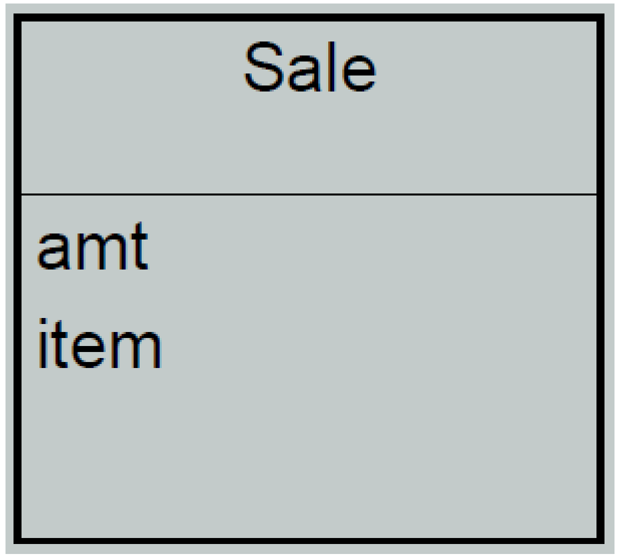
Class Model/Domain Model

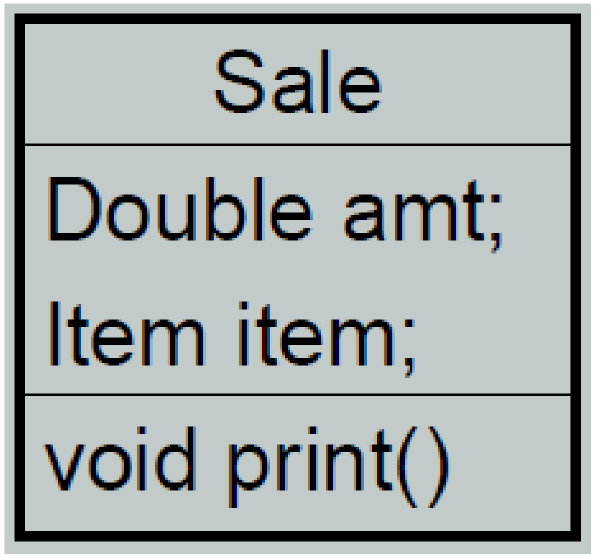
Class Diagram

What is a Domain Model?

- Illustrates meaningful conceptual classes in problem domain
- Represents real-world concepts, not software components



VS



Features of a domain model

- **Domain classes** – each domain class denotes a type of object.
- **Attributes** – an attribute is the description of a specified type in a domain class; each instance of the class separately holds a value.
- **Associations** – an association is a relationship between two (or more) domain classes that describes links between their object instances. Associations can have roles, describing the multiplicity and participation of a class in the relationship.

Steps to Create a Domain Model

- Identify candidate conceptual classes
- Draw them in a UML as a Class Diagram
- Add associations necessary to record the relationships that must be retained
- Add attributes necessary for information to be preserved

Domain classes

- Each domain class denotes a type of object. It is a descriptor for a set of things that share common features. Classes can be:-
    - *External entities*
    - *Things*
    - *Occurrences or events*
    - *Roles*
    - *Organizational units*
    - *Places*
    - *Structures*

Identifying Classes

- Identify, classify, and define objects, ideas, and events as **Classes** by examining the usage scenarios developed in the use case model, glossary, and other diagrams.
- Methods: -
  - Analyze the Requirements
  - Use Noun-Verb Analysis
  - Domain Knowledge
  - Identify Responsibilities
  - Identify Relationships

Identifying Classes

1. Analyze the Requirements:

- Examine use case descriptions to look for key entities in the problem domain that retain critical information to the system's functionality.

2. Use Noun-Verb Analysis:

- **Nouns as Classes:** Identify nouns in the requirements documentation, user stories, or use case descriptions. These nouns often represent potential classes.
- **Verbs as Methods:** Identify verbs associated with these nouns, as they often represent potential methods or responsibilities of the classes.

