



Lập trình Java - Inheritance & Polymorphism

Ths. Vũ Duy Khương

- 1 **Khái niệm Kế thừa**
- 2 **Tạo đối tượng & các biến tham chiếu đối tượng**
- 3 **Constructors**
- 4 **Phương thức, biến Class và Instance**
- 5 **Tầm tác dụng của biến**
- 6 **Từ khóa this**
- 7 **Case Studies**

Khái niệm Kế Thừa - Inheritance

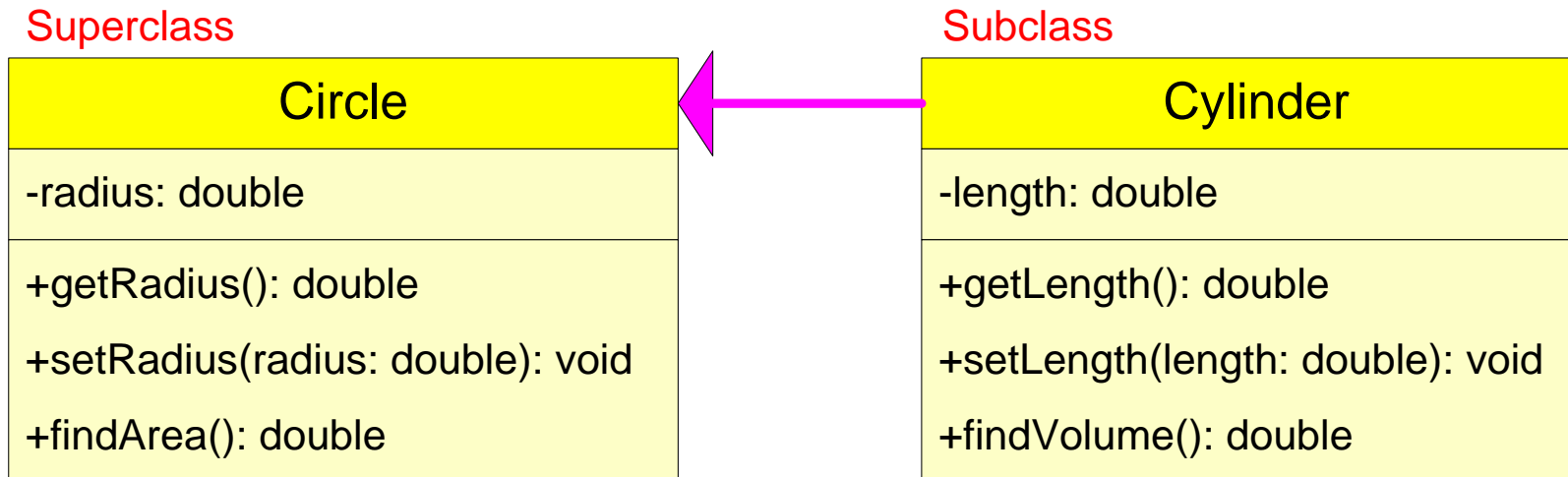
- **Kế thừa trong java** là sự liên quan giữa hai class với nhau, trong đó có class cha (superclass) và class con (subclass).
- Khi kế thừa class con được hưởng tất cả các phương thức và thuộc tính của class cha. Tuy nhiên, nó chỉ được truy cập các thành viên public và protected của class cha. Nó không được phép truy cập đến thành viên private của class cha.

Superclass và Subclass

- Lập trình hướng đối tượng cho phép bạn phát triển những lớp mới từ các lớp đã tồn tại.
- Ví dụ: lớp C1 được phát triển từ lớp C2:
 - C1: subclass, extended class, derived class
 - C2: superclass, parent class, base class
- Subclass thừa kế từ superclass các trường dữ liệu và phương thức có thể truy nhập được, và cũng có thể thêm vào các trường dữ liệu và phương thức mới.

Superclass và Subclass (tiếp)

- Thực tế, subclass thường được mở rộng để chứa nhiều thông tin chi tiết và nhiều chức năng hơn so với superclass của nó.



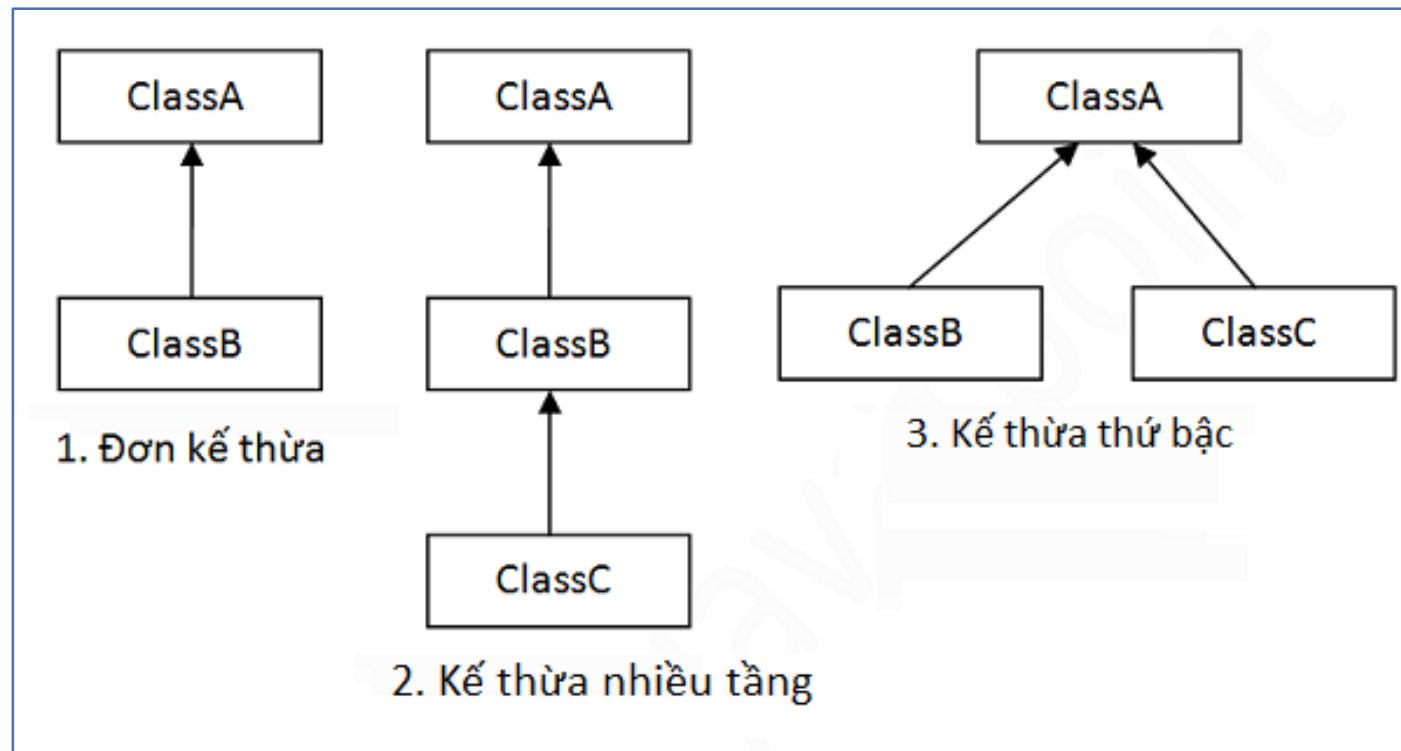
Cú pháp của kế thừa trong java

- Sử dụng từ khóa ***extends*** để kế thừa.

```
class Subclass-name extends Superclass-name {  
    //methods and fields  
}
```

Các kiểu kế thừa trong java

- Có 3 kiểu kế thừa trong java đó là đơn kế thừa, kế thừa nhiều cấp, kế thừa thứ bậc.



Ví dụ về đơn kế thừa

```
32 class Animal {
33     void eat() {
34         System.out.println("eating...");
35     }
36 }
37
38 class Dog extends Animal {
39     void bark() {
40         System.out.println("barking...");
41     }
42 }
43
44 public class TestInheritance1 {
45     public static void main(String args[]) {
46         Dog d = new Dog();
47         d.bark();
48         d.eat();
49     }
50 }
```


Ví dụ về kế thừa nhiều cấp

```
38 class Dog extends Animal {
39     void bark() {
40         System.out.println("barking...");
41     }
42 }
43
44 class BabyDog extends Dog {
45     void weep() {
46         System.out.println("weeping...");
47     }
48 }
49
50 public class TestInheritance2 {
51     public static void main(String args[]) {
52         BabyDog d = new BabyDog();
53         d.weep();
54         d.bark();
55         d.eat();
56     }
57 }
```

Ví dụ về kế thừa thứ bậc

```
44 class Dog extends Animal {
45     void bark() {
46         System.out.println("barking...");
47     }
48 }
49
50 class Cat extends Animal {
51     void meow() {
52         System.out.println("meowing...");
53     }
54 }
55
56 public class TestInheritance3 {
57     public static void main(String args[]) {
58         Cat c = new Cat();
59         c.meow();
60         c.eat();
61         // c.bark(); // compile error
62     }
63 }
```

Tại sao đa kế thừa không được support trong java?

- Để giảm thiểu sự phức tạp và đơn giản hóa ngôn ngữ, đa kế thừa không được support trong java
- Ví dụ: Có 3 lớp A, B, C. Trong đó lớp C kế thừa từ các lớp A và B. Nếu các lớp A và B có phương thức giống nhau và bạn gọi nó từ đối tượng của lớp con, như vậy khó có thể xác định được việc gọi phương thức của lớp A hay B.

Tại sao đa kế thừa không được support trong java?

```
32 class A {  
33     void msg() {  
34         System.out.println("Hello");  
35     }  
36 }  
37  
38 class B {  
39     void msg() {  
40         System.out.println("Welcome");  
41     }  
42 }  
43  
44 public class C extends A,B {  
45     public static void main(String args[]) {  
46         C obj = new C();  
47         obj.msg();  
48     }  
49 }
```

Từ khóa super trong java

- Từ khóa super trong java là một biến tham chiếu được sử dụng để tham chiếu trực tiếp đến đối tượng của lớp cha gần nhất.
- Có 3 cách sử dụng từ khóa *super*:
 - Từ khóa *super* được sử dụng để tham chiếu trực tiếp đến biến instance của lớp cha gần nhất.
 - *super()* được sử dụng để gọi trực tiếp Constructor của lớp cha.
 - Từ khóa *super* được sử dụng để gọi trực tiếp phương thức của lớp cha

Ví dụ sử dụng super trong java (1)

```
32 class Vehicle {
33     int speed = 50;
34 }
35
36 public class Bike extends Vehicle {
37     int speed = 100;
38
39     void display() {
40         System.out.println(speed);           //in speed của lớp Bike
41         System.out.println(super.speed);      //in speed của lớp Vehicle
42     }
43
44     public static void main(String args[]) {
45         Bike b = new Bike();
46         b.display();
47     }
48 }
```

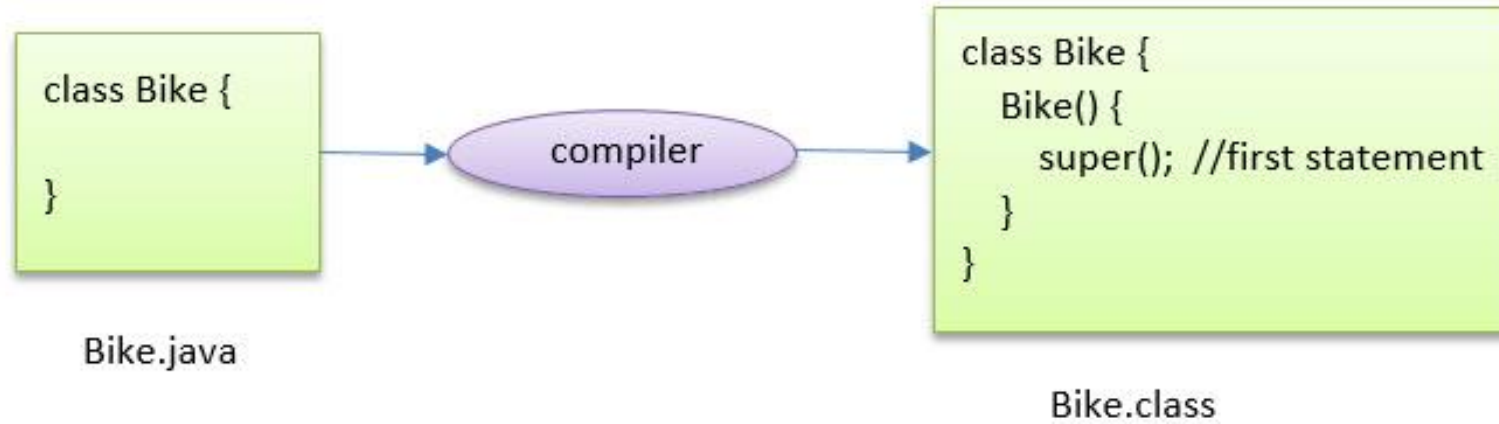
Ví dụ sử dụng super trong java (2)

➤ Trong java, `super()` được sử dụng để gọi trực tiếp Constructor của lớp cha.

```
32 class Vehicle {  
33     Vehicle() {  
34         System.out.println("Vehicle is created");  
35     }  
36 }  
37  
38 class Bike2 extends Vehicle {  
39     Bike2() {  
40         super(); //gọi Constructor của lớp cha  
41         System.out.println("Bike is created");  
42     }  
43  
44     public static void main(String args[]) {  
45         Bike2 b = new Bike2();  
46     }  
47 }
```

Chú ý: *super()* được tự động thêm vào mỗi Constructor của class bởi trình biên dịch.

Ví dụ sử dụng super trong java (3)



- Constructor được tạo ra tự động bởi trình biên dịch nhưng nó cũng thêm `super()` vào câu lệnh đầu tiên.
- Nếu bạn tạo Constructor và bạn không có `this()` hoặc `super()` ở dòng lệnh đầu tiên, trình biên dịch sẽ cung cấp `super()` của Constructor.

super được sử dụng để gọi trực tiếp phương thức của lớp cha

- Từ khóa `super` cũng có thể được sử dụng để gọi phương thức của lớp cha.

```
1 class Person {  
2     void message() {  
3         System.out.println("welcome");  
4     }  
5 }  
6  
7 public class Student extends Person {  
8     void message() {  
9         System.out.println("welcome to java");  
10    }  
11  
12    void display() {  
13        message();           // gọi phương thức message() của lớp hiện tại  
14        super.message();      // gọi phương thức message() của lớp cha  
15    }  
16  
17    public static void main(String args[]) {  
18        Student s = new Student();  
19        s.display();  
20    }  
21 }
```

Method Overloading

- **Nạp chồng phương thức** trong java xảy ra nếu một lớp có nhiều phương thức có tên giống nhau nhưng khác nhau về kiểu dữ liệu hoặc số lượng các tham số.
- Ví dụ:

```
class Adder{  
    static int add(int a, int b){    return a+b;  }  
    static int add(int a,int b,int c){  return a+b+c;  }  
    static double add(double a, double b){    return a+b;  }  
}  
class TestOverloading2{  
    public static void main(String[] args){  
        System.out.println(Adder.add(11,11));  
        System.out.println(Adder.add(12.3,12.6));  
    }  
}
```

Lợi ích của nạp chồng phương thức

- Sử dụng nạp chồng phương thức giúp tăng khả năng đọc hiểu chương trình.

Các cách nạp chồng phương thức

Có 2 cách nạp chồng phương thức trong java :

- Thay đổi số lượng các tham số
- Thay đổi kiểu dữ liệu của các tham số

Chú ý: Trong java, không thể nạp chồng phương thức bằng cách chỉ thay đổi kiểu trả về của phương thức

Các cách nạp chồng phương thức

(?) Tại sao không thể nạp chồng phương thức bằng cách chỉ thay đổi kiểu trả về của phương thức?

Ví dụ:

```
class Adder{
    static int add(int a,int b){ return a+b; }
    static double add(int a,int b){ return a+b; }
}
class TestOverloading3{
    public static void main(String[] args){
        System.out.println(Adder.add(11,11)); //ambiguity
    }
}
```

Có thể nạp chồng phương thức main() không?

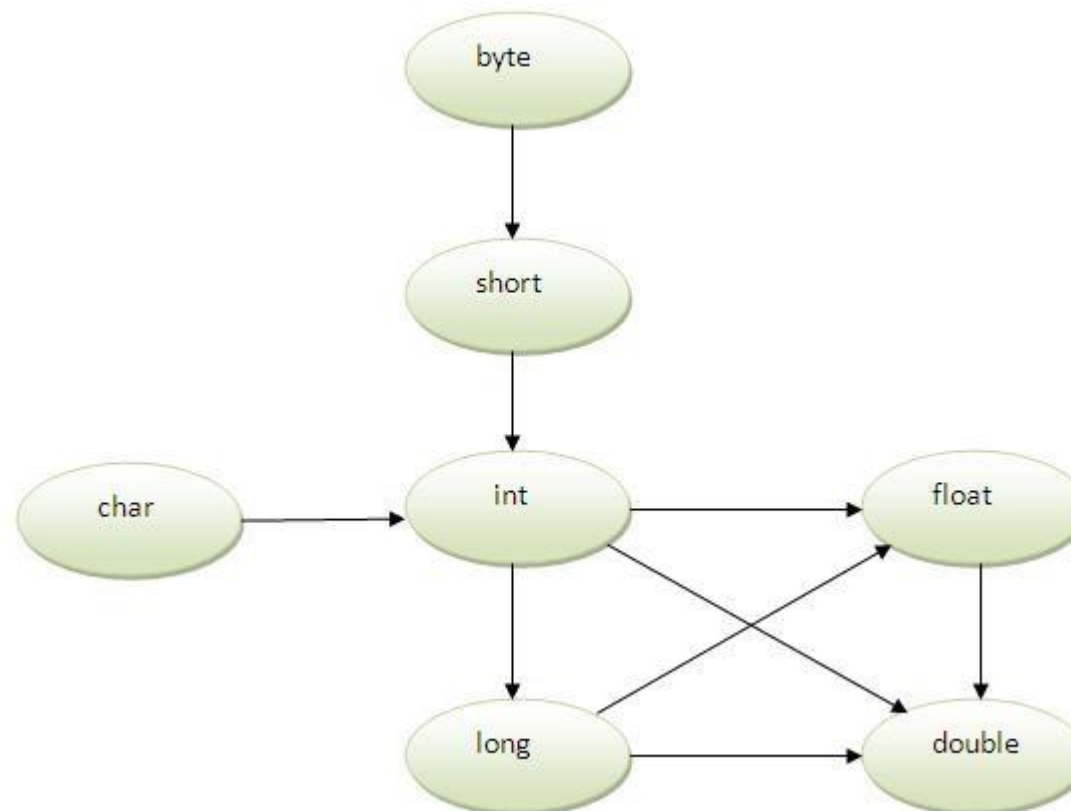
Ví dụ:

```
public class TestOverloading {  
    public static void main(String[] args) {  
        System.out.println("main with String[]");  
    }  
}
```

```
public static void main(String args) {  
    System.out.println("main with String");  
}
```

```
public static void main() {  
    System.out.println("main without args");  
}
```

Nạp chồng phương thức và sự thay đổi kiểu giá trị



Nạp chồng phương thức và sự thay đổi kiểu giá trị

```
public class OverloadingCalculation {  
    void sum(int a, long b) {  
        System.out.println("a method invoked");  
    }  
  
    void sum(long a, int b) {  
        System.out.println("b method invoked");  
    }  
  
    public static void main(String args[]) {  
        OverloadingCalculation obj = new OverloadingCalculation();  
        // không xác định được phương thức nào được gọi  
        obj.sum(20, 20);  
    }  
}
```

➔ Một kiểu không được tự động ép sang kiểu bé hơn, ví dụ kiểu double không được ép sang bất kỳ kiểu nào khác.

Ghi đè phương thức (method overriding)

Ghi đè phương thức trong java xảy ra nếu lớp con có phương thức giống lớp cha.

Nói cách khác, nếu lớp con cung cấp sự cài đặt cụ thể cho phương thức đã được cung cấp bởi một lớp cha của nó được gọi là ghi đè phương thức (method overriding) trong java.

Các nguyên tắc ghi đè phương thức

- Phương thức phải có tên giống với lớp cha.
- Phương thức phải có tham số giống với lớp cha.
- Lớp con và lớp cha có mối quan hệ kế thừa.

Ví dụ ghi đè phương thức

```
class Vehicle {  
    void run() {  
        System.out.println("Vehicle is running");  
    }  
}  
  
public class Bike extends Vehicle {  
    void run() {  
        System.out.println("Bike is running safely");  
    }  
  
    public static void main(String args[]) {  
        Bike obj = new Bike();  
        obj.run();  
    }  
}
```

Câu hỏi về ghi đè phương thức

- (?) Có ghi đè được phương thức static không?
- (?) Tại sao không ghi đè được phương thức static?
- (?) Có ghi đè phương thức main được không?

Sự khác nhau giữa overloading và overriding

Nạp chồng phương thức (overloading)	Ghi đè phương thức (overriding)
Nạp chồng phương thức được sử dụng để giúp code của chương trình <i>dễ đọc hơn</i> .	Ghi đè phương thức được sử dụng để cung cấp <i>cài đặt cụ thể</i> cho phương thức được khai báo ở lớp cha.
Nạp chồng được thực hiện bên <i>trong một class</i> .	Ghi đè phương thức xảy ra <i>trong 2 class</i> có quan hệ kế thừa.
Nạp chồng phương thức thì <i>tham số phải khác nhau</i> .	Ghi đè phương thức thì <i>tham số phải giống nhau</i> .
Nạp chồng phương thức là ví dụ về <i>đa hình lúc biên dịch</i> .	Ghi đè phương thức là ví dụ về <i>đa hình lúc runtime</i> .
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. <i>Giá trị trả về có thể giống hoặc khác</i> , nhưng tham số phải khác nhau.	Giá trị trả về phải giống nhau.

Tính đa hình - Polymorphism

- **Đa hình trong java (Polymorphism)** là một khái niệm mà chúng ta có thể thực hiện một hành động bằng nhiều cách khác nhau.
- Có hai kiểu của đa hình trong java:
 - Đa hình lúc biên dịch (compile)
 - Đa hình lúc thực thi (runtime).
- Có thể thực hiện đa hình trong java bằng cách nạp chồng phương thức và ghi đè phương thức.

Đa hình lúc runtime

➤ **Đa hình lúc runtime** là quá trình gọi phương thức đã được ghi đè trong thời gian thực thi chương trình.

➤ Ví dụ:


```
class Animal {  
    void run() {  
        System.out.println("running");  
    }  
}  
  
public class Lion extends Animal {  
    void run() {  
        System.out.println("running with 80km");  
    }  
  
    public static void main(String args[]) {  
        Animal b = new Lion();// upcasting  
        b.run();  
    }  
}
```

Ép kiểu đối tượng

```
m(new Student()) ;
```

thực hiện gán đối tượng new Student() cho một tham số kiểu Object, tương đương với 2 lệnh:


```
Object obj = new Student() ; // ép kiểu ngầm  
m(obj) ;
```



upcasting

➤ Muốn ấn định obj (kiểu Object) là một đối tượng Student:

```
Student std = (Student) obj ; //ép kiểu rõ ràng  
not Student std = obj ;
```



downcasting

Toán tử instanceof

Để ép kiểu đối tượng thành công, trước đó cần chắc chắn rằng đối tượng cần ép là 1 instance của đối tượng kia.

→ dùng toán tử **instanceof**

```
/** Giả sử myObj được khai báo kiểu Object */
```

```
/** Thực hiện ép kiểu nếu myObj là 1 instance của Cylinder */
```

```
if (myObj instanceof Cylinder) {  
    Cylinder myCyl = (Cylinder)myObj;  
    System.out.println("Thể tích hình  
        trụ là " + myCyl.findVolume());  
    ...  
}
```

Ví dụ về instanceof

```
class Animal { }
```

```
public class Dog extends Animal { // Dog inherits Animal
```

```
public static void main(String args[]) {
```

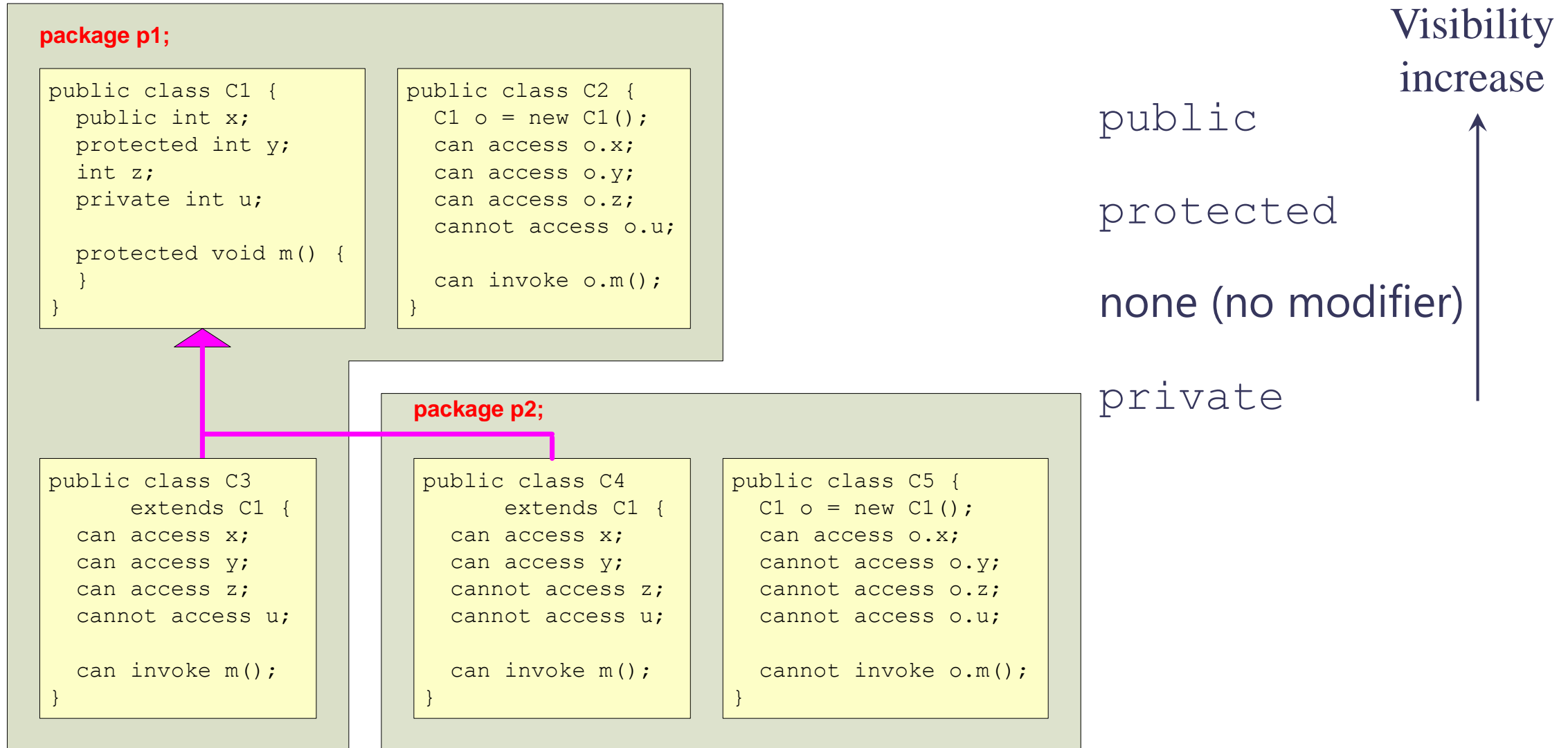
```
    Dog dog = new Dog();
```

```
    System.out.println(dog instanceof Animal); // true
```

```
}
```

```
}
```

Access Modifier trong Java

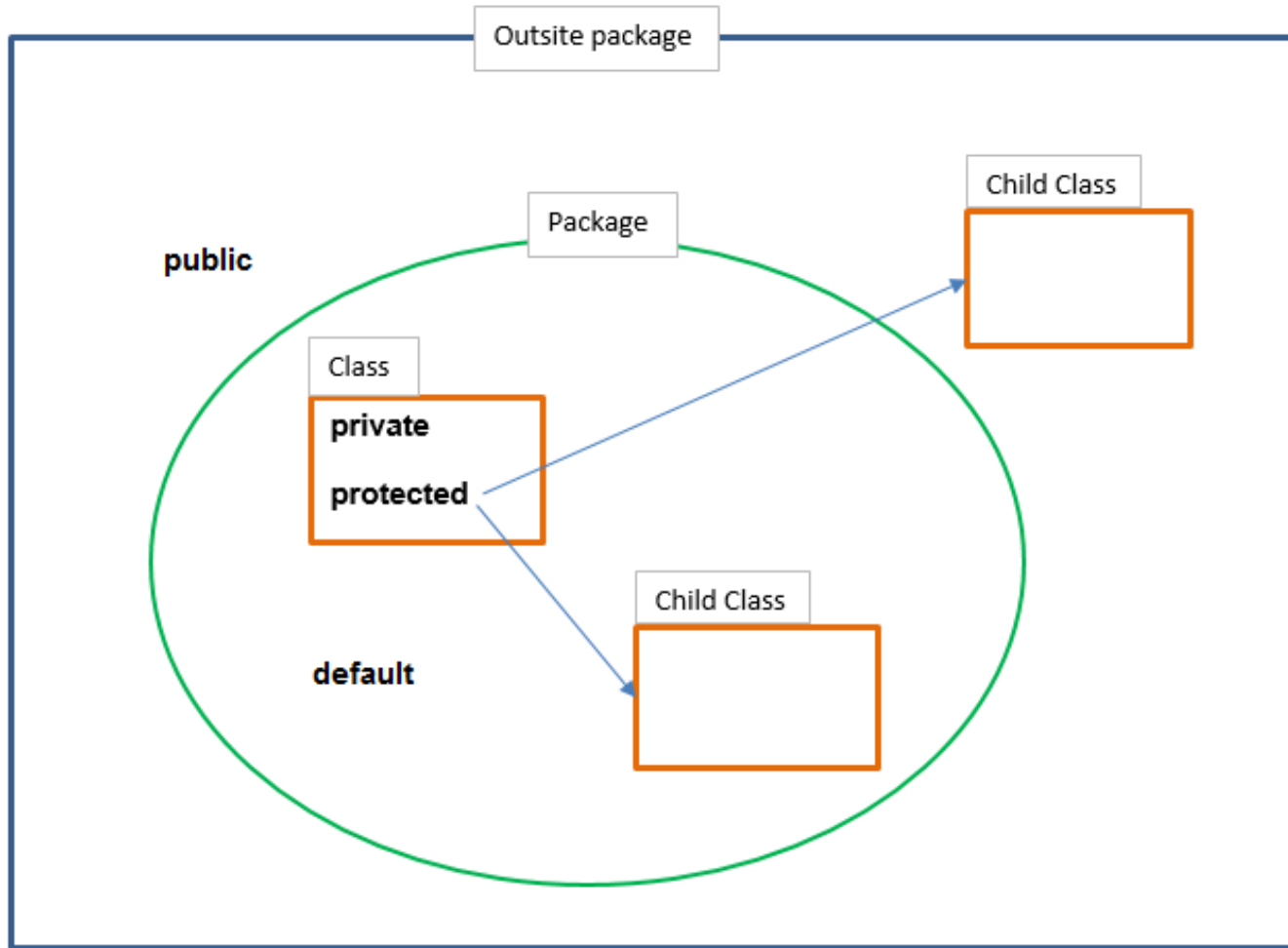


Access Modifier trong Java

- Có hai loại Access Modifier trong Java, đó là: **Access Modifier** và **Non-access Modifier**.
- Access Modifier trong Java xác định phạm vi có thể truy cập của biến, phương thức, constructor hoặc lớp.
- Trong java, có 4 phạm vi truy cập của **Access Modifier** như sau:
 - private**
 - default**
 - protected**
 - public**

Non-access Modifier như static, abstract, synchronized, native, volatile, transient,...

Access Modifier trong Java



Phạm vi truy cập private

```
class A {  
    private int data = 40;  
  
    private void msg() {  
        System.out.println("Hello java");  
    }  
}
```

```
public class Simple {  
    public static void main(String args[]) {  
        A obj = new A();  
        System.out.println(obj.data);  
        obj.msg();  
    }  
}
```

// Compile Time Error
// Compile Time Error

Phạm vi truy cập default

```
// Lưu file với tên A.java  
package demo;
```

```
class A {  
    void msg() {  
        System.out.println("Hello");  
    }  
}
```

```
// Lưu file với tên B.java  
package mypack;
```

```
import demo.*;
```

```
public class B {  
    public static void main(String args[]) {  
        A obj = new A(); // Compile Time Error  
        obj.msg();      // Compile Time Error  
    }  
}
```

Phạm vi truy cập protected

```
// Lưu file với tên A.java  
package demo;
```

```
public class A {  
    protected void msg() {  
        System.out.println("Hello");  
    }  
}
```

```
// Lưu file với tên B.java  
package mypack;
```

```
import demo.*;
```

```
public class B extends A {  
    public static void main(String args[]) {  
        B obj = new B();  
        obj.msg();  
    }  
}
```


Phạm vi truy cập public

```
// Lưu file với tên A.java  
package demo;
```

```
public class A {  
    public void msg() {  
        System.out.println("Hello");  
    }  
}
```

```
// Lưu file với tên B.java  
package mypack;
```

```
import demo.*;
```

```
public class B {  
    public static void main(String args[]) {  
        A obj = new A();  
        obj.msg();  
    }  
}
```

Lớp abstract trong java

- Một lớp được khai báo với từ khóa abstract là lớp abstract trong Java.
- Các abstract class có dữ liệu và phương thức tương tự như các class khác.
- Không thể dùng toán tử new để tạo các instance của abstract class.

Ví dụ header của 1 abstract class:

```
public abstract class GeometricObject
```

Tính trừu tượng trong java

- Tính trừu tượng là một tiến trình ẩn các cài đặt chi tiết và chỉ hiển thị tính năng tới người dùng.

Tính trừu tượng trong java (tiếp)

- Có 2 cách để đạt được sự trừu tượng hóa trong java
 - Sử dụng lớp *abstract*
 - Sử dụng *interface*

Phương thức trừu tượng

- Một phương thức được khai báo là abstract và không có trình triển khai thì đó là phương thức trừu tượng.
- *abstract void printStatus();*
// Khai báo phương thức với từ khóa abstract và không có thân phương thức

```
abstract class Bike{  
    abstract void run();  
}  
class Honda extends Bike{  
    void run() {  
        System.out.println("running safely..");  
    }  
}
```

```
public static void main(String args[]) {  
    Bike obj = new Honda();  
    obj.run();  
}
```

Interface

- Một Interface là một tập hợp các phương thức trừu tượng (abstract). Một class triển khai một interface, do đó kế thừa các phương thức abstract của interface.

Ví dụ về Interface

```
interface printable {  
    void print();  
}
```

```
class A6 implements printable {  
    public void print() {  
        System.out.println("Hello");  
    }  
}
```

```
public static void main(String args[]){  
    A6 obj = new A6();  
    obj.print();  
}
```

Interface vs. Abstract class

- Trong interface, dữ liệu phải là hằng, abstract class có thể có cả dữ liệu biến.
- Trong interface, phương thức chỉ có header mà ko có thực hiện, abstract class có thể có phương thức đầy đủ.
- Trong interface, tất cả dữ liệu là public final static và phương thức là public abstract

```
public interface T1{  
    public static final int k=1;  
    public abstract void p();  
}
```

==

```
public interface T1{  
    int k=1;  
    void p();  
}
```






THANK YOU