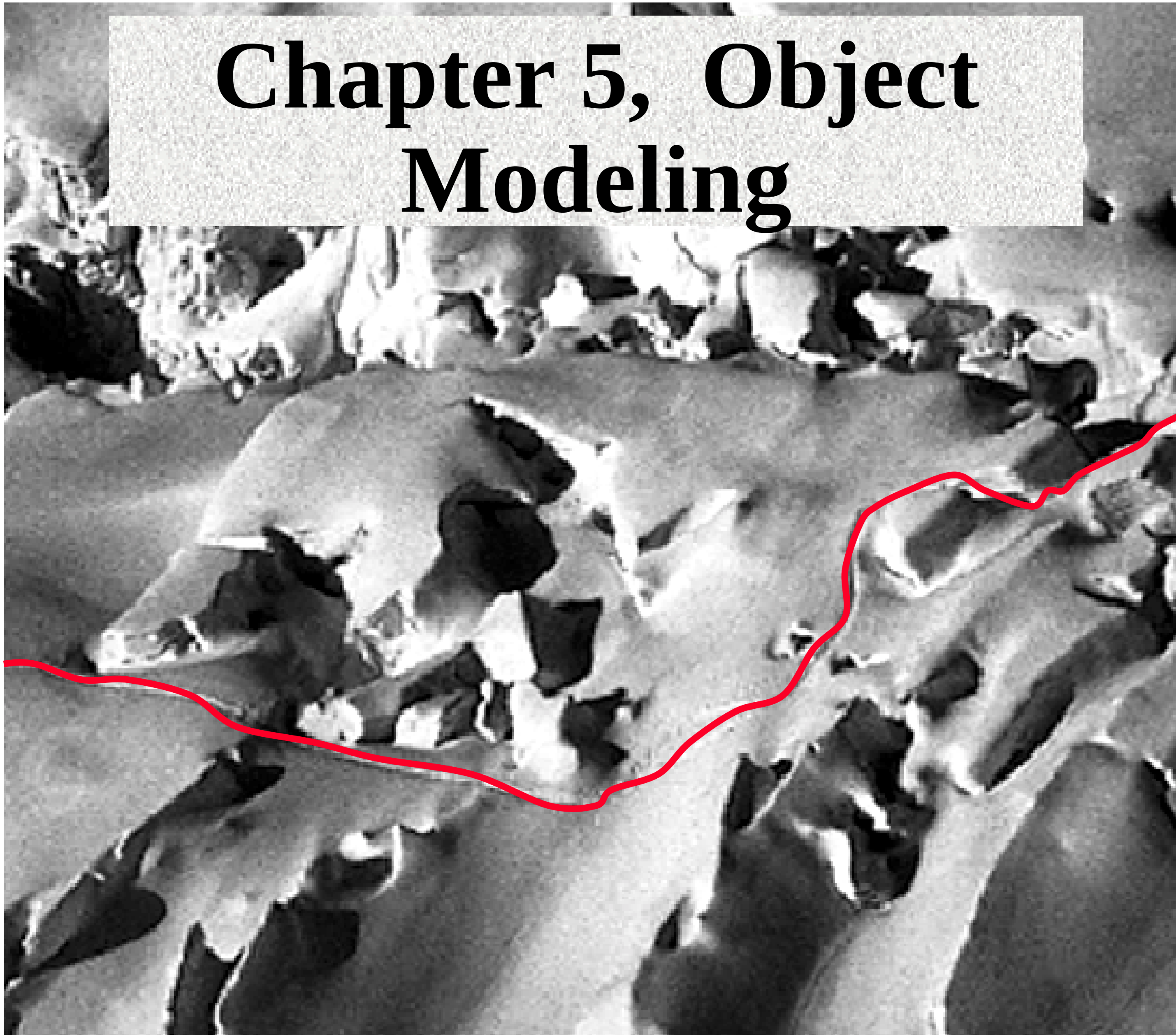
Object-Oriented Software Engineering
Using UML, Patterns, and Java

