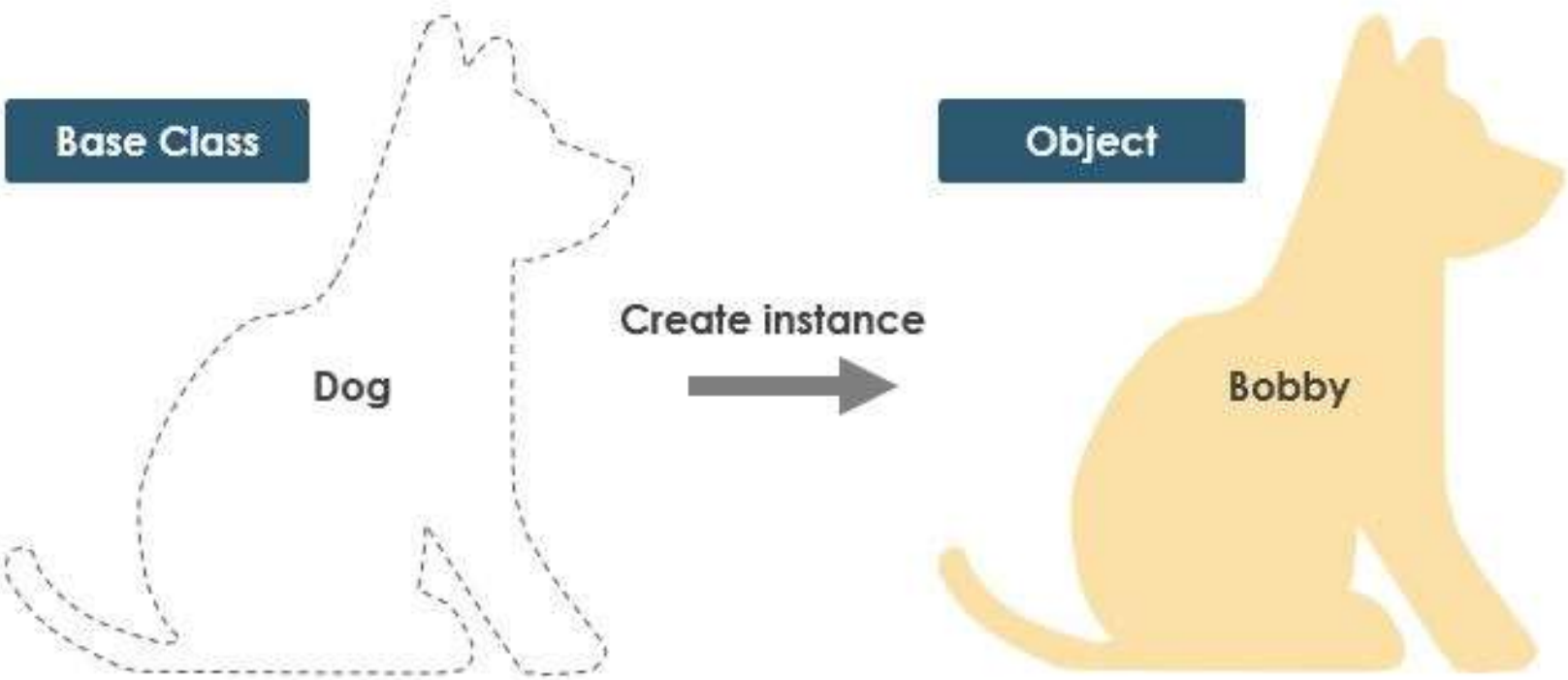
CLASS DIAGRAM

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

A class diagram in the Unified Modelling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's:

- **classes,**
- **their attributes,**
- **operations (or methods),**
- **and the relationships among objects.**



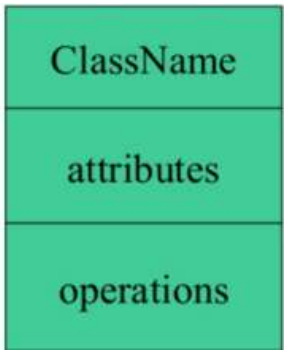
Properties	Methods
Color	Sit
Eye Color	Lay Down
Height	Shake
Length	Come
Weight	

