Object Life Cycle in Inheritance

Inst. Nguyễn Minh Huy

Contents

- Constructors in inheritance.
- Destructor in inheritance.
- The Big Three in inheritance.

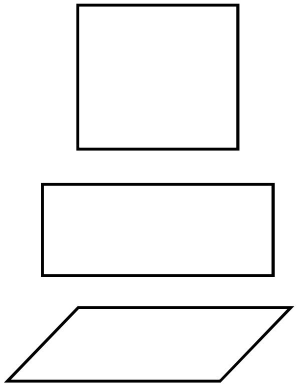
Contents

- **Constructors in inheritance.**
- Destructor in inheritance.
- The Big Three in inheritance.

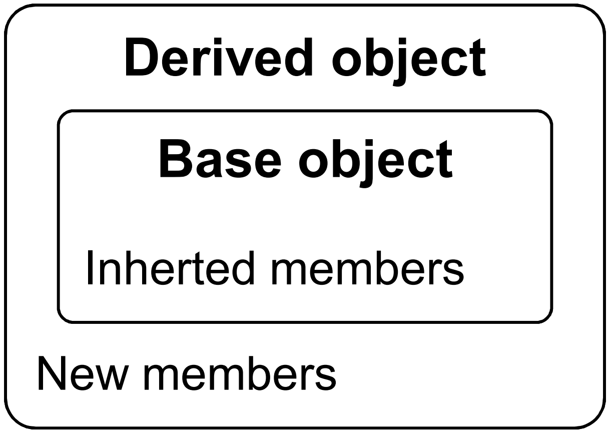
Constructor in inheritance

- Object initialization order:
  - Build house from the ground up.
  - Define concept from basic one.
  - Initialize object from “core” and “skin”:
    - Object “core”: inherited members.
    - Object “skin”: new members.

From the ground up



From basic concept



Derived object

Base object

Inherted members

New members

From “core” to “skin”

Constructor in inheritance

- Initialization order of derived object:
  - Base class constructor called first.

    ➔ **Initialize inherted members.**
  - Derived class constructor called after.

    ➔ **Initialize new members.**
  - Derived class can **decide** how to initialize its core.

    ➔ **Select base class constructor to call.**

    ➔ **Forget to select: default construct is called.**

Constructor in inheritance

- Example:

```
class Teacher
{
private:
        char        *m_name;
        float       m_salary;
        int         m_vacation;
public:
        Teacher();
        Teacher(char  *name,
                    float  salary,
                    int  vacation);
};
```

```
class HRTeacher : public Teacher
{
private:
        char        *m_classRoom;
public:
        HRTeacher();
        HRTeacher(char  *m_class);
        HRTeacher(char  *name,
                    float  salary,
                    int  vacation,
                    char  *m_classRoom);
};
```

Constructor in inheritance

- Example:

```
HRTeacher::HRTeacher(char  *cls) : Teacher(“John”, 1000, 0)
{
        m_classRoom = new char[strlen(cls) + 1];
        strcpy(m_classRoom, cls);
}
HRTeacher::HRTeacher(char  *name, float  salary, int  vacation, char  *cls)
                                    : Teacher(name, salary, vacation)
{
        m_classRoom = new char[strlen(cls) + 1];
        strcpy(m_classRoom, cls);
}
HRTeacher::HRTeacher()
{
}
```

Teacher() is called first!!

Contents

- Constructor in inheritance.
- **Destructor in inheritance.**
- The Big Three in inheritance.

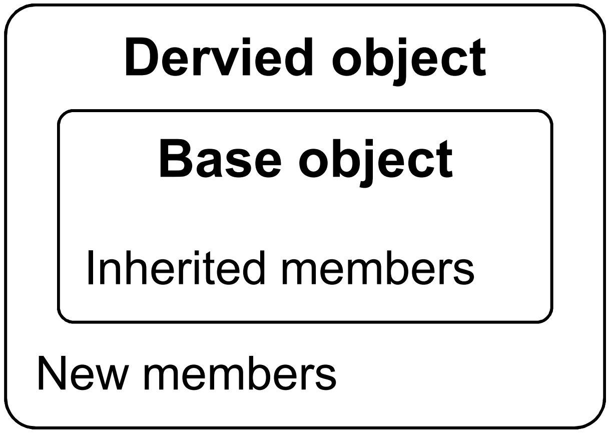
Destructor in inheritance

- Destruction order of derived object:
  - Initialization order in reverse.
  - Derived class destructor is called first.

    ➔ **Dispose object skin.**
  - Base class destructor is called after.