Identifying Classes

3. Domain Knowledge:

- Use your knowledge of the domain to identify important concepts, entities, and relationships that should be represented as classes.
- Engage with domain experts to ensure that you are capturing all relevant entities and their relationships.

4. Identify Responsibilities:

- Ensure that each class has a single, well-defined responsibility. This helps in identifying cohesive classes.
- Determine the roles that different entities play in the system and create classes accordingly.

5. Identify Relationships:

- **Aggregation and Composition:** Look for whole-part relationships (e.g., a car has an engine) that can help identify classes and their relationships.
- **Inheritance:** Identify generalization-specialization relationships (e.g., a dog is a type of animal) to determine superclass and subclass structures.

Step-by-step Identification of Classes

Use Case: “A customer can browse products, add products to a shopping cart, and place an order.”

Step 1: Examine use case descriptions to look for key entities in the problem domain that retain critical information to the system’s functionality.

- Key Entities: Customer, Product, Shopping Cart, Order

Step-by-step Identification of Classes

Use Case: "A customer can browse products, add products to a shopping cart, and place an order."

Step 2: Identify nouns in the requirements documentation, user stories, or use case descriptions. These nouns often represent potential classes.

- Nouns: Customer, Product, Shopping Cart, Order

Step 3: Identify verbs associated with these nouns, as they often represent potential methods or responsibilities of the classes.

- Verbs: Browse, Add, Place

Step-by-step Identification of Classes

Use Case: “A customer can browse products, add products to a shopping cart, and place an order.”

Step 4: Use your knowledge of the domain to identify important concepts, entities, and relationships that should be represented as classes.

- Common entities in e-commerce: Customer, Products, Categories, Shopping Cart, Orders, Payments, Shipping, Reviews, Promotion,…

Step-by-step Identification of Classes

Use Case: “A customer can browse products, add products to a shopping cart, and place an order.”

Step 5: Ensure that each class has a single, well-defined responsibility. This helps in identifying cohesive classes. Determine the roles that different entities play in the system and create classes accordingly.

- Customer: Manages customer information and actions.
- Product: Represents items available for purchase.
- Shopping Cart: Manages the collection of products selected by the customer.
- Order: Represents the completed purchase transaction.

Step-by-step Identification of Classes

Use Case: “A customer can browse products, add products to a shopping cart, and place an order.”

Step 6: Identify relationship such as aggregation, composition, and inheritance.

- Aggregation: Shopping Cart contains multiple Products.
- Aggregation: Order contains multiple Products
- Composition: Customer has multiple Orders.

Class selection characteristics

- Use six selection characteristics to select classes for the analysis model: -
  1. Retained information - if information about the class must be remembered so that the system can function
  2. Needed services - the potential class must have a set of identifiable operation that is needed by the system to operate
  3. Multiple attributes – a class with a single attribute may, in fact, be useful during design, but is probably better represented as an attribute of another class during the analysis activity

Class selection characteristics

- Use six selection characteristics to select classes for the analysis model: -

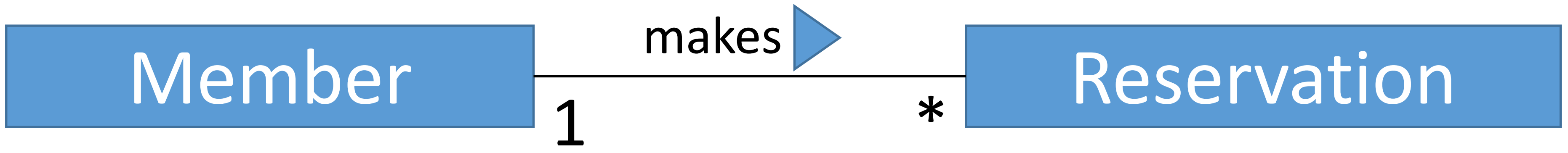
4. Common attributes - A set of attributes can be defined for the potential class and these attributes apply to all instances of the class.
5. Common operations - A set of operations can be defined for the potential class and these operations apply to all instances of the class.
6. Essential requirements - External entities that appear in the problem space and produce or consume information essential to the operation of the system

