UEE1303
Objective-Oriented Programming

C++_Lecture 10:

Inheritance (I) –

Basic and Single Inheritance

C: How to Program 8th ed.

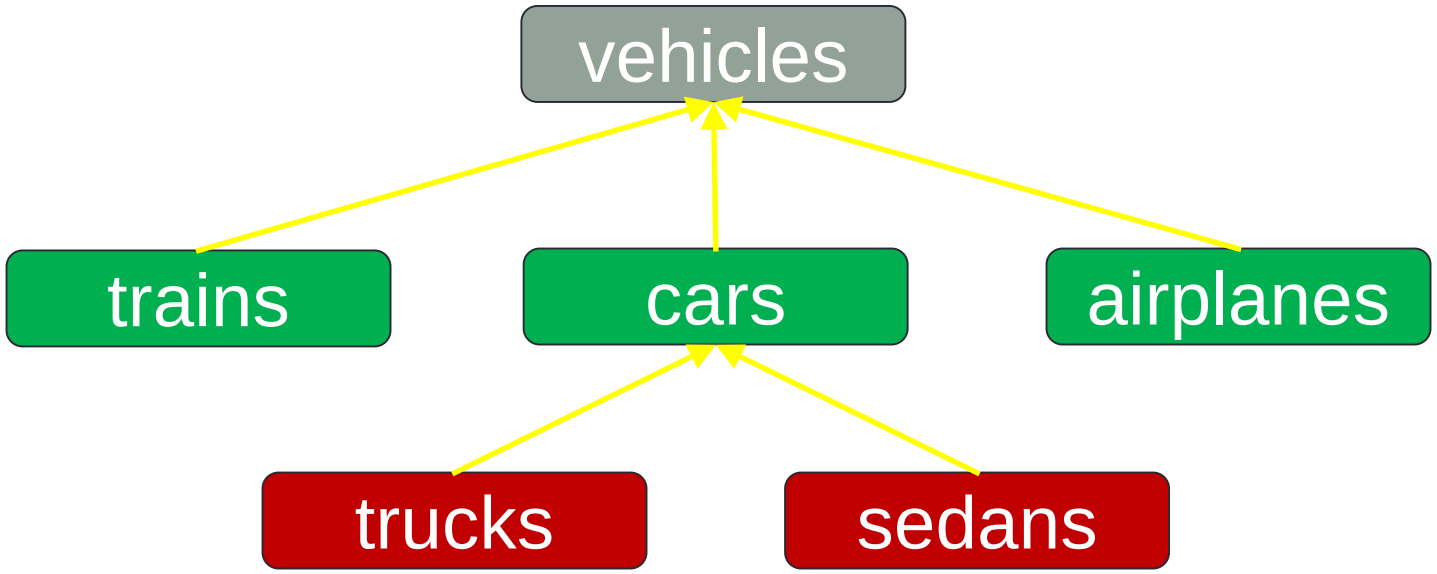
Agenda

- Concepts of Inheritance (Chapter 19.1~19.3)
  - as an is-a relationship
- How to publicly derive one class from another (Chapter 19.3)
  - constitution of a derived class
  - protected access
- Constructors/Destructors of Derived Class (Chapter 19.4~19.5)
  - public, private and protected inheritance
- Assignment operator and copy constructors

Fundamentals of Inheritance

- Code reusability is the key to enable object-oriented programming (OOP)
  - C++ implements inheritance
  - establish parent-child relationship among existing classes
- Inheritance (a.k.a. derivation) is a relationship between classes
  - one class inherits the properties (attributes and behaviors) of another class
  - Is an *is-a* relationship
  - enable a hierarchy of classes (from most general to most specific)

Hierarchy of Vehicles



- A vehicle contains its weight, seats, loading, etc..
- A car can be defined by either a redefinition or adding more properties based on vehicle

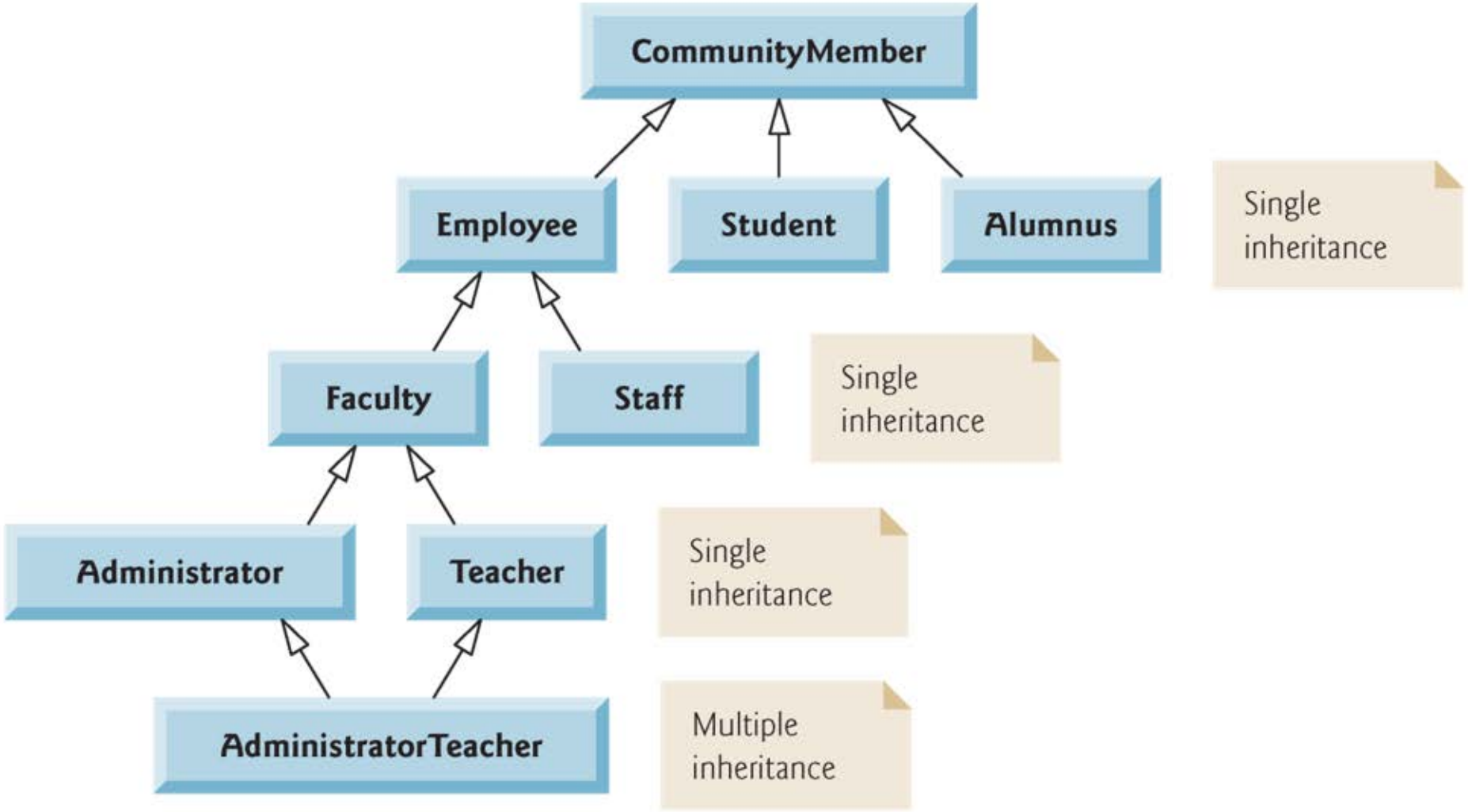
Class Inheritance

- A new class can be derived from an old class
  - inherit attributes/methods from the old class
  - create new attributes/methods
  - called *inheritance* and *derivation*
- Base (super, parent) class vs. derived (sub, child) class
  - base class is the class to be inherited
  - derived class is the new class on top of a base class
  - a base class has multiple derived classes

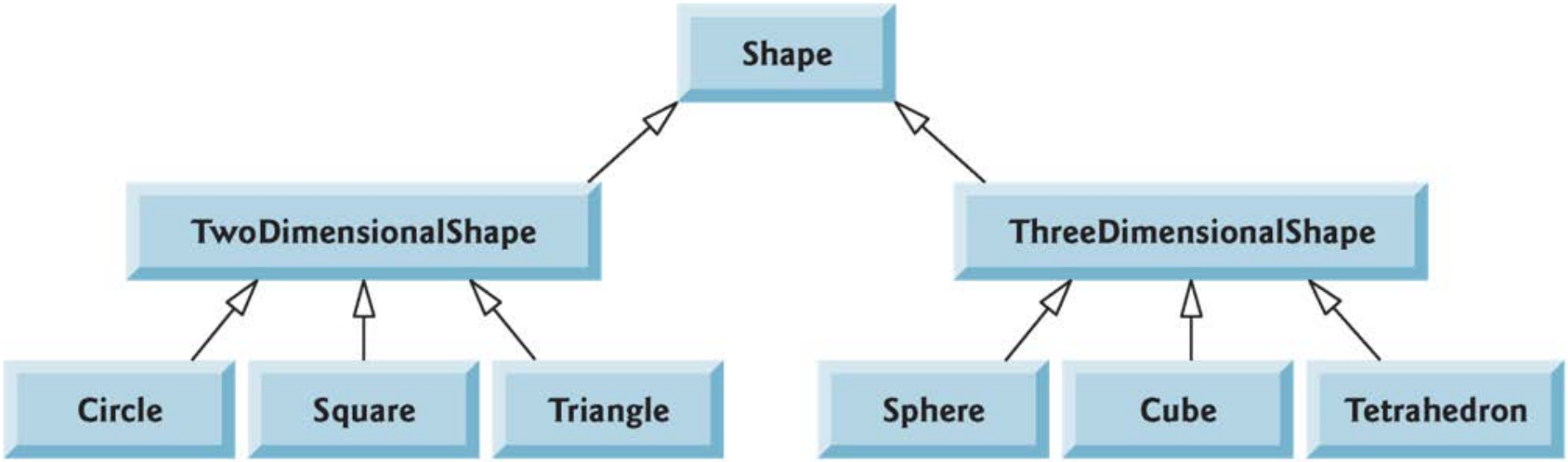
Another Inheritance Example (1)

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

Another Inheritance Example (2)



Another Inheritance Example (3)



Why We Need Inheritance?

- Consider two classes
  - Commission Employee
    - Paid a percentage of their sales
  - Base-salaried Commission Employee
    - Paid a *base salary* plus percentage of their sales
  - Common parts:
    - First name
    - Last name
    - Social security number
    - Commission rate
    - Gross sales amount

CommissionEmployee.h

```
// Fig. 19.4: CommissionEmployee.h
// CommissionEmployee class definition
#ifndef COMMISSION_H
#define COMMISSION_H
#include <string>
using namespace std;
class CommissionEmployee
{
public:
    CommissionEmployee (const string &, const string &,
const string &, double = 0.0, double = 0.0);
    void setFirstName(const string &); // set first name
    string getFirstName() const;
    void setLastName(const string &); // set last name
    string getLastName() const;
    void setSocialSecurityNumber(const string &); // set
SSN
    string getSocialSecurityNumber() const;
```

CommissionEmployee.h (cont.)

```
    void setGrossSales(double);  // set gross sales amount
    double getGrossSales() const;
    void setCommissionRate(double); // set rate (%)
    double getCommissionRate() const;
    double earnings() const; // calculate earnings
    void print() const;
private:
    string firstName;
    string lastName;
    string socialSecurityNumber;
    double grossSales;    // gross weekly sales
    double comissionRate; // commission percentage
}; // end class CommissionEmployee
#endif
```

CommissionEmployee.cpp

```
// Fig. 19.5: CommissionEmployee.cpp
#include <iostream>
#include "CommissionEmployee.h"
using namespace std;

// constructor
CommissionEmployee::CommissionEmployee(const string &first,
const string $last, const string &ssn, double sales,
double rate)
{
    firstName = first;
    lastName = last;
    socialSecurityNumber = ssn;
    setGrossSales(sales); //validate and store gross sales
    setCommissionRate(rate);
}
```

CommissionEmployee.cpp (cont.)

```
// set first name
void CommissionEmployee::setFirstName(const string &first)
{
    firstName = first;
}
// return first name
string CommissionEmployee::getFirstName() const
{
    return firstName;
}
…
void CommissionEmployee::setGrossSales(double sales)
{
    grossSales = (sales < 0.0) ? 0.0 : sales;
}
double CommissionEmployee::getGrossSales() const
{
    return grossSales;
}
```

CommissionEmployee.cpp (cont.)

```
…
// calculate earnings
double CommissionEmployee::earnings() const
{
    return commissionRate * grossSales;
}
// print CommissionEmployee object
void CommissionEmployee::print() const
{
    cout << "Commission employee: " << firstName << ' '
         << lastName << "\nsocial security number: "
         << socialSecurityNumber << "\ngross sales: "
         << grossSales << "\ncommission rate: "
         << commissionRate;
}
```

Fig19_06.cpp

```
// Fig. 19.6: fig19_06.cpp
// Testing class CommissionEmployee.
#include <iostream>
#include <iomanip>
#include "CommissionEmployee.h" // CommissionEmployee
class definition
using namespace std;

int main()
{
    // instantiate a CommissionEmployee object
    CommissionEmployee employee(
      "Sue", "Jones", "222-22-2222", 10000, .06 );
    // set floating-point output formatting
    cout << fixed << setprecision( 2 );
```

Fig19_06.cpp (cont.)

```
    // get commission employee data
    cout << "Employee information obtained by get
functions: \n"
         << "\nFirst name is " << employee.getFirstName()
         << "\nLast name is " << employee.getLastName()
         << "\nSocial security number is "
         << employee.getSocialSecurityNumber()
         << "\nGross sales is " << employee.getGrossSales()
         << "\nCommission rate is "
         << employee.getCommissionRate() << endl;

    employee.setGrossSales( 8000 ); // set gross sales
    employee.setCommissionRate( .1 ); // set commission
rate
    cout << "\nUpdated employee information output by
print function: \n" << endl;
    employee.print();
```

Fig19_06.cpp (cont.)

```
    // display the employee's earnings
    cout << "\n\nEmployee's earnings: $"
         << employee.earnings() << endl;
} // end main
```

```
Employee information obtained by get functions:

First name is Sue
Last name is Jones
Social security number is 222-22-2222
Gross sales is 10000.00
Commission rate is 0.06

Updated employee information output by print function:

commission employee: Sue Jones
social security number: 222-22-2222
gross sales: 8000.00
commission rate: 0.10

Employee's earnings: $800.00
```

BasePlusCommissionEmployee.h

```
// Fig. 19.7: BasePlusCommissionEmployee.h
// CommissionEmployee class definition
#ifndef BASEPLUS_H
#define BASEPLUE_H
#include <string>
using namespace std;
class BasePlusCommissionEmployee
{
public:
    BasePlusCommissionEmployee (const string &, const
string &, const string &, double = 0.0, double = 0.0,
double = 0.0);
    void setFirstName(const string &); // set first name
    string getFirstName() const;
    void setLastName(const string &); // set last name
    string getLastName() const;
    void setSocialSecurityNumber(const string &); // set
SSN
    string getSocialSecurityNumber() const;
```

BasePlusCommissionEmployee.h (cont.)

```
    void setGrossSales(double);  // set gross sales amount
    double getGrossSales() const;
    void setCommissionRate(double); // set rate (%)
    double getCommissionRate() const;
    void setBaseSalary(double); // set base salary
    double getBaseSalary() const;
    double earnings() const; // calculate earnings
    void print() const;
private:
    string firstName;
    string lastName;
    string socialSecurityNumber;
    double grossSales;    // gross weekly sales
    double comissionRate; // commission percentage
    double baseSalary;    // base salary
}; // end class BasePlusCommissionEmployee
#endif
```

BasePlusCommissionEmployee.cpp

```
// Fig. 19.8: BasePlusCommissionEmployee.cpp
#include <iostream>
#include "BasePlusCommissionEmployee.h"
using namespace std;

// constructor
BasePlusCommissionEmployee::BasePlusCommissionEmployee(con
st string &first, const string $last, const string &ssn,
double sales, double rate, double salary)
{
    firstName = first;
    lastName = last;
    socialSecurityNumber = ssn;
    setGrossSales(sales);
    setCommissionRate(rate);
    setBaseSalary(salary);
}
```

BasePlusCommissionEmployee.cpp (cont.)

```
void BasePlusCommissionEmployee::setFirstName(const string
&first)
{
    firstName = first;
}
string BasePlusCommissionEmployee::getFirstName() const
{
    return firstName;
}
…
void BasePlusCommissionEmployee::setBaseSalary(double
salary)
{
    baseSalary = (salary < 0.0) ? 0.0 : salary;
}
double BasePlusCommissionEmployee::getBaseSalary () const
{
    return baseSalary ;
}
```

BasePlusCommissionEmployee.cpp (cont.)

```
// calculate earnings
double BasePlusCommissionEmployee::earnings() const
{
    return baseSalary + commissionRate * grossSales;
}
// print BasePlusCommissionEmployee object
void BasePlusCommissionEmployee::print() const
{
    cout << "Commission employee: " << firstName << ' '
         << lastName << "\nsocial security number: "
         << socialSecurityNumber << "\ngross sales: "
         << grossSales << "\ncommission rate: "
         << commissionRate
         << "base salary: " << baseSalary;
}
```

Fig19_09.cpp

```
// Fig. 19.9: fig19_09.cpp
// Testing class BasePlusCommissionEmployee.
#include <iostream>
#include <iomanip>
#include "BasePlusCommissionEmployee.h"
using namespace std;

int main()
{
    // instantiate a BasePlusCommissionEmployee object
    BasePlusCommissionEmployee employee(
      "Sue", "Jones", "222-22-2222", 10000, .06, 300 );
    // set floating-point output formatting
    cout << fixed << setprecision( 2 );
```

Fig20_09.cpp (cont.)

```
    // get commission employee data
    cout << "Employee information obtained by get
functions: \n"
         << "\nFirst name is " << employee.getFirstName()
         << "\nLast name is " << employee.getLastName()
         << "\nSocial security number is "
         << employee.getSocialSecurityNumber()
         << "\nGross sales is " << employee.getGrossSales()
         << "\nCommission rate is "
         << employee.getCommissionRate()
         << "\nBase salary is " << employee.getBaseSalary()
         << endl;

    employee.setBaseSalary( 1000 ); // set base salary
    cout << "\nUpdated employee information output by
print function: \n" << endl;
    employee.print();
```

Fig19_09.cpp (cont.)

```
    // display the employee's earnings
    cout << "\n\nEmployee's earnings: $"
         << employee.earnings() << endl;
} // end main
```

```
Employee information obtained by get functions:

First name is Bob
Last name is Lewis
Social security number is 333-33-3333
Gross sales is 5000.00
Commission rate is 0.04
Base salary is 300.00

Updated employee information output by print function:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 1000.00

Employee's earnings: $1200.00
```

Declaration of Inheritance

- Base class defines
  - inheritable properties for derived classes
  - non-inheritable private properties of its own
- General form of a derived class

```
class ⟨Derived Class⟩:
      ⟨Access Specifier 1⟩⟨Base Class 1⟩,
      ⟨Access Specifier 2⟩⟨Base Class 2⟩,
      ...
{
    //specify properties of its own
};
```

  - only one "**:**"
  - if >=2 base classes ⇨ multiple inheritance
  - default as private access

Example of Inheritance

- Class `CCircle` and its definition

```
class CCircle {
protected:
    double radius;
public:
    CCircle(double r=1.0):radius(r) { ; }
    void setR(double r=1.0) { radius = r; }
    double calVol() const {
        return (PI*radius*radius);
    }
    void showVol() const {
        cout << "radius=" << radius << " ";
        cout << "volume=" << calVol() <<
endl;
    }
};
```

Example of Inheritance (cont.)

- Class `CCylinder` and its definition

```
class CCylinder : public CCircle {
protected:
    double length; //add one data member
public:
    CCylinder(double r=5.0, double l=1.0) :
        CCircle(r),length(l) { ; }
    void setRL(double r=5.0, double l=1.0){
        radius = r; length = l;
    }
    //calculate the volume of a cylinder
    double calVol() const {
        return (CCircle::calVol()*length);
    }
    //inherit showVol() from CCircle
    void displayVol() const {
        cout << "d radius=" << radius << " ";
        cout << "d volume=" << calVol() << endl;
    }
};
```

Example of Inheritance (cont.)

- `main` program

```
int main() {
    CCircle cr1, cr2(4);
    CCylinder cy1, cy2(2,3);

    cr1.setR(2);
    cr1.showVol(); cr2.showVol();
    cy1.setRL(3);
    cy1.showVol(); cy1.displayVol();
    cy2.showVol(); cy2.displayVol();

    cr1 = cy1; //Q: what happen?
    cr1.showVol();

    return 0;
}
```

Example of Inheritance (cont.)

- screen output

```
radius=2 volume=12.56
radius=4 volume=50.24
radius=3 volume=28.26
d radius=3 d volume=28.26
radius=2 volume=12.56
d radius=2 d volume=37.68
radius=3 volume=28.26
```

Derived Classes

- A derived class can
  - add *new* member (either data or function)
  - *overload* or *override* (*redefine*) member functions in the base class
  - change the accessibility of members of the base class in the derived class
- A derived class cannot inherit
  - *constructors/destructors* of the base class
  - *friends* of the base class
  - *static* data members and *static* member function

Choose Access Specifier

- Different ways of inheritance affects how the members in the base class can be accessed from the derived class

- The access specifier defines the way of inheritance
  - public inheritance (public derivation)
  - private inheritance (private derivation)
  - protected inheritance (protected derivation)

`public` Inheritance

- `public` inheritance is the most common one
  - maintain the accessibility of the base class
- Public and protected members of the base class remain the same accessibility in the derived class
  - member functions of the derived class *can access public and protected members* of the base class directly
  - *cannot* access the *private* members of the base class
- An object of the derived class can only access the public members of the base class

`public` Inheritance (cont.)

- `public` inheritance defines
  - the inheriting class as *public* derived class
  - the inherited class as *public* base class
- Accessibility of members in the base class

members in public base class	accessibility in public derived class
`public`	**`public`**
`private`	**`inaccessible`**
`protected`	**`protected`**

  - why cannot access private members?

    ⇨ maintain encapsulation of the base class

Example of public Inheritance

```
class CPoint        //base class: class CPoint
{
    double x, y;
public:
    CPoint(double a=0, double b=0): x(a), y(b) {}
    void SetPoint(double a=0, double b=0) {
        x = a; y = b;
    }
    void MovePoint(double dx, double dy) {
        x +=dx; y += dy;
    }
    double GetX() { return x; }
    double GetY() { return y; }
};
```

Example of public Inheritance (cont.)

```
class CRect: public CPoint { //derived class
//private:
    double h, w; //new private data
public:
    //inherit member functions from CPoint
    double GetH() const { return h; }
    double GetW() const { return w; }
    CRect(double a, double b, double c, double d):
        h(c), w(d) { SetPoint(a,b); }
    void SetRect(double a, double b, double c,
                 double d) {
    SetPoint(a,b);
    h = c; w = d; }
};
```

Example of public Inheritance (cont.)

```
int main()
{
    CRect cr1;
    cr1.SetRect(2,3,20,10);
    //visit base public members through
    //a derived object
    cr1.MovePoint(3,2);
    cout << cr1.GetX() << ','
         << cr1.GetY() << ','
          << cr1.GetH() << ','
          << cr1.GetW() << endl;
    return 0;
}
```

`private` Inheritance

- Public and protected members of the base class become *private* in the derived class
  - member functions of the derived class still can access public and protected member of the base class directly
  - *cannot* access the *private* members of the base class

- An object of the derived class cannot access any member (including public, protected and private) of the base class

`private` Inheritance (cont.)

- `private` inheritance defines
  - the inheriting class as *private* derived class
  - the inherited class as *private* base class
- Accessibility of members in the base class

members in private base class	accessibility in private derived class
`public`	**`private`**
`private`	**`inaccessible`**
`protected`	**`private`**

  - need not memorize the rules
    - inherit public/protected members as private

Example of `private` Inheritance

```
class CPoint        //base class: class CPoint
{
    double x, y;
public:
    CPoint(double a=0, double b=0): x(a), y(b) {}
    void SetPoint(double a=0, double b=0) {
        x = a; y = b;
    }
    void MovePoint(double dx, double dy) {
        x +=dx; y += dy;
    }
    double GetX() { return x; }
    double GetY() { return y; }
};
```

Example of `private` Inheritance (cont.)

```
class CRect: private CPoint //derived class
{
    double h, w; // new private data
public:
    double GetH() const { return h; }
    double GetW() const { return w; }
    // derived member functions can access
    // base member functions
    void SetRect(double a, double b, double c,
                 double d) {
        CPoint::SetPoint(a, b);
        h = c; w = d; }
    void MoveRect(double dx, double dy) {
        CPoint::MovePoint(dx, dy); }
};
```

Example of `private` Inheritance (cont.)

```
int main()
{
    CRect cr2;
    cr2.SetRect(2,3,20,10);
    //a derived object cannot access any
    //member of the base class
    cr2.MovePoint(3,2); //illegal!!!
    //use its own member function
    cr2.MoveRect(3,2);  //legal
    cout << cr2.GetX() << ',' //illegal!!!
         << cr2.GetY() << ',' //illegal!!!
         << cr2.GetH() << ','
         << cr2.GetW() << endl;
    return 0;
}
```

`protected` Inheritance

- Public and protected members of the base class become *protected* in the derived class
  - member functions of the derived class still can access public and protected member of the base class directly
  - *cannot* access the *private* members of the base class

- An object of the derived class cannot access any member (including public, protected and private) of the base class

`protected` Inheritance (cont.)

- `protected` inheritance defines
  - the inheriting class as *protected* derived class
  - the inherited class as *protected* base class
- Accessibility of members in the base class

members in protected base class	accessibility in protected derived class
`public`	`protected`
`private`	`inaccessible`
`protected`	`protected`

  - need not memorize the rules
    - inherit public/protected members as protected

`protected` Members

- For the base class where a protected member is defined,
  - the protected member works like a private member outside the base class
- For the derived class who inherited the protected member,
  - the protected member works like other public members
- For an object of the base class, protected members cannot be accessed

Example of `protected` Members

```
class A
{
private: double x;
protected: long y;
public: short z;
};
class B: public A {
public:
    void f() { //what's wrong below?
        x = 3.0; y = 4; z = 2; }
};
int main() {
    B b;
    b.x = 1.3; //what's wrong?!
    b.y = 2;   //what's wrong?!
    b.z = 5;
}
```

Example of `protected` Inheritance

```
class CBase
{
    double x;
protected:
    int GetX() { return x; }
public:
    void SetX(double a) { x = a; }
    void ShowX() { cout << x << endl; }
};
class CDerived: protected CBase {
    double y; //its own private member
public:
    void SetY(double b) { y = b; }
    //visit protected member of CBase
    void SetY() { y = GetX(); }
    void ShowY() { cout << y << endl; }
};
```

Example of `protected` Inheritance (cont.)

```
int main()
{
    CDerived cd;

    cd.SetX(15);  //what happen?
    cd.SetY(20);

    cd.ShowX();   //what happen?
    cd.ShowY();

    return 0;
}
```

- What's wrong with these calls?

Compare Different Inheritances

- Member functions of the derived class in different inheritances

inheritance method	accessibility of member functions in the derived class
public	can access public/protected members of the base class
private	can access public/protected members of the base class
protected	can access public/protected members of the base class

Compare Different Inheritances (cont.)

- Objects of the derived class in different inheritances

inheritance method	accessibility of objects
public	**can** access public members of both the derived and base classes
private	**cannot** access any member of the base class
protected	**cannot** access any member of the base class

Constructors of Derived Class

- Data members of a derived class consists of
  - old data members of the base class +
  - new data members of the derived class
- Constructor/destructor cannot be inherited

  ⇨ need to initialize *both* old and new data members by the constructor of the derived class
- Two steps to define a constructor of the derived
  - call the constructor of the base class for old data members
  - set up new data member properly

Constructors of Single Inheritance

- Single inheritance refers that the derived class has only one base class

- Format: a base class CB and a derived class CD

```
CD(⟨CD_para_list⟩):CB(⟨CB_para_list⟩) {
    //initialize new data members in CD
}
```

  - ⟨CD_para_list⟩ = ⟨CB_para_list⟩ + new data members
  - call CB's constructor

Example of Single Inheritance

```
class CPsn //base class
{
    string name;
    char sex;
public:
    CPsn(string sname, char s) {
        name = sname;
        sex = s;
    }
    ~CPsn() {}
    string getName (){return name;}
    char getSex (){return sex;}
};
```

Example of Single Inheritance (cont.)

```
class CStu : public CPsn {//derived class
    int age; string addr;
public:
    CStu(string sname, char s, int a,
         string ad): CPsn(sname, s) {
        age = a; addr = ad;
    }
    ShowCStu() {
        cout << "name=" << getName() <<
 endl;
        cout << "sex=" << getSex() << endl;
        cout << "age=" << age << endl;
        cout << "address=" << addr << endl;
    }
    ~CStu(){}
};
```

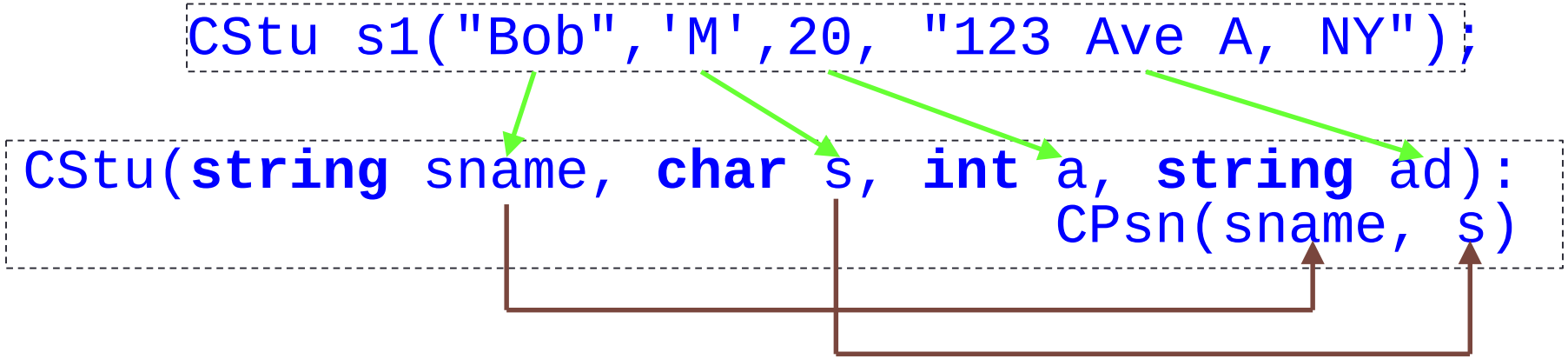
Example of Single Inheritance (cont.)

```
int main() {
    CStu s1("Bob",'M',20,"123 Ave A, NY");
    CStu s2("Jan",'F',18,"101 Ave Q, BJ");
    s1.ShowCStu();
    s2.ShowCStu();

    return 0;
}
```

Example of Single Inheritance (cont.)

- s1 assigns 5 arguments



- Alternative definition for the constructor

```
CStu(string sname, char s, int a, string ad):
            CPsn(sname, s), age(a), addr(ad) {}
```

- Release an `CStu` object
  - first call `~CStu()`, then call `~CPsn()`

Destructor of Derived Class

- Destructor of the derived class also cannot be inherited
  - need to be defined in the derived class
  - automatically (implicitly) call the destructor of the base class
- To invoke the destructor of the derived class, *opposite* of how constructors are called

- Example: C is derived from class B, which is derived from class A
  - 1st: class C destructor is called
  - 2nd: class B destructor is called
  - 3rd: class A destructor is called

CommissionEmployee.h

```
// Fig. 19.12: CommissionEmployee.h
// CommissionEmployee class definition
#ifndef COMMISSION_H
#define COMMISSION_H
#include <string>
using namespace std;
class CommissionEmployee
{
public:
    CommissionEmployee (const string &, const string &,
const string &, double = 0.0, double = 0.0);
    void setFirstName(const string &); // set first name
    string getFirstName() const;
    void setLastName(const string &); // set last name
    string getLastName() const;
    void setSocialSecurityNumber(const string &); // set
SSN
    string getSocialSecurityNumber() const;
```

CommissionEmployee.h (cont.)

```
    void setGrossSales(double);  // set gross sales amount
    double getGrossSales() const;
    void setCommissionRate(double); // set rate (%)
    double getCommissionRate() const;
    double earnings() const; // calculate earnings
    void print() const;
private:
    string firstName;
    string lastName;
    string socialSecurityNumber;
    double grossSales;    // gross weekly sales
    double comissionRate; // commission percentage
}; // end class CommissionEmployee
#endif
```

CommissionEmployee.cpp

```
// Fig. 19.14: CommissionEmployee.cpp
#include <iostream>
#include "CommissionEmployee.h"
using namespace std;

// constructor
CommissionEmployee::CommissionEmployee(const string &first,
const string $last, const string &ssn, double sales,
double rate): firstName(first), lastName(last),
socialSecurityNumber(ssn)
{
    setGrossSales(sales);
    setCommissionRate(rate);
}
```

CommissionEmployee.cpp (cont.)

```
// set first name
void CommissionEmployee::setFirstName(const string &first)
{
    firstName = first;
}
// return first name
string CommissionEmployee::getFirstName() const
{
    return firstName;
}
…
void CommissionEmployee::setGrossSales(double sales)
{
    grossSales = (sales < 0.0) ? 0.0 : sales;
}
double CommissionEmployee::getGrossSales() const
{
    return grossSales;
}
```

CommissionEmployee.cpp (cont.)

```
// calculate earnings
double CommissionEmployee::earnings() const
{
    return getCommissionRate()* getGrossSales();
}
// print CommissionEmployee object
void CommissionEmployee::print() const
{
    cout << "Commission employee: " << getFirstName()
         << ' ' << getLastName()
         << "\nsocial security number: "
         << getSocialSecurityNumber() << "\ngross sales: "
         << getGrossSales << "\ncommission rate: "
         << getCommissionRate();
}
```

BasePlusCommissionEmployee.h

```
// BasePlusCommissionEmployee.h
#ifndef BASEPLUS_H
#define BASEPLUE_H
#include <string>
#include "CommissionEmployee.h"
class BasePlusCommissionEmployee:public CommissionEmployee
{
public:
    BasePlusCommissionEmployee (const string &, const
string &, const string &, double = 0.0, double = 0.0,
double = 0.0);
    void setBaseSalary(double); // set base salary
    double getBaseSalary() const;
    double earnings() const; // calculate earnings
    void print() const;
private:
    double baseSalary;    // base salary
}; // end class BasePlusCommissionEmployee
#endif
```

BasePlusCommissionEmployee.cpp

```
// Fig. 19.15: BasePlusCommissionEmployee.cpp
#include <iostream>
#include "BasePlusCommissionEmployee.h"
using namespace std;

// constructor
BasePlusCommissionEmployee::BasePlusCommissionEmployee(con
st string &first, const string $last, const string &ssn,
double sales, double rate, double salary)
:CommissionEmployee(first, last, ssn, sales, rate)
{
    setBaseSalary(salary);
}
```

BasePlusCommissionEmployee.cpp (cont.)

```
void BasePlusCommissionEmployee::setBaseSalary(double
salary)
{
    baseSalary = (salary < 0.0) ? 0.0 : salary;
}
double BasePlusCommissionEmployee::getBaseSalary () const
{
    return baseSalary ;
}
// calculate earnings
double BasePlusCommissionEmployee::earnings() const
{
    return getBaseSalary()+CommissionEmployee::earnings();
}
```

BasePlusCommissionEmployee.cpp (cont.)

```
// print BasePlusCommissionEmployee object
void BasePlusCommissionEmployee::print() const
{
    cout << "base-salaried ";
    // invoke CommissionEmployee's print function
    CommissionEmployee::print();

    cout << "base salary: " << getBaseSalary();
}
```

Assignment and Copy Constructors

- Overloading assignment operators and copy constructors also cannot be not inherited!
  - but can be used in derived-class definitions
  - typically must be used
  - similar to how derived-class constructor invoke base-class constructor
- Example of assignment operator (=)

```
CDerived& CDerived::operator=(
                    const CDerived& rHand) {
    CBase::operator=(rHand);
    //...
}
```

Example of Copy Constructors

- If not defined, generate a default one
- After `:` is an invocation of base copy constructor
  - set inherited data members of derived-class object being created
  - `obj` is of type `CDerived`
  - `obj` is also of type `CBase` ⇨ argument is valid ⇨ involve class conversion

```
CDerived::CDerived(const CDerived& obj):
                                    CBase(obj) {
    //...
}
```

Conversion of Base/Derived Classes

- An object of the derived class can be used as an object of the base class, but not in reverse
  - can assign the object of the derived class to the object of the base class
  - can use the object of the derived class to initialize a reference to the base class
  - point to the object of the derived class can be assigned to pointer to the object of the base class
- Through the base class, the pointer can only be used onto the inherited members

Example of Class Conversion

```
class B0
{
public: void Show() {
    cout << "B0::Show()" << endl; }
};
class B1 : public B0
{
public: void Show() {
    cout << "B1::Show()" << endl; }
};
class D2 : public B1
{
public: void Show() {
    cout << "D2::Show()" << endl; }
};
void fun(B0* ptr) {
    ptr->Show();
}
```

Example of Class Conversion (cont.)

```
int main() {
    B0 r0;
    B1 r1;
    D2 r2;

    B0 *p;
    p = &r0; //B0 pointer to B0 object
    fun(p);
    p = &r1; //B0 pointer to B1 object
    fun(p);
    p = &r2; //B0 pointer to D2 object
    fun(p);

    return 0;
}
```

```
B0::Show()
B0::Show()
B0::Show()
```

Summary

- Inheritance provides code reuse
  - allow one class to “derive” from another, adding features
- Derived class objects inherit members of base class
  - may add members
- Private member variables in base class cannot be accessed “by name” in derived class
- Private member functions are not inherited

Summary (cont.)

- Can redefine inherited member functions
  - to perform differently in derived class
- Protected members in base class
  - Can be accessed “by name” in derived class member functions
- Overloaded assignment operators cannot be inherited
  - but can be invoked from derived class
- Constructors are not inherited
  - but ban be invoked from derived class’s constructor

References

- Paul Deitel and Harvey Deitel, “C How to Program” Sixth Edition
  - Chapter 19
- Paul Deitel and Harvey Deitel, “C++ How to Program (late objects version)” Seventh Edition
  - Chapter 12: Object-Oriented Programming: Inheritance
- W. Savitch, “Absolute C++,” Fourth Edition
  - Chapter 14