# Đa hình trong Java

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. Tính đa hình được suy ra từ hai từ Hy Lạp là Poly và Morphs. Poly nghĩa là nhiều và morphs nghĩa là hình, dạng. Có hai kiểu đa hình trong Java: Đa hình tại compile time và đa hình runtime. Chúng ta có thể thực hiện tính đa hình trong Java bởi nạp chồng phương thức và ghi đè phương thức.

Nếu bạn nạp chồng phương thức static trong Java, thì đó là ví dụ về đa hình tại compile time. Ở chương này chúng sẽ tập trung vào đa hình tại runtime trong Java.

Điều quan trọng để biết là có cách nào truy cập một đối tượng qua các biến tham chiếu. Một biến tham chiếu có thể chỉ là một kiểu. Khi được khai báo, kiểu của biến tham chiếu này không thể thay đổi.

Biến tham chiếu có thể được gán cho những đối tượng khác được cung cấp mà không được khai báo final. Kiểu của biến tham chiếu sẽ xác định phương thức mà có thể được triệu hồi trên đối tượng.

Một biến tham chiếu có thể được hướng đến bất kì đối tượng với kiểu khai báo hoặc bất kì kiểu con nào của kiểu khai báo. Một biến tham chiếu có thể được khai báo như là một class hoặc một interface.

## Đa hình tại runtime trong Java

Đa hình tại runtime là một tiến trình mà trong đó một lời gọi tới một phương thức được ghi đè được xử lý tại runtime thay vì tại compile time. Trong tiến trình này, một phương thức được ghi đè được gọi thông qua biến tham chiếu của một lớp cha. Việc quyết định phương thức được gọi là dựa trên đối tượng nào đang được tham chiếu bởi biến tham chiếu.

Trước khi tìm hiểu về đa hình tại runtime, chúng ta cùng tìm hiểu về Upcasting.

## Upcasting là gì?

Khi biến tham chiếu của lớp cha tham chiếu tới đối tượng của lớp con, thì đó là Upcasting. Ví dụ:

```
class A{}

class B extends A{}

A a=new B();//day la upcasting
```

# Ví dụ về đa hình tại runtime trong Java

Trong ví dụ, chúng ta tạo hai lớp Bike và Splendar. Lớp Splendar kế thừa lớp Bike và ghi đè phương thức run() của nó. Chúng ta gọi phương thức run bởi biến tham chiếu của lớp cha. Khi nó tham chiếu tới đối tượng của lớp con và phương thức lớp con ghi đè phương thức của lớp cha, phương thức lớp con được triệu hồi tại runtime.

Khi việc gọi phương thức được quyết định bởi JVM chứ không phải Compiler, vì thế đó là đa hình tại runtime.

```
class Bike{
  void run(){System.out.println("dang chay");}
}
class Splender extends Bike{
  void run(){System.out.println("chay an toan voi 60km");}

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

# Ví dụ thực về đa hình tại runtime trong Java

Giả sử Bank là một lớp cung cấp phương thức để lấy lãi suất. Nhưng lãi suất lại khác nhau giữa từng ngân hàng. Ví dụ, các ngân hàng VCB, AGR và CTG có thể cung cấp các lãi suất lần lượt là 8%, 7% và 9%. (Ví dụ này cũng có trong chương ghi đè phương thức nhưng không có Upcasting)

```
class Bank{
int getRateOfInterest(){return 0;}
}

class VCB extends Bank{
int getRateOfInterest(){return 8;}
}

class AGR extends Bank{
```

```
int getRateOfInterest(){return 7;}
}
class CTG extends Bank{
int getRateOfInterest(){return 9;}
}

class Test3{
public static void main(String args[]){
Bank b1=new VCB();
Bank b2=new AGR();
Bank b3=new CTG();
System.out.println("VCB lai suat la: "+b1.getRateOfInterest());
System.out.println("AGR lai suat la: "+b2.getRateOfInterest());
System.out.println("CTG lai suat la: "+b3.getRateOfInterest());
}
}
```

# Đa hình tại runtime trong Java với thành viên dữ liệu

Phương thức bị ghi đè không là thành viên dữ liệu, vì thế đa hình tại runtime không thể có được bởi thành viên dữ liệu. Trong ví dụ sau đây, cả hai lớp có một thành viên dữ liệu là speedlimit, chúng ta truy cập thành viên dữ liệu bởi biến tham chiếu của lớp cha mà tham chiếu tới đối tượng lớp con. Khi chúng ta truy cập thành viên dữ liệu mà không bị ghi đè, thì nó sẽ luôn luôn truy cập thành viên dữ liệu của lớp cha.

**Qui tắc**: Đa hình tại runtime không thể có được bởi thành viên dữ liệu.

```
class Bike{
 int speedlimit=90;
}
class Honda3 extends Bike{
 int speedlimit=150;

 public static void main(String args[]){
  Bike obj=new Honda3();
```

```
  System.out.println(obj.speedlimit);//90
}
```

# Đa hình tại runtime trong Java với kế thừa nhiều tầng (Multilevel)

Bạn theo dõi ví dụ sau:

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

class Dog extends Animal{
void eat(){System.out.println("an hoa qua");}
}

class BabyDog extends Dog{
void eat(){System.out.println("uong sua");}

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();
}
}
```

Và:

```
class Animal{
void eat(){System.out.println("animao dang an...");}
```

```
}


class Dog extends Animal{
void eat(){System.out.println("dog dang an...");}
}


class BabyDog1 extends Dog{
public static void main(String args[]){
Animal a=new BabyDog1();
a.eat();
}}
```

Vì, BabyDog không ghi đè phương thức eat(), do đó phương thức **eat()** của lớp Dog() được triệu hồi.