**BẢNG ĐÁNH GIÁ ĐIỂM THUYẾT TRÌNH**

**MÔN LẬP TRÌNH JAVA NÂNG CAO**

**LỚP CTK44-PM**

|  |  |
| --- | --- |
| **Thông tin đề tài thuyết trình** | **Thông tin sinh viên thực hiện đánh giá** |
| Tên đề tài: Lambda Expression trong Java | MSSV của sinh viên đánh giá: 2012353 |
| Điểm tổng kết: Ngày trình bày: 16/05/2023 | Họ tên sinh viên & ký xác nhận: Hạ  Đoàn Cao Nhật Hạ |
| ***Thông tin sinh viên thuyết trình*** |
| MSSV: 2012395  Họ tên: Nguyễn Hữu Trọng Vỹ |

1. **ĐÁNH GIÁ THUYẾT TRÌNH**

**Thang điểm: Kém 0-5; Trung bình: 5-7; Khá 7-8; Tốt 8-9; Xuất sắc: 9-10**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Bảng chấm điểm nhóm thuyết trình** | | | | | | |
| Các chủ đề chính có được xác định? (A) | Slide thuyết trình có thiết kế tốt (B) | Giao tiếp với người nghe bằng ánh mắt (C) | Giao tiếp với người nghe bằng giọng nói (D) | Khả năng trình bày nội dung (E) | Thảo luận & Hỏi đáp (F) | Điểm trung bình = (A+B+C+D+E+F)/6 |
| **10** | **10** | **9** | **9** | **9,5** | **9,5** | **9,5** |

1. **BÀI TẬP NẮM KIẾN THỨC**

Dựa vào nội dung vừa trình bày, trả lời các câu hỏi sau:

1. Trình bày tóm tắt nội dung anh/ chị đã nắm được về chủ đề vừa trình bày?

* Lambda expression:

+ Lambda expression (Biểu thức Lambda) là một tính năng mới và quan trọng được thêm vào Java SE 8.

+ Cung cấp một cách ngắn gọn và rõ ràng để biểu diễn một phương thức của interface sử dụng một biểu thức.

+ Lambda expression thực thi phương thức của một Functional Interface. Giúp không cần định nghĩa phương thức lại mà chỉ cần viết code để thực thi phương thức.

* Functional interface

+ Functional interface là một interface mà chỉ có một phương thức abstract duy nhất.

+ Java cung cấp một anotation @FunctionalInterface, được sử dụng để chỉ định một interface là Functional interface.

* Cú pháp:

+ (argument-list) -> {

        body

}

+ argument-list: danh sách tham số

+ Dấu “->”: được sử dụng để nối danh sách tham số và phần thân của biểu thức

+ body: chứa biểu thức và những câu lệnh

* Lợi ích:

+ Thực thi Functional interface.

+ Viết ít code hơn.

* Standard  Functional interface

+ Functional interface, được tổng hợp trong gói java.util.function, cung cấp khá nhiều những target type cho biểu thức Lambda. Với mỗi interface trong đó là Generic và Abstract.

@FunctionalInterface

**public** **interface** **Foo** {

    String **method**(String string);

}

**public** String **add**(String string, Foo foo) {

**return** foo.method(string);

}

**Foo** foo = parameter -> parameter + " from lambda"; **String** result = useFoo.add("Message ", foo);

**public** String **add**(String string, Function<String, String> fn) {

**return** fn.apply(string);

}

Function<String, String> fn = parameter -> parameter + " from lambda";

**String** result = useFoo.add("Message ", fn);

* Functional interface anotation

+ Khi sử dụng anotation @FunctionalInterface, trình biên dịch sẽ thông báo lỗi nếu như có sự thay đổi cấu trúc của một Functional interface.

Giúp cho các nhà phát triển khác dễ hiểu kiến trúc ứng dụng.

**public** **interface** **Foo** {

    String **method**();

}

@FunctionalInterface

**public** **interface** **Foo** {

    String **method**();

}

* Không lạm dụng default method trong functional interface

+ Có thể thêm phương thức với từ khóa “default” vào Functional interface. Điều này là chấp nhận được, miễn là Functional interface vẫn chỉ chứa duy nhất 1 abstract method.

