**Method Reference By Double Colon(::) Operator**

:: can be used for method reference and constrictor reference

:: (double colon operator) -> Alternative to lambda expression, its advantage is code reusability.

C++ double colon (scope resolution operator) and Java double colon operator concept and context is different. Don’t compare them.

We can use :: operator for method and constructor reference

======= Simple Lambda===

**public** **interface** MyInterface {

**public** **void** m1();

}

**public** **class** MethodReference {

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

MyInterface myInterface = () -> {

System.***out***.println("MethodReference.main()");

};

myInterface.m1();

}

}

==== using method reference===

**public** **class** MethodReference {

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

MyInterface myInterface = MethodReference::*m2*;

myInterface.m1();

}

**public** **static** **void** m2() {

System.***out***.println("MethodReference.m2()");

}

}

Output: MethodReference.m2()

Here (MethodReference::*m2* ) means we are referring method m2.

myInterface.m1(); means when ever we execute m1 then execute m2.

During execution of m1, it internally refers m2.

What is the advantage of method reference?

Code reusability is the biggest advantage.

Are there any restrictions?

Only the arguments types should match between m2 and m1 methods, return types and modifiers (public/private) . method names doesn’t matter.

In above example m2 is static. We can use concrete methods too (see below)

**public** **class** MethodReference {

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

MyInterface myInterface = **new** MethodReference()::m2;

myInterface.m1();

}

**public** **void** m2() {

System.***out***.println("MethodReference.m2()");

}

}

For static methods: MethodReference::*m2*;

For instance method: **new** MethodReference()::m2;

==Example

Runnable r = new MyRunnable();

Runnable r = lambda expression

Runnable r = method reference

**public** **class** MethodReferenceThread {

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

MethodReferenceThread methodReferenceThread = **new** MethodReferenceThread();

Runnable r = methodReferenceThread::m1;

Thread t = **new** Thread(r);

t.start();

**for** (**int** i = 0; i < 100; i++) {

System.***out***.println("MethodReferenceThread.main():" + i);

}

}

**public** **void** m1() {

**for** (**int** i = 0; i < 100; i++) {

System.***out***.println("MethodReferenceThread.m1():" + i);

}

}

}

Constructor Reference By Double Colon(::) Operator

**public** **class** ConstractorReference {

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

// MyInterface2 myInterface2 = () -> new TestClass();

// myInterface2.get();

MyInterface2 myInterface2 = TestClass::**new**;

myInterface2.get();

}

**public** **void** m2() {

System.***out***.println("MethodReference.m2()");

}

}

**class** TestClass {

**public** TestClass() {

System.***out***.println("TestClass constructor");

}

}

**interface** MyInterface2 {

**public** TestClass get();

}