UML Class Diagram:

Class diagrams (structural UML diagrams) are the main building blocks of every object oriented methods. The class diagram can be used *to show the classes, relationships, interface, association, and collaboration.* UML is standardized in class diagrams. Since classes are the building block of an application that is based on OOPs, so as the *class diagram has appropriate structure to represent the classes, inheritance, relationships, and everything that OOPs have in its context.* It describes various kinds of objects and the static relationship in between them.

The main purpose to use class diagrams are:

- This is the only UML which can appropriately depict various aspects of OOPs concept.
- Proper design and analysis of application can be faster and efficient.
- It is base for deployment and component diagram.

UML Class Notation

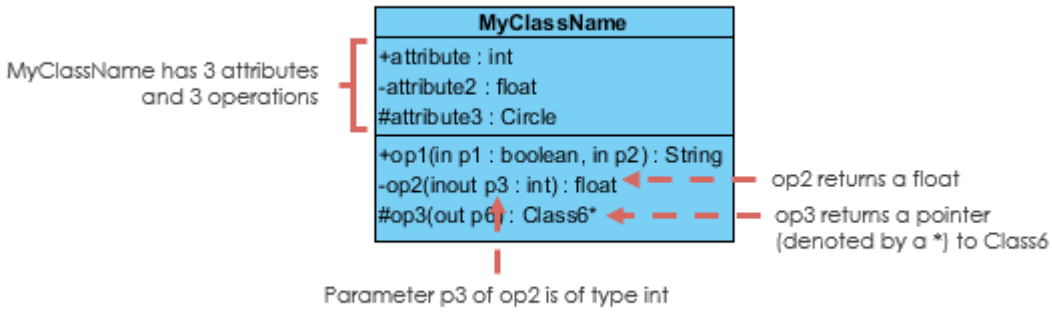
Each class is represented by a rectangle having a subdivision of three compartments *name, attributes and operation.*

There are three types of modifiers which are used to decide the visibility of attributes and operations:

- + is used for public visibility(for everyone)
- # is used for protected visibility (for friend and derived)
- - is used for private visibility (for only me)

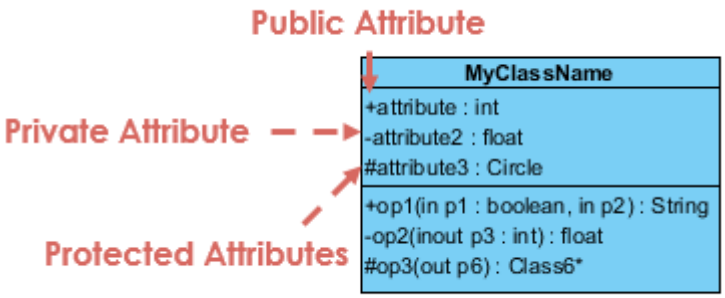
There are several diagram components which can be efficiently used while making/editing the model. These are as follows:

- Class { name, attribute, method}
- Objects
- Interface
- Relationships {inheritance, association, generalization}
- Associations {bidirectional, unidirectional}



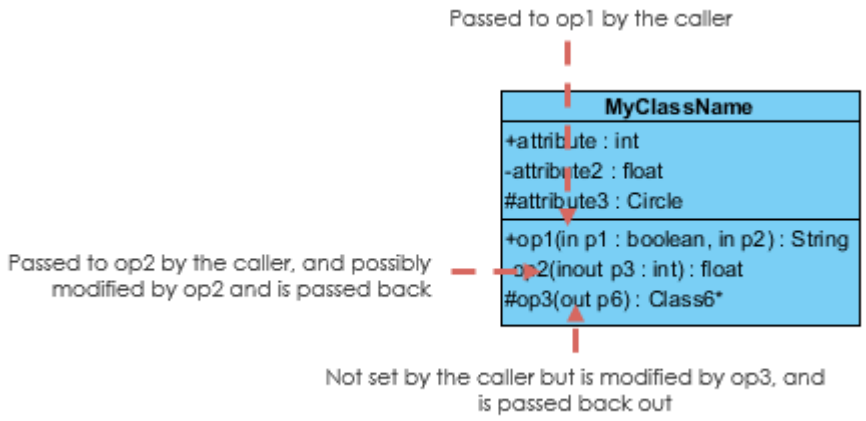
Class Visibility:

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



Parameter Directionality:

Each parameter in an operation (method) may be denoted as in, **out** or **inout** which specifies its direction with respect to the caller. This directionality is shown before the parameter name.



