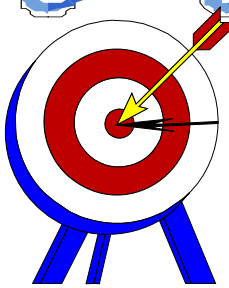
Object-Oriented Systems Development:
Using the Unified Modeling Language

Chapter 8:

Identifying Object Relationships, Attributes, and Methods



Goals

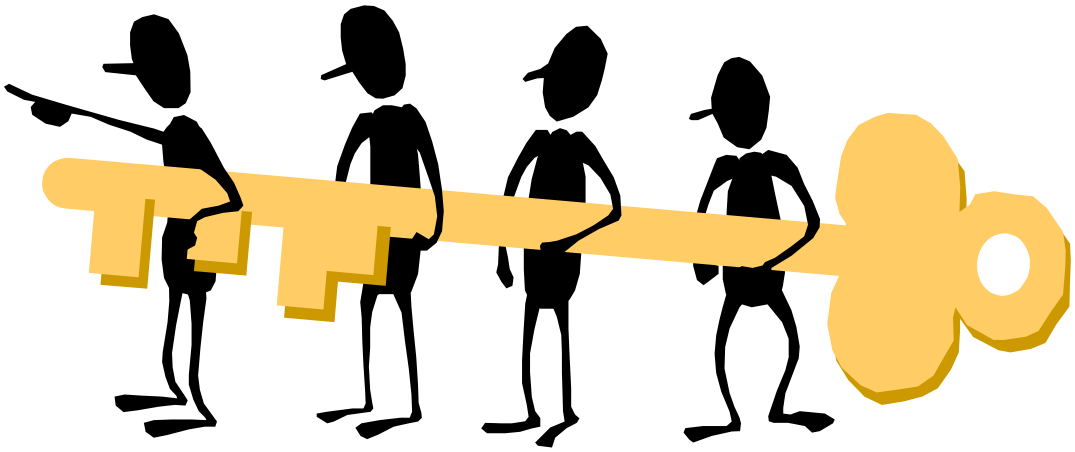
- Analyzing relationships among classes.
- Identifying association.
- Association patterns.
- Identifying super- and subclass hierarchies.

Introduction

- **Identifying aggregation or a-part-of compositions.**
- **Class responsibilities.**
- **Identifying attributes and methods by analyzing use cases and other UML diagrams.**

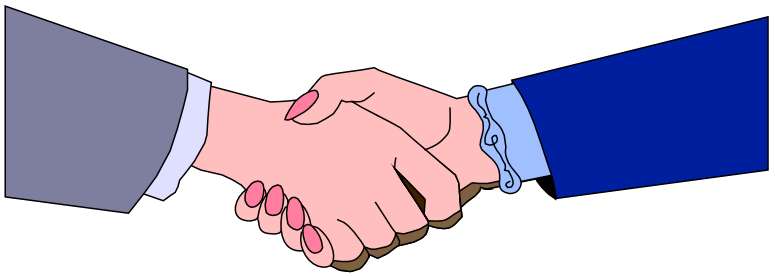
Objects contribute to the behavior of the system by collaborating with one another.

—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.

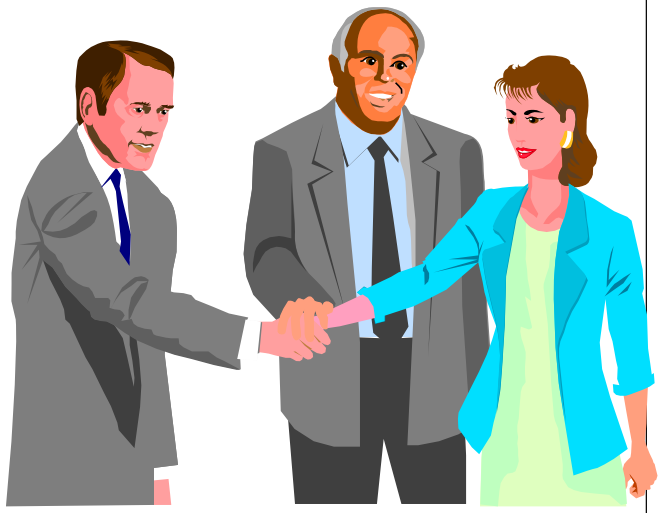


Objects Relationships

- **Three types of relationships among objects are:**
  - ***Association.***
  - ***Super-sub structure* (also known as *generalization hierarchy*).**
  - **Aggregation and a-part-of structure.**

