Instead of using

**AN ANONYMOUS CLASS**

you can use

**A LAMBDA EXPRESSION**

And if this just calls one method, you can use

**A METHOD REFERENCE**

**What is it?**

Java provides a new feature called method reference in Java 8. It is the shorthand syntax to a lambda expression that executes just one method.

**What syntax?**

In a method reference, you place the object (or class) that contains the method before the :: operator and the name of the method after it without arguments.

ObjectOrClass :: methodName

**There are four types of method references:**

* A method reference to ***a static method***
* A method reference to ***an instance method of an object of a particular type***
* A method reference to ***an instance method of an existing object***
* A method reference to ***a constructor***

**What Advantages or Disadvantages?**

|  |  |
| --- | --- |
| Advantages | Disadvantages |
| 1. Used to refer method of functional interface. | 1. A methods reference can't be used for any method. They can be used only to replace a single-method lambda expression. |
| 1. It is compact and easy form of lambda expression. Each time when you are using lambda expression to just referring a method, you can replace your lambda expression with method reference. |  |

**Example**

| Method Reference Type | Example | | |
| --- | --- | --- | --- |
| Static method | class Numbers {  public static boolean isMoreThanFifty(int n1, int n2) {  return (n1 + n2) > 50;  }  public static List<Integer> findNumbers(  List<Integer> l, BiPredicate<Integer, Integer> p) {  List<Integer> newList = new ArrayList<>();  for(Integer i : l) {  if(p.test(i, i + 10)) newList.add(i);  }  return newList;  }  } | | public class Test {  public static void main(String args[]) {  List<Integer> list = Arrays.asList(12,5,45,18,33,24,40);  }  } |
| **Equivalent** | **Syntax** | **Example** |
| **Method References** | Class::staticMethod | findNumbers(list, Numbers::isMoreThanFifty); |
| **Lambda Expression** | (args) -> Class.staticMethod(args) | findNumbers(list, (i1, i2) -> Numbers.isMoreThanFifty(i1, i2)); |
| **Anonymous Class** |  | findNumbers(list, new BiPredicate<Integer, Integer>() {  public boolean test(Integer i1, Integer i2) {  return Numbers.isMoreThanFifty(i1, i2);  }  }); |
| Instance method of an object of a particular type | class Shipment {  public double calculateWeight() {  double weight = 0;  // Calculate weight  return weight;  }  } | | public class Test {  public List<Double> calculateOnShipments(  List<Shipment> l, Function<Shipment, Double> f) {  List<Double> results = new ArrayList<>();  for(Shipment s : l) {  results.add(f.apply(s));  }  return results;  }  public static void main(String args[]) {  List<Shipment> l = new ArrayList<Shipment>();  }  } |
| **Equivalent** | **Syntax** | **Example** |
| **Method References** | ObjectType::instanceMethod | calculateOnShipments(l, Shipment::calculateWeight); |
| **Lambda Expression** | (obj, args) -> obj.instanceMethod(args) | calculateOnShipments(l, s -> s.calculateWeight()); |
| **Anonymous Class** |  | calculateOnShipments(l, new Function<Shipment, Double>() {  public Double apply(Shipment s) { // The object  return s.calculateWeight(); // The method  }  }); |
| Instance method of an existing object | class Car {  private int id;  private String color;  // More properties  // And getter and setters  }  class Mechanic {  public void fix(Car c) {  System.out.println("Fixing car " + c.getId());  }  } | | public class Test {  public void execute(Car car, Consumer<Car> c) {  c.accept(car);  }  public static void main(String args[]) {  final Mechanic mechanic = new Mechanic();  Car car = new Car();  }  } |
| **Equivalent** | **Syntax** | **Example** |
| **Method References** | obj::instanceMethod | execute(car, mechanic::fix); |
| **Lambda Expression** | (args) -> obj.instanceMethod(args) | execute(car, c -> mechanic.fix(c)); |
| **Anonymous Class** |  | execute(car, new Consumer<Car>() {  public void accept(Car c) {  mechanic.fix(c);  }  }); |
| Constructor | **Equivalent** | **Syntax** | **Example** |
| **Method References**  (no arguments) | ClassName::new | Supplier<List<String>> s = ArrayList::new;  List<String> l = s.get(); |
| **Lambda Expression**  (no arguments) | () -> new ClassName() | Supplier<List<String>> s = () -> new ArrayList<String>();  List<String> l = s.get(); |
| **Anonymous Class**  (no arguments) |  | Supplier<List<String>> s = new Supplier() {  public List<String> get() {  return new ArrayList<String>();  }  };  List<String> l = s.get(); |
| **Method References**  (an argument) | ClassName::new | Function<String, Integer> f = Integer::new;  Integer i = f.apply(100); |
| **Lambda Expression**  (an argument) | (args) -> new ClassName(args) | Function<String, Integer> f = s -> new Integer(s);  Integer i = f.apply(100); |
| **Anonymous Class**  (an argument) |  | Function<String, Integer> f =  new Function<String, Integer>() {  public Integer apply(String s) {  return new Integer(s);  }  };  Integer i = f.apply(100); |
| **Equivalent** | **Syntax** | **Example** |
| **Method References**  (two arguments) | ClassName::new | BiFunction<String, String, Locale> f = Locale::new;  Locale loc = f.apply("en","UK"); |
| **Lambda Expression**  (two arguments) | (args, args2) -> new ClassName(args, args2) | BiFunction<String, String, Locale> f = (lang, country) -> new Locale(lang, country);  Locale loc = f.apply("en","UK"); |
| **Anonymous Class**  (two arguments) |  | BiFunction<String, String, Locale> f = new BiFunction<String, String, Locale>() {  public Locale apply(String lang, String country) {  return new Locale(lang, country);  }  };  Locale loc = f.apply("en","UK"); |