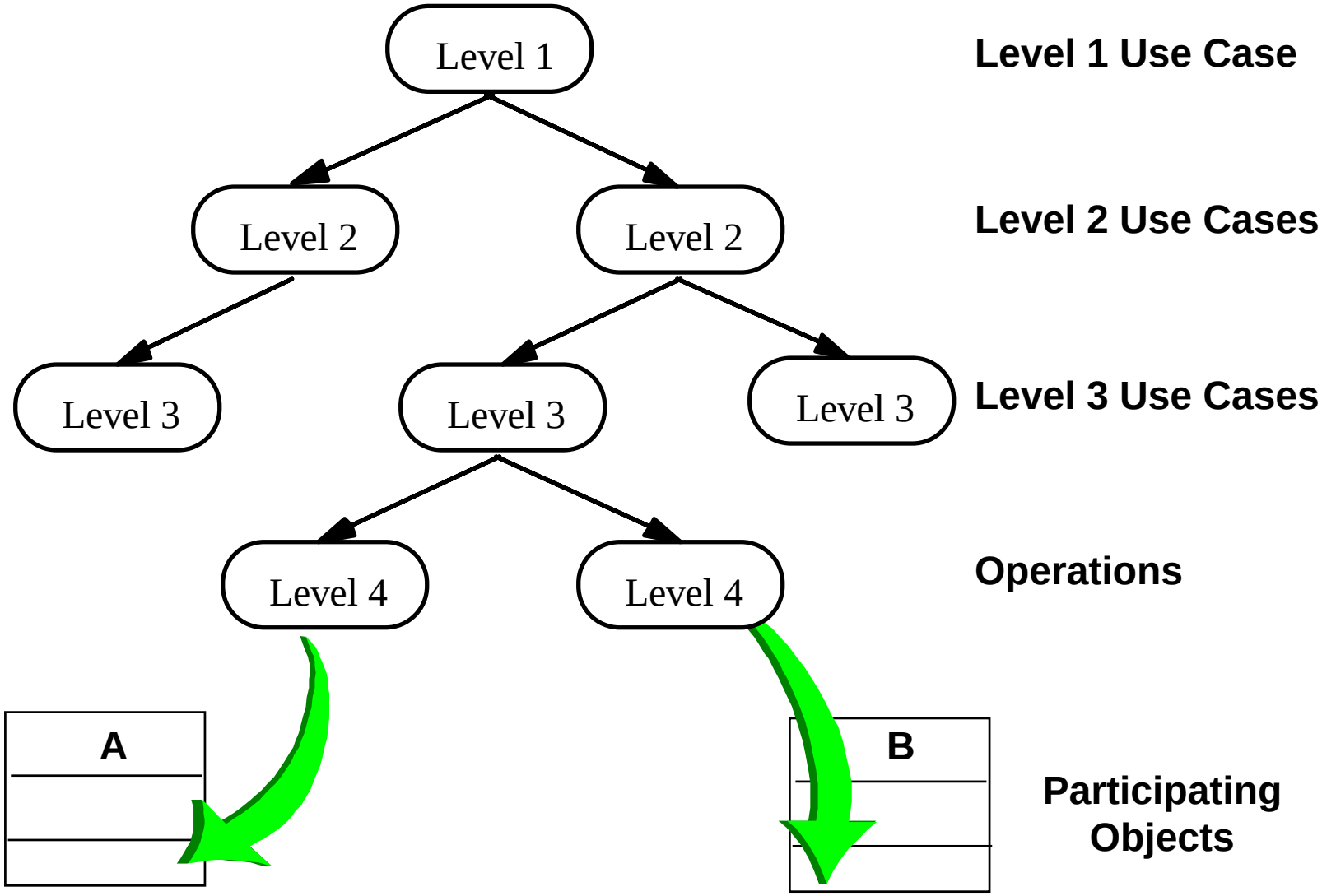
Chapter 5, Object Modeling



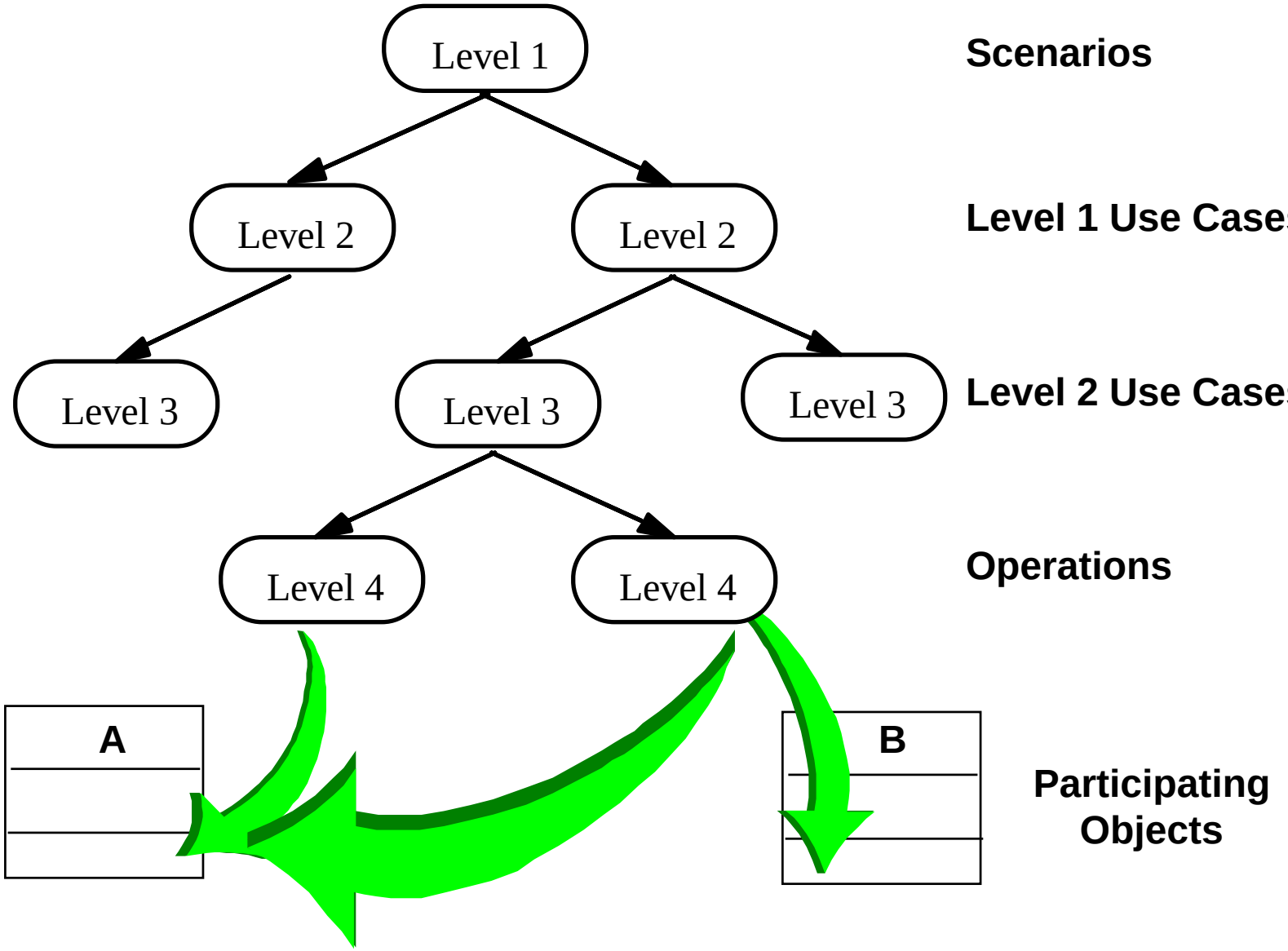
Outline

- From use cases to class diagrams
- Model and reality
- Activities during object modeling
- Object identification
- Object types
  - **entity, boundary and control objects**
- Object naming
- Abbott’s technique helps in object identification
- Users of class diagrams