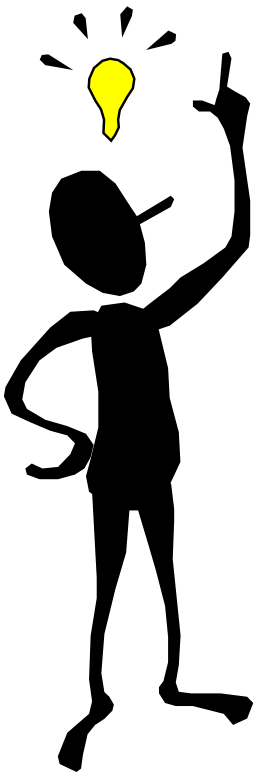
Associations

- **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.**



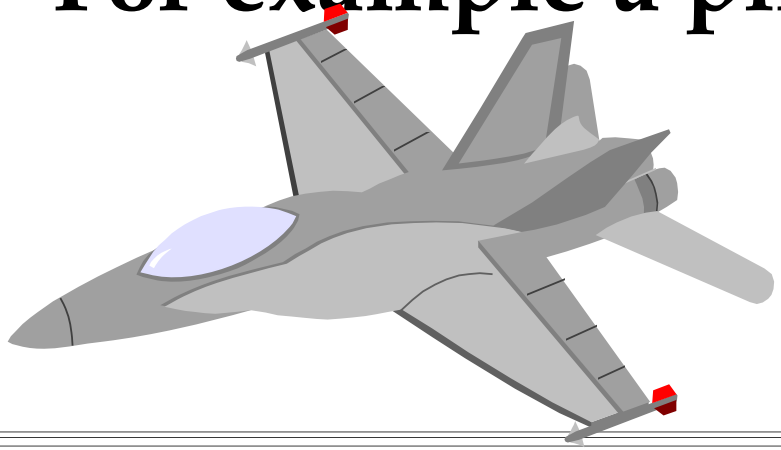
Associations (Con't)

- **Some associations are implicit or taken from general knowledge.**



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.**



Guidelines For Identifying Associations (Con't)

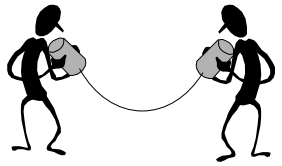
- **Association often corresponds to verb or prepositional phrases such as *part of*, *next to*, *works for*, *contained in*, etc.**

Common Association Patterns

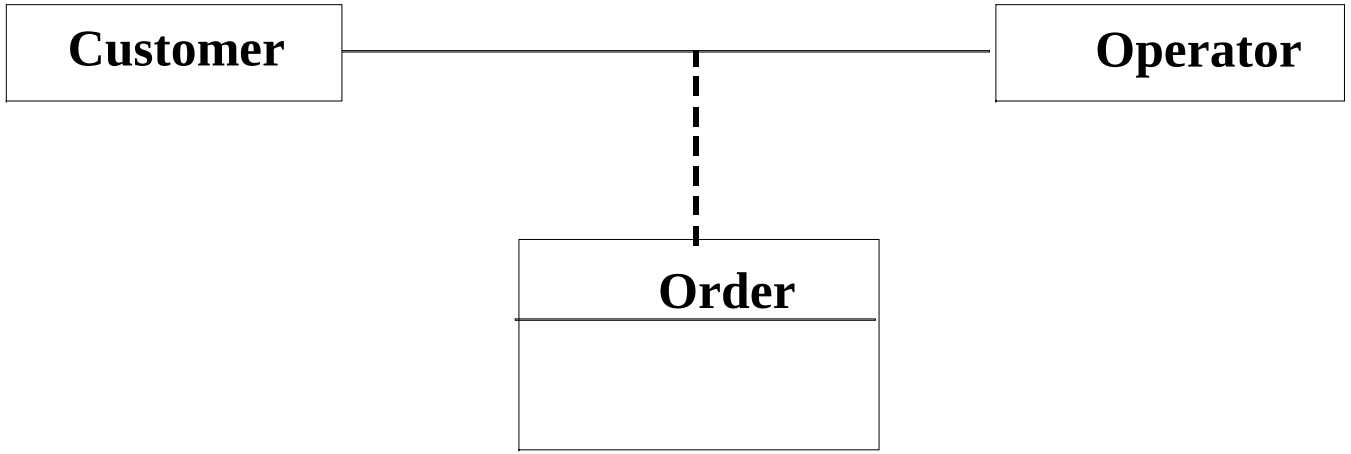
- **Common association patterns include:**
- **Location Association:** ***next To*****,** ***part of*****,** ***contained in, ingredient of*** ***etc.*** **:**
- **For example cheddar cheese is an** ***ingredient of*** **the French soup.**

