Trường Cao đẳng Công nghệ Thông tin TP.HCM

Khoa Công nghệ Thông tin – Điện tử

Chương 6:

Đa Hình

Giảng viên: Hà Mỹ Trinh

Email: trinhhm@itc.edu.vn

Nội dung

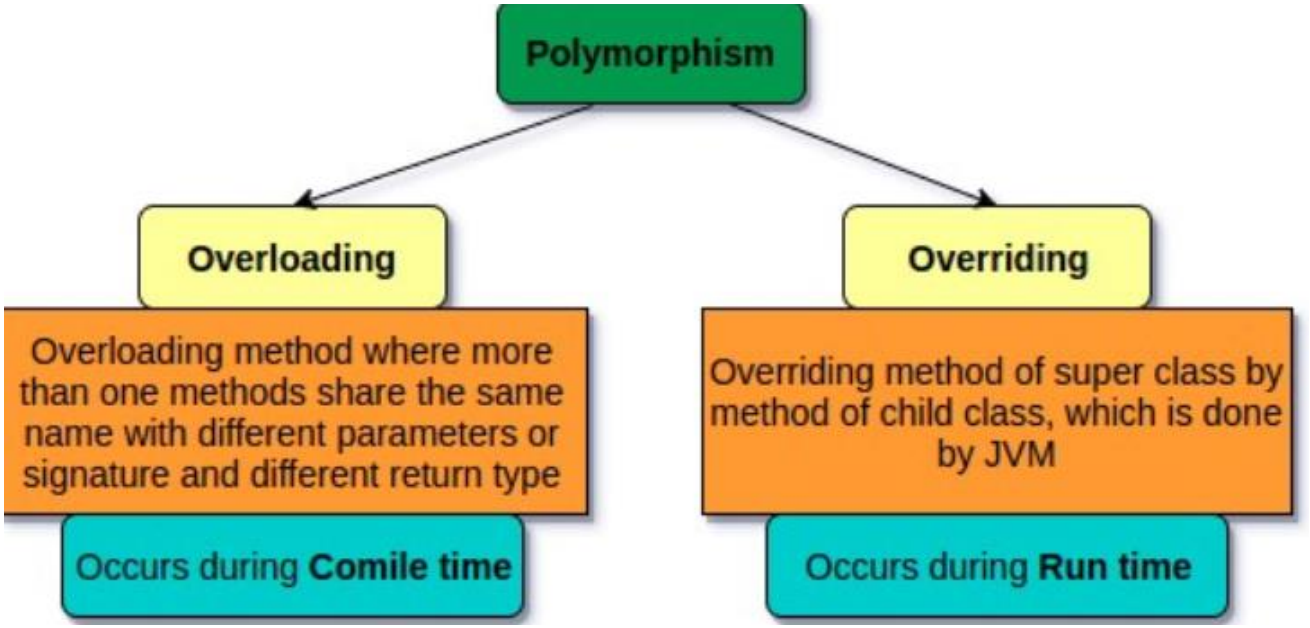
1. Đa hình
2. Upcasting và downcasting
3. Liên kết tĩnh và liên kết động

1. Đa hình (polymorphism): nạp chồng phương thức + ghi đè phương thức

- Polymorphism: nhiều hình thức thực hiện, nhiều kiểu tồn tại
- Overriding – ghi đè phương thức: thực hiện tính đa hình trong lập trình hướng đối tượng (một hành vi được thể hiện với các hình thái khác nhau)
- Gọi phương thức bị ghi đè được quyết định lúc chạy chương trình (runtime) chứ không phải lúc biên dịch chương trình (compile time)
- Đa hình trong lập trình
  - Đa hình phương thức:
    - Phương thức trùng tên, phân biệt bởi danh sách tham số.
  - Đa hình đối tượng
    - Nhìn nhận đối tượng theo nhiều kiểu khác nhau
    - Các đối tượng khác nhau cùng đáp ứng chung danh sách các thông điệp có giải nghĩa thông điệp theo cách thức khác nhau.

