Tutorial - UML Introduce 2

> Designed by ZHU Yueming
>
> Reference
>
> 1. Bernd Bruegge and Allen H.Dutoit, Object Oriented Software Engineering Using UML, Patterns, and Java Third Edition
> 2. Zhang Yuqun, Slides of Object Oriented Analyze and Design
> 3. Wang Wenmin, Slides of Object Oriented Analyze and Design

Experimental Objective

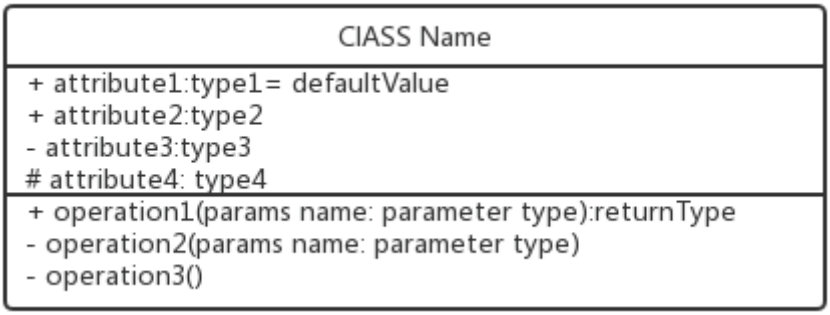
1. Learn how to design class diagram and the relationship between different class diagrams.
2. You need to understand how to write down the corresponding code to describe class diagrams and how to design class diagrams to describe the structure of your code.

Topic 2. Class Diagram

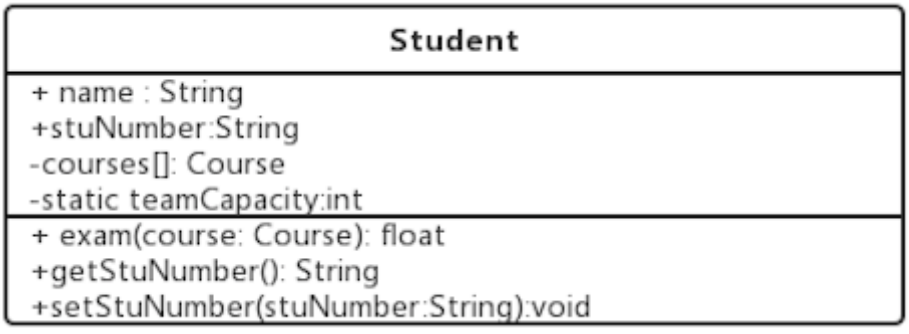
1. General Introduction

Class diagrams provide a way to document the structure of a system by defining what classes there are within it and how they are. In UML, classes and objects are depicted by boxes composed of three compartments. The top compartment displays the name of the class or object. The center compartment displays its attributes, and the bottom compartment displays it operations.

This is a model of class diagram.



This is an example of class diagram.



Following java code is a code framework for the class diagram above

```
public class Student {
```

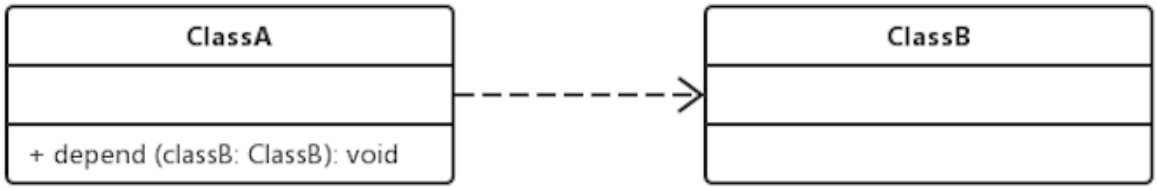
```
        private String name;
        private String stuNumber;
        private Course courses[];
        private static int teamCapacity;

        public float exam(Course course){
            return 0;
        }
        public String getStuNumber(){
            return null;
        }
        public String setStuNumber(String stuNumber){
            return null;
        }
    }
```

