

Bài 7
OOP -
Tính Kế thừa và
Đa hình

Biên soạn: **Huỳnh Thành Lộc**

Cập nhật: Tháng 01/2022

1

Nội dung (02 buổi = 06 tiết)

- ⌚ Tính Kế thừa
- ⌚ Tính Đa hình
- ⌚ Bài tập vận dụng

2

1

TÍNH KẾ THỪA

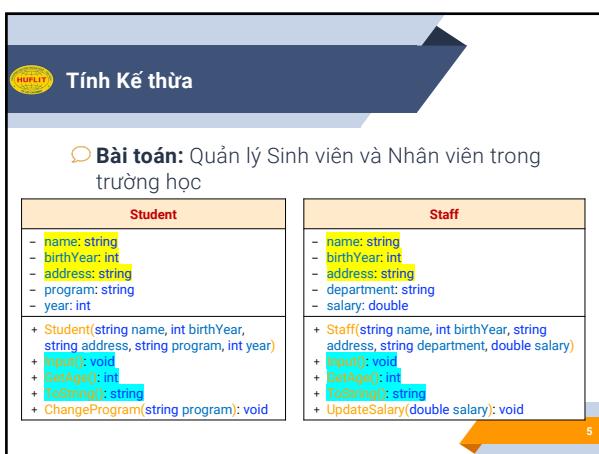
3

3

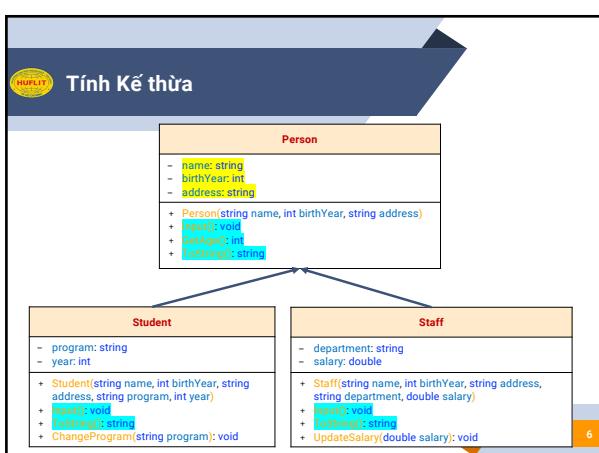
1250074 - Kỹ thuật lập trình



4



5



6

1250074 - Kỹ thuật lập trình

Tính Kế thừa

- Tính Kế thừa (*Inheritance*): cho phép định nghĩa một lớp:
 - Tái sử dụng (kế thừa) các thành phần của một lớp đã được định nghĩa trước
 - Thêm các thành phần riêng của lớp đó

The diagram illustrates inheritance. At the top is a box labeled 'Person'. Two arrows point down to boxes labeled 'Student' and 'Staff'. To the right of 'Person' is a callout box containing two points:

- Kế thừa các thành phần từ lớp Person
- Thêm các thành phần riêng của mỗi lớp

A small orange arrow at the bottom right points to the number '7'.

7

Tính Kế thừa

- Lớp Cơ sở (*Base class*): là lớp mà các thành phần của nó được lớp khác kế thừa
- Lớp Dẫn xuất (*Derived class*): là lớp kế thừa các thành phần của lớp cơ sở

The diagram shows 'Person' at the top with an arrow pointing down to 'Student' and 'Staff'. Callout boxes next to each class label indicate they are 'Base class' and 'Derived class' respectively. A small orange arrow at the bottom right points to the number '8'.

8

Tính Kế thừa

- C# chỉ hỗ trợ **đơn kế thừa** (một lớp dẫn xuất chỉ có thể kế thừa trực tiếp từ một lớp cơ sở)
- Tuy nhiên, việc kế thừa có tính chất bắc cầu

The diagram shows 'Person' at the top, which has arrows pointing down to 'Student' and 'Employee'. 'Employee' then has arrows pointing down to 'Staff' and 'Faculty'. Callout boxes indicate:

- Lớp Employee kế thừa các thành phần từ lớp Person
- Lớp Staff và Faculty kế thừa các thành phần từ lớp Employee và lớp Person

A small orange arrow at the bottom right points to the number '9'.

