timer

**57**

timProgram 2:er

**46**

**Program 2:**

**DESCRIPTION**

Your task here is to implement a **Java** code based on the following specifications. Note that your code should match the specifications in a precise manner. Consider default visibility of classes, data fields, and methods unless mentioned otherwise.

**Specifications:**

class Student

data members:

name: String

graduationYear: int

score: int

visibility: public

Student( String name, int graduationYear, int score): constructor **with** **public** visibility

**Define** getters **with** **public** visibility

toString() method has been implemented **for** you

**class** Implementation:

method definitions:

studentsSorted(Collection<Student> students):

**return** **type**: **List**<Student>

visibility: **public**

studentsByYear(Collection<Student> students):

**return** **type**: **Map**<Integer, **List**<Student>>

visibility: **public**

findOneBest(Collection<Student> students):

**return** **type**: Student

visibility: **public**

**Task:**

**class Student:**

**-**define **data member**saccording to the above specifications

**-**define a **constructor** and **getters** according to the above specifications

-**toString()** method has been implemented for you as a part of the code stub

**class Implementation:**

**Implement the below methods for this class using in Stream API:**

* **List<Student> studentsSorted(Collection<Student> students):**Sort students by decreasing results. In case of identical result, sort alphabetically.
* **Map<Integer, List<Student>> studentsByYear(Collection<Student> students):**Group students by graduation years. (Key is the year and value is the list of students of that year.)
* **Student findOneBest(Collection<Student> students):**Find one student that has obtained the highest score.

**Refer to sample output for more details**

**Sample Input**

Collection<Student> students;

            students = Arrays.asList(

**new** Student("S2014-11", 2014, 17),

**new** Student("S2014-18", 2014, 20),

**new** Student("S2013-18", 2013, 20)

            );

---------------------------------------------------------

studentsSorted(students);

studentsByYear(students);

findOneBest(students);

**Sample Output**

[Student{name='S2013-18', graduationYear=2013, score=20}, Student{name='S2014-18', graduationYear=2014, score=20}, Student{name='S2014-11', graduationYear=2014, score=17}]

----------------------------------------------------------

{2013=[Student{name='S2013-18', graduationYear=2013, score=20}], 2014=[Student{name='S2014-11', graduationYear=2014, score=17}, Student{name='S2014-18', graduationYear=2014, score=20}]}

----------------------------------------------------------

Student{name='S2014-18', graduationYear=2014, score=20}



import java.util.Arrays;

import java.util.Collection;

import java.util.List;

import java.util.Map;

import static java.util.Comparator.comparing;

import static java.util.stream.Collectors.groupingBy;

import static java.util.stream.Collectors.toList;

import static java.util.Comparator.comparingInt;

class Student {

public String name;

public int graduationYear;

public int score;

public Student(String name,int graduationYear,int score)

{

this.name=name;

this.graduationYear=graduationYear;

this.score=score;

}

public void setName(String name)

{

this.name=name;

}

public String getName()

{

return name;

}

public void setGraduationYear(int graduationYear)

{

this.graduationYear=graduationYear;

}

public int getGraduationYear()

{

return graduationYear;

}

public void setScore(int score)

{

this.score=score;

}

public String getScore()

{

return score;

}

@Override

public String toString() {

return "Student{" +

"name='" + name + '\'' +

", graduationYear=" + graduationYear +

", score=" + score +

'}';

}

}

class Implementation{

public List<Student> studentsSorted(Collection<Student> students)

{

}

public Map<Integer,List<Student>> studentsByYear(Collection<Student> students)

{

}

public Student findOneBest(Collection<Student> students)

{

}

public class Source {

public static void main(String args[] ) throws Exception {

/\* Enter your code here. Read input from STDIN. Print output to STDOUT \*/

}

}

Problem 3:

timer

**52**

**30**

**TIME LEFT IN THIS ASSIGNMENT SESSION**

1/1 Question attempted

navigate\_beforePREV[**1**](https://api.doselect.com/test/938vl/problem/9e54e5)NEXTnavigate\_next

### Author Of Books

[bookmark\_border](https://api.doselect.com/)

* subject Coding
* casino 100 points

**DESCRIPTION**

Your task here is to implement a **Java** code based on the following specifications. Note that your code should match the specifications in a precise manner. Consider default visibility of classes, data fields, and methods unless mentioned otherwise.

