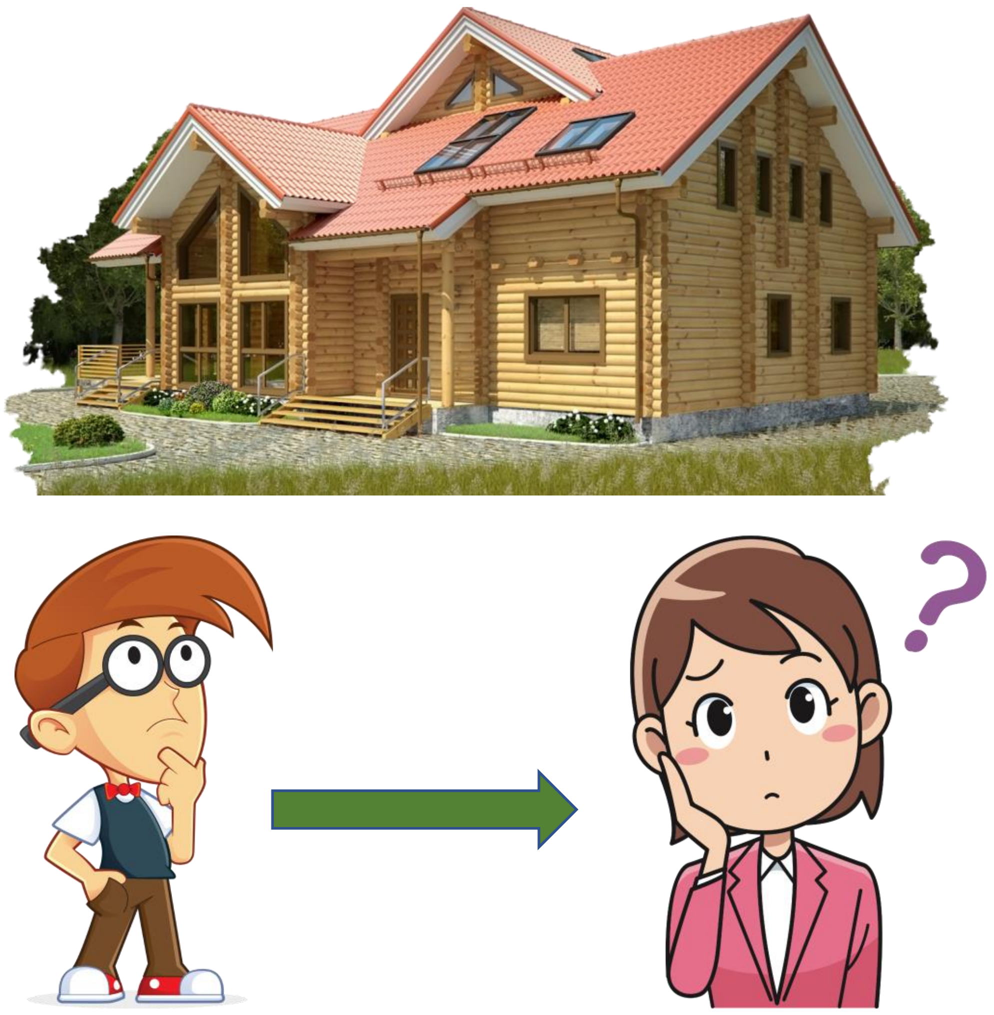
UML : Unified Modeling Language

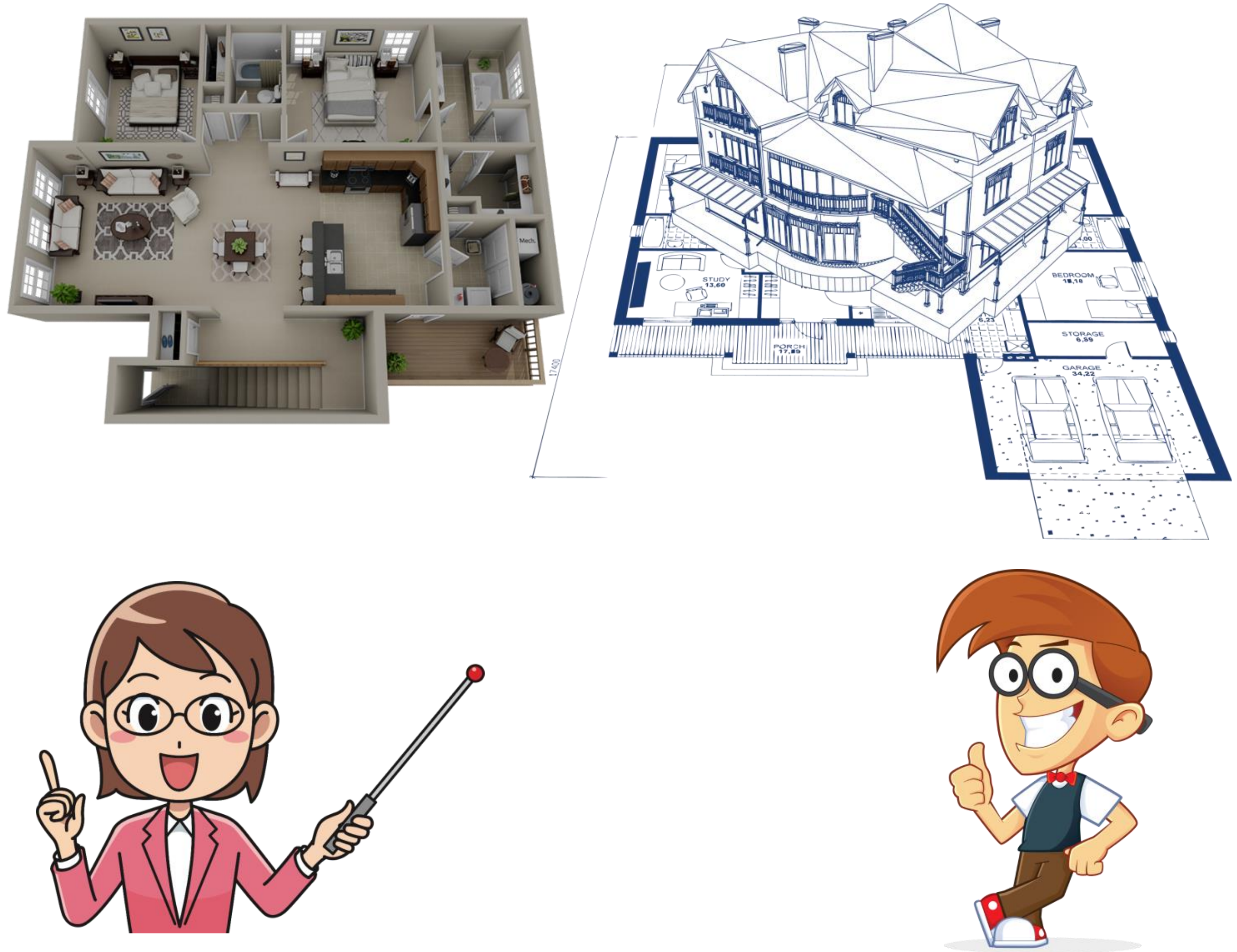
UML Diagrams

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.

Unified Modeling Language

- A well-defined and expressive notation is important to the process of software development.
- The Unified Modeling Language (UML) is the primary modeling language used to analyze, specify, and design software systems

Structure Diagrams

These diagrams are used to show the static structure of elements in the system. They may depict such things as the architectural organization of the system,

Behaviour Diagrams

Events happen dynamically in all software-intensive systems: Objects are created and destroyed, objects send messages to one another in an orderly fashion, and in some systems, external events trigger operations on certain objects

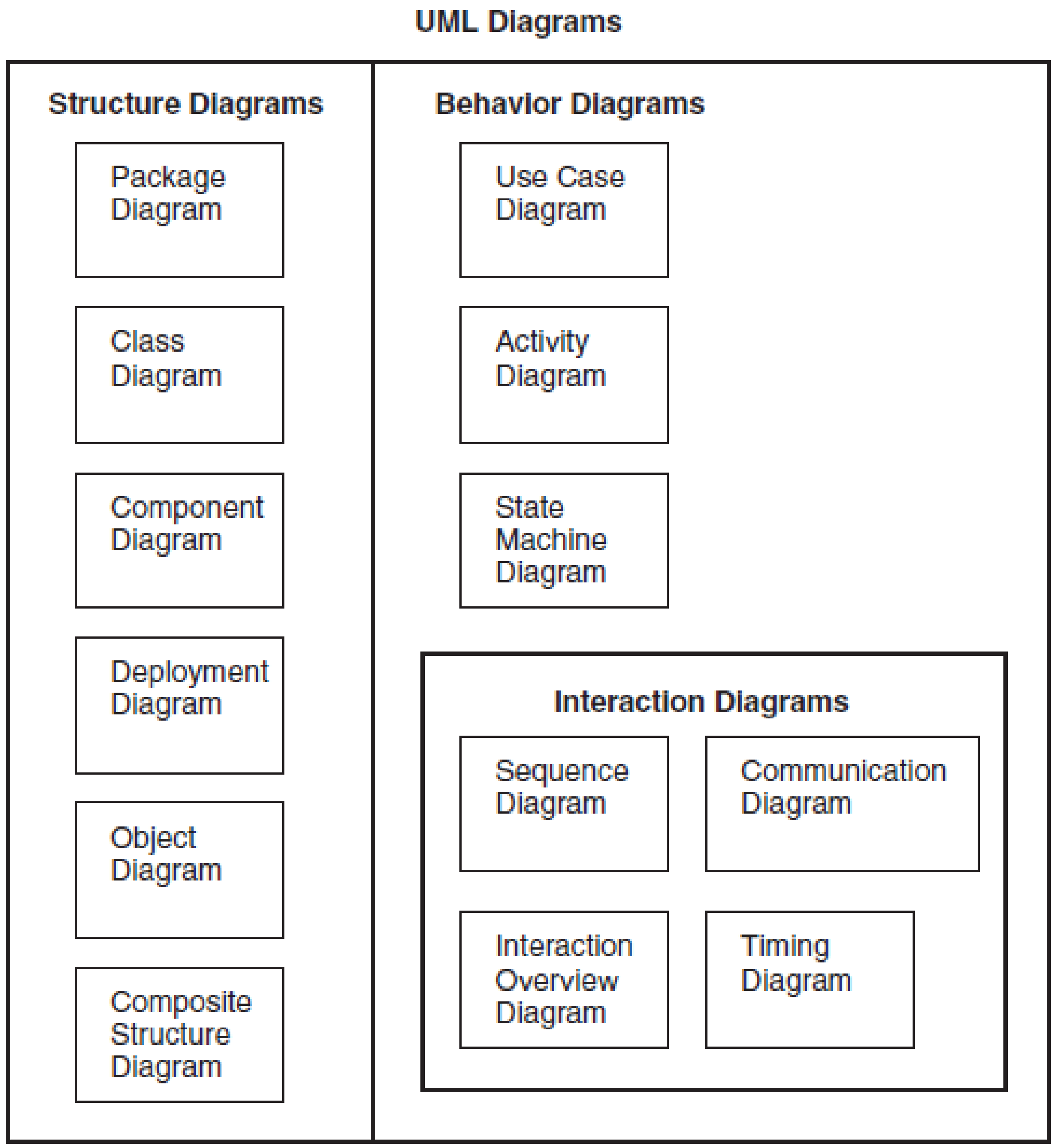
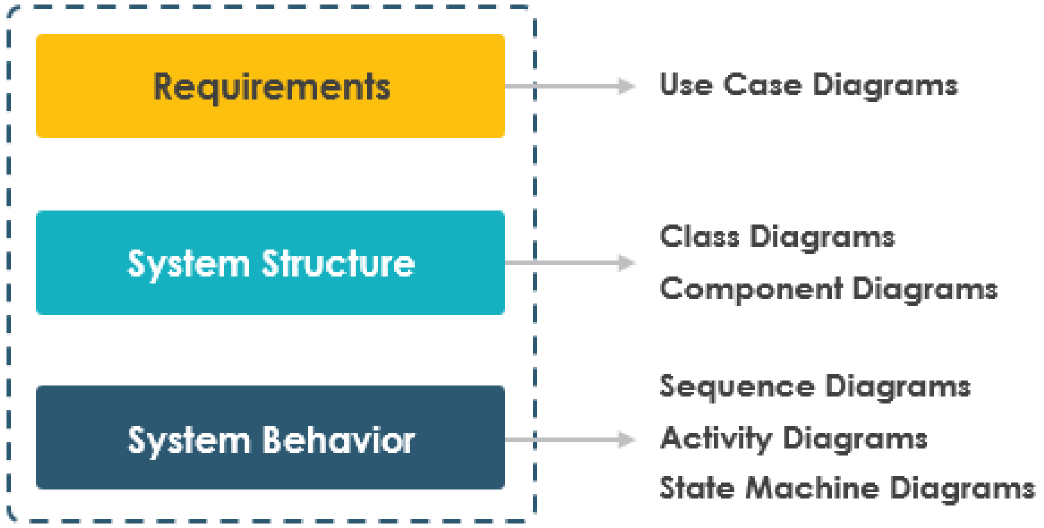
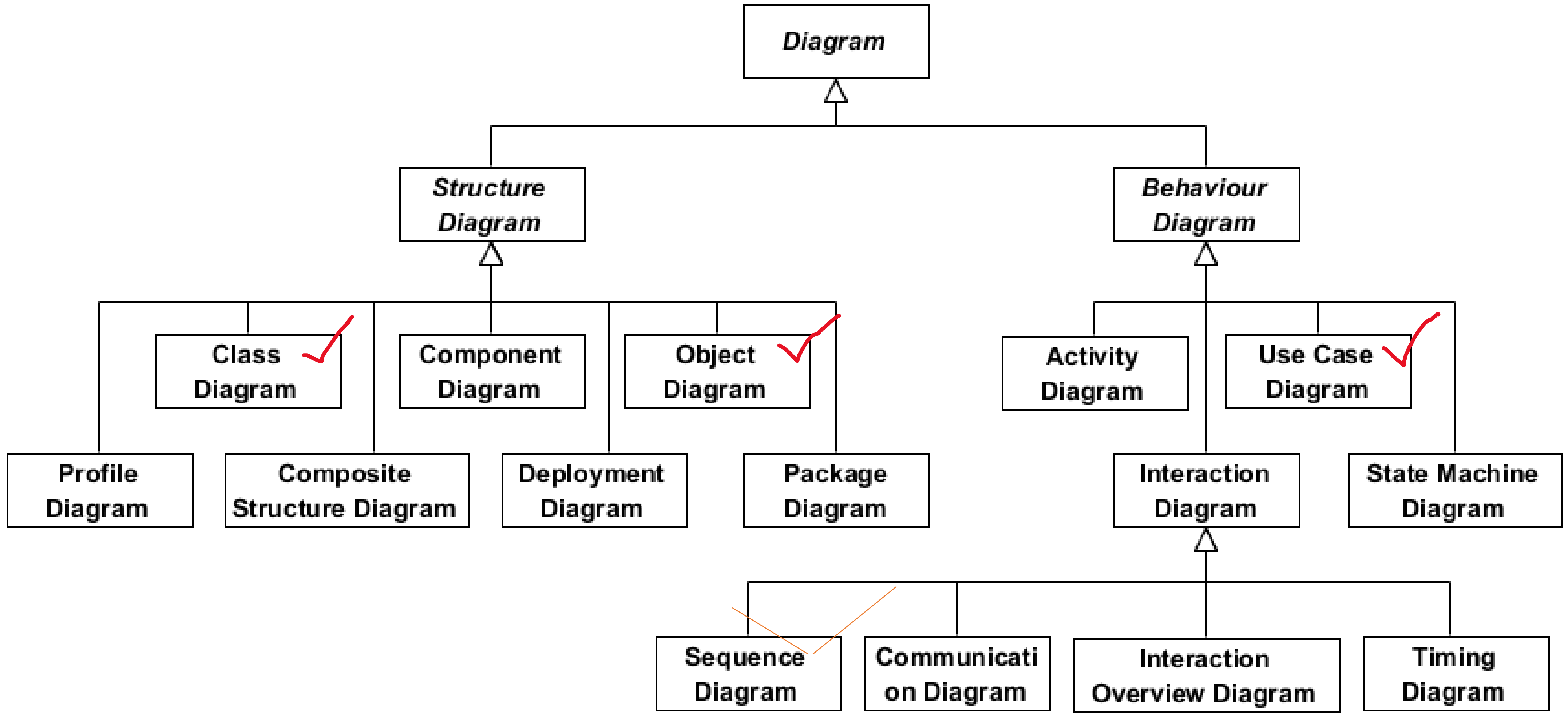


