Method References

eazy bytes

Sometimes, there might already a method that carries out exactly the action that you'd like to pass on inside lambda code. In such cases it would be nicer to pass on the method instead of duplicating the code again.

This problem is solved using method references in Java 8. Method references are a special type of lambda expressions which are often used to create simple lambda expressions by referencing existing methods.

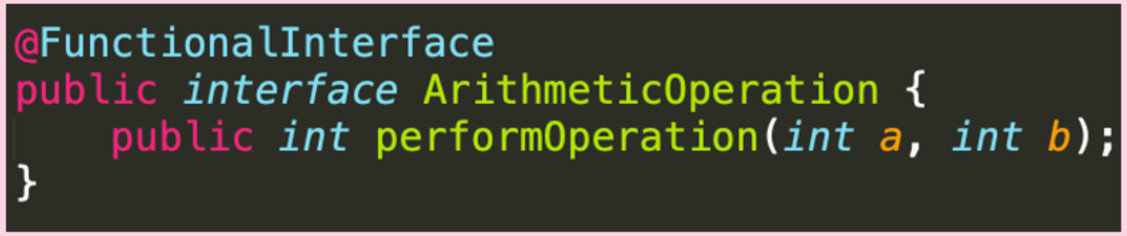
There are 4 types of method references introduced in Java 8,

- Static Method Reference (**Class::staticMethod**)
- Reference to instance method from instance (**objRef::instanceMethod**)
- Reference to instance method from class type (**Class::instanceMethod**)
- Constructor Reference (**Class::new**)

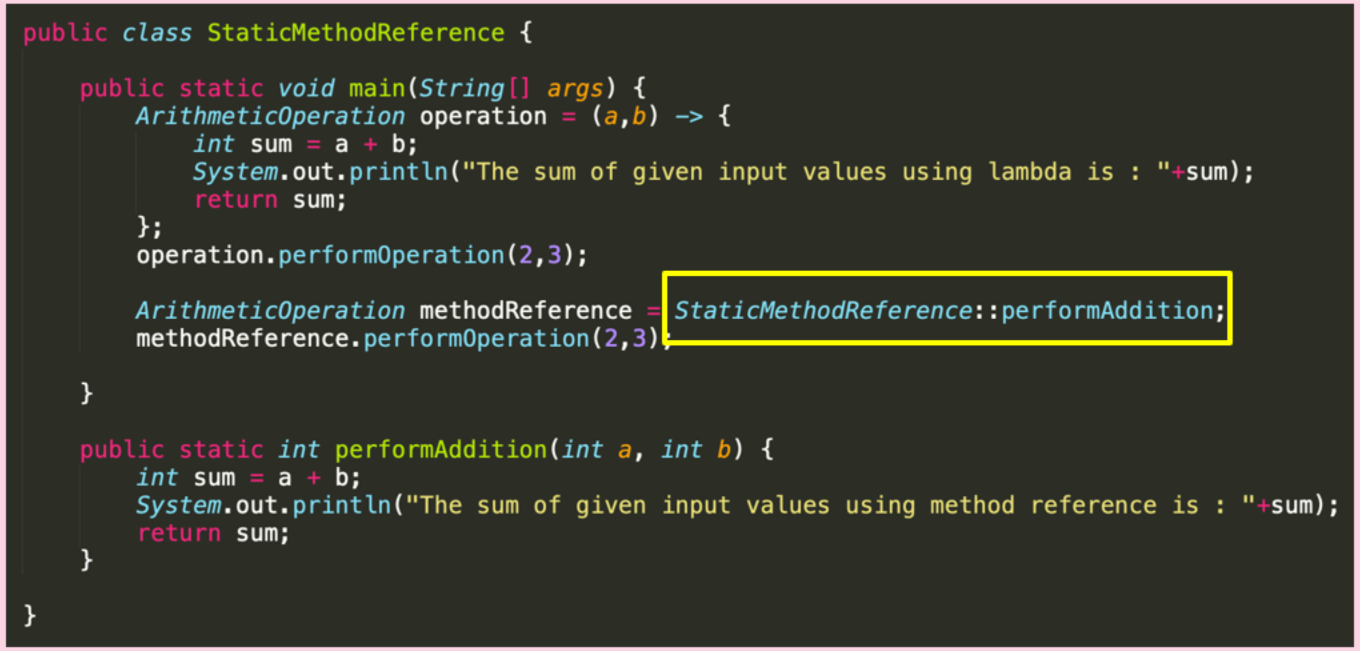
As you might have observed, Java has introduced a new operator **:: (double colon)** called as Method Reference Delimiter. In general, one don't have to pass arguments to method references.

