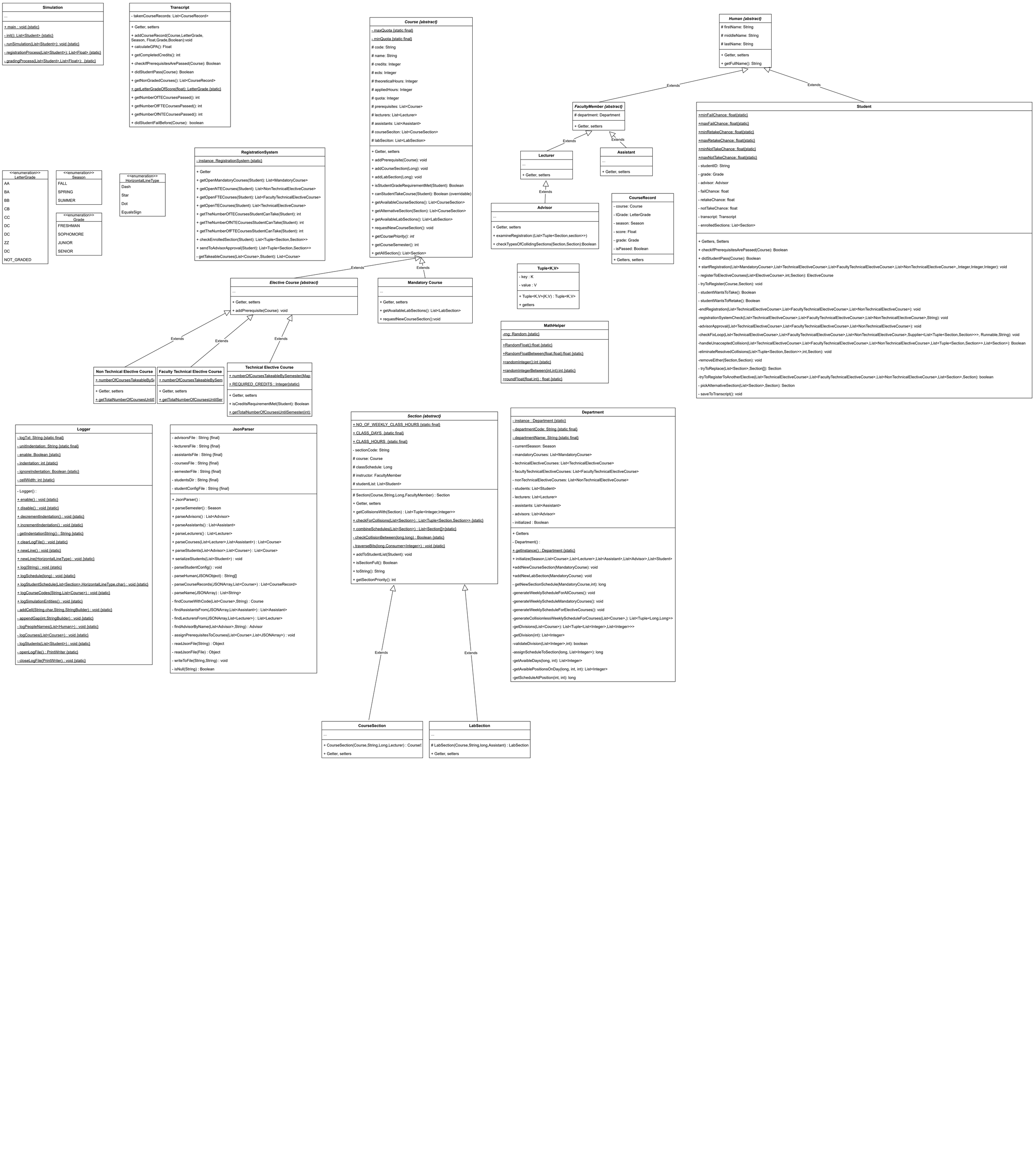
Simulation
...
