fit@hcmus

Object-Oriented Programming

Inheritance

Tran Duy Hoang

Topics Covered

- Introduction to Inheritance
- Types of Inheritance
- Method Overriding
- IS-A & HAS-A Relationships

Topics Covered

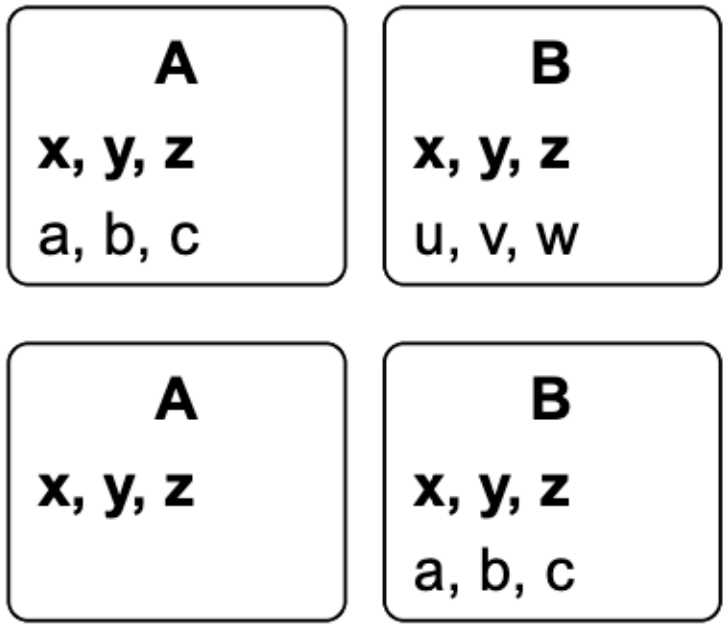
- **Introduction to Inheritance**
- Types of Inheritance
- Method Overriding
- IS-A & HAS-A Relationships

Redundency Problem

- Two classes have same information
  - Sharing: $A \cap B \neq \emptyset$
  - Extention: $B = A + \varepsilon$

- This can lead to:
  - More bugs
  - Harder maintenance
  - Slower development
  - Poor readability

