Static Modeling

SWE 321

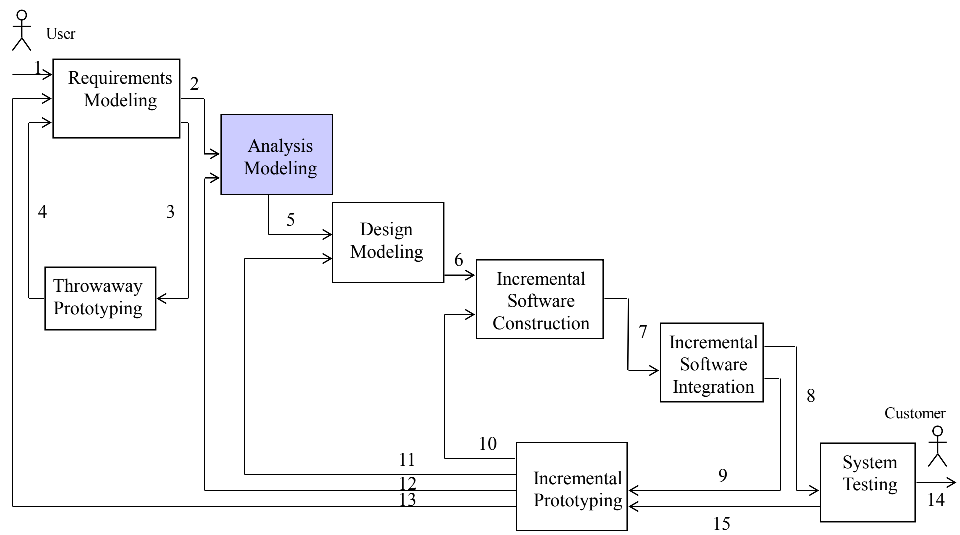
Fall2014

Steps in Using COMET/UML

1 Develop Software Requirements Model
- Develop Use Case Model (Chapter 6)

2 Develop Software Analysis Model
- Develop static model of problem domain (Chapter 7)
- Structure system into objects (Chapter 8)
- Develop statecharts for state dependent objects (Chapter 10)
- Develop object interaction diagrams for each use case (Chapter 9, 11)

3 Develop Software Design Model



Outline

- Building blocks of static modeling:
  - Objects and Classes
  - Class Diagrams
  - Relationships between classes
    - Associations
    - Composition / Aggregation
    - Generalization / Specialization
- Preliminary Class Design
  - Identifying candidate classes
  - Conceptual static model
  - System Context Class Diagram

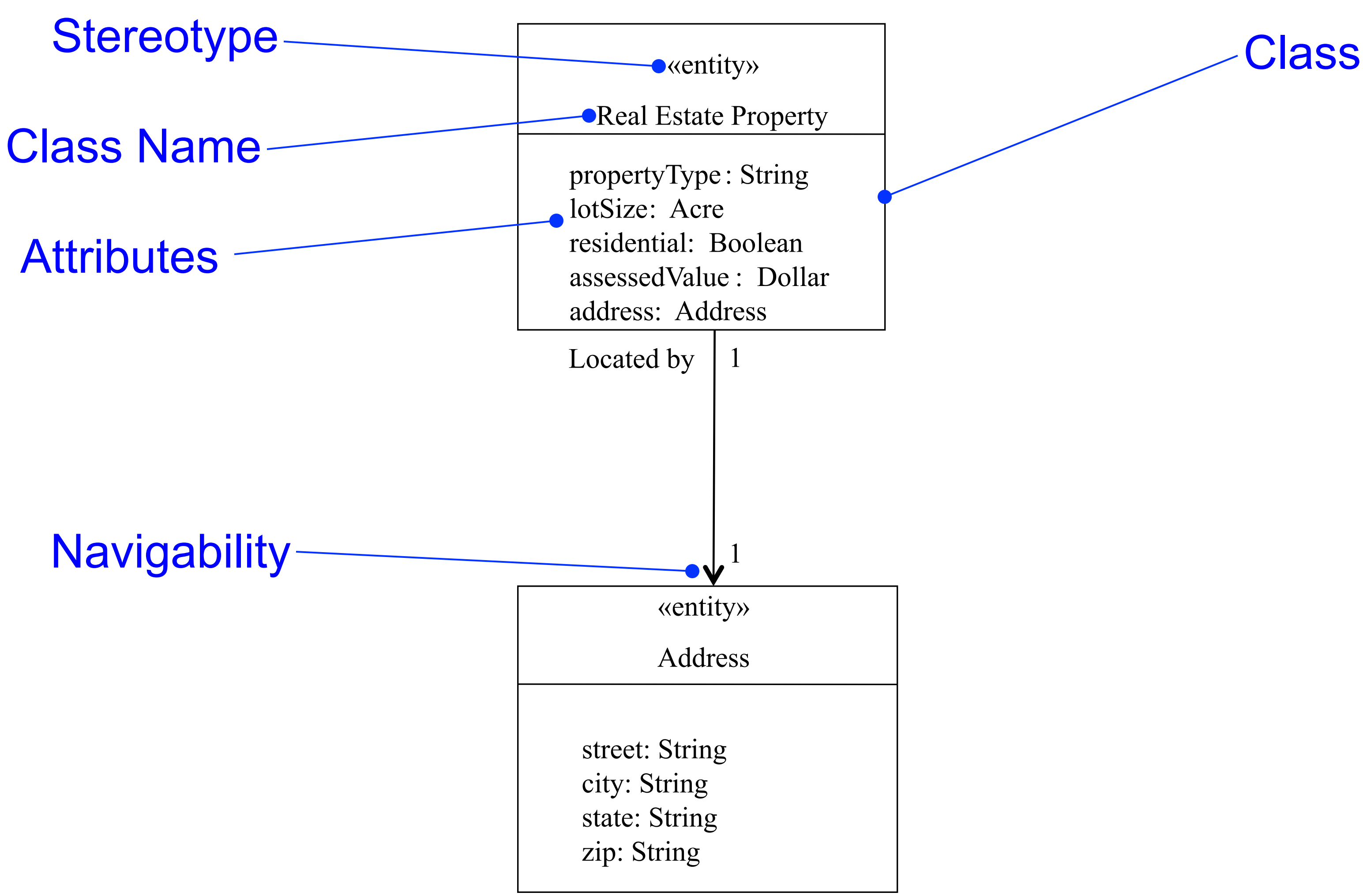
Class Diagrams

- UML class diagrams capture the static structure of a system
- Class
  - Analysis-level class often corresponds to real-world things
  - Represents a collection of identical objects (instances)
  - Described by means of attributes (data items)
  - Has operations to access data maintained by objects
  - Each object instance can be uniquely identified
- Relationships between classes
  - Associations
  - Composition / Aggregation
  - Generalization / Specialization