1. Đa hình (polymorphism)

- Ghi đè phương thức
  - Phương thức của lớp con có cùng tên và đặc điểm như phương thức của lớp cha (có cùng danh sách tham số và có cùng kiểu trả về).
- Khi phương thức ghi đè được triệu hồi từ lớp con, ta luôn có phiên bản phương thức ở lớp con được thực hiện. Xét chương trình Override sau:



1. Đa hình (polymorphism)

```
package chuong5;
class A {
    int i, j;
    A(int a, int b) {
      i = a;
      j = b;
    }

   // hiển thị i và j
    void show() {
        System.out.println("i and j: " + i + " " + j);
    }
}

class B extends A {
    int k;
    B(int a, int b, int c) {
        super(a, b);
        k = c;
    }
```

```
// hiển thị k – phương thức này ghi đè show()
của A
    void show() {
        System.out.println("k: " + k);
    }
}

public class Override {
  public static void main(String args[]) {
    B subOb = new B(1, 2, 3);
    subOb.show(); // this calls show() in B
  }
}
```

Kết quả là: k:3

2. Up – casting và Down - casting

2.1 Chuyển đổi kiểu dữ liệu nguyên thủy

- Java tự động chuyển đổi kiểu khi
  - Kiểu dữ liệu tương thích
  - Chuyển đổi từ kiểu hẹp hơn sang kiểu rộng hơn

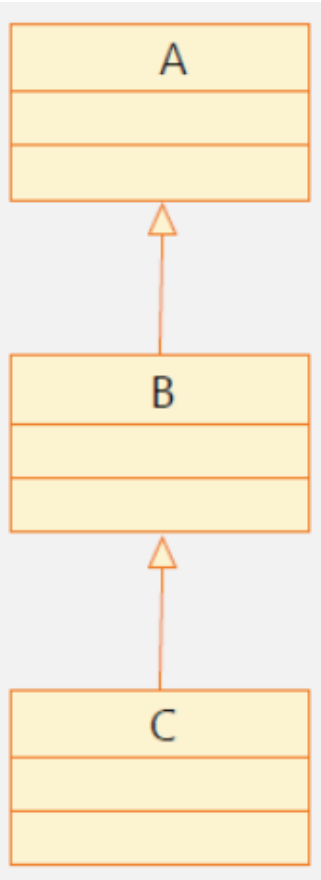
```
int i;
double d = i;
```

- Phải ép kiểu khi
  - Kiểu dữ liệu tương thích
  - Chuyển đổi từ kiểu rộng hơn sang kiểu hẹp hơn

```
int i;
~~byte b = i;~~ byte b = (byte)i;
```

2.1 Chuyển đổi kiểu dữ liệu tham chiếu

- Kiểu dữ liệu tham chiếu có thể được chuyển đổi kiểu khi
  - Kiểu dữ liệu tham chiếu (lớp) tương thích
    - Nằm trên cùng một cây phân cấp kế thừa
- Hai cách chuyển đổi
  - Up-casting
  - Down-casting



2.2. Up-casting

- Khi biến tham chiếu của lớp cha tham chiếu tới đối tượng của lớp con, thì đó là Upcasting

```
1  class A{}
2  class B extends A{}
```

```
1  A a=new B();//upcasting
```

Ví dụ về tính đa hình (polymorphism) trong lúc runtime

```java
class Bike {
    void run() {
        System.out.println("running");
    }
}

public class Splender extends Bike {
    void run() {
        System.out.println("running safely with 60km");
    }

    public static void main(String args[]) {
        Bike b = new Splender();// upcasting
        b.run();
    }
}
```

Kết quả:

```
running safely with 60km
```

Ví dụ về tính đa hình (polymorphism) trong lúc runtime

```
1  class Bank {
2      int getRateOfInterest() {
3          return 0;
4      }
5  }
6
7  class SBI extends Bank {
8      int getRateOfInterest() {
9          return 8;
10     }
11 }
12
13 class ICICI extends Bank {
14     int getRateOfInterest() {
15         return 7;
16     }
17 }
18
```

Ví dụ về tính đa hình (polymorphism) trong lúc runtime

```
19  class AXIS extends Bank {
20      int getRateOfInterest() {
21          return 9;
22      }
23  }
24
25  public class Test2 {
26      public static void main(String args[]) {
27          Bank b;
28          b = new SBI();
29          System.out.println("SBI Rate of Interest: " + b.getRateOfInterest());
30          b = new ICICI();
31          System.out.println("ICICI Rate of Interest: " + b.getRateOfInterest());
32          b = new AXIS();
33          System.out.println("AXIS Rate of Interest: " + b.getRateOfInterest());
34      }
35  }
```

Kết quả:

```
SBI Rate of Interest: 8
ICICI Rate of Interest: 7
AXIS Rate of Interest: 9
```

Ví dụ về tính đa hình (polymorphism) trong lúc runtime

```
class Polygon {
  //phương thức render của lớp Polygon
  public void render() {
    System.out.println("Rendering Polygon...");
  }
}
class Square extends Polygon {
  //ghi đè phương thức render
  public void render() {
  System.out.println("Rendering Square...");
  }
}
class Circle extends Polygon {
  //ghi đè phương thức render
  public void render() {
    System.out.println("Rendering Circle...");
  }
}
```

```
class Main {
  public static void main(String[] args) {
    // create an object of Square
    Square s1 = new Square();
    s1.render();
    // create an object of Circle
    Circle c1 = new Circle();
     c1.render();
  }
}
```

Kết quả:

Rendering Square...