Class Names

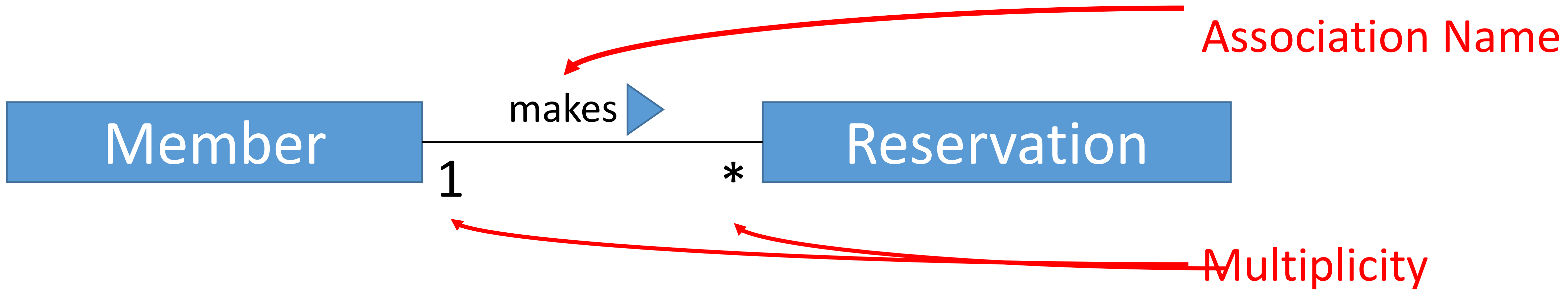
- **Class Name** creates the vocabulary of our analysis
  - Use nouns as class names
  - Verbs can also be made into nouns
- Use pronounceable names
  - If you cannot read aloud, it is not a good name
- Use CamelCasing
  - E.g., CardReader rather than CARDREADER or card_reader
- Avoid obscure, ambiguous abbreviations
  - E.g., is TermProcess something that terminates or something that runs on a terminal?
- Try *not* to use digits within a name, such as CardReader2

Associations

- Association is the relationship between 2 separate classes ("has a").
- For example:
  - Member class is associated to Reservation.

