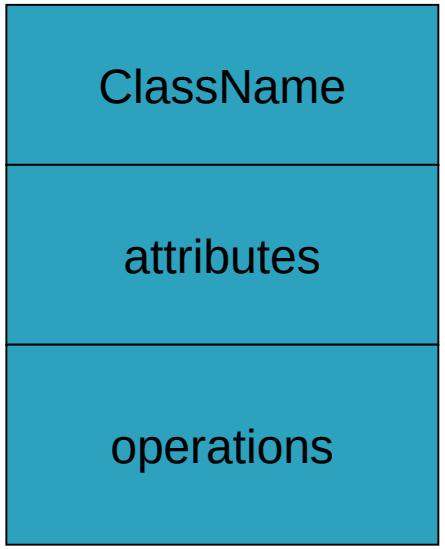
Unified Modeling Language (UML)

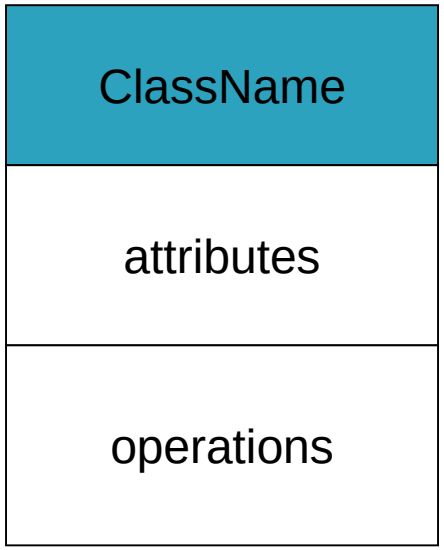
Class Diagram

Introduction

- A class diagram is a diagram used in designing and modeling software systems to describe classes and their relationships. Class diagrams enable us to model software in a high level of abstraction and without having to look at the source code.
- Classes in a class diagram correspond with classes in the source code.
- The diagram shows the names and attributes of the classes, connections between the classes, and sometimes also the methods of the classes.
- Class diagrams share many notations with ER diagrams and are often used to make ER diagrams
  - ER diagrams focus on data, not behavior
  - Class diagram includes behavior

Classes



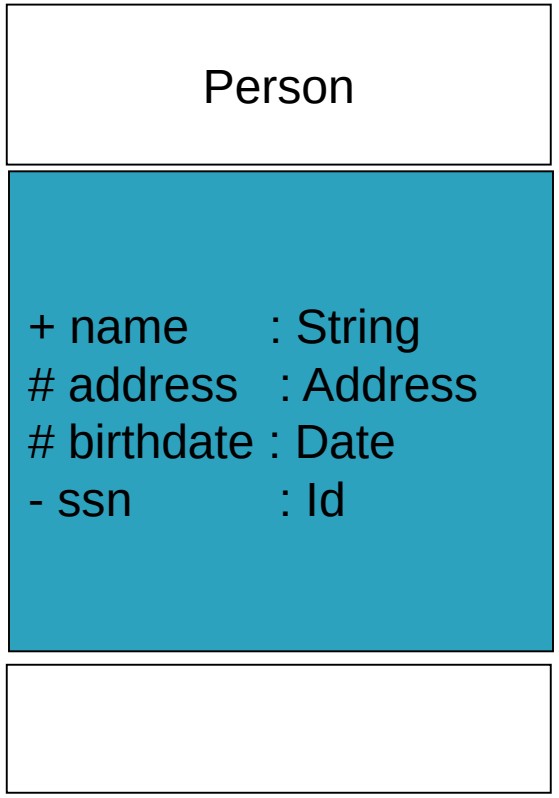


A *class* is a description of a set of objects that share the same attributes, operations, relationships, and semantics.

Graphically, a class is rendered as a rectangle, usually including its name, attributes, and operations in separate, designated compartments.

