Injibara University

College of Natural and Computational Science

Department of Software Engineering

Software Engineering

Course Code:CoSc3061

Molla K. (MSc)

Course Title: Software Engineering

Chapter -5 Analysis

An Overview of Analysis

- **Analysis** focuses on producing a model of the system, called the *Analysis Model*, which is correct, complete, consistent, and verifiable.
- **Analysis** is different from requirements elicitation in that developers focus on structuring and formalizing the requirements elicited from users.
- **Analysis** results in a model of a system that aims to be correct, complete, consistent and unambiguous.
- Formalize the requirements specification validate, correct and clarify the requirements specification if any errors or ambiguities are found.
- Generally the client and user are also involved.

An Overview of Analysis (Con'd)

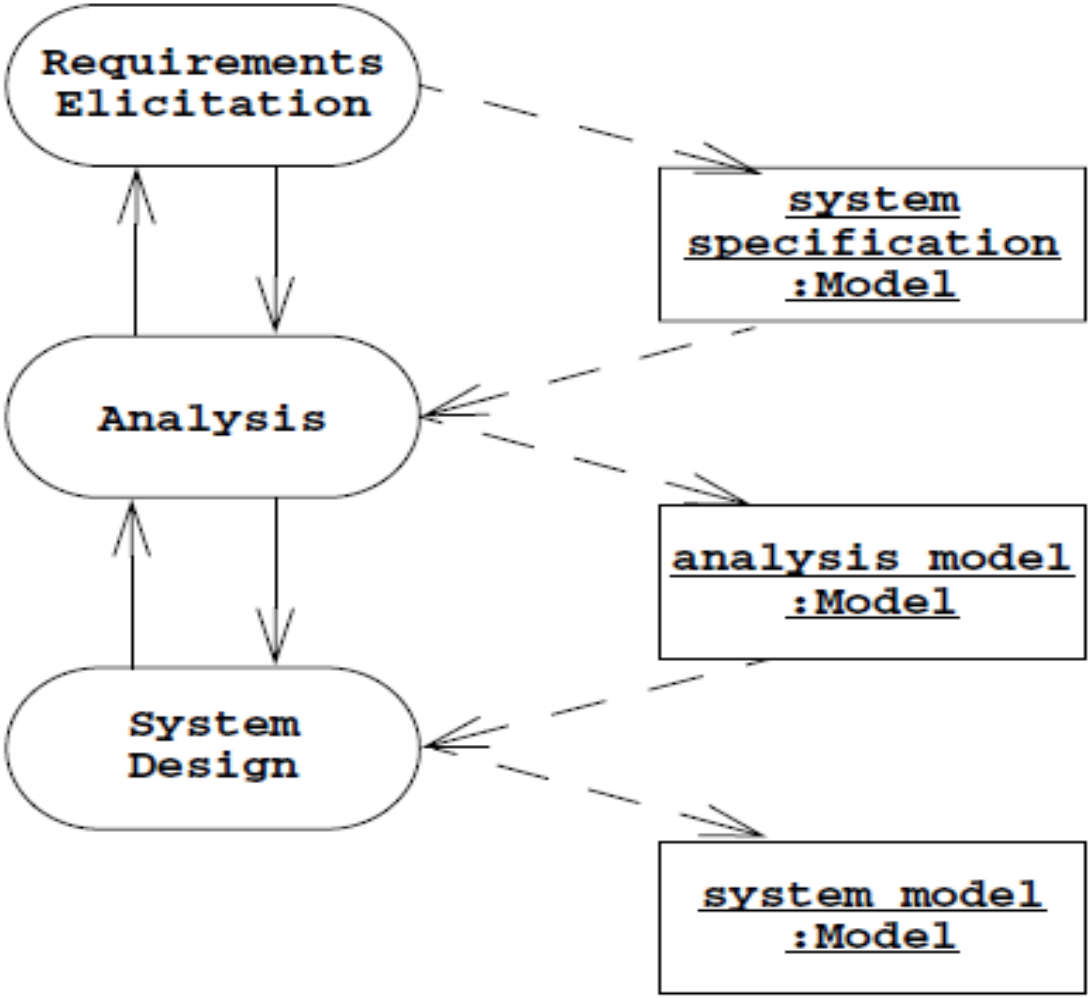


Figure 1: Products of requirements elicitation and analysis

Analysis Models

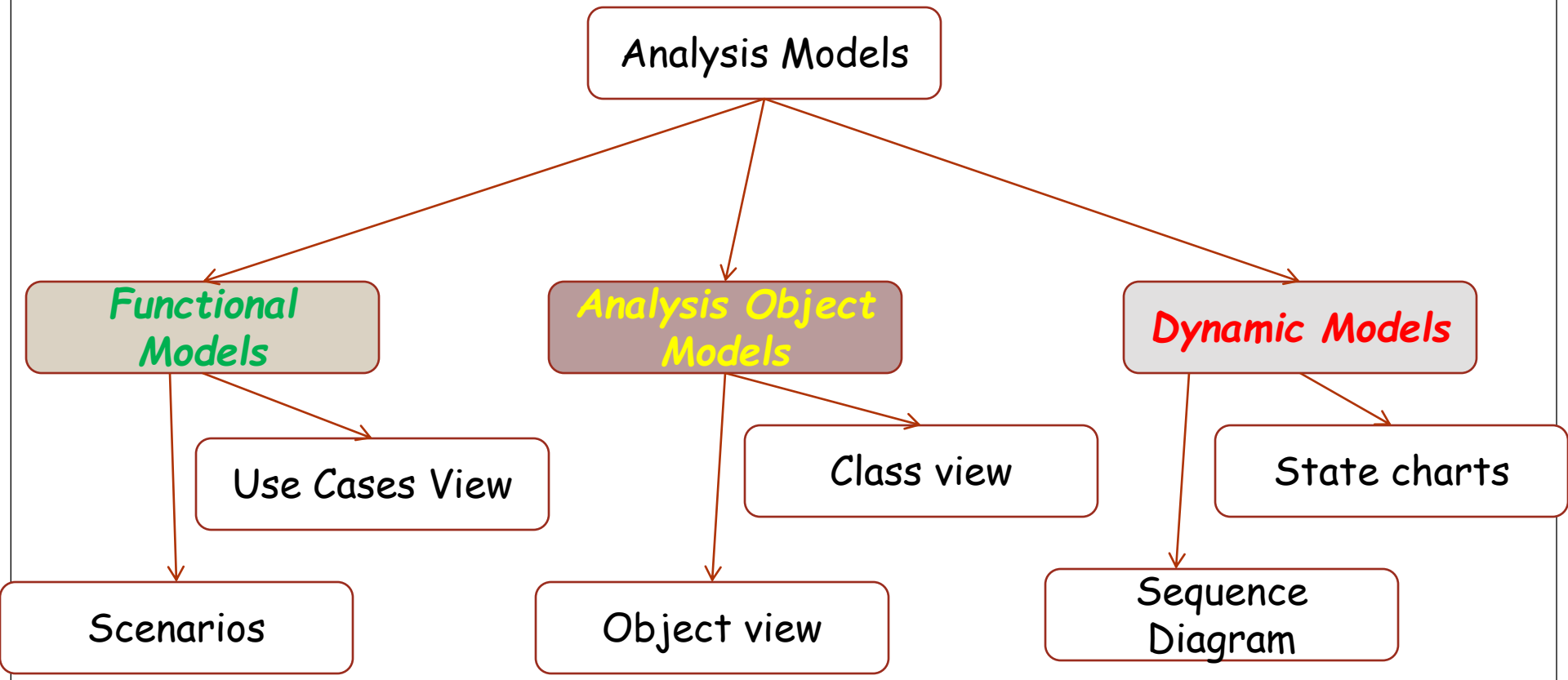
- The analysis model is composed of three individual techniques of models:
- The Functional Model, represented by use cases and scenarios and describes the functionality of the system from the user's point of view.
- The analysis Object Model, represented by class and object diagrams, and describes the structure of the system in terms of objects, attributes, associations, and operations.
- It develops the static structure of the system regarding to object.
- It recognizes the relationship ship between objects and classes.
- The analysis object model, depicted with UML class diagrams, includes classes, attributes, and operations.

Cont'd

- The Dynamic Model, represented in UML with **interaction** diagrams, State Chart, Sequence and Activity diagrams and describes the **internal behavior** of the system.
- The **dynamic model** focuses on the **behavior** of the system .
- The dynamic model **serves** to **assign responsibilities** to individual classes and, in the process, to identify new classes, associations, and attributes to be added to the analysis object model.
- Sequence diagrams represent the interactions among a set of objects during a single use case.

Analysis Models (con'd)

✓In **UML**, the **functional** model is represented with **use case** diagrams, the **object** model with **class diagrams**, and the **dynamic** model with **state machine** and **sequence** diagrams .



