Report: Object-Oriented Principles and Features Implemented in the Design and Code

In the presented design and code for the student database and evaluation system, several object-oriented principles and features have been implemented to ensure modularity, flexibility, and extensibility. Below are the key aspects:

1. Encapsulation:

- Encapsulation has been achieved by defining classes with **private** fields and providing **public** methods for accessing and modifying these fields. For example, fields such as "name", "roll", "email" in the "Student" class are private and accessed through **getter** methods. Most of the classes , their attributes and methods are created keeping this principal in mind .

2. Abstraction:

- Abstraction is utilized through abstract classes and methods. The `**Course**` class is abstract, providing a blueprint for **major** and **optional** courses. It declares abstract methods like `**calculateFinalScore()**` , leaving the implementation details to subclasses.

3. Inheritance:

- Inheritance is employed to establish an "**is-a**" relationship between classes. For instance, both `**MajorCourse**` and `**OptionalCourse**` inherit from the abstract class `**Course**`, inheriting its properties and behaviors.

4. Polymorphism:

- Polymorphism is demonstrated through **method overriding**. Subclasses such as `MajorCourse` and `OptionalCourse` **override the `calculateFinalScore()`** method to provide specialized implementations according to their requirements.

5. Composition:

- Composition is utilized to create complex objects by combining simpler ones. For instance, a `**Student**` object contains instances of `**MajorCourse**` and `**OptionalCourse**` objects, forming a composition relationship.

6. Modularity:

- The design promotes **modularity** by breaking down the system into smaller, manageable components such as `**StudentInfoDataBase**`, `**CourseInfoDataBase**`etc. Each class is responsible for a specific aspect of the system, enhancing maintainability and readability.

7. Flexibility and Extensibility:

- The design allows for easy incorporation of new courses, evaluation metrics, and assessment criteria. Courses and evaluation systems are defined flexibly, enabling future expansion without significant modifications to existing code.

8. Data Hiding:

- Implementation details of classes are hidden from external classes, promoting **Data hiding** and reducing dependencies. External classes interact with objects through public interfaces, enhancing encapsulation and abstraction.

9. Random Data Generation:

- The design includes a function to generate random student data, demonstrating a flexible approach to data generation. This feature enhances the realism and scalability of the system.

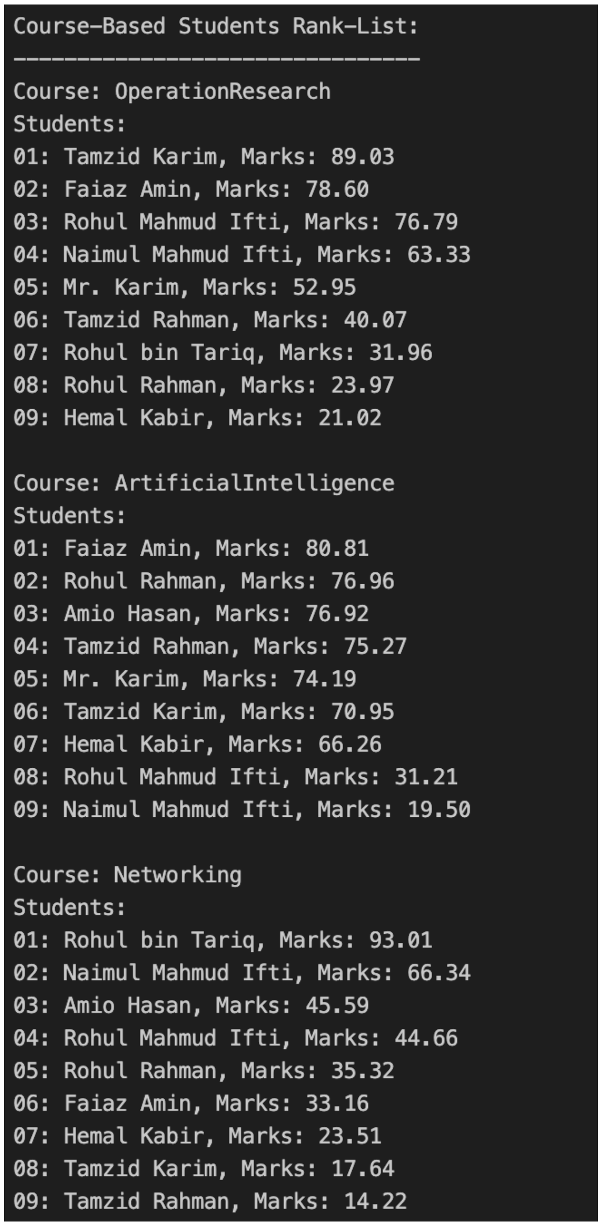
10. Sorting and Ranking:

- The system incorporates sorting algorithms to rank students and courses based on specified criteria. This feature showcases the flexibility and adaptability of the design to different evaluation scenarios.

Some Features were added to the Code according to the assignment requirement :

