

CS 320

Introduction to Software Engineering
Spring 2017

February 27, 2017

Today

UML crash course

- Class diagrams

Introduction to Software Design

- A little refresher on object-oriented programming

UML crash course

The main questions

- What is UML?
- Is it useful, why bother?
- When to (not) use UML?

What is UML?

- Unified Modeling Language.
- Developed in the mid 90's, improved since.
- Unifies existing, disparate notations.
- Standardizes the notation for modeling OO systems.

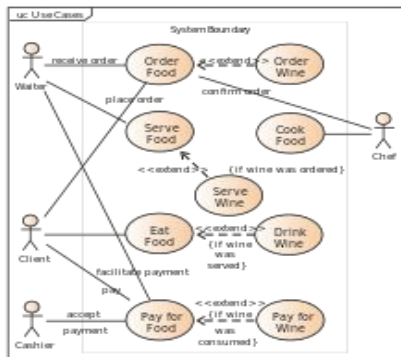
What is UML?

- Unified Modeling Language.
- Developed in the mid 90's, improved since.
- Unifies existing, disparate notations.
- Standardizes the notation for modeling OO systems.
- A collection of diagrams for different viewpoints:
 - Use case diagrams
 - Component diagrams
 - Class and Object diagrams
 - Sequence diagrams
 - Statechart diagrams
 - ...

What is UML?

- Unified Modeling Language.
- Developed in the mid 90's, improved since.
- Unifies existing, disparate notations.
- Standardizes the notation for modeling OO systems.
- A collection of diagrams for different viewpoints:
 - **Use case diagrams**
 - Component diagrams
 - Class and Object diagrams
 - Sequence diagrams
 - Statechart diagrams
 - ...

Use case diagrams



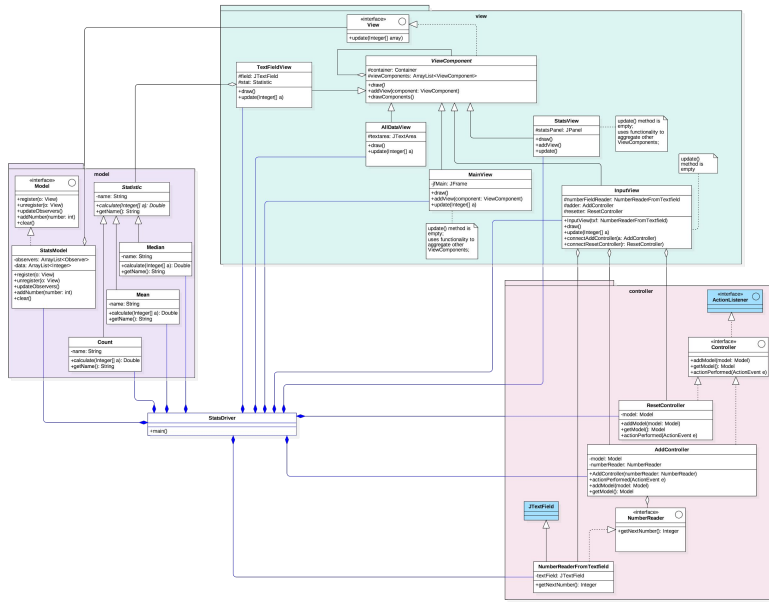
“For reasons that remain a mystery to me, many people have focused on the stick figures and ellipses in use case writing since Jacobson's first book came out, and neglected to notice that use cases are fundamentally a text form.”

Writing Effective Use Cases, Alistair Cockburn, 2000.

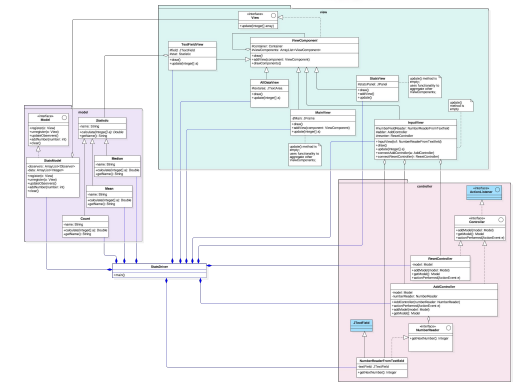
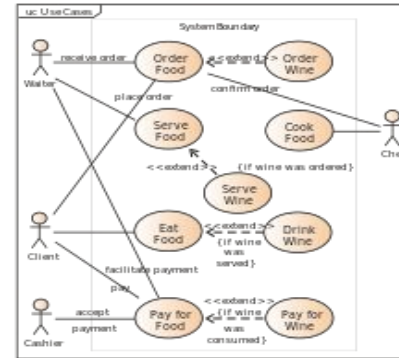
What is UML?