9

1250074 - Kỹ thuật lập trình

Tính Kế thừa

- Nguyên tắc trong kế thừa
 - Lớp dẫn xuất kế thừa tất cả các thành phần của lớp cơ sở, **nhưng chỉ có thể truy xuất** các thành phần có access modifier là **public** hoặc **protected**
 - Lớp dẫn xuất có thể **hiệu chỉnh** các phương thức của lớp cơ sở → Tính **Đa hình**

10

10

Tính Kế thừa

```
classDiagram
    class Person {
        -name: string
        -birthYear: int
        -address: string
        +Person(string name, int birthYear, string address)
        +ToString(): string
        +GetAge(): int
        +ToString(string)
    }
    class Student {
        -program: string
        -year: int
        +Student(string name, int birthYear, string address, string program, int year)
        +ToString(): void
        +ToString(string)
        +ChangeProgram(string program): void
    }
    class Staff {
        -department: string
        -salary: double
        +Staff(string name, int birthYear, string address, string department, double salary)
        +ToString(): void
        +ToString(string)
        +UpdateSalary(double salary): void
    }
    Person <|-- Student
    Person <|-- Staff
```

11

11

Tính Kế thừa

- Cú pháp:

```
[access modifier] class <ClassName> : <BaseClassName>
{
    ...
}
```

12

12

1250074 - Kỹ thuật lập trình

Tính Kế thừa

```
public class Person
{
    // Attributes
    private string name;
    private int birthYear;
    private string address;

    // Constructors
    public Person(string name, int birthYear, string address)
    {
        this.name = name;
        this.birthYear = birthYear;
        this.address = address;
    }

    // Methods
    public void Input() { . . . }
    public int GetAge() { . . . }
    public string ToString() { . . . }
}
```

13

13

Tính Kế thừa

```
public class Student : Person
{
    // Attributes
    private string program;
    private int year;

    // Constructors
    public Student(string name, int birthYear, string address, string program, int year)
        : base(name, birthYear, address)
    {
        this.program = program;
        this.year = year;
    }

    // Methods
    public void ChangeProgram(string program) { . . . }
}
```

14

Tính Kế thừa

```
public class Staff : Person
{
    // Attributes
    private string department;
    private int salary;

    // Constructors
    public Staff(string name, int birthYear, string address, string department, int salary)
        : base(name, birthYear, address)
    {
        this.department = department;
        this.salary = salary;
    }

    // Methods
    public void UpdateSalary(int salary) { . . . }
}
```

15

1250074 - Kỹ thuật lập trình

Tính Kế thừa

```
public class Program
{
    public static void Main(string args)
    {
        Student sv = new Student("Nguyễn An", 2003, "828 Su Van Hanh", "CNTT", 1);
        Console.WriteLine(sv.GetAge());
        Console.WriteLine(sv.ToString());

        Student gv = new Student("Nguyễn Bình", 1980, "304 Cao Thang", "CNTT", 1000);
        Console.WriteLine(gv.GetAge());
        Console.WriteLine(gv.ToString());
    }
}
```

Các đối tượng `sv` và `gv` vẫn có phương thức `GetAge()` và `ToString()` được kế thừa từ class `Person`

16

16

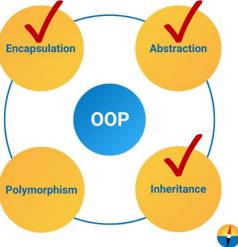
Tính Kế thừa

Ưu điểm <ul style="list-style-type: none">■ Cho phép chia sẻ các thông tin tin chung, tăng khả năng tái sử dụng, hạn chế viết lặp lại code■ Dễ dàng nâng cấp hệ thống	Nhược điểm <ul style="list-style-type: none">■ Chức năng được kế thừa hoạt động chậm hơn do nó được thực hiện gián tiếp từ lớp cơ sở■ Một thay đổi trong lớp cha sẽ ảnh hưởng đến tất cả các lớp con (có thể vừa là ưu điểm, vừa là nhược điểm)
---	---

