**UML Class Diagrams for all classes:**

Course class:

|  |  |  |  |
| --- | --- | --- | --- |
| Course | PrerequisiteCourse | Student | |
| - name : String  - number : int  - section : String  - semester : String  - year : int | - prerequisite : Course[]  - numOfPrequisites : int  - maxPrerequisites : int | - StudentID : int  - studentName : String  - email : String  - zip\_code : String  -lastGeneratedStudentID: int | |
| + Course()  + Course(n : String, num : int,  sect : String, sem : String, y : int )  + setCourseName(n : String) : void  + setNumber(num : int) : void  + setSection(sect : String) : void  + setSemester(sem : String) : void  + setYear(y : int) : void  + getCourseName() : String  + getNumber() : int  + getSection() : String  + getSemester() : String  + getYear() : int  + equals (obj : Object) : boolean  + toString() : String | + PrerequisiteCourse (n : String, numPre : int, sect : String, sem : String, y : int, maxPrerequisites: int)  + addPrerequisiteCourse(prerequisites : Course[]) : void  + getPrerequisiteCourses() : Course[]  + toString() : String | + Student()  + Student(studentName: String, email : String, zip : String)  + setStudentName(studentName : String) : void  + setEmail(email : String) : void  + setZip\_code(zip : String) : void  + getStudentID() : int  + getStudentName() : String  + getEmail() : String  + getZip\_code() : String  + isEmailValid() : boolean  + isZipCodeValid() : boolean  + equals (obj : Object) : boolean  + toString() : String | |
| StudentGradeRecord | | CourseGradeBook | |
| - grades: double []  - student: Student | | - caNames : String[]  - caWeights : double[]  - course : Course  - gradeList: StudentGradesRecord[]  - gradesRecordCount: int | |
| + StudentGradeRecord()  + StudentGradeRecord(grades: double[], student: Student)  + computeFinalGrade(assessmentWeights: double[]): double  + getGrade(k : int) : double  + setGrade(k : int, grd : double) : void  + getAllGrades() : double[]  + setAllGrades(allGrades : double[]) : void  + getStudent() : Student  + setStudent(student : Student) : void  + getNumberOfAssessments() : int  + computeLetterGrade(fin : double) : char  + equals (obj : Object) : boolean  + toString() : String | | + CourseGradeBook(course : Course, caNames : String[], caWeights : double[])  + addGradeRecord(student : Student, grades : double[])  + findArraySum(numbers : double[]) : double  + findArrayMaximum(numbers : double[]) : double  + findArrayMinimum(numbers : double[]) : double  + findArrayAverage(numbers: double[]) : double  + findArrayStandardDev(numbers: double[]) : double  + computeTotalWeight() : double  + getFinalsArray() : double[]  + getAssessmentArray(k : int) : double[]  + findMaxAssessment(k : int) : double  + findMinAssessment(k : int) : double  + findAvgAssessment(k : int) : double  + findStdevAssessment(k: int) : double  + toStringMaxAssessments() : String  + toStringMinAssessments() : String  + toStringAvgAssessments() : String  + toStringStdevAssessments() : String | |

**Pseudocode:**

**Course Class:**

**a) Data Members**

courseName: String

number: int

section: String

semester: String

year: Int

**b) Methods**

Course()

Course(n : String, num : int, sect : String, sem : String, y : int)

this.courseName = n

this.number = num

this.section = sect

this.semester = sem

this.year = y

setCourseName(n : String)

this.name = n

setNumber(num : int)

this.number = num

setSection(sect : String)

this.section = sect

setSemester(sem : String)

this.semester = sem

setYear(y : int)

this.year = y

getCourseName()

Return this.Coursename

getNumber()

Return this.number

getSection()

Return this.section

getSemester()

Return this.semester

getYear()

Return this.year

equals(obj)

equals(obj : Object)

If Object = obj

Return true

If obj is not equal to type of Object

Return false

toString(): Return data

**PrerequisiteCourse Class extends Course:**

**a) Data Members**

Prerequisite: Course[]

numOfPrerequisites: Int

maxPrerequisites: Int

**b) Methods**

PrerequisiteCourse(n: String, num: String, sect: String, sem: String, y: int, maxPrerequisites: int)

super(n, num, sect, sem, y)

this.maxPrerequisites = maxPrerequisites

this.numOfPrerequisites = 0

this.prerequisites = new Course[maxPrerequisites]

addPrerequisiteCourse(prerequisites: Course[]): void

if numOfPrerequisites < maxPrerequisites

prerequisites[numOfPrerequisites] = course

numOfPrerequisites++

else

Print Error

getPrerequisiteCourses(): Course[]

Return prerequisites array

toString(): String

for i = 0 to numOfPrerequisites - 1

Return data

**Student Class:**

**a) Data members**

StudentID: Int

studentName: String

email: String

zip\_code: String

lastGeneratedStudentID: Int = 30000000

**b) Methods**

Method Student():

studentID = 0

studentName = ""

email = ""

zip\_code = ""