**Method References**

**จุดสำคัญ (Key Points)**

* A method reference is the shorthand syntax to a lambda expression that executes just one method.

Method reference เป็น syntax อย่างย่อ เพื่อให้ Lambda Expression นั้นประมวลผลด้วยmethod เดียว

* The syntax of a lambda expression is:

ObjectOrClassName :: methodName

* In a method reference, you place the object (or class) that contains the method before the :: operator and the name of the method after it without arguments.

ใน Method Reference เราจะเขียน object หรือ class ไว้หน้า :: แล้วตามด้วยชื่อ method ใน class นั้น โดยไม่ต้องระบุ argument

* There are four types of method references:

A method reference to **a static method**

A method reference to **an instance method of an object of a particular type**

A method reference to **an instance method of an existing object**

A method reference to **a constructor**

* For static methods, we have a lambda expression like the following:

(args) -> Class.staticMethod(args)

* That can be turned into the following method reference:

Class::staticMethod

* For instance methods of objects of a particular type, we have a lambda expression like the following:

(obj, args) -> obj.instanceMethod(args)

* Where an instance of an object is passed as an argument and one of its methods is executed with some optional(s) parameter(s).

1. เป็นที่ให้ Object ที่ถูกสร้างขึ้น ส่งผ่านตัวแปรเป็น argument

Supplier<List<String>> s = ArrayList::new;

1. เป็นที่สำหรับประกาศ method ไว้ใช้ประมวลผลด้วยพารามิเตอร์ที่มีเงื่อนไขบางอย่าง

execute(car, mechanic::fix);

* And that can be turned into the following method reference:

ObjectType::instanceMethod

* For instance methods of existing objects, we have a lambda expression like the following:

(args) -> obj.instanceMethod(args)

* That can be turned into the following method reference:

obj::instanceMethod

* For creating objects (calling a constructor), we have a lambda expression like the following:

(args) -> new ClassName(args)

* That can be turned into the following method reference:

ClassName::new