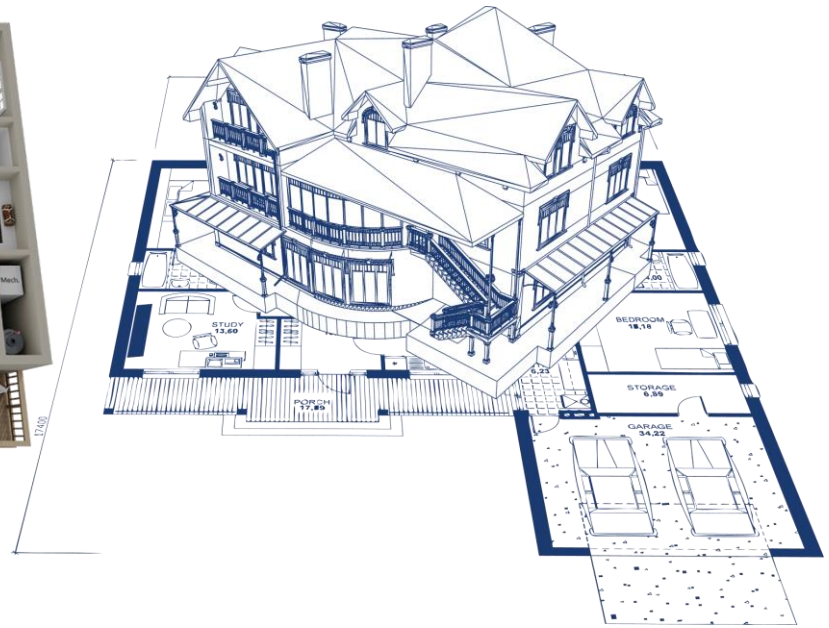
# UML : Unified Modeling Language

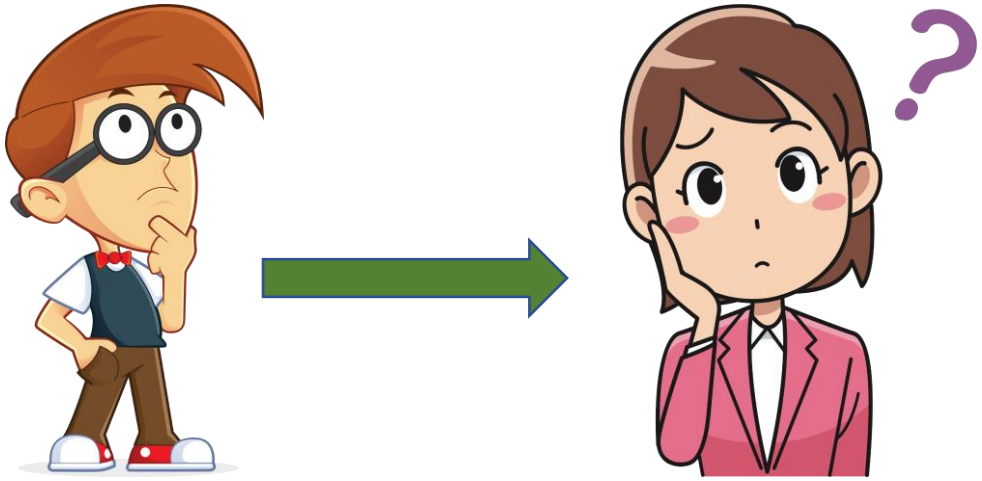
## UML Diagrams

# UML : Unified Modeling Language

- UML is modeling language used to **model** or **sketch** OO systems
- It is a collection **graphical design notations** to depict object-oriented systems.
- It **specifies**, **visualises** and **documents** all aspects of an OO system.
- UML originated in the mid-1990's from the efforts of Grady Booch, James Rumbaugh and Ivar Jacobson : Watch their video
- There many UML diagrams that **captures different aspects** of an OO system.

# Why Model a System?



Picture Courtesy: https://www.pngwing.com/

# Why UML Modeling?

## A model is a simplification of reality, providing blueprints of a system.

- In Unified Modeling Language (UML), a model may be **structural, emphasizing the organization** of the system or it may be **behavioral, emphasizing the dynamics** of the system.
- UML, in specific:
  - Permits you to specify the structure or behavior of a system.
  - Helps you visualize a system.
  - Provides template that guides you in constructing a system.
  - Helps to understand complex system part by part.
  - Document the decisions that you have made.
- We build model so that we can better understand the system we are developing. A model may encompass an overview of the system under consideration, as well as a detailed planning for system design, implementation and testing.

# CLASS DIAGRAM

**The UML Class diagram is a graphical notation used to construct and visualize object oriented systems.**

# Class Diagram

- A **class diagram** is used to show the **existence of classes** and their **relationships** in the logical view of a system.
- During **analysis**, we use class diagrams to indicate the **common roles and responsibilities of the entities** that provide the system's behavior.
- During **design**, we use class diagrams to capture the **structure of the classes** that form the system's architecture.

# Steps to follow to create a class diagram

- **Step 1**: Identify the class names
  - The first step is to identify the primary objects of the system.
- **Step 2**: Distinguish relationships
  - Next step is to determine how each of the classes or objects are related to one another. Look out for commonalities and abstractions among them; this will help you when grouping them when drawing the class diagram.
- **Step 3**: Create the Structure
  - First, add the class names and link them with the appropriate connectors. You can add attributes and functions/ methods/ operations later.

# Class Diagram Notations : Class icon

- The class icon consists of three compartments,
  - with the first occupied by the class name,
  - the second by the attributes,
  - and the third by the operations.

