Chapter 12

Polymorphism
（多态性）

Chapter 12 Topics

- Polymorphism in Programming Languages
- Override
- Static versus Dynamic Method Binding(绑定)
- Abstract Classes

Meaning of Polymorphism

In object-oriented languages, the term *polymorphism* means there is one name and many different meanings.

But names are used for a variety of purposes (variable names, function names, class names), and meanings can be defined in a number of different ways.

Major Forms of Polymorphism in Object Oriented Languages

- ***Overloading*** **(ad hoc polymorphism)**
  重载（专用多态）
- ***Overriding*** **(inclusion polymorphism)**
  改写（包含多态）
- ***Polymorphic Variable*** **(assignment polymorphism)**
  多态变量（赋值多态）
- ***Generics*** **(templates)**
  泛型（模板）

Overloading

The term *overloading* is used to describe the situation where a single function name (or method name) has several alternative implementations.

```
class OverLoader{
public:
    void example(int x){…}
    void example(int x,double y){…}
    void example(char * x){…}
…
};
```

Overriding

Overriding is in some sense a special case of overloading but occurs within the context of the parent class/child class relationship.

```
class Parent{
public:
    void example(int x){…}
…
};
```

```
class Child:public Parent{
public:
    void example(int x){…}
…
};
```

Polymorphic Variable

The *polymorphic variable* is a variable that is declared as one type but in fact holds a value of a different type.

```
Parent * p; //declared as parent
p=new Child( ); //holding child value
```

Generics

Generics provide a way of creating general tools or classes by parameterizing(参数化) on types.

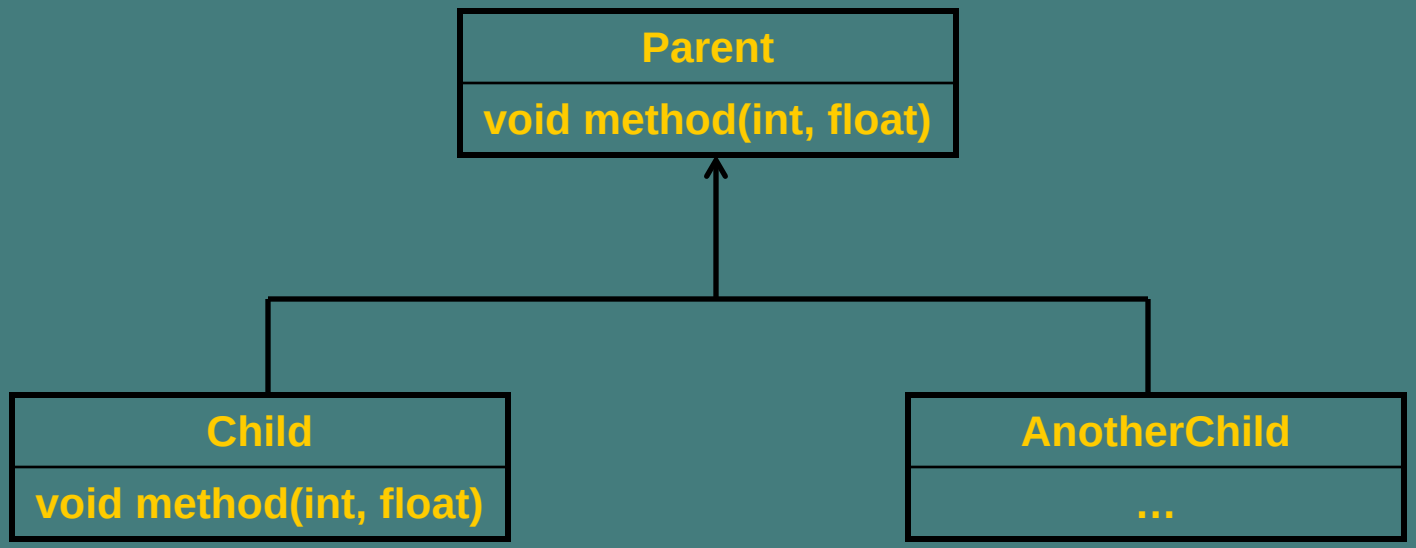
```
template <class T>
T max(T left, T right)
{
    if (left<right)
        return right;
    return left;
}
```

Chapter 12 Topics

- Polymorphism in Programming Languages
- Override
- Static versus Dynamic Method Binding
- Abstract Classes

A Definition of Override

We say that a method in a child class *overrides* a method in the parent class if the two methods have the same name and type signature.