The name of the class is the only required tag in the graphical representation of a class. It always appears in the top-most compartment.
write <<interface>> on top of interfaces' names
use italics for an abstract class name

Class Attributes



An *attribute* is a named property of a class that describes the object being modeled. In the class diagram, attributes appear in the second compartment just below the name-compartment.

attributes (fields, instance variables)

visibility name : *type* = *default_value*

visibility:

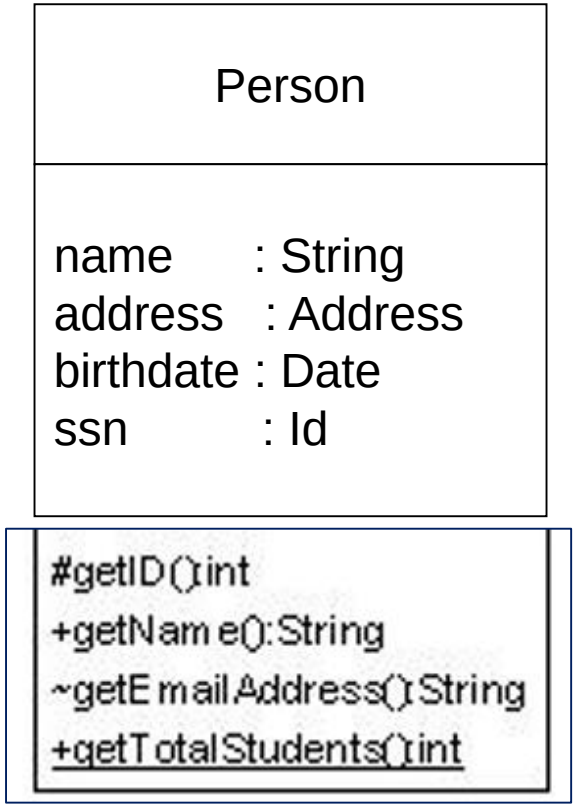
+	public
#	protected
-	private
~	package (default)

underline static attributes

attribute example:

- balance : double = 0.00

Class Operations

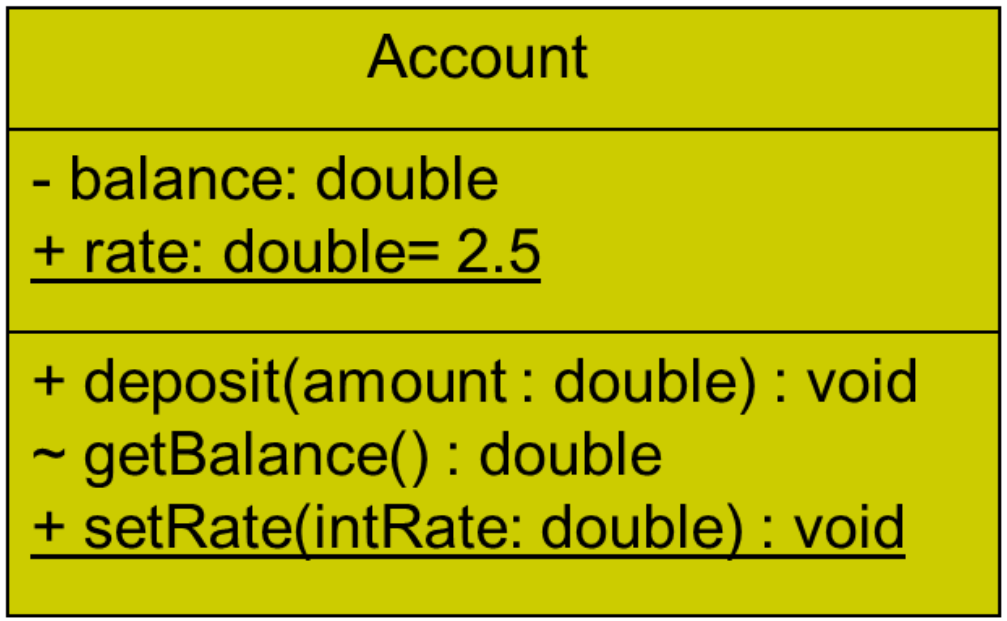


Operations describe the class behavior and appear in the third compartment.

1. You can specify an operation by stating its signature: listing the name, type, and default value of all parameters, and a return type.

2. Only add important methods: Not all methods should be added to the class diagram, methods like constructors, equals(), toString(), getters and setters should not be added to the diagram, those methods will not bring any information to the class diagram since every class should have them.