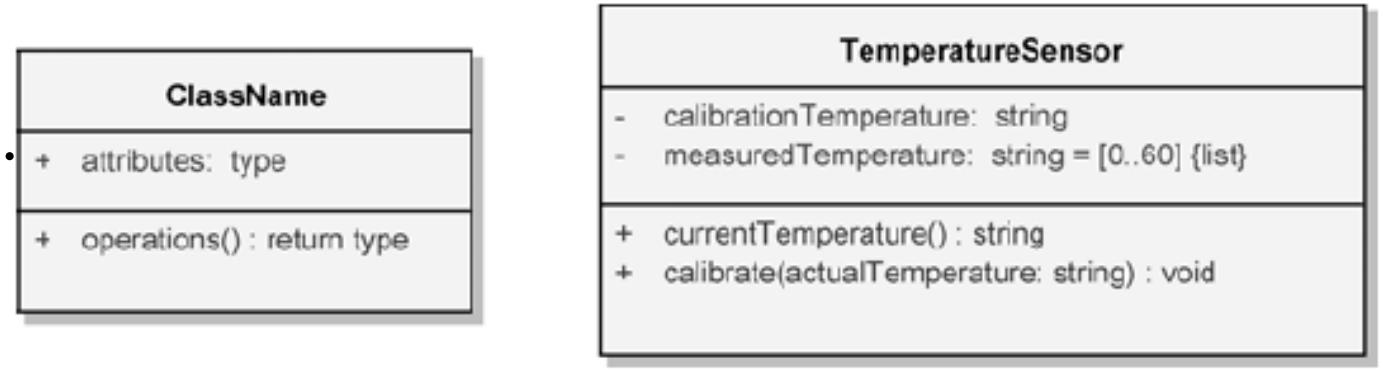



**Figure 5–33** A General Class Icon and an Example for the Gardening System

**Attribute specification format**:

***visibility attributeName : Type [multiplicity] = DefaultValue {property string}***

**Operation specification format:**

***visibility operationName (parameterName : Type) : ReturnType {property string}***

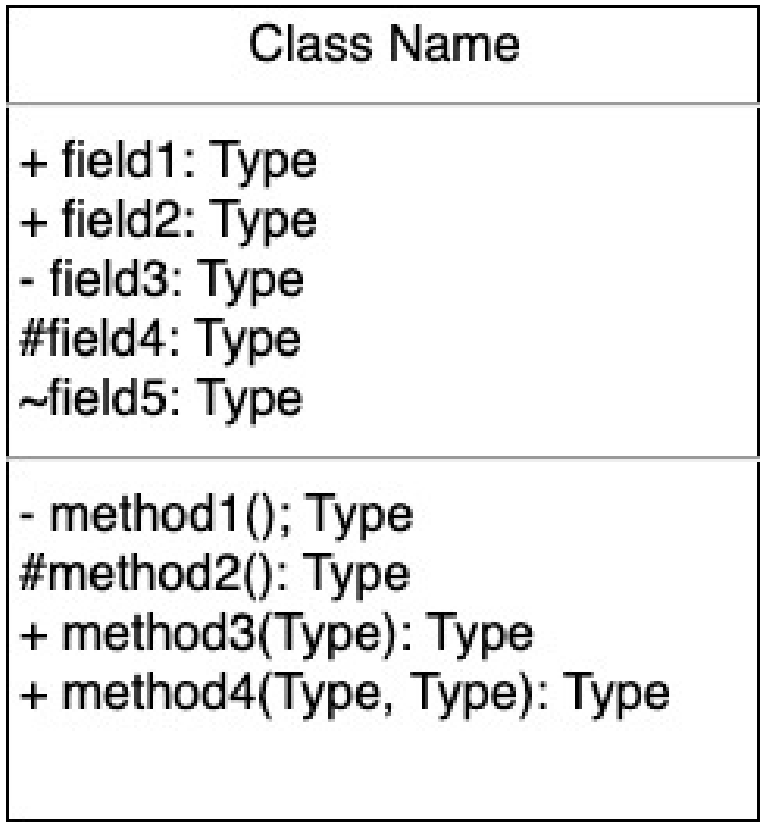
multiplicity of `[0..60]` on the `measuredTemperature` attribute indicates an array of 0 to 60 temperature measurements

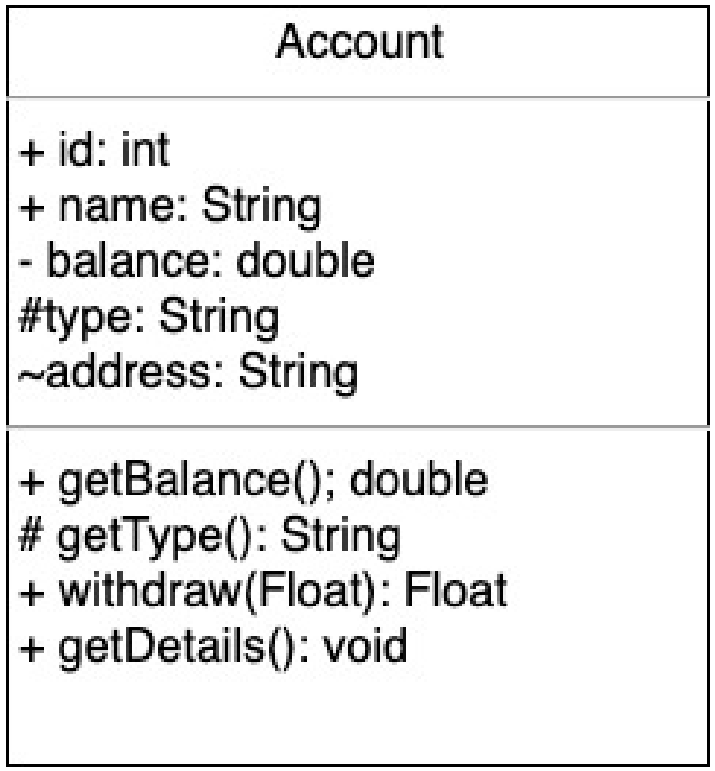
- We italicize the class name to show that we may have only instances of its subclasses., italize the operation to show it is abstract
- Class name begins in capital letters, and the space between multiple words is omitted
- The first letter of the attribute and operation names is lowercase, with subsequent words starting in uppercase, and spaces are omitted

# Member Visibility

- Data hiding leads to **member visibility** or access specification.
- Member access specification **defines how the member** will be accessed outside the class.
- The access specifiers are:
  - public – denoted by +
  - private – denoted by –
  - protected – denoted by #
  - default or package – denoted by ~

