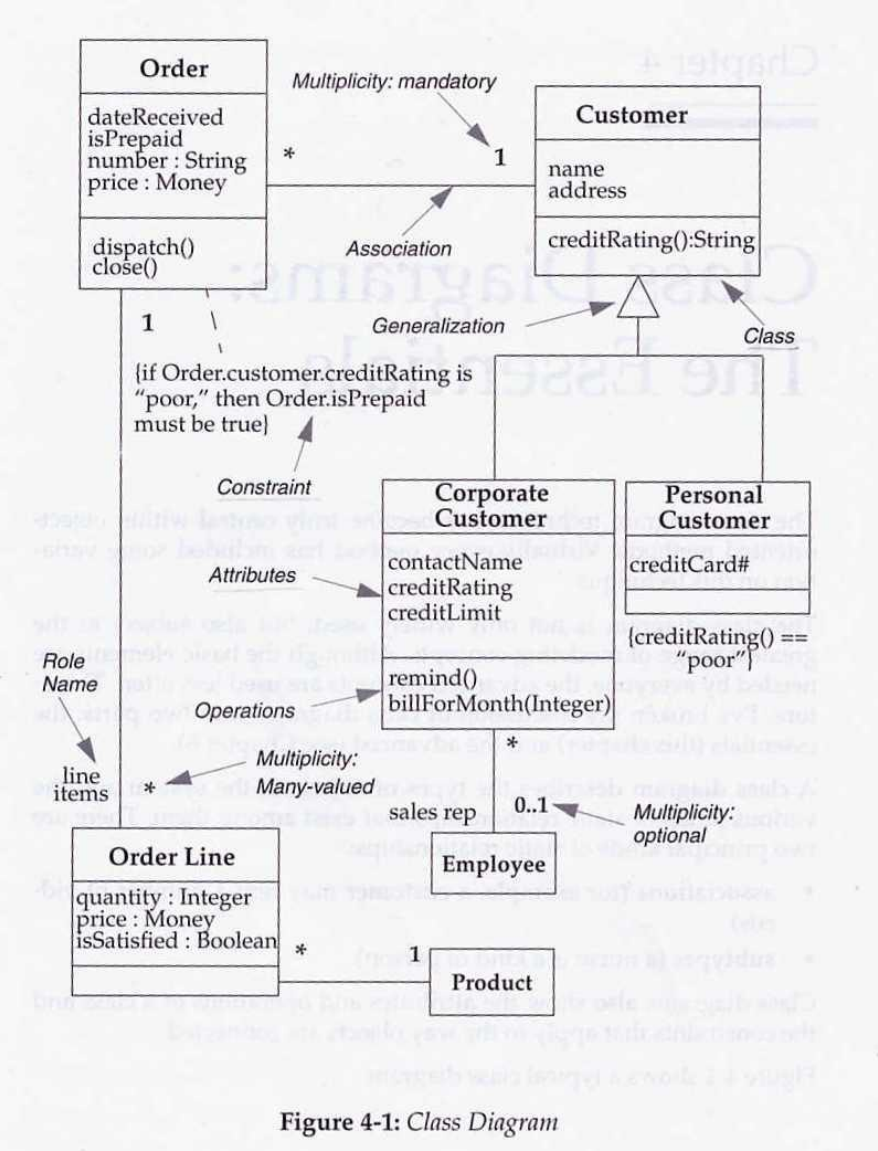


Figure 4-1: *Class Diagram*

Class Diagrams

Object-Oriented Analysis and Modeling
INFO3140

Dr. Emad Saleh

Class Diagram Concepts

- A class is a blueprint or a general template we use to define and create specific instances or objects.
  - It defines the structure and behavior of objects.
  - It encapsulates data (attributes) and functions (methods) that operate on that data.
  - (It organizes and models data and behavior)

- The UML Reference Manual states…

"The descriptor for a set of objects that share the same attributes, operations, methods, relationships and behaviour."

Class Diagram Concepts

- A static model that shows the classes and relationships among classes that remain constant in the system over time
  - A static model is a fundamental tool in software engineering that focuses on the unchanging structural aspects of a system.
  - It captures the classes and their relationships in the system that remain constant.
  - It allows for a comprehensive understanding of how objects interact within the system.
- Depicts classes which include both behaviors and states
  - The combination of states and behaviors within classes is crucial for modeling how objects interact with each other and respond to different events or actions in a software system.

Class Diagram Concepts

- Requirements of a well-formed design class
  - Completeness and sufficiency
    - it includes all the necessary attributes and methods to fulfill its purpose.
    - it doesn't contain unnecessary or redundant elements.
  - Primitiveness
    - a class should do one thing and do it well. This simplifies the class's design and makes it easier to understand and maintain.
  - High cohesion
    - the elements within a class should be closely related and work together to achieve a common goal (code readability and maintainability).
  - Low coupling
    - it refers to the level of interdependence between classes.
    - they are loosely connected to other classes. It enhances the flexibility of the system, making it easier to modify or extend without affecting unrelated parts of the code.
  - Aggregation vs inheritance
    - Aggregation is used when one class is composed of other classes as parts. ("has-a" relationship) (the "Car" class aggregates the "Engine" class.)
    - Inheritance establishes an "is-a" relationship, where one class derives from another. It's used when a class shares the characteristics and behaviors of another class. (the "Circle" and "Square" classes inherit from the "Shape" class.)

