Inheritance: Polymorphism and Virtual Functions

Lecture 25

Sections 15.1 - 15.4

Robb T. Koether

Hampden-Sydney College

Fri, Mar 24, 2017

1. Polymorphism
2. Abstract Classes
3. Assignment

Outline

Polymorphism

Definition (Polymorphism)

Polymorphism allows an object of one type to be treated as an object of a different type, provided that the IS-A relation holds. The "actual" type of the object may not be determined until run time. This is called late binding or dynamic binding (as opposed to the usual early binding or static binding.).

- A function that specifies a base-class object in its parameter list may accept a derived-class object in its place.
- Polymorphism works because the derived-class object IS-A base-class object.
- We have already seen this used in the constructors.

Polymorphism and Passing by Value

- If a function passes the base-class object *by value*, then the derived-class object is considered to be an object of the base class.
- Why does this happen?

Polymorphism and Passing by Value

- If a function passes the base-class object *by value*, then the derived-class object is considered to be an object of the base class.
- Why does this happen?
- This happens because the base-class copy constructor was used to create the local object.

Polymorphism and Passing by Value

- If a function passes the base-class object *by value*, then the derived-class object is considered to be an object of the base class.
- Why does this happen?
- This happens because the base-class copy constructor was used to create the local object.
- The local object loses its derived-class data members and functions.

Example

Example (Polymorphism)

```
int main()
{
    Man man("John");
    Woman woman("Jane");
    describe(man);
    describe(woman);
}

void describe(Person p)
{
    cout << p << endl;  // Is p a Man or a Woman?
                        // Or just a Person?
    return;
}
```

Polymorphism and Passing by Reference

- If the function passes the base-class object *by reference*, then the derived-class object may maintain its identity as an object of the derived class.

Polymorphism and Passing by Reference

- If the function passes the base-class object *by reference*, then the derived-class object may maintain its identity as an object of the derived class.
- However...

Example

Example (Polymorphism)

```
int main()
{
    Man man("John");
    Woman woman("Jane");
    describe(man);
    describe(woman);
}

void describe(Person& p)
{
    cout << p << endl;  // Is p a Man or a Woman?
                        // Or just a Person?
    return;
}
```

Outline

Virtual Functions

- When the base class and a derived class have distinct functions of the same name, how does the compiler know which one to invoke?
- If the base-class function is virtual, then the computer will invoke the member function of that name that is closest to the class of the invoking object.
- Write the keyword **`virtual`** at the beginning of the function prototype.

Example

Example (Virtual Functions)

```
class Person
{
    virtual void output(ostream& out) const
        out << m_name << ' ' << m_sex;
};
```

Example

Example (Virtual Functions)

```
int main()
{
    Man man("John");
    Woman woman("Jane");
    describe(man);
    describe(woman);
}

void describe(Person& p)
{
    cout << p << endl;  // What will happen?
    return;
}
```

Virtual Functions and Value Parameters

- What happens when the function is virtual and the parameter is a value parameter?

Example

Example (Virtual Functions)

```
int main()
{
    Man man("John");
    Woman woman("Jane");
    describe(man);
    describe(woman);
}

void describe(Person p)
{
    cout << p << endl;  // What will happen?
    return;
}
```

Pure Virtual Functions

- A function may be designated as a pure virtual function.
- Write

```
virtual function(parameters) = 0;
```

- The function is *not* instantiated at this level in the class hierarchy.
- However, the function *must* be instantiated in the nonabstract derived classes.

Pure Virtual Functions

- This is done when
  - The function must be implemented at a certain level in the hierarchy,
  - But there is not enough information at that level to implement it.
- Example?

Abstract Classes

Definition (Abstract Class)

An abstract class is a class that contains a pure virtual function. No object of an abstract class may be instantiated.

- Function parameters of an abstract class type must be passed by reference.
- Why?

Example

Example (Pure Virtual Functions)

```
class Person
{
    virtual void output(ostream& out) const = 0;
};
```

Example

Example (Pure Virtual Functions)

```
int main()
{
    Man man("John");
    Woman woman("Jane");
    describe(man);
    describe(woman);
}

void describe(Person& p)
{
    cout << p << endl;  // What will happen?
    return;
}
```

Abstract Classes

- Why include the `output()` function in the `Person` class at all?

