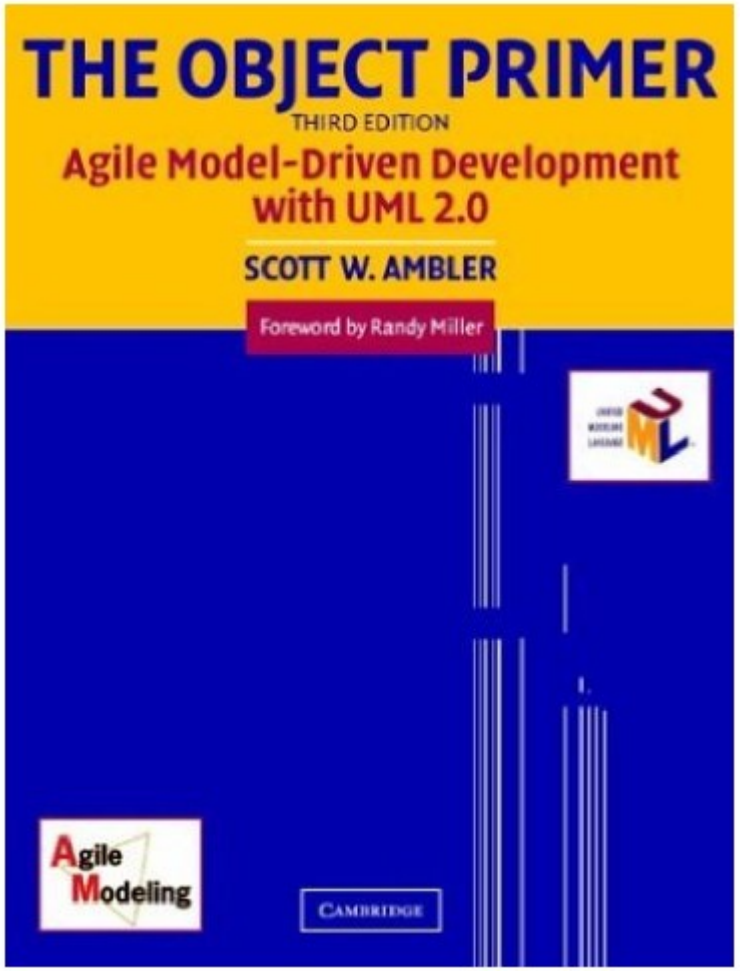
UML: introduction

Unified Modeling Language (UML) makes it possible to describe systems with words and pictures.

UML was standardized by **Object Management Group** (OMG) — http://www.omg.org, an international association that promotes open standards for object-oriented applications.

Modelling and the UML



Scott W. Ambler, The Object Primer. Agile Model-Driven Development with UML 2.0, Cambridge University

Overview of Object-Oriented Concepts

Term	Description
Abstract class	A class that does not have objects instantiated from it
Abstraction	The essential characteristics of an item (perhaps a class or operation)
Aggregation	Relationships between two classes or components defined as «is part of»
Aggregation hierarchy	A set of classes related through aggregation
Association	A relationship between two classes or objects
Attribute	Something a class knows (data/information)
Cardinality	The concept of «how many?»
Class	A software abstraction of similar objects, a template from which objects are created
Classifier	A UML term that refers to a collection of instances that have something in common That includes classes, components, data types, and use cases.
Cohesion	The degree of relatedness of an encapsulated unit (such as a component or a class)
Collaboration	Classes work together (collaborate) to fulfill their responsibilities
...	...

Scott W. Ambler, The Object Primer. Agile Model-Driven Development with UML 2.0, Cambridge University

4 basic Object-Oriented concepts

Term	Description
Class	A class is a software abstraction of an object, effectively, a template from which objects are created
Object	An object is a software construct that mirrors a concept in the real world, e.g. a person, place, thing.
Attribute	An attribute is equivalent to a data element in a record.
Method	A method can be thought of as either a function or procedure. Methods access and modify the attributes of an object.

The Diagrams of UML 2

- **Behavior diagrams.** This is a type of diagram that depicts behavioral features of a system or business process. This include *activity, state machine, use case, and interaction diagrams.*
- **Interaction diagrams.** This is a subset of behavior diagrams that emphasize object interactions. This includes *communication, interaction overview, sequence, and timing diagrams*.
- **Structure diagrams.** This is a type of diagram that depicts the static elements of a specification that are irrespective of time. This includes *class, composite structure, component, deployment, object, and package diagrams.*

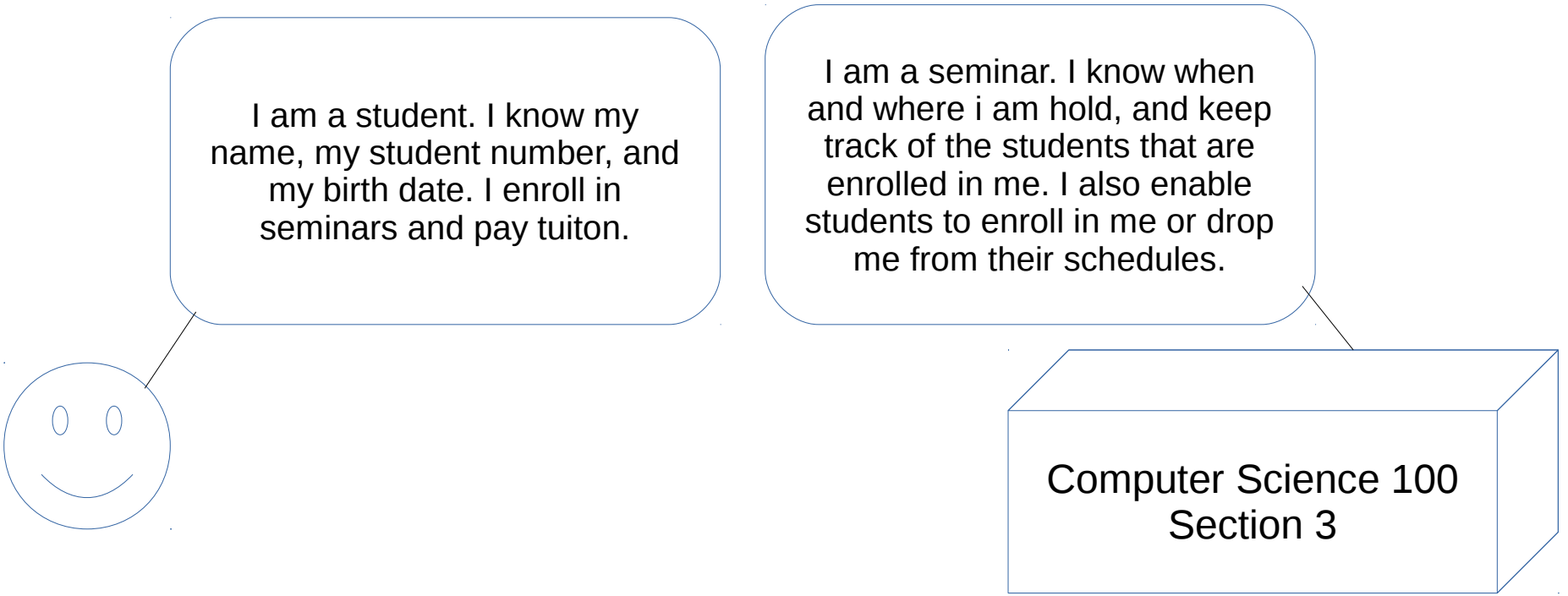
The diagrams of UML 2

Diagram	Description
Activity diagram	Depicts high-level business processes, including data flow, or to model the complex logic within a system.
Class diagram	Shows a collection of static model elements such as classes and types, their contents, and their relationships.
Communication diagram	Shows instances of classes, their interrelationships, and the message flow between them, and typically focuses on the structural organization of objects that send and receive messages.
Component diagram	Depicts the components, including their interrelationships, interactions, and public interfaces, that compose an application, system, or enterprise.
Composite structure diagram	Depicts internal structure of a classifier (such as a class, component, or use case), including the interaction points of the classifier to other parts of the system.
Deployment diagram	Shows the execution architecture of systems, including nodes, either hardware or software execution environments, and the middleware connecting them.
Interaction overview diagram	A variant of an activity diagram, which overview the control flow within a system or business process, whereby each node/activity within the diagram can represent another interaction diagram.
Object diagram	Depicts objects and their relationships at a point in time, typically a special case of either a class diagram or a communication diagram.

The diagrams of UML 2

Diagram	Description
Package diagram	Shows how model elements are organized into packages as well as the dependencies between packages
Sequence diagram	Models sequential logic, in effect th time ordering of messages between classifiers
State machine diagram	Describes the states an object or interaction may be in, as well as the transitions between states; formerly referred to as a state diagram, state chart diagram, or a state-transition diagram
Timing diagram	Depicts the change in state or condition of a classifier instance or role over time, and typically used to show the change in state of an object over time in response to external events.
Use Case Diagram	Shows use cases, actors, and their relationsips.

