

class Program

Main

timetable.TryToGenerateTimetable()

timetable.DisplayMenu()

class Timetable

Timetable

this.courses=courseCreator.GenerateCourses();

triedTimetables = 0;

electiveCourses: Collection

electiveCourse: CollectionItem

= ﬁrst Element of electiveCourses

[ electiveCourse is invalid ]

blocks[electiveCourse.takesPlaceAtBlock-1].electiveCourses.Add(electiveCourse)

electiveCourse: next Element of electiveCourses

TryToGenerateTimetable

readerWriter.Write("Stundenplan wird generiert...")

this.GenerateTimetable(0, 0)

[ courses.Count != 0 ]

readerWriter.Write("Generierung hat zu lange gedauert")

readerWriter.Write("Stundenplan wurde erfolgreich generiert")

Environment.Exit(0)

GenerateTimetable

int currentBlock, int currentCourse

[ courses.Count != 0 ]

[ currentBlock < blocks.Count ]

TryNextTimetable()

[ currentCourse < course.Count]

[ blocks[currentBlock]. CheckIfBlocked(courses[currentCourse]) == false && courses[currentCourse]. professor.IsProfessorBlocked(blocks[currentBlock]) == false ]

GenerateTimetable(currentblock+1, 0)

AddCourseToBlockOrTryNextCourse(currentblock, currentCourse) GenerateTimetable(currentblock, currentCourse+1)

AddCourseToBlockOrTryNextCourse

int currentBlock, int currentCourse

[ TryToAddRoomToCourse(currentBlock, currentCourse) ]

blocks[currentBlock].courses. Add(courses[currentCourse])

courses.Remove(courses[currentCourse])

GenerateTimetable(currentblock, currentCourse+1)

TryToAddRoomToCourse

int currentBlock, int currentCourse

rooms: Collection

room: CollectionItem

= ﬁrst Element of rooms

return false

[ room is invalid ]

[ CheckIfRoomIsFitting

(room, courses[currentCourse], blocks[currentBlock])) ]

room: next Element of rooms

courses[currentCourse].room = room;

return true

CheckIfRoomIsFitting

Room room, Course course, Block block

[ room.CheckRoomEquipment(course) == true

&& room.CheckRoomCapacity(course) == true

&& block.CheckIfRoomIsBlocked(room) == false

&& block.CheckIfRoomIsBlockedGeneraly(room) == false ]