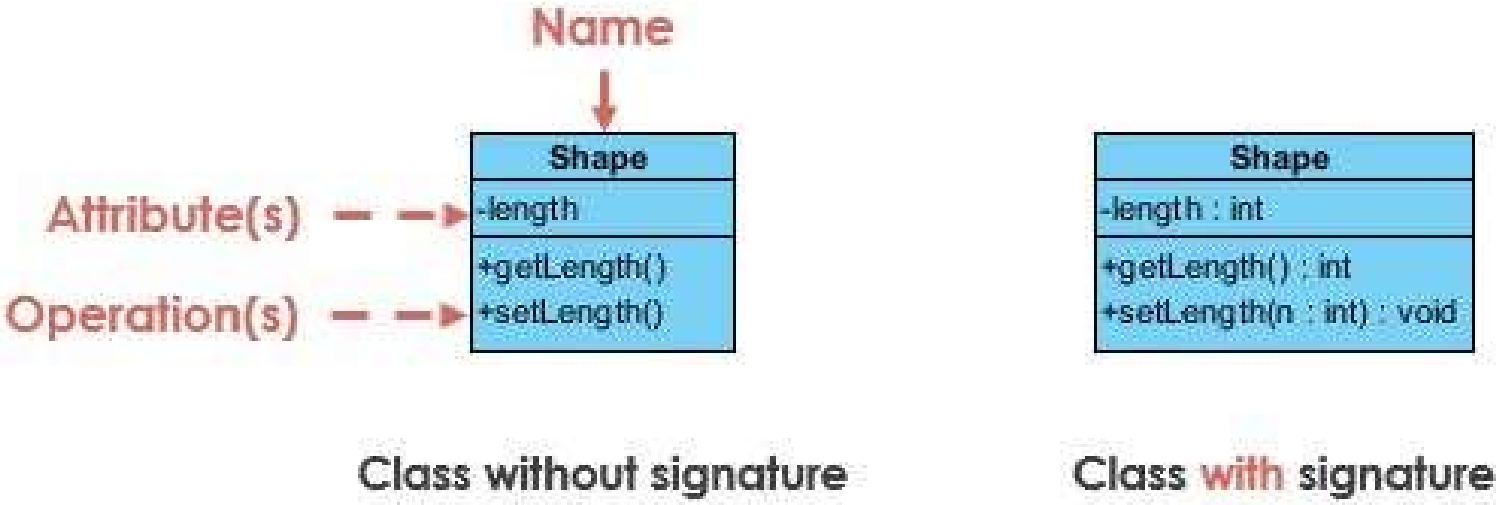
Property Values	Methods
Color: Yellow	Sit
Eye Color: Brown	Lay Down
Height: 17 in	Shake
Length: 35 in	Come
Weight: 24 pounds	

A dog has states - color, name, breed as well as behaviours - wagging, barking, eating. An object is an instance of a class.

UML Class Notation

A class represent a concept which encapsulates state (attributes) and behaviour (operations). Each attribute has a type. Each operation has a signature. The class name is the only mandatory information.


ClassName
attributes
operations


Name
Shape
Attribute(s)
-length
Operation(s)
+getLength()
+setLength()
Shape
-length : int
+getLength() : int
+setLength(n : int) : void
Class without signature
Class with signature

Class Name:

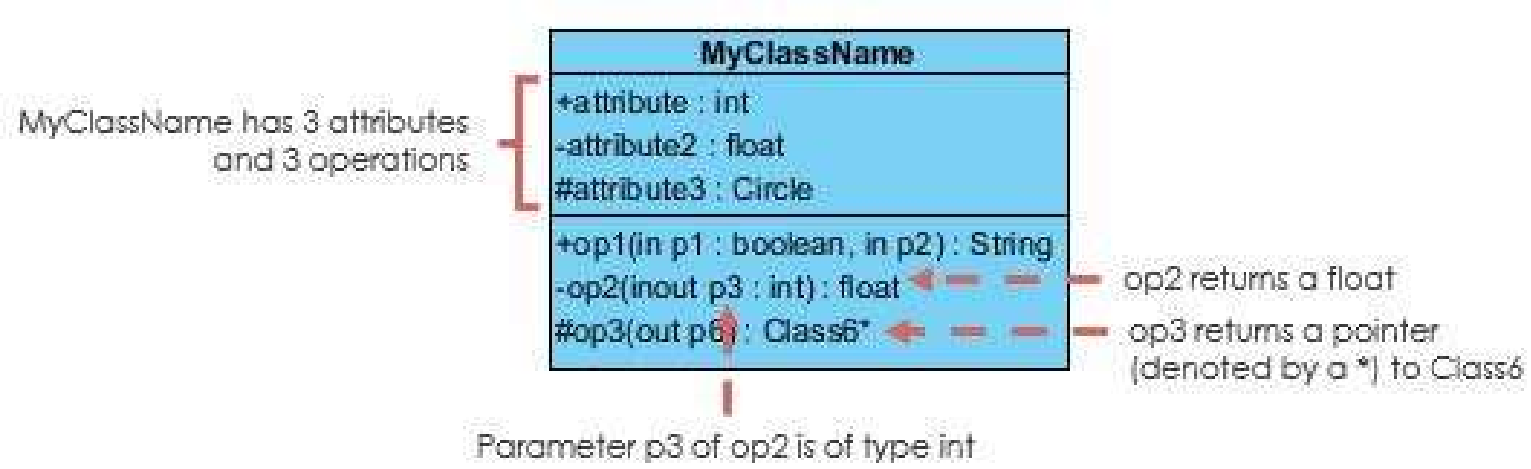
The name of the class appears in the first partition.

Class Attributes:

- Attributes are shown in the second partition.
- The attribute type is shown after the colon.
- Attributes map onto member variables (data members) in code.

Class Operations (Methods):

- Operations are shown in the third partition. They are services the class provides.
- The return type of a method is shown after the colon at the end of the method signature.
- The return type of method parameters are shown after the colon following the parameter name. Operations map onto class methods in code.



Class Visibility

The +, - and # symbols before an attribute and operation name in a class denote the visibility of the attribute and operation.

- **+ denotes public attributes or operations**
- **- denotes private attributes or operations**
- **# denotes protected attributes or operations**

Following rules must be taken care of while representing a class:

- A class name should always start with a capital letter.
- A class name should always be in the centre of the first compartment.
- A class name should always be written in bold format.
- An abstract class name should be written in italics format.

Attributes:

An attribute is named property of a class which describes the object being modelled. In the class diagram, this component is placed just below the name-compartment.

Relationships

There are mainly three kinds of relationships in UML:

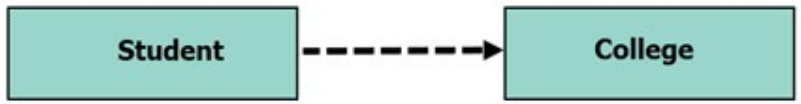
Dependencies

Generalizations

Associations

Dependency

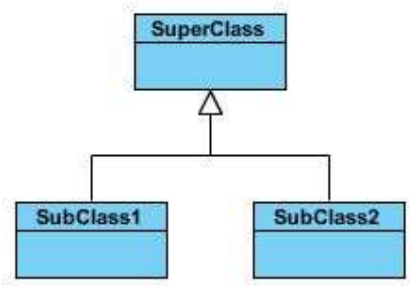
A dependency means the relation between two or more classes in which a change in one may force changes in the other. However, it will always create a weaker relationship. Dependency indicates that one class depends on another.