+ main : void {static}
- init(): List<Student> {static}
- runSimulation(List<Student>): void {static}
- registrationProcess(List<Student>): List<Float> {static}
- gradingProcess(List<Student>,List<Float>): {static}
Transcript
- takenCourseRecords: List<CourseRecord>
+ Getter, setters
+ addCourseRecord(Course,LetterGrade, Season, Float,Grade,Boolean):void
+ calculateGPA(): Float
+ getCompletedCredits(): int
+ checkIfPrerequisitesArePassed(Course): Boolean
+ didStudentPass(Course): Boolean
+ getNonGradedCourses(): List<CourseRecord>
+ getLetterGradeOfScore(float): LetterGrade {static}
+ getNumberOfTECoursesPassed(): int
+ getNumberOfFTECoursesPassed(): int
+ getNumberOfNTECoursesPassed(): int
+ didStudentFailBefore(Course): boolean
<<enumeration>> LetterGrade
AA
BA
BB
CB
CC
DC
DC
ZZ
DC
NOT_GRADED
<<enumeration>> Season
FALL
SPRING
SUMMER
<<enumeration>> Grade
FRESHMAN
SOPHOMORE
JUNIOR
SENIOR
<<enumeration>> HorizontalLineType
Dash
Star
Dot
EqualsSign
RegistrationSystem
- instance: RegistrationSystem {static}
+ Getter
+ getOpenMandatoryCourses(Student): List<MandatoryCourse>
+ getOpenNTECourses(Student): List<NonTechnicalElectiveCourse>
+ getOpenFTECourses(Student): List<FacultyTechnicalElectiveCourse>
+ getOpenTECourses(Student): List<TechnicalElectiveCourse>
+ getTheNumberOfTECoursesStudentCanTake(Student): int
+ getTheNumberOfNTECoursesStudentCanTake(Student): int
+ getTheNumberOfFTECoursesStudentCanTake(Student): int
+ checkEnrolledSection(Student): List<Tuple<Section,Section>>
+ sendToAdvisorApproval(Student): List<Tuple<Section,Section>>
- getTakeableCourses(List<Course>,Student): List<Course>
Course {abstract}
- maxQuota {static final}
- minQuota {static final}
code: String
name: String
credits: Integer
ects: Integer
theoreticalHours: Integer
appliedHours: Integer
quota: Integer
prerequisites: List<Course>
lecturers: List<Lecturer>
assistants: List<Assistant>
courseSeciton: List<CourseSection>
labSeciton: List<LabSection>
+ Getter, setters
+ addPrerequisite(Course): void
+ addCourseSection(Long): void
+ addLabSection(Long): void
+ isStudentGradeRequirementMet(Student): Boolean
+ canStudentTakeCourse(Student): Boolean (overridable)
+ getAvailableCourseSections(): List<CourseSection>
+ getAlternativeSection(Section): List<CourseSection>
+ getAvailableLabSections(): List<LabSection>
+ requestNewCourseSection(): void
+ getCoursePriority(): int
+ getCourseSemester(): int
+ getAllSection(): List<Section>
Extends
Extends
Elective Course {abstract}
...
+ Getter, setters
+ addPrerequisite(Course): void
Mandatory Course
...
+ Getter, setters
+ getAvailableLabSections(): List<LabSection>
+ requestNewCourseSection():void
Extends
Extends
Extends
Non Technical Elective Course
+ numberOfCoursesTakeableBySe
+ Getter, setters
+ getTotalNumberOfCoursesUntilS
Faculty Technical Elective Course
+ numberOfCoursesTakeableBySem
+ Getter, setters
+ getTotalNumberOfCoursesUntilSer
Technical Elective Course
+ numberOfCoursesTakeableBySemester(Map
+ REQUIRED_CREDITS : Integer{static}
+ Getter, setters
+ isCreditsRequirementMet(Student): Boolean
+ getTotalNumberOfCoursesUntilSemester(int):
Human {abstract}
firstName: String
middleName: String
lastName: String
+ Getter, setters
+ getFullName(): String
Extends
Extends
FacultyMember {abstract}
department: Department
+ Getter, setters
Extends
Extends
Lecturer
...
+ Getter, setters
Assistant
...
+ Getter, setters
Extends
Extends
Advisor
...
+ Getter, setters
+ examineRegistration:(List<Tuple<Section,section>>)
+ checkTypesOfCollidingSections(Section,Section):Boolean
Student
+minFailChance: float{static}
+maxFailChance: float{static}
+minRetakeChance: float{static}
+maxRetakeChance: float{static}
+minNotTakeChance: float{static}
+maxNotTakeChance: float{static}
- studentID: String
- grade: Grade
- advisor: Advisor
- failChance: float
- retakeChance: float
- notTakeChance: float
- transcript: Transcript
- enrolledSections: List<Section>
+ Getters, Setters
+ checkIfPrerequisitesArePassed(Course): Boolean
+ didStudentPass(Course): Boolean
+ startRegistration(List<MandatoryCourse>,List<TechnicalElectiveCourse>,List<FacultyTechnicalElectiveCourse>,List<NonTechnicalElectiveCourse>,,Integer,Integer,Integer): void
- registerToElectiveCourses(List<ElectiveCourse>,int,Section): ElectiveCourse
- tryToRegister(Course,Section): void
- studentWantsToTake(): Boolean
- studentWantsToRetake(): Boolean
-endRegistration(List<TechnicalElectiveCourse>,List<FacultyTechnicalElectiveCourse>,List<NonTechnicalElectiveCourse>): void
-registrationSystemCheck(List<TechnicalElectiveCourse>,List<FacultyTechnicalElectiveCourse>,List<NonTechnicalElectiveCourse>,String): void
-advisorApproval(List<TechnicalElectiveCourse>,List<FacultyTechnicalElectiveCourse>,List<NonTechnicalElectiveCourse>): void
-checkFixLoop(List<TechnicalElectiveCourse>,List<FacultyTechnicalElectiveCourse>,List<NonTechnicalElectiveCourse>,Supplier<List<Tuple<Section,Section>>>, Runnable,String): void
-handleUnacceptedCollision(List<TechnicalElectiveCourse>,List<FacultyTechnicalElectiveCourse>,List<NonTechnicalElectiveCourse>,List<Tuple<Section,Section>>,List<Section>): Boolean
-eliminateResolvedCollisions(List<Tuple<Section,Section>>,int,Section): void
-removeEither(Section,Section): void
- tryToReplace(List<Section>,Section[]): Section
-tryToRegisterToAnotherElective(List<TechnicalElectiveCourse>,List<FacultyTechnicalElectiveCourse>,List<NonTechnicalElectiveCourse>,List<Section>,Section): boolean
- pickAlternativeSection(List<Section>,Section): Section
- saveToTranscript(): void
CourseRecord
- course: Course
- lGrade: LetterGrade
- season: Season
- score: Float
- grade: Grade
- isPassed: Boolean
+ Getters, setters
Tuple<K,V>
- key : K
- value : V
+ Tuple<K,V>(K,V) : Tuple<K,V>
+ getters
MathHelper
-rng: Random {static}
+RandomFloat():float {static}
+RandomFloatBetween(float,float):float {static}
+randomInteger():int {static}
+randomIntegerBetween(int,int):int {static}
+roundFloat(float,int) : float {static}
Logger
- logTxt: String {static final}
- unitIndentation: String {static final}
- enable: Boolean {static}
- indentation: int {static}
- ignoreIndentation: Boolean {static}
- cellWidth: int {static}
- Logger() :
+ enable() : void {static}
+ disable() : void {static}
+ decrementIndentation() : void {static}
+ incrementIndentation() : void {static}
- getIndentationString() : String {static}
+ clearLogFile() : void {static}
+ newLine() : void {static}
+ newLine(HorizontalLineType) : void {static}
+ log(String) : void {static}
+ logSchedule(long) : void {static}
+ logStudentSchedule(List<Section>,HorizontalLineType,char) : void {static}
+ logCourseCodes(String,List<Course>) : void {static}
+ logSimulationEntities() : void {static}
- addCell(String,char,String,StringBuilder) : void {static}
- appendGap(int,StringBuilder) : void {static}
- logPeopleNames(List<Human>) : void {static}
- logCourses(List<Course>) : void {static}
- logStudents(List<Student>) : void {static}
- openLogFile() : PrintWriter {static}
- closeLogFile(PrintWriter) : void {static}
JsonParser
- advisorsFile : String {final}
- lecturersFile : String {final}
- assistantsFile : String {final}
- coursesFile : String {final}
- semesterFile : String {final}
- studentsDir : String {final}
- studentConfigFile : String {final}
+ JsonParser() :
+ parseSemester() : Season
+ parseAdvisors() : List<Advisor>
+ parseAssistants() : List<Assistant>
+ parseLecturers() : List<Lecturer>
+ parseCourses(List<Lecturer>,List<Assistant>) : List<Course>
+ parseStudents(List<Advisor>,List<Course>) : List<Course>
+ serializeStudents(List<Student>) : void
- parseStudentConfig() : void
- parseHuman(JSONObject) : String[]
- parseCourseRecords(JSONArray,List<Course>) : List<CourseRecord>
- parseName(JSONArray) : List<String>
- findCourseWithCode(List<Course>,String) : Course
- findAssistantsFrom(JSONArray,List<Assistant>) : List<Assistant>
- findLecturersFrom(JSONArray,List<Lecturer>) : List<Lecturer>
- findAdvisorByName(List<Advisor>,String) : Advisor
- assignPrerequisitesToCourses(List<Course>,List<JSONArray>) : void
- readJsonFile(String) : Object
- readJsonFile(File) : Object
- writeToFile(String,String) : void
- isNull(String) : Boolean
Section {abstract}
+ NO_OF_WEEKLY_CLASS_HOURS {static final}
+ CLASS_DAYS {static final}
+ CLASS_HOURS {static final}
- sectionCode: String
course: Course
classSchedule: Long
instructor: FacultyMember
studentList: List<Student>
Section(Course,String,Long,FacultyMember) : Section
+ Getter, setters
+ getCollisionsWith(Section) : List<Tuple<Integer,Integer>>
+ checkForCollisions(List<Section>) : List<Tuple<Section,Section>> {static}
+ combineSchedules(List<Section>) : List<Section[]>{static}
- checkCollisionBetween(long,long) : Boolean {static}
- traverseBits(long,Consumer<Integer>) : void {static}
+ addToStudentList(Student): void
+ isSectionFull(): Boolean
+ toString(): String
+ getSectionPriority(): int
Department
- instance : Department {static}
- departmentCode: String {static final}
- departmentName: String {static final}
- currentSeason: Season
- mandatoryCourses: List<MandatoryCourse>
- technicalElectiveCourses: List<TechnicalElectiveCourse>
- facultyTechnicalElectiveCourses: List<FacultyTechnicalElectiveCourse>
- nonTechnicalElectiveCourses: List<NonTechnicalElectiveCourse>
- students: List<Student>
- lecturers: List<Lecturer>
- assistants: List<Assistant>
- advisors: List<Advisor>
- initialized : Boolean
+ Getters
- Department() :
+ getInstance() : Department {static}
+ initialize(Season,List<Course>,List<Lecturer>,List<Assistant>,List<Advisor>,List<Student>
+addNewCourseSection(MandatoryCourse): void
+addNewLabSection(MandatoryCourse): void
- getNewSectionSchedule(MandatoryCourse,int): long
-generateWeeklyScheduleForAllCourses(): void
-generateWeeklyScheduleMandatoryCourses(): void
-generateWeeklyScheduleForElectiveCourses(): void
-generateCollisionlessWeeklyScheduleForCourses(List<Course>,): List<Tuple<Long,Long>>
-getDivisions(List<Course>): List<Tuple<List<Integer>,List<Integer>>>
-getDivision(int): List<Integer>
-validateDivision(List<Integer>,int): boolean
-assignScheduleToSection(long, List<Integer>): long
-getAvaibleDays(long, int): List<Integer>
-getAvaiblePositionsOnDay(long, int, int): List<Integer>
-getScheduleAtPosition(int, int): long
Extends
Extends
CourseSection
...
+ CourseSection(Course,String,Long,Lecturer) : Course
+ Getter, setters
LabSection
...
LabSection(Course,String,long,Assistant) : LabSection
+ Getter, setters