From Use Cases to Objects



From Use Cases to Objects: Why Functional Decomposition is not Enough



Reality and Model

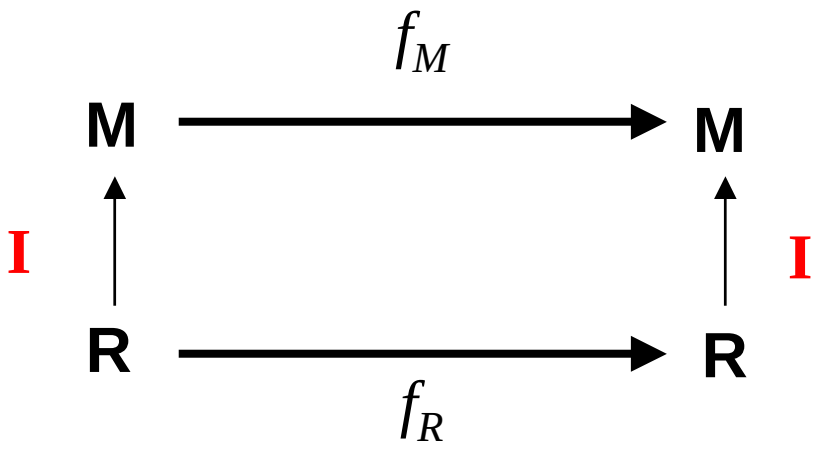
- Reality R: **Real Things, People, Processes happening during some time, Relationship between things**
- Model M: **Abstractions from (really existing or only thought of) things, people , processes and relationships between these abstractions.**

Why models?

- We use models
  - **To abstract away from details in the reality, so we can draw complicated conclusions in the reality with simple steps in the model**
  - **To get insights into the past or presence**
  - **To make predictions about the future**

What is a "good" model?

- **Relationships, which are valid in reality R, are also valid in model M.**
  - I : Mapping of real things in reality R to abstractions in the model M abbildet (Interpretation)
  - f_M: relationship between abstractions in M
  - f_R: relationship between real things inR
- **In a good model the following diagram is commutative:**

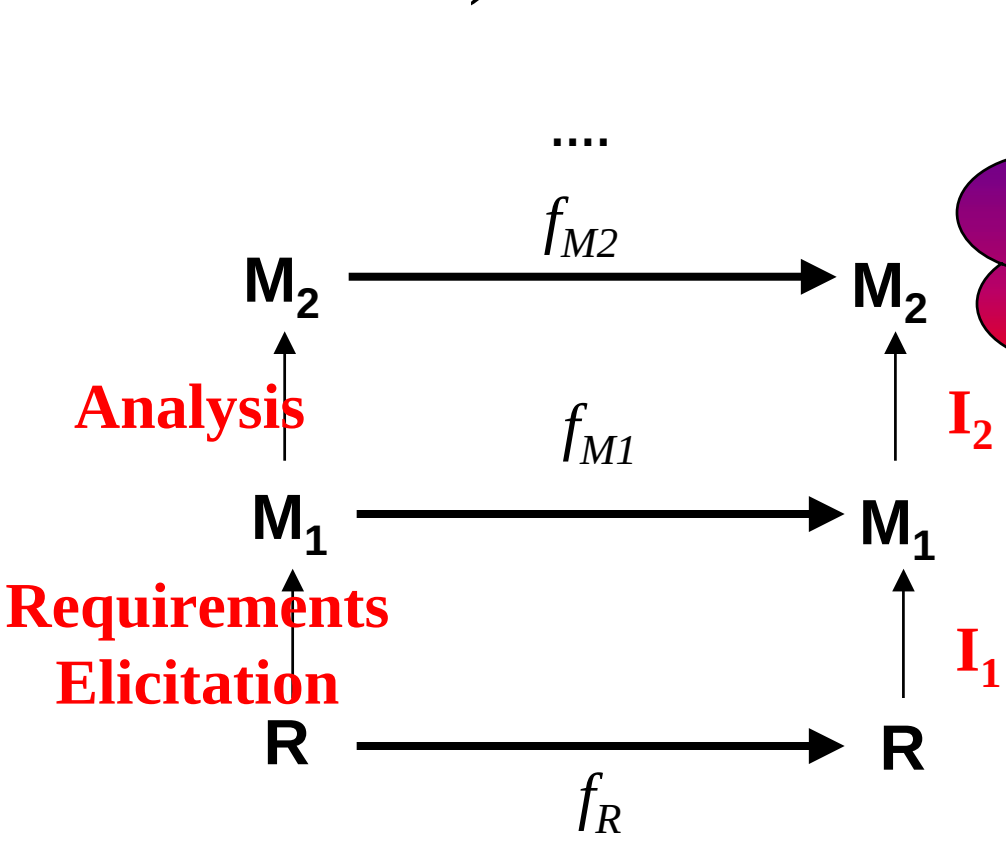


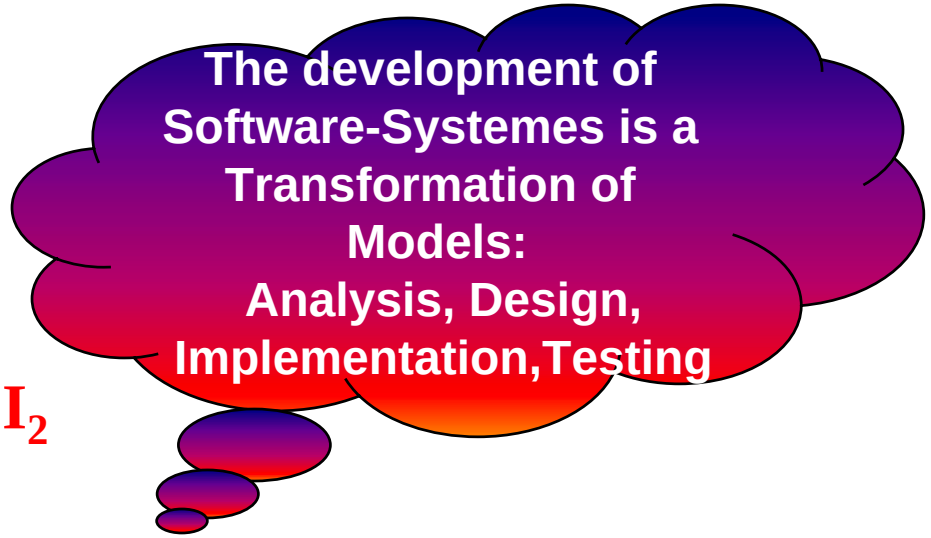
Models are falsifiable

- In the middle age people believed in truth
- Models of reality cannot be true
- A model is always an approximation
  - **We must say "according to our knowledge", or "with today's knowledge"**
- Popper ("Objective Knowledge):
  - **We can only build models from reality, which are "true" until, we have found a counter example** *(Principle of Falsification)*
    - **And even then we might stick with the model ("because it works quite well in most settings")**
- The falsification principle is the basis of software development
  - **The goal of prototypes, reviews and system testing is to falsify the software system**

Models of models of models...

