OBJECT-ORIENTED SYSTEMS DEVELOPMENT: USING THE UNIFIED MODELING LANGUAGE

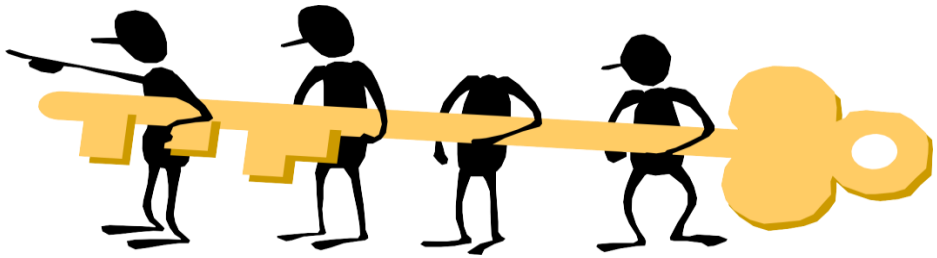
Identifying Object Relationships, Attributes, and Methods

GOALS

- Analyzing relationships among classes
- Identifying association
- Association patterns
- Identifying super- and subclass hierarchies
- Identifying aggregation or a-part-of compositions
- Class responsibilities
- Identifying attributes and methods by analyzing use cases and other UML diagrams

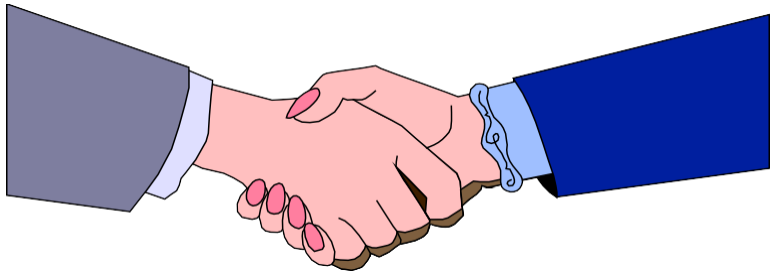
Objects contribute to the behavior of the system by collaborating with one another.

—Grady Booch



In OO environment, an application is the interactions and relationships among its domain objects

All objects stand in relationship to others, on whom they rely for services and controls

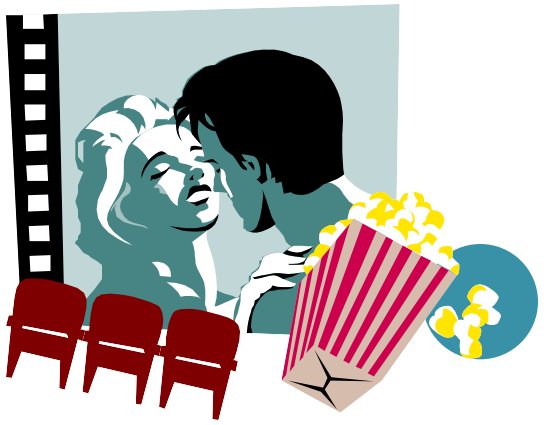


INTRODUCTION

- Objects do not exist in isolation but interact with each other.
- The object stands in relationship with each other for services and control.

OBJECTS RELATIONSHIPS

- Three types of relationships among objects are:
  - *Association*
    - How objects are associated?
  - *Super-sub structure* (also known as *generalization hierarchy*).
    - How objects are organized in subclasses and super classes?
  - Aggregation and a-part-of structure.
    - What is the composition of complex classes?

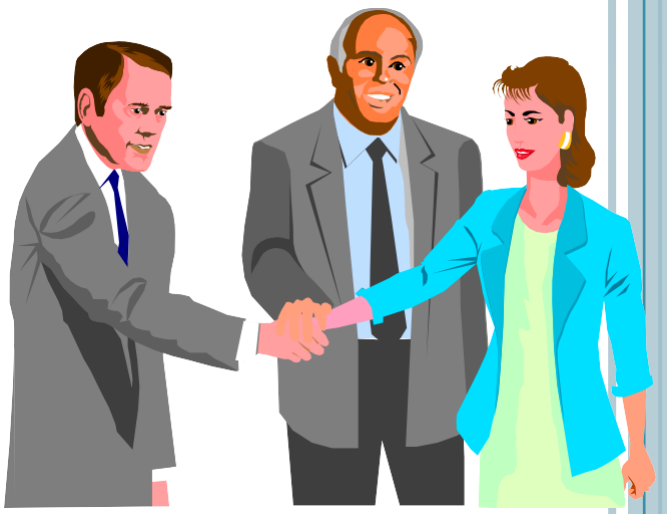


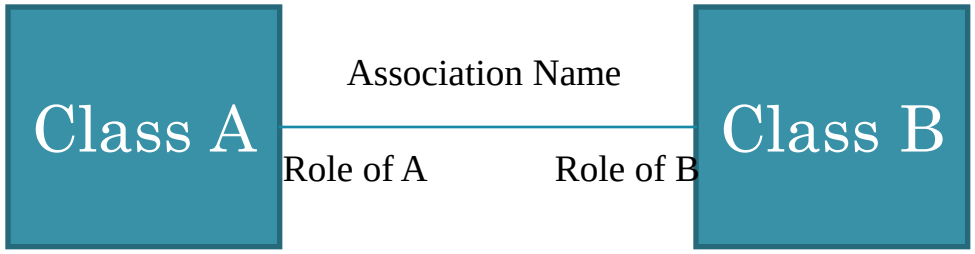
CONT..

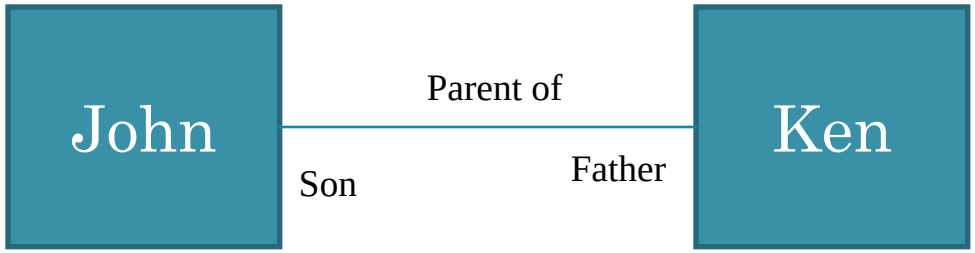
- In general the relationship between objects is called association.
  - E.g. a customer places an order for soup. So the order is an association between objects customer and soup.
- The hierarchical or super-sub relation allows the sharing of properties or inheritance.
  - E.g. car, bike, truck (subclasses) are vehicles (superclass)
- A part-of structure means organizing components in a bigger object.
  - E.g. walls, windows, doors are part of a bigger object: a building.

