
Lambda Expression & Method Reference



Inner class

Lớp lồng nhau (inner class) trong java là một lớp được khai báo trong lớp (class) hoặc interface khác.

Chúng ta sử dụng inner class để nhóm các lớp và các interface một cách logic lại với nhau ở một nơi để giúp cho code dễ đọc và dễ bảo trì hơn.

Thêm vào đó, nó có thể truy cập tất cả các thành viên của lớp bên ngoài (outer class) bao gồm các thành viên dữ liệu private và phương thức.



Cú pháp

```
public class Main {  
    class InnerClass{  
  
    }  
}
```

```
public class OuterClass {  
    int x = 10;  
    class InnerClass{  
        int y = 5;  
    }  
}
```

Tạo đối tượng outerClass

```
public class Main {  
    public static void main(String[] args) {  
        OuterClass outerClass = new OuterClass();  
        OuterClass.InnerClass innerClass = outerClass.new InnerClass();  
        System.out.println(outerClass.x + innerClass.y);  
    }  
}
```

Tạo đối tượng innerClass

```
public class OuterClass {  
    static class InnerClass2{  
        int y = 20;  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        OuterClass.InnerClass2 InnerClass2 = new OuterClass.InnerClass2();  
        System.out.println(InnerClass2.y);  
    }  
}
```

Anonymous Inner Class



Một lớp không có tên được gọi là lớp vô danh hay anonymous inner class. Nó nên được sử dụng nếu bạn phải ghi đè phương thức của lớp hoặc interface.

Anonymous inner class có thể được tạo bằng hai cách:

01

Class

02

Interface



Khi nào nên sử dụng lớp vô danh

Lớp vô danh thường được sử dụng khi bạn không muốn phải khai báo cụ thể lớp con của một lớp nào đó, kể cả khi bạn không muốn khai báo cụ thể lớp triển khai của một interface nào đó, mà vẫn muốn sử dụng các đối tượng của chúng

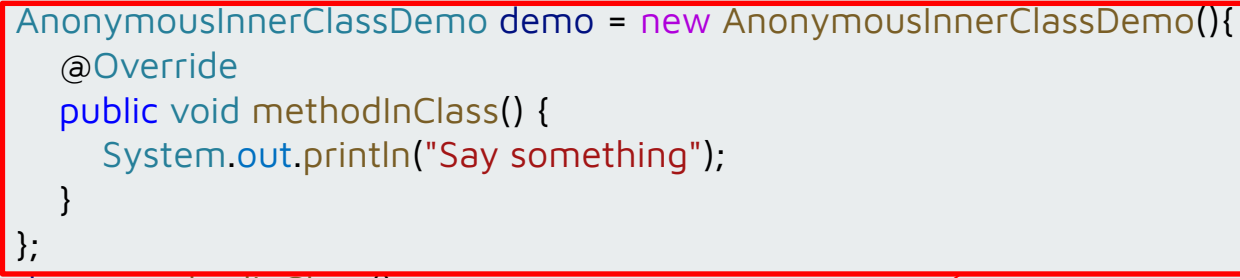


Đặc điểm của lớp vô danh

- ☐ Lớp vô danh chỉ có thể triển khai từ duy nhất một interface
- ☐ Lớp vô danh chỉ có thể kế thừa hoặc triển khai một lớp khác hoặc một interface khác
- ☐ Lớp vô danh không có constructor


```
public abstract class AnonymousInnerClassDemo {  
    public abstract void methodInClass();  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        AnonymousInnerClassDemo demo = new AnonymousInnerClassDemo(){  
            @Override  
            public void methodInClass() {  
                System.out.println("Say something");  
            }  
        };  
        demo.methodInClass();  
    }  
}
```



Anonymous inner class with class



Lambda Expression

Lambda Expression là một hàm không có tên với các tham số và nội dung thực thi. Nội dung thực thi của LE có thể là một khối lệnh hoặc 1 biểu thức

// Không có tham số, 1 câu lệnh

() -> expression

// 1 tham số, 1 câu lệnh

(parameters) -> expression

// các tham số và nội dung khối

(arg1, arg2, ...) -> {
 body-block
}

// các tham số, nội dung khối, dữ liệu trả về

(arg1, arg2, ...) -> {
 body-block;
 return return-value;
}

```
public class Main {  
  
    public static void main(String[] args) {  
        ArrayList<Integer> numbers = new ArrayList<Integer>();  
        numbers.add(5);  
        numbers.add(9);  
        numbers.add(8);  
        numbers.add(1);  
  
        for (Integer integers : numbers) {  
            System.out.println(integers);  
        }  
    }  
}
```



Sử dụng for each thông thường

Biểu thức lambda thường được truyền tham số dưới dạng tham số cho một hàm

```
numbers.forEach((n) -> {  
    System.out.println(n);  
});
```

Sử dụng Lambda
Expressions

//Sử dụng Comparator

```
list.sort(new Comparator<String>(){  
    public int compare(String o1, String o2) {  
        return o2.compareTo(o1);  
    };  
});
```

//Sử dụng lambda

```
list.sort((String o1, String o2) -> o1.compareTo(o2));
```

```
@FunctionalInterface
public interface AddNumber {
    public int add2Number(int a, int b);
}
```

```
public static void main(String[] args) {
    //Using anonymous inner class
    AddNumber addNumber = new AddNumber(){
        @Override
        public int add2Number(int a, int b) {
            return a+b;
        }
    };
}
```

```
public static void main(String[] args) {  
    //Using lambda  
    AddNumber addNumberUsingLambda = (a, b) -> {  
        return a+b;  
    };  
}
```


Method Reference

Method References (Phương thức tham chiếu) cung cấp các cú pháp hữu ích để truy cập trực tiếp tới constructor hoặc method đã tồn tại của các lớp hoặc đối tượng trong java mà không cần thực thi chúng





Method Reference

Method references là cú pháp viết tắt của biểu thức lambda để gọi phương thức. Ví dụ, nếu biểu thức lambda được viết như sau:

```
str -> System.out.println(str);
```

Có thể viết lại theo các của Method reference như sau:

```
System.out::println;
```



Các loại Method Reference

- Tham chiếu đến một static method – `Class::staticMethod`
- Tham chiếu đến một instance method của một đối tượng cụ thể
– `object::instanceMethod`
- Tham chiếu đến một instance method của một đối tượng tùy ý của một kiểu cụ thể
– `Class::instanceMethod`
- Tham chiếu đến một constructor – `Class::new`

```
ArrayList<Person> list = new ArrayList<>();  
list.add(new Person("Ngoc", 25));  
list.add(new Person("Hoang", 30));  
list.add(new Person("Tuan", 27));  
list.add(new Person("Hoa", 20));
```

```
//Sử dụng lambda  
list.forEach(n -> System.out.println(n));
```

```
//Sử dụng method reference  
list.forEach(System.out::println);
```

```
public static int compareByAge(Person p1, Person p2){  
    return p1.getAge() - p2.getAge();  
}
```

```
//Sử dụng lambda  
Collections.sort(list, (o1, o2) -> Person.compareByAge(o1, o2));
```

```
//Sử dụng method reference  
Collections.sort(list, Person::compareByAge);
```