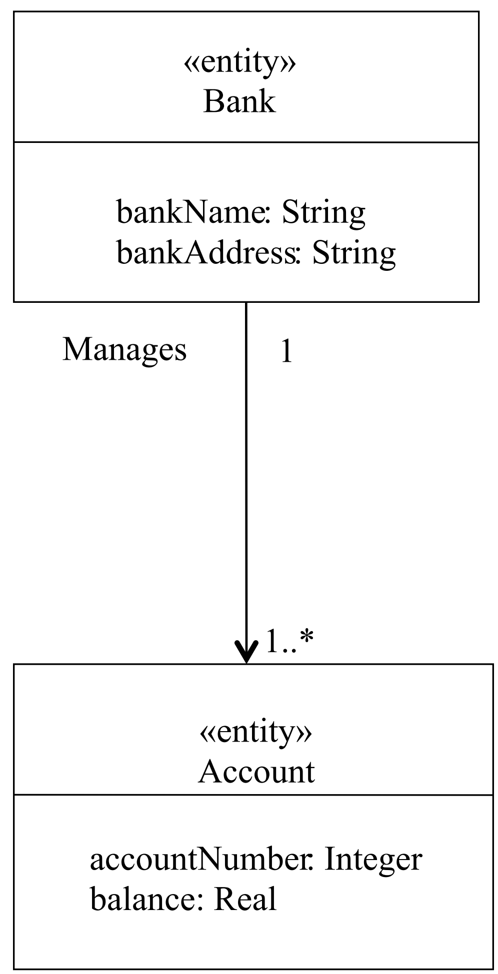
Associations

- Association is
  - Static, structural relationship between classes
  - E.g, Employee works in Department
- Multiplicity of Associations
  - Specifies how many instances of one class may relate to a single instance of another class
  - Options for multiplicity:
    - 1-to-1
    - 1-to-many
    - 0, 1, or many
    - Many-to-many

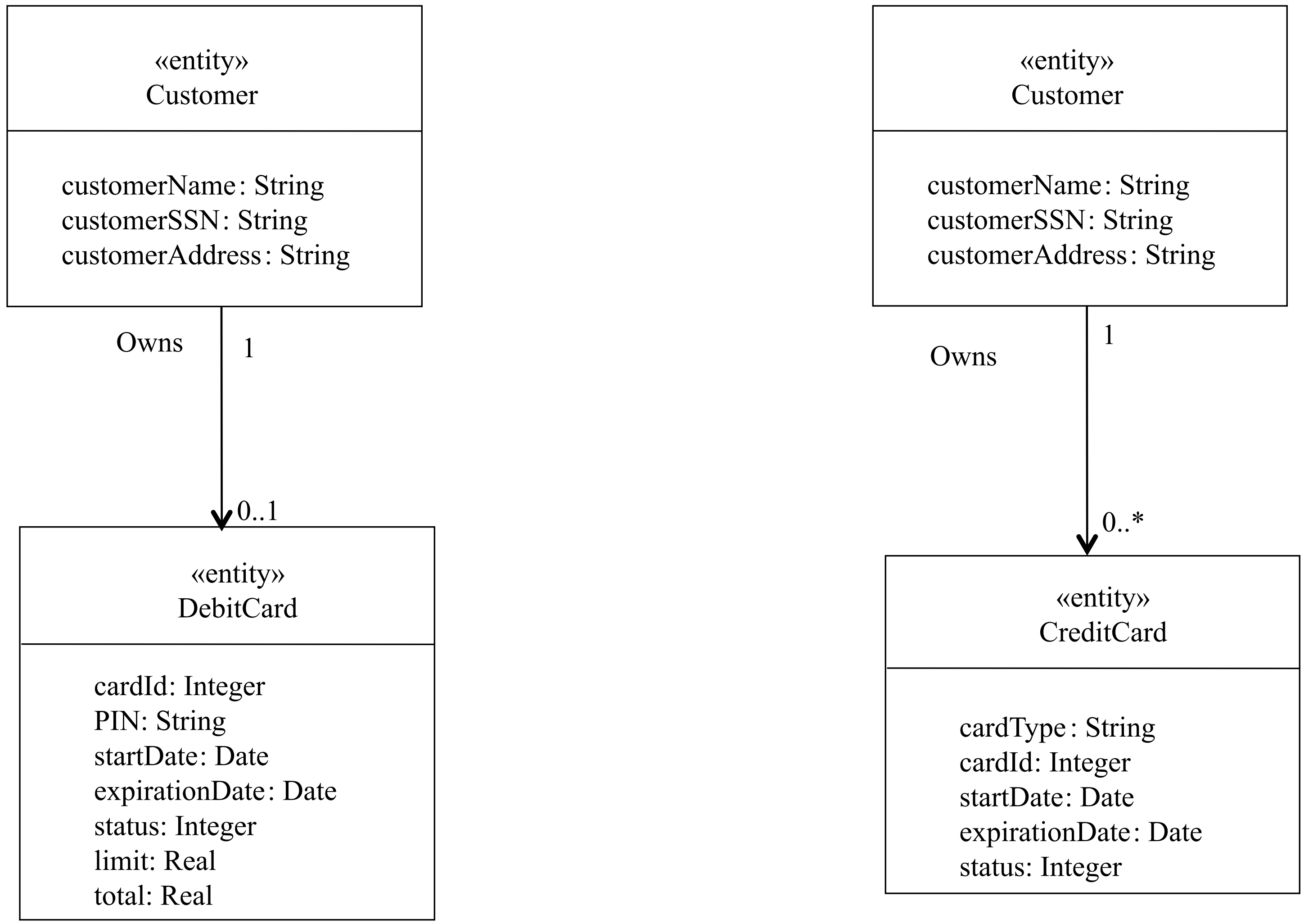
1-to-1 Associations



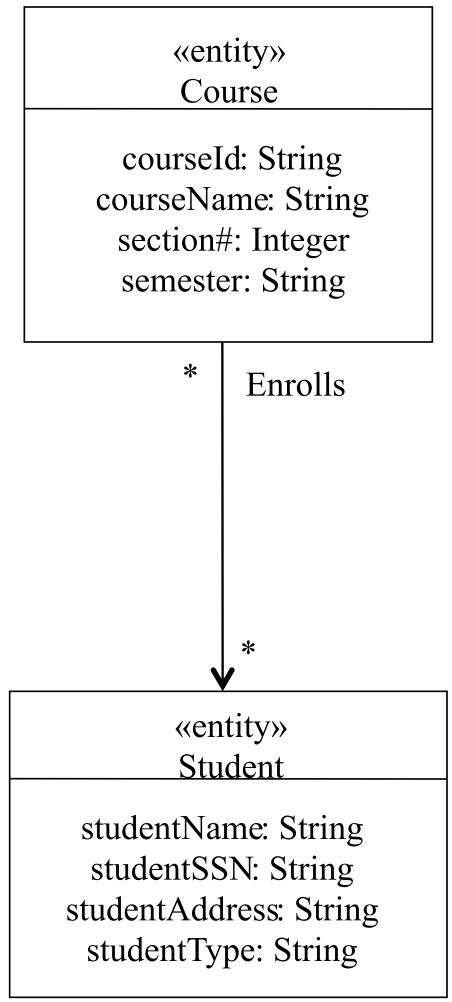
1-to-Many Associations



Optional (0, 1, or Many) Associations



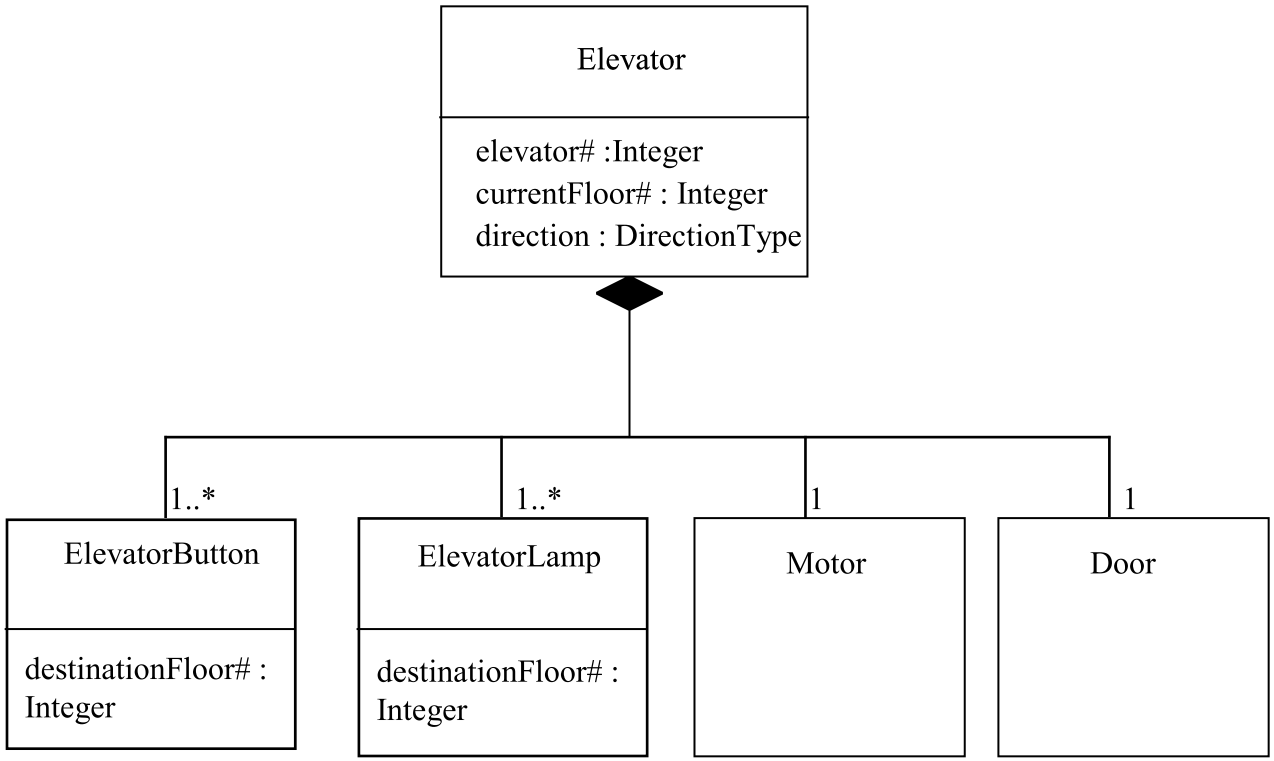
Many-to-Many Associations



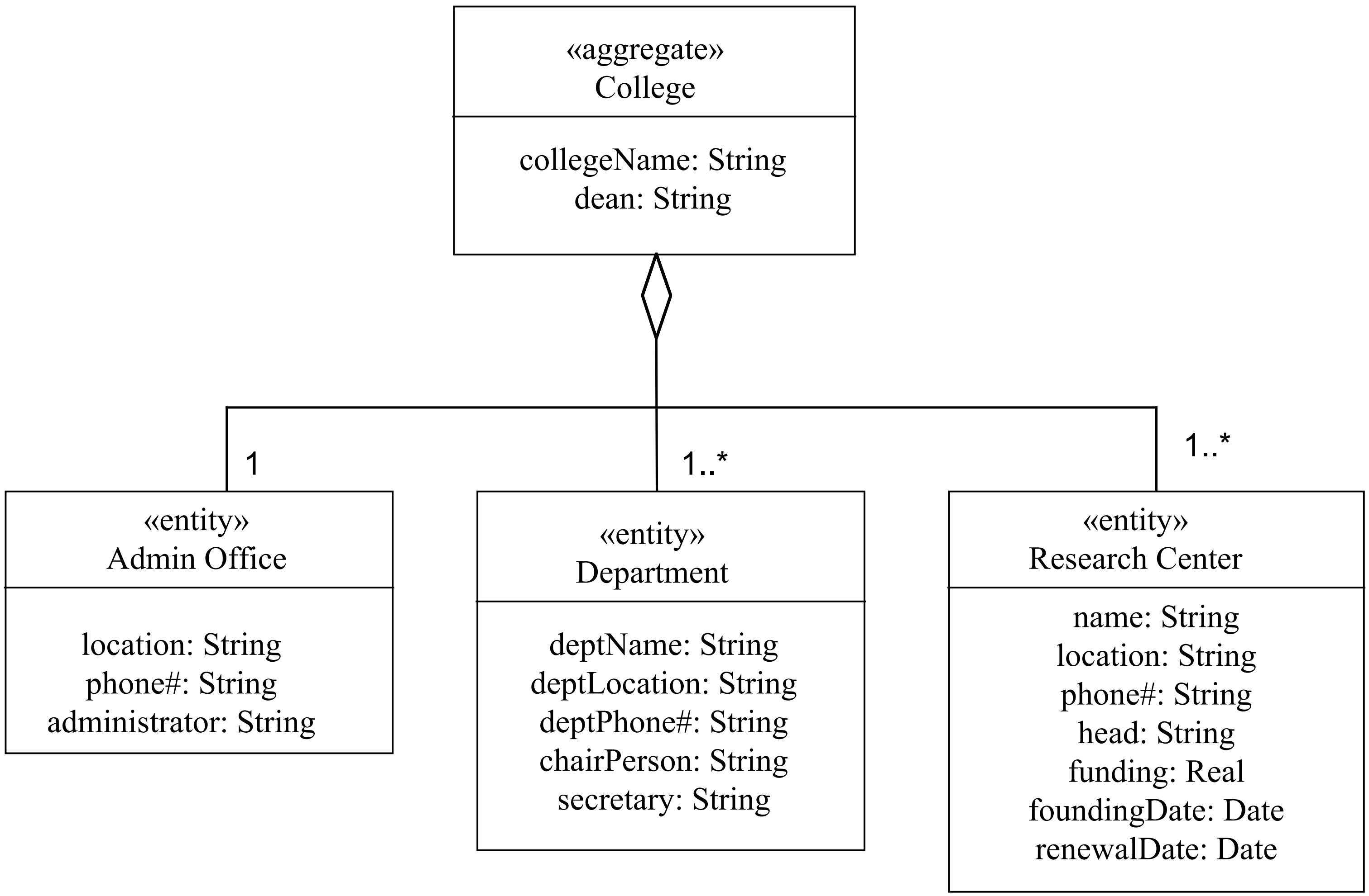
Composition and Aggregation Hierarchies

- Whole/Part Relationships
  - Show components of more complex class
  - Composition is stronger relationship than aggregation
- Composition Hierarchy
  - Whole and part objects are created, live, die together
  - Often also has a physical association
  - Association between instances
- Aggregation Hierarchy
  - Part objects of aggregate object may be created and deleted independently of aggregate object
  - Often used for more abstract whole/part relationships than composite objects
  - UML provides “light” semantic support for aggregation
    - Better to use composition in most cases
    - Aggregation can be modeled with basic associations