Attributes and Operations/Methods
Objects in «real world»



Attributes and Operations/Methods
Student and Seminar classes

Student
name phoneNumber studentNumber nextStudentNumber
findByName enrollInSeminar dropSeminar payTution requestTranscript

Seminar
instructors location listOfStudents
addStudent removeStudent

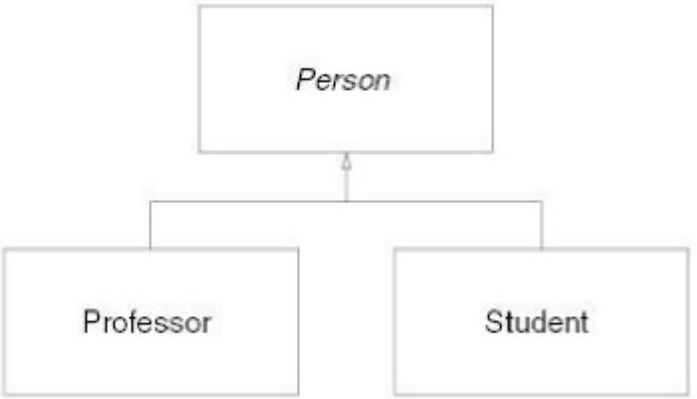
Abstraction, Encapsulation, And information Hiding

- Abstraction — determination of what a class knows and does.
- Encapsulation — hiding details of the implementation.
- Information Hiding — restricting access to attributes with adding of set get methods.

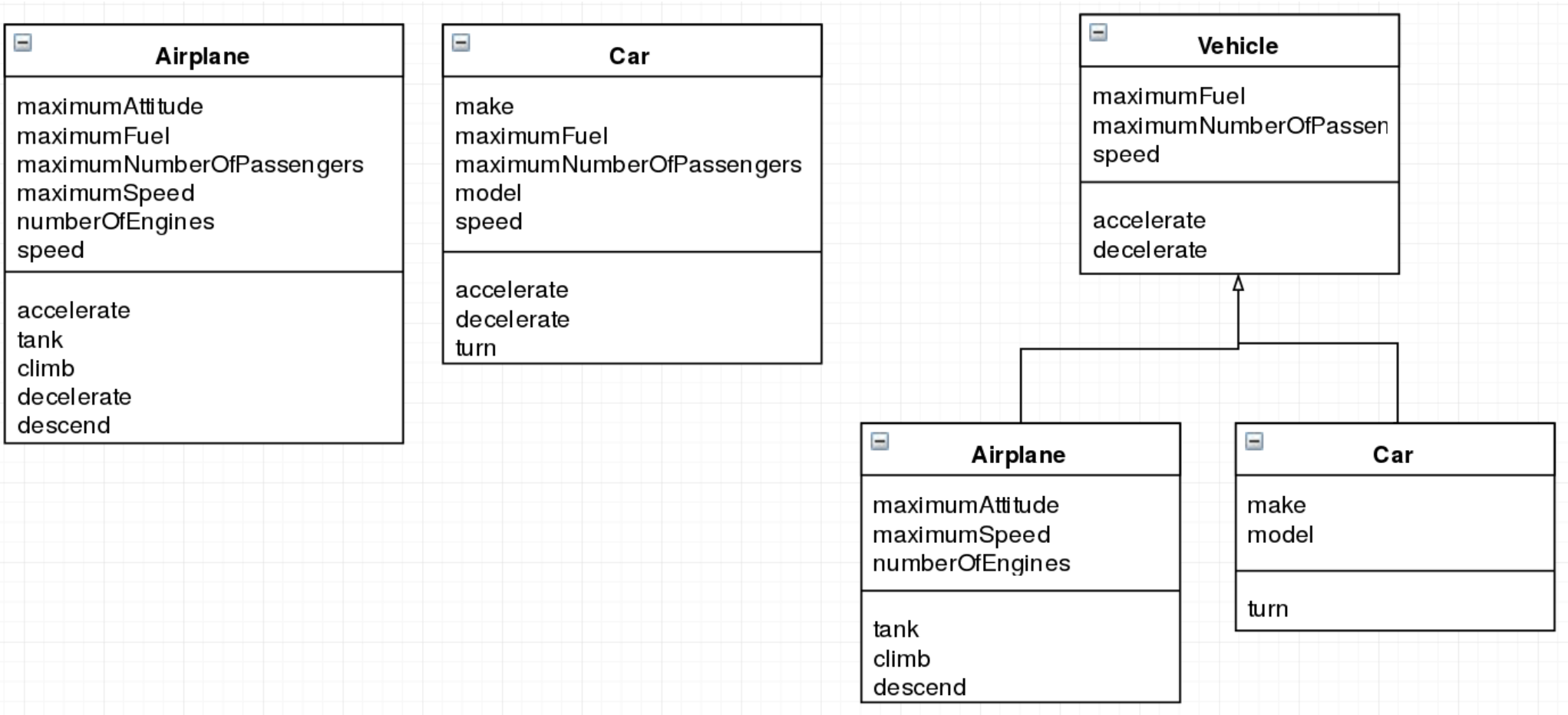
Inheritance

Tips to apply inheritance effectively

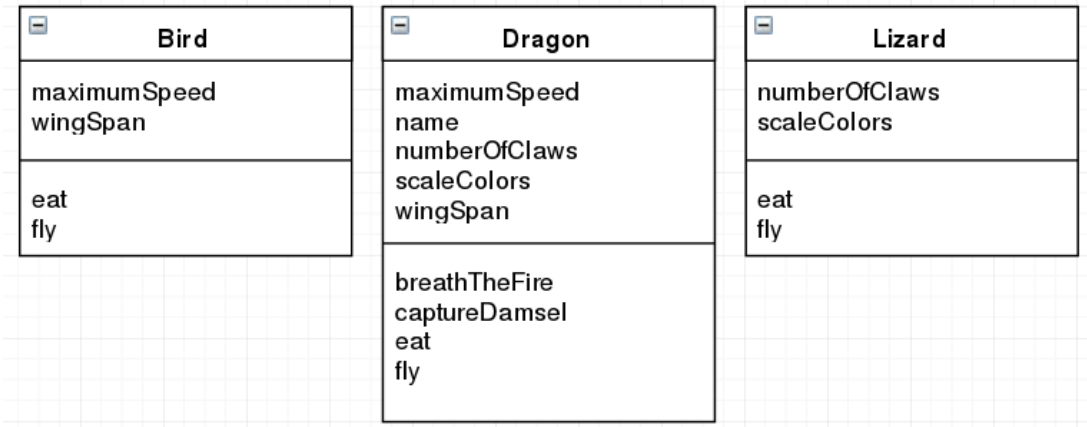
- Look for similarities (in classes, attributes, methods)
- Look for existing classes (when you identify a new class)
- Follow the sentence rule (a subclass is a kind of superclass)
- Avoid implementation inheritance
- Inherit everything

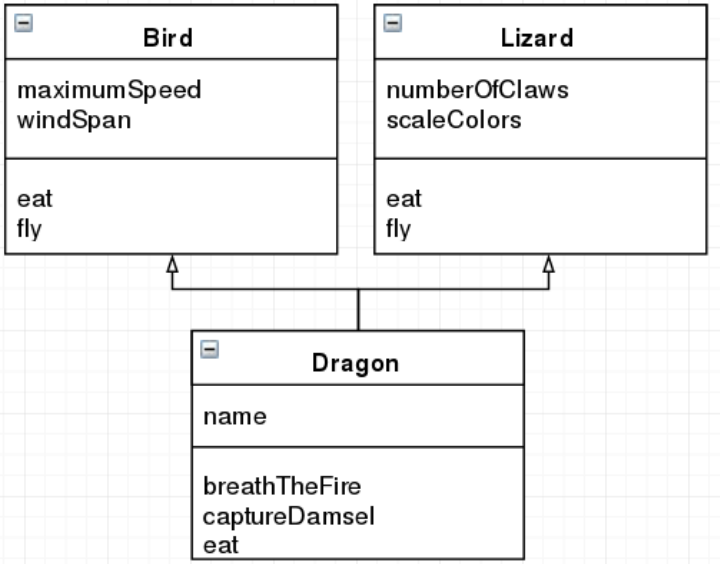


Inheritance



Inheritance





Requirements Artifacts

Artifact	Description
Business rules	A business rule defines or constrains one aspect of your business that is intended to assert business structure or influence the behavior of your business.
Constraints	A constraint is a restriction on the degree of freedom you have in providing a solution. Constraints will supplement other development artifacts, in particular architecture an design-oriented models.
Glossary	A glossary is a collection of definitions that supplements a wide range of development artifacts by defining a common business and technical vocabulary.
Technical requirements	A technical requirement pertrains to the technical aspects that your system must fulfill, such as performance-related, reliability, and availability issues.