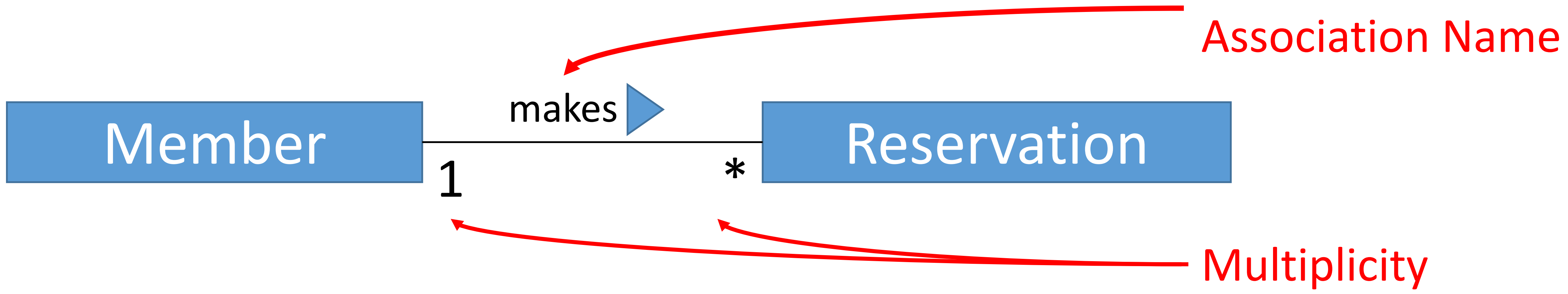


Association Name and Multiplicity



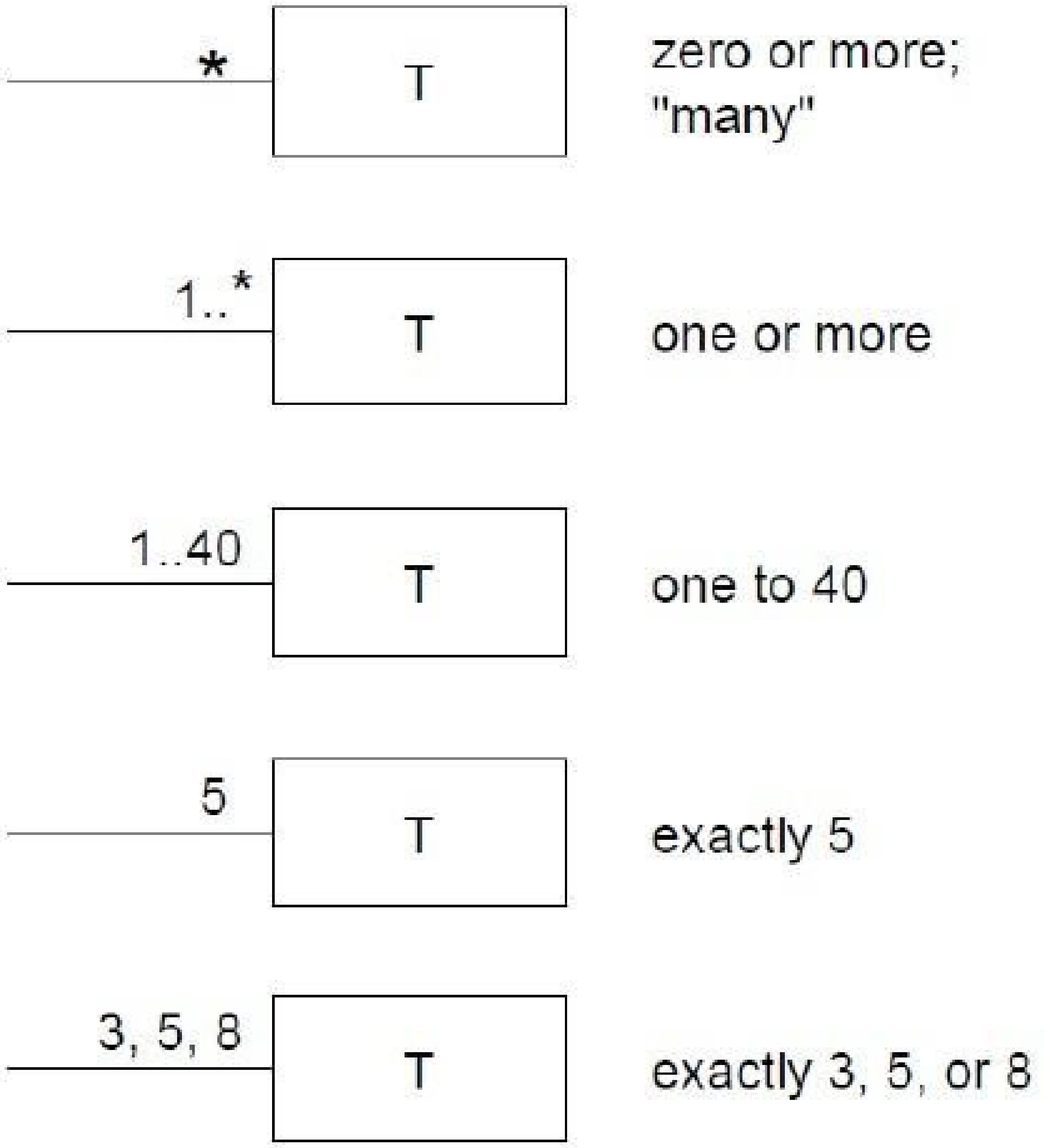
- Association can be further defined to indicate multiplicity.
- Multiplicity indicate the number of instances exist in an association
- For example:
  - Each Member ***makes*** none or many Reservation; Each Reservation belongs to 1 Member