- Unified Modeling Language.
- Developed in the mid 90's, improved since.
- Unifies existing, disparate notations.
- Standardizes the notation for modeling OO systems.
- A collection of diagrams for different viewpoints:
 - Use case diagrams
 - Component diagrams
 - **Class and Object diagrams**
 - Sequence diagrams
 - Statechart diagrams
 - ...

Class diagrams



Are UML diagrams useful?



Are UML diagrams useful?

Communication

- Forward design (before coding)
 - brainstorm ideas (on whiteboard or paper)
 - draft and iterate over software design

Documentation

- Backward design (after coding)
 - obtain diagram from code

Code generation

- Automatically derive code from diagrams



Are UML diagrams useful?

Communication

- Forward design (before coding)
 - brainstorm ideas (on whiteboard or paper)
 - draft and iterate over software design

Documentation

- Backward design (after coding)
 - obtain diagram from code

Code generation

- Automatically derive code from diagrams



Code generation can be useful for skeletons.

Classes vs. objects

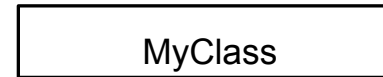
Class

- Grouping of similar objects.
 - Student
 - Car
- Abstraction of common properties and behavior.
 - Student: Name and Student ID
 - Car: Make and Model

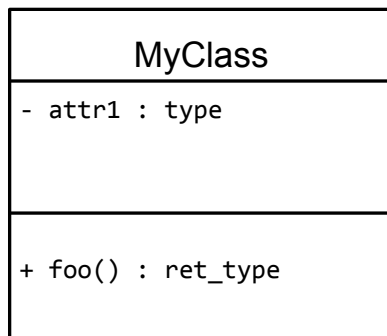
Object

- Entity from the real world.
- Instance of a class
 - Student: Joe (4711), Jane (4712), ...
 - Car: Audi A6, Honda Civic, ...

