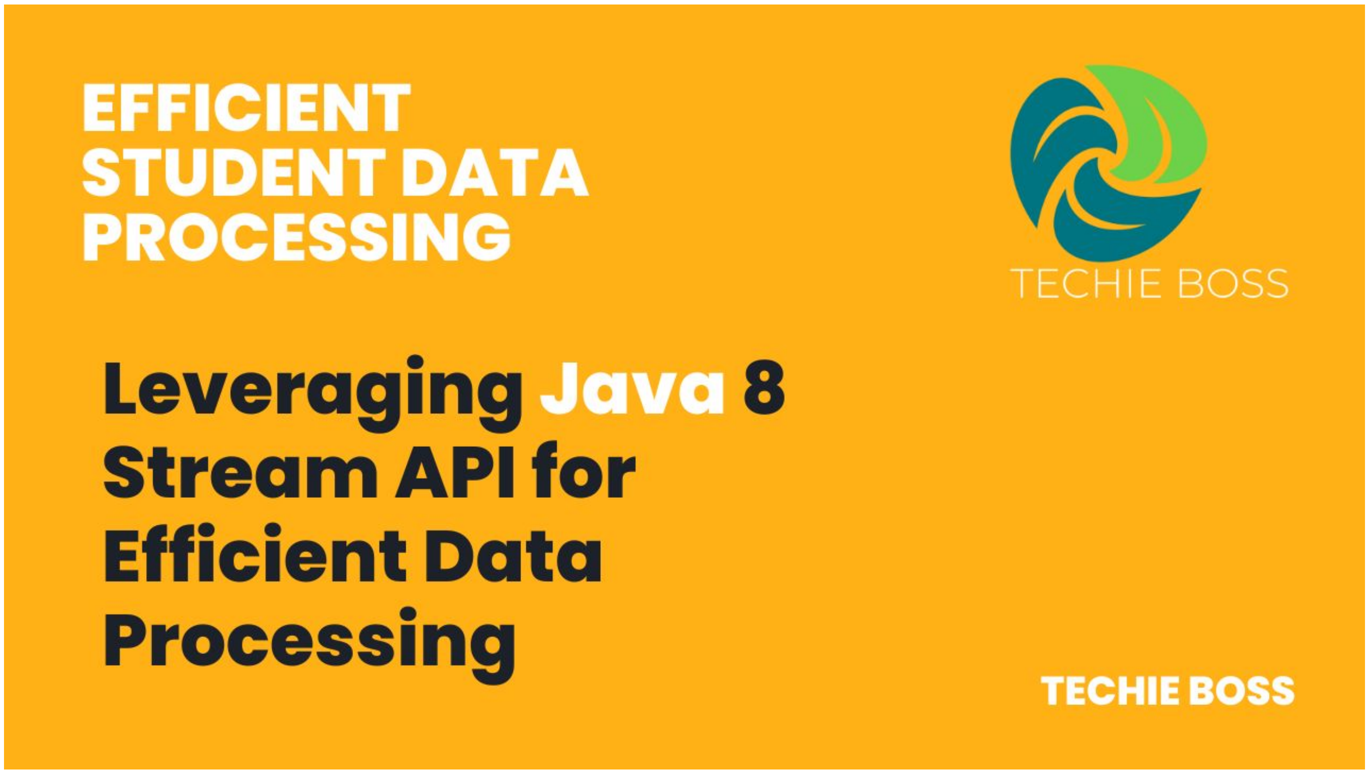
TECHIE BOSS


EFFICIENT STUDENT DATA PROCESSING
TECHIE BOSS
Leveraging Java 8 Stream API for Efficient Data Processing
TECHIE BOSS

Java 8 Stream API for Student Management

June 12, 2023 by ascblogger1

Java 8 Stream API For Student Management : Harnessing the Potential of Java 8 Stream API in a Student Management

System

Stream API is a Powerful Feature of Java 8 Release. Stream API Allows us to perform Various operation on Data . In this post we are going to Solve OOPS Based Problems With Stream API. Let's Get Started ……

Managing student information efficiently is a key aspect of any educational institution.

Student Management System

Implement a student management system using OOP principles. Create classes for **`Student`**, **Subject**, and **`School`**. By Using the Stream API we will perform below operations on the Student, Subject and School Objects.

By leveraging the functional programming capabilities of the Java 8 Stream API, we can simplify the implementation of these functionalities and achieve concise and readable code. This is not only saves time and effort but also enhances the overall efficiency of the student performance evaluation process.