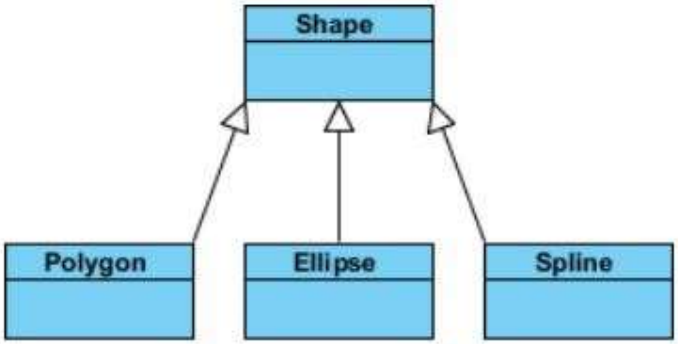
Student has a dependency on College

Generalization

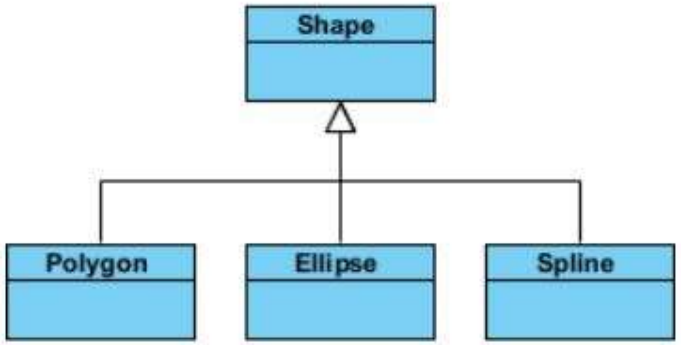
A generalization helps to connect a subclass to its superclass. A sub-class is inherited from its superclass. Generalization relationship can't be used to model interface implementation. Class diagram allows inheriting from multiple super classes.



SubClass1 and SubClass2 are derived from SuperClass.



Style 1: Separate target



Style 2: Shared target

Association

This kind of relationship represents static relationships between classes A and B. For example- an employee works for an organization.

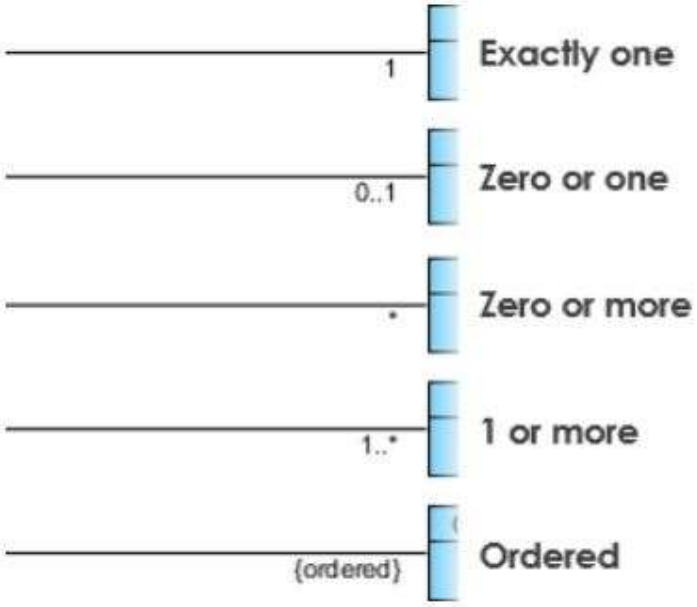
Rules for Association:

- Association is mostly verb or a verb phrase or noun or noun phrase.
- It should be named to indicate the role played by the class attached at the end of the association path.
- Mandatory for reflexive associations

Cardinality

Cardinality is expressed in terms of:

- one to one
- one to many
- many to many



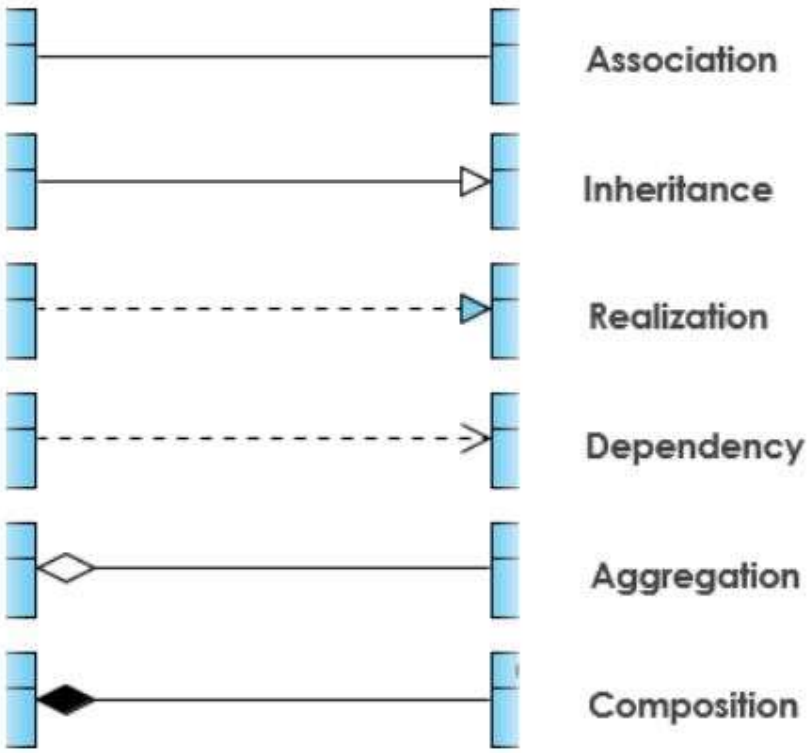
Aggregation

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

Composition

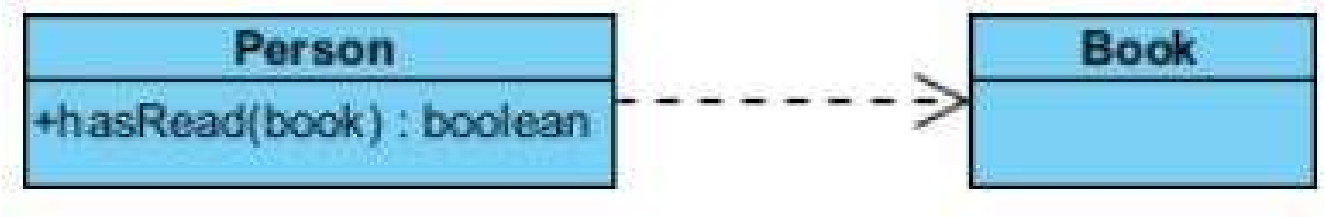
The composition is a special type of aggregation which denotes strong ownership between two classes when one class is a part of another class.

Notations


Association
Inheritance
Realization
Dependency
Aggregation
Composition

Dependency

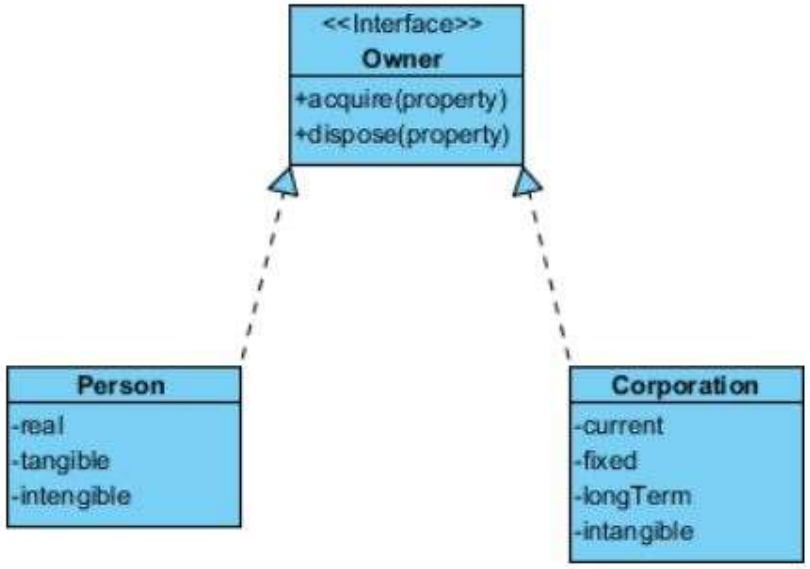
An object of one class might use an object of another class in the code of a method. If the object is not stored in any field, then this is modelled as a dependency relationship.