UML class diagram: basic notation



UML class diagram: basic notation



Name

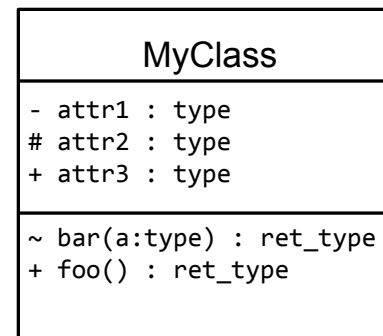
Attributes

<visibility> <name> : <type>

Methods

<visibility> <name>(<param>) : <return type>*
<param> := <name> : <type>

UML class diagram: basic notation



Name

Attributes

<visibility> <name> : <type>

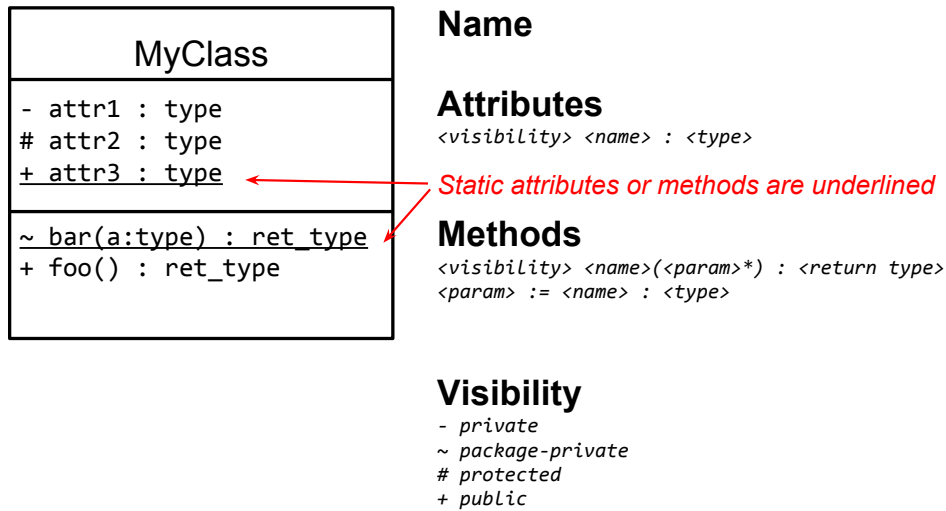
Methods

<visibility> <name>(<param>) : <return type>*
<param> := <name> : <type>

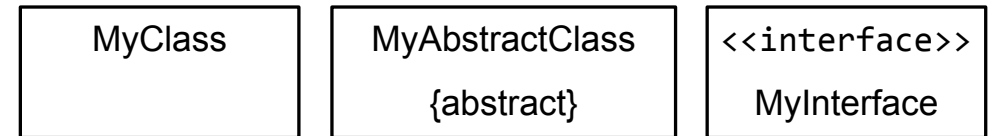
Visibility

- *private*
~ *package-private*
protected
+ *public*

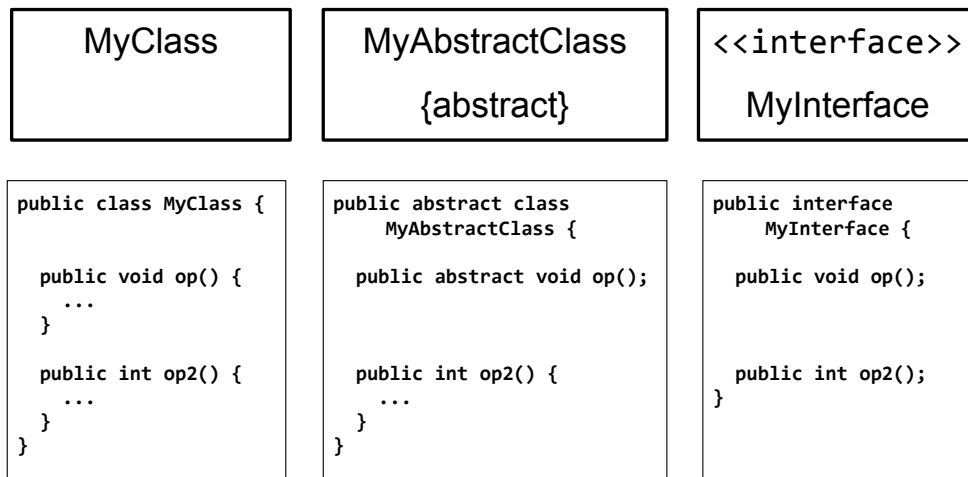
UML class diagram: basic notation



Classes, abstract classes, and interfaces

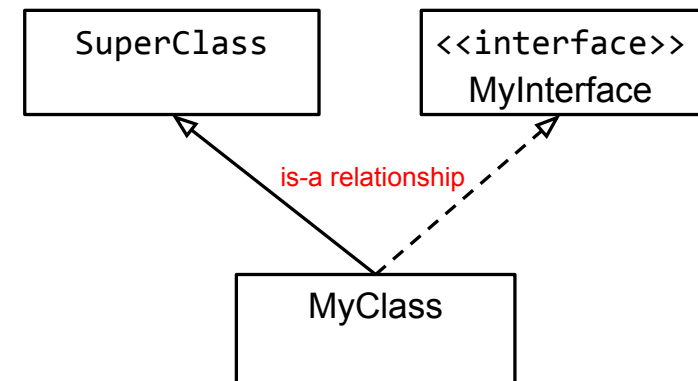


Classes, abstract classes, and interfaces



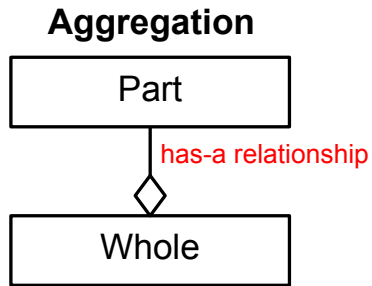
Level of detail in a given class or interface may vary and depends on context and purpose.

