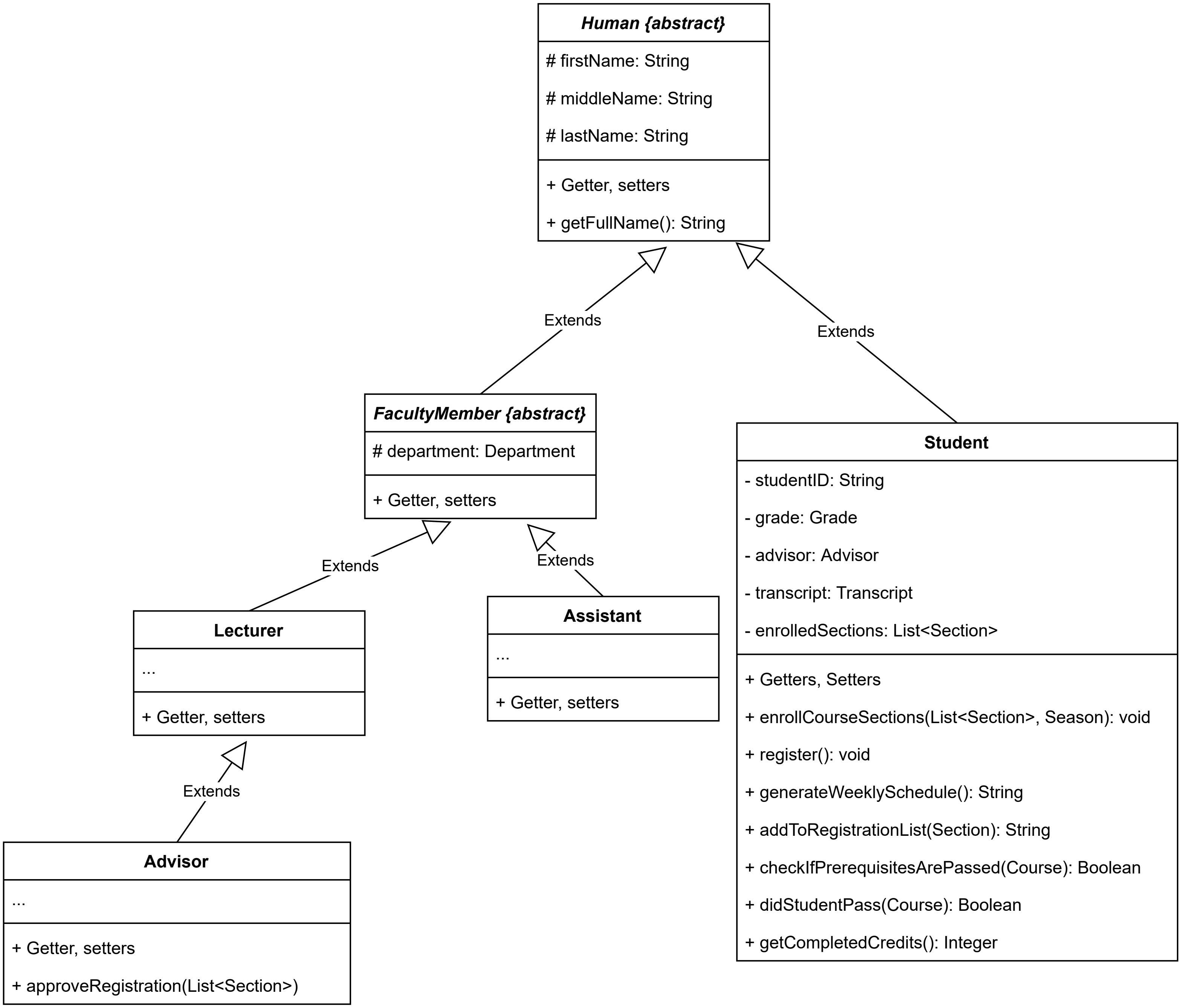