Figure 5–1 The Diagrams of the UML



UML Diagrams



PC: https://www.visual-paradigm.com/

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

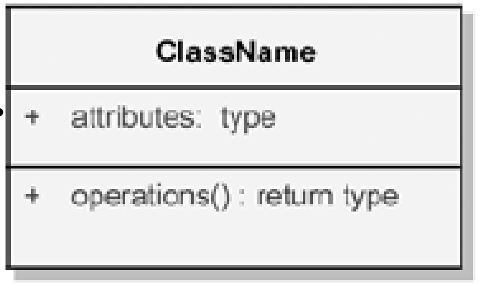
- **Step 1**: Identify the class names
  - The first step is to identify the primary objects of the system.
- **Step 2**: Distinguish relationships
  - Next step is to determine how each of the classes or objects are related to one another. Look out for commonalities and abstractions among them; this 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.

Attribute specification format:

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



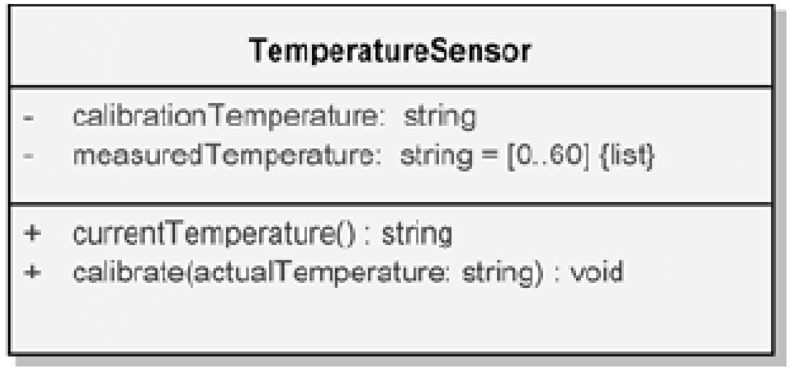

TemperatureSensor
- calibrationTemperature: string
- measuredTemperature: string = [0..60] {list}
+ currentTemperature() : string
+ calibrate(actualTemperature: string) : void

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

Operation specification format:

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

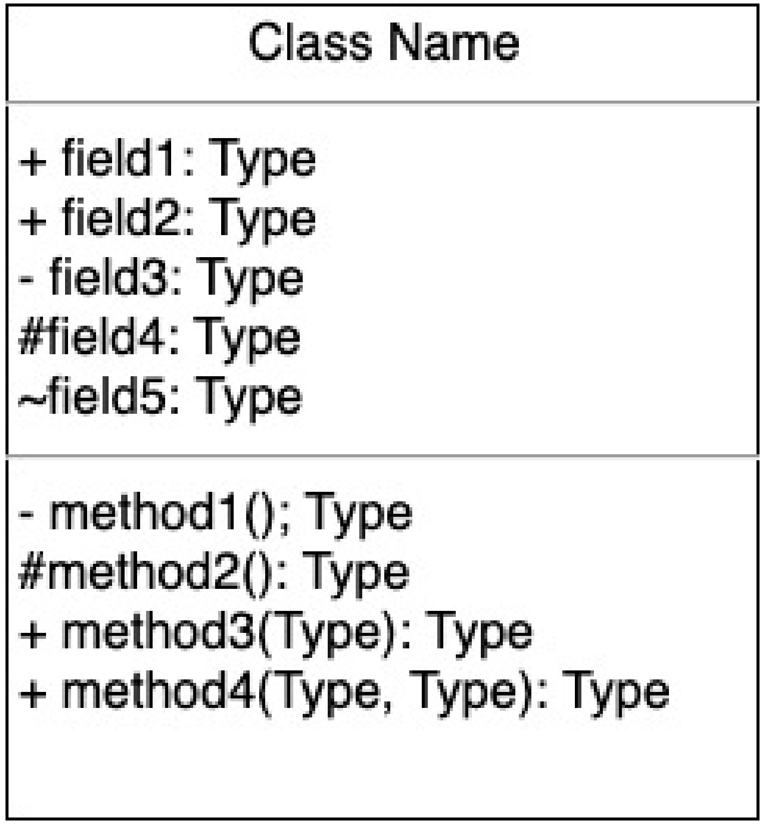
multiplicity of `[0..60]` on the `measuredTemperature` attribute indicates an array of 0 to 60 temperature measurements

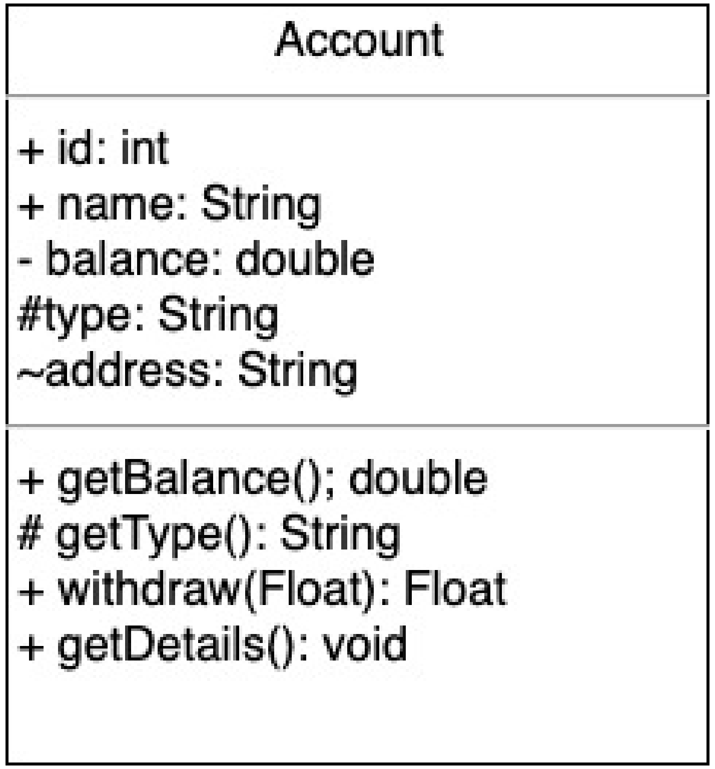
- We italicize the class name to show that we may have only instances of its subclasses., italize the operation to show it is abstract
- Class name begins in capital letters, and the space between multiple words is omitted
- The first letter of the attribute and operation names is lowercase, with subsequent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

