C++

PROGRAMMING LANGUAGE

L10-INHERITANCE

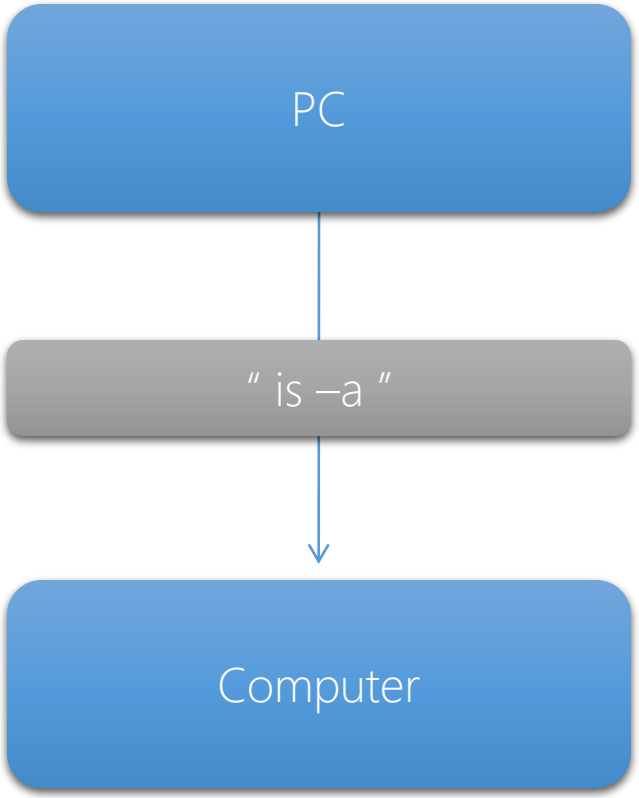
Mohammad Shaker
mohammadshaker.com
@ZGTRShaker
2010, 11, 12, 13, 14

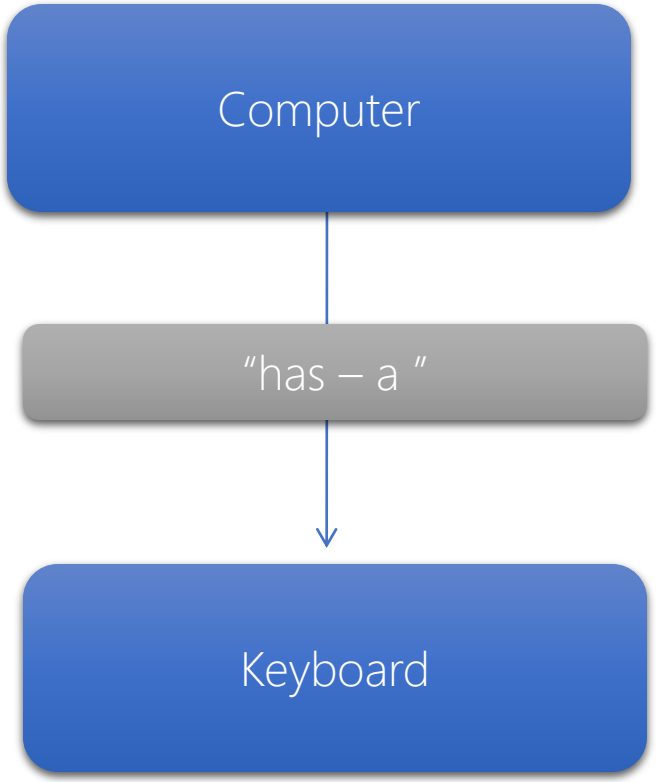
Inheritance

Introduction to Inheritance

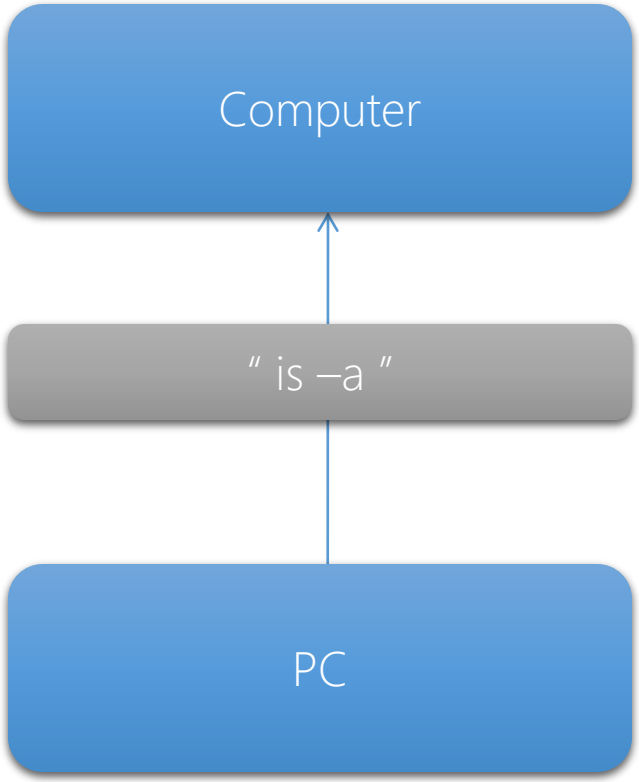
- Two common types:
  - "has – a "
    - Composition (as we have learned)
      - Object containing one or more objects of other classes as members.
        - » A car has a steering wheel
  - "Is – a "
    - Inheritance
      - Derived class object treated as base class objects
        - » A car is a vehicle
        - » Vehicle properties / behaviors also car properties / behaviors

Inheritance



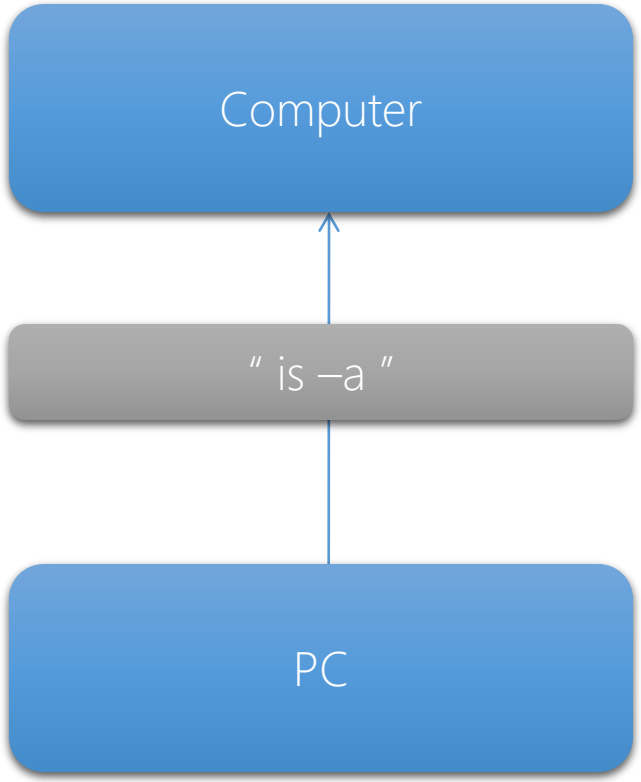


Inheritance, " is –a " Relationship

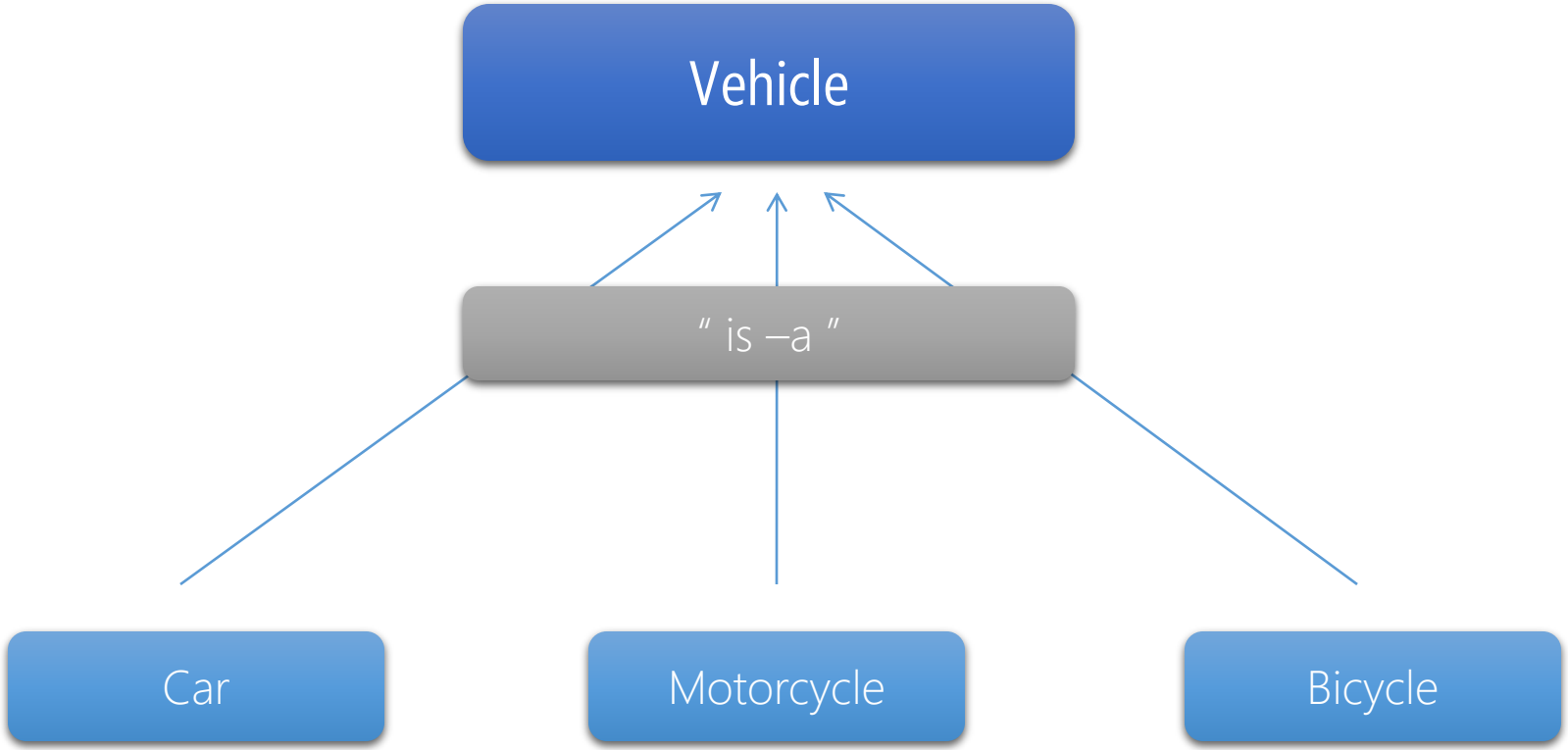


The link between classes in an "is – a " relationship comes from the fact that the subclasses share all the attributes and behaviors found in the supper class, plus more!