ASSOCIATIONS

- A reference from one class to another is an association.
- Basically a dependency between two or more classes is an association.
- For example, Jackie *works for* John







ASSOCIATIONS (CON'T)

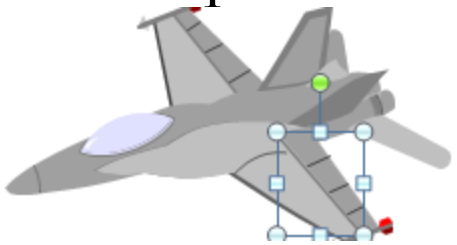
- Some associations are implicit or taken from general knowledge.
- It represents a physical or conceptual connection between two or more objects.

IDENTIFYING ASSOCIATIONS

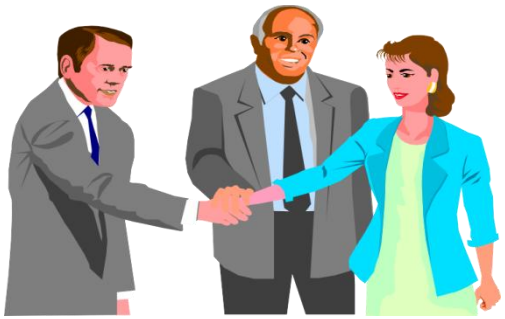
- It begins by analyzing the interactions between the classes.
- Any dependency is an association- so you must examine responsibilities to determine dependencies.
- An object lacking at knowledge or capacity to perform a specific task must delegate it to another.
- Following are the questions that can help us to identify associations:
  - Is the class capable of fulfilling the required task itself?
  - If not, what does it need?
  - From what other class can it acquire what it needs?
- Extract all candidates' associations from problem statement- it can be refined later.
- Distinguish aggregations from associations (it depends on the problem statement).

GUIDELINES FOR IDENTIFYING ASSOCIATIONS

- Association often appears as a verb in a problem statement and represents relationships between classes.
- For example a pilot *can fly* planes.



- Association often corresponds to verb or prepositional phrases such as *part of*, *next to*, *works for*, *contained in*, etc.

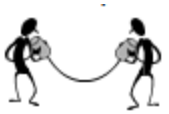


- A reference from one class to another is an association. Some are implicit or taken from general knowledge.

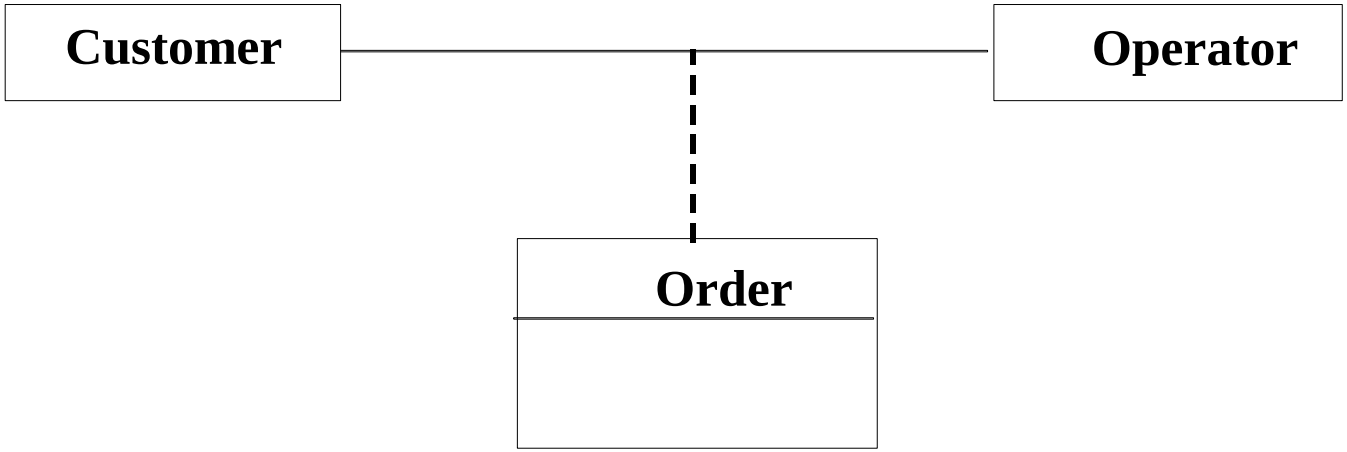
COMMON ASSOCIATION PATTERNS

- Common association patterns include:
  - Location Association: *next To*, *part of*, *contained in, ingredient of etc. The a-part-of relationship is a special type of association.*
  - For example cheese is an *ingredient of* the French soup.

COMMON ASSOCIATION PATTERNS (CON'T)



- Communication association—*talk to*, *order to*
  - For example, a customer places an order with an operator person.



ELIMINATE UNNECESSARY ASSOCIATIONS

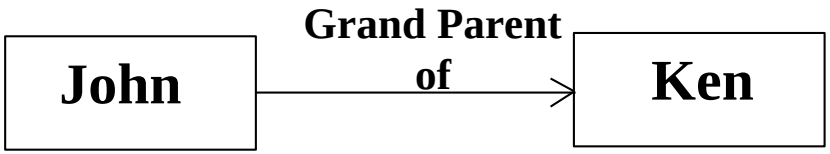
- *Implementation association*. Defer implementation-specific associations to the design phase.
  - Concerned with the implementation or design of the class within certain PL or development environment.
- *Ternary associations*. *Ternary* or n-ary *association* is an association among more than two classes.
  - They complicate the representation, hence restate them as binary associations.

ELIMINATE UNNECESSARY ASSOCIATIONS (CON'T)

- *Directed actions* (derived) *associations* can be defined in terms of other associations.
- Since they are redundant you should avoid these types of association.

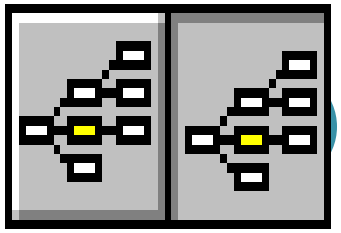
ELIMINATE UNNECESSARY ASSOCIATIONS (CON'T)

- Grandparent of Ken can be defined in terms of the parent association.



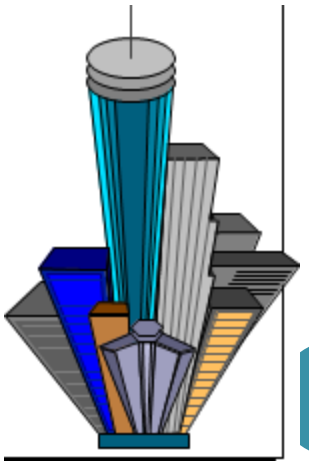
SUPER CLASS-SUBCLASS RELATIONSHIPS

- Its an identification of super-sub relation among classes known generalization hierarchy.
- A class is a part of a hierarchy of classes, where the top class is the most general one.
- The main advantage is that we can build on what we have, and reuse we already have.
- Recall that at the top of the class hierarchy is the most general class, and from it descend all other, more specialized classes.
- Sub-classes are more specialized versions of their super-classes.



GUIDELINES FOR IDENTIFYING SUPER-SUB RELATIONSHIPS: TOP-DOWN

- Look for noun phrases composed of various adjectives on class name. Avoid excessive refinement.
- Example, Military Aircraft and Civilian Aircraft.
- Only specialize when the sub classes have significant behavior.
  - E.g. a phone operator can be represented as a cook as well as a clerk or manager.