→ **Reusability**

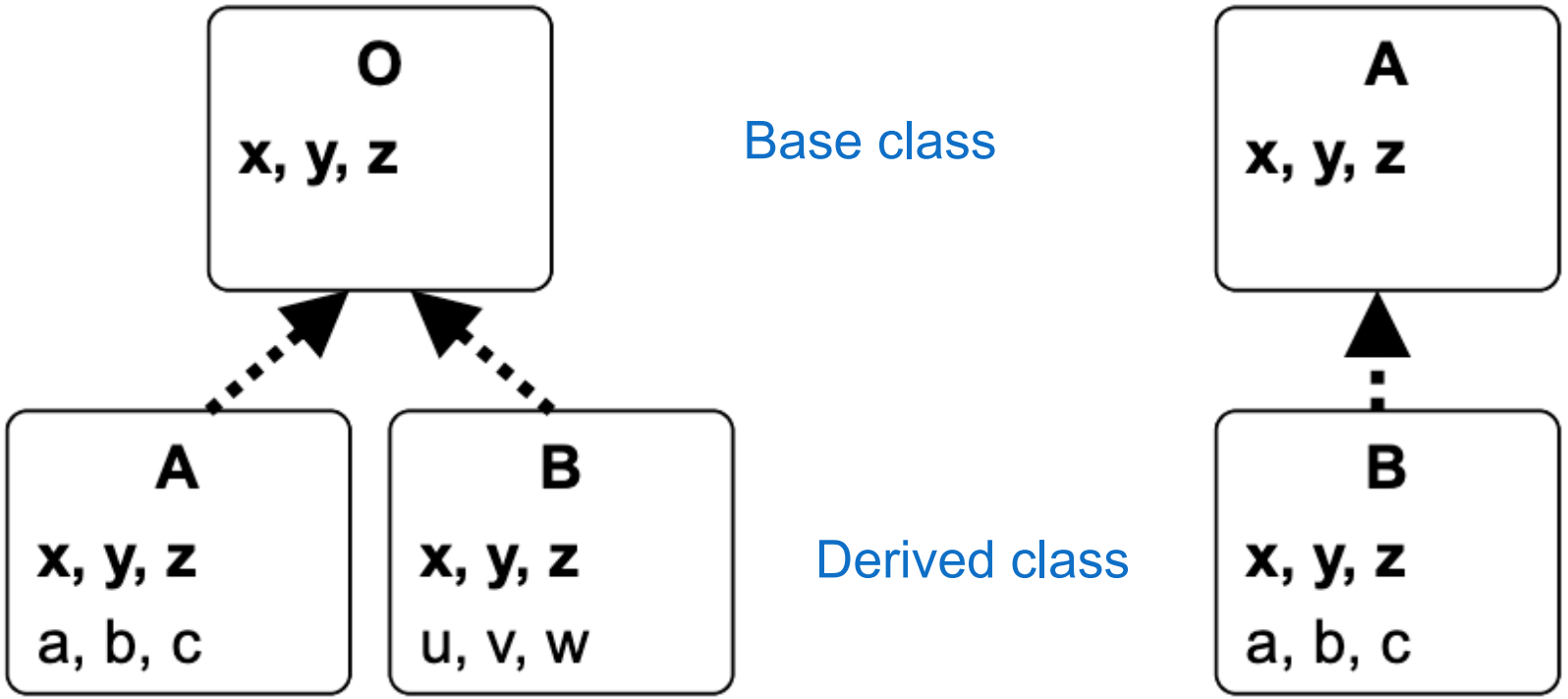


Inheritance

- Allows a class inherit attributes and methods from another class

- Base class:
  - original class that contains common attributes and methods
  - called Superclass or Parent class
- Derived class:
  - new class that inherits from the base class
  - called Subclass or Child class

Inheritance



Syntax

- In C++, the inheritance is described as:

```
class <Derived Class> : <TypeOfInheritance> <Base class>
{
private:
    <private attribute/methods>
protected:
    <protected attribute/methods>
public:
    <public attributes/methods>
};
```

Types of inheritance: *public*, *protected*, *private*

Protected Scope

- Members marked protected
  - can be accessed in derived classes
  - are not accessible outside the class

```
class A
{
protected:
    int t;
};
```

```
class B: public A
{
private:
    void test1(){ cout << t;}
protected:
    void test2(){ cout << t;}
public:
    void test3(){ cout << t;}
};
```

Example: Inheritance [1/2]

```
// Base class
class Student
{
protected:
    string name;
    int id;
    float gpa;

public:
    Student(string name, int id, float gpa);
    string getGrade(); // A, B, C, D
};

Student::Student(string name, int id, float gpa) {}
```

```
// Derived class
class ExchangeStudent : public Student
{
private:
    string country;
    int duration; // months

public:
    ExchangeStudent(string name, int id,
            float gpa, string country, int duration);
    void setDuration(int duration);
};

ExchangeStudent::ExchangeStudent(string name,
int id, float gpa, string country, int duration) :
Student(name, id, gpa) {}
```

Example: Inheritance [2/2]

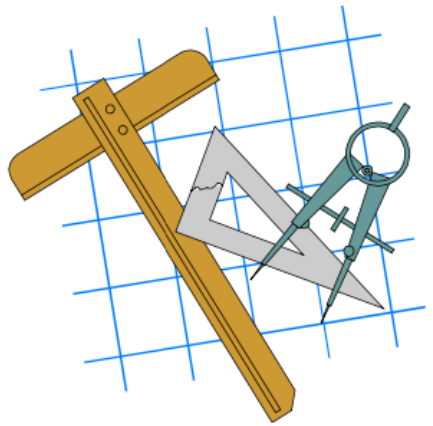
```
int main()
{
    // use Student
    Student student1("Alice", 101, 3.8);
    string grade1 = student1.getGrade();

    // use ExchangeStudent
    ExchangeStudent student2("Liam", 202, 3.9, "Germany", 6);
    string grade2 = student2.getGrade();
    student2.setDuration(9);

    // …
}
```

Exercise: Inheritance

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.