➢**The analysis model represents the system under development from the user's point of view.**

Analysis Concepts

- In this section, we describe the main analysis concepts used in this chapter.

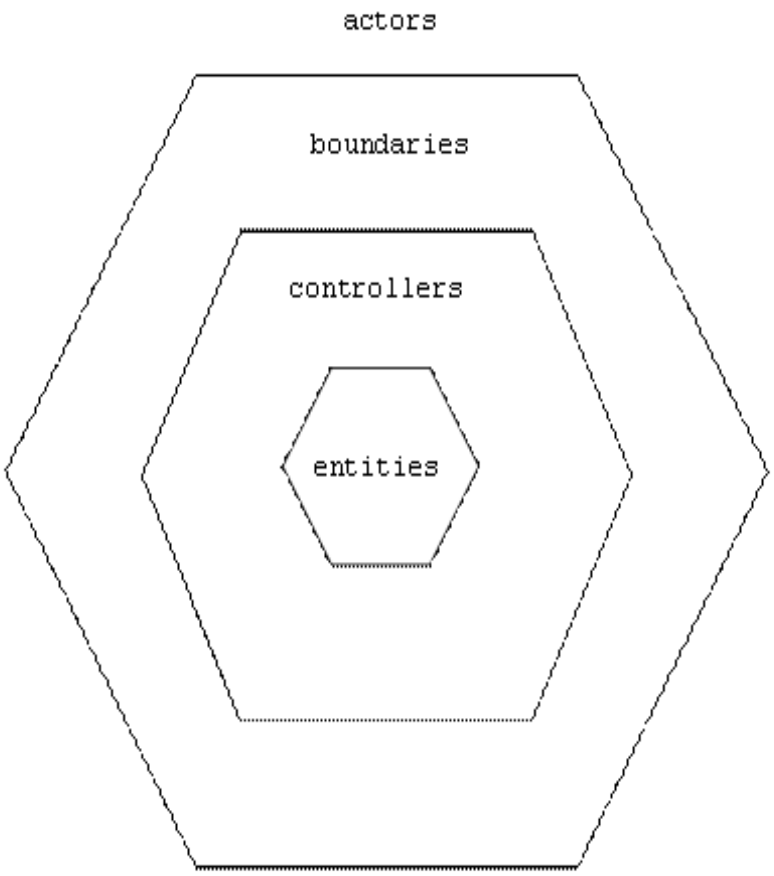
➢ In particular, we describe:

  - Entity, boundary, and control objects
  - Association multiplicity
  - Qualified associations
  - Generalization

Entity, Boundary and Control Objects

- The analysis object model consists of **Entity**, **Boundary**, and **Control** objects [Jacobson et al., 1999].
- **Entity objects:-** represent the **persistent** information tracked/followed by the **system**.
- **Boundary objects:-**represent the **interactions** between the **actors** and the **system**.
- **Control objects:-**represent the **tasks** that are performed by the **user** and supported by the **system**.