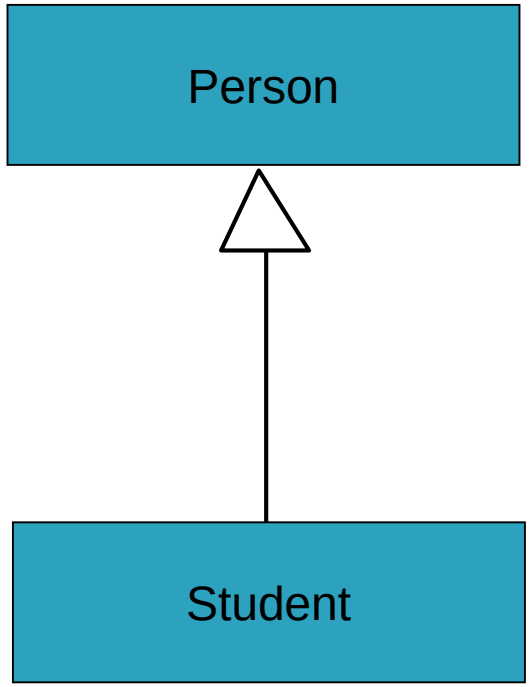
Example



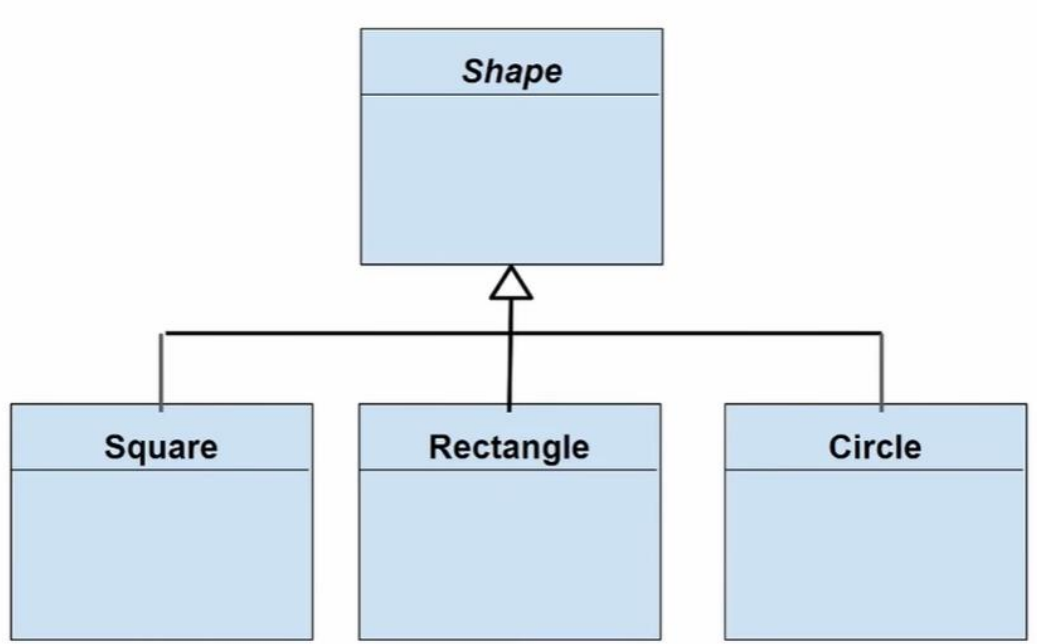

```
public class Account
{
   private double balance;
   public static double rate = 2.5;

   public void deposit (double amount)  {
                balance += amount;
   }
   /*package*/ double getBalance()  {
                return balance;
   }
   public static void setRate( double intRate ) {
                rate = intRate;
   }
}
```