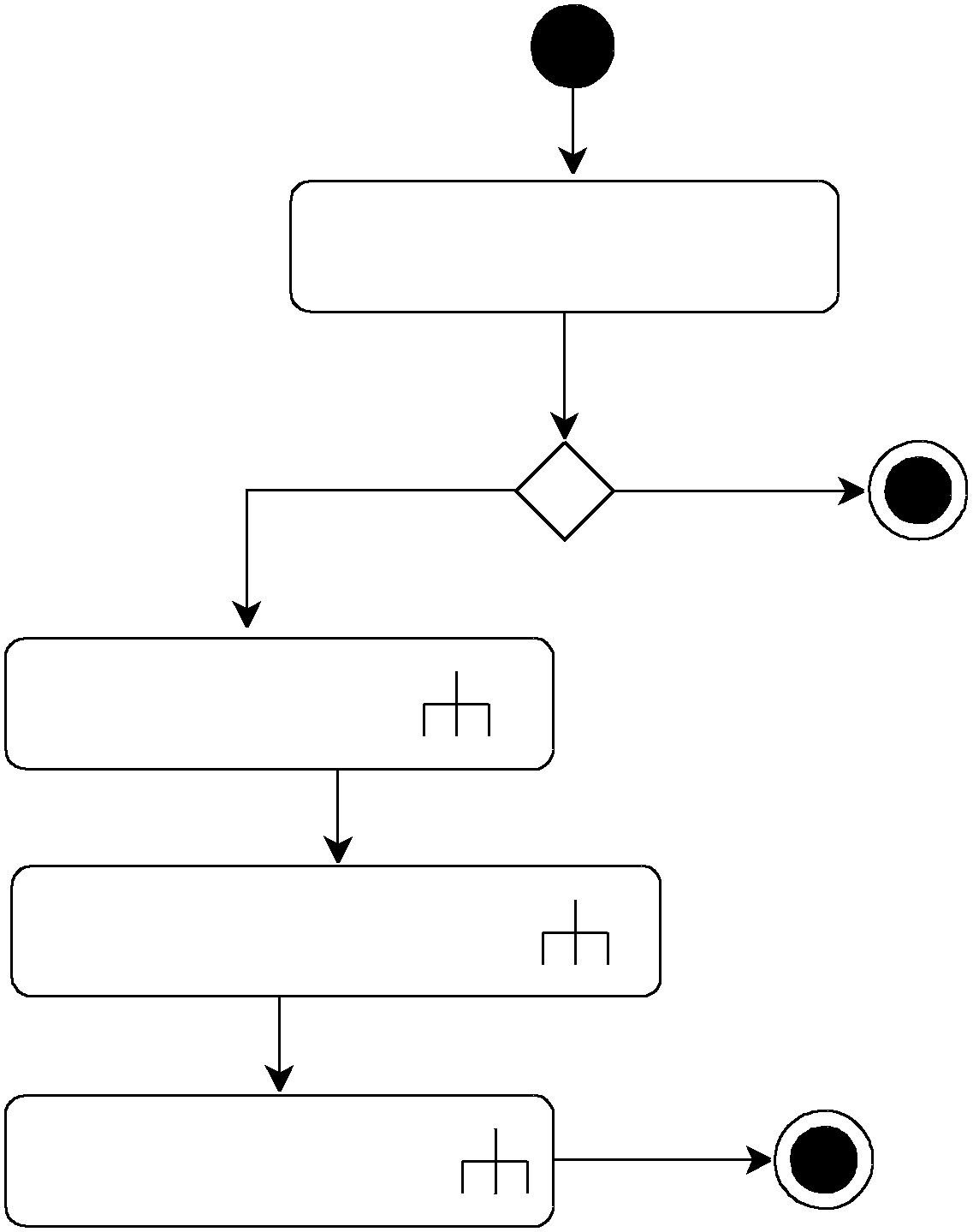
return true return false

TryNextTimetable

triedTimetables++

triedTimetables < 10000

ResetTimeTable()



ShufﬂeCourses(courses);

GenerateTimetable(0, 0)

ResetTimetable

blocks: Collection

blocks: CollectionItem

= ﬁrst Element of block

[ block is invalid ]

block: next Element of blocks

int i = 0

block.courses.Clear()

[ i < block.courses.Count ]

i++

courses.Add(block.courses[i])

ShufﬂeCourses

List<Course> courses

int listLength = courses.Count;

[ listLength > 1]

listLenght--

int random =

random.Next(listLength+1)

Course course =

courses[random]

courses[random] =

courses[listLength]

courses[listLength] =

course

DisplayMenu

[ while true ]

readerWriter.Write(Auswahlmöglichkeiten)

string input

= readerWriter.Read();

PrintCompleteTimetable()

[ 1 ]

[ 3 ]

ShowAvailabelOptions("professor")

[ 6 ]

Environment.Exit(0) Console.WriteLine

[ 5 ]

[ 4 ]

[ default ]

(Nutzerhinweis)

ShowAvailabelStudentOptions("electiveCourse")

ShowAvailabelOptions("room")

[ 2 ]

ShowAvailabelStudentOptions("students")

ShowAvailabelOptions

string info

[ professor ] [ room ]

int i = 0

int i = 0

[ i < professors.Count ]

[ i < rooms.Count ]

i++

readerWriter.Write(i +

") " + professors[i].name)

i++

readerWriter.Write( i + ") " rooms[i].name)

string professorChoice = Console.Readline()

string roomChoice = Console.Readline()

this.PrintSpeciﬁcTimetable

(professors[professorChoice].name)

this.PrintSpeciﬁcTimetable

(rooms[roomChoice].name)

ShowAvailabelStudentOptions

string option

int i = 0

i++

[ i < students.Count ]

readerWriter.Write( i +")" + students[i].courseOfStudies + students[i].semester);

int input = Convert.ToInt32(readerWriter.Read())

string info = students[input].courseOfStudies + students[input].semester

[ option == "electiveCourse" ]

[ option == "students" ]

this.PrintElectiveCourseTimetable(info) this.PrintSpeciﬁcTimetable(info)

string info

PrintSpeciﬁcCourseTimetable

blocks: Collection

block: CollectionItem

= ﬁrst Element of blocks

[ block is invalid ]

block.PrintSpeciﬁcCourse(info)

block: next Element of blocks

string info

PrintElectiveCourseTimetable

blocks: Collection

block: CollectionItem

= ﬁrst Element of blocks

[ block is invalid ]

block.PrintElectiveCourseInformation(info)

block: next Element of blocks

string info

PrintCompleteTimetable

blocks: Collection

block: CollectionItem

= ﬁrst Element of blocks

[ block is invalid ]

block.PrintAllCourses(info)

block: next Element of blocks

class CourseCreator

GenerateCourses

courses: Collection

course: CollectionItem

= ﬁrst Element of courses

[ course is invalid ]

AddProfessorsToCourses(courses)

course: next Element of courses

AddStudentsToCourses(courses)

[ course.CheckIfAllInformationIsAssigned()

== false ]

Console.WriteLine(course.errorReport)

Environment.Exit(0)

Console.WriteLine(Erfolgsmitteilung)

return courses

Course course

AddStudentsToCourses

students: Collection

student: CollectionItem

= ﬁrst Element of students

[ student is invalid ]

course.AddStudentsToCourse(students)

student: next Element of students

Course course

AddProfessorsToCourses

professors: Collection

professor: CollectionItem

= ﬁrst Element of professors

[ professor is invalid ]

course.AddProfessorsToCourse(professor)

professor: next Element of professors

class Room

CkeckRoomEqipment

Course course

bool roomIsFitting;

int i = 0

return roomIsFitting

i++

[ Array.IndexOf (this.equipment, course.equipment[i]) >= 0 ]

