

Method Reference in Java :

* It is a new feature introduced from JDK 1.8V.

* Actually, It is an improvement over lambda expression because while working with lambda we need to write the method body but by using method reference, we can refer an existing method from our project OR java API by using :: **operator**.

* It is mainly used to write concise coding as well as we can reuse an existing method multiple times.

There are 4 types of Method reference :

- 1) Instance Method Reference (objectReference :: instanceMethodName)
- 2) Static Method Reference (ClassName :: staticMethodName)
- 3) Constructor Reference (ClassName :: new)
- 4) Arbitrary (Object of any type) Method Reference (ClassName::instanceMethodName)

Example :

```
-----
package com.ravi.basic;

@FunctionalInterface
interface Worker
{
    void work();
}

public class MethodReferenceDemo1
{
    public static void main(String[] args)
    {
        //Lambda Expression
        Worker w1 = () -> System.out.println("Worker is working");
        w1.work();

        System.out.println(".....");

        Worker w2 = new Employee().:work;
        w2.work();
    }
}

class Employee
{
    public void work()
    {
        System.out.println("Employee is working..");
    }
}
```

```
package com.ravi.basic;

@FunctionalInterface
interface Worker
{
    void work();
}

public class MethodReferenceDemo1
{
    public static void main(String[] args)
    {
        Worker w1 = Employee::getSalary;
        w1.work();
    }
}

class Employee
{
    public static void getSalary()
    {
        System.out.println("Employee is working for Salary");
    }
}
```

```
package com.ravi.basic;

@FunctionalInterface
interface Worker
{
    void work(double salary);
}

public class MethodReferenceDemo1
{
    public static void main(String[] args)
    {
        Worker w1 = Employee::getSalary;
        w1.work(35000);
    }
}

class Employee
{
    public static void getSalary(double salary)
    {
        System.out.println("Employee Salary is :"+salary);
    }
}
```

Here parameter type and method return type both are compulsory.

```
class Foo
{
    public static void printData(String message)
    {
        System.out.println(message+" Enjoy your weekend");
    }
}

public class MethodReferenceDemo1
{
    public static void main(String[] args)
    {
        Consumer<String> cons = Foo::printData;
        cons.accept("Hello Everyone");
    }
}
```

Static Method Reference :(ClassName::staticMethodName)

```
-----
package com.ravi.static_method_ref;

import java.util.Vector;

class EvenOrOdd
{
    public static void isEven(int number)
    {
        if (number % 2 == 0)
        {
            System.out.println(number + " is even");
        }
        else
        {
            System.out.println(number + " is odd");
        }
    }
}

public class StaticMethodReferenceDemo1
{
    public static void main(String[] args)
    {
        Vector<Integer> numbers = new Vector<>();
        numbers.add(5);
        numbers.add(2);
        numbers.add(9);
        numbers.add(12);
        numbers.add(15);

        //By using Lambda Expression
        numbers.forEach(num -> EvenOrOdd.isEven(num));

        System.out.println(".....");

        //By using Method Reference
        numbers.forEach(EvenOrOdd::isEven);
    }
}
```

Instance Method Reference :

```
-----
package com.ravi.instance_method;

@FunctionalInterface
interface Trainer
{
    void getTraining(String name, int experience);
}

class InstanceMethod
{
    public void getTraining(String name, int experience)
    {
        System.out.println("Trainer name is :"+name+" having "+experience+" years of experience.");
    }
}

public class InstanceMethodDemo1
{
    public static void main(String[] args)
    {
        //Using Lambda Expression
        Trainer trainer = (name, exp) -> System.out.println("Trainer name is :"+name+" having "+exp+" years of experience.");
        trainer.getTraining("Mr Scott", 10);

        //By using Method reference
        InstanceMethod im = new InstanceMethod();
        Trainer t1 = im::getTraining;
        t1.getTraining("Mr Smith", 5);
    }
}
```
