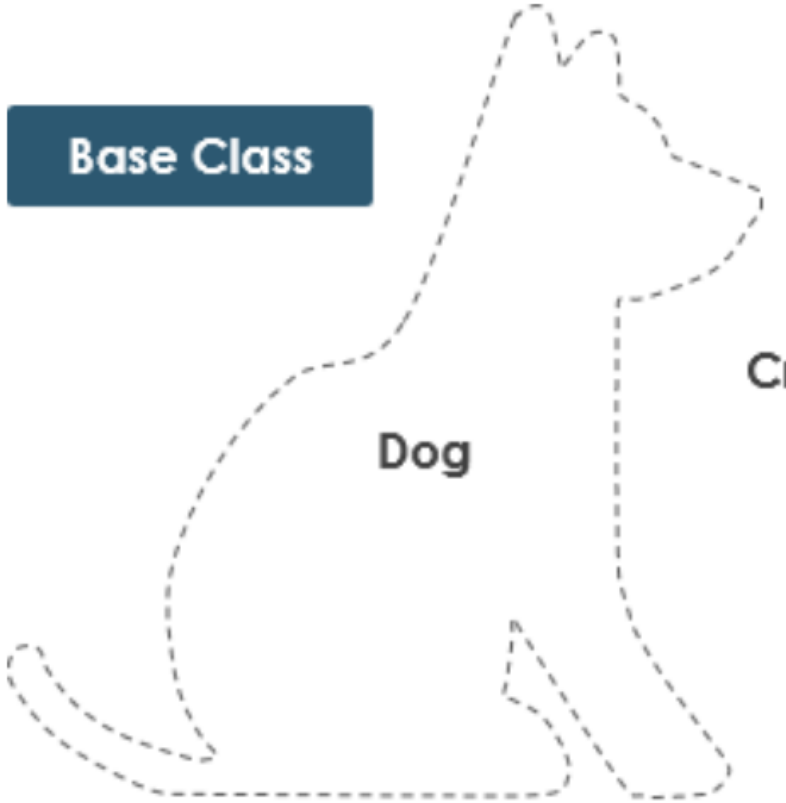
Information Systems Analysis & Modeling

Lecture 10: Class diagram

Mona Taghavi

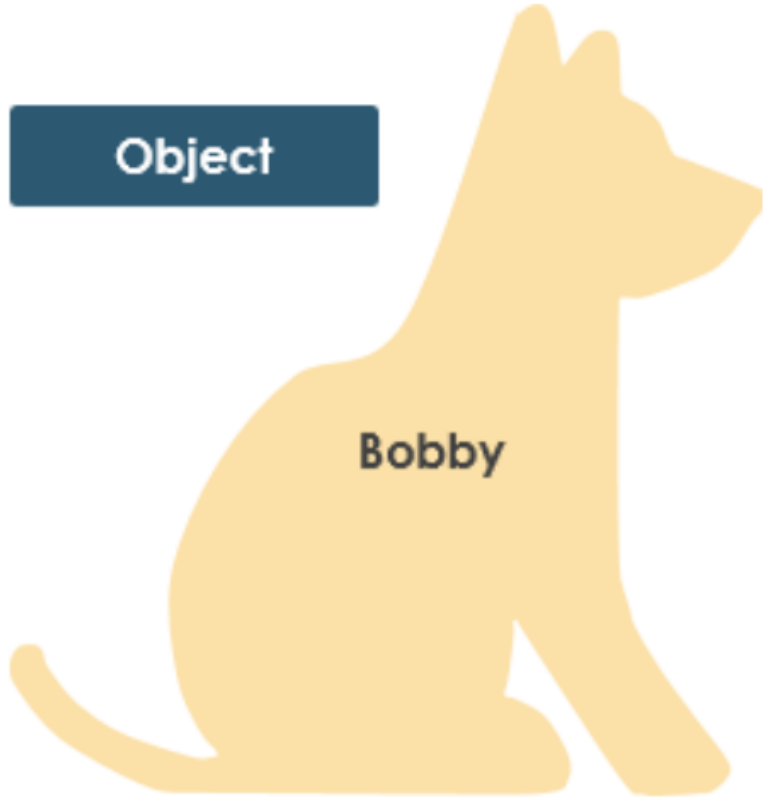
LaSalle College
Montréal

Base Class



Create instance

Object



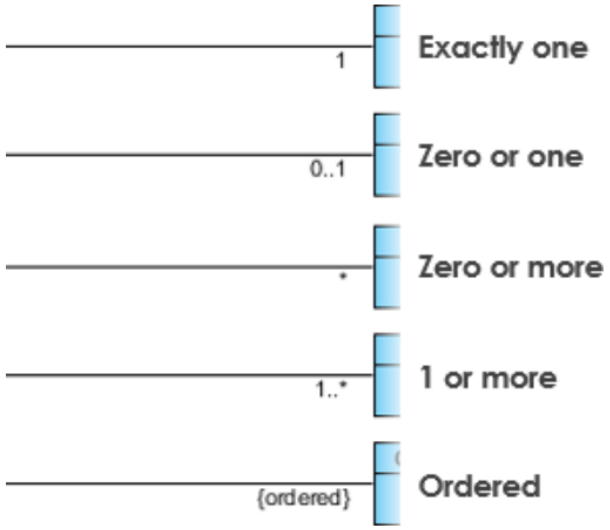
Properties	Methods
Color	Sit
Eye Color	Lay Down
Height	Shake
Length	Come
Weight	

