expert
    - uses class diagrams to model the application domain (including taxonomies)
      - during requirements elicitation and analysis
  - The developer
    - uses class diagrams during the development of a system
      - during analysis, system design, object design and implementation.

Who does not use Class Diagrams?

- The client and the end user are often not interested in class diagrams
  - Clients usually focus more on project management issues
  - End users usually focus on the functionality of the system.

“Historical” Readings

OMT:

- James Rumbaugh, Michael Blaha, William Premerlani, and Frederick Eddy. Object-Oriented Modeling and Design with UML (2nd Edition by Rumbaugh and Blaha), Prentice Hall 1991 and 2004.

OOSE and Use Cases:

- Ivar Jacobson, Magnus Christerson, Patrik Jonsson, Gunnar Overgaard *Object-Oriented Software Engineering: A Use Case Driven Approach (ACM Press) Addison-Wesley, 1992, ISBN 0201544350*

Booch Method:

- Booch, Grady (1993). *Object-oriented Analysis and Design with Applications,* 2nd ed., Redwood City: Benjamin Cummings.

Developers have different Views on Class Diagrams

- There are different roles in a software development project
  - Analyst
  - System Designer
  - Object Designer
  - Developer (Implementor)
- Each of these roles has a different view about the class diagram (the object model).

The View of the Analyst

- The analyst is interested
  - in application classes: The associations between classes are relationships between abstractions in the application domain
  - operations and attributes of the application classes (difference to E/R models!)
- The analyst uses inheritance in the model to reflect the taxonomies in the application domain
  - Taxonomy: An is-a-hierarchy of abstractions in an application domain
- The analyst is not interested
  - in the exact **signature** of operations
  - in solution domain classes.

The View of the Designer

- The designer focuses on the solution of the problem, that is, the solution domain
- The associations between classes now mean references (pointers) between classes in the application or solution domain
- An important design task is the specification of interfaces:
  - The designer describes the interface of classes and the interface of subsystems
  - Subsystems originate from modules (term often used during analysis):
    - **Module**: a collection of classes
    - **Subsystem**: a collection of classes with an interface
- Subsystems are modeled in UML with a package

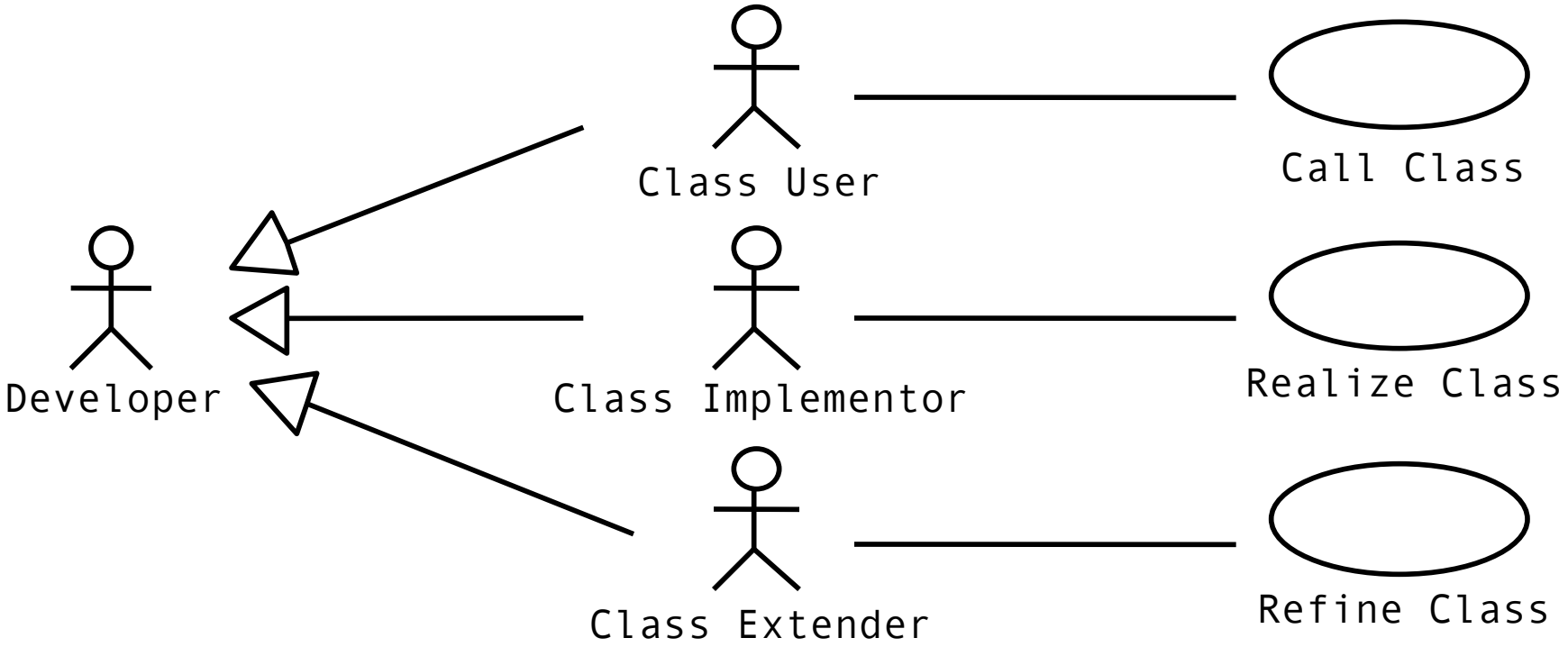
Goals of the Designer

- The most important design goals for the designer are design usability and design reusability
- **Design usability**: the subsystem interfaces are usable from as many classes as possible within in the system
- **Design reusability**: The subsystem interfaces are designed in a way, that they can also be reused by other (future) software systems
  - Class libraries
  - Frameworks
  - Design patterns

The View of the Developer

- **Class implementor**
  - Must realize the interface of a class in a programming language
  - Interested in appropriate data structures (for the attributes) and algorithms (for the operations)
- **Class extender**
  - Interested in how to extend a class to solve a new problem or to adapt to a change in the application domain
- **Class user**
  - The class user is interested in the signatures of the class operations and conditions, under which they can be invoked
  - The class user is not interested in the implementation of the class.

Class Implementor, Class Extender and Class User



Why do we distinguish different Users of Class Diagrams?

- Models often don't distinguish between application domain classes and solution domain classes
  - **Reason**: Modeling languages like UML allow the use of both types of classes in the same model
    - "address book", "array"
  - **Preferred**: No solution classes in the analysis model
- Many systems don't distinguish between the specification and the implementation of a class
  - **Reason**: Object-oriented programming languages allow the simultaneous use of specification and implementation of a class
  - **Preferred**: We distinguish between analysis model and object design model. The analysis design model does not contain any implementation specification.

Analysis Model vs. Object Design Model

- The analysis model is constructed during the analysis phase
  - Main stake holders: End user, customer, analyst
  - The class diagrams contains only application domain classes

- The object design model (sometimes also called specification model) is created during the object design phase
  - Main stake holders: class specifiers, class implementors, class extenders and class users
  - The class diagrams contain application domain as well as solution domain classes.