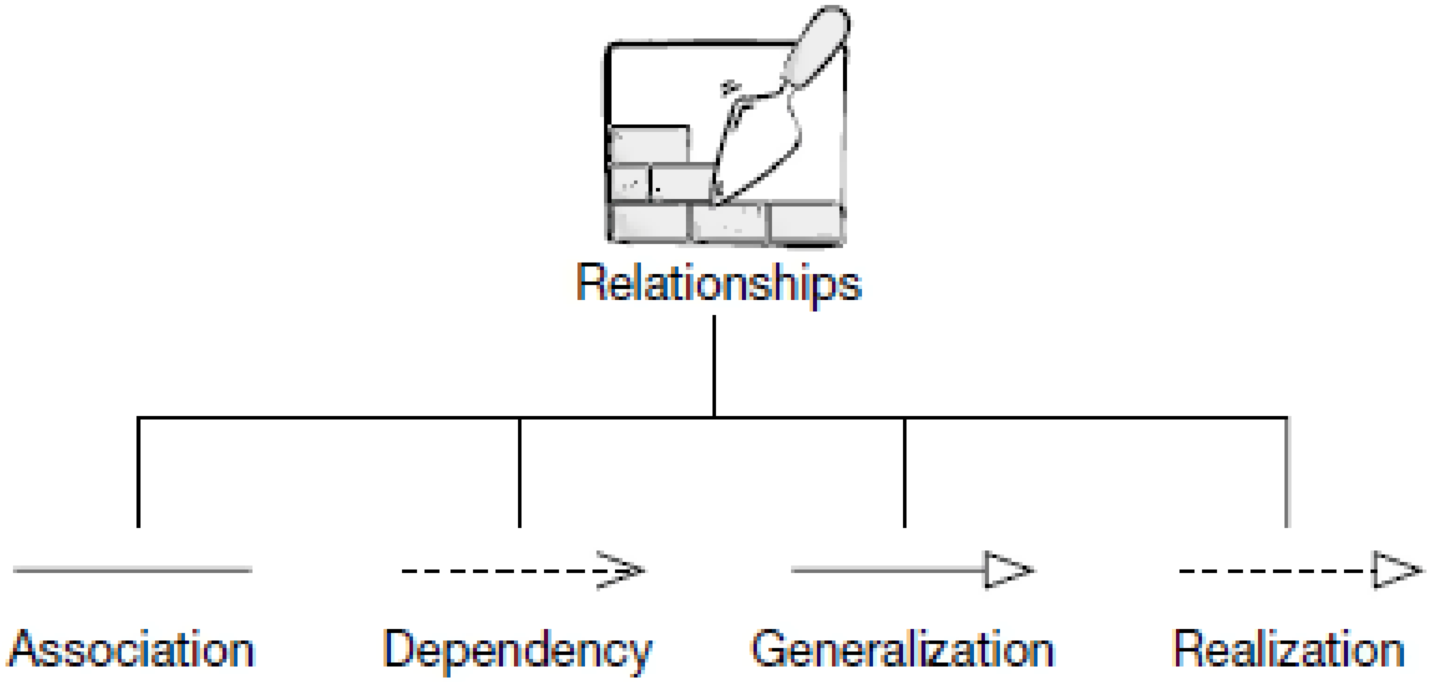
Update Class Diagram



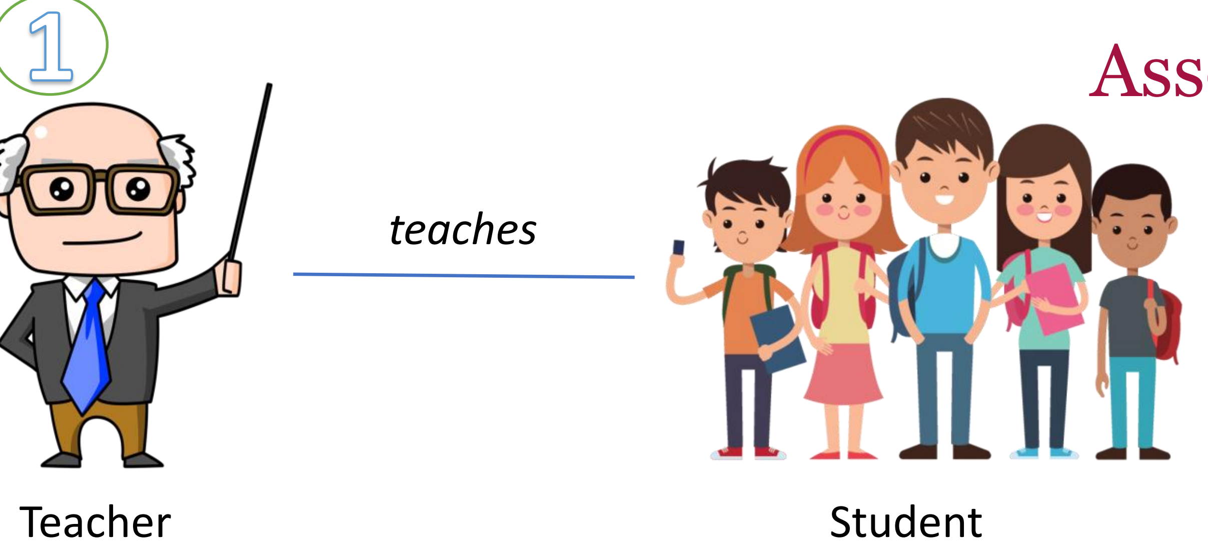


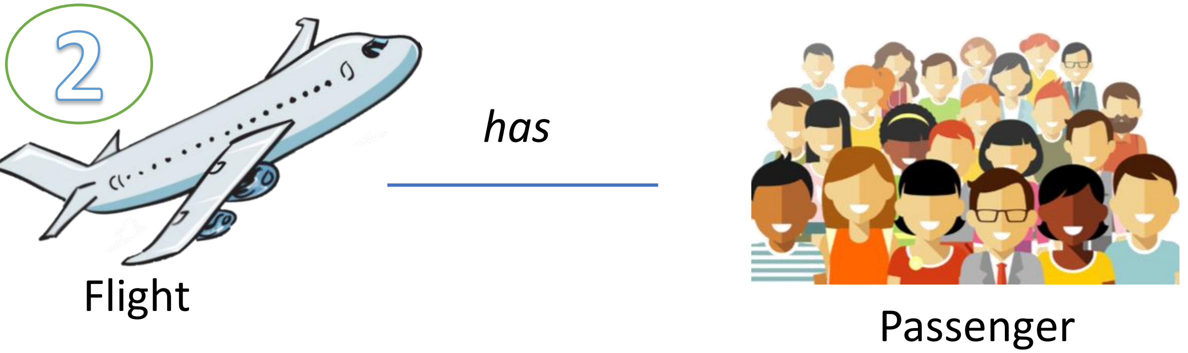
Relationships in UML

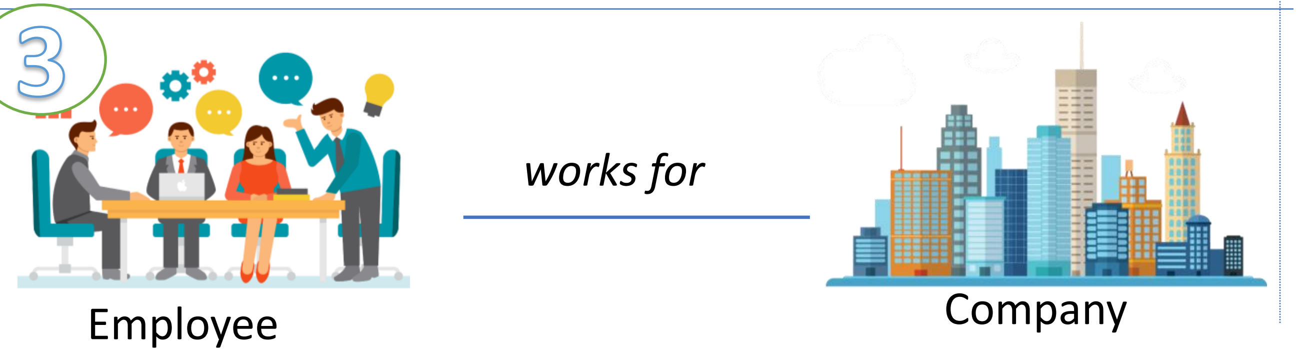
- There are mainly three kinds of relationships in UML:
- **Associations**
- **Generalizations**
- **Dependencies**

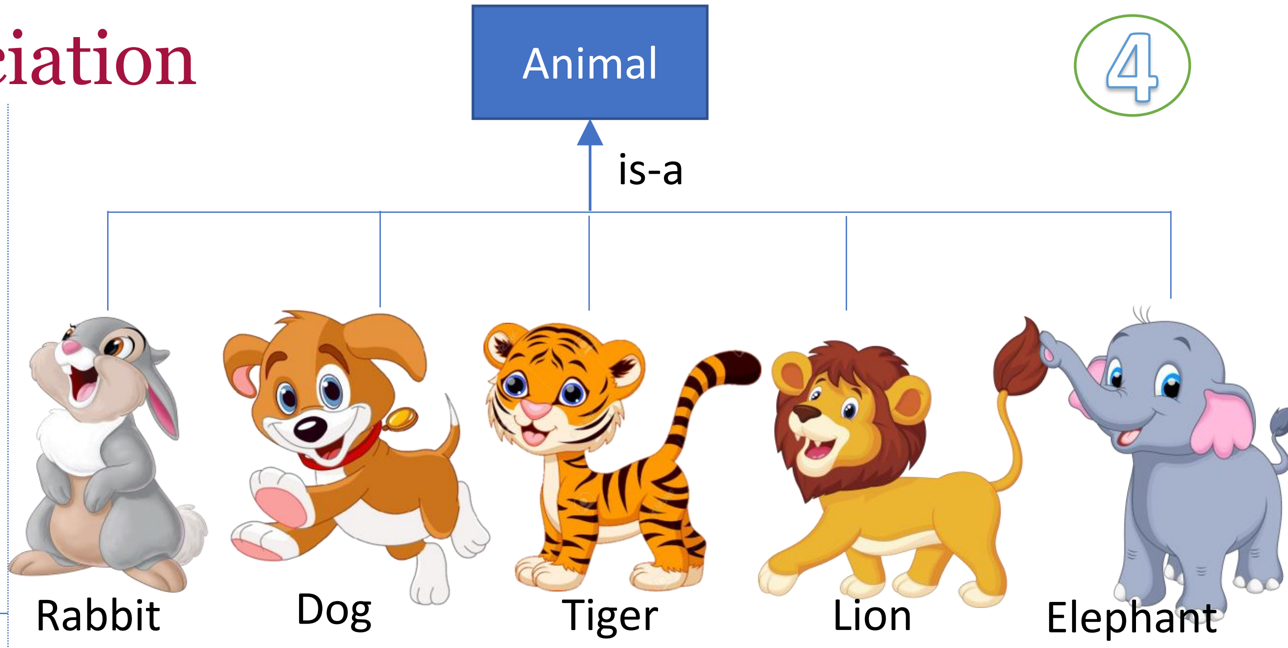


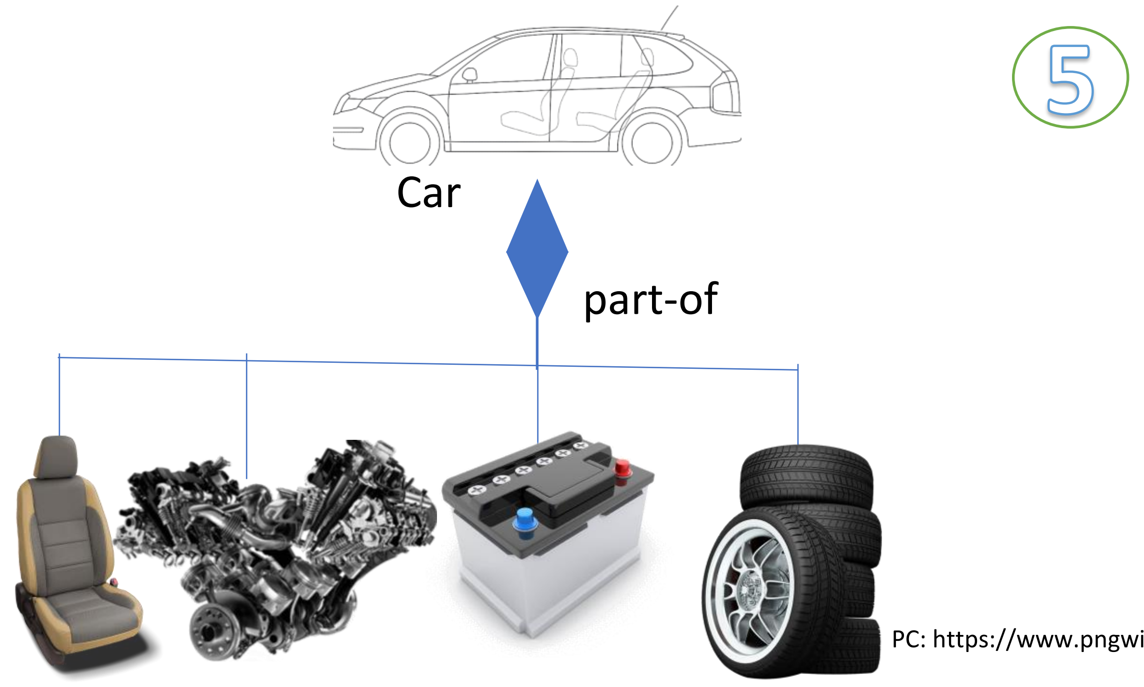
Association











PC: https://www.pngwing.com

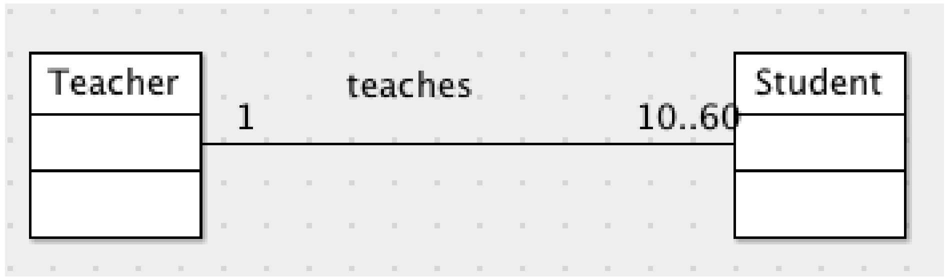
Association between Objects

- 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

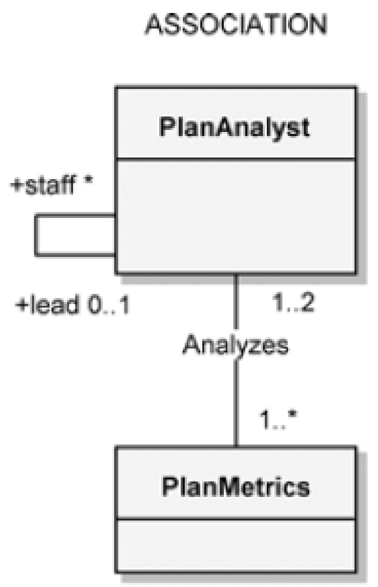
The association icon connects two classes and denotes a semantic connection

- 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.