[ i < course.equipment.Length ]

roomIsFitting = true roomIsFitting = false return roomIsFitting

CkeckRoomCapacity

Course course

int completeNumberOfStudents

int i = 0

[ i < course.students.Count ]

i++

completeNumberOfStudents +=

course.students[i].numberOfStudents

[ this.capacity >= completeNumberOfStudents ]

return true return false

class Students

CheckStudentsObligatoryEvents

Course course

[ this.obligatoryEvents.Contains(course.eventName) ]

return true return false

class Block

CheckIfBlocked

Course course

string eventProfessor = course.professor.name;

int i = 0

i++ return false

[ i < course.students.Count ]

string eventCourseOfStudies = course.students[i].courseOfStudies

int eventSemester = course.students[i].semester

int j = 0

j++

[ j < this.courses.Count ]

int k = 0

k++

[k < this.courses[j].students.Count ]

string courseOfStudies = this.courses[j].students[k].courseOfStudies

int semester = this.courses[j].students[k].semester

string professor = this.courses[j].professor.name;

[ courseOfStudies == eventCourseOfStudies &&

semester == eventSemester ||

professor == eventProfessor ]

return true

CheckIfRoomIsBlocked

Room room

courses: Collection

course: CollectionItem

= ﬁrst Element of courses

return false

[ course is invalid ]

[ course.room.name == room.name ]

course: next Element of courses

return true

CheckIfRoomIsBlockedGeneraly

Room room

electiveCourses: Collection

electiveCourse: CollectionItem

= ﬁrst Element of electiveCourses

return false

[ electiveCourse is invalid ]

[ electiveCourse.roomUsed == room.name ]

electiveCourse: next Element of electiveCourses

return true

PrintSpeciﬁcCourses

string info

readerWriter.Write(this.day + " " + this.blockNumber + ". Block " + this.blockTime);

courses: Collection

course: CollectionItem

= ﬁrst Element of courses

[ course is invalid ]

[ CheckIfCourseContainsInformation(info, course) ]

course: next Element of courses

PrintCourse(course)

CheckIfCourseContainsInformation

string info, Course course

int i = 0

return false

i++

[ i < course.students.Count ]

[ info==students ||

string students = course.students[i].courseOfStudies + course.students[i].semester

info == course.professor.name ||

info == course.room.name ]

return true

PrintCourse

Course course

readerWriter.Write(course.eventName);

int i = 0

i++

[ i < course.students.Count ]

readerWriter.Write(course.students[i].courseOfStudies + " " + course.students[i].semester)

readerWriter.Write(course.professor.name)

readerWriter.Write(course.room.name + "\n")

PrintAllCourses

readerWriter.Write(this.day + " " + this.blockNumber + ". Block " + this.blockTime);

courses: Collection

course: CollectionItem

= ﬁrst Element of courses

[ course is invalid ]

PrintCourse(course)

course: next Element of courses

PrintElectiveCourseInformation

string info

readerWriter.Write(this.day + " " + this.blockNumber + ". Block " + this.blockTime)

[ PrintElectiveCoursesTimetable(info) ]

electiveCourses: Collection

electiveCourse: CollectionItem

= ﬁrst Element of electiveCourses

[ electiveCourse is invalid ]

readerWriter.Write(electiveCourse.electiveCourseName)

electiveCourse: next Element of electiveCourses

readerWriter.Write(electiveCourse.professorName)

readerWriter.Write(electiveCourse.roomUsed + "\n")

PrintElectiveCoursesTimetable

string info

bool studentsHaveTime

courses: Collection

course: CollectionItem

= ﬁrst Element of courses

return studentsHaveTime

[ course is invalid ]

int i = 0

course: next Element of courses

i++

[ i < course.students.Count ]

string students = course.students[i].courseOfStudies + course.students[i].semester

[ info != students ]

studentsHaveTime = true studentsHaveTime = false; return studentsHaveTime

class ConsoleWriterReader

Write

string text

Read

Console.WriteLine(text);

return Console.ReadLine();

class Course CheckIfAllInformationIsAssigned

[ this.students == null ]

errorReport = "Keine Studenten für den Kurs

" + this.eventName + " gefunden!"

return false

[ this.professor == null ]

errorReport = "Kein professor für den Kurs

" + this.eventName + " gefunden!"

return false

return true

AddStudentsToCourse

Students students

AddProfessorsToCourse

Professor professor

[ students.CheckStudentsObligatoryEvents(this)

== true ]

[ professor.CheckProfessorCourses(this)

== true ]

this.students.Add(students)

this.professor = professor

class Professor

CheckProfessorCourses

Course course

[ this.courses.Contains(course.eventName) ]

return true return false

IsProfessorBlocked

Block block

[ this.blockedAtBlocks.Contains(block.blockIndex) ]

return true return false

class Json

deserializeRooms

List<Room> rooms = JsonConvert.DeserializeObject<List<Room>> (File.ReadAllText(@"room.json"), settings);

return rooms

(Für List<Students>, List<Professor>, List<Course>, List<ElectiveCourse> und List<Block>

auf dieselbe Art und Weise)