Cont'd

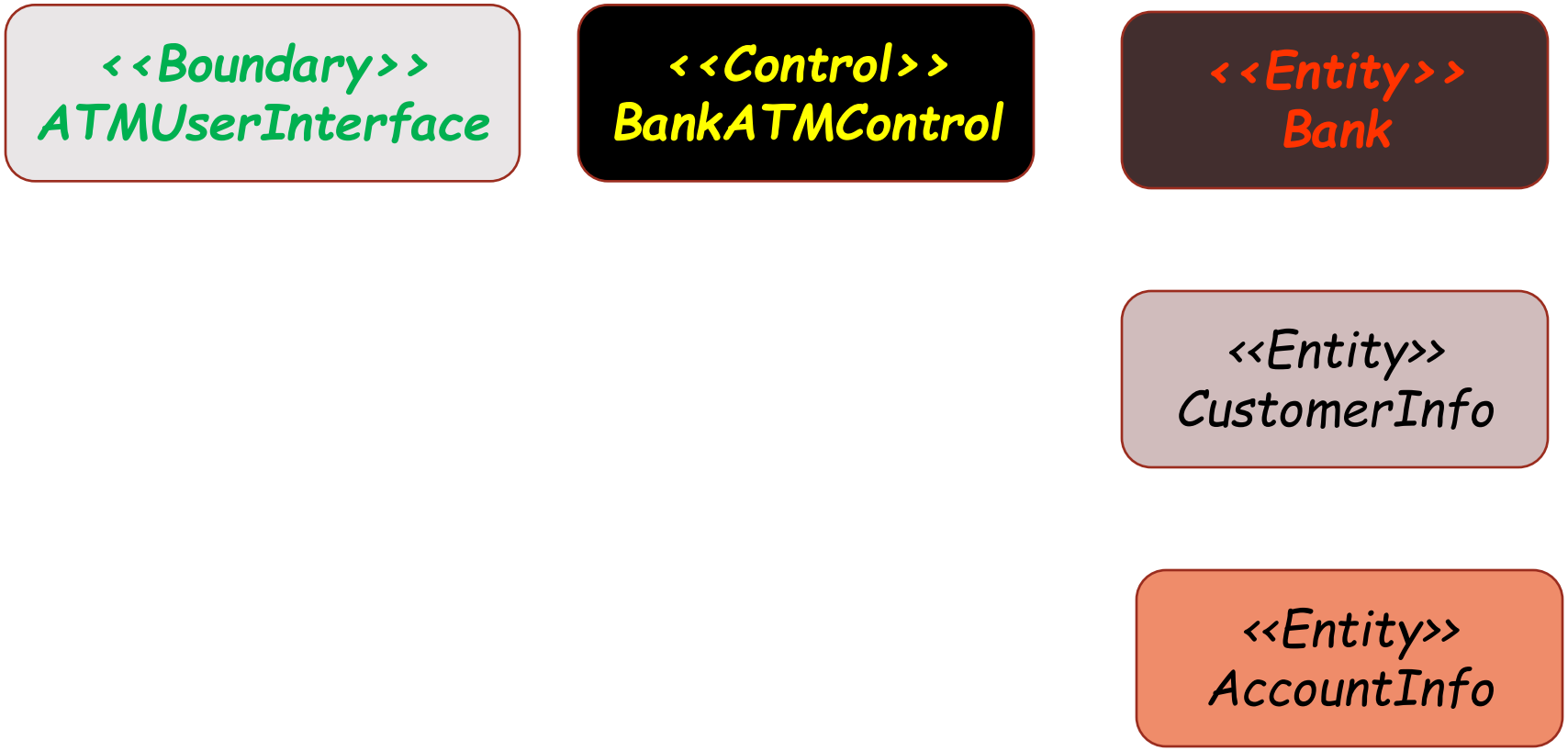


- ✓ **Actors** interact with boundary objects.
- ✓ **Boundary** objects issue commands to controller objects.
- ✓ **Controller** objects may send queries back to the boundary objects to get more information from the actors.
- ✓ **Controllers** then update **entities**.
- ✓ Boundaries refresh themselves as needed to reflect changes among the entities.

❖ **Four rules apply to their communication:**

- ➢ Actors can only talk to boundary objects.
- ➢ Boundary objects can only talk to controllers and actors.
- ➢ Entity objects can only talk to controllers.
- ➢ **Controllers** can talk to **boundary objects** and **entity objects**, and **to other controllers**, but not to actors