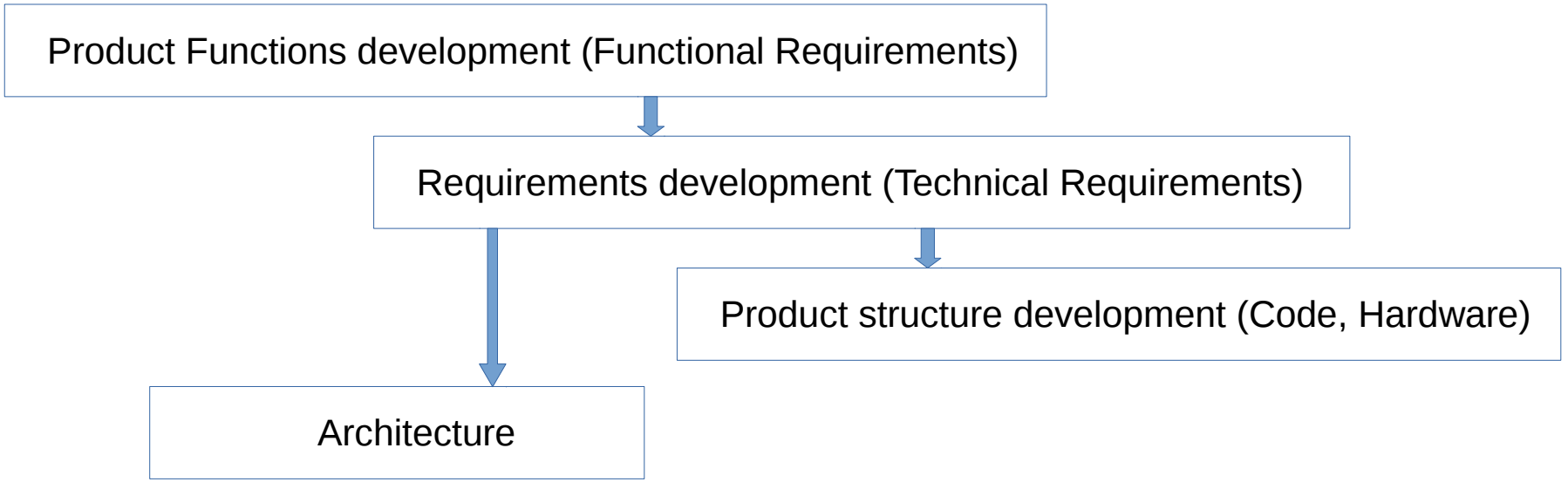
Business Rules

- BR123 Tenured professors may administer student grades
- BR124 Teaching assistants who have been granted authority by a tenured professor may administer student grades
- BR177 Table to convert between numeric grades and letter grades (f.e. «A-» to «95%»)
- BR245 All master's degree programs must include the development of a thesis

Technical Requirements

Two approaches:

- Requiremenst first — then product development.
- Product development first — then requirements (reverse engineering, Agile, project management mistake?).



Technical Requirements Organization

General Technical Requirements

- General System Technical Requirements
  - General Sub-System Technical Requirements
    - Component_1 Technical Requirements
    - Component_2 Technical Requirements
    - ...

HCPP (Hiérarchisation des caractéristiques Produit/Process)

The basic idea of HCPP is to rank requirements (because thousands of them) for further interaction with hardware suppliers (OEM).

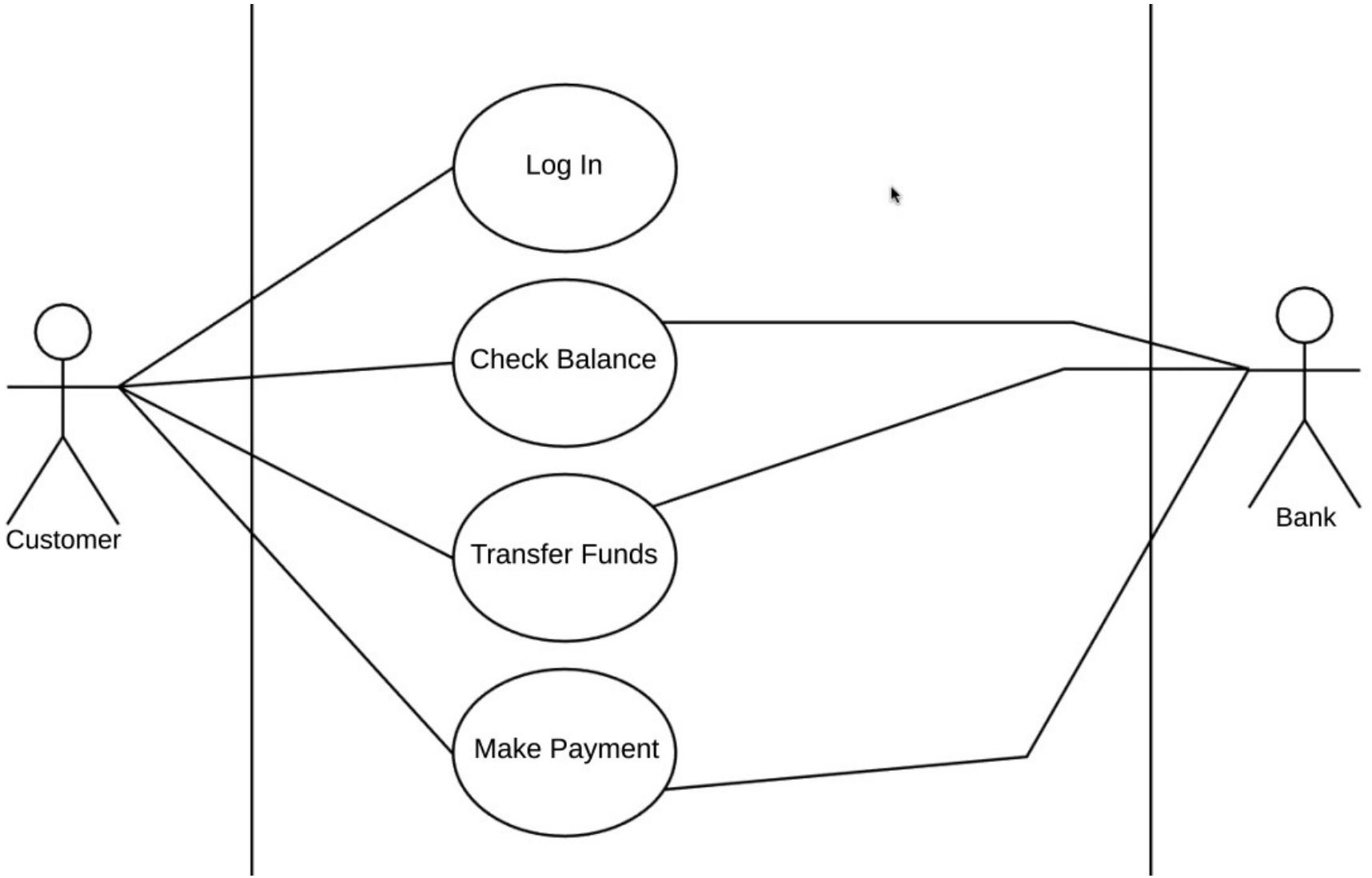
Basic principles:

- Each Technical Requirement has (or has not) Importance Rank: (Important for Safety, Important for Operability, Important for Customer …)
- Determine from each requirement, numerical values and admissible deviations of this values — Key Characteristics

Technical Requirement + Rank + Key Characteristics = **HCPP Map**

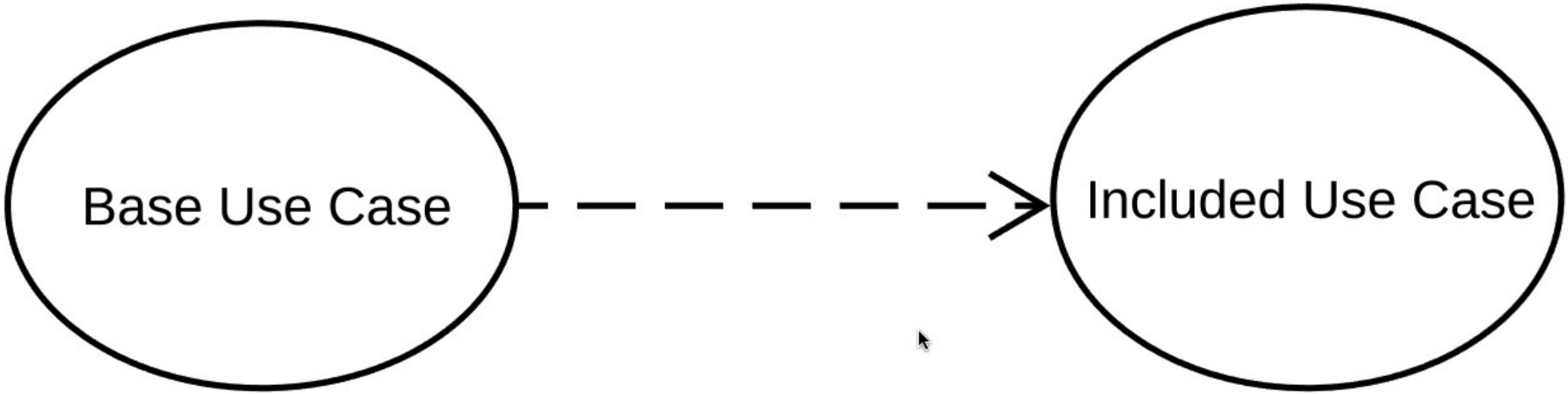
HCPP map used during Supplier Nomination process to determine: requirements that supplier can not fulfill — important for Product or not?