17

17

Các đặc trưng của OOP



18

18

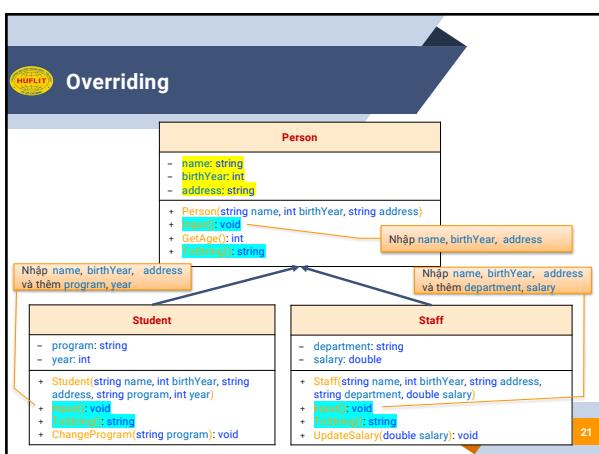
1250074 - Kỹ thuật lập trình



19

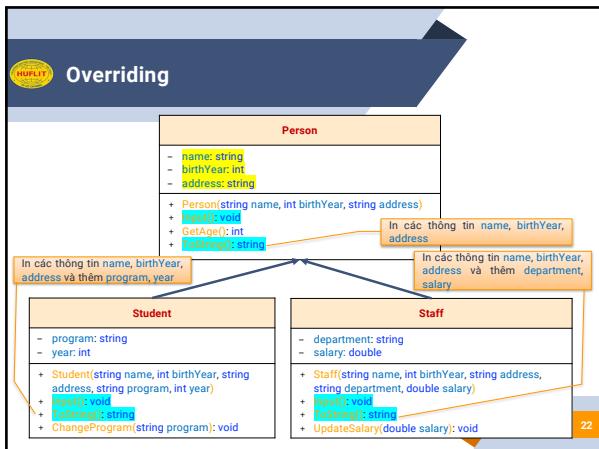


20

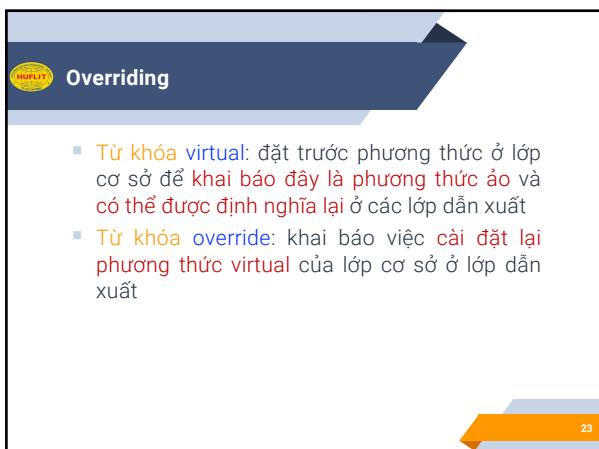


21

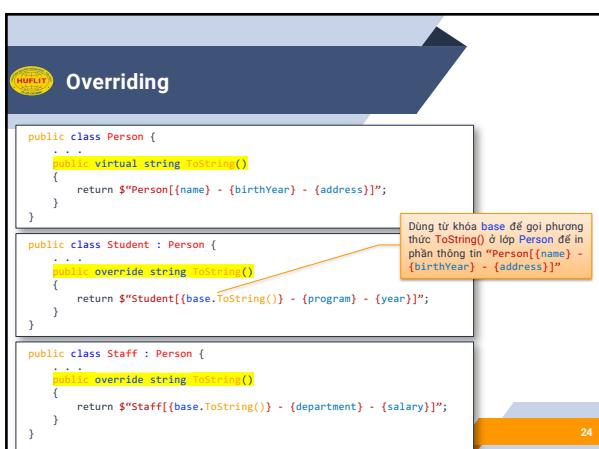
1250074 - Kỹ thuật lập trình



22



23



24

1250074 - Kỹ thuật lập trình

Overriding

○ Nhận xét:

- Phương thức **virtual** không bắt buộc phải được **override** ở các lớp dẫn xuất
- Nếu không thực hiện **override** lại các phương thức **virtual** thì đối tượng của lớp dẫn xuất sẽ thực hiện theo định nghĩa ở lớp cơ sở gần nhất

25

25

Overriding

```
class A {  
    public virtual void PrintInfo() { Console.WriteLine("Class A"); }  
}  
class B : A {  
    public override void PrintInfo() { Console.WriteLine("Class B"); }  
}  
class C : B {  
}  
public class Program  
{  
    public static void Main(string args)  
    {  
        B objB = new B();  
        C objC = new C();  
        objB.PrintInfo();  
        objC.PrintInfo();  
    }  
}
```

26

26

Overriding

- Từ khóa **sealed**: để ngăn chặn các lớp dẫn xuất tiếp theo **overriding** phương thức **virtual**

```
class A  
{  
    public virtual void DoWork() { . . . }  
}  
class B : A  
{  
    public sealed override void DoWork() { . . . }  
}  
class C : B  
{  
}
```

Từ đây các lớp dẫn xuất của lớp B không thể override phương thức **DoWork()** được nữa

27

27

1250074 - Kỹ thuật lập trình

Overloading

- Nạp chồng có 2 dạng:
 - Nạp chồng phương thức (method overloading): định nghĩa các phương thức có cùng tên nhưng khác nhau về số lượng và/hoặc kiểu dữ liệu của các tham số
 - Nạp chồng toán tử (operator overloading): định nghĩa chức năng của các toán tử có sẵn trong C# trên kiểu dữ liệu do người dùng định nghĩa (class).