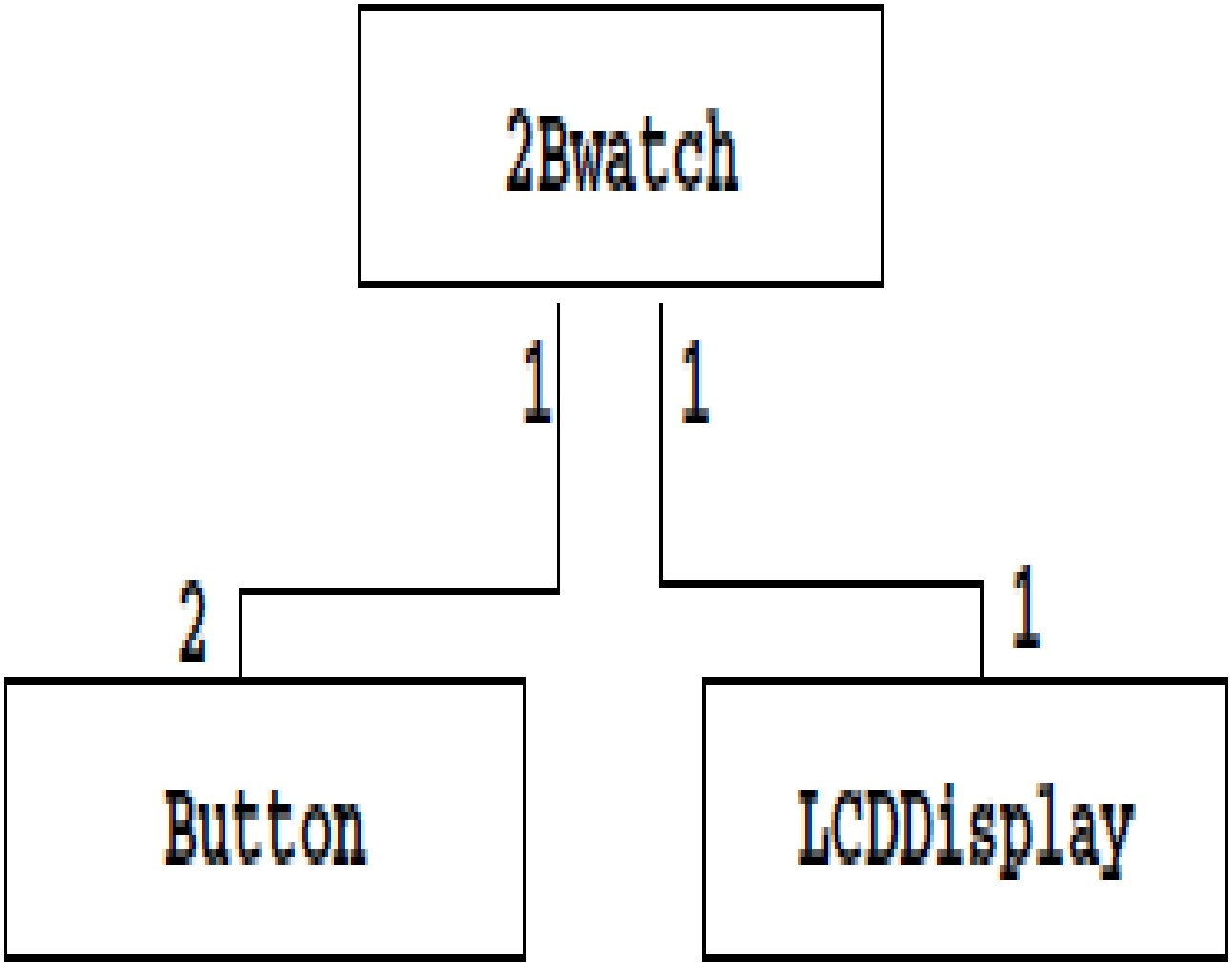
Example for ATM



Association Multiplicity

- The end of an association can be labeled by a set of integers called **Multiplicity**.
- The multiplicity indicates the number of links that can legitimately originate from an instance of the class connected to the association end.
- For example, in figure below, a **2Bwatch** has exactly two buttons and one **LCD** display.

Association Multiplicity (Con'd)