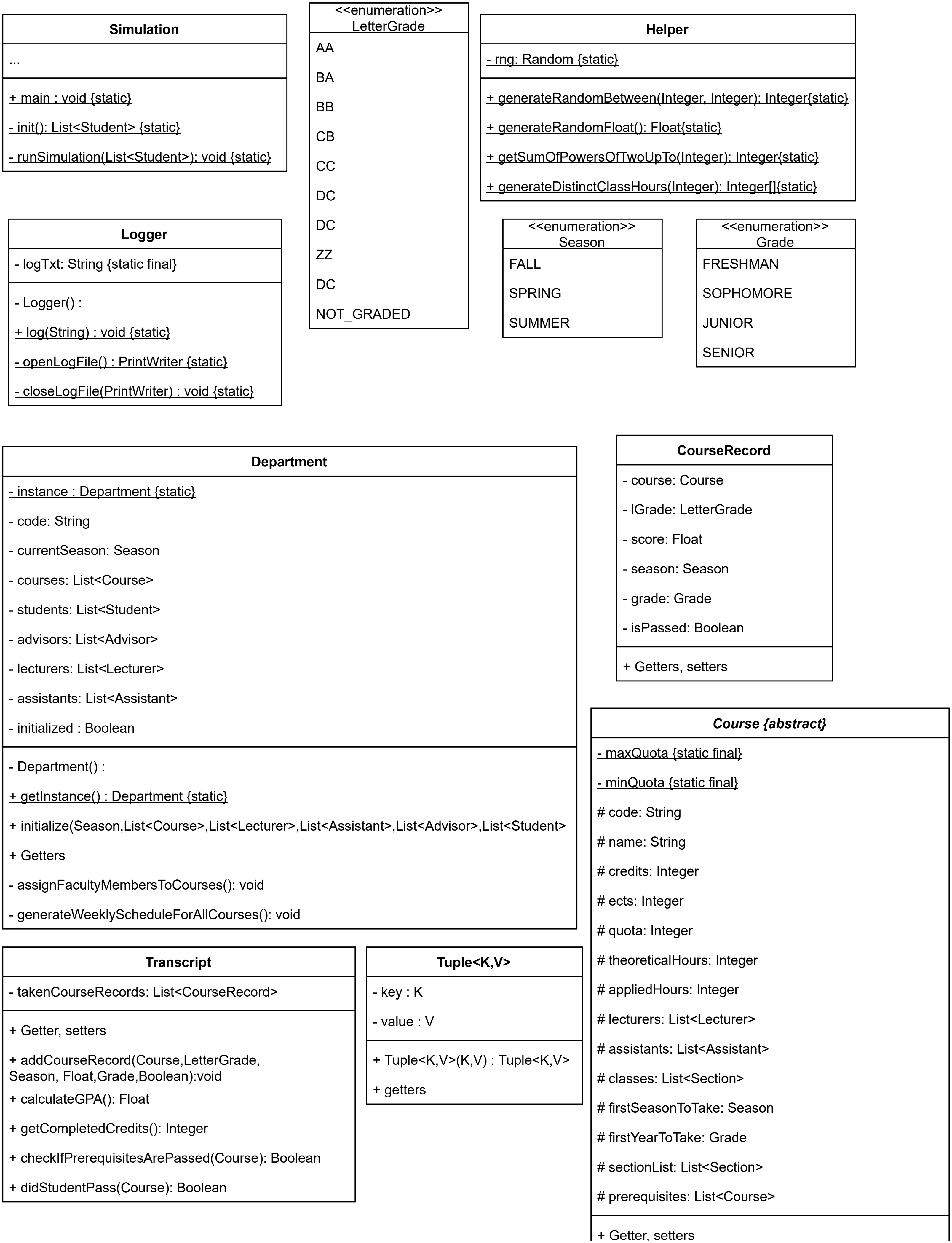