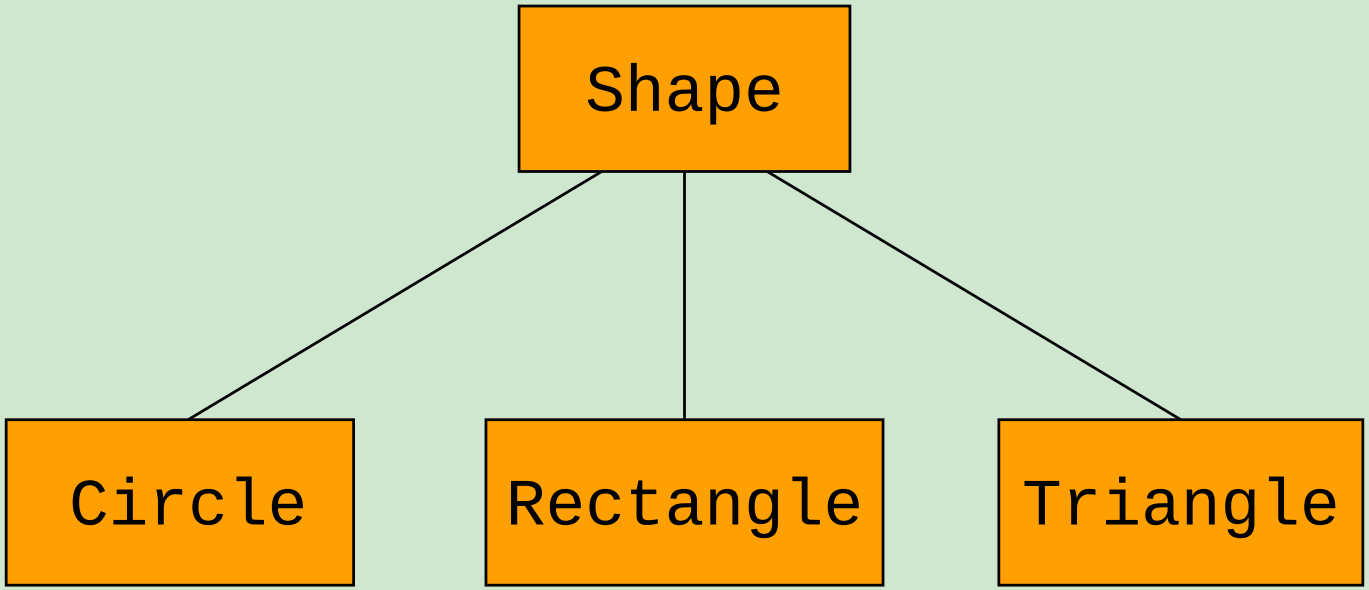
Example

Example (Abstract Class)

- Circles, squares, and triangles are shapes.
- Create a `Shape` class as a base class.

Example

Example (Abstract Class)



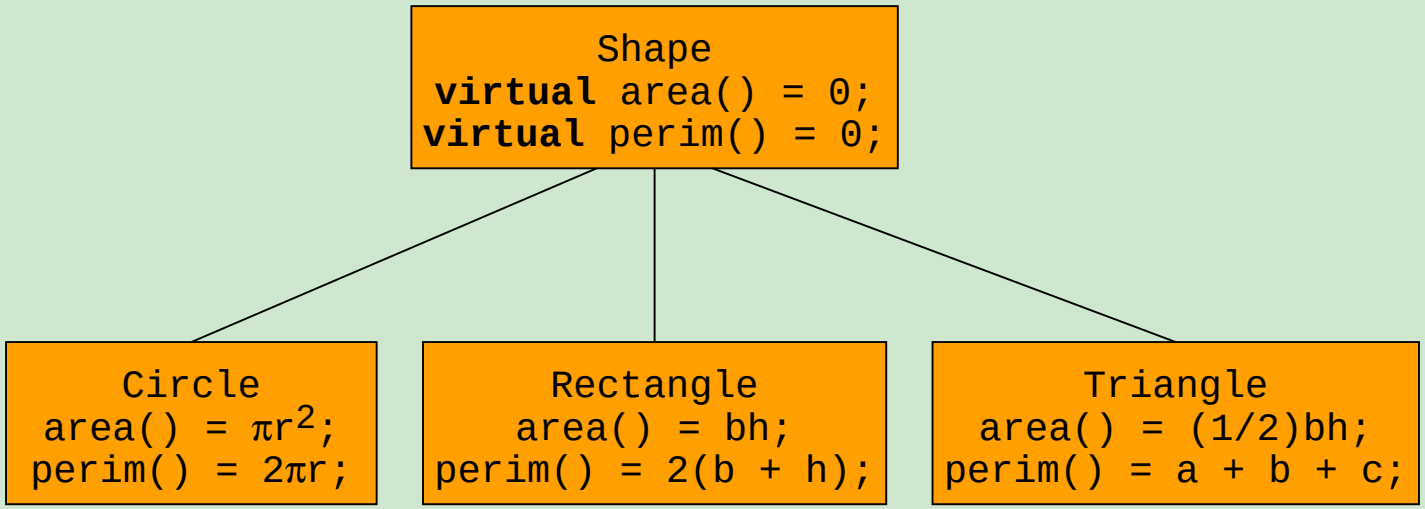
Example

Example (Abstract Class)

- Each shape has an area and a perimeter.
- However, we cannot find the area or perimeter until we know the *particular kind* of shape.
- Therefore, `Shape` should be an abstract class.

Example

Example (Abstract Class)



Outline

Assignment

Assignment

- Read Sections 15.1 - 15.4.