- Modeling is relative. **We can think of a model as reality and can build another model from it (with additional abstractions).**





Activities during Object Modeling

- Main goal: Find the important abstractions
- What happens if we find the wrong abstractions?
  - **Iterate and correct the model**
- Steps during object modeling
  - **1. Class identification**
    - **Based on the fundamental assumption that we can find abstractions**
  - **2. Find the attributes**
  - **3. Find the methods**
  - **4. Find the associations between classes**
- Order of steps
  - **Goal: get the desired abstractions**
  - **Order of steps secondary, only a heuristic**
  - **Iteration is important**

Class Identification

- Identify the boundaries of the system
- Identify the important entities in the system
- Class identification is crucial to object-oriented modeling
- Basic assumption:
  - **1. We can find the classes for a new software system (Forward Engineering)**
  - **2. We can identify the classes in an existing system (Reverse Engineering)**
- Why can we do this?
  - **Philosophy, science, experimental evidence**

Class identification is an ancient problem

- Objects are not just found by taking a picture of a scene or domain
- The application domain has to be analyzed.
- Depending on the purpose of the system different objects might be found
  - **How can we identify the purpose of a system?**
  - **Scenarios and use cases**
- Another important problem: Define system boundary.
  - **What object is inside, what object is outside?**

Pieces of an Object Model

- Classes
- Associations (Relations)
  - **Generic associations**
  - **Canonical associations**
    - **Part of- Hierarchy (Aggregation)**
    - **Kind of-Hierarchy (Generalization)**
- Attributes
  - **Detection of attributes**
  - **Application specific**
  - **Attributes in one system can be classes in another system**
  - **Turning attributes to classes**
- Operations
  - **Detection of operations**
  - **Generic operations: Get/Set, General world knowledge, design patterns**
  - **Domain operations: Dynamic model, Functional model**

Object vs Class

- Object (instance): Exactly one thing
  - **This lecture on Software Engineering**
- A class describes a group of objects with similar properties
  - **Game, Tournament, mechanic, car, database**
- *Object diagram:* A graphic notation for modeling objects, classes and their relationships ("associations"):
  - ***Class diagram:*** Template for describing many instances of data. Useful for taxonomies, patters, schemata...
  - ***Instance diagram:*** A particular set of objects relating to each other. Useful for discussing scenarios, test cases and examples

Class identification

- Finding objects is the central piece in object modeling
- Approaches
  - **Application domain approach (not a special lecture, examples):**
    - **Ask application domain expert to identify relevant abstractions**
  - **Syntactic approach (today):**
    - **Start with use cases. Extract participating objects from flow of events**
    - **Use noun-verb analysis (Abbot's technique) to identify components of the object model**
  - **Design patterns approach (Lecture on design patterns)**
    - **Use reusable design patterns**
  - **Component-based approach (Lecture on object design):**
    - **Identify existing solution classes**

How do you find classes?

- Finding objects is the central piece in object modeling
  - **Learn about problem domain: Observe your client**
  - **Apply general world knowledge and intuition**
  - **Take the flow of events and find participating objects in use cases**
  - **Try to establish a taxonomy**
  - **Do a syntactic analysis of *problem statement, scenario* or *flow of events***
  - **Abbott Textual Analysis, 1983, also called noun-verb analysis**
    - **Nouns are good candidates for classes**
    - **Verbs are good candidates for opeations**
  - **Apply design knowledge:**
    - **Distinguish different types of objects**
    - **Apply design patterns (Lecture on design patterns)**

How do you find classes?

- Finding objects is the central piece in object modeling
  - **Learn about problem domain: Observe your client**
  - **Apply general world knowledge and intuition**
  - **Take the flow of events and find participating objects in use cases**
  - **Try to establish a taxonomy**
  - **Apply design knowledge:**
    - **Distinguish different types of objects**
    - **Apply design patterns (Lecture on design patterns)**
  - **Do a syntactic analysis of *problem statement*, *scenario* or *flow of events***
  - **Abbott Textual Analysis, 1983, also called noun-verb analysis**
    - **Nouns are good candidates for classes**
    - **Verbs are good candidates for opeations**