Student(studentName: String, email: String, zip\_code: String)

studentID = generateStudentId()

this.studentName = studentName

this.email = email

this.zip\_code = zip

setStudentName(studentName: String):

this.studentName = studentName

setEmail(email: String):

this.email = email

setZipCode(zip: String):

this.zipCode = zip

getStudentID():

Return studentID

getStudentName():

Return studentName

getEmail():

Return email

getZipCode():

Return zipCode

generateStudentId():

Increment (++) lastGeneratedStudentID

Return lastGeneratedStudentID

isEmailValid():

atFound = false

dotFound = false

Array:

(For i from 0 to length of email – 1)

If email[i] equals @ and not atFound

atFound = true

Else

if email[i] equals dot and atFound and i is not length of email -1:

dotFound = true

Return atFound and dotFound

isZipCodeValid():

Return zip\_code matches

equals(obj : Object)

If Object = obj

Return true

If obj is not equal to type of Object

Return false

toString():

Return data

**StudentGradeRecord Class:**

**a) Data Members**

grades: double[] meaning it is an array.

student: Student

**b) methods**

StudentGradeRecord(grades: double[], student: Student)

this.grades = grades

this.student = Student

computeFinalGrade(assessmentWeights: double[]):

finalGrade = 0

Return finalGrade

getGrade(k: int): double

Return grades[k]

setGrade(k: int, grd: double)

this.grades[k] = grd

getAllGrades(): double[]

Return allGrades

setAllGrades(allGrades: double[])

If length of allGrades is equal from grades

Copy allGrades to grades

Else

Print Error

getStudent(): Student

Return student

setStudent(student: Student)

this.student = Student

getNumberOfAssessments(): int

Return grades length

computeLetterGrades(fin: double): char

If fin >= 90, return 'A'

Else if fin >= 80, return 'B'

Else if fin >= 70, return 'C'

Else if fin >= 60, return 'D'

Else return 'F'

equals(obj: Object): Boolean

If Object = obj

Return true

If obj is not equal to type of Object

Return false

toString(): String

Return data

**CourseGradeBook Class:**

**a) Data Members**

caNames: String[] (array)

caWeights: double [] (array)

course: Course

gradeList: StudentGradeRecord[] (array)

gradesRecordCount: int

**b) methods** **CourseGradeBook:**

CourseGradeBook(course : Course, caNames : String[], caWeights: double[])

this.course = Course

this.caNames = caNames;

this.caWeights = caWeights;

if length of caNames is not equal to length of caWeights

Print Error

addGradeRecord(student : Student, grades: Grades)

if length of grades is not equal to length of caNames

Print error

if gradesRecordCount is equal to length of gradeList

create new object

findArraySum(numbers : double[]) : double

sum = 0

sum = sum + number

return sum

findArrayMaximum(numbers : double[]) : double

max = numbers[0]

if number > max:

max = number

return max

findArrayMinimum(numbers : double[]) : double

min = numbers[0]

if number < min:

min = number

return min

findArrayAverage(numbers : double[]) : double

sum = findArraySum(numbers)

average = sum / length of numbers

return average

findArrayStandardDev(numbers : double[]) :double

average = findArrayAverage(numbers)

squaredSum = 0

diff = number - average

squaredSumDiff= squaredSum + (diff \* diff)

result = squaredSumDiff / length of numbers

standardDev = square root of result

return standardDev

computeTotalWeight() : double

totalWeight = findArraySum(caWeights)

return totalWeight

getFinalsArray() : double[]

finalsArray = new array of double that has the same length as gradesRecordCount

Array:

(for i from 0 to gradesRecordCount – 1)

finalsArray[i] = findArraySum(gradeList[i].grades \* caWeights)

return finalsArray

getAssessmentArray(k : int) : double[]

assessmentArray = new array of double that has the same length as gradesRecordCount

Array:

(for i in 0 to gradesRecordCount – 1)

assessmentArray[i] = gradeList[i].grades[k]

return assessmentArray

findMaxAssessment(k : int) : double

max = gradeList[0].grades[k]

if studentRecord.grades[k] > max

max = studentRecord.grades[k]

return max

findMinAssessment(k : int) : double

min = gradeList[0].grades[k]

if studentRecord.grades[k] < min:

min = studentRecord.grades[k]

return min

findAvgAssessment(k : int) : double

assessmentArray = getAssessmentArray(k)

avgAssessment = findArrayAverage(assessmentArray)

return avgAssessment

findStdevAssessment(k : int) : double

assessmentArray = getAssessmentArray(k)

stdevAssessment = findArrayStandardDev(assessmentArray)

return stdevAssessment

toStringMaxAssessments() : String

Array:

(for k from 0 to length of caNames – 1)

return result

toStringMinAssessments() : String

Array:

(for k from 0 to length of caNames – 1)

return result

toStringAvgAssessments() : String

Array:

(for k from 0 to length of caNames – 1)

return result

toStringStdevAssessments() : String

Array:

(for k from 0 to length of caNames – 1)

return result