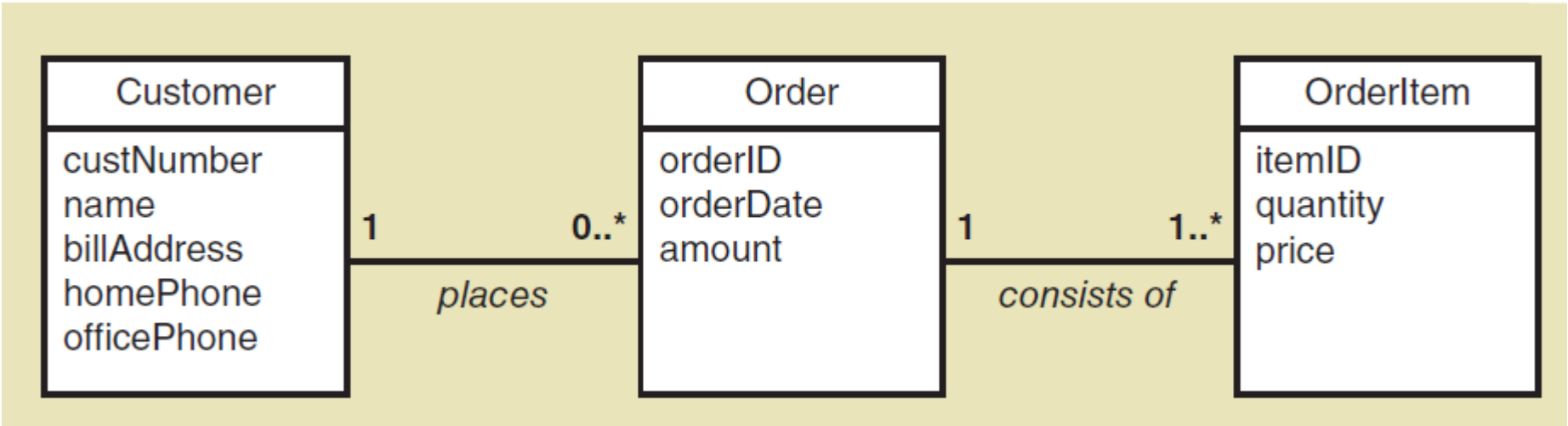
Property Values	Methods
Color: Yellow	Sit
Eye Color: Brown	Lay Down
Height: 17 in	Shake
Length: 35 in	Come
Weight: 24 pounds	

UML class diagram

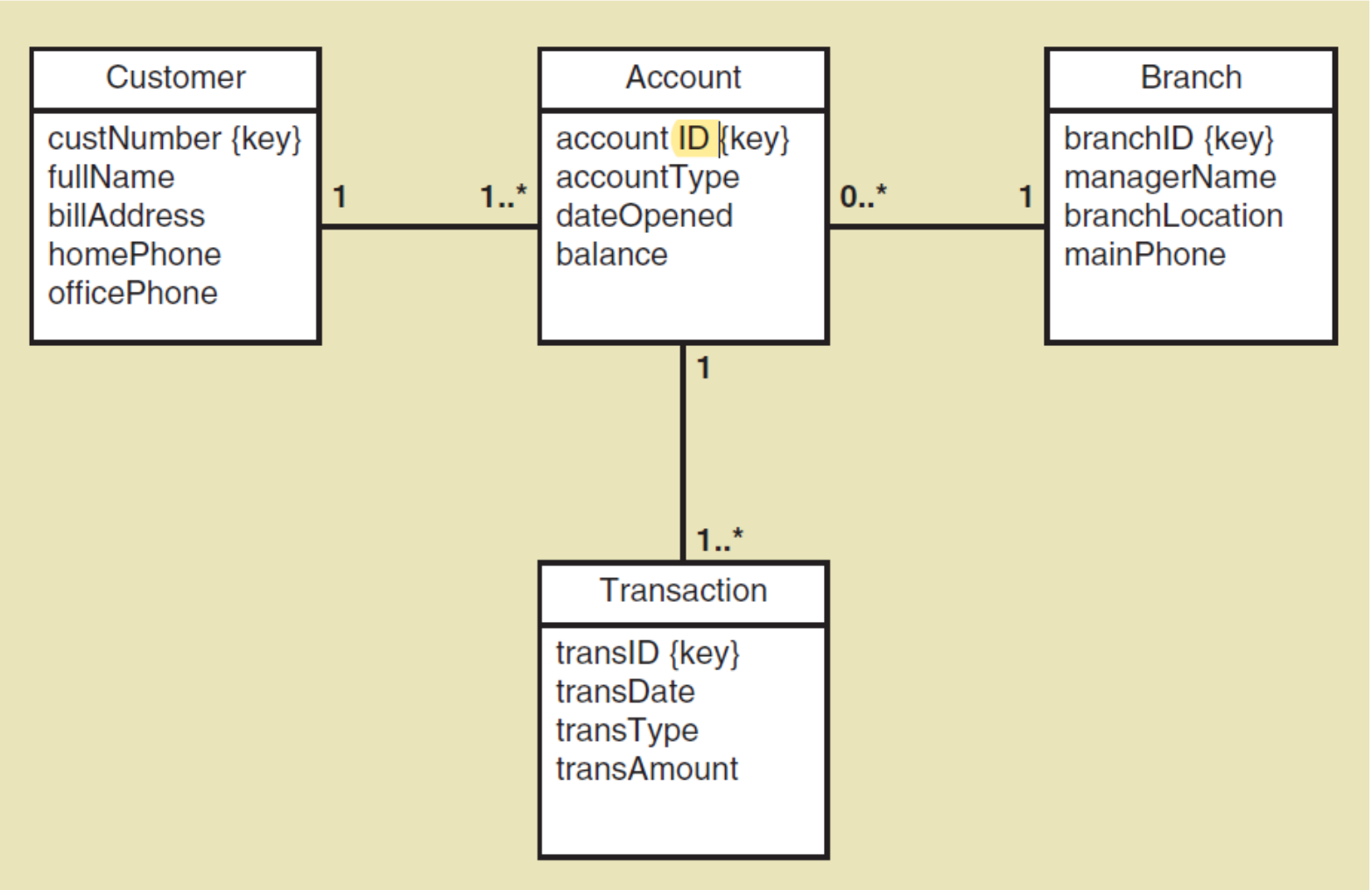
- The UML **class diagram** is used to show classes of objects for a system.
- Class names and attribute names use **camelback** (sometimes called **camelcase**) **notation**, in which the words run together without a space or underscore. Class names begin with a capital letter; attribute names begin with a lowercase letter.

