StudentId
- depCode: String = "1501" {readOnly}
- student: Student
- studentId: String
+ StudentId(student: Student)
+ StudentId(student: Student, studentId: studentId)
+ setStudentId()
+ getStudentId(): String
- getYearString(): String
- getRegistrationString(): String
+ toString(): String
Advisor
- name: String
+ Advisor(name: String)
+ approveCourse(courseSection:CourseSection, student: Student)
+ getName(): String
+ setName(name: String)
Schedule
+ DAYS: int = 5 {readOnly}
+ HOURS: int = 8 {readOnly}
- program: CourseSection[7][8]
- student: Student
+ Schedule (student: Student)
+ addToProgram(courseSection: CourseSection)
+ getCollidedHours(courseSection: CourseSection): List<CourseSection>)
+ isCollision(courseSection: CourseSection): boolean
+ getProgram(): CourseSection[][]
+ setProgram(program: CourseSection[][])
CourseSection
- course: Course
- registrationSystem: RegistrationSystem
- full: boolean
- sectionHour: int
- students: List<Student>
- courseProgram:boolean[7][8]
+ CourseSection(course:Course)
+ setCourseProgram()
+ collideWithSameSemester(randomHour: int, randomDay: int): boolean
+ addStudent(student:Student)
+ getQuota(): int
+ isFull(): boolean
+ getCourse(): Course
+ getCourseSectionCode(): String
+ setFull(full: boolean)
+ getCollisionStatistics(): int
+ setCollisionStatistics(collisionStatistics: int): int
+ getSectionHour(): int
+ setSectionHour()
+ getStudents(): List<Student>
+ setCourse(course: Course)
+ getCourseProgram(): boolean[][]
RegistrationSystem
- registrationSystem: RegistrationSystem
- isRegenerate: boolean
- semester: Semester
- totalStudents: int[]
- students: List<Student>
- advisors: List<Advisor>
- courses: List<Course>
- mandatoryCourses: List<MandatoryCourses>
- nonTechElectiveCourses: List<NonTechnicalUniversityElectiveCourse>
- techElectiveCourses: List<TechnicalElectiveCourse>
- facultyElectiveCourses: List<FacultyTechnicalElectiveCourse>
- passProbability : double
- advisorCount: int
- nonTechElectiveSemesters: List<Integer>
- techElectiveSemesters: List<Integer>
- facultyElectiveSemesters: List<Integer>
- statisticsBuffer: String
- RegistrationSystem()
+ getInstance(): RegistrationSystem
+ startTheSimulation()
- regenerateCheck()
- readStudents()
- statisticsOutput()
- registrationProcessOutput()
- printRegistrationProcess()
- printMandatoryStatistics()
- printFinalProjectStatistics()
- printElectiveStatistics()
- printStatistics()
- initializeAdvisors()
- initStudentsByCount(currentYear: int, numOfStudents: int)
- initializeStudents()
- appointAdvisors()
- addPastNTEs(student: Student)
- addPastFTEs(student: Student)
- addPastTEs(student: Student)
- addPastMandatories(student: Student)
- addPastSummerMandatories(student: Student)
- addPastCourses()
- requestCourses()
+ offeredTECount(student: Student): int
+ offeredFTECount(student: Student): int
+ offeredNTECount(student: Student): int
+ getOfferedMandatories(student: Student): List<CourseSection>
+ getOfferedElectives(student: Student): List<CourseSection>
-readMandatoryCourses(input: JSONObject)
-readFinalProjectCourses(input: JSONObject)
-readTechs(input: JSONObject)
-readNonTechs(input: JSONObject)
-readFacTechs(input: JSONObject)
-readGeneralInformation(input: JSONObject)
- readInput()
+ isThereEmptyNonTechSection(): boolean
+ isThereEmptyTechSection(): boolean
+ isThereEmptyFacTechSection(): boolean