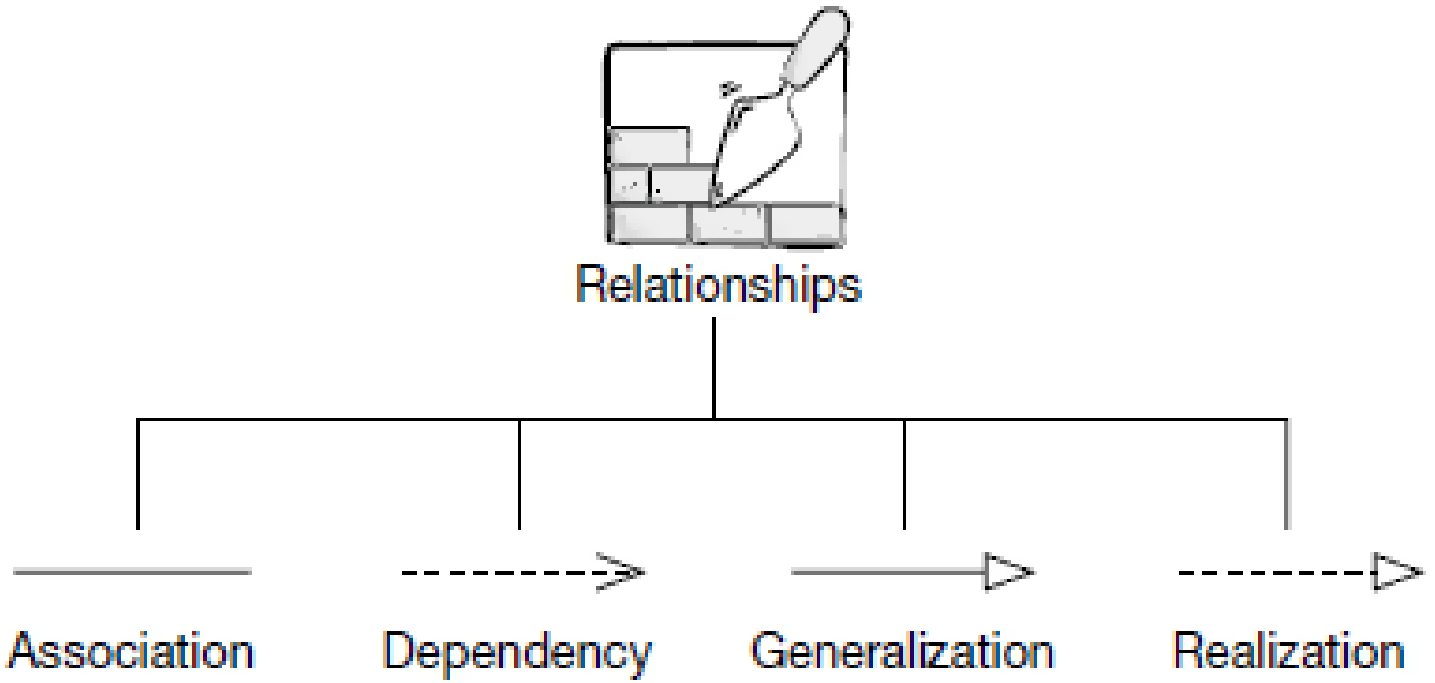
# Update Class Diagram



# Relationships in UML

- There are mainly three kinds of relationships in UML:
- **Associations**
- **Generalizations**
- **Dependencies**



# Association

1

Teacher

*teaches*

Student

2

Flight

*has*

Passenger

3

Employee

*works for*

Company

4


is-a

Rabbit

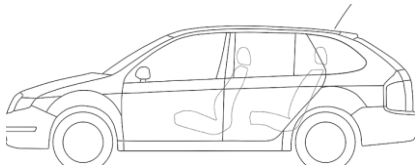
Dog

Tiger

Lion

Elephant

5



Car

part-of

PC: https://www.pngwing.com

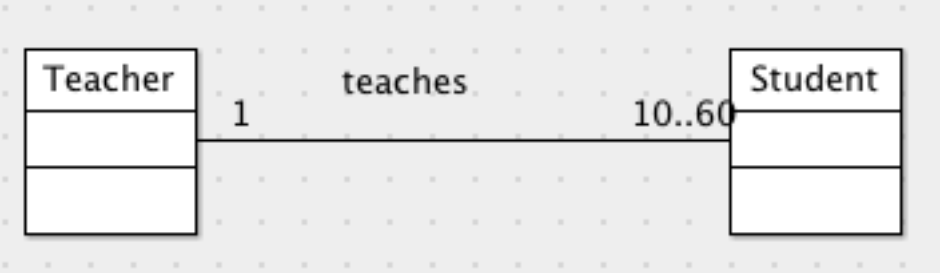
# Association between Objects

- Objects are not standalone entities : **Relationships** among Objects.
- They **collaborate** with one another : to do a task(s) in an OO system.
- Objects are **related** or **associated** with other objects.
- Types of Association relationships between objects:-
  - **Simple Association**
  - **Specialized Association**
    - **Aggregation**
    - **composition**

# Simple Association/Link

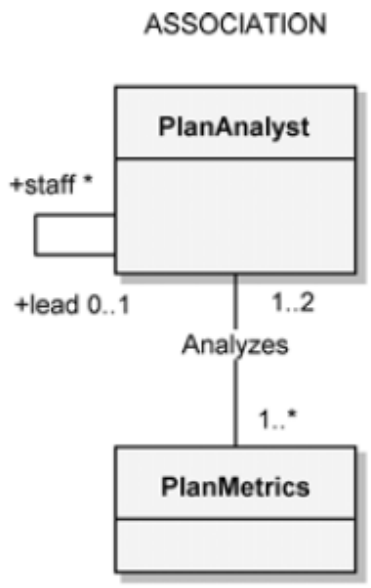
The association icon connects two classes and denotes a semantic connection

- An association is used when one object wants another object to **perform a service** for it.
- Denoted by a solid line connecting two classes.
- **Multiplicities** can also be mentioned in the association – indicates **number of objects** involved in the association.

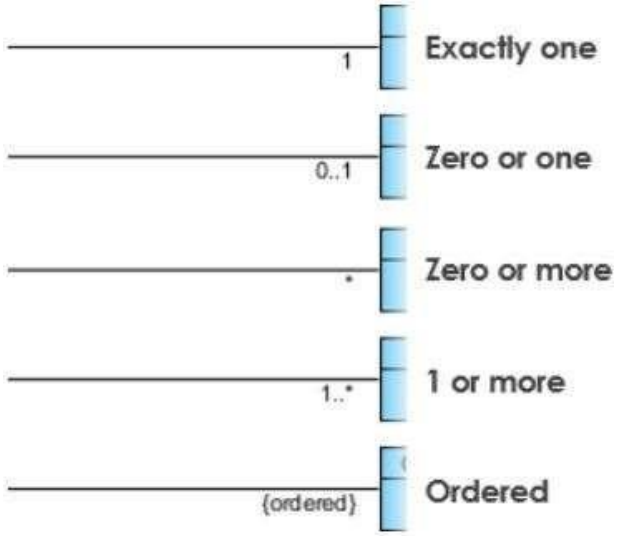


# Multiplicity

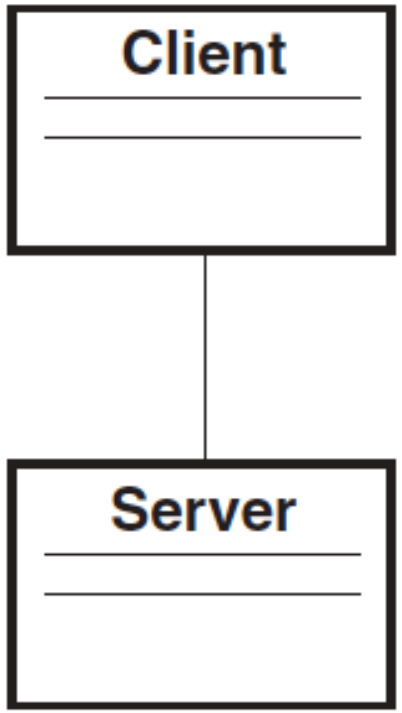
- The multiplicity adornment is applied to the target end of an association and denotes the number of links between each instance of the source class and instances of the target class
- Associations are often labeled with noun phrases, such as Analyzes, denoting the nature of the relationship. A class may have an association to itself (called a *reflexive association*), such as the collaboration among instances of the PlanAnalyst class

