

Chuyên đề mở rộng

Functional Programming in Java

&

Stream API

Mục tiêu

- ✓ Viết được biểu thức lambda, qua đó rút gọn và đơn giản hóa mã nguồn
- ✓ Vận dụng biểu thức lambda để viết được cú pháp gọi phương thức theo method reference để trỏ tới tên của chúng tương ứng ba loại (static, instance và constructor)
- ✓ Khai báo được default method giúp interface trong chương trình có thể tương thích ngược với các phiên bản mã nguồn cũ
- ✓ Sử dụng Stream API mới trong Java 8 để xử lý lấy dữ liệu từ tập hợp.

Bài thực hành số 1:

Yêu cầu: tạo lớp DemoLambda, trong đó:

- Tạo một interface Calculate và viết biểu thức lambda gọi tới nó.
- Vận dụng lambda để viết mã nguồn gọi interface Comparator thay vì viết anonymous class.

Code tham khảo:

```
package chuyende.lambda;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;

/**
 *
 * @author minhvuvc
 */
public class DemoLambda {
```

```
List<Employee> lstBkap = new ArrayList<>();

class Employee {

    String name;
    int age;
}

private void sortEmp() {
    // 1. Old case
    Comparator<Employee> oldCase = new Comparator<Employee>() {
        @Override
        public int compare(Employee o1, Employee o2) {
            return o1.name.compareTo(o2.name);
        }
    };

    // 2. Lambda
    // Viết tường minh có đầy đủ: kiểu dữ liệu của đối số + từ khóa return.
    Comparator<Employee> lambdaCase = (Employee emp1, Employee emp2) ->
    {
        return emp1.name.compareTo(emp2.name);
    };

    // Viết Lambda khai báo tường minh có return
    Comparator<Employee> lambdaCase2 = (emp1, emp2) -> {
        return emp1.age > emp2.age ? 1 : -1;
    };

    // Viết Labda lược bớt kiểu dữ liệu và return
    Comparator<Employee> lambdaCase3 = (o1, o2) ->
    o1.name.compareTo(o2.name);

    lstBkap.sort(lambdaCase3);
}

interface Calculate {

    int operator(int a, int b);
}

void sieuMayTinh() {
    Calculate tong = (a, b) -> a + b;
```

```

        Calculate tru = (a, b) -> a - b;
        System.out.println("Tổng: " + tong.operator(1, 2));
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        DemoLambda main = new DemoLambda();

    }
}

```

Bài thực hành số 2:

Yêu cầu: tạo lớp DemoFunctionInterface, trong đó:

1. Tạo phương thức có sử dụng Function Interface mới trong Java 8
2. Viết mã nguồn gọi method reference

Code tham khảo:

```

package chuyende.functioninterface;

import java.util.Arrays;
import java.util.List;
import java.util.function.BinaryOperator;
import java.util.function.Consumer;
import java.util.function.DoubleConsumer;
import java.util.function.Function;
import java.util.function.IntConsumer;
import java.util.function.Predicate;

/**
 *
 * @author minhvufc
 */
public class DemoFunctionalInterface {

    void evaluation(float number, Predicate<Float> pre) {
        // Sử dụng Predicate kiểm tra điều kiện quy ước có phù hợp không
        // "điều kiện quy ước" có thể tùy biến ở biểu thức lambda
        if (pre.test(number)) {
            System.out.println("#" + number);
        }
    }
}