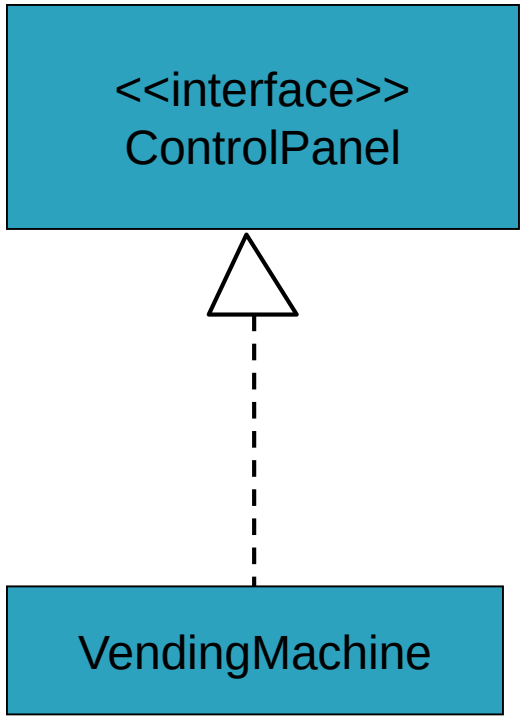
Generalization Relationships



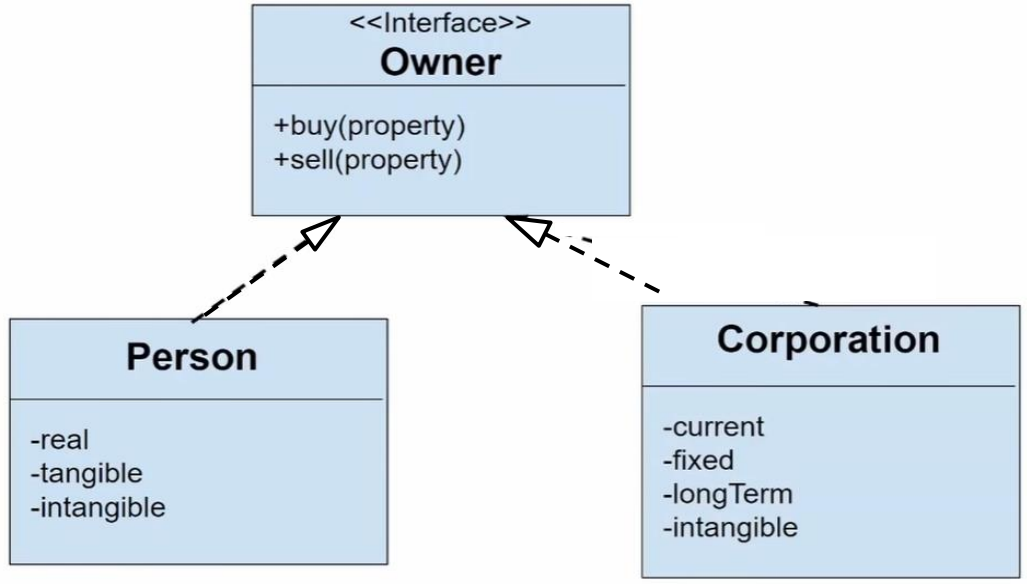
A generalization (inheritance) relationships connects a subclass to its superclass. It denotes an inheritance of attributes and behavior from the superclass to the subclass and indicates a specialization in the subclass of the more general superclass.



Interfaces Implementation



1. An *interface* is a named set of operations that specifies the behavior of objects without showing their inner structure.
2. It is a *realization* relationship connects a class with an interface that supplies its behavioral specification. It is rendered by a dashed line with a hollow triangle towards the specifier.



Relationships between Classes in UML

In UML, object interconnections are modeled as relationships.

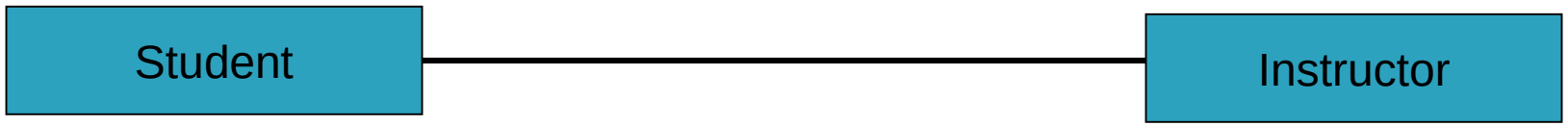
There are three kinds of relationships in UML:

- Association
- Composition
- Aggregation

Association Relationships

- If two classes in a model need to communicate with each other, there must be link between them.
- It is a relationship between two separate classes. An *association* denotes that link.
- We can model objects that contain other objects by special associations called:

  Aggregations and *Compositions.*

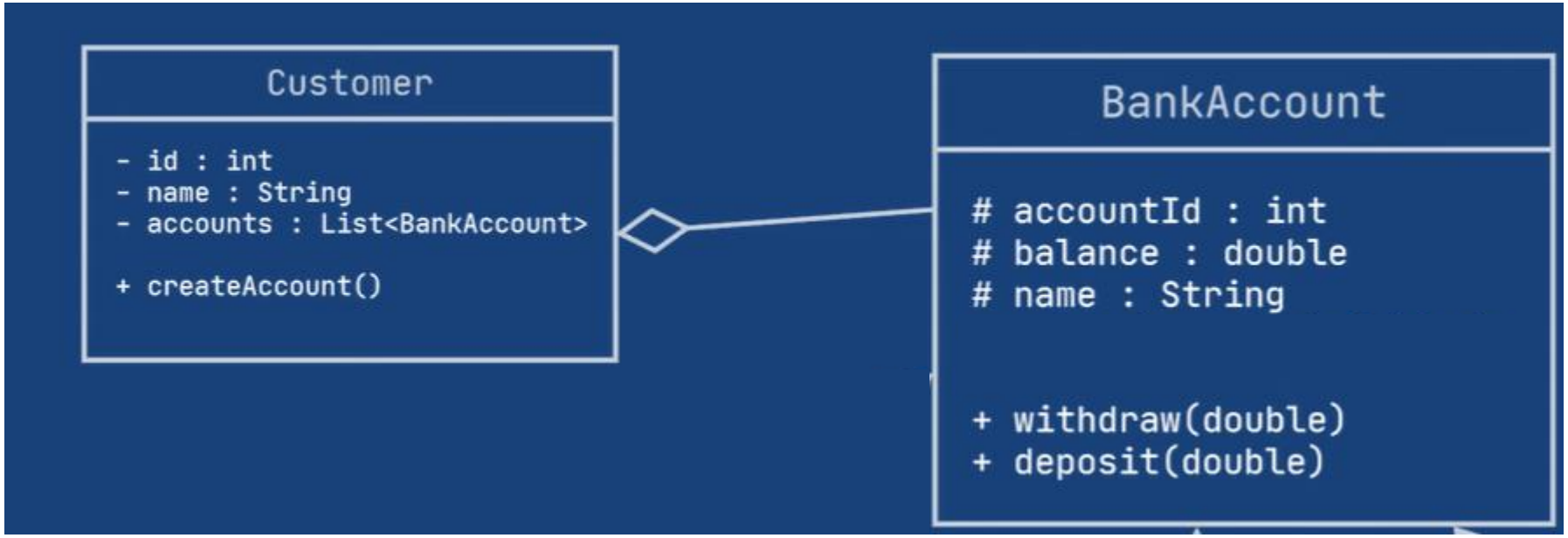


Aggregation

- Aggregation implies a relationship where the child can exist independently of the parent "is part of"
- An aggregation specifies a whole-part relationship between an aggregate, where the part can exist independently from the aggregate.
- The best way to understand this relationship is to call it a “has a” or “is part of” relationship
- Aggregations are denoted by a clear white diamond

Aggregation Example

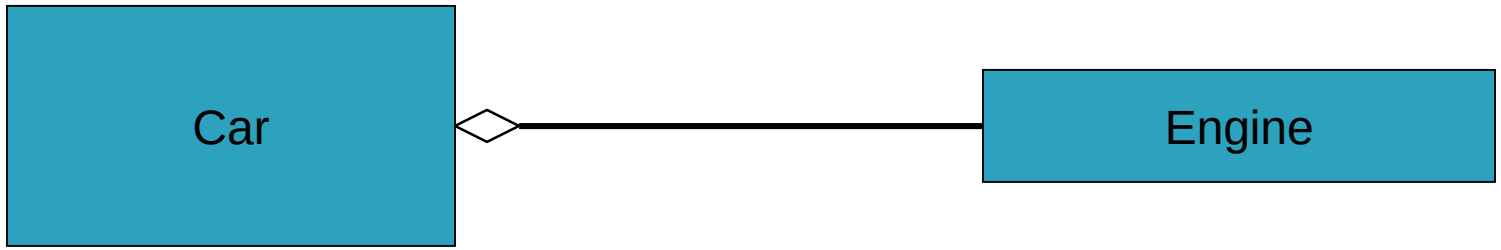
- A **Bank and Customer relationship, if we delete the Customer class the BankAccount still exist in our system.**
- A customer has_a **BankAccount**



