EEET2482/COSC2082

SOFTWARE ENGINEERING DESIGN, ADVANCED PROGRAMMING TECHNIQUES

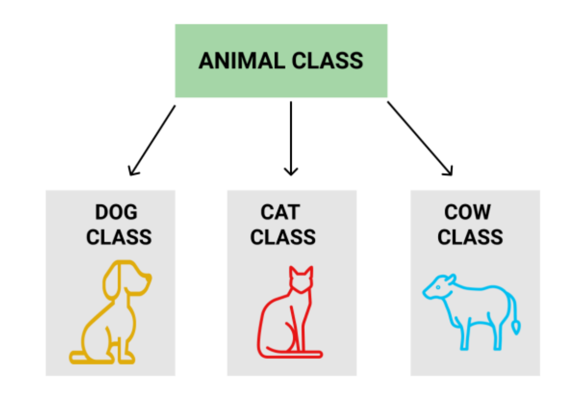
WEEK 8 – POLYMORPHISM & OBJECT ORIENTED DESIGN

LECTURER: LINH TRAN

RMIT UNIVERSITY

Inheritance Review

A class can inherit attributes and methods from another class



ANIMAL CLASS

DOG CLASS

CAT CLASS

COW CLASS

```
#include <iostream>
using namespace std;

// parent base class
class Animal {
public:
    void eat() {
        cout << "I can eat!" << endl;
    }
};

// child derived class
class Dog : public Animal {
public:
    void bark() {
        cout << "I can bark! Woof woof!!" << endl;
    }
};

int main() {

    Dog dog1;
    dog1.eat();  //dog1 has method from the parent class
    dog1.bark(); //dog1 has method from its own class

    return 0;
}
```

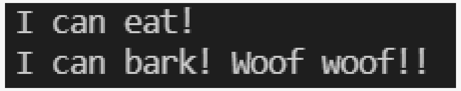
Syntax:

```
class child_class: access_mode parent_class
```

- **base class** (parent) - the class being inherited from
- **derived class** (child) - the class that inherits from another class

Note: *besides inherited properties, the child class can has its own attributes and methods.*

Output :



```
I can eat!
I can bark! Woof woof!!
```

Function Overriding

- A derived class may have a function with the same name as of the base class (overriding version of the base class).
- We can stil access the base class' version by using scope resolution operator (*preceeding by* ***classname::***)

Note: similarly, we can also ***override attributes*** *of the base class*

```cpp
#include <iostream>
using std::cout;

class Animal { // parent base class
public:
    void eat() {
        cout << "I can eat! \n";
    }
};

class Dog : public Animal { // child derived class
public:
    void eat() {
        cout << "The dog eat in his own way \n";
    }

    void bark() {
        cout << "I can bark! Woof woof!! \n";
    }
};

int main() {
    Dog dog1;
    dog1.Animal::eat();  //call method of the base class
    dog1.eat();  //call method of the derived class

    dog1.bark(); //call method of the derived class

    return 0;
}
```

Function Overriding and Virtual Functions

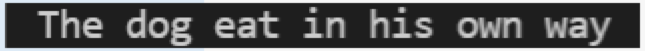
- A virtual function is a member function in the base class that we expect to overriden in derived classes.
- To **ensure that the derived class' version will be called**, we should declare the base class' version as **virtual function**. This especially applies to cases where ***a pointer/reference of base class points to an object of a derived class***

Note: *even declare the base class' version as virtual function, we can stil access it by using scope resolution operator as in previous slide.*

Example

```cpp
#include <iostream>
using std::cout;

class Animal { // parent base class
public:
    /*  IMPORTANT: remove virtual keyword will not
        call the child class' eat() version in activity() function
    */
    virtual void eat() {
        cout << "I can eat! \n";
    }
};

class Dog : public Animal { // child derived class
public:
    void eat() {
        cout << "The dog eat in his own way \n";
    }

    void bark() {
        cout << "I can bark! Woof woof!! \n";
    }
};

//A function that takes a pointer/reference of Animal class
void activity(Animal &anm) {
    anm.eat();
}

int main() {
    Dog dog1;
    activity(dog1); //call the function with Dog object

    return 0;
}
```

Output :



```
The dog eat in his own way
```

Polymorphism

Polymorphism means that it has **multiple forms.**

We can implement polymorphism in C++ using the following ways:

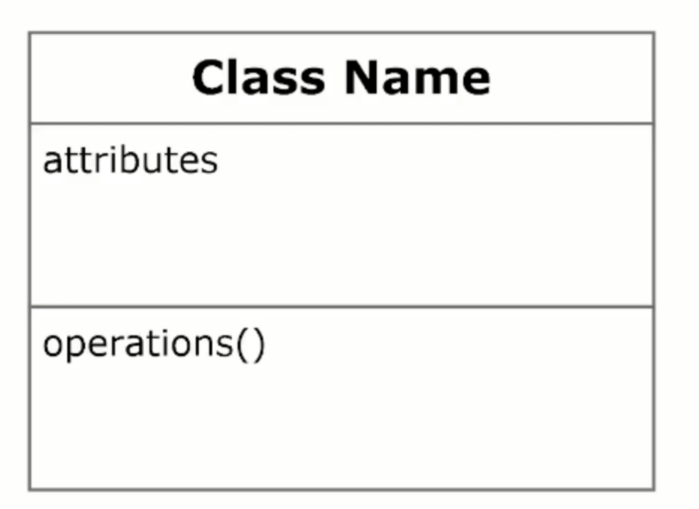
1. Function overloading
2. Operator overloading
3. Function overriding (with/ without declaration of **virtual functions**)

Object Oriented Design (OOD)

- **An approach for software design** using Object Oriented Programming (OOP) concepts.
- <u>Design</u>: specifying structure of how a software system will be developed, before writing the complete implementation.
- With OOD, the software design will be represented by <u>**class and object diagrams.**</u>
- <u>Guiding questions:</u>
  - ***What classes*** *will be implemented ?*
  - ***What attributes and methods*** *will each class have?*
  - *How will the classes* ***interact*** *with each other?*

Class Diagram (UML)

Notation:

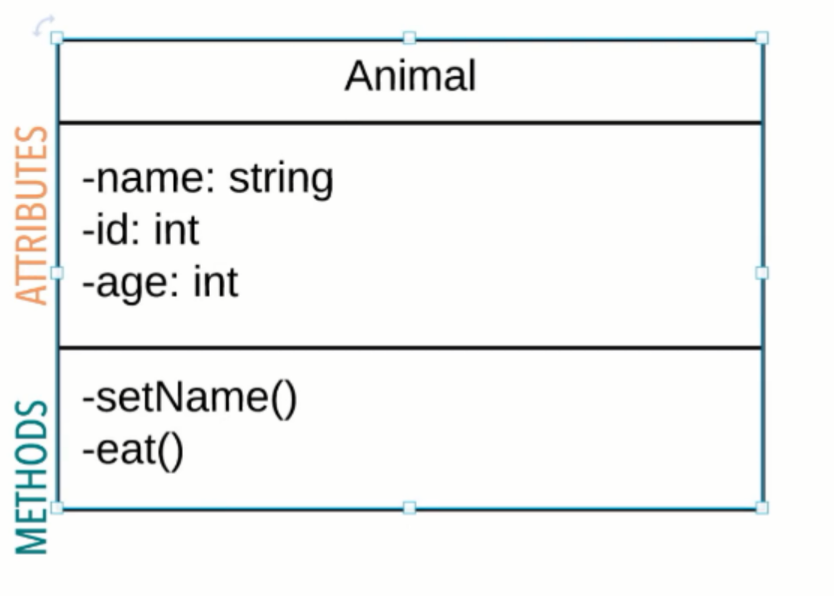


Class Name

attributes

operations()

Example:



Animal

-name: string
-id: int
-age: int

-setName()
-eat()

Access Specifier

- private

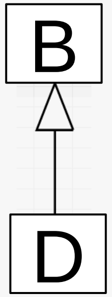
+ public

# protected

Note: ***complete reference for class diagram at this link***
https://www.uml-diagrams.org/class-reference.html

Relationships between Classes

- **Inheritance :**
  - "**is a**" relationship
  - A class inherit all attributes and methods of another class

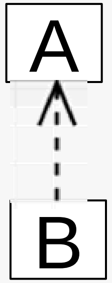


The derived class D "is a" type of the base class B

- **Dependency :**
  - "**depends**" relationship
  - Changes to definition of one class cause changes to the other (but not the vice versa).