Cardinality

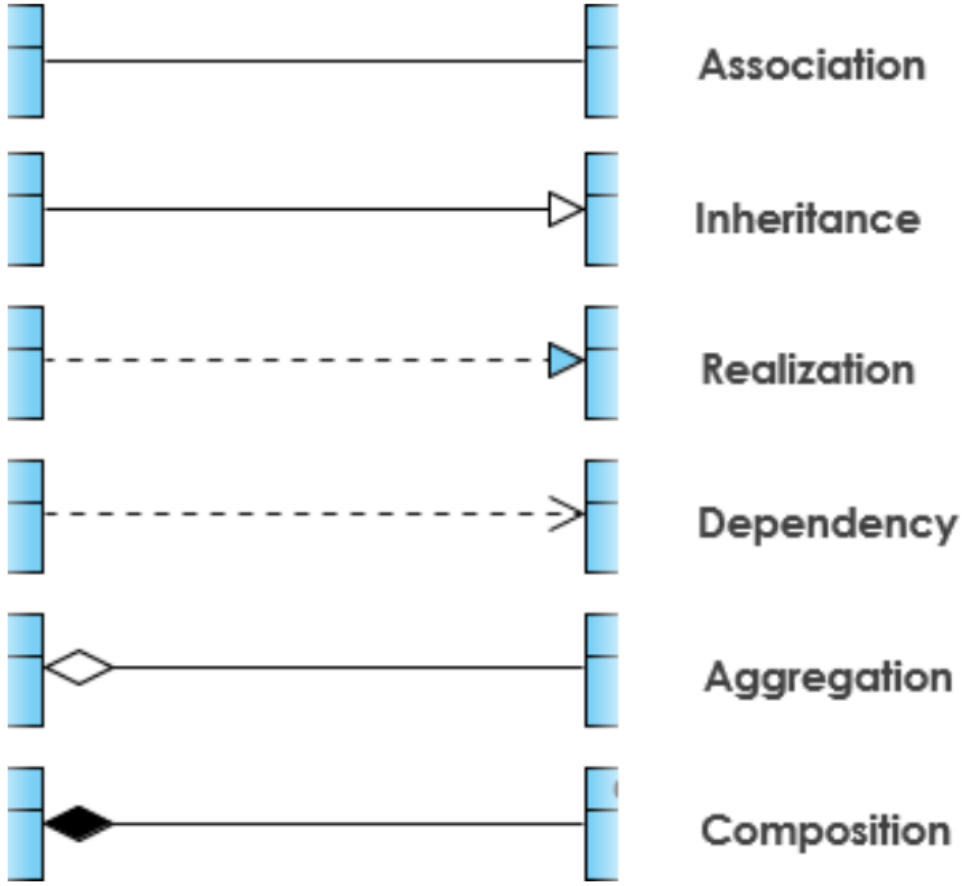



Customer
custNumber
name
billAddress
homePhone
officePhone
1
0..*
places
Order
orderID
orderDate
amount
1
1..*
consists of
OrderItem
itemID
quantity
price

