1. Converting a Stream of Strings to a Stream of Integers

List<String> listOfStrings = Arrays.asList("1", "2", "3", "4", "5");

List<Integer> listOfIntegers = listOfStrings.stream()

//.map(Integer::valueOf)  
 .map(n->Integer.valueOf(n))

.collect(Collectors.toList());

System.out.println(listOfIntegers);

1. Finding all distinct salaries among all employees  
   class Employee{  
    private int id;  
    private String name;

private double salary;  
 // getter, setter etc..  
}  
List<Employee> employeesList = Arrays.asList(

new Employee(1, "Alex", 100),

new Employee(2, "Brian", 100),

new Employee(3, "Charles", 200),

new Employee(4, "David", 200),

new Employee(5, "Edward", 300),

new Employee(6, "Frank", 300)

);

List<Double> distinctSalaries = employeesList.stream()

.map( e -> e.getSalary() )

.distinct()

.collect(Collectors.toList());

System.out.println(distinctSalaries);

1. Stream of Integers, Longs or Doubles  
   List<String> scoresAsString = Arrays.asList("85", "92", "78", "90", "88");

double averageScore = scoresAsString.stream()

.mapToInt(Integer::parseInt) // Convert strings to integers using mapToInt()

.average() // Calculate the average of the integers

.orElse(0.0); // Use 0.0 if there are no scores

System.out.println("Average score: " + averageScore);

1. Convert a Stream into a Map in Java

**Method 1: Using Collectors.toMap() Function**

The Collectors.toMap() method takes two parameters as the input:

KeyMapper: This function is used for extracting keys of the Map from stream value.

ValueMapper: This function used for extracting the values of the map for the given key.

4.1. Example 1

String input = "Keep learning and have fun";

// convert a string into a Map with the keys as the words of the string and the value as the  
// length of each word.

Map<String, Integer> lengthMap

= Arrays.stream(input.split(" "))

.collect(Collectors.toMap(

value -> value,

value -> value.length())

);

4.2. Example 2

public class User {

private int userId;

private String name;

private String city;  
// getter, setter etc..  
}

User user1 = new User(1, "Shalini", "Pune");

User user2 = new User(2, "Siya", "Mumbai");

User user3 = new User(3, "Asha", "Nagpur");

//Convert a list of users into a map where UserId is the key and the User is the value.

Map<Integer, User> userMap = Arrays.asList(user1, user2, user3)

.stream()

.collect(Collectors.toMap(

user -> user.getUserId(),

user -> user));

**Method 2: Using Collectors.toMap() Function**

The groupingBy collector takes one function as input and creates a group of stream objects using that function.

User user4 = new User(4, "Akhil", "Pune");

User user5 = new User(5, "Nikhil", "Mumbai");

//convert a user stream into a map whose key is the city and the value is the users living in   
// that city.

Map<String, List<User> > cityUserListMap

= Arrays.asList(user1, user2, user3, user4, user5)

.stream()

.collect(Collectors.groupingBy(

User::getCity));

// get the count of the users belonging to each city.

Map<String, Long>

cityUserCountMap

= Arrays.asList(user1, user2, user3,

user4, user5)

.stream()

.collect(

Collectors.groupingBy(

User::getCity,

Collectors.counting()));

1. Sorting the collection  
   5.1. Sort users with city  
   List<User> sorteduserwithcity =users.stream()

.sorted(Comparator.comparing(User::getCity))

.collect(Collectors.toList());

for(User user:sorteduserwithcity)

System.out.println(user);

5.1. Sort users with city and name  
List<User> sorteduserwithcityandname =users.stream()

.sorted(

Comparator.comparing(User::getCity)

.thenComparing(user -> user.getName()))

.collect(Collectors.toList());

for(User user:sorteduserwithcityandname)

System.out.println(user);

## Create class Person as follows: import java.util.List; import java.util.ArrayList; import java.time.chrono.IsoChronology; import java.time.LocalDate;

public class Person {

public enum Sex {

MALE, FEMALE

}

String name;

LocalDate birthday;

Sex gender;

String emailAddress;

Person(String nameArg, LocalDate birthdayArg,

Sex genderArg, String emailArg) {

name = nameArg;

birthday = birthdayArg;

gender = genderArg;

emailAddress = emailArg;

}

public int getAge() {

return birthday

.until(IsoChronology.INSTANCE.dateNow())

.getYears();

}

public void printPerson() {

System.out.println(name + ", " + this.getAge());

}

public Sex getGender() {

return gender;

}

public String getName() {

return name;

}

public String getEmailAddress() {

return emailAddress;

