Class Diagram

SMITA SANKHE
Assistant Professor
Department of Computer Engineering

Class Diagram

- Identify the classes and objects
- Identify relationship among classes and objects
- Identify attributes and operations of objects and links
- Identify associations between objects
- Organize and simplify objects classes using inheritance, generalization and specialization

Classes

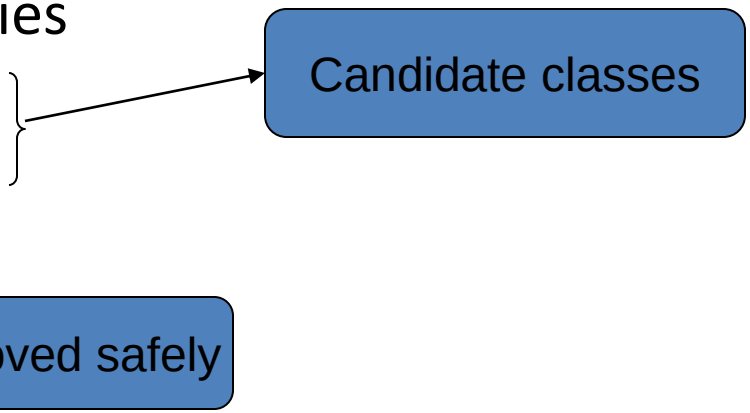
- Classes identified using noun phrase approach
  - Identify redundant classes
  - Identify final classes

Approaches for identifying classes

1. Noun phrase approach
2. Common class patterns approach
3. Use case driven ,sequence/collaboration approach
4. Classes, responsibilities and Collaborators (CRC) approach

Noun phrase approach

- Identify Noun phrases from requirements or use cases
- Nouns - classes
- Verbs - methods
- All plurals → singular
- Create a List of nouns
  - Divided into 3 categories
    - Relevant classes
    - Fuzzy classes
    - Irrelevant classes



Guidelines :Selecting classes from relevant and fuzzy category

- **Redundant classes**
  - Avoid
  - Choose more meaningful name and name used by user
- **Adjective classes**
  - Adjective can suggest
    - Different kind of object
    - Different use of same object
- **Attribute classes**
  - Objects used only as value can be treated as attribute instead of classes
- **Irrelevant classes**
  - Relevant class have statement of purpose.
  - Irrelevant classes - have no statement of purpose

Initial list of noun classes : in vianet bank

- Account
- Account balance
- Amount
- Approval process
- Atm card
- Atm machine
- Bank
- Bank client
- Card
- Cash
- Check
- Checking
- Checking account
- Client
- Client’s account
- Currency
- Dollar
- Envelope
- Four digits
- Fund
- Invalid pin
- Message
- Money
- Password
- PIN
- Pin code
- Record
- Savings
- Savings account
- Step
- System
- Transaction
- transaction history

Removing irrelevant classes

- Account
- Account balance
- Amount
- Approval process
- Atm card
- Atm machine
- Bank
- Bank client
- Card
- Cash
- Check
- Checking
- Checking account
- Client
- Client's account
- Currency
- Dollar
- Envelope
- ~~Four digits~~
- ~~Fund~~
- Invalid pin
- Message
- Money
- Password
- PIN
- Pin code
- Record
- Savings
- Savings account
- ~~Step~~
- System
- Transaction
- transaction history

Reviewing the class purpose

- Include classes with
  - Purpose
  - Clear definition
  - Necessary in achieving system goal
- Eliminate classes with no purpose
- Ex: Candidate class with purpose are
  - ATM machine class
  - ATM card class
  - Bankclient class
  - Bank class
  - Account class
  - Checking account class
  - Saving account class
  - Transaction class

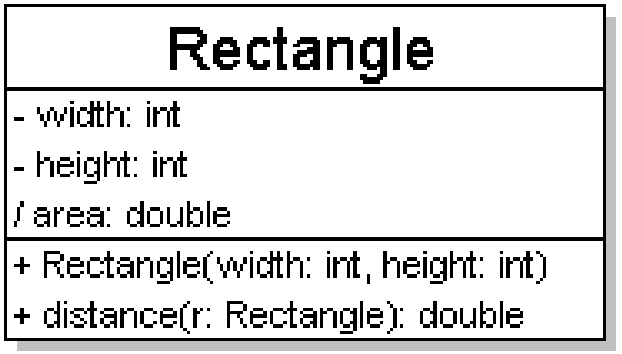
Design phase

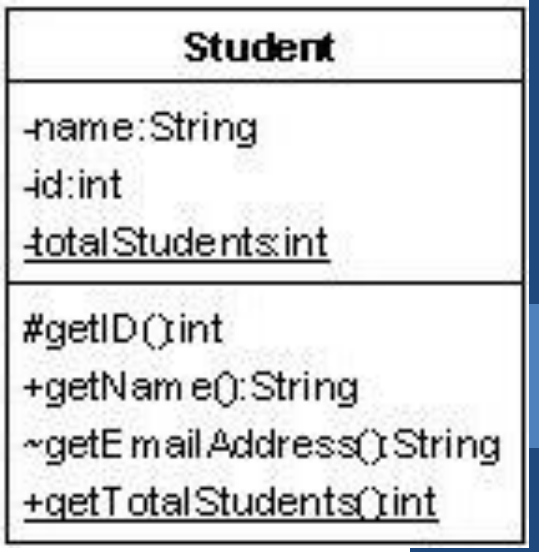
- **design**: specifying the structure of how a software system will be written and function, without actually writing the complete implementation

- a transition from "what" the system must do, to "how" the system will do it
  - What classes will we need to implement a system that meets our requirements?
  - What fields and methods will each class have?
  - How will the classes interact with each other?