Example - Composition



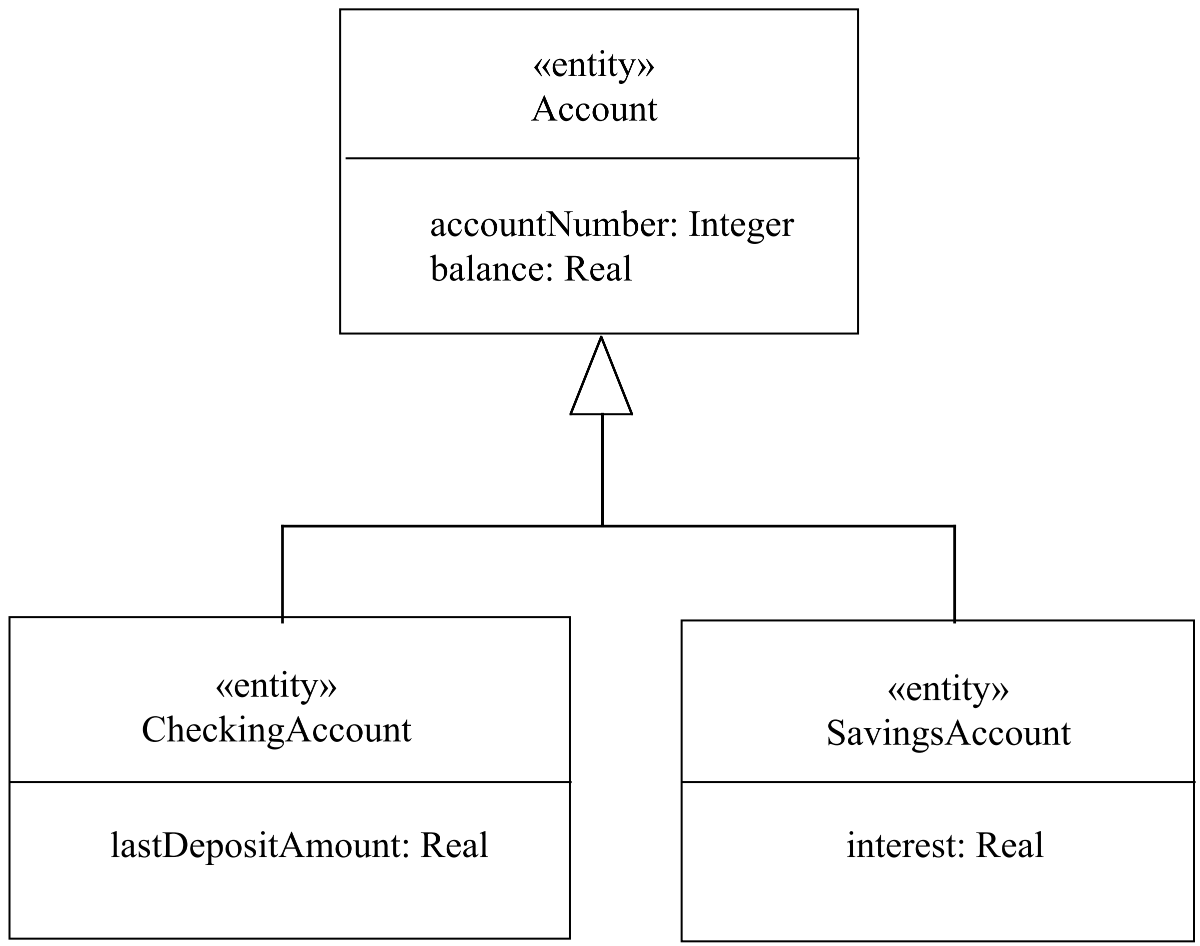
Example - Aggregation



Generalization / Specialization Hierarchy

- Some classes are similar but not identical
  - Have some attributes in common, others different
- Common attributes abstracted into generalized class (superclass)
  - E.g., Account (Account number, Balance)
- Different attributes are properties of specialized class (subclass)
  - E.g., Savings Account (Interest)
- Inheritance is best applied during design

Example - Generalization



Outline

- Building blocks of static modeling:
  - Class Diagrams
  - Relationships between classes
    - Associations
    - Composition / Aggregation
    - Generalization / Specialization
- Preliminary Class Design
  - System Context Class Diagram
  - Identifying candidate classes
  - Conceptual static model

Preliminary Class Design

- Static Model
  - Represents static structure of system
- Static Modeling
  - Analysis Modeling
    - Defines system context
    - Identifies problem-domain classes
    - Defines attributes of classes
    - Defines relationships between classes
  - Design Modeling
    - Defines operations of each class

Approach to Static Modeling

- Practitioners differ on how to apply static modeling
  - Model all classes
  - Only model entity (data) classes
  - Different modeling levels between analysis and design
- COMET Approach:
  - Conceptual static model early in analysis
    - Captures problem-domain entity classes and relationships
    - Deviation for this class:
      - Model all types of classes, not just entity classes
  - Context diagram
    - Identifies system context with respect to classes
  - Static model refined in "Class Design" to include solution-domain details

Preliminary Class Identification

- Determine all software objects in system
  - Use Object Structuring Criteria
  - Guidelines for identifying objects
  - Approach is iterative
    - Between analysis & design and static & dynamic models
- Structuring criteria depicted using stereotypes
- **Stereotype** used to further indicate types of classes
  - Can be thought of as a ***behavioral pattern*** for the class
    - «entity», «control», etc.
- Identify classes from:
  - Walking through use case specifications
  - Interface requirements
  - Persistent data requirements
  - Dynamic models

Application Class Stereotypes

- «boundary»
  - Provides the interface for external interactions
  - 1:1 correlation with «external» classes
- «entity»
  - Manages persistent data
- «control»
  - Central controller for one or more use case scenarios
- «timer»
  - Special purpose controller specifically used for time-triggered control
- «algorithm»
  - Encapsulates application specific algorithms

Analysis Classes: Level of Detail

- Each class represents a ***problem domain*** abstraction
  - Maps to real-world concept for your application
- Each class should contain:
  - Name
  - Stereotype
  - Attributes
    - Attribute name is mandatory
    - Attribute type and visibility are optional
  - Operations
    - Optional during object-structuring
    - Only high-level at this stage
      - Indicate main responsibilities of the class

Quality Factors of Analysis Classes

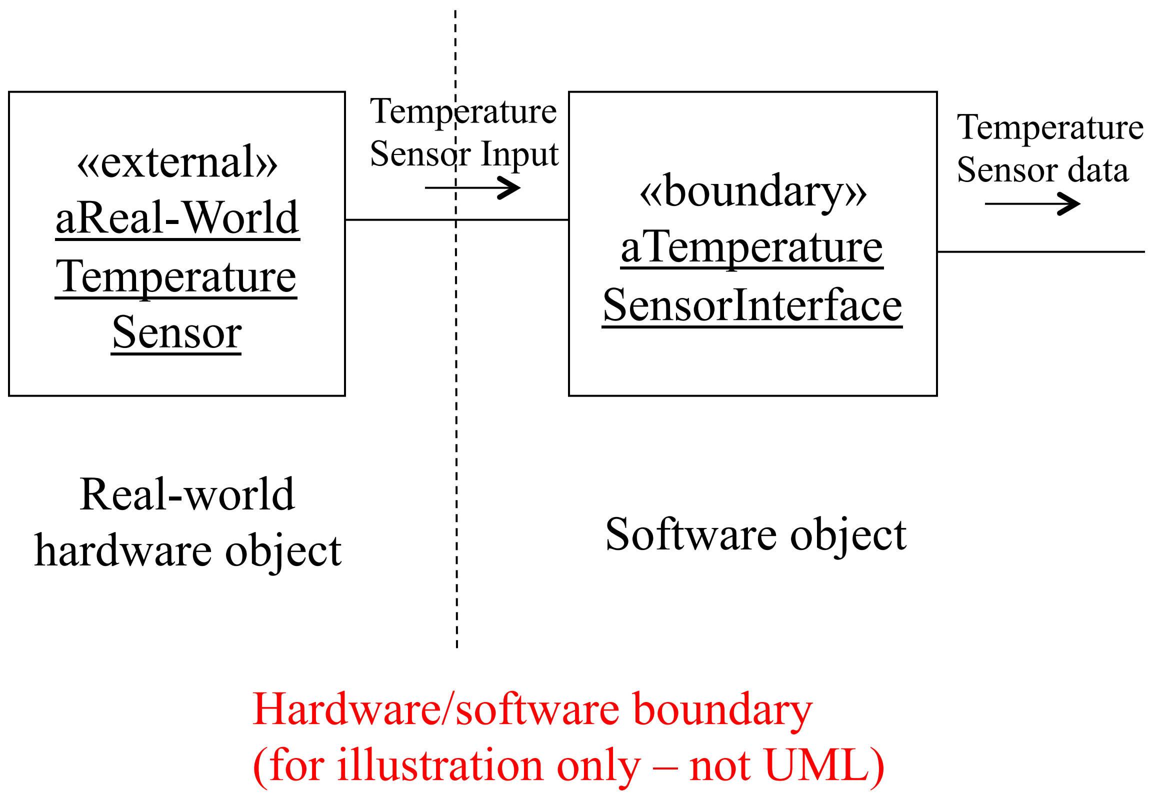
- The makings of a good analysis class:
  - Name indicates its purpose
  - Represents problem-domain abstraction
    - Recognizable by stakeholders
  - Single-purpose
    - Cohesive grouping of data and functions
    - Relatively small and focused set of responsibilities
  - Loosely coupled
    - Minimized dependencies on other classes

