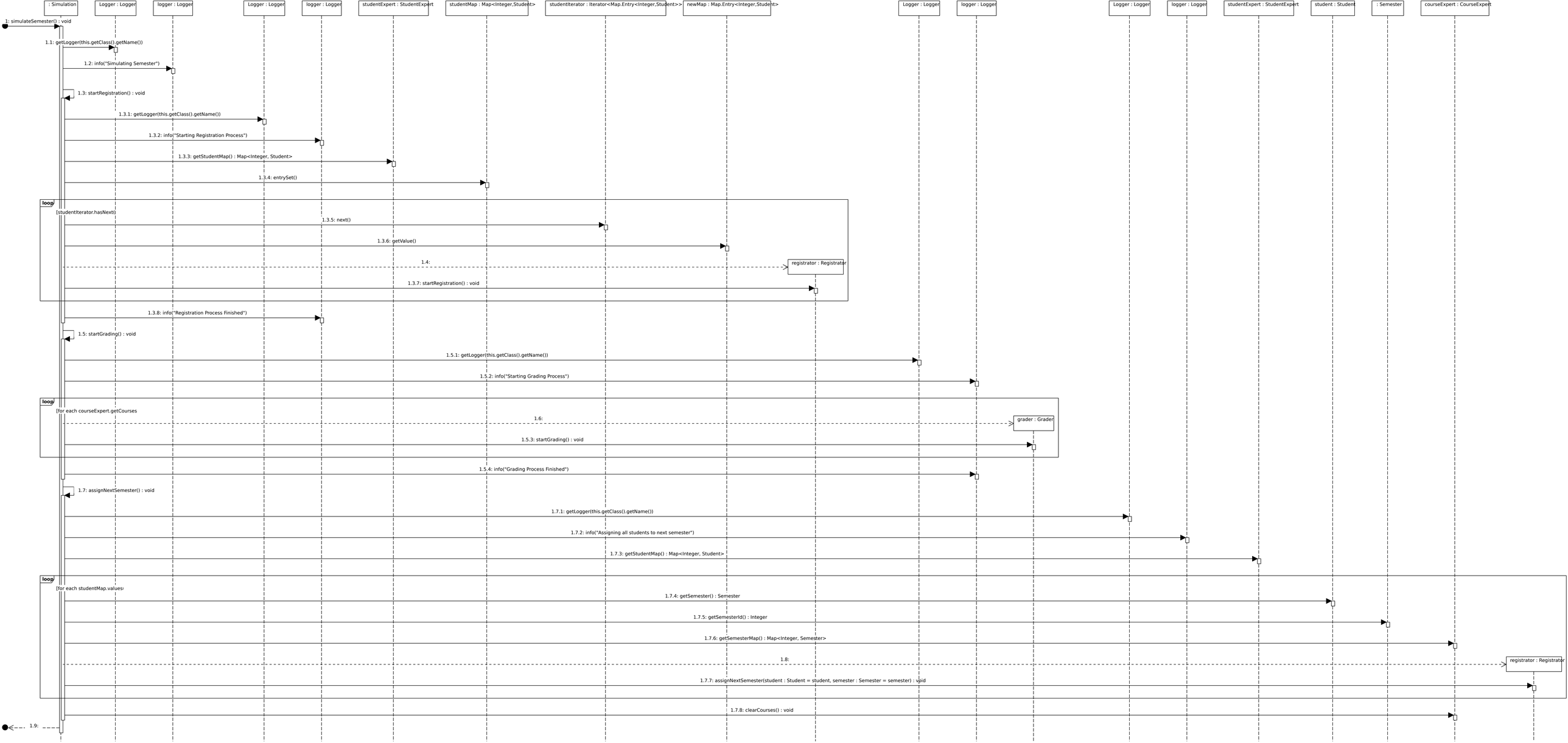
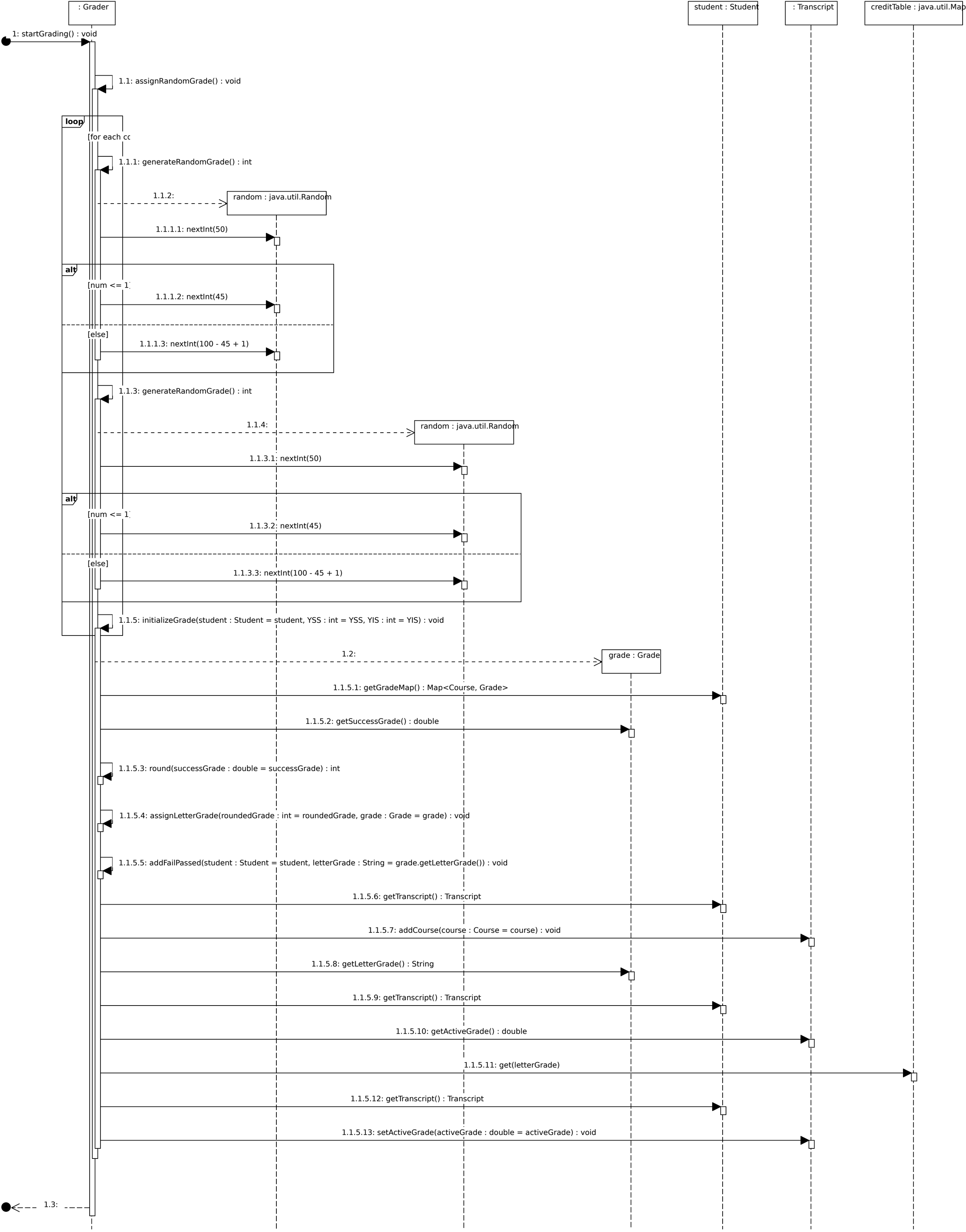
: Simulation
Logger : Logger
logger : Logger
inputReader : InputReader
studentExpert : StudentExpert
Logger : Logger
logger : Logger
inputReader : InputReader
transcriptReader : TranscriptReader
1: start() : void
1.1: getLogger(this.getClass().getName())
1.2: info("Simulation Started")
1.3: checkTranscriptFolder() : void
1.4: readInstructorJson(instructorExpert : InstructorExpert = instructorExpert) : void
1.5: readCourseJson(courseExpert : CourseExpert = courseExpert, instructorExpert : InstructorExpert = instructorExpert) : void
1.6: readPrerequisiteJson(courseExpert : CourseExpert = courseExpert) : void
1.7: setInstructors(instructors : List<Instructor> = new ArrayList<>(instructorExpert.getInstructorMap().values())) : void
1.8: readSeasonParameter() : String
1.9: readGenerationParameter() : String
1.10: info("Initializing Students")
1.11: initializeStudents(generation : String = generation, season : String = season) : void
1.11.1: getLogger(this.getClass().getName())
alt
[generation.equals("1"
1.11.2: info("Creating Random Students")
1.11.3: setLastStudent(lastStudent : int = lastStudent) : Simulation
1.11.4: setFirstStudent(firstStudent : int = firstStudent) : Simulation
alt
[season.equalsIgnoreCase("Fall"
[season.equalsIgnoreCase("Spring'
[else]
1.11.5: error("Error occured while initialization")
loop
[i < lastSemester
opt
[i % 2 == 1
1.11.6: getLastStudent() : int
1.11.7: readStudentJson(startIndex : int = superLast, studentExpert : StudentExpert = studentExpert, semester : Semester = courseExpert.getSemesterMap().get(1)) : int
1.11.8: setLastStudent(lastStudent : int = lastStudent) : Simulation
1.11.9: simulateSemester() : void
opt
[season.equalsIgnoreCase("Fall"
1.11.10: getLastStudent() : int
1.11.11: readStudentJson(startIndex : int = superLast, studentExpert : StudentExpert = studentExpert, semester : Semester = courseExpert.getSemesterMap().get(1)) : int
1.11.12: setLastStudent(lastStudent : int = ls) : Simulation
1.12: simulateSemester() : void
[else]
1.11.13: info("Reading Students From Existed Transcript")
1.11.14: readDepartmentOutput() : void
1.11.15: readTranscriptJson(studentExpert : StudentExpert = studentExpert, courseExpert : CourseExpert = courseExpert, instructorExpert : InstructorExpert = instructorExpert) : void
opt
[season.equalsIgnoreCase("Fall"
1.11.16: getLastStudent() : int
