Chương 2.

Lập trình hướng đối tượng trong Java (tiếp)

Biến this, Phương thức setter và getter

- **Biến This**: Biến **this** là một biến ẩn tồn tại trong tất cả các lớp trong ngôn ngữ java. Một class trong Java luôn tồn tại một biến this, biến **this** được sử dụng trong khi chạy và tham khảo đến bản thân lớp chứa nó.
- **Phương thức setter**:
  Phương thức setter sử dụng biến this để truy cập vào thuộc tính các đối tượng, gán giá trị cho các đối tượng.
  - ✓ Mỗi phương thức setter là một phương thức với ngầm định là truy cập vào các biến private và gán giá trị cho nó.
  - ✓ Tên phương thức (setHoten, setLop, setDiemtb) do người lập trình tự đặt, thông thường là set…
  - ✓ Mỗi phương thức setter có tham số truyền vào tương ứng, tùy thuộc vào chương trình.

Biến this, Phương thức setter và getter

- **Phương thức getter**:
  Getter cũng là một phương thức dùng để truy cập vào các thuộc tính của đối tượng. Tuy nhiên, ngược lại với phương thức setter, phương thức getter được dùng để trả về giá trị cho các thuộc tính của đối tượng.

- **Tạo nhanh các phương thức setter, getter trong eclipse**:
  Source → Generate Getter and Setter rồi chọn biến cần tạo phương thức getter và setter.

Biến this, Phương thức setter và getter

setter

```
class HocSinh {
    private String hoTen;
    private String lop;
    private float diemTb;

    public void setHoTen(String hoTen) {
        this.hoTen = hoTen;
    }

    public void setLop(String lop) {
        this.lop = lop;
    }

    public void setDiemTb(float diemTb) {
        this.diemTb = diemTb;
    }
}
public class SuDungSetter {
    public static void main(String[] args) {
        HocSinh a = new HocSinh();
        a.setHoTen("Vu Van Nam");
        a.setLop("54K");
        a.setDiemTb(7.5f);
    }
}
```

Biến this, Phương thức setter và getter

getter

```
class HocSinh {

    private String hoTen;
    private String lop;
    private float diemTb;

    public void setHoTen(String hoT
en) {
        this.hoTen = hoTen;
    }

    public void setLop(String lop) {
        this.lop = lop;
    }

    public String getHoTen() {
        return hoTen;
    }

    public String getLop() {
        return lop;
    }

    public float getDiemTb() {
        return diemTb;
    }

    public void setDiemTb(float diemTb) {
        this.diemTb = diemTb;
    }
}

public class SuDungSetterGetter {

    public static void main(String[] args) {
        HocSinh a = new HocSinh();
        a.setHoTen("Vu Van Nam");
        a.setLop(“54K");
        a.setDiemTb(7.5f);

        System.out.println("Họ tên: " + a.getHoTen());
        System.out.println("Lớp: " + a.getLop());
        System.out.println("Điểm Tb: " + a.getDiemTb());
    }
}
```

Phương thức toString()

- Phương thức toString() trả về biểu diễn chuỗi của đối tượng.
- Nếu in bất cứ đối tượng nào, trình biên dịch Compiler trong Java sẽ gọi nội tại phương thức toString() trên đối tượng. Vì thế việc ghi đè phương thức toString(), và trả về kết quả mong muốn, nó có thể là trạng thái của một đối tượng, tùy thuộc vào trình triển khai của người lập trình.

Ví dụ: …

OOP in Java (con.)

- Tính kế thừa
  - ✓ Lớp cha và lớp con
  - ✓ Hàm khởi tạo và hàm hủy
- Tính đa hình
  - ✓ Mối quan hệ giữa các đối tượng trong một hệ thống phân cấp kế thừa
  - ✓ Gọi các phương thức lớp cha từ các đối tượng lớp con
  - ✓ Sử dụng lớp cha qua các biến kiểu lớp con
  - ✓ Phương thức lớp con gọi qua các biến kiểu lớp cha
  - ✓ Các lớp và phương thức trừu tượng
  - ✓ Giao diện
  - ✓ Nạp chồng và ghi đè