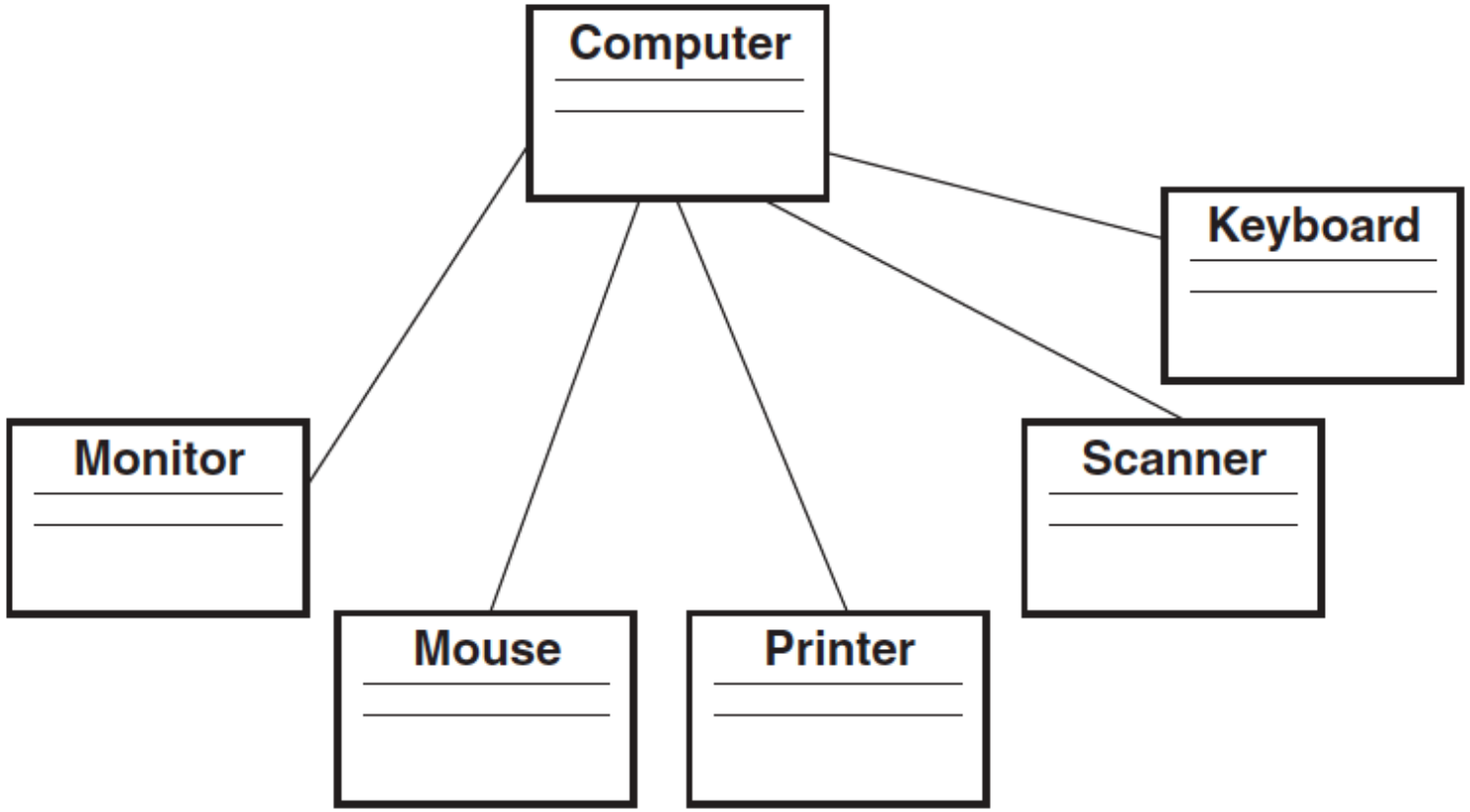



- `1` Exactly one
- `*` Unlimited number (zero or more)
- `0..*` Zero or more
- `1..*` One or more
- `0..1` Zero or one
- `3..7` Specified range (from three through seven, inclusive)



# Association Examples

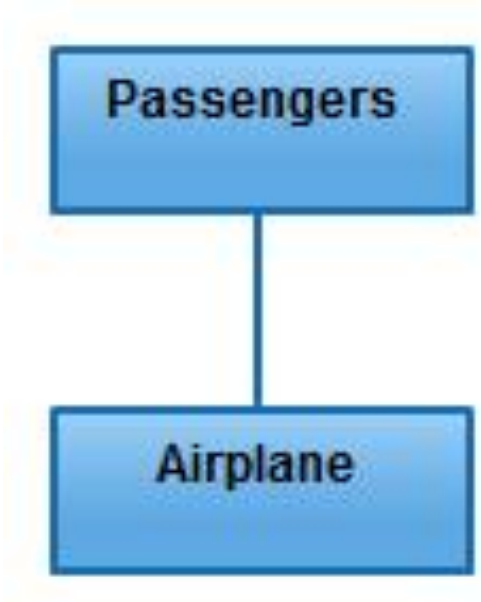


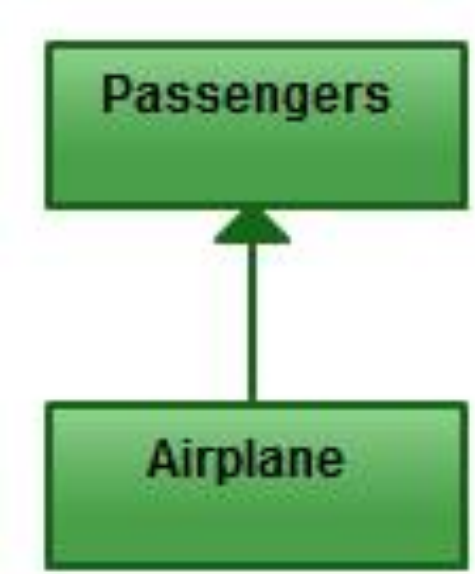

PC: Weisfeld Text 10.8 & 10.9

# Association

Encompasses about any logical connection or relationship between classes.