Rendering Circle...

Ví dụ về tính đa hình (polymorphism) trong lúc runtime

```
1  class Bike{
2   int speedlimit=90;
3  }
4  class Honda3 extends Bike{
5   int speedlimit=150;
6
7   public static void main(String args[]){
8    Bike obj=new Honda3();
9    System.out.println(obj.speedlimit);//90
10 }
```

Kết quả: 90

Đa hình lúc runtime với thuộc tính: KHÔNG BỊ GHI ĐÈ

Ví dụ về tính đa hình (polymorphism) trong lúc runtime

```
1   class Animal {
2       void eat() {
3           System.out.println("eating");
4       }
5   }
6
7   class Dog extends Animal {
8       void eat() {
9           System.out.println("eating fruits");
10      }
11  }
12
13  class BabyDog extends Dog {
14      void eat() {
15          System.out.println("drinking milk");
16      }
public static void main(String args[]) {
    Animal a1, a2, a3;
    a1 = new Animal();
    a2 = new Dog();
    a3 = new BabyDog();
    a1.eat();
    a2.eat();
    a3.eat();
```

Kết quả:

```
eating
eating fruits
drinking Milk
```

Đa hình lúc runtime với kế thừa nhiều tầng

Ví dụ về tính đa hình (polymorphism) trong lúc runtime

```
public class Animal {
    public void eat() {
        System.out.println("eating...");
    }
}
public class Cat extends Animal{
    public void meow() {
        System.out.println("meowing...");
    }
}
public class ViDu_UpCasting_2 {
    public static void main(String[] args) {
        Cat meo = new Cat();//Khai bao doi tuong meo thuoc lop Cat
        meo.eat();//goi phuong thuc cua lop CHA Animal - tinh ke thua
        meo.meow();//goi phuong thuc cua lop Cat
```

Kết quả:

```
eating...
meowing...
```

Ví dụ về tính đa hình (polymorphism) trong lúc runtime

```
public class Animal {
    public void eat() {
        System.out.println("eating...");
    }
}
public class Cat extends Animal{
    public void meow() {
        System.out.println("meowing...");
    }
}
//Khai bao doi tuong dongvat1 thuoc lop Animal
//va tham chieu den doi tuong meo
Animal dongvat1 = meo; //Chuyển kiểu không tường minh
dongvat1.eat();//goi phuong thuc cua lop Animal
//dongvat1.meow(); //=>KHONG goi duoc phuong thuc meow() cua lop Cat
```

Kết quả: `eating...`

Ví dụ về tính đa hình (polymorphism) trong lúc runtime

```
public class Animal {
    public void eat() {
        System.out.println("eating...");
    }
}
public class Cat extends Animal{
    public void meow() {
        System.out.println("meowing...");
    }
}
//Khai bao doi tuong dongvat2 thuoc lop Animal
//va tham chieu den doi tuong meo
Animal dongvat2 = (Animal) meo; // Chuyển kiểu tường minh
dongvat2.eat();//goi phuong thuc cua lop Animal
//dongvat2.meow(); // KHONG goi duoc phuong thuc meow() cua lop Cat
```

Kết quả: `eating...`

2.3 Down - casting

- Down casting: đi xuống cây phân cấp thừa kế (move back down the inheritance hierarchy)
- Down casting là khả năng nhìn nhận một đối tượng thuộc lớp cơ sở như một đối tượng thuộc lớp dẫn xuất.
- Không tự động chuyển đổi kiểu → Phải ép kiểu.

