**s1= new String(“A”);**

**s2= new String(“A”);**

**-> tạo 2 ô nhớ khác nhau**

**s3=”A”;**

**s4=”A”;**

**-> tạo 1 ô nhớ, trỏ s3,s4 vào ô nhớ đó**

**-> s1==s2 (false vì so sánh con trỏ)**

**s1.equals(s2) -> true (so sánh hàm)**

**1. Lịch sử Java**

<http://vietjack.com/java/lich_su_java.jsp>

**2. Tổng quan về Java**

<http://vietjack.com/java/tong_quan_ve_java.jsp>

**3. Giới thiệu về JDK, JRE, JRM**

<http://vietjack.com/java/gioi_thieu_jdk_jre_jvm_trong_java.jsp>

**4. Lớp Number trong Java**

Các phương thức và cách dùng

<http://vietjack.com/java/number_trong_java.jsp>  
**5. Lớp Char trong Java**

Các phương thức và cách dùng

<http://vietjack.com/java/character_trong_java.jsp>

**6. Nạp chồng và ghi đè**

**a. Nạp chồng**

- Các phương thức nằm trong cùng 1 lớp , có cùng tên với nhau nhưng có danh sách đối số khác nhau được gọi là các phương thức nạp chồng.  
- Constructor cũng có thể được nạp chồng.  
- Tùy theo ta gọi đối số thế nào mà nó sẽ gọi hàm tương ứng.  
- Nạp chồng là hình thức đa hình (polymorphism) trong quá trình biên dịch (compile time).

class Calculation{

void sum(int a,int b){System.out.println(a+b);}

void sum(int a,int b,int c){System.out.println(a+b+c);}

**Câu hỏi**: Chúng ta có thể nạp chồng phương thức main()?

Có. Bạn có thể có bất kỳ số lượng phương thức main nào trong môt lớp bởi nạp chồng phương thức.

**b. Ghi đè**

- Là Phương thức đã xuất hiện ở lớp cha và xuất hiện tiếp ở lớp con.

- Khi đối tượng thuộc lớp con gọi phương thức thì sẽ chọn lựa và chạy theo phương thức trong lớp con.

- Nếu lớp con không có phương thức đó thì mới lên kiếm ở lớp cha để chạy

- Ghi đè là hình thức đa hình (polymorphism) trong quá trình thực thi (Runtime)

class tenNguoiViet {  
    public String ten, ho;  
    public void show() {  
        System.out.println("Tên đầy đủ là: " + this.ho + " "+this.ten);  
    }  
}  
class tenNguoiNuocNgoai extends tenNguoiViet { //tenNguoiNuocNgoai kế thừa tenNguoiViet  
    @Override  
    public void show() {  
        System.out.println("Full name: " + this.ten + " " + this.ho);  
    }  
}

**Câu hỏi**: **Chúng ta có thể ghi đè phương thức static?**

Không, phương thức static không thể được ghi đè. Điều này có thể được chứng minh bởi tính đa hình tại runtime, chúng ta sẽ tìm hiểu sau.

**Câu hỏi: Tại sao chúng ta không thể ghi đè phương thức static?**

Bởi vì phương thức static được gắn kết với lớp trong khi đó phương thức instance được gắn kết với đối tượng. **Câu hỏi: Chúng ta có thể ghi đè phương thức main?**

Không, bởi vì main là phương thức static.

**7. Inner class**

**I**nner Class là một khái niệm chỉ về một lớp được khai báo bên trong lớp khác. Chúng ta sử dụng các Inner Class để nhóm các lớp và các Interface có quan hệ với nhau vào một vị trí để tăng tính đọc và tính duy trì.

Sử dụng Inner Class đem lại ba lợi thế chính là:

* Inner Class biểu diễn một kiểu quan hệ đặc biệt, đó là: nó có thể truy cập tất cả các thành viên (thành viên dữ liệu và các phương thức) của Outer Class, bao gồm cả các thành viên được khai báo là private.
* Inner Class được sử dụng để phát triển tính đọc và tính dễ dàng duy trì của code, bởi vì khi sử dụng Inner Class tức là bạn đã nhóm các lớp và các Interface có mối quan hệ với nhau vào một nơi.
* Giúp tối ưu hóa code, tức là bạn có thể viết ít code hơn nhưng có thể mang lại hiệu quả tương tự.

public class DemoJavaAndroidVn {

public static void main(String[] args) {

Outer out = new Outer();

out.show();

}

}

class Outer {

public void show() {

Inner in = new Inner();

in.display();

}

class Inner {

public void display() {

System.out.println("Đây là inner class.");

}

}

}

class B {

public void show() {

Outer.Inner in = new Outer().new Inner();

in.display();