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

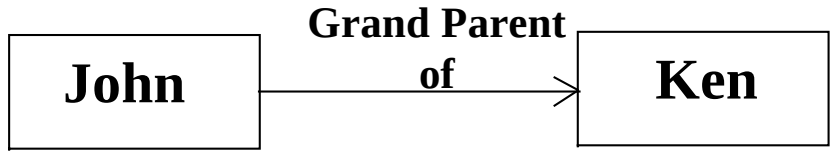
- ***Implementation association*. Defer implementation-specific associations to the design phase.**
- ***Ternary associations*. *Ternary* or n-ary *association* is an association among more than two classes**

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.**

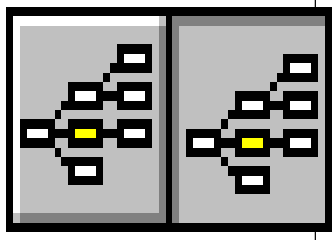
Eliminate Unnecessary Associations (Con't)

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



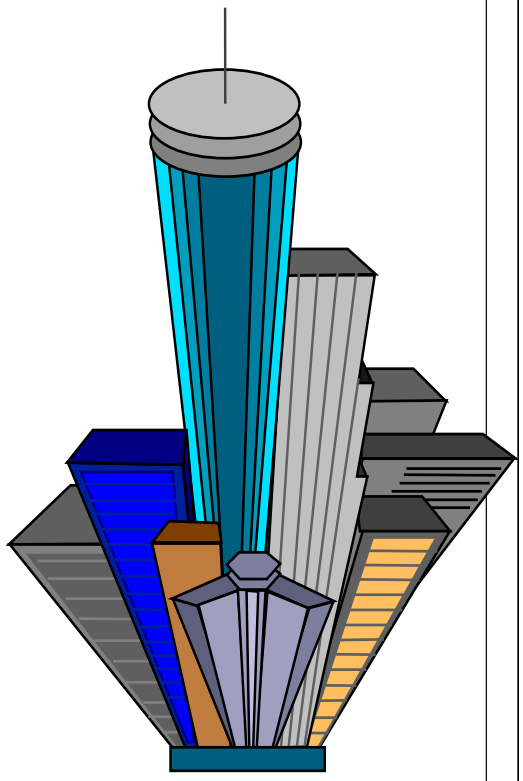
Superclass-Subclass Relationships

- **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.**
- **Example, Military Aircraft and Civilian Aircraft.**
- **Only specialize when the sub classes have significant behavior.**

