Tutorial 2 UML

[Experimental Objective]

1. Learn how to design use case diagram according to a paragraph of requirement description.
2. Understand different relationship between use cases.
3. Learn how to design class diagram and the relationship between different class diagrams.
4. 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.

[General Description]

Software engineers often have to communicate properties of the systems they develop with other stakeholders who may or may not be familiar with the technical details of the system that is being examined. UML provides a unified, consistent way to communicate information about software systems in a way designed to be intuitive for both technical and non-technical individuals. Many tools exist that support working with UML that serve to support the development of a UML model and expressing that model by generating graphical artifacts as specified by the UML standard.

In this laboratory, you can use any plot tools to accomplish exercise. We recommend **microsoft visio** or **ProcessOn** (online software). In this semester, we'll take two weeks to develop three most common UML diagrams:

1. Use case diagram
2. Class model diagram
3. Sequence diagram

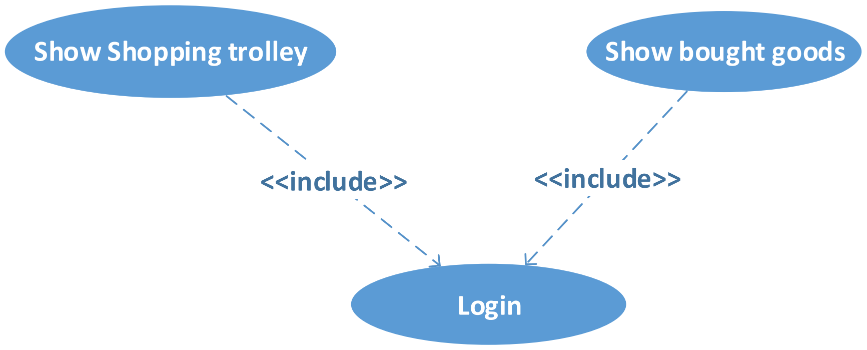
1. Use case diagram

Actor: An actor in a use case is an ***external*** agent that uses or interacts with the system. An actor can be a user or a role, such as a person or an external system characteristic of the environmental, such as time or temperature change.

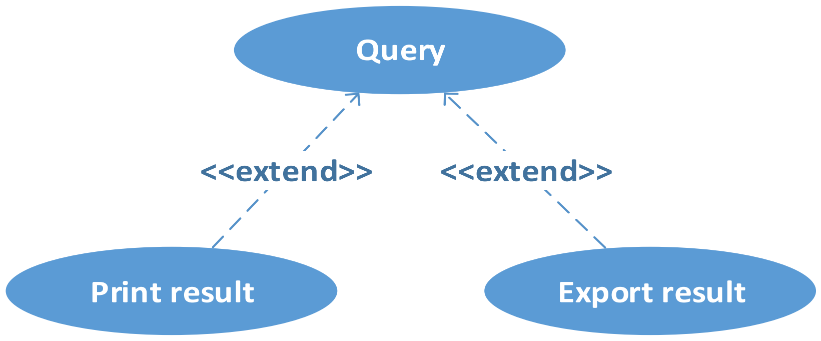
Use case: A use case is a collection of related scenarios, including normal and alternative scenarios. The scenarios can be regarded as a behaviorally related sequence of steps, automated or manual, for the purpose of completing a business task

Relationships of Use case diagram:
When describing a complex system, its use case model can become quite complex and can contain redundancy. We use these three types of relationships including inclusion, extension, and inheritance to reduce the complexity of the model.

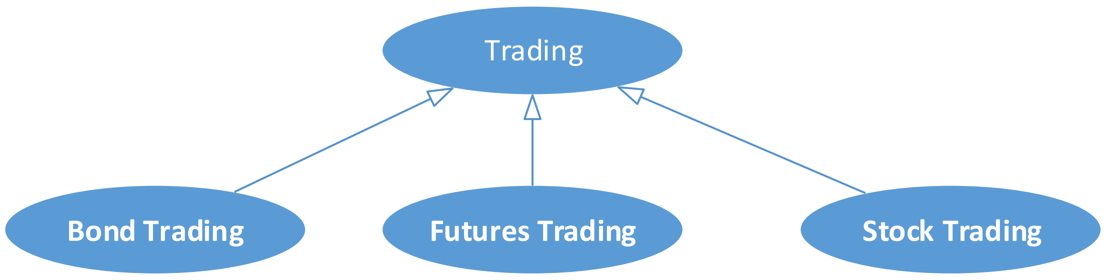
(1) Include (包含): Include is a directed relationship between two use cases, implying that the behavior of the included use case is inserted into the behavior of the including use case.