    ➔ **Dispose object core.**
  - Class has only 1 destructor:

    ➔ **Select destructor is not necessary.**



Dispose from skin to core

Destructor in inheritance

- Example:

```
Teacher::~Teacher()
{
        delete m_name;
}

HRTeacher::~HRTeacher()
{
        delete m_classRoom;
}
```

~Teacher() called after

~HRTeacher() called first

Contents

- Constructor in inheritance.
- Destructor in inheritance.
- **The Big Three in inheritance.**

The Big Three in inheritance

- **Rule of Three**:
  - Class having pointers and allocations:

    ➔ Implement **“The Big Three”** explicitly:
    - Destructor.
    - Copy constructor.
    - Assignment operator.
  - How about derived class?

The Big Three in inheritance

- Dr. Guru: **Rule of Three in Inheritance**
  - Derived class has pointer and allocate memory:
    - Implement “The Big Three” explicitly for derived class.
    - **Link “The Big Three” of derived to base class.**

The Big Three in inheritance

- Example:

Base class
“Big Three”

```
class A
{
public:
        A(const A &a);
        A& operator =(const A &a);
        virtual ~A();
};
class B: public A
{
public:
        B(const B &b);
        B& operator =(const B &b);
        ~B();
};
```

Derived class
“Big Three”

Link “Big Three”

```
B::B(const B &b) : A(b)
{
        // Copy new members...
}

B& operator =(const B &b)
{
        A::operator =(b);
        // Assign new members...
}

~B()  // Auto call ~A().
{
        // Dispose new members...
}
```

The Big Three in inheritance

- Dr. Guru: **Pointer usage rule.**
  - Should not use pointer at all!!
  - Use instead:
    - std::vector for array.
    - std::string for string.
    - Iterator for array iteration.
    - Functor for function pointer.
  - RAII - **R**esource **A**cquisition **I**s **I**nitialization:
    - Use object to manage memory.
    - Allocate memory in constructor.
    - De-allocate memory in destructor.

The Big Three in inheritance

- Dr. Guru: **Pointer usage rule.**
  - RAII - **R**esource **A**cquisition **I**s **I**nitialization:
    - Buit-in: **std::unique_ptr, std::shared_ptr, std::weak_ptr**.
    - Manual:

```
class PointerWrapper {
    int  *m_pointer;
public:
    PointerWrapper( int *pointer ): m_pointer( pointer ) { }
    ~PointerWrapper( ) { delete [ ]m_pointer; }
    // Implement The Big Three…
};
int main() {
    PointerWrapper        p1( new int[ 10 ] );
    std::unique_ptr<int> p2( new int[ 20 ] );
    // p1, p2 auto de-allocate memory.
}
```

Summary

- Constructors in inheritance:
  - Initialize from “core” to “skin”.
  - Build core: call base class constructor first.
  - Build skin: class derived class constructor after.
  - Select base class constructor to call.
- Destructor in inheritance:
  - Dispose from “skin” to “core.
  - Dispose skin: call derived class destructor.
  - Dispose core: call base class destructor.
- Rule of Three in inheritance.
- Rule of Pointer usage.

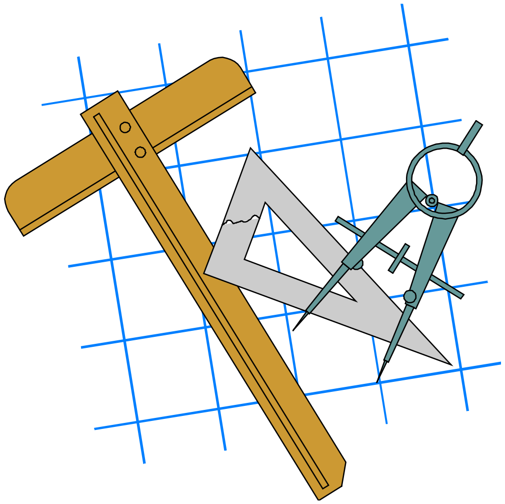
Practice

- Practice 8.1:

```
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 ) { }
};
```

Explain initialization orders of the followings:

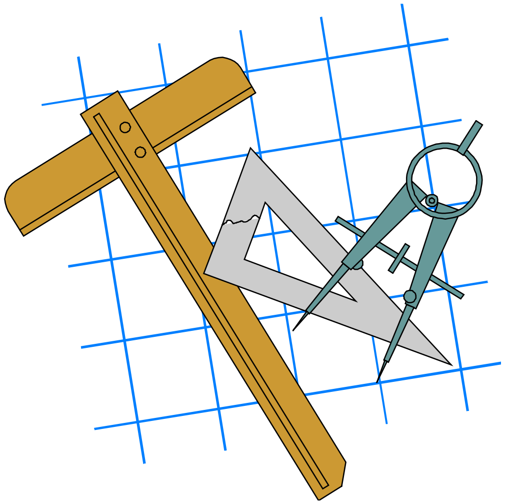
```
a) int main() {   C   obj( 1, 2, 3 );   }
b) int main() {   C   obj( 4 );         }
c) int main() {   C   obj;              }
```



Practice

- Practice 8.2:

  Apply Rule of Three for class **Teacher** and **HRTeacher**.