UML class diagram: Inheritance

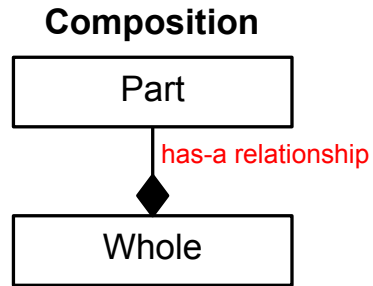


```
public class MyClass extends SuperClass implements MyInterface
```

UML class diagram: Aggregation and Composition

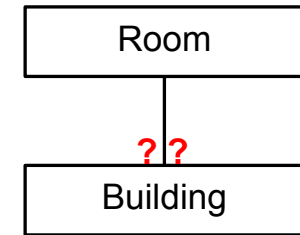
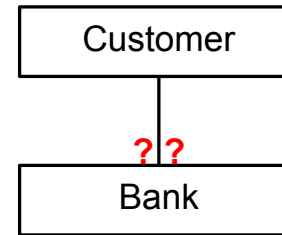


- Existence of Part does not depend on the existence of Whole.
- Whole does not own Part.
- Part might be shared with other instances of Whole.

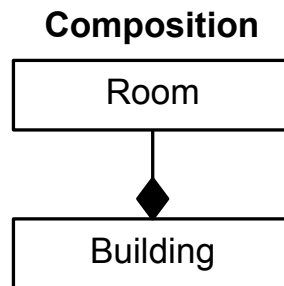
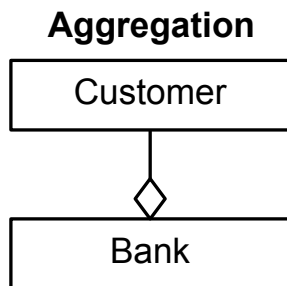


- Part cannot exist without Whole.
- The lifetime of Part is controlled by Whole.
- Whole is the single owner of Part.

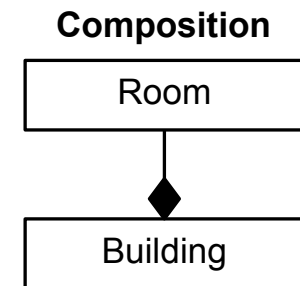
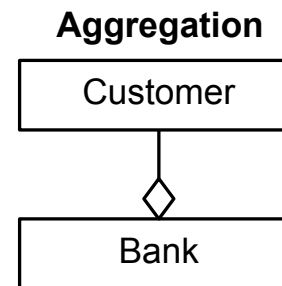
Aggregation or Composition?



Aggregation or Composition?

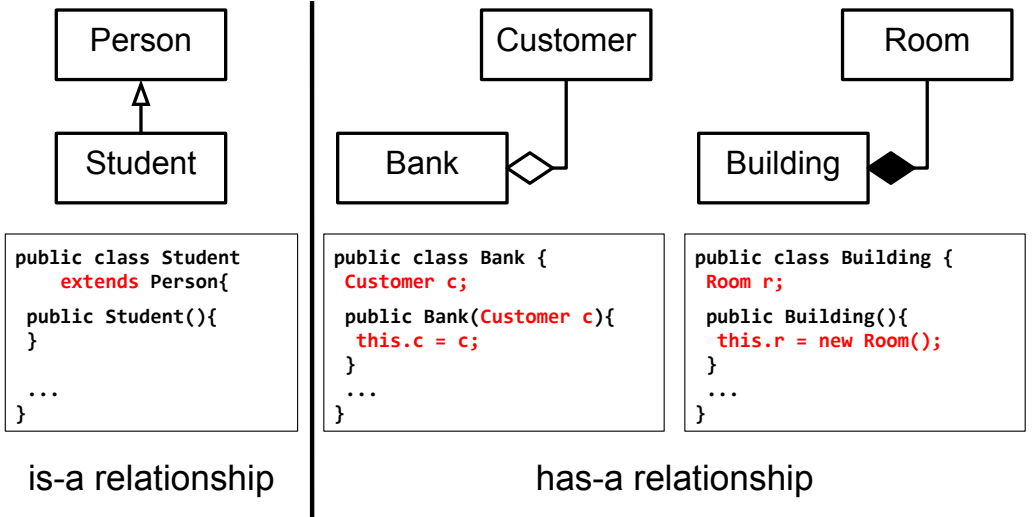


Aggregation or Composition?



What about class and students or body and body parts?

