**Lambda Expression** –

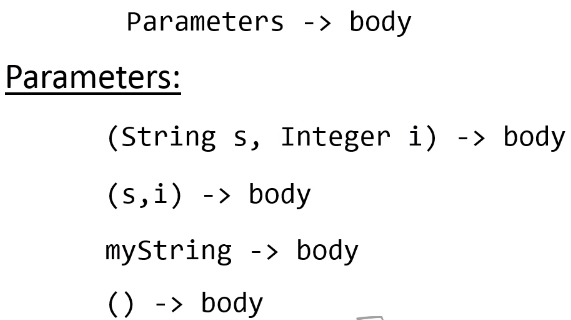
An anonymous code block or function without a name which can be passed inside another function and can be evaluated. Any interface in java 8 which has only one method can be used as a contract for lambda expression.

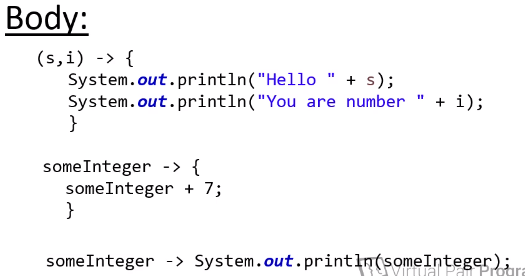
A lambda function in java is an object that implements a functional interface. This can be anonymous and can exist without any name or variable.

Functional interface – it is an interface which has single abstract method. As of java we can put static methods or default code blocks.

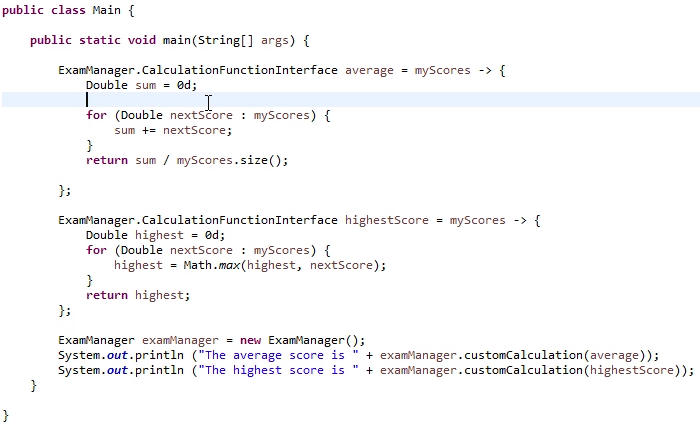
Syntax –

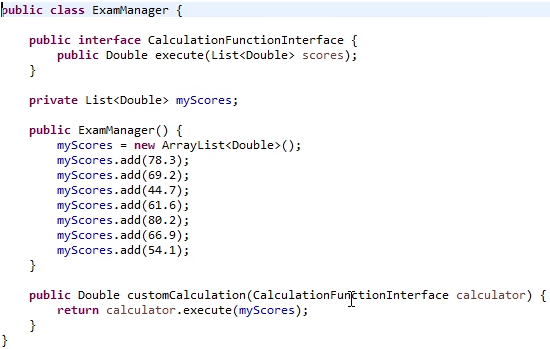
1. No need to provide variable type in parameters, it will match from the defined interface.
2. If only one parameter, no need to have parenthesis.
3. No parameters – empty parenthesis.
4. In body, no need to provide return statement, if only 1 statement.
5. In body with return type void and 1 statement, no need for braces.





Program example displays simple lambda expression demo. A class exam manager defines functional interface and the main class generates dynamically both the functionality of average and highest scores.

