**Biểu thức Lambda trong Java 8 – Lambda Expressions**

*Đăng vào 07/05/2018 Được đăng bởi*[**GP Coder**](https://gpcoder.com/author/gpadmin/)***25025****Lượt xem*

**Lambda Expressions (Biểu thức Lambda)** là một trong những tính năng mới được giới thiệu trong Java 8.

Trước khi Java 8, mọi thứ chủ yếu là hướng đối tượng. Ngoại trừ các kiểu dữ liệu nguyên thủy (primitive type), mọi thứ trong java tồn tại dưới dạng đối tượng. Tất cả các lời gọi đến các method/ function sẽ được thực hiện bằng cách sử dụng các class hoặc object. Các method/ function không tồn tại độc lập.

Với Java 8, lập trình chức năng (functional programming) đã được giới thiệu. Vì vậy, chúng ta có thể dễ dàng sử dụng các chức năng ẩn danh (anonymous functions). Nó tạo điều kiện cho các lập trình viên lập trình Functional và phát triển ứng dụng đơn giản hơn rất nhiều so với những phiên bản trước đó. Nó cung cấp một cách rõ ràng và ngắn ngọn để đại diện cho một **Functional Interface** sử dụng một biểu thức lamda.

**Nội dung**[[Ẩn](https://gpcoder.com/3898-bieu-thuc-lambda-trong-java-8-lambda-expressions/)]

* [1 Lambda expression là gì ?](https://gpcoder.com/3898-bieu-thuc-lambda-trong-java-8-lambda-expressions/#Lambda_expression_la_gi)
  + [1.1 Tại sao sử dụng biểu thức Lambda?](https://gpcoder.com/3898-bieu-thuc-lambda-trong-java-8-lambda-expressions/#Tai_sao_su_dung_bieu_thuc_Lambda)
  + [1.2 Cú pháp của biểu thức Lambda](https://gpcoder.com/3898-bieu-thuc-lambda-trong-java-8-lambda-expressions/#Cu_phap_cua_bieu_thuc_Lambda)
  + [1.3 So sánh Lambda expression vs method](https://gpcoder.com/3898-bieu-thuc-lambda-trong-java-8-lambda-expressions/#So_sanhLambda_expression_vs_method)
  + [1.4 Sử dụng biểu thức Lambda](https://gpcoder.com/3898-bieu-thuc-lambda-trong-java-8-lambda-expressions/#Su_dung_bieu_thuc_Lambda)
* [2 Ví dụ sử dụng biểu thức Lambda](https://gpcoder.com/3898-bieu-thuc-lambda-trong-java-8-lambda-expressions/#Vi_du_su_dung_bieu_thuc_Lambda)
* [3 Các ví dụ khác với biểu thức Lambda](https://gpcoder.com/3898-bieu-thuc-lambda-trong-java-8-lambda-expressions/#Cac_vi_du_khac_voi_bieu_thuc_Lambda)
  + [3.1 Ví dụ biểu thức Lambda không có tham số](https://gpcoder.com/3898-bieu-thuc-lambda-trong-java-8-lambda-expressions/#Vi_du_bieu_thuc_Lambda_khong_co_tham_so)
  + [3.2 Ví dụ biểu thức Lambda có một tham số duy nhất](https://gpcoder.com/3898-bieu-thuc-lambda-trong-java-8-lambda-expressions/#Vi_du_bieu_thuc_Lambda_co_mot_tham_so_duy_nhat)
  + [3.3 Ví dụ biểu thức Lambda có nhiều tham số, sử dụng hoặc không sử dụng từ khóa return](https://gpcoder.com/3898-bieu-thuc-lambda-trong-java-8-lambda-expressions/#Vi_du_bieu_thuc_Lambda_co_nhieu_tham_so_su_dung_hoac_khong_su_dung_tu_khoa_return)
  + [3.4 Ví dụ biểu thức Lambda với vòng lặp Foreach](https://gpcoder.com/3898-bieu-thuc-lambda-trong-java-8-lambda-expressions/#Vi_du_bieu_thuc_Lambda_voi_vong_lap_Foreach)
  + [3.5 Ví dụ biểu thức Lambda tạo Thread](https://gpcoder.com/3898-bieu-thuc-lambda-trong-java-8-lambda-expressions/#Vi_du_bieu_thuc_Lambda_tao_Thread)
  + [3.6 Ví dụ biểu thức Lambda với Filter Collection Data](https://gpcoder.com/3898-bieu-thuc-lambda-trong-java-8-lambda-expressions/#Vi_du_bieu_thuc_Lambda_voi_FilterCollection_Data)
* [4 Phạm vi truy cập trong biểu thức Lambda](https://gpcoder.com/3898-bieu-thuc-lambda-trong-java-8-lambda-expressions/#Pham_vi_truy_cap_trong_bieu_thuc_Lambda)
  + [4.1 Truy cập biến local – Accessing local variables](https://gpcoder.com/3898-bieu-thuc-lambda-trong-java-8-lambda-expressions/#Truy_cap_bien_local_8211_Accessing_local_variables)
  + [4.2 Truy cập Accessing fields and static variables](https://gpcoder.com/3898-bieu-thuc-lambda-trong-java-8-lambda-expressions/#Truy_cap_Accessing_fields_and_static_variables)
  + [4.3 Truy cập với phương thức mặc định trong Interface – Accessing Default Interface Methods](https://gpcoder.com/3898-bieu-thuc-lambda-trong-java-8-lambda-expressions/#Truy_cap_voi_phuong_thuc_mac_dinh_trong_Interface_8211_Accessing_Default_Interface_Methods)