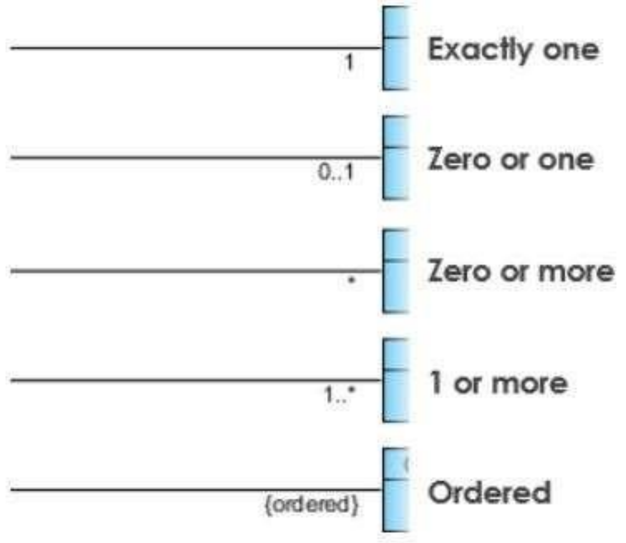


Multiplicity

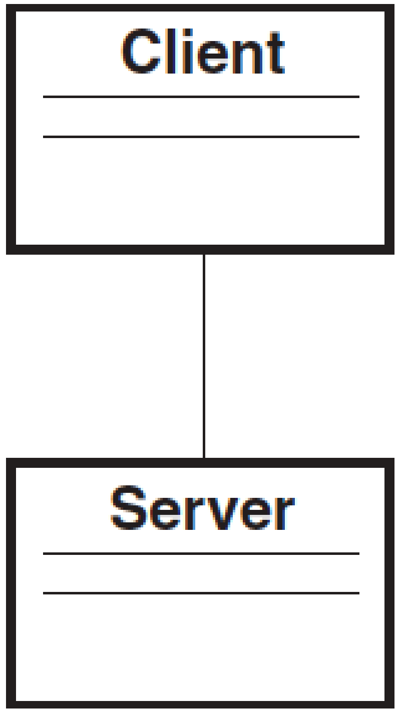
- The multiplicity adornment is applied to the target end of an association and denotes the number of links between each instance of the source class and instances of the target class
- Associations are often labeled with noun phrases, such as Analyzes, denoting the nature of the relationship. A class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

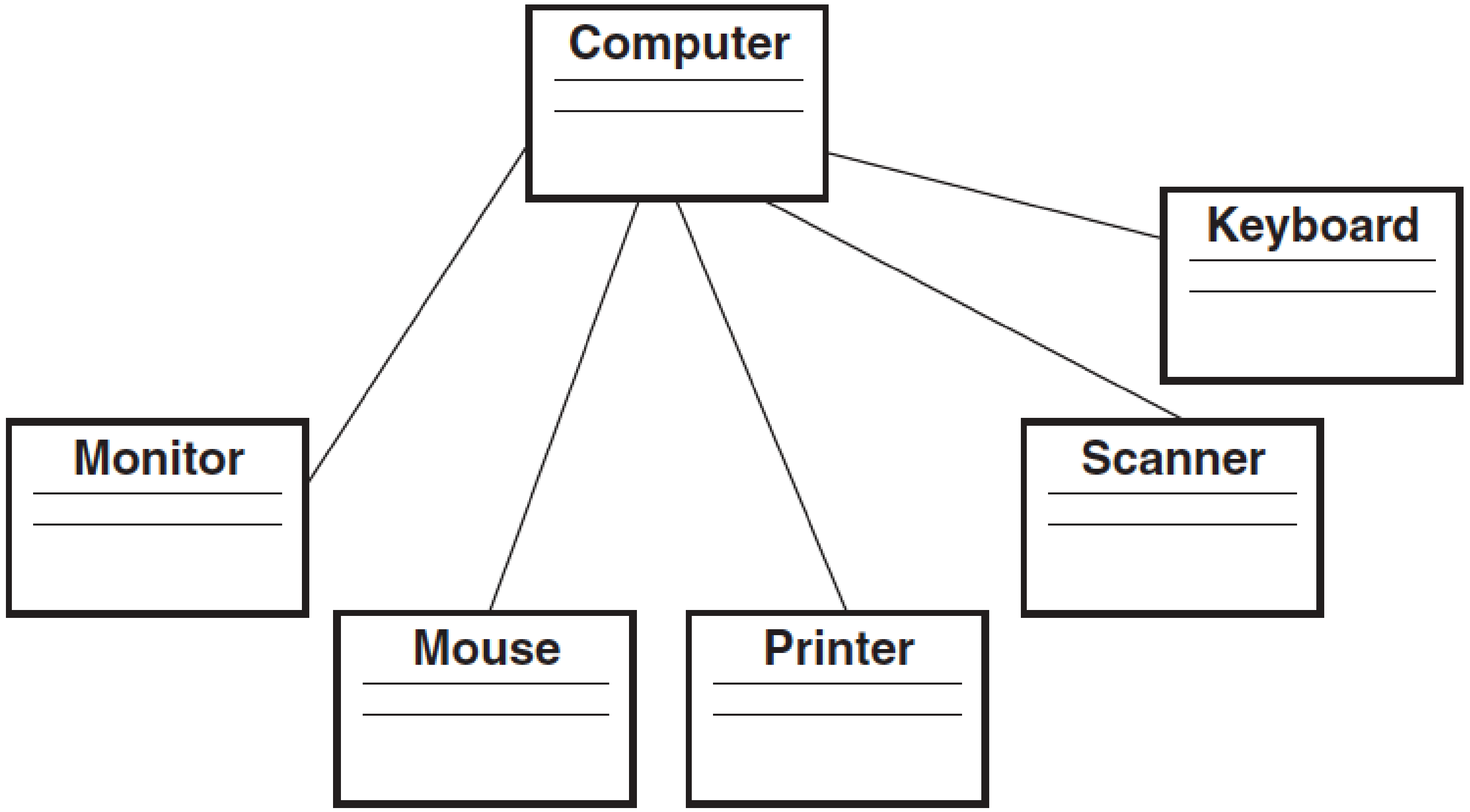


■ 1	Exactly one
■ *	Unlimited number (zero or more)
■ 0..*	Zero or more
■ 1..*	One or more
■ 0..1	Zero or one
■ 3..7	Specified range (from three through seven, inclusive)



Association Examples

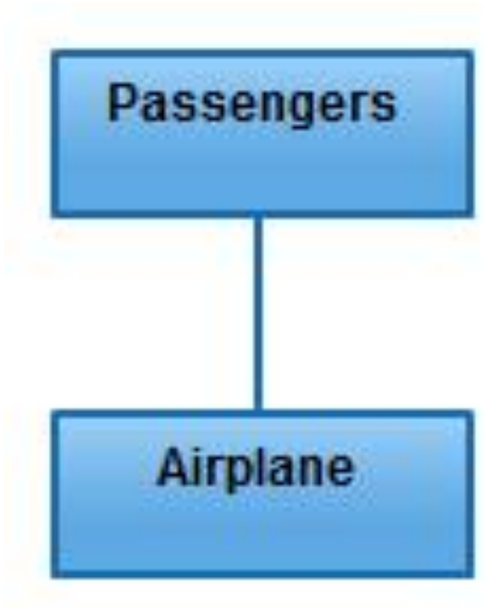




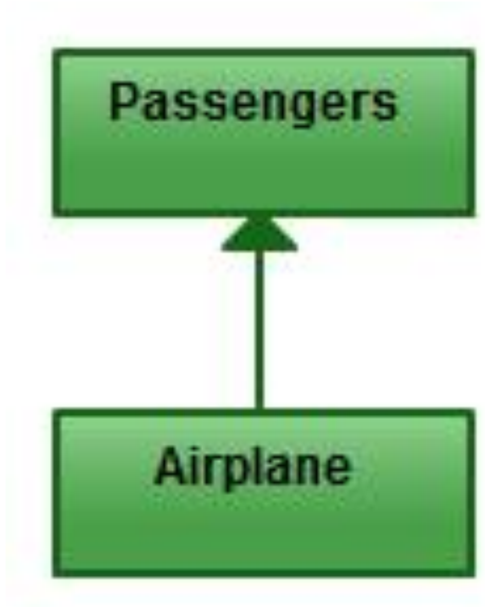
PC: Weisfeld Text 10.8 & 10.9

Association

Encompasses about any logical connection or relationship between classes.

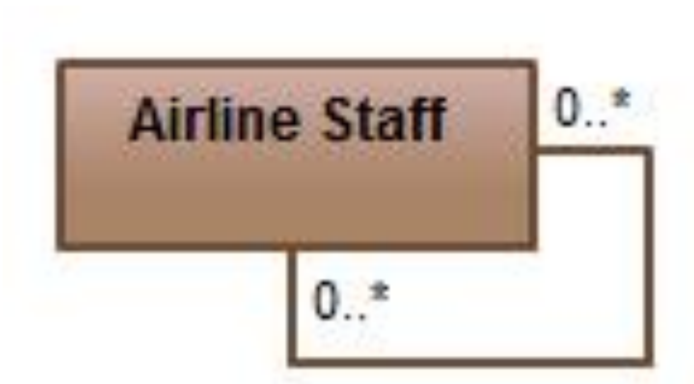


Association



Directed Association

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

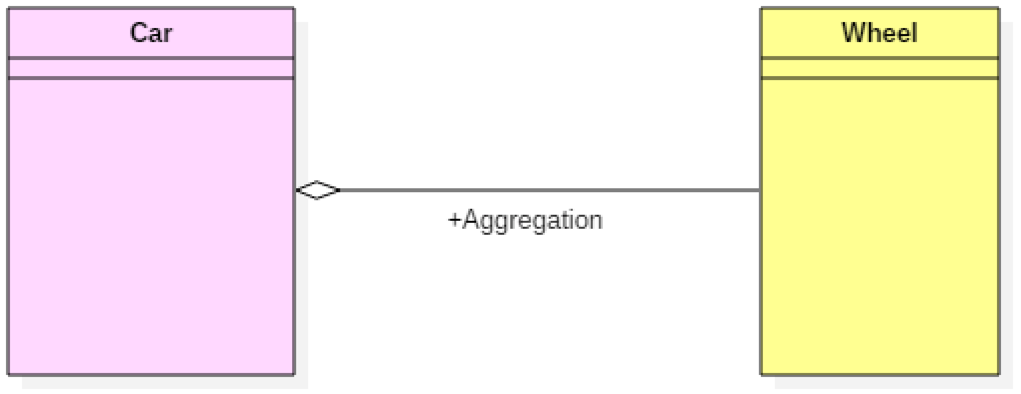


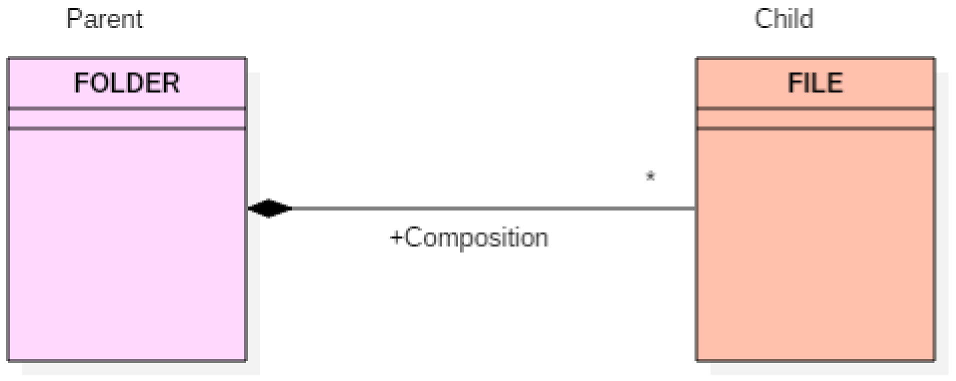
Reflexive Association

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.

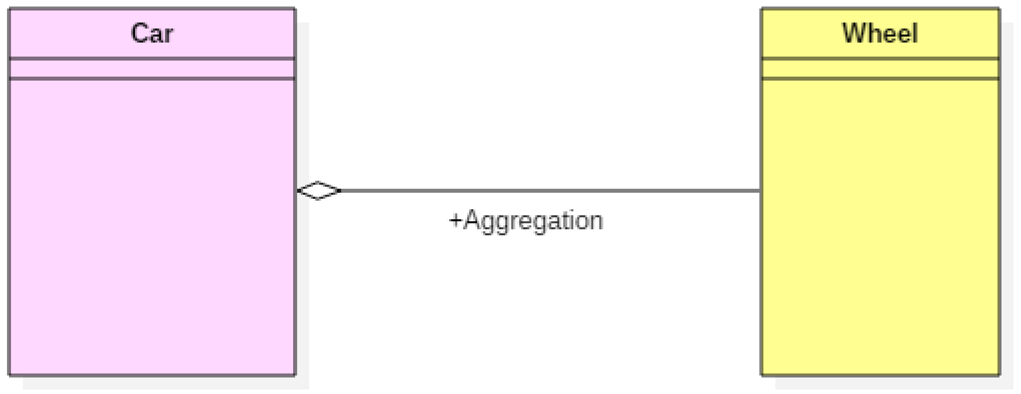
The composition is a special type of aggregation which denotes strong ownership between two classes when one class is a part of another class.