Use Case Diagram Example: Banking App



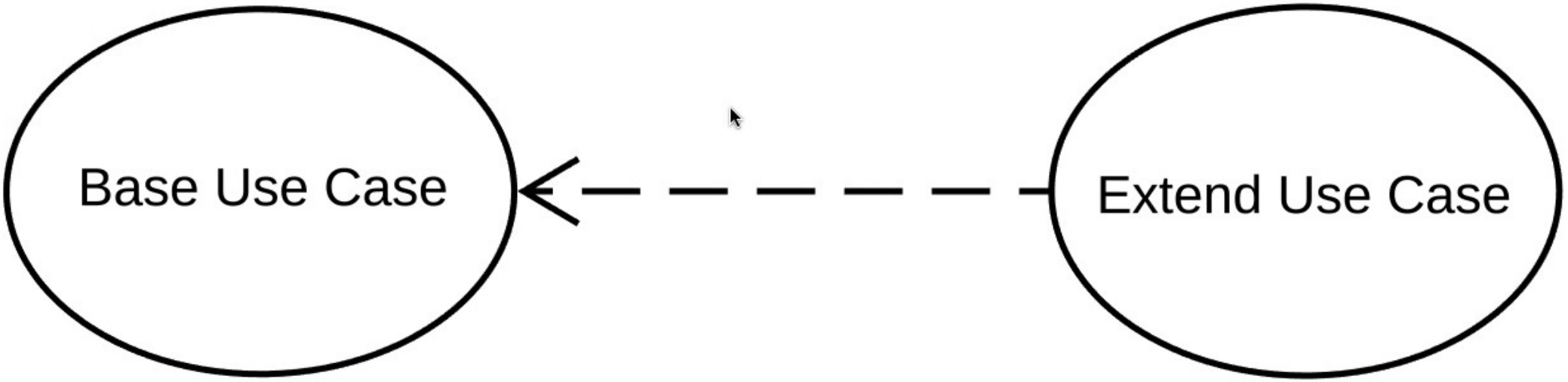
UML Use Case Diagram Tutorial by Lucidchart https://www.youtube.com/watch?v=zid-MVo7M-E

Use Case Diagram: **Include** relation



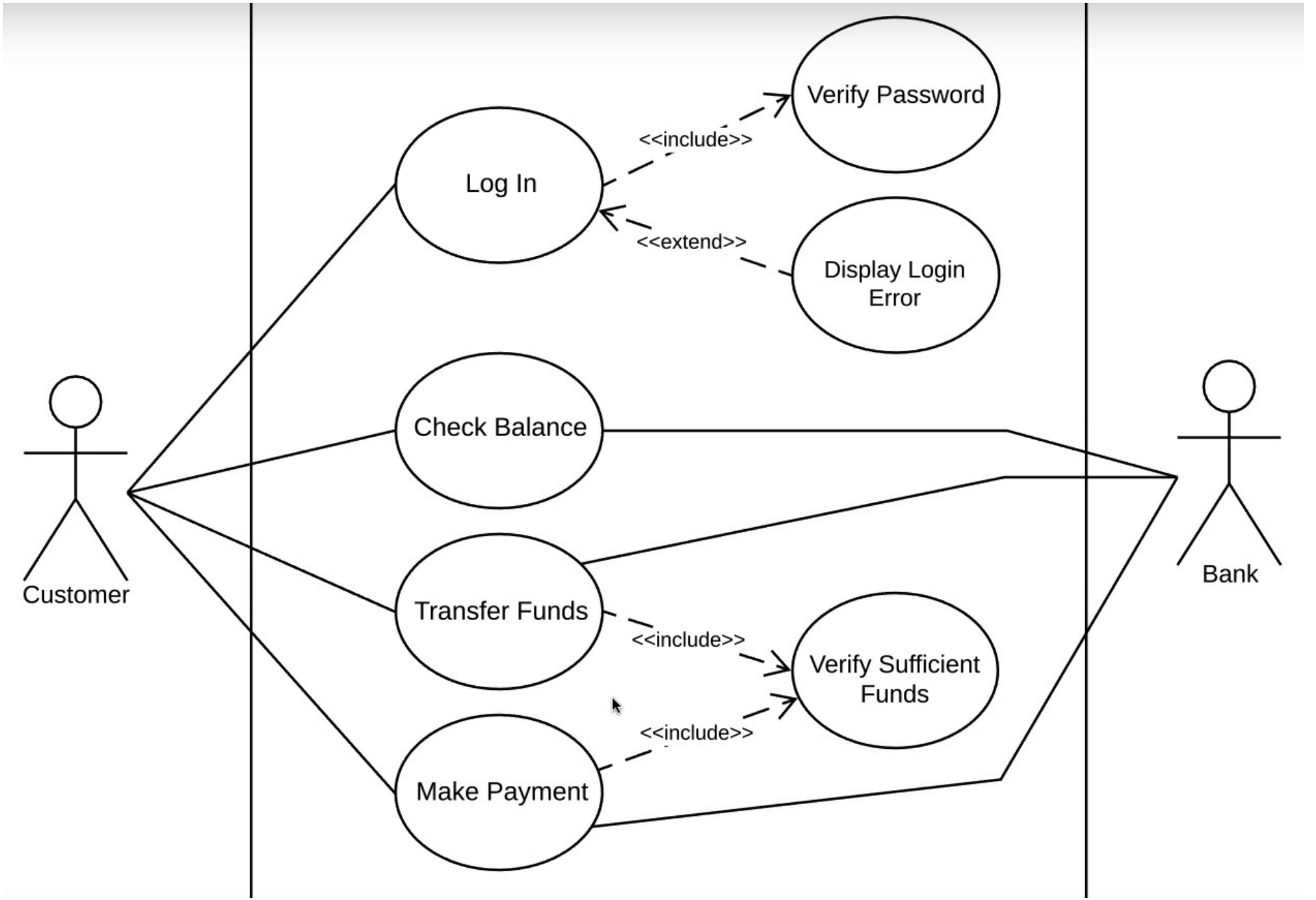
- If Base Use Case executed then Included Use Case executed
- Base Use Case requires Included Use Case to be complete

