**Student Grade Project**

package studentgrade;  
  
import java.util.Date;  
  
public class Student {  
  
 int studentId;  
 String departmentId;  
 String courseNumber;  
 Date courseDate;  
 double credits;  
 double grade;  
  
 public Student(int studentId, String departmentId, String courseNumber, Date courseDate, double credits, double grade) {  
 this.studentId = studentId;  
 this.departmentId = departmentId;  
 this.courseNumber = courseNumber;  
 this.courseDate = courseDate;  
 this.credits = credits;  
 this.grade = grade;  
 }  
  
 public int getStudentId() {  
 return studentId;  
 }  
  
 public void setStudentId(int studentId) {  
 this.studentId = studentId;  
 }  
  
 public String getDepartmentId() {  
 return departmentId;  
 }  
  
 public void setDepartmentId(String departmentId) {  
 this.departmentId = departmentId;  
 }  
  
 public String getCourseNumber() {  
 return courseNumber;  
 }  
  
 public void setCourseNumber(String courseNumber) {  
 this.courseNumber = courseNumber;  
 }  
  
 public Date getCourseDate() {  
 return courseDate;  
 }  
  
 public void setCourseDate(Date courseDate) {  
 this.courseDate = courseDate;  
 }  
  
 public double getCredits() {  
 return credits;  
 }  
  
 public void setCredits(double credits) {  
 this.credits = credits;  
 }  
  
 public double getGrade() {  
 return grade;  
 }  
  
 public void setGrade(double grade) {  
 this.grade = grade;  
 }  
  
 @Override  
 public String toString() {  
 return "Student{" +  
 "studentId=" + studentId +  
 ", departmentId='" + departmentId + '\'' +  
 ", courseNumber='" + courseNumber + '\'' +  
 ", courseDate=" + courseDate +  
 ", credits=" + credits +  
 ", grade=" + grade +  
 '}';  
 }  
}

package studentgrade;  
  
import java.text.DecimalFormat;  
import java.util.\*;  
import java.util.stream.Collectors;  
  
public class Main {  
 public static final int *n* = 100000;  
 public static void main(String[] args) {  
 DecimalFormat decimalFormat = new DecimalFormat("0.00");  
  
 List<Student> students = *loadStudent*();  
// for (Student student: students) {  
// System.out.println(student);  
// }  
  
 *serialStream*(decimalFormat, students);  
  
 System.*out*.println("\n");  
 *parallelStream*(decimalFormat, students);  
  
 }  
  
 private static void serialStream(DecimalFormat decimalFormat, List<Student> students) {  
 double currentTime = 0d;  
 Date start, end;  
 start = new Date();  
  
 double averageGpa = students.stream()  
 .filter(student -> student.departmentId == "MSCS")  
 .collect(Collectors.*averagingDouble*(Student::getGrade));  
 System.*out*.println("Average : "+ decimalFormat.format(averageGpa));  
  
 Optional<Student> maxGpa = students.stream()  
 .filter(student -> student.departmentId == "MSCS")  
 .max(Comparator.*comparing*(Student::getGrade));  
 System.*out*.println("Max GPA : "+decimalFormat.format(maxGpa.get().getGrade()));  
  
 Optional<Student> minGPA = students.stream()  
 .filter(student -> student.departmentId == "MSCS")  
 .min(Comparator.*comparing*(Student::getGrade));  
 System.*out*.println("Min GPA : "+decimalFormat.format(minGPA.get().getGrade()));  
  
 List<Student> sortedList = students.stream()  
 .filter(student -> student.departmentId == "MSCS")  
 .sorted(Comparator.*comparing*(Student::getGrade))  
 .collect(Collectors.*toList*());  
 Optional<Student> s = sortedList.stream()  
 .findFirst();  
 System.*out*.println("Min GPA with Sorted :"+decimalFormat.format(s.get().getGrade()));  
  
 end = new Date();  
  
 currentTime = end.getTime() - start.getTime();  
 System.*out*.println("Execution Time Serial: " + (currentTime / 1000)  
 + " seconds.");  
 }  
  
 private static void parallelStream(DecimalFormat decimalFormat, List<Student> students) {  
 double currentTime = 0d;  
 Date start, end;  
 start = new Date();  
  
 double averageGpa = students.parallelStream()  
 .filter(student -> student.departmentId == "MSCS")  
 .collect(Collectors.*averagingDouble*(Student::getGrade));  
 System.*out*.println("Average : "+ decimalFormat.format(averageGpa));  
  
 Optional<Student> maxGpa = students.parallelStream()  
 .filter(student -> student.departmentId == "MSCS")  
 .max(Comparator.*comparing*(Student::getGrade));  
 System.*out*.println("Max GPA : "+decimalFormat.format(maxGpa.get().getGrade()));  
  
 Optional<Student> minGPA = students.parallelStream()  
 .filter(student -> student.departmentId == "MSCS")  
 .min(Comparator.*comparing*(Student::getGrade));  
 System.*out*.println("Min GPA : "+decimalFormat.format(minGPA.get().getGrade()));  
  
 List<Student> sortedList = students.parallelStream()  
 .filter(student -> student.departmentId == "MSCS")  
 .sorted(Comparator.*comparing*(Student::getGrade))  
 .collect(Collectors.*toList*());  
 Optional<Student> s = sortedList.stream()  
 .findFirst();  
 System.*out*.println("Min GPA with Sorted :"+decimalFormat.format(s.get().getGrade()));  
  
 end = new Date();  
  
 currentTime = end.getTime() - start.getTime();  
 System.*out*.println("Execution Time Parallel: " + (currentTime / 1000)  
 + " seconds.");  
 }  
  
 public static List<Student> loadStudent() {  
 List<Student> studentList = new ArrayList<>();  
 Random random = new Random();  
 random.setSeed(1234567890);  
 for (int i=0;i<*n*;i++) {  
 double grade = random.nextDouble(0.0,4.0);  
 studentList.add(new Student((int) (100000 \* Math.*random*()),"MSCS","CS471",new Date(),120, grade));  
 }  
 return studentList;  
 }  
}

**Output:**

Average : 2.00

Max GPA : 4.00

Min GPA : 0.00

Min GPA with Sorted :0.00

Execution Time Serial: 0.329 seconds.

Average : 2.00

Max GPA : 4.00

Min GPA : 0.00

Min GPA with Sorted :0.00

Execution Time Parallel: 0.18 seconds.