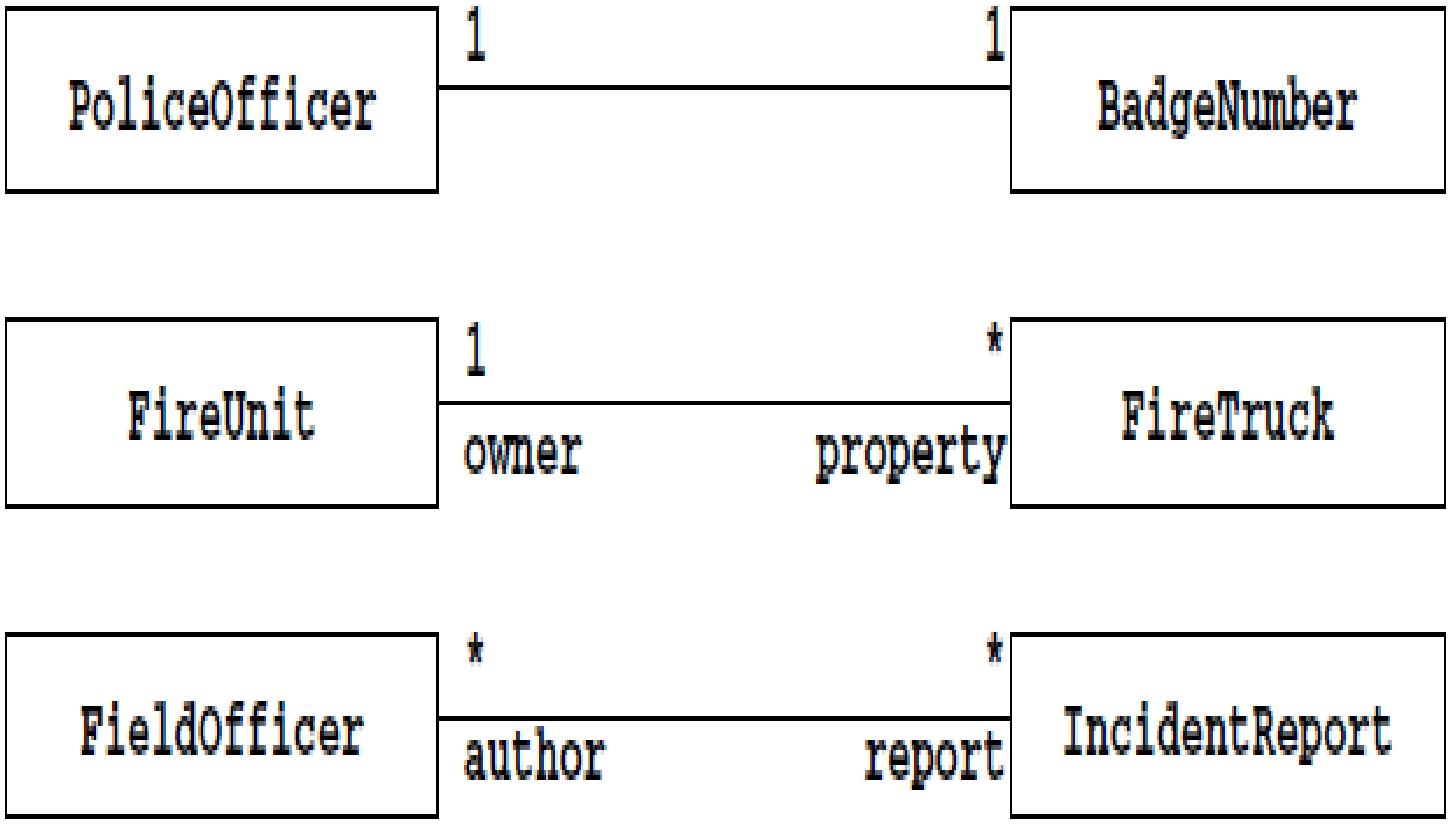
Association Multiplicity (Con'd)

- In UML, an association end can have an arbitrary set of integers as a multiplicity. For example, an association could allow only a prime number of links and, thus, would have a multiplicity 1, 2, 3, 5, 7, 11, 13, … .
- In practice, however, most of the associations we encounter belong to one of the following three types
- A **one-to-one** association has a multiplicity 1 on each end. A one-to-one association between two classes (e.g., PoliceOfficer and BadgeNumber), means that exactly one link exists between instances of each class (e.g., a PoliceOfficer has exactly one BadgeNumber , and a BadgeNumber denotes exactly one PoliceOfficer).

Association Multiplicity (con'd)

- A **one-to-many association** has a multiplicity 1 on one end and 0..n (also represented by a star) or 1..n on the other.
- A one-to-many association between two classes (e.g., Person and Car) denotes composition (e.g., a FireUnit owns one or more FireTrucks, a FireTruck is owned exactly by one FireUnit).
- A **many-to-many association** has a multiplicity 0..n or 1..n on both ends.
- A *many-to-many association* between two classes (e.g., FieldOfficer and IncidentReport) denotes that an arbitrary number of links can exist between instances of the two classes (e.g., a FieldOfficer can write for many IncidentReport, an IncidentReport can be written by many FieldOfficers).
- This is the most complex type of association.

Association Multiplicity (con'd)