Association Name and Multiplicity



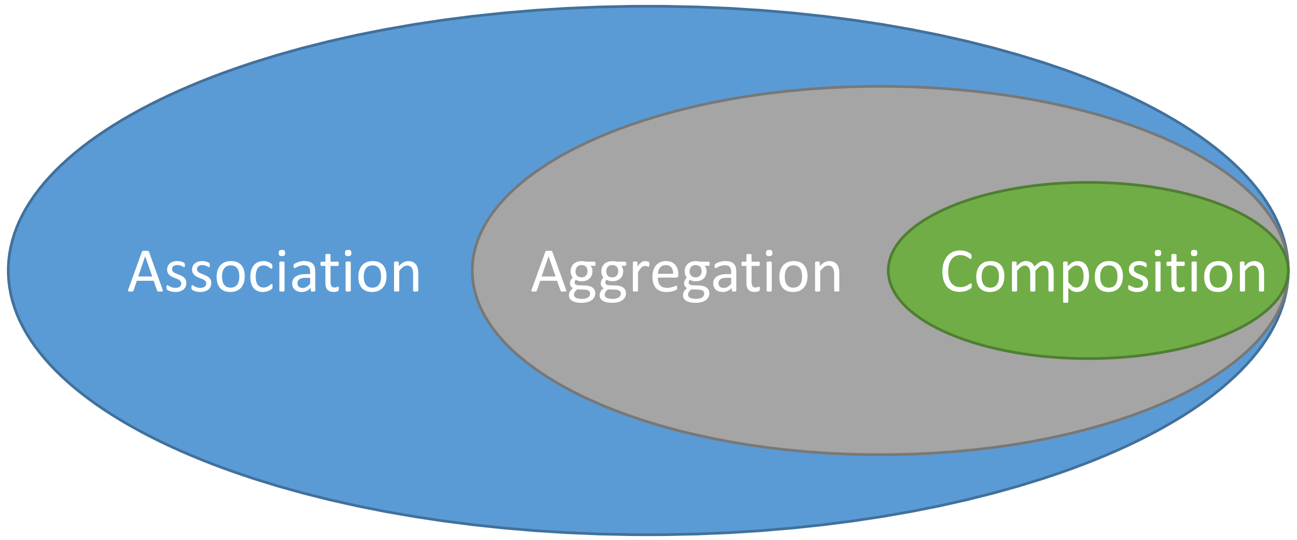
- Use verbs as the Association Name
- Use arrow to aid the reading of the Association Name

Examples of Multiplicity

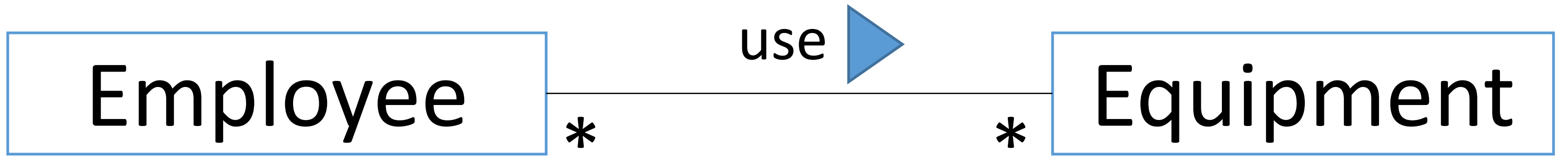


Association, Aggregation, Composition

- **Composition** and **Aggregation** are the two forms of association (object made up of another object).