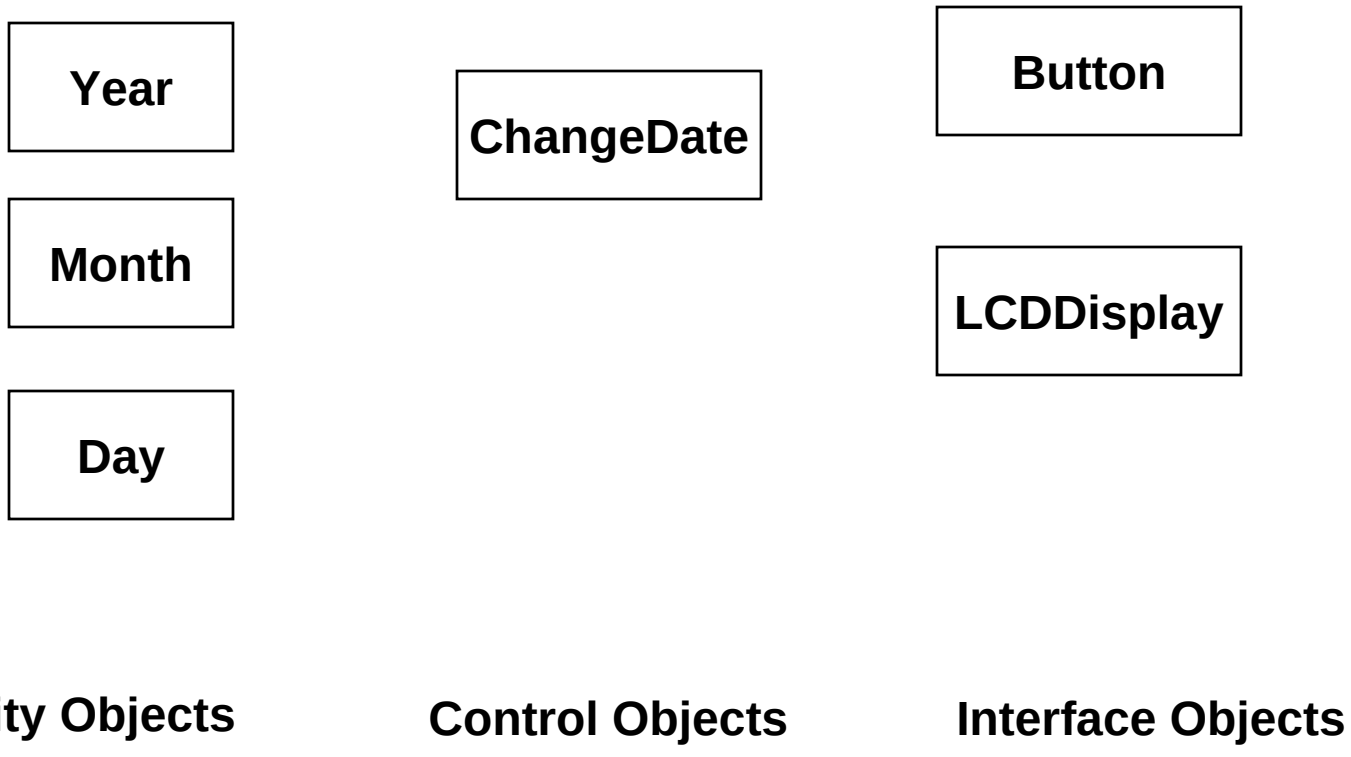
Finding Participating Objects in Use Cases

- Pick a ***use case*** and look at its ***flow of events***
  - **Find terms that developers or users need to clarify in order to understand the flow of events**
  - **Look for recurring nouns (e.g., Incident),**
  - **Identify real world entities that the system needs to keep track of (e.g., FieldOfficer, Dispatcher, Resource),**
  - **Identify real world procedures that the system needs to keep track of (e.g., EmergencyOperationsPlan),**
  - **Identify data sources or sinks (e.g., Printer)**
  - **Identify interface artifacts (e.g., PoliceStation)**
- Be prepared that some objects are still missing and need to be found:
    - **Model the flow of events with a sequence diagram**
- Always use the user's terms

Object Types

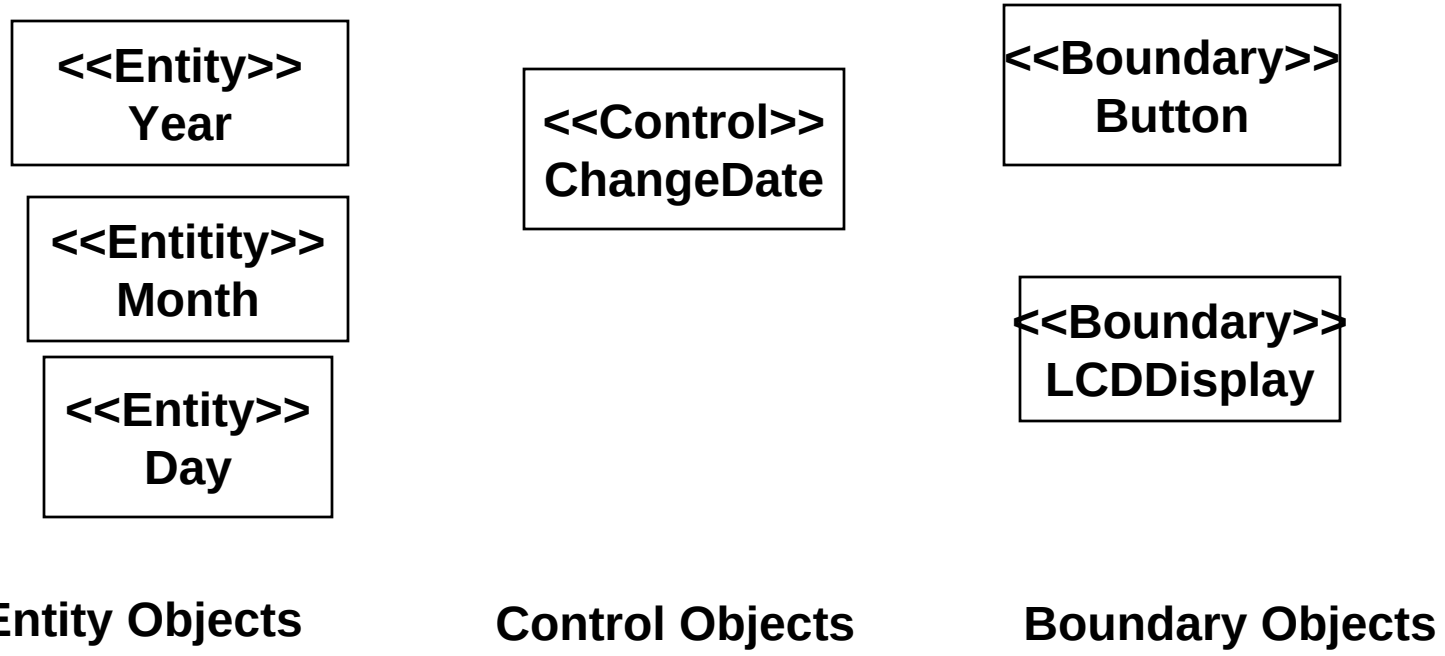
- Entity Objects
  - **Represent the persistent information tracked by the system (Application domain objects, “Business objects”)**
- Boundary Objects
  - **Represent the interaction between the user and the system**
- Control Objects:
  - **Represent the control tasks performed by the system**
- Having three types of objects leads to models that are more resilient to change.
  - **The interface of a system changes more likely than the control**
  - **The control of the system change more likely than the application domain**
- Object types originated in Smalltalk:
  - **Model, View, Controller (MVC)**