A domain model class diagram for a bank

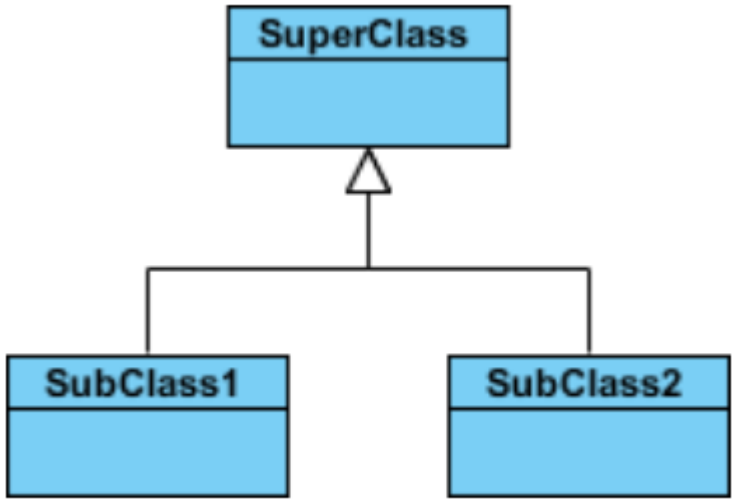


Relationships between classes

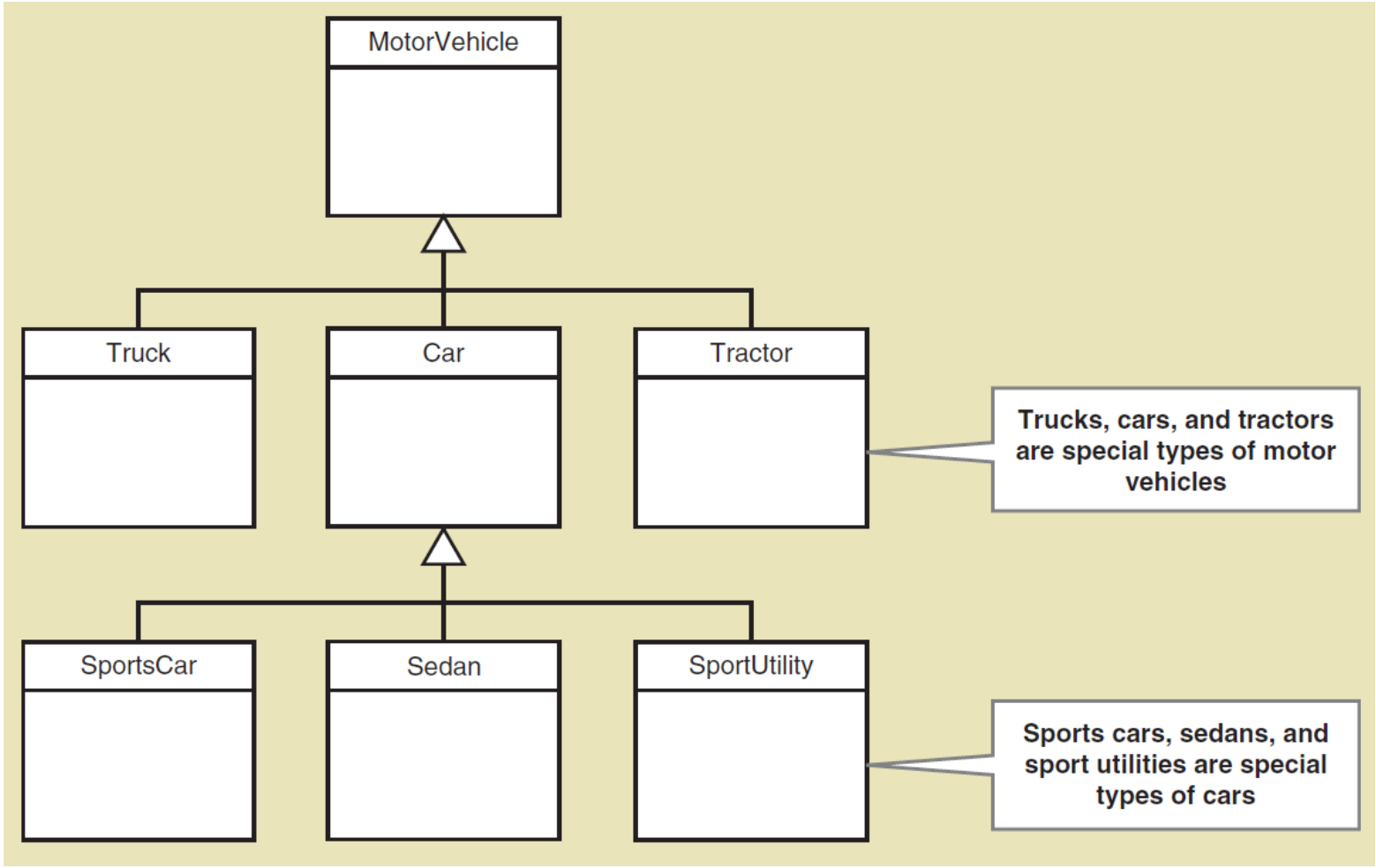


Inheritance (or Generalization)

- A generalization is a taxonomic relationship between a more general classifier and a more specific classifier. Each instance of the specific classifier is also an indirect instance of the general classifier. Thus, the specific classifier inherits the features of the more general classifier.
- Represents an "is-a" relationship.