2.4. Tính Kế thừa

- ✓Đôi khi thiết kế chúng ta gặp mối quan hệ sau đây:
  - Class2 là một dạng đặc biệt của Class1 (đã có)
- ✓Trong tình huống này, chúng ta không muốn sao lại tất cả các chức năng và thuộc tính của Class1.
- ✓Thay vì sao lại, chúng ta tạo Class2 như một lớp phụ (lớp dẫn xuất – lớp con - SubClass) của Class1 (lớp cơ sở - lớp cha - SuperClass).
- ✓Khi đó Class2 sẽ kế thừa tất cả các thuộc tính và phương thức được cung cấp trong Class1 và cũng có thể định nghĩa thêm các chức năng bổ sung cho những đặc điểm riêng của nó.

Tính Kế thừa

- ✓ Để khai báo một lớp kế thừa từ một lớp khác ta sử dụng từ khóa extends:

```
class SubClass extends SuperClass {
…
}
```

- ✓ Một lớp chỉ có thể là một lớp dẫn xuất của một lớp khác. Nói cách khác là chỉ có thể kế thừa từ một lớp.
- ✓ Lớp SubClass sẽ kế thừa tất cả các thuộc tính và phương thức của lớp SuperClass.
  - Các thành viên dữ liệu private của SuperClass tồn tại trong SubClass nhưng chúng không được truy cập trực tiếp bởi bất kỳ phương thức nào của lớp SubClass.
  - Các thành viên dữ liệu static của SuperClass cũng được kế thừa, có nghĩa chúng cùng chia sẻ một bản sao của các thành viên static.

Phương thức khởi tạo của lớp dẫn xuất - Lớp con (SubClass)

- ✓ Các phương thức khởi tạo (constructor) của lớp dẫn xuất (SubClass) đầu tiên cần có lời gọi một phương thức khởi tạo của lớp cơ sở (SuperClass).
- ✓ Thực hiện dựa vào từ khóa super như sau:

```
class SubClass extends SuperClass {
        public ClassA() {
                Super(…);
        }
```

- ✓ *Lưu ý*: Nếu không gọi constructor của lớp cha, Java sẽ tự động gọi constructor mặc định.

Quan hệ IS-A và HAS-A

- "IS-A" và "HAS-A"
  - ✓ Quan hệ "IS-A"
    - Kế thừa
    - Đối tượng của lớp con cũng là đối tượng của lớp cha.
    - Ví dụ: Car *is a* Vehicle
      - Vehicle properties/behaviors also car properties/behaviors
  - ✓ Quan hệ "HAS-A": Ta nói A HAS-A B nếu A có tham chiếu đến B.

Lớp Final

- Có thể khai báo 1 lớp là **`final`** (còn gọi là lớp vô sinh) – Không có sự kế thừa từ lớp này.
- Một lớp **`abstract`** không thể là một lớp **`final`**.

Các thành phần bổ nghĩa protected

- Bổ nghĩa truy nhập `protected`
  - ✓Là mức nằm giữa `public` và `private`
  - ✓Các thành phần `protected` có thể truy nhập tới:
    - Các thành phần của lớp cha.
    - Các thành phần của lớp con.
    - Các lớp thành phần trong một gói. (ngoài gói phải kế thừa)
  - ✓Lớp con truy nhập các thành phần của lớp cha
    - Dùng từ khóa `super` và dấu chấm (.)
    - Không có dạng super.super….
  - ✓Không được áp dụng cho lớp và interface.

2.5. TÍNH ĐA HÌNH

- Sức mạnh của lập trình hướng đối tượng là thông qua tính đa hình.
- Tính đa hình trong Java là một khái niệm mà từ đó chúng ta có thể thực hiện một hành động đơn theo nhiều cách khác nhau.
- Có thể thực hiện tính đa hình trong Java bởi nạp chồng phương thức (overloading) và ghi đè (overriding) phương thức.
- Trong Java, ta có thể gán một biến kiểu lớp nhất định bởi một thể hiện của lớp đó hay một thể hiện của lớp dẫn xuất.