Class Diagram Concepts

- ***Def***. classes whose specifications have been completed to a degree that they can be implemented (sufficiently designed and documented)
- Design classes come from 2 places
  - ***Problem domain*** – refinement of **analysis classes** by adding implementation details (**Design classes**)
    - analysis phase captures the essential components and relationships in the problem domain
    - Design classes enrich the initial abstractions with implementation details, such as attributes, methods, and interactions. It makes the classes ready for coding.
  - ***Solution domain*** –provides technical tools that allow you to implement the system, such as
    - **Reusable components;** which are pre-built and tested modules or libraries that can be integrated into the software
    - **Utility class libraries;** provide a collection of commonly used functions and classes that can simplify the development process.
    - **Databases;** design classes often represent data models, database connections, and data access logic. These classes interact with databases to store, retrieve, and manipulate data as needed.
    - **GUI frameworks;** these classes help manage user interfaces, handle user input, and display information in a visual way.

Class Diagram Concepts

- Analysis class may refine into 1 or more design classes or interfaces
  - refers to the process of transitioning from the analysis phase to the design phase
- Why? may need to be broken down into multiple design classes or interfaces.
  - **Modularity and Cohesion:** Breaking down large analysis classes into smaller design classes promotes modularity, which is a key design principle. Each design class can focus on a specific aspect or service, ensuring high cohesion and making the software more maintainable and understandable. (reusability and code efficiency)
- Analysis class is at high level of abstraction
  - it serves to capture the essential concepts and services offered by the class within the problem domain
- Is a sketch of key services offered by the class
  - These services can include actions, behaviors, or functionalities that the class needs to provide to fulfill its role in the system.
- When moving to design – must specify all of the operations and attributes so class may be too large and have to broken down into small, self-contained, cohesive units.

Class Diagram Concepts

- Refinement Example

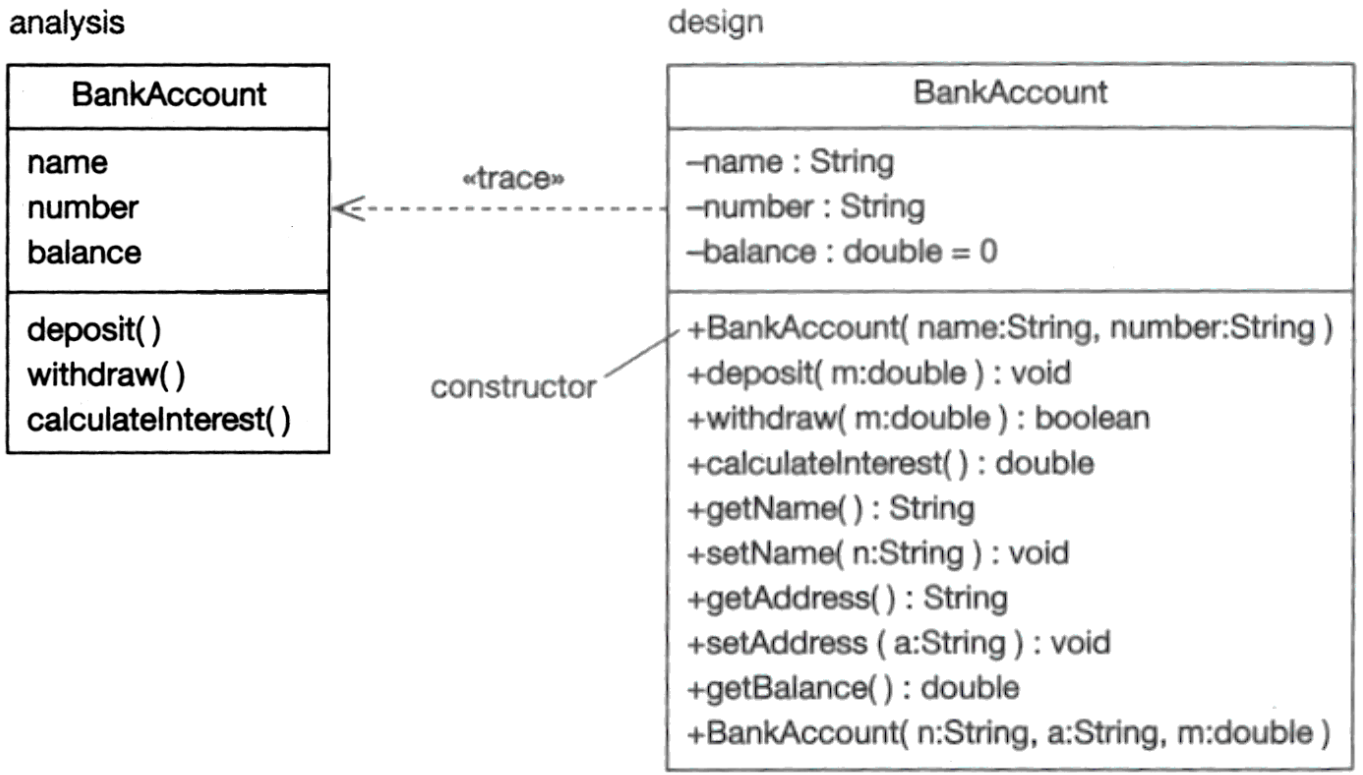


Figure 15.2

Class Diagram Concepts

- ***Operation***- high-level logical specification of a piece of functionality offered by a class
  - Each class can offer various functions or services to its objects
  - It focuses on the purpose and expected results of the operation
  - Example: in a "BankAccount" class, an operation named "withdraw." The high-level logical specification for this operation would describe that it allows account holders to withdraw a specified amount of money from their account. **The specification would not focus on the details of how the withdrawal is implemented in code** but would focus on the **functional aspect**.
- ***Method*** – fully specified function that can be implemented as source code
  - It is defined with details regarding its name, input parameters, return value, and the actual steps to perform its function. This specification outlines how the method should behave and what it should achieve.
  - Example, in a "BankAccount" class with a "withdrawal" method. This method takes an amount as input and performs a withdrawal operation from the account's balance. If the withdrawal is successful, it returns the withdrawn amount. If the withdrawal is invalid, it returns 0, indicating an unsuccessful withdrawal.
- 1 high level operation may resolve into 1 or more implementable methods
  - It allows for better code organization, reusability, and the efficient handling of complex tasks, making the software more modular and maintainable.

