fit@hcmus

Object-Oriented Programming

Object Life Cycle in Inheritance

Tran Duy Hoang

Topics Covered

- Constructor in Inheritance
- Destructor in Inheritance
- The Big Three in Inheritance

Topics Covered

- **Constructor in Inheritance**
- Destructor in Inheritance
- The Big Three in Inheritance

Order of Constructor Calls

- When you create an object of a derived class:
  1. The base class constructor is called first
  2. Then, the derived class constructor

- This ensures the base part of the object is properly initialized before initializing the derived part

Constructor Calling

- You can use an initialization list to call the base class constructor from the derived class constructor

```
class Base
{
private:
    int x;
public:
    Base(int x)
    {
        this->x = x;
    }
};
```

```
class Derived : public Base
{
private:
    int y;
public:
    Derived(int x, int y) : Base(x)
    {
        this->y = y;
    }
};
```

Constructor Calling

- If the base class has a default constructor, you don’t need to explicitly call it in the derived class:

```
class Base
{
private:
    int x;
public:
    Base()
    {
        this->x = 0;
    }
};
```

```
class Derived : public Base
{
private:
    int y;
public:
    Derived()
    {
        this->y = 0;
    }
};
```

Constructor Calling

- If the base class only has a parameterized constructor, must call it explicitly from the derived class, or get a compiler error

```
class Base
{
private:
    int x;
public:
    Base(int x)
    {
        this->x = x;
    }
};
```

```
class Derived : public Base
{
private:
    int y;
public:
    Derived() : Base(0)
    {
        this->y = 0;
    }
};
```

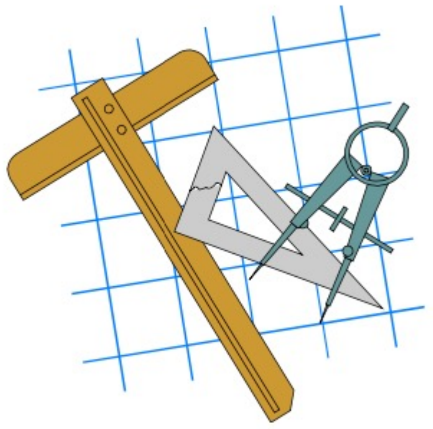
Exercise: Order of Constructor Calls

```
class A
{ public:
        A( int x ) {   }
};
class B: public A
{ public:
        B( ) {   }
        B( int x, int y ): A( x ) { }
};
class C: public B
{ public:
        C( ) {   }
        C( int z ) {   }
        C( int x, int y, int z ): B( x, y ) {   }
};
```

Point out initialization order of the followings:

```
a) void main() {   C   obj( 1, 2, 3 );   }
b) void main() {   C   obj( 4 );         }
c) void main() {   C   obj;              }
```

compiler error?



Topics Covered

- Constructor in Inheritance
- **Destructor in Inheritance**
- The Big Three in Inheritance

Order of Destructor Calls

- When a derived class object is destroyed:
    1. Derived class destructor is called first
    2. Then, the base class destructor is called

- This order is the reverse of constructor calls

Destructor Calling

- NOT need to call the base class destructor explicitly from the derived class destructor

```
class Base
{
private:
    int* x;
public:
    ~Base()
    {
        delete x;
    }
};
```

```
class Derived : public Base
{
private:
    int* y;
public:
    ~Derived()
    {
        delete y;
    }
};
```

Topics Covered

- Constructor in Inheritance
- Destructor in Inheritance
- **The Big Three in Inheritance**

The Big Three Rule

- C++ provides default versions:
  - The default destructor does nothing
  - The default copy constructor performs shallow copy
  - The default assignment operator performs shallow copy

- The Big Three Rule state that
  - If a class allocates resources dynamically, then it should implement all three: Destructor, Copy Constructor, Assignment Operator.

The Big Three in Inheritance

- If derived class also manage resources, then it should follow The Big Three too

```
class Base
{
private:
    int* x;
public:
    Base(const Base& a);
    Base& operator=(const Base& a);
    ~Base();
};
```

```
class Derived : public Base
{
private:
    int* y;
public:
    Derived(const Derived & b);
    Derived & operator=(const Derived & b);
    ~Derived();
};
```

The Big Three in Inheritance

- Link The Big Three of two classes

```
Derived::Derived(const Derived & b) : Base(b)
{
    // copy new attributes
}

Derived& Derived::operator=(const Derived & b)
{
    Base::operator=(b);
    // assign new attributes
}
```

```
Derived::~Derived()
{
    // dispose new attributes
}
```

Exercise 6.1

- Implement a class Array to manage a list of integers, and create a derived class HistogramArray that adds histogram functionality. The histogram should track how many times each value appears in the array.
  - Array
    - Attribute: int* data, int size
    - Method: constructor, destructor, copy constructor, assignment operator, set and get value at an index, display array elements
  - HistogramArray
    - Attribute: int* histogram, int histSize (value range, e.g. 0–9 → 10)
    - Method: compute histogram, display histogram, recomputes the histogram automatically every time a value is changed

Exercise 6.1

```
int main() {
    int arr[6] = {0, 1, 0, 2, 3, 4};
    Array a1(6, arr);
    a1[3] = 4;
    //...

    HistogramArray a2(6, arr, 5); // histSize = 5 -> [0, 4]
    a2.computeHist();
    a2.displayHist(); // 0->2, 1->1, 2->1, 3->1, 4->1
    a2[3] = 4; // [0, 1, 0, 4, 3, 4]
    a2.computeHist();
    a2.displayHist(); // 0->2, 1->1, 2->0, 3->1, 4->2
    //...
}
```

Exercise 6.2

- A cinema has multiple screens. A standard screen has M seats, and all seats have the same standard price. A VIP screen has fewer seats, and ticket prices include an additional charge based on a percentage rate. Additionally, the VIP screen includes a list of extra services, such as “Recliners”, “Free Snacks”, “Lounge Access”, etc. Design the classes: Screen, VIPScreen with the following functionalities:
    - Check if a seat is available
    - Calculate the price of a seat
    - Book a seat
    - Print the information of a screen
    - Calculate the total revenue of a screen