1.11.17: readStudentJson(startIndex : int = superLast, studentExpert : StudentExpert = studentExpert, semester : Semester = courseExpert.getSemesterMap().get(1)) : int
1.11.18: setLastStudent(lastStudent : int = ls) : Simulation
1.13: info("Preparing Department Output and Transcripts")
1.14: prepareErrorOutput() : Map<Course, List<Integer>>
1.15:
transcriptWriter : TranscriptWriter
1.16: startWriter() : void
1.17: prepareDepartmentOutput(firstStudent : int = firstStudent, lastStudent : int = lastStudent) : void
1.18: info("Simulation Finished")
1.19:


: Simulation
Logger : Logger
logger : Logger
Logger : Logger
logger : Logger
studentExpert : StudentExpert
studentMap : Map<Integer,Student>
studentIterator : Iterator<Map.Entry<Integer,Student>>
newMap : Map.Entry<Integer,Student>
Logger : Logger
logger : Logger
Logger : Logger
logger : Logger
studentExpert : StudentExpert
student : Student
: Semester
courseExpert : CourseExpert
1: simulateSemester() : void
1.1: getLogger(this.getClass().getName())
1.2: info("Simulating Semester")
1.3: startRegistration() : void
1.3.1: getLogger(this.getClass().getName())
1.3.2: info("Starting Registration Process")
1.3.3: getStudentMap() : Map<Integer, Student>
1.3.4: entrySet()
loop
[studentIterator.hasNext()]
1.3.5: next()
1.3.6: getValue()
1.4:
registrator : Registrator
1.3.7: startRegistration() : void
1.3.8: info("Registration Process Finished")
1.5: startGrading() : void
1.5.1: getLogger(this.getClass().getName())
1.5.2: info("Starting Grading Process")
loop
[for each courseExpert.getCourses]
1.6:
grader : Grader
1.5.3: startGrading() : void
1.5.4: info("Grading Process Finished")
1.7: assignNextSemester() : void
1.7.1: getLogger(this.getClass().getName())
1.7.2: info("Assigning all students to next semester")
1.7.3: getStudentMap() : Map<Integer, Student>
loop
[for each studentMap.values()]
1.7.4: getSemester() : Semester
1.7.5: getSemesterId() : Integer
1.7.6: getSemesterMap() : Map<Integer, Semester>
1.8:
registrator : Registrator
1.7.7: assignNextSemester(student : Student = student, semester : Semester = semester) : void
1.7.8: clearCourses() : void
1.9:

: Registrator
student : Student
Logger : Logger
logger : Logger
NontakenCourseList : List<Course>
iterator : Iterator<Course>
instructor : Instructor
courseExpert : CourseExpert
1: startRegistration() : void
1.1: getSemester() : Semester
1.2: getLogger(this.getClass().getName())