**Lambda expression là gì ?**

Lambda Expression (biểu thức Lambda) có thể được định nghĩa là một hàm ẩn danh, cho phép người dùng chuyển các phương thức làm đối số. Điều này giúp loại bỏ rất nhiều mã soạn sẵn.

Lambda Expression là một hàm không có tên và không thuộc bất kỳ lớp nào, không có phạm vi truy cập (private, public hoặc protected), không khai báo kiểu trả về.

**Tại sao sử dụng biểu thức Lambda?**

* Cung cấp bản implement cho Functional interface.
* Viết ít code hơn.
* Hiệu quả hơn nhờ hỗ trợ thực hiện tuần tự (sequential) và song song (parallel) thông qua Stream API.

**Cú pháp của biểu thức Lambda**

|  |  |
| --- | --- |
| 1 | (argument-list) -> {body} |

Biểu thức Lambda trong java gồm có 3 thành phần sau:

* **Argument-list**: có thể không có, có một hoặc nhiều tham số
* **Arrow-token**: được sử dụng để liên kết arguments-list và body của biểu thức.
* **Body**: chứa các biểu thức và câu lệnh cho biểu thức lambda.

Chúng ta có thể viết lambda expression bằng nhiều cách tùy thuộc vào việc sử dụng nó trong một số trường hợp như sau:

* **Tùy chọn khai báo kiểu dữ liệu**: Bạn không cần phải khai báo kiểu dữ liệu cho các parameter truyền vào. Trình biên dịch sẽ tự suy luận ra kiểu dữ liệu từ giá trị của các parameter.
* **Tùy chọn sử dụng dấu ngoặc ()**: Trong trường hợp bạn chỉ truyền vào một parameter duy nhất thì chúng ta có thể bỏ qua cặp dấu ngoặc **()**. Nếu như có nhiều parameter thì phải sử dụng dấu ngoặc.
* **Tùy chọn sử dụng dấu ngoặc {}**: Trong trường hợp phần body code của chúng ta chỉ thực hiện 1 statement duy nhất thì chúng ta cũng có thể loại bỏ luôn cặp dấu ngoặc **{}**.
* **Tùy chọn sử dụng lệnh return**: Trong biểu thức Lambda, nếu chỉ có một câu lệnh, bạn có thể sử dụng hoặc không sử dụng từ khoá return. Bạn phải sử dụng từ khóa return khi biểu thức lambda chứa nhiều câu lệnh.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | // No argument and one-statement method body    () -> expression    // One argument and one-statement method body    (parameters) -> expression    // Arguments separated by commas and block body    (arg1, arg2, ...) -> {      body-block  }    // With arguments and block body, return value    (arg1, arg2, ...) -> {      body-block;  **return** **return**-value;  } |

**So sánh Lambda expression vs method**

Một phương thức (method/ function) trong Java gồm các phần chính sau:

* Name: tên phương thức.
* Parameter list: danh sách các tham số.
* Body: biểu thức, câu lệnh xử lý.
* return type: kiểu dữ liệu trả về.