Use Case Diagram: **Extend** relation

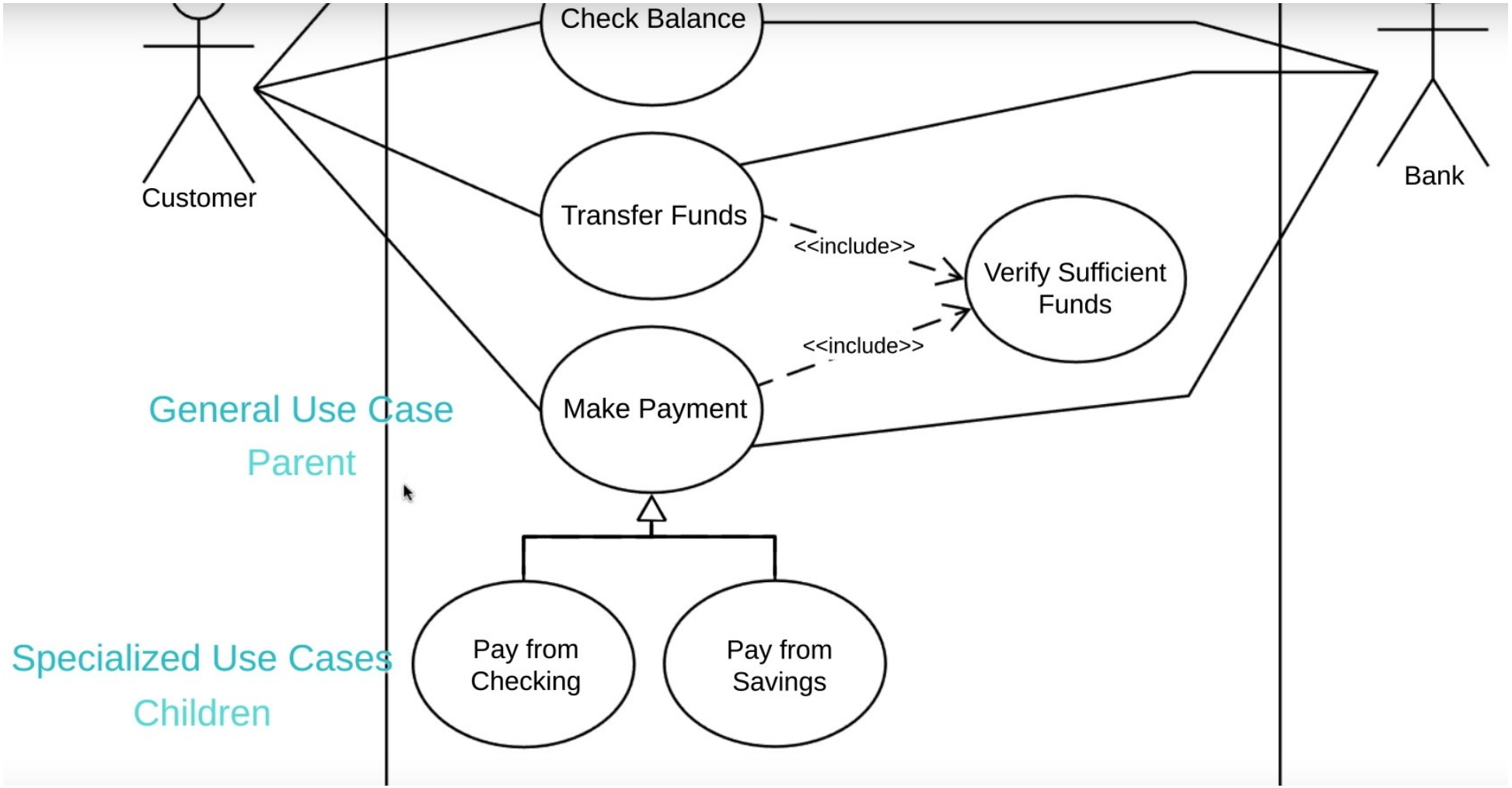


- If Base Use Case executed then Exten Use Case executed sometimes, but not everytime

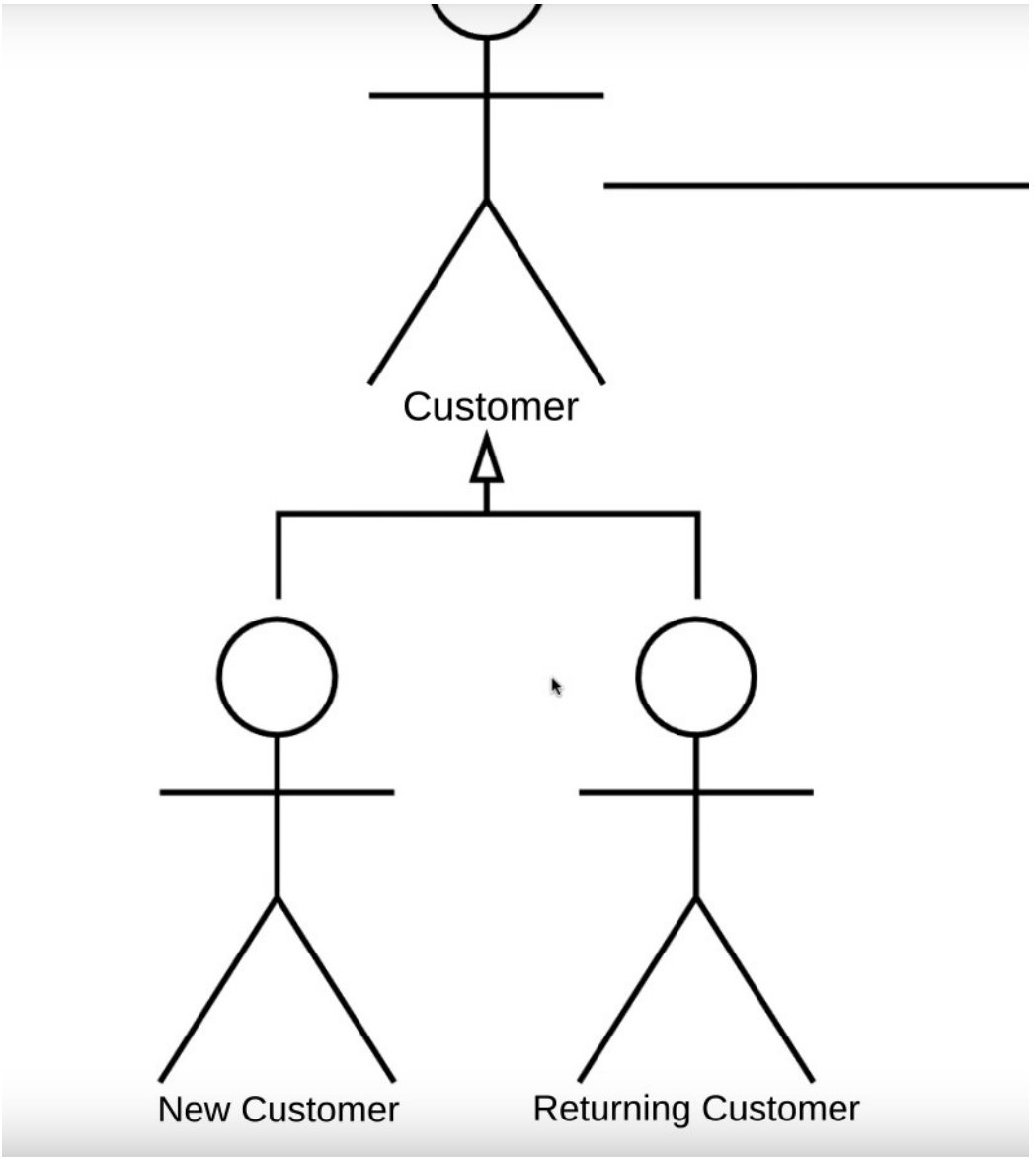
Use Case Diagram Example: Banking App



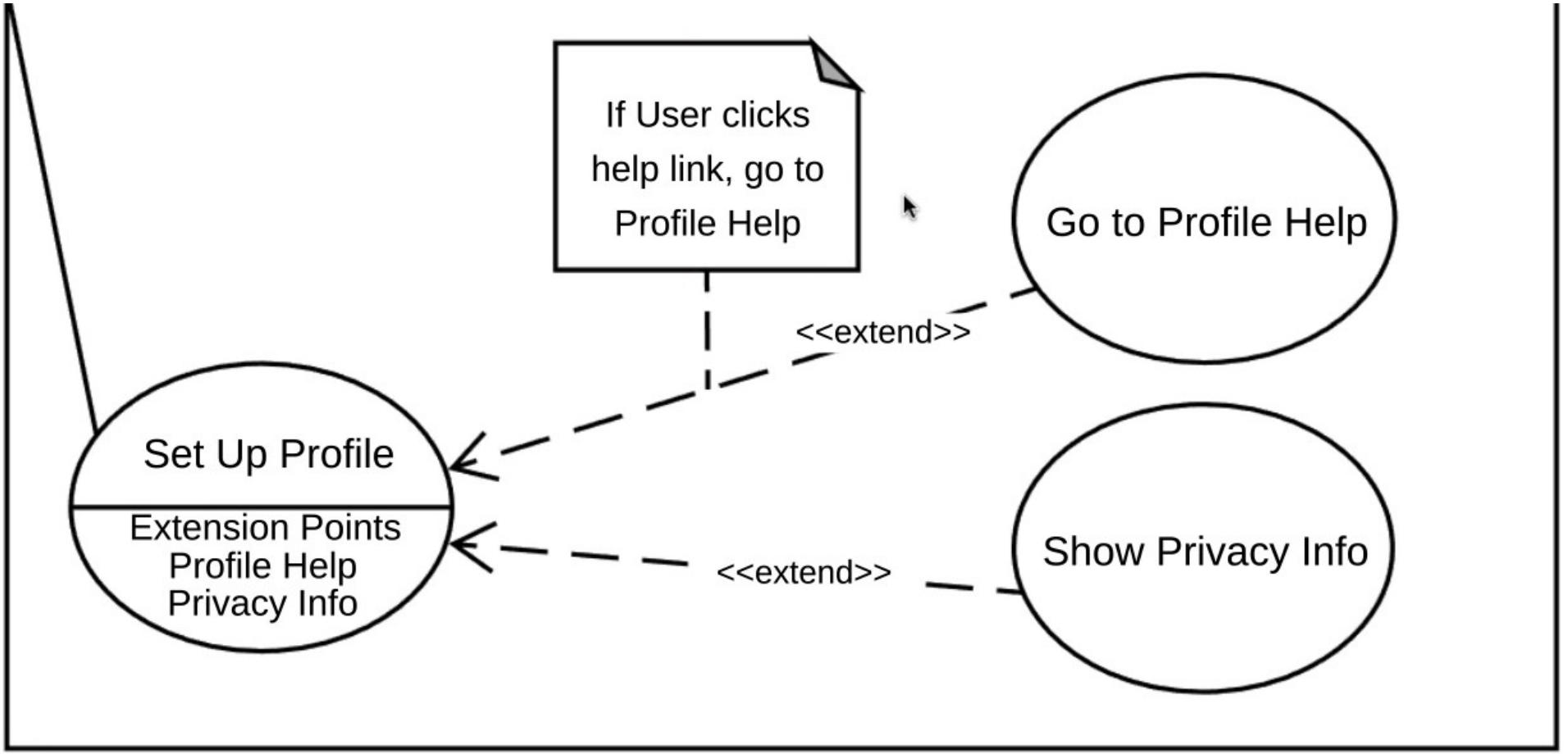
Use Case Diagram: **Generalizaion** relation