Class Diagram Concepts

- **Well – Formed Design Classes**
  - **Completeness**
    - Give the clients of the class what they expect
    - E.g. Bank account
      - If has withdrawals must include deposits
  - **Sufficiency**
    - Class must contain expected set of methods and no more
    - E.g. Bank account should *not* include credit card or insurance polices

Class Diagram Concepts

- **Primitiveness**
  - Method should offer a single atomic service
    - perform a single, well-defined task or operation.
  - Should not offer multiple ways of doing the same thing
    - Can lead to maintenance burdens and consistency problems
    - Eg Bank account class,
  - One method for handling single and multiple deposits
    - In the redundant approach, the "deposit" method offers multiple ways to deposit money: via a check or cash. This can lead to confusion, increased maintenance, and potential inconsistencies.
    - In the improved approach, the "BankAccount" class provides separate, clearly defined methods for different deposit scenarios. Each method offers a single way to perform the associated action, enhancing code clarity and reducing maintenance and consistency problems.

Class Diagram Concepts

- **High Cohesion**
  - A cohesive class has small set of responsibilities that are related
    - This minimizes the complexity of the class and makes it easier to understand
  - Most desirable feature of a class
    - leads to classes that are clear, focused, and maintainable. A cohesive class is more self-contained, meaning it doesn't have many dependencies on other parts of the code, making it easier to reuse.
  - Easy to understand, reuse and maintain

Class Diagram Concepts

- **Low coupling**
  - Class association should be limited to those which are necessary for realization of responsibilities
    - classes should interact with one another only when it's essential for the system's functioning.
  - Highly coupled object model is like “spaghetti code” in non-OO world
    - highly coupled systems can be difficult to understand, modify, and maintain.

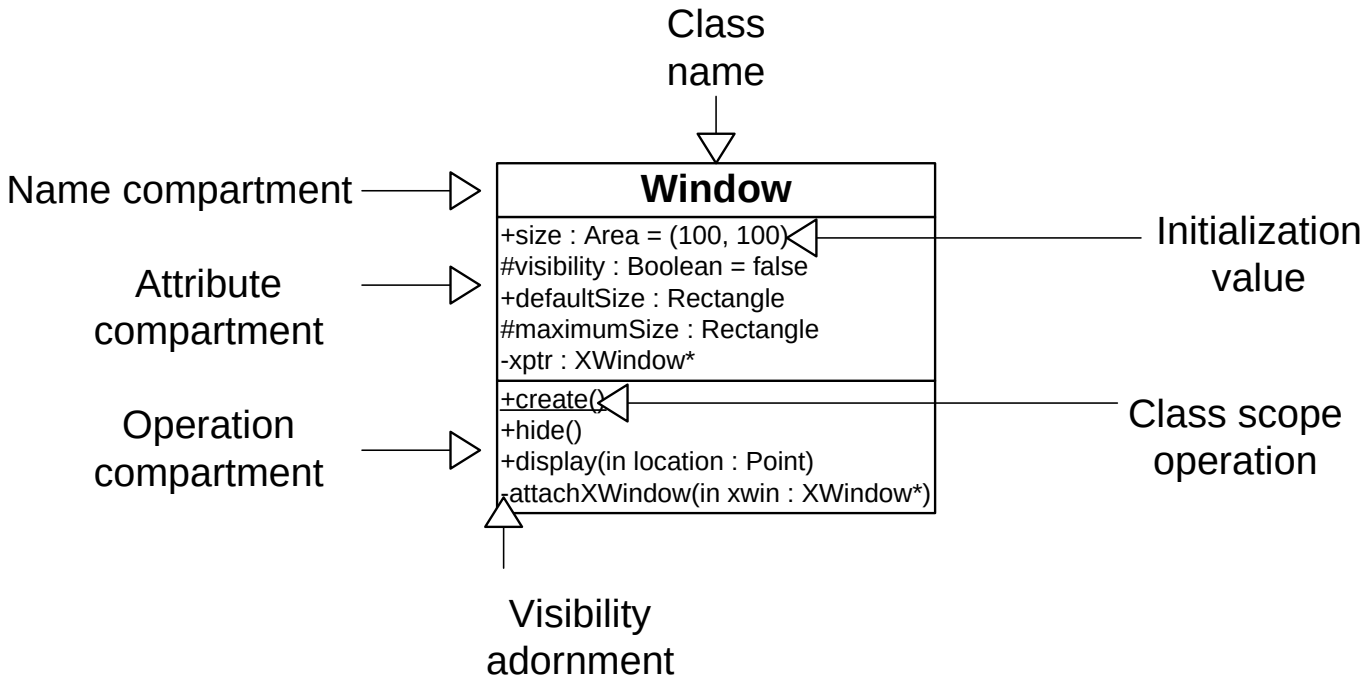
Class Diagram Concepts

- **Inheritance**
  - Using inheritance effectively
    - it allows a new class (subclass or derived class) to inherit properties and behaviors from an existing class (base class or superclass)
    - it supports the "is-a" relationship, meaning a derived class "is-a" type of the base class. It promotes code reuse, modularity, and polymorphism (the ability to treat objects of derived classes as objects of the base class).
    - Effective use of inheritance simplifies software design, enhances code reuse, and facilitates the organization and maintenance of classes and objects in an object-oriented system.