General Guidelines

- Keep it simple
  - 3-5 responsibilities per class
- No isolated classes
- Beware of classes that should really be attributes
  - Question models with many very small classes
- Beware of omnipotent classes
  - Question models with very few, but very large classes
- Beware of structured (functional) analysis disguised as OO
- Avoid overuse of inheritance

Identifying «boundary» Classes

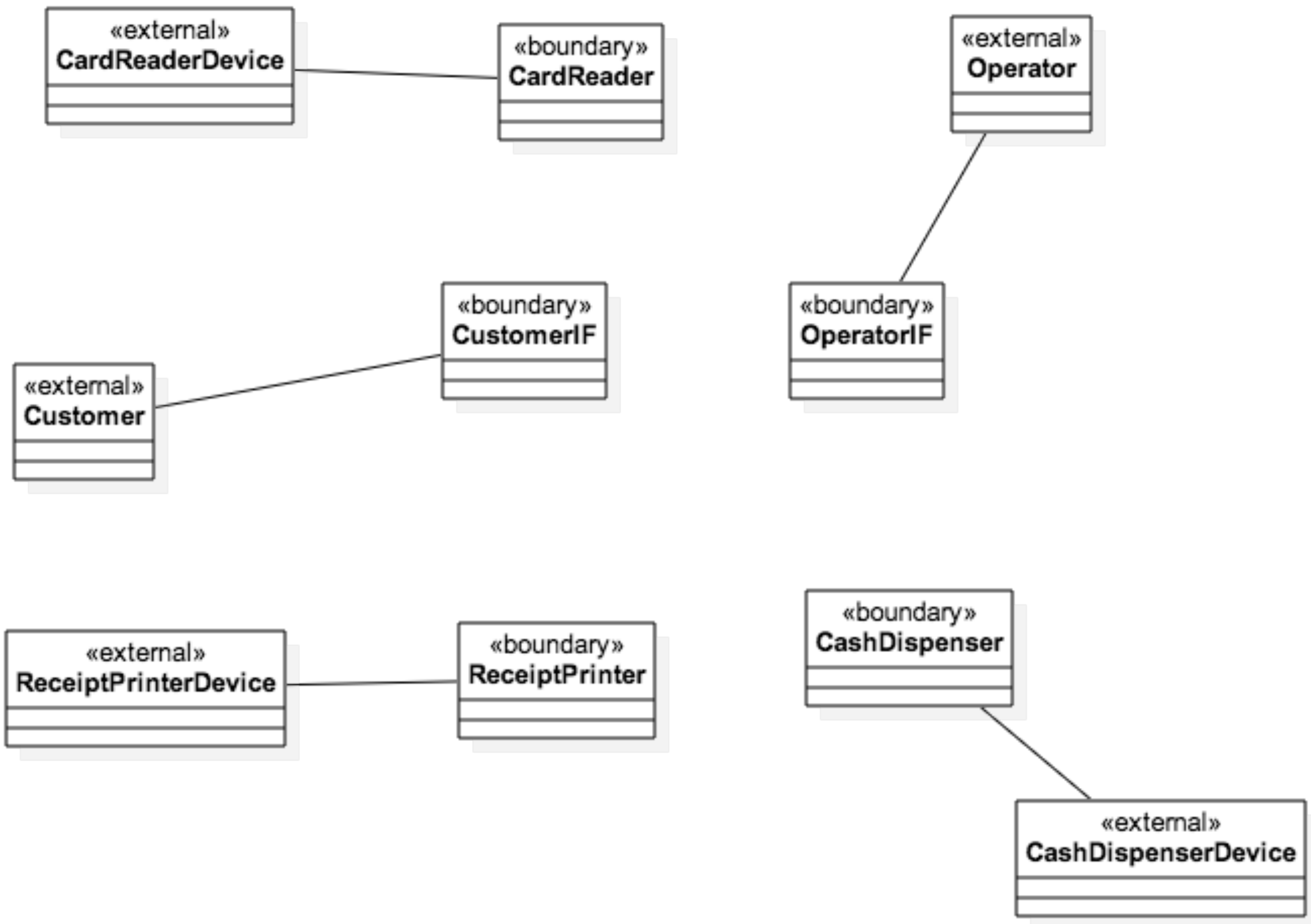
- One-to-one mapping with «external» objects



System Context Class Diagram

- Defines boundary between system and external environment
  - May be depicted on System Context Class Diagram
- External classes
  - External entities that the <u>software</u> system interfaces to
    - Devices, users, systems, timers, etc.
  - Not implemented by the software system
    - *Abstract* classes used to define the system boundaries
  - Stereotyped as «external»