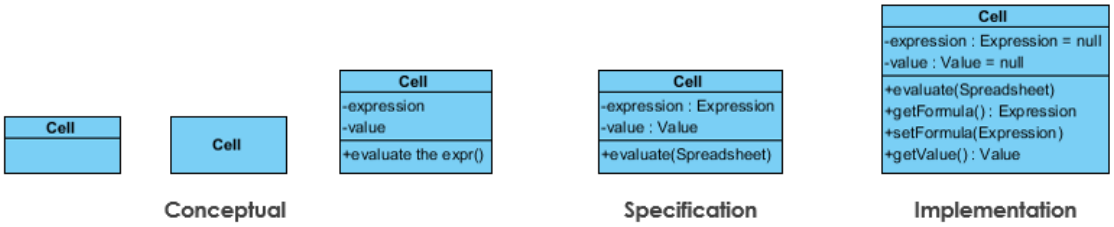
Perspectives of Class Diagram:

The choice of perspective depends on how far along you are in the development process. During the formulation of a **domain model**, for example, you would seldom move past the **conceptual perspective**. **Analysis models** will typically feature a mix of **conceptual and specification perspectives**. **Design model** development will typically start with heavy emphasis on the **specification perspective**, and evolve into the **implementation perspective**.

A diagram can be interpreted from various perspectives:

1. Conceptual: represents the concepts in the domain
2. Specification: focus is on the interfaces of Abstract Data Type (ADTs) in the software
3. Implementation: describes how classes will implement their interfaces

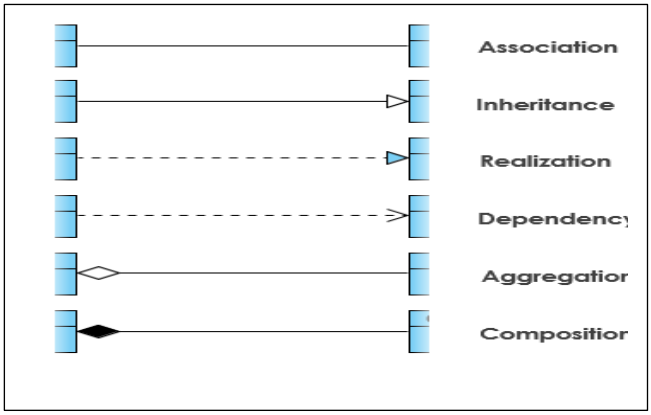
The perspective affects the amount of detail to be supplied and the kinds of relationships worth presenting. As we mentioned above, the class name is the only mandatory information.

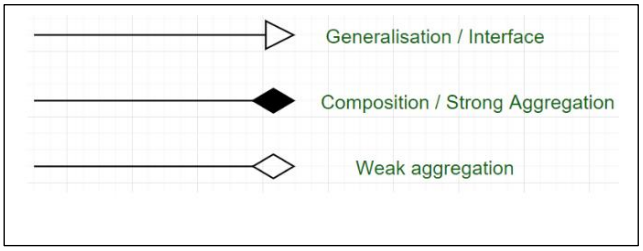


Relationships between classes:

UML is not just about pretty pictures. If used correctly, UML precisely conveys how code should be implemented from diagrams. If precisely interpreted, the implemented code will correctly reflect the intent of the designer.

A relationship can be one of the following types:



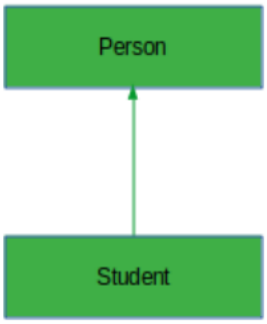


1. Inheritance (or Generalization): التعميم

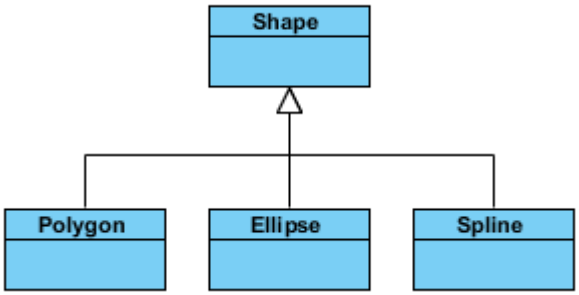
A generalization is a taxonomic relationship between a more general classifier and a more specific classifier. Each instance of the specific classifier is also an indirect instance of the general classifier. Thus, the specific classifier inherits the features of the more general classifier.