Example: passing objects of class A as arguments to methods of class B

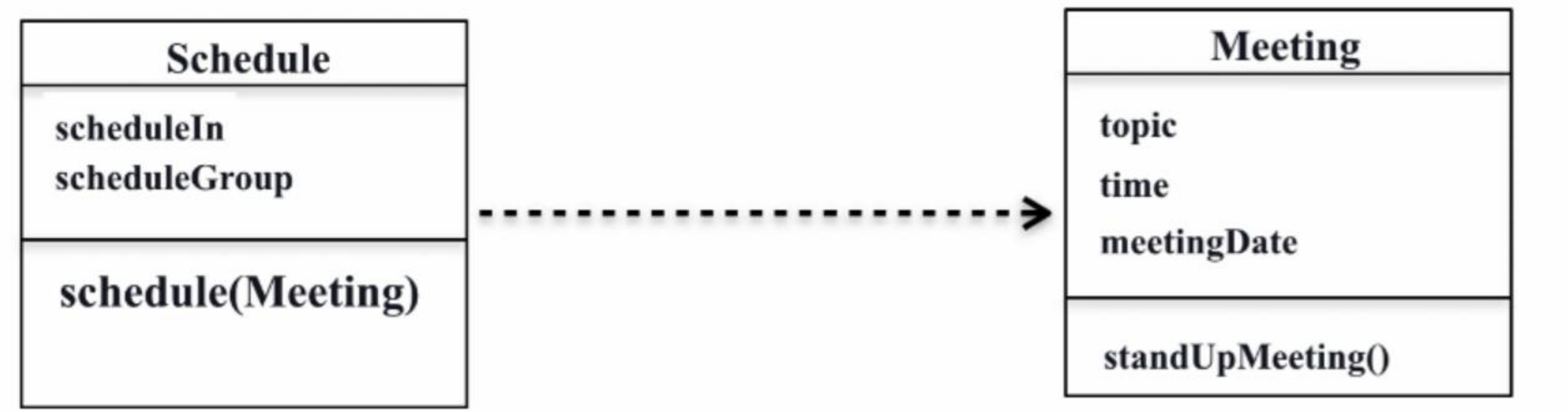
→ class B depends on class A



class B "depends" on class A

Example of Dependency

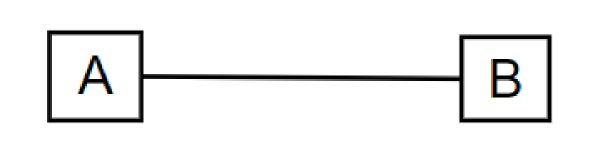
Schedule class depends on **Meeting** class
because it needs object of meeting class as a parameter for its methods.



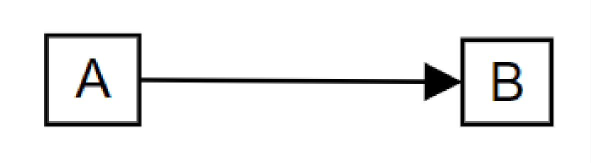
Relationships between Classes

- **Association:**
  - "**associate with**" relationship
  - A class associate/ interact with another class via their attributes/ methods