Ví dụ về tính đa hình (polymorphism) trong lúc runtime

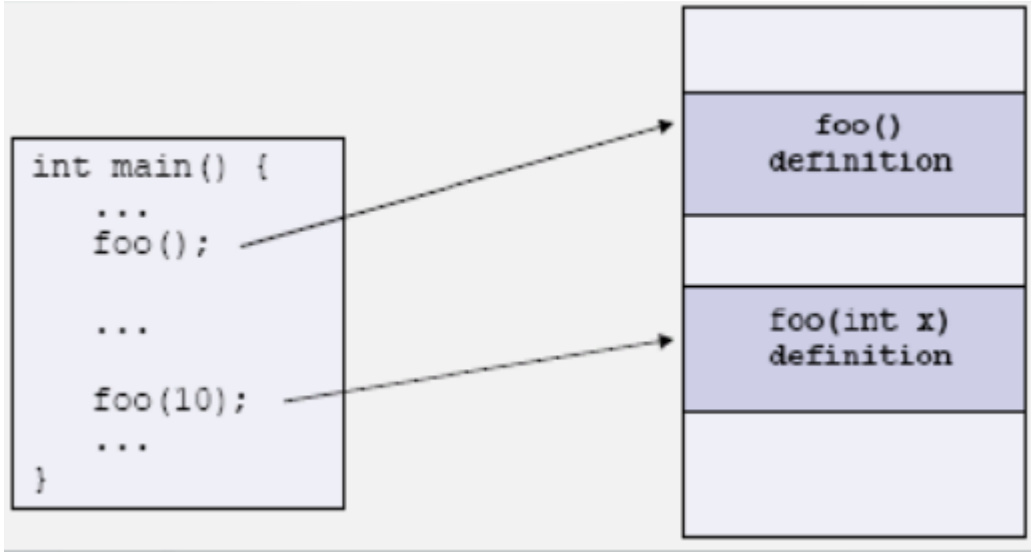
```
//Khai bao doi tuong dongvat1 thuoc lop Animal
//va tham chieu den doi tuong meo
Animal dongvat1 = meo; //Chuyển kiểu không tường minh
dongvat1.eat();//goi phuong thuc cua lop Animal
//dongvat1.meow(); //=>KHONG goi duoc phuong thuc meow() cua lop Cat
```

```
Animal dongvat2 = new Cat();
//dongvat3.meow(); //=> KHONG goi duoc phuong thuc meow() cua lop Cat
Cat meo2 = (Cat) dongvat2; //down-casting
meo2.meow();
```

Kết quả: `meowing...`

3. Liên kết tĩnh và liên kết động

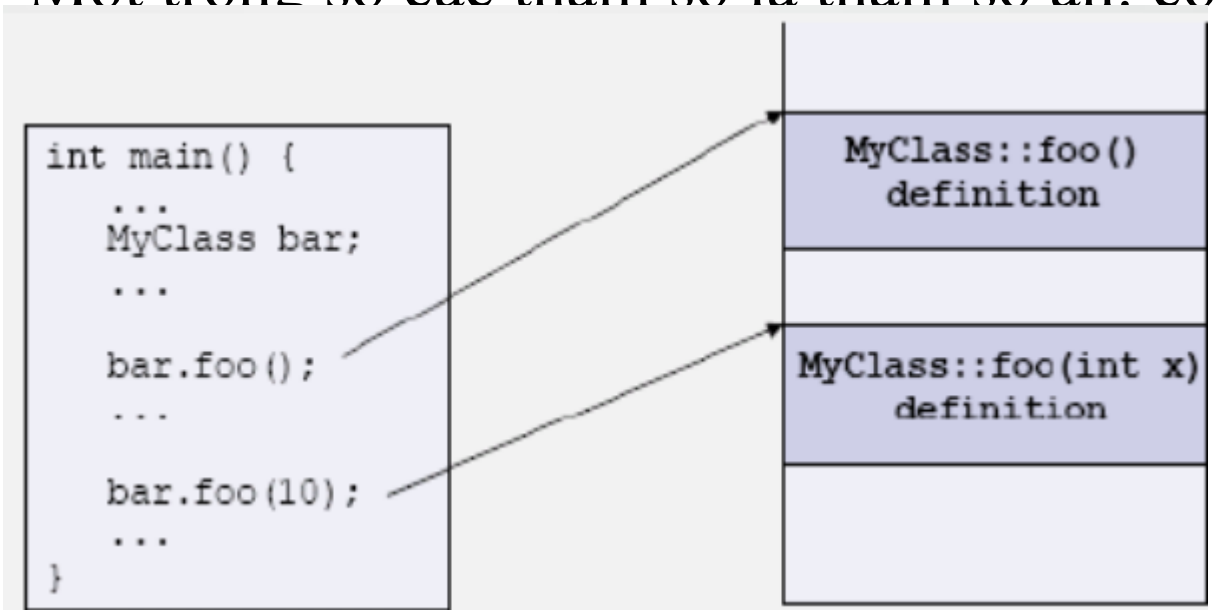
- Liên kết lời gọi hàm
- Liên kết lời gọi hàm (function call binding) là quy trình xác định khối mã hàm cần chạy khi một lời gọi hàm được thực hiện
- C: đơn giản vì mỗi hàm có duy nhất một tên
- C++: chồng hàm, phân tích chữ ký kiểm tra danh sách tham số.