Ví dụ

```
class ConNguoi {
    void hoatDong() {
        System.out.println("Hoat dong cua con nguoi la gi?");
    }
}

class khiONha extends ConNguoi {
    @Override
    public void hoatDong() {
        System.out.println("- Khi o nha thi phai nau com.");
    }
}

class khiOTruong extends ConNguoi {
    public void hoatDong() {
        System.out.println("- Khi o truong thi phai hoc bai.");
    }
}

public class NguoiDaHinh {
    public static void main(String[] args) {
        ConNguoi nguoi = new ConNguoi();
        nguoi.hoatDong();
        nguoi = new khiONha();
        nguoi.hoatDong();
        nguoi = new khiOTruong();
        nguoi.hoatDong();
    }
}
```

Lớp trừu tượng (Abstract Class)

- Lớp trừu tượng
  - ✓Khai báo bởi từ khóa **`abstract`**
  - ✓Chúng là lớp cha.
  - ✓Chúng không thể được khởi tạo.
  - ✓Bao gồm các phương thức trừu tượng hoặc không.
  - ✓Khi một lớp trừu tượng được kế thừa thì lớp con thường triển khai tất cả các phương thức trừu tượng trong lớp cha. Hơn nữa, nếu không triển khai thì lớp con cũng được khai báo trừu tượng.

Phương thức trừu tượng
Abstract Method

- Một phương thức trừu tượng (*abstract method*) là một phương thức được khai báo không có triển khai của nó.
  - ✓Ví dụ: `abstract void moveTo(int x, int y);`
- Nếu một lớp chứa các phương thức trừu tượng thì lớp đó cũng phải được khai báo trừu tượng.
- Tất cả các phương thức trừu tượng của *interface* là hoàn toàn trừu tượng, vì thế từ bổ nghĩa *abstract* không cần sử dụng với các phương thức trong nó.

Ví dụ

```
abstract class ConCho {
    abstract void Goi();
}

class MauMat extends ConCho {
    @Override
    void Goi() {
        System.out.println("Mat mau Xanh");
    }
}

class MauLong extends ConCho {
    void Goi(){
        System.out.println("Long mau Den");
    }
}


public class ChoDaHinh {
    public static void main(String[] args) {
        ConCho [] cho = new ConCho[2];
        cho[0] = new MauMat();
        cho[1] = new MauLong();
        cho[0].Goi();
        cho[1].Goi();
    }

}
```

Giao diện (Interface)

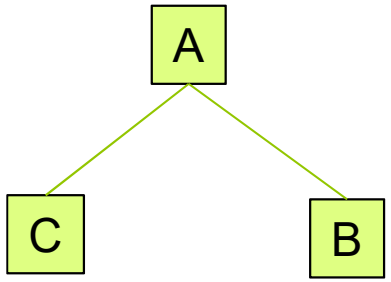
- Giao diện giống như một lớp trừu tượng "thuần chủng". Nó cho phép thiết lập hình dạng của một lớp: tên phương thức, danh sách các tham số và kiểu trả về, nhưng không có thân phương thức.
- Để tránh sự phức tạp của đa thừa kế, Java thay thế bằng các *interface*.
- Tất cả các phương thức của *interface* tự động là **public** và **abstract**.
- Có thể định nghĩa hằng trong *interface*.

Giao diện (Interface)

Một interface khác với một class ở một số điểm sau đây:

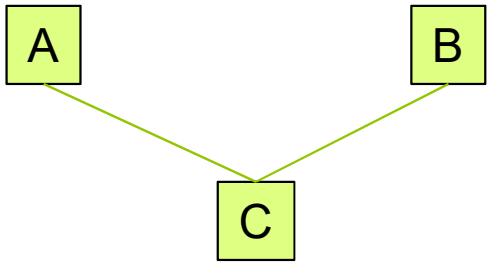
- Không thể khởi tạo một interface.
- Một interface không chứa bất cứ hàm contructor nào.
- Tất cả các phương thức của interface đều là abstract.
- Một interface không thể chứa một trường nào trừ các trường vừa static và final.
- Một interface không thể kế thừa từ lớp, nó được triển khai bởi một lớp.
- Một interface có thể kế thừa từ nhiều interface khác.

Khi nào thì sử dụng interface?