GUIDELINES FOR IDENTIFYING SUPER-SUB RELATIONSHIPS: BOTTOM-UP

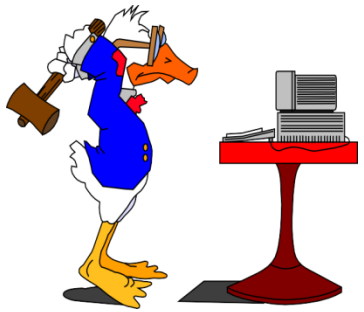
- Look for classes with similar attributes or methods.
- Group them by moving the common attributes and methods to super class.
- Do not force classes to fit a preconceived generalization structure.

GUIDELINES FOR IDENTIFYING SUPER-SUB RELATIONSHIPS: REUSABILITY

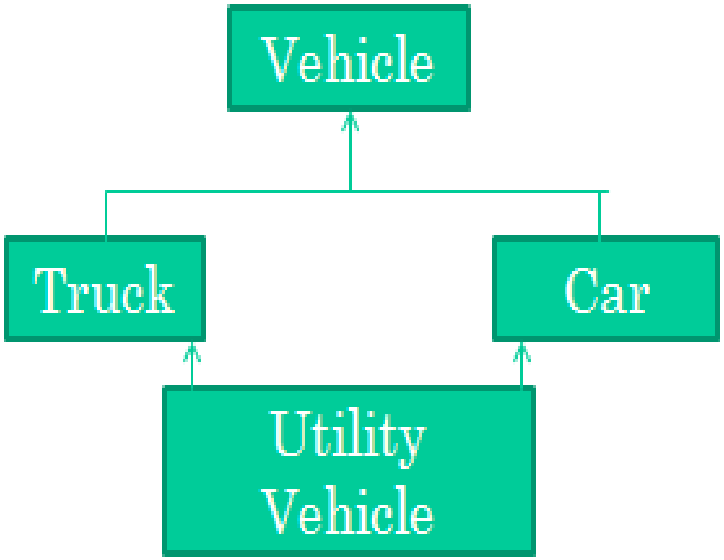
- Move attributes and methods as high as possible in the hierarchy.
- At the same time do not create very specialized classes at the top of hierarchy.
- This balancing act can be achieved through several iterations.

- This also ensures that you design objects that can be reused in another application.

GUIDELINES FOR IDENTIFYING SUPER-SUB RELATIONSHIPS: MULTIPLE INHERITANCE



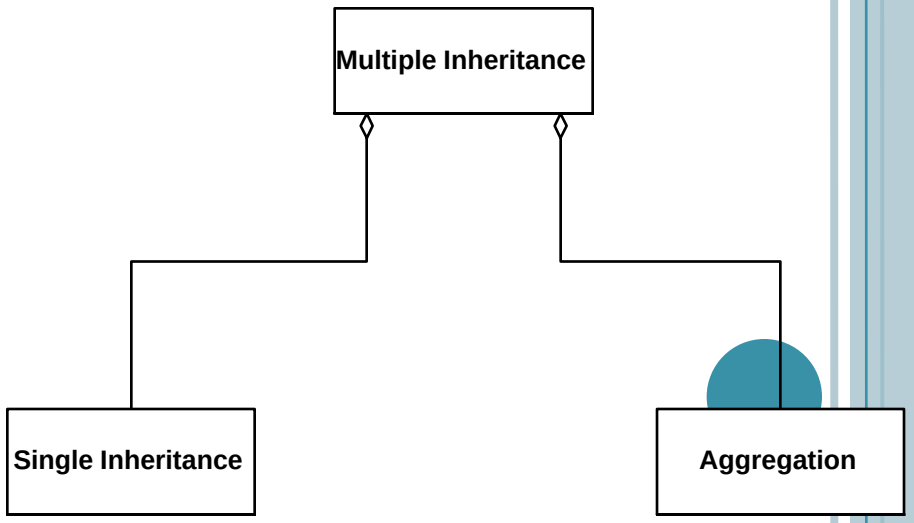
- Avoid excessive use of multiple inheritance.
  - E.g. when several ancestors define the same method.
- It is also more difficult to understand programs written in multiple inheritance system.



If stop method in both truck and car then which method to call from UtilityVehicle

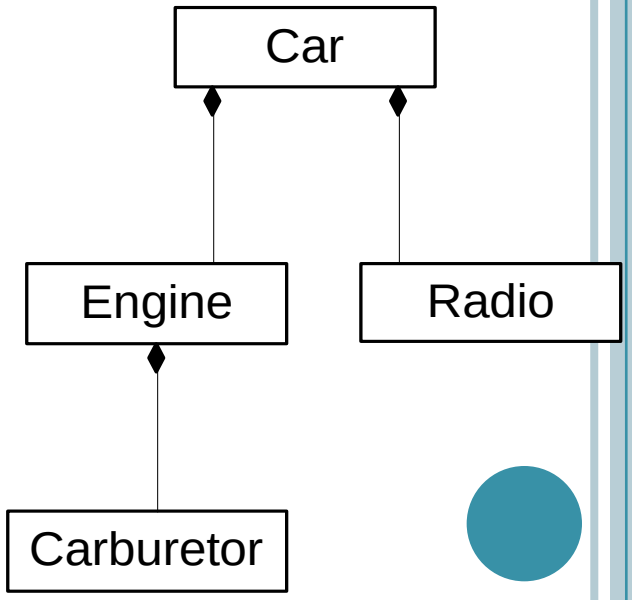
MULTIPLE INHERITANCE (CON'T)

- One way to achieve the benefits of multiple inheritance is to inherit from the most appropriate class and add an object of other class as an attribute.
- In essence, a multiple inheritance can be represented as an aggregation of a single inheritance and aggregation. This meta model reflects this situation.



A-PART-OF RELATIONSHIP - AGGREGATION

- *A-part-of relationship*, also called *aggregation*, represents the situation where a class consists of several component classes.
- This does not mean that the class behaves like its parts.
- For example, a car consists of many other classes, one of them is a radio, but a car does not behave like a radio.



A-PART-OF RELATIONSHIP - AGGREGATION (CON'T)

- Two major properties of a-part-of relationship are:
  - Transitivity
  - Antisymmetry
- Transitivity :
  - If A is part of B and B is part of C, then A is part of C.
  - For example, a carburetor is part of an engine and an engine is part of a car; therefore, a carburetor is part of a car.
- Antisymmetry :
  - If A is part of B, then B is not part of A.
  - For example, an engine is part of a car, but a car is not part of an engine.

A-PART-OF RELATIONSHIP - AGGREGATION (CON'T)

- A clear distinction between **the part** and **the whole** can help us decide where responsibilities for certain behavior must reside?
  - Does the part class belong to problem domain?
  - Is the part class within the system's responsibilities?
  - Does the part class capture more than a single value?
  - If it captures only a single value, then simply include it as an attribute with the whole class
  - Does it provide a useful abstraction in dealing with the problem domain?