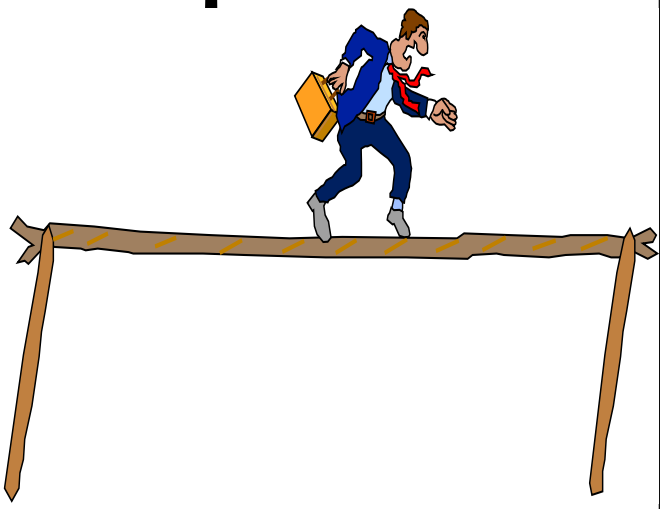


Guidelines For Identifying Super-sub Relationships: Bottom-up

- **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.**

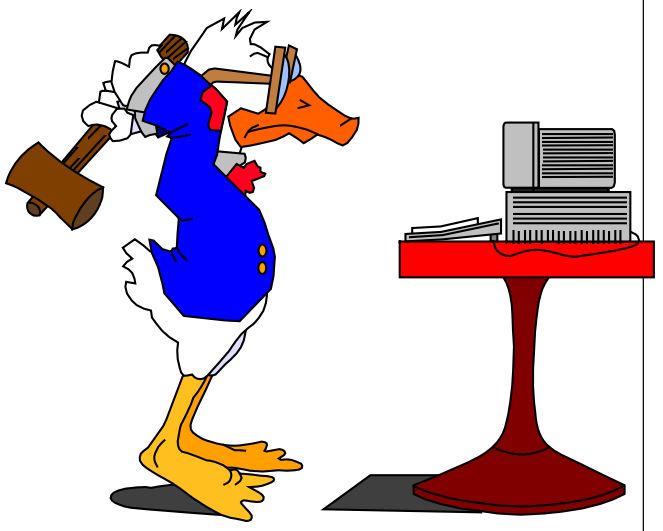
Guidelines For Identifying Super-sub Relationships: Reusability

- **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.**



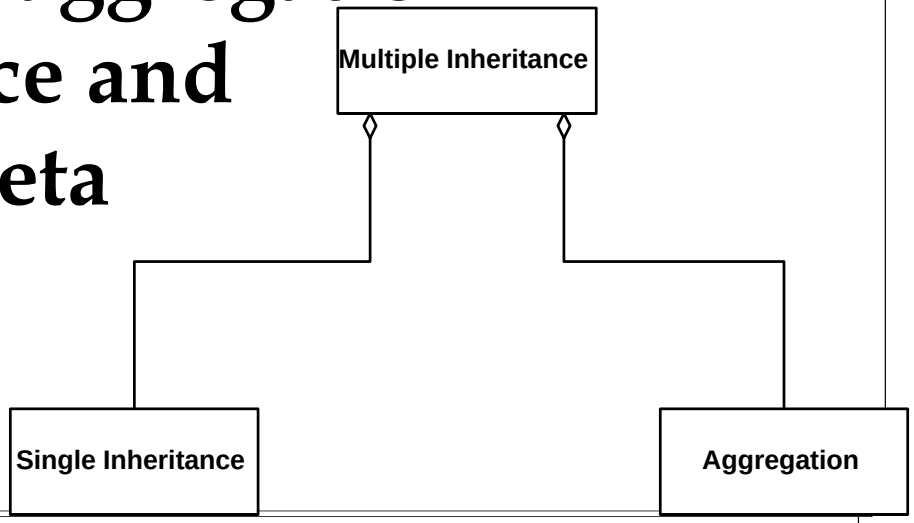
Guidelines For Identifying Super-sub Relationships: Multiple inheritance

- **Avoid excessive use of multiple inheritance.**
- **It is also more difficult to understand programs written in multiple inheritance system.**



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.**

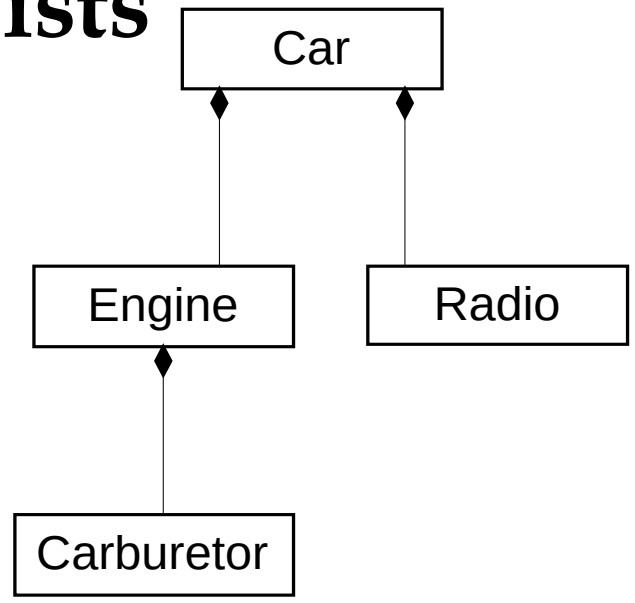


A-Part-of Relationship - Aggregation

- ***A-part-of relationship*, also called *aggregation*, represents the situation where a class consists of several component classes.**

A-Part-of Relationship - Aggregation (Con't)