**Association**




**Directed Association**

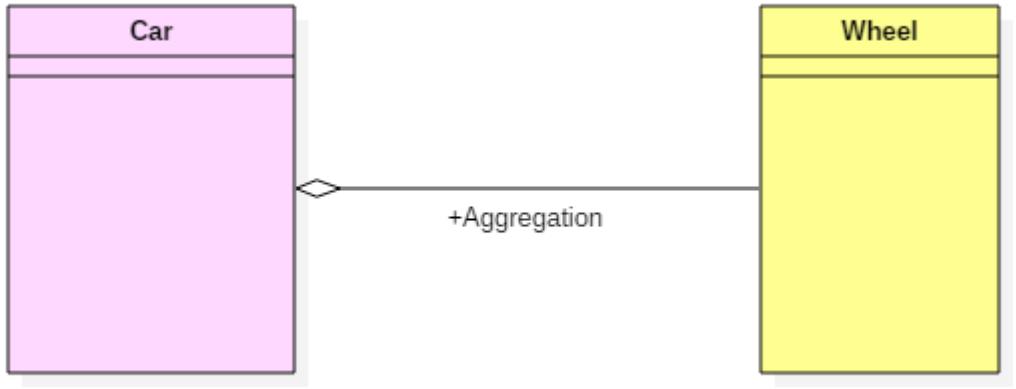

**Reflexive Association**

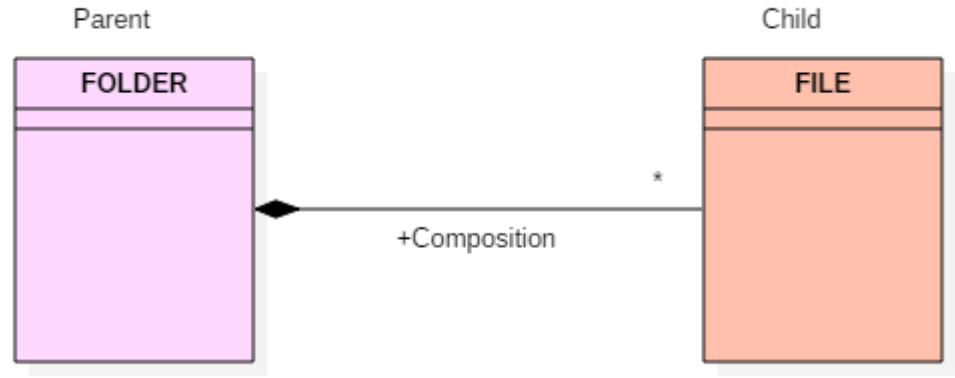
Directional relationship represented by a line with an arrowhead. The arrowhead depicts a container-contained directional flow.

one fleet may include multiple airplanes, while one commercial airplane may contain zero to many passengers. The notation 0..* in the diagram means “zero to many”.

# Specialized Associations

## Aggregation, Composition




**Aggregation is a special type of association that models a whole- part relationship between aggregate and its parts.**

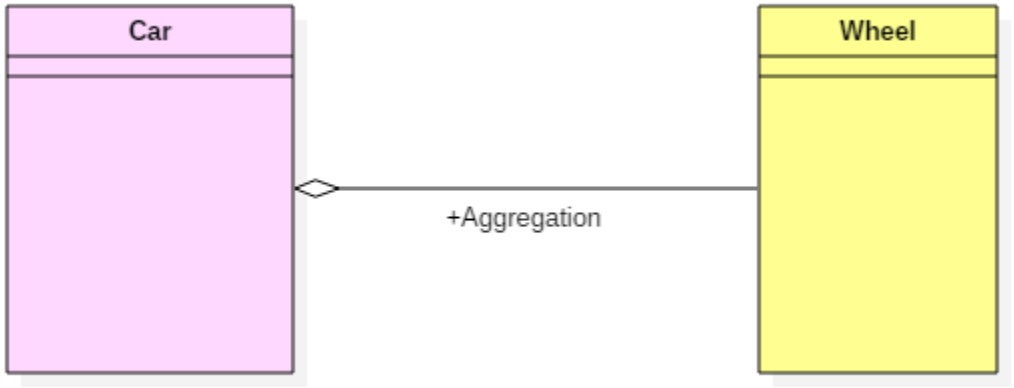
**The composition is a special type of aggregation which denotes strong ownership between two classes when one class is a part of another class.**

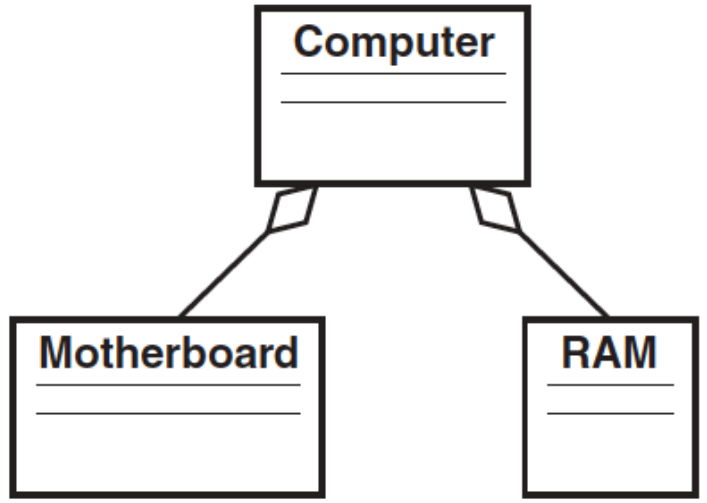
# Association vs Aggregation vs Composition

- **Aggregation** and **Composition** are subsets of association means- they are **specific cases of association**. In both aggregation and composition object of one class "owns" object of another class. But there is a subtle difference:
  - **Aggregation** implies a relationship where the child can exist independently of the parent. Example: Class (parent) and Student (child). Delete the Class and the Students still exist.
  - **Composition** implies a relationship where the child cannot exist independent of the parent. Example: House (parent) and Room (child). Rooms don't exist separate to a House.

# Aggregation

- Denoted by an arrowhead drawn as an unfilled diamond, **aggregation** can be read as **“is part of”** or, in the opposite direction as **“has a”**.