}

}

**8. Từ khoá SUPER**

### super được để tham chiếu biến instance của lớp cha gần nhất

### super() được sử dụng để triệu hồi Constructor của lớp cha gần nhất

### super được sử dụng để triệu hồi phương thức của lớp cha gần nhất

class Vehicle{   
 int speed=50;   
}   
class Bike3 extends Vehicle{   
 int speed=100;   
 void display(){   
 System.out.println(speed); //se in speed cua Bike

// System.out.println(super.speed); //bay gio se in speed cua Vehicle

super();//se trieu hoi constructor cua lop Vehicle

}   
 public static void main(String args[]){   
 Bike3 b=new Bike3();   
 b.display();   
}   
}

**9. Biến Final trong JAVA**

**Biến Final:** hằng số không thể thay đổi (int cube(final int n) -> tham số n ko đổi)

**Phương thức Final:**

➢ Được sử dụng để ngăn chặn việc ghi đè (override) hoặc che lấp (hidden) trong các lớp Java.  
➢ Phương thức được khai báo là private hoặc là một thành phần của lớp final thì được xem là phương  
thức hằng.  
➢ Phương thức hằng không thể khai báo là trừu tượng (abstract).

**Lớp Final**

Là lớp không có lớp con (lớp vô sinh)  
➢ Hay: là lớp không có kế thừa  
➢ Được sử dụng để hạn chế việc thừa kế và ngăn chặn việc sửa đổi một lớp.

class Bike{

final void run(){System.out.println("running");}

}

class Honda extends Bike{

void run(){System.out.println("Chay an toan voi 100kmph");}

public static void main(String args[]){

Honda honda= new Honda();

honda.run();

}

Không thể kế thừa lớp Final

final class Bike{}

class Honda1 extends Bike{

void run(){System.out.println("Chay an toan voi 100kmph");}

public static void main(String args[]){

Honda1 honda= new Honda();

honda.run();

}

}

}

**Câu hỏi**: Phương thức final có được kế thừa không?

Có, phương thức final được kế thừa nhưng **bạn không thể ghi đè nó**

**Câu hỏi**: Chúng ta có thể khởi tạo biến final trống không?

Có, nhưng chỉ trong Constructor.

**Câu hỏi**: Chúng ta có thể khai báo một constructor final không?

Không, bởi vì constructor không bao giờ được kế thừa.

**10. Abstract class và Interface**

<http://vietjack.com/java/interface_trong_java.jsp>

<http://vietjack.com/java/lop_abstract_trong_java.jsp>

<http://vietjack.com/java/truu_tuong_trong_java.jsp>

<http://vietjack.com/java/lop_abstract_va_interface_trong_java.jsp>

**Xem các tính chất OOP, ABSTRACT CLASS/ INTERFACE, CONSTRUCTOR TRONG FILE bài 3**

**11. Toán từ INSTANCEOF**

 Sử dụng để kiểm tra xem đối tượng có là instance của kiểu cụ thể: lớp hoặc lớp con hoặc interface hay không

class Simple1{

public static void main(String args[]){

Simple1 s=new Simple1();

System.out.println(s instanceof Simple);//true

}

}

//-------------

class Animal{}

class Dog1 extends Animal

{//Dog ke thua Animal

public static void main(String args[]){

Dog1 d=new Dog1();

System.out.println(d instanceof Animal);//true

}

}

12. Lớp Object

<http://vietjack.com/java/lop_object_trong_java.jsp>

13. Nhân bản đối tượng

http://vietjack.com/java/mo\_phong\_doi\_tuong\_trong\_java.jsp

14. Mảng

Int[] a;----------------int[][] a;

Int a[];------------------int a[][]; ------------int []a[]---int [][]a;

a= new int[x];

Int[] a=new int[x]; -------------- int[][] a= new int[x][x]

Int[] a= {x,y,z,..};

Sao chép một mảng trong Java

char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e',

'i', 'n', 'a', 't', 'e', 'd' };

char[] copyTo = new char[7];

System.arraycopy(copyFrom, 2, copyTo, 0, 7);

## 15. Lớp Array trong Java

**public static int binarySearch(Object[] a, Object key)**

**public static boolean equals(long[] a, long[] a2):** Trả về true nếu hai mảng long đã cho là cân bằng nhau.

**public static void fill(int[] a, int val):** Gán giá trị int đã cho tới mỗi phần tử của mảng int đã cho

**public static void sort(Object[] a):** Xếp thứ tự mảng các đối tượng đã cho theo thứ tự tăng dần

16. Date and Time

<http://vietjack.com/java/date_time_trong_java.jsp>

# 17. Regular Expression trong Java