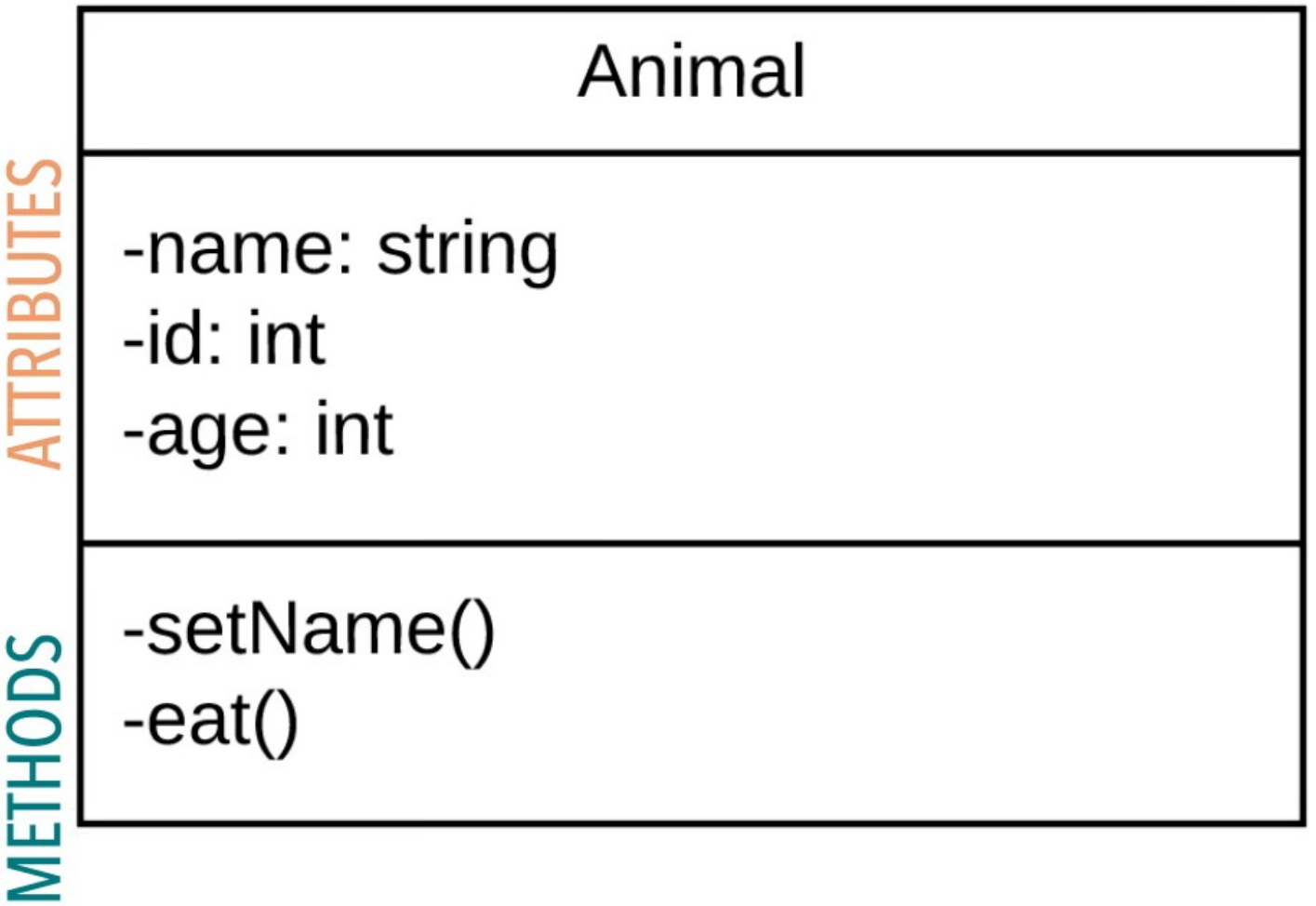
Use Case Diagram: **Generalizaion** relation



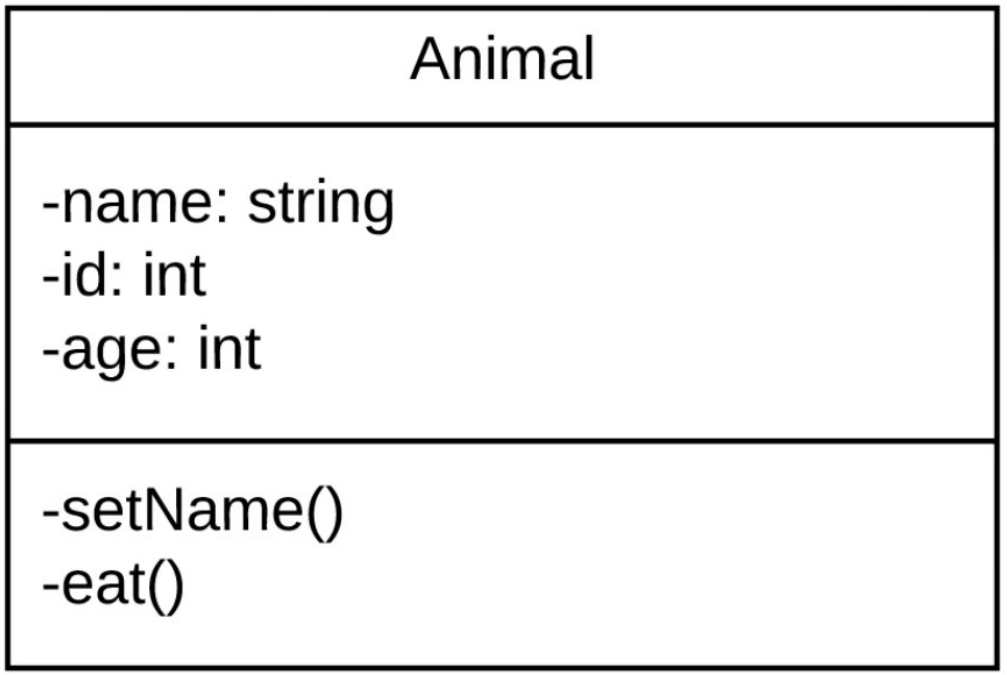
Use Case Diagram: **Extension** relation