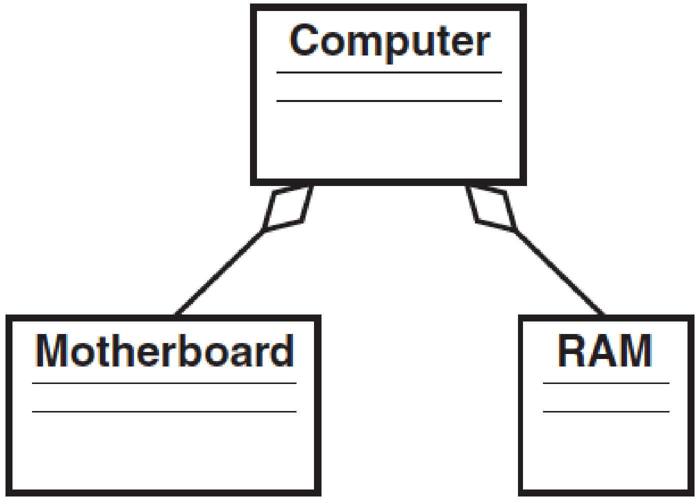
Association vs Aggregation vs Composition

- **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”**.

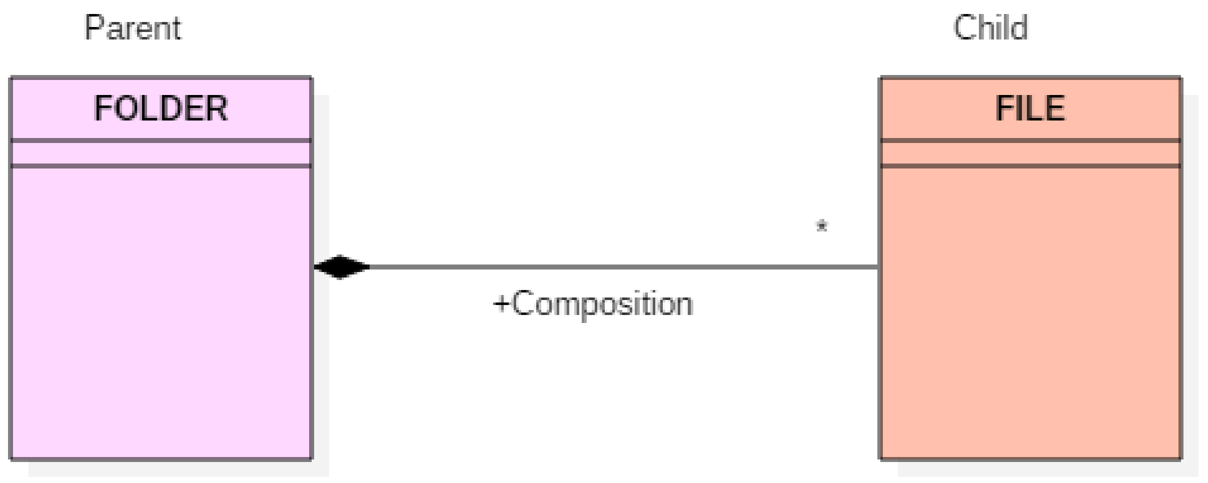




A car needs a wheel to function correctly, but a wheel doesn't always need a car. It can also be used with the bike, bicycle, or any other vehicles but not a particular car. Here, the wheel object is meaningful even without the car object. Such type of relationship is called an aggregation relation.

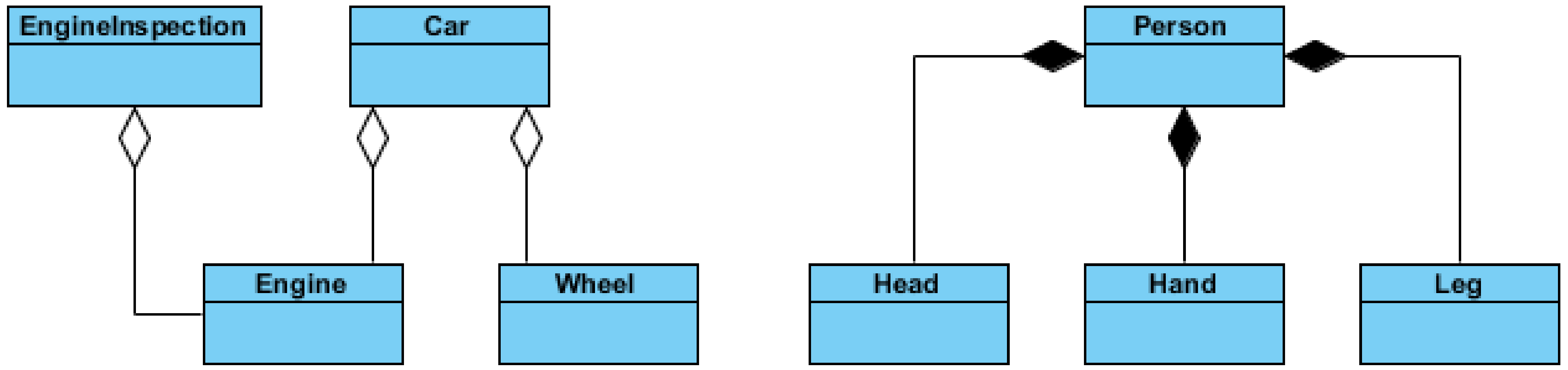
Composition- Strong aggregation relation

- 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.

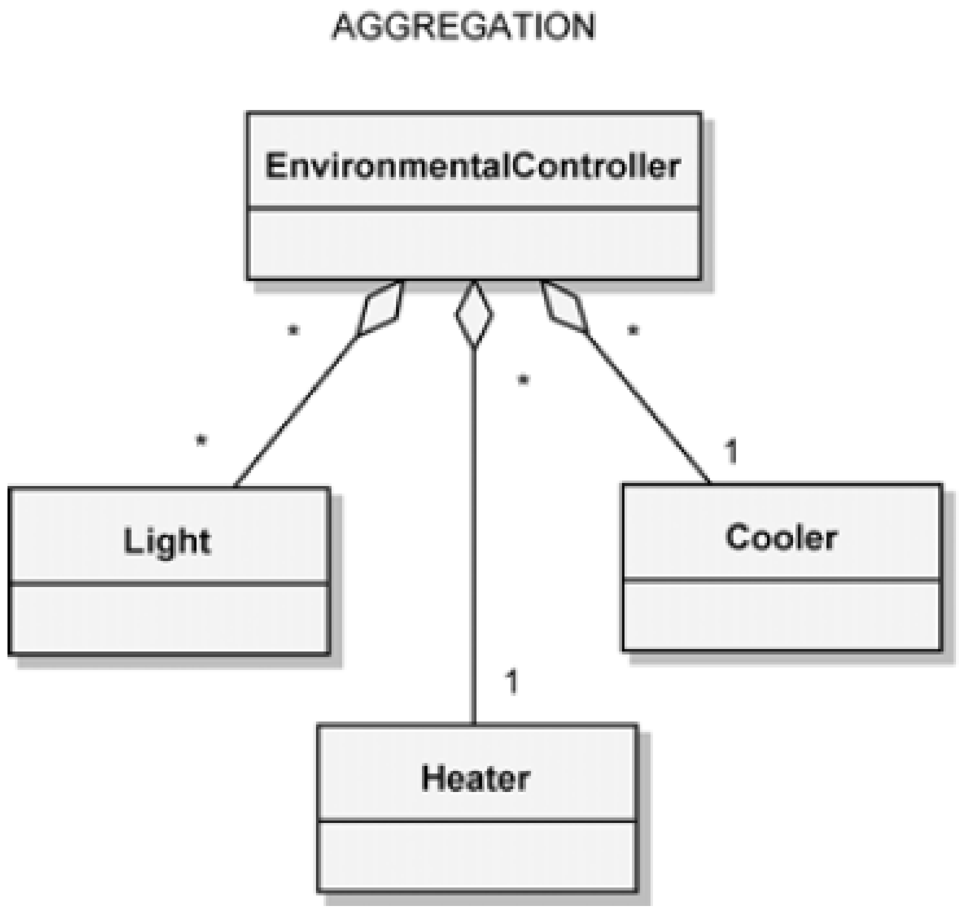


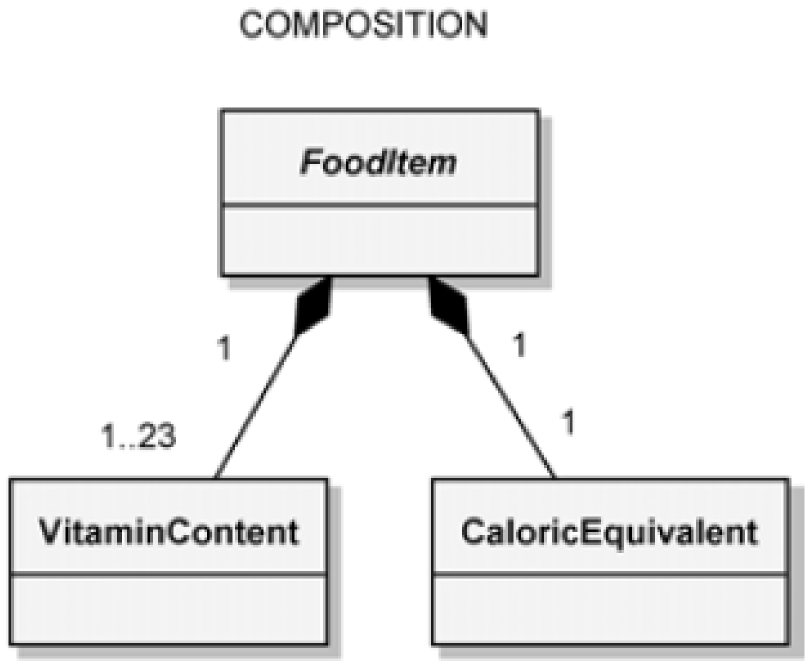
The folder could contain many files, while each File has exactly one Folder parent. If a folder is deleted, all contained files are removed as well.

Aggregation vs Composition



Aggregation vs Composition



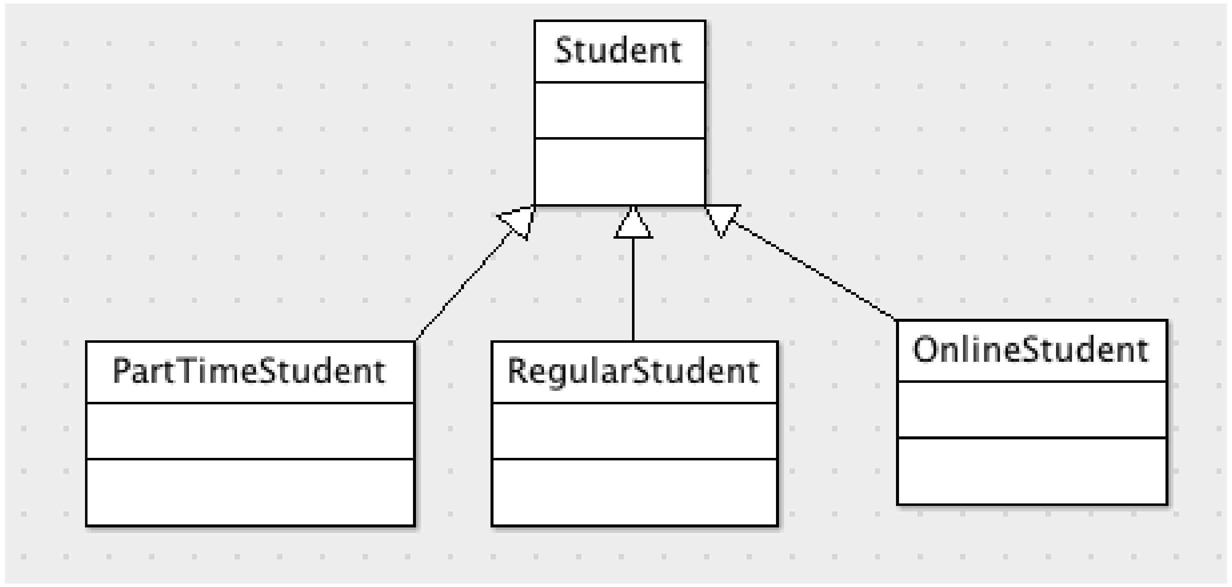


Generalization- “is-a”