```

```
} else {
    System.out.println("Wrong number");
}

static void printValue(String test) {
    System.out.println(test);
}

/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
    DemoFunctionalInterface main = new DemoFunctionalInterface();

    // Demo 1: Predicate
    Predicate<Float> p = (preNum) -> {
        return preNum > 2.7f; // "điều kiện quy ước" là số phải lớn hơn 2,7.
    };
    main.evaluation(3.14f, p);
    main.evaluation(0.69f, p);
    System.out.println("-----");

    // Demo 2: BinaryOperator
    BinaryOperator<Integer> boTong = (b1, b2) -> b1 + b2;
    BinaryOperator<Integer> boTru = (b1, b2) -> b1 - b2;
    System.out.println("Tổng: " + boTong.apply(5, 3));
    System.out.println("Trừ: " + boTru.apply(5, 3));
    System.out.println("-----");

    // Demo 3: IntConsumer
    IntConsumer intC = (value) -> {
        System.out.println("#" + value);
    };
    IntConsumer intC2 = (value) -> System.out.println("2# " + value);
    IntConsumer intC3 = (a) -> System.out.println("Square = " + a * a);
    intC.accept(5);
    intC2.accept(3);
    intC3.andThen(intC2).accept(6);

    // Method Reference và toán tử ::
    System.out.println("Method reference - double colon");
    DoubleConsumer douC = System.out::println; // Loại Instance method
    douC.accept(3.14);
}
```

```
// Sử dụng toán tử :: (double colon) viết tắt code
Consumer<String> con = DemoFunctionalInterface::printValue; // Loại Static
method

con.accept("Kiss more & more");

// Giải thích đoạn code bằng biểu thức lambda
// Consumer<String> con = (str) -> {
//     main.printValue(str);
// };
// Viết lại đoạn code trên tường minh bằng cách tạo anonymous class
// Consumer<String> con = new Consumer<String>() {
//     @Override
//     public void accept(String t) {
//         main.printValue(t);
//     }
// };
System.out.println("-----");

// Sử dụng trong List
List<String> name = Arrays.asList("Nam", "Tùng", "Bách", "Thu", "Nga",
"Hoàng");
name.forEach(con);
// Cách 1: viết bằng lambda
// name.forEach((t) -> {
//     System.out.println("- " + t);
// });
// Cách 2: Viết kiểu Method reference
name.forEach(System.out::println);

// Khởi tạo Function Interface FUNCTION bằng method reference
// gọi constructor (điều kiện là constructor có tham số)
Function<String, Integer> f0 = (t) -> {
    return new Integer(t); // Lambda
};
// method reference
Function<String, Integer> f01 = Integer::new;

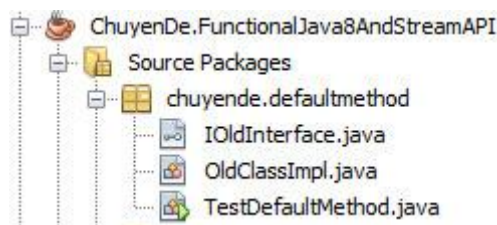
// Khởi tạo Function Interface FUNCTION bằng method reference
Function<Integer, String> f = String::valueOf;
// => Tường minh bằng biểu thức lambda
Function<Integer, String> f1 = (t) -> {
    return String.valueOf(t); //To change body of generated lambdas, choose
Tools | Templates.
};
```

```
}  
  
}
```

```
-----  
#3.14  
Wrong number  
-----  
Tổng: 8  
Trừ: 2  
-----  
#5  
2# 3  
Square = 36  
2# 6  
Method reference - double colon  
3.14  
Kiss more & more  
-----  
Nam  
Tùng  
Bách  
Thu  
Nga  
Hoàng  
Nam  
Tùng  
Bách  
Thu  
Nga  
Hoàng  
BUILD SUCCESSFUL (total time: 0 seconds)
```

Bài thực hành số 3:

Yêu cầu: viết interface có default method, tạo class thực thi interface này.



Code tham khảo:

IOldInterface.java

```
package chuyende.defaultmethod;
```

```
/**
```

```

*
* @author minhvuvc
*/
public interface IOldInterface {

    public void oldMethod();

    /**
     * @since 2.0
     */
    default void newMethod1() {
        System.out.println("This is new 1");
    }

    /**
     * Phương thức mới được thêm ở version 2
     *
     * @since 2.0
     */
    default void newMethod2() {
        System.out.println("This is new 2");
    }
}

```

OldClassImpl.java

```

package chuyende.defaultmethod;

/**
 *
 * @author minhvuvc
 */
public class OldClassImpl implements IOldInterface {

    @Override
    public void oldMethod() {
        System.out.println("This is old method");
    }

    @Override
    public void newMethod1() {
        IOldInterface.super.newMethod1(); // Call to parent
        System.out.println("and...this is inside child");
    }
}

```

```
}
```

TestDefaultMethod.java

```
package chuyende.defaultmethod;
```

```
/**
```

```
 *
```

```
 * @author minhvuvc
```

```
 */
```

```
public class TestDefaultMethod {
```

```
    /**
```

```
     * @param args the command line arguments
```

```
    */
```

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

```
        // Gọi class OldClass
```

```
        OldClassImpl oci = new OldClassImpl();
```

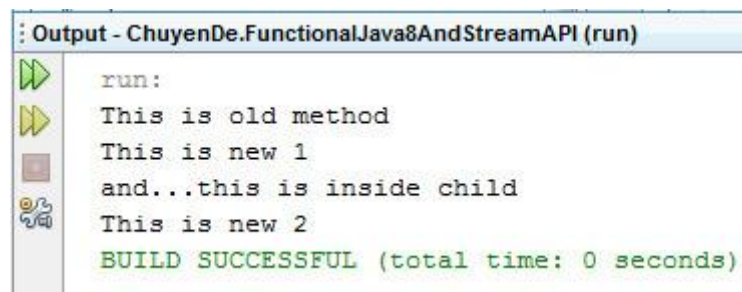
```
        oci.oldMethod();
```

```
        oci.newMethod1(); // Override phương thức ở interface (nếu cần)
```

```
        oci.newMethod2(); // Có thể sử dụng trực tiếp phương thức mới dc thêm
```

```
    }
```

```
}
```



Bài thực hành số 4:

Yêu cầu: tạo lớp DemoStreamJava8, trong đó viết các phương thức để thí nghiệm các tính năng xử lý dữ liệu tập hợp giới thiệu trong Stream API.