1.3: info("Registering " + student.getName() + " " + student.getSurname() + " to " + semester.getSemesterId() + ". semester")
loop
[for each student.getFailedCourses
opt
[approver.approveCourse(course
1.4: addBasket(course : Course = course) : void
1.5: getNonTakenCourses() : List<Course>
1.6: iterator()
loop
[iterator.hasNext()
1.7: next()
alt
[next instanceof ElectiveCours
1.8: selectRandomElective(course : Course = next) : Course
opt
[approver.approveCourse(course
1.9: addBasket(course : Course = course) : void
1.10: remove()
[approver.approveCourse(next
1.11: addBasket(course : Course = next) : void
1.12: remove()
loop
[for each seme
alt
[course instar
1.13: selectRandomElective(course : Course = course) : Course
loop
[!approver.approveCourse(electiv
alt
[course instanceof TechnicalElectiv
1.13.1: getTechnicalList() : List<ElectiveCourse>
[course instanceof NT_UElective
1.13.2: getNT_UList() : List<ElectiveCourse>
[course instanceof FacultyTechnicalElecti
1.13.3: getFacultyTechnicalList() : List<ElectiveCourse>
1.14: addBasket(course : Course = elective) : void
[approver.app
1.15: addBasket(course : Course = course) : void
1.16: info("Sending course basket to advisor approval")
1.17: getAdvisor() : Instructor
1.18: approveStudentBasket(student : Student = student) : void
1.19: addBasketToActiveCourse() : void
1.20:

: Approver
course : Course
student : Student
student : Student
student : Student
course : Course
approveCourse(course : Course) : boolean
opt
[course inst
1.1: capacityChecker() : boolean
1.1.1: getCapacity() : int
1.1.2: getStudents() : List<Student>
alt
[capacity > studentList.size
[else]
1.2:
quotaError : QuotaError
1.1.3: addError(error : Error = quotaError) : void
1.3: isElectiveTaken() : boolean
1.3.1: getPastCourses() : List<Course>
1.3.2: getCourseBasket() : List<Course>
alt
[pastCourse
[courseBask
1.4: prerequisiteChecker() : boolean
1.4.1: getPastCourses() : List<Course>
1.4.2: getPrerequisites() : List<Course>
loop
[for each prerequisiteCourse
opt
[!pastCourses.contains(require
1.5:
prerequisiteError : PrerequisiteError
1.4.3: addError(error : Error = prerequisiteError) : void
opt
[!student.getNonTakenCourses().contains(cour
1.4.4: addNonTakenCourse(course : Course = course) : void
1.6:


: Instructor
Logger : Logger
student : Student
: Semester
logger : Logger
Logger : Logger
logger : Logger
student : Student
courseBasket : List<Course>