A-PART-OF RELATIONSHIP PATTERNS

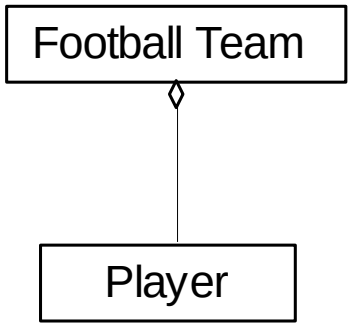
- Assembly
  - An assembly- Physical whole is constructed from physical parts
  - An assembly is constructed from its parts and an assembly-part situation physically exists.
  - For example, a French soup consists of onion, butter, flour, wine, French bread, cheddar cheese, etc.
  - Computer is assembly of floppy drive, motherboard, RAM etc.
  - House is assembly of bricks, concrete etc.

A-PART-OF RELATIONSHIP PATTERNS

- Container
  - A physical whole encompasses but is not constructed from physical parts;
  - e.g. a house can be considered as a container for furniture and appliances.
  - A case such as course-teacher situation, where a course is considered as a container. Teachers are assigned to specific courses.

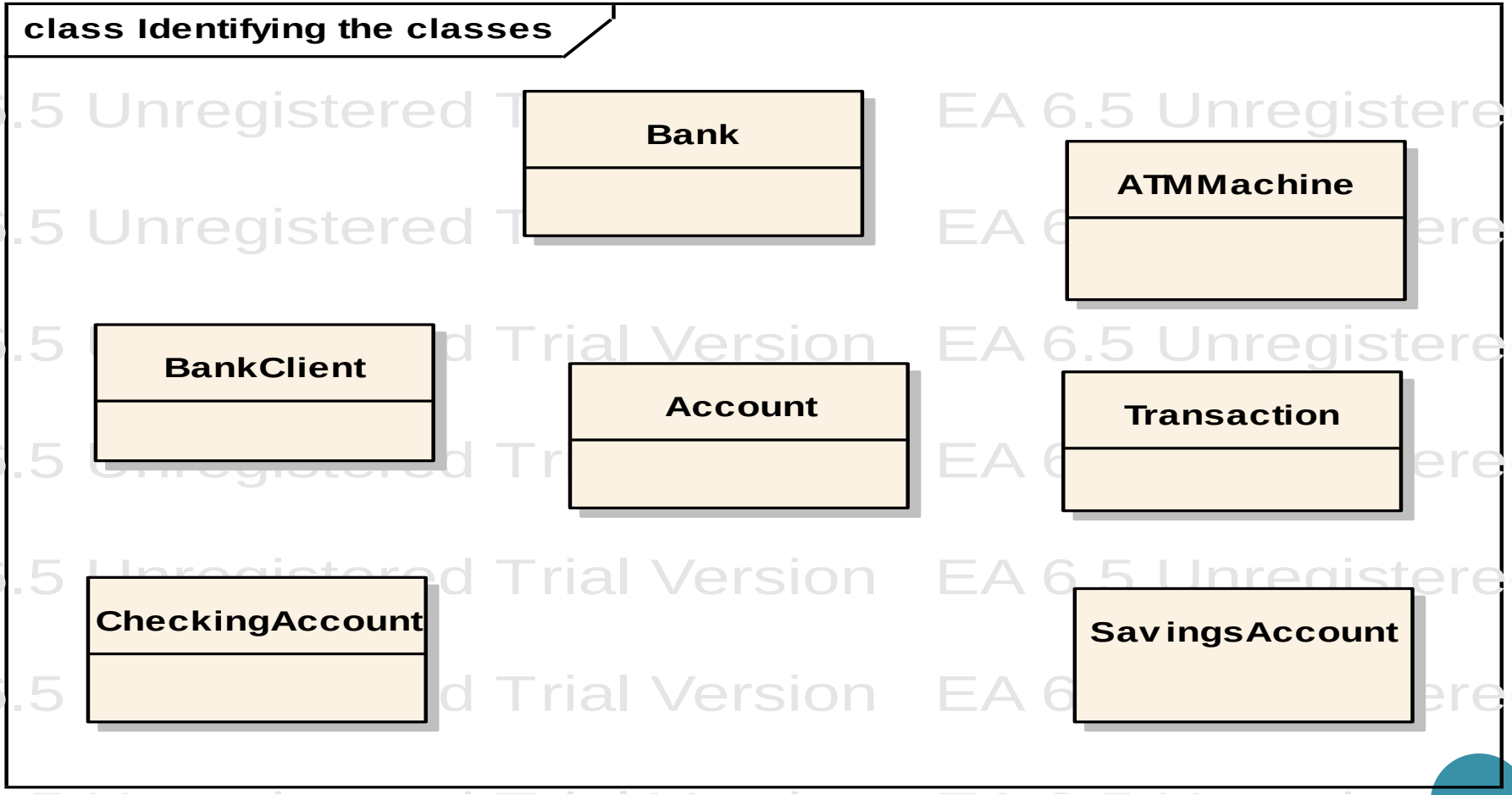
A-PART-OF RELATIONSHIP PATTERNS

- Collection-Member
  - A conceptual whole encompasses parts that may be physical or conceptual.
  - A soccer team is a collection of players.