Code tham khảo:

```
package chuyende.streamapi;
```

```
import java.util.Arrays;
```

```
import java.util.IntSummaryStatistics;
```

```
import java.util.List;
```

```
import java.util.Random;
```

```
import java.util.function.Predicate;
```

```
import java.util.function.ToIntFunction;
```



```
import java.util.stream.Collectors;

/**
 *
 * @author minhvuvc
 */
public class DemoStreamJava8 {

    List<String> listStr = Arrays.asList("Tao", "", "Le", "Bui", "Mang cau", "",
"Nho");

    private void streamsFilter() {
        // Lấy về stream và lọc dữ liệu rỗng
        Predicate<String> pre = (str) -> {
            return !str.isEmpty();
        };
        // Lấy về stream và lọc dữ liệu str > 3 ký tự
        Predicate<String> pre2 = (str) -> str.length() > 3;

        List<String> filter = listStr.stream().filter(pre).collect(Collectors.toList());
        System.out.println(filter);
    }

    /**
     * In 3 số ngẫu nhiên sử dụng ForEach
     */
    private void streamsForeach() {
        Random random = new Random();
        // random.ints().limit(2).forEach(System.out::println); // Method Reference

        // Full code
        // IntConsumer ic = (value) -> {
        //     value = value * 2;
        //     System.out.println("#" + value);
        // };
        // random.ints().limit(3).forEach(ic);
        // Short code 1
        // random.ints().limit(3).forEach((val) -> System.out.println("@" + val));
        // Short code 2
        random.ints().limit(3).forEach((val) -> {
            System.out.println("(.)");
            System.out.println("val = " + val);
        });
    }
}
```

```
/**
 * Phương thức map() sử dụng để khớp phần tử với kết quả tương ứng Đoạn
code
 * sau in ra bình phương một số (duy nhất) dựa trên số đầu vào
 */
private void streamsMap() {
    List<Integer> numbers = Arrays.asList(3, 9, 3, 0);
    //    Function<Integer, Integer> f = new Function<Integer, Integer>() {
    //        @Override
    //        public Integer apply(Integer t) {
    //            return (t * t);
    //        }
    //    };
    List<Integer> squareList = numbers.stream().map((t) -> t *
t).distinct().collect(Collectors.toList());
    System.out.println("List: " + squareList.toString());
}

/**
 * Sắp xếp phần tử sử dụng hàm sort()
 */
private void streamsSort() {
    Random random = new Random();
    random.ints().limit(5).sorted().forEach(System.out::println);
}

/**
 * Tận dụng phần cứng đa lõi (core) xử lý trên nhiều luồng
 */
private void streamsParallelProcessing() {
    long count = listStr.parallelStream().filter(string -> string.isEmpty()).count();
    System.out.println("Số phần tử rỗng: " + count);
}

/**
 * Nhóm các phần tử phù hợp
 */