- Represents an "is-a" relationship.
- An abstract class name is shown in italics.
- SubClass1 and SubClass2 are specializations of SuperClass.

يساعد التعميم على توصيل صنف فرعي (subclass) بصنفه الأعلى(superclass) ، يستمد الصنف الفرعي متغيراته (يرث) من الصنف الأعلى، لكن لا يمكن استخدام علاقة التعميم هذه لنمذجة تنفيذ الواجهة (interface implementation)، ومن الجدير بالذكر أيضاً أن مخطط الصنف يسمح بإستمداد المتغيرات (التوريث) من عدة أصناف أعلى. في هذا المثال، يتم تعميم الطالب في الفصل الدراسي من صنف الشخص.



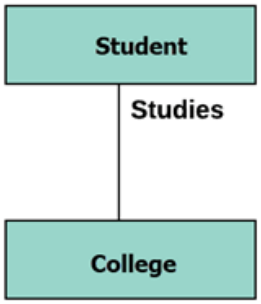
The figure below shows an example of inheritance hierarchy. SubClass1 and SubClass2 are derived from SuperClass.



2. Association: الارتباط

Associations are relationships between classes in a UML Class Diagram. They are represented by a solid line between classes. Associations are typically named using a verb or verb phrase which reflects the real world problem domain.

يمثل هذا النوع من العلاقات علاقات ثابتة بين الفئتين "أ" و "ب". يعني مثلا موظف يعمل في منظمة .في هذا المثال تظهر العلاقة بين الطالب والكلية وهي الدراسات .



Simple Association:

- A structural link between two peer classes.
- There is an association between Class1 and Class2

The figure below shows an example of simple association. There is an association that connects the <<control>> class Class1 and <<boundary>> class Class2.

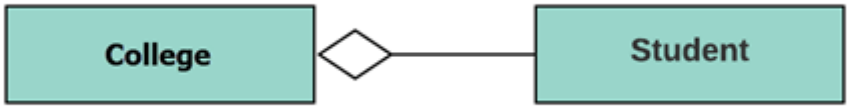
3. Aggregation: التجميع

A special type of association.

- It represents a "part of" relationship.
- Class2 is part of Class1.
- Many instances (denoted by the *) of Class2 can be associated with Class1.
- Objects of Class1 and Class2 have separate lifetimes.

The figure below shows an example of aggregation. The relationship is displayed as a solid line with a unfilled diamond at the association end, which is connected to the class that represents the aggregate.

التجميع (Aggregation) هو نوع خاص من الارتباط يمثل نموذجًا لعلاقة كاملة بين المجموع وأجزائه.



على سبيل المثال، يتكون صنف الكلية من طالب واحد أو أكثر، لا تعتمد الأصناف المتضمنة في التجميع بشكل كامل على المحتويات حيث سنرى هنا أنه سيظل صنف الكلية حتى لو لم يكن الطالب متاحًا.

4. Composition: التكوين

A special type of aggregation where parts are destroyed when the whole is destroyed.

- Objects of Class2 live and die with Class1.
- Class2 cannot stand by itself.

The figure below shows an example of composition. The relationship is displayed as a solid line with a filled diamond at the association end, which is connected to the class that represents the whole or composite.

التكوين (Composition) هو نوع خاص من التجميع الذي يشير إلى ملكية قوية بين صنفين عندما يكون صنف واحد جزءًا من صنف أخر.

على سبيل المثال، إذا كانت الكلية تتكون من فصول للطلبة حيث يمكن أن تحتوي الكلية على العديد من الطلاب، بينما ينتمي كل طالب إلى كلية واحدة فقط، لذلك إذا كانت الكلية لا تعمل فقد تمت إزالة جميع الطلاب أيضًا.