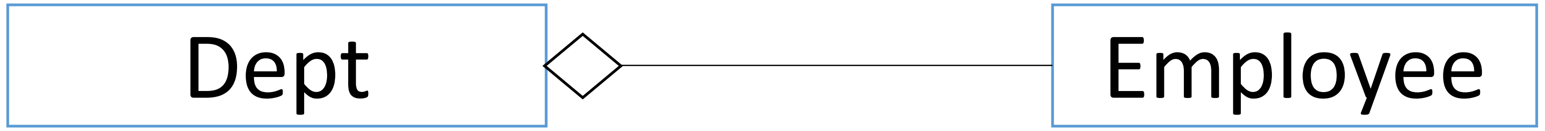


Association



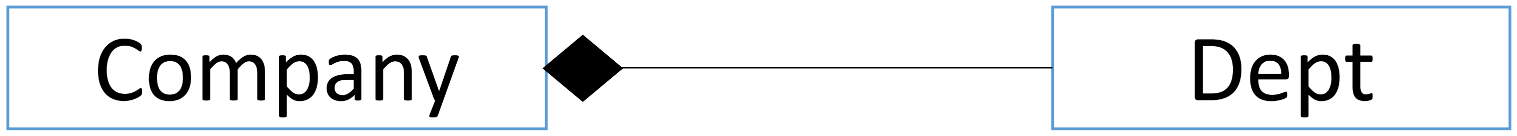
- **Association**
  - Relationship among objects
  - The objects merely aware of each other
  - One object can call the methods of the other object
  - The lifecycle of both objects are independent of each other

Aggregation



- **Aggregation**
  - Special type of Association
  - One object "containing" or "owning" the other object
  - "has a" relationship
  - The lifecycle of both objects are independent of each other
    - E.g. If I close the Dept, the Employee can still exist
    - Employee can work for many departments (non-unique relationship)

Composition



- **Composition**
  - A special type of Association, “Strong Association”
  - “part of” relationship
  - The containing object, the Company, exclusively “own” the Dept object
  - When containing object ended it’s lifecycle, the Dept object end with in

Implementation in Java

- Association
  - One object aware of the other object by maintaining a reference to the object
  - Creation and destruction of the reference object is not the major concern
- Aggregation
  - Containing object has the contained object(s) as one of the object properties
  - **Not responsible** for the creation and destruction of the contained object(s)
- Composition
  - Containing object has the contained object(s) as one of the object properties
  - **Responsible** for the creation and destruction of the contained object(s)

Exercise

- Identify the possible relationship between the following classes
    - Student, Instructor
    - Person, Head
    - Classroom, Desk
    - School, Programme
    - Staff, Task
    - Book, Page

Common Associations

- A is subpart/member of B. (SaleLineItem-Sale)
- A uses or manages B. (Cashier –Register, Pilot-airplane)
- A communicates with B. (Student -Teacher)
- A is transaction related to B. (Payment -Sale)
- A is owned by B. (Plane-Airline)
- A is an event related to B. (Sale-Store)