2. Relationship between classes

Classes are interrelated to each other in specific ways. In particular, relationships in class diagrams include different types of logical connections. The followings are such types of logical connections that are possible in UML:

(1) Dependency (依赖)

The UML *dependency* is the weakest relationship of them all. It means that one class uses the object of another class in method, and the object of another class is not declared in field.




```
public class ClassA {
public void depend(ClassB classB){}
}
public class ClassB {
}
```

(2) Association(关联)

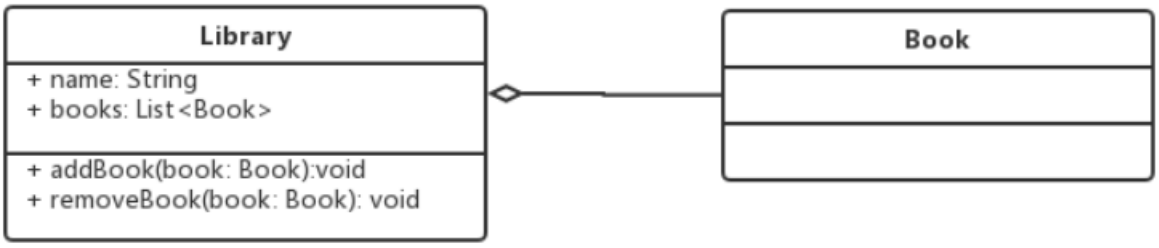
Unidirectional Associations(单向关联): The fields in `classA` contain at least one instances of `classB`, but the fields in `classB` does not contain any instances of `classA`.

Bidirectional Associations(双向关联): The fields in both `classA` and `classB` contain at least one instances of each other's.

(3) Aggregation(聚合)

It is a specific case of **Association** (unidirectional association) that has following features:

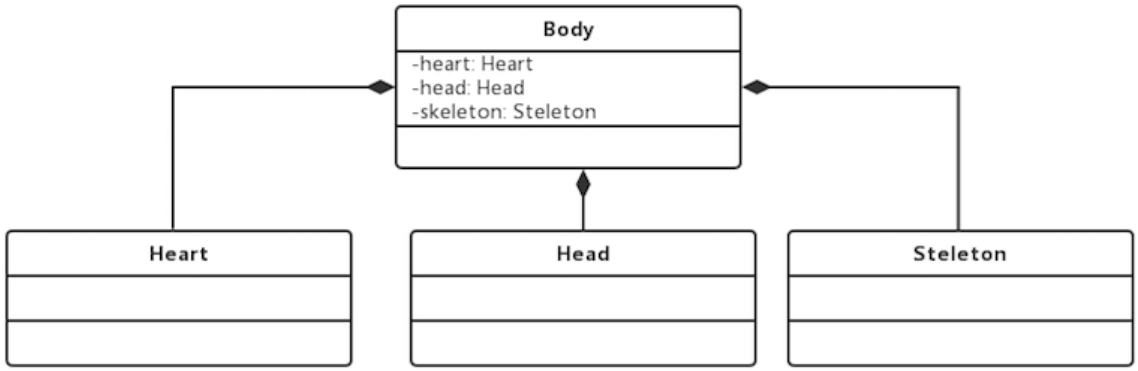
- The fields in `classA` contain at least one instances of `classB`
- Often describes "has a" relationship. `ClassA` has a `ClassB`, Library has a Book etc.
- `ClassB` can exist dependency of `ClassA` which means when destroy `ClassA` the instance of `ClassB` still exist.