مقارنة بين التجميع و التكوين

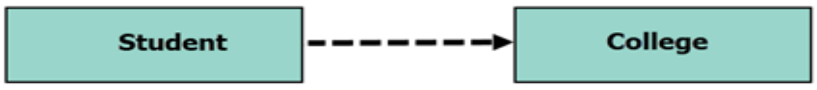
التجميع	التكوين
يشير التجميع إلى علاقة يمكن أن يكون الصنف الطفل فيها موجودًا بشكل منفصل عن الصنف الوالد. مثال: السيارات (الأصل) والسيارة (الطفل)، لذا إذا قمت بحذف السيارات، فإن السيارة الفرعية لا تزال موجودة.	علاقة تركيب ثابت حيث لن يكون الطفل موجودًا بشكل مستقل عن الوالد. مثال: المنزل (الوالد) والغرفة (الطفل)، لذلك نحن نعلم أنه لن تنفصل الغرف عن المنزل أبدًا.

5. Dependency: التبعية

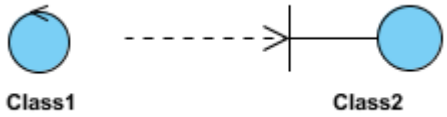
An object of one class might use an object of another class in the code of a method. If the object is not stored in any field, then this is modeled as a dependency relationship.

- A special type of association.
- Exists between two classes if changes to the definition of one may cause changes to the other (but not the other way around).
- Class1 depends on Class2

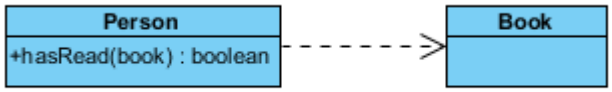
التبعية تعني العلاقة بين صنفين أو أكثر حيث قد يؤدي التغيير في صنف إلى فرض تغييرات في الآخر و تعتبر هذه علاقة ضعيفة حيث تشير التبعية إلى أن صنف واحد يعتمد على صنف أخر .في المثال التالي ، يعتمد الطالب على الكلية التي يتبع لها



The figure below shows an example of dependency. The relationship is displayed as a dashed line with an open arrow.



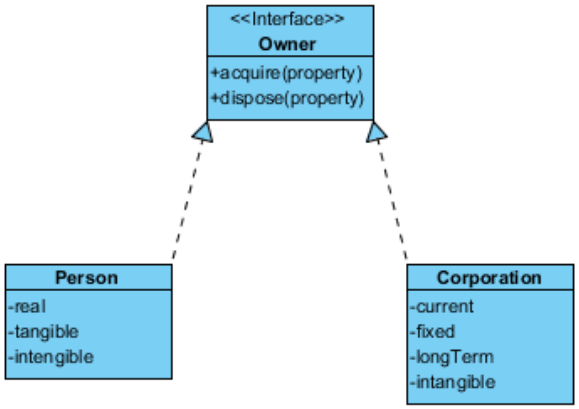
The figure below shows another example of dependency. The Person class might have a hasRead method with a Book parameter that returns true if the person has read the book (perhaps by checking some database).



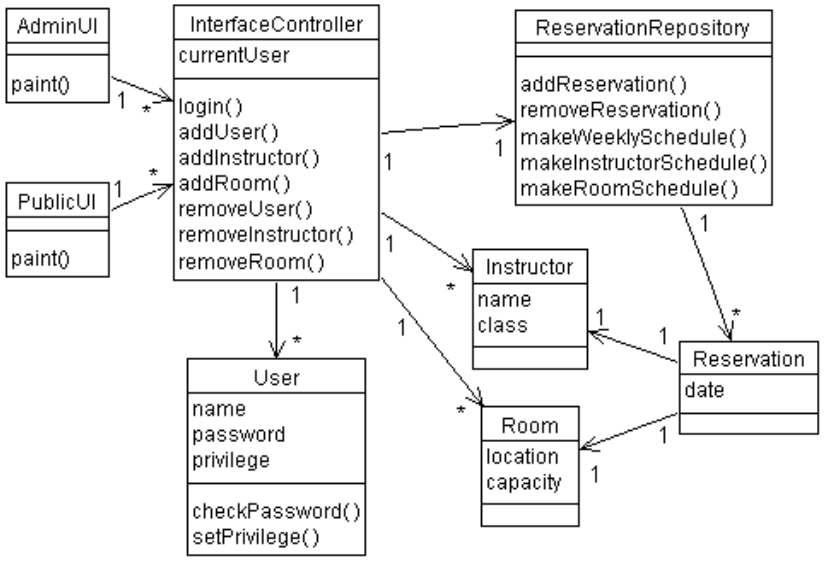
6. Realization:

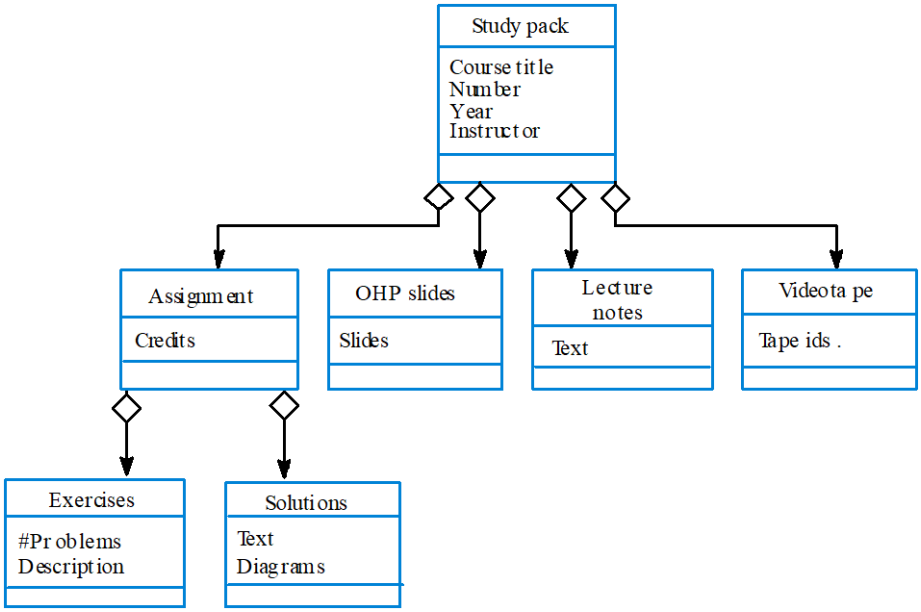
Realization is a relationship between the blueprint class and the object containing its respective implementation level details. This object is said to realize the blueprint class. In other words, you can understand this as the relationship between the interface and the implementing class.