Aggregation Example

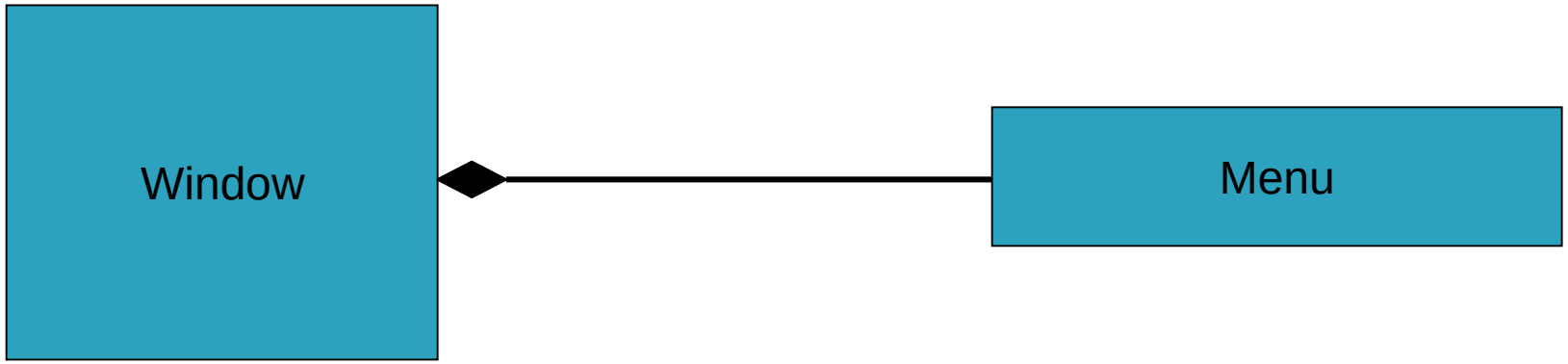
- A car is logically comprised of an Engine part (one, in this case). It does not make sense for Car object to exist independently of the Engine object, because the Car always must contain an Engine.

- However, the Engine can exist without the Car. That is, if the Car object is deleted, the Engine can still exist



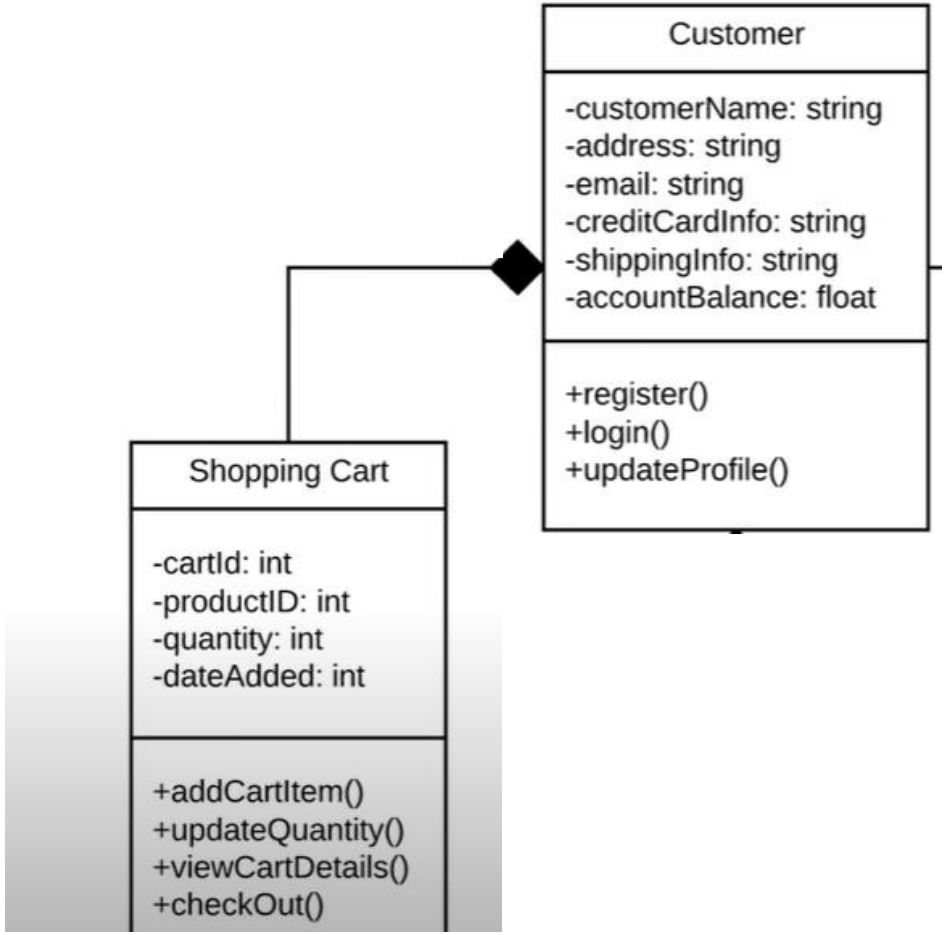
Composition

- Composition implies a relationship where the child cannot exist independent of the parent (restricted aggregation "is entirely made of").
- Compositions are denoted by a filled-diamond adornment on the association.
- A *composition* indicates a strong ownership and coincident lifetime of parts by the whole (*i.e.,* they live and die as a whole). Compositions are denoted by a filled-diamond adornment on the association.