Diagram of one class

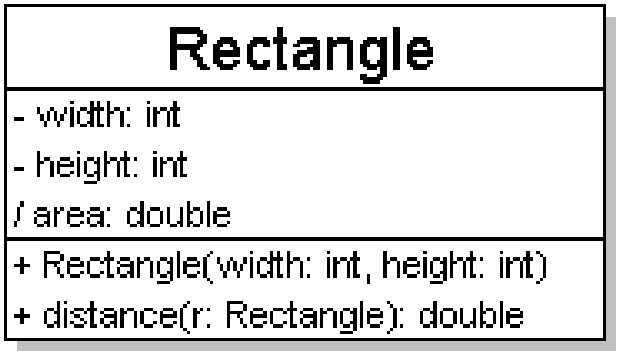
- class name in top of box
  - write <<interface>> on top of interfaces' names
  - use *italics* for an *abstract class* name
- attributes (optional)
  - should include all fields of the object
- operations / methods (optional)

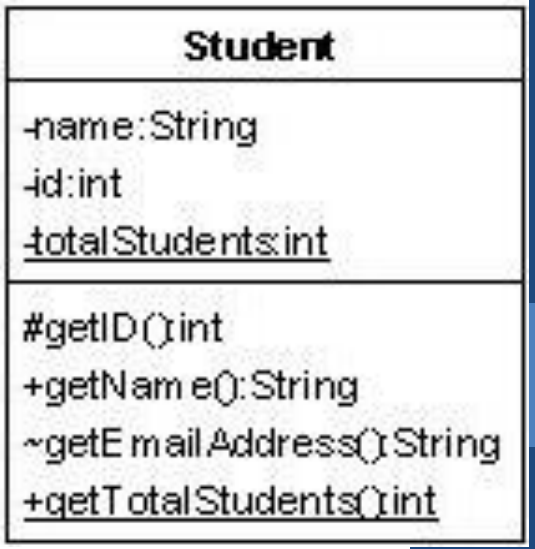




Class attributes

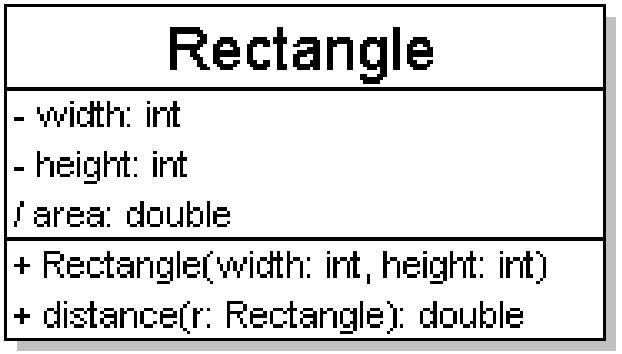
- attributes (fields, instance variables)
  - *visibility name* : *type* [*count*] = *default_value*
  - visibility:
    - \+ public
    - # protected
    - \- private
    - ~ package (default)
    - / derived
  - underline static attributes

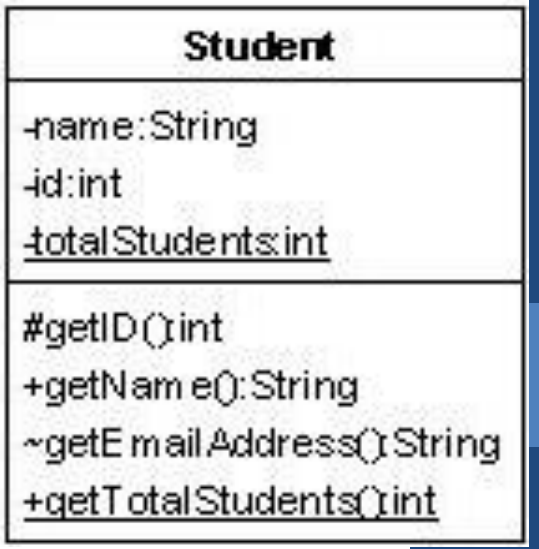




Class operations / methods

- operations / methods
  - *visibility name* (*parameters*) : *return_type*
  - visibility:
    - `+` public
    - `#` protected
    - `-` private
    - `~` package (default)
  - underline static methods
  - parameter types listed as (name: type)
  - omit *return_type* on constructors and when return type is void
  - method example:
    + distance(p1: Point, p2: Point): double





Relationships btwn. classes

- **generalization**: an inheritance relationship
  - inheritance between classes
  - interface implementation