- B và C là các lớp. Giả sử A là lớp cha của B và C như vậy A có thể chứa các phương thức và các thuộc tính chung của B và C.
- Chúng ta có thể tạo A là lớp trừu tượng. Các phương thức trong A biểu thị những phương thức nào được triển khai trong B và C.
- Đôi khi tất cả các phương thức của B được triển khai khác với các phương thức tương tự của C. Khi đó ta có thể khởi tạo A là một **interface.**
- Một interface giống như một lớp.

Khi nào thì sử dụng interface?



- Giả sử C là một lớp con của A và B. Từ lâu Java không hỗ trợ đa kế thừa, vì vậy A và B không thể là các lớp trừu tượng. Do đó chúng ta phải sử dụng interface ở đây.

Khởi tạo và sử dụng Interface

- Interface
  - ✓ Khai báo bởi từ khóa `interface`
  - ✓ Sử dụng từ khóa **implements** như sau:

  public class ClassName implements interface1, interface2,…
- Java cung cấp hỗ trợ đa kế thừa thông qua sử dụng interface
  - ✓ Các lớp chỉ có thể extends một và chỉ một lớp
  - ✓ Tuy nhiên, các lớp có thể thừa kế nhiều interface
- Một lớp implements một interface phải cài đặt các phương thức của interface.
- Nếu một lớp implements nhiều interface có các phương thức giống nhau thì chỉ cần cài đặt một phương thức cho chúng.

Ghi đè phương thức (Overriding)

- Một phương thức trong lớp con được khai báo tương tự như trong lớp cha sẽ ghi đè lên phương thức của lớp cha.
- Cho phép một lớp con kế thừa từ lớp cha định nghĩa lại hoặc triển khai các phương thức của lớp cha.
- Phương thức ghi đè có tên, số lượng và kiểu của các tham số, kiểu trả về tương tự phương thức mà nó ghi đè.
- Phải là quan hệ IS-A (quan hệ kế thừa)
- Không thể ghi đè một phương thức final

Ghi đè phương thức (Overriding)

- Từ bổ nghĩa truy cập của phương thức ghi đè có thể được phép mở rộng hơn (không được thu hẹp nhỏ hơn) đối với phương thức bị ghi đè.

Ví dụ:

Phương thức protected trong lớp cha thì phương thức ghi đè có thể là public, nhưng không thể là private, trong lớp con.

Overriding Example

```
public class Shape {
    public static void testClassMethod() {
        System.out.println("The class method in Shape.");
    }

    public void testInstanceMethod() {
        System.out.println("The instance method in Shape.");
    }
}

public class Circle extends Shape {
    public static void testClassMethod() {
        System.out.println("The class method in Circle.");
    }

    public void testInstanceMethod() {
        System.out.println("The instance method in Circle.");
    }
}
```

Overriding Example

```
public class TestOverridingAndHiding {
    public static void main(String[] args) {
        Circle myCircle = new Circle();
        Shape myShape = myCircle;

        Shape.testClassMethod();
        myShape.testInstanceMethod();
    }
}
```