Example: 2BWatch Objects



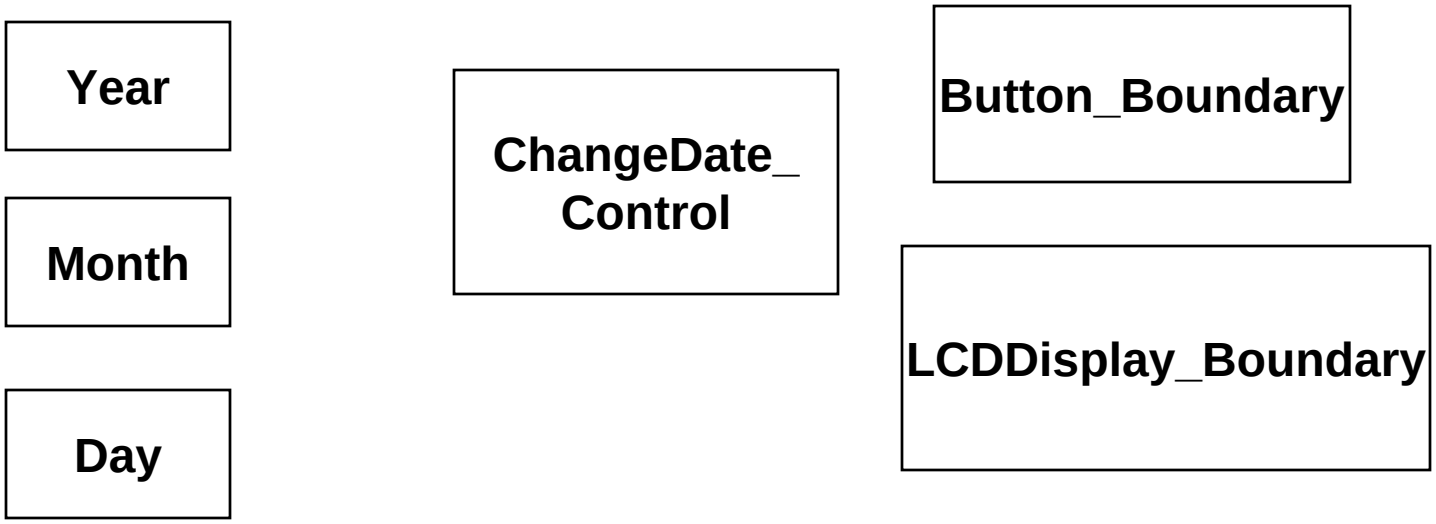
Naming of Object Types in UML

- UML provides several mechanisms to extend the language
- UML provides the stereotype mechanism to present new modeling elements



Recommended Naming Convention for Object Types

- To distinguish the different object tpyes on a syntactical basis, we recommend suffixes:
- Objects ending with the "_Boundary" suffix are boundary objects
- Objects ending with the "_Control" suffix are control objects
- Entity objects do not have any suffix appended to their name.



Example: Flow of events

- The customer enters a store with the intention of buying a toy for his child with the age of n.
- Help must be available within less than one minute.
- The store owner gives advice to the customer. The advice depends on the age range of the child and the attributes of the toy.
- The customer selects a dangerous toy which is kind of unsuitable for the child.
- The store owner recommends a more yellow doll.

Mapping parts of speech to object model components [Abbott, 1983]

Part of speech	*Model component*	*Example*
Proper noun	object	Jim Smith
Improper noun	class	Toy, doll
Doing verb	method	Buy, recommend
being verb	inheritance	is-a (kind-of)
having verb	aggregation	has an
modal verb	constraint	must be
adjective	attribute	3 years old
transitive verb	method	enter
intransitive verb	method (event)	depends on

Another Example

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.
- He tries a videogame, which uses a data glove and a head-mounted display. He likes it.

Is this a good use Case?

Not quite!

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".

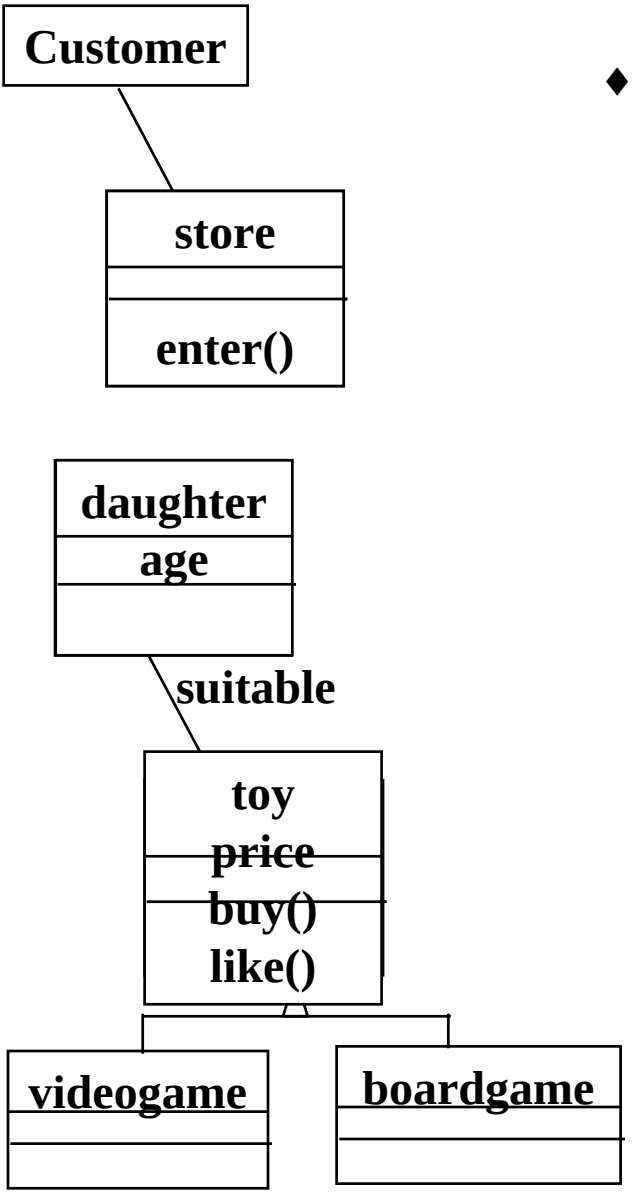
“Monopoly” is probably a left over from the scenario