- **association**: a usage relationship

  - aggregation
  - composition

Aggregation vs. Composition

- **Aggregation**

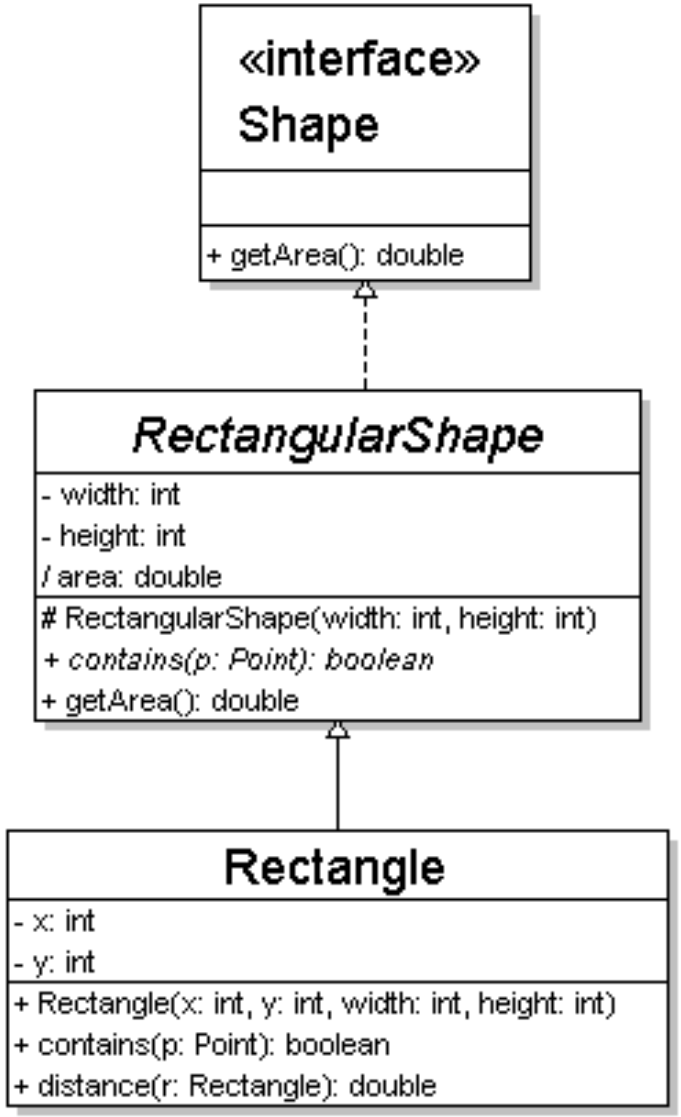
Aggregation indicates a relationship where the child can exist separately from their parent class. Example: Automobile (Parent) and Car (Child). So, If you delete the Automobile, the child Car still exist.

- **Composition**

Composition display relationship where the child will never exist independent of the parent. Example: House (parent) and Room (child). Rooms will never separate into a House.

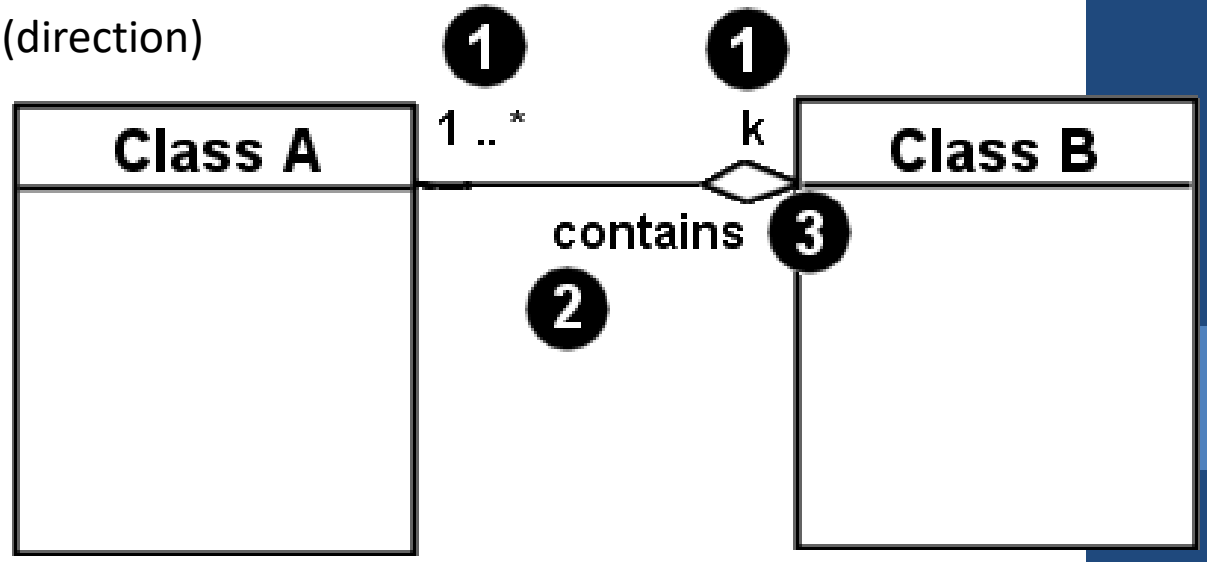
Generalization relationships

- generalization (inheritance) relationships
  - hierarchies drawn top-down with arrows pointing upward to parent
  - line/arrow styles differ, based on whether parent is a(n):
    - class:
      solid line, black arrow
    - interface:
      dashed line, white arrow