Practice

- Practice 8.3:

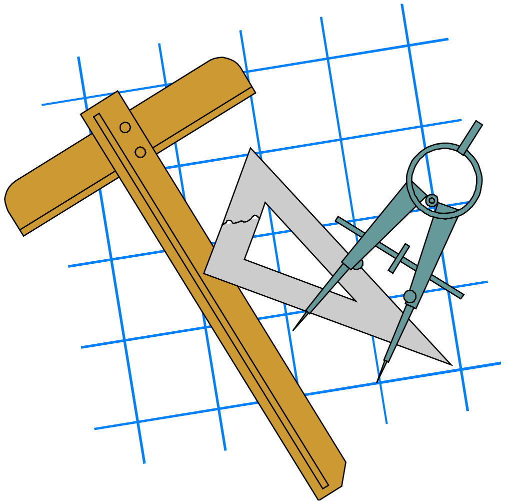
```
class X {   };

class Y: public X
{
public:
        Y( int i ) {   }
};

class Z: public Y
{
public:
        Z( int i ): Y( i++ ) {   }
};
```

Identify initialization orders of the followings:

a)
```
int main( )   {   Z   obj( 5 );   }
```

b)
```
int main( )
{
          Y   obj1( 6 );
          Y   obj2( obj1 );
}
```

c)
```
int main( )
{
          Z   obj1( 7 );
          Z   obj2( obj1 );
}
```



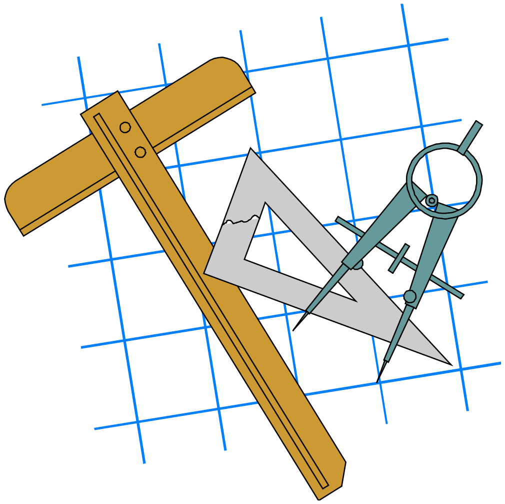
Practice

- Practice 8.4:

  Draw inheritance tree for the following classes:

  (create base classes as needed for reusability)

  - Square.
  - Circle.
  - Ellipse.
  - Rectangle.
  - Diamond.
  - Parallelogram.
  - Isosceles trapezoid.
  - Right trapezoid.
  - Right triangle.
  - Isosceles triangle.
  - Right isosceles triangle.
  - Equilateral triangle.



Practice

- Practice 8.5:

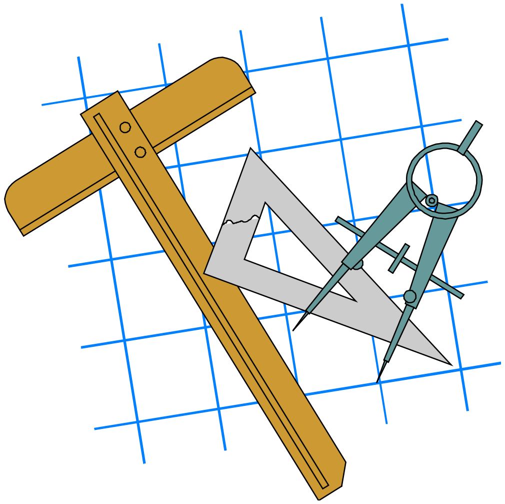
A cinema has M x N seats (M rows and N columns of seats).

Ticket price is calculated based on seat as follow:

- Seats at central row have max price.
- Seats at rows farther than central row have discount price ($0.5 for 1 row farther).

There are two types of cinemas:

- Standard cinema: price at central row $10.
- VIP cinema: price at central row $15, (discount 20% price on Thursday).



Practice

- Practice 8.5:

Construct class **StandardCinema** and **VIPCinema** as follow:

- Initialize a cinema with M x N seats.

- Check if a seat is empty.

- Tell price of a seat.

- Book a seat.

- Calculate total prices of sold tickets.