Composition Example

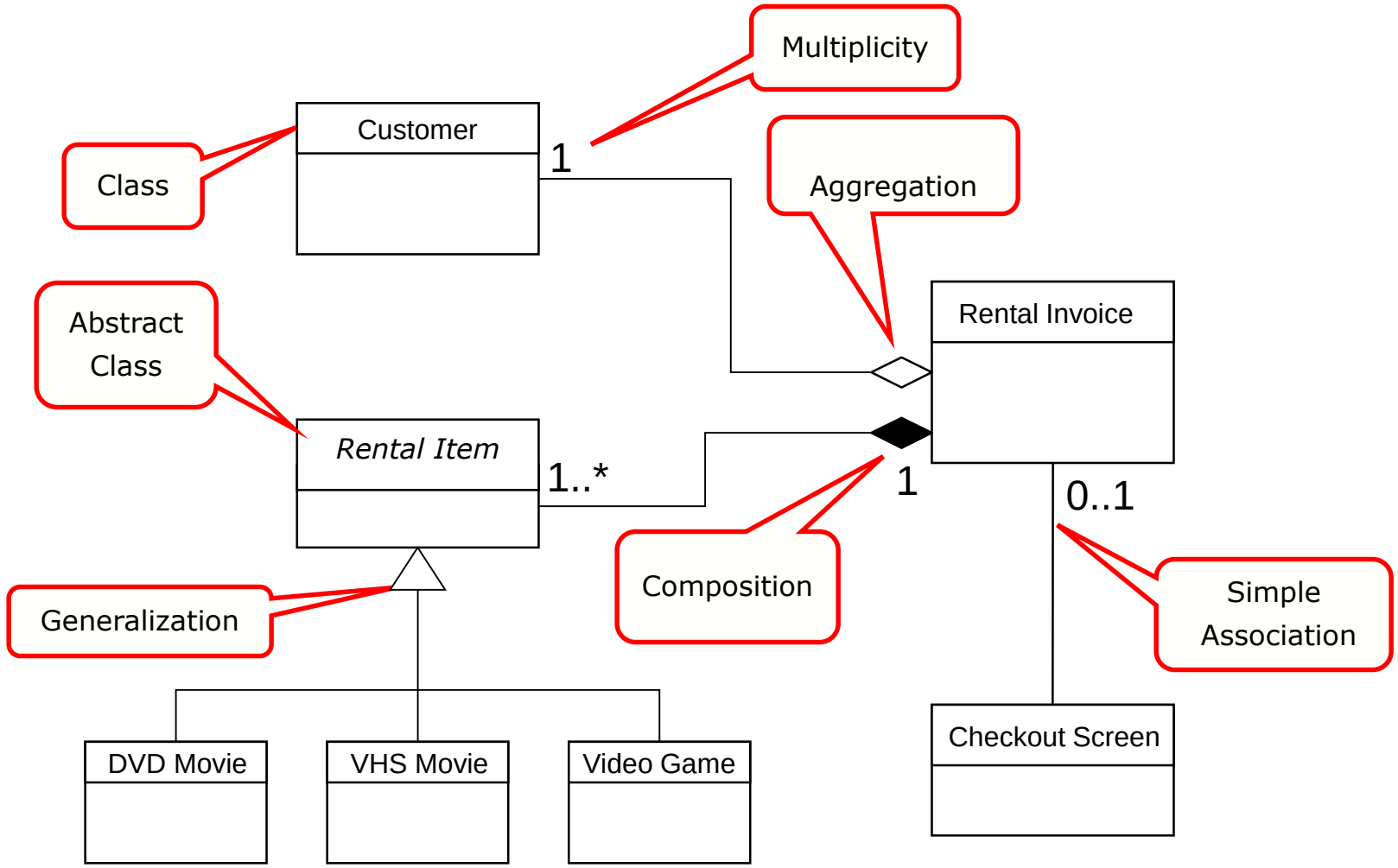
- The part cannot exist without the whole. If an instance of the customer class is destroyed his shopping cart will be also destroyed.