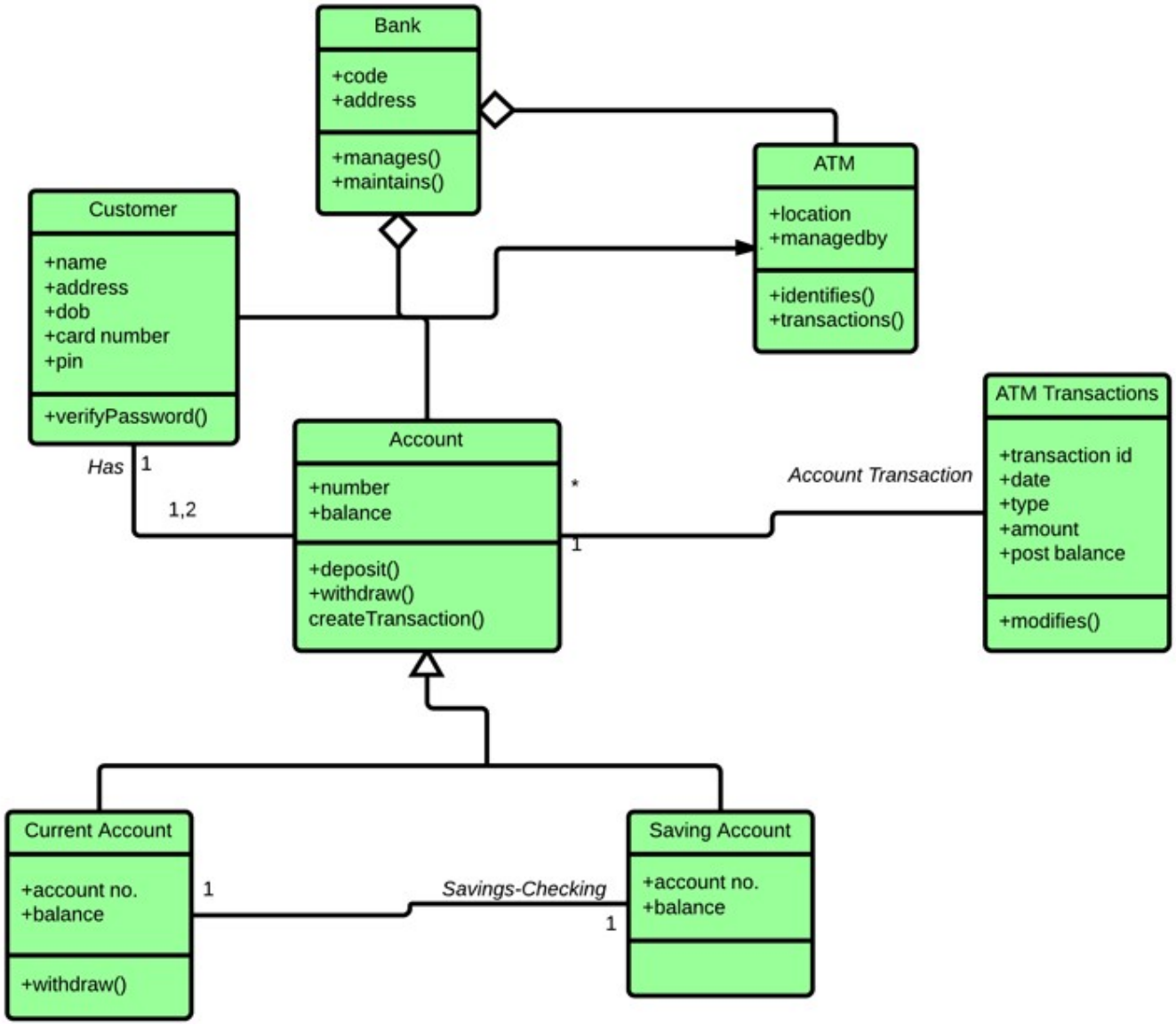
The Person class might have a hasRead method with a Book parameter that returns true if the person has read the book.

Realization

Shows the relationship between the interface and the implementing class.



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.


Bank
+code
+address
+manages()
+maintains()
ATM
+location
+managedby
+identifies()
+transactions()
Customer
+name
+address
+dob
+card number
+pin
+verifyPassword()
Has
1
1,2
Account
+number
+balance
+deposit()
+withdraw()
createTransaction()
*
1
Account Transaction
ATM Transactions
+transaction id
+date
+type
+amount
+post balance
+modifies()
Current Account
+account no.
+balance
+withdraw()
1
Savings-Checking
1
Saving Account
+account no.
+balance

Next: A Case Study on UML

References:

https://www.visual-paradigm.com

https://www.guru99.com/