**Specifications:**

class Author

data members:

firstName: String

lastName: String

Define a constructor **with** **public** visibility

**Define** getter setters **with** **public** visibility

toString() method has been implemented **for** you

**class** Book

**data** members:

title: **String**

Author author

**Define** a **constructor** **with** **public** visibility

**Define** getter setters **with** **public** visibility

toString() method has been implemented **for** you

**class** BookImplementation:

method definition:

firstNameOfAuthors(Author... **authors**):

**return** **type**: **List**<**String**>

visibility: **public**

titlesOf(Book... books):

**return** **type**: **List**<**String**>

visibility: **public**

**Task:**

**class Author:**

**-**define the **String** variable **firstName**

**-**define the **String** variable **lastName**

**-**define a **constructor** and **getter setters** according to the above specifications

-**toString()** method has been implemented for you as a part of the code stub

**class Book:**

**-**define the **String** variable **title**

**-**define a **constructor** and **getter setters** according to the above specifications

-**toString()** method has been implemented for you as a part of the code stub

class **BookImplementation:**

Implement the below method for this class using in **Stream API:**

* **List<String> firstNameOfAuthors(Author... authors):**

fetch the first name of Author from a list of authors and return it

* **List<String> titlesOf(Book... books):**

return the titles from a list of books.

Problem 4:

timer

**01**

**19**

**22**

**TIME LEFT IN THIS ASSIGNMENT SESSION**

0/1 Question attempted

navigate\_beforePREV[**1**](https://api.doselect.com/test/o9a18/problem/l9xna)NEXTnavigate\_next

### Car Story

[bookmark\_border](https://api.doselect.com/)

* subject Coding
* casino 100 points

**DESCRIPTION**

This is a short exercise in using the stream API.

Your task here is to implement a Java code based on the following specifications. Note that your code should match the specifications in a precise manner. Consider default visibility of classes, data fields and methods unless mentioned otherwise.

**Specifications:**

**class** **definitions**:

**class** **Car**:

      data fields:

**private** name: String variable

**private** carName: String Variable

**private** price: **double** variable

      method definitons:

          Car(String pName, String cName, **double** p): Constructor to initialize the **class**

**Getter** **methods**:

              getName()

              getCarName()

              getPrice()

             Setter methods:

              setName()

            setCarName()

             setPrice()

**class** **CarImplementation**:

    method definitions:

        sumOfPrices(List<Car> carList): to add the prices **and** **return** **using** forEach.

**return** type: **double**

            visibilty: **public**

**return**: totalPrice (sum of prices)

        printName(List<Car> carList) : to filter data of list

**return** type: List<String>

            visibility: **public**

**return**: carNameList(name of cars above 25000)

        maxPrice(List<Car> carList): to get maximum car price

**return** type: **double**

            visibility: **public**

**return**: maxPrice (maximum price of a car)

You don't need to implement the main() method. It has already been implemented as a part of the test-cases. It contains an **ArrayList** containing the **name**, **carName** and **price** of the car that will be used to create streams.

**Task:**

Create a Car class which has the following private members:

name String

carName String

price double

* Define parameterized constructor.
* Define getter method for all instance variables.(getName(), getcName(),....)
* Define setter methods for all instance variables.(setName(), setcName(),....)

Create a **CarImplementation** class which performs operations (as per the given requirements) in using Stream Api:

* Use map() and reduce() methods to add the prices of all the cars of the ArrayList.
* Create a List<String> that contains the name of cars which have a price greater than 25000. Use **filter()** and **forEach()** methods for the same.
* Fetch that car price which is maximum of all the cars

Implement using Lambda expressions.

**NOTE**

* Do not use any **for** loops or other control structures.
* Use the stream API methods for your implementations, else the test-cases might fail.
* You don’t need to write the main() function.

**Sample Output**

The given output will be obtained upon successful execution against the Sample test cases.

3375000.0

[Alfa Romeo, Bugatti, Chrysler, Dodge, Essen]

910000.0

import org.w3c.dom.NameList;

import java.util.ArrayList;

import java.util.List;

import java.util.Map;

import java.util.stream.Collectors;

class Car {

}

class CarImplementation {

// Implement the CarImplementation method...

}

Example 1:strange operations:

class People

data members:

name: String

age: Integer

gender: Gender

visibility: private

enum Gender:

  MAN,

        WOMEN

visibility: public

Define a constructor with public visibility

Define getter setters with public visibility

class PeopleImplementation:

method definition:

minimumAgeContainingE(Collection<People> peoples):

return type: Integer

visibility: public

getAgeOfOldestMan(Collection<People> peoples):

return type: Integer

visibility: public

**Task:**

**class People:**

**-**define **data member**s and **enum** **Gender**according to the above specifications

**-**define a **constructor** and **getter setters** according to the above specifications

**class PeopleImplementation:**

**Implement the below method for this class using in Stream API:**

* **﻿Integer minimumAgeContainingE(Collection<People> peoples):**Find and return the **minimum age**of a person who contains **"e"** in the name
* **Integer  getAgeOfOldestMan(Collection<People> peoples):** Find and return the **age**of the Oldest Man

**Refer to sample output for more details**

**Sample Input**

Collection<People> peoples = Arrays.asList(

**new** People("Vivek", 16, People.Gender.MAN),

**new** People("Kayle", 23, People.Gender.WOMEN),

**new** People("Jeremy", 42, People.Gender.MAN),

**new** People("Ivan", 69, People.Gender.MAN)

);

------------------------------------------------

peoples //Input for both the methods

**Sample Output**

16

----------------------------------------------------------------

69

# **NOTE**

You can make suitable function calls and use **the RUN CODE** button to check your **main()** method output.

