Evgen959 / Advanced_Backend Public

Code Issues Pull requests Actions Projects Security Insights

master Advanced_Backend / konsul / les045 / src / Main.java

Go to file

Evgen959 newKonsulLes45 65e4e1e · 1 minute ago

118 lines (99 loc) · 4.39 KB

Code Blame Raw

```java
/* 1. Дан список Person (String firstName, String lastName, int age).
Необходимо используя стрим реализовать метод, который вернет список String,
где в алфавитном порядке будут перечислены все  ФИО людей (в виде Иванов И.) старше 18 лет.*/



import java.util.*;
import java.util.function.Function;
import java.util.function.Predicate;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class Main {


    static class  AccWithPerson{
        private String account;
        private PersonWithAccounts person;

        public AccWithPerson(String account, PersonWithAccounts person) {
            this.account = account;
            this.person = person;
        }

        public String getAccount() {
            return account;
        }

        public PersonWithAccounts getPerson() {
            return person;
        }
    }
    public static void main(String[] args) {
        List<Person> people = List.of(
                new Person("Jack", 15),
                new Person("Leon", 18),
                new Person("Ann", 13),
                new Person("Nike", 22),
                new Person("Mike", 29),
                new Person("John", 24)
        );
        List<String> list = listHandler(
                people,
                person -> person.getAge() > 13,
                person -> getStringName(person),
                (n1, n2) -> n1.compareTo(n2)
        );

        System.out.println(list);
        //[Jack J., John J., Leon L., Mike M., Nike N.]
    /*
        Stream<Person> streamPerson = people.stream().filter(p -> p.getName().startsWith("J"));

        List<Person> list1 = streamPerson.sorted().toList();
        streamPerson.forEach(p-> System.out.println(p)); /// !!!!!Error Do not use finalized Stream
    */
        Set<Person> collect = listPersonHandler(people).collect(Collectors.toSet());
        System.out.println(collect);
        /*[Person{name='Ann', age=13}, Person{name='Leon', age=18}, Person{name='John', age=24},
        Person{name='Jack', age=15}, Person{name='Nike', age=22}, Person{name='Mike', age=29}]*/

        Map<String, Integer> collect1 = listPersonHandler(people)
                                           .collect(Collectors
                                           .toMap(p -> p.getName(), p -> p.getAge()));
        System.out.println(collect1);
        // {Ann=13, Nike=22, Mike=29, John=24, Jack=15, Leon=18}

        List<PersonWithAccounts> peopleWihAcc = List.of(
                new PersonWithAccounts("Jack", 15, List.of("1","4")),
                new PersonWithAccounts("Leon", 18, List.of("2")),
                new PersonWithAccounts("Ann", 13, List.of("3")),
                new PersonWithAccounts("Nike", 22, List.of("5","6")),
                new PersonWithAccounts("Mike", 29, List.of("6","7","8")),
                new PersonWithAccounts("John", 24, List.of("9"))
        );
        peopleWihAcc.stream()
                .flatMap(pwa->pwa.getAccounts().stream().map(s->new AccWithPerson(s,pwa)))
                .filter(s-> s.getAccount().compareTo("6")>0)
                .collect(Collectors.toMap(acc->acc.account, acc->acc.getPerson().getName()));
    }

    private static String getStringName(Person person) {
        StringBuilder sb = new StringBuilder(person.getName());
        return sb.append(" ")
                .append(person.getName().charAt(0))
                .append(".")
                .toString();
    }

    /*
    public static <T,R> List<R> listHandler(List<T>  list, Predicate<T> predicate, Function<T,R> function){
        List<R> resultList = new ArrayList<>();
        for (T element: list){
            if(predicate.test(element)){
                resultList.add(function.apply(element));
            }
        }
        return resultList;

    }*/

    public static <T,R> List<R> listHandler(List<T>  list,
                                           Predicate<T> predicate,
                                           Function<T,R> function,
                                           Comparator<R> comparator
    ){
```

```java
            return list.stream()
                    .filter(predicate)
                    .map(function)
                    .sorted()
                    .collect(Collectors.toList());

        }

        public static Stream<Person> listPersonHandler(List<Person> list){
            return list.stream().filter(p->p!=null);
        }
    }
```