Output:

```
The class method in Shape.
The instance method in Circle.
```

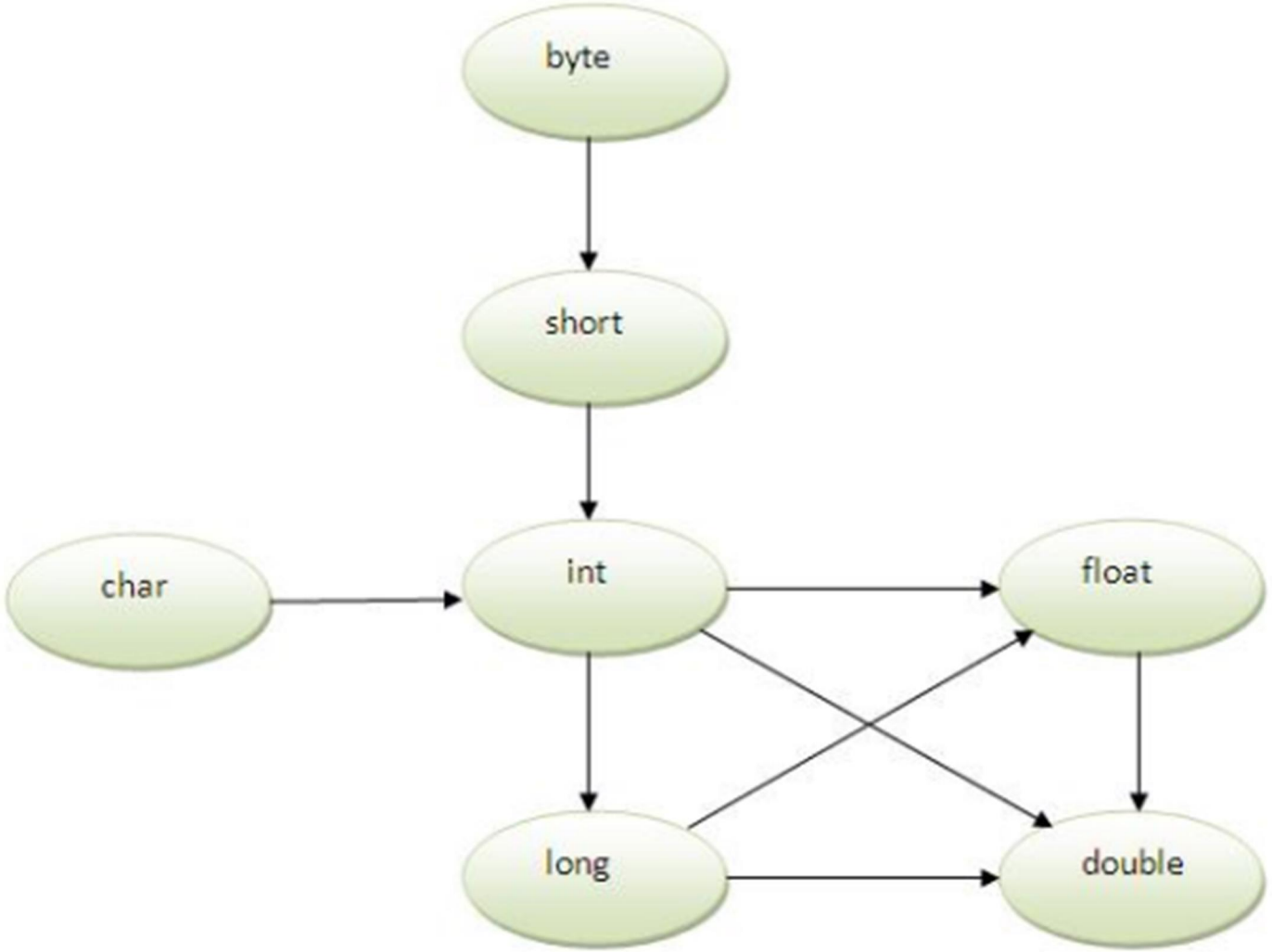
Nạp chồng phương thức (Overloading)

- Các phương thức được nạp chồng cho phép dùng lại tên phương thức trong một lớp nhưng khác nhau về tham số (số lượng, kiểu trả về).
- Trong lớp con, bạn có thể nạp chồng các phương thức kế thừa từ lớp cha.
- Những phương thức được nạp chồng:
  - ✓ Có mặt trong lớp cha cũng như lớp con.
  - ✓ Được định nghĩa lại trong lớp con.
- Những phương thức nạp chồng là một hình thức đa hình trong quá trình thực thi.

Nạp chồng phương thức (Overloading)

- Vài quy tắc chính trong việc nạp chồng một phương thức:
  - ✓ Các phương thức nạp chồng phải thay đổi danh sách tham số (số lượng, kiểu của các tham số).
  - ✓ Các phương thức nạp chồng không thể khác nhau kiểu trả về.
  - ✓ Các phương thức nạp chồng có thể thay đổi từ bổ nghĩa truy cập.
  - ✓ Một phương thức có thể nạp chồng trong cùng lớp hoặc trong một lớp con.

Nạp chồng phương thức và TypePromotion trong Java



Summary

- **Polymorphism**, which means "many forms," is the ability to treat an object of any subclass of a base class as if it were an object of the base class.
- **Abstract class** is a class that may contain abstract methods and implemented methods. An *abstract* method is one without a body that is declared with the reserved word *abstract*
- An **interface** is a collection of constants and method declarations. When a class *implements* an interface, it must declare and provide a method body for each method in the interface

Q&A