Associational relationships

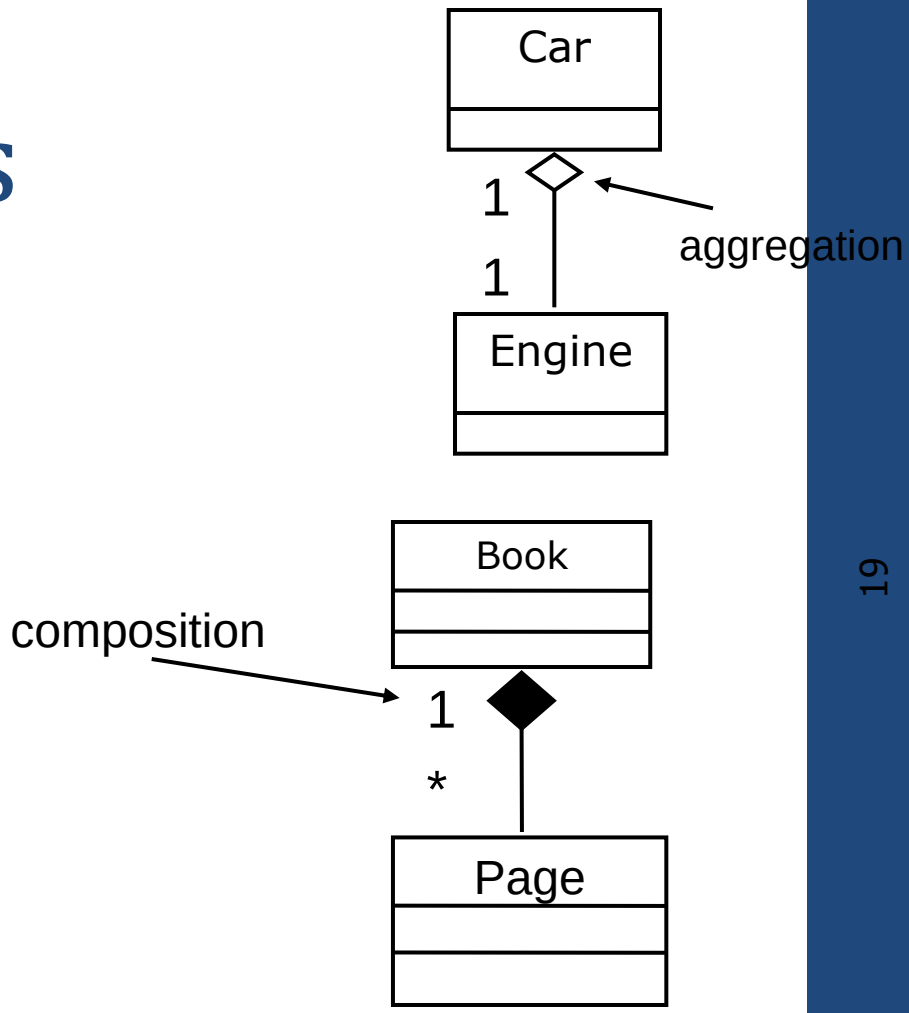
- associational (usage) relationships
  1. multiplicity (how many are used)
     - * ⇒ 0, 1, or more
     - 1 ⇒ 1 exactly
     - 2..4 ⇒ between 2 and 4, inclusive
     - 3..* ⇒ 3 or more
  2. name (what relationship the objects have)
  3. navigability (direction)



Association types

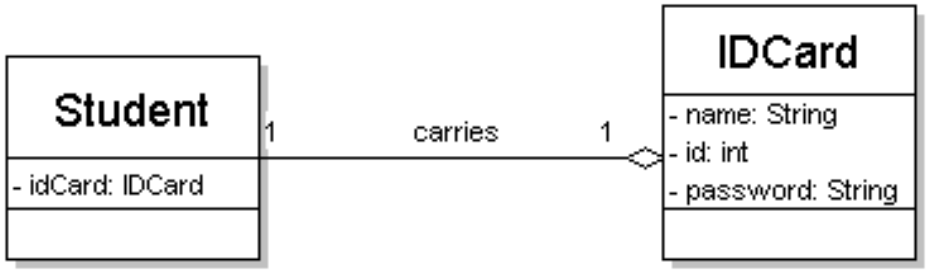
- **aggregation**: "is part of"
  - symbolized by a clear white diamond

- **composition**: "is entirely made of"
  - stronger version of aggregation
  - the parts live and die with the whole
  - symbolized by a black diamond

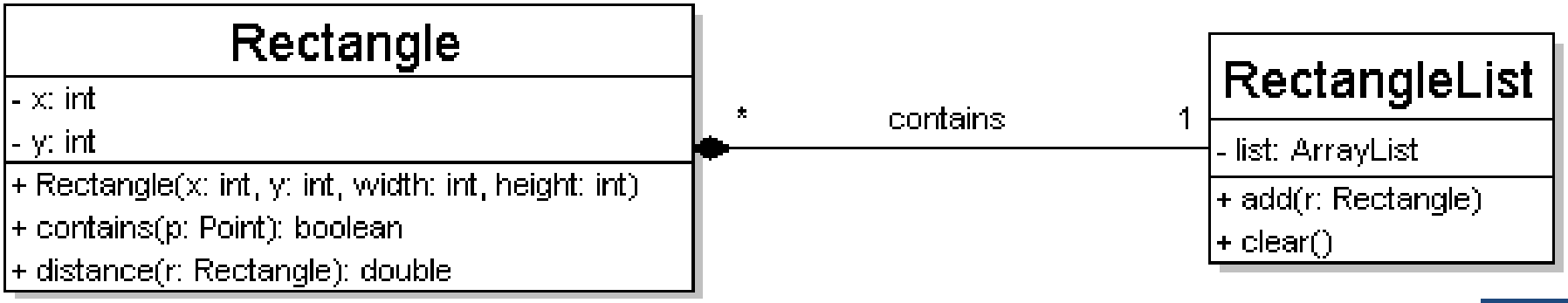


Multiplicity of associations

- one-to-one
  - each student must carry exactly one ID card



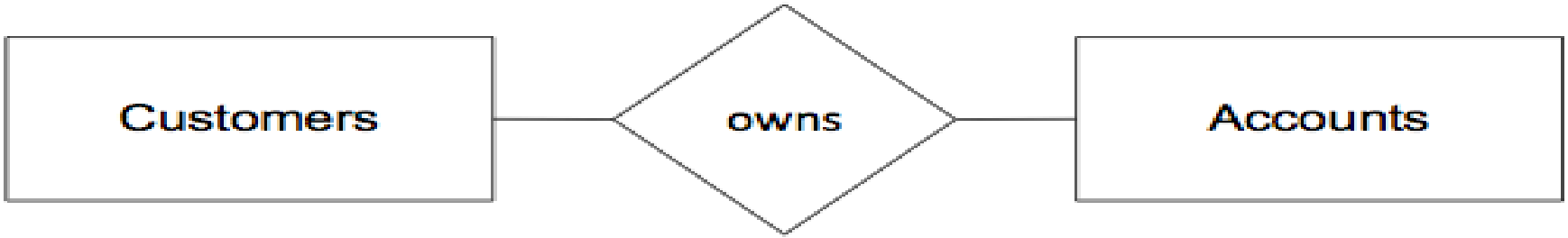
- one-to-many
  - one rectangle list can contain many rectangles