Một biểu thức Lambda (Lambda expression) trong Java gồm các phần chính sau:

* No name: không có tên phương thức, nó là một phương thức ẩn danh (anonymous method).
* Parameter list: danh sách các tham số.
* Body: biểu thức, câu lệnh xử lý.
* No return type: không có kiểu trả về tường minh, trình biên dịch Java 8 có thể tự suy luận ra kiểu dữ liệu trả về dựa vào code thực thi.

**Sử dụng biểu thức Lambda**

Để sử dụng biểu thức lambda, chúng ta cần tạo **Functional interface** của riêng mình hoặc sử dụng Functional interface do Java cung cấp. Một interface chỉ có 1 phương thức trừu tượng duy nhất được gọi là [Functional interface](https://gpcoder.com/3869-functional-interface-trong-java-8/). Ví dụ: Runnable, callable, Comparator, …

Sử dụng Functional interface:

* Trước Java 8: chúng ta tạo anonymous inner classes.
* Từ Java 8: sử dụng biểu thức lambda thay vì các anonymous inner classes.

**Ví dụ sử dụng biểu thức Lambda**

Ví dụ cách sắp xếp một tập hợp có sẵn trước phiên bản Java 8

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | **package** com.gpcoder.Lambda;    **import** java.util.Arrays;  **import** java.util.Collections;  **import** java.util.Comparator;  **import** java.util.List;    **public** **class** SortBefore8Example {    **public** **static** **void** main(String[] args) {            List<String> languages = Arrays.asList("Java", "C#", "C++", "PHP", "Javascript");            Collections.sort(languages, **new** Comparator<String>() {                @Override  **public** **int** compare(String o1, String o2) {  **return** o1.compareTo(o2);              }            });    **for** (String language : languages) {              System.out.println(language);          }      }  } |

Sử dụng Lambda của Java 8 ta có thể viết lại như sau:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | **package** com.gpcoder.Lambda;    **import** java.util.Arrays;  **import** java.util.Collections;  **import** java.util.List;    **public** **class** SortJava8Example {    **public** **static** **void** main(String[] args) {            List<String> languages = Arrays.asList("Java", "C#", "C++", "PHP", "Javascript");            Collections.sort(languages, (String o1, String o2) -> {  **return** o1.compareTo(o2);          });    **for** (String language : languages) {              System.out.println(language);          }      }  } |

Như bạn thấy sử dụng biểu thức Lambda code của chúng ta ngắn gọn hơn rất nhiều. Chúng ta không cần viết tên interface, tên phương thức.

Trong một khối lệnh nếu chỉ có một lệnh duy nhất, ta có thể bỏ đi **{ }**, khi đó đoạn code trên bạn có thể viết như sau:

|  |  |
| --- | --- |
| 1 | Collections.sort(languages, (String o1, String o2) -> o1.compareTo(o2)  ); |

Trình biên dịch (Compiler) của Java có thể biết được kiểu phần tử trong tập hợp cần sắp xếp là kiểu gì, trong ví dụ này là kiểu String, và vì vậy bộ so sánh ( Comparator) chắc chắn là so sánh các kiểu dữ liệu String. Đoạn code trên có thể viết ngắn gọn hơn như sau:

|  |  |
| --- | --- |
| 1 | Collections.sort(languages, (o1, o2) -> o1.compareTo(o2)); |

**Các ví dụ khác với biểu thức Lambda**

Một biểu thức lambda có thể có không, một hoặc nhiều tham số. Số lượng tham số của biểu thức Lambda phụ thuộc vào số lượng tham số của phương thức trừu tượng của Functional Interface.