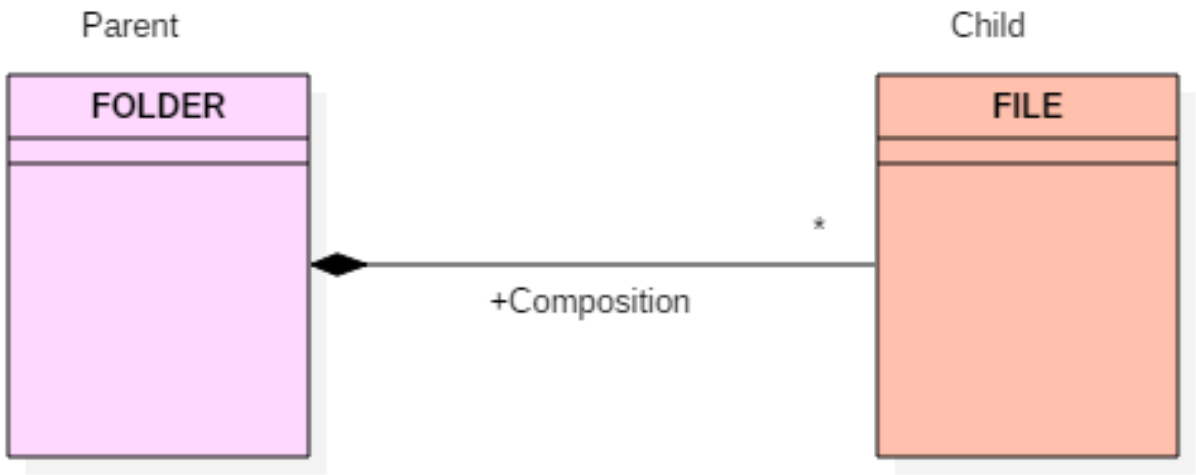



A car needs a wheel to function correctly, but a wheel doesn't always need a car. It can also be used with the bike, bicycle, or any other vehicles but not a particular car. Here, the wheel object is meaningful even without the car object. Such type of relationship is called an aggregation relation.

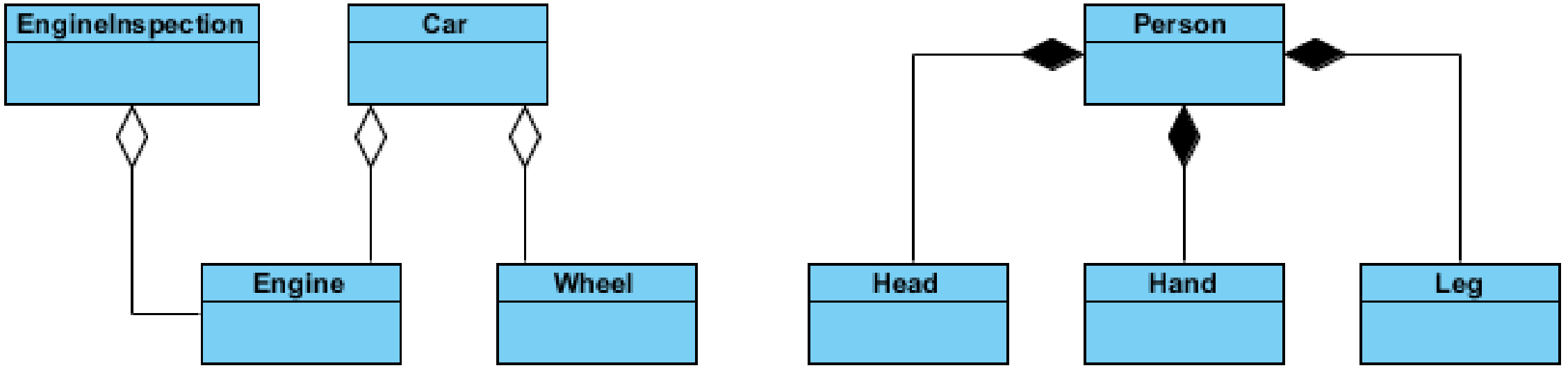
# Composition- Strong aggregation relation

- It is a two-way association between the objects.
- It is a whole/part relationship.
- If a composite is deleted, all other parts associated with it are deleted.