Inheritance vs. (Aggregation vs. Composition)



```
public class Student extends Person {  
    public Student() {  
    }  
    ...  
}
```

is-a relationship

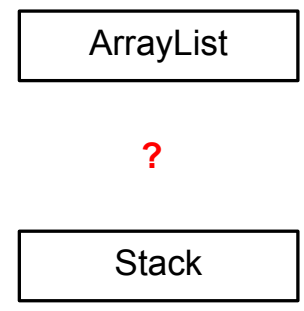
```
public class Bank {  
    Customer c;  
    public Bank(Customer c) {  
        this.c = c;  
    }  
    ...  
}
```

has-a relationship

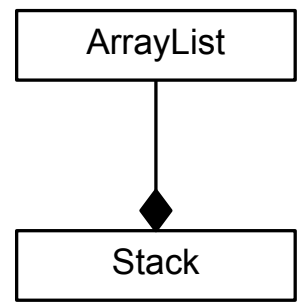
```
public class Building {  
    Room r;  
    public Building() {  
        this.r = new Room();  
    }  
    ...  
}
```

Simplified example: usually, a bank has more than one customer, and a building more than one room.

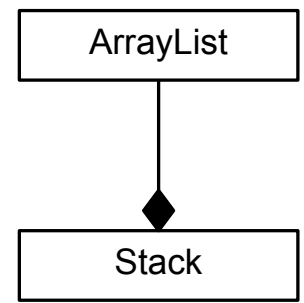
Inheritance or aggregation/composition?



Inheritance or aggregation/composition?

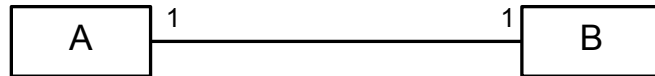


Inheritance or aggregation/composition?

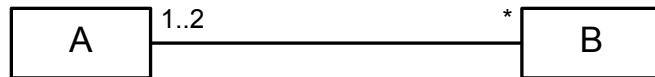


Aggregation may also be possible (e.g., if a Stack is used as a different representation of an existing ArrayList).

UML class diagram: multiplicity

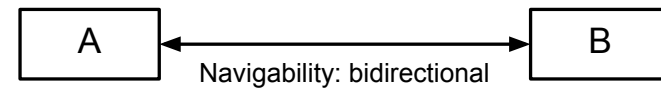
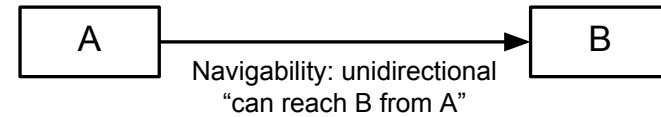
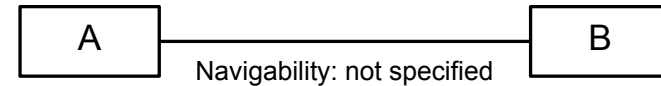


Each A is associated with exactly one B
Each B is associated with exactly one A

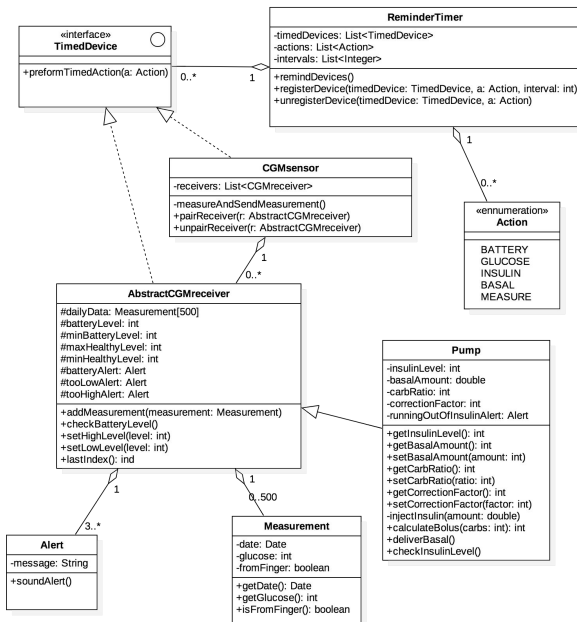


Each A is associated with any number of Bs
Each B is associated with exactly one or two As

UML class diagram: navigability



UML class diagram: example



Summary UML

- Unified notation for modeling OO systems.
- Allows different levels of abstraction.
- Suitable for design discussions and documentation.
- Generating code from diagrams is challenging.

In this class, we will use UML class diagrams mainly for visualization and discussion purposes.