Human {abstract}
# firstName: String
# middleName: String
# lastName: String
+ Getter, setters
+ getFullName(): String
Extends
Extends
FacultyMember {abstract}
# department: Department
+ Getter, setters
Student
- studentID: String
- grade: Grade
- advisor: Advisor
- transcript: Transcript
- enrolledSections: List<Section>
+ Getters, Setters
+ enrollCourseSections(List<Section>, Season): void
+ register(): void
+ generateWeeklySchedule(): String
+ addToRegistrationList(Section): String
+ checkIfPrerequisitesArePassed(Course): Boolean
+ didStudentPass(Course): Boolean
+ getCompletedCredits(): Integer
Extends
Extends
Lecturer
...
+ Getter, setters
Assistant
...
+ Getter, setters
Extends
Advisor
...
+ Getter, setters
+ approveRegistration(List<Section>)



Simulation

...

- + main : void {static}
- - init(): List<Student> {static}
- - runSimulation(List<Student>): void {static}

<<enumeration>>
LetterGrade

- AA
- BA
- BB
- CB
- CC
- DC
- DC
- ZZ
- DC
- NOT_GRADED

Helper

- - rng: Random {static}

- + generateRandomBetween(Integer, Integer): Integer{static}
- + generateRandomFloat(): Float{static}
- + getSumOfPowersOfTwoUpTo(Integer): Integer{static}
- + generateDistinctClassHours(Integer): Integer[]{static}

Logger

- - logTxt: String {static final}

- - Logger() :
- + log(String) : void {static}
- - openLogFile() : PrintWriter {static}
- - closeLogFile(PrintWriter) : void {static}

<<enumeration>>
Season

- FALL
- SPRING
- SUMMER

<<enumeration>>
Grade

- FRESHMAN
- SOPHOMORE
- JUNIOR
- SENIOR

Department

- - instance : Department {static}
- - code: String
- - currentSeason: Season
- - courses: List<Course>
- - students: List<Student>
- - advisors: List<Advisor>
- - lecturers: List<Lecturer>
- - assistants: List<Assistant>
- - initialized : Boolean

- - Department() :
- + getInstance() : Department {static}
- + initialize(Season,List<Course>,List<Lecturer>,List<Assistant>,List<Advisor>,List<Student>
- + Getters
- - assignFacultyMembersToCourses(): void
- - generateWeeklyScheduleForAllCourses(): void

CourseRecord

- - course: Course
- - lGrade: LetterGrade
- - score: Float
- - season: Season
- - grade: Grade
- - isPassed: Boolean

- + Getters, setters

Course {abstract}

- - maxQuota {static final}
- - minQuota {static final}
- # code: String
- # name: String
- # credits: Integer
- # ects: Integer
- # quota: Integer
- # theoreticalHours: Integer
- # appliedHours: Integer
- # lecturers: List<Lecturer>
- # assistants: List<Assistant>
- # classes: List<Section>
- # firstSeasonToTake: Season
- # firstYearToTake: Grade
- # sectionList: List<Section>
- # prerequisites: List<Course>

- + Getter, setters

Transcript

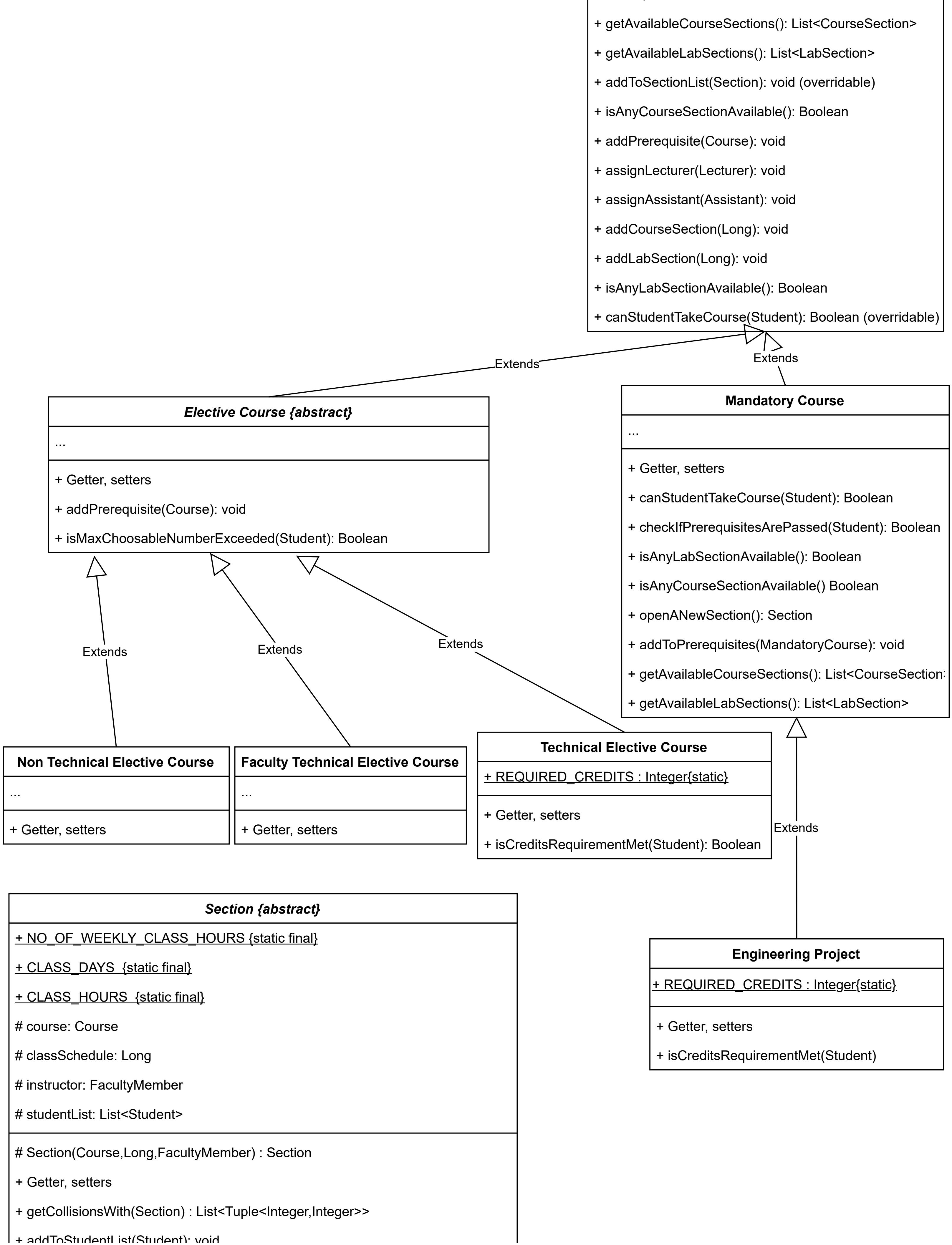
- - takenCourseRecords: List<CourseRecord>

- + Getter, setters
- + addCourseRecord(Course,LetterGrade, Season, Float,Grade,Boolean):void
- + calculateGPA(): Float
- + getCompletedCredits(): Integer
- + checkIfPrerequisitesArePassed(Course): Boolean
- + didStudentPass(Course): Boolean

Tuple<K,V>

- - key : K
- - value : V

- + Tuple<K,V>(K,V) : Tuple<K,V>
- + getters



+ getAvailableCourseSections(): List<CourseSection>

+ getAvailableLabSections(): List<LabSection>

+ addToSectionList(Section): void (overridable)

+ isAnyCourseSectionAvailable(): Boolean

+ addPrerequisite(Course): void

+ assignLecturer(Lecturer): void

+ assignAssistant(Assistant): void

+ addCourseSection(Long): void

+ addLabSection(Long): void

+ isAnyLabSectionAvailable(): Boolean

+ canStudentTakeCourse(Student): Boolean (overridable)

Elective Course {abstract}

...

+ Getter, setters

+ addPrerequisite(Course): void

+ isMaxChoosableNumberExceeded(Student): Boolean

Mandatory Course

...

+ Getter, setters

+ canStudentTakeCourse(Student): Boolean

+ checkIfPrerequisitesArePassed(Student): Boolean

+ isAnyLabSectionAvailable(): Boolean

+ isAnyCourseSectionAvailable() Boolean

+ openANewSection(): Section

+ addToPrerequisites(MandatoryCourse): void

+ getAvailableCourseSections(): List<CourseSection:

+ getAvailableLabSections(): List<LabSection>

Non Technical Elective Course

...

+ Getter, setters

Faculty Technical Elective Course

...

+ Getter, setters

Technical Elective Course

+ REQUIRED_CREDITS : Integer{static}

+ Getter, setters

+ isCreditsRequirementMet(Student): Boolean

Engineering Project

+ REQUIRED_CREDITS : Integer{static}

+ Getter, setters

+ isCreditsRequirementMet(Student)

Section {abstract}

+ NO_OF_WEEKLY_CLASS_HOURS {static final}

+ CLASS_DAYS {static final}

+ CLASS_HOURS {static final}

# course: Course

# classSchedule: Long

# instructor: FacultyMember

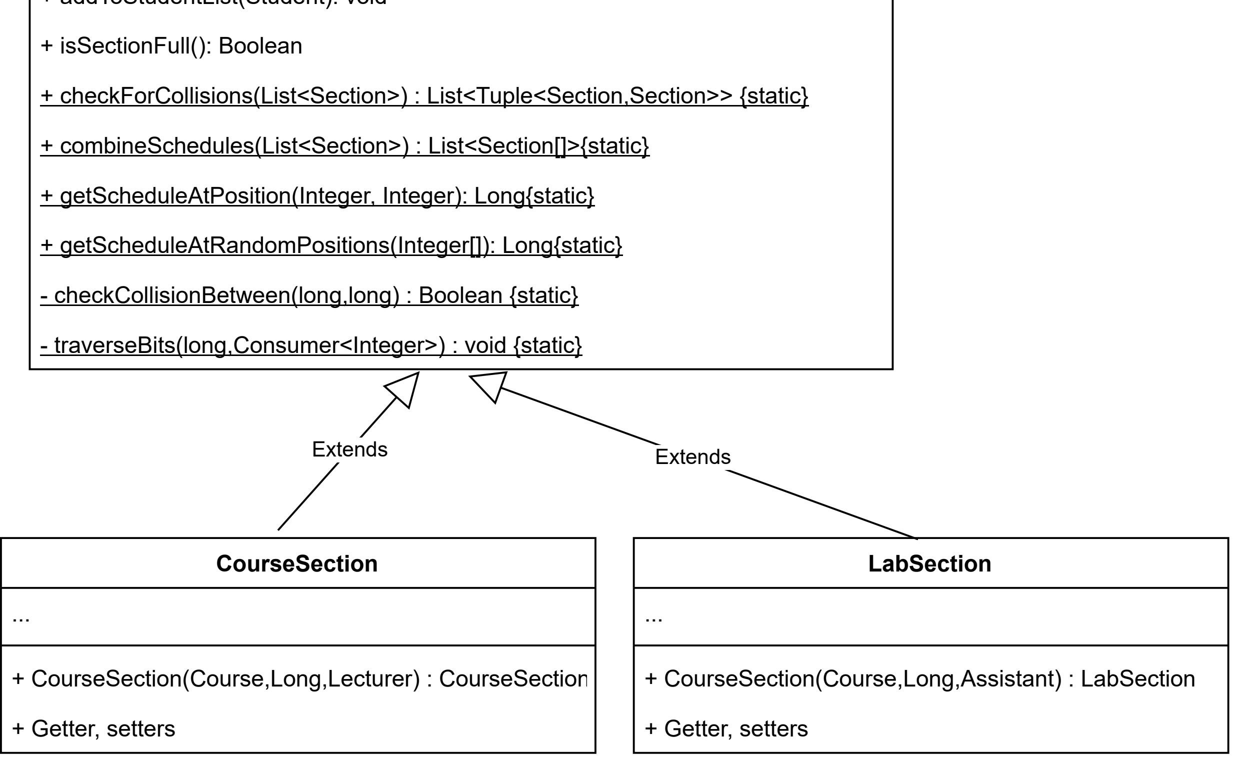
# studentList: List<Student>

# Section(Course,Long,FacultyMember) : Section

+ Getter, setters

+ getCollisionsWith(Section) : List<Tuple<Integer,Integer>>

+ addToStudentList(Student): void



+ addToStudentList(Student): void

+ isSectionFull(): Boolean

+ checkForCollisions(List<Section>) : List<Tuple<Section,Section>> {static}

+ combineSchedules(List<Section>) : List<Section[]>{static}

+ getScheduleAtPosition(Integer, Integer): Long{static}

+ getScheduleAtRandomPositions(Integer[]): Long{static}

- checkCollisionBetween(long,long) : Boolean {static}

- traverseBits(long,Consumer<Integer>) : void {static}

CourseSection

...

+ CourseSection(Course,Long,Lecturer) : CourseSection

+ Getter, setters

LabSection

...

+ CourseSection(Course,Long,Assistant) : LabSection

+ Getter, setters

JsonParser

- advisorsFile : String {final}
- lecturersFile : String {final}
- assistantsFile : String {final}
- coursesFile : String {final}
- semesterFile : String {final}
- studentsDir : String {final}

- JsonParser() :

+ parseSemester() : Season

+ parseAdvisors() : List<Advisor>

+ parseLecturers() : List<Lecturer>

+ parseAssistants() : List<Assistant>

+ parseCourses() : List<Course>

+ parseStudents(List<Advisor>,List<Course>) : List<Course>

+ serializeStudents(List<Student>) : void

- parseHuman(StringBuilder,StringBuilder,StringBuilder) : void
- parseHuman(StringBuilder,StringBuilder,StringBuilder) : void
- parseHuman(StringBuilder,StringBuilder,StringBuilder) : void
- isNull(String) : Boolean
- assignPrerequisitesToCourses(List<Course>,List<JSONArray>) : void
- findCourseWithCode(List<Course>,String) : Course
- parseCourseRecords(JSONArray,List<Course>) : Course
- findAdvisorByName(List<Advisor>,String) : Advisor
- readJsonFile(String) : Object
- readJsonFile(File) : Object
- writeToFile(String,String) : void

- writeToFile(String,String) : void