Static Method Reference (Class::staticMethod)

Sometimes, there might already a static method that carries out exactly the action that you'd like to pass on inside lambda code. In such cases it would be nicer to pass on the method instead of duplicating the code again.



```java
@FunctionalInterface
public interface ArithmeticOperation {
    public int performOperation(int a, int b);
}
```

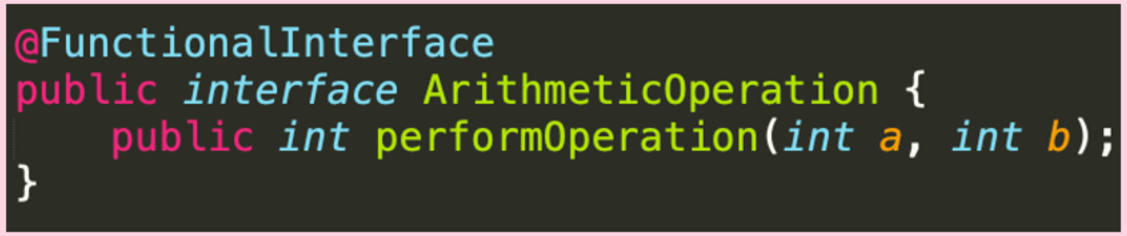


```java
public class StaticMethodReference {

    public static void main(String[] args) {
        ArithmeticOperation operation = (a,b) -> {
            int sum = a + b;
            System.out.println("The sum of given input values using lambda is : "+sum);
            return sum;
        };
        operation.performOperation(2,3);

        ArithmeticOperation methodReference = StaticMethodReference::performAddition;
        methodReference.performOperation(2,3);

    }

    public static int performAddition(int a, int b) {
        int sum = a + b;
        System.out.println("The sum of given input values using method reference is : "+sum);
        return sum;
    }

}
```

- ✓ As you can see first we have written a lambda expression code for the Functional interface 'ArithmeticOperation' to calculate the sum of given 2 integers.
- ✓ Later since we have same logic of code present inside a static method we used static method reference as highlighted
- ✓ This way we can leverage the existing static methods code to pass the behavior, instead of writing lambda code again
- ✓ Using method references is mostly advised if you already have code written inside a method or if your code inside lambda is large(we can put in separate method)

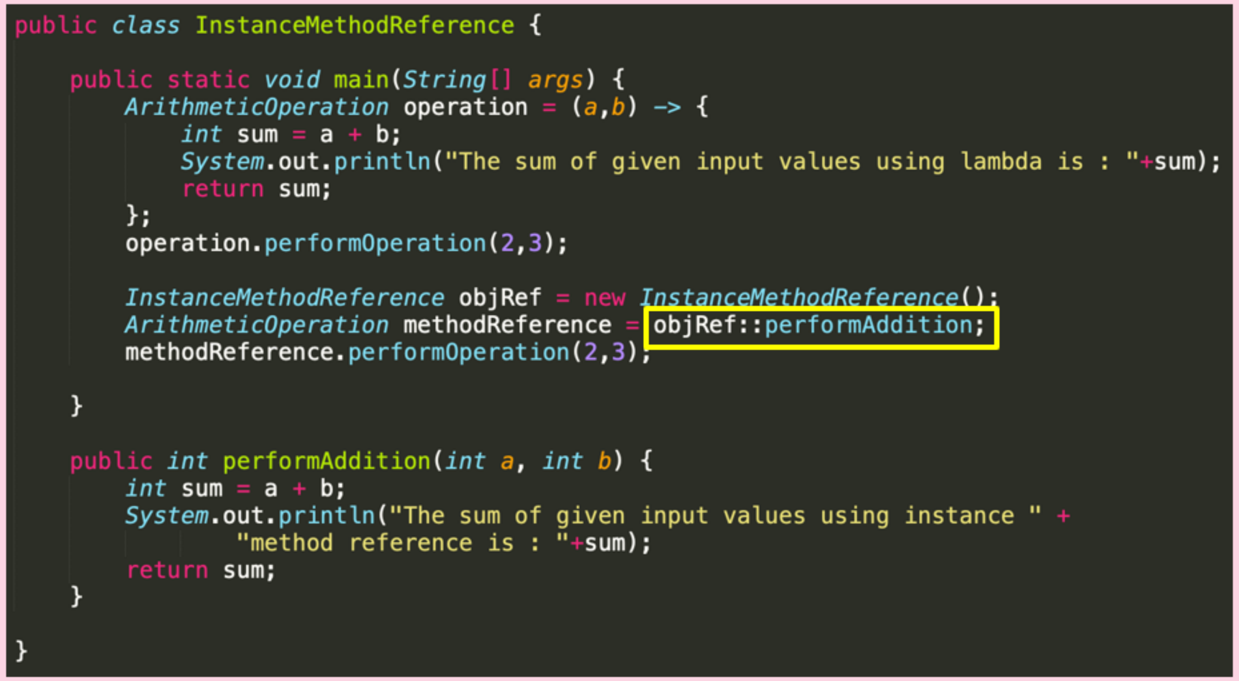
Instance method Reference using instance (objRef::instanceMethod)