(4) Composition(组合)

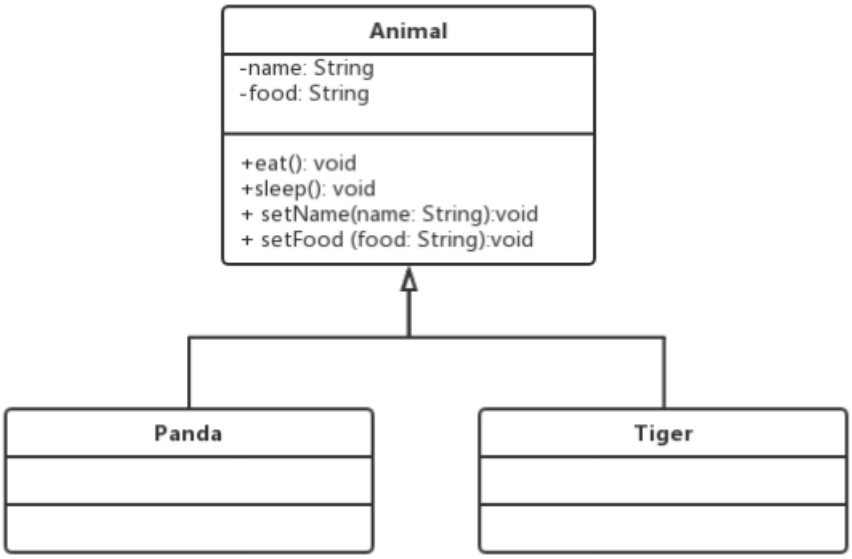
It is also a kind of **Association** but it describes a stronger dependency between two classes. Composition always has following features:

- The fields in `classA` contain at least one instances of `classB`
- Often represent a "part of" relationship. `ClassB` is a part of `ClassA`
- `ClassB` cannot exist dependency of `ClassA` which means when destroy `classA` the instance of `classB` in `classA` would not exist anymore. For example, a heart can not exist without a body.
- It often describe the inner class.



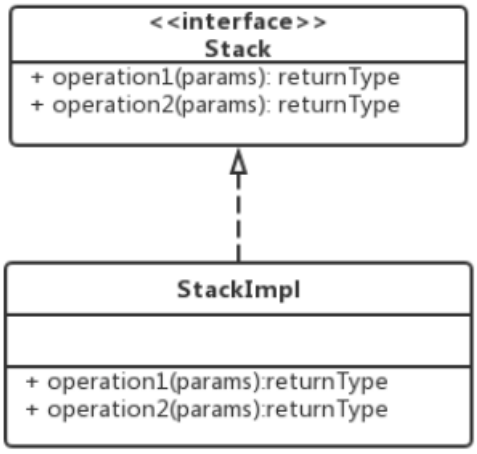
(5) Inheritance and Generalization(继承)

The inheritance relationship between classes.



(6) Realization(实现)

`classB` implements `classA`, so that `classA` always be an interface.



3. Basic Exercise

- **Exercise 1**

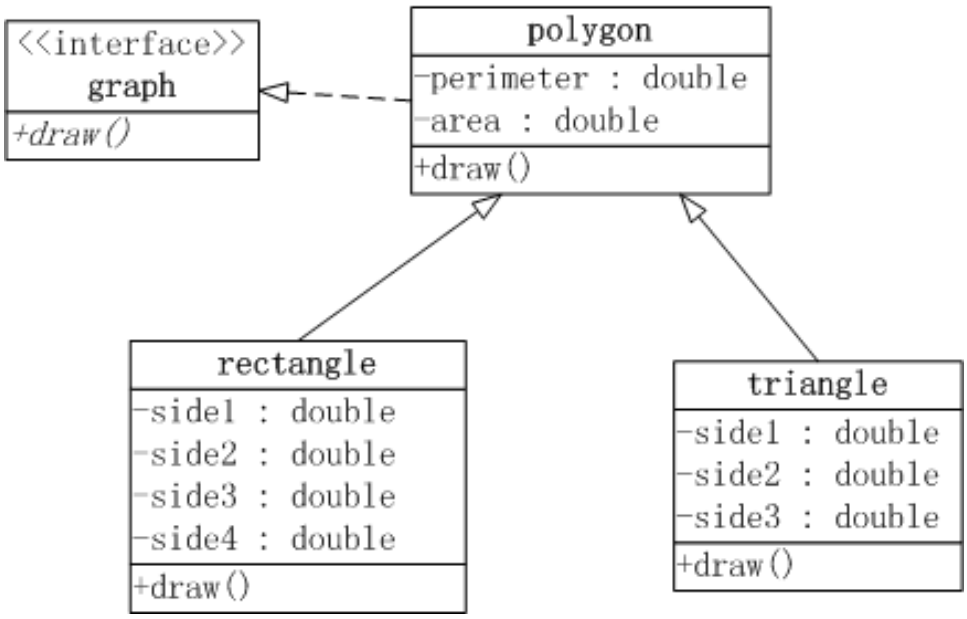
  According to the code framework below, please design the corresponding class diagram:

```
public class School {
    private List<Department>departments;

    public void SetUpDepart(){
        departments.add(new Department());
    }
}

public class Department{
    private Teacher[] teacher;
    public void displayTeacherName(){
    }
}
```

```
public class Teacher{
    private School school;
    private Department deparment;

    public void setSchool(School school){
        this.school=school;
    }
    public void setDepartment(Department department){
        this.deparment=department;
    }
}
```