Throughout this post, we will provide code examples and step-by-step explanations for each functionality, making it easier for you to understand and implement the Student Management System using Java 8 Stream API in your own projects.

So, let's dive into the details and explore how Java 8 Stream API can change student performance evaluation in a Student Management System.

1. Filtering students by subject
2. Finding the average grade
3. Sort students based on their grades
4. Finding Students with the Highest Grade
5. Grouping Students by Age
6. Counting Students by Subject
7. Finding Students with a Passing Grade
8. Calculating the Overall GPA of Each Student

Step 1 : Create The Subject Class

```
package com.techieboss.student_management_system;

```

```java
public class Subject {
    private String name;
    private int grade;

    public Subject(String name, int grade) {
        super();
        this.name = name;
        this.grade = grade;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getGrade() {
        return grade;
    }
    public void setGrade(int grade) {
        this.grade = grade;
    }
}
```

Step 2 : Create the **Student** Class

```java
package com.techieboss.student_management_system;

import java.util.ArrayList;
import java.util.List;

public class Student {
    private String name;
    private int age;
    private List<Subject> subjects;

    public Student(String name, int age) {
        this.name = name;
        this.age = age;
        subjects = new ArrayList<>();
    }

    public void addSubject(String subjectName, int grade) {
        Subject subject = new Subject(subjectName, grade);
        subjects.add(subject);
    }

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public List<Subject> getSubjects() {
        return subjects;
    }
    public void setSubjects(List<Subject> subjects) {
        this.subjects = subjects;
    }
}
```

Step 3 : Create the **School** Class with **List of Student**

```
package com.techieboss.student_management_system;

import java.util.ArrayList;
import java.util.List;

public class School {
    private List<Student> students;

    School() {
        students = new ArrayList<>();
    }

    public void addStudent(Student students) {
        this.students.add(students);
    }

    public List<Student> getStudents() {
        return students;
    }
    // Other methods like getStudents(), removeStudent(), etc.
}
```

Step 4 : Prepared the Data inside Test Class for Stream API Sample Examples

```
package com.techieboss.student_management_system;

public class Test {

    public static void main(String[] args) {
    Student student1 = new Student("John", 20);
        student1.addSubject("Math", 90);
        student1.addSubject("Science", 85);

        Student student2 = new Student("Alice", 22);
        student2.addSubject("Science", 95);
        student2.addSubject("History", 80);

        Student student3 = new Student("Bob", 21);
        student3.addSubject("Math", 92);
        student3.addSubject("History", 75);

        // Create School and add students
        School school = new School();
        school.addStudent(student1);
        school.addStudent(student2);
        school.addStudent(student3);
    }
}
```

Now Let’s find the Solution for First Problem :

1. Filtering Students by Subject :

In Below Code Snippet we have used stream() function inside filter() function . The Reason is we have List of Subjects attached with each Student so to apply filter operations over subjects nested stream() function is used.

```
     public static void filterStudentBySubject(School school) {
        String subjectName = "Math";
           List<Student> filteredStudents = school.getStudents()
                   .stream()
                   .filter(student -> student.getSubjects()
                           .stream()
                           .anyMatch(subject -> subject.getName().e
                   .collect(Collectors.toList());

           for (Student student : filteredStudents) {
               System.out.println(student.getName());
           }
    }

     Output :
     John
     Bob
```

2. Finding the average Grade :

```
public static void averageGrade(School school) {
        double averageGrade = school.getStudents()
                .stream()
                .flatMap(student -> student.getSubjects().stream())
                .mapToInt(Subject::getGrade)
                .average()
                .orElse(0.0);

        System.out.println("Average Grade :  " + averageGrade);
    }

Output :  Average Grade :  86.16666666666667
```

In this Solution we have used flatMap() , which might be quiet difficult to understand all , so i am going to explain in more detail.

In the context of the student management system, each student has a list of subjects. The goal is to calculate the average grade of all the subjects across all the students.

By using the `flatMap()` operation, we can efficiently flatten the nested structure of students and their subjects into a single stream of subjects. This allows us to perform subsequent operations on the subjects such as calculating the average grade in this scenario.

Step 1 : `school.getStudents().stream()` returns a stream of `Student` objects.

Step 2 : `.flatMap(student -> student.getSubjects().stream())` takes each `Student` object from the stream and applies the `getSubjects()` method to obtain a stream of Subjects for each student. The `flatMap()` operation effectively flattens the stream of streams of subjects into a single stream of subject.

For example, if there are three students, and each student has two subjects, the `flatMap()` operation will create a single stream containing all the subjects.

Let's understand this in some graphical way:

Before applying flatMap() how the structure look like :

```
[Student1] ---> [subject1, subject2, subject3]
[Student2] ---> [subject4, subject5]
[Student3] ---> [subject6, subject7, subject8, subject9]
```

After Applying flapMap() how the structure will look like :

```
[subject1, subject2, subject3, subject4, subject5, subject6, subject
```

Here the `flatMap()` operation transforms the stream of students into a flattened stream of subjects. The resulting stream consists of all the subjects from all the students, merged into a single stream.
I Hope you get the clarity about how the flatMap() is working.

Step 3 : .mapToInt(Subject::getGrade) applies the getGrade() method to each Subject object in the stream to extract the grades as an IntStream. This operation converts the stream of subjects into an IntStream of grades.

Step 4 : .average() calculates the average of all the grades in the IntStream. It returns an OptionalDouble representing the average value.

Step 5 : .orElse(0.0) is used to provide a default value of 0.0 in case the OptionalDouble is empty, which occurs if there are no grades available.

3. Sort Students Based on Their Grades

To sort students based on their grades in ascending order, we can use the sorted() method from the Stream API in Java 8.

For this Problem Statement First we should have average grade of Each students. There is two approach to find the average grade for Each Student.

1. **First approach** is we can introduce **new methods inside Student Class** to calculate the average grade for each student. so below is the code to calculating the average grade for each student.

```
public double getAverageGrade() {
        return this.calculateAverageGrade();
    }

    public double calculateAverageGrade() {
        return subjects.stream()
        .mapToInt(Subject::getGrade)
        .average()
```

```
            .average()
            .orElse(0.0);
    }
```

Now we can write the code to sort the student based on their Grades :

```
public static void sortStudentBasedOnGradesApproach1(School school)
        List<Student> sortedStudents = school.getStudents()
                .stream()
                .sorted(Comparator.comparingDouble(Student::calculat
                .collect(Collectors.toList());

        for (Student student : sortedStudents) {
            System.out.println(student.getName() + " - Average Grade
        }
    }
Output :
Bob - Average Grade: 83.5
John - Average Grade: 87.5
Alice - Average Grade: 87.5
```

2. Second Approach is without adding new method inside Student class , we can calculate using stream at the same time.

```
public static void sortStudentBasedOnGradesApproach2(School school)
         List<Student> sortedStudents = school.getStudents()
                    .stream()
                    .sorted(Comparator.comparingDouble(student -> st
                            .stream()
                            .mapToInt(Subject::getGrade)
                            .average()
                            .orElse(0.0)))
                    .collect(Collectors.toList());

            // Printing the sorted students
            for (Student student : sortedStudents) {
                System.out.println(student.getName() + " - Average G
                        student.getSubjects().stream().mapToInt(Subj
            }
    }

Output :
Bob - Average Grade: 83.5
John - Average Grade: 87.5
Alice - Average Grade: 87.5
```

4. Finding Students with the Highest Grade

```java
public static void studentWithHigheshGrade(School school) {
        // Find students with the highest grade
        Optional<Student> studentWithHighestGrade = school.getStuden
                .stream()
                .max(Comparator.comparingDouble(student -> student.g
                        .stream()
                        .mapToDouble(Subject::getGrade)
                        .max()
                        .orElse(0.0)));

        // Print the student with the highest grade
        studentWithHighestGrade.ifPresent(student -> {
            System.out.println("Student with the highest grade: " +
            System.out.println("Highest grade: " + student.getSubjec
                    .stream()
                    .mapToDouble(Subject::getGrade)
                    .max()
                    .orElse(0.0));
        });
    }
```

5. Group Students By Age

```java
    public static void groupStudentsByAge(School school) {
        Map<Integer, List<Student>> studentsByAge = school.getStuden
                .stream()
                .collect(Collectors.groupingBy(Student::getAge));

        studentsByAge.forEach((age, students) -> {
            System.out.println("Age: " + age);
            System.out.println("Students: " + students);
        });
    }

Output:
Age: 20
Students: [Student [name=John, age=20, subjects=[Subject [name=Math,
Age: 21
Students: [Student [name=Bob, age=21, subjects=[Subject [name=Math,
Age: 22
Students: [Student [name=Alice, age=22, subjects=[Subject [name=Scie
```

In this solution, we are using **groupingBy()** method from the Collectors class to group the students by their age. Let's see step by step

1. Calling **school.getStudents().stream()** to convert the list of students into a stream.

1.

2. Used the collect() method on the stream, along with **Collectors.groupingBy(Student::getAge)**. This creates a **Map<Integer, List<Student>>**

where the keys are the ages of the students, and the values are lists of students with that age.

3. **Student::getAge is a method reference** that specifies the property to group the students by the **age(in this case)**.
4. The **collect()** method collects the elements of the stream according to the grouping operation and returns the resulting map.

6. Counting Students by Subject

To count the number of students enrolled in each Subject, we can use the **Collectors.groupingBy()** method in combination with the **Collectors.counting()**

```
public static void countStudentsBySubject(School school) {
        Map<String, Long> studentCountBySubjects = school.getStudent
                .stream()
                .flatMap(student -> student.getSubjects().stream())
                .collect(Collectors.groupingBy(Subject::getName, Col


        studentCountBySubjects.forEach((subject, count) -> {
            System.out.println("Subject: " + subject);
            System.out.println("Number of students: " + count);
        });
    }

Output :
subject: Science
Number of students: 2
subject: History
Number of students: 1
subject: Math
Number of students: 2
```

7. Finding Students with a Passing Grade:

For this problem Statement we are going to use the filter() method to filter out students whose grade is less than **60 (assuming a passing grade is 60 or higher)**. The filtered whose grade is less than **60 (assuming a passing grade is 60 or higher).** The filtered students will collect into a list using the collect() method with Collectors.toList()

```
public static void studentWithPassingGrade(School school) {
        List<Student> studentWithPassingGrade = school.getStudents()
                .stream()

                .filter(student ->
                        student.getSubjects().stream().allMatch(subj
                .collect(Collectors.toList());

        studentWithPassingGrade.forEach(student -> System.out.printl
    }

Output :
John
Alice
Bob
```

8. Calculating the Overall GPA of Each Student:

For this Problem Statement we will calculate the overall GPA of each student directly within the toMap() method. We will perform inline calculation using a lambda expression. Inside the lambda expression, we will retrieve the list of subjects for each student, calculate the total subjects then sum the grades using mapToDouble(), and divide it by the total subjects to get the GPA.

```
public static void overallGPAOfEachStudent(School school) {
        Map<Student, Double> studentGPAMap = school.getStudents()
                .stream()
                .collect(Collectors.toMap(
                        student -> student,
                        student -> {
                            List<Subject> subjects = student.getSubj
                            int totalCredits = subjects.size();
                            double totalGradePoints = subjects.strea
                                    .mapToDouble(Subject::getGrade)
                                    .sum();
                            return totalGradePoints / totalCredits;
                        }
                ));

        // Print overall GPA of each student
        studentGPAMap.forEach((student, gpa) -> {
            System.out.println("Student: " + student.getName());
            System.out.println("Overall GPA: " + gpa);
        });
    }
```

```
Output :
Student: Bob
Overall GPA: 83.5
Student: John
Overall GPA: 87.5

Student: Alice
Overall GPA: 87.5
```

I have tried my best to explain everything in this post , please comment me below if you need any question/doubt . I will try to respond you immediately.

Related Posts

Java 8 Complete Features

Default Interface Method In Java 8

4 thoughts on “Java 8 Stream API for Student Management”

Sanchit Singh
June 13, 2023 at 7:38 am

Very helpful 👍

Reply

Aaditya
June 13, 2023 at 8:21 pm

Explained very well, thanks

Reply

Dhananjay Dubey
June 13, 2023 at 8:32 pm

Nice work really helps in excel you fullstack knowledge, thanks for great platform.

Reply

Farhan Usmani
June 14, 2023 at 7:33 pm

It's too use full & very help full

Mr. Frank

Reply

Leave a Comment

Name *

Email *

☐ Save my name, email, and website in this browser for the next time I comment.

Post Comment

Search

Search

Recent Posts

[More Insights In Functional Interface]

[Java 8 Stream API for Student Management]

[Java 8 Complete Features]

[Paytm Payment API Integration using SpringBoot : Complete Guide]

[What is the advantage of having Static method in Interface in java 8 ?]

About us

We are trying to make Techie Boss as a Plateform for the Begineers and Experienced Person who Really wants the Deeper Knowledge of the Any Technology.