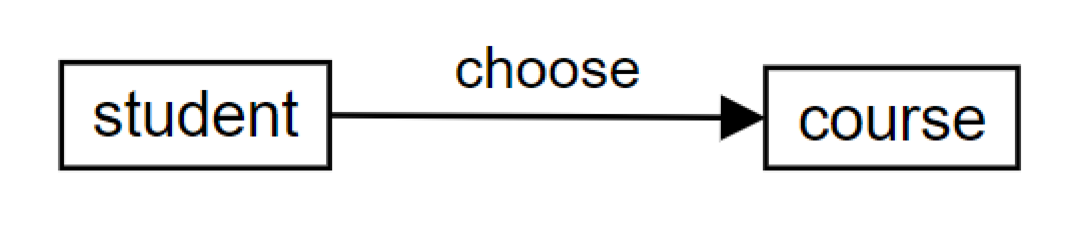
Bidirectional association

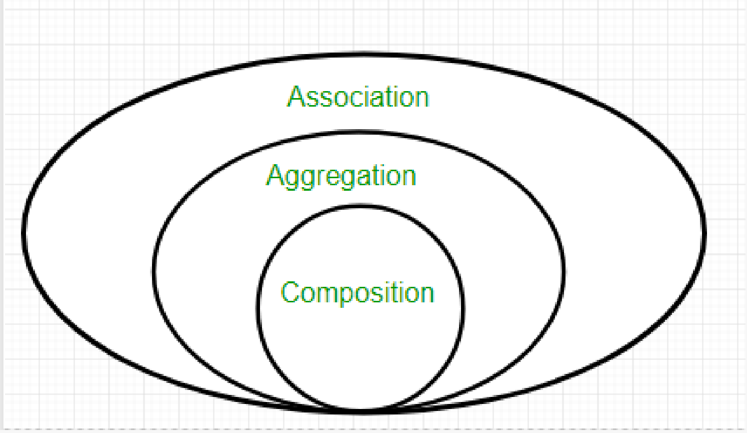


Directed association

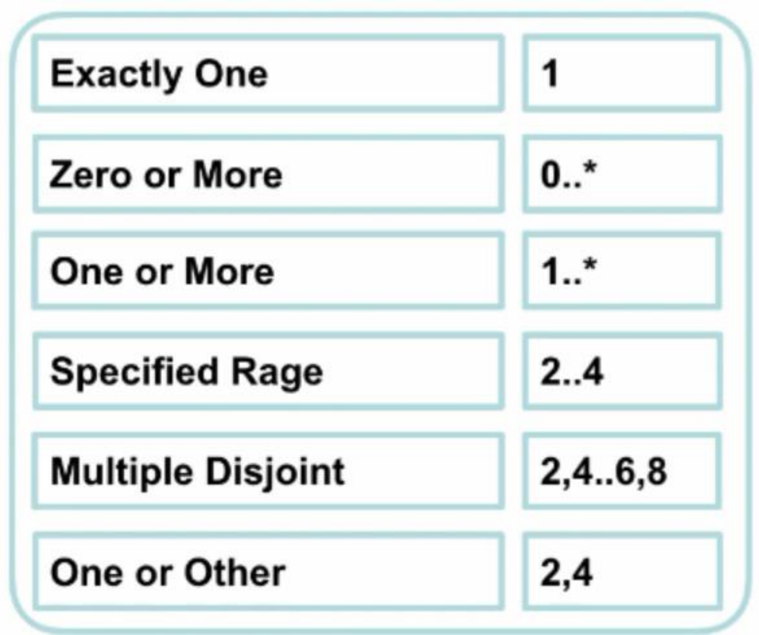


<u>Example</u>:

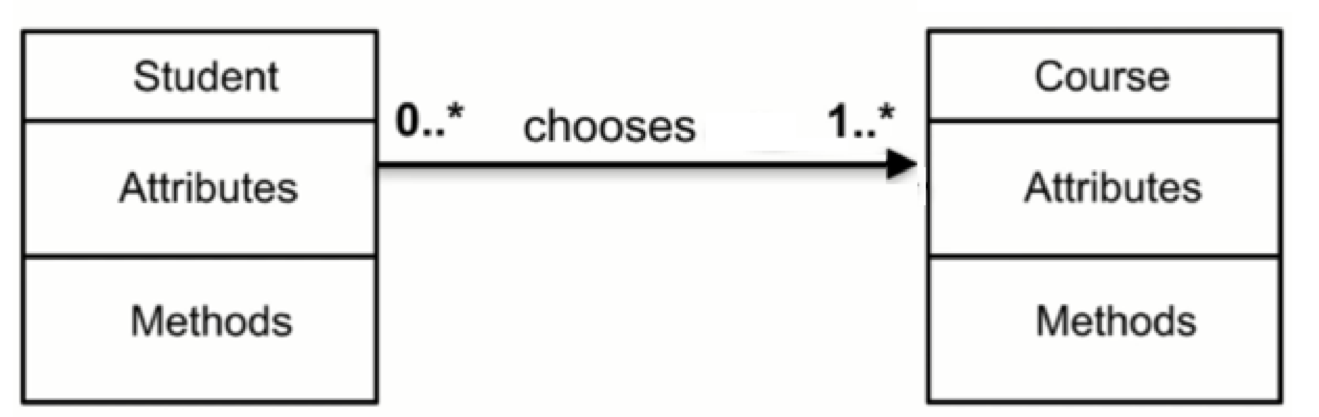




Multiplicity



Exactly One	1
Zero or More	0..*
One or More	1..*
Specified Rage	2..4
Multiple Disjoint	2,4..6,8
One or Other	2,4