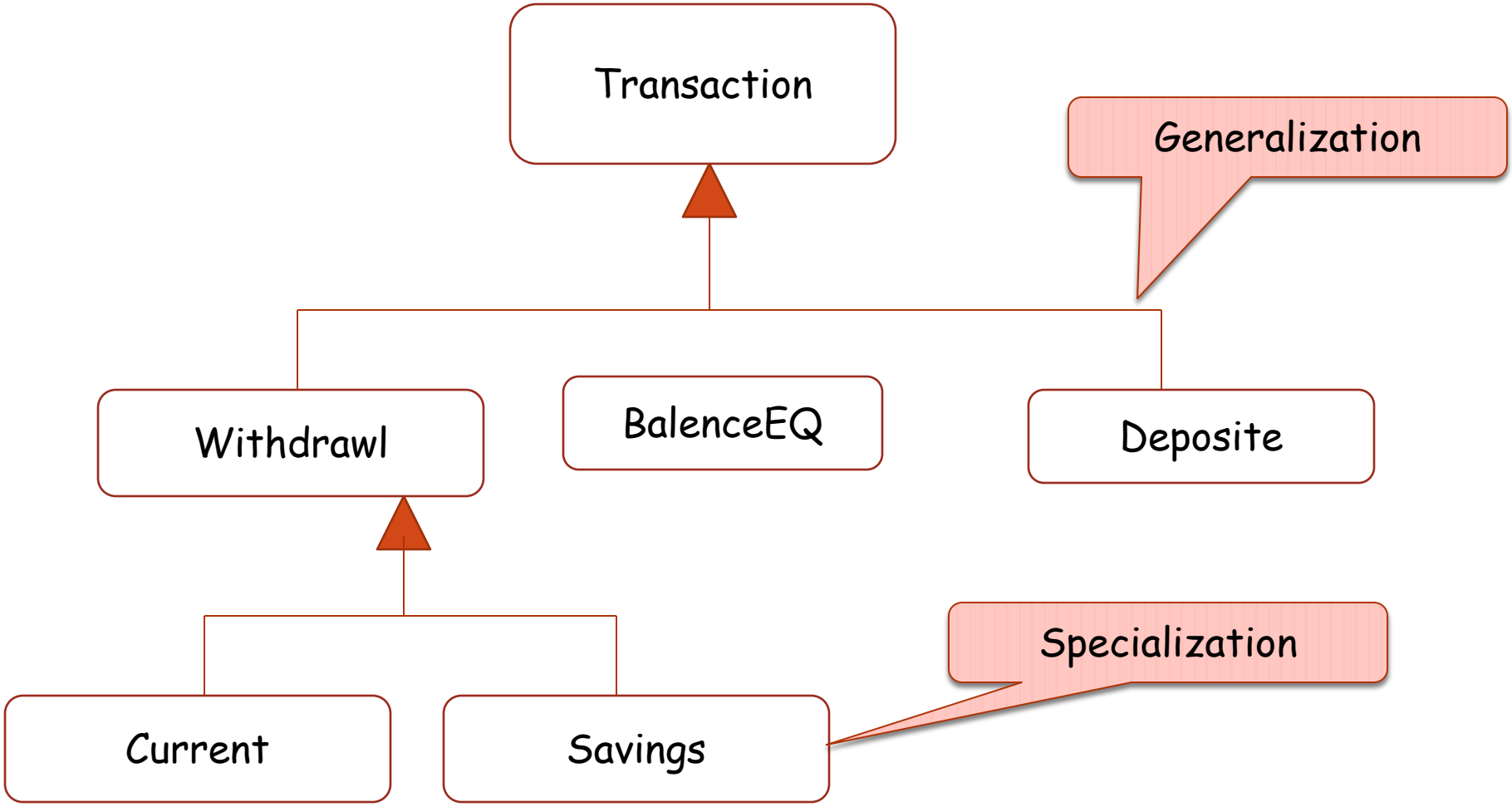
Qualified Associations

- **Qualification** is a technique for reducing multiplicity by using keys.
- Associations with a 0..1 or 1 multiplicity are easier to understand than associations with a 0..n or 1..n multiplicity.

Generalization

- **Generalization** enables us to organize concepts into hierarchies.
- At the top of the hierarchy is a general concept, and at the bottom of the hierarchy are the most specialized concepts
- There may be any number of intermediate levels in between, covering more-or-less generalized concepts .
- Such hierarchies allow us to refer to many concepts precisely

Generalization and specialization



Analysis Activities: From Use Cases to Objects

- In this section, we describe the activities that transform the use cases and scenarios produced during requirements elicitation into an analysis model. Analysis activities include:
- Identifying **Entity** objects
- Identifying **Boundary** objects
- Identifying **control** objects
- **Mapping use cases** to objects
- Identifying **Associations** among objects
- **Modeling Interactions** between objects: sequence diagrams
- Identifying **Associations**.
- Identifying **Attributes**
- **Reviewing** the analysis model

Identifying Entity Objects

- Participating objects are found by examining each use case and identifying candidate objects.
- ***Natural Language Analysis*** [**Abbott**, **1983**] is an intuitive set of heuristics for identifying objects, attributes, and associations from a system specification.
- Abbott's heuristics maps parts of speech (e.g., nouns, having verbs, being verbs, adjectives) to model components (e.g., objects, operations, inheritance relationships, classes).