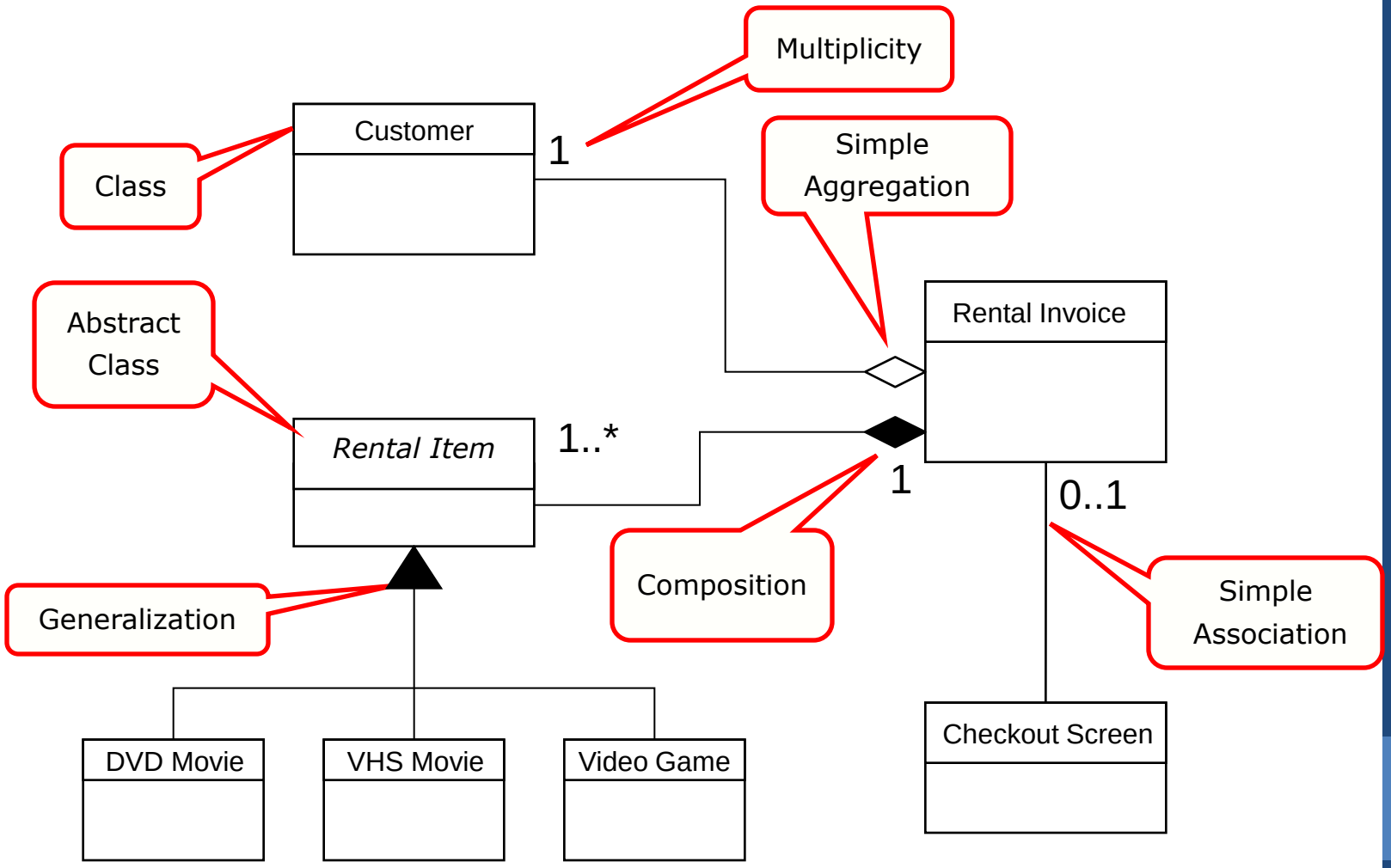
One-to-One

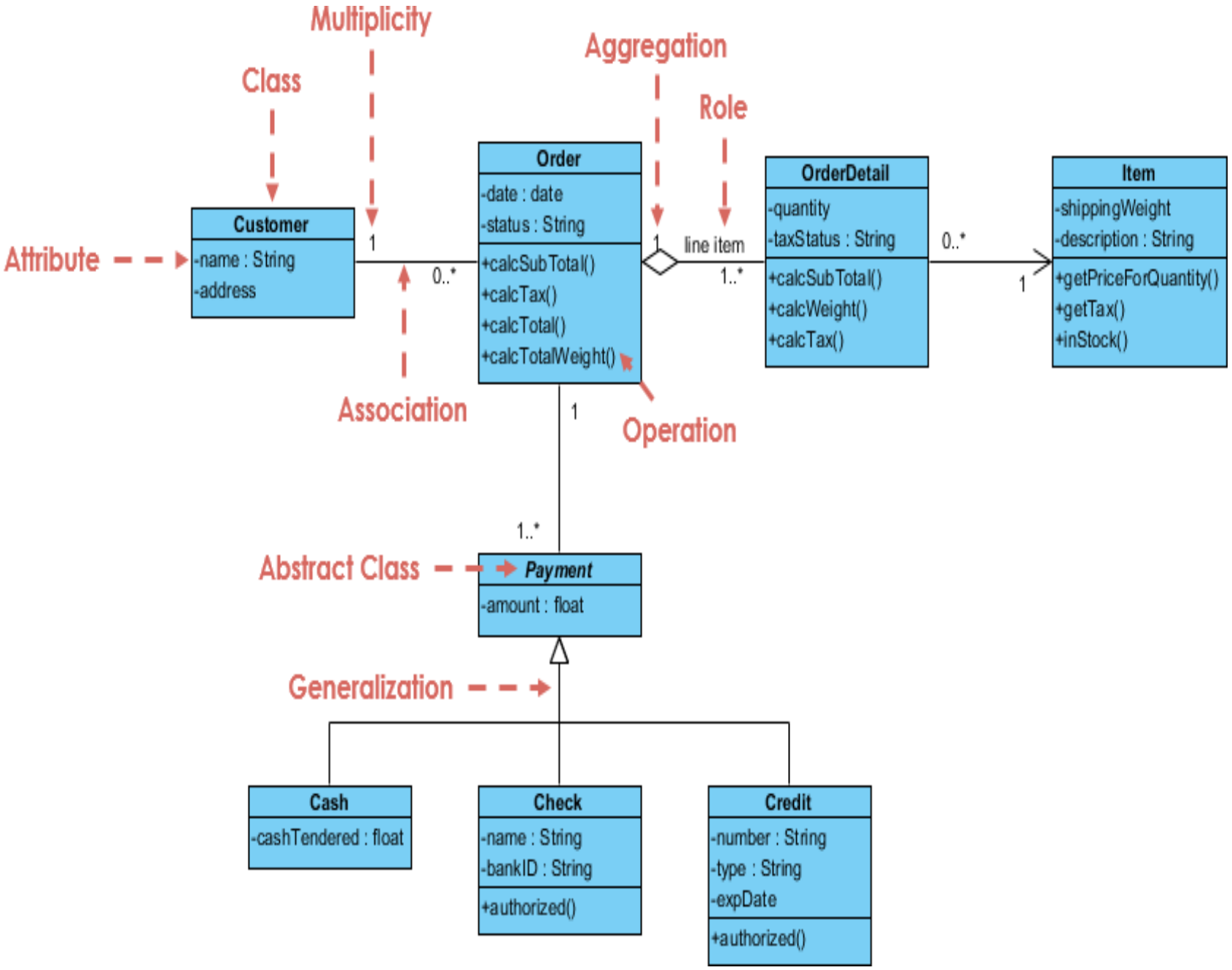
One-to-Many

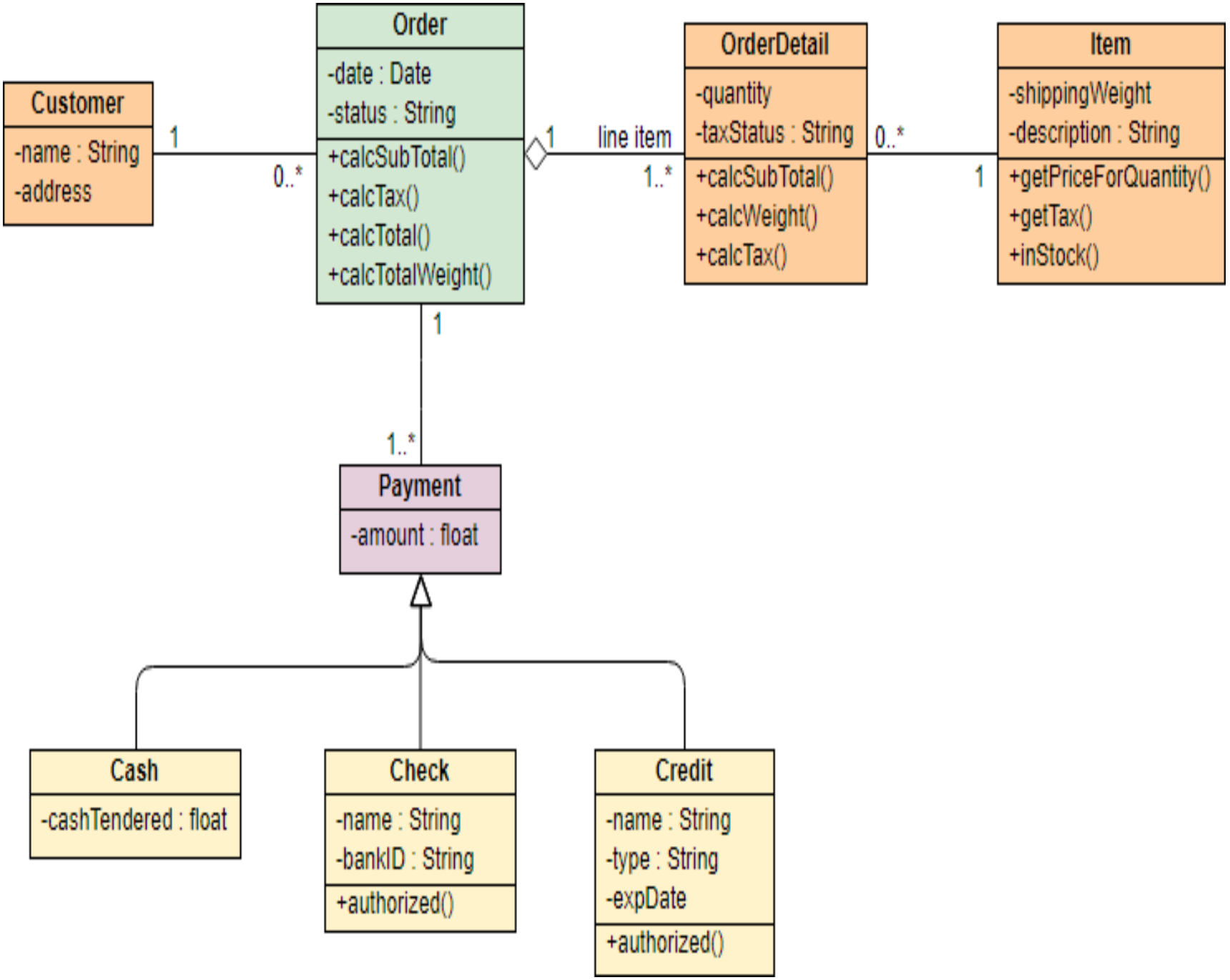
Many-to-Many

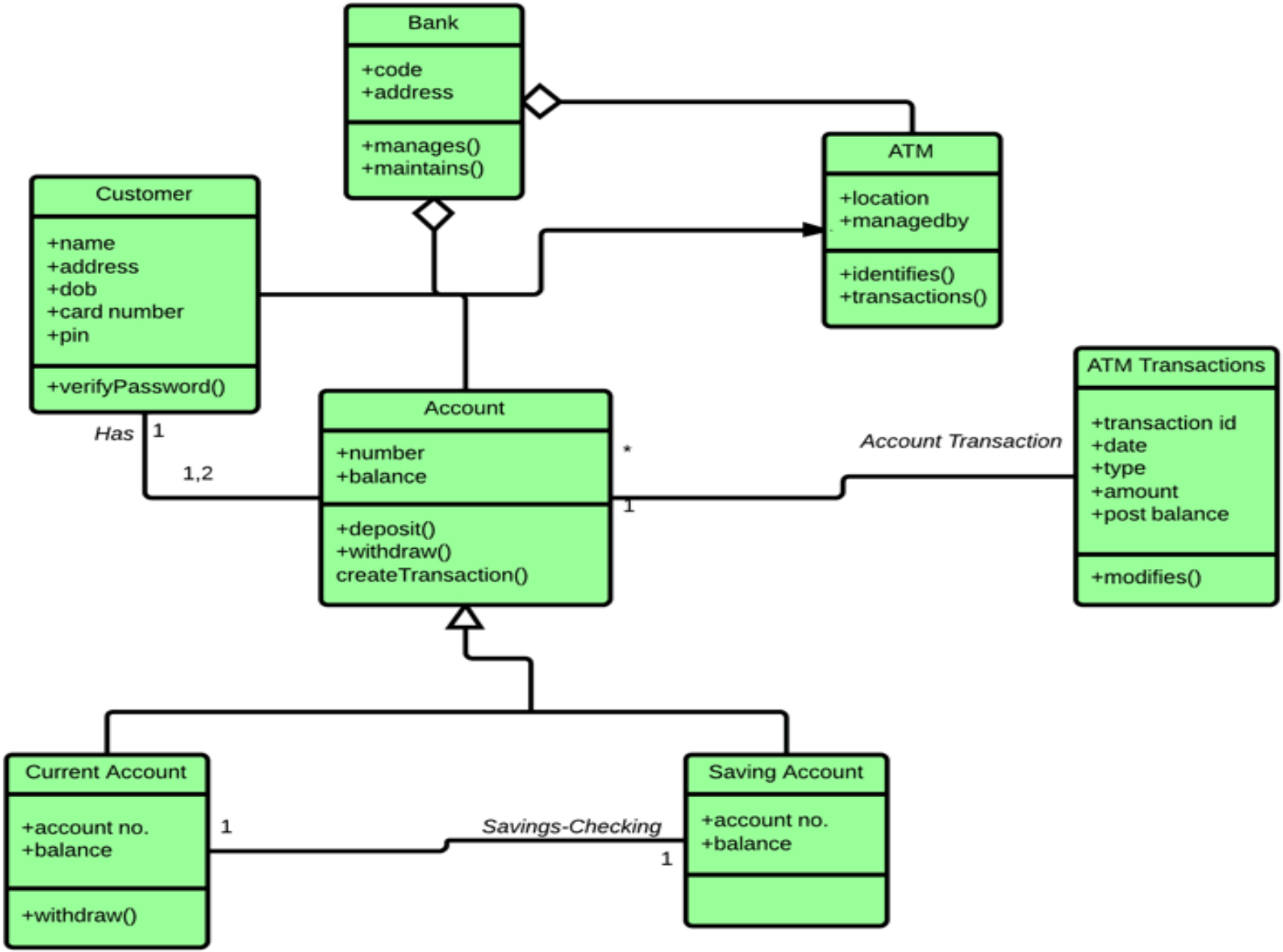


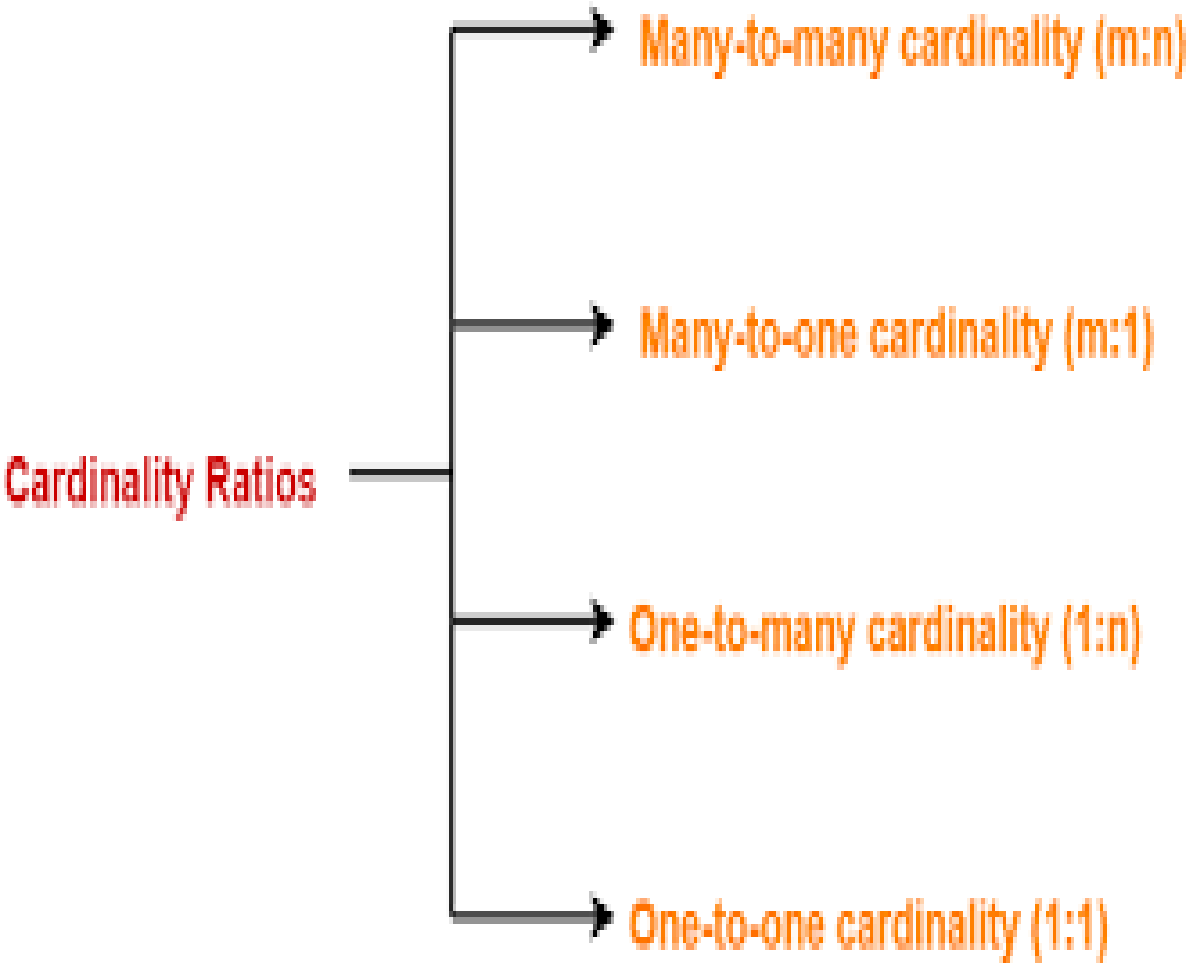
Class diagram example 1




Multiplicity
Class
Aggregation
Role
Attribute
Customer
-name : String
-address
1
0..*
Association
Order
-date : date
-status : String
+calcSubTotal()
+calcTax()
+calcTotal()
+calcTotalWeight()
1
line item
1..*
Operation
OrderDetail
-quantity
-taxStatus : String
+calcSubTotal()
+calcWeight()
+calcTax()
0..*
1
Item
-shippingWeight
-description : String
+getPriceForQuantity()
+getTax()
+inStock()
1
1..*
Abstract Class
Payment
-amount : float
Generalization
Cash
-cashTendered : float
Check
-name : String
-bankID : String
+authorized()
Credit
-number : String
-type : String
-expDate
+authorized()


Order
-date : Date
-status : String
+calcSubTotal()
+calcTax()
+calcTotal()
+calcTotalWeight()
Customer
-name : String
-address
1
0..*
1
line item
1..*
OrderDetail
-quantity
-taxStatus : String
+calcSubTotal()
+calcWeight()
+calcTax()
0..*
1
Item
-shippingWeight
-description : String
+getPriceForQuantity()
+getTax()
+inStock()
1
1..*
Payment
-amount : float
Cash
-cashTendered : float
Check
-name : String
-bankID : String
+authorized()
Credit
-name : String
-type : String
-expDate
+authorized()


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


Cardinality Ratios
Many-to-many cardinality (m:n)
Many-to-one cardinality (m:1)
One-to-many cardinality (1:n)
One-to-one cardinality (1:1)