

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 an account
- this is **inheritance**

```
class GradeList {...};, std::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 all **is-a** relationships.

Inheritance hierarchy

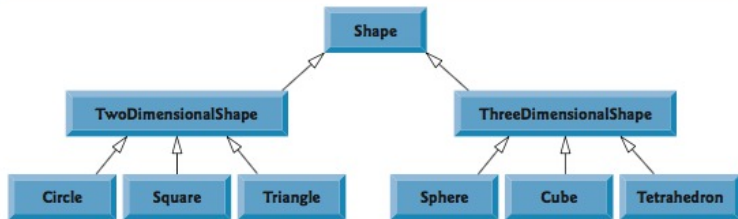


Fig. 12.3 | Inheritance hierarchy for Shapes.

Multiple levels of **is-a** relationships.

Inheritance declaration and terminology

```
class BaseClass {  
    // Base class definitions  
};  
class DerivedClass : public BaseClass {  
    // Derived class definitions  
}
```

- **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
 - 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`
- 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 an own implementation for base class's

C++ classes: An Account base class

// account.h

```
1  class Account {
2  public:
3      Account() : balance(0.0) { }
4      Account(double initial) : balance(initial) { }
5
6      void credit(double amt)    { balance += amt; }
7      void debit(double amt)     { balance -= amt; }
8      double get_balance() const { return balance; }
9  private:
10     double balance;
11 };
```

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

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

C++ classes: An Account base class usage

```
// account_main1.cpp
1  #include "account.h"
2  #include <iostream>
3
4  int main() {
5      Account acct(1000.0);
6      acct.credit(1000.0);
7      acct.debit(100.0);
8      std::cout << "Balance is: " << acct.get_balance() << std::endl;
9      return 0;
10 }

$ g++ -o account_main1 account_main1.cpp -std=c++11 -pedantic -Wall -Wextra

$ ./account_main1

Balance is: 1900
```

C++ classes: Inheritance example - Checking Account

// checking_account.h

```
1  class CheckingAccount : public Account {
2  public:
3      CheckingAccount(double initial, double atm) :
4          Account(initial), total_fees(0.0), atm_fee(atm) { }
5
6      void cash_withdrawal(double amt) {
7          total_fees += atm_fee;
8          debit(amt + atm_fee);
9      }
10
11     double get_total_fees() const { return total_fees; }
12
13 private:
14     double total_fees;
15     double atm_fee;
16 };
```

C++ classes: Inheritance - classing 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) { }
```

C++ classes: Inheritance example - Savings Account

```
// savings_account.h

1  class SavingsAccount : public Account {
2  public:
3      SavingsAccount(double initial, double rate) :
4          Account(initial), annual_rate(rate) { }
5
6      // Not implemented here; usual compound interest calc
7      double total_after_years(int years) const;
8
9  private:
10     double annual_rate;
11 };
```

C++ classes: Inheritance usage

```
// account_main2.cpp
1  #include <iostream>
2  #include "account.h"
3  #include "savings_account.h"
4  #include "checking_account.h"
5
6  int main() {
7      Account acct(1000.0);
8      acct.credit(1000.0);
9      acct.debit(100.0);
10     std::cout << "Account balance is: $" << acct.get_balance() << std::endl;
11
12     CheckingAccount checking(1000.0, 2.00);
13     checking.credit(1000.0);
14     checking.cash_withdrawal(100.0); // incurs ATM fee
15     std::cout << "Checking balance is: $" << checking.get_balance() << std::endl;
16     std::cout << "Checking total fees is: $" << checking.get_total_fees() << std::endl;
17
18     SavingsAccount saving(1000.0, 0.05);
19     saving.credit(1000.0);
20     std::cout << "Savings balance is: $" << saving.get_balance() << std::endl;
21     return 0;
22 }
```

```
$ g++ -o account_main2 account_main2.cpp -std=c++11 -pedantic -Wall -Wextra
```

```
$ ./account_main2
```

```
Account balance is: $1900
Checking balance is: $1898
Checking total fees is: $2
Savings balance is: $2000
```

C++ classes: 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
 - A chain call happens for multi-levels inheritances
- Note that constructors and destructors are executed in opposite orders!