SLIIT
Discover Your Future

IT1050-Object Oriented Concepts

Relationships and Class Diagram

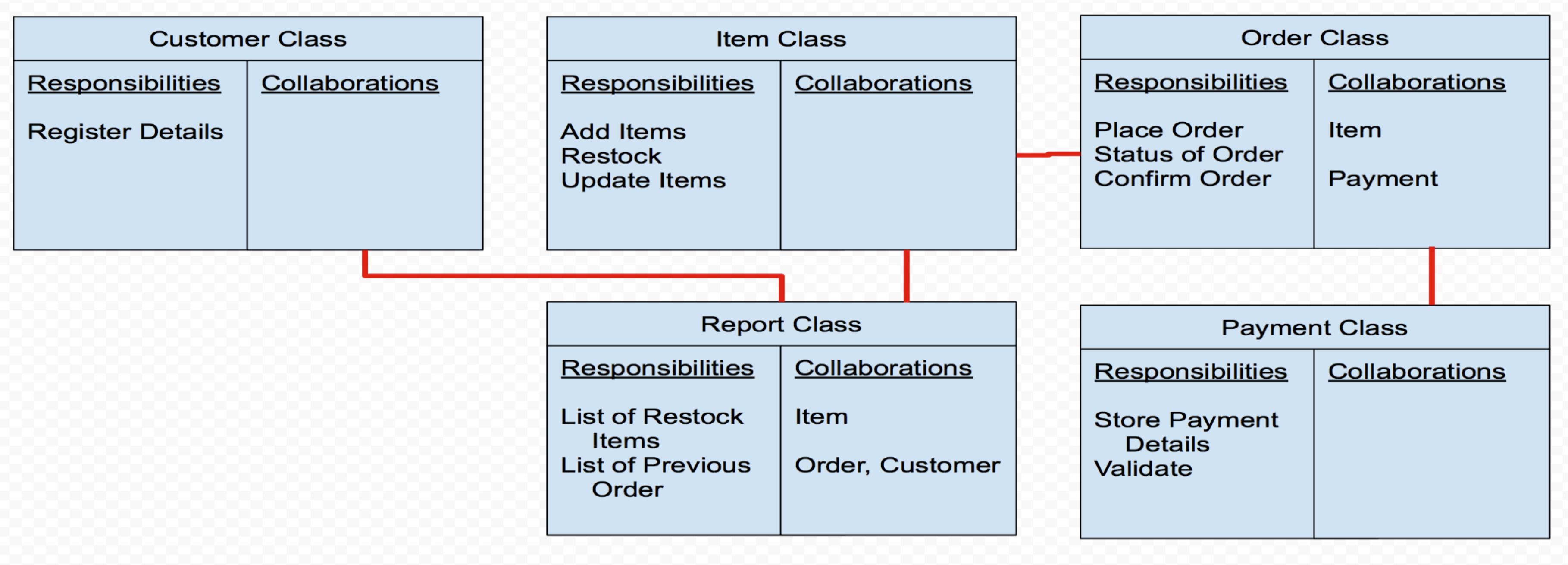
Lecture-10

SLIIT
FACULTY OF COMPUTING

Learning Outcomes

- At the end of the lecture, students should be able to
  - Identify Dependency, Association, Aggregation, Composition, and Inheritance relationships between classes
  - Create a class diagram in UML notation with the Dependency, Association, Aggregation, Composition and Inheritance relationships

Relationships between classes



Customer Class	
Responsibilities	Collaborations
Register Details	

Item Class	
Responsibilities	Collaborations
Add Items Restock Update Items	

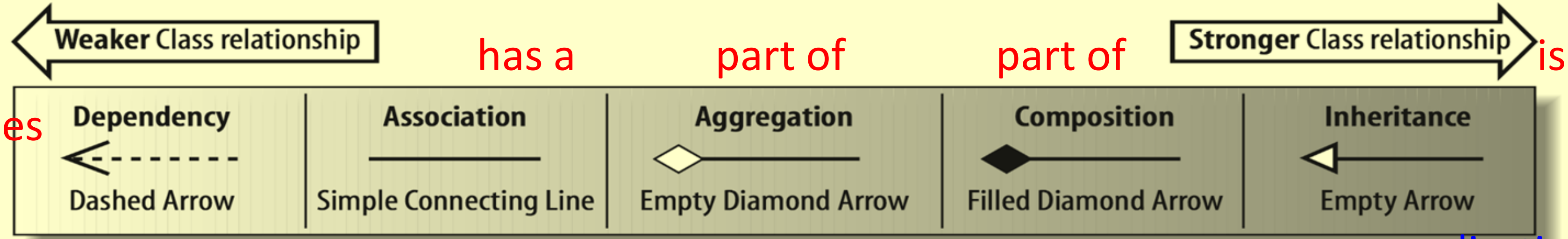
Order Class	
Responsibilities	Collaborations
Place Order Status of Order Confirm Order	Item Payment

Report Class	
Responsibilities	Collaborations
List of Restock Items List of Previous Order	Item Order, Customer

Payment Class	
Responsibilities	Collaborations
Store Payment Details Validate	

We can see that there are relationships between classes when we draw CRC cards. We can divide all relationships into five categories

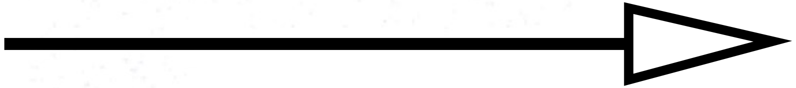
Relationships Between Classes



	Dependency	Association	Aggregation	Composition	Inheritance
	uses	has a	part of	part of	is a
	Dashed Arrow	Simple Connecting Line	Empty Diamond Arrow	Filled Diamond Arrow	Empty Arrow
					generalization
	When objects of one class **work briefly with** objects of another class	When objects of one class **work with** objects of another class for some prolonged amount of time	When one class **owns but shares a reference to** objects of another class	When one class **contains** objects of another class	When one class **is a** type of another class
	When objects of one class works briefly with with objects of another class	When objects of one class works with with objects of another class	When parts can exist without The whole	When parts cannot exist without the whole	When one class is a type of another class

Inheritance

- Otherwise Known as Generalization.
- Inheritance represents a “is-a-kind-of“ relationship.
- Inheritance is a relationship between a general thing (superclass/parent) and a more specific kind of a thing (subclass/child).
- Graphically, it is rendered as an empty block arrow.