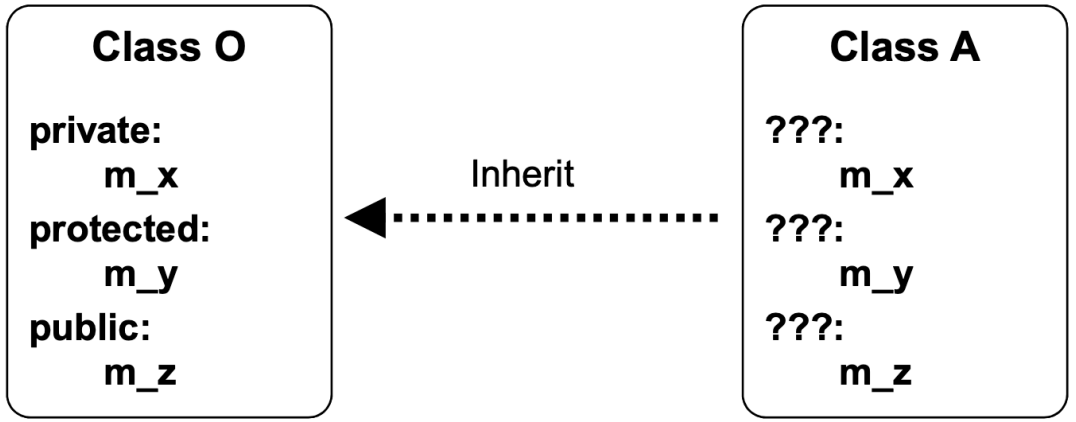


Topics Covered

- Introduction to Inheritance
- **Types of Inheritance**
- Method Overriding
- IS-A & HAS-A Relationships

Access Control in Inheritance

- Class A inherits from class O
  - A inherits all attributes and methods from O
  - Do scopes change during inheritance?



Decided by inheritance type

Inheritance Types

Inheritance Type	Syntax
Public	`class B : public A`
Protected	`class B : protected A`
Private	`class B : private A`

Access Behavior Table

Scope	public inheritance	protected inheritance	private inheritance
public	public	protected	private
protected	protected	protected	private
private	***inaccessible***	***inaccessible***	***inaccessible***

Exercise: Access Control

```
class Base {
public:
    int pub;
protected:
    int prot;
private:
    int priv;
};

class PublicDerived : public Base {
public:
    void access() {
        pub = 1;    // Line A
        prot = 2;   // Line B
        priv = 3;   // Line C
    }
};
```

```
class ProtectedDerived : protected Base {
public:
    void access() {
        pub = 1;    // Line D
        prot = 2;   // Line E
        priv = 3;   // Line F
    }
};

class PrivateDerived : private Base {
public:
    void access() {
        pub = 1;    // Line G
        prot = 2;   // Line H
        priv = 3;   // Line I
    }
};
```

```
int main() {
    PublicDerived p1;
    p1.pub = 10;    // Line J

    ProtectedDerived p2;
    p2.pub = 10;    // Line K

    PrivateDerived p3;
    p3.pub = 10;    // Line L

    return 0;
}
```

Topics Covered

- Introduction to Inheritance
- Types of Inheritance
- **Method Overriding**
- IS-A & HAS-A Relationships

Method Overriding

- **Re-define** a base class method in a derived class
- Derived class method must have **same signature**
- Method overriding vs. Method overloading?