(2) Extend (扩展): This relationship specifies that the behavior of a use case may be extended by the behavior of another use case.



(3) Generalization (泛化): 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.



Exercise1
Translate the following English statements into a representative use case diagram:
Library System: The system shall have the following four types of actors: BookBorrower, JournalBorrower, Browser, and Librarian. A user playing the role of a BookBorrower shall be able to borrow a book, return a book, and extend the loan of a book. A JournalBorrower shall be able to borrow a journal and return a journal. A Browser shall be able to browse, and a librarian shall be able to update the catalog.

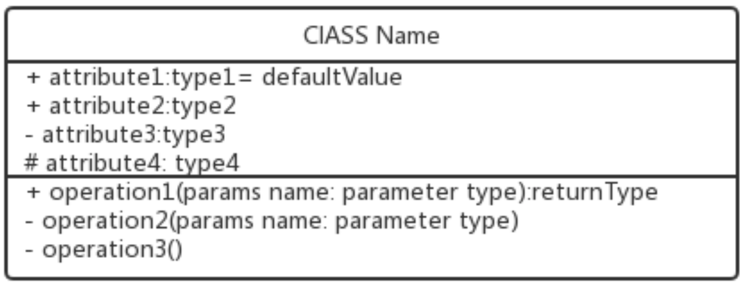
Exercise2
Translate the following English statements into a representative use case diagram. In this diagram you should add the use case relationship of inclusion and extension.
Fried System: When a fire occurs, the supervisor needs to report the disaster, and then the system will automatically send the disaster to dispatcher. After the dispatcher receives the report, he can send the fire resource allocation request to the system according to the fire situation and accident location. In this step, the dispatcher may

choice to open the map finding the specific location of disaster. At this point, the system will automatically calculate whether to meet the needs of dispatchers. When the fire resources in the inventory information database meet the needs of the dispatcher, the dispatcher can allocate the corresponding fire resources including allocating fire extinguishers and allocating fire trucks.
There are two actors in this use case diagram: supervisor and dispatcher. You need to find the use cases and the relationship of these use cases. Please begin to finish this use case diagram.

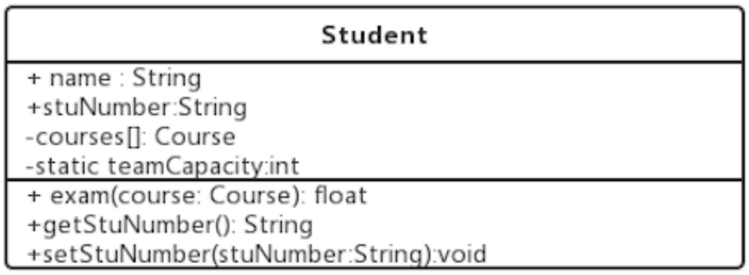
2. Class Diagram

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


ClASS Name
+ attribute1:type1= defaultValue
+ attribute2:type2
- attribute3:type3
# attribute4: type4
+ operation1(params name: parameter type):returnType
- operation2(params name: parameter type)
- operation3()

This is an example of class diagram



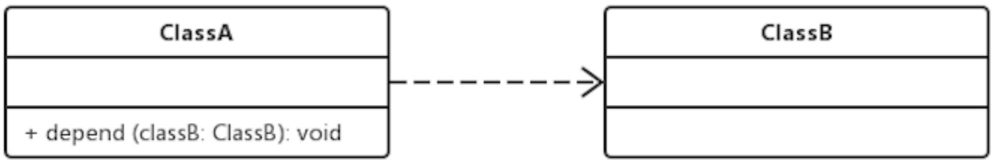
Following code is a code framework for the class diagram above

```
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;
      }
}
```

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

(1) Dependency (依赖)

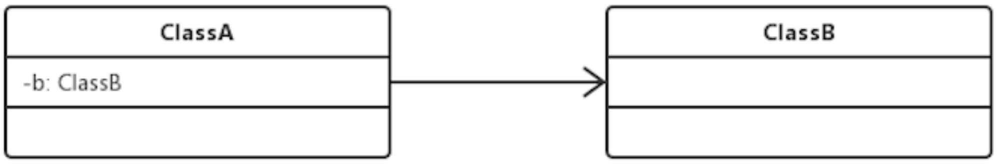
The UML *dependency* is the weakest relationship of them all. It means that one class uses the object of another class.



