

LAB 9: Inheritance

Bank Account

Rung-Bin Lin

International Bachelor Program in Informatics
Yuan Ze University

4/28/2021

Objectives

- Learn how to use inheritance for software reuse.

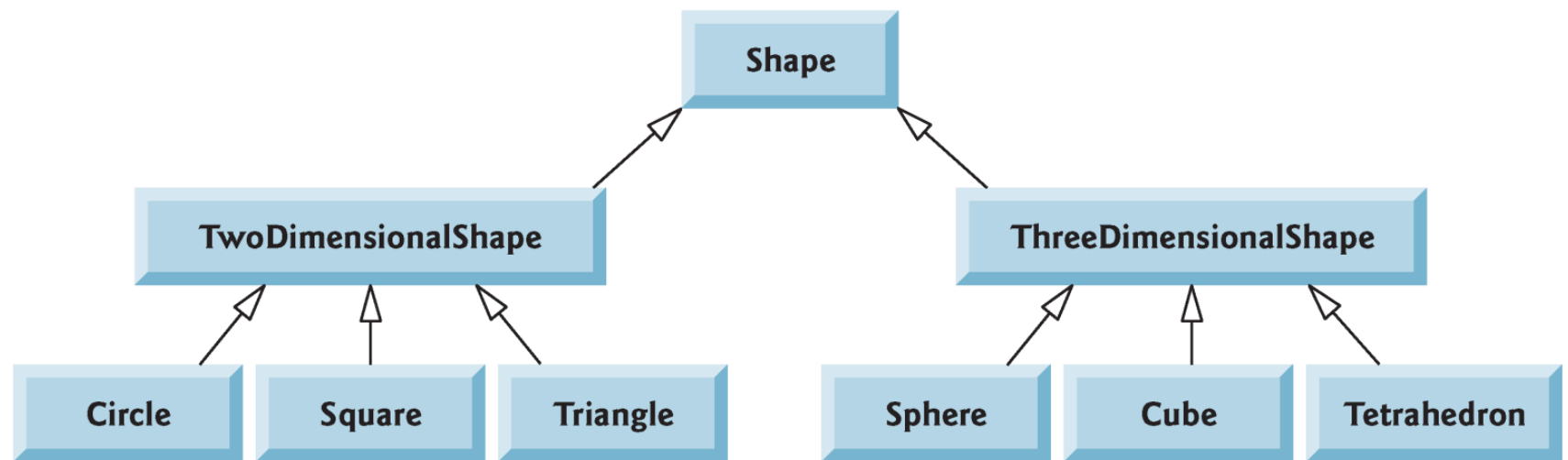


Fig. 12.3 | Inheritance hierarchy for Shapes.

Concept of Inheritance

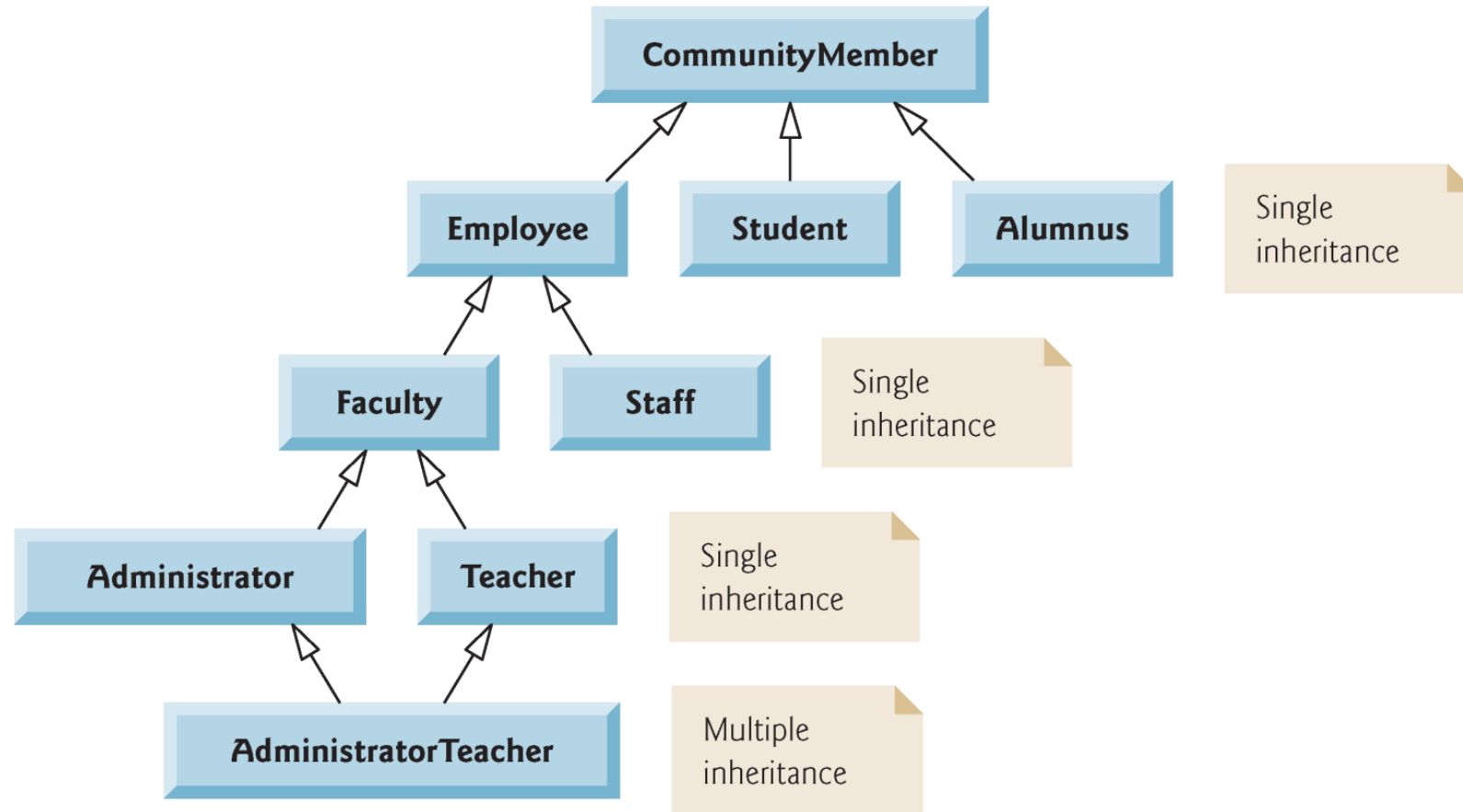


Fig. 12.2 | Inheritance hierarchy for university `CommunityMembers`.

Inheritance

• Inheritance is a form of software reuse.

- Create a class that absorbs an existing class's data and behaviors and enhance them with new capabilities.
- The existing class being used to create a new class is called **base class**.
- The newly created class is called **derived class**.

• Three types of inheritance

- **public**: most often used, a derived class's member function **not allowed** to directly access a private data member of base class.
- **protected**: more restricted, seldom used
- **private**: most restricted, rarely used.

• Inheritance is an “is” relationship

- A derived class's object is a base class's object, but a base class's object is usually not a derived class's object.
- For example, tiger is an animal, but an animal is usually not a tiger. So animal is the base class and tiger is a derived class.
- **Base class** defines more **general** objects whereas **derived class** defines more **specific** objects.

Things Not Inherited

- **Friend functions, destructors and constructors of base class are not inherited.**

Class Members

- **Three kinds of members**
 - **Public members**
 - **Protected members**
 - **Private members**
- **Members can be data or functions**

Member Access Issues with Public Inheritance

- A base class's **public members** are accessible within its body and anywhere that the program has a handle to an object of that class or one of its derived class.
 - A handle means either a name, a reference, or a pointer to an object.
- A base class's **protected members** can be accessed within its body, by members and friends of that base class, and by members and friends of any class derived from that based class.
- A base class's **private members** can be accessed within its body, by members and friends of that base class.

An Example of Public Inheritance

```
class CommissionEmployee // Base class
{
public:
    CommissionEmployee( const string &, const string &, const string &,
        double = 0.0, double = 0.0 );
    void setFirstName( const string & ); // set first name
    string getFirstName() const; // return first name
    void setLastName( const string & ); // set last name
    string getLastName() const; // return last name
    ...
    ...
    ...
    double earnings() const; // calculate earnings
    void print() const; // print CommissionEmployee object
private:
    string firstName;
    string lastName;
    string socialSecurityNumber;
    double grossSales; // gross weekly sales
    double commissionRate; // commission percentage
}; // end class CommissionEmployee
```


Derived class

```
#include <string> // C++ standard string class
#include "CommissionEmployee.h" // CommissionEmployee class declaration
using namespace std;
```

```
class BasePlusCommissionEmployee : public CommissionEmployee
```

```
{
```

```
public:
```

```
    firstName    lastName
```

```
    BasePlusCommissionEmployee( const string &, const string &,
                                const string &, double = 0.0, double = 0.0, double = 0.0 );
```

```
    socialSecurityNumber
```

```
    grossSales
```

```
    commissionRate
```

```
    baseSalary
```

```
    void setBaseSalary( double ); // set base salary
```

```
    double getBaseSalary() const; // return base salary
```

```
    double earnings() const; // calculate earnings
```

```
    void print() const; // print BasePlusCommissionEmployee object
```

```
private:
```

```
    double baseSalary; // base salary
```

```
}; // end class BasePlusCommissionEmployee
```

Constructor of Derived class

```
BasePlusCommissionEmployee::BasePlusCommissionEmployee(  
    const string &first, const string &last, const string &ssn,  
    double sales, double rate, double salary )  
    // explicitly call base-class constructor  
    : CommissionEmployee( first, last, ssn, sales, rate )  
{  
    setBaseSalary( salary ); // validate and store base salary  
} // end BasePlusCommissionEmployee
```

- Call the constructor of base class to initialize data members

Redefining a Member Function

- We can redefine member function in a derived class and use it for the application.

// calculate earnings

```
double BasePlusCommissionEmployee::earnings() const
{
    return getBaseSalary() + CommissionEmployee::earnings();
} // end function earnings
```

- Here we use the **earning** member function of the **base class** for redefining the **earning** member function of the **derived class**.

Suggest you study Fig. 12.17 through 12.21.

Lab 9: Bank Account

- Given the base class **Account**, you are asked to develop two **derived** classes **SavingAccount** and **CheckingAccount** respectively for saving and checking accounts using **public inheritance**.
- You should not modify the **main()** function.

Base Class Definition: Account

```
class Account
{
public:
    Account(double = 0.0, double =0.0);
    void credit(double =0.0); // Deposit money >0
    bool debit(double = 0.0); // Withdraw money>0
    double getBalance(); // Get balance
    double calculateInterest(); // Return interest and add
the interest to the balance
    void print(); // print balance and interest rate
private:
    double balance; // Account balance >=0
    double interestRate; // Interest rate >=0
};
```

Member Functions of Account

```
Account::Account(double bal,
double iRate )
{
    if (bal >0)
        balance = bal;
    else
        balance = 0;
    if(iRate >0)
        interestRate = iRate;
    else interestRate = 0;
}

void Account::credit(double
depos)
{
    if(depos > 0)
        balance = balance + depos;
}

double Account::getBalance()
{
    return balance;
}
```

```
bool Account::debit(double withdw)
{
    if(withdw >0 && withdw <= balance)
    {
        balance = balance - withdw;
        return true;
    }
    else if(withdw > balance)
    {
        cout << " Debit amount exceeded
account balance." << endl;
        return false;
    }
    return false;
}

void Account::print()
{
    cout << " Balance: " << balance <<
endl;
    cout << " Interest rate: " <<
interestRate << endl;
}
```

SavingAccount Class

- For your convenience, a summary of SavingAccount class is given below.

```
class SavingAccount {
```

```
public:
```

```
    SavingAccount(double = 0.0, double = 0.0, double = 3.0);
```

```
    // parameters: balance, interest rate, transaction fee.
```

```
    bool debit(double =0.0);
```

```
    void print();
```

```
private:
```

```
    double transactFee; // transaction fee for withdrawing
```

```
};
```

- **transactFee** is an amount of money paid to the bank by a saving account if a withdraw transaction is made on a saving account. No transaction fee is charged for deposition.
- **debit()** can only be done if balance remains positive after withdrawing. There is no transaction fee if a transaction fails.
- You should implement the member functions. No other member functions and data should be added.

CheckingAccount Class

```
class CheckingAccount {
```

```
public:
```

```
    CheckingAccount(double = 0.0, double =0.0, double =3.0, double = 2.0);
```

```
    // Parameters: balance, interest rate, transaction fee for withdraw, transaction fee for deposition
```

```
    bool debit(double =0.0); // return true if it can be done successfully.
```

```
    void credit(double =0.0);
```

```
    void print();
```

```
private:
```

```
    double transactFeeW; // withdraw
```

```
    double transactFeeD; // Deposit
```

```
}
```

- There is a transaction fee respectively for withdrawing and depositing if a transaction succeeds. Otherwise, no transaction fee is applied.
- debit() and credit() can only be done if their balance remains positive after transaction
- Implement the member functions. No other member functions and data should be added.

Extra Global Functions

bool CheckingToSaving(CheckingAccount&, SavingAccount&, const double);

- This function should transfer an amount of money from a checking account to a saving account. The checking account should pay a transaction fee for withdrawing. Return true when the transaction is successful.

bool SavingToChecking(SavingAccount&, CheckingAccount&, const double);

- This function should transfer an amount of money from a saving account to a checking account. The saving account should pay a transaction fee for withdrawing and the checking account should pay a transaction fee for deposition. Return true when the transaction is successful.
- **These two functions should *make friend* to CheckingAccount class and SavingAccount class.**

Forward Class Declaration

class Implementation;

// Forward class declaration should be made if a class uses another class whose definition appears later. Here, this is required by the statement in the red line below. We may need such a mechanism in this lab.

Class implementation; // forward class declaration

class Interface

{

public:

Interface(int); // constructor

void setValue(int); // same public interface as

int getValue() const; // class Implementation has

~Interface(); // destructor

private:

// requires previous forward declaration (line 6)

Implementation *ptr;

}; // end class Interface

Main()

```
int main()
{
    cout << "\nCreate a saving account." << endl;
    SavingAccount sAcnt(300.0, 0.05);
    sAcnt.print();
    sAcnt.debit(50.0);
    cout << "  New balance after withdrawing $50 from the saving account: " << sAcnt.getBalance() << endl;
    sAcnt.credit(150.0);
    cout << "  New balance after depositing $150 to the saving account: " << sAcnt.getBalance() << endl;
    sAcnt.print();
    cout << "  Interest of the saving account: " << sAcnt.calculateInterest() << endl;
    cout << "  New balance after adding interest: " << sAcnt.getBalance() << endl;
    cout << "Withdrawing 80 from the saving account." << endl;
    sAcnt.debit(80.0);

    cout << "\nCreate a checking account." << endl;
    CheckingAccount cAcnt(400.0, 0.02);
    cAcnt.print();
    cAcnt.debit(200.0);
    cout << "  New balance after withdrawing $200 from the checking account: " << cAcnt.getBalance() << endl;
    cAcnt.credit(150.0);
    cout << "  New balance after depositing $150 to the checking account: " << cAcnt.getBalance() << endl;

    cout << endl;
    cAcnt.print();
    sAcnt.print();

    cout << "\nAfter transfer $600 from cAcnt to sAcnt." << endl;
    CheckingToSaving(cAcnt, sAcnt, 600.0);
    cout << "New balance of cAcnt: " << cAcnt.getBalance() << "  New balance of sAcnt: " << sAcnt.getBalance() << endl;
    cout << "\nAfter transfer $800 from sAcnt to sAcnt." << endl;
    SavingToChecking(sAcnt, cAcnt, 800.0);
    cout << "New balance of cAcnt: " << cAcnt.getBalance() << "  New balance of sAcnt: " << sAcnt.getBalance() << endl;
    CheckingToSaving(cAcnt, sAcnt, 50.0);
    cout << "\nAfter transfer $50 from cAcnt to sAcnt." << endl;
    cout << "New balance of cAcnt: " << cAcnt.getBalance() << "  New balance of sAcnt: " << sAcnt.getBalance() << endl;
    SavingToChecking(sAcnt, cAcnt, 50.0);
    cout << "\nAfter transfer $50 from sAcnt to sAcnt." << endl;
    cout << "New balance of cAcnt: " << cAcnt.getBalance() << "  New balance of sAcnt: " << sAcnt.getBalance() << endl;
}
```

Example Output

```
Create a saving account.
Saving Account:
  Balance: 300
  Interest rate: 0.05
  Transaction fee of withdraw: 3
  New balance after withdrawing $50 from the saving account: 247
  New balance after depositing $150 to the saving account: 397
Saving Account:
  Balance: 397
  Interest rate: 0.05
  Transaction fee of withdraw: 3
  Interest of the saving account: 19.85
  New balance after adding interest: 416.85
Withdrawing 80 from the saving account:
  Debit amount exceeded account balance.

Create a checking account.
Checking Account:
  Balance: 400
  Interest rate: 0.02
  Transaction fee of withdraw: 3
  Transaction fee of deposition: 2
  New balance after withdrawing $200 from the checking account: 197
  New balance after depositing $150 to the checking account: 345

Checking Account:
  Balance: 345
  Interest rate: 0.02
  Transaction fee of withdraw: 3
  Transaction fee of deposition: 2
Saving Account:
  Balance: 416.85
  Interest rate: 0.05
  Transaction fee of withdraw: 3

After transfer $600 from cAcnt to sAcnt:
Transfer transaction fails.
New balance of cAcnt: 345   New balance of sAcnt: 416.85

After transfer $800 from sAcnt to sAcnt:
Transfer transaction fails.
New balance of cAcnt: 345   New balance of sAcnt: 416.85

After transfer $50 from cAcnt to sAcnt:
New balance of cAcnt: 292   New balance of sAcnt: 466.85

After transfer $50 from sAcnt to sAcnt:
New balance of cAcnt: 340   New balance of sAcnt: 413.85
```