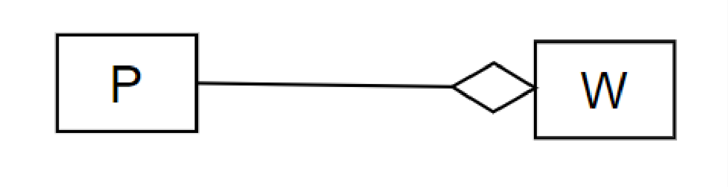
“Read at the other end”

- Each student can choose 1 or more courses
- Each course can be chosen by 0 or more students

Wherever the association relationship, there must be a multiplicity

Relationships between Classes

- **Aggregation: special form of Association**
  - **"has a" (belong to)** relationship
  - A class belongs to another class (***weak part-whole association***).
  - Destroy the whole may/may not destroy the part

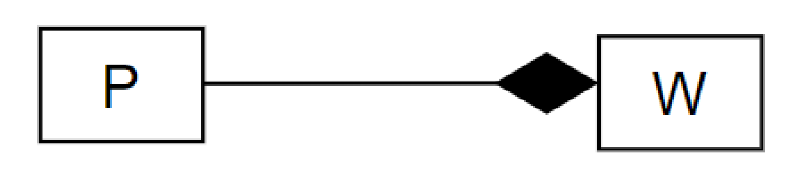


P is a part of w

Example: each teacher has a/ belong to a department.
However, delete the department will not eliminate its teachers

Relationships between Classes

- **Composition:** special form of Aggregation
  - **“Dependent part-of” relationship**
  - A class is a dependent part of another class (***strong part-whole association***).
  - Destroy the whole will always destroy the part



P is a dependent part of A

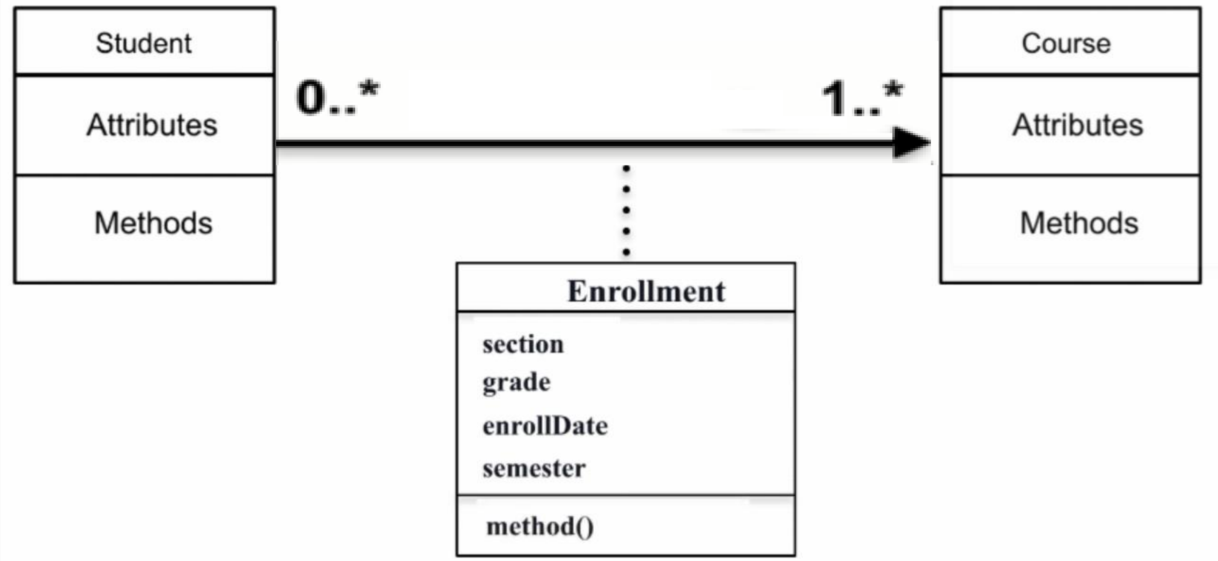
Example: each user may have multiple Ewallet accounts.
Delete a user will delete all of his/her accounts.

Association Class

- *A class attached to an association relationship* between two other classes. It has its own name, attributes operations, just like any other normal class.