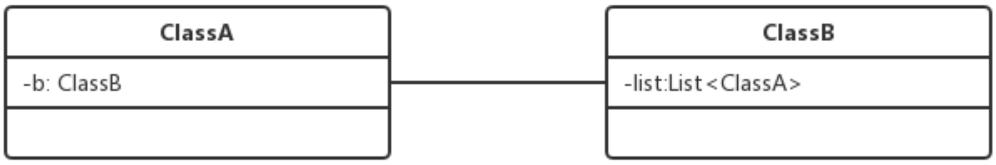
```
public class ClassA {
      public void depend(ClassB classB){}
}
public class ClassB {
}
```

(2) Association(关联)

Unidirectional Associations(单向关联): classA contains at least one instances of classB, but B does not contain any instances of classA.



Bidirectional Associations(双向关联): both classA and classB contains at least one instances of each other's.

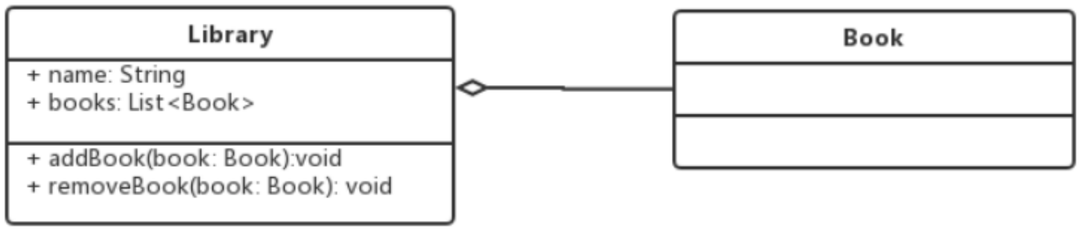


(3) Aggregation(聚合)

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

- ClassA needs to have at least one property of ClassB.
- Often decribes "**has a**" relationship. ClassA has a ClassB, Library has a Book etc.

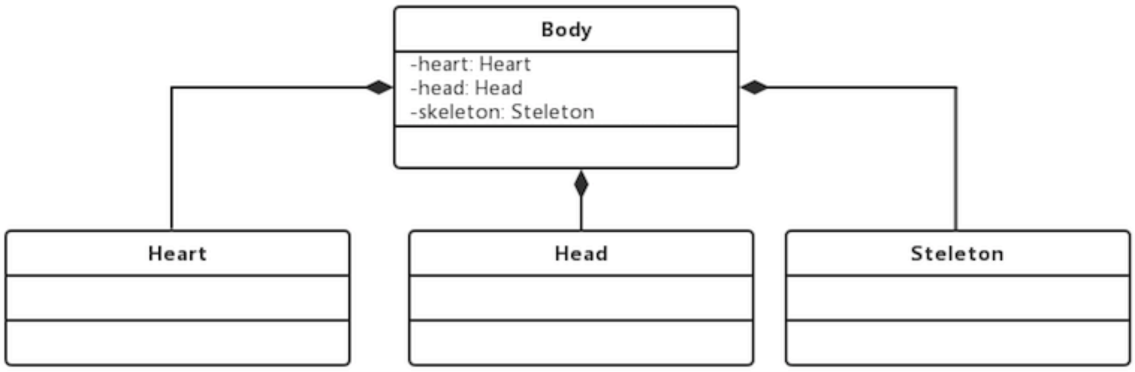
- ClassB can exist dependenty of ClassA which means when delete ClassA the ClassB still exist.



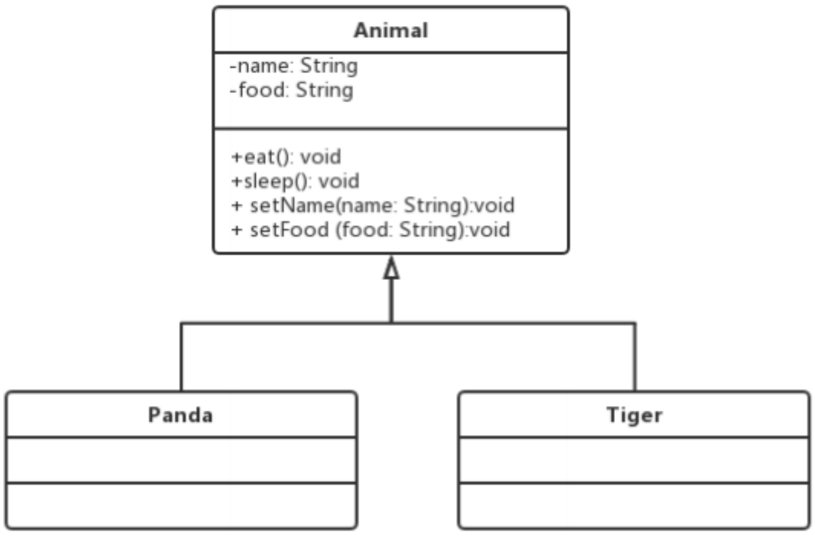
(4) Composition(组合)

It is also a kind of Association but it describes a stronger dependecy between two classes.