iterator : Iterator<Course>
Logger : Logger
logger : Logger
student : Student
courseBasket : List<Course>
Logger : Logger
logger : Logger
student : Student
courseBasket : List<Course>
Logger : Logger
logger : Logger
student : Student
courseBasket : List<Course>
Logger : Logger
logger : Logger
student : Student
courseBasket : List<Course>
iterator : Iterator<Course>
Logger : Logger
logger : Logger
courseBasket : List<Course>
basketIterator : Iterator<Course>
firstCourse : Course
firstSection : Section
secondIterator : Iterator<Course>
tempCourse : Course
secondSection : Section
firstScheduleList : List<Schedule>
student : Student
proveStudentBasket(student : Student) : void
1.1: getLogger(this.getClass().getName())
1.2: getSemester() : Semester
1.3: getSeason() : String
1.4: getSemester() : Semester
1.5: getSemesterId() : Integer
opt
["Fall".equals
1.6: checkTwoTechnical(student : Student = student) : void
1.6.1: getLogger(this.getClass().getName())
1.6.2: info("Checking for more than two Technical Elective")
1.6.3: getCourseBasket() : List<Course>
1.6.4: iterator()
loop
[iterator.hasNext()
1.6.5: next()
opt
[next instanceof TechnicalElectiv
opt
[count > 2]
1.7:
technicalElectiveError : TwoTechnicalElectiveError
1.6.6: addNonTakenCourse(course : Course = new TechnicalElective()) : void
2: checkFTE(student : Student) : void
2.1: getLogger(this.getClass().getName())
2.2: info("Checking for Faculty Technical Elective courses")
1.8: getCourseBasket() : List<Course>
1.9: stream()
opt
[course != null
1.10: remove(course)
1.11:
notInGraduationError : NotInGraduationError
1.12: addNonTakenCourse(course : Course = new FacultyTechnicalElective()) : void
1.13: addError(error : Error = notInGraduationError) : void
1.14: addError(error : Error = technicalElectiveError) : void
1.15: remove()
1.16:
opt
[!(semesterI
1.17: checkFTE(student : Student = student) : void
1.17.1: getLogger(this.getClass().getName())
1.17.2: info("Checking for Faculty Technical Elective courses")
1.17.3: getCourseBasket() : List<Course>
1.17.4: stream()
opt
[course != null
1.17.5: remove(course)
1.18:
notInGraduationError : NotInGraduationError
1.17.6: addNonTakenCourse(course : Course = new FacultyTechnicalElective()) : void
1.17.7: addError(error : Error = notInGraduationError) : void
1.19: graduationProjectCheck(student : Student = student) : void
1.19.1: getLogger(this.getClass().getName())
1.19.2: info("Checking for graduation project")