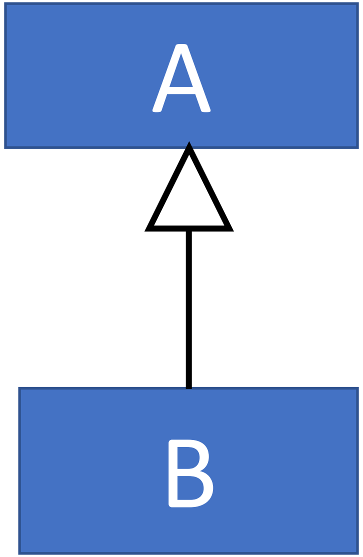


Inheritance

- Class A is a super class (parent) of class B if B directly inherits from A.
- Class B is a sub class (child) of class A id B directly inherits from A.
- Class A is an ancestor of class B if A is above B in the inheritance hierarchy
- Class B is a descendant of class A if B is below A in the inheritance hierarchy

Super class

Ancestor



Sub class

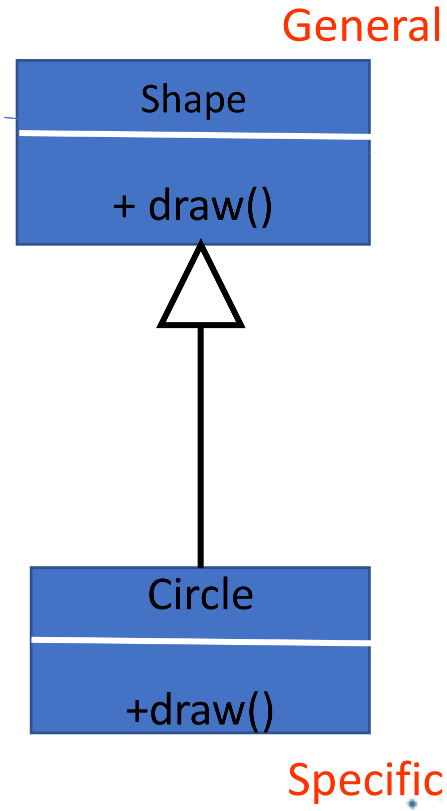
Descendent

Inheritance

- Generalization takes place from sub-class to super-class: the super-class is a **generalization** of the sub-class.

- Specialization takes place from super-class to sub-class: the sub-class is a **specialization** of the super-class.

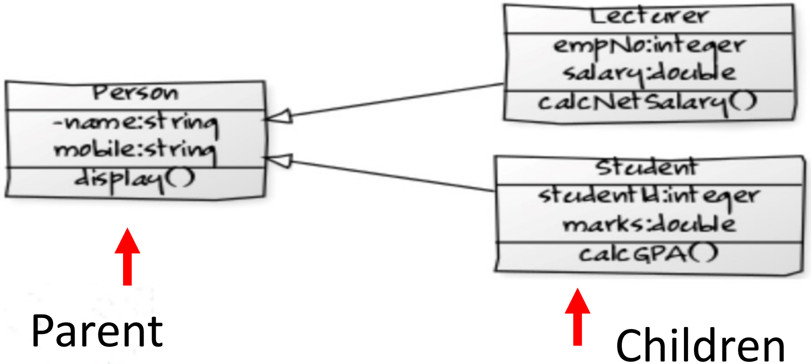
- The functionality of the child should be a specialization of the functionality of the parent.
- **e.g. the functionality of the draw method in Circle is more specific than the one in Shape**



Inheritance

- Super Class / Parent : Person
- Sub Classes / Children : Student, Lecturer

- Children (Student and Lecturer) inherit the properties of its parent’s attributes, operations, responsibilities, etc.).