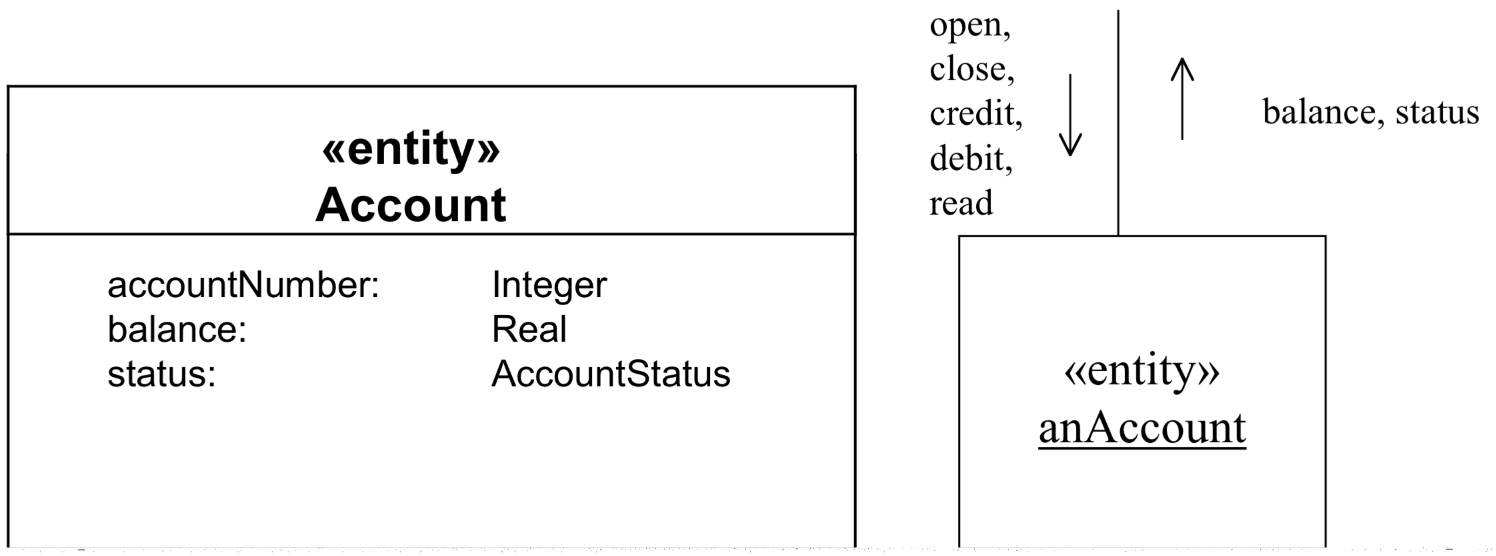
Example - Context Diagram



Identifying Entity Classes

- Entity classes
  - Long lasting objects that store information
    - Same object typically accessed by many use cases
    - Information persists over access by several use cases
      - E.g., Account, Customer
  - Entity classes and relationships shown on static model
  - Entity classes often mapped to relational database during design
  - Examples: Figs. 9.9 – 9.10

