Lambda functions - exercises

Software Development Methods
Master in Data Science and Scientific Computing

1. Define the lambda function **const2** of type **Function<Integer, Integer>**, which is equivalent to the constant function $\overline{\mathbf{2}}$ of the FP system.

const2.apply(5)	**const2.apply(11)**
2	**2**

2. Define the lambda function **half** of type **Function<Integer, Integer>**, which returns an integer which corresponds to the half of the integer value that was passed as an argument.

half.apply(4)	**half.apply(11)**
2	**5**

3. Define the lambda predicate **isEven**, which takes a single **Integer** as argument and returns true or false to indicate if the value is even or not.

isEven.test(4)	**isEven.test(5)**
true	**false**

4. Define the lambda function **quarter** of type **Function<Integer, Integer>**, which returns an integer which corresponds to the quarter of the integer value that was passed as an argument. Define it by **composing** previously defined functions.

quarter.apply(12)	**quarter.apply(4)**
3	**1**

5. Define the functional interface **ArrayFunction** with the method **int apply(int [])**, which can be used to defined lambda functions that takes an array of integers as argument and returns an integer value.

6. Define the lambda function **size** of type **ArrayFunction**, which returns the length of the array passed as argument.

   ```
   int[] a = {1,-2,-4,3,7};
   System.out.println(size.apply(a));
   5
   ```

7. Define the lambda function **positive** of type **ArrayFunction**, which returns the number of positive integers in the array passed as argument.

   ```
   int[] a = {1,-2,-4,3,7};
   System.out.println(positive.apply(a));
   3
   ```

8. Define the static method **select** which returns an object of type **ArrayList<Integer>** and takes three arguments. The first argument called **list** is of type **ArrayList<Integer>**, the second called **pred** is a **Predicate<Integer>** and the third called **func** is a **Function<Integer, Integer>**. The method must return a new **ArrayList** with only the elements from the original **list** that fulfill the condition indicated by the predicate **pred**, modified by the function **func**.

```
ArrayList<Integer> list1 = new ArrayList<>(Arrays.asList(1,8,5,2,3,4,15,20));
ArrayList<Integer> list2 = select(list1, isEven, half);
for (Integer x : list2)
  System.out.println(x);
```

```
4
1
2
10
```

9. Analyze the following piece of code:

```
BiFunction<Integer, Integer, Integer> f1 = (x, y) -> x + y;
Function<Integer, Function<Integer, Integer>> f2 = x -> y -> f1.apply(x, y);
System.out.println(f1.apply(2, 3));
System.out.println(f2.apply(2).apply(3));
```

```
5
5
```

The example uses a concept called "currying". Please investigate it and prepare a small example of "currying" to be discussed next class.