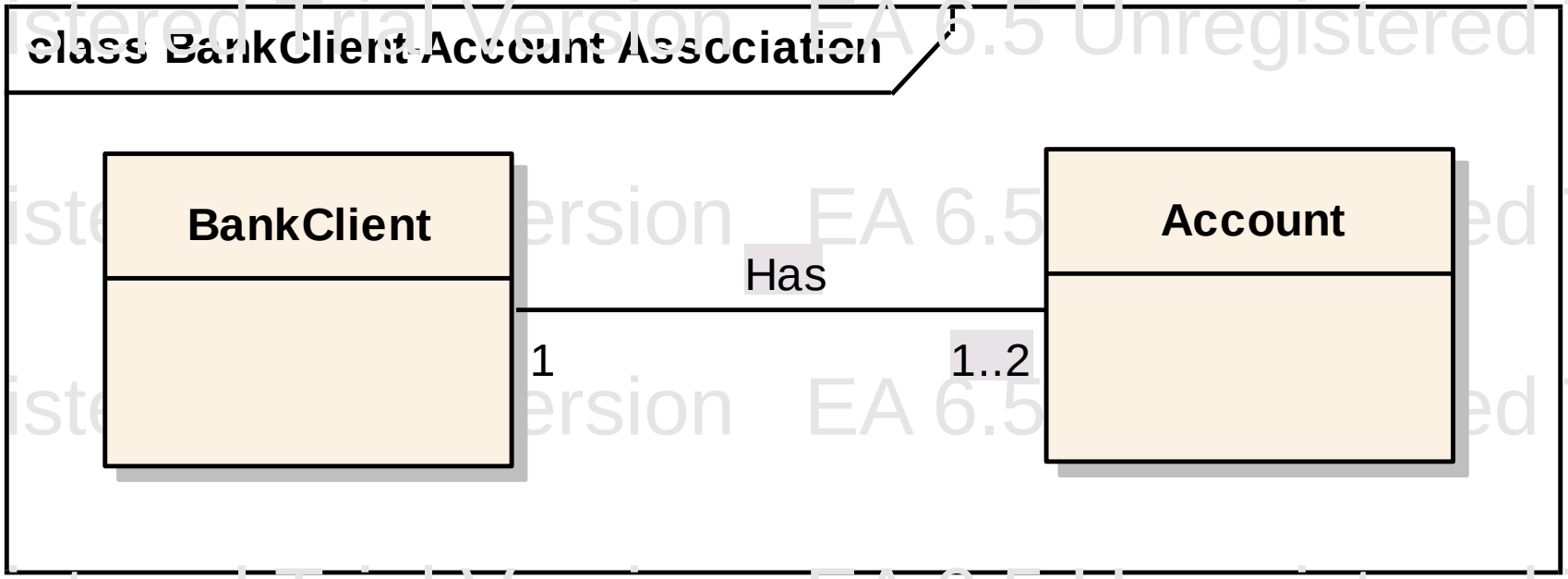


  - Relationship analysis for the VIA-NET Bank ATM System.

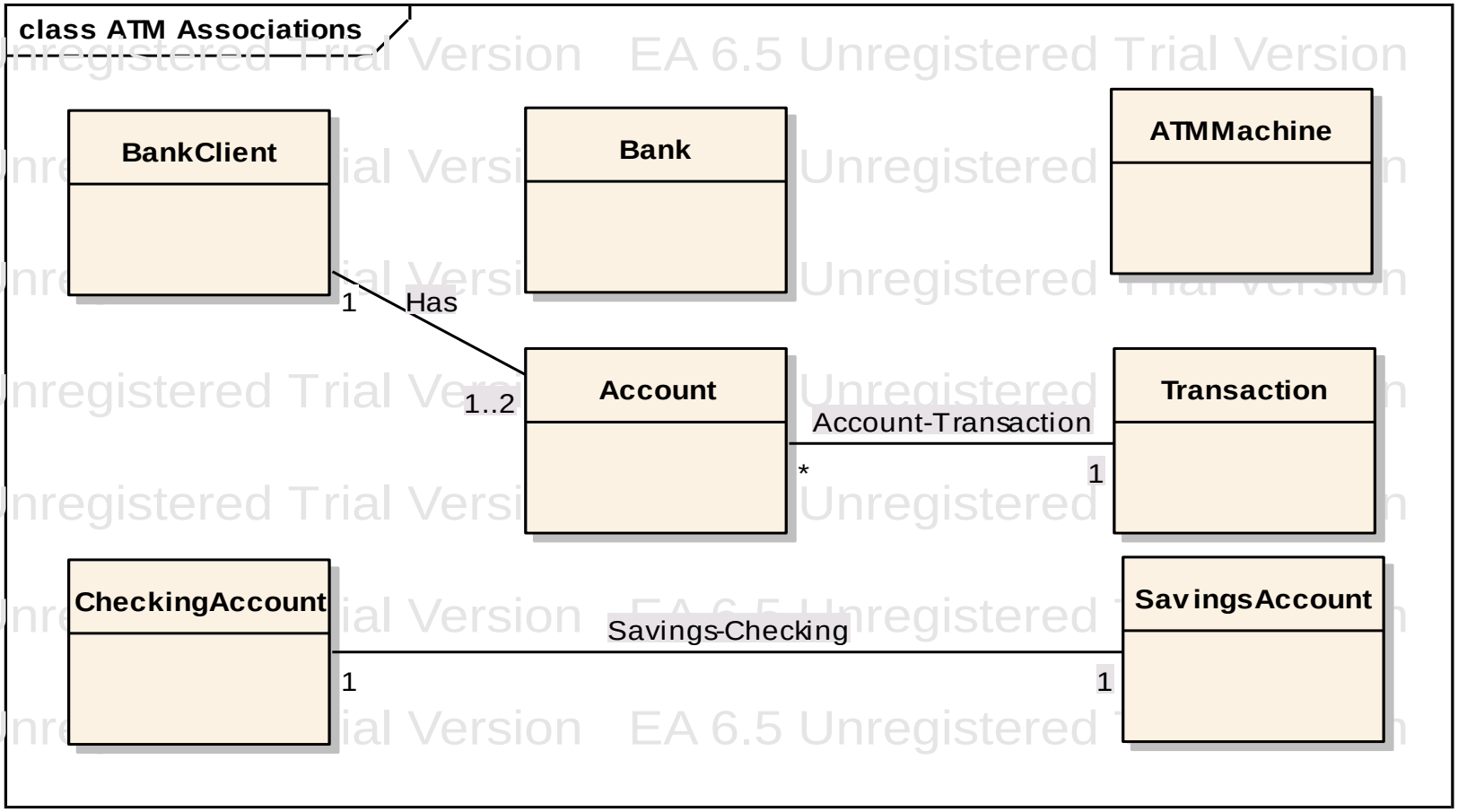
VIA-NET Bank ATM System : Relationship analysis
Identifying the Classes