3. Liên kết tĩnh và liên kết động

Liên kết lời gọi hàm trong ngôn ngữ hướng đối tượng

- Liên kết lời gọi phương thức
  - Đối với các lớp độc lập (không thuộc cây thừa kế nào), quy trình này gần như không khác với function call binding
    - So sánh tên phương thức, danh sách tham số để tìm định nghĩa tương ứng
    - Một trong số các tham số là tham số ẩn: con trỏ this



3.1 Liên kết tĩnh

- Liên kết tại thời điểm biên dịch
  - Early Binding/Compile-time Binding
  - Lời gọi phương thức được quyết định khi biên dịch, do đó chỉ có một phiên bản của phương thức được thực hiện
  - Nếu có lỗi thì sẽ có lỗi biên dịch
  - Ưu điểm về tốc độ
- C/C++ function call binding, và C++ method binding cơ bản đều là ví dụ của liên kết tĩnh (static function call binding)

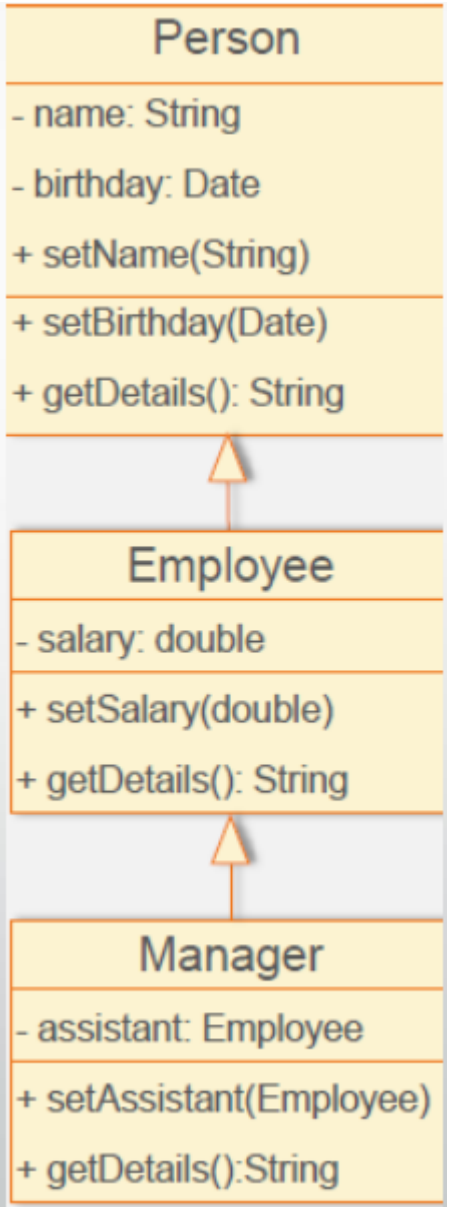
3.2 Liên kết động

- Lời gọi phương thức được quyết định khi thực hiện (run-time)
  - Late binding/Run-time binding
  - Phiên bản của phương thức phù hợp với đối tượng được gọi.
  - Java mặc định sử dụng liên kết động

3.2 Liên kết động

- Ví dụ:

```
public class Test {
  public static void main(String arg[]){
    Person p = new Person();
    // ...
    Employee e = new Employee();
    // ...
    Manager m = new Manager();
    // ...
    Person pArr[] = {p, e, m};
    for (int i=0; i< pArr.length; i++){
      System.out.println(
              pArr[i].getDetail());
    }
  }
}
```