eazy bytes



```java
@FunctionalInterface
public interface ArithmeticOperation {
    public int performOperation(int a, int b);
}
```

- ✓ As you can see first we have written a lambda expression code for the Functional interface 'ArithmeticOperation' to calculate the sum of given 2 integers.



```java
public class InstanceMethodReference {

    public static void main(String[] args) {
        ArithmeticOperation operation = (a,b) -> {
            int sum = a + b;
            System.out.println("The sum of given input values using lambda is : "+sum);
            return sum;
        };
        operation.performOperation(2,3);

        InstanceMethodReference objRef = new InstanceMethodReference();
        ArithmeticOperation methodReference = objRef::performAddition;
        methodReference.performOperation(2,3);

    }

    public int performAddition(int a, int b) {
        int sum = a + b;
        System.out.println("The sum of given input values using instance " +
                "method reference is : "+sum);
        return sum;
    }

}
```

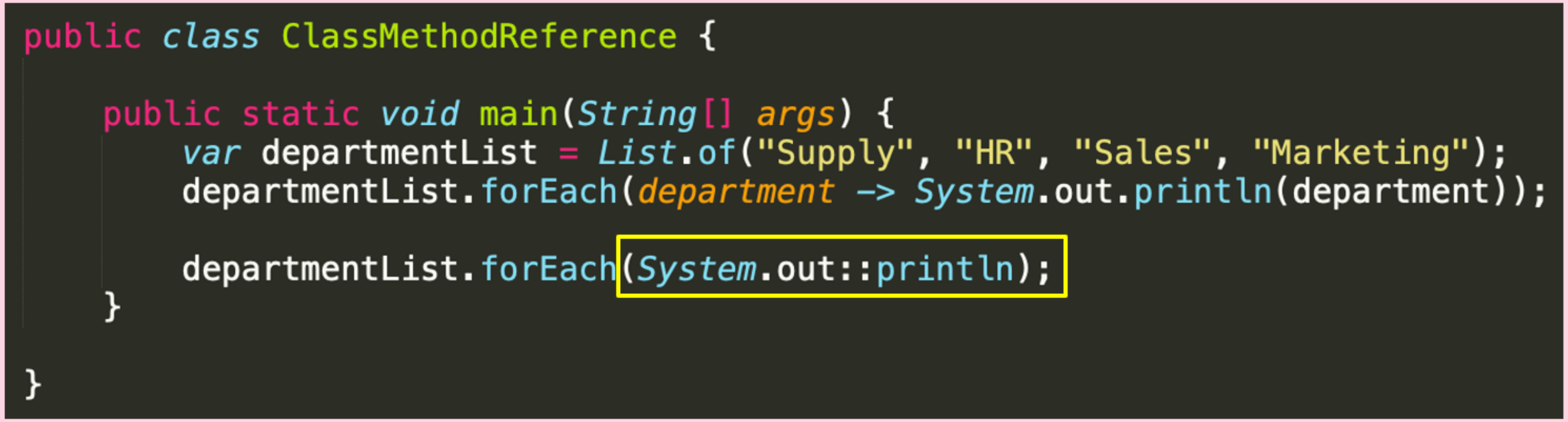
- ✓ Later since we have same logic of code present inside a instance method, we used instance method reference as highlighted using an object of the class

- ✓ This way we can leverage the existing static methods code to pass the behavior, instead of writing lambda code again

Instance method Reference using Class type (Class::instanceMethod)

eazy bytes

- This type of method reference is similar to the previous example, but without having to create a custom object
- As you can see first we have written a lambda expression code inside forEach method to print all the list elements
- Later since we have same logic of code present inside a method of System.out, we used instance method reference as highlighted using class type itself
- This way we can leverage the existing methods code to pass the behavior, instead of writing lambda code again

