fit@hcmus

Object-Oriented Programming

Multiple Inheritance

Tran Duy Hoang

Topics Covered

- Multiple inheritance
- Ambiguous member access
- Diamond problem
- Interface-style multiple inheritance

Multiple inheritance

- When a class has 2 or more direct base classes, it is called multiple inheritance.

```
class B {
public:
    void draw() {}
};

class C {
public:
    void cal() {}
};
```

```
class A: public B, public C {
public:
    void doSth() {}
};

int main() {
    A a;
    a.draw();
    a.cal();
    a.doSth();
}
```

Multiple inheritance

- C++ class can inherit from multiple base classes even if they have members (data or functions) with the same name.

```
class B {
public:
    void doSth() {}
};

class C {
public:
    void doSth() {}
};
```

```
class A: public B, public C {
public:
    void doSth() {}
};

int main() {
    A a;
    a.doSth();        //error: ambiguous
    a.B::doSth();     // OK
    a.C::doSth();     // OK
}
```

Multiple inheritance

- Virtual functions work as usual

```
class B {
public:
    virtual void draw() = 0;
};

class C {
public:
    virtual void cal() = 0;
};
```

```
class A: public B, public C {
public:
    void draw();      //override B::draw()
    void cal();       //override C::cal()
};
```

Ambiguous member access

- Function ambiguities from different base classes are not resolved based on function signatures

```
class B {
public:
    void doSth(int i) {}
};

class C {
public:
    void doSth(double f) {}
};
```

```
class A: public B, public C {
public:
    void doSth(char c) {}
};

int main() {
    A a;
    a.doSth(10);     //error: ambiguous
    a.doSth(5.2);    //error: ambiguous
    a.doSth('a');    //error: ambiguous
}
```

Ambiguous member access

- Fixes
  - Qualify the call (obj.A::start())

```
class B {
public:
    void doSth(int i) {}
};

class C {
public:
    void doSth(double f) {}
};
```

```
class A: public B, public C {
public:
    void doSth(char c) {}
};

int main() {
    A a;
    a.B::doSth(10);
    a.C::doSth(5.2);
    a.A::doSth('a');
}
```

Ambiguous member access

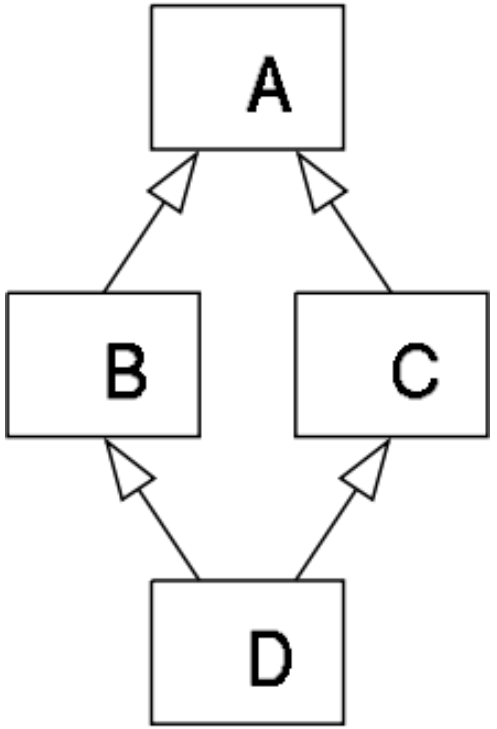
- Fixes
  - ***using*** declaration can bring the functions into a common scope

```
class B {
public:
    void doSth(int i) {}
};

class C {
public:
    void doSth(double f) {}
};
```

```
class A: public B, public C {
    using B::doSth;
    using C::doSth;
    void doSth(char c);
};

int main() {
    A a;
    a.doSth(10); // B::doSth(int)
    a.doSth('a'); // A::doSth(char)
    a.doSth(5.2); // C::doSth(double)
}
```

Diamond problem

- When two base classes share a common base and a derived class inherits from both



→ D ends up containing two separate A subobjects: one via B, one via C

Diamond problem

- Have two copies of A::doSth()

```
class A {
public:
    void doSth() {}
};

class B : public A {};

class C : public A {};

class D : public B, public C {};
```

```
int main() {
    D d;
    d.doSth();        //error: ambiguous
    d.B::doSth();     // replicated
    d.C::doSth();     // replicated
}
```

Diamond problem

- The standard fix: virtual inheritance
  - D has only 1 class virtual A

```
class A {
public:
    void doSth() {}
};

class B : public virtual A {};

class C : public virtual A {};

class D : public B, public C {};
```

```
int main() {
    D d;
    d.doSth();        // single Base
}
```

Interface-style multiple inheritance

- A class inherits from multiple “interfaces”—base classes that declare pure virtual functions (and typically have no data)

```
class Printable {
public:
    virtual void print() const = 0;
    virtual ~Printable() = default;
};

class Serializable {
public:
    virtual std::string toJson() const = 0;
    virtual ~Serializable() = default;
};
```

```
class Report : Printable, Serializable {
public:
    void print() const override;
    std::string toJson() const override;
};
```