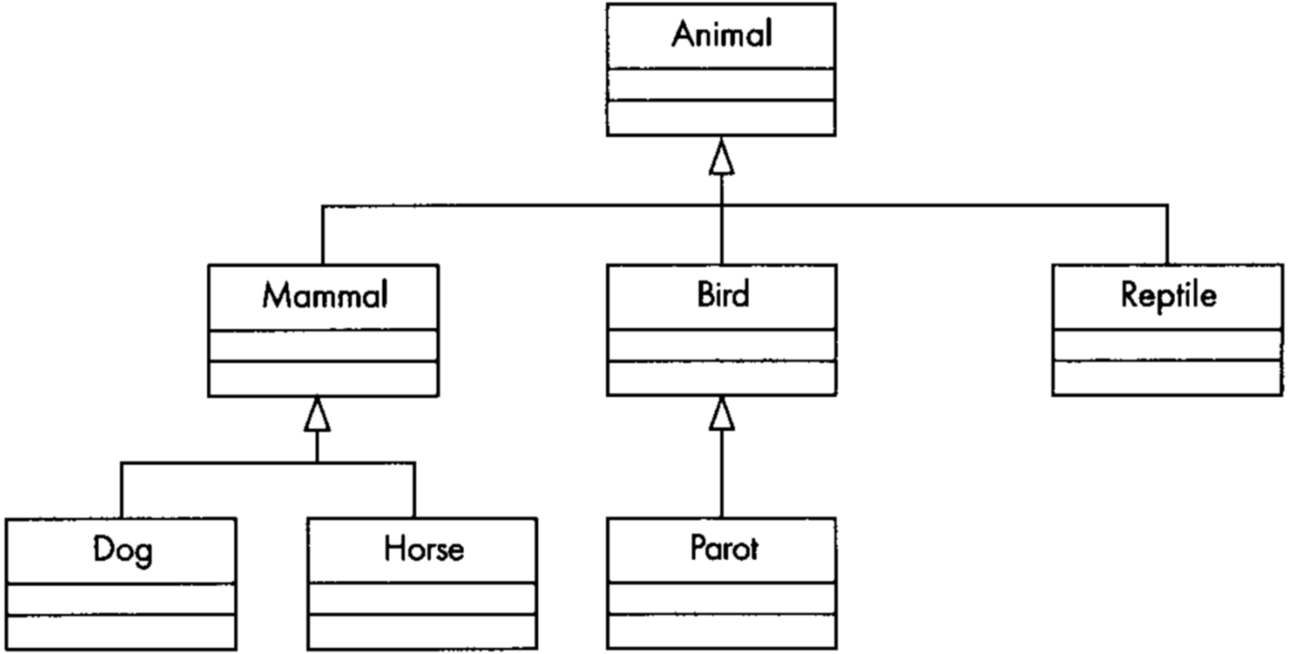
Parent

Children

Note : Normally this diagram is drawn vertically

Inheritance – is A

Animal, Mammal, Bird, Reptile, Dog, Horse, Parrot



Whole – Part Relationships

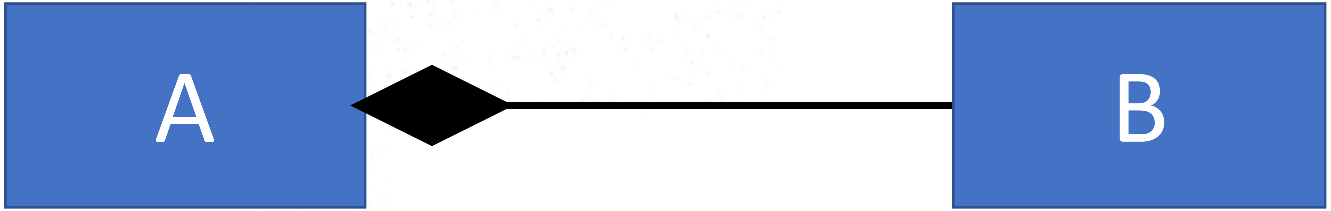
- Aggregation
- Composition

Whole – Part Relationships

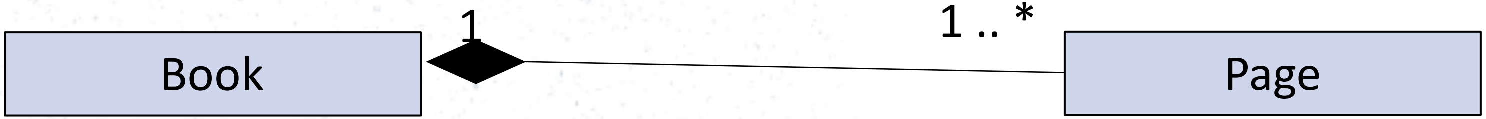
- A "whole/part“ relationship refers to a fairly strong connection between two classes.
- One class represents a larger thing("whole"),which consists of smaller things ("parts").
- This means “part of” relationship. Meaning that an object of the whole has objects of the part.
- Two types;
  - Aggregation (Relatively Weak)
  - Composition (Relatively Strong)

Composition

- Composition is a strong form of whole-part relationship
- Graphically, a dependency is rendered as a filled diamond arrow.
- Composition should have a relationship with a multiplicity of “1 .. “



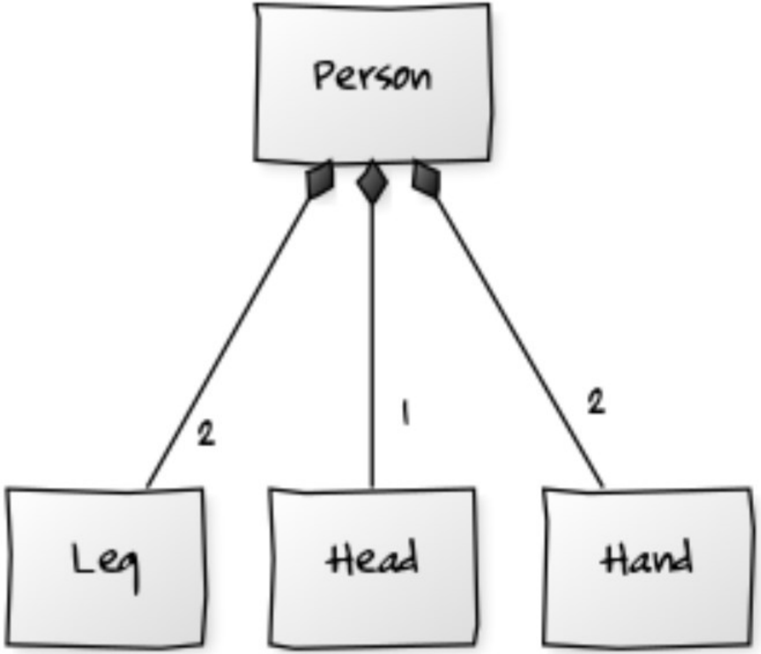
Composition



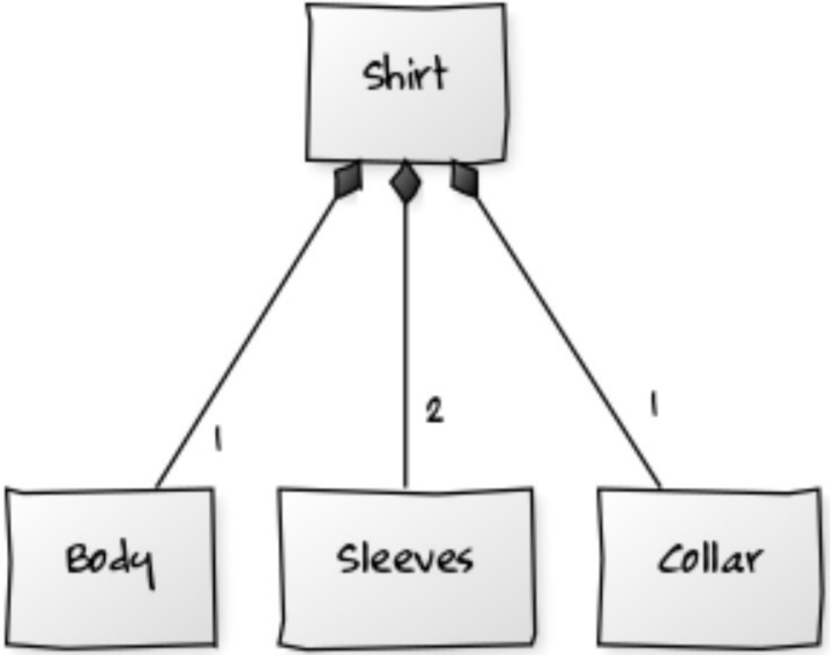
- Whole : Book
- Part : Page
- A Book has pages,
- A Page cannot exist without the Book.
- Implies that the “Part cannot exist without Whole”