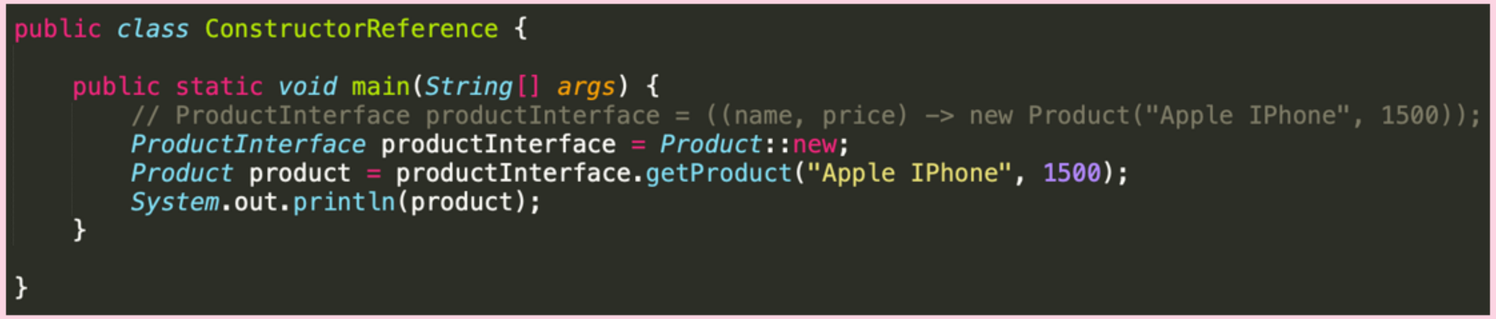


```java
public class ClassMethodReference {

    public static void main(String[] args) {
        var departmentList = List.of("Supply", "HR", "Sales", "Marketing");
        departmentList.forEach(department -> System.out.println(department));

        departmentList.forEach(System.out::println);
    }

}
```

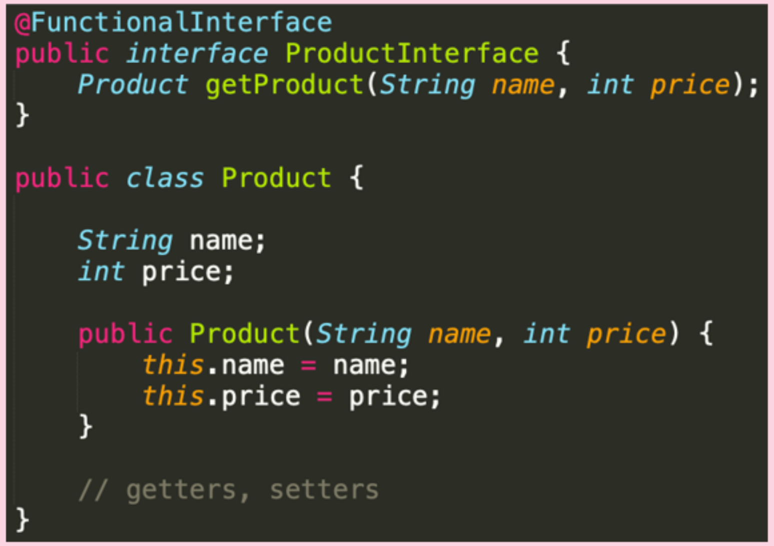
Constructor Reference (Class::new)

eazy bytes

- ✓ Sometimes, the body of a lambda expression will be just an object creation expression. In such scenarios, we can use Constructor Reference instead of writing the lambda expression
- ✓ Constructor references are just like method references, except that the name of the method is new. For example, Product ::new is a reference to a Product constructor
- ✓ As you can see we have used constructor reference in the place of lambda code to create a new product details using functional interface 'ProductInterface'



```java
public class ConstructorReference {

    public static void main(String[] args) {
        // ProductInterface productInterface = ((name, price) -> new Product("Apple IPhone", 1500));
        ProductInterface productInterface = Product::new;
        Product product = productInterface.getProduct("Apple IPhone", 1500);
        System.out.println(product);
    }

}
```



```java
@FunctionalInterface
public interface ProductInterface {
    Product getProduct(String name, int price);
}

public class Product {

    String name;
    int price;

    public Product(String name, int price) {
        this.name = name;
        this.price = price;
    }

    // getters, setters
}
```

Few other examples of Constructor reference are,

- String::new
- Integer::new
- ArrayList::new
- UserDetail::new