- 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

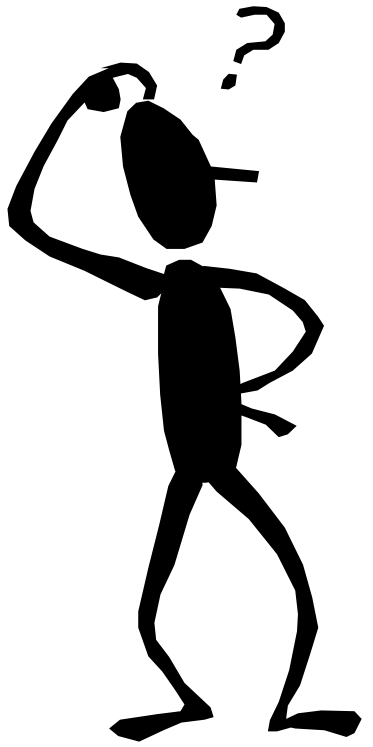
- **If *A* is part of *B* and B is part of *C*, then *A* is part of *C*.**
- **For example, a carburetor is part of an engine and an engine is part of a car; therefore, a carburetor is part of a car.**

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.**

Where responsibilities for certain behavior must reside?

- **Does the part class belong to problem domain?**
- **Is the part class within the system's responsibilities?**



where responsibilities ...(Con't)

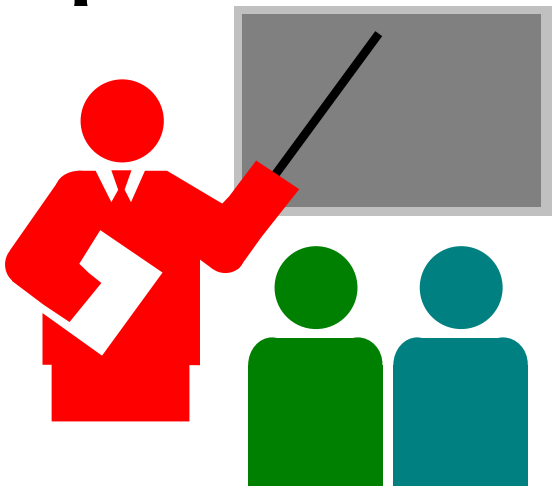
- **Does the part class capture more than a single value?**
- **If it captures only a single value, then simply include it as an attribute with the whole class.**
- **Does it provide a useful abstraction in dealing with the problem domain?**

A-Part-of Relationship Patterns Assembly

- **An assembly-Part situation physically exists.**
- **For example, a French soup consists of onion, butter, flour, wine, French bread, cheddar cheese, etc.**

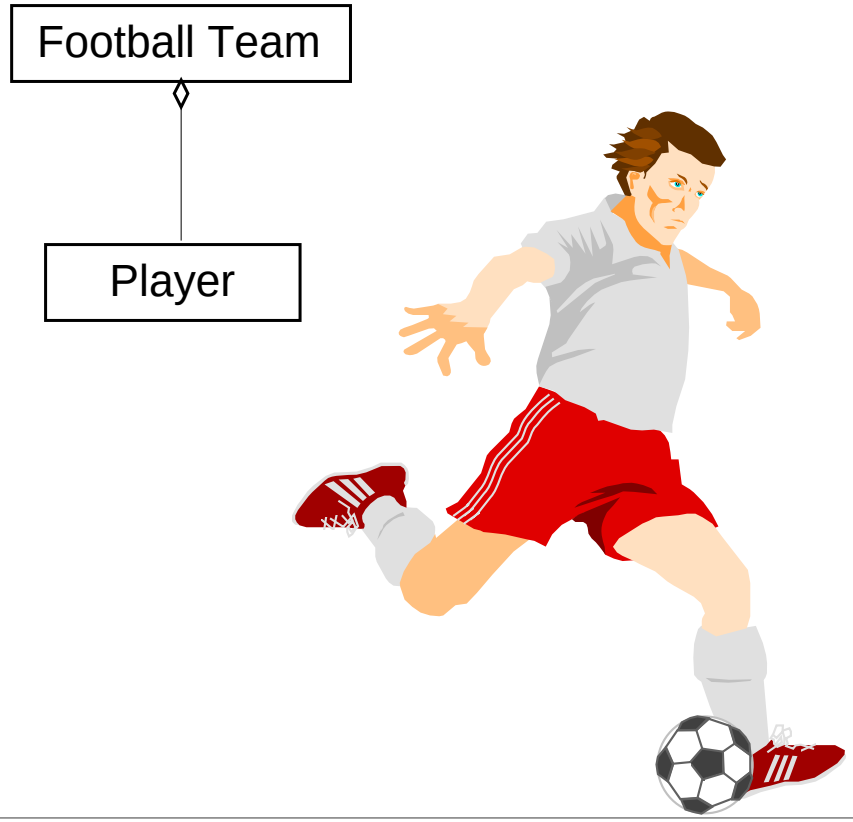
A-Part-of Relationship Patterns Container