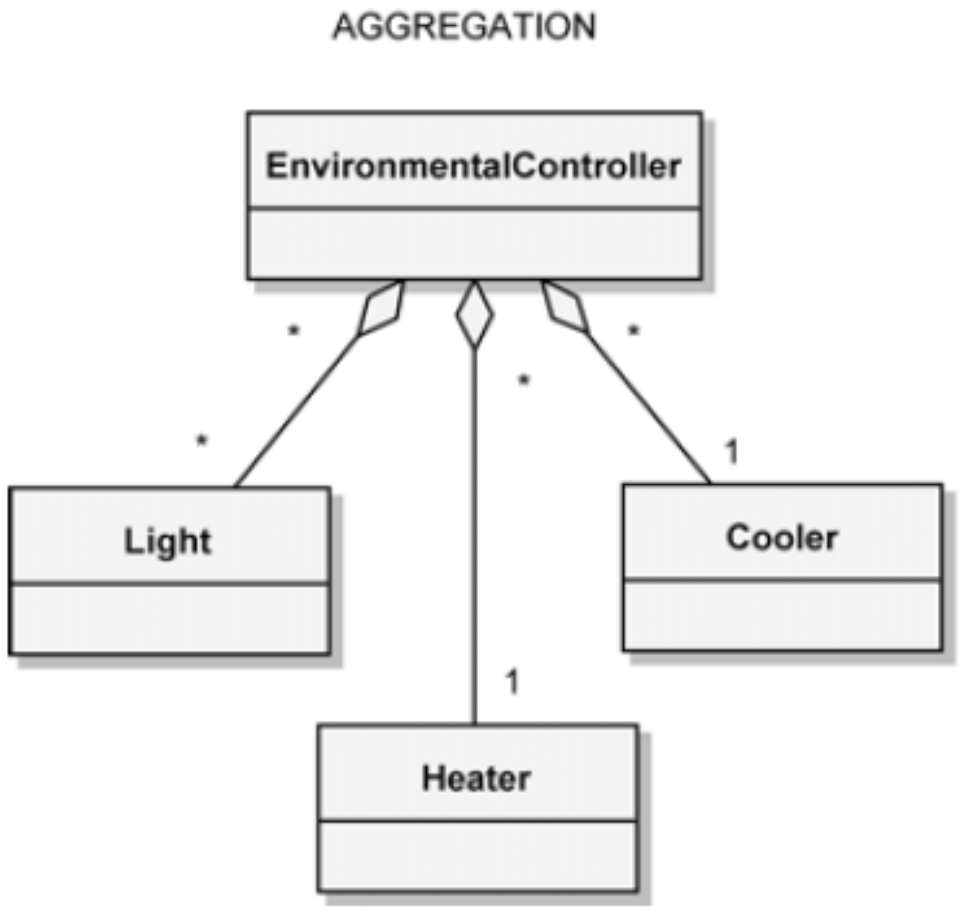


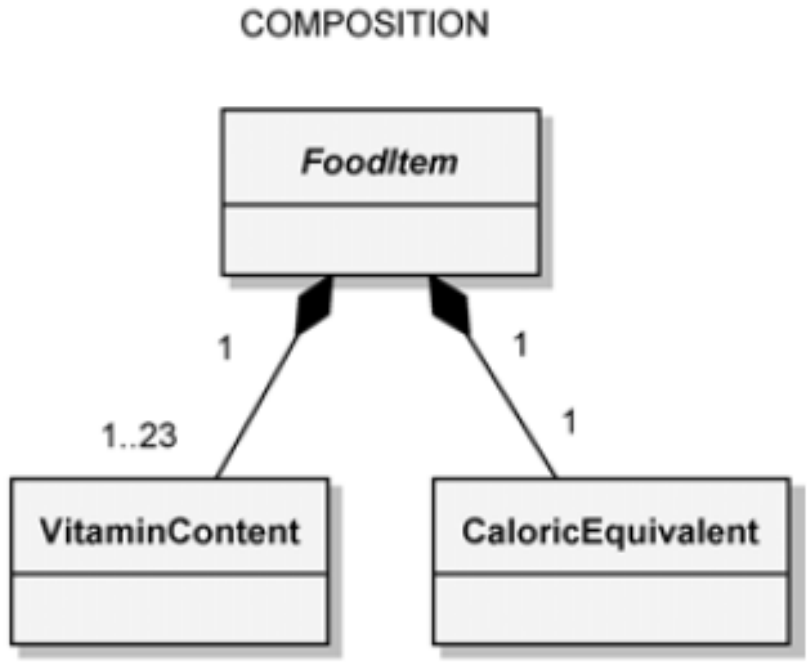

The folder could contain many files, while each File has exactly one Folder parent. If a folder is deleted, all contained files are removed as well.

# Aggregation vs Composition



# Aggregation vs Composition



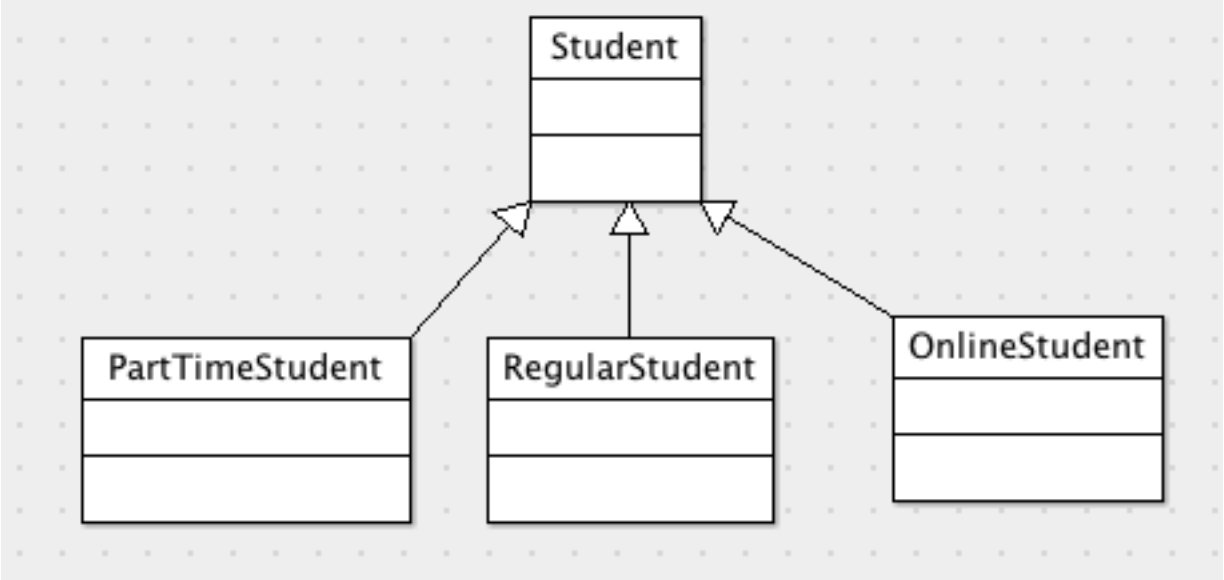



# Generalization- “is-a”

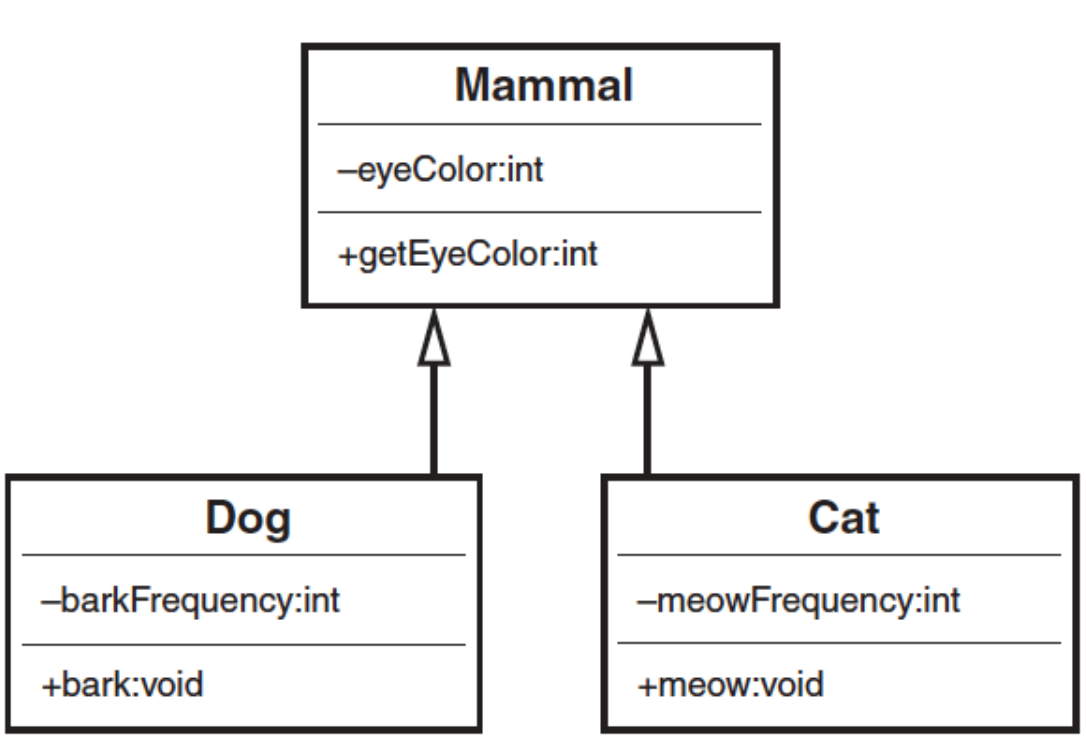
- Generalization is a relationship between a **general class** and a more **specific class**.
- “**is-a**” relation
- This relationship is achieved by an OO property : **Inheritance**.
  - General class : **Superclass** or **Parent** class or **Base** class
  - The specific/specialized class : **Subclass** or **Child** class or **Derived** class