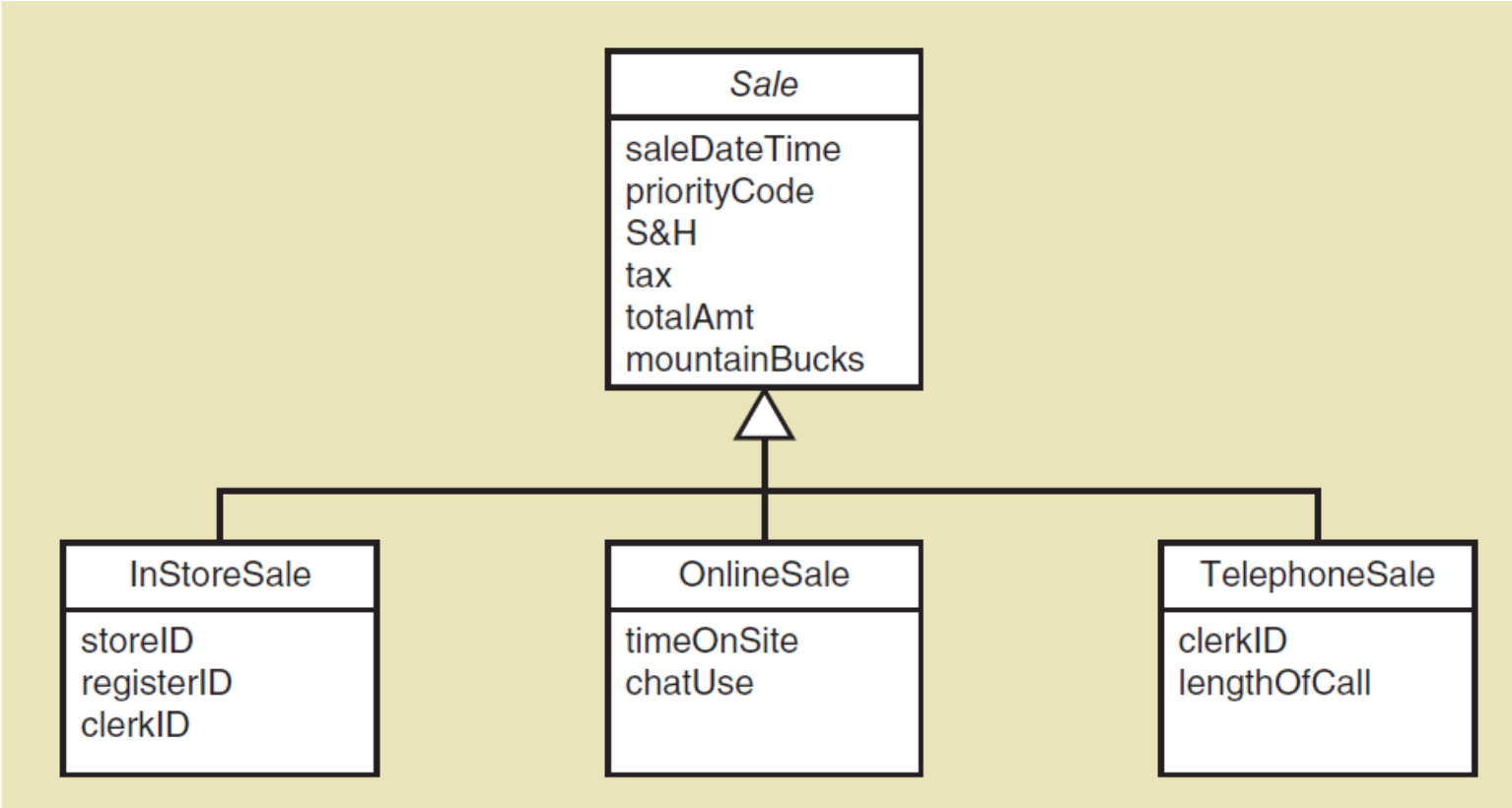


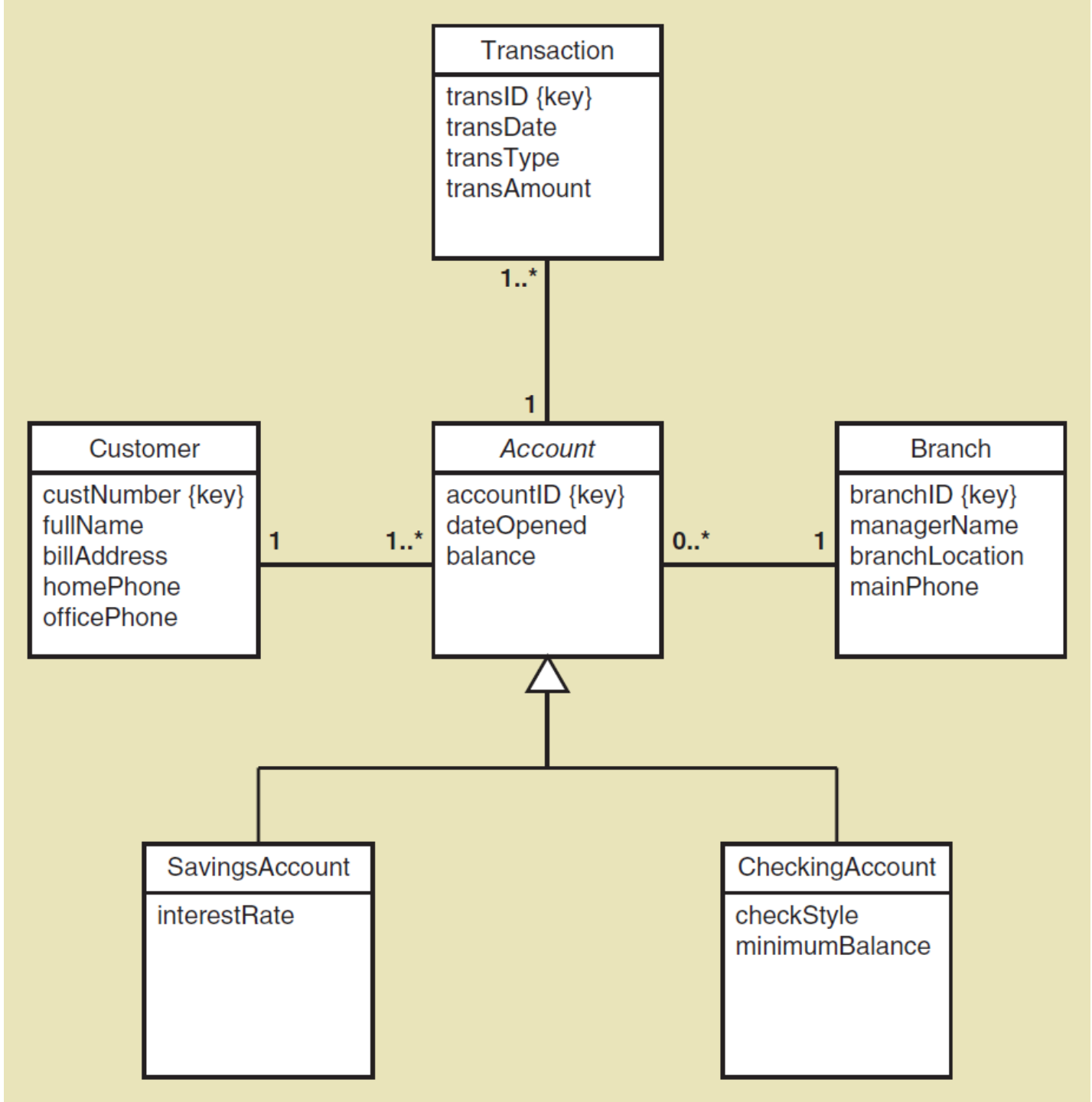
Generalization/specialization relationships for motor vehicles



