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 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

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 void setAge(int age)

{

this.age=age;

}

public void setGender(Gender gender)

{

this.gender=gender;

}

public int getAge()

{

return this.age;

}

public String getName()

{

return this.name;

}

public Gender getGender()

{

return this.gender;

}

}

class PeopleImplementation

{

public Integer minimumAgeContainingE(Collection <People> peoples)

{

Integer temp=-1;

for(People i:peoples)

{

if(i.getName().contains("e") || i.getName().contains("E"))

{

temp=i.getAge();

}

}

for(People i:peoples)

{

if(i.getName().contains("e") || i.getName().contains("E"))

{

if(temp>i.getAge())

{

temp=i.getAge();

}

}

}

return temp;

}

public Integer getAgeOfOldestMan(Collection <People> peoples)

{

int temp=peoples.iterator().next().getAge();

for(People i:peoples)

{

if(i.getAge()>temp)

{

temp=i.getAge();

}

}

return temp;

}

}

public class Source

{

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

{

Collection<People> peoples=Arrays.asList(

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

new People("Kayle",23,People.Gender.MAN),

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

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

);

PeopleImplementation imp=new PeopleImplementation();

System.out.println(imp.minimumAgeContainingE(peoples));

}

}

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}

class Student {

//Write Your Code Here.

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 String getName(){

return this.name;

}

public int getGraduationYear(){

return this.graduationYear;

}

public int getScore(){

return this.score;

}

@Override

public String toString() {

return "Student{" +

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

", graduationYear=" + graduationYear +

", score=" + score +

'}';

}

}

class Implementation{

//Write Your Code Here..

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

Comparator<Student> compareByScore=Comparator.comparing(Student::getScore).reversed();

Comparator<Student> compareByName=Comparator.comparing(Student::getName);

Comparator<Student> compare=compareByScore.thenComparing(compareByName);

return students.stream().sorted(compare).collect(toList());

}

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

return students.stream().collect(groupingBy(Student::getGraduationYear));

}

public Student findOneBest(Collection<Student> students){

Comparator<Student> compareByScore=Comparator.comparing(Student::getScore).reversed();

List<Student> s=students.stream().sorted(compareByScore).collect(toList());

return s.iterator().next();

}

}

public class Source {

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

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

Collection<Student> students=Arrays.asList(new Student("S2014-11",2014,17),

new Student("S2014-18",2014,20),new Student("S2013-18",2013,20));

Implementation i=new Implementation();

System.out.println(i.findOneBest(students));

}

}

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

**Sample Input**

Author joshuaBloch = **new** Author("Joshua", "Bloch");

Author brianGoetz = **new** Author("Brian", "Goetz");

Author barryBurd = **new** Author("Barry", "Burd");

Book Java1 = **new** Book("Effective Java", joshuaBloch);

Book Java2 = **new** Book("Java Concurrency In Practice", brianGoetz);

Book Java3 = **new** Book("Java For Dummies", barryBurd);

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

Java1, Java2, Java3

//Input **for** first method

joshuaBloch, brianGoetz, barryBurd

//Input **for** second method

**Sample Output**

[Effective Java, Java Concurrency In Practice, Java For Dummies]

[Joshua, Brian, Barry]

class Author {

private String firstName;

private String lastName;

public Author(String firstName,String lastName) {

this.firstName=firstName;

this.lastName=lastName;

}

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

@Override

public String toString() {

return "Author{" +

"firstName='" + firstName + '\'' +

", lastName='" + lastName + '\'' +

'}';

}

}

class Book{

//Write Your Code Here..

private String title;

private Author author;

public Book(String title,Author author)

{

this.title=title;

this.author=author;

}

public String getTitle() {

return title;

}

public void setTitle(String title) {

this.title = title;

}

public Author getAuthor() {

return author;

}

public void setAuthor(Author author) {

this.author = author;

}

@Override

public String toString() {

return "Book{" +

"title='" + title + '\'' +

", author=" + author +

'}';

}

}

class BookImplementation{

//Write Your Code Here..

public List<String>firstNameOfAuthors(Author...authors)

{

return Arrays.stream(authors).map(auth -> auth.getFirstName()).

collect(Collectors.toList());

}

public List<String>titlesOf(Book...books){

return Arrays.stream(books).map(j->j.getTitle()).collect(Collectors.toList());

}

}

public class Source {

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

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

}

}

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

class Car {

private String name,carName;

private double price;

Car(String pName,String cName,double p){

this.name=pName;

this.carName=cName;

this.price=p;

}

public void setName(String name){

this.name=name;

}

public void setCarName(String carName){

this.carName=carName;

}

public void setPrice(double price){

this.price=price;

}

public String getName(){

return name;

}

public String getCarName(){

return carName;

}

public double getPrice(){

return price;

}

}

class CarImplementation {

public double sumOfPrices(List<Car> carList){

return carList.stream().map(car->car.getPrice()).

reduce(0.0,(price1,price2)->price1+price2);

}

public List<String> printName(List<Car> carList){

return carList.stream().filter(car->car.getPrice()>25000.00).

map(car->car.getCarName()).collect(Collectors.toList());

}

public double maxPrice(List<Car> carList){

return carList.stream().map(car->car.getPrice()).max(Double::compare).get();

}

}

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

class Employee {

String name;

int salary;

Employee(String name, int salary)

{

this.name=name;

this.salary=salary;

}

public void setName(String name)

{

this.name=name;

}

public String getName()

{

return name;

}

public void setSalary(int salary)

{

this.salary=salary;

}

public int getSalary()

{

return salary;

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder("<");

sb.append("name: ");

sb.append(name);

sb.append(" salary: ");

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

return sb.toString();

}

}

class EmployeeInfo{

enum SortMethod

{

BYNAME,

BYSALARY

}

public List<Employee> sort(List<Employee> emps, final SortMethod method)

{

if(method == SortMethod.BYNAME)

{

return emps.stream().sorted(Comparator.comparing(Employee :: getName))

.collect(Collectors.toList());

}

else

{

return emps.stream().sorted(Comparator.comparing(Employee :: getSalary)

.thenComparing(Employee :: getName)).collect(Collectors.toList());

}

}

public Employee getThirdInCollection(Collection<Employee> entities)

{

return entities.stream().skip(2).findFirst().get();

}

public boolean isCharacterPresentInAllNames(Collection<Employee> entities,String character)

{

return entities.stream().allMatch(name -> name.getName().contains(character));

}

}