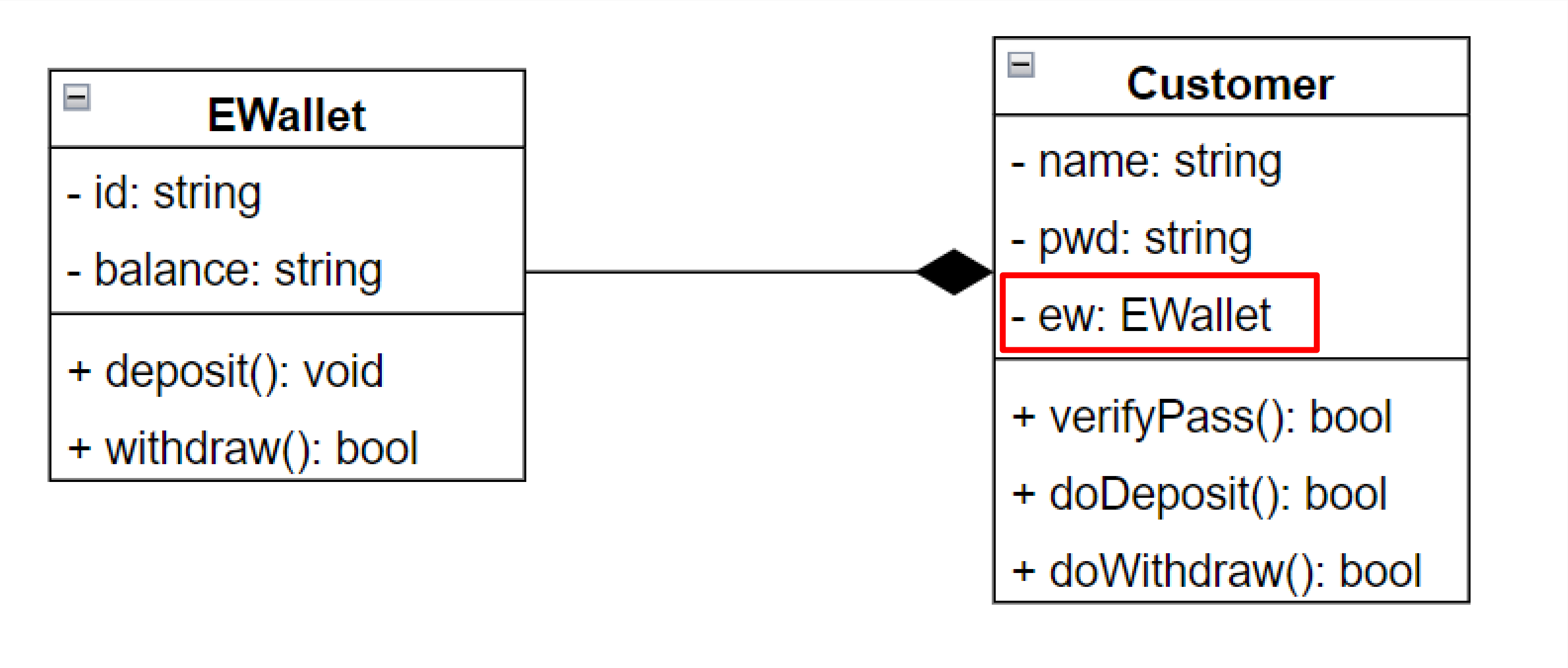
Example: **Students enroll** in **Courses**