- **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 soccer team is a collection of players.**



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 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.**

Responsibility

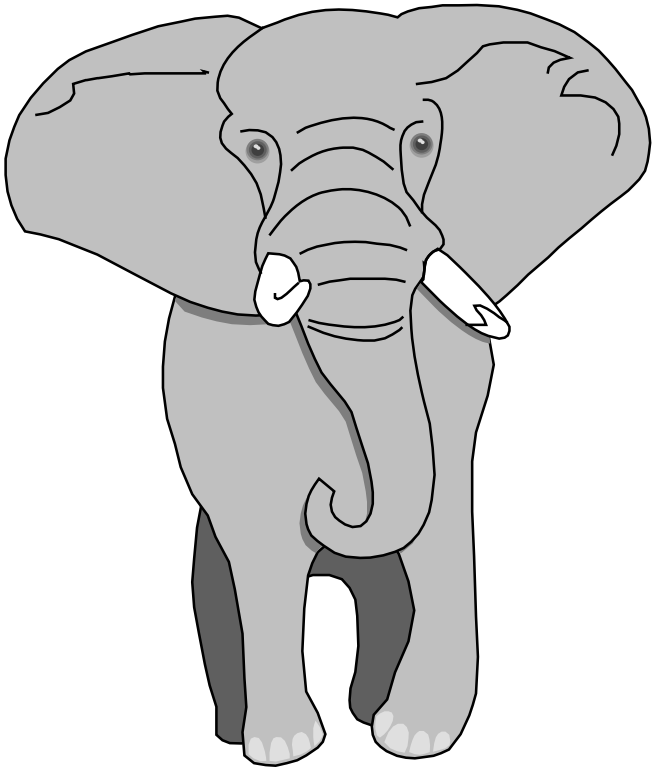
- How am I going to be used?
- How am I going to collaborate with other classes?
- How am I described in the context of this system's responsibility?
- What do I need to know?
- What state information do I need to remember over time?
- What states can I be in?

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 possessive phrases such as *cost of* the soup.**

Guidelines For Identifying Attributes Of Classes (Con't)

- **Keep the class simple; only state enough attributes to define the object state.**

Guidelines For Identifying Attributes Of Classes (Con't)

- **Attributes are less likely to be fully described in the problem statement.**
- **You must draw on your knowledge of the application domain and the real world to find them.**

Guidelines For Identifying Attributes Of Classes (Con't)

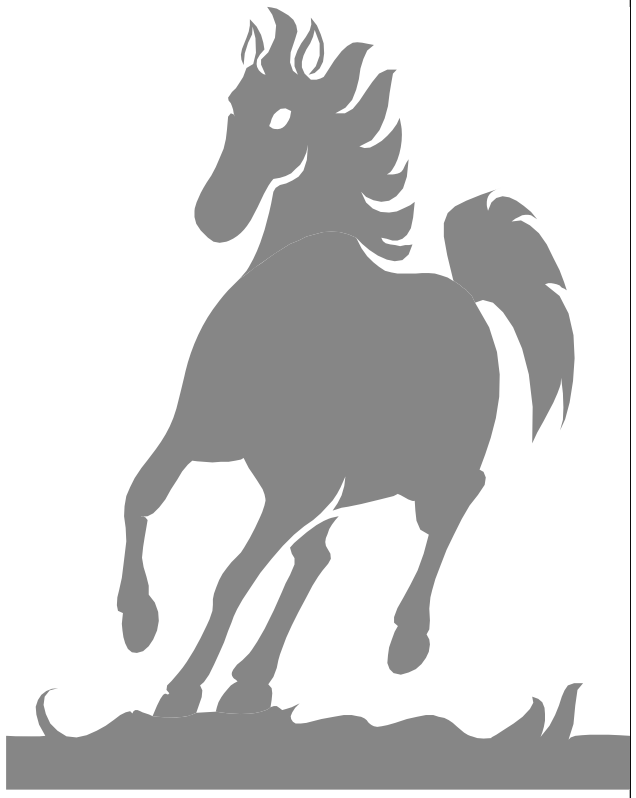
- **Omit derived attributes.**
- **For example, don't use age as an attribute since it can be derived from date of birth.**
- **Drive attributes should be expressed as a method.**

