Chapter 15: Polymorphism and Virtual Functions

Starting Out with C++
Early Objects
Seventh Edition
by Tony Gaddis, Judy Walters, and Godfrey Muganda

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Topics
15.1 Type Compatibility in Inheritance Hierarchies
15.2 Polymorphism and Virtual Member Functions
15.3 Abstract Base Classes and Pure Virtual Functions
15.4 Composition Versus Inheritance

15.1 Type Compatibility in Inheritance Hierarchies

• Classes in a program may be part of an inheritance hierarchy

Animal
Cat
Dog
Poodle

• Classes lower in the hierarchy are special cases of those above

Type Compatibility in Inheritance

• A pointer to a derived class can be assigned to a pointer to a base class. Another way to say this is:

Animal *pA = new Cat;

Using Type Casts with Base Class Pointers

• C++ uses the declared type of a pointer to determine access to the members of the pointed-to object

• If an object of a derived class is pointed to by a base class pointer, all members of the derived class may not be accessible

• Type cast the base class pointer to the derived class (via static_cast) in order to access members that are specific to the derived class

15-2

15-3

15-4

15-5

15-6
15.2 Polymorphism and Virtual Member Functions

- **Polymorphic code**: Code that behaves differently when it acts on objects of different types.
- **Virtual Member Function**: The C++ mechanism for achieving polymorphism.

---

### Polymorphism

Consider the Animal, Cat, Dog hierarchy where each class has its own version of the member function `id()`.

```
class Animal{
public: void id(){cout << "animal";}
}

class Cat : public Animal{
public: void id(){cout << "cat";}
}

class Dog : public Animal{
public: void id(){cout << "dog";}
}
```

---

The code is not polymorphic because in the expression `pA[k]->id()` the compiler sees only the type of the pointer `pA[k]`, which is pointer to `Animal`. Compiler does not see type of actual object pointed to, which may be `Animal`, or `Dog`, or `Cat`.

---

Polymorphic code would have printed "animal dog cat" instead of "animal animal animal".
Virtual Functions

Declaring a function **virtual** will make the compiler check the type of each object to see if it defines a more specific version of the virtual function.

If the member functions `id()` are declared virtual, then the code

```cpp
Animal *pA[] = {new Animal, new Dog, new Cat};
for(int k=0; k<3; k++)
pA[k]->id();
```

will print `animal dog cat`.

How to declare a member function virtual:

```cpp
class Animal{
    public: virtual void id(){cout << "animal";}
}
class Cat : public Animal{
    public: virtual void id(){cout << "cat";}
}
class Dog : public Animal{
    public: virtual void id(){cout << "dog";}
}
```

Function Binding

- **Static binding** chooses the function in the class of the base class pointer, ignoring any versions in the class of the object actually pointed to.
- **Static binding** is done at compile time.

- **Dynamic Binding** determines the function to be invoked at execution time.
- Can look at the actual class of the object pointed to and choose the most specific version of the function.
- **Dynamic binding** is used to bind virtual functions.
15.3 Abstract Base Classes and Pure Virtual Functions

- An abstract class is a class that contains no objects that are not members of subclasses (derived classes)
- For example, in real life, Animal is an abstract class: there are no animals that are not dogs, or cats, or lions…

Abstract Base Classes and Pure Virtual Functions

- Abstract classes are an organizational tool: useful in organizing inheritance hierarchies
- Abstract classes can be used to specify an interface that must be implemented by all subclasses

Abstract Functions

- The member functions specified in an abstract class do not have to be implemented
- The implementation is left to the subclasses
- In C++, an abstract class is a class with at least one abstract member function

Pure Virtual Functions

- In C++, a member function of a class is declared to be an abstract function by making it virtual and replacing its body with = 0;
  ```cpp
class Animal{
public:
    virtual void id()=0;
};
```
- A virtual function with its body omitted and replaced with =0 is called a pure virtual function, or an abstract function

Abstract Classes

- An abstract class can not be instantiated
- An abstract class can only be inherited from: that is, you can derive classes from it
- Classes derived from abstract classes must override the pure virtual function with a concrete member function before they can be instantiated.

15.4 Composition vs. Inheritance

- Inheritance models an 'is a' relation between classes. An object of a derived class 'is a(n)' object of the base class
- Example:
  - an UnderGrad is a Student
  - a Mammal is an Animal
  - a Poodle is a Dog
Composition vs. Inheritance

• When defining a new class:
  • Composition is appropriate when the new class needs to use an object of an existing class
  • Inheritance is appropriate when
    – objects of the new class are a subset of the objects of the existing class, or
    – objects of the new class will be used in the same ways as the objects of the existing class