**Java Lambda Expressions**

Lambda expression is a new and important feature of Java which was included in Java SE 8. It provides a clear and concise way to represent one method interface using an expression. It is very useful in collection library. It helps to iterate, filter and extract data from collection.

The Lambda expression is used to provide the implementation of an interface which has functional interface. It saves a lot of code. In case of lambda expression, we don't need to define the method again for providing the implementation. Here, we just write the implementation code.

Java lambda expression is treated as a function, so compiler does not create .class file.

**Functional Interface**

Lambda expression provides implementation of *functional interface*. An interface which has only one abstract method is called functional interface. Java provides an anotation @*FunctionalInterface*, which is used to declare an interface as functional interface.

**Why use Lambda Expression**

1. To provide the implementation of Functional interface.
2. Less coding.

**Java Lambda Expression Syntax**

1. (argument-list) -> {body}

Java lambda expression is consisted of three components.

**1) Argument-list:** It can be empty or non-empty as well.

**2) Arrow-token:** It is used to link arguments-list and body of expression.

**3) Body:** It contains expressions and statements for lambda expression.

Let's see a scenario. If we don't implement Java lambda expression. Here, we are implementing an interface method without using lambda expression.

**EXAMPLE WITHOUT LAMBDA EXPRESSION**

|  |  |
| --- | --- |
| **package** com.snow; **import** java.time.\*; **import** java.time.format.DateTimeFormatter; **public class** Main {   **public static void** main(String[] args) {  **new** Thread(**new** codetorun()).start();   *//using anonymous class* **new** Thread(**new** Runnable() {  @Override  **public void** run() {  System.***out***.println(**"from runable anonymous class"**);  }  }).start();  } } **class** codetorun **implements** Runnable{  @Override  **public void** run() {  System.***out***.printf(**"from runnable class"**);  } } | from runable anonymous class  from runnable class |

**With lambda expression**

|  |  |
| --- | --- |
| **package** com.snow;  **public class** Main {   **public static void** main(String[] args) {  *//WITH LAMBDA EXPRESSION* **new** Thread(() -> System.***out***.println(**"from lambda expression"**)).start();  } } | from lambda expression |

**WITH LAMBDA EXPRESSION - with multiple line argument**

|  |  |
| --- | --- |
| **package** com.snow;  **public class** Main {   **public static void** main(String[] args) {  *//WITH LAMBDA EXPRESSION - with multiple line argument* **new** Thread(() ->{  System.***out***.println(**"from runable"**);  System.***out***.println(**"line 2"**);  System.***out***.format(**"the value is %d "**, 3);  }).start();  } } | from runable  line 2  the value is 3 |

**WITHOUT LAMBDA EXPRESSION - more about collection class**

|  |  |
| --- | --- |
| **package** com.snow;  **import** java.util.ArrayList; **import** java.util.Collections; **import** java.util.Comparator;  **public class** Main {   **public static void** main(String[] args) {  *//WITHOUT LAMBDA EXPRESSION - more about collection class* Employee snow = **new** Employee(**"Jon snow"**, 10);  Employee tygerrian = **new** Employee(**"Tygerrian"**, 5);  Employee ned = **new** Employee(**"Ned Stark"**, 25);  Employee lenisters = **new** Employee(**"Kingslayer"**, 35);   ArrayList<Employee> employees = **new** ArrayList<>();  employees.add(snow);  employees.add(tygerrian);  employees.add(ned);  employees.add(lenisters);   Collections.*sort*(employees, **new** Comparator<Employee>() {  @Override  **public int** compare(Employee emp1, Employee emp2) {  **return** emp1.getName().compareTo(emp2.getName());  }  });   **for** (Employee emp : employees){  System.***out***.println(emp.getName());  }  } } **class** Employee{  String **name**;  **int experience**;   *//constructor* **public** Employee(String name, **int** experience) {  **this**.**name** = name;  **this**.**experience** = experience;  }  *//getter and setter* **public** String getName() {  **return name**;  }  **public void** setName(String name) {  **this**.**name** = name;  }  **public int** getExperience() {  **return experience**;  }  **public void** setExperience(**int** experience) {  **this**.**experience** = experience;  } } | Jon snow  Kingslayer  Ned Stark  Tygerrian |