We can make **Enrollment** process becomes an association class

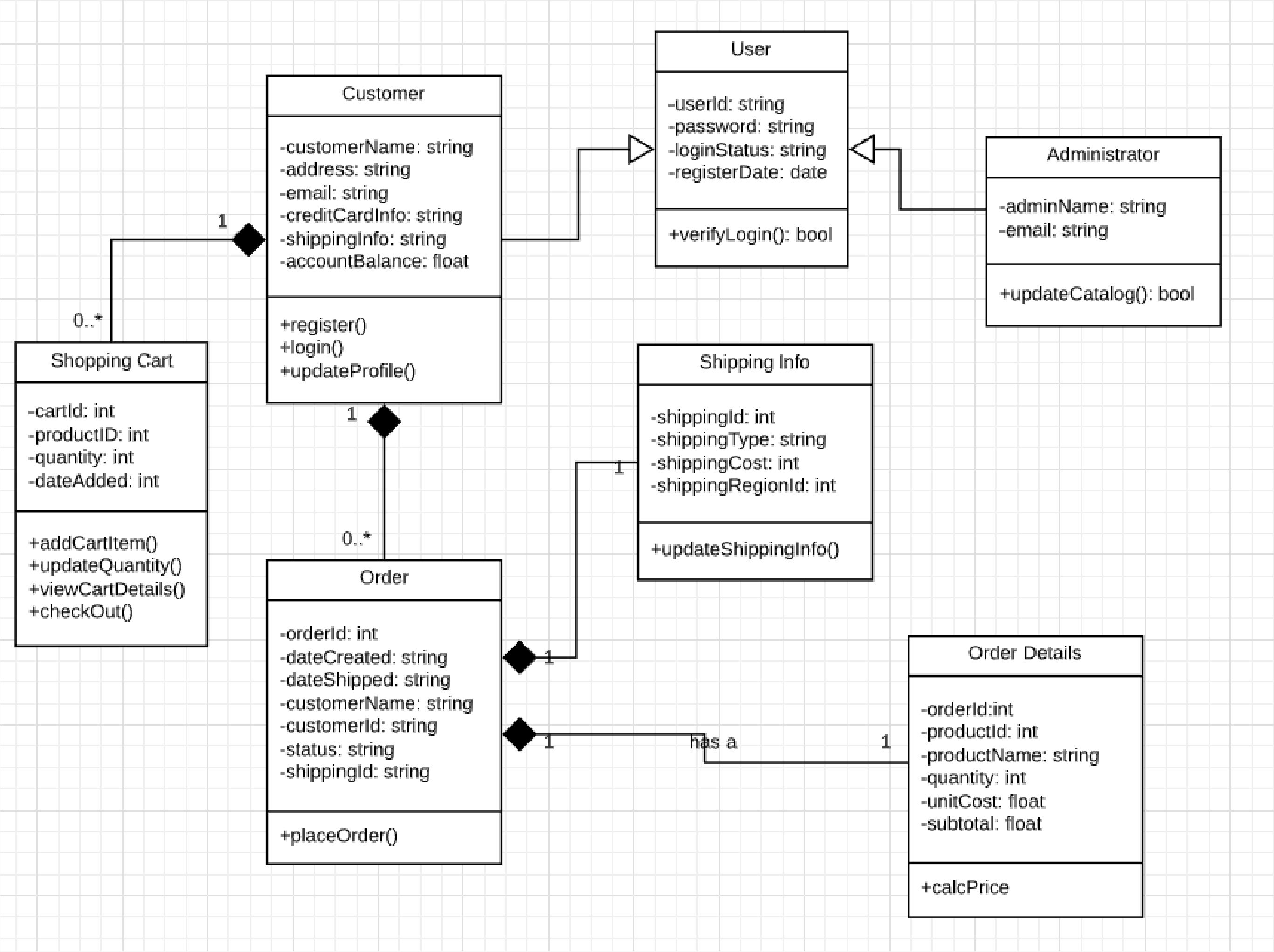


Class Diagram Example

- **Example from Lab Assessment 1:**



Example: Online Shopping



User

-userId: string
-password: string
-loginStatus: string
-registerDate: date

+verifyLogin(): bool

Customer

-customerName: string
-address: string
-email: string
-creditCardInfo: string
-shippingInfo: string
-accountBalance: float

+register()
+login()
+updateProfile()

Administrator

-adminName: string
-email: string

+updateCatalog(): bool

Shopping Cart

-cartId: int
-productID: int
-quantity: int
-dateAdded: int

+addCartItem()
+updateQuantity()
+viewCartDetails()
+checkOut()

Shipping Info

-shippingId: int
-shippingType: string
-shippingCost: int
-shippingRegionId: int

+updateShippingInfo()

Order

-orderId: int
-dateCreated: string
-dateShipped: string
-customerName: string
-customerId: string
-status: string
-shippingId: string

+placeOrder()

Order Details

-orderId:int
-productId: int
-productName: string
-quantity: int
-unitCost: float
-subtotal: float