Hãy xem các ví dụ sau:

**Ví dụ biểu thức Lambda không có tham số**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | **package** com.gpcoder.Lambda;    @FunctionalInterface  **interface** Sayable1 {  **public** String say();  }    **public** **class** LambdaExpression\_1 {  **public** **static** **void** main(String[] args) {          Sayable1 s = () -> {  **return** "I have nothing to say.";          };          System.out.println(s.say());      }  } |

**Ví dụ biểu thức Lambda có một tham số duy nhất**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | **package** com.gpcoder.Lambda;    @FunctionalInterface  **interface** Sayable2 {  **public** String say(String name);  }    **public** **class** LambdaExpression2 {  **public** **static** **void** main(String[] args) {            // Lambda expression with single parameter.          Sayable2 s1 = (name) -> {  **return** "Hello, " + name;          };          System.out.println(s1.say("GP Coder"));            // Shorter          Sayable2 s2 = name -> {  **return** "Hello, " + name;          };          System.out.println(s2.say("GP Coder"));            // Shortest          Sayable2 s3 = name -> "Hello, " + name;          System.out.println(s3.say("GP Coder"));      }  } |

**Ví dụ biểu thức Lambda có nhiều tham số, sử dụng hoặc không sử dụng từ khóa return**

Trong biểu thức Lambda, nếu chỉ có một câu lệnh, bạn có thể sử dụng hoặc không sử dụng từ khoá return. Bạn phải sử dụng từ khóa return khi biểu thức lambda chứa nhiều câu lệnh.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42 | **package** com.gpcoder.Lambda;    @FunctionalInterface  **interface** Addable {  **int** add(**int** a, **int** b);  }    **public** **class** LambdaExpression3 {    **public** **static** **void** main(String[] args) {            // Multiple parameters with data type in lambda expression          Addable ad1 = (**int** a, **int** b) -> (a + b);          System.out.println(ad1.add(10, 20));            // Multiple parameters in lambda expression          Addable ad2 = (a, b) -> (a + b);          System.out.println(ad2.add(10, 20));            // Lambda expression without return keyword.          Addable ad3 = (a, b) -> (a + b);          System.out.println(ad3.add(10, 20));            // Lambda expression with return keyword.          Addable ad4 = (a, b) -> {  **return** (a + b);          };          System.out.println(ad4.add(10, 20));            // Lambda expression without return keyword.          Addable ad5 = (a, b) -> (a + b);          System.out.println(ad5.add(10, 20));            // Lambda expression with multi-statement          Addable ad6 = (a, b) -> {  **int** sum = (a + b);  **return** sum;          };          System.out.println(ad6.add(10, 20));      }    } |

**Ví dụ biểu thức Lambda với vòng lặp Foreach**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | **package** com.gpcoder.Lambda;    **import** java.util.Arrays;  **import** java.util.List;    **public** **class** LambdaExpression4 {    **public** **static** **void** main(String[] args) {            List<String> languages = Arrays.asList("Java", "C#", "C++", "PHP", "Javascript");            // Using Lambda expression          languages.forEach(n -> System.out.println(n));      }  } |

**Ví dụ biểu thức Lambda tạo Thread**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | **package** com.gpcoder.Lambda;    **public** **class** LambdaExpression5 {    **public** **static** **void** main(String[] args) {          // Using an anonymous inner class          Runnable r1 = **new** Runnable() {  **public** **void** run() {                  System.out.println("Runnable 1");              }          };            // Using Lambda Expression for Funcational Interface          Runnable r2 = () -> System.out.println("Runnable 2");            r1.run();          r2.run();      }  } |

**Ví dụ biểu thức Lambda với Filter Collection Data**

Ví dụ hiển thị danh sách sản phẩm có giá 20000.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | **package** com.gpcoder.lambda;    **import** java.util.ArrayList;  **import** java.util.List;  **import** java.util.stream.Stream;    **class** Product {  **int** id;      String name;  **float** price;    **public** Product(**int** id, String name, **float** price) {  **super**();  **this**.id = id;  **this**.name = name;  **this**.price = price;      }  }    **public** **class** LambdaExpression6 {    **public** **static** **void** main(String[] args) {            List<Product> list = **new** ArrayList<>();          list.add(**new** Product(1, "Samsung A5", 17000f));          list.add(**new** Product(3, "Iphone 6S", 65000f));          list.add(**new** Product(2, "Sony Xperia", 25000f));          list.add(**new** Product(4, "Nokia Lumia", 15000f));          list.add(**new** Product(5, "Redmi4 ", 26000f));          list.add(**new** Product(6, "Lenevo Vibe", 19000f));            // using lambda to filter data          Stream<Product> filtered\_data = list.stream().filter(p -> p.price > 20000);            // using lambda to iterate through collection          filtered\_data.forEach(product -> System.out.println(product.name + ": " + product.price));      }  } |