Should be able to find a course-based rank list

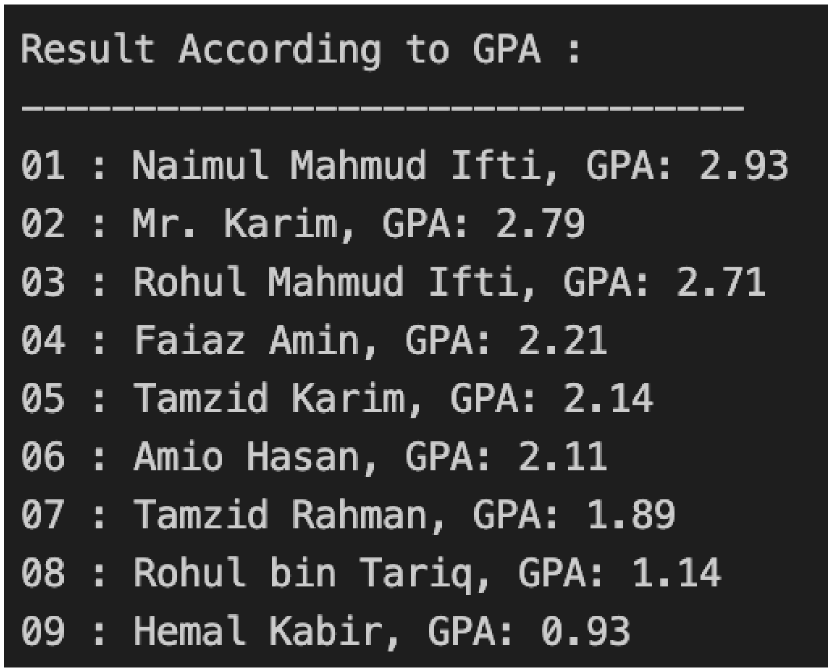
Should be able to provide a comprehensive list regarding a particular course to state the students that have enrolled in that course



```
Course-Based Students Rank-List:
--------------------------------
Course: OperationResearch
Students:
01: Tamzid Karim, Marks: 89.03
02: Faiaz Amin, Marks: 78.60
03: Rohul Mahmud Ifti, Marks: 76.79
04: Naimul Mahmud Ifti, Marks: 63.33
05: Mr. Karim, Marks: 52.95
06: Tamzid Rahman, Marks: 40.07
07: Rohul bin Tariq, Marks: 31.96
08: Rohul Rahman, Marks: 23.97
09: Hemal Kabir, Marks: 21.02

Course: ArtificialIntelligence
Students:
01: Faiaz Amin, Marks: 80.81
02: Rohul Rahman, Marks: 76.96
03: Amio Hasan, Marks: 76.92
04: Tamzid Rahman, Marks: 75.27
05: Mr. Karim, Marks: 74.19
06: Tamzid Karim, Marks: 70.95
07: Hemal Kabir, Marks: 66.26
08: Rohul Mahmud Ifti, Marks: 31.21
09: Naimul Mahmud Ifti, Marks: 19.50

Course: Networking
Students:
01: Rohul bin Tariq, Marks: 93.01
02: Naimul Mahmud Ifti, Marks: 66.34
03: Amio Hasan, Marks: 45.59
04: Rohul Mahmud Ifti, Marks: 44.66
05: Rohul Rahman, Marks: 35.32
06: Faiaz Amin, Marks: 33.16
07: Hemal Kabir, Marks: 23.51
08: Tamzid Karim, Marks: 17.64
09: Tamzid Rahman, Marks: 14.22
```

Should be able to calculate grades per course, based on the achieved number from the evaluation categories

Should be able to find the overall rank list based on the GPA



```
Result According to GPA :
--------------------------------
01 : Naimul Mahmud Ifti, GPA: 2.93
02 : Mr. Karim, GPA: 2.79
03 : Rohul Mahmud Ifti, GPA: 2.71
04 : Faiaz Amin, GPA: 2.21
05 : Tamzid Karim, GPA: 2.14
06 : Amio Hasan, GPA: 2.11
07 : Tamzid Rahman, GPA: 1.89
08 : Rohul bin Tariq, GPA: 1.14
09 : Hemal Kabir, GPA: 0.93
```

Data Generation with Randomness:

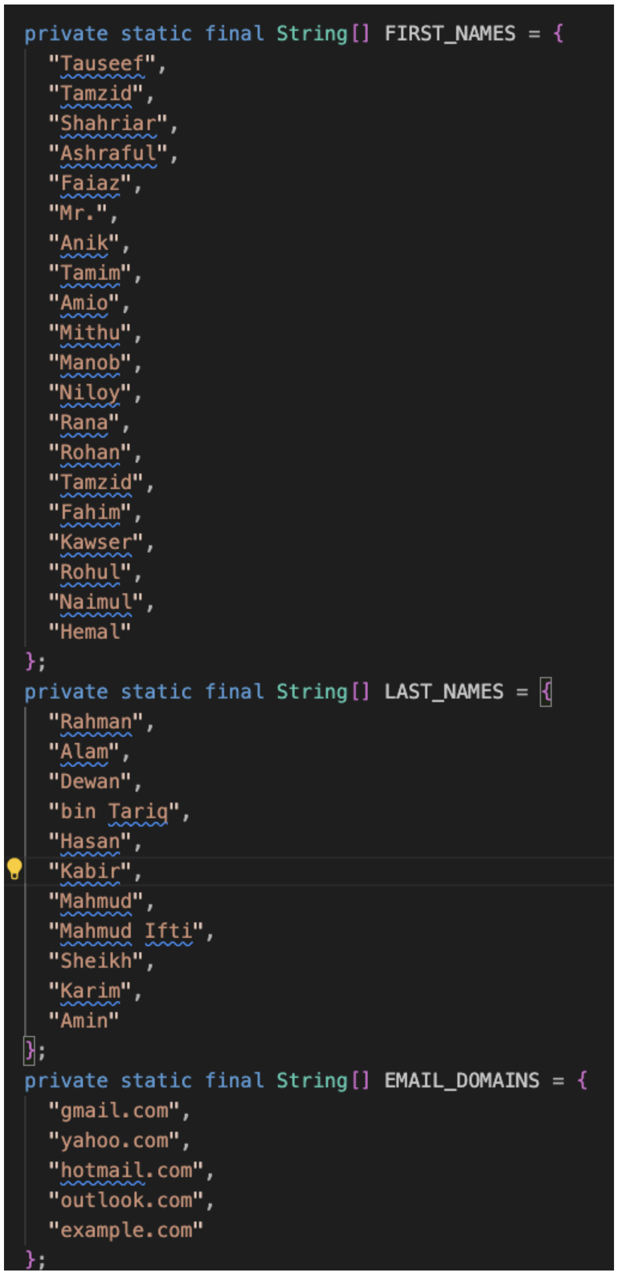
3. Take a variable N = 20 to denote the number of students
4. For each student, add some data to integrate their name, roll and email as per the given statement
5. Now, for each student assign four courses randomly (3 major courses, 1 optional course)
6. Now, for each student for each course, randomly pick an evaluation system maintaining the constraint of having 100 in each course
7. Now, randomly assign value for each course, for each evaluation metric (Final, mid, attendance, assignment, etc. as required by the chosen metric)

While generating Random Data all the points were taken in mind and were followed by.