28

28

Method Overloading

```
class Math
{
    public int Add(int a, int b)
    {
        return a + b;
    }
    public double Add(double a, double b)
    {
        return a + b;
    }
    public int Add(int a, int b, int c)
    {
        return a + b + c;
    }
}

public class Program
{
    public static void Main(string args)
    {
        Math m = new Math();
        int sum1 = m.Add(5, 9);
        double sum2 = m.Add(2.4, 9.8);
        int sum3 = m.Add(5, 9, 6);
    }
}
```

29

Operator overloading

💡 Vấn đề: Thực hiện phép toán cộng 2 phân số

```
public class Fraction
{
    // Attributes
    private int numerator;
    private int denominator;

    // Constructors
    public Fraction(int numerator, int denominator)
    {
        int g = GCD(numerator, denominator);
        this.numerator = numerator/g;
        this.denominator = denominator/g;
    }

    // Methods
    public Fraction Add(Fraction f)
    {
        int n = numerator * f.denominator + denominator * f.numerator;
        int d = denominator * f.denominator;
        return new Fraction(n, d);
    }
}

public class Program
{
    public static void Main(string args)
    {
        Fraction f1 = new Fraction(1, 2);
        Fraction f2 = new Fraction(3, 4);

        Fraction sum1 = f1.Add(f2);
        Fraction sum2 = f1 + f2;
    }
}
```

Muốn gọi được như thế này cần định nghĩa toán tử "+" đối với kiểu dữ liệu Fraction → Nạp chồng toán tử (operator overloading)

30

30

1250074 - Kỹ thuật lập trình

Operator overloading

- Cú pháp: sử dụng từ khóa "operator"
 - Bắt buộc dùng từ khóa public và static
 - Nếu là toán tử 1 ngôi (+, -, !, ...) thì cần 1 tham số.
Nếu là toán tử 2 ngôi (+, -, *, /, %, >, <, >=, <=, !=, ...) thì cần 2 tham số
 - Trong tham số, phải có ít nhất 1 tham số thuộc kiểu dữ liệu đang định nghĩa operator

```
public static <return-type> operator <operator-sign> (parameters)
```

31

31

Operator overloading

- Nạp chồng toán tử 2 ngôi "+" 2 phân số

```
public class Fraction{  
    // Attributes  
    private int numerator;  
    private int denominator;  
  
    // Constructors  
    public Fraction(int numerator, int denominator)  
    {  
        int g = GCD(numerator, denominator);  
        this.numerator = numerator/g;  
        this.denominator = denominator/g;  
    }  
  
    // Operator overloading  
    public static Fraction operator + (Fraction a, Fraction b)  
    {  
        int n = a.numerator * b.denominator + a.denominator * a.numerator;  
        int d = a.denominator * b.denominator;  
        return new Fraction(n, d);  
    }  
}  
  
public class Program  
{  
    public static void Main(string args)  
    {  
        Fraction f1 = new Fraction(1, 2);  
        Fraction f2 = new Fraction(3, 4);  
  
        Fraction sum2 = f1 + f2;  
    }  
}
```

32

32

Operator overloading

- Nạp chồng toán tử 1 ngôi "-" đổi dấu phân số

```
public class Fraction{  
    // Attributes  
    private int numerator;  
    private int denominator;  
  
    // Constructors  
    public Fraction(int numerator, int denominator)  
    {  
        int g = GCD(numerator, denominator);  
        this.numerator = numerator/g;  
        this.denominator = denominator/g;  
    }  
  
    // Operator overloading  
    public static Fraction operator - (Fraction a)  
    {  
        return new Fraction(- a.numerator, b.denominator);  
    }  
}  
  
public class Program  
{  
    public static void Main(string args)  
    {  
        Fraction a = new Fraction(1, 2);  
  
        a = -a;  
    }  
}
```

33

33

1250074 - Kỹ thuật lập trình

Operator overloading

- Lưu ý: nếu nạp chồng các toán tử quan hệ thì phải định nghĩa theo từng cấp, không nạp chồng đơn lẻ:
 - == và !=
 - > và <
 - >= và <=