Kết quả thực thi chương trình:

|  |  |
| --- | --- |
| 1  2  3 | Iphone 6S: 65000.0  Sony Xperia: 25000.0  Redmi4 : 26000.0 |

**Phạm vi truy cập trong biểu thức Lambda**

Việc truy cập các biến phạm vi bên ngoài từ các biểu thức lambda rất giống với các đối tượng ẩn danh (anonymous objects). Bạn có thể truy cập bất kỳ biến final, static hoặc biến chỉ được gán một lần. Biểu thức Lambda throw một lỗi biên dịch, nếu một biến được gán một giá trị lần thứ hai.

**Truy cập biến local – Accessing local variables**

Chúng ta có thể truy cập các biến final và biến chỉ được gán một lần.

Ví dụ:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | **package** com.gpcoder.Lambda;    @FunctionalInterface  **interface** Converter<F, T> {      T convert(F from);  }    **public** **class** Java8Scope1 {    **public** **static** **void** doSomething1() {  **final** **int** num = 1;          Converter<Integer, String> stringConverter = (from) -> String.valueOf(from + num);            stringConverter.convert(2); // 3      }    **public** **static** **void** doSomething2() {  **int** num = 1;          Converter<Integer, String> stringConverter = (from) -> String.valueOf(from + num);            stringConverter.convert(2); // 3      }    **public** **static** **void** doSomething3() {  **int** num = 1;          Converter<Integer, String> stringConverter = (from) -> String.valueOf(from + num);            stringConverter.convert(2); // 3            // Lambda expression will throw error at num varialbe.          // Local variable num defined in an enclosing scope must be final or effectively          // num = 3;      }    } |

**Truy cập Accessing fields and static variables**

Chúng ta có thể truy cập và thay đổi các trường hoặc biến static.

Ví dụ:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | **package** com.gpcoder.Lambda;    **public** **class** Java8Scope2 {    **static** **int** outerStaticNum;  **int** outerNum;    **void** testScopes() {          Converter<Integer, String> stringConverter1 = (from) -> {              outerNum = 1;  **return** String.valueOf(from);          };            Converter<Integer, String> stringConverter2 = (from) -> {              outerStaticNum = 1;  **return** String.valueOf(from);          };      }    } |

**Truy cập với phương thức mặc định trong Interface – Accessing Default Interface Methods**

Phương thức mặc định (Default methods) không thể truy cập bên trong Biểu thức Lambda.

**Tài liệu tham khảo:**

* <https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html>
* <https://www.javatpoint.com/java-lambda-expressions>
* <https://www.ntu.edu.sg/home/ehchua/programming/java/JDK8_NewFeatures.html>

**4.9**

10

**Nếu bạn thấy hay thì hãy chia sẻ bài viết cho mọi người nhé!**

SHARES

*Chuyên mục:*[**Java 8**](https://gpcoder.com/category/java-core/java-8/)*Được gắn thẻ:*[**Java 8**](https://gpcoder.com/tag/java-8/)

[Phương thức tham chiếu trong Java 8 – Method References](https://gpcoder.com/3882-phuong-thuc-tham-chieu-trong-java-8-method-references/)

[Phương thức forEach() trong java 8](https://gpcoder.com/3919-phuong-thuc-foreach-trong-java-8/)

**Có thể bạn muốn xem:**

* [Predicate trong Java 8](https://gpcoder.com/3963-predicate-trong-java-8/) **(21/05/2018)**
* [Base64 encoding và decoding trong Java 8](https://gpcoder.com/4144-base64-encoding-va-decoding-trong-java-8/) **(26/07/2018)**
* [Giới thiệu về Stream API trong Java 8](https://gpcoder.com/3923-gioi-thieu-ve-stream-api-trong-java-8/) **(14/05/2018)**
* [String Joiner trong Java 8](https://gpcoder.com/4139-string-joiner-trong-java-8/) **(23/07/2018)**
* [Phương thức forEach() trong java 8](https://gpcoder.com/3919-phuong-thuc-foreach-trong-java-8/) **(10/05/2018)**

**Bình luận**

1 bình luận