Class Diagram Concepts

Term and Definition	Symbol
A class Represents a kind of person, place, or thing about which the system must capture and store information Has a name typed in bold and centered in its top compartment Has a list of attributes in its middle compartment Has a list of operations in its bottom compartment Does not explicitly show operations that are available to all classes	
An attribute Represents properties that describe the state of an object Can be derived from other attributes, shown by placing a slash before the attribute's name	Attribute name /derived attribute name
A method Represents the actions or functions that a class can perform Can be classified as a constructor, query, or update operation Includes parentheses that may contain special parameters or information needed to perform the method	Method name ()
An association Represents a relationship between multiple classes, or a class and itself Is labeled using a verb phrase or a role name, whichever better represents the relationship Can exist between one or more classes Contains multiplicity symbols, which represent the minimum and maximum times a class instance can be associated with the related class instance	

Class Diagram Concepts



Class Diagram Concepts

- Name compartment
  - Class Name usually in title case “Bid”
- Attribute compartment
  - Each attribute has
    - Visibility
      - this controls access to the features of the class
      - +(public),#(protected), or –(private), /(derived)
    - Name (mandatory) – usually lower case
    - Multiplicity – arrays e.g. [10]; null values e.g. [0..1];
    - Type
    - Default values (name: type = default value)

Class Diagram Concepts

- Operation/Methods compartment
  - Each operation may have
    - Visibility – Public, Private and Protected
    - Name (mandatory) – usually lower case
    - Parameter list (name and type for each parameter)
    - Return type

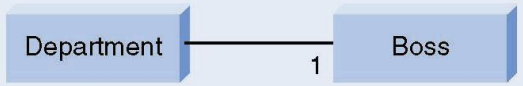
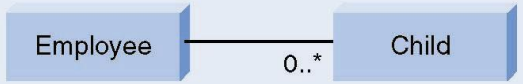
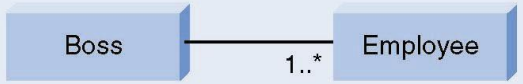
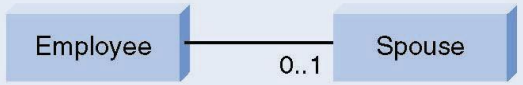
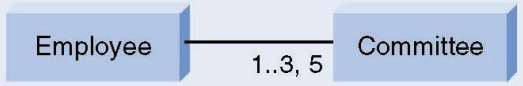
Class Diagram Concepts

- Types of Methods
  - Constructor methods create new instances of a class
    - Constructor methods are responsible for creating and initializing new instances of a class (objects). They are typically used to set the initial state of an object when it is created.
  - Query methods determine the state of an object and make information about that state available to the system
    - Query methods provide information about state of an object to the system. They do not modify the object's state but rather provide access to it.
  - Update methods will change the value of some or all of the object’s attributes, resulting in a change of state
    - Update methods, also known as mutator methods, are responsible for changing the values of an object's attributes, which results in a change of the object's state.

Class Diagram Concepts

- Associations
  - Represent associations between instances of classes
    - Associations in UML diagrams represent relationships or connections between instances of classes. They illustrate how different classes are related to each other by defining how objects collaborate and communicate in a system.
  - Both ends of each line can have a role name associated with it
    - Role names indicate the purpose or role that each end plays in the relationship.
      - clarify the meaning of the association. Example, "Teacher" and "Student," are associated, to describe the relationship.
    - Role Names:
    - "teaches" (role of the Teacher)
    - "is taught by" (role of the Student)
  - Multiplicity indicates lower and upper bounds for the participating objects
    - It specifies how many instances of one class can be associated with instances of another class.
    - Teacher end: "1" (each teacher teaches at least one student)
    - Student end: "0..*" (each student may be taught by multiple teachers)

Class Diagram Concepts

Instance(s)	Representation of Instance(s)	Diagram Involving Instance(s)	Explaination of Diagram
Exactly one	1		A department has one and only one boss.
Zero or more	0..*		An employee has zero to many children.
One or more	1..*		A boss is responsible for one or more employees.
Zero or one	0..1		An employee can can be married to zero or no spouse.
Specified range	2..4		An employee can take between two to four vacations each year.
Multiple, disjoint ranges	1..3, 5		An employee is a member of one to three or five committees.

Class Diagram Concepts

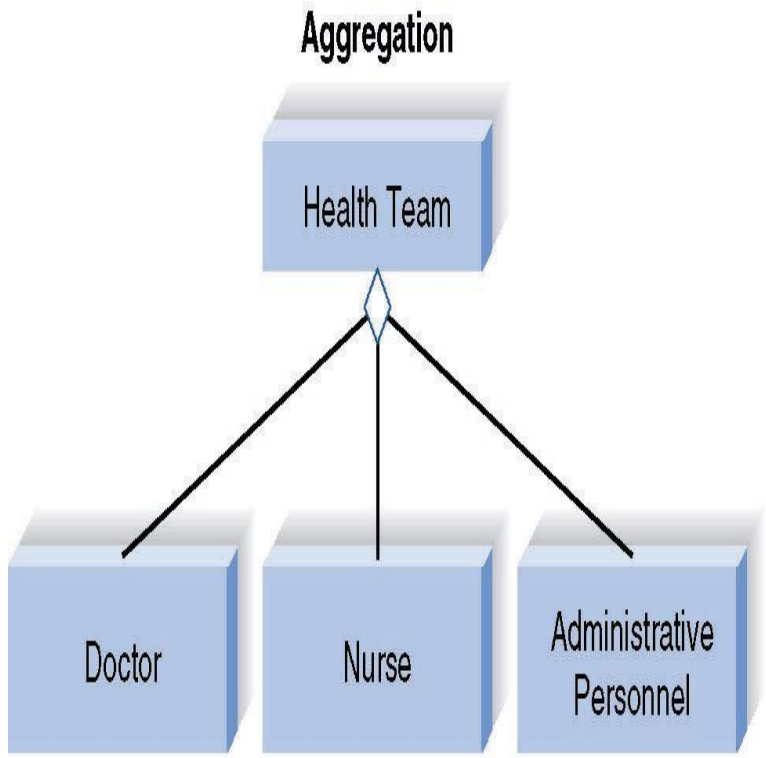
- Constraint Rules
  - Constraints in a UML class diagram are used to specify additional rules, conditions, or restrictions that apply to a class or its elements.