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.**

Object Responsibility: Methods & Messages

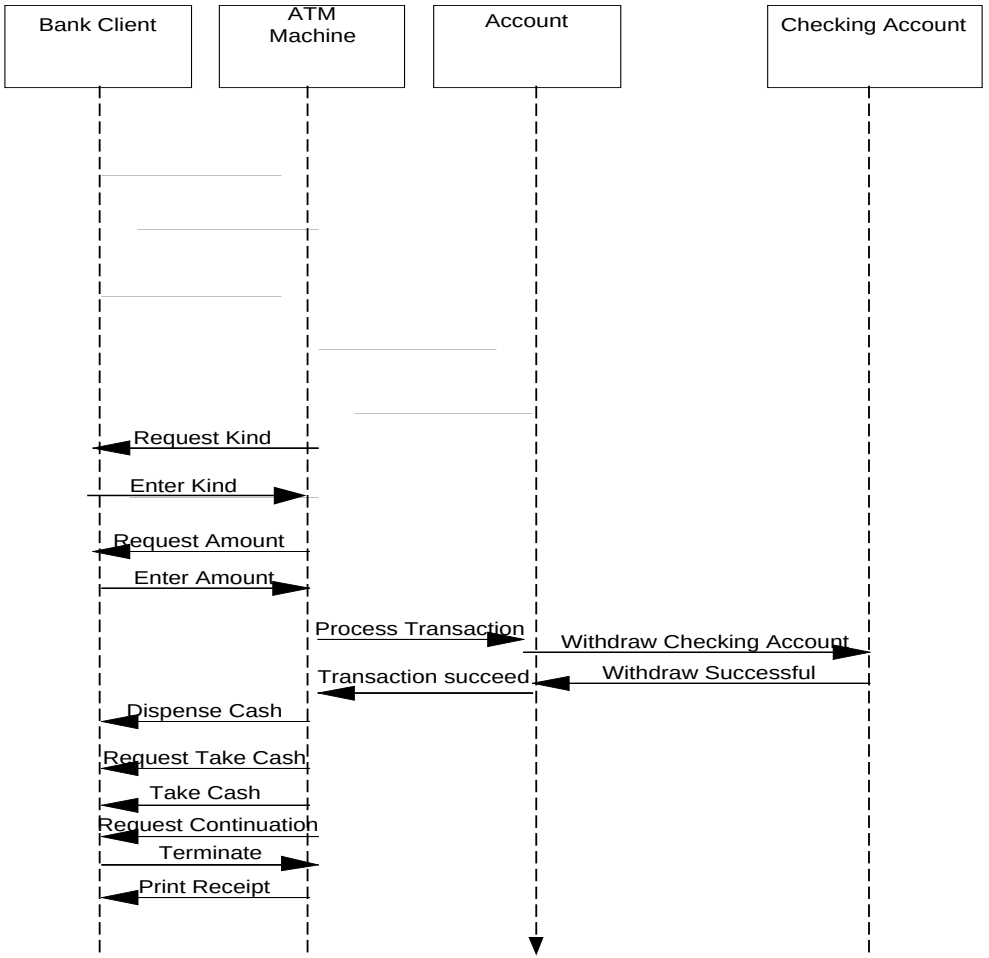
- **Methods and messages are the work horses of object-oriented systems.**
- **In O-O environment, every piece of data, or object, is surrounded by a rich set of routines called methods.**



Identifying Methods by Analyzing UML Diagrams and Use Cases

- **Sequence diagrams can assist us in defining the services the objects must provide.**

Identifying Methods (Con't)



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.**

Identifying Methods (Con't)

- **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?**

Identifying Methods (Con't)

- **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.**

Identifying Methods (Con't)

- **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**

Summary (Con't)

- **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.**

Summary (Con't)

- **Every class is responsible for storing certain information from domain knowledge .**
- **Every class is responsible for performing operations necessary upon that information.**