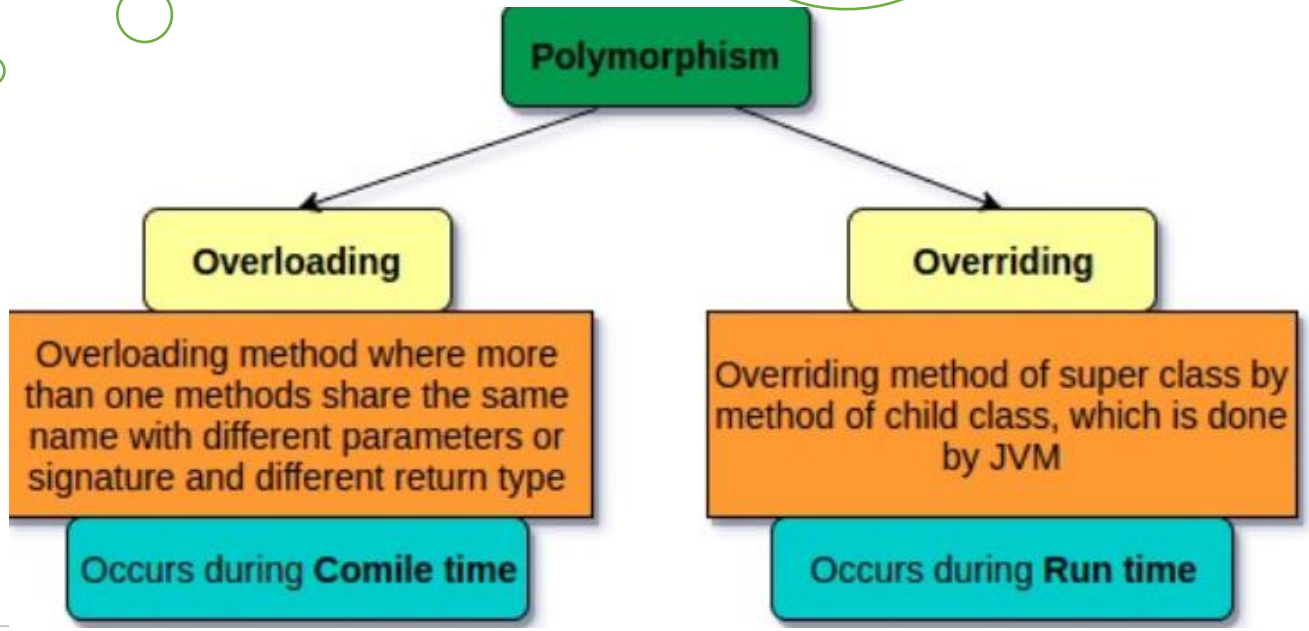
4. Toán tử Instanceof

- Kiểm tra xem một đối tượng có phải là thể hiện của một lớp nào đó không

```
public class Employee extends Person {}
public class Student extends Person {}

public class Test{
   public doSomething(Person e) {
      if (e instanceof Employee) {...
      } else if (e instanceof Student) {...
      } else {...
      }
   }
}
```

Tại sao sử dụng tính đa hình (polymorphism)



No.	Nạp chồng phương thức (overloading)	Ghi đè phương thức (overriding)
1)	Nạp chồng phương thức được sử dụng để giúp code của chương trình *dễ đọc hơn*.	Ghi đè phương thức được sử dụng để cung cấp *cài đặt cụ thể* cho phương thức được khai báo ở lớp cha.
2)	Nạp chồng được thực hiện bên *trong một class*.	Ghi đè phương thức xảy ra *trong 2 class* có quan hệ kế thừa.
3)	Nạp chồng phương thức thì *tham số phải khác nhau*.	Ghi đè phương thức thì *tham số phải giống nhau*.
4)	Nạp chồng phương thức là ví dụ về *đa hình lúc biên dịch*.	Ghi đè phương thức là ví dụ về *đa hình lúc runtime*.
5)	Trong java, nạp chồng phương thức không thể được thực hiện khi chỉ thay đổi kiểu giá trị trả về của phương thức. Kiểu giá trị trả về có thể giống hoặc khác. *Giá trị trả về có thể giống hoặc khác*, nhưng tham số phải khác nhau.	Giá trị trả về phải giống nhau.

Sự khác nhau giữa NẠP CHỒNG PHƯƠNG THỨC (overloading) & GHI ĐÈ PHƯƠNG THỨC (overriding)

```java
public class OverloadingExample {
    static int add(int a, int b) {
        return a + b;
    }

    static int add(int a, int b, int c) {
        return a + b + c;
    }
}
```

Nạp chồng phương thức

```java
class Animal {
    void eat() {
        System.out.println("eating...");
    }
}

class Dog extends Animal {
    void eat() {
        System.out.println("eating bread...");
    }
}
```

Ghi đè phương thức


Q&A