- **Exercise 2**

  According to the class diagram below, please design the corresponding code framework.



4. ECB pattern

From the perspective of the functional differences, classes often divided into three different categories.

- **Entity class**:

  Entity class represents entities used to encapsulate data, transfer instance. (e. g. Customer, Record)

- **Control class**:

  Control class is the medium of interaction between entity class and boundary class, which is often used to describe a series of logical processes.

- **Boundary class**

  Boundary class represents the boundary of interaction with system actors. (e. g. Page)

For more detailed introduction: click here

5. Comprehensive Exercise

5.1. WeChat Payment

Mobile payment is very popular in China. Many people use `WeChat` as their third-party payment platform. The following description is a simple design of its payment function.

You can do multiple operations in `WeChat` pay, including checking how much balance you have in your wallet, adding bank cards to recharge or withdraw, looking up transaction history, sending lucky money to your contacts, and making payment to strangers by a QR code. In last case, we have two ways of transaction, either the payer scanning the QR code of payee's or the payee scanning the QR code of payer's. In addition, user also can change his nickname, password or other information.

Question 1: Draw an use case diagram according to the above scenario

Question 2: Class diagram: Finding out entity class according to your design, and given the class diagram. In this section, you can only provide the class name, necessary attributes, and the indicate the relationship between those classes.

2.2. Parting

In ShenZhen city, a automatic parking payment system is widely used by almost all shopping malls. Then, it is a time for you to help a newly opened shopping mall design the system. The following description is a simple requirement of the parking management:

Before a car enter the parking lot, there is an automatic scanning device that can register the time and the license plate number of the car. Before leaving, the customer has to pay the parking fee, otherwise he/she cannot drive out of the parking lot. When paying, the customer can pay the parking fees directly, or he/she can choice to deduct the payment with his/her VIP points if he/she is the VIP of this shopping mall. Customers need to drive out of the parking lot within 30 minutes after the successful payment time, otherwise, the cost will be recalculated. There are several administrators who can view the parking list, and they can also query and modify the parking info of each car. On the other hand, in order to promote sales, the shopping mall set a VIP Day on the 1st day of each month. On the 12:00 am in each VIP Day, the system could send points in each VIP account automatically.

Question 1: Draw an use case diagram according to the above scenario

Question 2: Class diagram: Finding out entity class according to your design, and given the class diagram. In this section, you can only provide the class name, necessary attributes, and the indicate the relationship between those classes.

In this section, you need to indicate the class name, relevant attributes, the necessary methods (at least include the ones listed below) and the relationship between those classes. Please highlight the control class according to your design.

- findAllCarRecord
- findOneCarRecord
- modifyOneCarRecord
- createCarRecord
- getCarObjectByPlateNumber
- payment
- calculateFee
- changeCarStatus

- getEnterTime
- getLeavingTime
- addvipPoint
- consumevipPoints