- Any number of constraints can be captured on a class diagram
  - Constraints are used to provide additional information about the class, its attributes, operations, or relationships. {age >= 6}
- The only rule is that all constraints must be placed inside braces {}
- UML does provide a formal Object Constraint Language but this is optional
  - UML provides the Object Constraint Language (OCL) as a formal language for specifying constraints. OCL is a precise and formal way to express constraints using a formal syntax. {age >= 6} is written in OCL and is placed within braces.

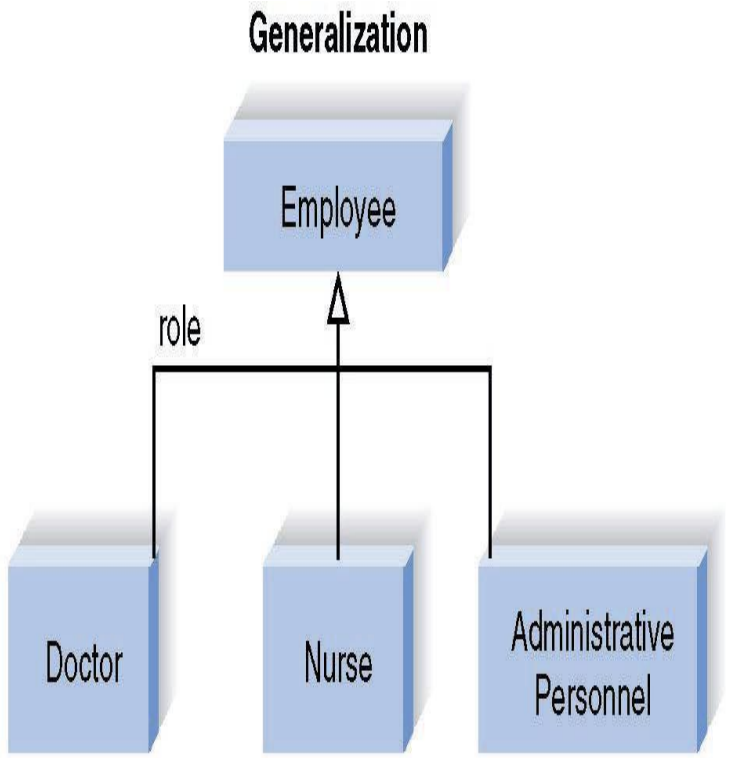
Class Diagram Concepts

- Generalizations
  - Generalizations describe the hierarchical relationship between classes. (The parent class is more general, while the child class is more specific)
  - It describes the 'family tree'. Classes may inherit attributes and behavior from a parent class (which may in turn be the child of another class). Showing how various classes are related through inheritance.
  - This tree of inherited characteristics and behavior allows the designer the ability to collect common functionality in root classes and refine and specialize that behavior over one or more generations.

Class Diagram Concepts



Doctors, nurses, and administrative personnel are "a part of" a health care team



Doctors, nurses, and administrative personnel are "a kind of" employee

Class Diagram Concepts

- Aggregation
  - Aggregation relationships define whole/part relationships. Where the "whole" class encompasses or is composed of one or more "part" classes
  - For example, a train journey may be composed of many journey legs.

The overall journey then aggregates or is composed of the separate legs.

  - The train journey is the "whole," and each journey leg is a "part" that contributes to the overall journey.

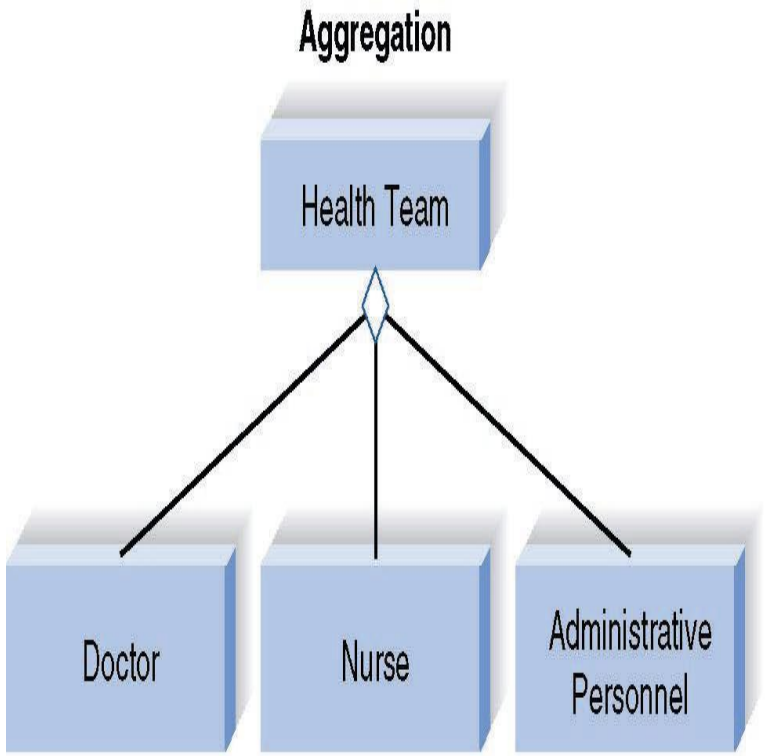
The stronger form of **aggregation** is **composition** and infers that a class not only collects another class, but is actually composed of that collection.

  - In composition, the "whole" class manages the lifecycle of the "parts." If the whole is destroyed, the parts are also typically destroyed.