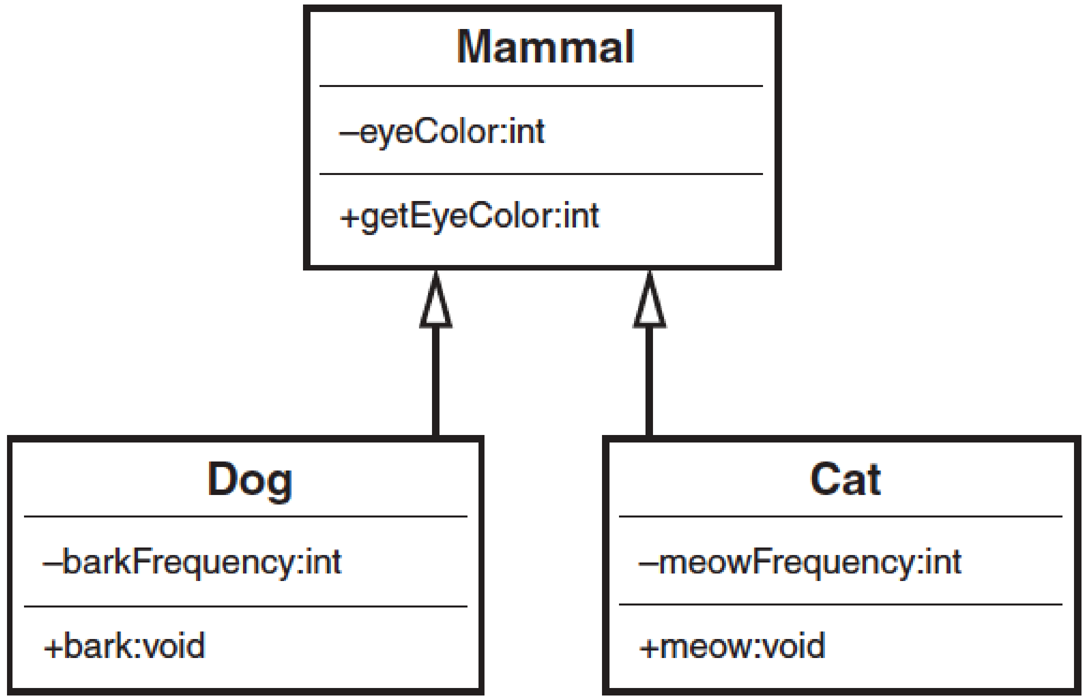
- Generalization is a relationship between a **general class** and a more **specific class**.
- “**is-a**” relation
- This relationship is achieved by an OO property : **Inheritance**.
  - General class : **Superclass** or **Parent** class or **Base** class
  - The specific/specialized class : **Subclass**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

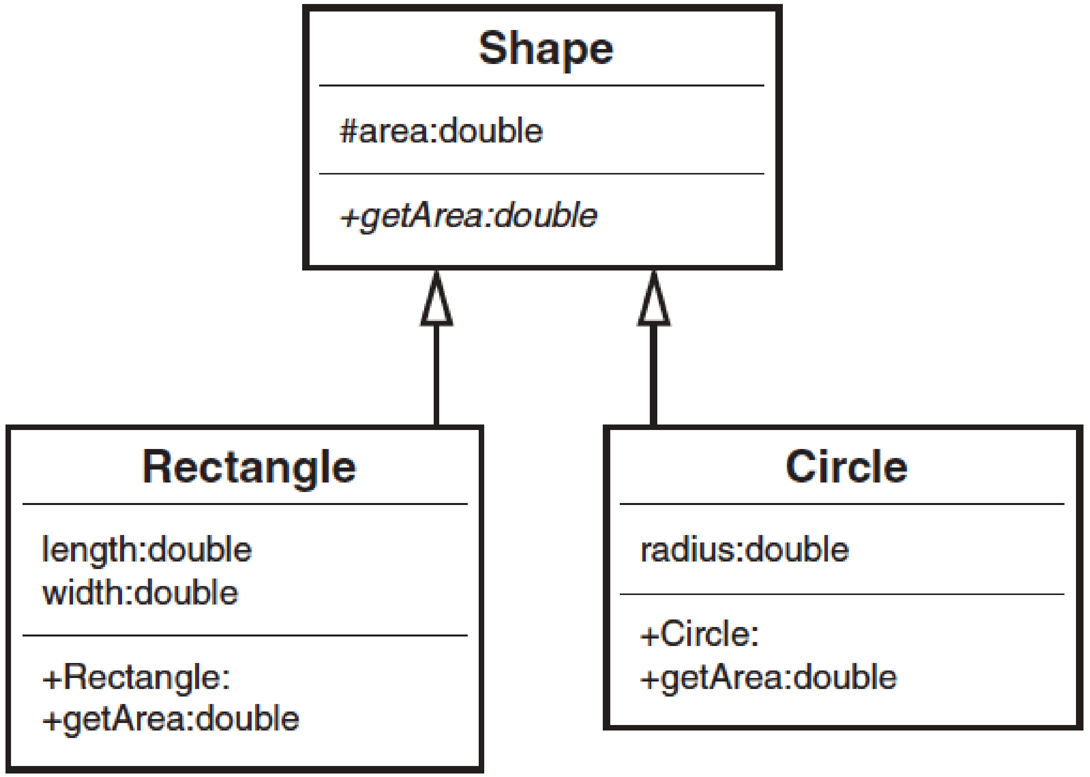
Generalization – Notation

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

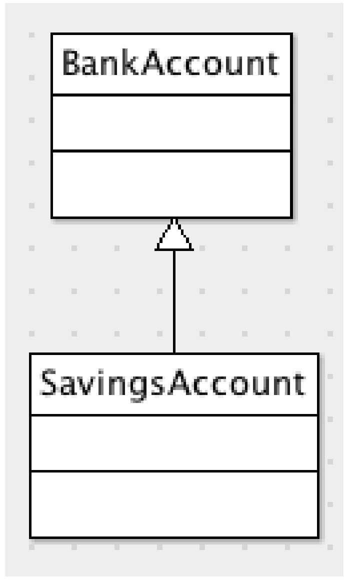


Inheritance – an example : is-a relation

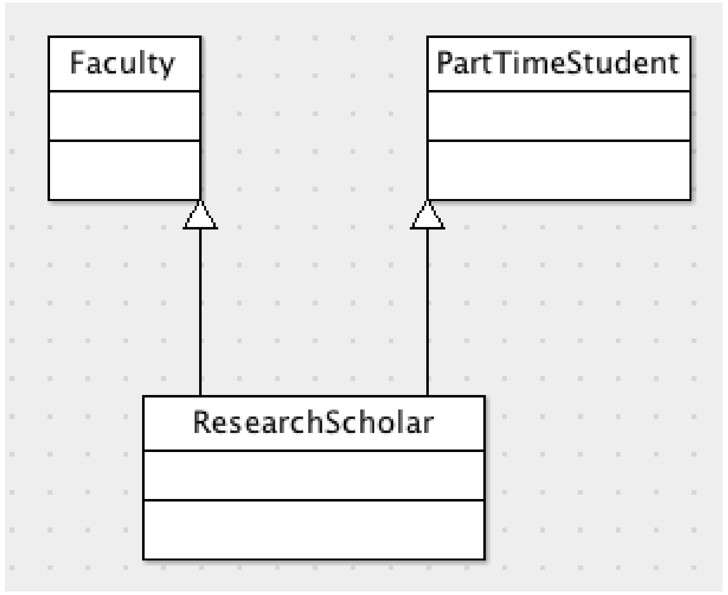




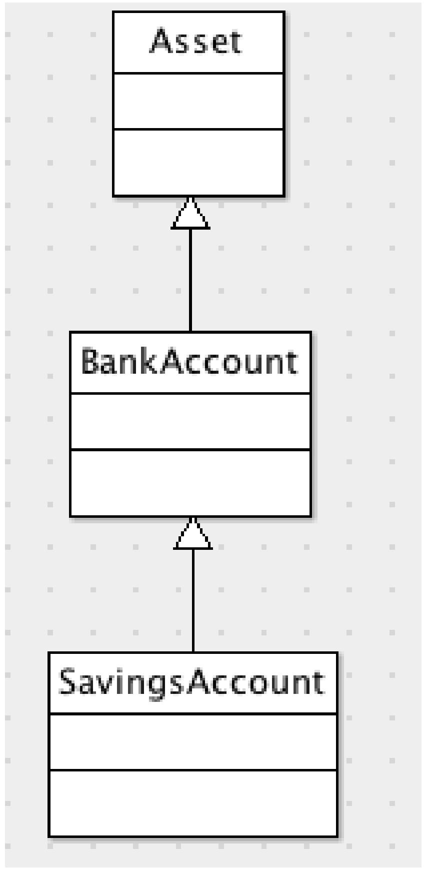
Types of Inheritance



Single Inheritance
A class has only
one superclass



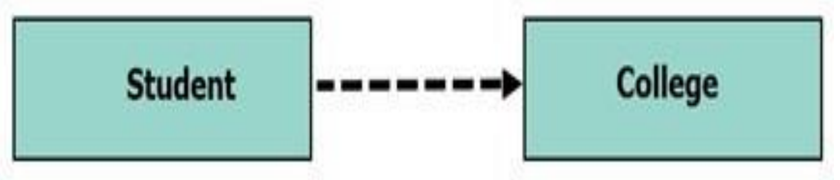
Multiple Inheritance
A class has
two or more superclasses

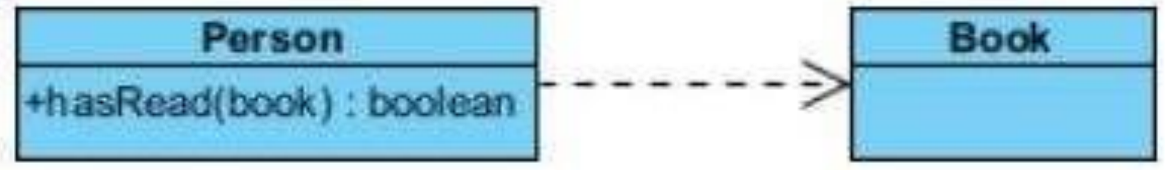


Multilevel Inheritance
A subclass can be
superclass to another class

Dependency

- **An object of one class might use an object of another class in the code of a method.** If the object is not stored in any field, then this is modelled as a dependency relationship\
- A dependency means the relation between two or more classes in which **a change in one may force changes in the other**. However, it will always create a weaker relationship.
- Dependency indicates that **one class depends on another**.

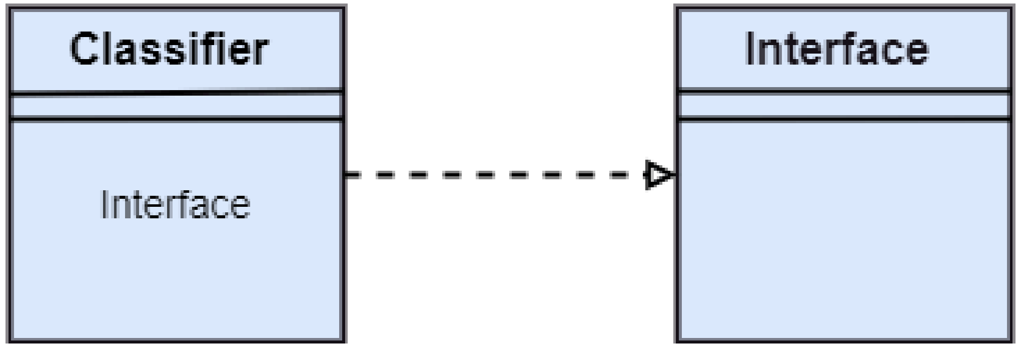




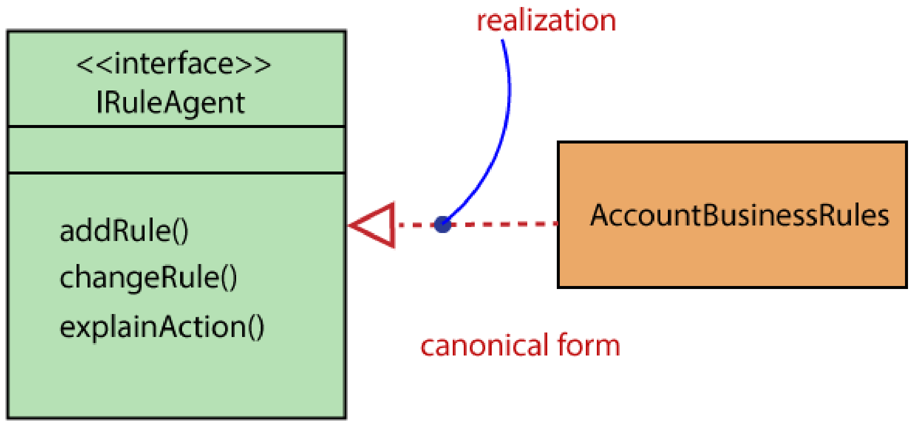
Student has a dependency on College

The Person class might have a hasRead method with a Book parameter that returns true if the person has read the book.