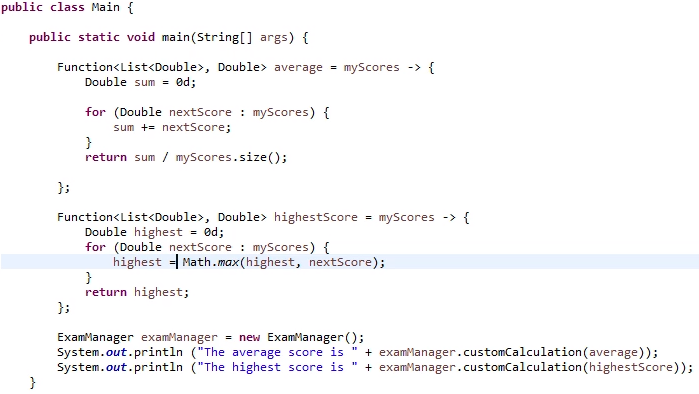


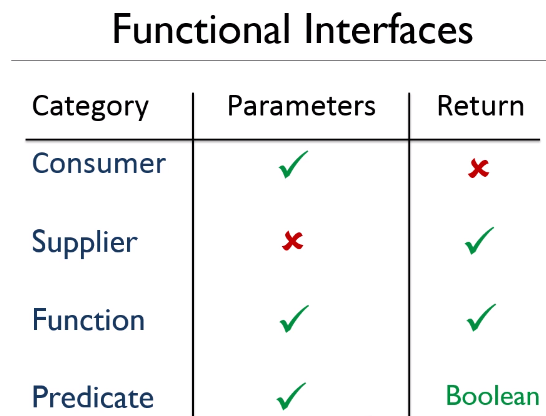


**Built in functional interfaces in java 8 –**

There are several built in functional interfaces, which can be directly used so we do not write our custom interfaces. These standard interfaces reside in java.lang.functions. So in the above example we can remove CalculationFunctionInterface and use a built in standard Function interface. *So we can use those built in interfaces as long as it matches our requirement.*







We can use a ToDoubleFunction built in functional interface also

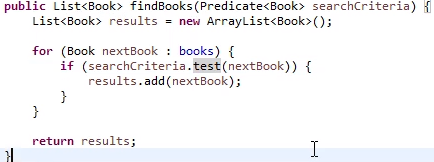


We can also use functional interfaces inline also. Below code have only one parameter and one statement so it is a shortcut way of defining. This is more usual way of writing functions.



Now we can use a Predicate which returns true or false so again a short cut

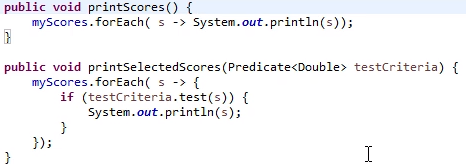




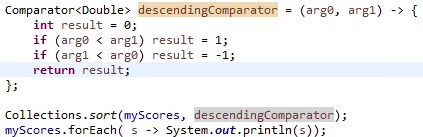
**Lambdas with collections –**

A new foreach loop has been introduced in java 8. Here foreach loop will need a **consumer** functional interface, which needs one parameter with void return type.



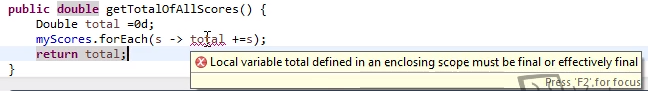


Collections also support java 8 functional interfaces, so there is no need to have extra classes to implement Comparator interface.

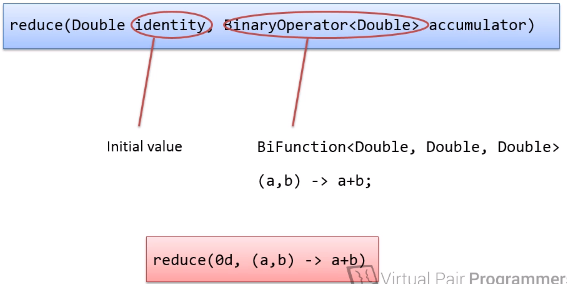


**Streams –**

It gives very powerful methods to manipulate collections. There are some limitations of iterating in collections in the scope of lambda expressions. Suppose a variable is defined outside of the scope of lambda expression, it cannot be changed inside the scope of lambda expression.



Now to solve this issue we will use a reduce function for the stream. The identity variable is the starting value which will be updated during the execution. Accumulator will be a BiFunction lambda expression which takes to parameters and returns a parameter.



Now to get back the list from the stream object, collect method is used. Also to change a value in the stream, use a **map** method which takes a Functional interface.