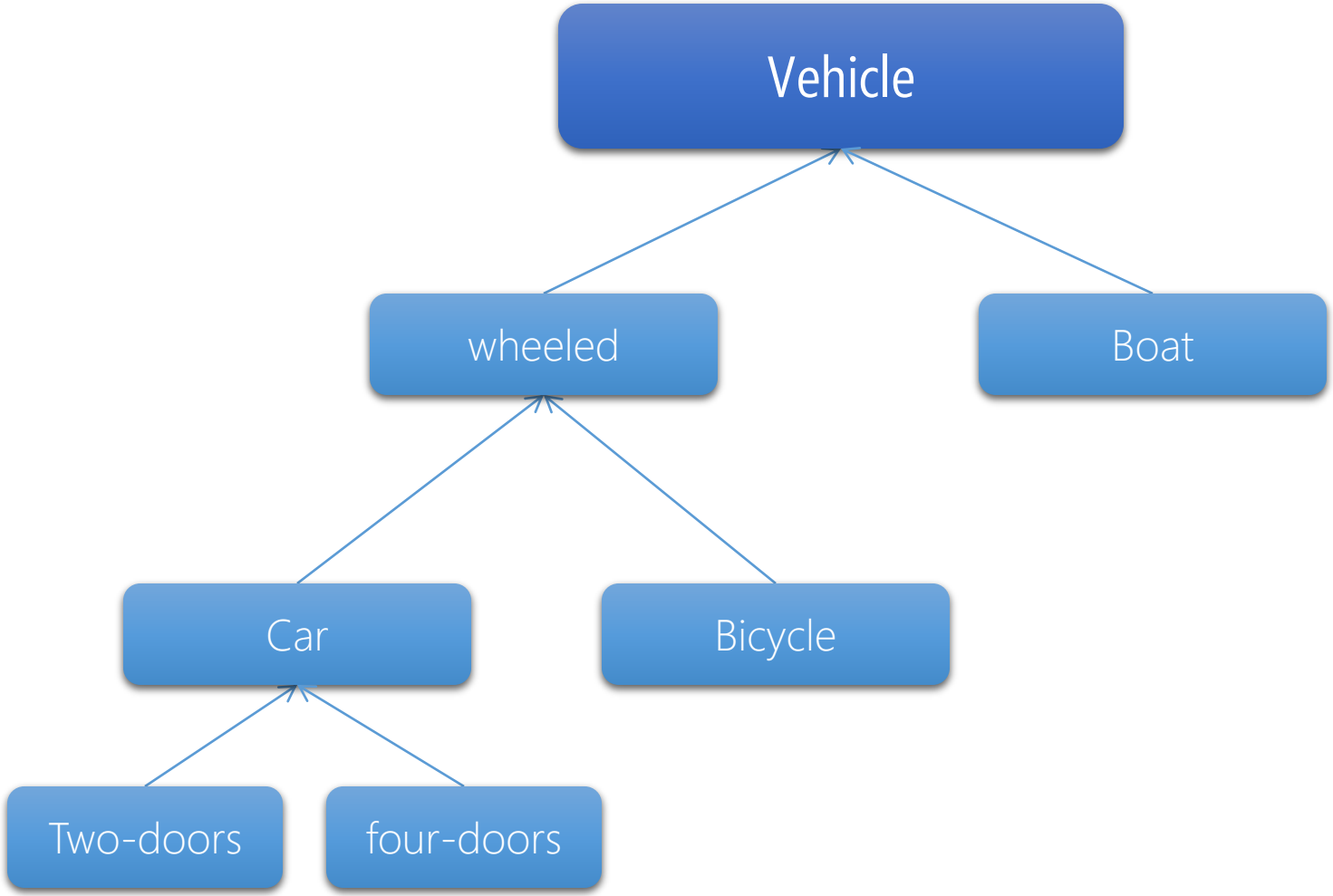
Inheritance, " is –a " Relationship



Inheritance, " is –a " Relationship



Inheritance, " is –a " Relationship



Inheritance

- Let's have the following relationship
  - Student and instructor (teacher)
    - Fields:
      - Name, address, phone
    - Now, let's build a class called person

```
class Person
  {
    private:
      int idNum;
      char lastName[20];
      char firstName[15];
    public:
      void setFields(int num, char last[], char first[]);
      void outputData();
  };
void Person::setFields(int num, char last[], char first[])
  {
    idNum = num;
    strcpy(lastName, last);
    strcpy(firstName,first);
  }
void Person::outputData()
  {
    cout<<"ID #"<<idNum<<" Name: "<<firstName<<" "<<
     lastName<<endl;
  }
```

Inheritance

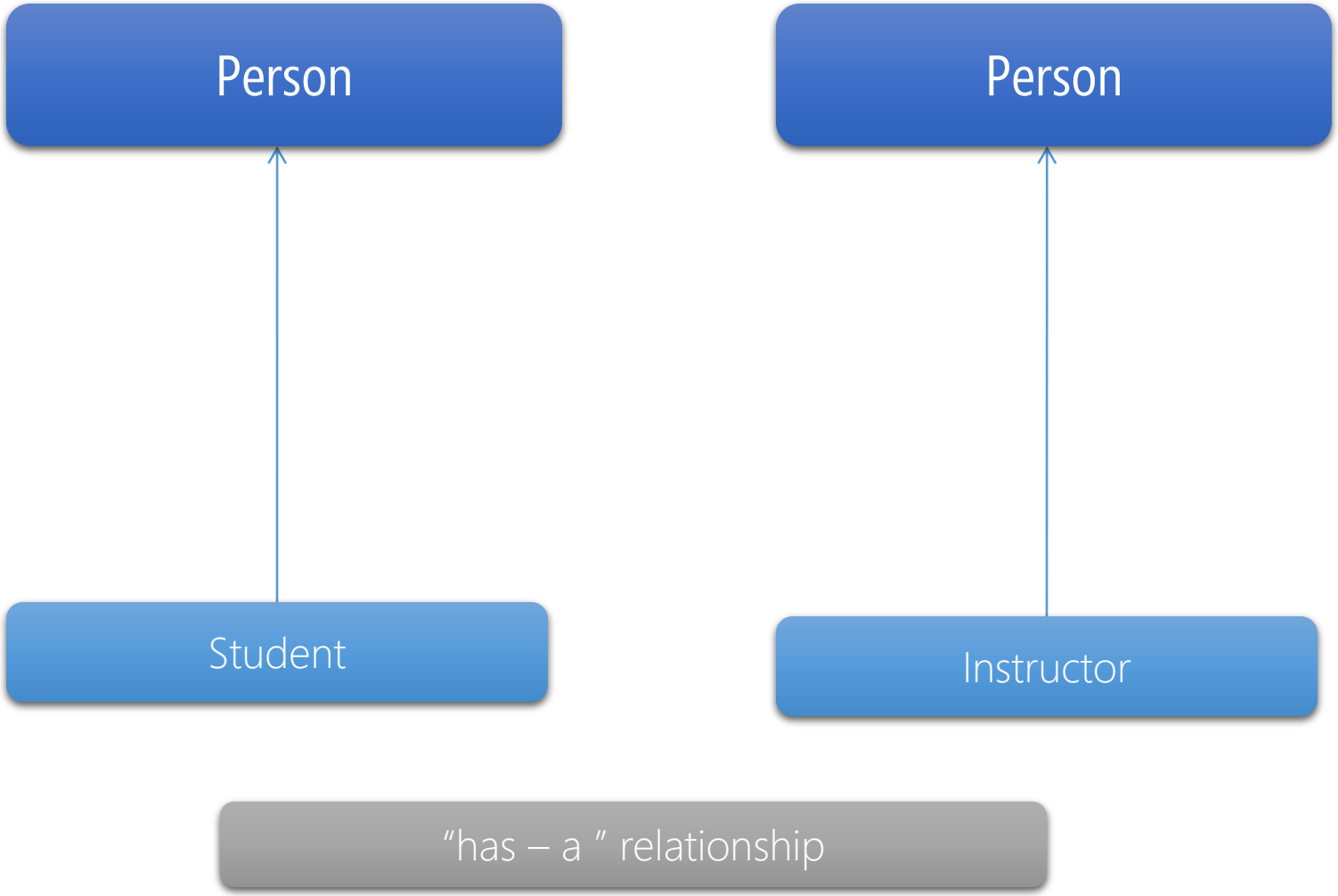
- Now, we think that we can create the following class and carry on

```
class Student
{
private:
          person UniversityStudent;
          int StuID;
          int StuRate;
};
```

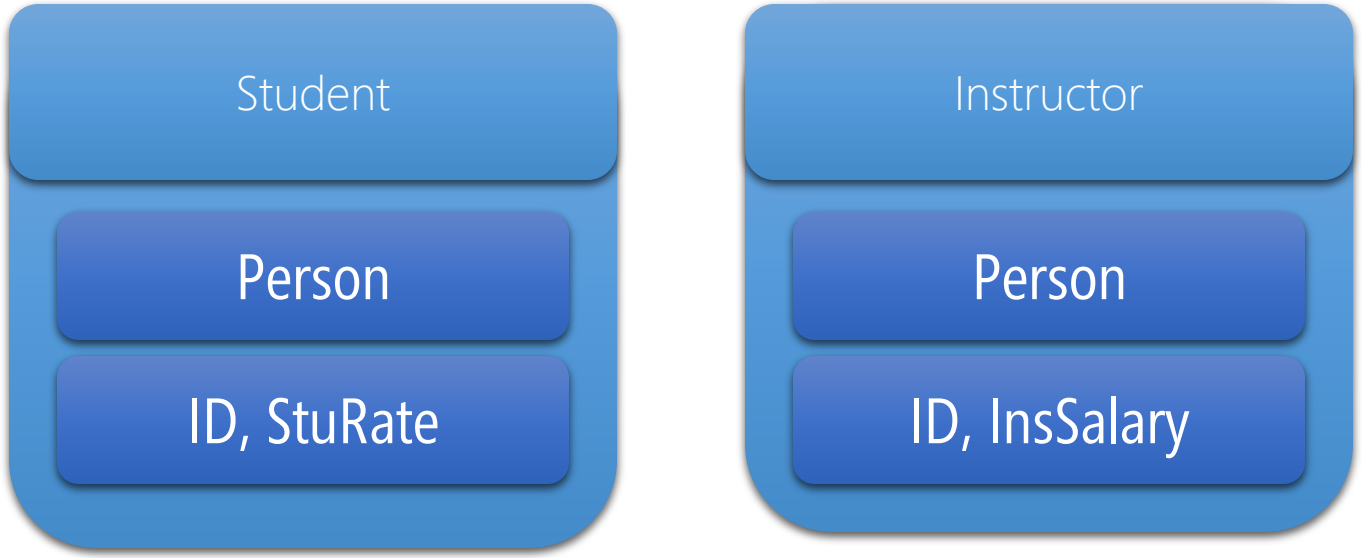
```
class Instructor
{
private:
          person UniversityInstructor;
          int InsID;
          int InsSalary;
};
```

- But we have the idea?
  - Inheritance!

