Method and constructor references by using :: operator

Functional interface can refer lambda expression and function interface can also refer method references.

1) Functional interface method can be mapped to a specific method definition by using :: (double colon) operator.

This is called as method reference.

2) The specified method can be either static method or instance method(non static).

3) Functional interface method and the specified method must have same number of arguments and same type of arguments.

expect this the remaining things like return type, method name, modifiers etc are not required to be matched.

if the specified method is static method

Classname :: method-name

if the specified method is instacne method

objref :: method-name

Hence lambda expression can be replace with methd reference.

Hence method reference is alternative syntax to lambda expression.

Runnable r = () -> {

for(int i=0;i<65535;i++)

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

};

Thread t = new Thread(r);

t.start();

for(int i=0;i<65535;i++)

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

=====================================

public class Demo1{

public static void main(String...x){

Demo1 d = new Demo1();

Runnable r1 = Demo1 :: method1;

Runnable r2 = d :: method2;

Thread t1 = new Thread(r1);

Thread t2 = new Thread(r2);

t1.start();

t2.start();

for(int i=0;i<65535;i++)

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

}

public static void method1(){

for(int i=0;i<65535;i++)

System.out.println("child1 thread");

}

public void method2(){

for(int i=0;i<65535;i++)

System.out.println("child2 thread");

}

}

===============================================

interface Test{

public void method(int i);

}

class MyTest{

public void function(int n){

for(int i=0;i<n;i++){

System.out.println("execution done by the function");

}

}

}

public class MethodRefDemo2 {

public static void main(String[] args){

MyTest mt = new MyTest();

Test t = mt::function;

t.method(5);

}

}

The main advantage of method referene is we can use already existing code to implement function interfaces (code resusability).

**Constrcutor references**

constructor arguments must be matched with functional interface abstract method arguments.

syntax: classname :: new

example:

class Point{

private int x;

private int y;

Point() {

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

}

Point(int x, int y){

System.out.println("Argument constructor invoked");

this.x = x;

this.y = y;

}

public void setX(int a){

x = a;

}

public void setY(int b){

y = b;

}

public String toString(){

return x+","+y;

}

}

interface PointLambda2{

public abstract Point getPoint();

}

interface PointLambda{

public abstract Point getPoint(int x, int y);

}

public class ConstructorRef{

public static void main(String[] args){

PointLambda p = (a,b) ->{

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

Point point = new Point();

point.setX(a); point.setY(b);

return point;

};

PointLambda p1 = (a,b) ->{

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

return new Point(a,b);

};

PointLambda p2 = Point::new;

PointLambda2 p3 = Point::new;

System.out.println(p.getPoint(2,3));

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

System.out.println(p1.getPoint(2,3));

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

System.out.println(p2.getPoint(2,3));

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

System.out.println(p3.getPoint());

}

}

Note: In method and constructor references the argument types must be matched.