Constructor Reference : [ClassName::new]
----
* We can refer any constructor by using ClassName::new

```java
package com.ravi.contrustor_ref;

import java.util.function.Supplier;

class Sample
{
    public Sample()
    {
        System.out.println("No Argument constructor!!!");
    }

    @Override
    public String toString()
    {
        return "Sample []";
    }
}

public class ConstructorRefDemo1
{
    public static void main(String[] args)
    {
        Supplier<Sample>  sup = Sample::new;
        Sample sample = sup.get();
        System.out.println(sample);

    }
}
```

```java
package com.ravi.contrustor_ref;

@FunctionalInterface
interface A
{
    Test createObject();
}

class Test
{
    public Test()
    {
        System.out.println("No Argument constructor of Test class!!!");
    }
}


public class ConstructorReferenceDemo2
{
    public static void main(String[] args)
    {
        //By using Lambda
        A a1 =   () -> new Test();
        a1.createObject();

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

        //By using Constructor Reference
        A a2 = Test::new;
        a2.createObject();


    }
}
```

```java
package com.ravi.contrustor_ref;

import java.util.function.Function;

class Employee
{
    private String name;

    public Employee(String name)
    {
        super();
        this.name = name;
    }

    @Override
    public String toString()
    {
        return "Employee [name=" + name + "]";
    }
}

public class ConstructorRefDemo3
{
    public static void main(String[] args)
    {
        Function<String, Employee> f1 = Employee::new;
        Employee employee = f1.apply("Scott");
        System.out.println(employee);
    }
}
```

Note : If the parameterized constructor contains two parameters like Integer id, String name then we can use BiFunction<T,U,R>

How to create an array Object by using Constructor reference :

```java
package com.ravi.contrustor_ref;

import java.util.Arrays;
import java.util.Scanner;
import java.util.function.Function;

record Product(Integer id, String name, Double price)
{
}


public class ConstructorRefeDemo4
{
    public static void main(String[] args)
    {
        //Product []products = new Product[5];

        Scanner sc = new Scanner(System.in);

        Function<Integer, Product[]> fn2 = Product[]::new;

        System.out.println("Enter the size of the Array :");
        int size = Integer.parseInt(sc.nextLine());

        Product[] products = fn2.apply(size);

        //Initialization of Array
        for(int i=0; i<products.length; i++)
        {
            System.out.println("Enter Product Id :");
            int id = Integer.parseInt(sc.nextLine());

            System.out.println("Enter Product Name :");
            String name = sc.nextLine();

            System.out.println("Enter Product Price :");
            double price = Double.parseDouble(sc.nextLine());
            products[i] = new Product(id, name, price);
        }


        System.out.println(Arrays.toString(products));


    }

}
```

Arbitrary Reference type :
----
* In this Arbitrary Reference type we can directly call non static method with the help of class name. Actually Here Arbitrary means Object of any type which is automatically passed by JVM at runtime.

Example :

Lambda	Arbitrary Reference Type
str -> str.length();	String::length
(i1, i2) -> i1.compareTo(i2);	Integer::compareTo
(s1, s2) -> s1.compareTo(s2);	String::compareTo

Here internally JVM will pass this keyword to the method which referes the current Object.It does not refer to a particular object, but any instance of the class provided at runtime.

It will work with the interfaces which are working as a method parameter because this keyword is available as a method parameter.

Internally javac compiler converts method reference into Lambda.

It uses LambdaMetafactory, which dynamically generates a class at runtime implementing the target functional interface (e.g., Comparator<String>). It is from JDK 8V, Earlier we had Anonymous inner class

```java
package com.ravi.arbit_ref_type;

import java.util.function.Function;

public class ArbitraryRefDemo1 {

    public static void main(String[] args)
    {
        Function<String,String> fn1 = String::toUpperCase;
        System.out.println(fn1.apply("java"));
    }

}
```

```java
import java.util.TreeSet;

public class ArbitraryRefDemo2 {

    public static void main(String[] args)
    {
        TreeSet<String> ts = new TreeSet<>(String::compareTo);
        ts.add("C");
        ts.add("B");
        ts.add("A");

        System.out.println(ts);
    }

}
```

```java
package com.ravi.arbit_ref_type;

import java.util.Arrays;
import java.util.Collections;
import java.util.List;

public class ArbitraryRefDemo3
{
    public static void main(String[] args)
      {
        List<Integer> listOfNumbers = Arrays.asList(9,5,6,2,4,1);

        //By Using Lambda Expression
        Collections.sort(listOfNumbers, (Collections.reverseOrder()));
        System.out.println(listOfNumbers);

        //By using Method Reference
        Collections.sort(listOfNumbers, Integer::compareTo);
        System.out.println(listOfNumbers);


       //By Using Lambda Expression
       String [] players = {"Virat", "Rohit", "Zaheer", "Rishab", "Abhishek"};
       Arrays.sort(players,(s1, s2)-> s2.compareTo(s1));
       System.out.println(Arrays.toString(players));

       //By using Method Reference
       String [] players1 = {"Virat", "Rohit", "Zaheer", "Rishab", "Abhishek"};
       Arrays.sort(players1, String::compareTo);
       System.out.println(Arrays.toString(players1));

      }
}
```

```java
package com.ravi.arbit_ref_type;

@FunctionalInterface
interface MyFunction<T,U,V,R>
{
    R myApply(T t, U u, V v);
}

class Addition
{
    public Integer doSum(String x, String y)
    {
        return Integer.parseInt(x) + Integer.parseInt(y);
    }
}

public class ArbitRefDemo4
{
    public static void main(String[] args)
    {
     //Lambda
     MyFunction<Addition,String ,String, Integer> fn1 =
             (add, x, y) -> add.doSum(x, y);
     System.out.println(fn1.myApply(new Addition(), "100", "200"));


     //By Using Arbitrary Reference
     MyFunction<Addition,String ,String, Integer> fn2 = Addition::doSum;
     System.out.println(fn2.myApply(new Addition(), "500", "500"));

    }

}
```