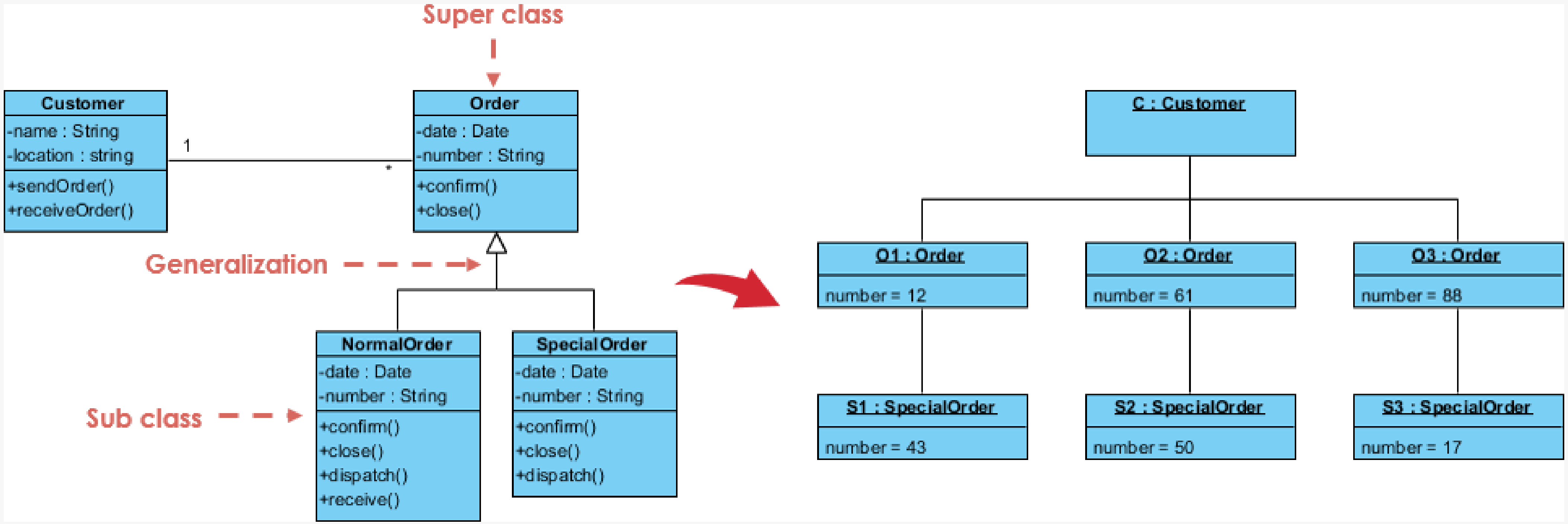
+calcPrice

Object Diagram (UML)

- **An instance of class diagram** in a particular moment in runtime that have its own state and data values.
- Shows a snapshot of the detailed state of a system at a point in time.

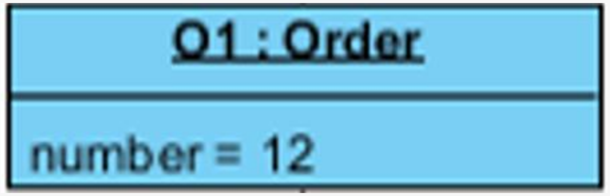
Class diagram

Object diagram

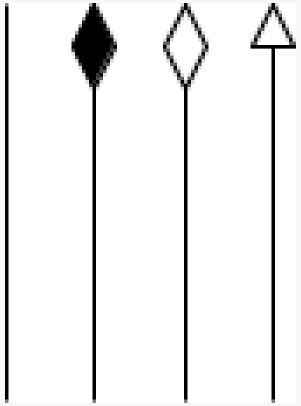


Object Diagram components

- Objects
  - **Object Name: Class name**
  - **Attribute = value** *(for each attribute)*



- Links:
  - Connecting lines of one object to another.
  - You can draw a link while using the lines utilized in class diagrams *(however, most of cases will be only association links)*



Useful Tips for Class & Object Diagram

- It is usually best to keep the diagram as **simple as possible**.
- Only add further information if it is really likely to be **useful in understanding the requirements** or **producing a suitable design.**
- Useful reference links:
  - https://www.lucidchart.com/pages/uml-class-diagram
  - https://ecs.syr.edu/faculty/fawcett/Handouts/cse687/lectures/StudyGuideClassRelationships.htm
- Tools to draw diagrams (*select one of below*):
  1. app.diagrams.net/
  2. www.lucidchart.com
  3. https://staruml.io/download