The use case should terminate with the customer leaving the store

Textual Analysis using Abbot's technique

Example	*Grammatical construct*	*UML Component*
"Monopoly"	Concrete Person, Thing	Object
"toy"	noun	class
"3 years old"	Adjective	Attribute
"enters"	verb	Operation
"depends on…."	Intransitive verb	Operation (Event)
"is a" ,"either..or", "kind of…"	Classifying verb	Inheritance
"Has a ", "consists of"	Possessive Verb	Aggregation
"must be", "less than…"	modal Verb	Constraint

Generation of 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

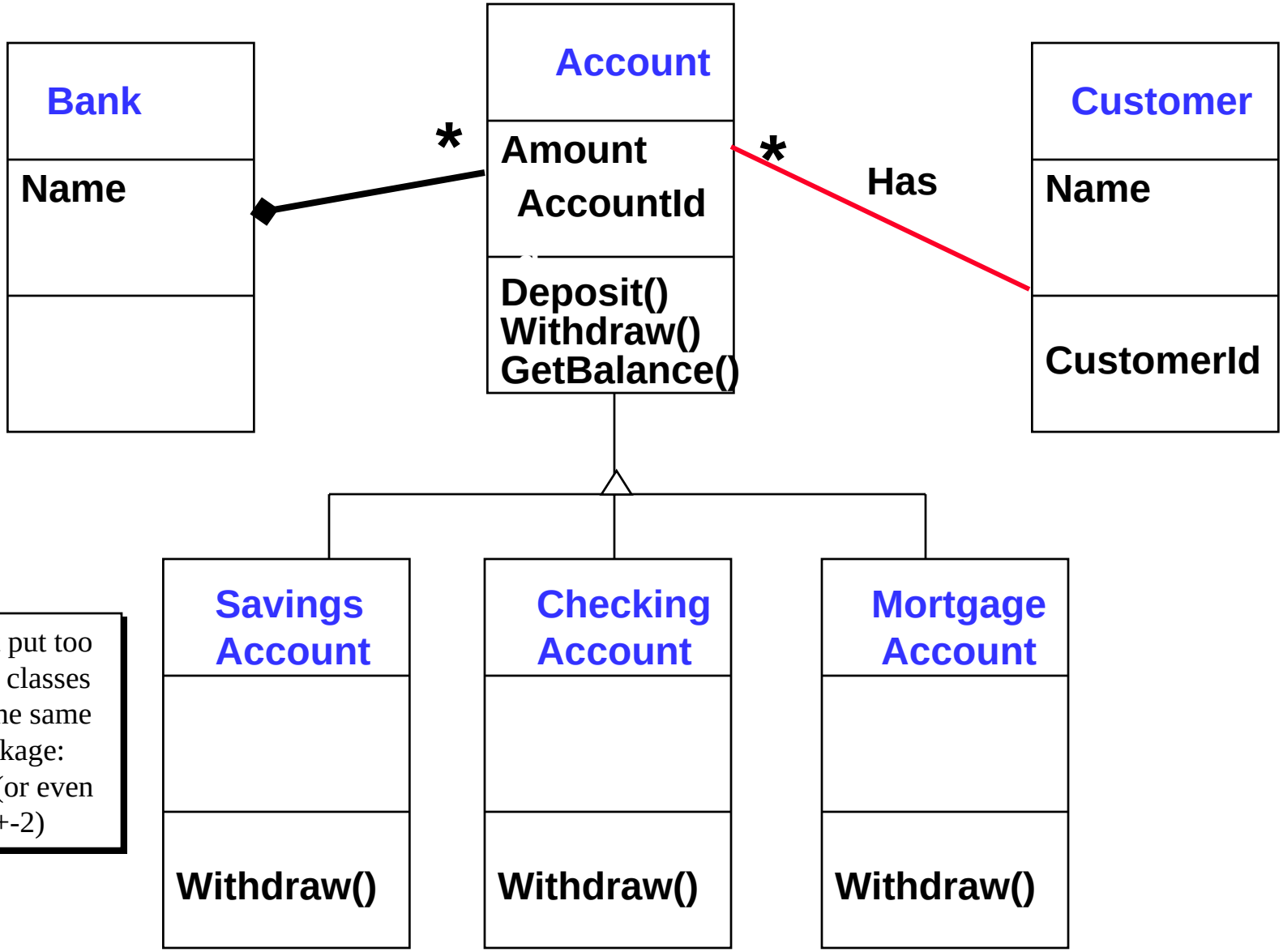
Order of activities in modeling

1. Formulate a few scenarios with help from the end user and/or application domain expert.
2. Extract the use cases from the scenarios, with the help of application domain expert.
3. Analyse the flow of events, for example with Abbot's textual analysis.
4. Generate the class diagrams, which includes the following steps:
    1. **Class identification (textual analysis, domain experts).**
    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 constraints**