+ setSemester(semester: String)
- findCourse(courseCode: String):Course
+ getSemester(): String
+ getPassProbability(): double
+ setPassProbability(passProbability: double)
+ getStudentCount(): int
+ setStudentCount(studentCount: int)
+ getAdvisorCount(): int
+ setAdvisorCount(advisorCount: int)
+ getCourses(): List<Course>
+ getTotalStudents(): int[]
+ getStudents(): List<Student>
+ getAdvisors(): List<Advisor>
+ getMandatoryCourses(): List<MandatoryCourse>
+ getNontechElectiveCourses(): List<NonTechnicalUniversityElectiveCourse>
+ getTechElectiveCourses(): List<TechnicalElectiveCourse>
+ getFacultyElectiveCourses(): List<FacultyTechnicalElectiveCourse>
+ getNonTechElectiveSemesters(): List<Integer>
+ getTectElectiveSemesters(): List<Integer>
+ getFacTechElectiveSemesters(): List<Integer>
+ getStatisticsBuffer(): String
Student
- name: String
- studentId: StudentId
- registrationOrder: int
- currentYear: int
- semesterNumber: int
- advisor: Advisor
- schedule: Schedule
- transcript: Transcript
- registrationSystem: RegistrationSystem
- executionTrace: StringBuilder
+ Student:(name: String, studentId: String, registrationSystem: RegistrationSystem, semesterNumber: int))
+ Student:(name: String, currentYear: int, registrationOrder: int, registrationSystem: RegistrationSystem)
+ getNumOfPastElectives(semesterNums: List<Integer>): int
+ addToCurrentCourses(courseSection: CourseSection)
+ requestCourseSection(courseSection: CourseSection)
+ requestCourses()
+ requestMandatoryCourses()
+ requestElectiveCourses()
+ getExecutionTrace(): StringBuilder
+ setExecutionTrace(executionTrace: StringBuilder)
+ getName(): String
+ getRegistrationOrder(): int
+ getStudentId(): StudentId
+ getCurrentYear(): int
+ getAdvisor(): Advisor
+ setAdvisor(advisor: Advisor): void
+ getSemesterNumber(): int
+ getSchedule(): Schedule
+ setSchedule(schedule: Schedule): void
+ getTranscirpt(): Transcript
+ toString(): String
Transcript
- student: Student
- currentCourses: List<Course>
- grades: <Grade>
+ Transcript(student: Student)
+ addPassedCourse(course: Course)
+ getCompletedCredits(): int
+ getPassedCourses(): List<Course>
+ getTakenCourses(): List<Course>
+ hasPassedCourse(course: Course): booelan
+ hasPassedCourses(course: List<Course>): booelan
+ addPassedCourse(course: Course)
+ addFailedCourse(course: Course)
+ getStudent(): Student
+ setStudent(student: Student)
+ getGrades(): List<Grades>
+ setGrades(grades: List<Grades>)
+ getCurrentCourses(): List<Course>
+ setCurrentCourses(currentCourses: List<Course>)
+ toString(): String
Grade
- intGrade: int
- course: Course
+ Grade(intGrade: int, course: Course)
+ isPassed(): booelan
+ getLetterGrade: String
+ getIntGrade(): int
+ getCourse(): Course
+ setCourse(course: Course)
Course {abstract}
- courseCode: String
- quota: int
- credits: int
- theoretical: int
- practical: int
- courseSection: CourseSection
- registrationSystem: RegistrationSystem
- nonRegisteredCollisiion: Set<Student>
- nonRegisteredQuota: Set<Student>
# Course(courseCode: String, quota: int, credits: int, theoretical: int, practical: int)
+ isEligiblePastCourse(student: Student): boolean
+ onRequested(student: Student): booelan
+ getSectionHours(): int
+ setSectionHours(theoretical: int, practical: int)
+ getCourseCode(): String
+ getQuota(): int
+ getCredits(): int
+ getTheoretical(): int
+ getPractical(): int
+ getRegistrationSystem(): RegistrationSystem
+ getCourseSection(): int
+ setCourseSection(courseSection: CourseSection)
+ setNonRegisteredCollision(nonRegisteredCollision: Set<Student>)
+ getNonRegisteredCollision(): List<Student>
+ setNonRegisteredQuota(nonRegisteredQuota: Set<Student>)
+ getNonRegisteredQuota(): List<Student>
+ toString(): String
ElectiveCourse {abstract}
- semesters: List<Integer>
+ ElectiveCourse(courseCode: String, quota: int, credits : int, theoretical: int, practical: int, semesters: List<Integer>
+ onRequested(student: Student): boolean
+ offeredElectiveCount(student: Student): int
+ onRequested(student: Student): boolean
+ whenRejected(student: Student): {abstract}
+ getRandomElective(): Course {abstract}
+ getSemesters(): List<Integer>
+ setSemesters(semesters: List<Integer>)
MandatoryCourse
- semesterNumber: float
- semester: Semester
- nonRegisteredPrereq: Set<Student>
- preRequisties: List<Course>
+ MandatoryCourse(courseCode: String, quota: int, credits : int, theretical: int, practical: int, preRequesities: List<Course>)
+ isEligiblePastCourse(student: Student): boolean
+ isOfferableForStudent(student: Student): booelan
+ onRequested(student: Student): boolean
+ setSemesterNumber(semesters: float)
+ getSemesterNumber(): float
+ setSemester(): Semester
+ getPreRequesities(): List<Course>
+ setPreRequesities(preRequisites: List<Course>)
+ getNonRegisteredPrereq(): List<Student>
+ setNonRegisteredPrereq(nonRegisteredPrereq: Set<Student>)
FacultyTechnicalElecetiveCourse
+ FacultyTechnicalElectiveCourse(courseCode: String, quota: int, credits : int, theretical: int, practical: int, semesters: List<Integer>)
+ whenRejected(student: Student)
+ getRandomElective(): Course
+ toString(): String
NonTechnicalUniversityElectiveCourse
+ NonTechnicalUnversityElectiveCourse(courseCode: String, quota: int, credits : int, theretical: int, practical: int, semesters: List<Integer>)
+ whenRejected(student: Student)
+ getRandomElective(): Course
+ toString(): String
TechnicalElectiveCourse
- requiredCredits: int
- preRequisites: List<Course>
- nonRegisteredStudents: Set<Student>
+ TechnicalElectiveCourse(courseCode: String, quota: int, credits : int, theretical: int, practical: int, semesters: List<Integer>, requiredCredits: preRequisites: List<Course>)
+ isEligiblePastCourse(student: Student): boolean
+ whenRejected(student: Student)
+ getRandomElective(): Course
+ onRequested(student: Student): boolean
+ checkCreditCondition(student: Student): boolean
+ getRequiredCredits(): int
+ setRequiredCredits(requiredCredits: int)
+ getPreRequesities(): List<Course>
+ setPreRequesities(preRequisites: List<Course>)
+ getNonRegisteredStudents(): Set<Student>
+ setNonRegisteredStudents(unregisteredStudents: Set<Student)
+ toString(): String
FinalProjectMandatoryCourse
- requiredCredits: int
- nonRegisteredCredit: Set<Student>
+ FinalProjectMandatoryCourseCourse(courseCode: String, semester: float, quota: int, credits : int, theretical: int, practical: int, preRequesities: List<Course>, requiredCredits: int)
+ isEligiblePastCourse(student: Student): boolean
+ onRequested(student: Student): boolean
+ checkReqCredits(student: Student): boolean
+ getRequiredCredits(): int
+ setRequiredCredits(requiredCredits: int)
+ getNonRegisteredCredit(): Set<Student>
+ setNonRegisteredCredit(nonRequesedCredtis: Set<Student>)
+ toString(): String