Java 8 new Feature

Lê Thanh Nhân - LAB

Những tính năng mới trong Java8 nổi bật như lamda Expression, interface default và static methods, method reference. Một số snippet minh hoạ phục vụ LAB.

1. **Lamda Expessions:**

   Biểu thức Lambda trong Java8 được xem là tính năng mới nhất. cú pháp đơn giản

   parameter -> { expression body }

   Có các dạng:

   ***1.1. Optional type declaration**: Không cần khai báo kiểu (type) tham số.*

   ***1.2. Optional parenthesis around parameter**: Không cần khai báo tham số trong ngoặc đơn. Tuy nhiên, bắt buộc dùng dấu ngoặc đơn nếu có nhiều tham số.*

   ***1.3. Optional curly braces**: Không cần sử dụng dấu ngoặc nhọn trong phần thân (expression body) nếu nó chỉ chứa một câu lệnh.*

   ***1.4. Optional return keyword**: Tuỳ chọn từ khoá return nếu phần thân chỉ có một biểu thức duy nhất.*

   Snippet:

```
interface Calculator {
      int cal(int n, int m);
}

interface Message {
      void display(String message);
}

public class One_LamdaExpression{
    int test(int a, int b, Calculator calculator) {
      return calculator.cal(a, b);
    }
    public static void main(String[] args) {
        One_LamdaExpression one = new One_LamdaExpression();
        //Continue...
    }
}
```

```
public class One_LamdaExpression{
    int test(int a, int b, Calculator calculator) {
      return calculator.cal(a, b);
    }
    public static void main(String[] args) {
        One_LamdaExpression one = new One_LamdaExpression();
        // Without parenthesis
        Message first = mes -> System.out.println(mes);
        first.display("Without parenthesis");
        // With parenthesis
        Message second = (mes) -> System.out.println(mes);
        second.display("With parenthesis");
        // Without type declaration
        Calculator add = (n, m) -> n + m;
        System.out.println("20 + 10 = " + one.test(20, 10, add));
        // With type declaration
        Calculator sub = (int n, int m) -> n - m;
        System.out.println("20 - 10 = " + one.test(20, 10, sub));
        // Without return statement and without curly braces
        Calculator multiple = (n, m) -> n * m;
        System.out.println("20 x 10 = " + one.test(20, 10, multiple));
        // With return statement along with curly braces
        Calculator div = (n, m) -> { return n / m; };
        System.out.println("20 / 10 = " + one.test(20, 10, div));
    }
}
```

Output:

```
run:
Without parenthesis
With parenthesis
20 + 10 = 30
20 - 10 = 10
20 x 10 = 200
20 / 10 = 2
BUILD SUCCESSFUL (total time: 0 seconds)
```

2. Interface Default and Static Methods

Trước đây, các interface chỉ có thể có các abstract methods (public). Với Java8, các interface có thể có các static hay default methods.

Snippet:

```
interface DefaultStaticMethods{
    default void testDefault(){
        System.out.println("1.1. Default Methods is new Feature.");
    }

    static void testStatic(){
        System.out.println("1.2. Static Methods is too.");
    }
}

public class Two_DefaultStaticMethods implements DefaultStaticMethods{

    public static void main(String[] args) {
        new Two_DefaultStaticMethods().testDefault();
        DefaultStaticMethods.testStatic();
    }
}
```

Output:

```
run:
1.1. Default Methods is new Feature.
1.2. Static Methods is too.
BUILD SUCCESSFUL (total time: 0 seconds)
```

3. Method References

Method References có thể được sử dụng như là một thay thế ngắn hơn và dễ đọc hơn cho biểu thức lambda chỉ gọi một phương thức hiện có. Có bốn dạng tham chiếu:

3.1. Reference to a Static Method (*Using interface Message in class One_LamdaExpression*)

Snippet:

```
public class Three_MethodReferences01 {
    static void show(String s){
        System.out.println(s);
    }

    public static void main(String[] args) {

        // Reference to the static method
        Message mes = Three_MethodReferences01::show;
        // Call interface method
        mes.display("Reference to a Static Method.");
    }
}
```

Output:

```
run:
Reference to a Static Method.
BUILD SUCCESSFUL (total time: 0 seconds)
```

3.2. Reference to an Instance Method

Snippet:

```java
interface MyReferences{
    void display(String message);
}

public class Three_MethodReferences02 {
    public void methodRef(String s){
        System.out.println(s);
    }

    public static void main(String[] args) {
        Three_MethodReferences02 three = new Three_MethodReferences02();
        // Using Lamda Expression
        MyReferences mr = mes -> System.out.println(mes);
        mr.display("Lamda Expression.");
        /*
            Using Reference to an Instance Method
            1. Method reference using the Instance of the class
            2. Calling the method of functional interface
        */
        MyReferences ref = three::methodRef;
        ref.display("Reference to an Instance Method.");
    }
}
```

Output:

```
run:
Lamda Expression.
Reference to an Instance Method.
BUILD SUCCESSFUL (total time: 0 seconds)
```

3.3. Reference to a Constructor

Snippet:

```
interface Template{
    Three_MethodReferences03 get(String no, String name);
}

public class Three_MethodReferences03 {
    String no, name;

    public Three_MethodReferences03(String no, String name){
        this.no     = no;
        this.name   = name;
    }
    @Override
    public String toString(){
        return "No: " + this.no +"\nName: " + name;
    }

    public static void main(String[] args) {
        Template tmp;
        Three_MethodReferences03 three;
        tmp = Three_MethodReferences03::new;
        three = tmp.get("Student01","Lê Văn A");
        System.out.println(three.toString());
    }
}
```

Output:

```
run:
No: Student01
Name: Lê Văn A
BUILD SUCCESSFUL (total time: 0 seconds)
```