34

34

Operator overloading

- Lưu ý: Các toán tử không thể overloading:
 - Toán tử kép: +, -, *, /, % (nhưng khi overloading các toán tử +, -, *, /, % thì C# sẽ thực hiện được các phép toán +=, -=, *=, /=, %=)
 - Toán tử kết hợp điều kiện: && và ||
 - Toán tử ép kiểu: (T)x → sẽ có cách định nghĩa riêng (conversion operator)

35

35

Conversion Operators

○ **Vấn đề:** Chuyển đổi giữa từ Fraction → double và từ int → Fraction

```
public class Fraction {  
    // Attributes  
    private int numerator;  
    private int denominator;  
  
    // Constructors  
    public Fraction(int numerator, int denominator)  
    {  
        int g = GCD(numerator, denominator);  
        this.numerator = numerator/g;  
        this.denominator = denominator/g;  
    }  
    public Fraction(int numerator)  
    {  
        this.numerator = numerator;  
        this.denominator = 1;  
    }  
    // Methods  
    public double ToDecimal()  
    {  
        return (1.0 * numerator) / denominator;  
    }  
}
```

```
public class Program  
{  
    public static void Main(string args)  
    {  
        Fraction f1 = new Fraction(1, 2);  
        double d = f1.ToDecimal();  
        Fraction f2 = new Fraction(3);  
    }  
}
```

Cách viết ngắn gọn:
• double d = (double)f1;
• Fraction f2 = 3;

36

36

1250074 - Kỹ thuật lập trình

Conversion Operators

- Có 2 dạng toán tử chuyển kiểu:
 - Chuyển kiểu **ngầm định (implicit)**: trình biên dịch sẽ **tự động chuyển đổi** từ một kiểu dữ liệu này sang kiểu dữ liệu khác
 - Chuyển kiểu **tường minh (explicit)**: lập trình viên **phải khai báo** để chuyển đổi từ một kiểu dữ liệu này sang kiểu dữ liệu khác

37

37

Conversion Operators

- Cú pháp: sử dụng từ khóa "operator" và "implicit" / "explicit"

```
public static implicit/explicit operator <return-type> (parameters)  
{  
    ...  
}
```

38

38

Conversion Operators

○ **Vấn đề:** Chuyển đổi giữa từ `Fraction` → `double` và từ `int` → `Fraction`

```
public class Fraction {  
    // Attributes  
    private int numerator;  
    private int denominator;  
  
    // Constructors  
    public Fraction(int numerator, int denominator)  
    {  
        int g = GCD(numerator, denominator);  
        this.numerator = numerator/g;  
        this.denominator = denominator/g;  
    }  
  
    // Conversion operators  
    public static explicit operator double (Fraction f)  
    {  
        return (1.0 * numerator) / denominator;  
    }  
    public static implicit operator Fraction (double d)  
    {  
        return new Fraction(d,1);  
    }  
}  
  
public class Program  
{  
    public static void Main(string args)  
    {  
        Fraction f1 = new Fraction(1, 2);  
        double d = (double)f1;  
  
        Fraction f2 = 3;  
    }  
}
```

39

39

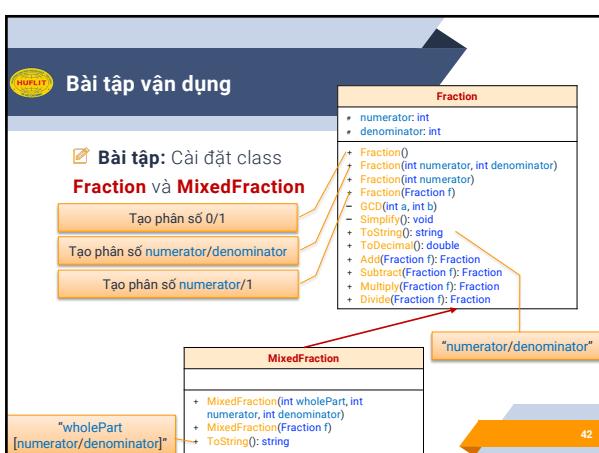
1250074 - Kỹ thuật lập trình



40



41



42

 **Bài tập vận dụng**

 **Bài tập:** Cài đặt class **Fraction** và **MixedFraction**

- Nạp chồng toán tử một ngôi “-” để đổi dấu phân số
- Nạp chồng các toán tử hai ngôi “+, -, *, /” để thực hiện các phép toán giữa 2 phân số
- Nạp chồng các toán tử so sánh 2 phân số “==, !=, >, <, >=, <=”

43

43





Q & A

*“One who never asks
either knows everything or nothing”*

Malcolm S. Forbes

44

44