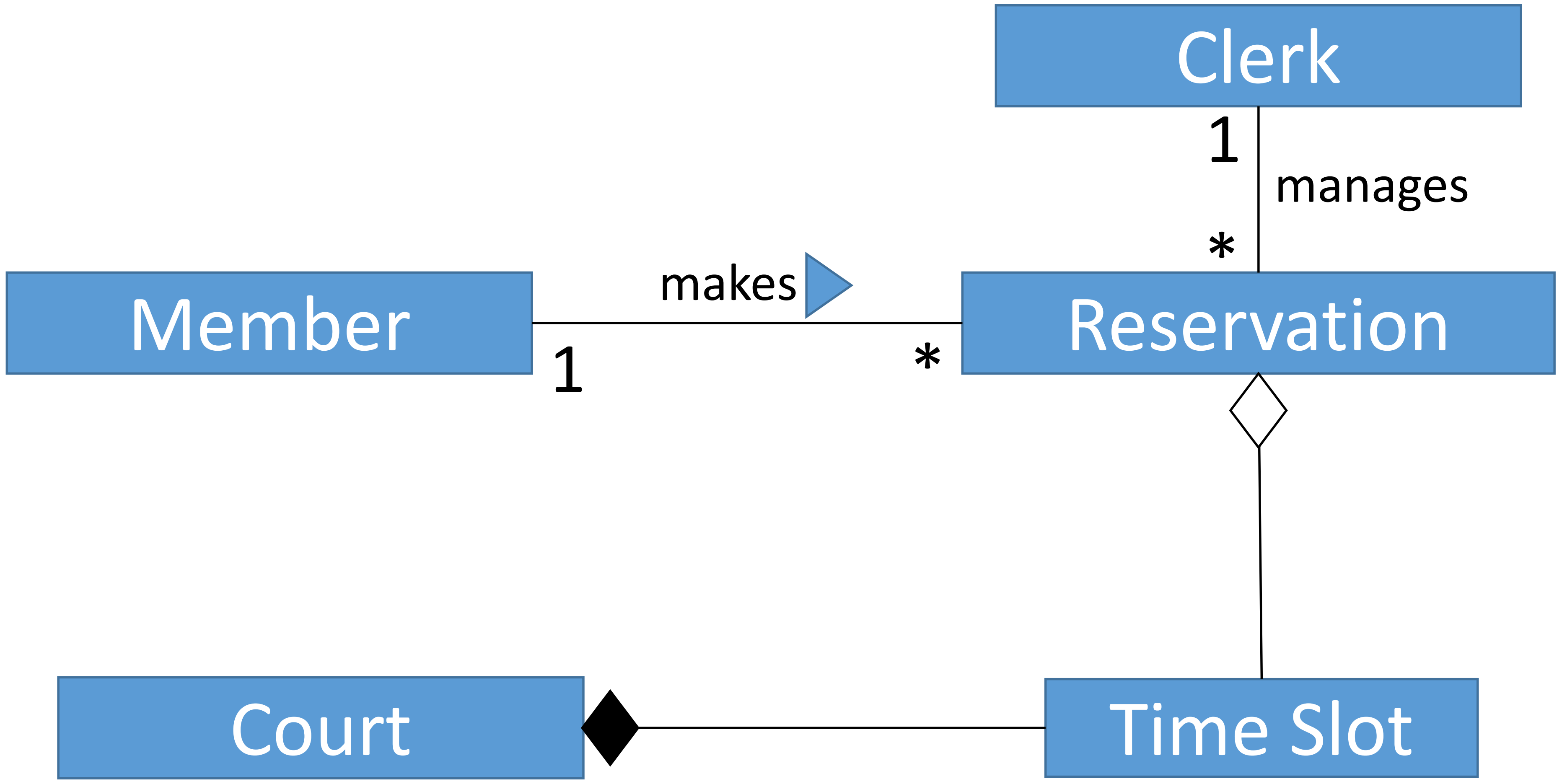
Badminton Court Reservation Process

- In the badminton court reservation system, the process begins when a player, who must be a registered member, contacts the stadium to book a court. Only members are allowed to make reservations. The player communicates their preferred date and time for the reservation to the clerk. Additionally, the player can inquire about available time slots for that day.
- The clerk responds with the available time slots. A time slot represents a specific period during which a badminton court is reserved or available for reservation. Based on this information, the player selects and confirms their preferred time slot for the reservation. Sometimes, a reservation may include multiple time slots to allow the member to play for an extended period.
- Once the player confirms their preferred time slot(s), the clerk updates the reservation records accordingly, ensuring that a badminton court is reserved for the specified time slot(s).

Reservation Case Study

- Key entities and nouns
  - Player/member, court, reservation, preferred date and time, clerk, available time slot
- Associated verbs
  - contact, book/reserve, communicate, inquire, select and confirm, update
- Domain knowledge
  - Ignored
- Responsibilities and roles
  - Member: Represents member information and operation
  - Reservation: Records of reservation make by members at specific time for specific court.
  - Preferred date and time: Single attribute. Can be recorded in Reservation as an attribute.
  - Clerk: Represents a person who operate the system.
  - Available time slot: Represents a specific period during which a court is reserved or available for reservation
  - Court: Represent a badminton court and its status. We may also ignore if it contain only single attribute.
- Relationship
  - A Member can make multiple Reservations.
  - A Clerk manages Reservations.
  - A Reservation can include multiple Time Slots.
  - A Court can have multiple Time Slots.

Classes with Relationship



Class Attribute

- Attributes describe an object
- Study use case to select the attribute belonging to a particular class
  - What data items fully define the class in the domain problem?
  - What information needs to be retained for the system to function as required?

Class Attribute

- In the badminton court reservation case study, which attributes are required for the Reservation class? TimeSlot class?

- Some of the attributes are the compounded attribute that you may or may not want to break it down to elementary attributes, and vice versa.
- In the modeling for requirement analysis, data type and access scope of the attributes are not required