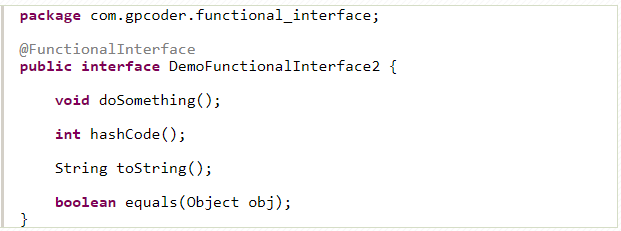
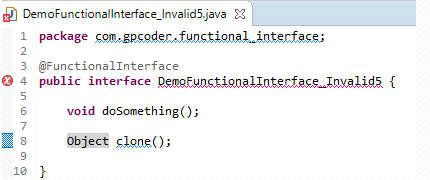
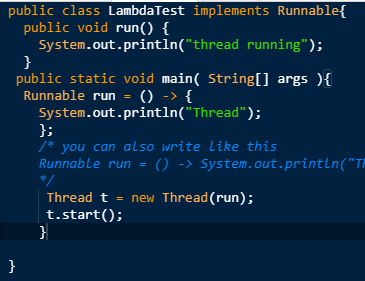
# Functional interface

* @Annotation: Functional Interface to express that this interface contains only 1 single abstract



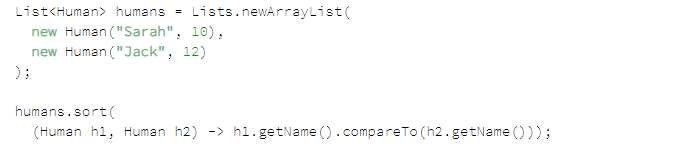
* All abstract methods of functional interface must be public



* Static/Default methods don’t break the rule.
* Functional Interface can only extend other functional interface when it doesn’t have any abstract methods
* Used with lambda

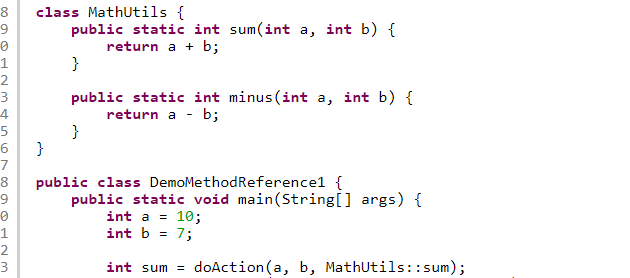
# Lambda expressions

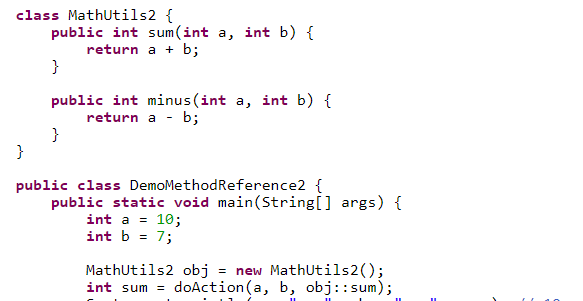
* Express instances of functional interface -> a function can be created without belonging to any classes.
* Decrease lines of code.
* Support Sequential and Parallel
* Mostly use with collections to sort collections instead of using Comparator



* Syntax: lambda operator -> body
* Zero parameter: () -> System.out.println("Zero parameter lambda");
* 1 parameter: (p) -> System.out.println("One parameter: " + p);
* Multiple parameters : (p1, p2) -> System.out.println("Multiple parameters: " + p1 + ", " + p2);
* arrL.forEach(n -> System.out.println(n));
* arrL.forEach(n -> { if (n%2 == 0) System.out.println(n); });

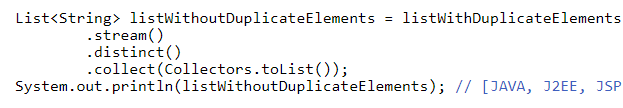
# Method Reference: Syntax: ContainingClass::(double colons)MethodName

* Reference to a static method 
* Reference to instance method



* Reference to constructor

# Stream

* Source -> Intermediate operations: filter(), limit(), skip(), map(), sorted()->Terminal operations: forEach(), collect(), anyMatch(), allMatch(), noneMatch(), count(), min(), max(), reduce(), summaryStatistics()
* Lazy evaluation: Stream has an important point that is: Using lazy evaluation to optimize the performance( don’t procedure until needed)
* Remove duplicate elements in array : 

# Optional

* Check Null :

+ : OrElse():getResource(resourceId).orElse(getHeavyResource()) => Always call getHeavyResource whether null or not.

+OrElseGet() => just call getHeavyResource when null

+ isPresent()

* Used in stream (map, filter) avoid Null Pointer Exception:

