LAMBDA/ STREAM

Exercises

MAP

1. Get the List of Joining Dates. Use Method References?
2. Given a list (gotten from the above exercise) of Joining dates, get the list of months.
3. Extract the lambdas from the above to make them reusable and Create a third lambda by composing the lambdas used in above 2 examples.
4. Use this Lambda to map List of Employees to List of Months!!!

FILTER

1. Get the list of Only Male Employees
2. Get the list of Employees those who joined in a particular Month (Say April)
3. Extract the lambdas in above 2 examples and compose a third lambda
4. Use this Lambda to get the Employees that are Female and have joined in the Month of June

SORT

1. Get a Sorted list of Employees based on their first name in Ascending order.
2. Get a Sorted list of Employees based on their first name in Descending order.
3. Get a Sorted list of Employees based on their last name
4. Extract the lambdas in above 2 examples and compose a third lambda
5. Get a Sorted list of Employees based on their first name and then on last name

COLLECTOR

1. Get the Unique List of Departments
2. Get Department wise Employees (Multi-Map)
3. Get Department wise and then "Month of Joining" wise Employees (Nested Multi-Map)

FLATMAP

1. Get All the movies (Unique List, no duplicates) in the gang
   "gang = MovieClub.president().getFriendCircle()"

MORE COLLECTORS

From all Movies in Gang (President's Friend Circle)

"gang = MovieClub.president().getFriendCircle()"

1. Get All the movies by Actor (MultiMap: <key=Actor, value=movie list>)
2. Get movies with each friend/person (MultiMap, key:Person, value=List of Movies)
3. Get Total Count of movies with each friend (Map, key:Person, value=Integer)
4. Get Total Cost of movies with each friend (Map, key:Person, value=Integer)
5. Get the Movies with Each friend and each actor, Nested Multimap(Outer key:person, inner key Actor, value=Movie)

CUSTOM COLLECTORS

1. Get the map of Movie vs. List of people, to find out who all possess a particular movie in a friend Circle (Map: <key: Movie, value:<list of Person>>). Solve the problem by using the standard Collectors in Java 8
2. Now write a custom collector that does the above

LAMBDA EXERCISES

1. Write an interface named "Operation", and make it SAM (Single abstract method) interface by declaring only one abstract method, say "int operate(int x, int y)"
2. For using the Lambda create a class called "Calculator" with a constructor that takes 2 parameter (operands),
3. Declare a concrete method "int calculate(Operation op)"
4. Implement the method by delegating the calculation logic to Operation and return the result.
5. Now you are done with developing your library
6. As a user of your own library (client code), pass Lambda expressions to calculate the sum, difference, product, and division to the calculate method
7. TIP: This was just for exercise, every time you don't have to write your own SAM interface, Java provide standard interface in java.util.function package. The above could be achieved by using BiFunction<Integer,Integer, Integer> or more optimally by IntBinaryOperator (if we are sure that the parameters are going to be primitive and don't want to pay the cost of boxing/unboxing)