Abstract class

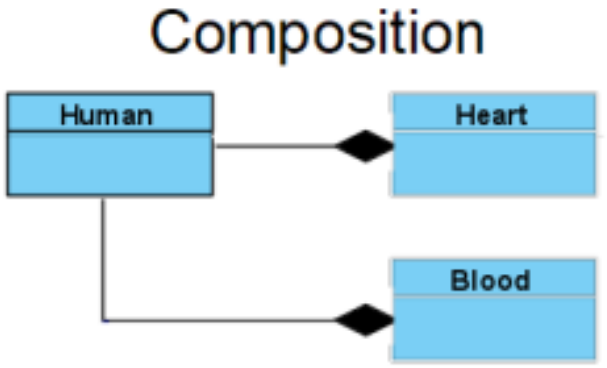
- An **abstract class** is a class that only exists so subclasses can inherit from it.
- A **concrete class** is a class that does have actual objects.

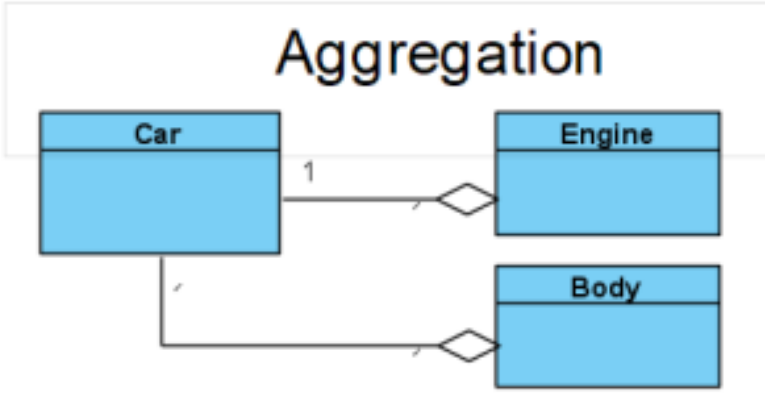



Transaction
transID {key}
transDate
transType
transAmount
1..*
1
Customer
custNumber {key}
fullName
billAddress
homePhone
officePhone
1
1..*
Account
accountID {key}
dateOpened
balance
0..*
1
Branch
branchID {key}
managerName
branchLocation
mainPhone
SavingsAccount
interestRate
CheckingAccount
checkStyle
minimumBalance