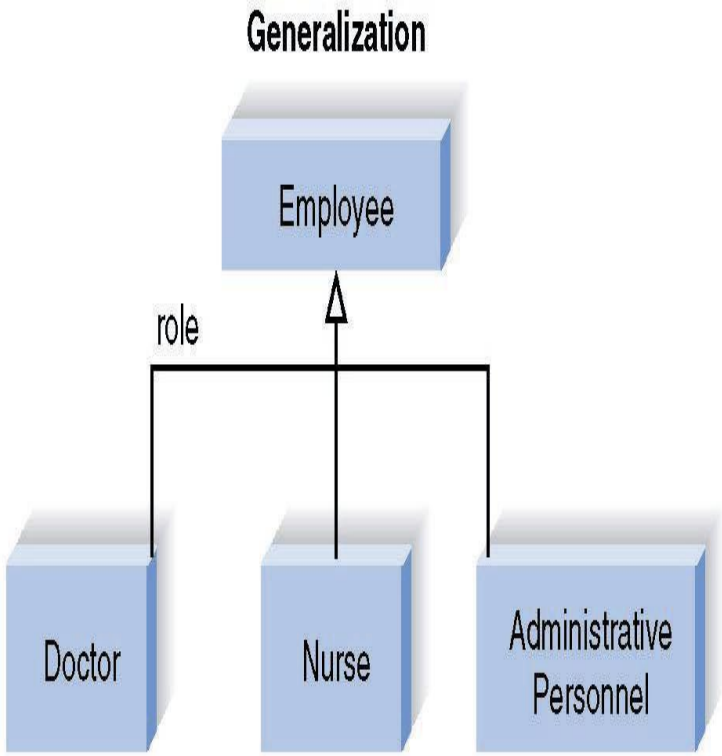
In UML diagrams, **aggregation** is represented using a **diamond-shaped arrow** on the side of the whole class. A line with an **empty diamond head signifies aggregation**, while a line with a **filled diamond head indicates composition**.

  - **Composition:**
  - A car (whole) is composed of an engine, wheels, and other components (parts).
  - The car has a composition relationship with its components.
  - If a car is destroyed, its components (engine, wheels, etc.) are destroyed as well.

Class Diagram Concepts



Doctors, nurses, and administrative personnel are "a part of" a health care team



Doctors, nurses, and administrative personnel are "a kind of" employee

Class Diagram Concepts

- **Association Class**
  - A class that describes a relationship between two classes or between instances of two classes.
    - An association class in UML is a class that represents a relationship between two other classes or between instances of two classes. It is used to capture additional information or attributes related to the association itself. Association classes are a way to add more details to an association without having to modify the original classes involved in the relationship.