Composition

Person, Head, Leg, Hand

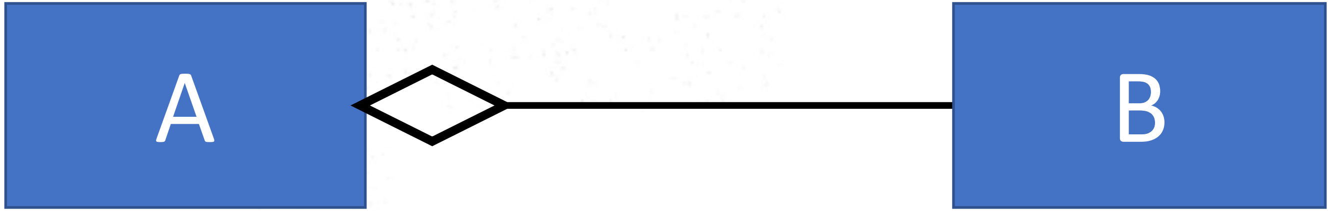


Shirt, Body, Sleeve, Collar



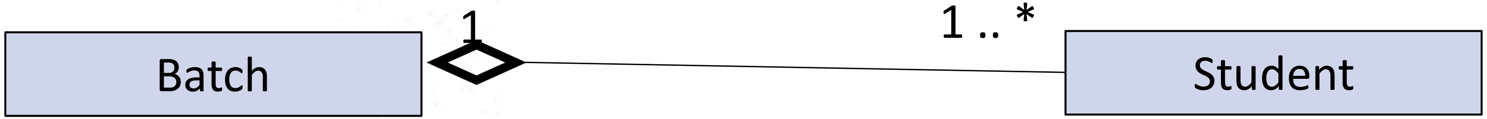
Aggregation

- Aggregation is just a special kind of association
- Aggregation is a weak form of whole-part relationship
- Graphically, a dependency is rendered as an empty diamond arrow.
- This means that A aggregates B.

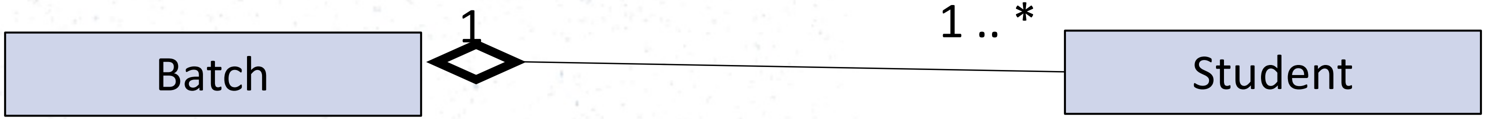


Aggregation

- The same rule for multiplicity can be applied for aggregation relationship.
- In some situations Aggregation can also have a multiplicity of “**0..**”
- e.g. : A batch consists of of **one or more students**



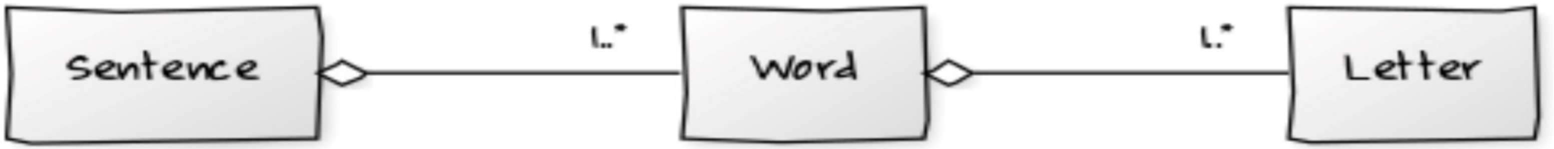
Aggregation



- Whole : Batch
- Part : Student
- A Student can exist without the Batch.
- This implies that the Part can exist without the Whole.

Aggregation

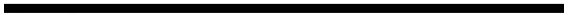
Sentence, Word, Letter



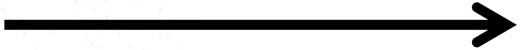
Association

- Association defines a has a relationship.

- Association connects one instance of a class with an instance of another class.



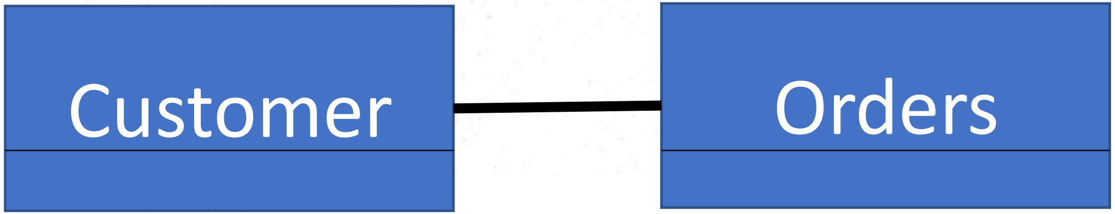
- The relationship can be bi-directional (two way) or uni-directional (one way)



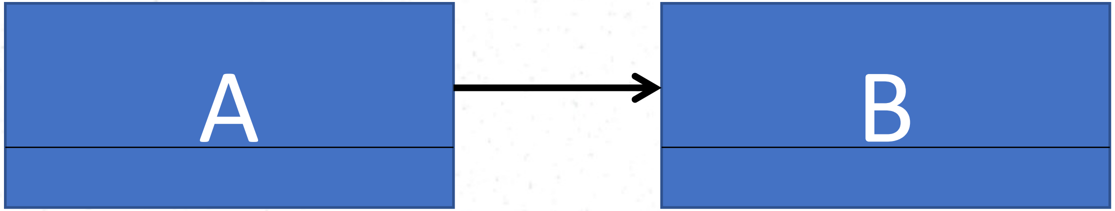
- If there is an arrowhead, it means there is a one-way relationship.

