Lecture 11
OOP

Khola Naseem
khola.naseem@uet.edu.pk

Association:

- Association in C++ is a relationship between two classes where one class uses the functionalities provided by the other class.
- In other words, an association represents the connection or link between two classes. In an association, one class instance is connected to one or more instances of another class.
- Unlike aggregation, where the relationship is always unidirectional, in an association, the relationship may be unidirectional or bidirectional (where the two objects are aware of each other).
- The relationship between doctors and patients is a great example of an association. The doctor clearly has a relationship with his patients, but conceptually it's not a part/whole (object composition) relationship. A doctor can see many patients in a day, and a patient can see many doctors (perhaps they want a second opinion, or they are visiting different types of doctors). Neither of the object's lifespans are tied to the other.
- The Bank class has a direct association with the Account class because it uses objects of the Account class as parameters in its method.

Association:

- Example:

```
class Account {
public:
    Account(int id, double balance) : id(id), balance(balance) {}
    int getId() { return id; }
    void setbalance(double bal)
    {
        balance=bal;
    }
    double getBalance() { return balance; }
private:
    int id;
    double balance;
};
class Bank {
public:
    void transferMoney(Account* fromAccount, Account* toAccount, double amount) {
        double newbal= fromAccount->getBalance()-amount;
        fromAccount->setbalance(newbal);
        newbal= toAccount->getBalance()+amount;
        toAccount->setbalance(newbal);
    }
};
int main() {
    Account* account1= new Account(123, 1000.00);
    Account* account2= new Account(456, 1500.00);
    Bank bank;
    bank.transferMoney(account1, account2, 250.00);
    cout<<account1->getBalance()<<endl;
    cout<<account2->getBalance();
    return 0;
}
```

Output:

```
750
1750
```

Credit: Khola Naseem

Difference:

Aggregation	Association
It is represented by a “has a”+ “whole-part” relationship	It is represented by a “has a” relationship
Uni directional	Uni or bi directional
Ownership	No ownership

Credit: Khola Naseem

Composition vs aggregation

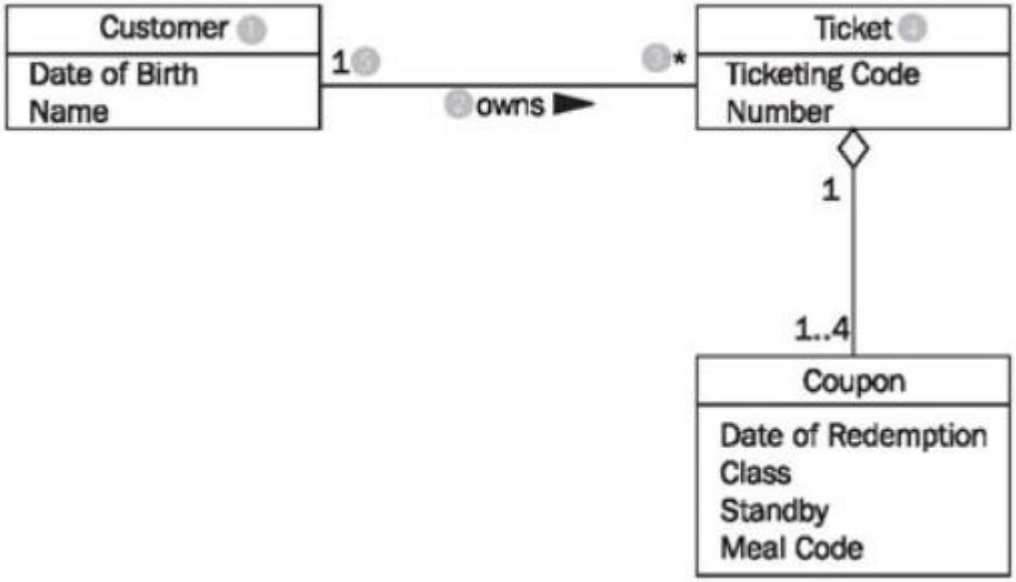
Composition: every car has an engine.