}

public LocalDate getBirthday() {

return birthday;

}

public static int compareByAge(Person a, Person b) {

return a.birthday.compareTo(b.birthday);

}

public static List<Person> createRoster() {

List<Person> roster = new ArrayList<>();

roster.add(

new Person(

"Fred",

IsoChronology.INSTANCE.date(1980, 6, 20),

Person.Sex.MALE,

"fred@example.com"));

roster.add(

new Person(

"Jane",

IsoChronology.INSTANCE.date(1990, 7, 15),

Person.Sex.FEMALE, "jane@example.com"));

roster.add(

new Person(

"George",

IsoChronology.INSTANCE.date(1991, 8, 13),

Person.Sex.MALE, "george@example.com"));

roster.add(

new Person(

"Bob",

IsoChronology.INSTANCE.date(2000, 9, 12),

Person.Sex.MALE, "bob@example.com"));

return roster;

}

}

1. Create Test class to implement the following functionalities:

List<Person> roster = Person.createRoster();  
System.out.println("Contents of roster:");

* 1. Average age of male members  
     double average = roster.stream()  
      .filter(p -> p.getGender() == Person.Sex.MALE)  
      .mapToInt(Person::getAge)  
      .average()  
      .getAsDouble();

System.out.println("Average age (bulk data operations): " + average);

* 1. Sum of ages with sum operation  
     Integer totalAge = roster

.stream()

.mapToInt(Person::getAge)

.sum();

System.out.println("Sum of ages (sum operation): " +

totalAge);

// 3. Sum of ages with reduce(identity, accumulator)

Integer totalAgeReduce = roster

.stream()

.map(Person::getAge)

.reduce(

0,

(a, b) -> a + b);

System.out.println(

"Sum of ages with reduce(identity, accumulator): " +

totalAgeReduce);

// 4. Average of male members with collect operation

/\* Averager averageCollect = roster.stream()

.filter(p -> p.getGender() == Person.Sex.MALE)

.map(Person::getAge)

.collect(Averager::new, Averager::accept, Averager::combine);

System.out.println("Average age of male members: " +

averageCollect.average());\*/

// 5. Names of male members with collect operation

System.out.println("Names of male members with collect operation: ");

List<String> namesOfMaleMembersCollect = roster

.stream()

.filter(p -> p.getGender() == Person.Sex.MALE)

.map(p -> p.getName())

.collect(Collectors.toList());

namesOfMaleMembersCollect

.stream()

.forEach(p -> System.out.println(p));

// 6. Group members by gender

System.out.println("Members by gender:");

Map<Person.Sex, List<Person>> byGender =

roster

.stream()

.collect(

Collectors.groupingBy(Person::getGender));

List<Map.Entry<Person.Sex, List<Person>>>

byGenderList =

new ArrayList<>(byGender.entrySet());

byGenderList

.stream()

.forEach(e -> {

System.out.println("Gender: " + e.getKey());

e.getValue()

.stream()

.map(Person::getName)

.forEach(f -> System.out.println(f)); });

// 7. Group names by gender

System.out.println("Names by gender:");

Map<Person.Sex, List<String>> namesByGender =

roster

.stream()

.collect(

Collectors.groupingBy(

Person::getGender,

Collectors.mapping(

Person::getName,

Collectors.toList())));

List<Map.Entry<Person.Sex, List<String>>>

namesByGenderList =

new ArrayList<>(namesByGender.entrySet());

namesByGenderList

.stream()

.forEach(e -> {

System.out.println("Gender: " + e.getKey());

e.getValue()

.stream()

.forEach(f -> System.out.println(f)); });

// 8. Total age by gender

System.out.println("Total age by gender:");

Map<Person.Sex, Integer> totalAgeByGender =

roster

.stream()

.collect(

Collectors.groupingBy(

Person::getGender,

Collectors.reducing(

0,

Person::getAge,

Integer::sum)));

List<Map.Entry<Person.Sex, Integer>>

totalAgeByGenderList =

new ArrayList<>(totalAgeByGender.entrySet());

totalAgeByGenderList

.stream()

.forEach(e ->

System.out.println("Gender: " + e.getKey() +

", Total Age: " + e.getValue()));

// 9. Average age by gender

System.out.println("Average age by gender:");

Map<Person.Sex, Double> averageAgeByGender =

roster

.stream()

.collect(

Collectors.groupingBy(

Person::getGender,

Collectors.averagingInt(Person::getAge)));

for (Map.Entry<Person.Sex, Double> e : averageAgeByGender.entrySet()) {

System.out.println(e.getKey() + ": " + e.getValue());

}

}

}