Difference from Overloading

- Overriding only occurs in the context of the parent/child relationship
- The type signatures must match
- Overridden methods are sometimes combined together
- Overriding is resolved at run-time, not at compile time.

Syntax for Overriding

```
//parent class
class GraphicalObject {
public:
    virtual void draw( ); // can be overridden
};
//child class
class Ball : public GraphicalObject {
public:
    [virtual] void draw( ); // virtual optional here
};
```

Chapter 12 Topics

- Polymorphism in Programming Languages
- Override
- Static versus Dynamic Method Binding
- Abstract Classes

What do the terms Static and Dynamic Mean?

Static almost always means fixed or bound at compile time, and cannot thereafter be changed.

Dynamic almost always means not fixed or bound until run time, and therefore can change during the course of execution.

Method Binding

In almost all object-oriented programming languages, the binding of a method to execute in response to a message is determined by the dynamic value currently being held by the receiver.

Let's say that a variable is polymorphic if the binding of message to a method is determined by the type associated with the value most recently assigned to the variable.

Rules for Member Function Binding in C++

- With variables that are declared normally, the binding of member function name to function body is based on the static type of the argument (regardless whether the function is declared virtual or not).
- With variables that are declared as references or pointers, the binding of the member function name to function body is based on the dynamic type if the function is declared as virtual, and the static type if not.

Normal Method Binding

```
class Animal{
public:
   virtual void speak( ){cout<<“Animal Speak!”<<endl;}
   void reply( ){cout<<“Animal Reply!”<<endl;}
};
class Dog:public Animal{
public:
   virtual void speak( ){cout<<“Woof!”<<endl;}
   void reply( ){cout<<“Woof Again!”<<endl;}
};
class Bird:public Animal{
public:
   virtual void speak( ){cout<<“Tweet!”<<endl;}
};
```

```
Animal a;
Dog b;
b.speak();
a=b;
a.speak();
Bird c;
c.speak();
a=c;
a.speak();
```

Woof!

Animal Speak!

Tweet!

Animal Speak!

Pointer Using in Method Binding

```
class Animal{
public:
    virtual void speak( ){cout<<“Animal Speak!”<<endl;}
    void reply( ){cout<<“Animal Reply!”<<endl;}
};
class Dog:public Animal{
public:
    virtual void speak( ){cout<<“Woof!”<<endl;}
    void reply( ){cout<<“Woof Again!”<<endl;}
};
class Bird:public Animal{
public:
    virtual void speak( ){cout<<“Tweet!”<<endl;}
};
```

```
Animal * d;
Dog b;
b.speak( );
d=&b;
d->speak( );
Bird c;
c.speak( );
d=&c;
d->speak( );
```

Woof!

Woof!

Tweet!

Tweet!

Reference Using in Method Binding

```
class Animal{
public:
    virtual void speak( ){cout<<“Animal Speak!”<<endl;}
    void reply( ){cout<<“Animal Reply!”<<endl;}
};
class Dog:public Animal{
public:
    virtual void speak( ){cout<<“Woof!”<<endl;}
    void reply( ){cout<<“Woof Again!”<<endl;}
};
class Bird:public Animal{
public:
    virtual void speak( ){cout<<“Tweet!”<<endl;}
};
```

```
Dog b;
b.speak( );
Animal &e=b;
e.speak( );
Bird c;
c.speak( );
Animal &f=c;
f.speak( );
```

Woof!

Woof!

Tweet!

Tweet!

Shadowing Methods

```
class Animal{
public:
    virtual void speak( ){cout<<“Animal Speak!”<<endl;}
    void reply( ){cout<<“Animal Reply!”<<endl;}
};
class Dog:public Animal{
public:
    virtual void speak( ){cout<<“Woof!”<<endl;}
    void reply( ){cout<<“Woof Again!”<<endl;}
};
class Bird:public Animal{
public:
    virtual void speak( ){cout<<“Tweet!”<<endl;}
};
```

```
Animal * g;
Dog b;
b.reply( );
g=&b;
g->reply( );
Bird c;
g=&c;
g->reply( );
```

Woof Again!

Animal Reply!

Animal Reply!

Overriding, Shadowing and Redefinition

- ***Overriding*** **—The type signatures are the same in both parent and child classes, and the method is declared as virtual in the parent class.**
- ***Shadowing*** **—The type signatures are the same in both parent and child classes, but the method was not declared as virtual in the parent class.**
- ***Redefinition*** **—The type signature in the child class differs from that given in the parent class.**

Chapter 12 Topics

- Polymorphism in Programming Languages
- Override
- Static versus Dynamic Method Binding
- Abstract Classes

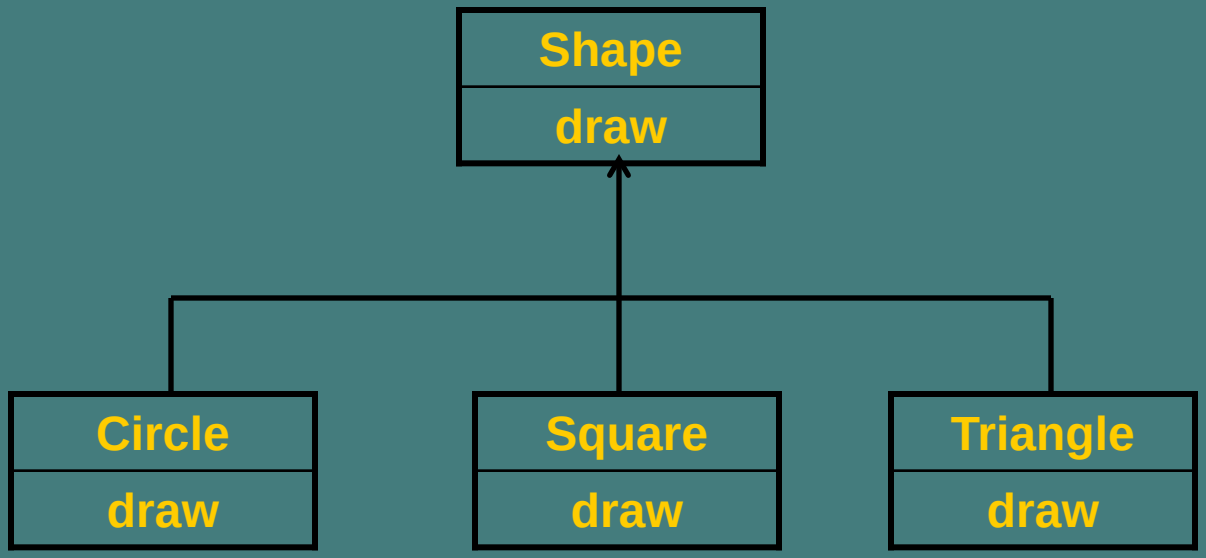
Interfaces and Abstract Classes

An *interface* is similar to a class, but does not provide any implementation. A child class must override all methods.

A middle ground is an *abstract class*. Here some methods are defined, and some (abstract methods) are undefined. A child class must fill in the definition for abstract methods. In C++ an abstract method is called a *pure virtual method*(纯虚方法).

Advantages of Deferred Method

- Allow the programmer to think of an activity as associated with an abstraction at a **higher level** than may actually be the case.



Syntax for Abstract Classes

```
class Shape{
public:
    virtual void draw( )=0; //pure virtual method
};
```