Class Diagram: Class block



Class Diagram: Class block



Visibility

- \- private
- \+ public
- \# protected
- ~ package/default

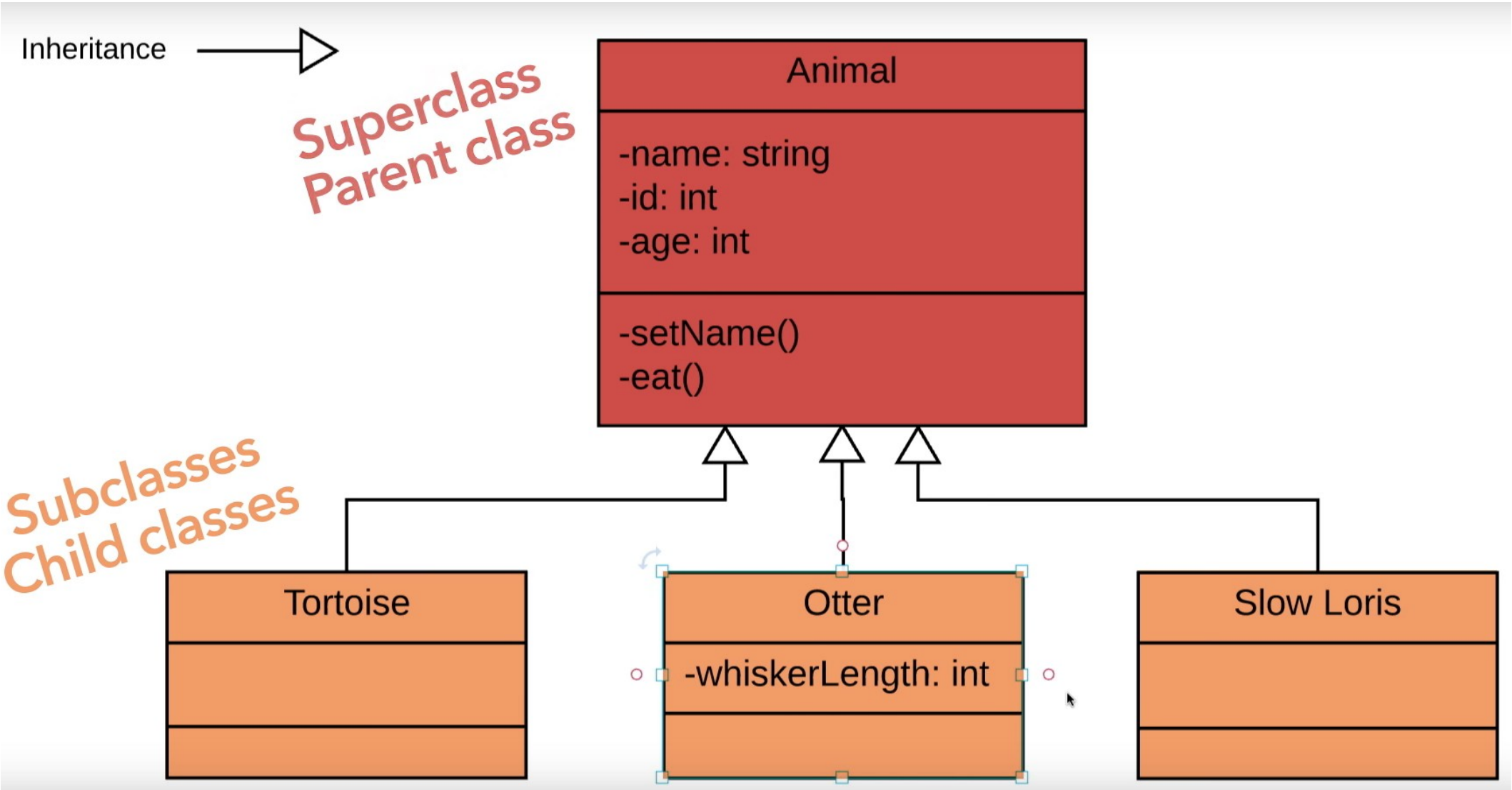
private — can note be accessed by any class or sub-class

public — can be accessed by any class or sub-class

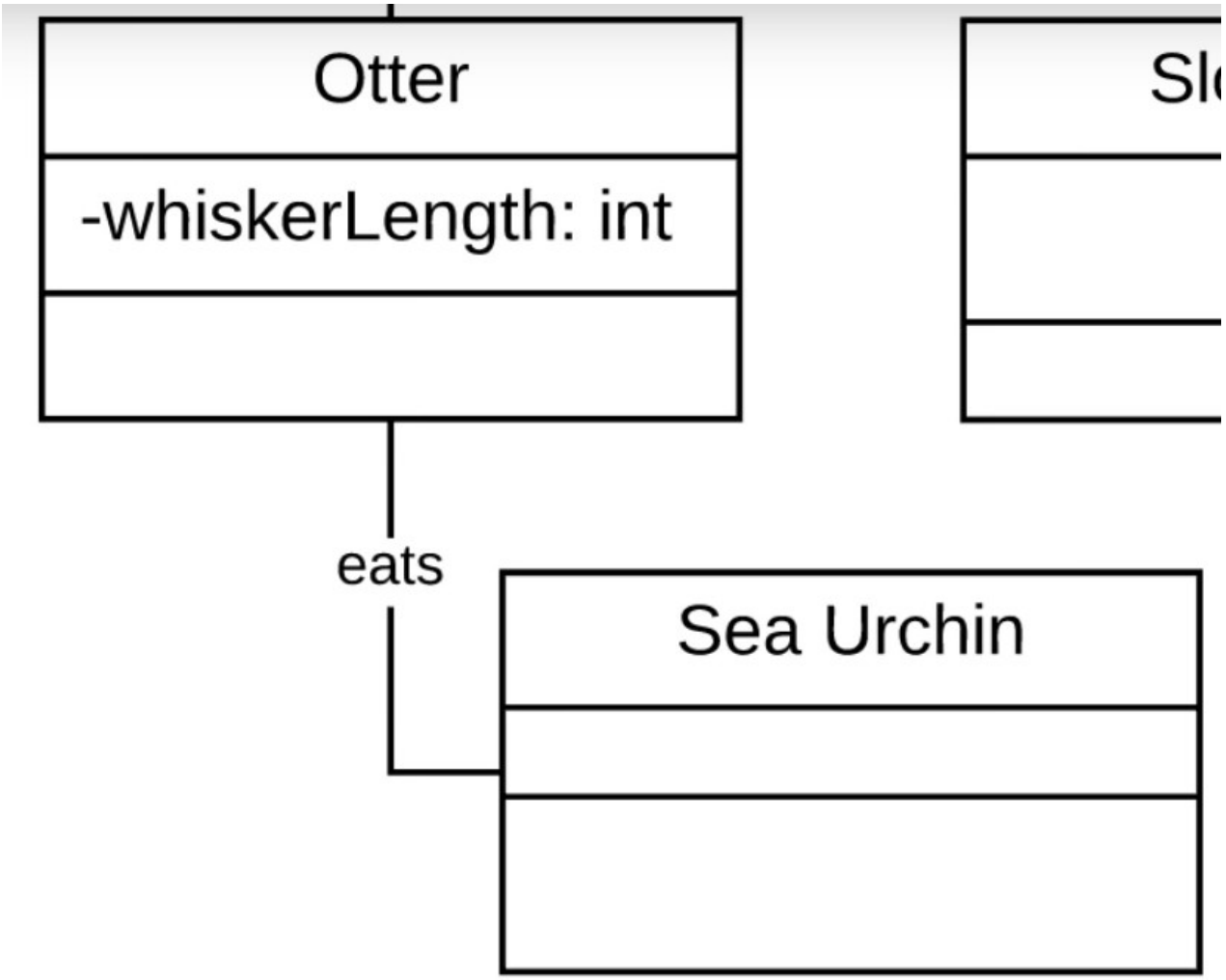
protected — can only be accessed by same class or sub-classes