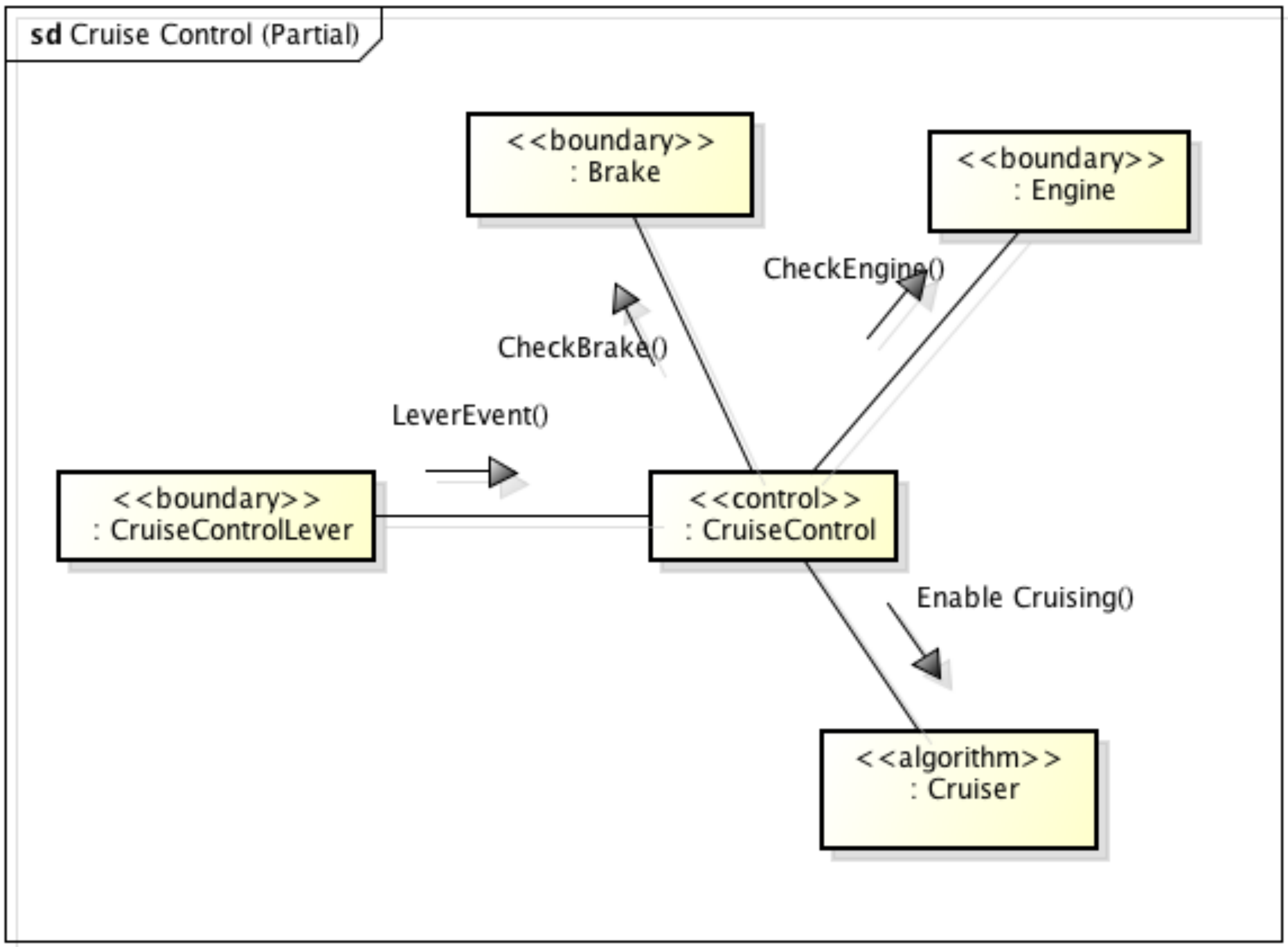
Example – Entity Class



Identifying Control Classes

- Control classes
  - Provide overall coordination for execution of use case
  - Glue that unites other objects that participate in use case
  - Makes overall decisions
  - Decides when, and in what order, other objects participate in use case.
    - Entity objects
    - boundary objects
  - Simple use cases do not need control classes
  - More complex use case usually has at least one control class
  - Often discovered during dynamic modeling

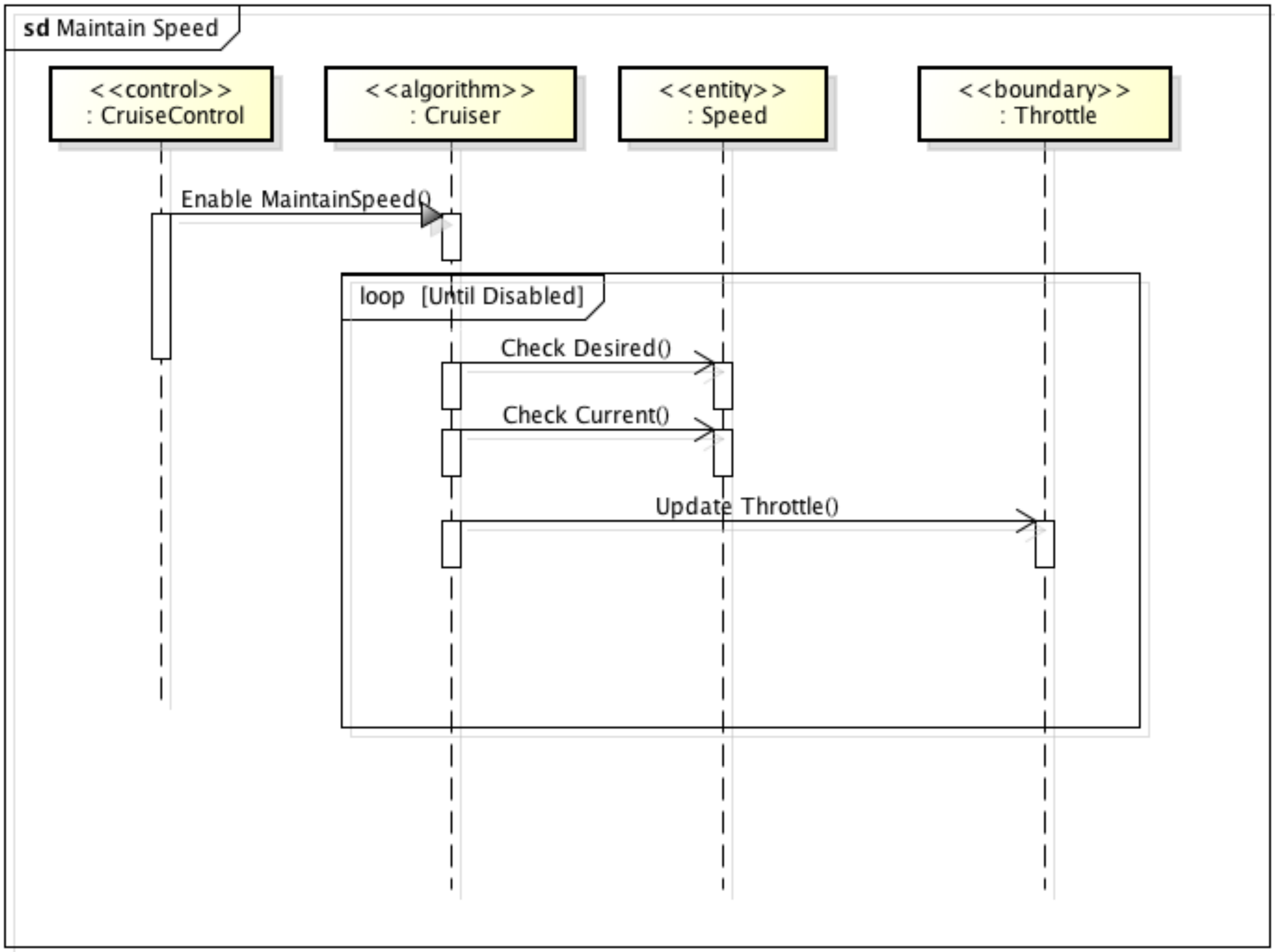
Example – Control Objects



Identifying Algorithm Classes

- «algorithm»
  - Encapsulates algorithm used in problem domain
  - Application specific logic
  - Used when you want to capture a specific algorithm or piece of business logic that needs to be maintained as its own class
    - For maintainability
    - To be able to easily change algorithms in the application
  - Example: Fig. 9.14

Example – Algorithm Object



Identifying Classes - Recap

- Analysis
  - Classes should directly relate to problem domain
- Design
  - Software objects added to address solution domain
- Identify objects from…
  - Use case model
  - Requirements specification
  - Dynamic model
- Classes created on a UML class diagram
- Static model captures structural relationships between classes and class instances (objects)

Summary: Analysis Level Static Modeling

- During Analysis Modeling
  - Conceptual static model
  - Emphasizes real-world classes in the problem domain
  - Does not specifically address software solution classes
  - Preliminary set of problem-domain classes and their structural relationships
  - In COMET, the use of inheritance is minimized until later in the design process
  - Initial static model will continue to evolve through the design lifecycle