**EXECUTION TIME LIMIT**

10 seconds

**Solution code**Please choose a language and write your code.

done\_allSUBMIT

info\_outlineUNSOLVED

CODEINPUTOUTPUT

CODEINPUTOUTPUT

**Java 8**

play\_circle\_filledRUN CODEplaylist\_add\_checkVERIFYmore\_vert



1

2

3

4

5

import java.io.\*;

import java.util.\*;

import java.text.\*;

import java.math.\*;

import java.util.regex.\*;

class People {

private String name;

private Integer age;

private Gender gender;

public enum Gender{MAN,WOMEN};

public People(String name,Integer age,Gender gender)

{

this.name=name;

this.age=age;

this.gender=gender;

}

public void setName(String name)

{

this.name=name;

}

public String getName()

{

return name;

}

public void setAge(Integer age)

{

this.age=age;

}

public Integer getAge()

{

return age;

}

public void setGender(Gender gender)

{

this.gender=gender;

}

public Gender getGender()

{

return gender;

}

}

class PeopleImplementation {

public Integer minimumAgeContainingE(Collection<People> people)

{

}

public Integer getAgeOfOldestMan(Collection<People> people)

{

}

}

public class Source {

public static void main(String args[] ) throws Exception {

);

}

}

Problem 5:

### Employee Verification

**DESCRIPTION**

Your task here is to implement a Java code based on the following specifications. Note that your code should match the specifications in a precise manner. Consider default visibility of classes, data fields and methods unless mentioned otherwise.

**Specifications:**

**class** **definitions**:

**class** **Employee**:

    data fields:

      name: String

      salary: **int**

    Implement a Constructor **using** the **class** **variables**.

**Implement** **getter** **setter** **methods** **for** **the** **same**.

**class** **EmployeeInfo**:

    method definitions:

      SortMethod :

**return** type: **enum**

          named Constants: BYNAME, BYSALARY

      sort(List<Employee> emps, final SortMethod method): Method to **return** sorted list by name **and** by salary **using** SortMethod

           Return type: List<Employee>

           Visibility: **public**

      getThirdInCollection(Collection<Employee> entities): method to **return** third element in list **using** skip method

           Return type: Employee

           Visibility: **public**

      isCharacterPresentInAllNames(Collection<Employee> entities, String character): method to check **if** Employee list contains a specific character out

           Return type: boolean

           Visibility: **public**

**Task:**

Create an Employee class which has the following members:

**String** name;

**int** salary;

* Define parameterized **constructor**.
* Define **getter** method for all instance variables.(getName(),...)
* Define **setter** methods for all instance variables.(setName(),....)

Create an **EmployeeInfo** class which performs following operations (as per the given requirements) using **StreamAPI**:

* **enum SortMethod** : representing a group of named constants **BYNAME** and **BYSALARY**
* **sort(List<Employee> emps, final SortMethod method)**: Method to return sorted list by **name** and by **salary** using **SortMethod**
* **getThirdInCollection(Collection<Employee> entities)**: Method to return **third element**in list using **skip** method
* **isCharacterPresentInAllNames(Collection<Employee> entities, String character)**: Method to check if Employee list contains a**specific character** out

Implement using **Lambda expressions**.

Following has been done for you:

* **Main()** method containing list of **Employees**
* **String toString()** method, it's part of code stub, don't edit it else your**test-cases might fail**

**Sample Input**

List<Employee> emps = **new** ArrayList<>();

emps.**add**(**new** Employee("Mickey", 100000));

emps.**add**(**new** Employee("Timmy", 50000));

emps.**add**(**new** Employee("Annny", 40000));

**Sample Output**

[<**name:** Annny salary: 40000>, <**name:** Mickey salary: 100000>, <**name:** Timmy salary: 50000>]

[<**name:** Annny salary: 40000>, <**name:** Timmy salary: 50000>, <**name:** Mickey salary: 100000>]

<**name:** Mickey salary: 100000>

false

**NOTE**

* Do not use any **for** loops or other control structures.
* Use the stream API methods for your implementations, else the test-cases might fail.
* You CAN implement the **main()** method to check the implementation of your methods in the solution.
* Upon implementation of **main()** method, you can use the **RUN CODE** button to pass input data in the method calls and arrive at some output.

