Lớp và các thành phần của Lớp các đối tượng

- ***Các nội dung chính:***
- Cấu trúc lớp và khai báo các thành phần của lớp,
- Định nghĩa hàm thành phần và cơ chế nạp chồng, viết đè trong Java,
- Các thuộc tính kiểm soát truy nhập các thành phần của lớp,
- Toán tử tạo lập các đối tượng,
- Kế thừa giữa các lớp đối tượng,
- Các giao diện (*interface*) và sự mở rộng quan hệ kế thừa (đa kế thừa) trong Java.

4.1 Định nghĩa lớp

[<Phạm vi hoặc kiểm soát truy nhập>] **class** <Tên lớp>

[**extends** <Tên lớp cha>] [**implements** <Tên giao diện>]

{

<Các thành phần của lớp>

}

Trong đó:

+ **class**, **extends**, **implements** là các từ khoá.

+ Những phần trong cặp [và] là tùy chọn.

a lớp

<Tên lớp>

ts <Tên giao diện>]

<Các thành phần của lớp>

}

Trong đó:

+ class, ex

+ Những

```
import java.io.*;
class Person{
    private String hoTen;
    private String diaChi;
    private int namSinh;

    Person(){}
    Person(String ht, String dc, int ns){
    /*Person(String hoTen, String diaChi, int namSinh){
    Person(String ht, String dc){
    public String nhapXau(){
    public int doiInt(String st){
    public void nhapTT(){
    public void hienThiTT(){
}
```

```
public class NhanVien extends Person{
    private int tongLuong;

    NhanVien(){}
    NhanVien(String ht, String dc, int ns, int tl){
    public void nhapTT(){
    public void hienThiTT(){
    public static void main(String arg[]){
        NhanVien nv=new NhanVien();
        nv.nhapTT();
        nv.hienThiTT();
    }
}
```

4.2 Định nghĩa hàm thành phần

[<P.Vi hoặc TT K.Soát truy nhập>]<Kiểu trả lại> <Tên hàm >(

[<Danh sách tham biến hình thức>]) [<Mệnh đề **throws**>]

{

<Nội dung của hàm>

}

Trong đó:

+ <Kiểu trả lại> có thể là kiểu nguyên thủy, kiểu lớp hoặc không có giá trị trả lại (kiểu *void*).

+ <Danh sách tham biến hình thức> bao gồm dãy các tham biến (kiểu và tên) phân cách với nhau bởi dấu phẩy

Ví dụ 4.1 // Tệp Demo.java

```
public class Demo{
  public static void main(String args[]){
            if (args.length == 0) return; //Hàm kiểu void có thể sử dụng return;
            output(checkValue(args.length));
   }
 static int sum(int m, int n){     // Hàm kiểu int; tính tổng giá trị của các tham số
            int kq=m+n;
            System.out.println("Sum ="+kq);
            return kq;
            }
  static int checkValue(int i){ // Hàm khác kiểu void phải sử dụng return để trả
            if (i > 3) return 1;            // lại giá trị.
            else return 2;
            }
}
```

4.3 Phạm vi và các thuộc tính kiểm soát truy nhập các thành phần của lớp

- **4.3.1 Phạm vi của các thành phần**
- *Phạm vi lớp của các thành phần*: Quyền truy nhập của chúng thường được xác định thông qua các *bổ ngữ (modifier): public, protected, private.*
- *Phạm vi khối của các biến cục bộ* (*local*): Tuân theo *Luật phạm vi:* một biến được khai báo ở trong một khối có phạm vi xác định bên trong khối và không xác định ở bên ngoài khối đó (Khối được bắt đầu bởi "{", kết thúc bởi "}")

4.3 Phạm vi và các thuộc tính kiểm soát truy nhập các thành phần của lớp

4.3.2 Các T.Tính K.Soát truy nhập các thành phần của lớp

- **public**: Công cộng, công khai
- **protected**: Được bảo vệ
- **mặc định** (không sử dụng thêm bổ ngữ khi định nghĩa lớp)
- **private**: Sở hữu riêng
- **static**: Chung cho cả lớp; không phải riêng của đối tượng nào cả
- **final**: Sau lần gán giá trị đầu tiên sẽ trở thành hằng số
- **abstract**: Thành phần trừu tượng

- **synchronized**: là khả năng kiểm soát truy cập của nhiều luồng đến bất kỳ nguồn tài nguyên chia sẻ (shared resource).
- **native**: Từ khoá native trong Java được sử dụng để chỉ định một hàm được triển khai bởi các ngôn ngữ khác như C, C++. Sử dụng các mã nguồn có sẵn mà không phải được viết bằng Java.
- **transient**: được sử dụng trong serialization. Nếu bạn định nghĩa bất kỳ thành viên dữ liệu nào là transient, nó sẽ không được đánh dấu là tuần tự (serialize).
- **volatile:** Biến volatile trong Java có tác dụng thông báo sự thay đổi giá trị của biến tới các thread khác nhau nếu biến này đang được sử dụng trong nhiều thread.

Một số lưu ý với từ khóa final

- Thuộc tính final
  - hằng số, chỉ được gán giá trị khởi tạo một lần, không thay đổi được giá trị
- Phương thức final
  - không cho phép định nghĩa lại ở lớp dẫn xuất
- Tham số final
  - không thay đổi được giá trị của tham chiếu
- Lớp final
  - không định nghĩa được lớp dẫn xuất

4.4 Tạo đối tượng bằng toán tử new

- **Một đối tượng (object)** trong Java là một thực thể của lớp. Để khai báo một đối tượng thuộc lớp, ta sử dụng cú pháp:
- <TênLớp> tên đối tượng;

- Ví dụ 1: SinhVien sv1;
- Ví dụ 2: Sach s1;

- Chúng ta sử dụng toán tử **new** và gọi đến **hàm khởi tạo** của lớp để khởi tạo đối tượng (Hàm **constructor** có tên trùng với tên của lớp).
- SinhVien sv=new SinhVien();
- Sach s=new Sach();

- Khi khởi tạo, đối tượng được cấp phát một vùng nhớ riêng để lưu trữ các dữ liệu của đối tượng đó.

- **Truy cập đến thuộc tính và phương thức của đối tượng**

- Sử dụng **toán tử chấm "."** để truy cập đến thuộc tính và phương thức của đối tượng. Nếu truy cập từ trong cùng một lớp thì có thể không cần dùng **toán tử chấm "."**.

```
BankAccount acount = new BankAccount() ;
account.setOwner("Alan Kay") ;
account.credit(5000000.00) ;
```

BankAcount method

```
public void credit(double amount) {
    setBalance(getBalance() + amount) ;
}
```

Đối tượng tự tham chiếu với từ khóa this

Với từ khóa this, cho phép truy cập vào đối tượng hiện tại của lớp, nhằm xóa đi sự nhập nhằng giữa biến cục bộ, tham số với thành phần dữ liệu lớp. Đặc biệt, trong trường hợp chúng cùng tên với nhau.

```
public class Account {
    String owner;
    long balance;

    void setAccountInfo(String owner, long balance) {
        this.owner = owner;
        this.balance = balance;
    }
}
```

Trong lớp **Account**, sử dụng tử khóa this để chỉ rõ các thuộc tính của lớp và gán giá trị cho chúng. Đó là tự tham chiếu với từ khóa this.

- Ví dụ về xây dựng hàm và lớp trong java.
- Xây dựng lớp toán học chứa 2 phương thức tính tổng, hiệu 2 số a, b;
- Xây dựng Vận dụng chứa hàm main() gọi tới 2 hàm của lớp trên.

Bài 1. Viết chương trình tính diện tích, chu vi hình chữ nhật.

- Hãy viết lớp HinhChuNhat gồm có:

+ Thuộc tính : chiều dài, chiều rộng.

+ Phương thức để nhập từ bàn phím, và hiển thị thông tin chiều dài, chiều rộng.

+ Phương thức tính chu vi.

+ Phương thức tính diện tích.

- Xây dựng lớp chứa hàm main chứa cac hàm trên để thực hiện nhập vào chiều dài, chiều rộng; Tính chu vi và diện tích của hình chữ nhật đó.

Xây dựng lớp **ToanHoc** có một biến thành phần là số nguyên n, với phương thức tĩnh(stattic) sau:

1. Kiểm tra xem một số nguyên k có phải là số nguyên tố hay không ?

public static boolean kiemtraNguyenTo(int k)

2. Kiểm tra xem một nguyên n có phải là số hoàn hảo hay không ?

public static boolean kiemtraHoanHao(int n)

Xây dựng lớp VanDung có phương thức main() sử dụng các phương thức trong lớp ToanHoc ở trên.

4.5 Toán tử tạo lập đối tượng

- Mục đích chính của toán tử tạo lập (*constructor*): Đặt các giá trị khởi tạo cho các đối tượng khi một đối tượng được tạo ra bằng toán tử *new*.
- Khai báo:

```
[<Thuộc tính>] <Tên lớp> ([<danh sách tham biến>]) {
        // Nội dung cần tạo lập
}
```

4.5 Toán tử tạo lập đối tượng

- Toán tử tạo lập giống như các hàm thành phần và chỉ khác ở phần đầu của định nghĩa:
- <Thuộc tính> chỉ có thể sử dụng: *public*, *protected*, *private* hoặc mặc định như đã trình bày ở phần trên.
- <Tên lớp> luôn trùng với tên của lớp chứa toán tử đó.

Một số lưu ý đối với các toán tử tạo lập:

- Không sử dụng các bổ ngữ khác với toán tử tạo lập,
- Toán tử tạo lập là loại hàm đặc biệt không có kiểu trả lại, ngay cả kiểu *void* cũng không được sử dụng,
- Toán tử tạo lập chỉ sử dụng được với toán tử *new*.

4.5.1 Toán tử tạo lập mặc định

- Toán tử tạo lập mặc định (*default*) là toán tử tạo lập không có tham biến.
- Cấu trúc: <Tên lớp> () {/* ... */ }

- Khi một lớp không định nghĩa một toán tử tạo lập nào cả thì hệ thống sẽ tự động cung cấp toán tử tạo lập mặc định không tường minh. Toán tử tạo lập mặc định không tường minh tương đương với dạng:
- <Tên lớp> () { } // Không làm gì cả
- *Lưu ý*: Khi đã định nghĩa toán tử tạo lập thì hệ thống sẽ không cung cấp toán tử tạo lập mặc định không tường minh nữa.

Ví dụ: Viết chương trình quản lý đối tượng Người gồm các thông tin: Họ tên, địa chỉ, năm sinh.

- Viết lớp Person như sau:

+ Thuộc tính: Họ và tên, địa chỉ, năm sinh.

+ Tạo ra 1 toán tử tạo lập mặc định.

+ Tạo toán tử tạo lập thứ hai nhận đầy đủ thông tin để khởi tạo giá trị cho các biến.

+ Viết hàm nhapTT() và hienTT().

- Xây dựng lớp class chứa hàm main: tạo 2 đối tượng p, p1. Trong đó:

+ p1 chứa thông tin của chính mình – sử dụng toán tử tạo lập đủ thông số.

+ p tạo bằng toán tử tạo lập mặc định, nhập các thông tin của sv p từ bàn phím, in kết quả ra màn hình.

Ví dụ 4.2 // Tệp NhanVien.java

```
class Person{
    private String hoTen;
    private String diaChi;
    private int namSinh;
    Person(){}
    Person(String hoTen, String diaChi, int namSinh){
        this.hoTen=hoTen;
        this.diaChi=diaChi;
        this.namSinh=namSinh;
    }
    public void nhapTT(){
        Scanner sc=new Scanner(System.in);
        System.out.print("Ho ten :"); hoTen=sc.nextLine();
        System.out.print("Dia chi :"); diaChi=sc.nextLine();
        System.out.print("Nam sinh :"); namSinh=sc.nextInt();
    }
    public void hienThiTT(){
        System.out.println("Ho ten =" + hoTen);
        System.out.println("Dia chi =" + diaChi);
        System.out.println("Nam sinh =" + namSinh);
    }
}
```

Ví dụ: Viết chương trình quản lý đối tượng Sinh Viên.

- Viết lớp SinhVien như sau:

+ Thuộc tính: Mã sinh viên, Họ tên, điểm LT, điểm TH, Điểm TB.

+ Tạo ra 1 toán tử tạo lập mặc định.

+ Tạo toán tử tạo lập thứ hai nhận đầy đủ thông tin để khởi tạo giá trị cho các biến.

+ Viết hàm nhapTT() và hienTT().

+ Viết hàm tính điểm trung bình của sinh viên đó.

- Xây dựng lớp class chứa hàm main: tạo 2 đối tượng sv1, sv2. Trong đó:

+ sv1 chứa thông tin của chính mình – sử dụng toán tử tạo lập đủ thông số.

+ sv2 tạo bằng toán tử tạo lập mặc định, nhập các thông tin của sv p từ bàn phím, in kết quả ra màn hình.

4.5.2 Nạp chồng toán tử tạo lập

```
class BongDen{
        // Biến thành phần

         int soWatts;
         boolean batTat;
         String viTri;
    // Định nghĩa toán tử tạo lập  mặc định
    BongDen(){ soWatts = 40;  batTat = true; viTri = new String("XX"); }
    // Định nghĩa toán tử tạo lập không mặc định
   BongDen(int w, boolean s, String v){ soWatts = w;          batTat = s;
         viTri = new String(v);
        }
}
```

4.6 Quan hệ kế thừa giữa các lớp

Java chỉ hỗ trợ kế thừa đơn (tuyến tính), nghĩa là một lớp chỉ kế thừa được từ một lớp cha.

Trong Java lớp cơ sở nhất, làm cơ sở (gốc của cấu trúc phân cấp) cho tất cả các lớp kế thừa là lớp *Object*.

Khai báo:

```
class <Tên lớp con> extends <Tên lớp cha> {
        // Các thuộc tính dữ liệu bổ sung
        // Các hàm thành phần bổ sung hay viết đè
}
```

Các mức kiểm soát truy nhập

Modifier	Same class	Same package	Subclass	Universe
private	Yes			
package (*default*)	Yes	Yes		
protected	Yes	Yes	Yes	
public	Yes	Yes	Yes	Yes

4.6.1 Toán tử móc xích giữa các lớp kế thừa this() và super()

Toán tử tạo lập this():

- *Được sử dụng để tạo ra đối tượng của lớp hiện thời.*
- Liên kết với đối tượng của lớp hiện thời

Toán tử tạo lập super():

- Được sử dụng trong các toán tử tạo lập của lớp con (*subclass*) để gọi tới các toán tử tạo lập của lớp cha (*superclass*) trực tiếp.
- Liên kết tới đối tượng của lớp cha trực tiếp

4.6.1 Toỏn tử this

Tạo ra đối tượng của lớp hiện thời

Liên kết với đối tượng của lớp hiện thời

```
Person(){}
/*Person(String ht, String dc, int ns){
Person(String ht, String dc){
    this(ht,dc,2016);
    System.out.println("Toan tu tao lap su dung voi this");
}
Person(String hoTen, String diaChi, int namSinh){
    this.hoTen=hoTen;
    this.diaChi=diaChi;
    this.namSinh=namSinh;
}
```

4.6.1. Toán tử super

```
class Person{
    private String hoTen;
    private String diaChi;
    private int namSinh;
    Person(){}
    /*Person(String ht, String dc, int ns){
    Person(String ht, String dc){
    Person(String hoTen, String diaChi, int namSinh){
    public String nhapXau(){
    public int doiInt(String st){
    public void nhapTT(){
    public void hienThiTT(){
}
public class NhanVien extends Person{
    private int tongLuong;
    NhanVien(){}
    NhanVien(String ht, String dc, int ns, int tl){
        super(ht, dc,ns);
        tongLuong=tl;
    }
    public void nhapTT(){
        super.nhapTT();
        System.out.print("Tong luong :"); tongLuong=doiInt(nhapXau());
    }
```

Gọi tới các toán tử tạo lập của lớp cha

Liên kết tới đối tượng của lớp cha trực tiếp

4.6.1 Toán tử móc xích giữa các lớp kế thừa this() và super()

Một số lưu ý khi sử dụng super() và this():

- Chúng chỉ sử dụng để xây dựng các toán tử tạo lập của các lớp và khi sử dụng thì chúng luôn phải là lệnh đầu tiên trong định nghĩa của toán tử tạo lập.
- *This()* được sử dụng để móc xích với cùng lớp chứa nó còn *super()* lại được sử dụng để móc xích với lớp cha của lớp đó.

4.6.2 Quan hệ kế thừa và quan hệ kết tập

- Vấn đề rất quan trọng trong phát triển phần mềm HĐT là xây dựng được biểu đồ lớp mô tả đầy đủ mối quan hệ giữa các lớp của hệ thống ứng dụng. Hai lớp có thể có các quan hệ sau:
  - Quan hệ liên kết (*Association*),
  - Quan hệ kết tập (*Aggregation*),
  - Quan hệ kế thừa (*Inheritance*),
  - Quan hệ phụ thuộc (*Dependency*).
- Hai quan hệ kế thừa và kết tập đều có đặc điểm chung là loại quan hệ giữa "một bộ phận" và phía "tổng thể". Quan hệ kế thừa là dạng "is a" (là một thành viên) còn quan hệ kết tập có dạng "has a" (có một thành viên).
- Vấn đề đặt ra cho người lập trình là chọn quan hệ nào thích hợp cho bài toán cần giải quyết

4.7 Giao diện và sự mở rộng quan hệ kế thừa trong Java

- **Định nghĩa interface:**

  interface định nghĩa mẫ… …ảo các hàm mẫu (*prototype*… hiện.

- **Khai báo**:

interface <Tên interface > {

  <Nội dung của…

}

- Trong đó: interfa…

```
public interface MyInterface{
    public void add(int x, int y);
    public void volume(int x, int y, int z);
}
interface Const{
    public static final double price = 100.0;
    public static final int counter = 5;
}
```

```
public interface MyInterface2{
    abstract public String nhapXau();
    abstract public int doiInt(String st);
    public double doiDouble(String st);
    public double diemTB(double x, double y, double z);
    public int namSinh(int tuoi);
}
```

4.7 Giao diện và sự mở rộng quan hệ kế thừa trong Java

- <Nội dung của interface>: thường chứa danh sách các hàm mẫu và các hằng.
- Giao diện
  hàm thành
  được giá t
- Những hà
  mới theo c

```
class
// Bổ
// Cà
}
```

```
import java.io.*;
public class DemoInterface implements MyInterface{
    public void add(int x, int y){
        System.out.println("x + y = " + (x + y));
    }
    public void volume(int x, int y, int z){
        System.out.println("x * y * z = " + (x * y*z));
    }
    public static void main(String args[]){
        DemoInterface d = new DemoInterface();
        d.add(3, 5);
        d.volume(2,4,5);
    }
}
```

4.7 Giao diện và sự mở rộng quan hệ kế thừa trong Java

- Giống như cấu trúc lớp, *interface* cũng có thể được mở rộng từ nhiều *interface* khác bằng mệnh đề *extends*. Tuy nhiên khác với sự mở rộng tuyến tính của các lớp, các *interface* được phép mở rộng không tuyến tính, nghĩa là có thể mở rộng nhiều hơn một *interface*.

```
class MyClass implements Interface1, Interface2 {
        //  Cài đặt một số hàm mẫu của Interface1, Interface2;
}
interface Interface1{
// Khai báo các hằng và hàm mẫu;
}
interface Interface2{
// Khai báo các hằng và hàm mẫu;
}
```

4.7 Giao diện và sự mở rộng quan hệ kế thừa trong Java

Một số lưu ý:

- Các hàm mẫu của *interface* được mặc định là *abstract* và không được khai báo *static*;
- Một lớp có thể chọn một số hàm mẫu để cài đặt, nghĩa là có thể chỉ cài đặt một phần của giao diện *interface*;
- Một giao diện có thể kế thừa từ nhiều hơn một giao diện;
- Một giao diện có thể kế thừa từ nhiều hơn một giao diện và lớp;

http://vietjack.com/index.jsp