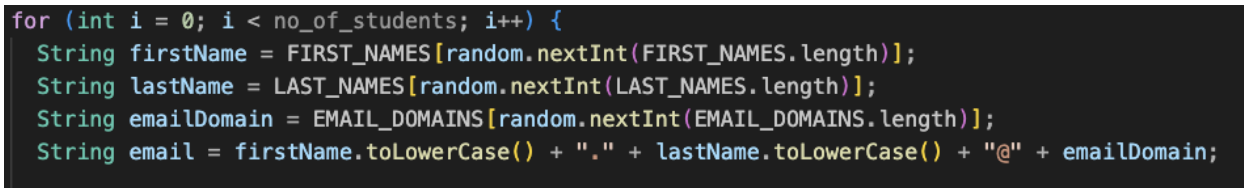
There were a few functionalities asked in the question , but to be precise no 8 , 11 and 12 were asking about implementing particular functions for the showcase of data which are implemented properly.

Unique Features :

01/ I have generated Random Students info . For this I stored some dummy first names and second names and used the Built in Random class to generate Random names and email of a user.

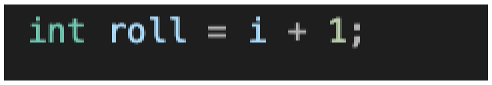


```java
private static final String[] FIRST_NAMES = {
    "Tauseef",
    "Tamzid",
    "Shahriar",
    "Ashraful",
    "Faiaz",
    "Mr.",
    "Anik",
    "Tamim",
    "Amio",
    "Mithu",
    "Manob",
    "Niloy",
    "Rana",
    "Rohan",
    "Tamzid",
    "Fahim",
    "Kawser",
    "Rohul",
    "Naimul",
    "Hemal"
};
private static final String[] LAST_NAMES = {
    "Rahman",
    "Alam",
    "Dewan",
    "bin Tariq",
    "Hasan",
    "Kabir",
    "Mahmud",
    "Mahmud Ifti",
    "Sheikh",
    "Karim",
    "Amin"
};
private static final String[] EMAIL_DOMAINS = {
    "gmail.com",
    "yahoo.com",
    "hotmail.com",
    "outlook.com",
    "example.com"
};
```



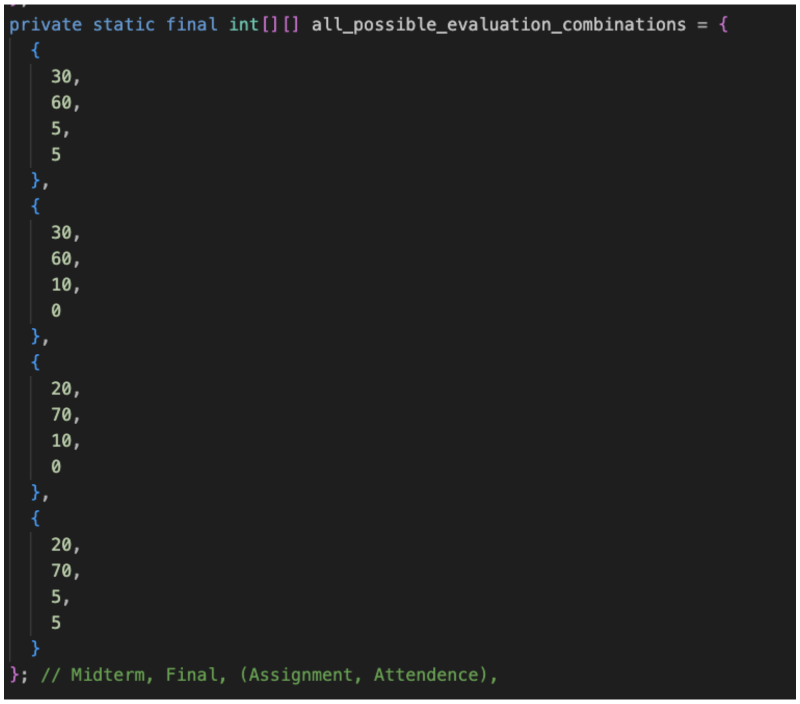
```java
for (int i = 0; i < no_of_students; i++) {
    String firstName = FIRST_NAMES[random.nextInt(FIRST_NAMES.length)];
    String lastName = LAST_NAMES[random.nextInt(LAST_NAMES.length)];
    String emailDomain = EMAIL_DOMAINS[random.nextInt(EMAIL_DOMAINS.length)];
    String email = firstName.toLowerCase() + "." + lastName.toLowerCase() + "@" + emailDomain;
```

Now , the roll number can not be random . It has to be unique . So , I insured uniqueness in roll number.

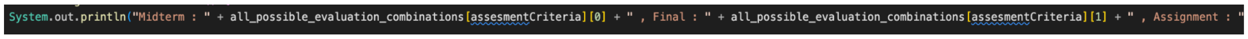


```
int roll = i + 1;
```

02/ **for each student**, **for each course,** randomly picking an evaluation system maintaining the constraint of having 100 in each course was quite difficult . I created a 2D array having all possible evaluation numbers .

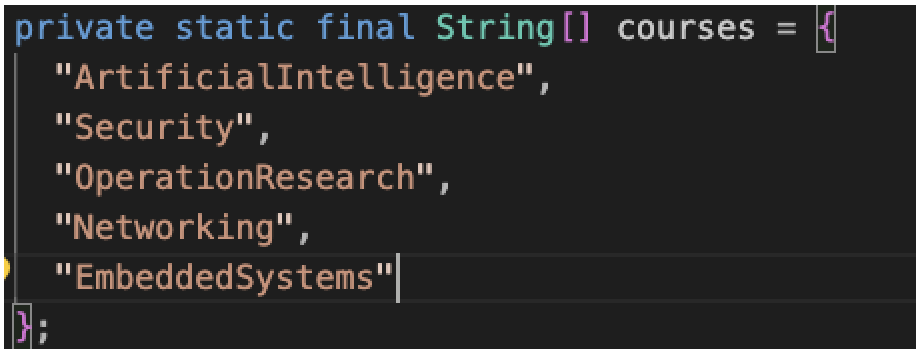


```
private static final int[][] all_possible_evaluation_combinations = {
  {
    30,
    60,
    5,
    5
  },
  {
    30,
    60,
    10,
    0
  },
  {
    20,
    70,
    10,
    0
  },
  {
    20,
    70,
    5,
    5
  }
}; // Midterm, Final, (Assignment, Attendence),
```



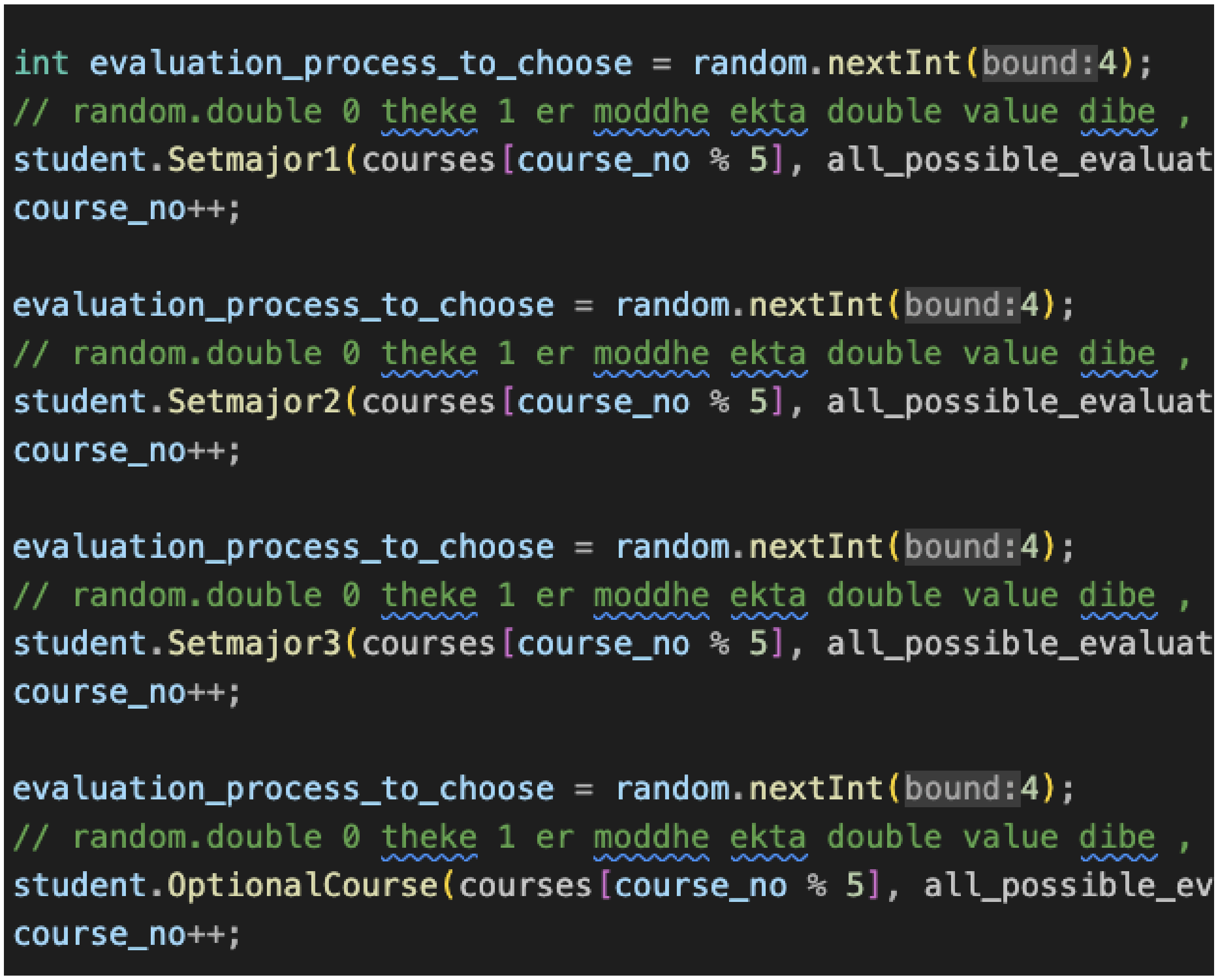
```
System.out.println("Midterm : " + all_possible_evaluation_combinations[assesmentCriteria][0] + " , Final : " + all_possible_evaluation_combinations[assesmentCriteria][1] + " , Assignment : "
```

03/ To ensure some sort of uniqueness among the course chosen by each student , I again used random function to generate a number between 0 and 4.



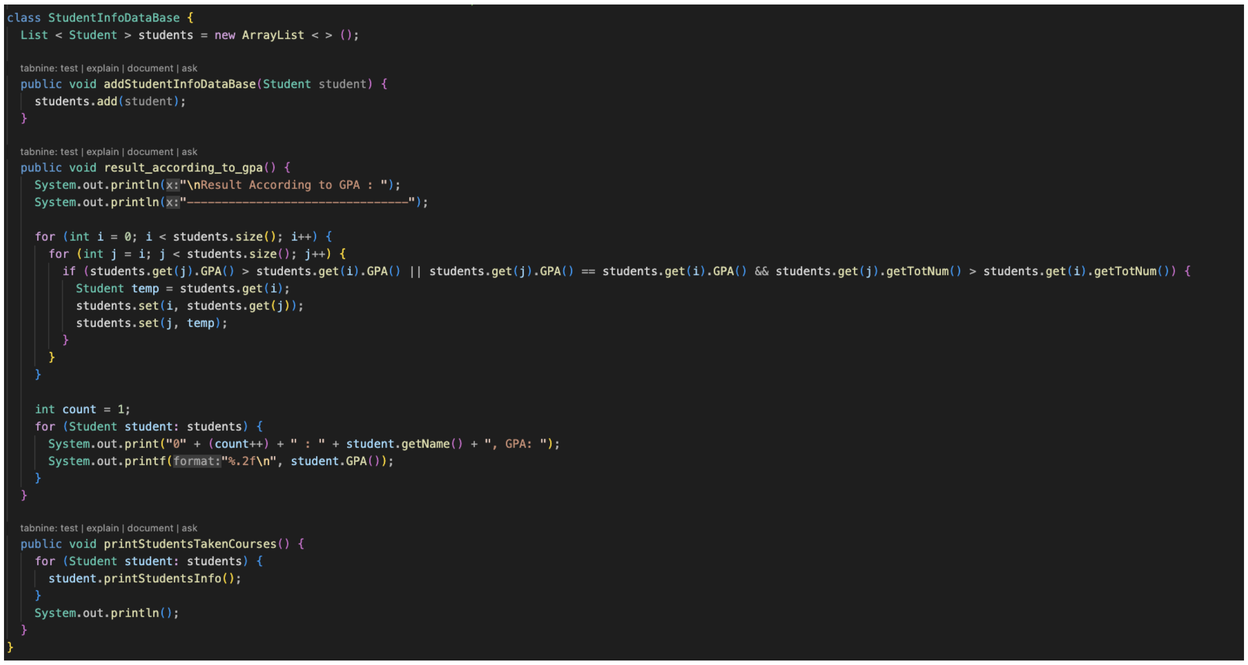
```
private static final String[] courses = {
  "ArtificialIntelligence",
  "Security",
  "OperationResearch",
  "Networking",
  "EmbeddedSystems"
};
```

Then I starting setting major from that subject and for every increasing count I modded the count , which makes one course not to be chosen every time.



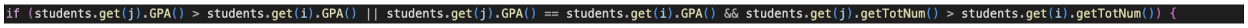
```
int evaluation_process_to_choose = random.nextInt(bound:4);
// random.double 0 theke 1 er moddhe ekta double value dibe ,
student.Setmajor1(courses[course_no % 5], all_possible_evaluat
course_no++;

evaluation_process_to_choose = random.nextInt(bound:4);
// random.double 0 theke 1 er moddhe ekta double value dibe ,
student.Setmajor2(courses[course_no % 5], all_possible_evaluat
course_no++;

evaluation_process_to_choose = random.nextInt(bound:4);
// random.double 0 theke 1 er moddhe ekta double value dibe ,
student.Setmajor3(courses[course_no % 5], all_possible_evaluat
course_no++;

evaluation_process_to_choose = random.nextInt(bound:4);
// random.double 0 theke 1 er moddhe ekta double value dibe ,
student.OptionalCourse(courses[course_no % 5], all_possible_ev
course_no++;
```

04/ To store the students' info I created a list and stored them there. When I had to rank them according to their GPA I used Bubble sort on them to sort them .



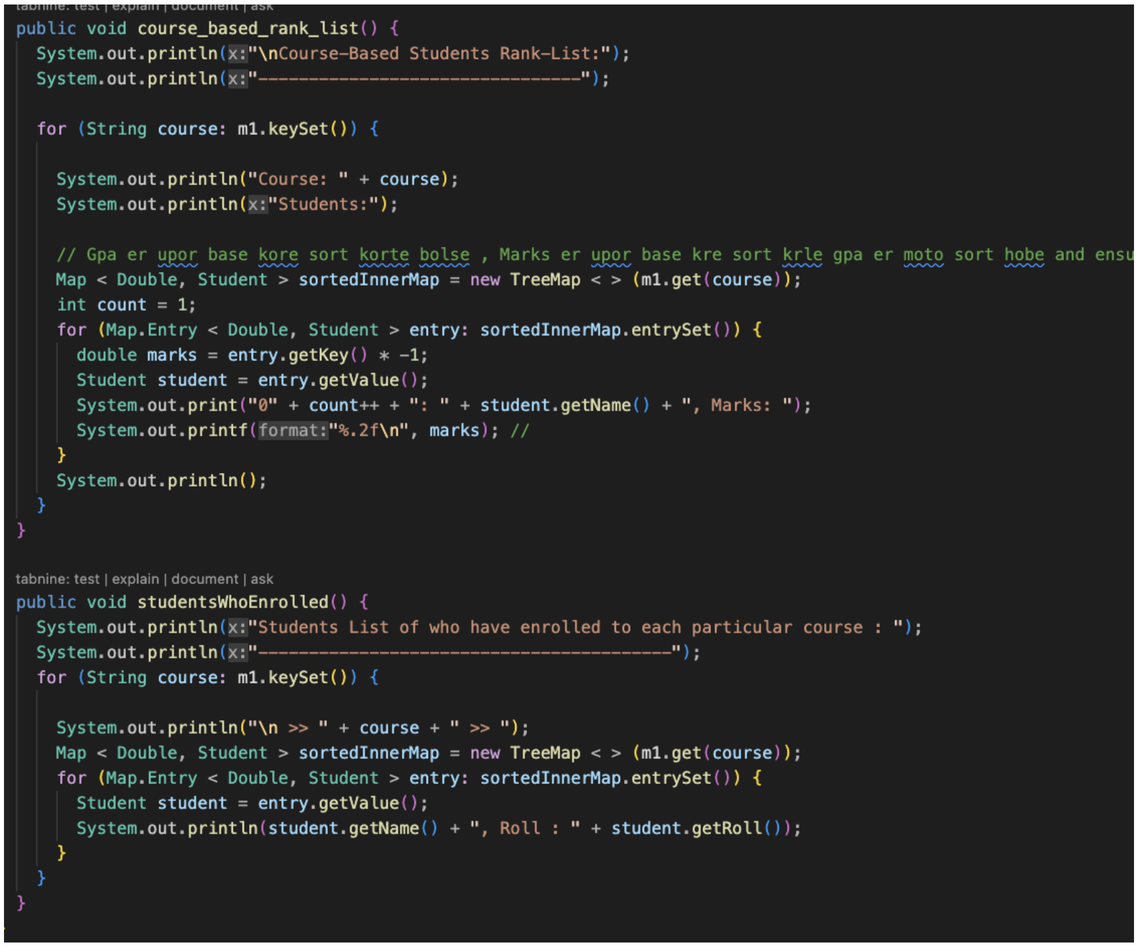
```java
class StudentInfoDataBase {
  List < Student > students = new ArrayList < > ();

  tabnine: test | explain | document | ask
  public void addStudentInfoDataBase(Student student) {
    students.add(student);
  }

  tabnine: test | explain | document | ask
  public void result_according_to_gpa() {
    System.out.println(x:"\nResult According to GPA : ");
    System.out.println(x:"--------------------------------");

    for (int i = 0; i < students.size(); i++) {
      for (int j = i; j < students.size(); j++) {
        if (students.get(j).GPA() > students.get(i).GPA() || students.get(j).GPA() == students.get(i).GPA() && students.get(j).getTotNum() > students.get(i).getTotNum()) {
          Student temp = students.get(i);
          students.set(i, students.get(j));
          students.set(j, temp);
        }
      }
    }

    int count = 1;
    for (Student student: students) {
      System.out.print("0" + (count++) + " : " + student.getName() + ", GPA: ");
      System.out.printf(format:"%.2f\n", student.GPA());
    }
  }

  tabnine: test | explain | document | ask
  public void printStudentsTakenCourses() {
    for (Student student: students) {
      student.printStudentsInfo();
    }
    System.out.println();
  }
}
```

Ensuring if GPA is equal then the sorting will be based on total marks.



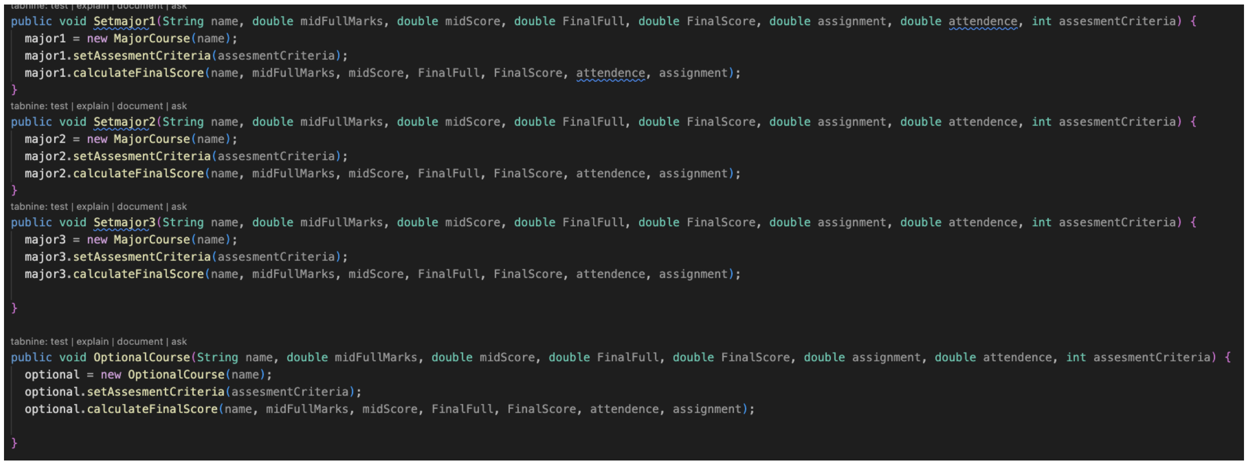
```java
if (students.get(j).GPA() > students.get(i).GPA() || students.get(j).GPA() == students.get(i).GPA() && students.get(j).getTotNum() > students.get(i).getTotNum()) {
```

05/ For Each course, <courses < Mark, Student>> were kept inside a Map of Map. To rank the student for each course sorting on map was done based on the marks got .

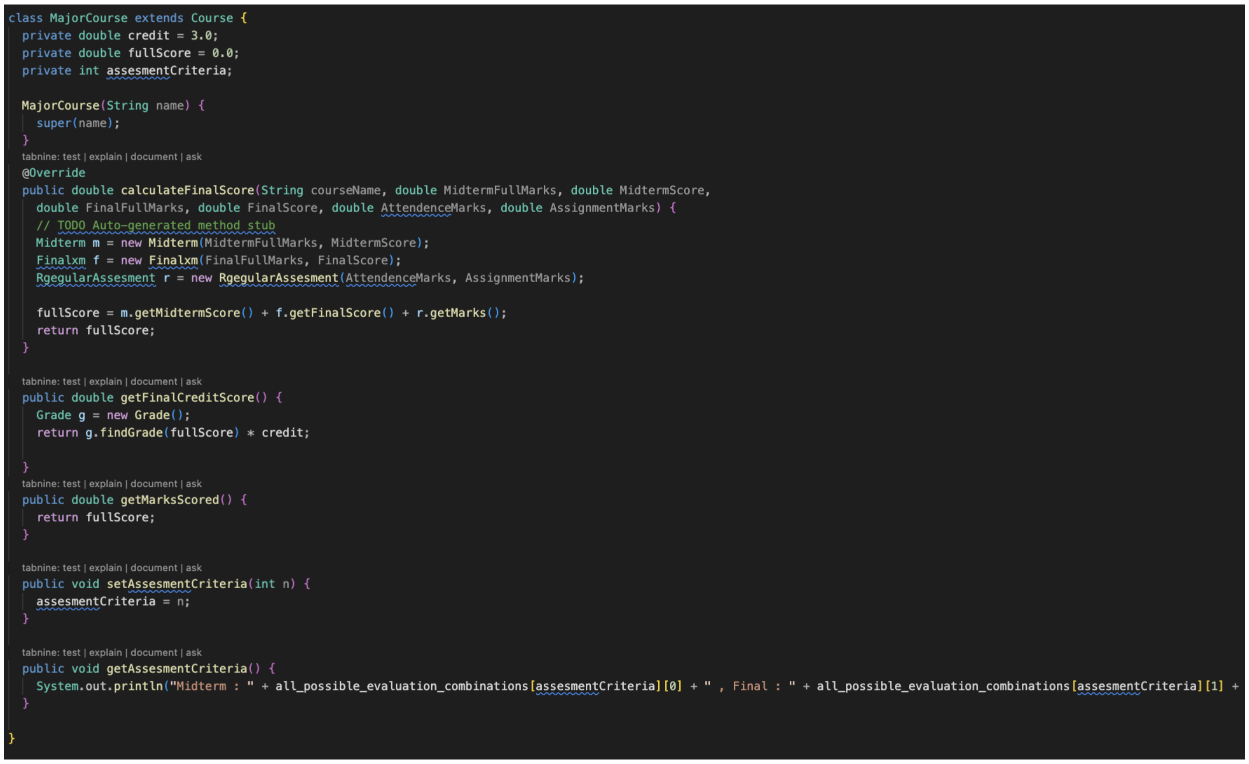


```
tabnine: test | explain | document | ask
public void course_based_rank_list() {
  System.out.println(x:"\nCourse-Based Students Rank-List:");
  System.out.println(x:"--------------------------------");

  for (String course: m1.keySet()) {

    System.out.println("Course: " + course);
    System.out.println(x:"Students:");

    // Gpa er upor base kore sort korte bolse , Marks er upor base kre sort krle gpa er moto sort hobe and ensu
    Map < Double, Student > sortedInnerMap = new TreeMap < > (m1.get(course));
    int count = 1;
    for (Map.Entry < Double, Student > entry: sortedInnerMap.entrySet()) {
      double marks = entry.getKey() * -1;
      Student student = entry.getValue();
      System.out.print("0" + count++ + ": " + student.getName() + ", Marks: ");
      System.out.printf(format:"%.2f\n", marks); //
    }
    System.out.println();
  }
}

tabnine: test | explain | document | ask
public void studentsWhoEnrolled() {
  System.out.println(x:"Students List of who have enrolled to each particular course : ");
  System.out.println(x:"----------------------------------------");
  for (String course: m1.keySet()) {

    System.out.println("\n >> " + course + " >> ");
    Map < Double, Student > sortedInnerMap = new TreeMap < > (m1.get(course));
    for (Map.Entry < Double, Student > entry: sortedInnerMap.entrySet()) {
      Student student = entry.getValue();
      System.out.println(student.getName() + ", Roll : " + student.getRoll());
    }
  }
}
```

06/ The way handled marks that each student got is that I created 4 course Object individual marks on them .

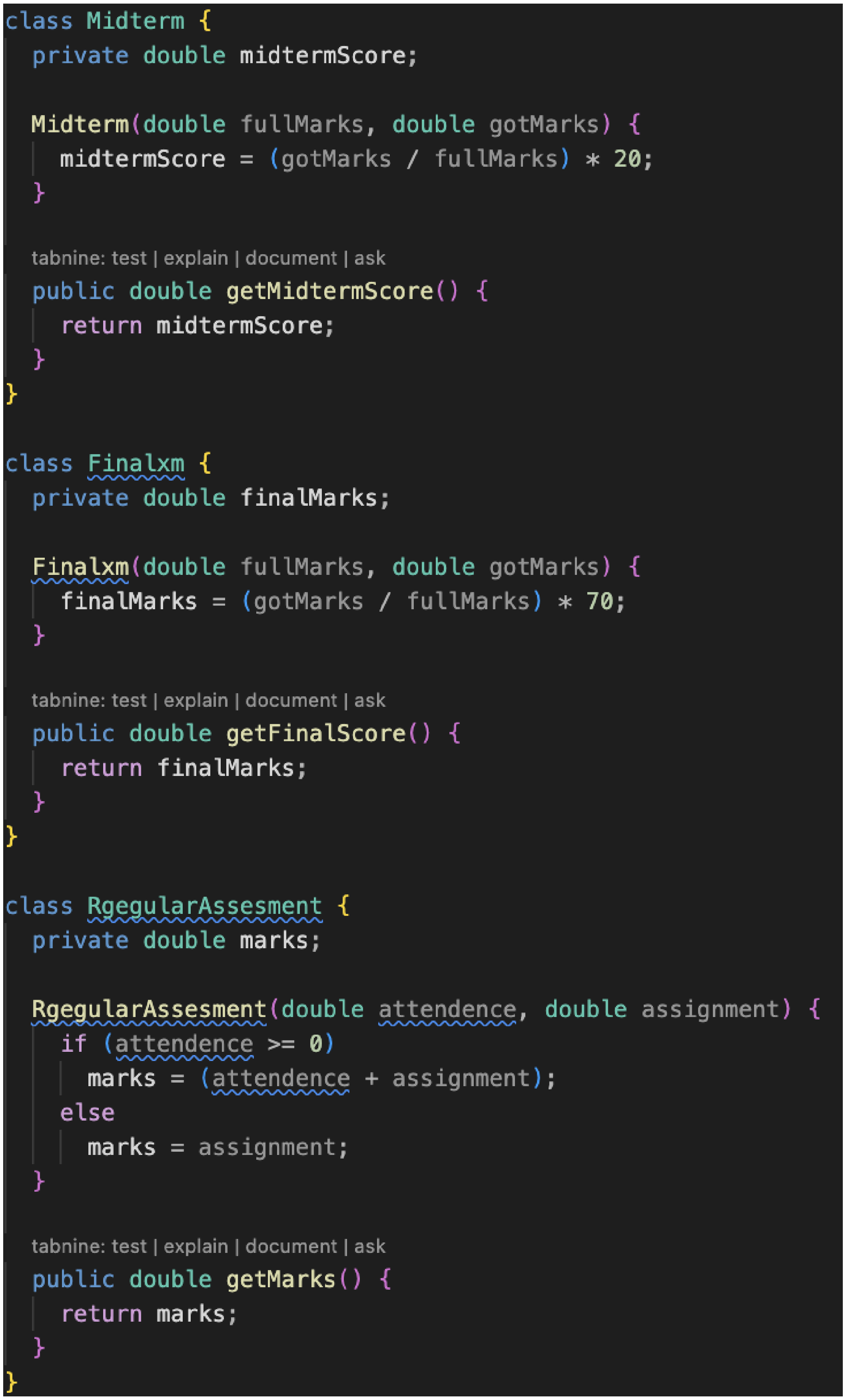


```
tabnine: test | explain | document | ask
public void Setmajor1(String name, double midFullMarks, double midScore, double FinalFull, double FinalScore, double assignment, double attendence, int assesmentCriteria) {
  major1 = new MajorCourse(name);
  major1.setAssesmentCriteria(assesmentCriteria);
  major1.calculateFinalScore(name, midFullMarks, midScore, FinalFull, FinalScore, attendence, assignment);
}
tabnine: test | explain | document | ask
public void Setmajor2(String name, double midFullMarks, double midScore, double FinalFull, double FinalScore, double assignment, double attendence, int assesmentCriteria) {
  major2 = new MajorCourse(name);
  major2.setAssesmentCriteria(assesmentCriteria);
  major2.calculateFinalScore(name, midFullMarks, midScore, FinalFull, FinalScore, attendence, assignment);
}
tabnine: test | explain | document | ask
public void Setmajor3(String name, double midFullMarks, double midScore, double FinalFull, double FinalScore, double assignment, double attendence, int assesmentCriteria) {
  major3 = new MajorCourse(name);
  major3.setAssesmentCriteria(assesmentCriteria);
  major3.calculateFinalScore(name, midFullMarks, midScore, FinalFull, FinalScore, attendence, assignment);

}

tabnine: test | explain | document | ask
public void OptionalCourse(String name, double midFullMarks, double midScore, double FinalFull, double FinalScore, double assignment, double attendence, int assesmentCriteria) {
  optional = new OptionalCourse(name);
  optional.setAssesmentCriteria(assesmentCriteria);
  optional.calculateFinalScore(name, midFullMarks, midScore, FinalFull, FinalScore, attendence, assignment);

}
```



```java
class MajorCourse extends Course {
  private double credit = 3.0;
  private double fullScore = 0.0;
  private int assesmentCriteria;

  MajorCourse(String name) {
    super(name);
  }
  tabnine: test | explain | document | ask
  @Override
  public double calculateFinalScore(String courseName, double MidtermFullMarks, double MidtermScore,
    double FinalFullMarks, double FinalScore, double AttendenceMarks, double AssignmentMarks) {
    // TODO Auto-generated method stub
    Midterm m = new Midterm(MidtermFullMarks, MidtermScore);
    Finalxm f = new Finalxm(FinalFullMarks, FinalScore);
    RgegularAssesment r = new RgegularAssesment(AttendenceMarks, AssignmentMarks);

    fullScore = m.getMidtermScore() + f.getFinalScore() + r.getMarks();
    return fullScore;
  }

  tabnine: test | explain | document | ask
  public double getFinalCreditScore() {
    Grade g = new Grade();
    return g.findGrade(fullScore) * credit;

  }
  tabnine: test | explain | document | ask
  public double getMarksScored() {
    return fullScore;
  }

  tabnine: test | explain | document | ask
  public void setAssesmentCriteria(int n) {
    assesmentCriteria = n;
  }

  tabnine: test | explain | document | ask
  public void getAssesmentCriteria() {
    System.out.println("Midterm : " + all_possible_evaluation_combinations[assesmentCriteria][0] + " , Final : " + all_possible_evaluation_combinations[assesmentCriteria][1] +
  }

}
```

07/ Again each course Object had three objects in each of them , mid, regular assessment, final exam . Which individual would calculate their number as the evaluation criteria could differ .



```
class Midterm {
  private double midtermScore;

  Midterm(double fullMarks, double gotMarks) {
    midtermScore = (gotMarks / fullMarks) * 20;
  }

  tabnine: test | explain | document | ask
  public double getMidtermScore() {
    return midtermScore;
  }
}

class Finalxm {
  private double finalMarks;

  Finalxm(double fullMarks, double gotMarks) {
    finalMarks = (gotMarks / fullMarks) * 70;
  }

  tabnine: test | explain | document | ask
  public double getFinalScore() {
    return finalMarks;
  }
}

class RgegularAssesment {
  private double marks;

  RgegularAssesment(double attendence, double assignment) {
    if (attendence >= 0)
      marks = (attendence + assignment);
    else
      marks = assignment;
  }

  tabnine: test | explain | document | ask
  public double getMarks() {
    return marks;
  }
}
```

Now, the calculation was made on percentage based , as for each exam percentage was calculated first and then mid was converted to 20 , final to 70 and regular to 10 . Regulars could have attendance on it.

In summary, the design and code exhibit a strong adherence to object-oriented principles such as encapsulation, abstraction, inheritance, polymorphism, composition, modularity, and flexibility. Also all the functions and functionalities are performed as mentioned in the question. These features contribute to the scalability, maintainability, and extensibility of the student database and evaluation system.