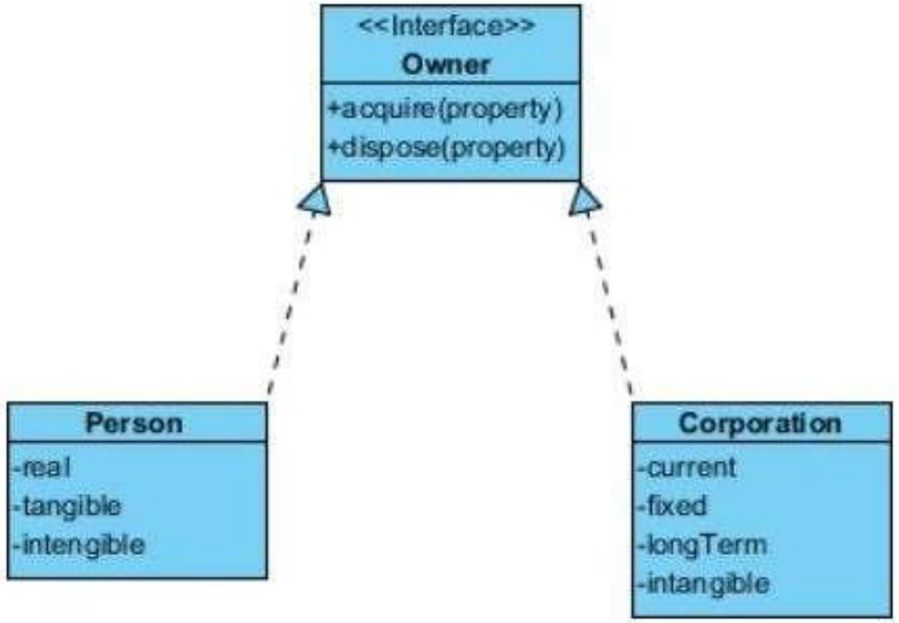
Realization



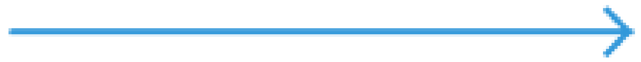
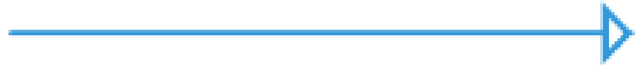
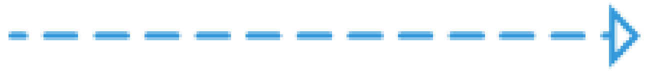
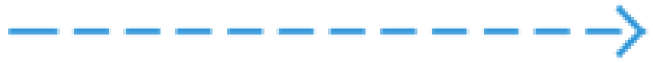
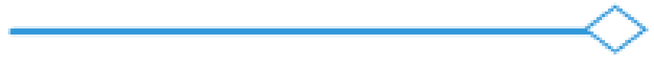
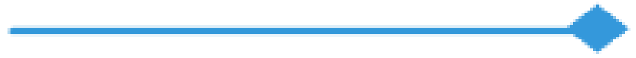
- Realization is a relationship between two objects, where the client (one model element) implements the responsibility specified by the supplier (another model element).



Account Business Rules realizes the interface **Iruleagent**

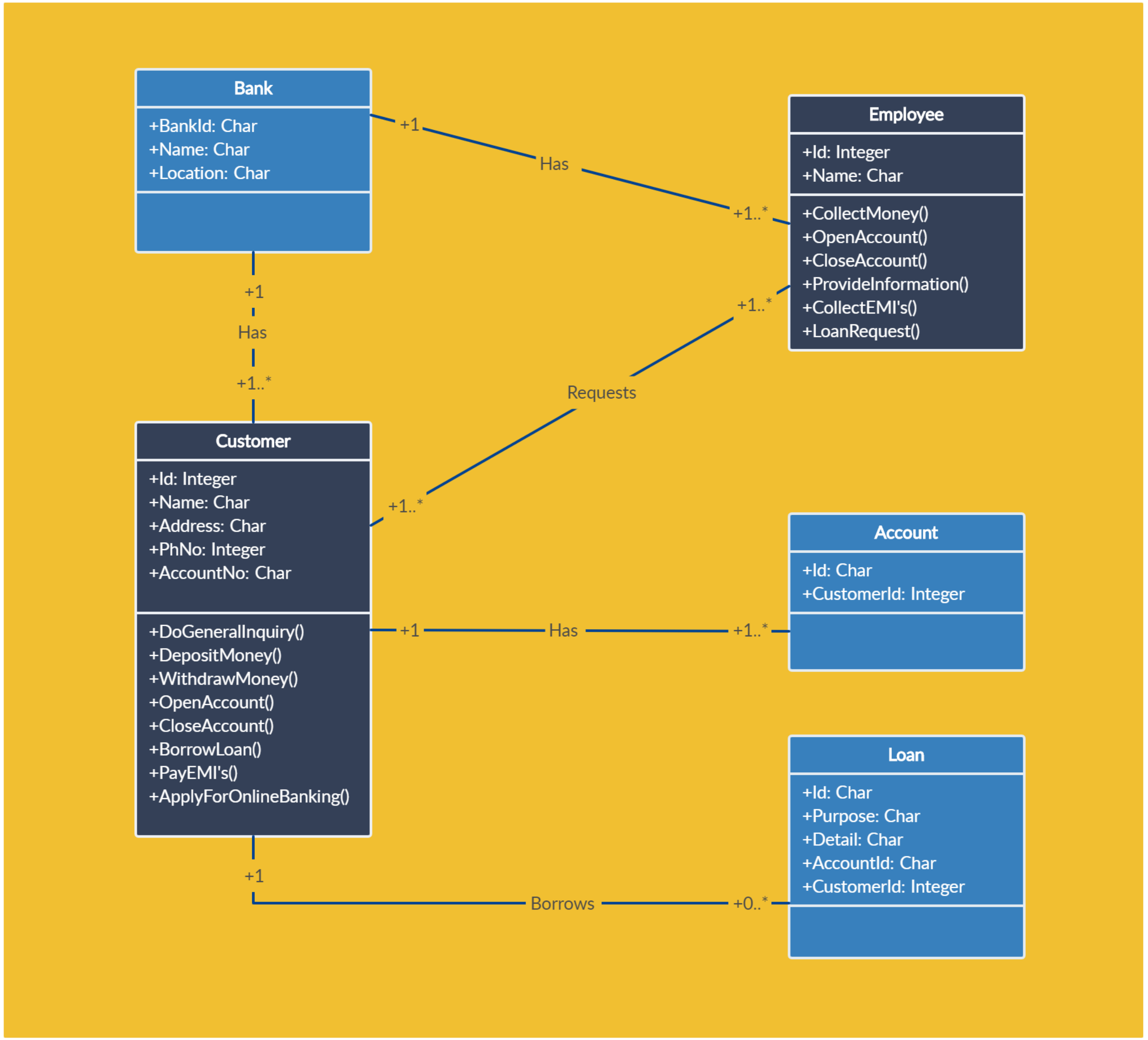


The Owner interface might specify methods for acquiring property and disposing of property. The Person and Corporation classes need to implement these methods, possibly in very different ways.

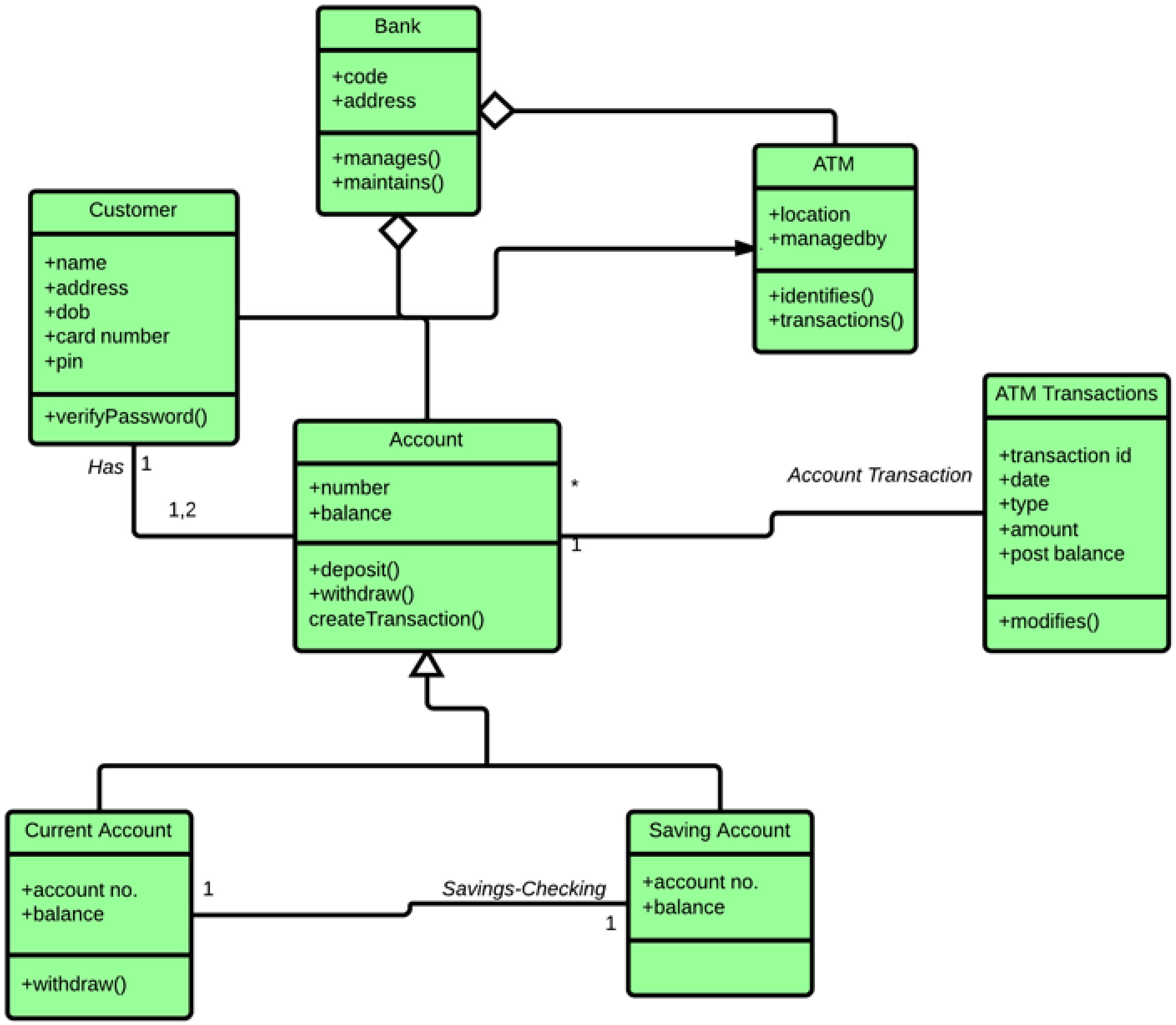
Class Diagram Relationship Type	Notation
Association	
Inheritance	
Realization/ Implementation	
Dependency	
Aggregation	
Composition	

