Final Exam Question: Java OOP-Based CRUD Operations Using HashMaps

Problem Description

You are required to develop a simple student management system using Object-Oriented Programming (OOP) principles in Java. The system should manage a collection of students, allowing for the addition, removal, updating, and searching of student records. Each student will be represented as an object, and you will store these objects in a `` `HashMap` `` where the key is the student's ID and the value is the `` `Student` `` object. In addition to basic CRUD operations, you will implement more complex operations, such as filtering students based on specific criteria and calculating statistics.

Requirements

1. **Student Class**
   - Create a class named `` `Student` `` with the following attributes:
     - `` `studentId` `` (String): The unique ID for each student.
     - `` `name` `` (String): The name of the student.
     - `` `age` `` (int): The age of the student.
     - `` `gpa` `` (double): The GPA (Grade Point Average) of the student.
   - Implement the following methods in the `` `Student` `` class:
     - `` `public Student(String studentId, String name, int age, double gpa)` ``: Constructor to initialize all attributes.
     - `` `public String getStudentId()` ``: Returns the `` `studentId` `` of the student.
     - `` `public String getName()` ``: Returns the `` `name` `` of the student.
     - `` `public int getAge()` ``: Returns the `` `age` `` of the student.
     - `` `public double getGpa()` ``: Returns the `` `gpa` `` of the student.
     - `` `public void setGpa(double gpa)` ``: Sets the `` `gpa` `` of the student.
     - `` `@Override public String toString()` ``: Returns a string representation of the student, including all attributes.
2. **StudentManagement Class**
   - Create a class named `` `StudentManagement` `` that manages a collection of `` `Student` `` objects using a `` `HashMap<String, Student>` ``.
   - Implement the following methods in the `` `StudentManagement` `` class:
     - `` `public void addStudent(Student student) throws DuplicateStudentException` ``: Adds a new student to the system. Ensure that a student with the same `` `studentId` `` does not already exist in the system.
     - `` `public void removeStudent(String studentId) throws StudentNotFoundException` ``: Removes a student from the system based on their `` `studentId` ``. If the student does not exist, throw a custom exception `` `StudentNotFoundException` ``.
     - `` `public void updateStudentGpa(String studentId, double newGpa) throws StudentNotFoundException` ``: Updates the GPA of a student based on their `` `studentId` ``. If the student does not exist, throw a custom exception `` `StudentNotFoundException` ``.

    - `` `public Student searchStudentById(String studentId) throws StudentNotFoundException` ``: Searches for a student by their `` `studentId` `` and returns the `` `Student` `` object. If the student is not found, throw a custom exception `` `StudentNotFoundException` ``.
    - `` `public List<Student> getStudentsByGpa(double minGpa, double maxGpa)` ``: Returns a list of students whose GPA falls within the specified range.
    - `` `public List<Student> getStudentsByAgeRange(int minAge, int maxAge)` ``: Returns a list of students whose age falls within the specified range.
    - `` `public double calculateAverageGpa()` ``: Returns the average GPA of all students in the system.
    - `` `public Map<String, List<Student>> groupStudentsByAge()` ``: Groups students by age and returns a map where the key is the age, and the value is a list of students with that age.

3. Custom Exception Classes
    - Create two custom exception classes:
        - `` `StudentNotFoundException` ``: Thrown when a student with a specified `` `studentId` `` is not found in the system.
        - `` `DuplicateStudentException` ``: Thrown when trying to add a student with a `` `studentId` `` that already exists in the system.
    - Each exception should have at least one constructor that takes a `` `String` `` message and passes it to the superclass `` `Exception` ``.
4. Main Application
    - Create a `` `Main` `` class with a `` `main` `` method to test all the functionalities of the `` `StudentManagement` `` class.
    - Demonstrate adding, removing, updating, and searching for students.
    - Demonstrate handling of the custom exceptions by providing invalid inputs (e.g., searching for a student that doesn't exist).
    - Demonstrate the more complex operations such as filtering by GPA, grouping by age, and calculating the average GPA.

Example Scenario

1. Add a student with `` `studentId = "S001"` ``, `` `name = "Alice"` ``, `` `age = 20` ``, `` `gpa = 3.5` ``.
2. Add another student with `` `studentId = "S002"` ``, `` `name = "Bob"` ``, `` `age = 22` ``, `` `gpa = 3.8` ``.
3. Attempt to add a student with `` `studentId = "S001"` `` again (should throw `` `DuplicateStudentException` ``).
4. Update the GPA of the student with `` `studentId = "S001"` `` to 3.9.
5. Remove the student with `` `studentId = "S003"` `` (should throw `` `StudentNotFoundException` ``).
6. Search for a student by `` `studentId = "S002"` `` and display their details.
7. Retrieve a list of students with a GPA between 3.0 and 4.0 and display the results.
8. Group students by age and display the result.
9. Calculate and display the average GPA of all students in the system.