Aggregation: cars may have passengers, they come and go

Association

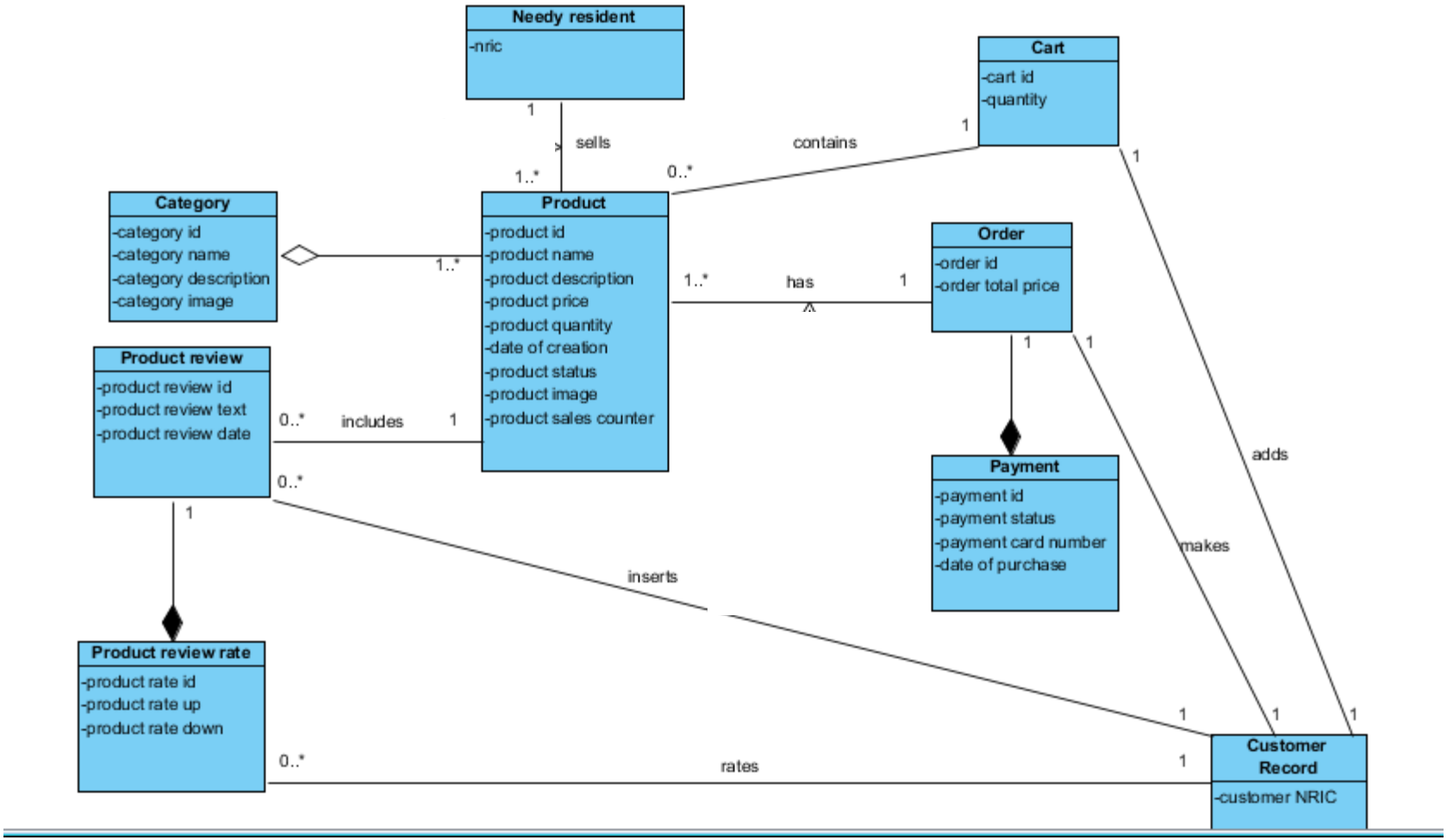
Credit: Khola Naseem

Association vs aggregation



Credit: Khola Naseem

Class diagram:



Credit: Khola Naseem

One to many

- Example:

```
class Address {
    private:
    string street;
    string city;
    string state;
    string zip;
public:
    Address(string street, string city, string state, string zip): street(street), city(city), state(state), zip(zip) {}
    void display()
    {
        cout<<street<<endl;
    }
};
class Person {
    private:
    string name;
    Address* address;
public:
    Person(std::string name) : name(name), address(NULL) {}
    void setAddress(Address* address) {
        this->address = address;
        address->display();
    }
};
int main() {
    Address* address = new Address("123 Main St.", "Anytown", "CA", "12345");
    Person person("John Doe");
    Person person2("ali");
    Person person3("umer");
    person.setAddress(address);
    person2.setAddress(address);
    person3.setAddress(address);
    return 0;
}
```

Credit: Khola Naseem

Many to Many:

- Example:

```
class Person {
    private:
    string name;

public:
    Person(string name) : name(name) {}
      string displayp()
    {

        return name;
    }

};
class Waiter{
    private:
    string name;

public:
    Waiter(string name) : name(name) {}
     string displayw()
    {

        return name;
    }

};
class Meal {
    private:
    Waiter w;
    Person p;
public:
    Meal(Waiter w,Person p): w(w), p(p) {}
    void display()
    {

        cout<<w.displayw()<<" will serve the food to "<<p.displayp()<<endl;
    }
};
```

```
int main() {
    Person p1("ali");
    Person person("John Doe");
    Person person2("ANNA");
    Person person3("umer");
    Waiter w1("Rose");
    Waiter w2("Ross");
    Waiter w3("Joe");
    Meal m1(w1,p1);
    Meal m2(w2,p1);
    Meal m3(w3, person);
    Meal m4(w3, person3);
    m1.display();
    m2.display();
    m3.display();
    m4.display();
    return 0;
}
```

```
Rose will serve the food to ali
Ross will serve the food to ali
Joe will serve the food to John Doe
Joe will serve the food to umer
```

Credit: Khola Naseem

Classes in multiple files:

- Example:
- Date.h file

```
#ifndef DATE_H
#define DATE_H

class Date
{
    private :
        int day;
        int month;
        int year;
    public :

        Date(const int d = 0, const int m  = 0, const int y = 0);
        void setDate(const int d, const int m, const int y);
        void print() const;
        bool equals(const Date&);
};
#endif
```

1/ 2 bool Date::equals (const Date &otherDate)

Classes in multiple files:

- Second cpp file:

```cpp
#include <iostream>
#include "partial.h"

using namespace std;


Date :: Date(const int d, const int m, const int y)
  : day(d), month (m), year(y)
{}

void Date :: setDate(const int d, const int m, const int y)
{
    day = d;
    month = m;
    year = y;
}

void Date :: print() const
{
    cout << day << "/" << month<< "/" << year << "\n";

}

bool Date :: equals(const Date &otherDate)
{
    if(day == otherDate.day
        && month == otherDate.month
        && year == otherDate.year)
        return true;
    else
        return false;
}
```

```cpp
int main()
{
    Date t1(10, 50, 59);
    t1.print();
    Date t2;
    t2.print();
    //t2.setDate(6, 39, 9);
    t2.setDate(10, 50, 59);
    t2.print();

    if(t1.equals(t2))
        cout << "Two Dates are equal\n";
    else
        cout << "Two Dates are not equal\n";

    return 0;
}
```

Credit: Khola Naseem

Classes in multiple files:

- Friend.h:

```
#ifndef COUNTER_H
#define COUNTER_H
class counter
{
    int val1;

    public:
        counter();
        friend counter operator ++(counter &c);
        friend counter operator ++(counter &c,int unused);
        void display();

};

#endif
```