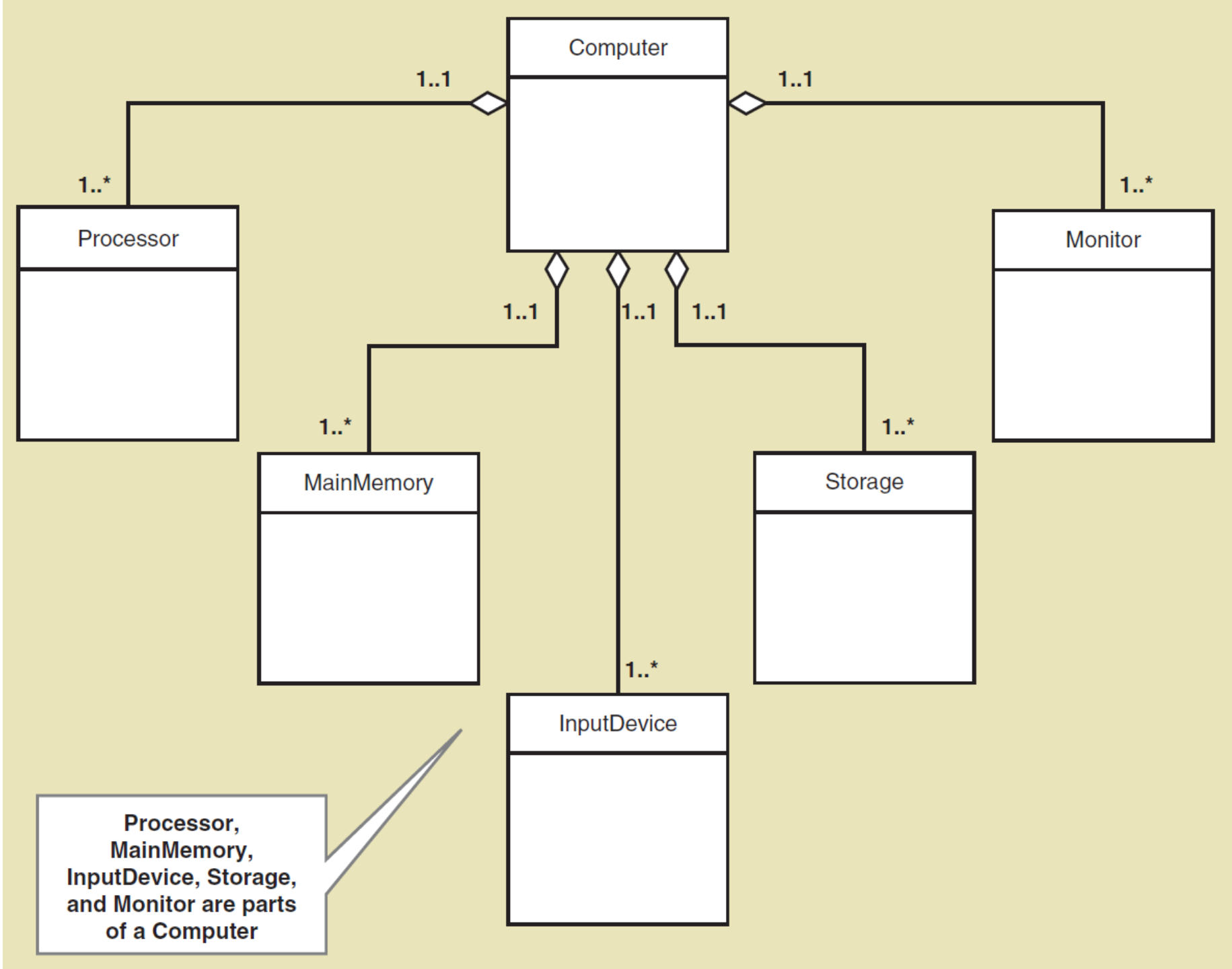
Whole-part relationships

- There are two types of whole-part relationships: aggregation and composition.
- **Aggregation** refers to a type of whole-part relationship between the aggregate (whole) and its components (parts), where the parts can exist separately.
- **Composition** refers to whole-part relationships that are even stronger, where the parts, once associated, can no longer exist separately.