Identifying Entity Objects (con'd)

Part of speech	Model component	Examples
Proper noun	Object	Alice
Common noun	Class	FieldOfficer
Doing verb	Operation	Creates, submits, selects
Being verb	Inheritance	Is a kind of, is one of either
Having verb	Aggregation	Has, consists of, includes
Modal verb	Constraints	Must be
Adjective	Attribute	Incident description

Table : Abbott's heuristics for mapping parts of speech to model components [Abbott, 1983]

Heuristics for Identifying Entity Objects

- Terms that developers or users need to clarify in order to understand the use case.
- Recurring nouns in the use cases (ex: session)
- Real-world entities that the system needs to track (ex: *Customer, Admin, FieldOfficer, Dispatcher, Resource)*
- Real world activities that the system needs to track (Ex:transaction)
- Use cases (e.g., Report emergency)
- Data sources or sinks (e.g., Printer)
- Always use the user's terms

Identifying Boundary Objects

- Boundary objects represent the system interface with the actors.
- In each use case, each actor interacts with at least one boundary object.
- The boundary object collects the information from the actor and translates it into an interface neutral form that can be used by the entity objects and also by the control objects.

Heuristics for Identifying the Boundary Object

- Identify user interface controls that the user needs to initiate the use case (ex: press OK button)
- Identify forms and windows the users needs to enter data into the system (ex: form to fill details)
- Identify notices and messages the system uses to respond to the user (ex: acknowledge Notice by system)
- Do not model the user interface details

Identifying Control Objects

- Control objects are responsible for coordinating boundary and entity objects.
- Control objects usually do not have a concrete counterpart /equivalent in the real world.
- There is often a close relationship between a use case and a control object.
- A control object is usually created at the beginning of a use case and ceases to exist at its end.
- It is responsible for collecting information from the boundary objects and dispatching it to entity objects.

Heuristics for Identifying the Control Object

- Identify one **control object** per **use case**
- Identify one **control object** per **actor** in the use case
- The life span of the control object should cover the extent of the use case or the extent of a user session.
- If it is difficult to identify the beginning and the end of a control object activation, the corresponding use case probably does not have well defined **entry** and **exit** condition.

Modeling Interactions Between Objects: Sequence Diagrams

- A sequence diagram ties use cases with objects.
- It shows how the behavior of a use case (or scenario) is distributed among its participating objects.
- Sequence diagrams are usually not a good medium for communication with the user.
- They represent, however, another shift in perspective and allow the developers to find missing objects or grey areas in the system specification.

Heuristics for Drawing **Sequence** Diagrams

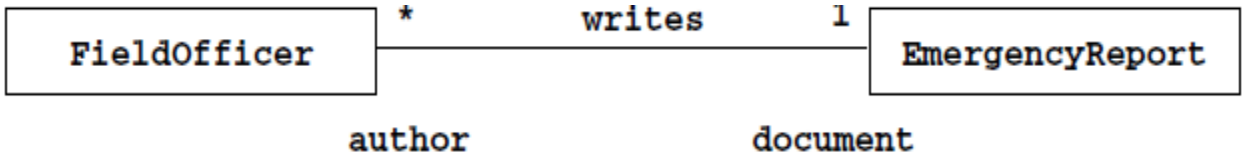
- The **first column** should correspond to the actor who initiated the use case.
- The **second column** should be an ***boundary object*** (that the actor used to initiate the use case).
- The **third column** should be the ***control object*** that manages the rest of the use case.
- Control objects are created by boundary objects initiating use cases.
- Boundary objects are created by control objects.
- Entity objects are accessed by control and boundary objects.
- Entity objects never access boundary or control objects, this makes it easier to share entity objects across use cases.

Identifying Associations

- Sequence diagrams allow developers to represent interactions among objects over time, class diagrams allow developers to describe the spatial connectivity of objects.
- An association shows a relationship between two or more classes.
- Identifying associations has two advantages.
- First, it clarifies the analysis model by making relationships between objects explicit
- Second, it enables the developer to discover boundary cases associated with links.
- Boundary cases are exceptions that need to be clarified in the model.

Identifying Associations (con'd)

- Associations have several properties:
- **A name**, to describe the association between the two classes (e.g., Writes in Figure below).Association names are optional and need not be unique globally.
- **A role** at each end, identifying the function of each class with respect to the associations (e.g., author is the role played by FieldOfficer in the Writes association).
- **A multiplicity** at each end, identifying the possible number of instances (e.g., * indicates a FieldOfficer may write zero or more EmergencyReports, whereas 1 indicates that each EmergencyReport has exactly one FieldOfficer as author.



Identifying Attributes

- Attributes are properties of individual objects.
- Attributes have:
  - A Name identifying them within an object.
  - A brief description.
  - A type describing the legal values it can take.

Z End of This Chapter!