Advantages of Inheritance

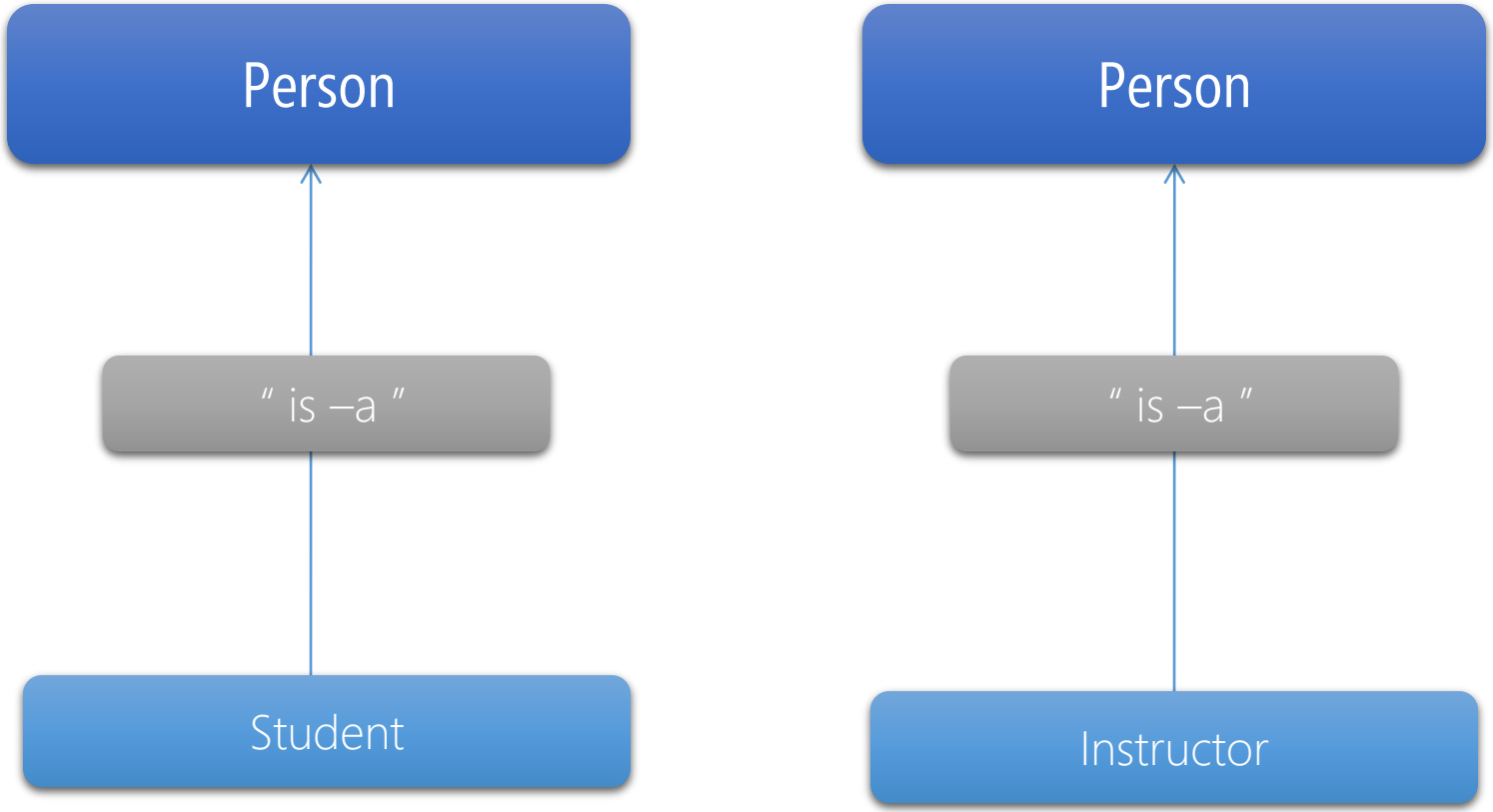


Advantages of Inheritance

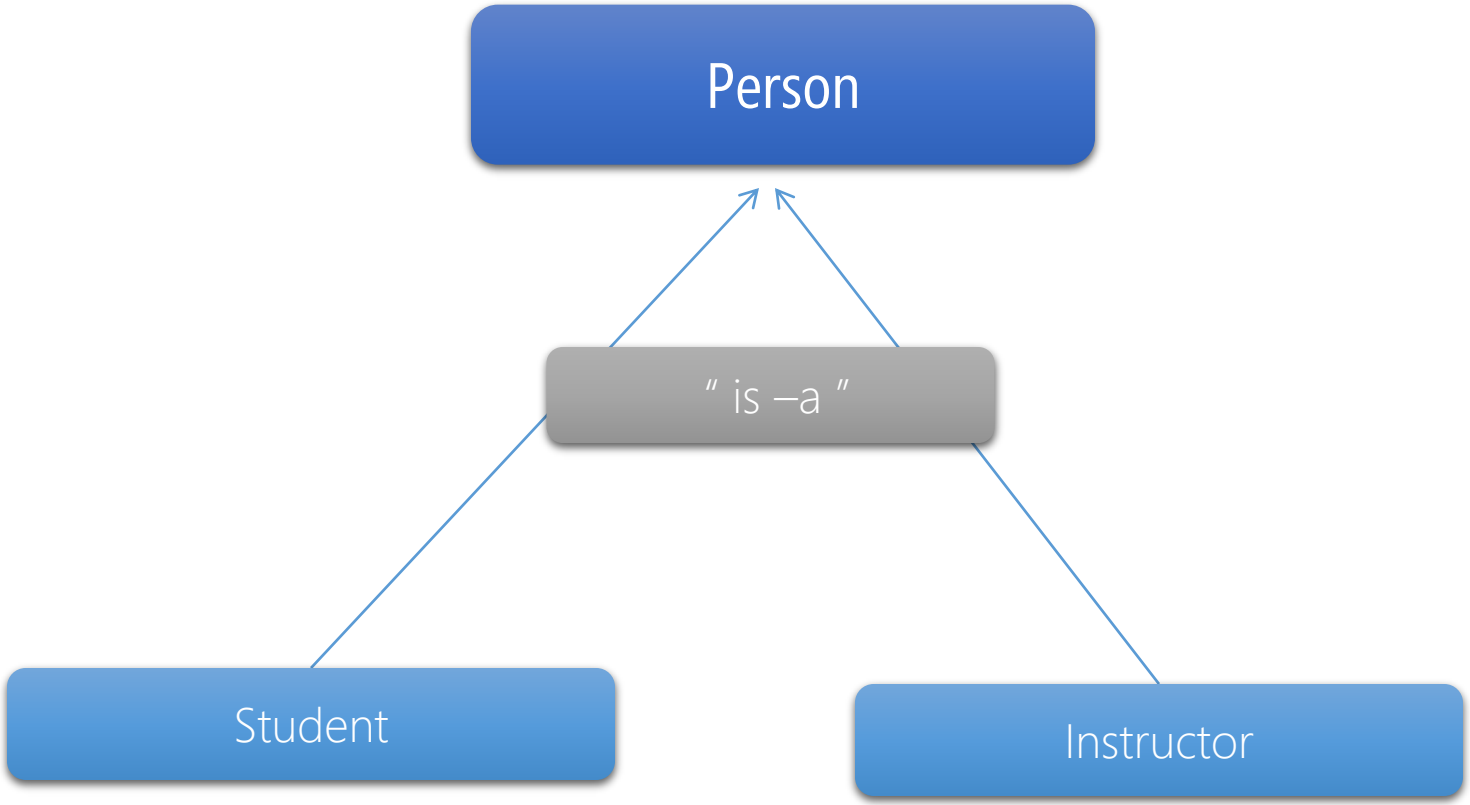


"has – a " relationship

Advantages of Inheritance



Advantages of Inheritance



Advantages of Inheritance

- What we do now, is that
  - We create a “person” class then
    - Inherit from it!
    - What to inherit from\to?
      - We Inherit class person to class
        - » Student
        - » Instructor
      - And then, we carry on and add for each class of them their unique features! (ID, StuRate, StuSalary etc)

Advantages of Inheritance

- Advantages of inheritance:
  - Saving time
  - After saving time building the first part of the solution
    - We can concentrate on building more complex materials up ahead
  - Can extend or revise (edit) the parent class without corrupting the existing parent class features
  - Absorb existing parent class's data and behaviors
    - All public, protected member variables in the main class are accessible from the derived class
    - All behaviors inherited too
  - Enhance with new capabilities
    - Customizing
    - Additional behaviors

Thinking Inheritance

- C++ \ Usability
- Creating a “Derived class” from “Base Class” *
- Base Class usually represent:
  - Larger set of objects than derived class
  - Characteristics that are shared between all of the derived classes
- Derived classes usually represent more specialized group of objects
- Sometimes
  - Derived class inherit data members and data functions it does \ doesn’t need or should \ shouldn’t not have

---

*Base Class: also called Parent class, Super class, Ancestor

*Derived Class: also called Child Class, Sub Class, Descendant

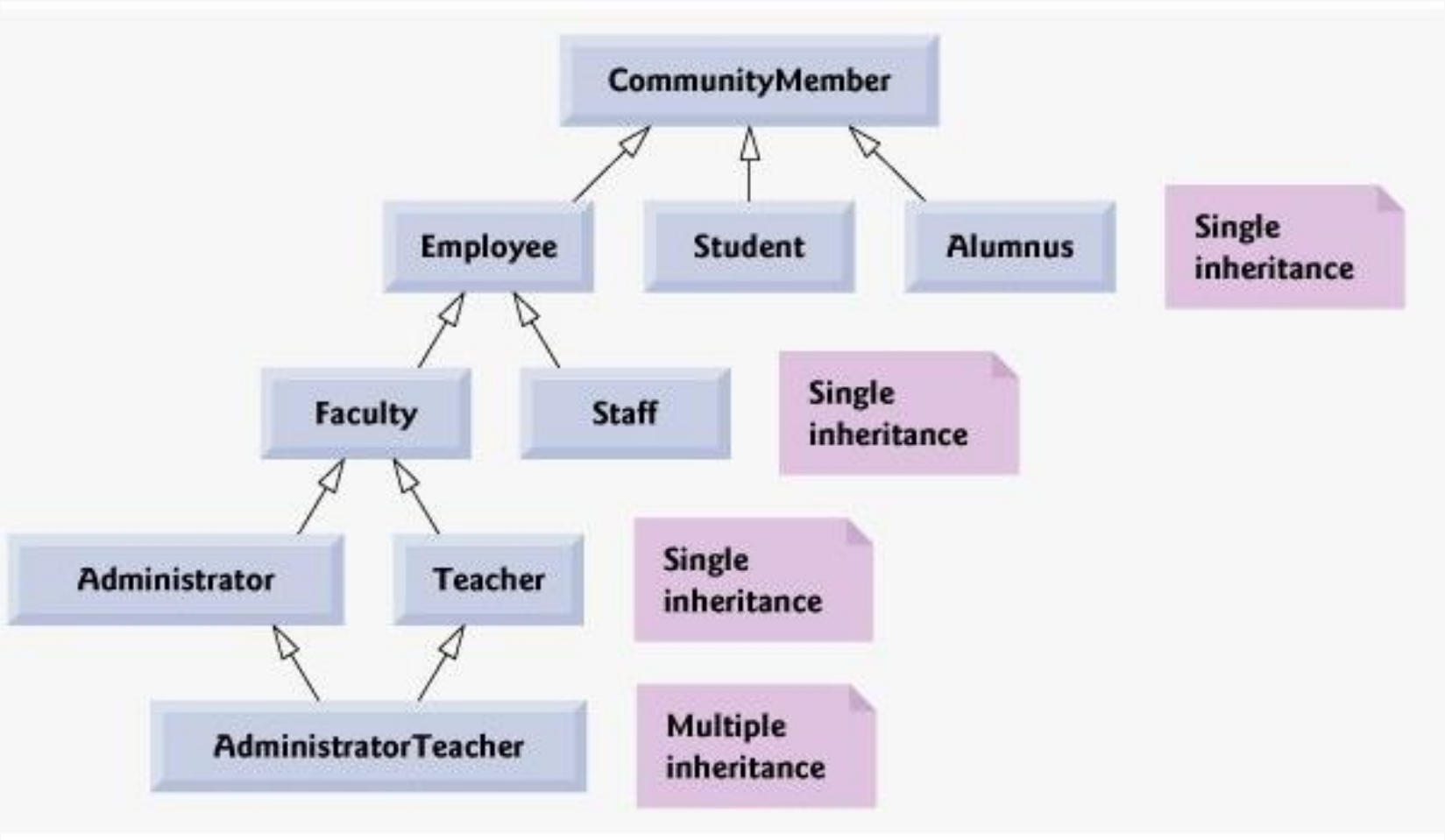
Class Hierarchy types

- Direct \ Indirect:
  - Direct class access
    - Inherited explicitly (one level up)
  - Indirect class access
    - (one or two levels up hierarchy)

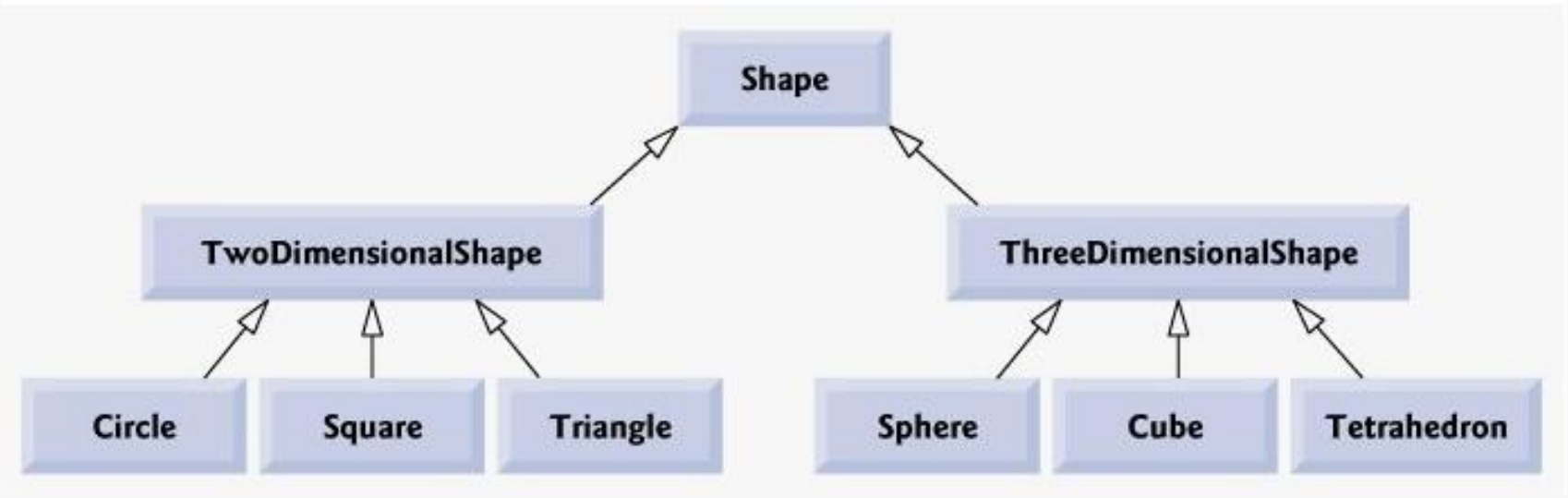
- Single \ Multiple:
  - Single Inheritance
    - From one base class
  - Multiple Inheritance
    - From multiple base class (only in C++, not in C++.NET, C#, Java, etc.)

Inheritance Examples

Inheritance



Inheritance



Inheritance

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

Types of Inheritance

Types of Inheritance

Base-class member-access specifier	Type of inheritance		
	`public` inheritance	`protected` inheritance	`private` inheritance
`public`	`public` in derived class. Can be accessed directly by member functions, `friend` functions and nonmember functions.	`protected` in derived class. Can be accessed directly by member functions and `friend` functions.	`private` in derived class. Can be accessed directly by member functions and `friend` functions.
`protected`	`protected` in derived class. Can be accessed directly by member functions and `friend` functions.	`protected` in derived class. Can be accessed directly by member functions and `friend` functions.	`private` in derived class. Can be accessed directly by member functions and `friend` functions.
`private`	Hidden in derived class. Can be accessed by member functions and `friend` functions through `public` or `protected` member functions of the base class.	Hidden in derived class. Can be accessed by member functions and `friend` functions through `public` or `protected` member functions of the base class.	Hidden in derived class. Can be accessed by member functions and `friend` functions through `public` or `protected` member functions of the base class.

Creating a Derived Class

Creating a Derived Class

- Notes:
  - When inheriting with public specifier
    - friend Functions are not inherited!
  - When inheriting with private specifier
    - Non of the public methods of the base class are accessible to the users of the derived class

Creating a Derived Class

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


class BaseClass
{

};

class DerivedClass: private BaseClass
{

};
```

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


class BaseClass
{

};

class DerivedClass: protected BaseClass
{

};
```

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


class BaseClass
{

};

class DerivedClass: public BaseClass
{

};
```

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


class BaseClass
{

};

class DerivedClass:BaseClass
{

};
```

When ommiting private, public or protected then the default is private

Creating a Derived Class

```
class Vehicle
{
public:
            Vehicle ()
            {           weight = 0; }
            Vehicle (int wt)
            {           weight = wt;            };
            int GetWeight ()
            {           return weight; }
            void SetWeight (int x )
            {           weight = x;};
private:
            int weight;
};
class Car: public Vehicle
{
public:
            Car()
            {           speed = 0; }
            Car(int wt, int sp)
            {           SetWeight(wt);          speed = sp;             }
            int GetSpeed ()
            {           return speed; }
            void SetSpeed (int x )
            {           speed = x; }
private:
            int speed;
};
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;



void main()
{
            Car MyCar(2,3);
            cout << MyCar.GetSpeed() << endl;
            cout << MyCar.GetWeight() << endl;

            MyCar.SetWeight(5);     // member function in base class
            MyCar.SetSpeed(10);

            cout << "Printing" << endl;
            cout << MyCar.GetSpeed() << endl;
            cout << MyCar.GetWeight() << endl;

}
```

```
3
2
Printing
10
5
Press any key to continue
```


Live Demo
with Lord of the Rings
(Humans and Goblins inherit from Creatures)

Don’t forget

`friend` functions are not inherited!

Examples

Creating a Derived Class

```
class Vehicle
{
public:
            Vehicle ()
            {           weight = 0; }
            Vehicle (int wt)
            {           weight = wt;            };
            int GetWeight ()
            {           return weight; }
            void SetWeight (int x )
            {           weight = x;};
private:
            int weight;
};

class Car: public Vehicle
{
public:
            friend void foo(Car &C);
            Car()
            {           speed = 0; }
            Car(int wt, int sp)
            {           SetWeight(wt);          speed = sp;             }
            int GetSpeed ()
            {           return speed; }
            void SetSpeed (int x )
            {           speed = x; }
private:
            int speed;
};
void foo(Car &C)
{           C.speed = 9; }
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;
void main()
{
            Car MyCar(2,3);
            cout << MyCar.GetSpeed() << endl;
            cout << MyCar.GetWeight() << endl;
            cout << "After 1st change" << endl;
            MyCar.SetWeight(5);
            MyCar.SetSpeed(10);
            cout << MyCar.GetSpeed() << endl;
            cout << MyCar.GetWeight() << endl;
            foo(MyCar);
            cout << "After 2nd change" << endl;
            cout << MyCar.GetSpeed() << endl;
            cout << MyCar.GetWeight() << endl;
}
```

```
3
2
After 1st change
10
5
After 2nd change
9
5
Press any key to continue
```

Creating a Derived Class

```
class Vehicle
{
public:
            Vehicle ()
            {           weight = 0; }
            Vehicle (int wt)
            {           weight = wt;            };
            int GetWeight ()
            {           return weight; }
            void SetWeight (int x )
            {           weight = x;};
private:
            int weight;
};

class Car: public Vehicle
{
public:
            friend void foo(Car C);
            Car()
            {           speed = 0; }
            Car(int wt, int sp)
            {           SetWeight(wt);          speed = sp;             }
            int GetSpeed ()
            {           return speed; }
            void SetSpeed (int x )
            {           speed = x; }
private:
            int speed;
};
void foo(Car C)
{           C.speed = 9; }
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;
void main()
{
            Car MyCar(2,3);
            cout << MyCar.GetSpeed() << endl;
            cout << MyCar.GetWeight() << endl;
            cout << "After 1st change" << endl;
            MyCar.SetWeight(5);
            MyCar.SetSpeed(10);
            cout << MyCar.GetSpeed() << endl;
            cout << MyCar.GetWeight() << endl;
            foo(MyCar);
            cout << "After 2nd change" << endl;
            cout << MyCar.GetSpeed() << endl;
            cout << MyCar.GetWeight() << endl;
}
```

```
3
2
After 1st change
10
5
After 2nd change
10
5
Press any key to continue
```

Creating a Derived Class

```
class Vehicle
{
public:
            Vehicle ()
            {           weight = 0; }
            Vehicle (int wt)
            {           weight = wt;            };
            int GetWeight ()
            {           return weight; }
            void SetWeight (int x )
            {           weight = x;};
private:
            int weight;
};

class Car: public Vehicle
{
public:
            friend void foo(Car C);
            Car()
            {           speed = 0; }
            Car(int wt, int sp)
            {           SetWeight(wt);          speed = sp;             }
            int GetSpeed ()
            {           return speed; }
            void SetSpeed (int x )
            {           speed = x; }
private:
            int speed;
};
void foo(Car C)
{           C.weight= 9; }
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;
void main()
{
            Car MyCar(2,3);
            cout << MyCar.GetSpeed() << endl;
            cout << MyCar.GetWeight() << endl;
            cout << "After 1st change" << endl;
            MyCar.SetWeight(5);
            MyCar.SetSpeed(10);
            cout << MyCar.GetSpeed() << endl;
            cout << MyCar.GetWeight() << endl;
            foo(MyCar);
            cout << "After 2nd change" << endl;
            cout << MyCar.GetSpeed() << endl;
            cout << MyCar.GetWeight() << endl;
}
```

```
Compiler error, function foo must not access private data
members of Vehicle
```

Creating a Derived Class

```
class Vehicle
{
public:
            Vehicle ()  {           weight = 0; }
            Vehicle (int wt)        {           weight = wt;
            };
            int GetWeight ()        {           return weight; }
            void SetWeight (int x ) {           weight = x;};
            int tempWeight;
private:
            int weight;
};

class Car: public Vehicle
{
public:
            friend void foo(Car &C);
            Car()       {speed = 0; }
            Car(int wt, int sp){    SetWeight(wt);          speed = sp;
            }
            int GetSpeed ()         {           return speed; }
            void SetSpeed (int x )  {           speed = x; }
private:
            int speed;
};
void foo(Car &C)
{           C.speed = 9;}
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;

void main()
{
            Car MyCar(2,3);
            cout << MyCar.GetSpeed() << endl;
            cout << MyCar.GetWeight() << endl;
            MyCar.tempWeight = 5;
            cout << MyCar.GetSpeed() << endl;
            cout << MyCar.GetWeight() << endl;
}
```

```
3
2
3
2
Press any key to continue
```

Creating a Derived Class

```
class Vehicle
{
public:
            Vehicle ()  {           weight = 0; }
            Vehicle (int wt)        {           weight = wt;
            };
            int GetWeight ()        {           return weight; }
            void SetWeight (int x ) {           weight = x;};
            int tempWeight;
private:
            int weight;
};

class Car: public Vehicle
{
public:
            friend void foo(Car &C);
            Car()       {speed = 0; }
            Car(int wt, int sp){    SetWeight(wt);          speed = sp;
            }
            int GetSpeed ()         {           return speed; }
            void SetSpeed (int x )  {           speed = x; }
private:
            int speed;
};
void foo(Car &C)
{           C.speed = 9; C.TempWeight = 10; }
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;

void main()
{
            Car MyCar(2,3);
            cout << MyCar.GetSpeed() << endl;
            cout << MyCar.GetWeight() << endl;
            cout << MyCar.tempWeight << endl;
            MyCar.tempWeight = 5;
            foo(MyCar);
            cout << MyCar.GetSpeed() << endl;
            cout << MyCar.GetWeight() << endl;
            cout << MyCar.tempWeight << endl;

}
```

```
3
2
-858993460
9
2
10
Press any key to continue
```

`protected` Members

`protected` Members

- Base class protected members
  - Intermediate level of protection between public and private
  - Protected data members are accessible through their own class and their own class's derived classes
    - Accessible to
      - Base class members (like public)
      - Base class friends (like public, private)
      - Derived class members (like public)
      - Derived class friends (like public)
    - BUT NO ONE using base class directly can access them directly (like private)

protected Members

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

class Vehicle
{
public:
            Vehicle ()  {           weight = 0; }
            Vehicle (int wt)        {           weight = wt;            };
            int GetWeight ()        {           return weight; }
            void SetWeight (int x ){            weight = x;};
private:
            int weight;
};

class Car: public Vehicle
{
public:
            friend void foo(Car &C);
            Car()       {speed = 0; }
            Car(int wt, int sp){    SetWeight(wt);          speed = sp;             }
            int GetSpeed ()         {           return speed; }
            void SetSpeed (int x ) {speed = x; }
            void Print() const { cout << speed << "\n" <<  weight << endl;   }
private:
            int speed;
};
void foo(Car &C)
{           C.speed = 9;}
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;

void main()
{
            Car MyCar(2,3);
            MyCar.Print();
}
```

Compile error Car can't access private data membes declared in its base class

protected Members

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

class Vehicle
{
public:
            Vehicle ()  {           weight = 0; }
            Vehicle (int wt)        {           weight = wt;            };
            int GetWeight ()        {           return weight; }
            void SetWeight (int x ){            weight = x;};
protected:
            int weight;
};

class Car: public Vehicle
{
public:
            friend void foo(Car &C);
            Car()       {speed = 0; }
            Car(int wt, int sp){    SetWeight(wt);          speed = sp;             }
            int GetSpeed ()         {           return speed; }
            void SetSpeed (int x ) {speed = x; }
            void Print() const { cout << speed << "\n" <<  weight << endl;   }
private:
            int speed;
};
void foo(Car &C)
{           C.speed = 9;}
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;

void main()
{
            Car MyCar(2,3);
            MyCar.Print();
}
```

```
3
2
```

protected Members

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

class Vehicle
{
public:
            Vehicle ()  {           weight = 0; }
            Vehicle (int wt)        {           weight = wt;            };
            int GetWeight ()        {           return weight; }
            void SetWeight (int x ){            weight = x;};
protected:
            int weight;
};

class Car: public Vehicle
{
public:
            friend void foo(Car &C);
            Car()       {speed = 0; }
            Car(int wt, int sp){    SetWeight(wt);          speed = sp;             }
            int GetSpeed ()         {           return speed; }
            void SetSpeed (int x ) {speed = x; }
            void Print() const { cout << speed << "\n" <<  weight << endl;   }
private:
            int speed;
};
void foo(Car &C)
{           C.speed = 9;}
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;

void main()
{
            Car MyCar(2,3);
            MyCar.weight = 3;
            cout << MyCar.weight << endl;
}
```

Compiler error, can't access protected data members directly from outside the class

protected Members

```
class Vehicle
{
public:
            Vehicle ()  {           weight = 0; }
            Vehicle (int wt)        {           weight = wt;            };
            int GetWeight ()        {           return weight; }
            void SetWeight (int x ){            weight = x;};
protected:
            int weight;
};

class Car: public Vehicle
{
public:
            friend void foo(Car &C);
            Car()       {speed = 0; }
            Car(int wt, int sp){    weight = wt;            speed = sp;             }
            int GetSpeed ()         {           return speed; }
            void SetSpeed (int x ) {speed = x; }
            void Print() const { cout << speed << "\n" <<  weight << endl;   }
private:
            int speed;
};
void foo(Car &C)
{           C.speed = 9;}
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;

void main()
{
            Car MyCar(2,3);
            MyCar.Print();
}
```

```
3
2
```

`protected` Members

- Problems of protected
  - No validity checking between derived class and the base one
  - Derived class can assign illegal values to protected members (since they can access them directly.)
  - Implementation dependent
    - Fragile (brittle) software

protected Members

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

class Vehicle
{
public:
            Vehicle ()  {           weight = 0; }
            Vehicle (int wt)        {           weight = wt;            };
            int GetWeight() const   {           return weight; }
            void SetWeight (int x ){            weight = x;};
            void GetName1() const { cout << "I'm a vehicle " << endl; }
private:
            int weight;
};

class Car: public Vehicle
{
public:
            friend void foo(Car &C);
            Car()       {speed = 0; }
            Car(int wt, int sp){    SetWeight(wt);          speed = sp;             }
            int GetSpeed ()         {           return speed; }
            void SetSpeed (int x ) {speed = x; }
            void GetName2() const { cout << "I'm a car " << endl; }
private:
            int speed;
};
void foo(Car &C)
{           C.speed = 9;}
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;

void main()
{
            Car MyCar(2,3);
            MyCar.GetName2();
}
```

```
I'm a car
Press any key to continue
```

protected Members

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

class Vehicle
{
public:
            Vehicle ()  {           weight = 0; }
            Vehicle (int wt)        {           weight = wt;            };
            int GetWeight() const   {           return weight; }
            void SetWeight (int x ){            weight = x;};
            void GetName1() const { cout << "I'm a vehicle " << endl; }
private:
            int weight;
};

class Car: public Vehicle
{
public:
            friend void foo(Car &C);
            Car()       {speed = 0; }
            Car(int wt, int sp){    SetWeight(wt);          speed = sp;             }
            int GetSpeed ()         {           return speed; }
            void SetSpeed (int x ) {speed = x; }
            void GetName2() const { cout << "I'm a car " << endl; }
private:
            int speed;
};
void foo(Car &C)
{           C.speed = 9;}
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;

void main()
{
            Vehicle MyVehicle;
            MyVehicle.GetName1();
            Car MyCar(2,3);
            MyCar.GetName2();
}
```

```
I'm a vehicle
I'm a car
Press any key to continue
```

protected Members

```
class Vehicle
{
public:
            Vehicle ()  {           weight = 0; }
            Vehicle (int wt)        {           weight = wt;            };
            int GetWeight() const   {           return weight; }
            void SetWeight (int x ){            weight = x;};
            void OverFun(){cout<< "In Vehicle class \n";}
private:
            int weight;
};

class Car: public Vehicle
{
public:
            friend void foo(Car &C);
            Car()       {speed = 0; }
            Car(int wt, int sp){    SetWeight(wt);          speed = sp;             }
            int GetSpeed ()         {           return speed; }
            void SetSpeed (int x ) {speed = x; }
            void OverFun()
            {
                        cout << "In car class" << endl; Vehicle::OverFun();
                         // overriding here with::
            }
private:    int speed;};
void foo(Car &C)
{           C.speed = 9;}
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;

void main()
{
            Vehicle MyVehicle;
            Car MyCar(2,3);
            cout << "Printing with Vehicle:" << endl;
            MyVehicle.OverFun();
            cout << "Printing with car:" << endl;
            MyCar.OverFun();
}
```

```
Printing with Vehicle:
In Vehicle class
Printing with car:
In car class
In Vehicle class
Press any key to continue
```

What You Can’t Inherit

What You Can't Inherit

- The following never ever can be inherited
  - Constructors
  - Destructors
  - Friends
  - Static data members
  - Static member functions
  - new operator
  - "=" operator

Overriding Base Class Behavior

Overriding Base Class Behavior

- What's overriding \ overloading?
  - Any child class function with the same name and same argument list as the parent
    - overrides the parent function
  - Any child class with the same name, yet with different argument list as for the parent
    - overloads the parent function

Constructors and Inheritance

Constructors and Inheritance

- When instantiate a derived class, a constructor for its base class is called first followed by the derived class constructor.
- Chains of constructors calls
  - Base class constructor called last, but
  - First to finish executing
- Example: Point \ Circle \ Cylinder
  - Point constructor called last, but
  - Point constructor body finishes executing first

Constructors and Inheritance

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

class Vehicle
{
public:
            Vehicle ()  {cout << "I'm a vehicle!" << endl; weight = 0; }
            Vehicle (int wt)        {weight = wt;           };
            int GetWeight() const   {return weight; }
            void SetWeight (int x ){weight = x;};
private:
            int weight;
};

class Car: public Vehicle
{
public:
            friend void foo(Car &C);
            Car()       { cout << "I'm a car!" << endl; speed = 0;   }
            Car(int wt, int sp){    SetWeight(wt);          speed = sp;             }
            int GetSpeed ()         {           return speed; }
            void SetSpeed (int x ) {speed = x; }
private:
            int speed;
};
void foo(Car &C)
{           C.speed = 9;}
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;

void main()
{
            Vehicle MyVehicle;
}
```

```
I'm a vehicle!
Press any key to continue
```

Constructors and Inheritance

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

class Vehicle
{
public:
            Vehicle ()  {cout << "I'm a vehicle!" << endl; weight = 0; }
            Vehicle (int wt)        {weight = wt;           };
            int GetWeight() const   {return weight; }
            void SetWeight (int x ){weight = x;};
private:
            int weight;
};

class Car: public Vehicle
{
public:
            friend void foo(Car &C);
            Car()       { cout << "I'm a car!" << endl; speed = 0;   }
            Car(int wt, int sp){    SetWeight(wt);          speed = sp;             }
            int GetSpeed ()         {           return speed; }
            void SetSpeed (int x ) {speed = x; }
private:
            int speed;
};
void foo(Car &C)
{           C.speed = 9;}
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;

void main()
{
            Car MyCar;
}
```

```
I'm a vehicle!
I'm a car!
Press any key to continue
```

Constructors and Inheritance

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

class Vehicle
{
public:
            Vehicle ()  {cout << "I'm a vehicle!" << endl; weight = 0; }
            Vehicle (int wt)        {weight = wt;           };
            int GetWeight() const   {return weight; }
            void SetWeight (int x ){weight = x;};
private:
            int weight;
};

class Car: public Vehicle
{
public:
            friend void foo(Car &C);
            Car()       { cout << "I'm a car!" << endl; speed = 0;   }
            Car(int wt, int sp){    SetWeight(wt);          speed = sp;             }
            int GetSpeed ()         {           return speed; }
            void SetSpeed (int x ) {speed = x; }
private:
            int speed;
};
void foo(Car &C)
{           C.speed = 9;}
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;

void main()
{
            Car MyCar();
}
```

```
Press any key to continue
```

Constructors and Inheritance

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

class Vehicle
{
public:
            Vehicle ()  {cout << "I'm a vehicle!" << endl; weight = 0; }
            Vehicle (int wt)        {weight = wt;           };
            int GetWeight() const   {return weight; }
            void SetWeight (int x ){weight = x;};
private:
            int weight;
};

class Car: public Vehicle
{
public:
            friend void foo(Car &C);
            Car()       { cout << "I'm a car!" << endl; speed = 0;   }
            Car(int wt, int sp){    SetWeight(wt);          speed = sp;             }
            int GetSpeed ()         {           return speed; }
            void SetSpeed (int x ) {speed = x; }
private:
            int speed;
};
void foo(Car &C)
{           C.speed = 9;}
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;

void main()
{
            Vehicle My();
}
```

```
Press any key to continue
```

Constructors and Inheritance

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

class Vehicle
{
public:
            Vehicle ()  {cout << "I'm a vehicle!" << endl; weight = 0; }
            Vehicle (int wt)        {weight = wt;           };
            int GetWeight() const   {return weight; }
            void SetWeight (int x ){weight = x;};
private:
            int weight;
};

class Car: public Vehicle
{
public:
            friend void foo(Car &C);
            Car()       { cout << "I'm a car!" << endl; speed = 0;   }
            Car(int wt, int sp){    SetWeight(wt);          speed = sp;             }
            int GetSpeed ()         {           return speed; }
            void SetSpeed (int x ) {speed = x; }
private:
            int speed;
};
void foo(Car &C)
{           C.speed = 9;}
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;

void main()
{
            Vehicle My();
            My.GetWeight();
}
```

```
Compiler error
```

Constructors and Inheritance

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

class Vehicle
{
public:
            Vehicle ()  {cout << "I'm a vehicle!" << endl; weight = 0; }
            Vehicle (int wt)        {           weight = wt;            };
            int GetWeight() const   {return weight; }
            void SetWeight (int x ){weight = x;};
private:
            int weight;
};

class Car: public Vehicle
{
public:
            friend void foo(Car &C);
            Car()       { cout << "I'm a car!" << endl; speed = 0;   }
            Car(int wt, int sp){    SetWeight(wt);          speed = sp;             }
            int GetSpeed ()         {           Vehicle::Vehicle(); return speed; }
            void SetSpeed (int x ) {speed = x; }
private:
            int speed;
};
void foo(Car &C)
{           C.speed = 9;}
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;

void main()
{
            Car MyCar;
            cout << MyCar.GetSpeed() <<
endl;
}
```

```
I'm a vehicle!
I'm a car!
I'm a vehicle!
0
Press any key to continue
```

Constructors and Inheritance

```
class Vehicle
{
public:
            Vehicle ()  {cout << "I'm a vehicle!" << endl; weight = 0; }
            Vehicle (int wt){cout << "I'm a vehicle!" << endl; weight += wt;      };
            int GetWeight() const   {           return weight; }
            void SetWeight (int x ){            weight = x;};
private:
            int weight;
};

class Car: public Vehicle
{
public:
            friend void foo(Car &C);
            Car()       { cout << "I'm a car!" << endl; speed = 0;   }
            Car(int wt, int sp){    SetWeight(wt);          speed = sp;             }
            int GetSpeed ()         {           Vehicle::Vehicle(5); return speed; }
            void SetSpeed (int x ) {speed = x; }
private:
            int speed;
};
void foo(Car &C)
{           C.speed = 9;}
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;

void main()
{
            Car MyCar;
            cout << MyCar.GetWeight() << endl;
            cout << MyCar.GetSpeed() << endl;
            cout << MyCar.GetSpeed() << endl;
            cout << MyCar.GetWeight() << endl;
}
```

```
I'm a vehicle!
I'm a car!
0
I'm a vehicle!
0
I'm a vehicle!
0
0
Press any key to continue
```

Constructors and Inheritance

```
class Vehicle
{
public:
            Vehicle ()  {cout << "I'm a vehicle!" << endl; weight = 0; }
            Vehicle (int wt){cout << "I'm a vehicle!" << endl; weight += wt;      };
            int GetWeight() const   {           return weight; }
            void SetWeight (int x ){            weight = x;};
private:
            int weight;
};

class Car: public Vehicle
{
public:
            friend void foo(Car &C);
            Car()       { cout << "I'm a car!" << endl; speed = 0;   }
            Car(int wt, int sp){    SetWeight(wt);          speed = sp;             }
            int GetSpeed ()         {           Vehicle::Vehicle(5); return speed; }
            void SetSpeed (int x ) {speed = x; }
private:
            int speed;
};
void foo(Car &C)
{           C.speed = 9;}
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;

void main()
{
            Car MyCar;
            cout << MyCar.GetWeight() << endl;
            cout << MyCar.GetSpeed() << endl;
            MyCar.SetWeight(4);
            cout << MyCar.GetWeight() << endl;
}
```

```
I'm a vehicle!
I'm a car!
0
I'm a vehicle!
0
4
Press any key to continue
```

Constructors and Inheritance

```
class Vehicle
{
public:
            Vehicle ()  {cout << "I'm a vehicle!" << endl; weight = 0; }
            Vehicle (int wt){cout << "I'm a vehicle!" << endl; weight += wt;      };
            int GetWeight() const   {           return weight; }
            void SetWeight (int x ){            weight = x;};
private:
            int weight;
};

class Car: public Vehicle
{
public:
            friend void foo(Car &C);
            Car()       { cout << "I'm a car!" << endl; speed = 0;   }
            Car(int wt, int sp){    SetWeight(wt);          speed = sp;             }
            int GetSpeed ()         {           Vehicle::Vehicle(5); return speed; }
            void SetSpeed (int x ) {speed = x; }
private:
            int speed;
};
void foo(Car &C)
{           C.speed = 9;}
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;

void main()
{
            Car MyCar;
            cout << MyCar.GetWeight() << endl;
            cout << MyCar.GetSpeed() << endl;
            MyCar.SetWeight(4);
            cout << MyCar.GetWeight() << endl;
            cout << MyCar.GetSpeed() << endl;
            cout << MyCar.GetWeight() << endl;
}
```

```
I'm a vehicle!
I'm a car!
0
I'm a vehicle!
0
4
I'm a vehicle!
0
4
Press any key to continue
```

Constructors and Inheritance

```
class Vehicle
{
public:
            Vehicle ()  { weight = 0; }
            Vehicle (int wt){  weight = wt;    };
            int GetWeight() const  {          return weight; }
            void SetWeight (int x ){          weight = x;};
private:
            int weight;
};

class Car: public Vehicle
{
public:
            friend void foo(Car &C);
            Car(int wt, int sp){   SetWeight(wt);          speed = sp;            }
            int GetSpeed ()        {           return speed; }
            void SetSpeed (int x ) {speed = x; }
private:
            int speed;
};
void foo(Car &C)
{           C.speed = 9;}
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;

void main()
{
            Car MyCar (2,3);
            cout << MyCar.GetSpeed() << endl;
            cout << MyCar.GetWeight() << endl;


}
```

```
3
2
Press any key to continue
```

Code Cracking

Code Cracking

```
class Vehicle
{
public:
            Vehicle (int wt)
            {           weight = wt;          };
            int GetWeight ()
            {           return weight; }
            void SetWeight (int x )
            {           weight = x;};
private:
            int weight;
};
class Car: public Vehicle
{
public:
            Car()
            {           speed = 0; }
            Car(int wt, int sp)
            {           SetWeight(wt);        speed = sp;           }
            int GetSpeed ()
            {           return speed; }
            void SetSpeed (int x )
            {           speed = x; }
private:
            int speed;
};
```

Compiler error. We didn't define the 0 parameter Vehicle constructor and but we used it in the (0 parameter constructor of the) derived class

Code Cracking

```
class Vehicle
{
public:
            Vehicle (int wt)
            {           weight = wt;          };
            int GetWeight ()
            {           return weight; }
            void SetWeight (int x )
            {           weight = x;};
private:
            int weight;
};
class Car: public Vehicle
{
public:
            Car()
            {           speed = 0; }
            Car(int wt, int sp)
            {           SetWeight(wt);        speed = sp;           }
            int GetSpeed ()
            {           return speed; }
            void SetSpeed (int x )
            {           speed = x; }
private:
            int speed;
};
```

The derived class implicitly call the base class constructor!

Code Cracking

```
class Vehicle
{
public:
            Vehicle (int wt)
            {           weight = wt;          };
            int GetWeight ()
            {           return weight; }
            void SetWeight (int x )
            {           weight = x;};
private:
            int weight;
};
class Car: public Vehicle
{
public:
            Car()
            {           speed = 0; }
            Car(int wt, int sp)
            {           SetWeight(wt);        speed = sp;           }
            int GetSpeed ()
            {           return speed; }
            void SetSpeed (int x )
            {           speed = x; }
private:
            int speed;
};
```

The compiler looks for a matching non-parameter base class constructor but can't find one!

Code Cracking

```
class Vehicle
{
public:
            Vehicle (int wt)
            {           weight = wt;          };
            int GetWeight ()
            {           return weight; }
            void SetWeight (int x )
            {           weight = x;};
private:
            int weight;
};
class Car: public Vehicle
{
public:
            Car()
            {           speed = 0; }
            Car(int wt, int sp)
            {           SetWeight(wt);        speed = sp;
            int GetSpeed ()
            {           return speed; }
            void SetSpeed (int x )
            {           speed = x; }
private:
            int speed;
};
```

To solve this, 2 ways:

1. define an non-parameter base class constructor.
2. call the parametric base class constructor (the one with parameters) and make an explicit call for it from the derived class.

Code Cracking

```
class Vehicle
{
public:
            Vehicle ()
            {           weight = 0; }
            Vehicle (int wt)
            {           weight = wt;          };
            int GetWeight ()
            {           return weight; }
            void SetWeight (int x )
            {           weight = x;};
private:
            int weight;
};
class Car: private Vehicle
{
public:
            Car()
            {           speed = 0; }
            Car(int wt, int sp)
            {           SetWeight(wt);        speed = sp;           }
            int GetSpeed ()
            {           return speed; }
            void SetSpeed (int x )
            {           speed = x; }
private:
            int speed;
};
```

```
Error       1           error C2247: 'Vehicle::GetWeight' not accessible because 'Car' uses 'private' to inherit from 'Vehicle'
            c:\users\zmee\documents\visual studio 2008\projects\exam\exam\exam.cpp  11          Exam
Error       2           error C2247: 'Vehicle::SetWeight' not accessible because 'Car' uses 'private' to inherit from 'Vehicle'
            c:\users\zmee\documents\visual studio 2008\projects\exam\exam\exam.cpp  13          Exam
```

Code Cracking

```
class Vehicle
{
public:
            Vehicle ()
            {           weight = 0; }
            Vehicle (int wt)
            {           weight = wt;          };
            int GetWeight ()
            {           return weight; }
            void SetWeight (int x )
            {           weight = x;};
private:
            int weight;
};
class Car: public Vehicle
{
public:
            Car()
            {           speed = 0; }
            Car(int wt, int sp)
            {           weight = wt;          speed = sp;           }
            int GetSpeed ()
            {           return speed; }
            void SetSpeed (int x )
            {           speed = x; }
private:
            int speed;
};
```

Compiler error. Can’t access private data member of the base class directly

Creating a Derived Class

As we have seen, derived class can affect state changes in private base-class members.

BUT ONLY THROUGH, Non-private functions provided in the base class and inherited into the derived class.

Code Cracking

```
class Vehicle
{
public:
            friend void foo(Vehicle &V);
            Vehicle ()
            {           weight = 0; }
            Vehicle (int wt)
            {           weight = wt;            };
            int GetWeight ()
            {           return weight; }
            void SetWeight (int x )
            {           weight = x;};
private:
            int weight;
};

class Car: public Vehicle
{
public:
            Car()
            {           speed = 0; }
            Car(int wt, int sp)
            {           SetWeight(wt);          speed = sp;             }
            int GetSpeed ()
            {           return speed; }
            void SetSpeed (int x )
            {           speed = x; }
private:
            int speed;
};


void foo(Vehicle &V)
{           V.weight = 9; }
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;



void main()
{
            Vehicle V;
            Car MyCar(2,3);
            MyCar.SetWeight(5);     // member function in base class
            MyCar.SetSpeed(10);
            MyCar.foo(V);
            cout << "Printing" << endl;
            cout << MyCar.GetSpeed() << endl;
            cout << MyCar.GetWeight() << endl;
}
```

```
Printing
10
5
Press any key to continue
```

Code Cracking

```
class Vehicle
{
public:
            friend void foo(Vehicle &V);
            Vehicle ()
            {           weight = 0; }
            Vehicle (int wt)
            {           weight = wt;            };
            int GetWeight ()
            {           return weight; }
            void SetWeight (int x )
            {           weight = x;};
private:
            int weight;
};

class Car: public Vehicle
{
public:
            Car()
            {           speed = 0; }
            Car(int wt, int sp)
            {           SetWeight(wt);          speed = sp;             }
            int GetSpeed ()
            {           return speed; }
            void SetSpeed (int x )
            {           speed = x; }
private:
            int speed;
};


void foo(Vehicle &V)
{           V.weight = 9; }
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;



void main()
{
            Vehicle V;
            Car MyCar(2,3);
            MyCar.SetWeight(5);     // member function in base class
            MyCar.SetSpeed(10);
            foo(V);
            cout << "Printing" << endl;
            cout << MyCar.GetSpeed() << endl;
            cout << MyCar.GetWeight() << endl;

}
```

Compiler error. foo is not a member functions in Car. It's not inherited!

Constructors and Destructors

Constructors and Destructors

- Chain of destructors calls
  - Reverse order of constructor chain
  - Destructor of the derived class called first
  - Destructor of the next class up hierarchy called next
    - Continue up hierarchy until final base class is reached
    - After final base class destructor, object is removed from memory

Constructors and Destructors

```
class Vehicle
{
public:
            Vehicle ()  { weight = 0; }
            Vehicle (int wt){ weight = wt; cout <<"Vehicle class " <<
weight << endl;};
            ~Vehicle(){cout << "Destructing vehicle class for " << weight
<< endl; }
            int GetWeight() const   {           return weight; }
            void SetWeight (int x ) {           weight = x;};
private:
            int weight;
};

class Car: public Vehicle
{
public:
            friend void foo(Car &C);
            Car(int wt, int sp):Vehicle(wt){speed = sp; cout <<"Car class
for  "
            << speed << "-" << GetWeight() << endl;
}
            ~Car(){cout << "Destructing car class for " << speed << "-"
<< GetWeight()          << endl; }
            int GetSpeed ()         {            return speed; }
            void SetSpeed (int x )  {speed = x; }
private:
            int speed;
};
void foo(Car &C)
{           C.speed = 9;}
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;

void main()
{
            Vehicle V1;
            {
                        Vehicle V2;
                        Car MyCar1 (2,3);
            }
            Car MyCar2 (2,3);
            Car MyCar3 (2,3);
            Vehicle V3;
}
```

```
Vehicle class 2
Car class for  3-2
Destructing car class for 3-2
Destructing vehicle class for 2
Destructing vehicle class for 0
Vehicle class 2
Car class for  3-2
Vehicle class 2
Car class for  3-2
Destructing vehicle class for 0
Destructing car class for 3-2
Destructing vehicle class for 2
Destructing car class for 3-2
Destructing vehicle class for 2
Destructing vehicle class for 0
Press any key to continue
```

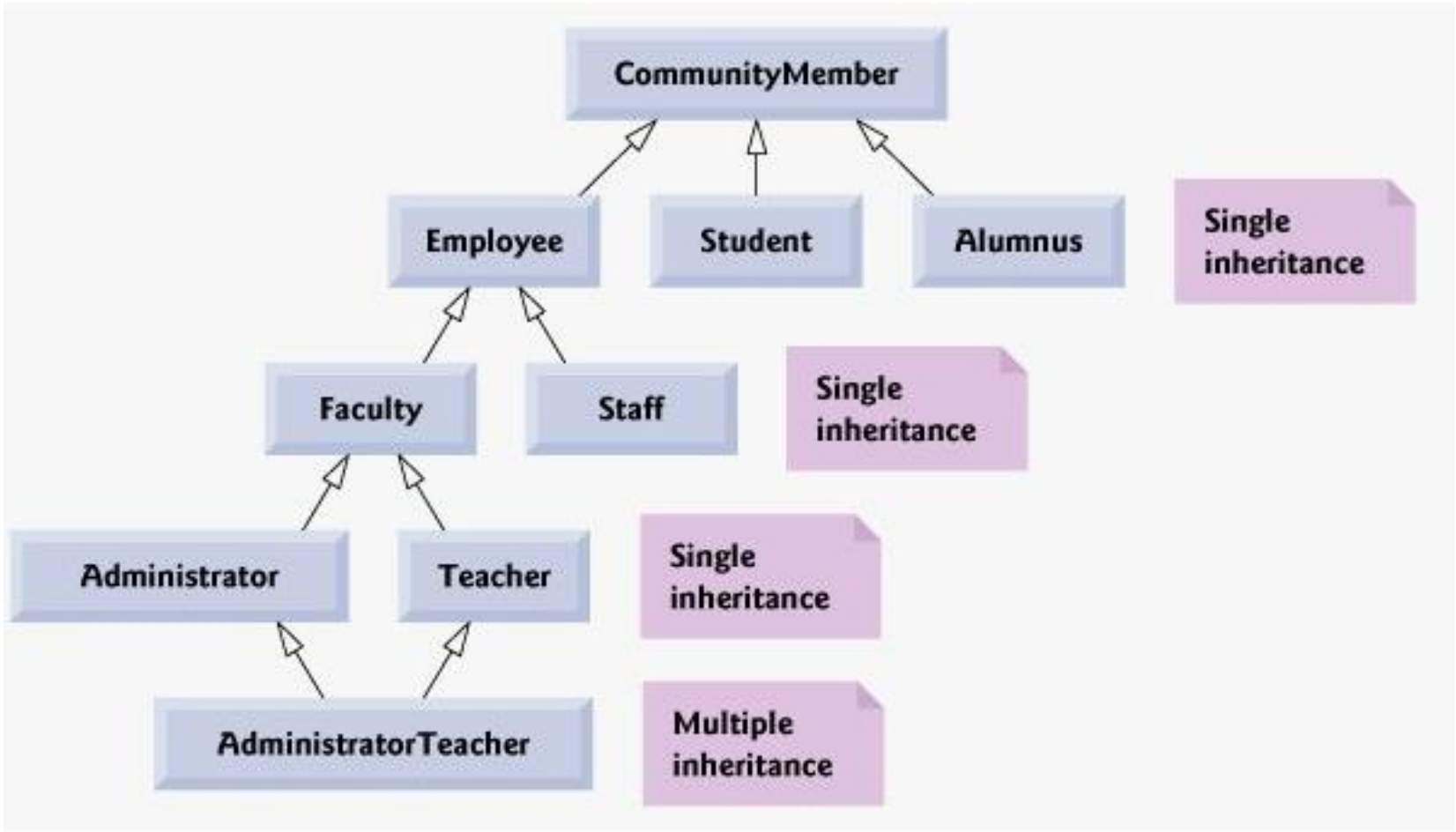
Constructors and Destructors

```
class Vehicle
{
public:
            Vehicle ()  { weight = 0; }
            Vehicle (int wt){ weight = wt; cout <<"Vehicle class " <<
weight << endl;};
            ~Vehicle(){cout << "Destructing vehicle class for " << weight
<< endl; }
            int GetWeight() const   {           return weight; }
            void SetWeight (int x ) {           weight = x;};
private:
            int weight;
};

class Car: public Vehicle
{
public:
            friend void foo(Car &C);
            Car(int wt, int sp):Vehicle(wt){speed = sp; cout <<"Car class
for  "
            << speed << "-" << GetWeight() << endl;
}
            ~Car(){cout << "Destructing car class for " << speed << "-"
<< weight   << endl; }
            int GetSpeed ()         {            return speed; }
            void SetSpeed (int x )  {speed = x; }
private:
            int speed;
};
void foo(Car &C)
{           C.speed = 9;}
```

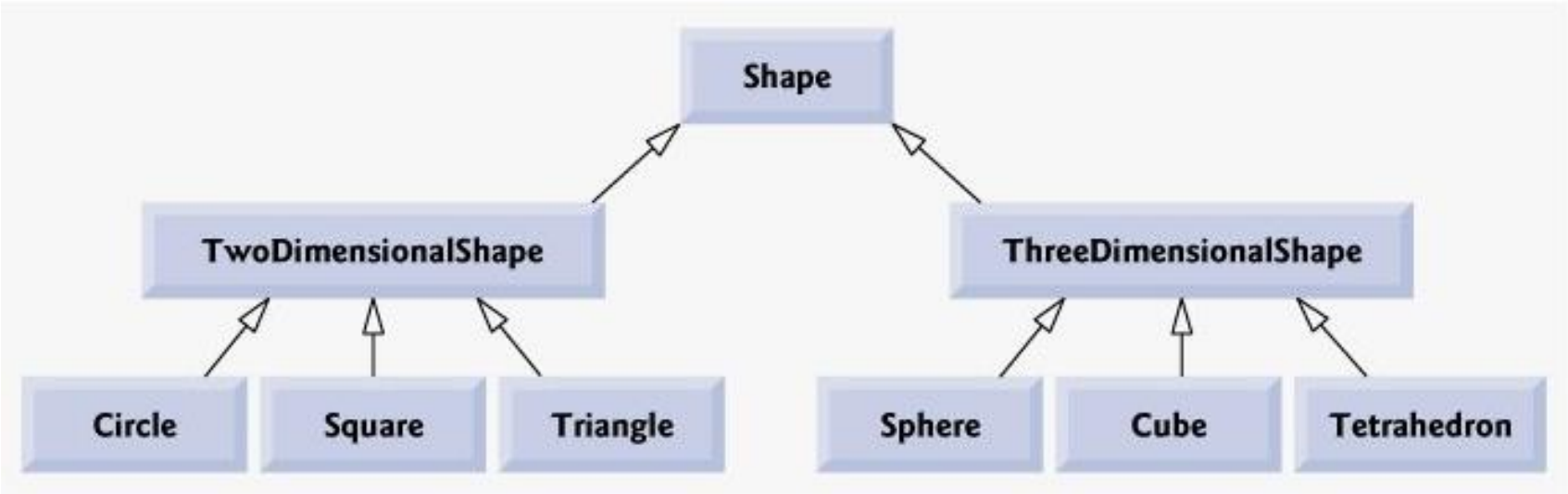
Compiler error, accessing private data member of vehicle

Multiple Inheritance

Single VS Multiple Inheritance



Single VS Multiple Inheritance



Multiple Inheritance

- Looks like the single inheritance
- Except!
  - There's multiple base classes
  - Separated by commas
  - Individually can be declared as public, protected or private
    - Default is private
  - Inherited member variables are accessible according to the rules of single inheritance

- Name conflicts
  - Can result in a member of a base class being hidden by the member of the derived class with the same name.

Multiple Inheritance

```
#include <iostream>
using namespace::std;
class OptionList
{
public:
            OptionList(){cout << "OptionList constructor \n";};
            ~OptionList(){cout << "OptionList destructor \n";};
};

class Window
{
public:
            Window(){cout << "Window constructor \n";};
            ~Window(){cout << "Window destructor \n";};
};

class Menu: public OptionList, public Window
{
public:
            Menu(){cout << "Menu constructor \n";};
            ~Menu(){cout << "Menu destructor \n";};
};
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;

void main()
{
            Menu M;
}
```

```
OptionList constructor
Window constructor
Menu constructor
Menu destructor
Window destructor
OptionList destructor
Press any key to continue
```

Multiple Inheritance

```
#include <iostream>
using namespace::std;
class OptionList
{
public:
            OptionList(){cout << "OptionList constructor \n";};
            ~OptionList(){cout << "OptionList destructor \n";};
};

class Window
{
public:
            Window(){cout << "Window constructor \n";};
            ~Window(){cout << "Window destructor \n";};
};

class Menu: public Window, public OptionList
{
public:
            Menu(){cout << "Menu constructor \n";};
            ~Menu(){cout << "Menu destructor \n";};
};
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;

void main()
{
            Menu M;
}
```

```
Window constructor
OptionList constructor
Menu constructor
Menu destructor
OptionList destructor
Window destructor
Press any key to continue
```

The constructors runs in order of declaration

Multiple Inheritance

```
#include <iostream>
using namespace::std;
class OptionList
{
public:
            OptionList(){cout << "OptionList constructor \n";};
            ~OptionList(){cout << "OptionList destructor \n";};
};

class Window
{
public:
            Window(){cout << "Window constructor \n";};
            ~Window(){cout << "Window destructor \n";};
};

class Menu: public Window, public OptionList
{
public:
            Menu(){cout << "Menu constructor \n";};
            ~Menu(){cout << "Menu destructor \n";};
};
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;

void main()
{
            Menu M();
}
```

```
Press any key to continue
```

Multiple Inheritance

```
#include <iostream>
using namespace::std;
class OptionList
{
public:
            //OptionList(){cout << "OptionList constructor \n";};
            ~OptionList(){cout << "OptionList destructor \n";};
};

class Window
{
public:
            //Window(){cout << "Window constructor \n";};
            ~Window(){cout << "Window destructor \n";};
};

class Menu: public Window, public OptionList
{
public:
            Menu(){cout << "Menu constructor \n";};
            ~Menu(){cout << "Menu destructor \n";};
};
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;

void main()
{
            Menu M;
}
```

```
Menu constructor
Menu destructor
OptionList destructor
Window destructor
Press any key to continue
```

Multiple Inheritance

```
#include <iostream>
using namespace::std;
class OptionList
{
public:
            //OptionList(){cout << "OptionList constructor \n";};
            OptionList(int x){cout << "OptionList constructor \n";};
            ~OptionList(){cout << "OptionList destructor \n";};
};

class Window
{
public:
            //Window(){cout << "Window constructor \n";};
            Window(int y){cout << "Window constructor \n";};
            ~Window(){cout << "Window destructor \n";};
};

class Menu: public Window, public OptionList
{
public:
            Menu(){cout << "Menu constructor \n";};
            ~Menu(){cout << "Menu destructor \n";};
};
```

```
Compiler error
```

Multiple Inheritance

```
#include <iostream>
using namespace::std;
class OptionList
{
public:
            //OptionList(){cout << "OptionList constructor \n";};
            OptionList(int x){cout << "OptionList constructor \n";};
            ~OptionList(){cout << "OptionList destructor \n";};
};

class Window
{
public:
            //Window(){cout << "Window constructor \n";};
            Window(int y){cout << "Window constructor \n";};
            ~Window(){cout << "Window destructor \n";};
};

class Menu: public Window, public OptionList
{
public:
            Menu(int n, int m):Window(n),OptionList(m){cout<<"Menu constructor
\n";};
            ~Menu(){cout << "Menu destructor \n";};
};
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;

void main()
{
            Menu M;
}
```

```
Window constructor
OptionList constructor
Menu constructor
Menu destructor
OptionList destructor
Window destructor
Press any key to continue
```

Multiple Inheritance

```
#include <iostream>
using namespace::std;
class OptionList
{
public:
            //OptionList(){cout << "OptionList constructor \n";};
            OptionList(int x){cout << "OptionList constructor \n";};
            ~OptionList(){cout << "OptionList destructor \n";};
};

class Window
{
public:
            //Window(){cout << "Window constructor \n";};
            Window(int y){cout << "Window constructor \n";};
            ~Window(){cout << "Window destructor \n";};
};

class Menu: public Window, public OptionList
{
public:
            Menu(int n, int m):OptionList(m),Window(n){cout<<"Menu constructor
\n";};
            ~Menu(){cout << "Menu destructor \n";};
};
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;

void main()
{
            Menu M;
}
```

```
Window constructor
OptionList constructor
Menu constructor
Menu destructor
OptionList destructor
Window destructor
Press any key to continue
```

Multiple Inheritance

```
#include <iostream>
using namespace::std;
class OptionList
{
public:
            OptionList(int x){cout << "OptionList constructor \n";};
            ~OptionList(){cout << "OptionList destructor \n";};
            void HighLighting(){cout << "In OptionList \n";};
};

class Window
{
public:
            Window(int y){cout << "Window constructor \n";};
            ~Window(){cout << "Window destructor \n";};
            void HighLighting(){cout << "In Window\n";};
};

class Menu: public Window, public OptionList
{
public:
            Menu(int n, int m):OptionList(n), Window(m){cout << "Menu constructor
\n";};
            ~Menu(){cout << "Menu destructor \n";};
};
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;

void main()
{
            Menu M(2,3);
}
```

Compile and run

Multiple Inheritance

```
#include <iostream>
using namespace::std;
class OptionList
{
public:
            OptionList(int x){cout << "OptionList constructor \n";};
            ~OptionList(){cout << "OptionList destructor \n";};
            void HighLighting(){cout << "In OptionList \n";};
};

class Window
{
public:
            Window(int y){cout << "Window constructor \n";};
            ~Window(){cout << "Window destructor \n";};
            void HighLighting(){cout << "In Window\n";};
};

class Menu: public Window, public OptionList
{
public:
            Menu(int n, int m):OptionList(n), Window(m){cout << "Menu constructor
\n";};
            ~Menu(){cout << "Menu destructor \n";};
};
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;

void main()
{
            Menu M(2,3);
            M.HighLighting();
}
```

Now, it's a compiler error

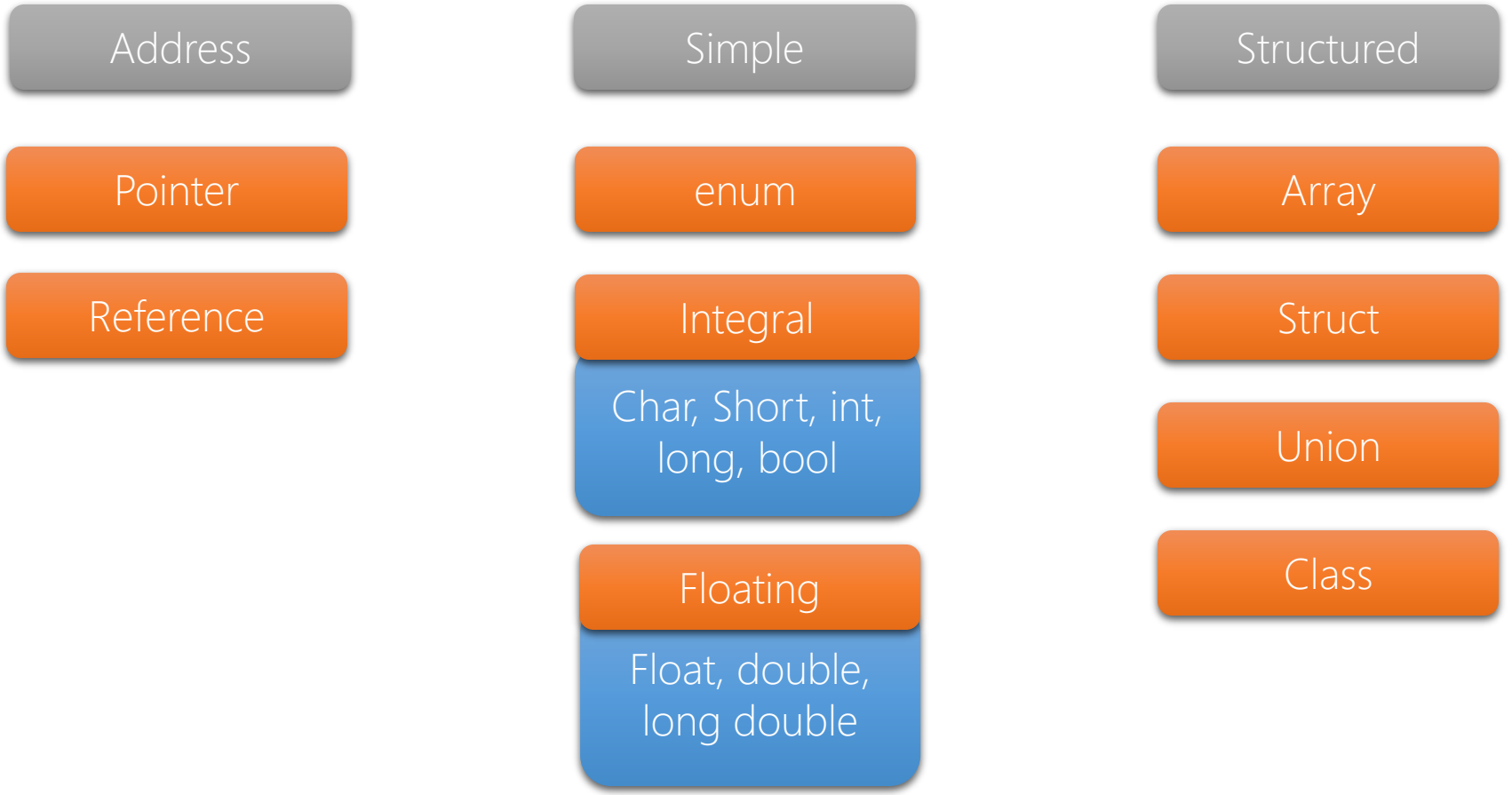
Multiple Inheritance

```
#include <iostream>
using namespace::std;
class OptionList
{
public:
            OptionList(int x){cout << "OptionList constructor \n";};
            ~OptionList(){cout << "OptionList destructor \n";};
            void HighLighting(){cout << "In OptionList \n";};
};

class Window
{
public:
            Window(int y){cout << "Window constructor \n";};
            ~Window(){cout << "Window destructor \n";};
            void HighLighting(){cout << "In Window\n";};
};

class Menu: public Window, public OptionList
{
public:
            Menu(int n, int m):OptionList(n), Window(m){cout << "Menu constructor
\n";};
            ~Menu(){cout << "Menu destructor \n";};
};
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;

void main()
{
            Menu M(2,3);
            M.Window::HighLighting();
            M.OptionList::HighLighting();
}
```

```
Window constructor
OptionList constructor
Menu constructor
In Window
In OptionList
Menu destructor
OptionList destructor
Window destructor
Press any key to continue
```

C++ data types


Address
Pointer
Reference
Simple
enum
Integral
Char, Short, int, long, bool
Floating
Float, double, long double
Structured
Array
Struct
Union
Class

Nested Classes

Classes that are defined inside another classes are called nested classes
In the public, private or protected section
Such a nested class can be considered a member of the outer class

Nested classes

- Visibility
  - In public section:
    - Visible outside the outer class
  - In private section:
    - Visible only for the member of the outer class
  - In protected section:
    - Subclasses, derived from the outer class

Nested classes

```
class Surround
{
public:
            class FirstWithin
            {
            public:
                        FirstWithin();
                        int GetVarFirst()
                        {           return xFirst; }
            private:
                        int xFirst;
            };

private:
            class SecondWithin
            {
            public:
                        SecondWithin();
                        int GetVarSecond()
                        {           return xSecond
                        }
            private:
                        int xSecond;
            };
            SecondWithin obj;
            int xSurround;
};
```

Class `FirstWithin`

- Visibility
  - Both outside and inside
  - Member functions are globally visible
  - `Int` variable is only visible to the class `FirstWithin`
  - Neither `Surround` nor `SecondWithin` can access the variable of the class `FirstWithin` directly

Nested classes

```
class Surround
{
public:
            class FirstWithin
            {
            public:
                        FirstWithin();
                        int GetVarFirst()
                        {           return xFirst; }
            private:
                        int xFirst;
            };

private:
            class SecondWithin
            {
            public:
                        SecondWithin();
                        int GetVarSecond()
                        {           return xSecond
                        }
            private:
                        int xSecond;
            };
            SecondWithin obj;
            int xSurround;
};
```

Class `FirstWithin`

- Visibility
  - Both outside and inside
  - Member functions are globally visible
  - `Int` variable is only visible to the class `FirstWithin`
  - Neither `Surround` nor `SecondWithin` can access the variable of the class `FirstWithin` directly

Class `SecondWith`

- Visibility
  - Just inside surround
  - Member functions can only be reached by the members of class `SecondWithin` only
  - `Int` variable is only visible to the class `SecondWithin`
  - Neither `Surround` nor `FirstWithin` can access the variable of the class `SecondWithin` directly

Nested classes

```
class Surround
{
public:
            class FirstWithin
            {
            public:
                        FirstWithin();
                        int GetVarFirst()
                        {           return xFirst; }
            private:
                        int xFirst;
            };

private:
            class SecondWithin
            {
            public:
                        SecondWithin();
                        int GetVarSecond()
                        {           return xSecond
                        }
            private:
                        int xSecond;
            };
            SecondWithin obj;
            int xSurround;
};
```

```
class Surround
{
public:
            class FirstWithin
            {
            public:
                        FirstWithin();
                        int GetVarFirst();
            private:
                        int xFirst;
            };

private:
            class SecondWithin
            {
            public:
                        SecondWithin();
                        int GetVarSecond();
            private:
                        int xSecond;
            };
            SecondWithin obj;
            int xSurround;
};
int Surround::FirstWithin::GetVarFirst()
{           return xFirst; }
int Surround::SecondWithin::GetVarSecond()
{           return xSecond; }
```

Note how we define the member functions here

Nested classes

```
class Surround
{
public:
		class FirstWithin
		{
		public:
			FirstWithin();
			int GetVarFirst()
			{		return xFirst; }
		private:
			int xFirst;
		};

private:
		class SecondWithin
		{
		public:
			SecondWithin();
			int GetVarSecond()
			{		return xSecond
			}
		private:
			int xSecond;
		};
		SecondWithin obj;
		int xSurround;
};
```

```
#include <iostream>
#include "MyFile.h"
using namespace::std;

void main()
{
		Surround s;
		Surround::FirstWithin First;

}
```

Compile & Run

```
class Surround
{
public:
		class FirstWithin
		{
		public:
			FirstWithin();
			int GetVarFirst();
		private:
			int xFirst;
		};

private:
		class SecondWithin
		{
		public:
			SecondWithin();
			int GetVarSecond();
		private:
			int xSecond;
		};
		SecondWithin obj;
		int xSurround;
};
int Surround::FirstWithin::GetVarFirst()
{		return xFirst; }
int Surround::SecondWithin::GetVarSecond()
{		return xSecond; }
```

Nested classes

```
class Surround
{
public:
            class FirstWithin
            {
            public:
                        FirstWithin();
                        int GetVarFirst()
                        {           return xFirst; }
            private:
                        int xFirst;
            };

private:
            class SecondWithin
            {
            public:
                        SecondWithin();
                        int GetVarSecond()
                        {           return xSecond
                        }
            private:
                        int xSecond;
            };
            SecondWithin obj;
            int xSurround;
};
```

```
void main()
{
            Surround s;
            Surround::FirstWithin First;
            Surround::SecondWithin Second;

}
```

Compiler error, can't access private data members SecondWithin

```
class Surround
{
public:
            class FirstWithin
            {
            public:
                        FirstWithin();
                        int GetVarFirst();
            private:
                        int xFirst;
            };

private:
            class SecondWithin
            {
            public:
                        SecondWithin();
                        int GetVarSecond();
            private:
                        int xSecond;
            };
            SecondWithin obj;
            int xSurround;
};
int Surround::FirstWithin::GetVarFirst()
{           return xFirst; }
int Surround::SecondWithin::GetVarSecond()
{           return xSecond; }
```

Nested classes

```
class Surround
{
public:
            class FirstWithin
            {
            public:
                        FirstWithin();
                        int GetVarFirst()
                        {           return xFirst; }
            private:
                        int xFirst;
            };

private:
            class SecondWithin
            {
            public:
                        SecondWithin();
                        int GetVarSecond()
                        {           return xSecond
                        }
            private:
                        int xSecond;
            };
            SecondWithin obj;
            int xSurround;
};
```

```
void main()
{
            Surround::FirstWithin First;
            Surround s;
            s.obj.GetVarSecond();
}
```

Compile & Run

```
class Surround
{
public:
            class FirstWithin
            {
            public:
                        FirstWithin();
                        int GetVarFirst();
            private:
                        int xFirst;
            };

private:
            class SecondWithin
            {
            public:
                        SecondWithin();
                        int GetVarSecond();
            private:
                        int xSecond;
            };
            SecondWithin obj;
            int xSurround;
};
int Surround::FirstWithin::GetVarFirst()
{           return xFirst; }
int Surround::SecondWithin::GetVarSecond()
{           return xSecond; }
```

How can we access private data members is Nested Classes?

friend!

`friend!`

Allowing the surrounding class to access private data members of the nested classes.
The nested class to access the private data members of the surrounding class and other nested classes.

Access Private Members in Nested classes

```
class Surround
{
public:
		class FirstWithin
		{
			friend class Surround;
			public:
				FirstWithin();
				int GetVarFirst();
			private:
				static int xFirst;
		};

private:
		class SecondWithin
		{
			friend class surround;
			public:
				SecondWithin();
				int GetVarSecond();
			private:
				static int xSecond;
		};
public:
		SecondWithin obj;
		int xSurround;
};

int Surround::FirstWithin::GetVarFirst()
{		FirstWithin::xFirst = SecondWithin::xSecond;  return xFirst; }

int Surround::SecondWithin::GetVarSecond()
{		FirstWithin::xFirst = 9; return xSecond; }
```

Allowing The surrounding class to access private data members of the nested classes

Access Private Members in Nested classes

```
class Surround
{
public:
		class FirstWithin
		{
				friend class Surround;
				public:
						FirstWithin();
						int GetVarFirst();
				private:
						static int xFirst;
		};

private:
		class SecondWithin
		{
				friend class surround;
				public:
						SecondWithin();
						int GetVarSecond();
				private:
						static int xSecond;
		};
public:
		SecondWithin obj;
		int xSurround;
};

int Surround::FirstWithin::GetVarFirst()
{		FirstWithin::xFirst = SecondWithin::xSecond;  return xFirst; }

int Surround::SecondWithin::GetVarSecond()
{		FirstWithin::xFirst = 9; return xSecond; }
```

But we are still missing something
What is it?

Access Private Members in Nested classes

```
class Surround
{
                class FirstWithin;
                class SecondWithin;
                friend class FirstWithin;
                friend class SecondWithin;
public:
                class FirstWithin
                {
                friend class Surround;
                public:
                                FirstWithin();
                                int GetVarFirst();
                private:
                                static int xFirst;
                };

private:
                class SecondWithin
                {
                friend class surround;
                public:
                                SecondWithin();
                                int GetVarSecond();
                private:
                                static int xSecond;
                };
public:
                SecondWithin obj;
                int xSurround;
};
```

We should use Forward Declaration
So that classes can know other classes exist

Quiz

Quiz #1

```
class Vehicle
{
public:
            Vehicle (char *n)       { name = n; weight = 0; }
            Vehicle (int wt, char* n ){ name=n; cout <<"Vehicle class " << name <<
weight << endl;         weight = wt; };
            ~Vehicle(){cout << "Destructing vehicle class for " << name << weight
<< endl; }
            int GetWeight() const   {           return weight; }
            void SetWeight (int x ){            weight = x;};
private:
            int weight;
            char *name;
};

class Car: public Vehicle
{
public:

            Car(int wt, int sp, char *n1, char *n2):Vehicle(wt,n2){name = n1;   cout
<<"Car class for   " << name << speed << "-" << GetWeight() << endl;   speed = sp;  }
            ~Car(){cout << "Destructing car class for " << name << speed << "-" <<
GetWeight() << endl; }
            int GetSpeed ()         {           return speed; }
            void SetSpeed (int x ) {speed = x; }
private:
            int speed;
            char *name;
};
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;

static Vehicle VG1("VG1");
Vehicle VG2("VG2");
void main()
{
            Vehicle V1("1");
            {
                        static Vehicle V2("2");
                        Car MyCar1 (2,3,"1","temp1");
            }
            Car MyCar2 (2,3,"2","temp2");
            Car MyCar3 (2,3,"3","temp3");
            Vehicle V3 ("3");
}
```

```
Vehicle class temp1-858993460
Car class for  1-858993460-2
Destructing car class for 13-2
Destructing vehicle class for temp12
Vehicle class temp2-858993460
Car class for  2-858993460-2
Vehicle class temp3-858993460
Car class for  3-858993460-2
Destructing vehicle class for 30
Destructing car class for 33-2
Destructing vehicle class for temp32
Destructing car class for 23-2
Destructing vehicle class for temp22
Destructing vehicle class for 10
Destructing vehicle class for 20
Destructing vehicle class for VG20
Destructing vehicle class for VG10
Press any key to continue
```

Quiz #2

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

class Vehicle
{
public:
            Vehicle ()
            {           weight = 0; }
            Vehicle (int wt)
            {           weight = wt;            };
            int GetWeight ()
            {           return weight; }
            void SetWeight (int x )
            {           weight = x;};
private:
            int weight;
};

class Car: public Vehicle
{
public:
            friend void foo(Car &C);
            Car()
            {           speed = 0; }
            Car(int wt, int sp)
            {           SetWeight(wt);          speed = sp;             }
            int GetSpeed ()
            {           return speed; }
            void SetSpeed (int x )
            {           speed = x; }
private:
            int speed;
};
void foo(Car &C)
{           static int x = 6;       C.speed =+ x;           x++; }
```

```
#include <iostream>
#include "Testing.h"
using namespace::std;
void main()
{
            Car MyCar(2,3);
            cout << MyCar.GetSpeed() << endl;
            cout << MyCar.GetWeight() << endl;
            foo(MyCar);
            MyCar.SetWeight(5);
            MyCar.SetSpeed(10);
            foo(MyCar);
            cout << MyCar.GetSpeed() << endl;
            cout << MyCar.GetWeight() << endl;
            foo(MyCar);
            cout << MyCar.GetSpeed() << endl;
            cout << MyCar.GetWeight() << endl;
            foo(MyCar);
            cout << MyCar.GetSpeed() << endl;
            cout << MyCar.GetWeight() << endl;
}
```

```
3
2
7
5
8
5
9
5
```