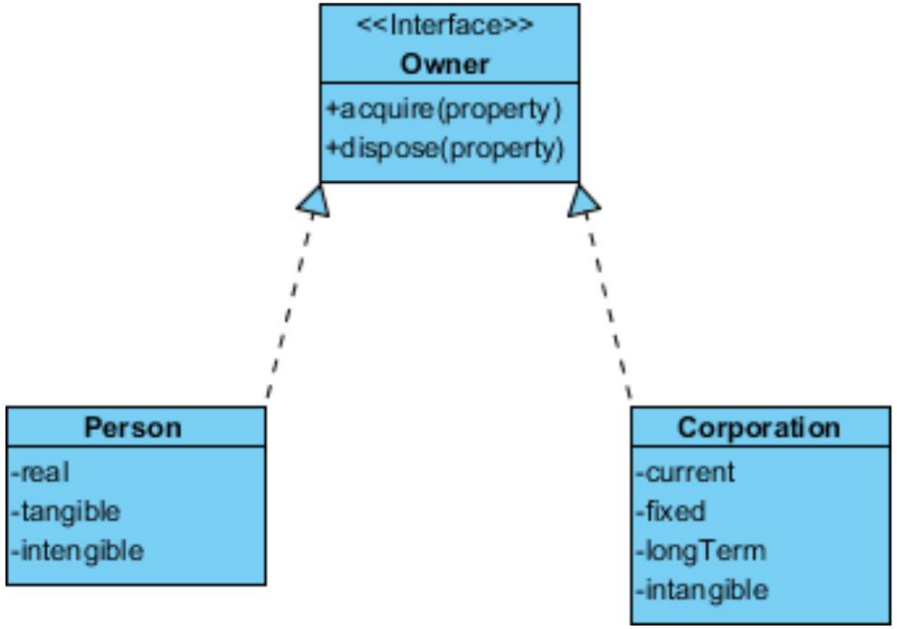


Aggregation in a computer system



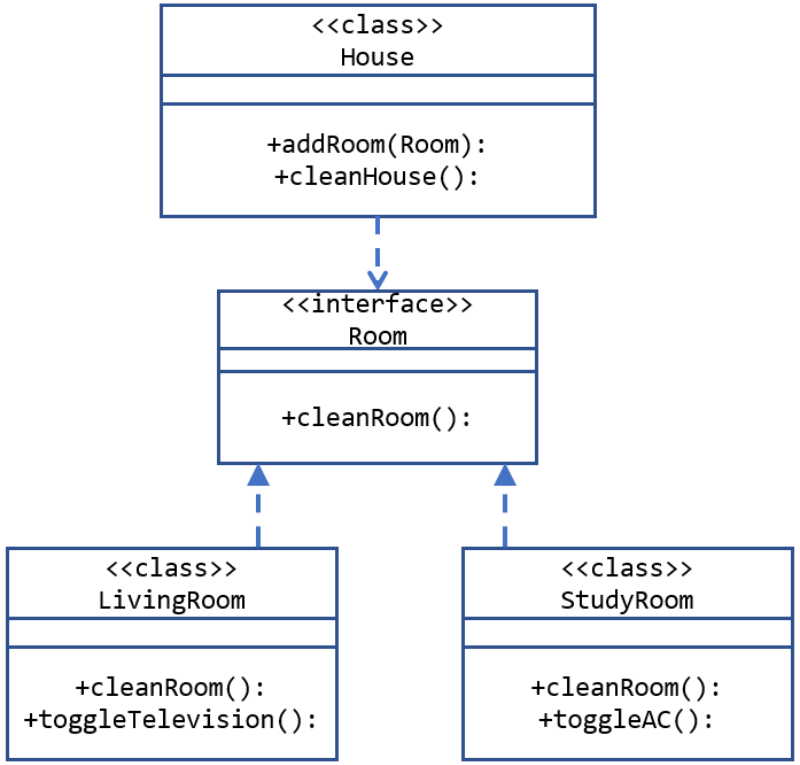
Realization

- Realization is a relationship between the blueprint class and the object containing its respective implementation level details. In other words, you can understand this as the relationship between the interface and the implementing class.
- For example, the Owner interface might specify methods for acquiring property and disposing of property. The Person and Corporation classes need to implement these methods, possibly in very different ways.

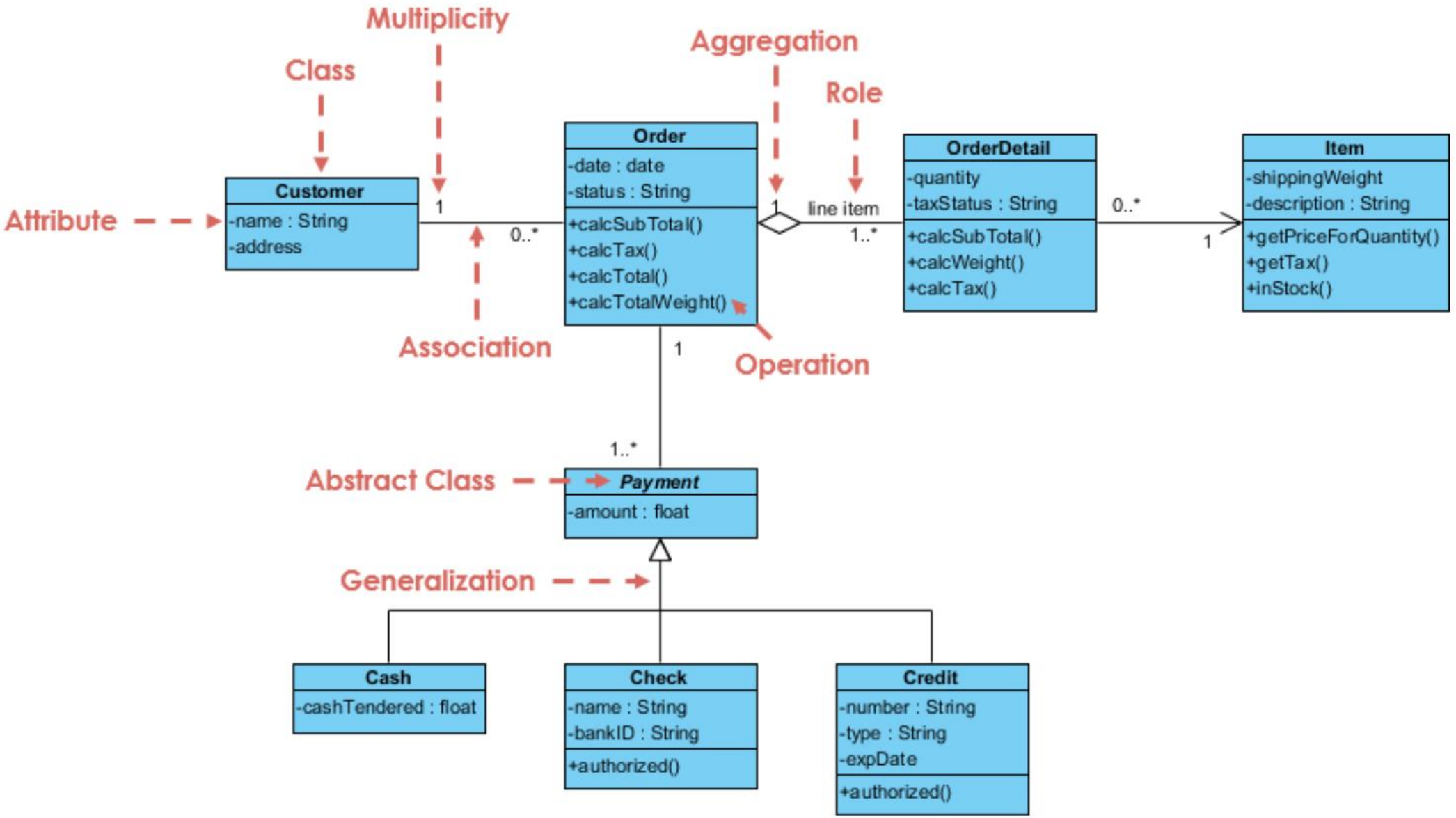


Dependency

- An object of one class might use an object of another class in the code of a method. If the object is not stored in any field, then this is modeled as a dependency relationship.
- Exists between two classes if changes to the definition of one may cause changes to the other (but not the other way around).



Order System (in parts)



Class exercise

Exercise-Complementary explanation

- Adding a domain class, called an **association class**, to represent the association between Student and CourseSection provides a place in which to store the missing attribute for grade.