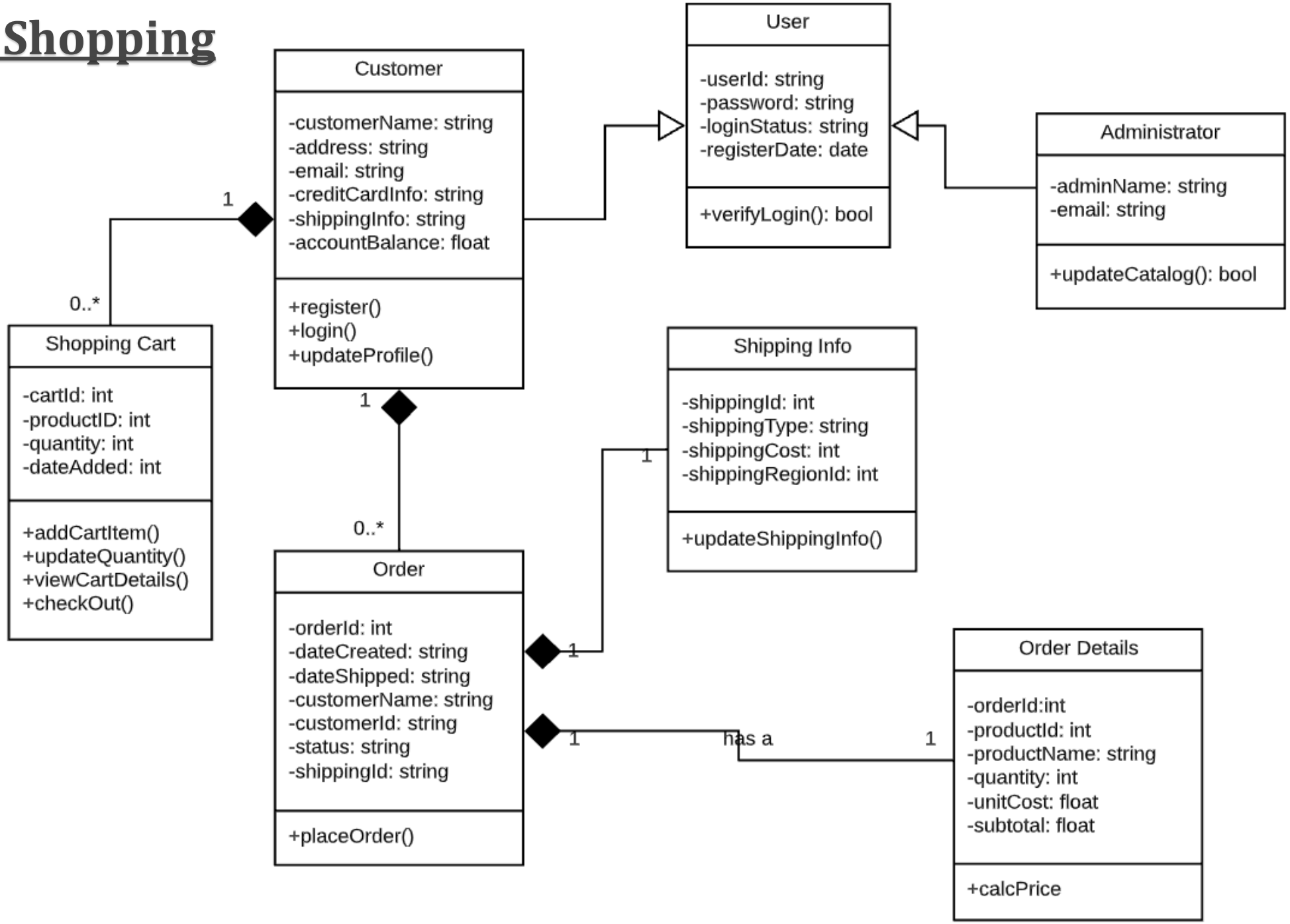
Multiplicity of Associations

- Specifies how many instances of one class can associate with instances of another class.
- For example,
  - 1 patient can have only 1 nationality,
  - 1 customer can purchase up to 100 books,
  - 1 student must register for at least 6 credits per semester but cannot enroll in more than 18.

Class Example



Online Shopping



Summary

- There are also several different types of relationships that exist within UML Class Diagrams. Inheritance is when a child class (or subclass) takes on all the attributes and methods of the parent class (or superclass).
- Association is a very basic relationship where there's no dependency.
- Aggregation is a relationship where the part can exist outside the whole.
- And finally, Composition is when a part cannot exist outside the whole. A class would be destroyed if the class it's related to is destroyed.