Some issues in object modeling

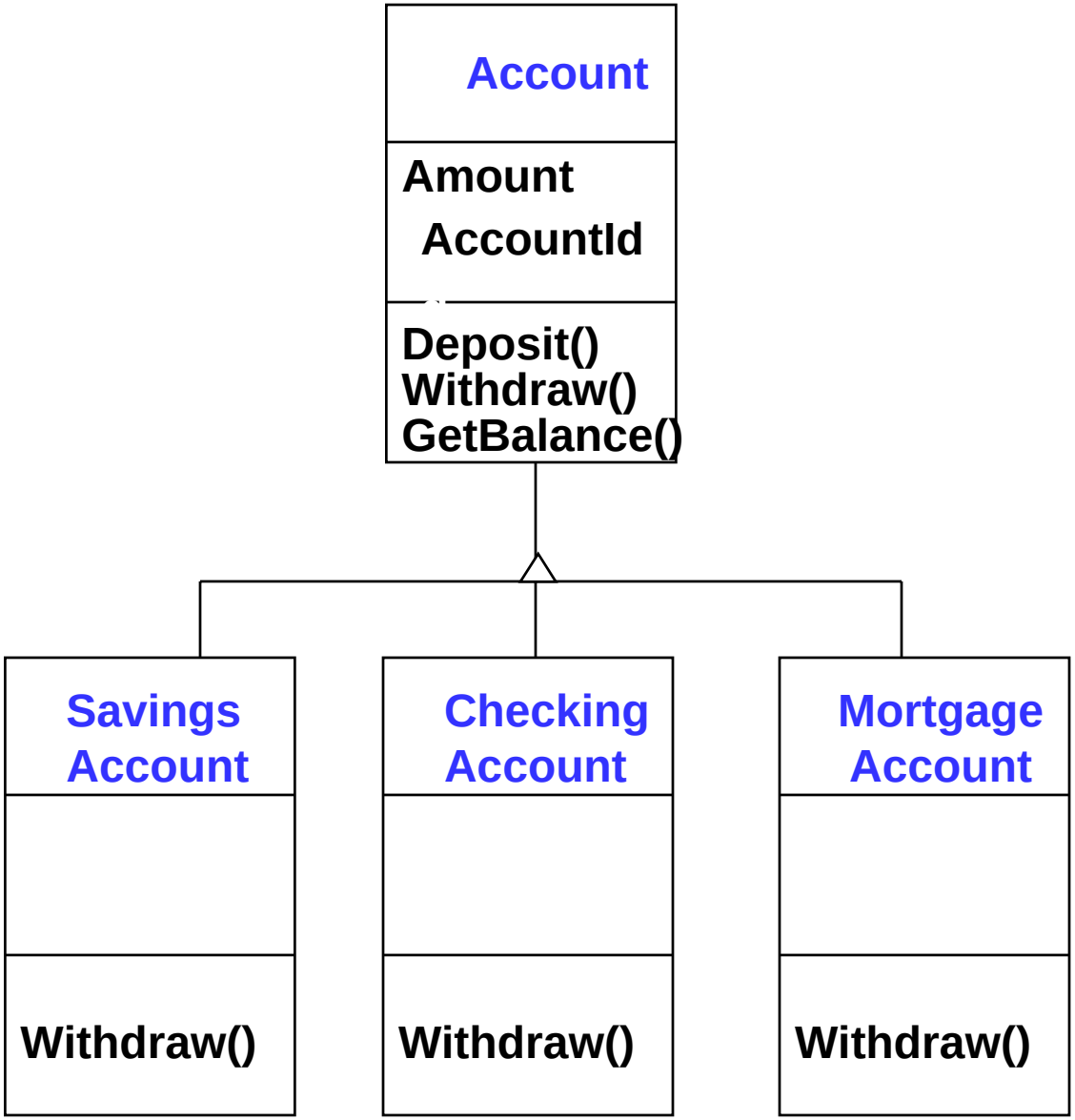
- Improving the readability of class diagrams
- Managing object modeling
- Different users of class diagrams

Avoid Ravioli Models



Don't put too many classes into the same package: 7+-2 (or even 5+-2)

Put Taxonomies on a separate Diagram



Project Management Heuristics

- Explicitly schedule meetings for object identification
- First just find objects
- Then try to differentiate them between entity, interface and control objects
- Find associations and their multiplicity
  - **Unusual multiplicities usually lead to new objects or categories**
- Identify Inheritance: Look for a Taxonomy, Categorize
- Identify Aggregation

- Allow time for brainstorming , Iterate, iterate

Who uses class diagrams?

- Purpose of Class diagrams :
  - **The description of the static properties of a system (main purpose)**
- Who uses class diagrams?
- The **customer** and the **end user** are often not interested in class diagrams. They usually focus more on the functionality of the system.
- **The application domain expert** uses class diagrams to model the application domain
- The **developer** uses class diagrams during the development of a system,that is, during analysis, system design, object design and implementation.

Class-diagrams have different types of „users“

- According to the development activity, the developer plays different roles.
  - Analyst
  - System-Designer,
  - DetailedDesigner
  - Implementor**.**
- In small systems some of the roles do not exist or are played by the same person.
- Each of these roles has a different view about the models.
- Before I describe these different views, I want to distinguish the types of classes that appear in class diagrams.
  - **Application domain classes**
  - **Solution domain classes**

Application domain vs solution domain

- Application domain:
  - **The problem domain (financial services, meteorology, accident management, architecture, …).**
- Application domain class:
  - **An abstraction in the application domain. If we model business applications, these classes are also called business objects.**
  - **Example: Board game, Tournament**
- Solution domain:
  - **Domains that help in the solution of problems (tele communication, data bases, compiler construction, operting systems, ….)**
- Solution domain class:
- An abstraction, that is introduced for technical reasons, because it helps in the solution of a problem.
  - **Examples: Tree, Hashtable, Scheduler**

Analysis model

- The Analysis modell is constructure during the analyse phase.
  - **Main stake holders: End user, Customer, Analyst.**
  - **The diagram contains only application domain classes.**

- The analysis model is the base for communication between analyists, experts in the application domain and end users of the system.

Object design model

- The object design model (sometimes also called **specification model) is created during the object design phase**
  - **Main stake holders are class specificiers, class implementors and class users**
  - **The class diagrams contain applikation and solution domain classes.**
- The object design model is the basis of communication between designers and implementors.

Summary

- Modeling vs reality
- System modeling
  - **Object model**
  - **Dynamic model**
  - **Functional model**
- Object modeling is the central activity
  - **Class identification is a major activity of object modeling**
  - **There are some easy syntactic rules to find classes/objects**
- Different roles during software development
- Requirements Analysis Document Structure