Polymorphism & Virtual Methods

Week 6

Gaddis:15.6-15.8

CS 5301
Fall 2013

Jill Seaman

Polymorphism

- The Greek word poly means many, and the Greek word morphism means form.
- So, polymorphism means 'many forms'.
- In object-oriented programming (OOP), polymorphism refers to
  - identically named (and redefined) methods
  - that have different behavior depending on the (specific derived) type of object that they are called on.

Example of polymorphism?

```
class Animal {
  private:
    ...
  public:
    void speak() { cout << “none ”; }
};

class Cat : public Animal {
  private:
    ...
  public:
    void speak() { cout << “meow ”; }
};
class Dog : public Animal {
  private:
    ...
  public:
    void speak() { cout << “bark ”; }
};
```

Example of polymorphism?, part 2

```
void f (Animal a) {
  a.speak();
}

int main() {
   Cat c;
   Dog d;
   f(c);
   f(d);
}
```

- IF the output is “meow bark”, this (function f) is an example of polymorphism.
  - The behavior of a in f would depend on its specific (derived type).
- IF the output is “none none”, it’s not polymorphism.

Polymorphism in C++

- Polymorphism in C++ is supported through:
  - virtual methods AND
  - pointers to objects OR reference variables/ parameters.
- without these, C++ determines which method to invoke at compile time (using the variable type).
- when virtual methods and pointer/references are used together, C++ determines which method to invoke at run time (using the specific type of the instance currently referenced by the variable).

Virtual methods

- Virtual member function: function in a base class that expects to be redefined in derived class
- Function defined with key word virtual:

```
virtual void Y() {...}
```

- Supports dynamic binding: functions bound at run time to function that they call
- Without virtual member functions, C++ uses static (compile time) binding

Example virtual methods

```
class Animal {
  public:
  virtual void speak();
  int age();
};
class Cat : public Animal
{
  public:
  virtual void speak(); //redefining a virtual
  int age();            //redefining a normal function
};
int main()
{
  Cat morris;
  Animal *pA = &morris;
  pA -> age();   // Animal::age() is invoked (base) (not virtual)
  pA -> speak(); // Cat::speak()  is invoked (derived)
...
}
```

Virtual methods

- In compile-time binding, the data type of the pointer resolves which method is invoked.
- In run-time binding, the type of the object pointed to resolves which method is invoked.

```
void f (Animal &a) {
  a.speak();
}

int main() {
   Cat c;
   Dog d;
   f(c);
   f(d);
}
```

- Assuming speak is virtual, and a is passed by reference, the output is:

```
meow bark
```

Example version 1:

```
class COne {
  public:
    void vWhoAmI() { cout << "I am One" << endl; }
};
class CTwo : public COne {
  public:
    void vWhoAmI() { cout << "I am Two" << endl; }
};
class CThree : public CTwo {
  public:
    void vWhoAmI() { cout << "I am Three" << endl; }
};
```

```
int main() {
{
    COne *apCOne[3] = { new COne, new CTwo, new CThree };
    for (int i = 0; i < 3; i++)
        apCOne[i] -> vWhoAmI();
}
```

Output:

```
I am One
I am One
I am One
```

Example version 2:

```
class COne {
  public:
    virtual void vWhoAmI() { cout << "I am One" << endl; }
};
class CTwo : public COne {
  public:
    void vWhoAmI() { cout << "I am Two" << endl; }
};
class CThree : public CTwo {
  public:
    void vWhoAmI() { cout << "I am Three" << endl; }
};
```

```
int main() {
{
    COne *apCOne[3] = { new COne, new CTwo, new CThree };
    for (int i = 0; i < 3; i++)
        apCOne[i] -> vWhoAmI();
}
```

Output:

```
I am One
I am Two
I am Three
```

Abstract classes and Pure virtual functions

- Pure virtual function: a virtual member function that **must** be overridden in a derived class.
- Abstract base class contains at least one pure virtual function:

```
virtual void Y() = 0;
```

- The = 0 indicates a pure virtual function
- Must have no function definition in the base class.

Abstract classes and Pure virtual functions

- Abstract base class: a class that can have no objects (instances).
- Serves as a basis for derived classes that will have objects
- A class becomes an abstract base class when one or more of its member functions is a pure virtual function.

Example: Abstract Class

```
class CShape {
  public:
    CShape ( ) { }
    virtual void vDraw ( ) const = 0; // pure virtual method
};
```

- An abstract class may **not** be used as an argument type, as a function return type,or as the type of an explicit conversion.
- Pointers and references to an abstract class may be declared.

```
CShape CShape1;       // Error: object of abstract class
CShape* pCShape;      // Ok
CShape CShapeFun();   // Error: return type
void vg(CShape);      // Error: argument type
CShape& rCShapeFun(CShape&); // Ok
```

Example: Abstract Class

- Pure virtual functions are inherited as pure virtual functions.

```
class CAbstractCircle : public CShape {
  private:
    int m_iRadius;
  public:
    void vRotate (int) {}
    // CAbstractCircle ::vDraw() is a pure virtual function
};
```

- Or else:

```
class CCircle : public CShape {
  private:
    int m_iRadius;
  public:
    void vRotate (int) {}
    void vDraw();   //define here or in impl file
};
```

Heterogeneous collections

```
class Animal {
  private:
    string name;
  public:
    Animal(string n) {name = n;}
    virtual void speak() = 0;
};
class Cat : public Animal {
  public:
    Cat(string n) : Animal(n) { };
    void speak()  {cout << "meow "; }
};
class Dog : public Animal {
  public:
    Dog(string n) : Animal(n) { };
    void speak()  {cout << "bark "; }
};
class Pig : public Animal {
  public:
    Pig(string n) : Animal(n) { };
    void speak()  {cout << "oink "; }
};
```

Heterogeneous collections

- Driver:

```
int main()
{
    Animal* animals[ ] = {
        new Cat("Charlie"),
        new Cat("Scamp"),
        new Dog("Penny"),
        new Cat("Libby"),
        new Cat("Patches"),
        new Dog("Milo"),
        new Pig("Wilbur") };

    for (int i=0; i< 7; i++) {
        animals[i]->speak();
    }
}
```

```
meow meow bark meow meow bark oink
```