Classes in multiple files:

- .cpp:

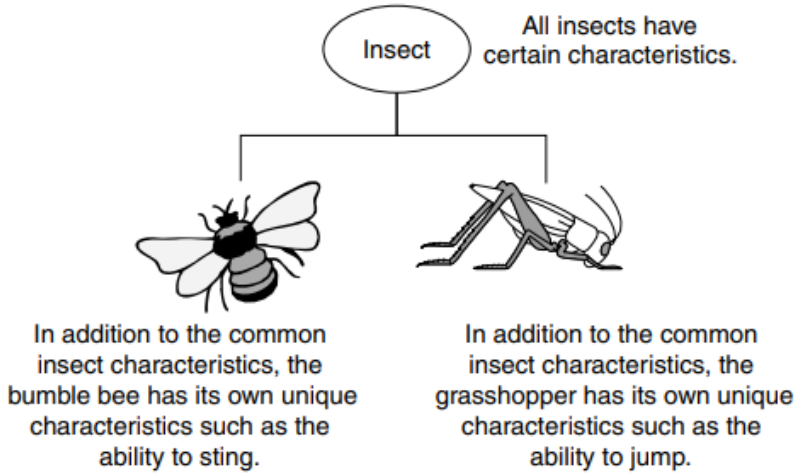
```cpp
#include <iostream>
#include "Friend.h"
/* run this program using the console pauser or add your own getch, system("pause") or input loop */
using namespace std;
counter::counter()
{
    val1=5;

}
void counter::display()
{
    cout<<"Answer is: "<<val1;
}
//prefix
counter operator ++(counter &c)
        {

            c.val1++;
            return c;

        }
        //postfix
counter operator ++(counter &c, int unused)
        {
          counter c9=c;
            c.val1++;
            return c9;

        }
int main(int argc, char** argv) {
    counter c,c2;
    c2=++c;
    counter c3=c++;
    c2.display();
    cout<<"value of c3   " ;
    c3.display();
}
```

```
E:\UET\Spring 23\OOP\Class\Friend partial.exe
Answer is: 6value of c3   Answer is: 6
--------------------------------
Process exited after 0.1401 seconds with return value 0
Press any key to continue . . .
```

Credit: Khola Naseem

Inheritance

- Inheritance?
- Inheritance allows a new class to be based on an existing class. The new class inherits all the member variables and functions (except the constructors and destructor) of the class it is based on.
- Generalization and Specialization

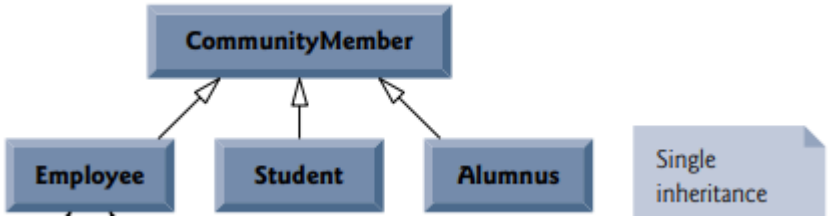


Credit: Khola Naseem

Inheritance

- Inheritance Examples:

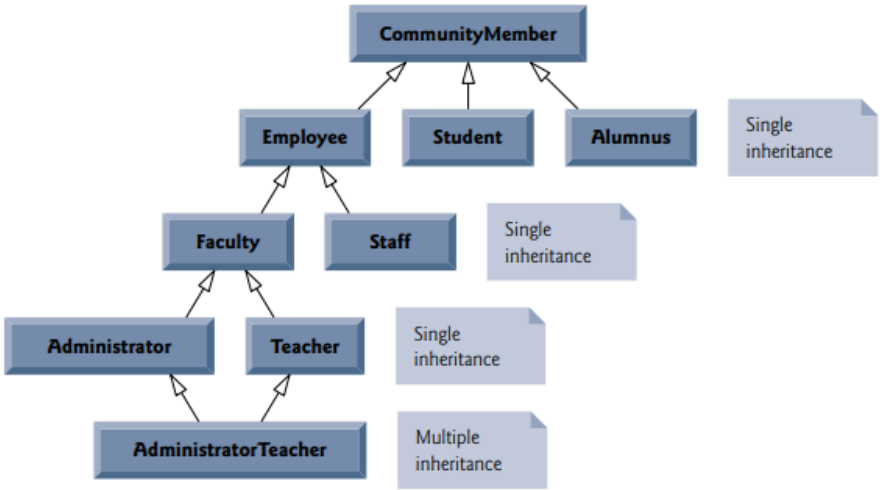
Base class	Derived classes
Student	GraduateStudent, UndergraduateStudent
Shape	Circle, Triangle, Rectangle, Sphere, Cube
Loan	CarLoan, HomeImprovementLoan, MortgageLoan
Employee	Faculty, Staff
Account	CheckingAccount, SavingsAccount



Credit: Khola Naseem

Inheritance

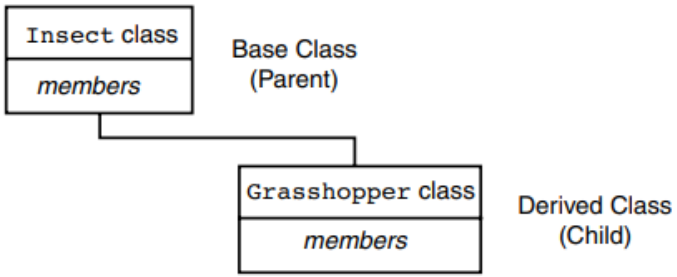
- Inheritance



- With single inheritance, a class is derived from one base class.
- With multiple inheritance, a derived class inherits simultaneously from two or more
- direct base class
- Indirect base class

Inheritance

- Inheritance and the "**Is a**" Relationship
- When one object is a specialized version of another object, there is an "is a" relationship between them. For example, a grasshopper is an insect
- Examples:
  - A poodle is a dog.
  - A car is a vehicle.
  - A tree is a plant.
  - A rectangle is a shape.
  - A football player is an athlete
- Inheritance involves a base class and a derived class. The base class is the general class and the derived class is the specialized class.



Credit: Khola Naseem

Inheritance

- Inheritance:
  - class derived-class: access-specifier base-class
- Where access-specifier is one of **public, protected,** or **private**, and base-class is the name of a previously defined class. If the access-specifier is not used, then it is private by default.
- Example:

```
class Shape
{

};
class TwoDimensionalShape : public Shape
{

}
```

Inheritance

- Example:

```
class Shape
{
    public:
    void displayshape() {
        cout << "i am function of shape class" << endl;
    }

};
class TwoDimensionalShape : public Shape
{
    public:
     void displayTshape() {
        cout << "i am function of TwoDimensionalShape class" << endl;
    }

};


int main(int argc, char** argv) {
    TwoDimensionalShape t;
    t.displayshape();
    t.displayTshape();
}
```

- Output:

```
i am function of shape class
i am function of TwoDimensionalShape class
```

Credit: Khola Naseem

The access modifier protected is especially relevant when it comes to C++ inheritance.

Inheritance

- **Access Modes of Inheritance in C++:**
- This is an example of public inheritance, the most commonly used form. The other types are private inheritance and protected inheritance
- With all forms of inheritance, private members of a base class are not accessible directly from that class's derived classes, but these private base-class members are still inherited
- With **public inheritance**, all other base-class members retain their original member access when they become members of the derived class
  - e.g., public members of the base class become public members of the derived class, and, protected members of the base class become protected members of the derived class
- With **private inheritance ,**all the members of the base class become private members in the derived class.

```
class TwoDimensionalShape : private Shape
```

- With **protected inheritance** The public members of the base class become protected members in the derived class.

