Week 13 - Lambda expressions

This page is a draft, treat it accordingly,

Topics covered in this week

- syntax
- functional interfaces
- method references
- exception handling

Reading material

- https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html (examples, syntax, scope)
- https://docs.oracle.com/javase/tutorial/java/javaOO/methodreferences.html (method references)
- https://www.baeldung.com/java-8-functional-interfaces (functional interfaces, examples)
- https://www.baeldung.com/java-8-lambda-expressions-tips (best practices)
- https://www.baeldung.com/java-lambda-exceptions (exception handling)

Homework

Difficulty	Problem	Notes

EASY	Create a functional interface which combines four parameters into a single return value from the same generic data type.
	
	Using this functional interface create some lambda expression for the following:
	
	1. sum of integers
	2. product of integers
	3. concatenation of strings
	4. concatenation of strings with spaces between them
	
	Now create a method which could use your lambda expressions. This method should:
	
	• handle a generic array as input, checking that it has a minimum of 4 length
	• handle any instance of your functional interface
	• handle any instance of a lambda expression, which takes an argument and returns nothing - applying to the result of your functional interface instance. Hint: take a look for existing solution in java.util.function package.
	
	Write a program which calls this method for the array of Integers [1,2,3,4] with the 1. and 2. lambda expression you wrote, and for the array of Strings of ["one", two", "three", "four"] with the 3. and 4. lambda expression you wrote, all having the result printed to the console. E.g.
	
	`applyTransformations(new Integer[]{1,2,3,4}, summer, printer);`
	
	where applyTransformations is your method, the first parameter is the array of data, the second is your combiner functional interface instance, and the third parameter is the functional interface instance, which consumes the result of the combination.
		Example solution:
		
		`@FunctionalInterface`
		`interface Combiner<T> {`
		`    T combine(T arg1, T arg2, T arg3, T arg4);`
		`}`
		
		`public static <T> void applyCombination(T[] data, Combiner<T> combiner, java.util.function.Consumer<T> consumer) {`
		`    if (data.length >= 4) {`
		`        consumer.accept(combiner.combine(data[0], data[1], data[2], data[3]));`
		`    }`
		`}`

		`Combiner<Integer> summer = (first, second, third, fourth) -> first + second + third + fourth; Combiner<Integer> productor = (first, second, third, fourth) -> first * second * third * fourth; Combiner<String> concatenator = (first, second, third, fourth) -> first + second + third + fourth; Combiner<String> concatenatorWithSpaces = (first, second, third, fourth) -> first + " " + second + " " + third + " " + fourth;` `applyCombination(new Integer[]{1,2,3,4}, summer, System.out::println); applyCombination(new Integer[]{1,2,3,4}, productor, (res) -> System.out.println(res)); applyCombination(new String[]{"1","2","3","4"}, concatenator, System.out::println);` `applyCombination(new String[]{"1","2","3","4"}, concatenatorWithSpaces, (res) -> System.out.println(res));`
EASY		
EASY		