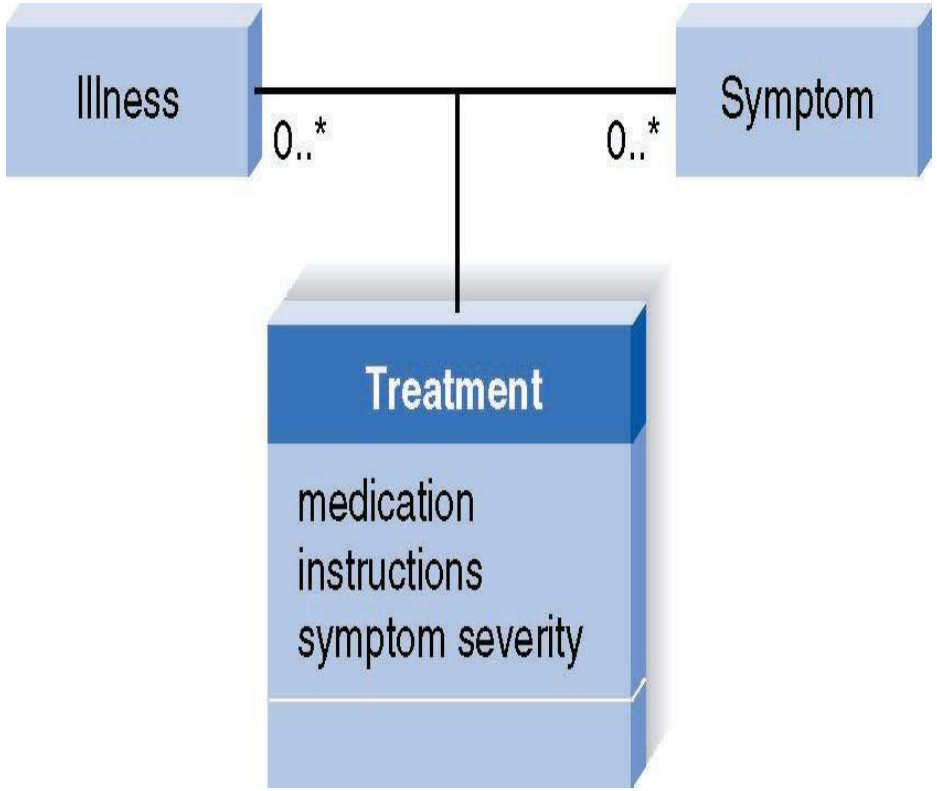
Class Diagram Concepts

Association Class

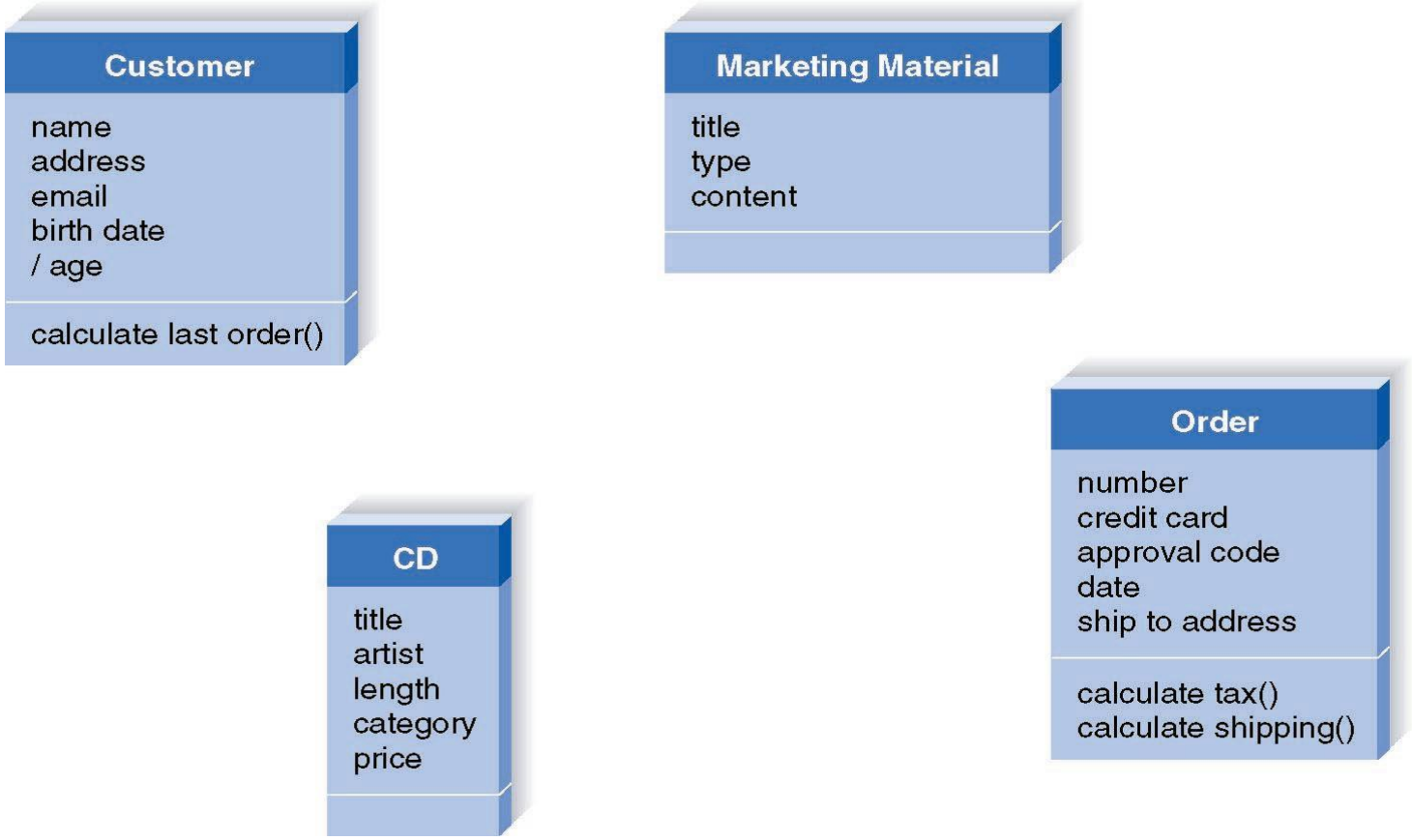
- Example with association classes to represent the relationship between "Illness", "Symptom," and an association class "Treatment." In this example, we'll capture the details of how an illness is associated with its symptom and the treatments that can be applied.

- "Illness" represents a class for various illnesses.
- "Symptom" represents a class for various symptoms.
- "Treatment" represents an association class, which captures information related to treatments for specific illnesses.

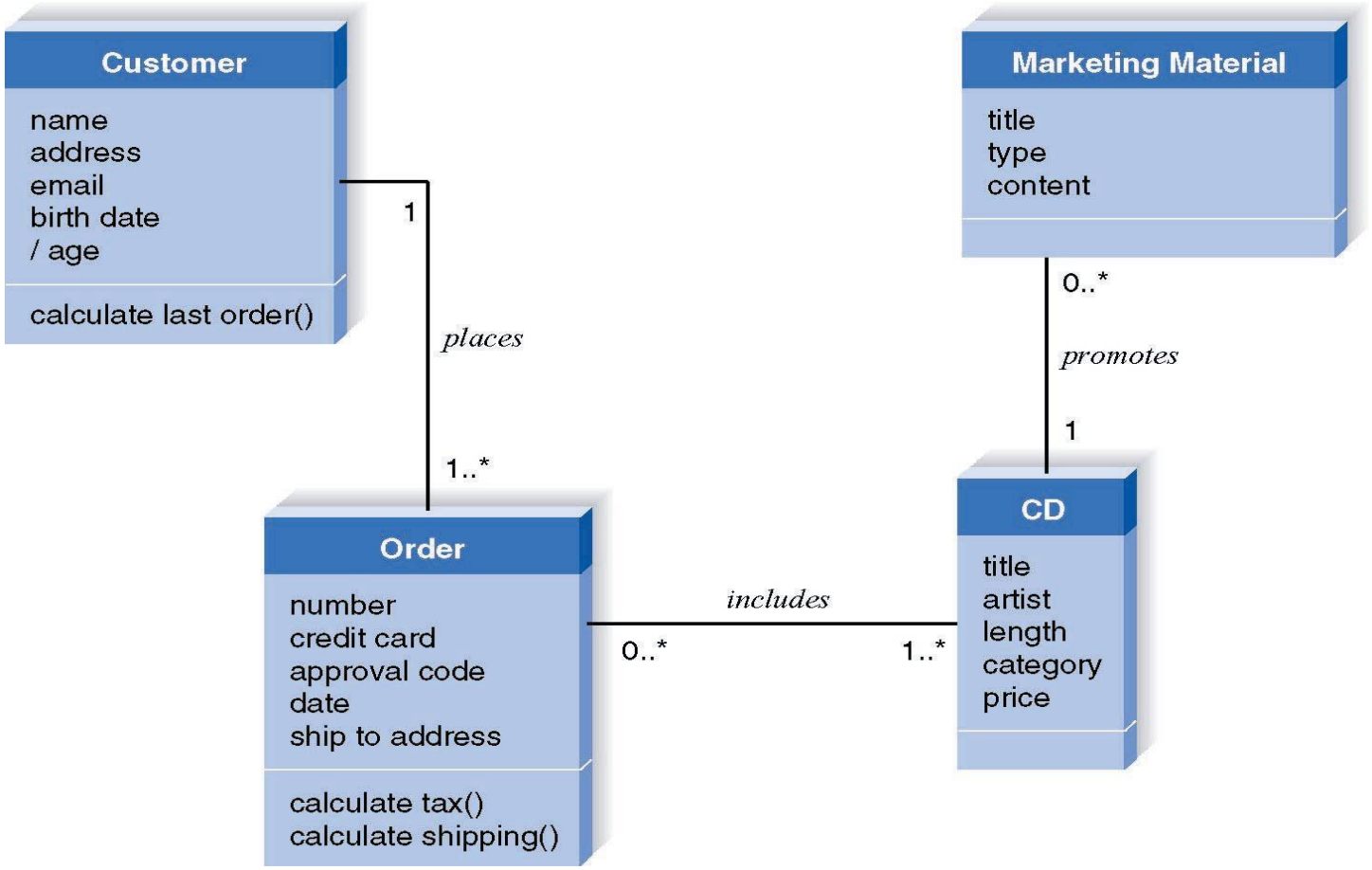
- This UML diagram models the relationship between illnesses, their symptoms, and the treatments that can be applied to those symptoms. The "Treatment" association class allows you to capture additional information related to the treatment of symptoms without directly modifying the "Illness" or "Symptoms" classes.



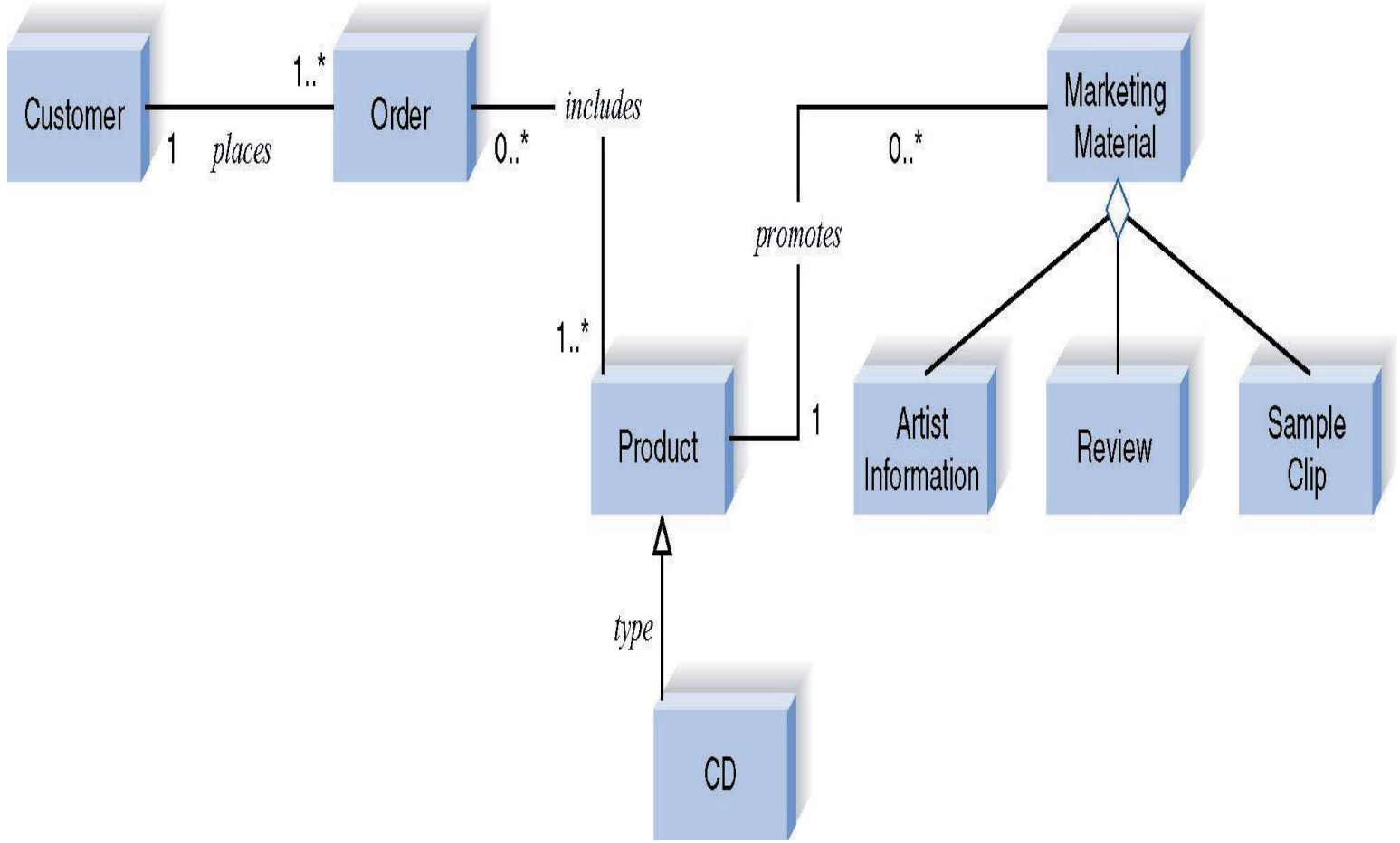
Class Diagram Concepts



Class Diagram Concepts



Class Diagram Concepts



Class Diagram Concepts