**EXECUTION TIME LIMIT**

10 seconds

**Solution code**Please choose a language and write your code.

done\_allSUBMIT

createATTEMPTED

CODEINPUTOUTPUT

CODEINPUTOUTPUT

**Java 8**

play\_circle\_filledRUN CODEplaylist\_add\_checkVERIFYmore\_vert



1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

import java.util.\*;

import java.util.function.\*;

import java.util.stream.Stream;

class Employee {

String name;

int salary

public Employee(Stri)

public String toString() {

StringBuilder sb = new StringBuilder("<");

sb.append("name: ");

sb.append(name);

sb.append(" salary: ");

sb.append("" + salary+">");

return sb.toString();

}

}

class EmployeeInfo{

}

timer

**01**

**10**

**59**

**TIME LEFT IN THIS ASSIGNMENT SESSION**

1/1 Question attempted

navigate\_beforePREV[**1**](https://api.doselect.com/test/646po/problem/n65qo)NEXTnavigate\_next

### Employee Verification

[bookmark\_border](https://api.doselect.com/)

* subject Coding
* casino 100 points

**DESCRIPTION**

Your task here is to implement a Java code based on the following specifications. Note that your code should match the specifications in a precise manner. Consider default visibility of classes, data fields and methods unless mentioned otherwise.

**Specifications:**

**class** **definitions**:

**class** **Employee**:

    data fields:

      name: String

      salary: **int**

    Implement a Constructor **using** the **class** **variables**.

**Implement** **getter** **setter** **methods** **for** **the** **same**.

**class** **EmployeeInfo**:

    method definitions:

      SortMethod :

**return** type: **enum**

          named Constants: BYNAME, BYSALARY

      sort(List<Employee> emps, final SortMethod method): Method to **return** sorted list by name **and** by salary **using** SortMethod

           Return type: List<Employee>

           Visibility: **public**

      getThirdInCollection(Collection<Employee> entities): method to **return** third element in list **using** skip method

           Return type: Employee

           Visibility: **public**

      isCharacterPresentInAllNames(Collection<Employee> entities, String character): method to check **if** Employee list contains a specific character out

           Return type: boolean

           Visibility: **public**

**Task:**

Create an Employee class which has the following members:

**String** name;

**int** salary;

* Define parameterized **constructor**.
* Define **getter** method for all instance variables.(getName(),...)
* Define **setter** methods for all instance variables.(setName(),....)

Create an **EmployeeInfo** class which performs following operations (as per the given requirements) using **StreamAPI**:

* **enum SortMethod** : representing a group of named constants **BYNAME** and **BYSALARY**
* **sort(List<Employee> emps, final SortMethod method)**: Method to return sorted list by **name** and by **salary** using **SortMethod**
* **getThirdInCollection(Collection<Employee> entities)**: Method to return **third element**in list using **skip** method
* **isCharacterPresentInAllNames(Collection<Employee> entities, String character)**: Method to check if Employee list contains a**specific character** out

Implement using **Lambda expressions**.

Following has been done for you:

* **Main()** method containing list of **Employees**
* **String toString()** method, it's part of code stub, don't edit it else your**test-cases might fail**

**Sample Input**

List<Employee> emps = **new** ArrayList<>();

emps.**add**(**new** Employee("Mickey", 100000));

emps.**add**(**new** Employee("Timmy", 50000));

emps.**add**(**new** Employee("Annny", 40000));

**Sample Output**

[<**name:** Annny salary: 40000>, <**name:** Mickey salary: 100000>, <**name:** Timmy salary: 50000>]

[<**name:** Annny salary: 40000>, <**name:** Timmy salary: 50000>, <**name:** Mickey salary: 100000>]

<**name:** Mickey salary: 100000>

false

**NOTE**

* Do not use any **for** loops or other control structures.
* Use the stream API methods for your implementations, else the test-cases might fail.
* You CAN implement the **main()** method to check the implementation of your methods in the solution.
* Upon implementation of **main()** method, you can use the **RUN CODE** button to pass input data in the method calls and arrive at some output.

**EXECUTION TIME LIMIT**

10 seconds

import java.util.\*;

import java.util.function.\*;

import java.util.stream.Stream;

class Employee {

String name;

int salary

public Employee(String name,int salary)

{

this.name=name;

this.salary=salary;

}

enum sortMethod

{

BYNAME,BYSALARY;

}

public String toString() {

StringBuilder sb = new StringBuilder("<");

sb.append("name: ");

sb.append(name);

sb.append(" salary: ");

sb.append("" + salary+">");

return sb.toString();

}

}

class EmployeeInfo{

}