@FunctionalInterface

**public** **interface** **Foo** {

    String **method**(String string);

**default** **void** **defaultMethod**() {}

}

@FunctionalInterface

**public** **interface** **FooExtended extends** **Baz**, Bar {}

@FunctionalInterface

**public** **interface** **Baz** {

    String **method**(String string);

**default** String **defaultBaz**() {} }

@FunctionalInterface

**public** **interface** **Bar** {

    String **method**(String string);

**default** String **defaultBar**() {} }

* Lambda expression không phải là inner class

+ Một điều khác nhau cần lưu ý nhất là: PHẠM VI.

+ Khi sử dụng một inner class, nó sẽ tạo ra một phạm vi mới. Có thể ẩn các biến cục bộ khỏi phạm vi kèm theo bằng cách khởi tạo các biến cục bộ mới có cùng tên. Cũng có thể sử dụng từ khóa this bên trong inner class để tham chiếu đến instance của nó.

+ Tuy nhiên, biểu thức lambda hoạt động với phạm vi kèm theo. Không thể ẩn các biến khỏi phạm vi kèm theo bên trong phần thân của lambda. Trong trường hợp này, từ khóa this là tham chiếu đến một instance kèm theo.

**private** **String** value = "Enclosing scope value";

**public** String **scopeExperiment**() {

**Foo** fooIC = **new** **Foo**() {

**String** value = "Inner class value";

        @Override **public** String **method**(String string) {

**return** this.value;

        }

};

**String** resultIC = fooIC.method("");

**Foo** fooLambda = parameter -> {

**String** value = "Lambda value";

**return** this.value;

};

**String** resultLambda = fooLambda.method("");

**return** "Results: resultIC = " + resultIC + ", resultLambda = " + resultLambda; }

* Giữ lambda expression ngắn gọn và dễ hiểu

+ Nếu có thể, nên viết biểu thức lambda trên 1 dòng thay vì dùng một khối code lớn.

+ Chủ yếu là về phong cách code, không ảnh hưởng đến hiệu suất.

* Tránh viết các khối code trong th n lambda expression

+ Nếu có khối lượng lớn code, có thể tách phần xử lý ra một hàm riêng, sau đó gọi hàm trong phần thân của Lambda expression.

**Foo** foo = parameter -> {

**String** result = "Something " + parameter;

    //many lines of code

**return** result;

};

**Foo** foo = parameter -> buildString(parameter);

**private** String **buildString**(String parameter) {

**String** result = "Something " + parameter;

    //many lines of code

**return** result;

}

* Tránh chỉ định kiểu cụ thể trong lambda expression

+  Compiler (trình biên dịch) trong hầu hết các trường hợp đều có khả năng tự suy ra kiểu. Vì vậy có thể bỏ qua việc xác định kiểu

(String a, String b) -> a.toLowerCase() + b.toLowerCase();

(a, b) -> a.toLowerCase() + b.toLowerCase();

* Tránh sử dụng dấu ()

+ Tránh sử dụng cặp dấu () nếu chỉ có 1 tham số duy nhất.

Điều này giúp code ngắn gọn hơn đôi chút.

1. -> a.toLowerCase();

a -> a.toLowerCase();

* tránh sử dụng mệnh đề return và dấu {}

+ Tránh sử dụng từ khóa return nếu chỉ có 1 tham số duy nhất.

+ Tránh sử dụng cặp dấu ngoặc nhọn nếu chỉ có 1 câu lệnh hoặc 1 biểu thức.

+ Điều này giúp code ngắn gọn hơn đôi chút.

a -> {

    return a.toLowerCase()

};

a -> a.toLowerCase();

* tránh lỗi “lạ” xảy ra

+ Mô hình “hiệu quả cuối cùng” giúp ích rất nhiều ở đây, nhưng không phải trong mọi trường hợp. Lambdas không thể thay đổi giá trị của một đối tượng khỏi phạm vi kèm theo

+ Đoạn mã dưới đây là hợp lệ, vì biến total vẫn là "effectively final", nhưng liệu đối tượng mà nó tham chiếu có cùng trạng thái sau khi thực thi lambda không? KHÔNG!

**int**[] total = **new** **int**[1];

**Runnable** r = () -> total[0]++;

r.run();

- Tổng kết

+ Bản chất của Lambda Expression - Biểu thức Lambda trong java là tạo ra một đối tượng nặc danh cho một Functional Interface, và cung cấp thân hàm cho phương thức trừu tượng duy nhất của Functional Interface.

+ Functional Interface trong Java là một interface chỉ được chứa một hàm trừu tượng duy nhất

1. Anh/ chị hãy đưa ra các nhận xét và góp ý cho sinh viên vừa trình bày?