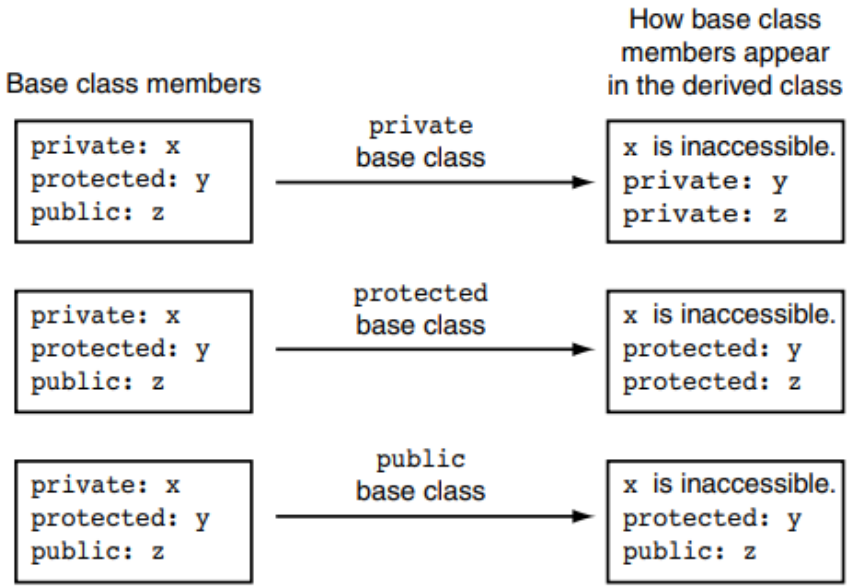
```
class TwoDimensionalShape : protected Shape
```

Credit: Khola Naseem

The access modifier protected is especially relevant when it comes to C++ inheritance.

Inheritance

- **Access Modes of Inheritance in C++:**
- With **public inheritance**, all other base-class members retain their original member access when they become members of the derived class
- With **private inheritance ,**all the members of the base class become private members in the derived class.
- With **protected inheritance** The public members of the base class become protected members in the derived class.



Credit: Khola Naseem

The access modifier protected is especially relevant when it comes to C++ inheritance.

Inheritance

- **Protected access modifier :**
- The access modifier protected is especially relevant to inheritance.
- Like private members, protected members are inaccessible outside of the class. However, they can be accessed by derived classes and friend classes/functions.
- We need protected members if we want to hide the data of a class, but still want that data to be inherited by its derived classes.

```
class Shape
{
    private:
       string color;
    protected:
      int Dim;
    public:
    void displayshape() {
        cout << "i am function of shape class" << endl;
    }
    void setcolor(string col)
    {
        color=col;
    }
    string getcolor()
    {
        return color;
    }

};
```

The access modifier protected is especially relevant when it comes to C++ inheritance.

Inheritance

- **Protected access modifier :**

```
class Shape
{
    private:
       string color;
    protected:
      int Dim;
    public:
    void displayshape() {
        cout << "i am function of shape class" << endl;
    }
    void setcolor(string col)
    {
        color=col;
    }
    string getcolor()
    {
        return color;
    }

};
class TwoDimensionalShape : public Shape

{
    public:
     void displayTshape() {
        cout << "i am function of TwoDimensionalShape class" << endl;
    }
    void setDim(int a)
    {
        Dim=a;
    }
    void displayData(string c) {
        cout << "Color of the shape is : " <<c <<" and Dimension is :  " <<Dim<< endl;
    }

};
```

Credit: Khola Naseem

Reference material

For Practice Questions, refer to these books

- C++ Programming From Problem Analysis To Program Design, 5th Edition, D.S.Malik. Chapter 12.
- C++ How to Program, Deitel & Deitel, 5th Edition, Prentice Hall.
- Object Oriented Programming in C++ by Robert Lafore.
- Object Oriented Software Construction, Bertrand Meyer's
- Object-Oriented Analysis and Design with applications, Grady Booch et al, 3Rd Edition, Pearson, 2007
- Web