**Java 8**

**Method Reference**

**It is used to simply the code written inside {} of Lambda Expression**.

**Syntax :**

**ClassName::methodName**

Ex,

// Using Lambda Expression  
Function<String,String> usingLambda = word -> word.toUpperCase();  
  
// Using Method Reference  
**Function<String,String> usingMethodReference = String::toUpperCase;**

// In Functional Interface  
// Using Lambda Expression  
 Function<String,String> usingLambda = word -> word.toUpperCase();  
 System.*out*.println(usingLambda.apply("AadaPadaKonPada"));  
  
 // Using Method Reference  
 **Function<String,String> usingMethodReference = String::toUpperCase;**  
 System.*out*.println(usingMethodReference.apply("bantiWedsBabli"));  
  
 // In Consumer Interface  
 List<Student> studentList = StudentDataBase.*getAllStudents*();  
 // Using Lambda Expression  
 Consumer<Student> usingLambda1 = curStudent -> System.*out*.println(curStudent);  
 studentList.forEach(usingLambda1);  
  
 // Using Method Reference  
 Consumer<Student> usingMethodReference1 = System.*out*::println;  
 studentList.forEach(usingMethodReference1);  
  
 Consumer<Student> usingLambda2 = curStudent -> curStudent.printActivities();  
 studentList.forEach(usingLambda2);  
  
 Consumer<Student> usingMethodReference2 = Student::printActivities;  
 studentList.forEach(usingMethodReference2);

**Implementing Method Reference where it cannot be directly implemented**

public class MethodReferenceIndirectImplementation {  
 // Implementing Method Reference where it cannot be directly implemented  
 static Supplier<Student> *studentSupplier* = () -> new Student("bimlesh",3,4.4,"male", Arrays.*asList*("kirket", "tebal tanis","fud"));  
  
  
 // We cannot directly create method reference for Predicate Interface. But we can create boolean method and use it.  
 // Using Lambda Expression  
 static Predicate<Student> *usingLamda* = student -> student.getGpa() > 3;  
  
 // Using Method Reference Expression  
 static boolean gpaGreater3 (Student s){  
 return s.getGpa() > 3;  
 }  
 static Predicate<Student> *usingMethodReference* = MethodReferenceIndirectImplementation::*gpaGreater3*;  
  
 public static void main(String[] args) {  
 System.*out*.println(*usingLamda*.test(*studentSupplier*.get()));  
 System.*out*.println(*usingMethodReference*.test(*studentSupplier*.get()));  
 }  
}

**Note: It’s not necessary that every code must be compatible with Method Reference. Bss ye aur short code likhne ke liye bana h.**

**Constructor Reference**

**It is used to create a new object of a class but only using Functional Interface. Bss itna hi.**

**Syntax:**

**ClassName::new**

Eg,

Supplier<Student> newStudent = Student::new;  
// Note upar koi input parameter nhi h. Therefore, we must have a Default Constructor in that mentioned class. (here in Student Class).

System.*out*.println(newStudent.get());  
  
Function<String,Student> newStu = Student::new; // Created a new object which takes  
// Note upar ek input parameter h. Therefore, we must have a parameterized constructor with one parameter in that mentioned class (here in Student Class).

System.*out*.println(newStu.apply("champak"));



Output ->

**Limitation using Lamda Expression**

1. It’s not allowed to use the same local variable name as a lambda parameter or insidethe lambda body. Eg,

public static void main(String[] args) {  
 int i = 5;  
 Consumer<Integer> cs = i -> System.*out*.println(i); // Error -> java: variable i is already defined in method main  
 cs.accept(34);  
}

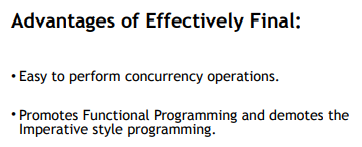
1. It’s not allowed to reassign a new value to a local variable in the lambda expression. Eg,

public static void main(String[] args) {  
 int i = 5;  
 Consumer<Integer> cs = j -> {  
 i =10; // Error dega  
 System.*out*.println(j);  
 };

.

But we can reassign Reference Variable in Lambda Body.

public class LamdaRestriction {  
 static int *z* = 6;  
 public static void main(String[] args) {  
 Consumer<Integer> c = x -> {  
 *z*++; // chalega  
 System.*out*.println(i);  
 };  
 }  
}

****

**Effectively Final:** Even though the local variables are not declared final, they are still not allowed to be modified in the Lamda Body. Bss issi ko bolte Effectively Final

**Streams API**