1.19.3: getCourseBasket() : List<Course>
opt
[student.getCompletedCredit() < 1
1.19.4: stream()
opt
[course != null
1.19.5: remove(course)
1.20:
projectError : ProjectError
1.19.6: addNonTakenCourse(course : Course = course) : void
1.19.7: addError(error : Error = projectError) : void
1.21: checkMinimumCredit(student : Student = student) : void
1.21.1: getLogger(this.getClass().getName())
1.21.2: info("Checking for minimum credit")
1.21.3: getCourseBasket() : List<Course>
1.21.4: iterator()
loop
[iterator.hasNext()
1.21.5: next()
opt
[next instanceof TechnicalElective && student.getCompletedCredit() < ((TechnicalElective) next).getMinin
1.22:
uncompletedCreditError : UncompletedCreditError
1.21.6: addNonTakenCourse(course : Course = new TechnicalElective()) : void
1.21.7: addError(error : Error = uncompletedCreditError) : void
1.21.8: remove()
1.23: getCourseBasket() : List<Course>
1.24: collisionCheck(courseBasket : List<Course> = courseBasket, student : Student = student) : void
1.24.1: getLogger(this.getClass().getName())
1.24.2: info("Checking for collisions")
1.25:
willBeRemoved : java.util.ArrayList
1.24.3: iterator()
loop
[basketIterator.hasNext(
1.24.4: next()
alt
[willBeRemoved.contains(firstCours
[else]
1.24.5: iterator()
1.24.6: getSection() : Section
1.24.7: getScheduleList() : List<Schedule>
loop
[secondIterator.hasNext
1.24.8: next()
alt
[willBeRemoved.contains(tempCours
[else]
alt
[firstCourse.equals(tempCourse
[else]
1.24.9: getSection() : Section
1.24.10: getScheduleList() : List<Schedule>
1.24.11: stream()
opt
[!collisions.isEmpty()
alt
[firstCourse instanceof MandatoryCour
1.24.12: selectNonTakenCourse(course : Course = tempCourse) : Course
1.26:
error : CollisionError
1.24.13: addError(error : Error = error) : void
1.24.14: addNonTakenCourse(course : Course = selectedCourse) : void
1.24.15: add(tempCourse)
[else]
1.24.16: selectNonTakenCourse(course : Course = firstCourse) : Course
1.27:
error : CollisionError
1.24.17: addError(error : Error = error) : void
1.24.18: addNonTakenCourse(course : Course = selectedCourse) : void
1.24.19: add(firstCourse)
1.28: info("Course basket approved by " + this.fullName)
1.29:
1.24.20: removeAll(willBeRemoved)


: Grader
student : Student
: Transcript
creditTable : java.util.Map
1: startGrading() : void
1.1: assignRandomGrade() : void
loop
[for each cc
1.1.1: generateRandomGrade() : int
1.1.2:
random : java.util.Random