Association

- Example: “A Customer has Orders” (A list of previous orders made)

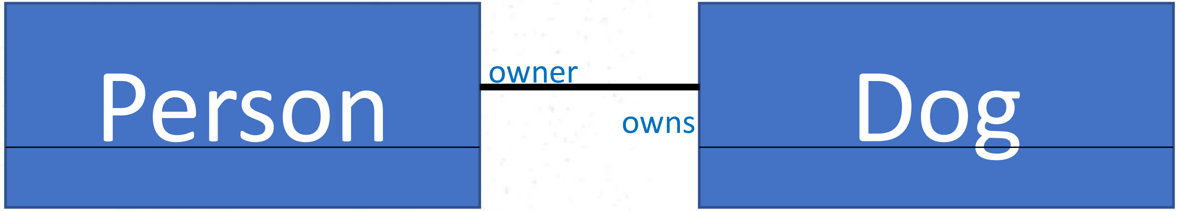


Association (one way)



- Here it means that;
  - Class A is associated with class B
  - Class A uses and contains one instance of class B, but B does not know about it or contain any instances of class A.
- In an Association relationship, the dependent class (A) defines an instance of the associated class (B).

Association (Two way)

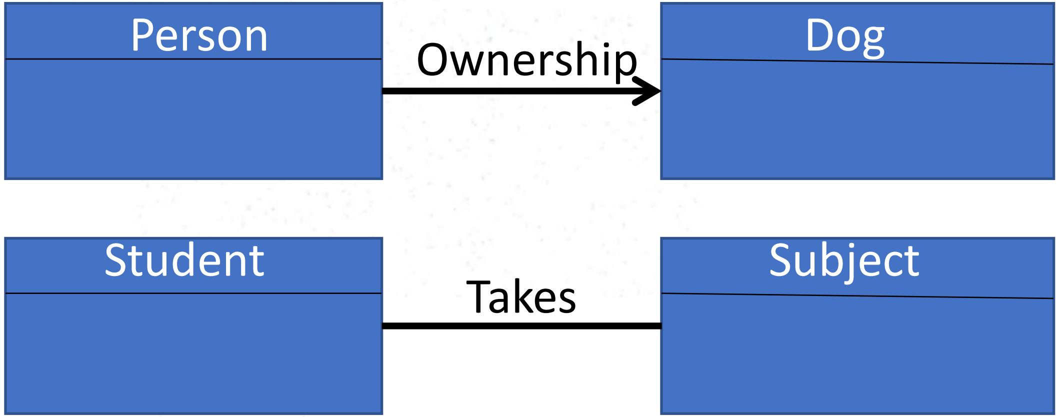


- The person class and the Dog class are associated.
- Here. It is bi-directional
  - The person is related to the dog in some way
  - The Dog is also related to the person in some way

Association

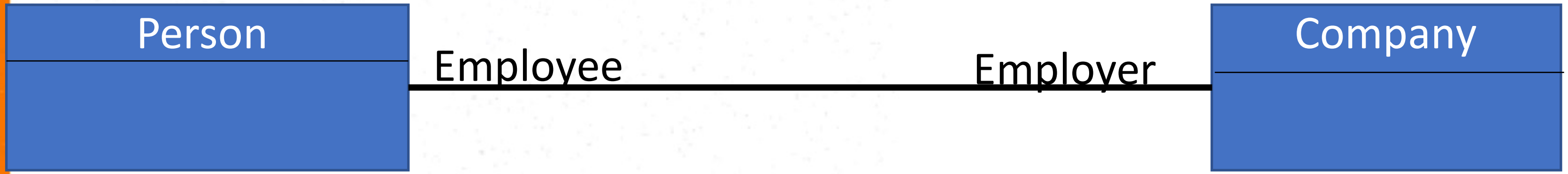
- Association Name :
- An Association can be given a name:

(This is only shown if it helps to clarify the association)



Association

- Association Role:
- When a class participates in an association, it has a specific role that it plays in that relationship;
- A role is just the face the class at the near end of the association presents to the class at the other end of the association.



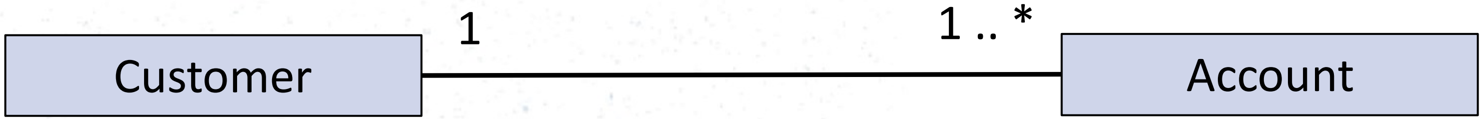
A Person playing the role of employee is associated with a Company playing the role of employer.

Association

- Association multiplicity:
  - Multiplicity defines the number of objects associated with an instance of the association
  - This “How many” is the multiplicity of an association’s role.

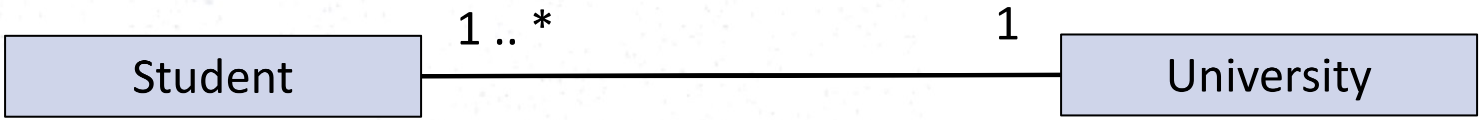
Format	Meaning
1	Exactly one
n	Exactly n
0 .. 1	Zero or one
n .. m	Between n and m (inclusive)
0 .. *	Zero or more
1 .. *	One or more
0 .. 1, 3 .. 5	Zero or one, or between 3 and 5 inclusive

Association (Two Way)



Customer 1 — 1 .. * Account

Customer can own 1 or more Accounts. Account belongs to only one customer.



Student 1 .. * — 1 University

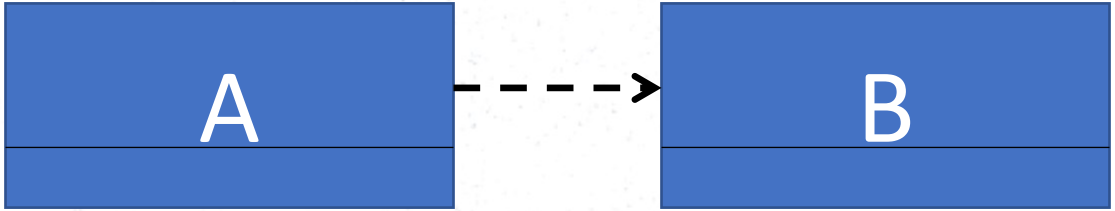
University class has 1 or more students and a student is attached to only one university.