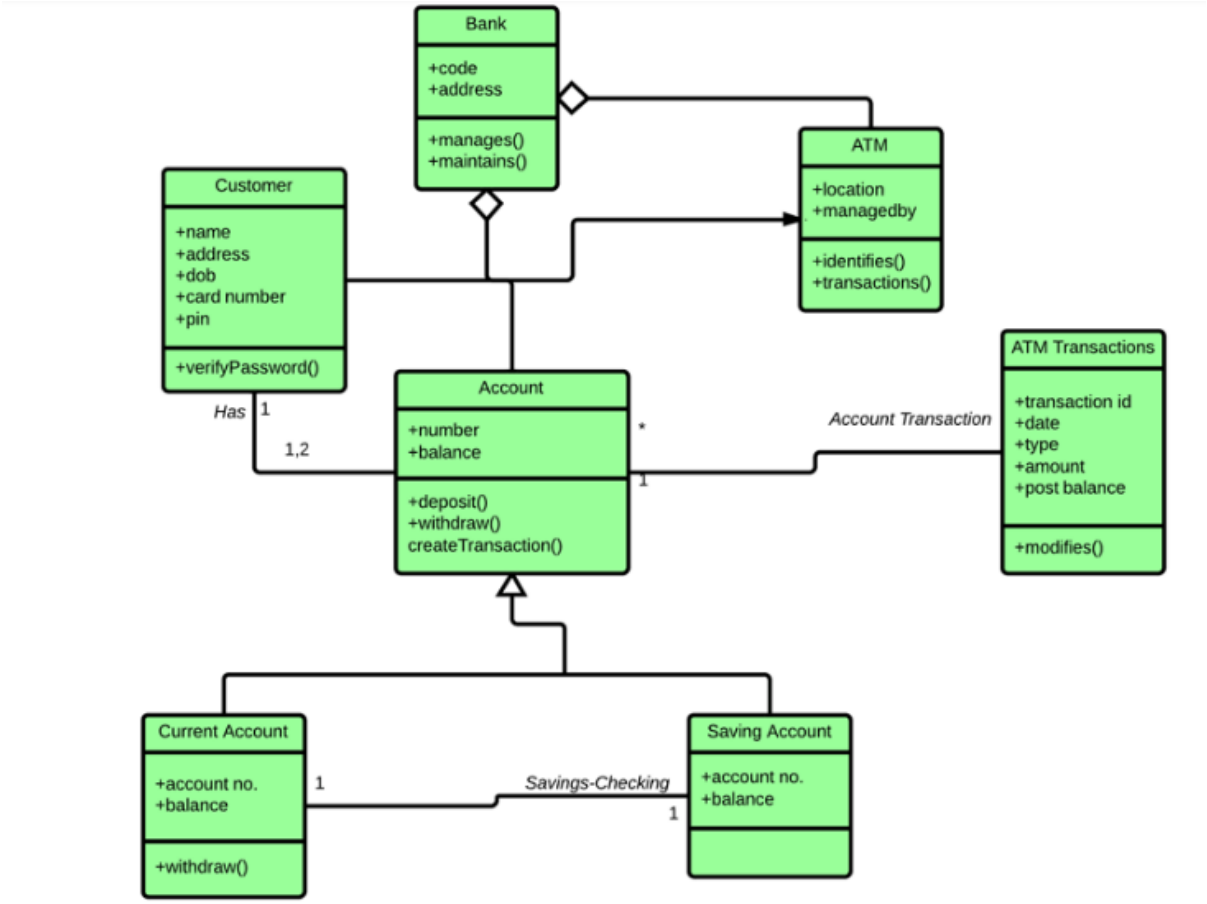
For example, the Owner interface might specify methods for acquiring property and disposing of property. The Person and Corporation classes need to implement these methods, possibly in very different ways.



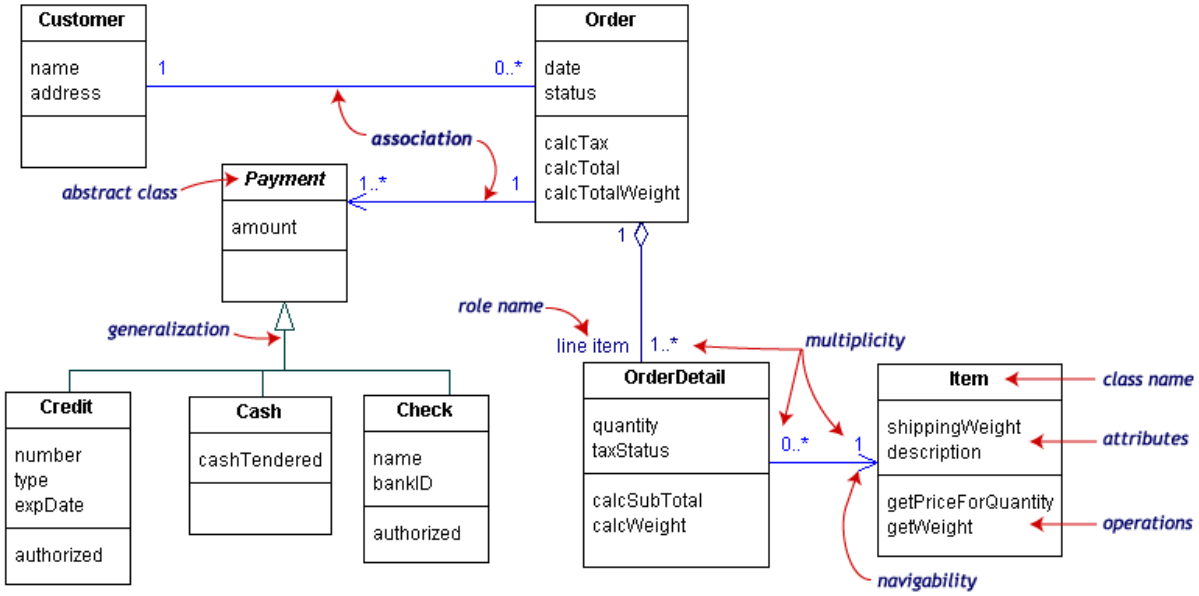
- **Class diagram examples (A classroom scheduling system: specification perspective.)**


AdminUI
paint()
PublicUI
paint()
InterfaceController
currentUser
login()
addUser()
addInstructor()
addRoom()
removeUser()
removeInstructor()
removeRoom()
ReservationRepository
addReservation()
removeReservation()
makeWeeklySchedule()
makeInstructorSchedule()
makeRoomSchedule()
Instructor
name
class
Reservation
date
User
name
password
privilege
checkPassword()
setPrivilege()
Room
location
capacity

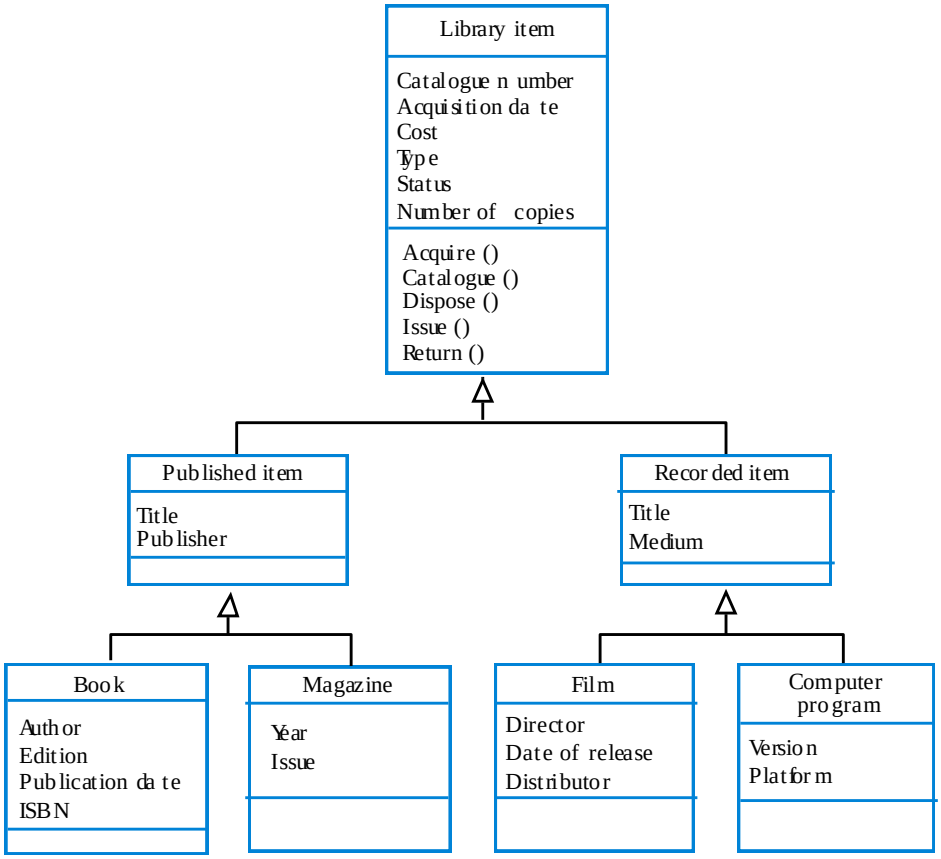

Study pack
Course title
Number
Year
Instructor
Assignment
Credits
OHP slides
Slides
Lecture notes
Text
Videota pe
Tape ids .
Exercises
#Problems
Description
Solutions
Text
Diagrams



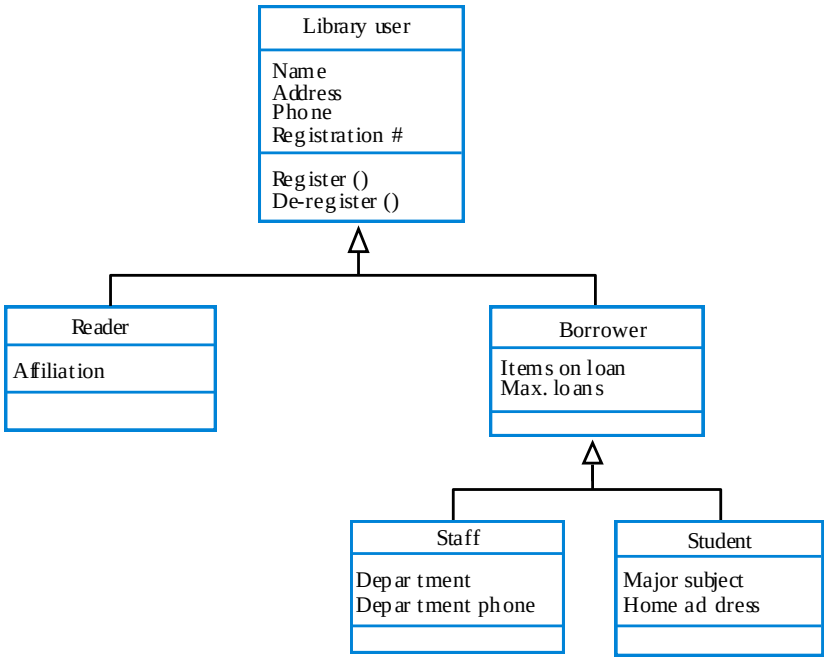
- **UML Class Example**



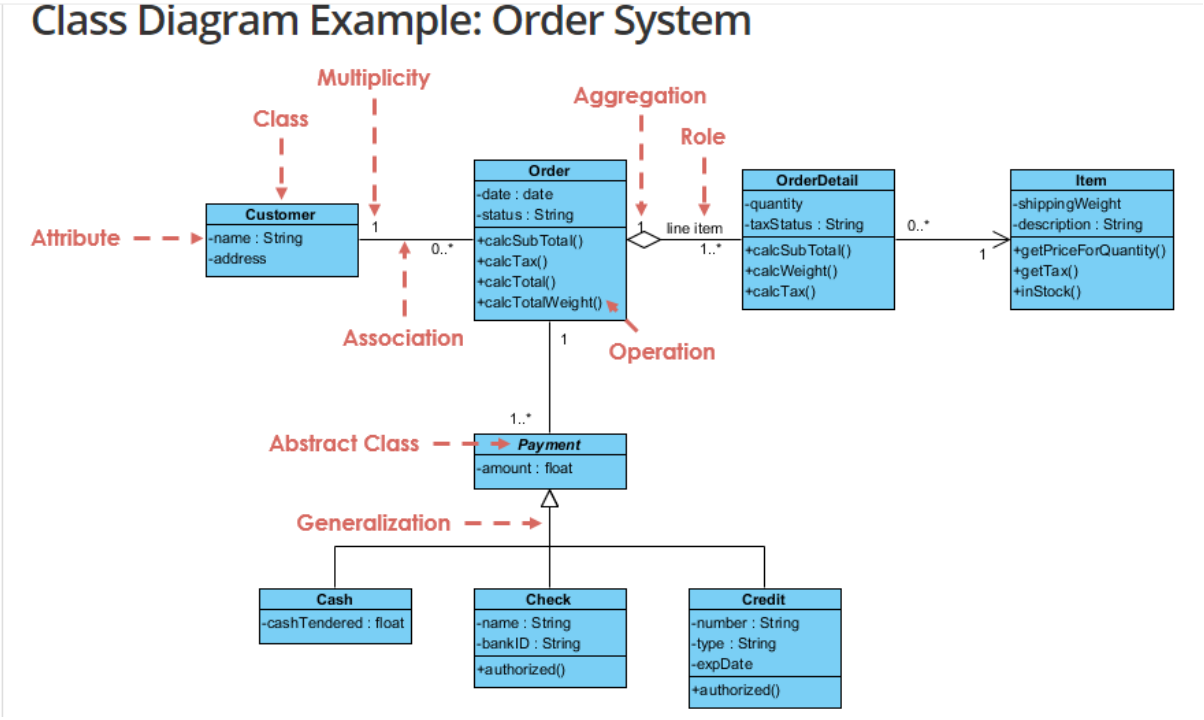
- **UML class diagrams** : Library class hierarchy



- **UML class diagrams**: User class hierarchy



-



A class diagram may also have notes attached to classes or relationships.