Method References in java

1. Provides implementation for Functional Interfae
2. Method references always pointing or referencing implementation of a interface method.
3. It is the simple way of providing implementation for functional interface by writing less code.
4. Method references internlly creates lambda expression
5. Method references always uses existed method logic as implementation for functional interface
6. Method references internally creates all four types of lambdas....
7. The signature of existed method should be same as functional interface method signature
8. "::" --> method reference operator

NARESH i technologies

```java
@FunctionalInterface
interface I{
    public abstract void m1();
}
class A{
    static public void test() {
        System.out.println("implementation for m1-I");
    }
}
public class Test {

    public static void main(String[] args) {
        I obj = A::test;

        /*
         * () -> { System.out.println("implementation for m1-I"); };
         */
    }
}
```

--> Method references ka use karke ham existing methods ko refer kar sakte hain bina unhein dobara define kiye hue.

--> jab ham mtd. reference ka use kar kai call karta hai "::" isa use kar kai to background mai ek lambda expresion create hota hai jo ise case mai test() mth. ko call karta aur hai uska andar ka content ko implement karta hai.

Q. Are static methods of functional interface available in the implementation classes?

Ans.: In a functional interface, static methods are not considered abstract and, therefore, are not required to be implemented by the classes that implement the functional interface. Static methods in a functional interface are generally utility methods associated with the interface itself and are meant to be invoked on the interface rather than on instances of its implementation classes.

```java
@FunctionalInterface
interface MyFunctionalInterface {
    void myMethod();

    // Static method in the interface
    static void staticMethod() {
        System.out.println("Static method in the interface");
    }
}

class MyImplementationClass implements MyFunctionalInterface {
    @Override
    public void myMethod() {
        System.out.println("Implementation of myMethod");
    }
}

public class Main {
    public static void main(String[] args) {
        MyImplementationClass obj = new MyImplementationClass();
        obj.myMethod();  // Output: Implementation of myMethod

        // Static method called on the interface
        MyFunctionalInterface.staticMethod();  // Output: Static method in the interface
    }
}
```

--> In this example, MyFunctionalInterface is a functional interface with a static method staticMethod(). The MyImplementationClass implements the MyFunctionalInterface but does not need to provide an implementation for the static method.

You can invoke the static method directly on the interface, as shown in the Main class. The static method is associated with the interface itself and is not tied to instances of the implementing classes.

```java
interface I{
    //public abstract void m1();
    public abstract int m2(int x);
}

class A{
    /*static public void test() {
        System.out.println("this implementation for m1-I");
    }*/
    static public int test(int x) {
        return x*x;
    }
}
public class Test{

    public static void main(String[] args) {
        I obj = A::test;
        int result = obj.m2(5);
        System.out.println("result: "+result);
```

--> Jab ham mtd. referencing ka use karta hai too hmasa yaad rakhna chahiya ki dono class kai mtd. ka signature same hona chahiyai.

--> Iska mtlb yai hai ki unka return type and jo parameter ham pass kar raha hai mtd. sai wo dono same hona chahiyai nahi too error show hoga.

Test.java

```java
@FunctionalInterface
interface I{
    public abstract void m1();
}

class A{
    public  void test() {
        System.out.println("implementation for m1");
    }
}
public class Test{

    public static void main(String[] args) {
        A obj1 = new A();
        I obj2 = obj1::test;
        obj2.m1();

        I obj = new A()::test;
        obj.m1();
    }
}
```

mtd I

mtd II

--> ya par dia gya dona mtd. mai non-static referencing ko initiate karna ka method diya hua hai. Ine mai sai kisi bhi method ka use kar ka non-static referencing koi implement kar sakta hai.

--> yai ek trah ki summary hai, jismai ham kis referencing mtd. ka liya kis keyword ka use karta hai wo btaya ja raha hai.

constructor method reference

==========

static method reference	classname::methodname
instance method reference	obj/referenece::methodname
costructor method reference	classname::new

Test.java

```java
@FunctionalInterface
interface I{
    public abstract void m1();
}
class A{
    A(){
        System.out.println("this logic is the implementation for m1-I");
    }
}
public class Test{
    public static void main(String[] args) {
        I obj = A::new;
        obj.m1();
    }
}
```

--> Yaha par constructor mtd. reference ka kiase use hota hai wo dikhaya gya hai.

--> Constructor reference ka use karna ka liya ham "classname::new;" ise keyword ka use karta hai.