**WITH LAMBDA EXPRESSION - more about collection class**

|  |  |
| --- | --- |
| **package** com.snow;  **import** java.util.ArrayList; **import** java.util.Collections; **import** java.util.Comparator;  **public class** Main {   **public static void** main(String[] args) {  *//WITH LAMBDA EXPRESSION - more about collection class* Employee snow = **new** Employee(**"Jon snow"**, 10);  Employee tygerrian = **new** Employee(**"Tygerrian"**, 5);  Employee ned = **new** Employee(**"Ned Stark"**, 25);  Employee lenisters = **new** Employee(**"Kingslayer"**, 35);   ArrayList<Employee> employees = **new** ArrayList<>();  employees.add(snow);  employees.add(tygerrian);  employees.add(ned);  employees.add(lenisters);  *//with lambda implementation* Collections.*sort*(employees, (Employee emps1, Employee emps2) -> emps1.getName().compareTo(emps2.getName()));  *//it doesn’t mean you gotta use a EMPLOYEE type*  Collections.*sort*(employees, (emps1, emps2) ->   emps1.getName().compareTo(emps2.getName()));  **for** (Employee emp : employees){  System.***out***.println(emp.getName());  }  } } **class** Employee{  String **name**;  **int experience**;   *//constructor* **public** Employee(String name, **int** experience) {  **this**.**name** = name;  **this**.**experience** = experience;  }  *//getter and setter* **public** String getName() {  **return name**;  }  **public void** setName(String name) {  **this**.**name** = name;  }  **public int** getExperience() {  **return experience**;  }  **public void** setExperience(**int** experience) {  **this**.**experience** = experience;  } } | Jon snow  Kingslayer  Ned Stark  Tygerrian |

**WITHOUT LAMBDA EXPRESSION – ABOUT interface**

|  |  |
| --- | --- |
| **package** com.snow;  **import** java.util.ArrayList; **import** java.util.Collections; **import** java.util.Comparator;  **public class** Main {   **public static void** main(String[] args) {  *//WITH LAMBDA EXPRESSION - more about collection class* Employee snow = **new** Employee(**"Jon snow"**, 10);  Employee tygerrian = **new** Employee(**"Tygerrian"**, 5);  Employee ned = **new** Employee(**"Ned Stark"**, 25);  Employee lenisters = **new** Employee(**"Kingslayer"**, 35);   ArrayList<Employee> employees = **new** ArrayList<>();  employees.add(snow);  employees.add(tygerrian);  employees.add(ned);  employees.add(lenisters);   *//with lambda implementation* Collections.*sort*(employees, (emps1, emps2) ->  emps1.getName().compareTo(emps2.getName()));   **for** (Employee emp : employees){  System.***out***.println(emp.getName());  }  *//this gonna concat and make it upper case as well* String sillystring = *dotheinterface*(**new** upper() {  @Override  **public** String upperConcat(String s1, String s2) {  **return** s1.toUpperCase() + s2.toUpperCase();  }  },  employees.get(0).getName(), employees.get(1).getName());  System.***out***.printf(sillystring);  }   *//takes an object for the instance* **public final static** String dotheinterface(upper u, String s1, String s2){  **return** u.upperConcat(s1, s2);  }  } **class** Employee{  String **name**;  **int experience**;   *//constructor* **public** Employee(String name, **int** experience) {  **this**.**name** = name;  **this**.**experience** = experience;  }  *//getter and setter* **public** String getName() {  **return name**;  }  **public void** setName(String name) {  **this**.**name** = name;  }  **public int** getExperience() {  **return experience**;  }  **public void** setExperience(**int** experience) {  **this**.**experience** = experience;  } }  **interface** upper{  **public** String upperConcat(String s1, String s2); } | Jon snow  Kingslayer  Ned Stark  Tygerrian  JON SNOWKINGSLAYER |

**WITH LAMBDA EXPRESSION – ABOUT interface**

|  |  |
| --- | --- |
| **package** com.snow;  **import** java.util.ArrayList; **import** java.util.Collections; **import** java.util.Comparator;  **public class** Main {   **public static void** main(String[] args) {  *//WITH LAMBDA EXPRESSION - more about collection class* Employee snow = **new** Employee(**"Jon snow"**, 10);  Employee tygerrian = **new** Employee(**"Tygerrian"**, 5);  Employee ned = **new** Employee(**"Ned Stark"**, 25);  Employee lenisters = **new** Employee(**"Kingslayer"**, 35);   ArrayList<Employee> employees = **new** ArrayList<>();  employees.add(snow);  employees.add(tygerrian);  employees.add(ned);  employees.add(lenisters);   *//with lambda implementation* Collections.*sort*(employees, (emps1, emps2) ->  emps1.getName().compareTo(emps2.getName()));   **for** (Employee emp : employees){  System.***out***.println(emp.getName());  }  *//this gonna concat and make it upper case as well* upper up = (s1, s2) -> s1.toUpperCase() + s2.toUpperCase();  String sillystring = *dotheinterface*(up, employees.get(0).getName(), employees.get(1).getName());  System.***out***.printf(sillystring);  }   *//takes an object for the instance* **public final static** String dotheinterface(upper u, String s1, String s2){  **return** u.upperConcat(s1, s2);  }  } **class** Employee{  String **name**;  **int experience**;   *//constructor* **public** Employee(String name, **int** experience) {  **this**.**name** = name;  **this**.**experience** = experience;  }  *//getter and setter* **public** String getName() {  **return name**;  }  **public void** setName(String name) {  **this**.**name** = name;  }  **public int** getExperience() {  **return experience**;  }  **public void** setExperience(**int** experience) {  **this**.**experience** = experience;  } }  **interface** upper{  **public** String upperConcat(String s1, String s2); } | Jon snow  Kingslayer  Ned Stark  Tygerrian  JON SNOWKINGSLAYER |