package/default — can be used by any other class in the same package

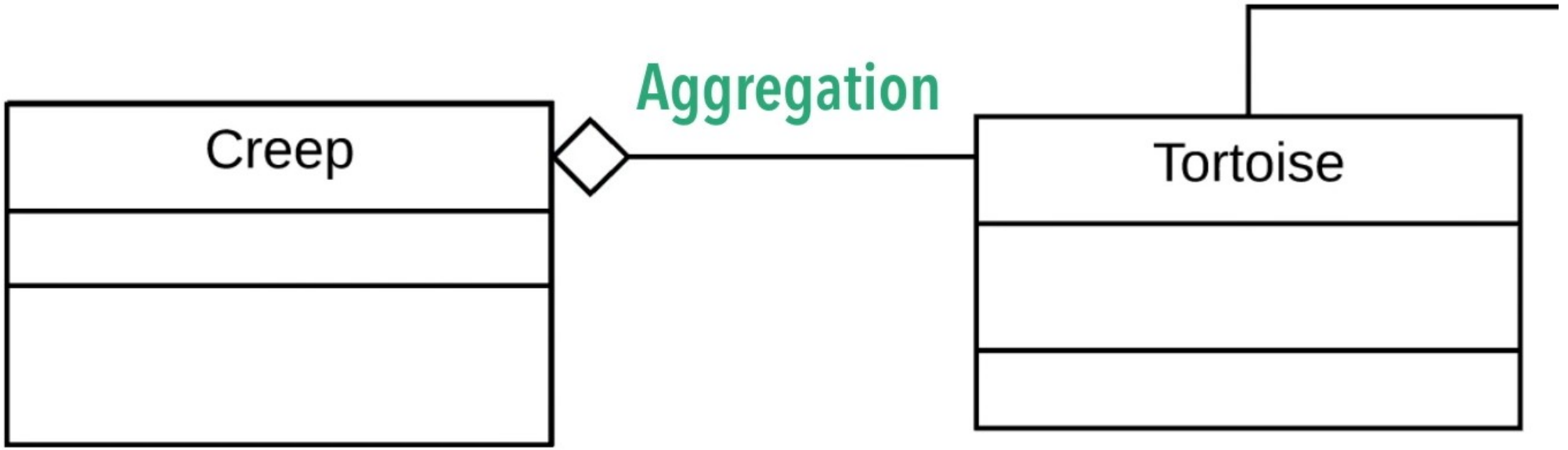
Class Diagram: **Inheritance** relation



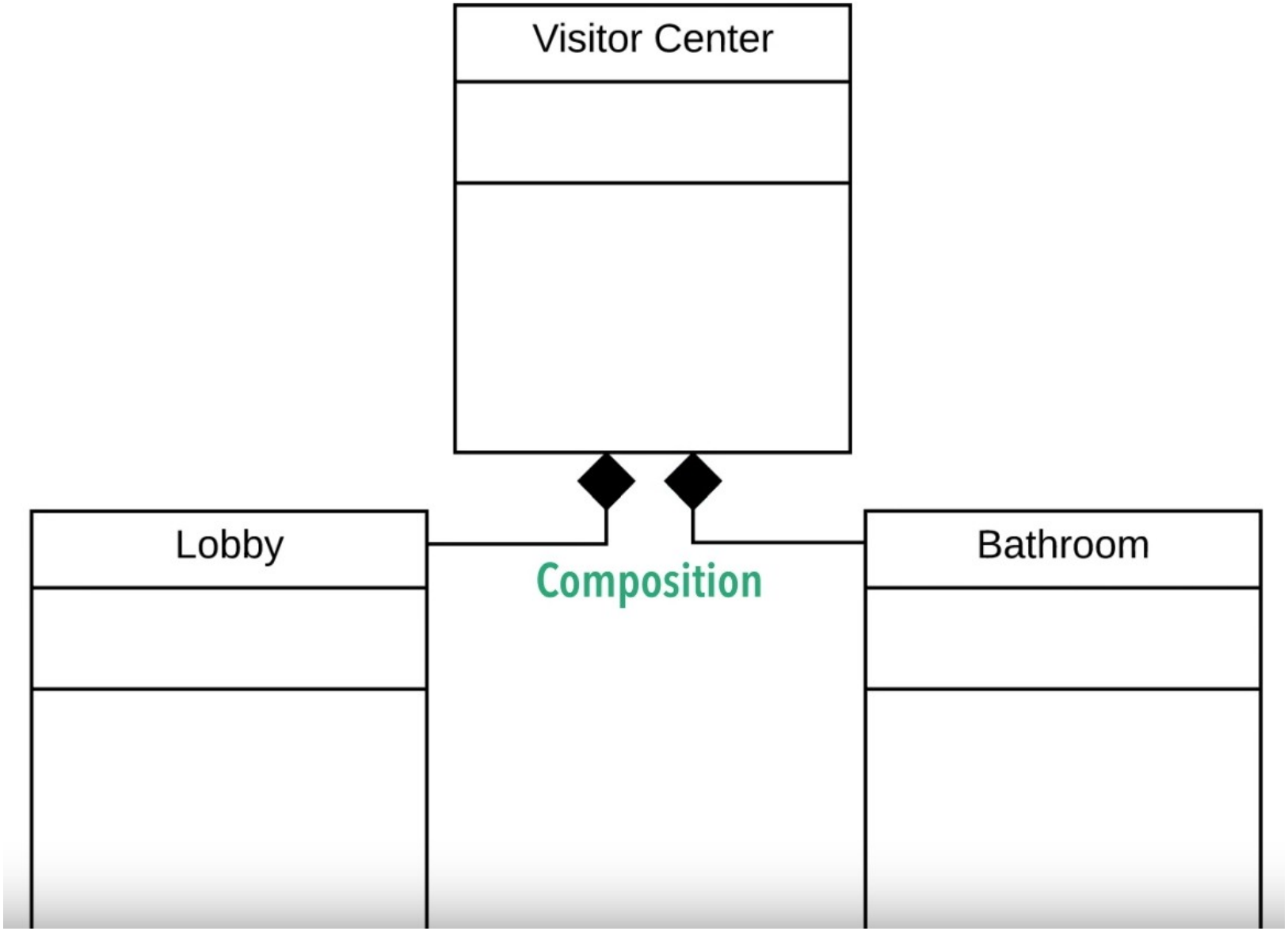
Class Diagram: **Association** relation



Class Diagram: **Aggregation** relation



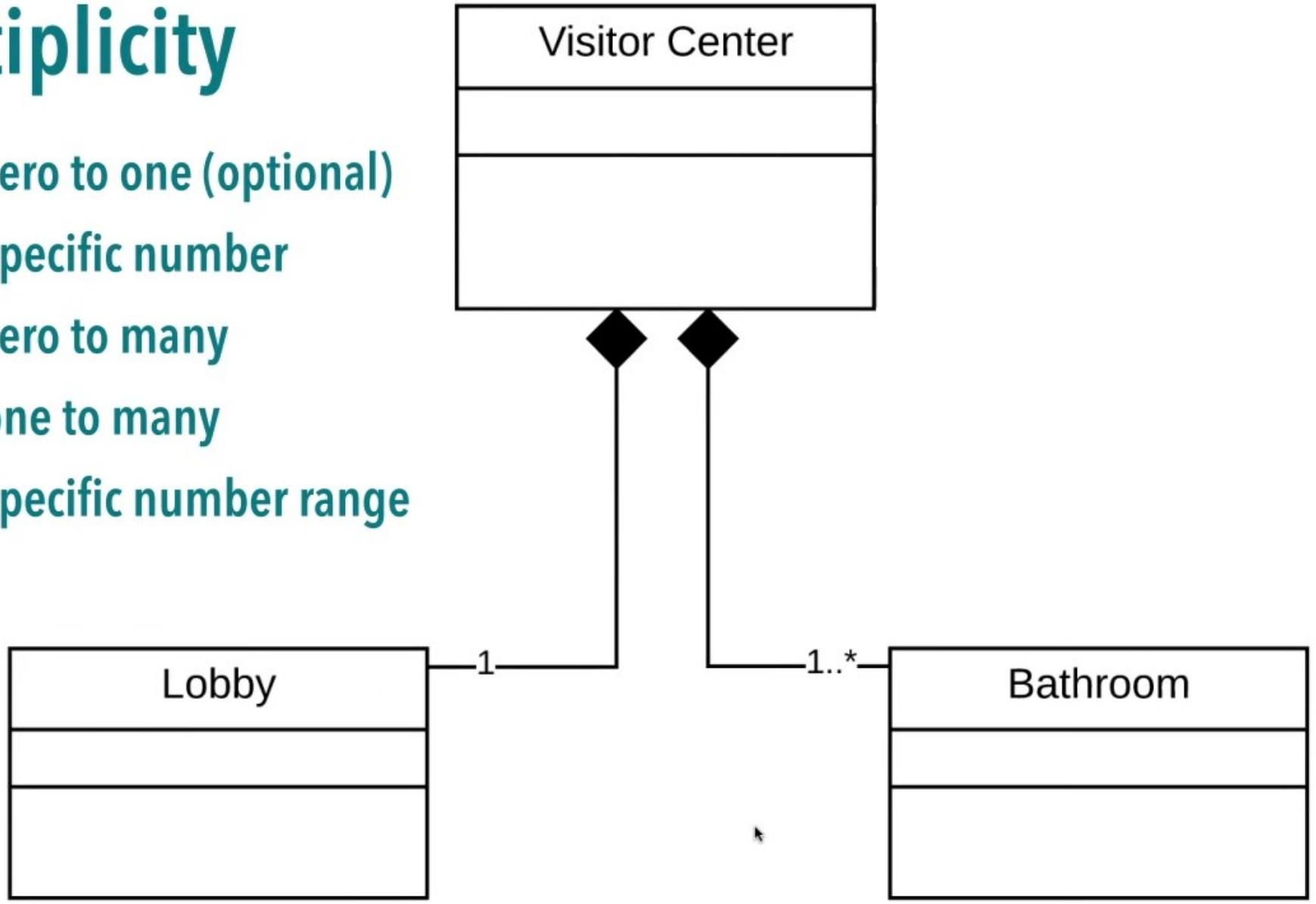
Class Diagram: **Composition** relation



Class Diagram: **Multiplicity**

Multiplicity

0..1	zero to one (optional)
n	specific number
0..*	zero to many
1..*	one to many
m..n	specific number range



Class Diagram Example: **Market**