Example: Method Overriding [1/2]

```
// Base class
class Student
{
protected:
    string name;
    int id;
    float gpa;

public:
    Student(string name, int id, float gpa);
    string getGrade(); // A, B, C, D
    void displayInfo();
};

void Student::displayInfo() {}
```

```
// Derived class
class ExchangeStudent : public Student
{
private:
    string country;
    int duration; // months

public:
    ExchangeStudent(string name, int id,
            float gpa, string country, int duration);
    void setDuration(int duration);
    void displayInfo();
};

void ExchangeStudent::displayInfo() {}
```

Example: Method Overriding [2/2]

```
int main()
{
    // use Student
    Student student1("Alice", 101, 3.8);
    string grade1 = student1.getGrade();
    student1.displayInfo();

    // use ExchangeStudent
    ExchangeStudent student2("Liam", 202, 3.9, "Germany", 6);
    string grade2 = student2.getGrade();
    student2.setDuration(9);
    student2.displayInfo();
}
```

Subtle Problem

When a derived class overrides one overloaded method in the base class, it hides the other overloaded methods.

```
class A {
public:
    void test();
    void test(int);
    void test(int, int);
};

class B : public A
{
public:
    void test(int);
};
```

```
int main()
{
    B b;
    int x, y;

    b.test(x); // OK
    b.test(x, y); // error
}
```

Subtle Problem

Solution: Use "using" to bring base overloads into scope

```
class A {
public:
    void test();
    void test(int);
    void test(int, int);
};

class B : public A
{
public:
    using A::test;
    void test(int);
};
```

```
int main()
{
    B b;
    int x, y;

    b.test(x); // OK
    b.test(x, y); // OK
}
```

Topics Covered

- Introduction to Inheritance
- Types of Inheritance
- Method Overriding
- **IS-A & HAS-A Relationships**

IS-A & HAS-A Relationships

- IS-A Relationship (Inheritance)
  - A derived class **is a type of** the base class

- Examples
  - A ExchangeStudent is a Student
  - A Dog is an Animal
  - A Car is a Vehicle

```
// base class
class Student {};

// ExchangeStudent IS-A Student
class ExchangeStudent : public Student {};
```

IS-A & HAS-A Relationships

- HAS-A Relationship (Composition or Aggregation)
  - One class **has** another class as a member
- Examples
  - A Student has an Address
  - A Car has an Engine
  - A Book has an Author

```
class Address {
private:
    string street;
    string city;
    string state;
};

class Student {
private:
    Address addr;  // Student HAS-A Address
};
```

Example: IS-A vs. HAS-A

#	Class A	Class B	Relationship
1	Car	Engine	
2	Engine	FuelInjector	
3	Professor	Person	
4	Library	Book	
5	Laptop	Device	
6	House	Room	
7	Doctor	Hospital	
8	Rectangle	Shape	
9	Phone	Camera	
10	Teacher	Department	
11	ElectricCar	Car	
12	Employee	Company	
13	School	Principal	
14	Driver	License	

Exercise 5.1

- Implement Address, Student, and ExchangeStudent classes
  - Address
    - Attribute: street, city, state
    - Method: display (prints full address)
  - Student
    - Attribute: name, student ID, gpa, address
    - Method: get grade (returns a letter grade based on GPA), display (prints student info)
  - ExchangeStudent (inherits from Student)
    - Attribute: home country, exchange duration (months)
    - Method: constructor (parameterized), update duration, display (prints exchange student info)

Exercise 5.2

- Implement Client, Account, and SavingAccount classes
  - Client
    - Attribute: name, client ID
    - Method: display (prints client info)
  - Account
    - Attribute: client, account number, balance
    - Method: get balance, deposit (balance += amount), withdraw (balance -= amount), display (prints account info)
  - SavingAccount (inherits from Account)
    - Attribute: interest rate
    - Method: apply interest (balance += balance * interestRate), display (prints saving account info)