1.1.1.1: nextInt(50)
alt
[num <= 1
1.1.1.2: nextInt(45)
[else]
1.1.1.3: nextInt(100 - 45 + 1)
1.1.3: generateRandomGrade() : int
1.1.4:
random : java.util.Random
1.1.3.1: nextInt(50)
alt
[num <= 1
1.1.3.2: nextInt(45)
[else]
1.1.3.3: nextInt(100 - 45 + 1)
1.1.5: initializeGrade(student : Student = student, YSS : int = YSS, YIS : int = YIS) : void
1.2:
grade : Grade
1.1.5.1: getGradeMap() : Map<Course, Grade>
1.1.5.2: getSuccessGrade() : double
1.1.5.3: round(successGrade : double = successGrade) : int
1.1.5.4: assignLetterGrade(roundedGrade : int = roundedGrade, grade : Grade = grade) : void
1.1.5.5: addFailPassed(student : Student = student, letterGrade : String = grade.getLetterGrade()) : void
1.1.5.6: getTranscript() : Transcript
1.1.5.7: addCourse(course : Course = course) : void
1.1.5.8: getLetterGrade() : String
1.1.5.9: getTranscript() : Transcript
1.1.5.10: getActiveGrade() : double
1.1.5.11: get(letterGrade)
1.1.5.12: getTranscript() : Transcript
1.1.5.13: setActiveGrade(activeGrade : double = activeGrade) : void
1.3:

: Simulation
Logger : Logger
logger : Logger
studentExpert : StudentExpert
student : Student
: Semester
courseExpert : CourseExpert
Logger : Logger
student : Student
transcript : Transcript
logger : Logger
1: assignNextSemester() : void
1.1: getLogger(this.getClass().getName())
1.2: info("Assigning all students to next semester")
1.3: getStudentMap() : Map<Integer, Student>
loop
[for each studentMap.values
1.4: getSemester() : Semester
1.5: getSemesterId() : Integer
1.6: getSemesterMap() : Map<Integer, Semester>
1.6.1: getLogger(this.getClass().getName())
alt
[try]
1.6.2: getTranscript() : Transcript
1.6.3: getActiveGrade() : double
1.6.4: getCumulativeGrade() : double
1.6.5: setCumulativeGrade(cumulativeGrade : double = cumulativeGrade) : void
1.6.6: calculateGPA() : void
1.6.7: setActiveGrade(activeGrade : double = 0) : void
1.6.8: addGPA() : void
1.6.9: setActiveCredit(activeCredit : double = 0) : void
1.6.10: setGpa(gpa : float = 0) : void
1.6.11: calculateCGPA() : void
1.6.12: setSemester(semester : Semester = semester) : void
1.6.13: addSemester(semester : Semester = semester) : void
1.6.14: clearActiveCourses() : void
[catch NullPointerExceptior
1.6.15: error(e.getMessage())
1.6.16:
1.7:
registrator : Registrator
1.8: assignNextSemester(student : Student = student, semester : Semester = semester) : void
1.9: clearCourses() : void
1.10: