

# 601.220 Intermediate Programming

## Inheritance

# Inheritance

Classes we use are often related to each other

```
class Account {...};, class CheckingAccount {...};
```

- “is a” relationship; a checking account is a kind of account
- this is inheritance

```
class GradeList {...};, vector<double>
```

- “has a” relationship; a grade list *has* a vector of grades as part of it
- this is composition or aggregation

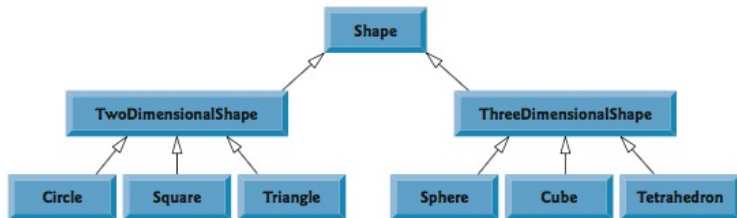
# Inheritance examples

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

**Fig. 12.1** | Inheritance examples.

These are “is a” relationships

# Inheritance hierarchy



**Fig. 12.3** | Inheritance hierarchy for Shapes.

Multiple levels of “is a” relationships

## Inheritance declaration and terminology

```
class BaseClass {  
    // Definitions for BaseClass  
};  
class DerivedClass : public BaseClass {  
    // Definitions for DerivedClass  
};
```

- *Derived class* inherits from *base class*
- Java-like vocab: *subclass* inherits from *superclass* (We'll typically say "derived" and "base")
- This is "public inheritance" – by far the most common kind
  - access of members in the base class is passed down and preserved
- protected & private inheritance also possible, but rarely used. Note: if you forget to explicitly say `public`, the default is private and that can get you into trouble

# Inheritance access

`protected` is an access modifier we haven't used yet

`protected` fields & functions can only be accessed:

- from member functions of their class
- from member functions defined in derived classes

## Inheritance access

Base-class members marked `public` or `protected` can be accessed from member functions defined in the derived class

Base-class members marked `private` *cannot* be accessed from member functions defined in the derived class

- They're still there, and *base class* member functions can still use them, but *derived class* member functions can't use them (without `public` or `protected` accessor or mutator functions)

## C++ classes: inherit what?

- Derived class *inherits most members* of base class, whether public, protected or private
  - can only access public and protected members directly
- Does not inherit
  - constructors
  - assignment operator if explicitly defined
- Derived class cannot delete things it inherited; cannot pick and choose what to inherit
- But derived class can *override* inherited member functions
  - override = substitute own implementation for base class's



## C++ classes: Bank Account base class

```
// account.h:  
class Account {  
public:  
    Account() : balance(0.0) { }  
    Account(double initial) : balance(initial) { }  
  
    void credit(double amt)    { balance += amt; }  
    void debit(double amt)     { balance -= amt; }  
    double get_balance() const { return balance; }  
private:  
    double balance;  
};
```

Default constructor sets balance to 0; non-default constructor sets according to argument

balance is private, modified via `credit(amt)/debit(amt)`

## C++: Account usage

```
// account_main1.cpp:
#include <iostream>
#include "account.h"

using std::cout; using std::endl;

int main() {
    Account acct(1000.0);
    acct.credit(1000.0);
    acct.debit(100.0);
    cout << "Balance is: " << acct.get_balance() << endl;
    return 0;
}

$ g++ -c account_main1.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o account_main1 account_main1.o
$ ./account_main1
Balance is: 1900
```

# C++: Inheritance example Checking

```
// checkingaccount.h:
class CheckingAccount : public Account {
public:
    CheckingAccount(double initial, double atm) :
        Account(initial), total_fees(0.0), atm_fee(atm) { }

    void cash_withdrawal(double amt) {
        total_fees += atm_fee;
        debit(amt + atm_fee);
    }

    double get_total_fees() const { return total_fees; }

private:
    double total_fees;
    double atm_fee;
};
```

## C++: Calling base class constructor

- Derived classes don't inherit constructors, but (usually) need to call their base class constructor to initialize inherited data members
  - We do this with the base class name in C++ (no `super()` like in Java)
  - The base class constructor call must be the first thing in the derived class constructor
  - If the base class constructor call is missing, then a default constructor for the base class will be called automatically; error if one doesn't exist!

```
CheckingAccount(double initial, double atm) :  
    Account(initial), total_fees(0.0), atm_fee(atm) { }
```

Notice the `Account(initial)` call to the base class constructor

## C++: Inheritance example Savings

```
// savingsaccount.h:  
class SavingsAccount : public Account {  
public:  
    SavingsAccount(double initial, double rate) :  
        Account(initial), annual_rate(rate) { }  
  
    // Not implemented here; usual compound interest calc  
    double total_after_years(int years) const;  
  
private:  
    double annual_rate;  
};
```

# Inheritance usage

```
// account_main2.cpp:
#include <iostream>
#include "account.h"
#include "savingsaccount.h"
#include "checkingaccount.h"

using std::cout; using std::endl;

int main() {
    Account acct(1000.0);
    acct.credit(1000.0);
    acct.debit(100.0);
    cout << "Account balance is: $" << acct.get_balance() << endl;

    CheckingAccount checking(1000.0, 2.00);
    checking.credit(1000.0);
    checking.cash_withdrawal(100.0); // incurs ATM fee
    cout << "Checking balance is: $" << checking.get_balance() << endl;
    cout << "Checking total fees is: $" << checking.get_total_fees() << endl;

    SavingsAccount saving(1000.0, 0.05);
    saving.credit(1000.0);
    cout << "Savings balance is: $" << saving.get_balance() << endl;
    return 0;
}
```

# Inheritance

```
$ g++ -c account_main2.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o account_main2 account_main2.o
$ ./account_main2
Account balance is: $1900
Checking balance is: $1898
Checking total fees is: $2
Savings balance is: $2000
```

## C++: Inheritance & Destructors

- When a derived class object is created, its inherited (base) parts must be initialized before any newly defined parts by executing a base constructor (default or explicit call to one)
- When the lifetime of a derived class object is about to end, two destructors are called: the one defined for the derived object and then the one defined for the base class
  - Either destructor may be explicitly defined, or just the provided default
- Note that constructors and destructors are executed in opposite orders!