Summary

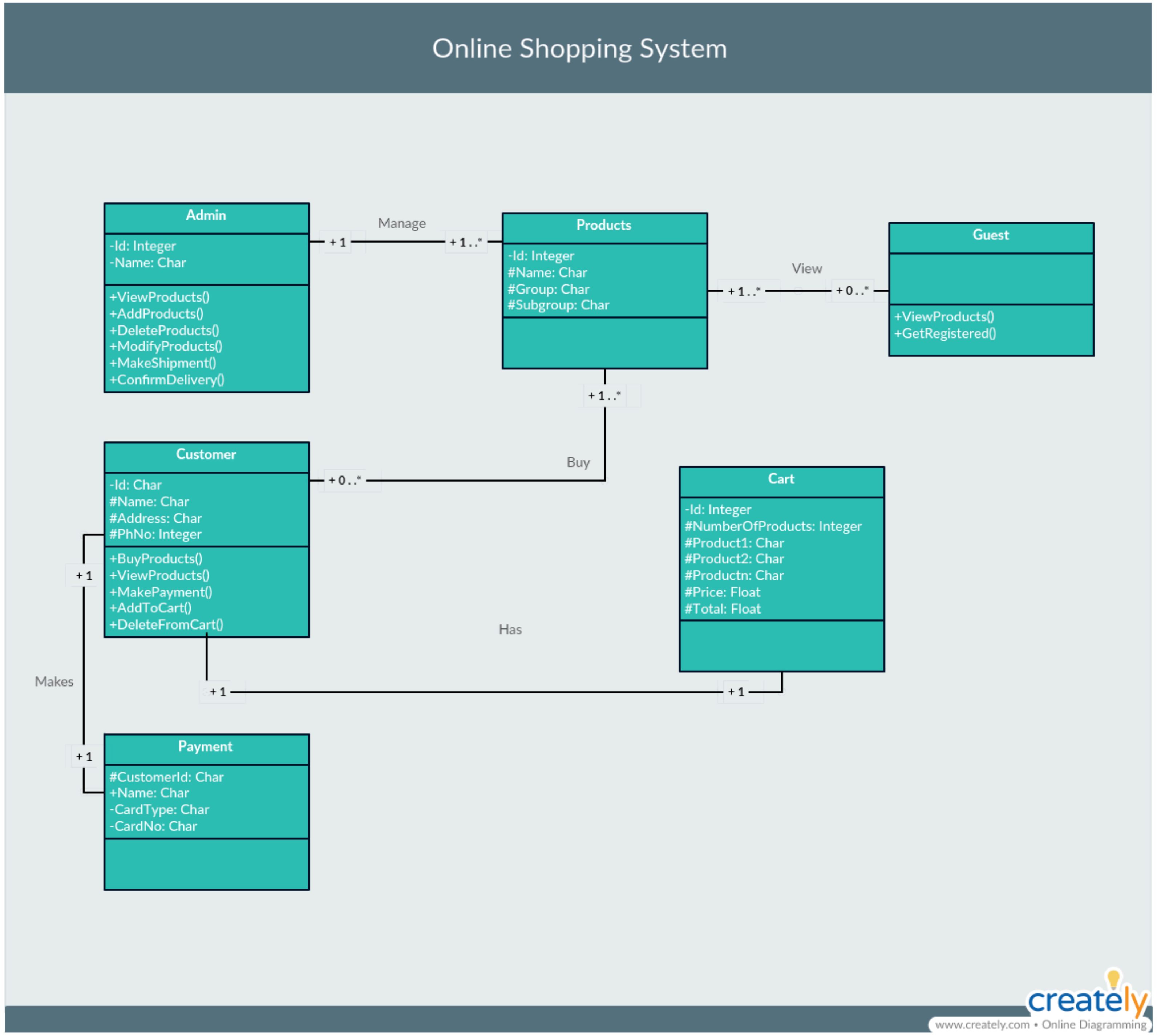
Banking System



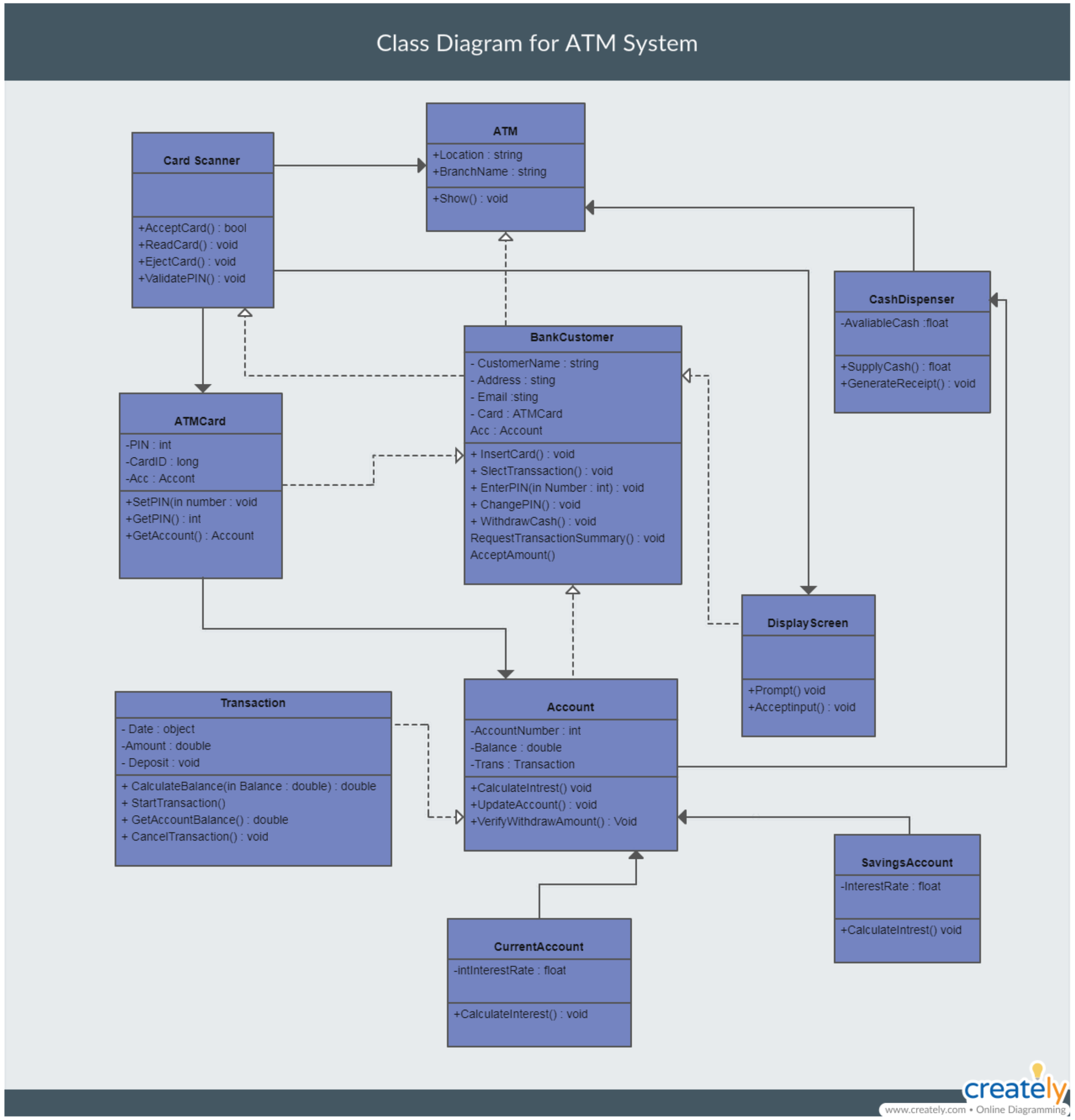
Banking System



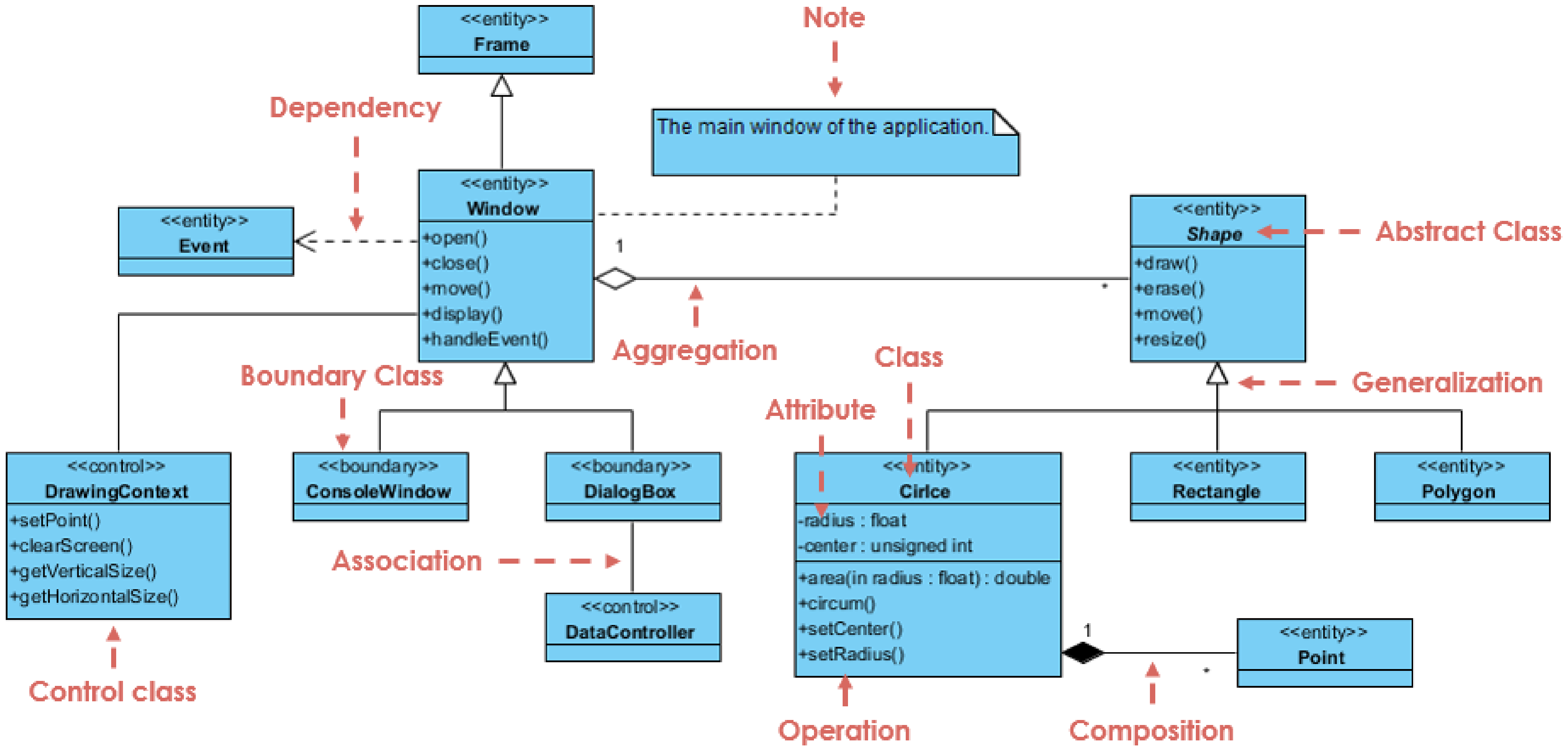
Online Shopping System



ATM System


Class Diagram for ATM System
Card Scanner
+AcceptCard() : bool
+ReadCard() : void
+EjectCard() : void
+ValidatePIN() : void
ATM
+Location : string
+BranchName : string
+Show() : void
CashDispenser
-AvaliableCash :float
+SupplyCash() : float
+GenerateReceipt() : void
BankCustomer
- CustomerName : string
- Address : sting
- Email :sting
- Card : ATMCard
Acc : Account
+ InsertCard() : void
+ SlectTransaction() : void
+ EnterPIN(in Number : int) : void
+ ChangePIN() : void
+ WithdrawCash() : void
RequestTransactionSummary() : void
AcceptAmount()
ATMCard
-PIN : int
-CardID : long
-Acc : Accont
+SetPIN(in number : void
+GetPIN() : int
+GetAccount() : Account
DisplayScreen
+Prompt() void
+Acceptinput() : void
Transaction
- Date : object
-Amount : double
- Deposit : void
+ CalculateBalance(in Balance : double) : double
+ StartTransaction()
+ GetAccountBalance() : double
+ CancelTransaction() : void
Account
-AccountNumber : int
-Balance : double
-Trans : Transaction
+CalculateIntrest() void
+UpdateAccount() : void
+VerifyWithdrawAmount() : Void
SavingsAccount
-InterestRate : float
+CalculateIntrest() void
CurrentAccount
-intInterestRate : float
+CalculateInterest() : void
creately
www.creately.com • Online Diagramming

UI Design



Note
Dependency
<<entity>>
Frame
The main window of the application.
<<entity>>
Window
+open()
+close()
+move()
+display()
+handleEvent()
<<entity>>
Event
1
*
<<entity>>
Shape
+draw()
+erase()
+move()
+resize()
Abstract Class
Aggregation
Boundary Class
Class
Generalization
Attribute
<<control>>
DrawingContext
+setPoint()
+clearScreen()
+getVerticalSize()
+getHorizontalSize()
<<boundary>>
ConsoleWindow
<<boundary>>
DialogBox
<<entity>>
Cirlce
-radius : float
-center : unsigned int
+area(in radius : float) : double
+circum()
+setCenter()
+setRadius()
<<entity>>
Rectangle
<<entity>>
Polygon
Association
<<control>>
DataController
1
*
<<entity>>
Point
Control class
Operation
Composition

- Shape is an abstract class. It is shown in Italics.
- Shape is a superclass. Circle, Rectangle and Polygon are derived from Shape. In other words, a Circle is-a Shape. This is a generalization / inheritance relationship.
- There is an association between DialogBox and DataController.
- Shape is part-of Window. This is an aggregation relationship. Shape can exist without Window.
- Point is part-of Circle. This is a composition relationship. Point cannot exist without a Circle.
- Window is dependent on Event. However, Event is not dependent on Window.
  - The attributes of Circle are radius and center. This is an entity class.
  - The method names of Circle are area(), circum(), setCenter() and setRadius().
  - The parameter radius in Circle is an in parameter of type float.
  - The method area() of class Circle returns a value of type double.
- The attributes and method names of Rectangle are hidden. Some other classes in the diagram also have their attributes and method names hidden.

Namah Shivaya