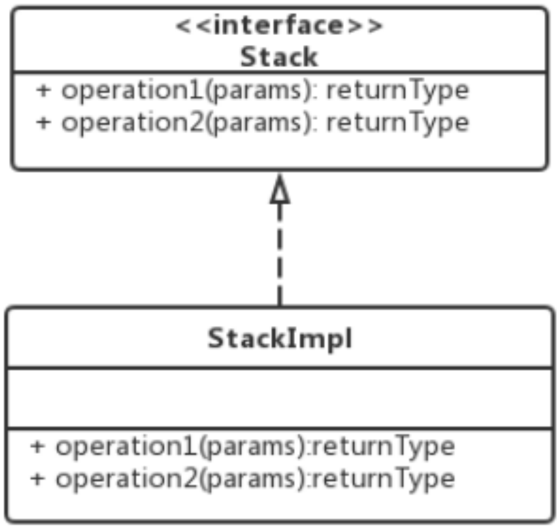
- ClassA needs to have at least one property of ClassB.
- Often represent a “**part of**” relationship. ClassB is a part of ClassA.
- ClassB cannot exist dependency of ClassA. For example, a heart can not exist without a body.



(5) Inheritance and Generalization(继承)



(6) Realization(实现)



<<interface>>
Stack
+ operation1(params): returnType
+ operation2(params): returnType
StackImpl
+ operation1(params):returnType
+ operation2(params):returnType

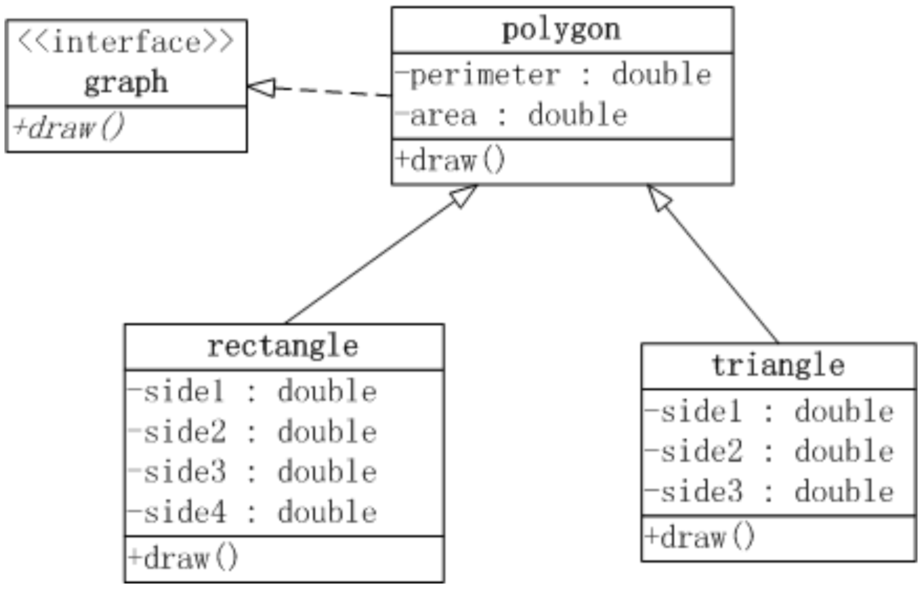
Exercise3

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;
      }
}
```

Exercise4

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



Comprehensive Exercise: 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.

Suppose there can be two type of servers: one is used for manage account information, including payment password, another is used for manage user wallet, which is responsible for handling money transaction. When transferring money, you should provide identification information to the account manager server. If the identification information is incorrect, WeChat will display an error message. Otherwise, the transferring notification will be send to the receiver, and he/she can confirm this transaction to receive the money. However, if the money is not received within 24 hours, the transaction will be expired and money will be returned back to the payer. The payer will be notified in both cases.

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.

Question 3

Please given the code structure according to your class diagram.

[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*

[What to Submit]

None