Lecture #10 | Inheritance

SE271 Object-oriented Programming (2017)

Prof. Min-gyu Cho

Today’s topic

- Brief introduction of OOP features
  - Abstraction
  - Encapsulation
  - Inheritance
  - Polymorphism
- Inheritance

Abstraction in OOP

- **Abstraction**: most fundamental concept in OOP
  - Powerful mechanism to manage complex systems
  - Managed by well-defined objects and their hierarchical classification
- Example
  - Car: composed of engine, tire, gearing system, steering mechanism, etc., but in real-life, we often consider a car as a single object without discussing/mentioning the details of it

OOP Features

- **Encapsulation**: group data and methods to manipulate them, and define & expose *interface* to hide detailed implementation
- **Inheritance**: allow code to be reused between related types for extending/modifying existing data & methods
- **Polymorphism**: allow a value to be one of several types, and determining at runtime which functions to call on it based on its type

What is inheritance?

- When class A inherits class B, class B is called ‘a base class’ of A and A is ‘a derived class’ of B
- A share all the features (i.e., member variables and member functions) of B
- A may add new features or override existing features

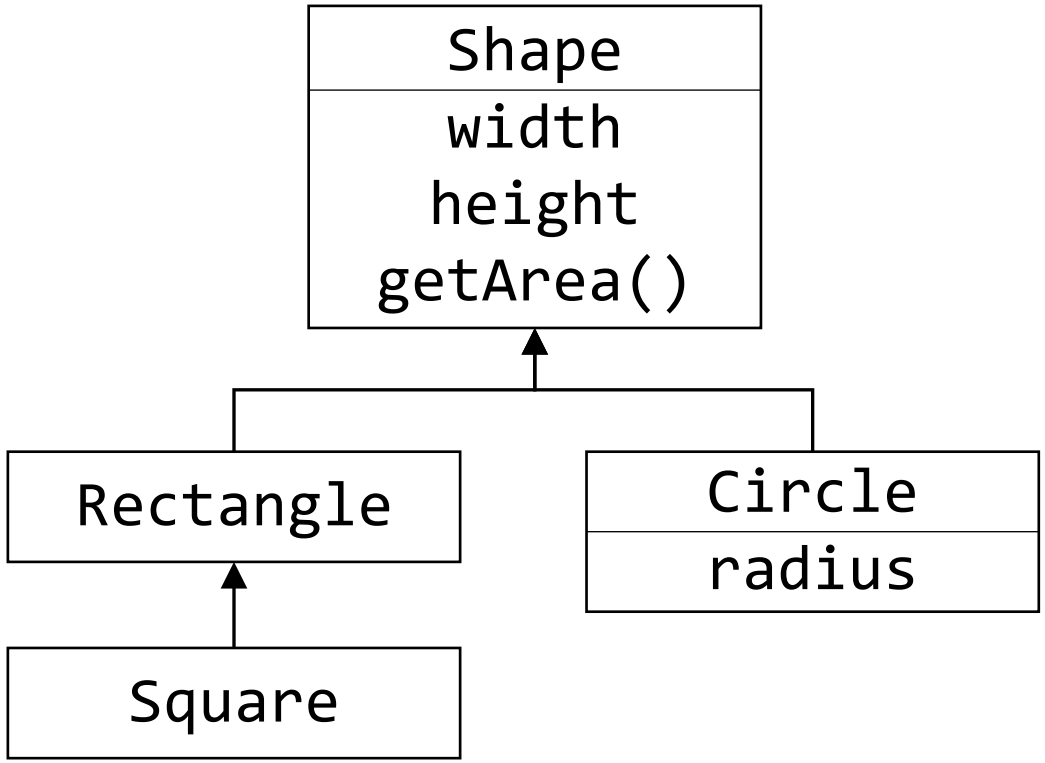
```
class Car : public Vehicle {
    ...
};
```

- Car is “a kind of a” Vehicle
- Car is “derived from” Vehicle
- Car is “a specialized” Vehicle
- Car is a “subclass” of Vehicle
- Car is a “derived class” of Vehicle
- Vehicle is the “base class” of Car
- Vehicle is the “superclass” of Car (not common in the C++ community)

Example: shapes

- Shape is a base class of all the different shapes
  - Characteristics
    - width
    - height
  - Behavior
    - getArea()

- Class hierarchy diagram (simplified)



Early implementation

```
class Shape {
protected:
    double width;
    double height;
public:
    Shape(double w, double h): width{w},
height{h} {}
    double getArea();
};

class Rectangle : public Shape {
public:
    Rectangle(double w, double h):
Shape{w, h} {}
    double getArea() { return width *
height; }
};
```

```
class Square : public Rectangle {
public:
    Square(double length) :
Rectangle{length, length} {}
};

class Circle : public Shape {
private:
    double radius;
public:
    Circle(double radius) : Shape{radius
* 2., radius * 2.} { this->radius =
radius; }
    double getArea() { return width *
width * M_PI / 4.; }
    double getRadius() { return radius; }
};
```

Early implementation (cont.)

```
#include "shapes.h"
int main()
{
    Rectangle rectangle {1, 2};
    Square square {3};
    Circle circle {2};
    cout << "Area of rectangle: " << rectangle.getArea() << endl;
    cout << "Area of square: " << square.getArea() << endl;
    cout << "Area of circle: " << circle.getArea() << endl;
    cout << "Radius of circle: " << circle.getRadius() << endl;
}
```

Constructor & destructor of a base/derived class

```
class Base {
protected:
    int val;
public:
    Base(int v = 0.0) { val = v; cout << "Base: constructor\n";}
    ~Base() { cout << "Base: destructor\n"; }
    int getVal() { return val; }
};
class Derived : public Base {
public:
    Derived(int v) : Base(v) { cout << "Derived: constructor\n"; }
    ~Derived() { cout << "Derived: destructor\n"; }
};
```

Constructor & destructor of base/derived class (cont.)

```
int main()
{
    cout << "Derived ----------------\n";
    Derived d1(42);
    cout << "d1=" << d1.getVal() << endl;
}
```

- Output

```
Derived1 ----------------
Base: constructor
Derived: constructor
d1=42
Derived: destructor
Base: destructor
```

- Order of constructor call
  - base → derived
- Order of destructor call
  - derived → base
- Why?

Access control with inheritance

```
class Base {
private/protected/public:
};

class Derived : public Base {
    ...
};
```

Types of inheritance

Access specifiers in the base classes

	private	protected	public
private	inaccessible	private	private
protected	inaccessible	protected	protected
public	inaccessible	protected	public

Reading list

- Learn C++
  - Inheritance: Ch. 11 1-4
- OOP in Java: http://beginnersbook.com/2013/03/oops-in-java-encapsulation-inheritance-polymorphism-abstraction/

ANY QUESTIONS?