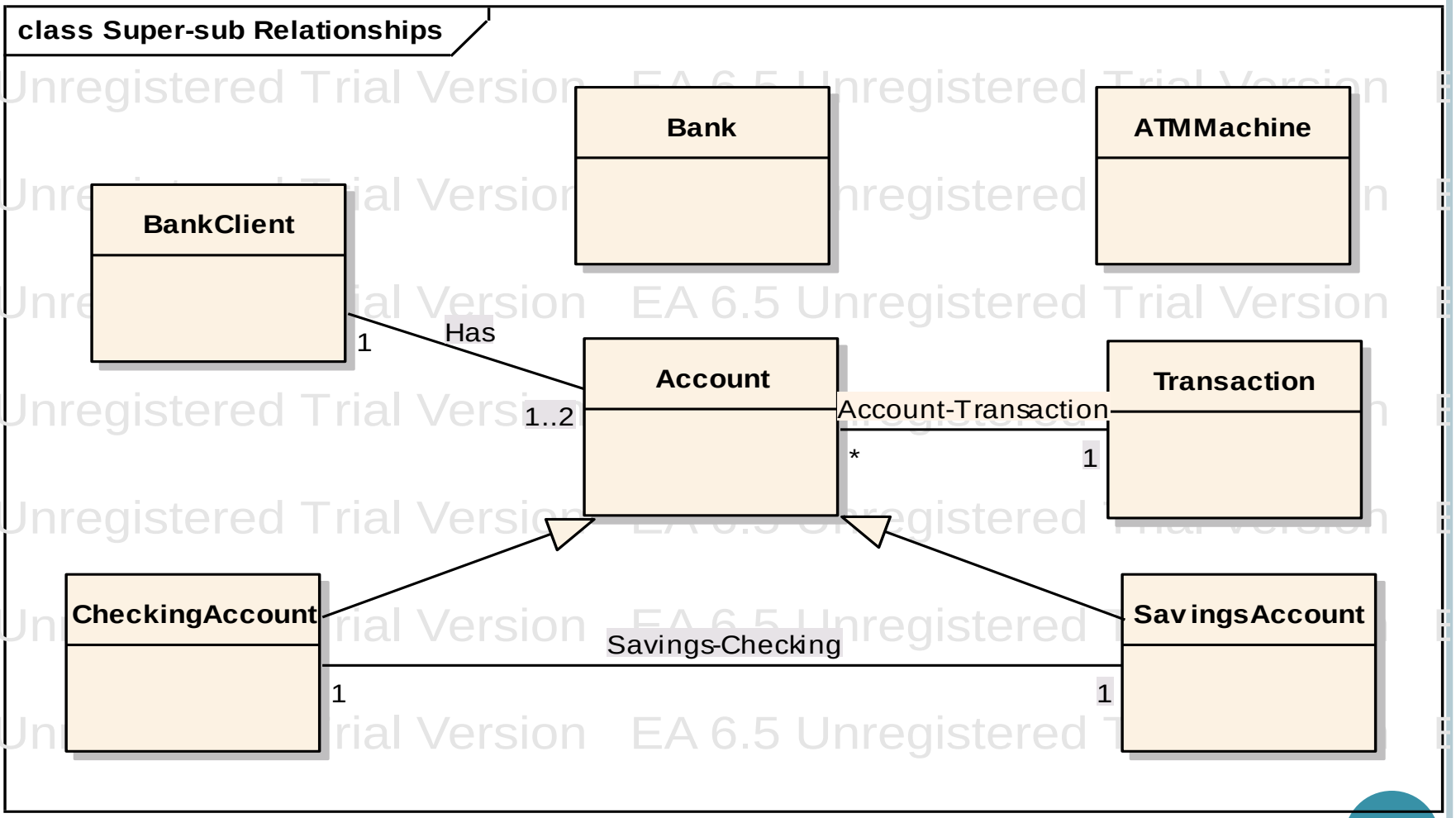
Relationship



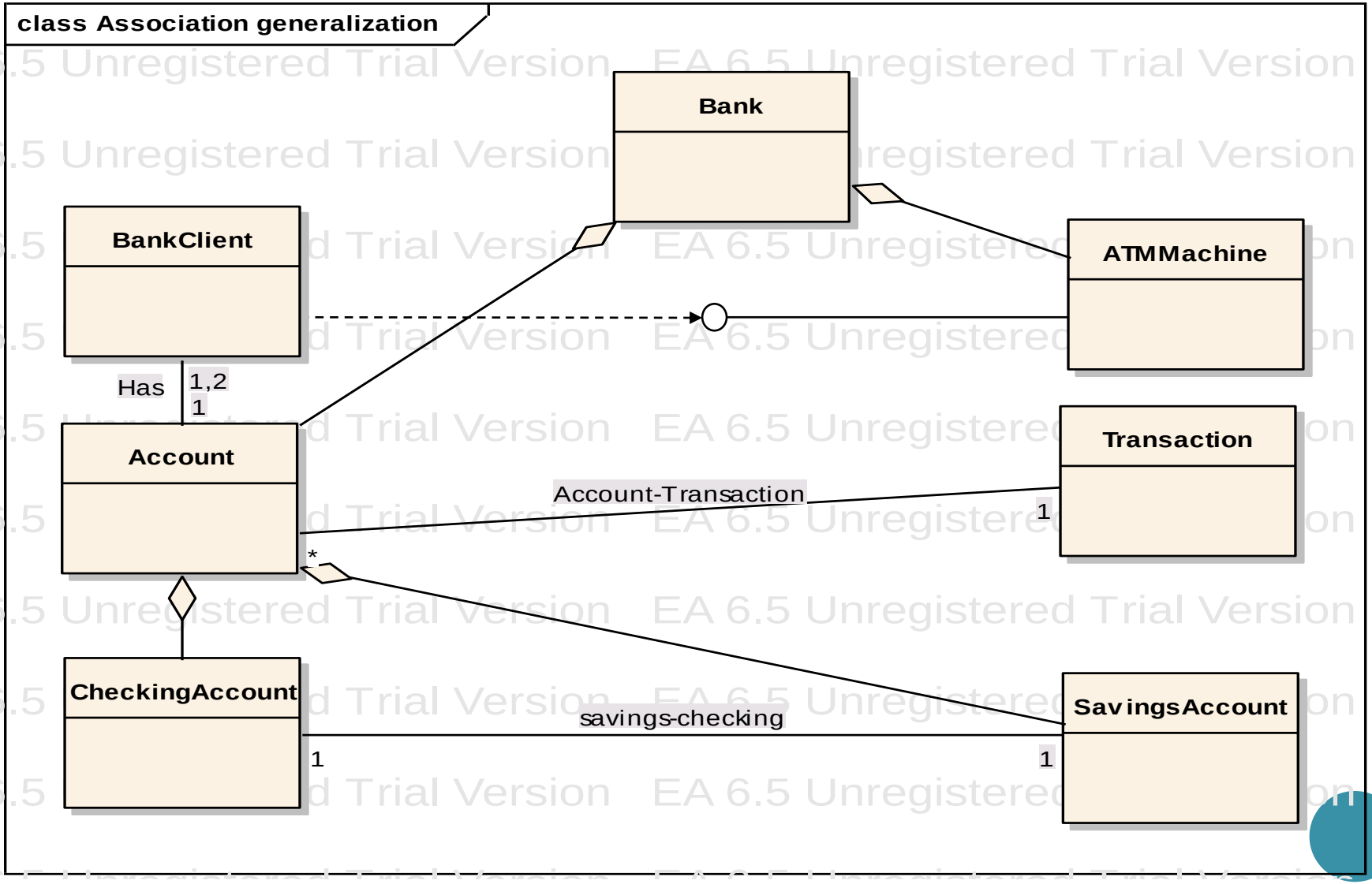
Associations



Super-sub relationships



Association, generalization, aggregation and interface



CLASS RESPONSIBILITY: IDENTIFYING ATTRIBUTES AND METHODS

- Identifying attributes and methods, like finding classes, is a difficult activity.
- The use cases and other UML diagrams will be our guide for identifying attributes, methods, and relationships among classes.
- Identifying the class's attributes starts with understanding the systems responsibilities.
- And systems responsibilities can be identified by developing use cases.
- Following questions can help in identifying the responsibilities:
  - What information about an object should we keep track of?
  - What services must a class provide?
- The answer of first question will help to identify the attributes of the class.
- And that of second question allow us to identify methods.

IDENTIFYING CLASS RESPONSIBILITY BY ANALYZING USE CASES AND OTHER UML DIAGRAMS

- Attributes can be identified by analyzing the use cases, sequence/collaboration, activity, and state diagrams.
- The basic goal is to understand what the class is responsible for knowing.

RESPONSIBILITY

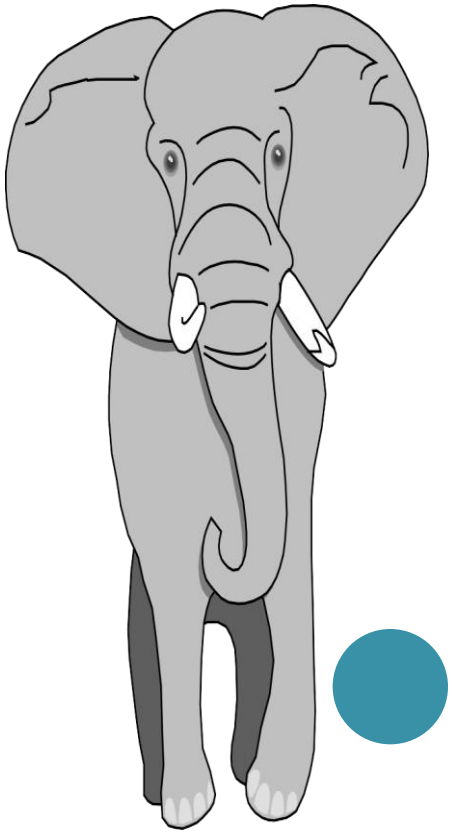
- How am I going to be used?
- How am I going to collaborate with other classes?
- How am I described in the context of this system's responsibility?
- What do I need to know?
- What state information do I need to remember over time?
- What states can I be in?

ASSIGN EACH RESPONSIBILITY TO A CLASS

- Assign each responsibility to the class that it logically belongs to.
- This also aids us in determining the purpose and the role that each class plays in the application.

OBJECT RESPONSIBILITY: ATTRIBUTES

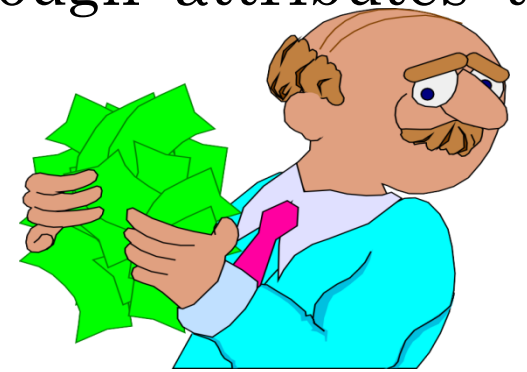
- Information that the system needs to remember



GUIDELINES FOR IDENTIFYING ATTRIBUTES OF CLASSES

- Attributes usually correspond to nouns

  followed by prepositional phrases such as *cost of* the soup
- Attributes may also corresponds to adjectives or adverbs
- Keep the class simple; state only enough attributes to define the object state



- Attributes are less likely to be fully described in the problem statement

GUIDELINES FOR IDENTIFYING ATTRIBUTES OF CLASSES

- You must draw on your knowledge of the application domain and the real world to find them

- Omit derived attributes
- For example, don't use age as an attribute since it can be derived from date of birth
- Drive attributes should be expressed as a method –

GUIDELINES FOR IDENTIFYING ATTRIBUTES OF CLASSES (CON'T)

- Do not carry discovery of attributes to excess.
- You can always add more attributes in the subsequent iterations.

CONT..

- Conclusion
  - You may think of many attributes that can be associated with the class but add only those necessary for the design at hand
  - E.g. the library Member class may have attributes such as Name, SSN, Age and Weight
    - Here the attributes Name and Weight may be important for the class Member in personal system
    - But not in the scope of this system since there is no scenario in Library Borrow Books that requires to keep track of weight and age of a member
- Defining attributes for VIA-NET bank objects

OBJECT RESPONSIBILITY: METHODS & MESSAGES

- Methods and messages are the work horses of object-oriented systems
- In O-O environment, every piece of data, or object, is surrounded by a rich set of routines called methods

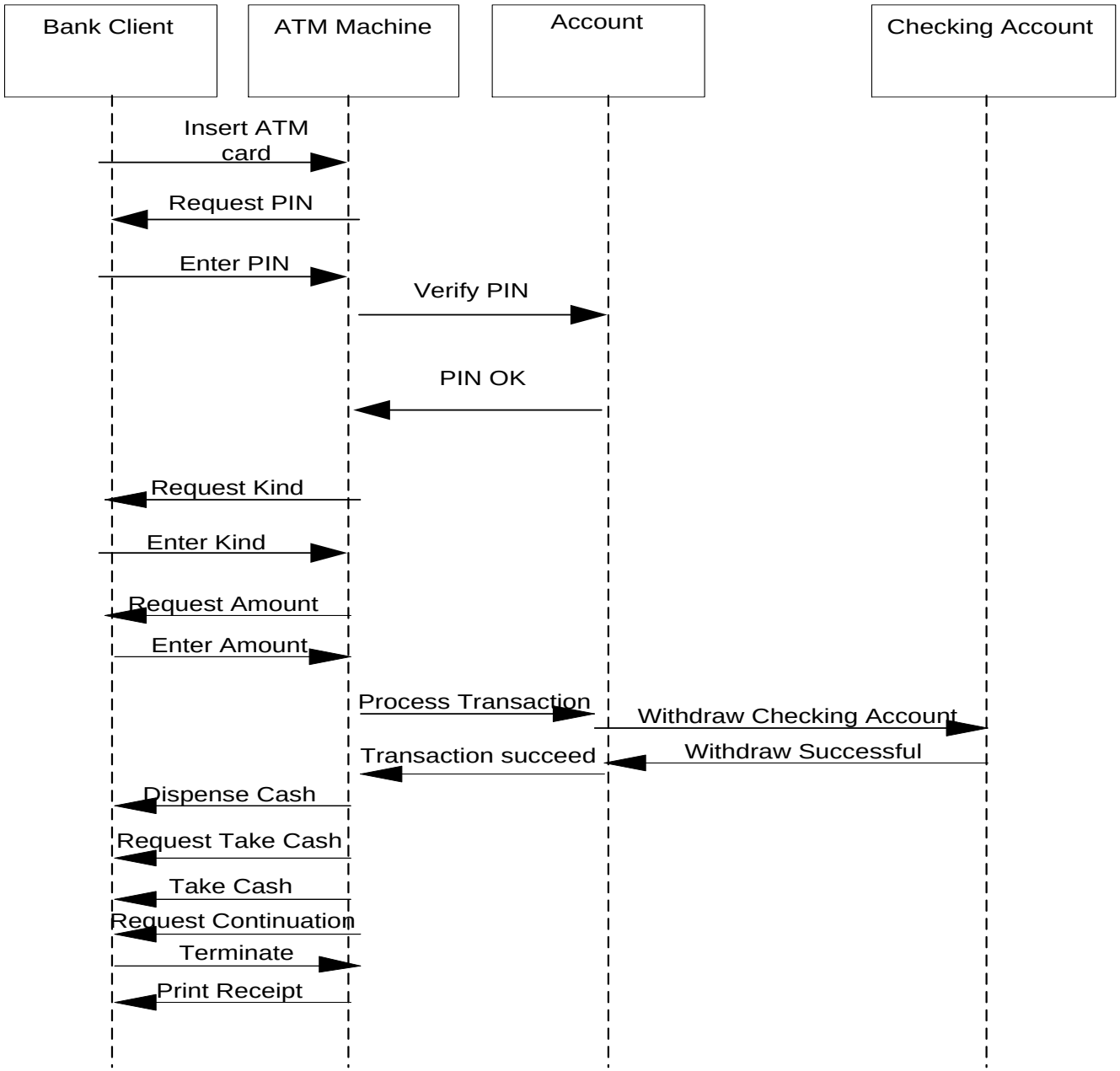
CONT..

- Methods or behavior in the OO system usually corresponds to queries about attributes (sometimes the associations) between objects
- i.e. the methods are responsible for managing the values of attributes such as query, updating, reading and writing;
  - E.g. getBalance operation returns the current balance. In the same way setBalance operation to set the value

IDENTIFYING METHODS BY ANALYZING UML DIAGRAMS AND USE CASES

- Sequence diagrams can assist us in defining the services the objects must provide
- How?
  - Every event can be considered to be an action that transmit the information
  - These actions are operations that the objects must perform
  - Methods like attributes also can be derived from scenario testing
- Consider the sequence diagram for the Withdraw Checking use case as given below:

IDENTIFYING METHODS (CON'T)



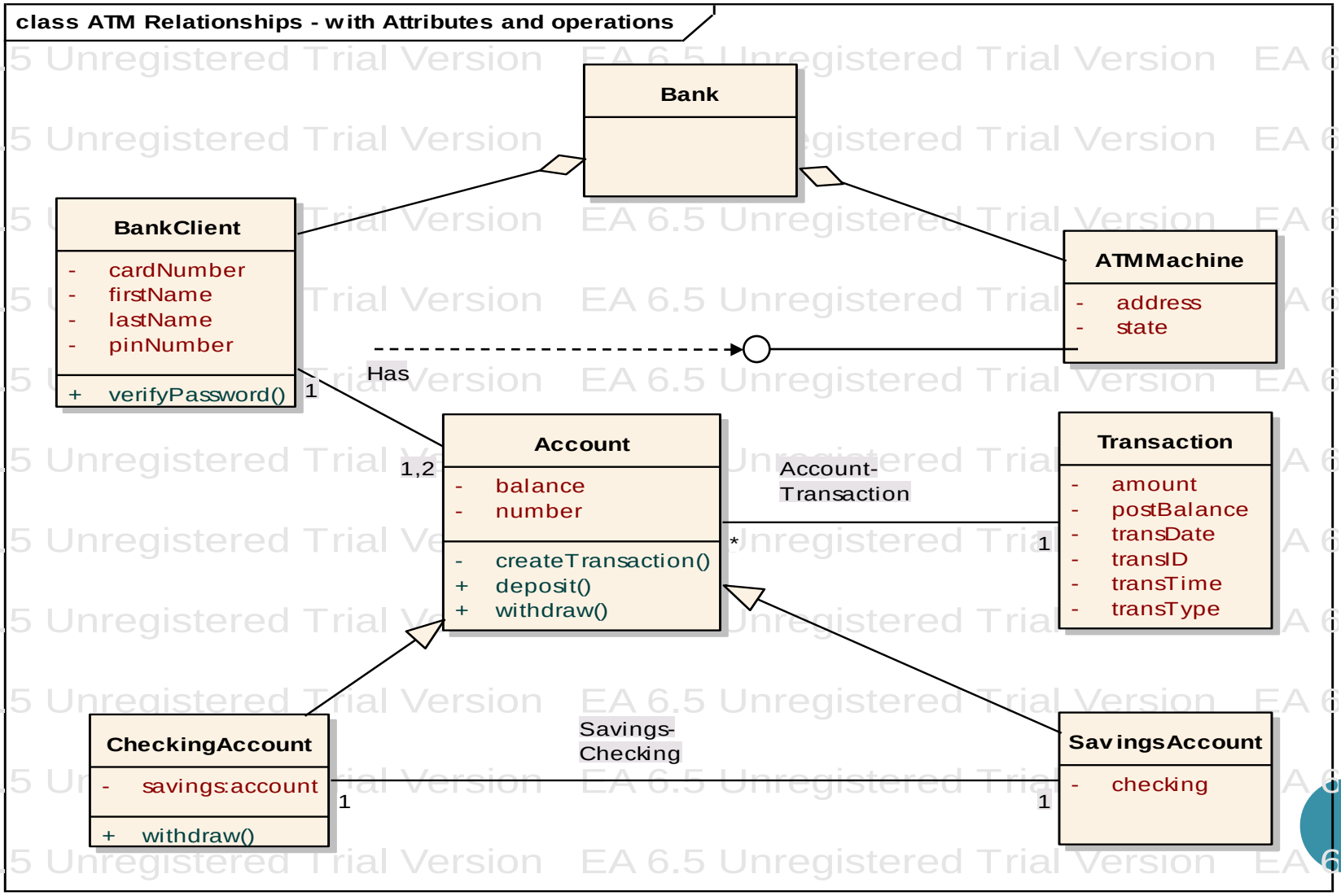
CONT...

- Following use cases can be considered
  - Deposit Checking
  - Deposit Savings
  - Withdraw Checking
  - Withdraw More from Checking
  - Withdraw Savings
  - Withdraw Savings Denied
  - Checking Transaction History
  - Savings Transaction History
- E.g. by studying the sequence diagram for Withdraw Checking its clear that the Account class must provide a service such as withdrawal
- Same way for Deposit Checking

Con't

- Account must also be able to create transaction records of any deposit or withdrawal.
- Methods of Account Class
  - deposit
  - withdrawal
  - createTransaction
- Subclass will either inherit generic services as it is or enhance according to needs.
- So, here saving a/c will not change but Checking a/c will override withdraw method.

ATM- with attributes and operations



IDENTIFYING METHODS (CON'T)

- Methods usually correspond to queries about attributes (and sometimes association) of the objects
- Methods are responsible for managing the value of attributes such as query, updating, reading and writing
- For example, we need to ask the following questions about soup class:
  - What services must a soup class provide? And
  - What information (from domain knowledge) is soup class responsible for storing?
- Let's first take a look at its attributes which are:
  - name
  - preparation
  - price
  - preparation time and
  - oven temperature

IDENTIFYING METHODS (CON'T)

- Now we need to add methods that can maintain these attributes
- For example, we need a method to change a price of a soup and another operation to query about the price
- setName
- getName
- setPreparation
- get Preparation
- setCost
- getCost
- setOvenTemperature
- getOvenTemperature
- setPreparationTime
- getPreparationTime

SUMMARY

- We learned how to identify three types of object relationships:
  - Association
  - Super-sub Structure (Generalization Hierarchy)
  - A-part-of Structure
- The hierarchical relation allows the sharing of properties or inheritance
- A reference from one class to another is an association
- The A-Part-of Structure is a special form of association
- Every class is responsible for storing certain information from domain knowledge
- Every class is responsible for performing operations necessary upon that information