private void streamCollector() {
    List<String> list3Char = listStr.stream().filter(str -> str.length() >
3).collect(Collectors.toList());
    String str3Char = listStr.stream().filter(str -> str.length() >
3).collect(Collectors.joining(" - "));

    System.out.println("List 3 character: " + list3Char);
}
```

```

        System.out.println("String of 3 character: " + str3Char);
    }

    private void streamStatistics() {
        List<Integer> lstNumber = Arrays.asList(3, 5, 9, 2, 0, 4, 6);
        IntSummaryStatistics iss = lstNumber.stream().mapToInt(n ->
n).summaryStatistics();

        System.out.println("Max: " + iss.getMax());
        System.out.println("Min: " + iss.getMin());
        System.out.println("Average: " + iss.getAverage());
        System.out.println("Sum: " + iss.getSum());
        System.out.println("Count: " + iss.getCount());
    }

    private void streamStatisticsPerson() {
        List<Person> lstPerson = Arrays.asList(new Person("Minh", 25), new
Person("Tuấn", 19), new Person("Vũ", 30));
        ToIntFunction<Person> tifPer = (Person per) -> per.age;
        IntSummaryStatistics iss =
lstPerson.parallelStream().mapToInt(tifPer).summaryStatistics();

        System.out.println(".:Person:");
        System.out.println("Max: " + iss.getMax());
        System.out.println("Min: " + iss.getMin());
        System.out.println("Average: " + iss.getAverage());
        System.out.println("Sum: " + iss.getSum());
        System.out.println("Count: " + iss.getCount());
    }

    class Person {

        String name;
        int age;

        public Person(String name, int age) {
            this.name = name;
            this.age = age;
        }

    }

    /**
     * @param args the command line arguments

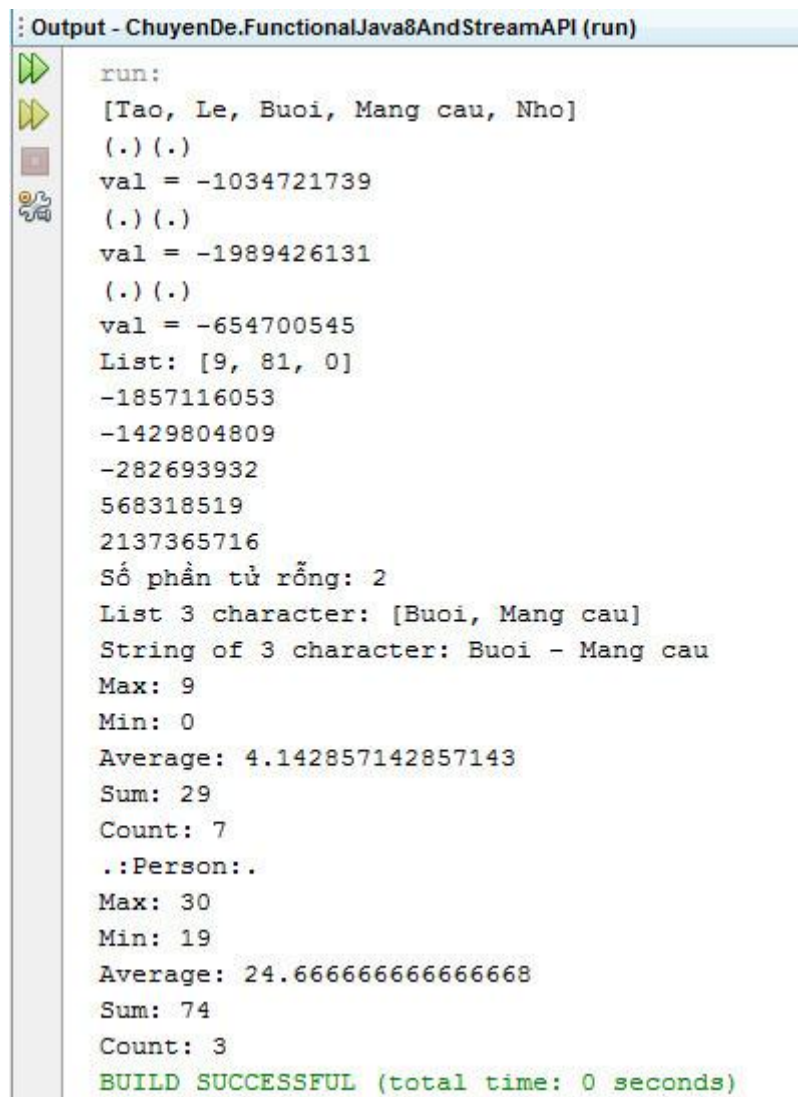
```

*/

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

```
    DemoStreamJava8 dmStream = new DemoStreamJava8();
    dmStream.streamsFilter();
    dmStream.streamsForeach();
    dmStream.streamsMap();
    dmStream.streamsSort();
    dmStream.streamsParallelProcessing();
    dmStream.streamCollector();
    dmStream.streamStatistics();
    dmStream.streamStatisticsPerson();
}
```

```
}
```



```
Output - ChuyenDe.FunctionalJava8AndStreamAPI (run)
run:
[Tao, Le, Buoi, Mang cau, Nho]
(.) (.)
val = -1034721739
(.) (.)
val = -1989426131
(.) (.)
val = -654700545
List: [9, 81, 0]
-1857116053
-1429804809
-282693932
568318519
2137365716
Số phần tử rỗng: 2
List 3 character: [Buoi, Mang cau]
String of 3 character: Buoi - Mang cau
Max: 9
Min: 0
Average: 4.142857142857143
Sum: 29
Count: 7
.:Person:..
Max: 30
Min: 19
Average: 24.666666666666668
Sum: 74
Count: 3
BUILD SUCCESSFUL (total time: 0 seconds)
```