Dependency

- Dependency is a weaker form of relationship which indicates that one class depends on another because it uses it at some point in time.
- It implies that a change to one class may affect the other but not vice versa.

Dependency

- Graphically, a dependency is rendered as a dashed directed line, directed to the thing being depended on.



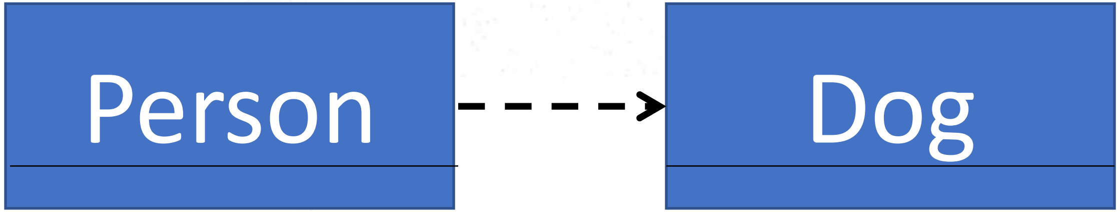
- It mean that;
  - A uses B, but A does not contain as instance of B as part of its own state.

Dependency

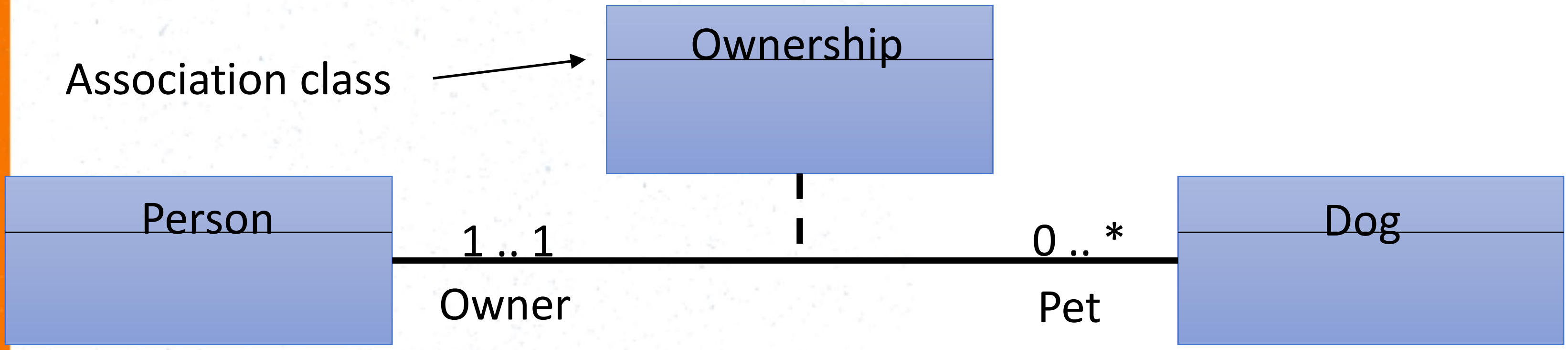
- e.g :
- The “Person” class depends on “Dog” class. Changes to “Dog” class may affect the “Person” class, but not vice versa.
- A “Person” object uses a “Dog” object somewhere

  as a parameter in one of its methods.

  void Person::walk(Dog dog)

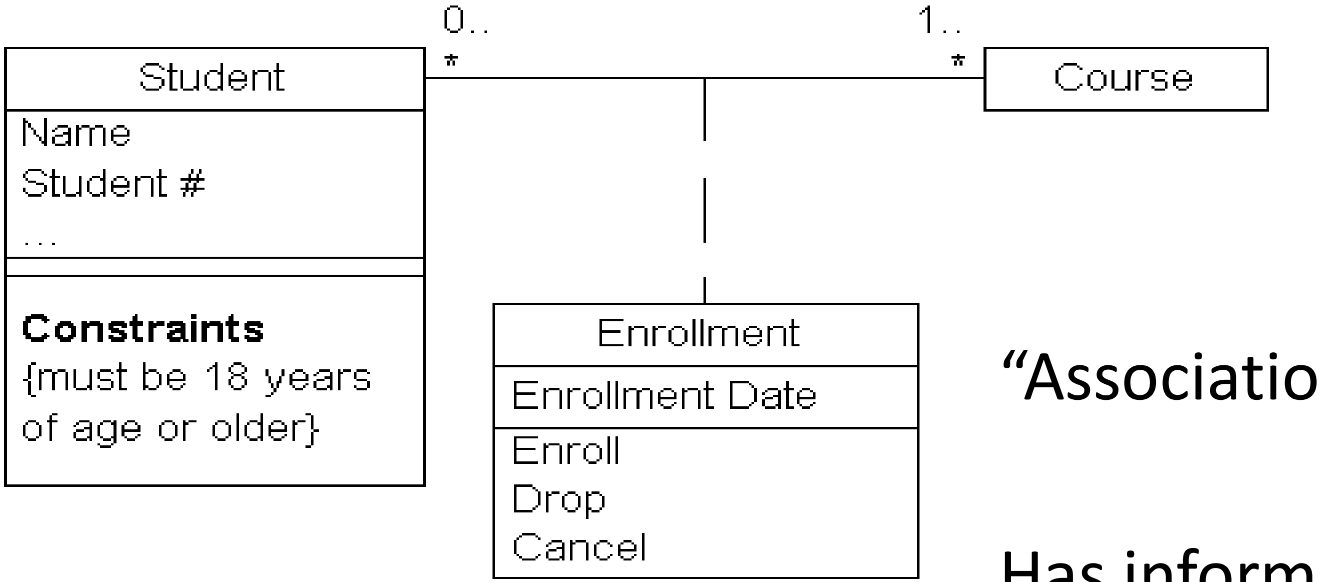


Association class



- The association can also be promoted to a class
- This places the responsibility for maintaining information pertaining to the association with the Ownership class.

Association



"Association Class"

Has information related to the association.

Deciding on Relationships

- Is-A relationship – Inheritence
- Part of – Part cannot exist without the whole – Composition

  e.g. Pages are part of a Book
- Part of – Part can exist without the whole – Aggregation

  e.g. Students are part of a Batch
- Has a – Not a direct Part of Relationship - Association

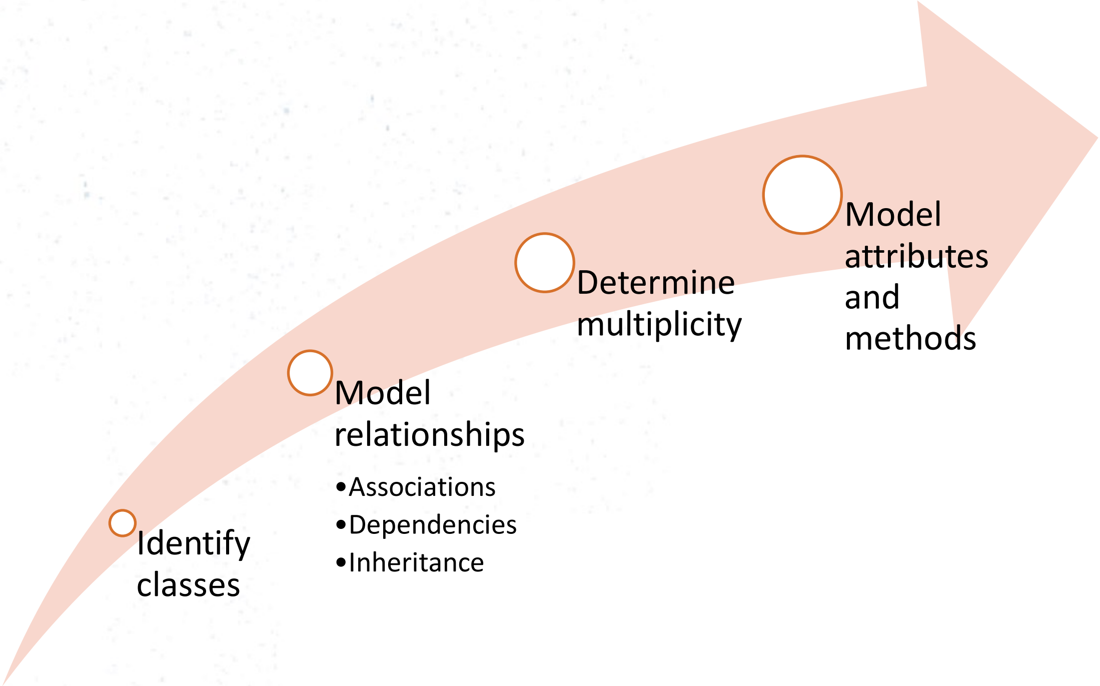
  e.g. Customer has multiple Orders

  Airplane can carry (has) multiple Passengers
- Uses - An object is used as a parameter of a method – Dependency

  e.g. void Player::ThrowDice(Dice mydice);

  Here the ThrowDice method in the Player Class has a parameter of the Dice Class

Creating a Class Diagram



Identify classes

Model relationships

- Associations
- Dependencies
- Inheritance

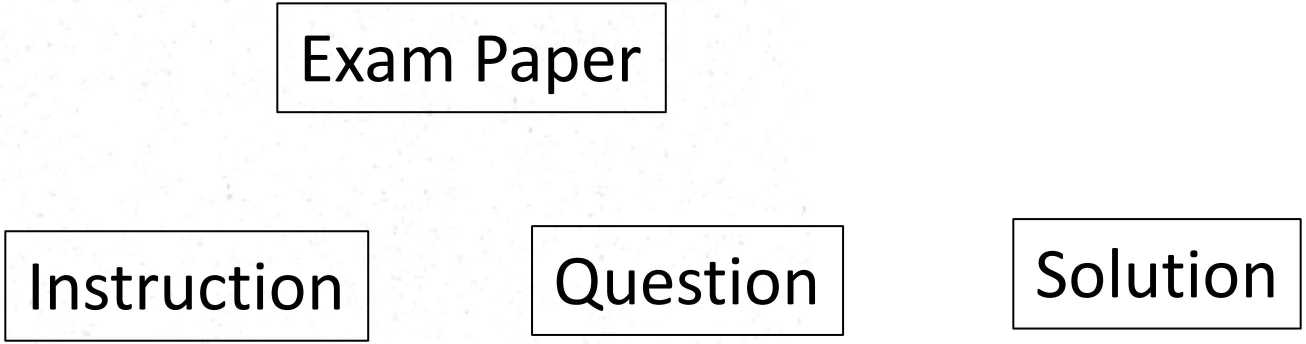
Determine multiplicity

Model attributes and methods

Class Diagram – The Case

- An Exam Paper has many instructions and has one or more Questions. Each Question has a solution.
- Here you can assume that Questions can only be in Exam papers.

Class Diagram – Identifying Classes



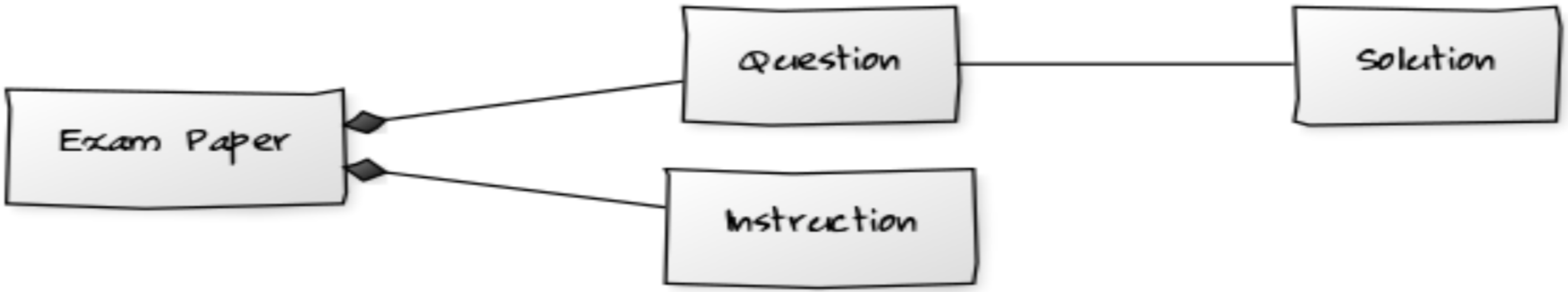
Exam Paper

Instruction

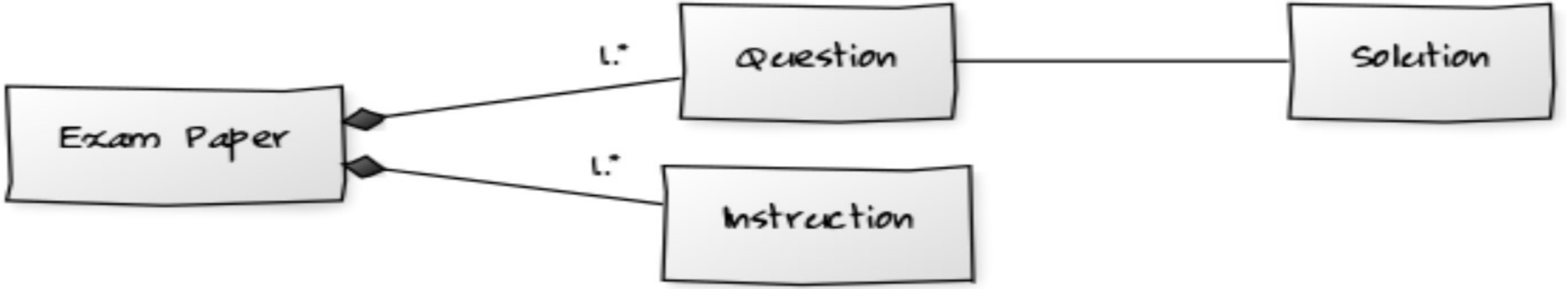
Question

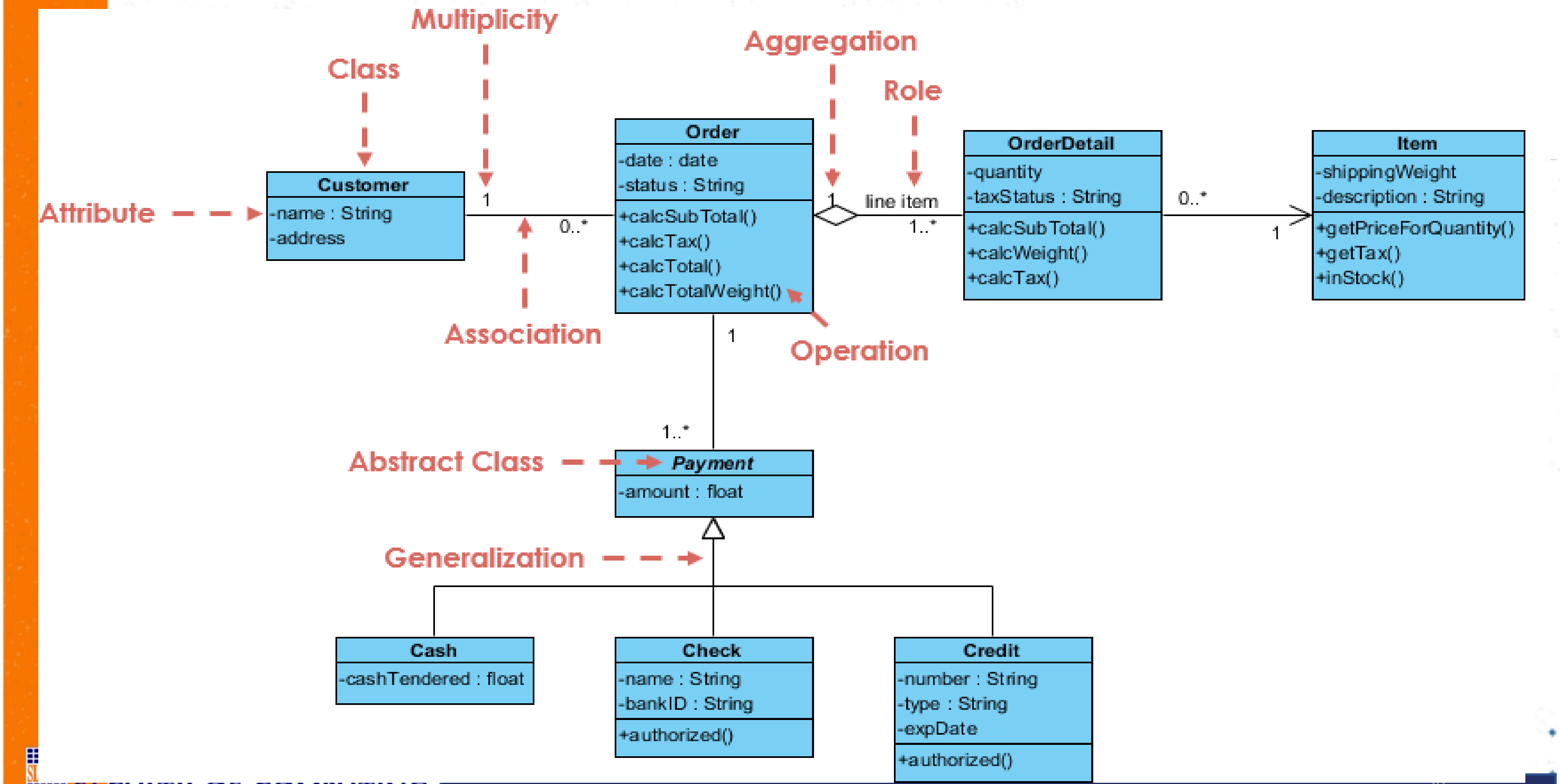
Solution

Class Diagram – Modeling Relationships



Class Diagram – Multiplicity





Multiplicity

Class

Aggregation

Role

Attribute

Association

Operation

Abstract Class

Generalization

References

- UML Distilled by Martin Fowler, chapters 3 and 5
- Fundamentals of Object-Oriented Design in UML by Page-Jones, M (2000). Chapters 4 & 12.