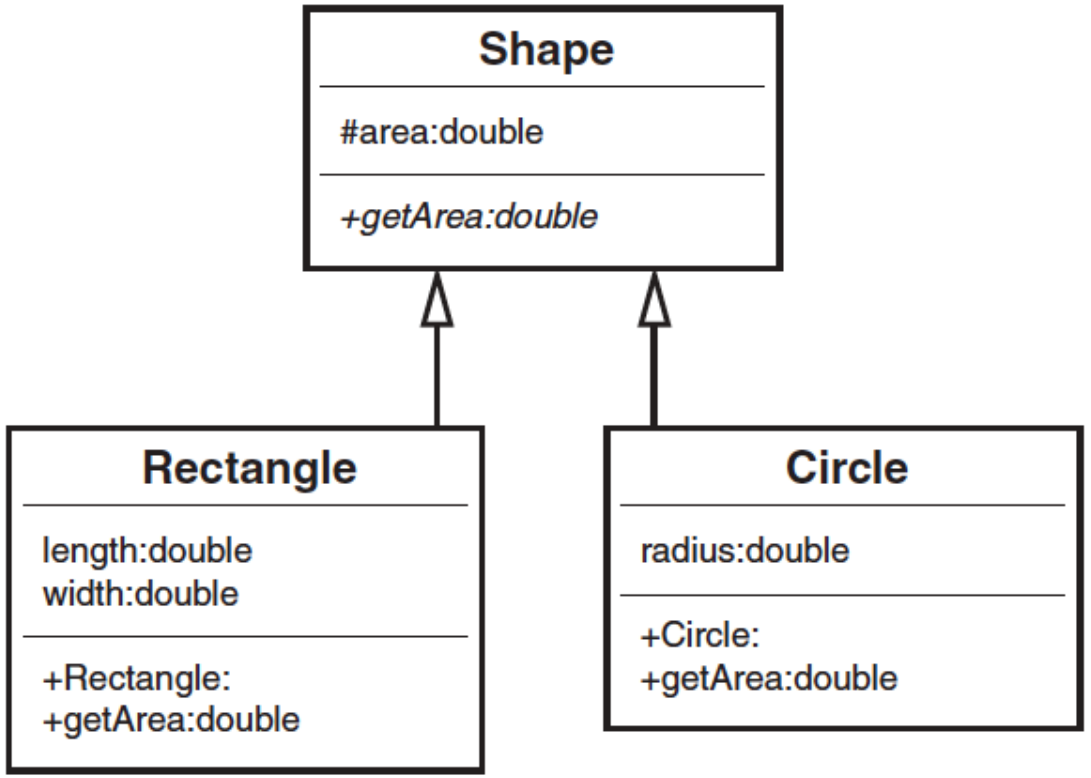
# Generalization – Notation

- Denoted by directed line with a closed, hollow arrowhead or triangle at the superclass end.

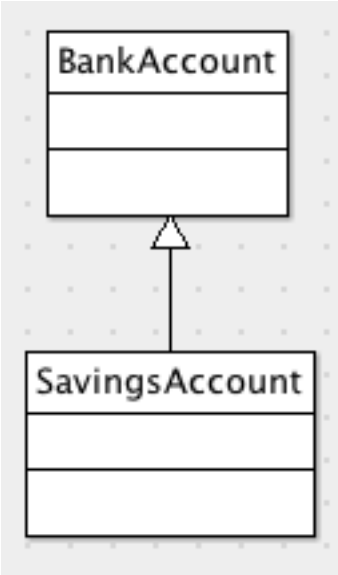


# Inheritance – an example : is-a relation

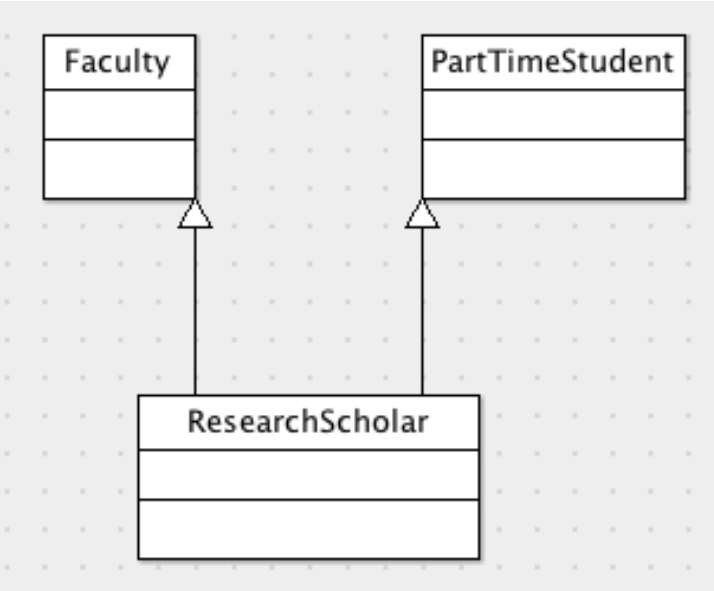


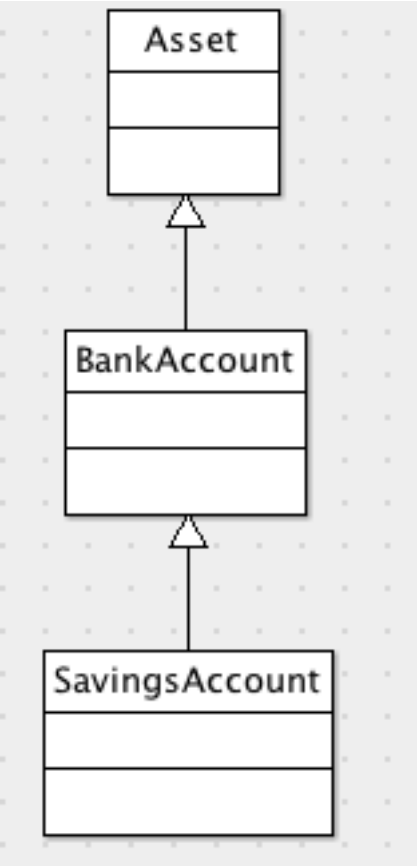
# Types of Inheritance




**Single Inheritance**
A class has only
one superclass




**Multiple Inheritance**
A class has
two or more superclasses




**Multilevel Inheritance**
A subclass can be
superclass to another class

# Namah Shivaya